Kubernetes

注解

Dolphin

EDITED BY XIAOQIANG JIANG

HTTP://JIANGXIAOQIANG.GITHUB.COM/

Version 21:06, September 22, 2019

Contents

参考资料https://zh.wikisource.org/zh/%E8%A9%A9%E7%B6%93:

风

1. Kubernetes 基础

1.1 Kubernetes 部署

部署 Kubernetes 最难的点是获取部署文件，这个问题解决了，其他问题就都顺利了。Kubernetes 是 Google 基于 Borg 开源的容器编排调度引擎，作为 CNCF（Cloud Native Computing Foundation）最重要的组件之一，它的目标不仅仅是一个编排系统，而是提供一个规范，可以让你来描述集群的架构，定义服务的最终状态，Kubernetes 可以帮你将系统自动地达到和维持在这个状态。安装步骤来源于 handbook[1]。后来发现 Kubernetes 官方文档[2]描述更详细、更清晰。还可以参考 Kubernetes 交互式部署手册[3]。

定义整个 kubernetes 集群都需要使用到的环境变量 (Environment Variable)[4]：

[1]参考地址：https://jimmysong.io/kubernetes-handbook/practice/create-tls-and-secret-key.html

[2]https://kubernetes.io/zh/docs/concepts/cluster-administration/certificates/

[3]https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-interactive/

[4]来源地址：https://github.com/opsnull/follow-me-install-kubernetes-cluster

Table 1.1: 组件分布

主机	角色	组件
172.19.104.231	k8s master	etcd，kube-apiserver，kube-controller-manager，kube-scheduler
172.19.104.230	k8s node01	docker，kubelet，kube-proxy
172.19.150.82	k8s node02	docker，kubelet，kube-proxy

```
#!/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=(172.19.104.231 172.19.104.230 172.19.150.82)

# 集群各 IP 对应的主机名数组
# Azshara 是游戏《魔兽世界》中的角色
export NODE_NAMES=(azshara-k8s01 azshara-k8s02 azshara-k8s03)

# etcd 集群服务地址列表
export ETCD_ENDPOINTS="https://172.19.104.231:2379,https
        ://172.19.104.230:2379,https://172.19.150.82:2379"

# etcd 集群间通信的 IP 和端口
export ETCD_NODES="azshara-k8s01=https://172.19.104.231:2380,
        azshara-k8s02=https://172.19.104.230:2380,azshara-k8s03=
        https://172.19.150.82:2380"

# kube-apiserver 的反向代理 (kube-nginx) 地址端口
export KUBE_APISERVER="https://127.0.0.1:8443"
```

```
21
22 # 节点间互联网络接口名称
23 export IFACE="eth0"
24
25 # etcd 数据目录
26 export ETCD_DATA_DIR="/opt/k8s/etcd/data"
27
28 # etcd WAL 目录，建议是 SSD 磁盘分区
29 # 或者和 ETCD_DATA_DIR 不同的磁盘分区
30 export ETCD_WAL_DIR="/opt/k8s/etcd/wal"
31
32 # k8s 各组件数据目录
33 export K8S_DIR="/opt/k8s/k8s"
34
35 # docker 数据目录
36 export DOCKER_DIR="/opt/k8s/docker"
37
38 # 以下参数一般不需要修改
39
40 # TLS Bootstrapping 使用的 Token
41 # 可以使用命令 head -c 16 /dev/urandom|od -An -t x|tr -d ' ' 生成
42 BOOTSTRAP_TOKEN="41f7e4ba8b7be874fcff18bf5cf41a7c"
43
44 # 最好使用当前未用的网段来定义服务网段和 Pod 网段
45
46 # 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)
47 SERVICE_CIDR="10.254.0.0/16"
48
49 # Pod 网段，建议/16 段地址
50 # 部署前路由不可达，部署后集群内路由可达 (flanneld 保证)
51 CLUSTER_CIDR="172.30.0.0/16"
52
53 # 服务端口范围 (NodePort Range)
```

```
export NODE_PORT_RANGE="30000-32767"

# 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
```

这里设置数据目录时，如果是多主机挂载的同一个磁盘，那么数据目录就不能够放在这个挂载的磁盘下，原因很简单，后面等于多个主机的数据放在一个目录下，会冲突，除非确定这个数据只有一个主机会用到。因为用阿里云挂载的数据盘很大，开始时，将 etcd 的数据文件放在 data 目录 (data 目录挂载的一个 500GB 的数据盘) 下，深深的为自己的高瞻远瞩和机制折服，后面 etcd 启动的时候总是提示文件锁定，更奇怪的是启动了 a 节点，b 节点宕掉了，百撕不得其解。还好反应过来了，否则不知道要掉坑里多久。调整环境变量中的路径，重新运行脚本生成服务即可。

创建 TLS 证书和秘钥

安装证书创建工具 CFSSL

生成证书可以通过 easyrsa、openssl 或 cfssl 生成证书。创建传输层安全 TLS(Transport Layer Security) 证书和密钥在服务器 172.19.104.231 上进行。直接使用二进制源码包安装 CFSSL(CloudFlare Secure Sockets Layer):

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

```
4
5  wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
6  chmod +x cfssljson_linux-amd64
7  mv cfssljson_linux-amd64 /usr/local/bin/cfssljson
8
9  wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
10 chmod +x cfssl-certinfo_linux-amd64
11 mv cfssl-certinfo_linux-amd64 /usr/local/bin/cfssl-certinfo
12
13 git clone https://github.com/cloudflare/cfssl.git $GOPATH/src/
       github.com/cloudflare/cfssl
14
15 export PATH=/usr/local/bin:$PATH
```

遗憾的是，1.2 版本貌似有 bug[5]。在后续使用 1.2 版本 cfssl 工具生成 Kubernetes 证书时提示错误:This certificate lacks a "hosts" field. This makes it unsuitable for websites. For more information see the Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.1.6, from the CA/Browser Forum (https://cabforum.org); specifically, section 10.2.3 ("Information Requirements"). 避免此问题，可升级 cfssl 工具的版本，使用如下命令升级：

```
1  # 下载 binary 包
2  wget -c https://dl.google.com/go/go1.12.9.linux-amd64.tar.gz
3  # 安装 go
4  tar -C /usr/local -xzf go1.12.9.linux-amd64.tar.gz
5  # 升级 cfssl 到最新版本
6  # 此命令下载、编译、安装 cfssl 工具
7  # 安装路径在 GOPATH/bin/cfssl
8  /usr/local/go/bin/go get -u github.com/cloudflare/cfssl/cmd/cfssl
9  # 查看 cfssl 版本
10 /root/go/bin/cfssl version
```

[5]参考 Issue：https://github.com/cloudflare/cfssl/issues/717

需要注意到细节是，截至到 2019 年 8 月 25 日，采用 go 命令安装 cfssl 需要 go 到版本是 1.12+，使用默认的 yum 仓库的 go 之前，预先看一看版本是否符合要求。

生成 CA 证书和私钥

CA 证书是根证书 (Root Certificate)，一般网站都希望用户知道他们建立的网络通道是安全的，所以会向 CA 机构购买证书来验证 domain，所以我们也可以在很多 HTTPS 的网页地址栏看到一把小绿锁。然而在一些情况下，我们没必要去 CA 机构购买证书，比如在内网的测试环境中，在此次内网部署 Kubernetes 集群中，为了验证 HTTPS 下的一些问题，不需要部署昂贵的证书，这个时候自建 Root CA，给自己颁发证书就可以搞事情[6]。

创建 CA 根证书配置文件

输入如下命令创建 CA 根证书 (Root Certificate) 配置文件：

```
mkdir /root/ssl
cd /root/ssl
cfssl print-defaults config > config.json
cfssl print-defaults csr > csr.json
# 根据config.json文件的格式创建如下的ca-config.json文件
# 过期时间设置成了 87600h
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
            "signing",
            "key encipherment",
            "server auth",
```

[6]文字来源小胡子哥的博客:https://www.barretlee.com/blog/2016/04/24/detail-about-ca-and-certs/。就我所知，目前除了阮一峰老师、陈浩老师的博客，小胡子的博客文章质量上乘。

```
            "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 证书中 CA=TRUE；
- server auth：表示 client 可以用该 CA 对 server 提供的证书进行验证；
- client auth：表示 server 可以用该 CA 对 client 提供的证书进行验证；

创建 CA 证书签名请求

创建 ca-csr.json 文件，内容如下：

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

```
        "expiry": "87600h"
    }
}
```

algo 是算法 (algorithm) 的缩写，表示 CA 证书使用的什么加密算法，这里指定的是公钥加密算法，也叫做非对称加密算法，RSA 是设计算法的三位数学家 Rivest、Shamir 和 Adleman 名字英文首字母缩写。"CN"：Common Name，kube-apiserver 从证书中提取该字段作为请求的用户名 (User Name)；浏览器使用该字段验证网站是否合法；"O"：Organization，kube-apiserver 从证书中提取该字段作为请求用户所属的组 (Group)；输入如下命令生成证书和私钥：

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

生成的文件如下表所示：

Table 1.2: CA 证书文件列表

文件名称	作用	备注
ca-config.json	CA 证书配置文件	
ca.csr	CA 证书	
ca-csr.json	CA 证书签名请求	
ca-key.pem	CA 证书私钥	
ca.pem	CA 证书公钥	

创建 Kubernetes 证书

创建证书的操作在 Master 主机上执行，这里使用的 Master 主机是 172.19.104.231。创建 Kubernetes 证书签名请求 (Certificate Sign Request) 文件 kubernetes-csr.json：

```
{
    "CN": "kubernetes",
    "hosts": [
      "127.0.0.1",
      "172.19.104.230",
```

```
      "172.19.150.82",
      "172.19.104.231"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "ST": "BeiJing",
            "L": "BeiJing",
            "O": "k8s",
            "OU": "System"
        }
    ]
}
```

172.19.104.230 为 Master IP，172.19.150.82 为 Cluster Master IP，是 API 服务器的服务集群 IP。运行如下命令生成 Kubernetes 证书和私钥，签发证书使用的是上一步生成的根证书 (Root Certificate)：

```
/root/go/bin/cfssl gencert -ca=ca.pem -ca-key=ca-key.pem \
-config=ca-config.json \
-profile=kubernetes \
kubernetes-csr.json | cfssljson -bare kubernetes
```

创建 admin 证书

创建 admin 证书签名请求文件 admin-csr.json：

```
{
  "CN": "admin",
  "hosts": [],
```

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

生成 admin 证书和私钥：

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

创建 kube-proxy 证书

创建 kube-proxy 证书签名请求文件 kube-proxy-csr.json：

```
{
  "CN": "system:kube-proxy",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
```

```
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
```

生成 kube-proxy 客户端证书和私钥:

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

将生成的证书和秘钥文件（后缀名为.pem）拷贝到所有机器的 /etc/kubernetes/ssl 目录下备用，PEM(Privacy Enhanced Mail) 是 X.509 其中的一种编码格式, 是文本格式, 以"——BEGIN..." 开头, "——END..." 结尾, 内容是 BASE64 编码。DER - Distinguished Encoding Rules, 是二进制格式, 不可读。虽然有 PEM 和 DER 这两种编码格式, 但文件扩展名并不一定就叫"PEM" 或者"DER", 常见的扩展名除了 PEM 和 DER 还有 CRT、CER、KEY、CSR、PFX/P12、JKS, 它们除了编码格式可能不同之外, 内容也有差别, 但大多数都能相互转换编码格式.

客户端节点可能拒绝承认自签名 CA 证书有效。对于非生产环境的部署，或运行在企业防火墙后的部署，用户可以向所有客户端分发自签名 CA 证书，并刷新本地的有效证书列表。在每个客户端上执行以下操作 (注意不同的操作系统类型或者版本执行的脚本不同，此处使用的操作系统是 CentOS 7.6)：

```
openssl x509 -outform der -in ca.pem -out ca.crt
# 拷贝 CA 根证书
cp /data/k8s/ssl/ca.crt /etc/pki/ca-trust/source/anchors/
update-ca-trust
```

安装 kubectl 命令行工具

kubectl 可以操控 Kubernetes 集群，安装 kubectl(Kubernetes Control) 命令行工具在 master 节点上操作。安装 kubectl 需要对应版本或者是高版本，尽量避免使用过低版本的 client，避免遇到兼容性等不可预知的问题，这里部署的 kubenetes 是 v1.15 版本[7]。查看当前最新的稳定版本的 kubectl 版本号：

```
curl -s https://storage.googleapis.com/kubernetes-release/release/
        stable.txt
```

下载 kubectl，注意这里的下载需要通过代理方式，直接访问是无法下载的：

```
curl -LO https://storage.googleapis.com/kubernetes-release/release
        /v1.15.2/bin/linux/amd64/kubectl
```

由于在终端中无法下载 kubectl，浪费了不少时间，直接拷贝到浏览器下载即可，下载下来就一个 kubectl 文件。

创建 kubeconfig 文件

kubernetes 1.4 开始支持由 kube-apiserver 为客户端生成 TLS 证书的 TLS Bootstrapping 功能，这样就不需要为每个客户端生成证书了；该功能当前仅支持为 kubelet 生成证书；以下操作只需要在 master 节点上执行，生成的 *.kubeconfig 文件可以直接拷贝到 node 节点的/etc/kubernetes 目录下。

创建 TLS Bootstrapping Token

Token 可以是任意的包含 128 bit 的字符串，可以使用安全的随机数发生器生成。使用如下命令来完成：

```
export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x |
        tr -d ' ')
cat > token.csv <<EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-
        bootstrap"
EOF
```

[7]kubectl 官方安装文档：https://kubernetes.io/docs/tasks/tools/install-kubectl/

创建 kubelet bootstrapping kubeconfig 文件

安装 kubectl 后，执行如下命令：

```
cd /etc/kubernetes
# APISERVER 就是 master 节点
# 此处的 master 节点 IP 是 172.19.104.231
export KUBE_APISERVER="https://172.19.104.231:6443"

# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=bootstrap.kubeconfig

# 设置客户端认证参数
kubectl config set-credentials kubelet-bootstrap \
  --token=${BOOTSTRAP_TOKEN} \
  --kubeconfig=bootstrap.kubeconfig

# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kubelet-bootstrap \
  --kubeconfig=bootstrap.kubeconfig

# 设置默认上下文
kubectl config use-context default --kubeconfig=bootstrap.
        kubeconfig
```

创建 kube-proxy kubeconfig 文件

```
export KUBE_APISERVER="https://172.19.104.231:6443"
```

```
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.kubeconfig
# 设置客户端认证参数
kubectl config set-credentials kube-proxy \
  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \
  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-proxy.kubeconfig
# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=kube-proxy.kubeconfig
# 设置默认上下文
kubectl config use-context default --kubeconfig=kube-proxy.
        kubeconfig
```

生成的 kubeconfig 被保存到 /.kube/config 文件， /.kube/config 文件拥有对该集群的最高权限，需要妥善保管。

创建高可用 etcd 集群

Kuberntes 系统使用 etcd 存储所有数据，etcd 是 CoreOS 团队于 2013 年 6 月发起的开源项目，它的目标是构建一个高可用的分布式键值 (key-value) 数据库。这个词的发音是“et-cee-dee”，表示在多台机器上分发 Unix 系统的“/etc”目录，其中包含了大量的全局配置文件。它是许多分布式系统的主干，为跨服务器集群存储数据提供可靠的方式。它适用于各种操作系统，包括 Linux、BSD 和 OS X。etcd 内部采用 raft 协议作为一致性算法，etcd 基于 Go 语言实现。

Table 1.3: etcd 组件分布

主机	角色	组件
172.19.104.231	Master	etcd 集群 Leader 节点
172.19.104.230	Worker	etcd 集群 Follower 节点
172.19.150.82	Worker	etcd 集群 Follower 节点

移除旧文件

后续有一个节点无法加入集群，为了熟悉 etcd，再重新安装一遍，重新安装之前删除旧的数据：

```
cd /var/lib/etcd
```

下载二进制文件

部署 etcd 3.3.13：

```
# 下载 etcd 二进制包
wget -c https://github.com/coreos/etcd/releases/download/v3.3.13/
        etcd-v3.3.13-linux-amd64.tar.gz
tar -zxvf etcd-v3.3.13-linux-amd64.tar.gz
```

将二进制文件分发到集群的各个节点，注意如果目标集群没有对应文件夹，新建文件夹即可，当前操作的节点与其他节点要做免密登录：

```
cd /opt/k8s/work
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_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

创建证书签名请求 (Certificate Sign Request)：

```
cd /opt/k8s/work
cat > etcd-csr.json <<EOF
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "172.19.104.231",
    "172.19.104.230",
    "172.19.150.82"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "4Paradigm"
    }
  ]
}
EOF
```

生成证书和私钥：

```
cd /opt/k8s/work
/root/go/bin/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
```

分发生成的证书和私钥到各 etcd 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/etcd/cert"
    scp etcd*.pem root@${node_ip}:/etc/etcd/cert/
  done
```

创建和分发 etcd 的 systemd unit 文件

利用创建的 system Unit 文件模版，执行如下脚本生成各个节点的 system unit 文件，在/usr/lib/systemd/system/目录下创建文件 etcd.service，内容如下。注意替换 IP 地址为你自己的 etcd 集群的主机 IP。

```
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]}/" etcd.service.template > etcd-${NODE_IPS[i
        ]}.service
  done
```

环境变量配置文件/etc/etcd/etcd.conf:

```
# [member]
ETCD_NAME=infra1
ETCD_DATA_DIR="/var/lib/etcd"
ETCD_LISTEN_PEER_URLS="https://172.19.150.82:2380"
ETCD_LISTEN_CLIENT_URLS="https://172.19.150.82:2379"

# [cluster]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://172.19.150.82:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_ADVERTISE_CLIENT_URLS="https://172.19.150.82:2379"
```

启动 etcd 服务

如果有用其他方式安装过 etcd，启动时可能提示文件存在，删除之前创建的链接即可:

```
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 ${ETCD_DATA_DIR} ${ETCD_WAL_DIR}
        "
    ssh root@${node_ip} "systemctl daemon-reload && systemctl
        enable etcd && systemctl restart etcd " &
  done
```

启动之后，检查当前节点的状态:

```
# 查看状态
/opt/k8s/bin/etcdctl endpoint health
# 启动状态日志
```

```
systemctl status etcd.service
```

在所有的 Kubernetes Master 节点重复上面的步骤 (节点 1:172.19.104.231，节点 2:172.19.104.230，节点 3:172.19.150.82)，直到所有机器的 etcd 服务都已启动。Kubernetes 的 master 服务主要包括 etcd(数据存储)、control-manager(控制器)、scheduler(调度器)、apiserver(服务接口)，我们将其部署到多节点实现容错。etcd 服务启动失败时，使用如下命令查看日志：

```
journalctl -xe
```

journal 是 systemd 的日志系统，是 systemd 的一部分，从而使得 systemd 不依赖于其他组件，例如不用等待 syslog 守护进程。启动时提示 Failed at step CHDIR spawning /usr/local/bin/etcd: No such file or directory 错误，原因是/var/lib/etcd 这个目录不存在，创建该目录即可。如果无法以服务的方式启动，在机器 172.19.104.231 机器上，尝试通过命令行方式启动 etcd 的 infra1 节点：

```
/usr/local/bin/etcd \
  --name infra1 \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls https://172.19.104.231:2380 \
  --listen-peer-urls https://172.19.104.231:2380 \
  --listen-client-urls https://127.0.0.1:2379 \
  --advertise-client-urls https://172.19.104.231:2379 \
  --initial-cluster-token etcd-cluster \
  --initial-cluster infra1=https://172.19.104.231:2380,infra2=
        https://172.19.104.230:2380,infra3=https
        ://172.19.150.82:2380 \
  --initial-cluster-state new \
```

```
  --data-dir=/var/lib/etcd
```

- listen-peer-urls 是表示 etcd 和同伴 (peer) 通信的地址，多个使用逗号分隔，注意要使得所有节点都能访问，不要使用 localhost 作为配置的 URL 的主机标记。
- listen-client-urls 对外提供服务的地址：比如 http://ip:2379,http://127.0.0.1:2379，客户端会连接到这里和 etcd 交互
- advertise-client-urls 是 etcd 客户端使用，客户端通过该地址与本 member 交互信息。一定要保证从客户侧能可访问该地址。
- initial-advertise-peer-urls：其他 member 使用，其他 member 通过该地址与本 member 交互信息。一定要保证从其他 member 能可访问该地址。静态配置方式下，该参数的 value 一定要同时在–initial-cluster 参数中存在。memberID 的生成受–initial-cluster-token 和–initial-advertise-peer-urls 影响。

etcd 目前默认使用 2379 端口提供 HTTP API 服务，2380 端口和 peer 通信（这两个端口已经被 IANA 官方预留给 etcd）。启动后使用命令 etcdctl member list 查看 etcd 集群状态时提示：x509: certificate signed by unknown authority。解决此问题，指定 etcdctl 版本：

```
export ETCDCTL_API=3
```

启动节点 1：

```
/usr/local/bin/etcd \
  --name infra3 \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls https://172.19.150.82:2380 \
  --listen-peer-urls https://172.19.150.82:2380 \
  --listen-client-urls http://127.0.0.1:2379 \
  --advertise-client-urls https://172.19.150.82:2379 \
```

```
  --initial-cluster-token etcd-cluster \
  --initial-cluster infra1=https://172.19.104.231:2380,infra2=
        https://172.19.104.230:2380,infra3=https
        ://172.19.150.82:2380 \
  --initial-cluster-state new \
  --data-dir=/var/lib/etcd
```

启动节点 2:

```
/usr/local/bin/etcd \
  --name infra2 \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls https://172.19.104.230:2380 \
  --listen-peer-urls https://172.19.104.230:2380 \
  --listen-client-urls http://127.0.0.1:2379 \
  --advertise-client-urls https://172.19.104.230:2379 \
  --initial-cluster-token etcd-cluster \
  --initial-cluster infra1=https://172.19.104.231:2380,infra2=
        https://172.19.104.230:2380,infra3=https
        ://172.19.150.82:2380 \
  --initial-cluster-state new \
  --data-dir=/var/lib/etcd
```

提示错误：error "remote error: tls: bad certificate", ServerName ""。输入如下命令检查 etcd 集群：

```
/usr/local/bin/etcdctl --endpoint https://127.0.0.1:2379 \
--ca-file=/etc/kubernetes/ssl/ca.pem        \
```

```
--cert-file=/etc/kubernetes/ssl/kubernetes.pem    \
--key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
cluster-health
# 查看集群单个节点状态
etcdctl endpoint status
# 查看集群多个节点状态
etcdctl endpoint status --cluster \
--cacert=/etc/kubernetes/ssl/ca.pem \
--cert=/etc/kubernetes/ssl/kubernetes.pem \
--key=/etc/kubernetes/ssl/kubernetes-key.pem \

etcdctl endpoint https://127.0.0.1:2379 \
--cacert=/etc/kubernetes/ssl/ca.pem \
--cert=/etc/kubernetes/ssl/kubernetes.pem \
--key=/etc/kubernetes/ssl/kubernetes-key.pem \

# etcd v3 查看集群状态
etcdctl endpoint health --cluster \
--cacert=/etc/kubernetes/ssl/ca.pem \
--cert=/etc/kubernetes/ssl/kubernetes.pem \
--key=/etc/kubernetes/ssl/kubernetes-key.pem
# v3 版本查看方式
etcdctl endpoint health
# 查看 etcd 集群的所有节点
etcdctl member list
etcdctl member add infra2 http://172.19.104.230:2380
```

健康的 etcd 集群使用如下命令可以检查:

```
[root@iZuf63refzweg1d9dh94t8Z ~]# etcdctl endpoint health
        -endpoints=localhost:2379
localhost:2379 is healthy: successfully committed proposal: took =
         3.656346ms
```

输入如下命令查看 etcd 集群信息：

```
# 查看 etcd 集群信息
curl http://localhost:2379/metrics
# 添加 etcd 节点
# 添加节点失败，提示错误 etcdserver: unhealthy cluster
# 删除集群数据 (/var/lib/etcd)，重新安装集群解决
etcdctl member add infra2 --peer-urls="http://172.19.104.230:2380"
etcdctl member add infra2 http://172.19.104.230:2380
```

v3 版本通过 REST Api 添加集群节点：

```
curl http://127.0.0.1:2379/v3beta/cluster/member/add -XPOST -H "
        Content-Type: application/json" -d '{"peerURLs": ["http
        ://172.19.104.230:2380"]}'

curl http://172.19.104.231:2379/health
```

查看集群健康状态：

```
etcdctl --endpoints=https://172.19.104.231:2379 \
--ca-file=/etc/kubernetes/ssl/kubernetes.pem \
--cert-file=/etc/kubernetes/pki/etcd/client.pem \
--key-file=/etc/kubernetes/pki/etcd/client-key.pem cluster-health

/opt/k8s/bin/etcdctl endpoint health --cluster --cacert=/etc/
        kubernetes/cert/ca.pem --cert=/etc/etcd/cert/etcd.pem --
        key=/etc/etcd/cert/etcd-key.pem

# 查看 etcd 集群健康状态
/opt/k8s/bin/etcdctl endpoint health --cluster

# 查看集群信息
curl -k --cert /etc/etcd/cert/etcd.pem --key /etc/etcd/cert/etcd-
```

```
        key.pem https://172.19.150.82:2379/metrics
```

以表格形式查看集群信息：

```
/opt/k8s/bin/etcdctl endpoint status --write-out=table
```

etcd 部署完毕后，使用如下脚本查看集群的 Leader 信息[8]：

```
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
```

输出结果如图1.1所示，可以看到集群的 ID、集群数据库的大小、集群的版本、Term，Term 可以理解为周期（第几届、任期）的概念[9]，用 Term 作为一个周期，每个 Term 都是一个连续递增的编号，每一轮选举都是一个 Term 周期，在一个 Term 中只能产生一个 Leader；每次 Term 的递增都将发生新一轮的选举，Raft 保证一个 Term 只有一个 Leader，在 Raft 正常运转中所有的节点的 Term 都是一致的，如果节点不发生故障一个 Term（任期）会一直保持下去，当某节点收到的请求中 Term 比当前 Term 小时则拒绝该请求。

```
[root@iZuf63refzweg1d9dh94t8Z work]# ./check-etcd-cluster-leader.sh
+----------------------------+------------------+---------+---------+-----------+-----------+------------+
|          ENDPOINT          |        ID        | VERSION | DB SIZE | IS LEADER | RAFT TERM | RAFT INDEX |
+----------------------------+------------------+---------+---------+-----------+-----------+------------+
| https://172.19.104.231:2379 | 5ab2d0e431f00a20 |  3.3.13 |   20 kB |      true |         5 |         16 |
| https://172.19.104.230:2379 | 56298c42af788da7 |  3.3.13 |   20 kB |     false |         5 |         16 |
|  https://172.19.150.82:2379 | 84c70bf96ccff30f |  3.3.13 |   20 kB |     false |         5 |         16 |
+----------------------------+------------------+---------+---------+-----------+-----------+------------+
```

Figure 1.1: etcd 查看 Leader 节点

[8]内容来源：https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master

[9]内容来源：http://www.solinx.co/archives/415

部署 master 节点

kubernetes master 节点包含的组件如下：

- kube-apiserver
- kube-scheduler
- kube-controller-manager

kube-scheduler、kube-controller-manager 和 kube-apiserver 三者的功能紧密相关。同时只能有一个 kube-scheduler、kube-controller-manager 进程处于工作状态，如果运行多个，则需要通过选举产生一个 leader；

下载最新版本的 Kubernetes 二进制文件

在中国大陆境内无法直接下载 Kubernetes 二进制文件，奇怪的是虽然使用了代理，但是还是无法通过 dl.k8s.io 域名进行下载，没有找到有价值的原因分析。所以此处采取的方式是在国外的主机上下载完毕后，拷贝文件到国内到云主机上进行部署 (当然可以下载源码自己编译，不过估计要花不少时间和精力来处理其中遇到的问题)，国外主机下载速度较快，在 50MB 每秒。注意以下命令在国外的服务器上执行，这里是用的美国俄亥俄州的节点，当然香港、新加坡、台湾等物理距离较近的节点更好。

```
wget -c https://github.com/kubernetes/kubernetes/releases/download
        /v1.15.2/kubernetes.tar.gz
tar -xzvf kubernetes.tar.gz
cd kubernetes
./cluster/get-kube-binaries.sh
```

从 Github 上下载下来的仅仅是一些脚本，这里下载的是 v1.15.2 版本的脚本，大概有 600KB 左右，完整的 Binary 文件估计有 1.5GB+。下载完毕后拷贝到阿里云服务器：

```
scp -i dolphin.pem -r ubuntu@ec2-23-53-88-011.us-east-2.compute.
        amazonaws.com:~/* .
```

将二进制文件拷贝到指定路径:

```
cp -r server/bin/{kube-apiserver,kube-controller-manager,kube-
        scheduler,kubectl,kube-proxy,kubelet} /usr/local/bin/
```

配置和启动 kube-apiserver

kube-apiserver 是在部署 kubernetes 集群是最需要先启动的组件，也是和集群交互的核心组件。创建 kube-apiserver 的 service 配置文件 (路径：/usr/lib/systemd/system)，service 配置文件 kube-apiserver.service 内容：

```
[Unit]
Description=Kubernetes API Service
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=etcd.service

[Service]
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/apiserver
ExecStart=/usr/local/bin/kube-apiserver \
        $KUBE_LOGTOSTDERR \
        $KUBE_LOG_LEVEL \
        $KUBE_ETCD_SERVERS \
        $KUBE_API_ADDRESS \
        $KUBE_API_PORT \
        $KUBELET_PORT \
        $KUBE_ALLOW_PRIV \
        $KUBE_SERVICE_ADDRESSES \
        $KUBE_ADMISSION_CONTROL \
        $KUBE_API_ARGS
Restart=on-failure
Type=notify
LimitNOFILE=65536

```

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

/etc/kubernetes/config 文件的内容为：

```
#
# kubernetes system config
#
# The following values are used to
# configure various aspects of all
# kubernetes services, including
#
# kube-apiserver.service
# kube-controller-manager.service
# kube-scheduler.service
# kubelet.service
# kube-proxy.service
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed
# to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=true"

# How the controller-manager, scheduler
# and proxy find the apiserver
KUBE_MASTER="--master=http://172.19.104.231:8080"
```

该配置文件同时被 kube-apiserver、kube-controller-manager、kube-scheduler、kubelet、kube-proxy 使用。添加 api server 配置文件/etc/kubernetes/apiserver，内容为：

```
KUBE_API_ADDRESS="--advertise-address=172.19.104.231 --bind-
        address=172.19.104.231 --insecure-bind-address
        =172.19.104.231"
#
# The port on the local server to listen on.
#
# Port minions listen on
#
# Comma separated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd-servers=https://172.19.104.231:2379,
        https://172.19.104.230:2379,https://172.19.150.82:2379"
#
# Address range to use for services
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16"
#
# default admission control policies
KUBE_ADMISSION_CONTROL="--admission-control=ServiceAccount,
        NamespaceLifecycle,NamespaceExists,LimitRanger,
        ResourceQuota"
#
# Add your own!
KUBE_API_ARGS="--authorization-mode=RBAC --runtime-config=rbac.
        authorization.k8s.io/v1beta1 --kubelet-https=true --enable
        -bootstrap-token-auth --token-auth-file=/etc/kubernetes/
        token.csv --service-node-port-range=30000-32767 --tls-cert
        -file=/etc/kubernetes/ssl/kubernetes.pem --tls-private-key
        -file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-
        file=/etc/kubernetes/ssl/ca.pem --service-account-key-file
        =/etc/kubernetes/ssl/ca-key.pem --etcd-cafile=/etc/
        kubernetes/ssl/ca.pem --etcd-certfile=/etc/kubernetes/ssl/
        kubernetes.pem --etcd-keyfile=/etc/kubernetes/ssl/
        kubernetes-key.pem --enable-swagger-ui=true --apiserver-
```

```
        count=3 --audit-log-maxage=30 --audit-log-maxbackup=3 --
        audit-log-maxsize=100 --audit-log-path=/var/lib/audit.log
        --event-ttl=1h"
```

输入如下命令启动 kube-apiserver：

```
# 调整配置文件后执行此命令重新加载配置
systemctl daemon-reload
systemctl start kube-apiserver.service
systemctl status kube-apiserver.service
```

启动 kube-apiserver 服务时提示 unknown flag: –experimental-bootstrap-token-auth，原来 Bootstrap Token Authentication 在 1.9 版本已经变成了正式 feature(此次部署的版本是 v1.15)，将 kube-apiserver 配置文件/etc/kubernetes/apiserver 相应参数改为–enable-bootstrap-token-auth 即可。启动时提示错误：

```
clientconn.go:1251] grpc: addrConn.createTransport failed to
        connect to {172.19.150.82:2379 0  <nil>}. Err :connection
        error: desc = "transport: authentication handshake failed:
         x509: certificate signed by unknown authority (possibly
        because of \"crypto/rsa: verification error\" while trying
         to verify candidate authority certificate \"kubernetes\")
        ". Reconnecting...
```

将 pem 格式的证书转换为 crt 格式：

```
openssl x509 -outform der -in ca.pem -out ca.crt
# 拷贝 CA 根证书
cp /data/k8s/ssl/ca.crt /etc/pki/ca-trust/source/anchors/
update-ca-trust
```

创建 kube-controller-manager 集群

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",
      "172.19.104.230",
      "172.19.104.231",
      "172.19.150.82"
    ],
    "names": [
      {
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:kube-controller-manager",
        "OU": "4Paradigm"
      }
    ]
}
EOF
```

hosts 列表包含所有 kube-controller-manager 节点 IP。创建 kube-controller-manager 的 serivce 配置文件，文件路径/usr/lib/systemd/system/kube-controller-manager.service[10]。

```
[Unit]
Description=Kubernetes Controller Manager
```

[10]参考链接：https://jimmysong.io/kubernetes-handbook/practice/master-installation.html

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

[Service]
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/controller-manager
ExecStart=/usr/local/bin/kube-controller-manager \
        $KUBE_LOGTOSTDERR \
        $KUBE_LOG_LEVEL \
        $KUBE_MASTER \
        $KUBE_CONTROLLER_MANAGER_ARGS
Restart=on-failure
LimitNOFILE=65536

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

配置文件/etc/kubernetes/controller-manager。

```
###
# The following values are used to configure the kubernetes
        controller-manager

# defaults from config and apiserver should be adequate

# Add your own!
KUBE_CONTROLLER_MANAGER_ARGS="--address=127.0.0.1 --service-
        cluster-ip-range=10.254.0.0/16 --cluster-name=kubernetes
        --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem --
        cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem
        --service-account-private-key-file=/etc/kubernetes/ssl/ca-
        key.pem --root-ca-file=/etc/kubernetes/ssl/ca.pem --leader
        -elect=true"
```

通过如下命令启动 controller-manager[11]：

```
systemctl start kube-controller-manager
```

无法通过服务启动时，可直接通过命令行启动 Controller Manager：

```
nohup /usr/local/bin/kube-controller-manager --address=127.0.0.1
        --service-cluster-ip-range=10.254.0.0/16 --cluster-name=
        kubernetes --cluster-signing-cert-file=/etc/kubernetes/ssl
        /ca.pem --cluster-signing-key-file=/etc/kubernetes/ssl/ca-
        key.pem  --service-account-private-key-file=/etc/
        kubernetes/ssl/ca-key.pem --root-ca-file=/etc/kubernetes/
        ssl/ca.pem --leader-elect=true --master=http:
        //172.19.104.231:8080 &
```

配置和启动 kube-scheduler

创建 kube-scheduler 的 serivce 配置文件，文件路径/usr/lib/systemd/system/kube-scheduler.service。使用命令行启动 kubernetes scheduler：

```
/usr/local/bin/kube-scheduler --master=http://172.19.104.231:8080
        --leader-elect=true --address=127.0.0.1
```

验证 master 节点功能

master 所有组件启动完成后，使用如下命令查看组件状态：

```
kubectl get componentstatuses
```

提示错误：The connection to the server localhost:8080 was refused - did you specify the right host or port? 此错误表示 kubectl 未正确配置。配置 kubeconfig 如下：

```
export KUBE_APISERVER="http://172.19.104.231:8080"
# 设置集群参数
```

[11] 来源链接：https://jimmysong.io/kubernetes-handbook/practice/master-installation.html

```
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER}
# 设置客户端认证参数
kubectl config set-credentials admin \
  --client-certificate=/etc/kubernetes/ssl/admin.pem \
  --embed-certs=true \
  --client-key=/etc/kubernetes/ssl/admin-key.pem
# 设置上下文参数
kubectl config set-context kubernetes \
  --cluster=kubernetes \
  --user=admin
# 设置默认上下文
kubectl config use-context kubernetes
```

添加部分基础配置后，连接拒绝的错误消失。输出各个组件的状态 (Component Status)：

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

安装 flannel 网络插件

覆盖网络 (Overlay Network) 就是应用层网络，它是面向应用层的，不考虑或很少考虑网络层，物理层的问题。详细说来，覆盖网络是指建立在另一个网络上的网络。该网络中的结点可以看作通过虚拟或逻辑链路而连接起来的。虽然在底层有很多条物理链路，但是这些虚拟或逻辑链路都与路径一一对应。例如：许多 P2P 网络就是覆盖网络，因为它运行在互连网的上层。覆盖网络允许对没有 IP 地址标识的目的主机路由信息，例如：Freenet 和 DHT（分布式哈希表）可以路由信息到一个存储特定文件的

结点，而这个结点的 IP 地址事先并不知道。覆盖网络被认为是一条用来改善互连网路由的途径，让二层网络在三层网络中传递，既解决了二层的缺点，又解决了三层的不灵活！Flannel 实质上是一种“覆盖网络 (overlay network)”，也就是将 TCP 数据包装在另一种网络包里面进行路由转发和通信，目前已经支持 UDP、VxLAN、AWS VPC 和 GCE 路由等数据转发方式。数据从源容器中发出后，经由所在主机的 docker0 虚拟网卡转发到 flannel0 虚拟网卡，这是个 P2P 的虚拟网卡，flanneld 服务监听在网卡的另外一端。Flannel 通过 Etcd 服务维护了一张节点间的路由表，详细记录了各节点子网网段。源主机的 flanneld 服务将原本的数据内容 UDP 封装后根据自己的路由表投递给目的节点的 flanneld 服务，数据到达以后被解包，然后直接进入目的节点的 flannel0 虚拟网卡，然后被转发到目的主机的 docker0 虚拟网卡，最后就像本机容器通信一下的有 docker0 路由到达目标容器。所有的 node 节点都需要安装网络插件才能让所有的 Pod 加入到同一个局域网中，本文是安装 flannel 网络插件的参考文档。

下载分发 flannel 二进制文件

获取 flannel 部署文件[12]:

```
cd /opt/k8s/work
mkdir flannel
wget https://github.com/coreos/flannel/releases/download/v0.11.0/
        flannel-v0.11.0-linux-amd64.tar.gz
tar -xzvf flannel-v0.11.0-linux-amd64.tar.gz -C flannel
```

service 配置文件/usr/lib/systemd/system/flanneld.service。

```
[Unit]
Description=Flanneld overlay address etcd agent
After=network.target
After=network-online.target
Wants=network-online.target
After=etcd.service
Before=docker.service

```

[12]来源链接：https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master

```
[Service]
Type=notify
EnvironmentFile=/etc/sysconfig/flanneld
EnvironmentFile=-/etc/sysconfig/docker-network
ExecStart=/usr/bin/flanneld-start \
  -etcd-endpoints=${FLANNEL_ETCD_ENDPOINTS} \
  -etcd-prefix=${FLANNEL_ETCD_PREFIX} \
  $FLANNEL_OPTIONS
ExecStartPost=/usr/libexec/flannel/mk-docker-opts.sh -k
        DOCKER_NETWORK_OPTIONS -d /run/flannel/docker
Restart=on-failure

[Install]
WantedBy=multi-user.target
RequiredBy=docker.service
```

/etc/sysconfig/flanneld 配置文件：

```
# Flanneld configuration options

# etcd url location.  Point this to the server where etcd runs
FLANNEL_ETCD_ENDPOINTS="https://172.19.104.231:2379,https:
        //172.19.104.230:2379,https://172.19.150.82:2379"

# etcd config key.  This is the configuration key that flannel
        queries
# For address range assignment
FLANNEL_ETCD_PREFIX="/kube-centos/network"

# Any additional options that you want to pass
FLANNEL_OPTIONS="-etcd-cafile=/etc/kubernetes/ssl/ca.pem -etcd-
        certfile=/etc/kubernetes/ssl/kubernetes.pem -etcd-keyfile
        =/etc/kubernetes/ssl/kubernetes-key.pem"
```

执行下面的命令为 docker 分配 IP 地址段。

```
etcdctl --endpoints=https://172.19.104.231:2379,https
        ://172.19.104.230:2379,https://172.19.150.82:2379 \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  mkdir /kube-centos/network
etcdctl --endpoints=https://172.19.104.231:2379,https
        ://172.19.104.230:2379,https://172.19.150.82:2379 \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  mk /kube-centos/network/config '{"Network":"172.30.0.0/16","
        SubnetLen":24,"Backend":{"Type":"vxlan"}}'
```

启动 flannel:

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

也可直接执行如下命令启动:

```
/usr/bin/flanneld -etcd-endpoints=https://172.19.104.231:2379,
        https://172.19.104.230:2379,https://172.19.150.82:2379 -
        etcd-prefix=/kube-centos/network -etcd-cafile=/etc/
        kubernetes/ssl/ca.pem -etcd-certfile=/etc/kubernetes/ssl/
        kubernetes.pem -etcd-keyfile=/etc/kubernetes/ssl/
        kubernetes-key.pem
/root/flanneld -etcd-endpoints=https://172.19.104.231:2379,https
```

```
        ://172.19.104.230:2379,https://172.19.150.82:2379 -etcd-
        prefix=/kube-centos/network -etcd-cafile=/etc/kubernetes/
        ssl/ca.pem -etcd-certfile=/etc/kubernetes/ssl/kubernetes.
        pem -etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem
```

启动时提示错误：failed to retrieve network config: 100: Key not found (/kube-centos)。要先设置 key：

```
# v2 版本写法
etcdctl set /kube-centos/network/config '{ "Network":"10.1.0.0/16"
         }'
# v3 版本写法
etcdctl put /kube-centos/network/config '{ "Network":"10.1.0.0/16"
         }'
etcdctl get /kube-centos/network/config
# 查看所有的 key(etcd v3 写法)
etcdctl get / --prefix --keys-only
# 查看集群信息
ETCDCTL_API=3 etcdctl get /registry/pods --prefix -w json|python -
        m json.tool
```

实际操作过程中，即使设置好了 key，也还是会抛出 Key not found 的异常，初步确定是由于 flannel 截止到 2019 年 09 月 06 日，还没有兼容分布式数据库 etcd v3 版本，读取分布式数据库还是采用的 v2 版本的读取方式，而 v3 版本和 v2 版本的读取方式是不兼容的，所以会出现数据库明明已经设置了对应的 key，但是 flannel 还是始终无法读取到，提示 key 未找到错误。Etcd V2 和 V3 之间的数据结构完全不同，互不兼容，也就是说使用 V2 版本的 API 创建的数据只能使用 V2 的 API 访问，V3 的版本的 API 创建的数据只能使用 V3 的 API 访问。这就造成我们访问 etcd 中保存的 flannel 的数据需要使用 etcdctl 的 V2 版本的客户端，而访问 kubernetes 的数据需要设置 ETCDCTL_API=3 环境变量来指定 V3 版本的 API。安装好 flannel 插件后，可以在 etcd 中检查是否有相应的数据，来判断是否运行正常。查询 etcd 中的内容可以看到:

```
/usr/local/bin/etcdctl --endpoints=${ETCD_ENDPOINTS} \
```

```
--cacert=/etc/kubernetes/ssl/ca.pem \
--cert=/etc/kubernetes/ssl/kubernetes.pem \
--key=/etc/kubernetes/ssl/kubernetes-key.pem \
```

注意不同版本的 etcdctl 其命令的参数有可能不一致，此处使用的 etcdctl 版本是 3.3.13。

安装完毕后验证

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
```

输出的内容如下：

```
>>> 172.19.104.231
    inet 172.30.224.0/32 scope global flannel.1
>>> 172.19.104.230
    inet 172.30.208.0/32 scope global flannel.1
>>> 172.19.150.82
    inet 172.30.184.0/32 scope global flannel.1
```

在各节点上 ping 所有 flannel 接口 IP，确保能通：

```
source /opt/k8s/bin/environment.sh
for node_ip in ${NODE_IPS[@]}
  do
    echo ">>> ${node_ip}"
    ssh ${node_ip} "ping -c 1 172.30.80.0"
```

```
    ssh ${node_ip} "ping -c 1 172.30.32.0"
    ssh ${node_ip} "ping -c 1 172.30.184.0"
  done
```

此脚本会登陆各台主机，分别 ping 所有主机，保证所有主机之间能够互相 ping 通。在安装的过程中，遇到了本主机的 IP 可以 ping 通，但是其他主机的 IP 无法 ping 通的情况。添加 iptables 规则解决此问题 (实际上主机的 iptables 服务是没有启动的)：

```
sudo iptables -P INPUT ACCEPT
sudo iptables -P OUTPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
```

由于 82 这台主机使用的是 firewalld 防火墙，暂时还不清楚 flannel 服务需要开启哪个端口才可以正常访问，暂时开启所有端口来解决部署问题：

```
firewall-cmd --permanent --zone=public --add-port=0-65525/tcp
firewall-cmd --permanent --zone=public --add-port=0-65535/udp
firewall-cmd --reload
```

1.1.1 部署 kubelet 组件

创建和分发 kubelet systemd unit 文件

创建 kubelet systemd unit 文件模板，创建模版时，检查启动命令不要有 allow-privileged 参数，由于此处部署的是 Kubernetes v1.15.2 版本，allow-privileged 已经在 1.15.0 版本中删除[13]：

```
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
```

[13] 内容来源：https://github.com/opsnull/follow-me-install-kubernetes-cluster/blob/master/07-2.kubelet.md

```
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=${K8S_DIR}/kubelet
ExecStart=/opt/k8s/bin/kubelet \\
  --bootstrap-kubeconfig=/etc/kubernetes/kubelet-bootstrap.
      kubeconfig \\
  --cert-dir=/etc/kubernetes/cert \\
  --cni-conf-dir=/etc/cni/net.d \\
  --container-runtime=docker \\
  --container-runtime-endpoint=unix:///var/run/dockershim.sock \\
  --root-dir=${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/
      images_k8s/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
```

启动 kubelet 服务

如下指令启动 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
```

kubelet 启动后使用–bootstrap-kubeconfig 向 kube-apiserver 发送 CSR(Certificate Signing Request) 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。启动时提示错误：unknown flag: –allow-privileged，是由于 kubelet 1.15.0 的启动参数中 allow-privileged 已被删除，所以在启动参数中将之移除。

查看 kubelet 的情况

等待一段时间 (1-10 分钟)，三个节点的 CSR 都被自动 approved：

```
/opt/k8s/bin/kubectl get csr
```

如果执行命令提示 No resources found，则可以使用如下命令查看 kubelet 服务运行日志：

```
systemctl status kubelet -l
systemctl status kubelet.service | tail
```

提示错误 Failed to list *v1.Pod: Unauthorized。第一确认 kube-apiserver 的确配置了 Node 授权模式：–authorization-mode=Node,RBAC。/etc/kubernetes/cert/目录下有 kubelet-

client 和 kubelet-server 证书。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:<Token ID>，group 设置为 system:bootstrappers，这一过程称为 Bootstrap Token Auth。默认情况下，这个 user 和 group 没有创建 CSR 的权限，kubelet 启动失败。解决办法是：创建一个 clusterrolebinding，将 group system:bootstrappers 和 clusterrole system:node-bootstrapper 绑定：

```
/opt/k8s/bin/kubectl create clusterrolebinding kubelet-bootstrap
        --clusterrole=system:node-bootstrapper --group=system:
        bootstrappers
```

执行完毕以上动作，查看集群 CSR(Certificate Signing Request) 时，输出如下内容：

```
[root@ops001 work]# kubectl get csr
NAME         AGE     REQUESTOR                    CONDITION
csr-2k5rp    8m9s    system:bootstrap:6o33uh      Pending
csr-4q7qw    23m     system:bootstrap:27qa60      Pending
csr-4rtpz    4m57s   system:bootstrap:58ndy6      Pending
csr-92bs9    8m8s    system:bootstrap:27qa60      Pending
csr-c8gql    23m     system:bootstrap:6o33uh      Pending
csr-csxnq    23m     system:bootstrap:58ndy6      Pending
csr-dr6ld    23m     system:bootstrap:27qa60      Pending
csr-hcppn    23m     system:bootstrap:58ndy6      Pending
csr-k6khk    23m     system:bootstrap:6o33uh      Pending
csr-t4xk7    19m     system:bootstrap:58ndy6      Pending
```

可以看出当前所有的 CSR 都处于 Pending 状态，需要注意的是，在查看 CSR 的时候，由于生成 CSR 需要较长时间，当前输出的可能是部分需要 approve 的 CSR，根据实际经验，即使在手动 approve 到最后的 CSR 时，仍然会有新的 CSR 产生。

手动 Approve Server Certificate CSR

基于安全性考虑，CSR approving controllers 不会自动 approve kubelet server 证书签名请求，需要手动 approve，输入如下命令手动 Approval 证书：

```
/opt/k8s/bin/kubectl certificate approve csr-2k5rp
```

1.1.2 部署 CoreDNS 插件

CoreDNS 是一个 Go 语言实现的链式插件，是一个高性能、易扩展的 DNS 服务端。集群中的每个流程都需要知道自己是如何与其他服务联系，但集群 IP 地址往往是动态的，很难通过 IP 来解决服务问题。CoreDNS 为服务提供了一种发现彼此的方式，开发者只需知道服务名称，CoreDNS 就可回复相应 IP 地址。CoreDNS 在 Kubernetes1.11 版本已经做为 GA 功能释放，成为 Kubernetes 默认的 DNS 服务替代了 Kube-DNS，目前是 kubeadm、kube-up、minikube 和 kops 安装工具的默认选项。kuberntes 自带插件的 manifests yaml 文件使用 gcr.io 的 docker registry，国内被墙，需要手动替换为其它 registry 地址，这里替换成了微软中国镜像[14]地址 (微软提供镜像服务确实很赞，又多了一个选择，如果是过去几年微软提供镜像部署 Google 的服务是不大可能的)，例如 CoreDNS 的地址是 gcr.azk8s.cn/google-containers/coredns:1.3.1，替换 CoreDNS 地址后，可以使用如下命令验证地址是否可用：

```
# 测试 Kubenetes Dashboard 镜像地址是否可达
docker pull gcr.azk8s.cn/google_containers/kubernetes-dashboard-
        amd64:v1.10.1
# 测试 CoreDNS 镜像地址是否可达
docker pull gcr.azk8s.cn/google-containers/coredns:1.3.1
```

调整 CoreDNS 配置文件 (/opt/k8s/work/kubernetes/cluster/addons/dns/coredns)，需要将 __PILLAR__DNS__DOMAIN__ 设置为集群环境中的域名，这个域名需要和 kubelet 的–cluster-domain 参数值一致。查看集群的 cluster Domain 在配置文件/etc/kubernetes/kubelet-config.yaml 中，显示的是 cluster.local。需要将 __PILLAR__DNS__SERVER__ 设置为集群环境的 IP，这个 IP 需要和 kubelet 的–cluster-dns 参数值一致，kubelet 的 cluster DNS 配置也在 kubelet config 文件中，值是 10.254.0.2。使用如下命令部署 CoreDNS：

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

[14]镜像地址：http://mirror.azk8s.cn，其中 GCR Proxy Cache 服务帮助地址：http://mirror.azk8s.cn/help/gcr-proxy-cache.html，截止到 2019 年 09 月 20 日，该地址可以正常访问。

1.1.3 安装 Kubernetes Dashboard 插件

Kubernetes 的国外安装其实非常简单，国内安装的主要问题在于 kubernetes 部件所需的官方镜像在 http://gcr.io(Google Cloud Container Registry) 上，但是此网站在国内是无法访问的。当前安装 Kubernetes Dashboard 的时间是 2019 年 08 月 30 日，本文档中安装的是 kubernetes dashboard v1.6.0[15]。执行如下命令部署 Dashboard Web UI(此方法不可取，在实际部署过程中，由于国内特殊的网络环境，部署所需要的部分镜像无法获取，在部署之前，还需要修改部分镜像的地址配置，将地址修改为国内网络可以访问的地址[16]，方可继续部署，否则后面需要花费大量的事件处理各种莫名的问题，浪费时间和精力)：

```
# 安装 Dashboard
kubectl apply -f https://raw.githubusercontent.com/kubernetes/
        dashboard/v1.10.1/src/deploy/recommended/kubernetes-
        dashboard.yaml
# 卸载 Dashboard
kubectl delete -f https://raw.githubusercontent.com/kubernetes/
        dashboard/v1.10.1/src/deploy/recommended/kubernetes-
        dashboard.yaml
```

部署完毕后，查看 Pods：

```
kubectl get pod --namespace=kube-system
# 查看所有命名空间的 Pods
kubectl get pods --all-namespaces
kubectl get pods,svc --all-namespaces -o wide
```

查看到镜像一直处于 Pending 状态，查看镜像的日志：

```
kubectl --namespace kube-system logs kubernetes-dashboard-7
        d75c474bb-b2lwd
```

[15]参考来源：https://jimmysong.io/kubernetes-handbook/practice/dashboard-addon-installation.html

[16]具体可以参考文章:https://jimmysong.io/kubernetes-handbook/practice/dashboard-upgrade.html

```
# 查看 Dashboard 对外映射的端口
kubectl get svc -n kube-system
```

如果日志没有输出，查看事件：

```
kubectl get events --namespace=kube-system
```

通过事件可以看出 kube-dns 和 metrics-server 服务 IP 范围指定与原有的 IP 范围不一致，提示 ClusterIPOutOfRange 错误。查看镜像的配置：

```
./kubectl describe pod kubernetes-dashboard-7d75c474bb-b2lwd --
        namespace="kube-system"
```

启动 Scheduler 组件时，提示 Failed to update lock: Operation cannot be fulfilled on endpoints "kube-scheduler": StorageError: invalid object, Code: 4, Key: /registry/services/endpoints/kube-system/kube-scheduler, ResourceVersion: 0, AdditionalErrorMsg: Precondition failed: UID in precondition: 121bc661-80f5-11e9-b3ce-00163e086f0c, UID in object meta: 49e84916-589a-4da5-b78a-761a1fe78285 错误。

查看集群信息：

```
# 查看集群 IP 信息
kubectl cluster-info dump | grep service-cluster-ip-range
kubectl get cm kubeadm-config -n kube-system -o yaml
```

通过代理访问 dashboard

可以使用 kubectl 命令行工具访问 Dashboard，启动代理命令如下：

```
kubectl proxy --address=0.0.0.0 --port=8001 --accept-hosts='^*$'
```

accept-hosts 设置 API 接收所有主机的请求。输入如下链接通过 kubectl proxy 访问 Dashboard：

```
curl http://172.19.150.82:8001/api/v1/namespaces/kube-system/
        services/https:kubernetes-dashboard:/proxy/
```

```
http://kubernetes.example.com/api/v1/namespaces/kube-system/
        services/kube-ui/#/dashboard/
curl http://172.19.104.231:8001/api/v1/proxy/namespaces/kube-
        system/services/kubernetes-dashboard/
# 通过域名访问的访问地址
# 不过返回的 Json 字符串
https://kubernetes.ttt208.com/api/v1/namespaces/kube-system/
        services/kubernetes-dashboard
```

需要注意的是，Kubernetes Web UI 默认只支持在直接执行代理操作命令的机器上访问。查看 kube-system 命名空间下 pod 状态：

```
kubectl get pod --namespace=kube-system
```

访问 Kubenetes Dashboard 时提示 no endpoints available for service "kubernetes-dashboard"。

```
kubectl get pods --namespace kube-system
kubectl describe pod kubernetes-dashboard-74d7cc788-mk9c7 --
        namespace kube-system
kubectl logs -n kube-system kubernetes-dashboard-74d7cc788-mk9c7 -
        c dnsmasq
```

发现问题是 no nodes available to schedule pods，无可用节点，原来是没有安装 kubelet 组件导致，安装组件将节点修复即可。

通过 kube-apiserver 访问 Dashboard

```
https://172.19.104.231:6443/api/v1/namespaces/kube-system/services
        /https:kubernetes-dashboard:/proxy
```

在公网访问 Dashboard 链接：

```
http://kubernetes.ttt208.com/api/v1/namespaces/kube-system/
        services/https:kubernetes-dashboard:/proxy/
```

```
http://172.19.104.231:8001/api/v1/namespaces/kube-system/services/
        https:kubernetes-dashboard:/proxy/
http://172.19.150.82:8001/api/v1/namespaces/kube-system/services/
        https:kubernetes-dashboard:/proxy/
curl -L http://172.19.150.82:31085/ui
```

提示错误：Error: 'dial tcp 172.17.0.6:8443: connect: connection refused'，Trying to reach: 'https://172.17.0.6:8443/'。在主机 172.19.104.231 上可以 ping 通此 IP，使用如下命令列出主机 172.19.104.231 所有 IP：

```
docker inspect --format='{{.Name}} - {{range .NetworkSettings.
        Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)
```

输出结果如下：

```
/elegant_rosalind - 172.17.0.10
/confluence - 172.17.0.5
/gitlab - 172.17.0.2
/nexus3 - 172.17.0.9
/jenkins - 172.17.0.6
/mysql_atlassian - 172.17.0.7
/jira - 172.17.0.8
/nginx - 172.17.0.4
/rancher-server - 172.17.0.3
```

说明访问 Dashboard UI 时，系统指向了主机上 jenkins 的容器地址，jenkins 容器自然没有启动 8443 端口，所以会提示错误。而登陆 Dashboard 部署的主机查看，刚好发现 Dashboard 容器的 IP 也是 172.17.0.6，说明不同主机的 Docker 分配到了相同的 IP，这就是一个跨主机容器 IP 冲突的问题了。尝试将 Jenkins 的 IP 重新获取，避免和 Kubernetes 容器的 IP 冲突，发现也无法解决问题。经过分析，各个主机的 IP 无法联通，本质上还是属于跨主机容器通信的问题。为什么会产生这样的问题？是由于 Kubernetes 是后来部署的，在部署 flannel 网络组件之前，主机上已经启动了 Docker 并部署了其他应用，后来在加入 flannel 组件时，原来的应用没有纳入 flannel 的网络管理，而从公网访问 Kubernetes 集群又由早先部署在 Docker 中的 Nginx 应用转发，

Nginx 和 Dashboard 分别部署在不同主机的 Docker 中，在没有采用 flannel 做跨主机容器通信之前，Nginx 当然无法访问后来新增的 Kubernetes Dashboard。所以尝试将不同主机的 Docker 纳入 flannel 统一分配网段，才是解决问题的根本办法。

Docker 加入 Flannel 网络

如果需要跨主机容器间能够通信，考虑将原有的 Docker 容器加入到 flannel 网络中。Docker 安装完成以后，需要修改其启动参数以使其能够使用 flannel 进行 IP 分配，以及网络通讯。在 flannel 运行之后，主要做了以下几步的工作：从 etcd 中获取 network 的配置信息；划分 subnet，并在 etcd 中进行注册；将子网信息记录到/run/flannel/subnet.env 中。生成一个环境变量文件，包含了当前主机要使用 flannel 通讯的相关参数，如下：

```
[root@ops001 conf.d]# cat /run/flannel/subnet.env
FLANNEL_NETWORK=172.30.0.0/16
FLANNEL_SUBNET=172.30.224.1/21
FLANNEL_MTU=1450
FLANNEL_IPMASQ=true
```

最大传输单元（Maximum Transmission Unit，MTU）是指一种通信协议的某一层上面所能通过的最大数据报大小（以字节为单位）。IP Masquerade 是 Linux 發展中的一種網路功能．如果一台 Linux 主機使用 IP Masquerade 功能連線到網際網路上，那麼接上它的電腦（不論是在同一個區域網路上或藉由數據機連線）也可以接觸網際網路，即使它們沒有獲得正式指定的 IP 位址．這使得一些電腦可以隱藏在閘道 (gateway) 系統後面存取網際網路而不被發現，看起來就像只有這個系統在使用網際網路．突破設定良好的偽裝 (masquerade) 系統之安全防護應該會比突破良好的封包過濾式防火牆 (packet filter firewall) 來得更加困難[17]．输入如下命令将 subnet.env 转写成 Docker 启动参数，创建好的启动参数默认生成在/run/docker_opts.env 文件中：

```
[root@ops001 conf.d] cat /run/docker_opts.env
DOCKER_OPTS=" --bip=172.30.224.1/21 --ip-masq=false --mtu=1450"
```

修改 docker 的服务启动文件如下：

```
# vim /lib/systemd/system/docker.service
```

[17]内容来源：http://www.e-infomax.com/ipmasq/howto-trans/zh/ipmasq-HOWTO-zh-2.html

```
2
3 EnvironmentFile=/run/docker_opts.env
4 ExecStart=/usr/bin/dockerd $DOCKER_OPTS -H fd://
```

重启 Docker，这时可以看到 docker0 的 ip 已经位于 flannel 网卡的网段之中。

1.1.4 集群状态检查

查看所有节点：

```
1 # 查看 pods
2 /opt/k8s/bin/kubectl get pods
3 # 查看集群健康状态
4 /opt/k8s/bin/kubectl get cs -o wide
5 # 查看 kube-apiserver 日志
6 journalctl -f -u kube-apiserver
7 kubectl logs kubernetes-dashboard-74d7cc788-mk9c7 -n kube-system
```

Bibliography