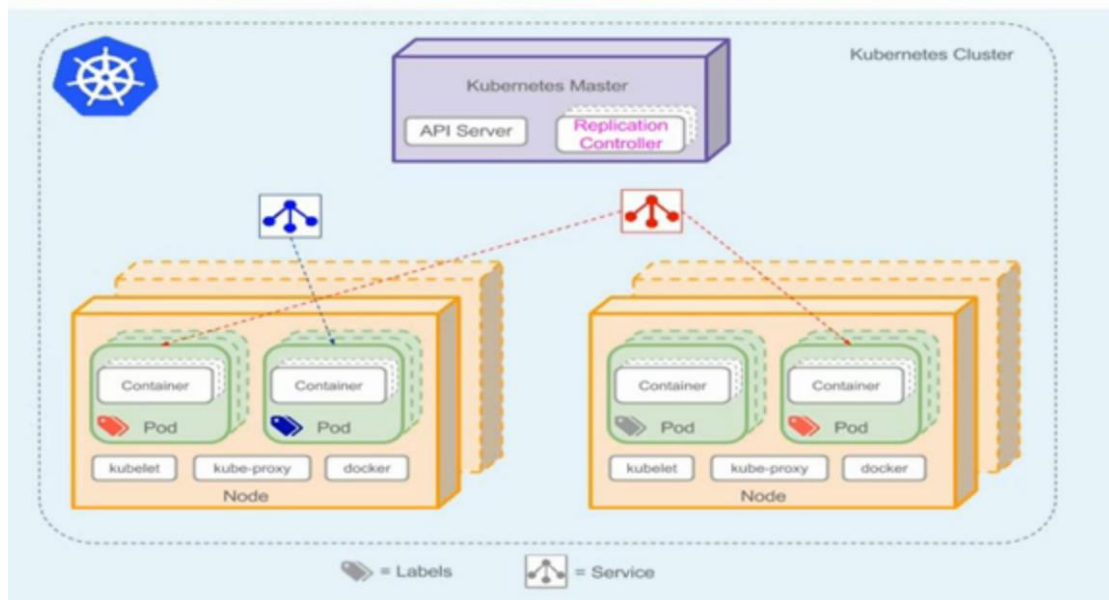
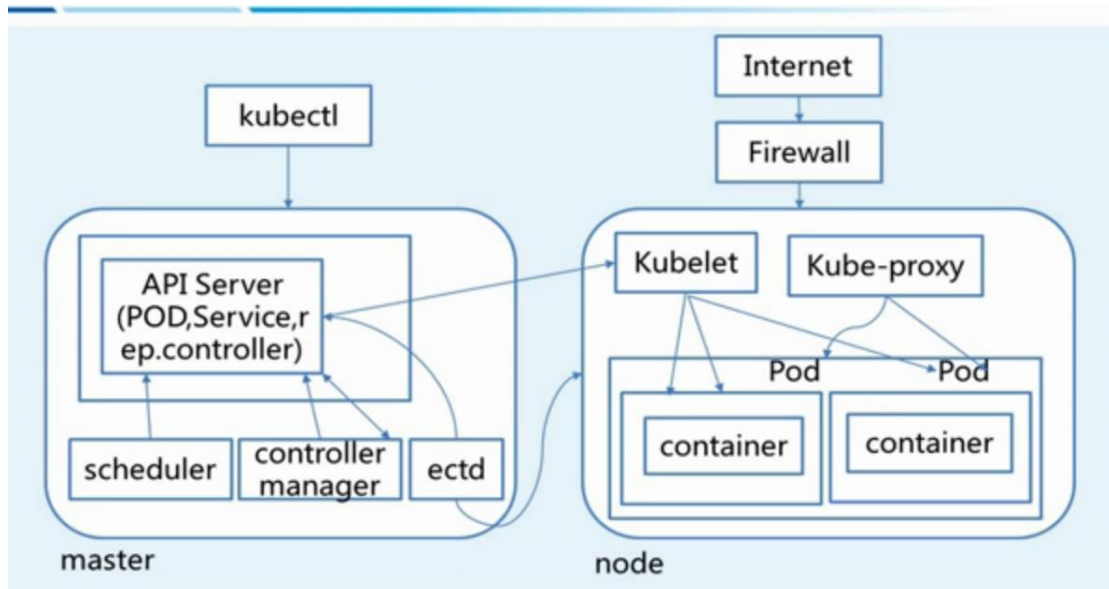


整体介绍

Kubernetes 架构



K8S 架构图



工作流程大概是这样的

1. kubectrl (一个命令, 控制 k8s 的命令) 说创建三个容器,
2. 一条命令发给 apiserver, 说要创建容器的副本数量为 3.
3. 这时候 scheduler 收到消息, 查看有那些 node, 选一个空闲的, 让他创建

4. 此时 apiserver 就会通知 node 节点上的 kubelet，这时候 kubelet 就会帮他创建容器。
5. contrller manager 里面有个复制控制器 replcation contrller，它看创建了几个容器，是两个就改成三个。（复制容器）
6. 所有数据存在 ectd 中
7. kuber-proxy 来实现三个容器的负载均衡

1.1、Master 服务

API Server: 供 Kubernetes API 接口，主要处理 REST 操作以及更新 ETCD 中的对象，所有资源增删改查的唯一入口。

scheduler: 服务做调度，负责 pod 到 Node 的调度

controller manager: 所有其他群集级别的功能，目前由控制器 Manager 执行，资源对象的控制中心

etcd: 所有持久化的状态信息存储在 ETCD 中。

本次学习目标



1.2、Node 节点服务介绍

1. kubelet: 管理容器的。和 master 进行通信，master 说创建一个容器就创建一个容器，说关就关。（传统 mstart 和 agent 的结构）
2. kube-proxy: 用来做代理的，部署之后才能理解。相当于做负载均衡的（这个版本有两种方式，iptables 和 lvs），实现与 service 通信。

3. Docker Engine: kubelet 是对宿主机管理的，真正创建容器的是这个 docker（个人理解）

<https://github.com/unixhot/salt-kubernetes> 地址最下面有手动部署

环境准备



k8s环境准备.pdf

1.1.1 环境搭建主机规划

主机名 (FQDN)	IP 地址 (NAT)
linux-node1.example.com	eth0:192.168.56.11
linux-node2.example.com	eth0:192.168.56.12
linux-node2.example.com	eth0:192.168.56.13

1.1.2 环境优化

- 1、关闭 NetworkManager 和防火墙开启自启动

```
systemctl disable firewalld  
systemctl disable NetworkManager
```

- 2、设置主机名

```
[root@localhost ~]# vi /etc/hostname  
linux-node1.example.com
```

- 3、设置主机名解析

```
[root@linux-node1 ~]# cat /etc/hosts  
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
```

```
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.56.11 linux-node1 linux-node1.example.com
192.168.56.12 linux-node2 linux-node2.example.com
192.168.56.13 linux-node3 linux-node3.example.com
4、设置 DNS 解析
[root@localhost ~]# vi /etc/resolv.conf
nameserver 192.168.56.2

5、安装 EPEL 仓库和常用命令
[root@linux-node1 ~]# rpm -ivh http://mirrors.aliyun.com/epel/epel-release-latest-7.noarch.rpm
[root@linux-node1 ~]# yum install -y net-tools vim lrzsz tree screen lsof tcpdump nc mtr nmap

6、关闭并确认 SELinux 处于关闭状态
[root@linux-node1 ~]# vim /etc/sysconfig/selinux SELINUX=disabled #修改为 disabled
```

1.2、系统环境初始化

1.2.1 第一步：使用国内 Docker 源

```
cd /etc/yum.repos.d/
wget \
https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

1.2.2 第二步：Docker 安装：

```
[root@linux-node1 ~]# yum install -y docker-ce
```

1.2.3 第三步：启动后台进程：

```
[root@linux-node1 ~]# systemctl start docker
```

1.3、准备部署目录

```
mkdir -p /opt/kubernetes/{cfg,bin,ssl,log} #所有机器都需要创建
```

目录注释：

1. cfg 配置文件
2. bin 二进制文件

3. ssl 证书文件
4. log 日志

1.4、准备软件包

百度网盘下载地址：

<https://pan.baidu.com/s/1zs8sCouDeCQJ9lghH1BPiw>

cd /usr/local/src

[root@linux-node1 src]# ls -lhi

total 586M

33636301 -rwxr-xr-x 1 root root 6.3M Mar 30 2016 cfssl-certinfo_linux-amd64

33898268 -rwxr-xr-x 1 root root 2.2M Mar 30 2016 cfssljson_linux-amd64

33898267 -rwxr-xr-x 1 root root 9.9M Mar 30 2016 cfssl_linux-amd64

33898269 -rw-r--r-- 1 root root 17M Apr 12 17:35 cni-plugins-amd64-v0.7.1.tgz #网络接口一插件

33898270 -rw-r--r-- 1 root root 11M Mar 30 01:58 etcd-v3.2.18-linux-amd64.tar.gz #存储

33981463 -rw-r--r-- 1 root root 9.3M Jan 24 2018 flannel-v0.10.0-linux-amd64.tar.gz #网络

33981464 -rw-r--r-- 1 root root 13M Apr 13 01:51 kubernetes-client-linux-amd64.tar.gz

33981465 -rw-r--r-- 1 root root 108M Apr 13 01:51 kubernetes-node-linux-amd64.tar.gz

33981466 -rw-r--r-- 1 root root 409M Apr 13 01:51 kubernetes-server-linux-amd64.tar.gz

33981467 -rw-r--r-- 1 root root 2.6M Apr 13 01:51 kubernetes.tar.gz #源码包

官方下载地址

<https://github.com/kubernetes/kubernetes>

1.5、解压软件包

#注意 tar 这种写法，一层层的写入文件夹。之前没有留意过

```
tar xzf kubernetes.tar.gz
```

```
tar xzf kubernetes-server-linux-amd64.tar.gz
```

```
tar xzf kubernetes-client-linux-amd64.tar.gz
```

```
tar xzf kubernetes-node-linux-amd64.tar.gz
```

添加环境变量

```
[root@linux-node1 ~]# vim .bash_profile
```

```
# .bash_profile
```

```
# Get the aliases and functions
```

```
if [ -f ~/.bashrc ]; then
```

```
    . ~/.bashrc
```

```
fi
```

```
# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/opt/kubernetes/bin

export PATH

source .bash_profile
```

遇到问题：不生效，退出后重新登陆搞定

ca 证书

ca 证书的创建和分发

1.1、证书介绍

Kubernetes 系统各组件需要使用 TLS 证书对通信进行加密

CA 证书管理

1. easyrsa : easyrs openvpn 用的是这个证书
2. openssl :
3. cfssl : cfssl 是用的最多的，相对简单，通过用 json 文件的形式把证书的东西配置进去。下载地址：<https://pkg.cfssl.org/>

上面三个都可以用来加密，此次配置选用的是 cfssl

1.2、ca 认证原理

参考文章

<http://blog.itpub.net/28624388/viewspace-2152064/>

一：前言

SSL : Secure Sockets Layer, 标准化后叫"TLS", http协议默认情况下是不加密内容的, 这样就很可能在内容传播的时候被别人监听到, 对于安全性要求较高的场合, 必须要加密, https就是带加密的http协议。

消息-->[公钥]-->加密后的信息-->[私钥]-->消息

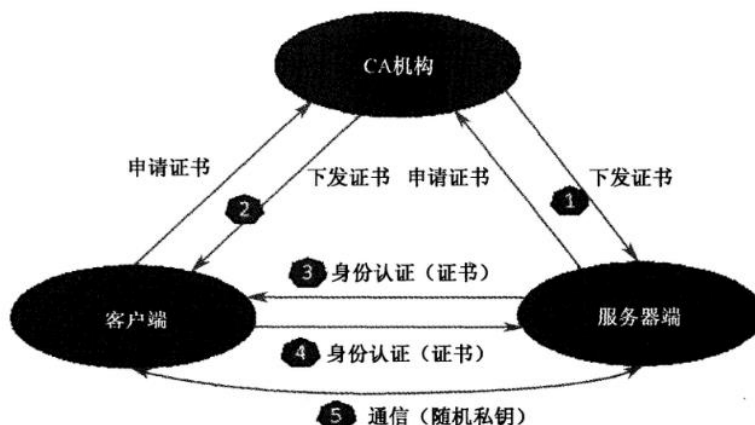


图 3.11 CA 认证流程

如图 3.11 所示，SSL 双向认证大概包含下面几个步骤。

(1) HTTPS 通信双方的服务器端向 CA 机构申请证书，CA 机构是可信的第三方机构，它可以是一个公认的权威的企业，也可以是企业自身。企业内部系统一般都用企业自身的认证系统。CA 机构下发根证书、服务端证书及私钥给申请者。

(2) HTTPS 通信双方的客户端向 CA 机构申请证书，CA 机构下发根证书、客户端证书及私钥给申请者。

(3) 客户端向服务器端发起请求，服务器端下发服务端证书给客户端。客户端接收到证书后，通过私钥解密证书，并利用服务器端证书中的公钥认证证书信息比较证书里的消息，例如域名和公钥与服务器刚刚发送的相关消息是否一致，如果一致，则客户端认可这个服务器的合法身份。

(4) 客户端发送客户端证书给服务器端，服务器端接收到证书后，通过私钥解密证书，获得客户端证书公钥，并用该公钥认证证书信息，确认客户端是否合法。

(5) 客户端通过随机密钥加密信息，并发送加密后的信息给服务器端。服务器端和客户端协商好加密方案后，客户端会产生一个随机的密钥，客户端通过协商好的加密方案，加密该随机密钥，并发送该随机密钥到服务器端。服务器端接收这个密钥后，双方通信的所有内容都通过该随机密钥加密。

1.3、手工制作 CA 证书

1.3.1 安装 CFSSL

```
[root@linux-node1 ~]# cd /usr/local/src
[root@linux-node1 src]# wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
[root@linux-node1 src]# wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
```

```
[root@linux-node1 src]# wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
chmod +x cfssl*
mv cfssl-certinfo_linux-amd64 /opt/kubernetes/bin/cfssl-certinfo
mv cfssljson_linux-amd64 /opt/kubernetes/bin/cfssljson
mv cfssl_linux-amd64 /opt/kubernetes/bin/cfssl
```

复制 cfssl 命令文件到 k8s-node1 和 k8s-node2 节点。如果实际中多个节点，就都需要同步复制。

```
ssh-keygen -t rsa #创建证书 ssh 免密钥登陆
ssh-copy-id linux-node1 #本地也要复制一份
ssh-copy-id linux-node2
ssh-copy-id linux-node3
scp /opt/kubernetes/bin/cfssl* 192.168.56.12:/opt/kubernetes/bin #记得节点创建和 master 一样的目录
scp /opt/kubernetes/bin/cfssl* 192.168.56.13:/opt/kubernetes/bin
```

1.3.2 初始化 cfssl

```
[root@linux-node1 src]# mkdir ssl && cd ssl
注释：下面两个命令也是生成证书和私钥的。但是，没有使用。因为下面手动的有 k8s 相关的信息。
[root@linux-node1 ssl]# cfssl print-defaults config > config.json
[root@linux-node1 ssl]# cfssl print-defaults csr > csr.json
```

1.3.3 创建用来生成 CA 文件的 JSON 配置文件

```
[root@linux-node1 ssl]# vim ca-config.json #这些证书是官方提供的
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
          "client auth"
        ],
        "expiry": "8760h"
      }
    }
  }
}
```



```
}  
}  
}
```

1.3.4 创建用来生成 CA 证书签名请求（CSR）的 JSON 配置文件

```
[root@linux-node1 ssl]# vim ca-csr.json  
{  
  "CN": "kubernetes",  
  "key": {  
    "algo": "rsa",  
    "size": 2048  
  },  
  "names": [  
    {  
      "C": "CN",  
      "ST": "BeiJing",  
      "L": "BeiJing",  
      "O": "k8s",  
      "OU": "System"  
    }  
  ]  
}
```

1.3.5 生成 CA 证书（ca.pem）和密钥（ca-key.pem）

```
[root@linux-node1 ssl]# cfssl gencert -initca ca-csr.json | cfssljson -bare ca  
[root@linux-node1 ssl]# ls -l ca* #此时生成三个文件  
-rw-r--r-- 1 root root 290 Mar  4 13:45 ca-config.json  
-rw-r--r-- 1 root root 1001 Mar  4 14:09 ca.csr  
-rw-r--r-- 1 root root 208 Mar  4 13:51 ca-csr.json  
-rw----- 1 root root 1679 Mar  4 14:09 ca-key.pem  
-rw-r--r-- 1 root root 1359 Mar  4 14:09 ca.pem
```

1.3.6 分发证书

```
# cp ca.csr ca.pem ca-key.pem ca-config.json /opt/kubernetes/ssl  
SCP 证书到 k8s-node1 和 k8s-node2 节点  
# scp ca.csr ca.pem ca-key.pem ca-config.json 192.168.56.12:/opt/kubernetes/ssl #以后添加节点也需要有这个证书，所以新开的要把这个弄进去  
# scp ca.csr ca.pem ca-key.pem ca-config.json 192.168.56.13:/opt/kubernetes/ssl
```

ETCD 集群

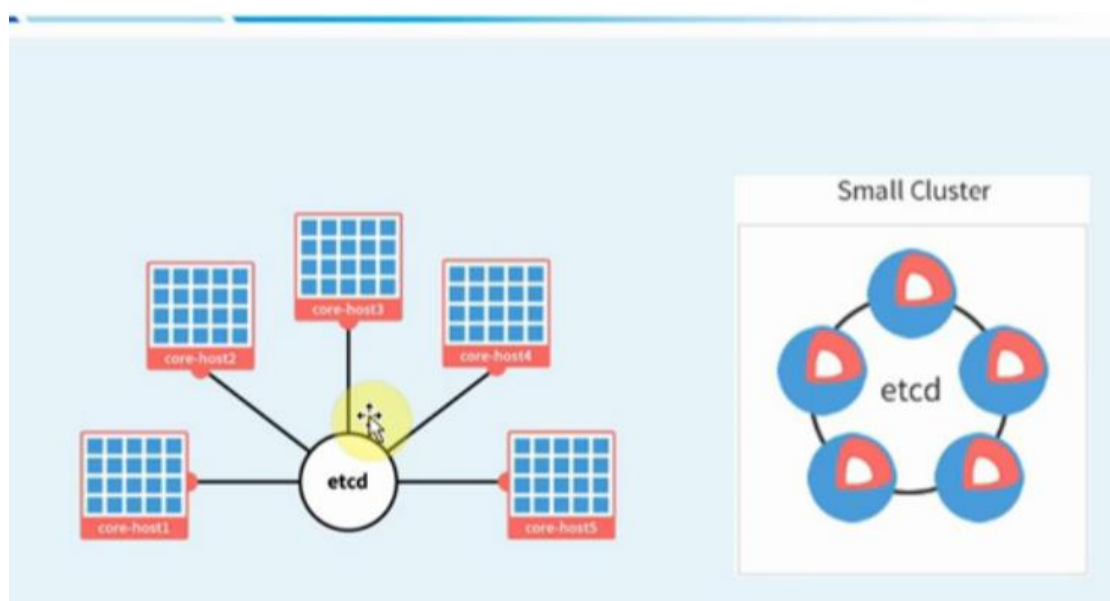
官方下载地址

<https://github.com/coreos/etcd>



下面演示的是三个节点都部署 ETCD 集群

ETCD 集群图解



1.1、准备 etcd 软件包

```
wget https://github.com/coreos/etcd/releases/download/v3.2.18/etcd-v3.2.18-linux-amd64.tar.gz
#自己的已经下载完了
[root@linux-node1 ssl]# cd /usr/local/src/
[root@linux-node1 src]# ll
total 580288
-rw-r--r-- 1 root root 17108856 Apr 12 17:35 cni-plugins-amd64-v0.7.1.tgz
-rw-r--r-- 1 root root 10562874 Mar 30 01:58 etcd-v3.2.18-linux-amd64.tar.gz
-rw-r--r-- 1 root root 9706487 Jan 24 2018 flannel-v0.10.0-linux-amd64.tar.gz
drwxr-xr-x 11 root root 211 Apr 12 11:16 kubernetes
```

```
-rw-r--r-- 1 root root 13344537 Apr 13 01:51 kubernetes-client-linux-amd64.tar.gz
-rw-r--r-- 1 root root 112427817 Apr 13 01:51 kubernetes-node-linux-amd64.tar.gz
-rw-r--r-- 1 root root 428337777 Apr 13 01:51 kubernetes-server-linux-amd64.tar.gz
-rw-r--r-- 1 root root 2716855 Apr 13 01:51 kubernetes.tar.gz
drwxr-xr-x 2 root root          93 Aug 27 05:06 ssl
[root@linux-node1 src]# tar zxf etcd-v3.2.18-linux-amd64.tar.gz
[root@linux-node1 src]# cd etcd-v3.2.18-linux-amd64
[root@linux-node1 etcd-v3.2.18-linux-amd64]# cp etcd etcdctl /opt/kubernetes/bin/
[root@linux-node1 etcd-v3.2.18-linux-amd64]# scp etcd etcdctl 192.168.56.12:/opt/kubernetes/bin/
[root@linux-node1 etcd-v3.2.18-linux-amd64]# scp etcd etcdctl 192.168.56.13:/opt/kubernetes/bin/
```

安装、配置、启动 三个步骤，到启动的时候不要着急，启动也是需要证书的。这也是手动安装比较麻烦的地方

1.2、创建 etcd 证书签名请求：

生成 etcd 证书签名请求的 json 文件

```
[root@linux-node1 opt]# cd /usr/local/src/ssl
现在的规范：所有相关证书的创建都放在 master 节点中的这个目录下，创建好之后，在 scp 到其他节点。
注释：下面的 hosts 是指定，etcd 节点的 ip 地址，正常应该只填写一个当前的 ip。
[root@linux-node1 ~]# vim etcd-csr.json  #证书签名请求，etcd 的证书签名请求
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "192.168.56.11",
    "192.168.56.12",
    "192.168.56.13"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
```

1.3、生成 etcd 证书和私钥：

生成这个证书的时候和上面的 k8s 证书关联起来了

```
[root@linux-node1 ~]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \  
-ca-key=/opt/kubernetes/ssl/ca-key.pem \  
-config=/opt/kubernetes/ssl/ca-config.json \  
-profile=kubernetes etcd-csr.json | cfssljson -bare etcd
```

会生成以下证书文件

```
[root@k8s-master ~]# ls -l etcd*      #此时 etcd 生成了三个证书文件，json 那个是上面自己  
配置的
```

```
-rw-r--r-- 1 root root 1045 Mar  5 11:27 etcd.csr  
-rw-r--r-- 1 root root  257 Mar  5 11:25 etcd-csr.json  
-rw----- 1 root root 1679 Mar  5 11:27 etcd-key.pem  
-rw-r--r-- 1 root root 1419 Mar  5 11:27 etcd.pem
```

1.4、将证书移动到/opt/kubernetes/ssl 目录下

```
cp etcd*.pem /opt/kubernetes/ssl  
scp etcd*.pem 192.168.56.12:/opt/kubernetes/ssl  
scp etcd*.pem 192.168.56.13:/opt/kubernetes/ssl  
#rm -f etcd.csr etcd-csr.json
```

1.5、设置 ETCD 配置文件

```
[root@linux-node1 ~]# vim /opt/kubernetes/cfg/etcd.conf  
#[member]  
ETCD_NAME="etcd-node1"  
#注意此名称是计算机的名称，一定不能一样  
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"  
#ETCD_SNAPSHOT_COUNTER="10000"  
#ETCD_HEARTBEAT_INTERVAL="100"  
#ETCD_ELECTION_TIMEOUT="1000"  
#监听两个端口，2380 集群之间通信用的，2379 给客户端用的。  
ETCD_LISTEN_PEER_URLS="https://192.168.56.11:2380"  
ETCD_LISTEN_CLIENT_URLS="https://192.168.56.11:2379,https://127.0.0.1:2379"  
#ETCD_MAX_SNAPSHOTS="5"  
#ETCD_MAX_WALS="5"  
#ETCD_CORS=""  
#[cluster]  
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.56.11:2380"  
# if you use different ETCD_NAME (e.g. test),  
# set ETCD_INITIAL_CLUSTER value for this name, i.e. "test=http://..."
```

```
#初始化的集群，三个节点所以三个初始化
ETCD_INITIAL_CLUSTER="etcd-node1=https://192.168.56.11:2380,etcd-
node2=https://192.168.56.12:2380,etcd-node3=https://192.168.56.13:2380"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="k8s-etcd-cluster"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.56.11:2379"
#[security]
#所有的证书
CLIENT_CERT_AUTH="true"
ETCD_CA_FILE="/opt/kubernetes/ssl/ca.pem"
ETCD_CERT_FILE="/opt/kubernetes/ssl/etcd.pem"
ETCD_KEY_FILE="/opt/kubernetes/ssl/etcd-key.pem"
PEER_CLIENT_CERT_AUTH="true"
ETCD_PEER_CA_FILE="/opt/kubernetes/ssl/ca.pem"
ETCD_PEER_CERT_FILE="/opt/kubernetes/ssl/etcd.pem"
ETCD_PEER_KEY_FILE="/opt/kubernetes/ssl/etcd-key.pem"
```

1.6、创建 ETCD 系统服务

```
[root@linux-node1 ~]# vim /etc/systemd/system/etcd.service    #服务的脚本都放在这个下
面，自己需要了解
[Unit]
Description=Etcd Server
After=network.target

[Service]
Type=simple
WorkingDirectory=/var/lib/etcd
EnvironmentFile=-/opt/kubernetes/cfg/etcd.conf
# set GOMAXPROCS to number of processors
ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /opt/kubernetes/bin/etcd"
Type=notify

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

1.7、重新加载系统服务

```
#不要着急启动服务，因为集群的通信其他节点没有完成会卡住。
scp /opt/kubernetes/cfg/etcd.conf 192.168.56.12:/opt/kubernetes/cfg/
scp /etc/systemd/system/etcd.service 192.168.56.12:/etc/systemd/system/
scp /opt/kubernetes/cfg/etcd.conf 192.168.56.13:/opt/kubernetes/cfg/
scp /etc/systemd/system/etcd.service 192.168.56.13:/etc/systemd/system/
```

注意上面配置文件，cp 之后要在对应节点改成其 ip 地址，更改四个 ip 地址和主机名称

```
anaconda-ks.cfg
[root@linux-node2 ~]# vim /opt/kubernetes/cfg/etcd.conf
#member
ETCD_NAME="etcd-node1"
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
#ETCD_SNAPSHOT_COUNTER="10000"
#ETCD_HEARTBEAT_INTERVAL="100"
#ETCD_ELECTION_TIMEOUT="1000"
ETCD_LISTEN_PEER_URLS="https://192.168.56.11:2380"
ETCD_LISTEN_CLIENT_URLS="https://192.168.56.11:2379,https://127.0.0.1:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
#ETCD_CORS=""
#[cluster]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.56.11:2380"
# if you use different ETCD_NAME (e.g. test),
# set ETCD_INITIAL_CLUSTER value for this name, i.e. "test=http://..."
ETCD_INITIAL_CLUSTER="etcd-node1=https://192.168.56.11:2380,etcd-node2=https://192.168.56.12:2380,etcd-node3=https://192.168.56.13:2380"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="k8s-etcd-cluster"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.56.11:2379"
#security
```

mkdir /var/lib/etcd #在所有节点上创建 etcd 存储目录并启动 etcd

systemctl daemon-reload #重载所有修改过的配置文件

systemctl enable etcd #加入开机自启动

systemctl start etcd

systemctl status etcd

下面需要大家在所有的 etcd 节点重复上面的步骤，直到所有机器的 etcd 服务都已启动。

检查是否启动

```
[root@linux-node1 ssl]# systemctl status etcd.service
```

```
[root@linux-node1 ssl]# netstat -lntup
```

1.8、验证集群

```
[root@linux-node1 ~]# etcdctl --endpoints=https://192.168.56.11:2379 \
```

```
--ca-file=/opt/kubernetes/ssl/ca.pem \
```

```
--cert-file=/opt/kubernetes/ssl/etcd.pem \
```

```
--key-file=/opt/kubernetes/ssl/etcd-key.pem cluster-health
```

```
member 435fb0a8da627a4c is healthy: got healthy result from https://192.168.56.12:2379
```

```
member 6566e06d7343e1bb is healthy: got healthy result from https://192.168.56.11:2379
```

```
member ce7b884e428b6c8c is healthy: got healthy result from https://192.168.56.13:2379
```

```
cluster is healthy
```

🌀 master 节点部署

需要部署三个服务

一、API Server:提供集群管理的 REST API 接口，包括认证授权，数据校验以及集群状态变更

1. 只有 API Server 才直接操作 etcd
2. 其他模块通过 API Server 查询或者修改数据
3. 提供其他模块之间的数据交互和通信的枢纽

二、Scheduler：最小单位不是容器，而是 pod，pod 里面有容器

1. 监听 kube-apiserver，查询还未分配 Node 的 Pod
2. 根据调度策略为这些 pod 分配节点

三、Controller Manager：集群说有 80 个容器，要是 70 个则加 10 个

由一系列的控制器组成，它通过 API Server 监控整个集群的状态，并确保集群处于预期的工作状态

1.1、部署 API Server 服务部署

1.1.1 准备三个服务端软件包

```
cd /usr/local/src/kubernetes
cp server/bin/kube-apiserver /opt/kubernetes/bin/
cp server/bin/kube-controller-manager /opt/kubernetes/bin/
cp server/bin/kube-scheduler /opt/kubernetes/bin/
```

1.1.2 创建生成签名请求 CSR 的 JSON 配置文件

麻烦的地方，第一步又是要创建证书

```
[root@linux-node1 src]# cd /opt/kubernetes/ssl/
[root@linux-node1 src]# vim kubernetes-csr.json
#疑惑，这个 10.1.0.1 的 ip 是干嘛的？
{
  "CN": "kubernetes",
  "hosts": [
    "127.0.0.1",
    "192.168.56.11",
    "10.1.0.1",
    "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": "System"
  }
]
```

1.1.3 生成 kubernetes 证书和私钥

```
[root@linux-node1 src]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \
-ca-key=/opt/kubernetes/ssl/ca-key.pem \
-config=/opt/kubernetes/ssl/ca-config.json \
-profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes
#
scp kubernetes*.pem 192.168.56.12:/opt/kubernetes/ssl/
scp kubernetes*.pem 192.168.56.13:/opt/kubernetes/ssl/
```

1.1.4 创建 kube-apiserver 使用的客户端 token 文件

疑问：这一步和下一步的密码都是干嘛的？

```
[root@linux-node1 ~]# head -c 16 /dev/urandom | od -An -t x | tr -d ' '
ad6d5bb607a186796d8861557df0d17f
[root@linux-node1 ~]# vim /opt/kubernetes/ssl/bootstrap-token.csv
ad6d5bb607a186796d8861557df0d17f,kubelet-bootstrap,10001,"system:kubelet-bootstrap"
```

1.1.5 创建基础用户名/密码认证配置

```
[root@linux-node1 ~]# vim /opt/kubernetes/ssl/basic-auth.csv
admin,admin,1
readonly,readonly,2
```

1.1.6 部署 Kubernetes API Server

这个是部署服务，没有配置文件，因为通过各种参数都传进去了

```
[root@linux-node1 ~]# vim /usr/lib/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
ExecStart=/opt/kubernetes/bin/kube-apiserver \
  --admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,R
esourceQuota,NodeRestriction \
  --bind-address=192.168.56.11 \
  --insecure-bind-address=127.0.0.1 \
  --authorization-mode=Node,RBAC \
  --runtime-config=rbac.authorization.k8s.io/v1 \
  --kubelet-https=true \
  --anonymous-auth=false \
  --basic-auth-file=/opt/kubernetes/ssl/basic-auth.csv \
  --enable-bootstrap-token-auth \
  --token-auth-file=/opt/kubernetes/ssl/bootstrap-token.csv \
  --service-cluster-ip-range=10.1.0.0/16 \
  --service-node-port-range=20000-40000 \
  --tls-cert-file=/opt/kubernetes/ssl/kubernetes.pem \
  --tls-private-key-file=/opt/kubernetes/ssl/kubernetes-key.pem \
  --client-ca-file=/opt/kubernetes/ssl/ca.pem \
  --service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \
  --etcd-cafile=/opt/kubernetes/ssl/ca.pem \
  --etcd-certfile=/opt/kubernetes/ssl/kubernetes.pem \
  --etcd-keyfile=/opt/kubernetes/ssl/kubernetes-key.pem \
  --etcd-
servers=https://192.168.56.11:2379,https://192.168.56.12:2379,https://192.1
68.56.13:2379 \
  --enable-swagger-ui=true \
  --allow-privileged=true \
  --audit-log-maxage=30 \
  --audit-log-maxbackup=3 \
  --audit-log-maxsize=100 \
  --audit-log-path=/opt/kubernetes/log/api-audit.log \
  --event-ttl=1h \
  --v=2 \
  --logtostderr=false \
  --log-dir=/opt/kubernetes/log
Restart=on-failure
```

```
RestartSec=5
Type=notify
LimitNOFILE=65536

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

1.1.7 启动 API Server 服务

```
systemctl daemon-reload
systemctl enable kube-apiserver
systemctl start kube-apiserver
systemctl status kube-apiserver
#API server 监听的是 6443 端口，同时还监听了本地的 8080，这个是给 controller manager 和
kube**什么玩意服务用的
```

1.2、部署 Controller Manager 服务

1.2.1 配置服务

```
[root@linux-node1 ~]# vim /usr/lib/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/opt/kubernetes/bin/kube-controller-manager \
  --address=127.0.0.1 \
  --master=http://127.0.0.1:8080 \
  --allocate-node-cidrs=true \
  --service-cluster-ip-range=10.1.0.0/16 \
  --cluster-cidr=10.2.0.0/16 \
  --cluster-name=kubernetes \
  --cluster-signing-cert-file=/opt/kubernetes/ssl/ca.pem \
  --cluster-signing-key-file=/opt/kubernetes/ssl/ca-key.pem \
  --service-account-private-key-file=/opt/kubernetes/ssl/ca-key.pem \
  --root-ca-file=/opt/kubernetes/ssl/ca.pem \
  --leader-elect=true \
  --v=2 \
  --logtostderr=false \
  --log-dir=/opt/kubernetes/log
```

```
Restart=on-failure
RestartSec=5

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

1.2.2 启动 Controller Manager

```
systemctl daemon-reload
systemctl enable kube-controller-manager
systemctl start kube-controller-manager
监听端口：127.0.0.1:10252
```

1.2.3 查看服务状态

```
[root@linux-node1 scripts]# systemctl status kube-controller-manager
```

1.3、部署 Scheduler 服务

1.3.1 配置服务

```
[root@linux-node1 ~]# vim /usr/lib/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/opt/kubernetes/bin/kube-scheduler \
  --address=127.0.0.1 \
  --master=http://127.0.0.1:8080 \
  --leader-elect=true \
  --v=2 \
  --logtostderr=false \
  --log-dir=/opt/kubernetes/log

Restart=on-failure
RestartSec=5

[Install]
```

```
WantedBy=multi-user.target
```

1.3.2 部署服务

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

1.3.3 检查服务

监听 120251 端口

```
systemctl status kube-scheduler
```

1.4、部署 kubectl 命令行工具

kubectl 来管理 apiserver

1.4.1 生成证书和密钥

1.准备二进制命令包

```
cd /usr/local/src/kubernetes/client/bin
cp kubectl /opt/kubernetes/bin/
```

2.创建 admin 证书签名请求

#与 apiserver 通信，需要证书所以，这个 kubectl 也显得比较麻烦

```
[root@linux-node1 ~]# cd /usr/local/src/ssl/
```

```
[root@linux-node1 ssl]# vim admin-csr.json
```

```
{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
```

```
"O": "system:masters",
"OU": "System"
}
]
}
```

3.生成 admin 证书和私钥：

```
cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \
  -ca-key=/opt/kubernetes/ssl/ca-key.pem \
  -config=/opt/kubernetes/ssl/ca-config.json \
  -profile=kubernetes admin-csr.json | cfssljson -bare admin
[root@linux-node1 ssl]# ls -l admin*
-rw-r--r-- 1 root root 1009 Mar  5 12:29 admin.csr
-rw-r--r-- 1 root root 229 Mar  5 12:28 admin-csr.json
-rw----- 1 root root 1675 Mar  5 12:29 admin-key.pem
-rw-r--r-- 1 root root 1399 Mar  5 12:29 admin.pem

[root@linux-node1 src]# cp admin*.pem /opt/kubernetes/ssl/
```

1.4.2 生成 kubectl 与 apiserver 通信文件

1、设置集群参数

因为要和 apiserver 的通信，而 apiserver 的验证方法预定义了一些角色。所以要把角色相关信息设置上

```
[root@linux-node1 src]# kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=https://192.168.56.11:6443
Cluster "kubernetes" set.
```

2.设置客户端认证参数

```
[root@linux-node1 src]# kubectl config set-credentials admin \
  --client-certificate=/opt/kubernetes/ssl/admin.pem \
  --embed-certs=true \
  --client-key=/opt/kubernetes/ssl/admin-key.pem
User "admin" set.
```

3.设置上下文参数

```
[root@linux-node1 src]# kubectl config set-context kubernetes \
  --cluster=kubernetes \
  --user=admin
Context "kubernetes" created.
```

4.设置默认上下文

```
[root@linux-node1 src]# kubectl config use-context kubernetes
Switched to context "kubernetes".
```

上面这个一大堆实际上就是在一个文件中生成一大堆验证字符，文件目录如下

```
[root@linux-node1 ~]# cat .kube/config
kubectl 与 apiserver 通信用的就是这个文件，其他节点如果希望通信把这个 copy 到对应即可
```

1.4.3 验证 kubectl

使用 kubectl 工具

```
[root@linux-node1 ~]# kubectl get cs
```

NAME	STATUS	MESSAGE	ERROR
controller-manager	Healthy	ok	
scheduler	Healthy	ok	
etcd-1	Healthy	{"health":"true"}	
etcd-2	Healthy	{"health":"true"}	
etcd-0	Healthy	{"health":"true"}	

Node 节点部署

1、部署 kubelet

1.1、二进制包准备 将软件包从 linux-node1 复制到其他节点

```
cd /usr/local/src/kubernetes/server/bin/
cp kubelet kube-proxy /opt/kubernetes/bin/
scp kubelet kube-proxy 192.168.56.12:/opt/kubernetes/bin/
scp kubelet kube-proxy 192.168.56.13:/opt/kubernetes/bin/
```

1.2、创建角色绑定

kubectl 启动时候要和 server api 通信，访问的时候 apiserver 会自动给它分配证书

```
[root@linux-node1 ~]# kubectl create clusterrolebinding kubelet-bootstrap --
clusterrole=system:node-bootstrapper --user=kubelet-bootstrap
```

1.3、创建 bootstrap.kubeconfig 配置文件

下面所有的步骤都是填充 bootstrap.kubeconfig 这个文件，生成好之后把它 copy 到对应的节点中

1、设置集群参数

```
[root@linux-node1 ssl]# cd /opt/kubernetes/ssl/
[root@linux-node1 ~]# kubectl config set-cluster kubernetes \
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=https://192.168.56.11:6443 \
  --kubeconfig=bootstrap.kubeconfig
```

2、设置客户端认证参数

```
[root@linux-node1 ~]# kubectl config set-credentials kubelet-bootstrap \
  --token=ad6d5bb607a186796d8861557df0d17f \
  --kubeconfig=bootstrap.kubeconfig
User "kubelet-bootstrap" set.
```

注意：

这个 token 是不是 apiserver 时候创建的 token，

查看创建的 token 位置， `cat /usr/lib/systemd/system/kube-apiserver.service`

```
authorization-mode=Node,RBAC \
--runtime-config=rbac.authorization.k8s.io/v1 \
--kubelet-https=true \
--anonymous-auth=false \
--basic-auth-file=/opt/kubernetes/ssl/basic-auth.csv \
--enable-bootstrap-token-auth \
--token-auth-file=/opt/kubernetes/ssl/bootstrap-token.csv \
--service-cluster-ip-range=10.1.0.0/16 \
--service-node-port-range=20000-40000 \
--tls-cert-file=/opt/kubernetes/ssl/kubernetes.pem \
--tls-private-key-file=/opt/kubernetes/ssl/kubernetes-key.pem \
--client-ca-file=/opt/kubernetes/ssl/ca.pem \
--service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \
--etcd-cafile=/opt/kubernetes/ssl/ca.pem \
--etcd-certfile=/opt/kubernetes/ssl/kubernetes.pem \
--etcd-keyfile=/opt/kubernetes/ssl/kubernetes-key.pem \
```

3、设置上下文参数

```
[root@linux-node1 ~]# kubectl config set-context default \
  --cluster=kubernetes \
  --user=kubelet-bootstrap \
  --kubeconfig=bootstrap.kubeconfig
Context "default" created.
```

4、选择默认上下文

```
[root@linux-node1 ~]# kubectl config use-context default --kubeconfig=bootstrap.kubeconfig
```

```
[root@linux-node1 ssl]# cat bootstrap.kubeconfig 上面的一堆命令就是为了生成这个文件
```

1.4、配置文件 copy 到对应 node 节点

```
cp bootstrap.kubeconfig /opt/kubernetes/cfg
scp bootstrap.kubeconfig 192.168.56.12:/opt/kubernetes/cfg
scp bootstrap.kubeconfig 192.168.56.13:/opt/kubernetes/cfg
```

1.5、部署 kubelet

1.5.1 设置 CNI 支持

注释：cni 是 k8s 网络接口的一个插件

```
[root@linux-node2 ~]# mkdir -p /etc/cni/net.d          #全部节点都要创建
[root@linux-node2 ~]# vim /etc/cni/net.d/10-default.conf
{
    "name": "flannel",
    "type": "flannel",
    "delegate": {
        "bridge": "docker0",
        "isDefaultGateway": true,
        "mtu": 1400
    }
}
scp /etc/cni/net.d/10-default.conf 192.168.56.12:/etc/cni/net.d/10-default.conf
scp /etc/cni/net.d/10-default.conf 192.168.56.13:/etc/cni/net.d/10-default.conf
```

创建 kubelet 目录

为了 kubelet 的数据放在这个下面
mkdir /var/lib/kubelet #所有节点创建

1.5.2 创建 kubelet 服务

注意下面 ip 地址的修改，犯错没有改就启动服务。

```
[root@k8s-node2 ~]# vim /usr/lib/systemd/system/kubelet.service
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service
```



```
[Service]
WorkingDirectory=/var/lib/kubelet
ExecStart=/opt/kubernetes/bin/kubelet \
  --address=192.168.56.12 \
  --hostname-override=192.168.56.12 \
  --pod-infra-container-image=mirrorgooglecontainers/pause-amd64:3.0 \
  --experimental-bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \
  --kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \
  --cert-dir=/opt/kubernetes/ssl \
  --network-plugin=cni \
  --cni-conf-dir=/etc/cni/net.d \
  --cni-bin-dir=/opt/kubernetes/bin/cni \
  --cluster-dns=10.1.0.2 \
  --cluster-domain=cluster.local. \
  --hairpin-mode hairpin-veth \
  --allow-privileged=true \
  --fail-swap-on=false \
  --logtostderr=true \
  --v=2 \
  --logtostderr=false \
  --log-dir=/opt/kubernetes/log
Restart=on-failure
RestartSec=5
```

1.5.3 启动 Kubelet

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

1.5.4 其他节点配置

配置文件复制到其他节点上去，记得改监听的 ip 地址

```
scp /usr/lib/systemd/system/kubelet.service 192.168.56.12:/usr/lib/systemd/system/kubelet.service
scp /usr/lib/systemd/system/kubelet.service 192.168.56.13:/usr/lib/systemd/system/kubelet.service
```

验证

```
systemctl daemon-reload
```

```
systemctl enable kubelet
systemctl start kubelet
systemctl status kubelet
```

1.6、验证 kubelet 状态

查看 csr 请求 注意是在 linux-node1 上执行。

```
[root@linux-node1 ~]# kubectl get csr #会有证书的请求，可以查看到对应节点，之后下面
在进行批准
```

NAME	REQUESTOR	CONDITION	AGE	
node-csr-0_w5F1FM_la_SeGiu3Y5xELRpYUjjT2icIFk9gO9KOU	bootstrap	Pending	1m	kubelet-

批准 kubelet 的 TLS 证书请求

```
[root@linux-node1 ~]# kubectl get csr | grep 'Pending' | awk 'NR>0{print $1}' | xargs kubectl
certificate approve
kubectl get csr #此时命令已经是批准状态
之后 ssl 下面会多一个证书文件
```

```
[root@linux-node1 ssl]# kubectl get node NAME STATUS ROLES AGE VERSION
Error from server (NotFound): nodes "NAME" not found
[root@linux-node1 ssl]# ll
total 76
-rw----- 1 root root 1675 Aug 26 03:41 admin-key.pem
-rw-r--r-- 1 root root 1399 Aug 26 03:41 admin.pem
-rw-r--r-- 1 root root 35 Aug 26 03:14 basic-auth.csv
-rw----- 1 root root 2167 Aug 26 08:31 bootstrap.kubeconfig
-rw-r--r-- 1 root root 84 Aug 26 03:05 bootstrap-token.csv
-rw-r--r-- 1 root root 291 Aug 25 12:06 ca-config.json
-rw-r--r-- 1 root root 1001 Aug 25 12:06 ca.csr
-rw----- 1 root root 1675 Aug 25 12:06 ca-key.pem
-rw-r--r-- 1 root root 1359 Aug 25 12:06 ca.pem
-rw----- 1 root root 1675 Aug 25 14:28 etcd-key.pem
-rw-r--r-- 1 root root 1436 Aug 25 14:28 etcd.pem
-rw-r--r-- 1 root root 1046 Aug 26 09:22 kubelet-client.crt
-rw----- 1 root root 227 Aug 26 09:08 kubelet-client.key
-rw-r--r-- 1 root root 2185 Aug 26 09:08 kubelet.crt
-rw-r--r-- 1 root root 1679 Aug 26 09:08 kubelet.key
-rw-r--r-- 1 root root 1245 Aug 26 02:57 kubernetes.csr
-rw-r--r-- 1 root root 436 Aug 26 02:56 kubernetes-csr.json
-rw----- 1 root root 1675 Aug 26 02:57 kubernetes-key.pem
-rw-r--r-- 1 root root 1610 Aug 26 02:57 kubernetes.pem
[root@linux-node1 ssl]#
```

执行完毕后，查看节点状态已经是 Ready 的状态了

```
[root@linux-node1 ssl]# kubectl get node 用这个命令查看状态
```

```
[root@linux-node1 ssl]# kubectl get node
```

NAME	STATUS	ROLES	AGE	VERSION
------	--------	-------	-----	---------

192.168.56.11	Ready	<none>	5m	v1.10.1
192.168.56.13	Ready	<none>	5m	v1.10.1

2、部署 Kubernetes Proxy

2.1.1 1.配置 kube-proxy 使用 LVS

```
[root@linux-node2 ~]# yum install -y ipvsadm ipset conntrack
```

2.1.2 生成证书

创建 kube-proxy 证书请求

```
[root@linux-node1 ~]# cd /usr/local/src/ssl/
[root@linux-node1 ~]# vim kube-proxy-csr.json #首先证书签名的请求配置文件
{
  "CN": "system:kube-proxy",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
```

生成证书

```
[root@linux-node1~]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \
  -ca-key=/opt/kubernetes/ssl/ca-key.pem \
  -config=/opt/kubernetes/ssl/ca-config.json \
  -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy
```

分发证书到所有 Node 节点

```
cp kube-proxy*.pem /opt/kubernetes/ssl/
```

```
scp kube-proxy*.pem 192.168.56.12:/opt/kubernetes/ssl/  
scp kube-proxy*.pem 192.168.56.13:/opt/kubernetes/ssl/
```

2.1.3 创建 kube-proxy 配置文件

下面执行这么多就是为了生成 kube-proxy.kubeconfig 这个配置文件而已，后面都是追加了一些配置。

```
[root@linux-node2 ~]# kubectl config set-cluster kubernetes \  
  --certificate-authority=/opt/kubernetes/ssl/ca.pem \  
  --embed-certs=true \  
  --server=https://192.168.56.11:6443 \  
  --kubeconfig=kube-proxy.kubeconfig  
Cluster "kubernetes" set.
```

```
[root@linux-node2 ~]# kubectl config set-credentials kube-proxy \  
  --client-certificate=/opt/kubernetes/ssl/kube-proxy.pem \  
  --client-key=/opt/kubernetes/ssl/kube-proxy-key.pem \  
  --embed-certs=true \  
  --kubeconfig=kube-proxy.kubeconfig  
User "kube-proxy" set.
```

```
[root@linux-node2 ~]# kubectl config set-context default \  
  --cluster=kubernetes \  
  --user=kube-proxy \  
  --kubeconfig=kube-proxy.kubeconfig  
Context "default" created.
```

```
[root@linux-node2 ~]# kubectl config use-context default --kubeconfig=kube-  
proxy.kubeconfig  
Switched to context "default".
```

分发 kubeconfig 配置文件

```
cp kube-proxy.kubeconfig /opt/kubernetes/cfg/  
scp kube-proxy.kubeconfig 192.168.56.12:/opt/kubernetes/cfg/  
scp kube-proxy.kubeconfig 192.168.56.13:/opt/kubernetes/cfg/
```

2.1.4 创建 kube-proxy 服务配置

```
[root@linux-node2 bin]# mkdir /var/lib/kube-proxy      #未来 proxy 的所有的配置文件都放  
在这里  
[root@k8s-node2 ~]# vim /usr/lib/systemd/system/kube-proxy.service  
[Unit]
```

```
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
```

[Service]

```
WorkingDirectory=/var/lib/kube-proxy
ExecStart=/opt/kubernetes/bin/kube-proxy \
    --bind-address=192.168.56.12 \
    --hostname-override=192.168.56.12 \
    --kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig \
--masquerade-all \
    --feature-gates=SupportIPVSProxyMode=true \
    --proxy-mode=ipvs \
    --ipvs-min-sync-period=5s \
    --ipvs-sync-period=5s \
    --ipvs-scheduler=rr \
    --logtostderr=true \
    --v=2 \
    --logtostderr=false \
    --log-dir=/opt/kubernetes/log
```

```
Restart=on-failure
```

```
RestartSec=5
```

```
LimitNOFILE=65536
```

[Install]

```
WantedBy=multi-user.target
```

把配置文件 **copy** 到其他 node 上

```
scp /usr/lib/systemd/system/kube-proxy.service 192.168.56.12:/usr/lib/systemd/system/kube-
proxy.service
scp /usr/lib/systemd/system/kube-proxy.service 192.168.56.13:/usr/lib/systemd/system/kube-
proxy.service
#记得修改对应节点的监听 ip
```

2.1.5 启动 Kubernetes Proxy

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

2.1.6 检查 LVS 状态

```
[root@linux-node2 ~]# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
```

```
Prot LocalAddress:Port Scheduler Flags
```

```
-> RemoteAddress:Port          Forward Weight ActiveConn InActConn
```

```
TCP    10.1.0.1:443 rr persistent 10800
```

```
-> 192.168.56.11:6443           Masq     1         0         0
```

如果你在两台实验机器都安装了 kubelet 和 proxy 服务，使用下面的命令可以检查状态：

```
[root@linux-node1 ss]# kubectl get node
```

NAME	STATUS	ROLES	AGE	VERSION
192.168.56.12	Ready	<none>	22m	v1.10.1
192.168.56.13	Ready	<none>	3m	v1.10.1

Flannel 网络部署

1、前置知识理解

1.1.1 POD

k8s 管理的是 pod，pod 里面是容器，可能多个容器。

1. pod 是需要 ip 地址的
2. 每个 pod 上面都个标签

1.1.2 RC

1. RC (replication controller) 管理 pod 的服务，RS 是 RC 的升级，直接面对用户的应用是 deployment
2. RC 是 K8S 集群中最早的保证 pod 高可用的 API 对象，通过监控运行中的 POD 来保证集群中运行指定树木的 Pod 副本。
3. 指定的数目可以是多个也可以是 1 个，少于指定数目，RC 就会启动运行新的 pod 副本，多余指定数目，RC 就会杀死多余的 POD 副本
4. 即使在指定数目为 1 的情况下，通过 RC 运行 Pod 也比直接运行 Pod 更明智，因为 RC 也可以发挥它高可用的能力，保证永远有 1 个 Pod 在运行

1.1.3 RS

1. RS 是新一代 RC，提供同样的高可用能力，区别主要在于 RS 后来居上，能支持更多中的匹配模式。副本集对象一般不单独使用，而是作为部署的理想状态参数使用。
2. 是 K8S1.2 中出现的概念，是 RC 的升级。一般和 Deployment 共同使用

1.1.4 K8S 的 IP 地址

- ✚ Node IP: 节点设备的 IP，如物理机，虚拟机等容器宿主机的实际 IP。
- ✚ Pod IP: Pod 的 IP 地址，是根据 docker0 网络 IP 段进行分配的
- ✚ Cluster IP: Service 的 IP，是一个虚拟 IP，仅作用与 Service 对象，由 K8S 管理和分配，需要结合 service port 才能使用，单独的 IP 没有通信功能，集群外访问需要一些修改
- ✚ 在 k8s 集群内部，上面的三种 IP 的通性机制是由 K8S 制定的路由规则，不是 IP 路由

1.1.5 Deployment

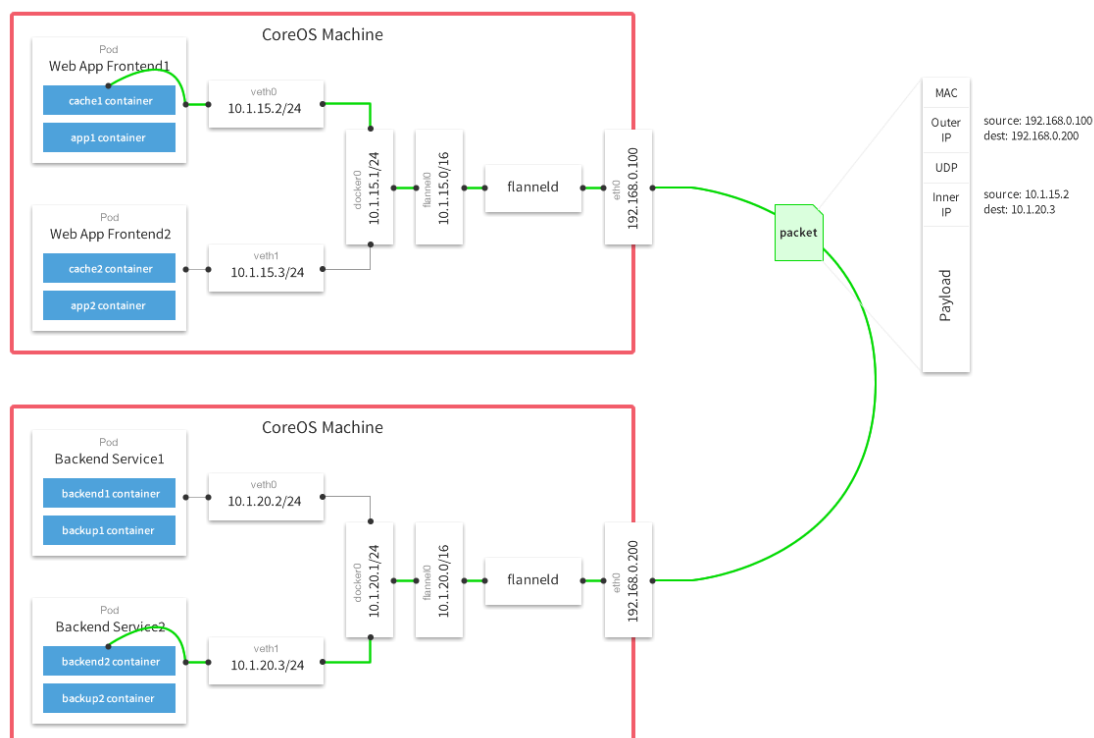
- ✚ Deployment 表示用户对 K8s 集群的一次更新操作。Deployment 是一个比 RS 应用模式更广的 API 对象，
- ✚ 可以是创建一个新的服务，更新一个新的服务，也可以是滚动升级一个服务。滚动升级一个服务，实际是创建一个新的 RS，然后逐渐将新 RS 中副本数增加到理想状态，将旧 RS 中的副本数减小到 0 的复合操作
- ✚ 这样一个复合操作用一个 RS 是不太好描述的，所以用一个更通用的 Deployment 来描述。

1.1.6 service

- ✚ RC、RS 和 Deployment 只是保证了支持服务 POD 的数量，但是没有解决如何访问这些服务的问题。一个 Pod 只是一个运行服务的实例，随时可能在一个节点上停止，在另一个节点以一个新的 IP 启动一个新的 Pod，因此不能以确定的 IP 和端口号提供服务。

- ✚ 要稳定地提供服务需要服务有发现和负载均衡能力。服务发现完成的工作，是针对客户端访问的服务，找到对应的后端服务实例。
- ✚ 在 K8S 集群中，客户端只要访问的服务就是 Service 对象。每个 Service 会对应一个集群内部有效的虚拟 IP，集群内部通过虚拟 IP 访问一个服务。

1.1.7 Flannel



2、Flannel 网络部署

2.1.1 为 Flannel 生成证书

```
[root@linux-node1 ssl]# cd /usr/local/src/ssl
```

1、创建 flanneld 证书签名请求

```
[root@linux-node1 ~]# vim flanneld-csr.json
```

```
{  
  "CN": "flanneld",  
  "hosts": [],  
  "key": {  
    "algo": "rsa",
```



```
"size": 2048
},
"names": [
  {
    "C": "CN",
    "ST": "Beijing",
    "L": "Beijing",
    "O": "k8s",
    "OU": "System"
  }
]
}
```

2.生成证书

```
[root@linux-node1 ~]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \
-ca-key=/opt/kubernetes/ssl/ca-key.pem \
-config=/opt/kubernetes/ssl/ca-config.json \
-profile=kubernetes flanneld-csr.json | cfssljson -bare flanneld
```

3.分发证书

```
cp flanneld*.pem /opt/kubernetes/ssl/
scp flanneld*.pem 192.168.56.12:/opt/kubernetes/ssl/
scp flanneld*.pem 192.168.56.13:/opt/kubernetes/ssl/
```

2.1.2 下载 Flannel 软件包

```
[root@linux-node1 ~]# cd /usr/local/src
# wget
https://github.com/coreos/flannel/releases/download/v0.10.0/flannel-v0.10.0-linux-amd64.tar.gz
tar xzf flannel-v0.10.0-linux-amd64.tar.gz
cp flanneld mk-docker-opts.sh /opt/kubernetes/bin/
```

复制到 linux-node2 节点

```
scp flanneld mk-docker-opts.sh 192.168.56.12:/opt/kubernetes/bin/
scp flanneld mk-docker-opts.sh 192.168.56.13:/opt/kubernetes/bin/
```

复制对应脚本到/opt/kubernetes/bin 目录下。#上面的脚本没有完事，还有下面这个脚本

```
cd /usr/local/src/kubernetes/cluster/centos/node/bin/
cp remove-docker0.sh /opt/kubernetes/bin/
scp remove-docker0.sh 192.168.56.12:/opt/kubernetes/bin/
scp remove-docker0.sh 192.168.56.13:/opt/kubernetes/bin/
```

2.1.3 生成 Flannel 配置文件

1、生成配置文件

```
[root@linux-node1 ~]# vim /opt/kubernetes/cfg/flannel
FLANNEL_ETCD="--etcd-
endpoints=https://192.168.56.11:2379,https://192.168.56.12:2379,https://192.168.56.13:2379"
FLANNEL_ETCD_KEY="--etcd-prefix=/kubernetes/network"
FLANNEL_ETCD_CAFILE="--etcd-cafile=/opt/kubernetes/ssl/ca.pem"
FLANNEL_ETCD_CERTFILE="--etcd-certfile=/opt/kubernetes/ssl/flanneld.pem"
FLANNEL_ETCD_KEYFILE="--etcd-keyfile=/opt/kubernetes/ssl/flanneld-key.pem"
```

2、复制配置到其它节点上

```
scp /opt/kubernetes/cfg/flannel 192.168.56.12:/opt/kubernetes/cfg/
scp /opt/kubernetes/cfg/flannel 192.168.56.13:/opt/kubernetes/cfg/
```

2.1.4 生成 Flannel 系统服务

```
[root@linux-node1 ~]# vim /usr/lib/systemd/system/flannel.service
[Unit]
Description=Flanneld overlay address etcd agent
After=network.target
Before=docker.service
[Service]
EnvironmentFile=-/opt/kubernetes/cfg/flannel
ExecStartPre=/opt/kubernetes/bin/remove-docker0.sh
ExecStart=/opt/kubernetes/bin/flanneld ${FLANNEL_ETCD} ${FLANNEL_ETCD_KEY}
${FLANNEL_ETCD_CAFILE} ${FLANNEL_ETCD_CERTFILE}
${FLANNEL_ETCD_KEYFILE}
ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -d /run/flannel/docker
Type=notify
[Install]
WantedBy=multi-user.target
RequiredBy=docker.service
```

复制系统服务脚本到其它节点上

```
scp /usr/lib/systemd/system/flannel.service 192.168.56.12:/usr/lib/systemd/system/
scp /usr/lib/systemd/system/flannel.service 192.168.56.13:/usr/lib/systemd/system/
```

2.2、Flannel CNI 集成

2.2.1 下载 CNI 插件

```
https://github.com/containernetworking/plugins/releases
wget https://github.com/containernetworking/plugins/releases/download/v0.7.1/cni-plugins-
```

```
amd64-v0.7.1.tgz
mkdir /opt/kubernetes/bin/cni #所有节点创建
cd /usr/local/src
tar xzf cni-plugins-amd64-v0.7.1.tgz -C /opt/kubernetes/bin/cni
scp -r /opt/kubernetes/bin/cni/* 192.168.56.12:/opt/kubernetes/bin/cni/
scp -r /opt/kubernetes/bin/cni/* 192.168.56.13:/opt/kubernetes/bin/cni/
```

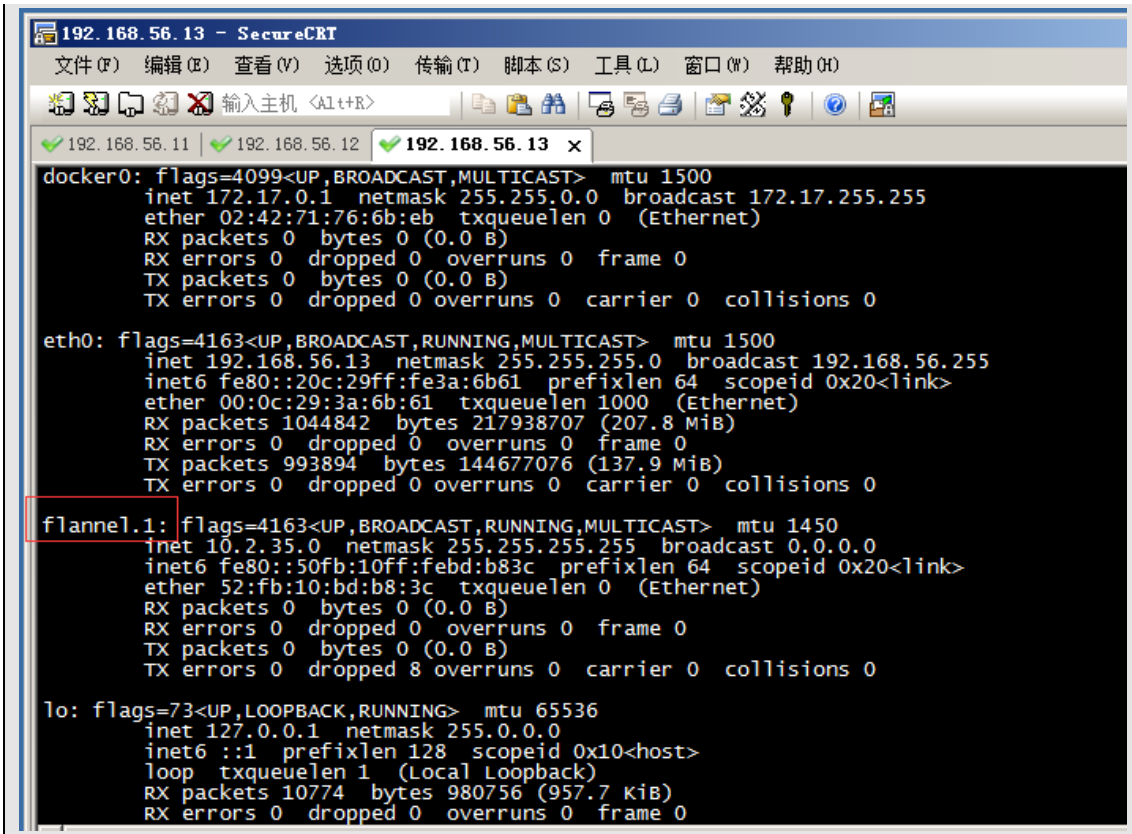
2.2.2 创建 Etcd 的 key，一个节点创建就行

```
/opt/kubernetes/bin/etcdctl --ca-file /opt/kubernetes/ssl/ca.pem --cert-file
/opt/kubernetes/ssl/flanneld.pem --key-file /opt/kubernetes/ssl/flanneld-key.pem \
--no-sync -C
https://192.168.56.11:2379,https://192.168.56.12:2379,https://192.168.56.13:2379 \
mk /kubernetes/network/config '{ "Network": "10.2.0.0/16", "Backend": { "Type": "vxlan", "VNI":
1 } }' >/dev/null 2>&1
```

2.2.3 启动 flannel

```
systemctl daemon-reload
systemctl enable flannel
chmod +x /opt/kubernetes/bin/*
systemctl start flannel
systemctl status flannel
```

每个机器都会多一个 Flannel 的网卡，并且每个网卡的网段都不一样。这就是 flannel 的作用



```
192.168.56.13 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 窗口(W) 帮助(H)
输入主机 <Alt+R>
192.168.56.11 | 192.168.56.12 | 192.168.56.13 x
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 02:42:71:76:6b:eb txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.56.13 netmask 255.255.255.0 broadcast 192.168.56.255
inet6 fe80::20c:29ff:fe3a:6b61 prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:3a:6b:61 txqueuelen 1000 (Ethernet)
RX packets 1044842 bytes 217938707 (207.8 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 993894 bytes 144677076 (137.9 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

flannel1.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
inet 10.2.35.0 netmask 255.255.255.255 broadcast 0.0.0.0
inet6 fe80::50fb:10ff:febd:b83c prefixlen 64 scopeid 0x20<link>
ether 52:fb:10:bd:b8:3c txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 8 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1 (Local Loopback)
RX packets 10774 bytes 980756 (957.7 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
```

2.2.4 配置 docker 使用 Flannel

作用：

使 flannel 先起来，docker 后起来。并且在 docker 起来之前加一个环境变量，后面会自动给 docker 生成 ip 地址

```
[root@linux-node1 ~]# vim /usr/lib/systemd/system/docker.service
```

[Unit] #在 Unit 下面修改 After 和增加 Requires

After=network-online.target firewall.service [flannel.service](#)

Wants=network-online.target

[Requires=flannel.service](#)

[Service] #增加 EnvironmentFile=-/run/flannel/docker

#注释：意思就是 flannel 一起来就会创建后面这个 docker 的文件，查看这个文件的内容就知道对应意思了。

Type=notify

[EnvironmentFile=-/run/flannel/docker](#)

ExecStart=/usr/bin/dockerd [\\$DOCKER_OPTS](#)

2.2.5 将配置复制到另外两个节点

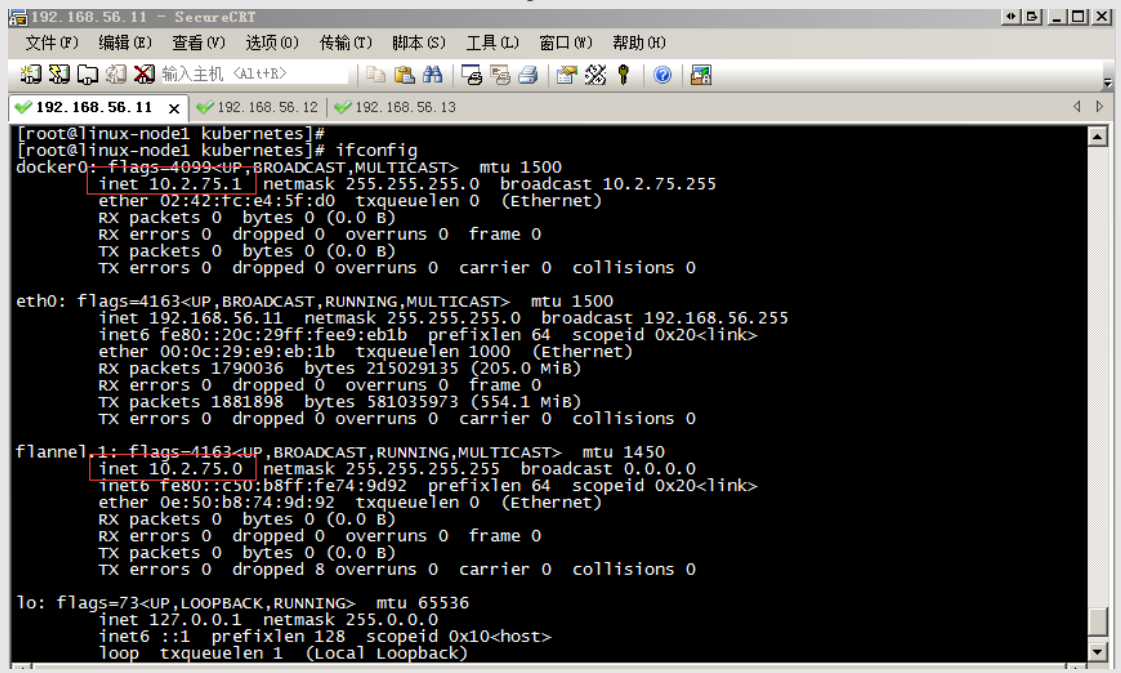
```
scp /usr/lib/systemd/system/docker.service 192.168.56.12:/usr/lib/systemd/system/  
scp /usr/lib/systemd/system/docker.service 192.168.56.13:/usr/lib/systemd/system/
```

2.2.6 重启 Docker

```
systemctl daemon-reload
```

```
systemctl restart docker
```

重启之后，flannel 就会重新分配 docker 的 ip 地址，且和 flannel 是一个网段



```
[root@linux-node1 ~]# ifconfig  
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500  
    inet 10.2.75.1 netmask 255.255.255.0 broadcast 10.2.75.255  
    ether 02:42:fc:e4:5f:d0 txqueuelen 0 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.56.11 netmask 255.255.255.0 broadcast 192.168.56.255  
    inet6 fe80::20c:29ff:fee9:eb1b prefixlen 64 scopeid 0x20<link>  
    ether 00:0c:29:e9:eb:1b txqueuelen 1000 (Ethernet)  
    RX packets 1790036 bytes 215029135 (205.0 MiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 1881898 bytes 581035973 (554.1 MiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
flannel1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450  
    inet 10.2.75.0 netmask 255.255.255.255 broadcast 0.0.0.0  
    inet6 fe80::c50:b8ff:fe74:9d92 prefixlen 64 scopeid 0x20<link>  
    ether 0e:50:b8:74:9d:92 txqueuelen 0 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 8 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1 (Local Loopback)
```

验证 k8s 集群

1.1、简单测试

1.1.1 创建一个测试用的 deployment

```
[root@linux-node1 ~]# kubectl run net-test --image=alpine --replicas=2 sleep 360000
```

1.1.2 查看获取 IP 情况

```
[root@linux-node1 ~]# kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE					

net-test-74f45db489-gmgv8	1/1	Running	0	1m	10.2.83.2
192.168.56.13					
net-test-74f45db489-pr5jc	1/1	Running	0	1m	10.2.59.2
192.168.56.12					

1.1.3 测试联通性

```
ping 10.2.83.2
```

1.2、nginx 案例

1.2.1 创建 deployment.yaml 文件

注释直接 copy 这个文件内容，会多出很多井号注释，自己采用的办法是把对应文本上次系统中

```
[root@linux-node1 kubernetes]# vim nginx-deployment.yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2 # tells deployment to run 2 pods matching the template
  template: # create pods using pod definition in this template
    metadata:
      # unlike pod-nginx.yaml, the name is not included in the meta data as a unique name is
      # generated from the deployment name
    labels:
      app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

1.2.2 命令演示

1. kubectl describe deployment nginx-deployment #查看 nginx-deployment 文件描述
2. kubectl get pods -l app=nginx #显示 nginx 的 pod

3. `kubectl get pod` #显示全部 pod, `kubectl get pod -o wide` 这个命令更加详细
4. `kubectl describe pod nginx-deployment-75675f5897-w7vqw` #查看镜像的当前详情, 比如失败, 当前动作等
5. `curl --head http://10.2.35.6` #测试是否可以访问
6. `kubectl set image deployment/nginx-deployment nginx=nginx:1.12.2 --record` #更新 nginx 镜像, --record 的意思是记录日志
7. `kubectl get deployment -o wide` #查看更新后的 deployment
8. `kubectl rollout history deployment/nginx-deployment` #查看更新历史
9. `kubectl rollout history deployment/nginx-deployment --revision=1` #查看具体某一版本的升级历史
10. `kubectl rollout undo deployment/nginx-deployment` #快速回滚到上一个版本

1.2.3 service.yaml 创建

创建 vip, pod 的 ip 是经常变化的。vip 是 lvs 的负载 vip

```
[root@linux-node1 kubernetes]# cat nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
kubectl create -f nginx-service.yaml
kubectl get service
ipvsadm -Ln
curl --head http://10.1.135.124 #因为 node1 节点没有安装 proxy 网络所以不能访问, node2 就可以
kubectl scale deployment nginx-deployment --replicas 5 #扩容五个节点, 一条命令搞定。
ipvsadm -Ln #此时查看 vip 对应五个节点, 所以日常做的就是做镜像, 编写 deployment.yaml 和 service.yaml
```



```
UtYWNjb3VudC51aWQiOiI0MGViZTA4OC1hYjYzLTExZTgtOTJhMy0wMDBjMjllOWViMWIiL
CjzdWlI0ijzeXN0ZW06c2VydmljZWFiY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.1jlw
TGeNnqKkedmZpSW4pozmpEHLAEsBB5yvXJVDI-
N_5TQZ4uI4YpxJiwi3JpfUAw_D8ttGTD50XmnOi2kjcgpVVKbs-
k2yKwdbbh5zq0LmEzSRHGCZm1JgJxKz3lf2NXIEAS_vI5zzF1WshckEnLU9NTOLTsHmfCJOhof
DnNR7o3TsBz7AAH-LdJb-EiwPcVJHOepFEOQoRB1RQvEZhLBrrd_s9rR9P_OWTaIR-
HxO7hUg_x-
FwW9q5p080gN0s39n4CVheW9OGaOW_gYG9hPr08cSwLmBIu3wjBttELlMDWReY6kwXgAHC
W9fY-IWDVZX_KhjuH-PZ4TtBpFe9bg
```

验证

```
[root@linux-node1 src]# kubectl get pod -n kube-system
NAME                                READY    STATUS    RESTARTS
AGE
kubernetes-dashboard-66c9d98865-twjjr  1/1      Running   0          8m
#查看系统日志，有个 down 镜像的过程要等一会
[root@linux-node1 src]# kubectl logs pod/kubernetes-dashboard-66c9d98865-twjjr -n kube-system
[root@linux-node1 src]# kubectl get service -n kube-system #查看 service
NAME                                TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)
AGE
kubernetes-dashboard               NodePort       10.1.92.75    <none>          443:28419/TCP  11m
```