源码搭建 k8s

一、环境准备

1. 主机规划

172.30.170.76：k8s 的 master1 和 etcd

172.30.170.77：k8s 的 master2 和 etcd

172.30.170.78：k8s 的 master3 和 etcd

172.30.170.79：k8s 的 node1

172.30.170.80：k8s 的 node2

172.30.170.99：高可用的 vip

2. 各个组件使用的版本

k8s：kubernetes-v1.10.5

docker：docker-ce-17.03.3.ce-1.el7

etcd：etcd-3.3.11-2.el7

3. 配置相关的主机映射

3.1 为每台主机设置相应的永久主机名

hostnamectl set-hostname master1（master2、master3、node1、node2）

3.2 配置主机映射文件

echo “172.30.170.76 mater1

172.30.170.77 master2

172.30.170.78 master3

172.30.170.79 node1

172.30.170.80 node2

172.30.170.76 master” >> /etc/hosts

为了方便后面的操作可以设置 master1 对其他主机的免密登录，设置过程不赘述

4. 关闭防火墙、swap、selinux

systemctl stop firewalld

systemctl disable firewalld

vim /etc/selinux/config

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

swapoff -a

sed -i 's/.*swap.*/#&/' /etc/fstab

5. 调整内核参数

modprobe br_netfilter #加载内核模块

vim /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

sysctl -p /etc/sysctl.d/k8s.conf　#使配置文件生效

6. 修改系统资源配置文件，调整文件打开数等参数

echo "* soft nofile 655360" >> /etc/security/limits.conf

echo "* hard nofile 655360" >> /etc/security/limits.conf

echo "* soft nproc 655360" >> /etc/security/limits.conf

echo "* hard nproc 655360" >> /etc/security/limits.conf

echo "* soft memlock unlimited" >> /etc/security/limits.conf

echo "* hard memlock unlimited" >> /etc/security/limits.conf

echo "DefaultLimitNOFILE=1024000" >> /etc/systemd/system.conf

echo "DefaultLimitNPROC=1024000" >> /etc/systemd/system.conf

7. 配置国内 yum 源

yum install -y wget $ rm -rf　/etc/yum.repos.d/*　#安装下载工具，删除原有 yum 源

wget -O /etc/yum.repos.d/CentOS-Base.repo

http://mirrors.cloud.tencent.com/repo/centos7_base.repo # 配置基础 yum 源

wget -O /etc/yum.repos.d/epel.repo http://mirrors.cloud.tencent.com/repo/epel-7.repo # 配置企业附件 yum 源

yum-config-manager　--add-repo　https://download.docker.com/linux/centos/docker-ce.repo #添加 docker 的镜像源

yum clean all && yum makecache && yum repolist #刷新 yum 仓库记录

8. 安装可能会用到的依赖包

yum install　-y　conntrack ipvsadm ipset jq sysstat curl iptables libseccomp bash-completion yum-utils device-mapper-persistent-data lvm2 net-tools conntrack-tools vim libtool-ltdl

9. 配置时间同步，由于集群是分布式的所以一定要保证时间同步

yum install chrony －y

systemctl enable chronyd.service && systemctl start chronyd.service && systemctl status chronyd.service

chronyc sources

```
[root@master1 yum.repos.d]# chronyc sources
210 Number of sources = 4
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* ntp6.flashdance.cx           2   6    35     5  +5169us[-7022us] +/-  165ms
^+ 119.28.206.193               2   6    17    15  -2600us[  -15ms] +/-   58ms
^? ntp7.flashdance.cx           2   6     1    15    -15ms[  -15ms] +/-  161ms
^- ntp.wdc1.us.leaseweb.net     2   6    17    18    +85ms[  +73ms] +/-  491ms
[root@master1 yum.repos.d]#
```

使用 date 命令查看时间同步状态

10. 所有的环境准备已经完成，重启一下主机，然后再确认下相关配置

ps：以上操作需要在所有主机上进行。

ping 每个节点 hostname 看是否能 ping 通

执行 date 命令查看每个节点时间是否正确

执行 ulimit -Hn 看下最大文件打开数是否是 655360

cat /etc/sysconfig/selinux |grep disabled 查看下每个节点 selinux 是否都是 disabled 状态

等等

二、安装 docker

k8s-v1.10.x 版本支持的 docker 版本请查阅：

https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.10.md#external-dependencies

- The supported etcd server version is 3.1.12, as compared to 3.1.10 in v1.9 (#60998)
- The validated docker versions are the same as for v1.9: 1.11.2 to 1.13.1 and 17.03.x (ref)

1. 移除旧版本的 docker

yum remove -y docker docker-ce docker-common docker-selinux docker-engine

2. 列出 docker 版本并安装指定版本

yum list docker-ce --showduplicates | sort -r #列出 docker 版本

yum -y install docker-ce-17.03.3.ce-1.el7 #安装指定版本的 docker，有可能会安装失败，可以先把 docker-ce 和 docker-ce-selinux 下载下来再本地安装

wget https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-17.03.3.ce-1.el7.x86_64.rpm

wget https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-selinux-17.03.3.ce-1.el7.noarch.rpm

yum -y localinstall docker-ce-selinux-17.03.3.ce-1.el7.noarch.rpm && yum -y localinstall docker-ce-17.03.3.ce-1.el7.x86_64.rpm && docker version #安装 docker 并查看版本

```
Client:
 Version:      17.03.3-ce
 API version:  1.27
 Go version:   go1.7.5
 Git commit:   e19b718
 Built:        Thu Aug 30 01:06:10 2018
 OS/Arch:      linux/amd64
```

3. 启动并开机自启动 docker

systemctl start docker && systemctl enable docker

systemctl status docker #查看 docker 进程状态

```
[root@node1 ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2019-02-15 13:15:54 CST; 54s ago
     Docs: https://docs.docker.com
 Main PID: 5344 (dockerd)
   CGroup: /system.slice/docker.service
           ├─5344 /usr/bin/dockerd
           └─5350 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-contain...
```

三、创建 kubernetes 各组件需要使用的证书和密钥

1. 创建 k8s 的命令目录并放在$PATH 路径中（所有主机中）

mkdir -p /usr/k8s/bin

export PATH=/usr/k8s/bin:$PATH # 可将其写入到/etc/rc.local 中使其长久生效

2. 下载安装 cfssl（只需要在 master1 上操作）

wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64

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

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64

将下载的文件移动到/usr/k8s/bin 下，并改成较为好记的名字，添加执行权限

chmod +x cfssl_linux-amd64

chmod +x cfssljson_linux-amd64
chmod +x cfssl-certinfo_linux-amd64
mv cfssl_linux-amd64 /usr/k8s/bin/cfssl
mv cfssljson_linux-amd64 /usr/k8s/bin/cfssljson
mv cfssl-certinfo_linux-amd64 /usr/k8s/bin/cfssl-certinfo

3. 生成默认的配置文件和证书签名请求文件

cfssl print-defaults config > ca-config.json
cfssl print-defaults csr > ca-csr.json

4. 创建 ca

4.1 修改 ca-config.json 文件

cat ca-config.json

```
{
    "signing": {
        "default": {
            "expiry": "87600h"
        },
        "profiles": {
            "kubernetes": {
                "expiry": "87600h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            }
        }
    }
}
```

说明：

config.json：可以定义多个 profiles，分别指定不同的过期时间、使用场景等参数；后续在签名证书时使用某个 profile；

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 提供的证书进行校验；

client auth: 表示 server 可以用该 CA 对 client 提供的证书进行验证。

4.2 修改 ca-csr.json 文件

cat ca-csr.json

```
{
    "CN": "kubernetes",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
```

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

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)；

4.3 生成 ca 证书和私钥

将上面修改的文件均放在名为 ssl 的目录中，并在 ssl 目录中执行命令

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

```
[root@master1 ssl]# cfssl gencert -initca ca-csr.json | cfssljson -bare ca
2019/02/15 14:15:42 [INFO] generating a new CA key and certificate from CSR
2019/02/15 14:15:42 [INFO] generate received request
2019/02/15 14:15:42 [INFO] received CSR
2019/02/15 14:15:42 [INFO] generating key: rsa-2048
2019/02/15 14:15:43 [INFO] encoded CSR
2019/02/15 14:15:43 [INFO] signed certificate with serial number 383684172420340027770874695096895135288265473138
[root@master1 ssl]# ls
ca-config.json  ca.csr  ca-csr.json  ca-key.pem  ca.pem
[root@master1 ssl]#
```

5. 分发证书

将生成的 CA 证书、密钥文件、配置文件拷贝到所有机器的/etc/kubernetes/ssl 目录下面

mkdir -p /etc/kubernetes/ssl

cp ca* /etc/kubernetes/ssl

四、部署高可用的 etcd 集群

kubernetes 系统使用 etcd 存储所有的数据，我们这里部署 3 个节点的 etcd 集群，这 3 个节点直接复用 kubernetes master 的 3 个节点。

1. 直接 yum 安装 etcd

yum -y install etcd

2. 创建 tls 密钥和证书

为了保证通信安全，客户端(如 etcdctl)与 etcd 集群、etcd 集群之间的通信需要使用 TLS 加密。

2.1 创建 etcd 证书签名请求：

cd /etc/kubernetes/ssl

vim etcd-csr.json

```
{
  "CN": "etcd",
  "hosts": [
    "127.0.0.1",
    "${NODE_IP}" #当前节点 ip
  ],
  "key": {
```

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

2.2 生成 etcd 证书和私钥：

cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem -ca-key=/etc/kubernetes/ssl/ca-key.pem -config=/etc/kubernetes/ssl/ca-config.json -profile=kubernetes etcd-csr.json | cfssljson -bare etcd

```
[root@master1 ssl]# cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem  -ca-key=/etc/kubernetes/ssl
/ca-key.pem -config=/etc/kubernetes/ssl/ca-config.json  -profile=kubernetes etcd-csr.json | c
fssljson -bare etcd
2019/02/15 14:49:59 [INFO] generate received request
2019/02/15 14:49:59 [INFO] received CSR
2019/02/15 14:49:59 [INFO] generating key: rsa-2048
2019/02/15 14:50:01 [INFO] encoded CSR
2019/02/15 14:50:01 [INFO] signed certificate with serial number 260180658863635302707022147
8
71707324049917236238
2019/02/15 14:50:01 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitabl
e 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").
```

```
[root@master1 ssl]# ls etcd*
etcd.csr  etcd-csr.json  etcd-key.pem  etcd.pem
```

将生成的文件移动到 etcd/ssli 下面

mkdir -p /etc/etcd/ssl

mv etcd*.pem /etc/etcd/ssl/

3. 修改 etcd 配置文件

vim /etc/etcd/etcd.conf

```
#[Member]
#ETCD_CORS=""
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
#ETCD_WAL_DIR=""
ETCD_LISTEN_PEER_URLS="https://172.30.170.76:2380"
ETCD_LISTEN_CLIENT_URLS="http://localhost:2379,https://172.30.170.76:2379"
#ETCD_MAX_SNAPSHOTS="5"
#ETCD_MAX_WALS="5"
ETCD_NAME="db3"
#ETCD_SNAPSHOT_COUNT="100000"
#ETCD_HEARTBEAT_INTERVAL="100"
```

```
#ETCD_ELECTION_TIMEOUT="1000"
#ETCD_QUOTA_BACKEND_BYTES="0"
#ETCD_MAX_REQUEST_BYTES="1572864"
#ETCD_GRPC_KEEPALIVE_MIN_TIME="5s"
#ETCD_GRPC_KEEPALIVE_INTERVAL="2h0m0s"
#ETCD_GRPC_KEEPALIVE_TIMEOUT="20s"
#
#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://172.30.170.76:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://172.30.170.76:2379"
#ETCD_DISCOVERY=""
#ETCD_DISCOVERY_FALLBACK="proxy"
#ETCD_DISCOVERY_PROXY=""
#ETCD_DISCOVERY_SRV=""
ETCD_INITIAL_CLUSTER="db1=https://172.30.170.76:2380,db2=https://172.30.170.77:2380,db3
=https://172.30.170.78:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
#ETCD_STRICT_RECONFIG_CHECK="true"
#ETCD_ENABLE_V2="true"
#
#[Proxy]
#ETCD_PROXY="off"
#ETCD_PROXY_FAILURE_WAIT="5000"
#ETCD_PROXY_REFRESH_INTERVAL="30000"
#ETCD_PROXY_DIAL_TIMEOUT="1000"
#ETCD_PROXY_WRITE_TIMEOUT="5000"
#ETCD_PROXY_READ_TIMEOUT="0"
#
#[Security]
ETCD_CERT_FILE=""
ETCD_KEY_FILE=""
ETCD_CLIENT_CERT_AUTH="true"
ETCD_TRUSTED_CA_FILE=""
#ETCD_AUTO_TLS="false"
ETCD_PEER_CERT_FILE=""
ETCD_PEER_KEY_FILE=""
ETCD_PEER_CLIENT_CERT_AUTH="true"
ETCD_PEER_TRUSTED_CA_FILE=""
ETCD_PEER_AUTO_TLS="false"

#[Logging]
#ETCD_DEBUG="false"
#ETCD_LOG_PACKAGE_LEVELS=""
```

```
#ETCD_LOG_OUTPUT="default"
#
#[Unsafe]
#ETCD_FORCE_NEW_CLUSTER="false"
#
#[Version]
#ETCD_VERSION="false"
#ETCD_AUTO_COMPACTION_RETENTION="0"
#
#[Profiling]
#ETCD_ENABLE_PPROF="false"
#ETCD_METRICS="basic"
#
#[Auth]
#ETCD_AUTH_TOKEN="simple"
```

4. 修改 service 文件（必须修改，etcd 的 service 文件有点坑爹）

vim /usr/lib/systemd/system/etcd.service

```
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
EnvironmentFile=-/etc/etcd/etcd.conf
User=etcd
# set GOMAXPROCS to number of processors
ExecStart=/usr/bin/etcd --name=${ETCD_NAME} --data-dir=${ETCD_DATA_DIR} \
--cert-file=/etc/kubernetes/ssl/etcd.pem \
  --key-file=/etc/kubernetes/ssl/etcd-key.pem \
  --peer-cert-file=/etc/kubernetes/ssl/etcd.pem \
  --peer-key-file=/etc/kubernetes/ssl/etcd-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls ${ETCD_INITIAL_ADVERTISE_PEER_URLS} \
  --listen-peer-urls ${ETCD_LISTEN_PEER_URLS} \
  --listen-client-urls ${ETCD_LISTEN_CLIENT_URLS}\
  --advertise-client-urls ${ETCD_ADVERTISE_CLIENT_URLS} \
  --initial-cluster-token ${ETCD_INITIAL_CLUSTER_TOKEN} \
  --initial-cluster ${ETCD_INITIAL_CLUSTER} \
  --initial-cluster-state ${ETCD_INITIAL_CLUSTER_STATE}
Restart=on-failure
```

LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
在另外两台机器上重复 1-3 步骤，注意修改相对应的 ip。
启动 etcd
systemctl start etcd && systemctl enable etcd
ps：有可能会失败，通过 journalctl -xe 查看

```
[root@master2 etcd]# journalctl -xe
Feb 15 15:24:17 master2 etcd[5859]: The scheme of client url http://localhost:2379 is HTTP wh
Feb 15 15:24:17 master2 etcd[5859]: listening for client requests on localhost:2379
Feb 15 15:24:17 master2 etcd[5859]: open /etc/etcd/ssl/etcd-key.pem: permission denied
Feb 15 15:24:17 master2 systemd[1]: etcd.service: main process exited, code=exited, status=1/
Feb 15 15:24:17 master2 systemd[1]: Failed to start Etcd Server.
-- Subject: Unit etcd.service has failed
```

给 etcd-key.pem 添加读权限
chmod +r /etc/etcd/ssl/etcd-key.pem
再重新启动 etcd 即可
systemctl status etcd　#查看 etcd 进程状态
5.　查看 etcd 集群状态
在集群任意节点执行
etcdctl --ca-file=/etc/etcd/etcdSSL/ca.pem --cert-file=/etc/etcd/etcdSSL/etcd.pem
--key-file=/etc/etcd/etcdSSL/etcd-key.pem　member list
etcdctl --ca-file=/etc/etcd/etcdSSL/ca.pem --cert-file=/etc/etcd/etcdSSL/etcd.pem
--key-file=/etc/etcd/etcdSSL/etcd-key.pem cluster-health

```
[root@harbor-slave etcd]# etcdctl   --ca-file=/etc/etcd/etcdSSL/ca.pem   --cert-file=/etc/etcd/etcdSSL/etcd.pem   --key-file=/etc/etcd/etcdSSL/etcd-key.pem  cluster-health
member 25ab5ffda65fdb25 is healthy: got healthy result from https://172.30.170.71:2379
member 4266582b4b4545cd is healthy: got healthy result from https://172.30.170.60:2379
member 8575f8a1c950be42 is healthy: got healthy result from https://172.30.170.68:2379
cluster is healthy
[root@harbor-slave etcd]# etcdctl   --ca-file=/etc/etcd/etcdSSL/ca.pem   --cert-file=/etc/etcd/etcdSSL/etcd.pem   --key-file=/etc/etcd/etcdSSL/etcd-key.pem  member list
25ab5ffda65fdb25: name=db3 peerURLs=https://172.30.170.71:2380 clientURLs=https://172.30.170.71:2379 isLeader=true
4266582b4b4545cd: name=db1 peerURLs=https://172.30.170.60:2380 clientURLs=https://172.30.170.60:2379 isLeader=false
8575f8a1c950be42: name=db2 peerURLs=https://172.30.170.68:2380 clientURLs=https://172.30.170.68:2379 isLeader=false
[root@harbor-slave etcd]#
```

五、配置 kubectl 命令行工具

1.　下载 kubectl（kubectl 是和 kube-apiserver 进行交互的命令行工具，在集群中任意的节点安装都可以，著需要该节点有 kubectl 的二进制文件和~/.kube/config 文件即可（我现在 master1 上安装）
ps：kubectl 默认是从~/.kube/config 配置文件中读取 kube-apiserver 地址、证书、用户名等信息，所以需要正确配置该文件才能正常的使用 kubectl 命令
wget https://dl.k8s.io/v1.8.2/kubernetes-client-linux-amd64.tar.gz　#需要翻墙才能下载，可以下载到物理机上面在上传到虚拟机上（推荐一个工具 lrzsz，传文件十分方便，而且 yum 源可以直接安装）
tar -xf kubernetes-client-linux-amd64.tar.gz #解压缩，生成 kubernetes 目录
cp kubernetes/client/bin/* /usr/k8s/bin #将 kubectl 命令移动到我们的 PATH 变量中，方便使用
2.　创建 admin 证书
kubectl 与 kube-apiserver 的安全端口通信，需要为安全通信提供 TLS 证书和密钥。
2.1 创建 admin 证书签名请求：
vim /etc/kubernetes/ssl/admin-csr.json

```
{
  "CN": "admin",
```

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

后续 kube-apiserver 使用 RBAC 对客户端(如 kubelet、kube-proxy、Pod)请求进行授权

kube-apiserver 预定义了一些 RBAC 使用的 RoleBindings，如 cluster-admin 将 Group system:masters 与 Role cluster-admin 绑定，该 Role 授予了调用 kube-apiserver 所有 API 的权限

O 指定了该证书的 Group 为 system:masters，kubectl 使用该证书访问 kube-apiserver 时，由于证书被 CA 签名，所以认证通过，同时由于证书用户组为经过预授权的 system:masters，所以被授予访问所有 API 的劝降

hosts 属性值为空列表

2.2 生成 admin 证书和私钥：

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

```
[root@master1 ssl]# cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem -ca-key=/etc/kubernetes/ssl/
ca-key.pem -config=/etc/kubernetes/ssl/ca-config.json  -profile=kubernetes admin-csr.json | c
fssljson -bare admin
2019/02/15 16:22:13 [INFO] generate received request
2019/02/15 16:22:13 [INFO] received CSR
2019/02/15 16:22:13 [INFO] generating key: rsa-2048
2019/02/15 16:22:14 [INFO] encoded CSR
2019/02/15 16:22:14 [INFO] signed certificate with serial number 388583602129063793508111095O
23520970732264457215
2019/02/15 16:22:14 [WARNING] This certificate lacks a "hosts" field. This makes it unsuitabl
e 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").
[root@master1 ssl]# ls admin*
admin.csr  admin-csr.json  admin-key.pem  admin.pem
```

3. 创建 kubectl kubeconfig 文件

3.1 生成 TLS Bootstrapping 使用的 Token

head -c 16 /dev/urandom | od -An -t x | tr -d ' '

8d01427e640b99acf1a53ceab04ac2e0

3.2 设置集群参数

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=https://master:6443

3.3 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 --token=8d01427e640b99acf1a53ceab04ac2e0

3.4 设置上下文参数

kubectl config set-context kubernetes --cluster=kubernetes --user=admin

3.5 设置默认上下文

kubectl config use-context kubernetes

```
[root@master1 ssl]# head -c 16 /dev/urandom | od -An -t x | tr -d ' '
8d01427e640b99acf1a53ceab04ac2e0
[root@master1 ssl]# kubectl config set-cluster kubernetes --certificate-authority=/etc/kubern
etes/ssl/ca.pem --embed-certs=true --server=https://master:6443
Cluster "kubernetes" set.
[root@master1 ssl]# kubectl config set-credentials admin --client-certificate=/etc/kubernetes
/ssl/admin.pem --embed-certs=true --client-key=/etc/kubernetes/ssl/admin-key.pem --token=8d01
427e640b99acf1a53ceab04ac2e0
User "admin" set.
[root@master1 ssl]# kubectl config set-context kubernetes --cluster=kubernetes --user=admin
Context "kubernetes" created.
[root@master1 ssl]# kubectl config use-context kubernetes
Switched to context "kubernetes".
[root@master1 ssl]#
```

说明：admin.pem 证书 O 字段值为 system:masters，kube-apiserver 预定义的 RoleBinding cluster-admin 将 Group system:masters 与 Role cluster-admin 绑定，该 Role 授予了调用 kube-apiserver 相关 API 的权限

生成的 kubeconfig 被保存到 ~/.kube/config 文件，可以到~/.kube/config 文件中查看

若有其他继续需要运行 kubectl 命令，著需要将 kubectl 二进制文件和~/.kube/config 拷贝到相应的机器即可。

六、部署 master 节点

kubernetes master 节点包含的组件有：

kube-apiserver

kube-scheduler

kube-controller-manager

目前这 3 个组件需要部署到同一台机器上：（后面再部署高可用的 master）

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；

1. 下载二进制文件

wget https://dl.k8s.io/v1.8.2/kubernetes-server-linux-amd64.tar.gz

tar -xf kubernetes-server-linux-amd64.tar.gz #解压文件 ，生成 kubernetes 目录

cd kubernetes/server/bin

cp kube-apiserver kube-controller-manager kube-scheduler /usr/k8s/bin/

2. 创建 kubernetes 证书

2.1 创建签名请求：

cd /etckubernetes/ssl

vim kubernetes-scr.json

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

```
    "127.0.0.1",
    "172.30.170.76",
    "master",
    "10.254.0.1",
    "kubernetes",
    "kubernetes.default",
    "kubernetes.default.svc",
    "kubernetes.default.svc.cluster",
    "kubernetes.default.svc.cluster.local"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
```

说明：如果 hosts 字段不为空则需要指定授权使用该证书的 IP 或域名列表，所以上面分别指定了当前部署的 master 节点主机 IP 以及 apiserver 负载的内部域名

还需要添加 kube-apiserver 注册的名为 kubernetes 的服务 IP (Service Cluster IP)，一般是 kube-apiserver --service-cluster-ip-range 选项值指定的网段的第一个 IP，如 “10.254.0.1”

2.2 生成 kubernetes 证书和私钥

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

```
[root@master1 ssl]# cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem  -ca-key=/etc/kubernete
s/ssl/ca-key.pem -config=/etc/kubernetes/ssl/ca-config.json -profile=kubernetes kubernet
es-csr.json | cfssljson -bare kubernetes
2019/02/18 09:07:35 [INFO] generate received request
2019/02/18 09:07:35 [INFO] received CSR
2019/02/18 09:07:35 [INFO] generating key: rsa-2048
2019/02/18 09:07:36 [INFO] encoded CSR
2019/02/18 09:07:36 [INFO] signed certificate with serial number 463378422189497611823421399787627538358766119367
2019/02/18 09:07:36 [WARNING] 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").
[root@master1 ssl]# ls
admin.csr       admin.pem       ca-csr.json  kubernetes.csr       kubernetes.pem
admin-csr.json  ca-config.json  ca-key.pem   kubernetes-csr.json
admin-key.pem   ca.csr          ca.pem       kubernetes-key.pem
```

3. 配置和启动 kube-apiserver

3.1 创建 kube-apiserver 使用的 token 文件

kubelet 首次启动时向 kube-apiserver 发送 TLS Bootstrapping 请求，kube-apiserver 验证请求中的 token 是否与它配置的 token.csv 一致，如果一致则自动为 kubelet 生成证书和密钥。

vim /etc/kubernetes/token.csv

```
8d01427e640b99acf1a53ceab04ac2e0,kubelet-bootstrap,10001,"system:kubelet-bootstrap"
```

3.2 创建 kube-apiservere 的配置文件

vim /etc/kubernetes/kube-apiserver.conf

```
CONTROLL="--admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota"
ADDRESS="--advertise-address=172.30.170.76 --bind-address=0.0.0.0
--insecure-bind-address=172.30.170.76"
TOKEN="--enable-bootstrap-token-auth --token-auth-file=/etc/kubernetes/token.csv"
KUBE_CLUSTER_IP="--service-cluster-ip-range=10.254.0.0/16"
SERVICE_PORT_RANGE="--service-node-port-range=30000-60000"
TLS_FILE="--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_ENDPOINT="--etcd-servers=http://172.30.170.76:2379,http://172.30.170.77:2379,http://172.30.170.78:2379"
ETCD_SSL="--etcd-cafile=/etc/kubernetes/ssl/ca.pem
--etcd-certfile=/etc/kubernetes/ssl/kubernetes.pem
--etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem"
AUDIT="--audit-log-maxage=30 --audit-log-maxbackup=3 --audit-log-maxsize=100
--audit-log-path=/var/lib/audit.log --audit-policy-file=/etc/kubernetes/audit-policy.yaml"
ARGS="--authorization-mode=Node,RBAC --runtime-config=rbac.authorization.k8s.io/v1beta1
--kubelet-https=true --enable-swagger-ui=true --allow-privileged=true --event-ttl=1h
--logtostderr=true --v=6"
```

说明：如果你安装的是 1.9.x 版本的，一定要记住上面的参数 experimental-bootstrap-token-auth，需要替换成 enable-bootstrap-token-auth，因为这个参数

在 1.9.x 里面已经废弃掉了
kube-apiserver 1.6 版本开始使用 etcd v3 API 和存储格式
--authorization-mode=RBAC 指定在安全端口使用 RBAC 授权模式，拒绝未通过授权的请求
kube-scheduler、kube-controller-manager 一般和 kube-apiserver 部署在同一台机器上，它们使用非安全端口和 kube-apiserver 通信
kubelet、kube-proxy、kubectl 部署在其它 Node 节点上，如果通过安全端口访问 kube-apiserver，则必须先通过 TLS 证书认证，再通过 RBAC 授权
kube-proxy、kubectl 通过使用证书里指定相关的 User、Group 来达到通过 RBAC 授权的目的
如果使用了 kubelet TLS Boostrap 机制，则不能再指定 --kubelet-certificate-authority、--kubelet-client-certificate 和 --kubelet-client-key 选项，否则后续 kube-apiserver 校验 kubelet 证书时出现 ”x509: certificate signed by unknown authority“ 错误
--admission-control 值必须包含 ServiceAccount，否则部署集群插件时会失败
--bind-address 不能为 127.0.0.1
--service-cluster-ip-range 指定 Service Cluster IP 地址段，该地址段不能路由可达
--service-node-port-range=${NODE_PORT_RANGE} 指定 NodePort 的端口范围
缺省情况下 kubernetes 对象保存在 etcd/registry 路径下，可以通过 --etcd-prefix 参数进行调整
kube-apiserver 1.8 版本后需要在--authorization-mode 参数中添加 Node，即：--authorization-mode=Node,RBAC，否则 Node 节点无法注册
注意要开启审查日志功能，指定--audit-log-path 参数是不够的，这只是指定了日志的路径，还需要指定一个审查日志策略文件：--audit-policy-file，我们也可以使用日志收集工具收集相关的日志进行分析。
3.3 创建日志审计策略文件
vim /etc/kubernetes/audit-policy.yaml

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
- level: Metadata
```

这只是简单的审计文件，更多请查看：
https://kubernetes.io/docs/tasks/debug-application-cluster/audit/
3.4 创建 service 文件，并使用 systemd 启动 kube-apiserver
vim /usr/lib/systemd/system/kube-apiserver.service

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

[Service]
EnvironmentFile=/etc/kubernetes/kube-apiserver.conf
ExecStart=/usr/k8s/bin/kube-apiserver $CONTROLL $ADDRESS $TOKEN $KUBE_CLUSTER_IP
$SERVICE_PORT_RANGE  $TLS_FILE $ETCD_ENDPOINT $ETCD_SSL $AUDIT $ARGS
Restart=on-failure
RestartSec=5
```

Type=notify
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
systemctl enable kube-apiserver && systemctl start kube-apiserver #启动并开机自启
systemctl status kube-apiserver

```
[root@master1 ~]# systemctl status kube-apiserver
● kube-apiserver.service - Kubernetes API Server
   Loaded: loaded (/usr/lib/systemd/system/kube-apiserver.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Mon 2019-02-18 13:31:55 CST; 8min ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
 Main PID: 15520 (kube-apiserver)
   CGroup: /system.slice/kube-apiserver.service
           └─15520 /usr/k8s/bin/kube-apiserver NamespaceLifecycle,LimitRanger,Service...

Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.738931   15520 handle...ul
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.738980   15520 pathre...i/
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.739002   15520 handle...v1
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.741166   15520 wrap.g...6]
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.741473   15520 round_...ds
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.742162   15520 handle...ul
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.742185   15520 pathre...i/
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.742204   15520 handle...v1
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.745715   15520 wrap.g...6]
Feb 18 13:40:26 master1 kube-apiserver[15520]: I0218 13:40:26.746103   15520 round_...ds
Hint: Some lines were ellipsized, use -l to show in full.
[root@master1 ~]#
```

4. 配置和启动 kube-controller-manager

4.1 创建 kube-controller-manager 配置文件

vim /etc/kubernetes/kube-controller-manager.conf

DDRESS="--address=127.0.0.1"
MASTER="--master=http://master"
CLUSTER_IP="--service-cluster-ip-range=10.254.0.0/16"
CLUSTER_CIDR="--cluster-cidr=192.168.0.0/16"
TLS_FILE="--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"
ARGS="--cluster-name=kubernetes --leader-elect=true --v=2 --allocate-node-cidrs=true"

说明：--address 值必须为 127.0.0.1，因为当前 kube-apiserver 期望 scheduler 和 controller-manager 在同一台机器

--master=http://${MASTER_URL}:8080：使用 http(非安全端口)与 kube-apiserver 通信，需要下面的 haproxy 安装成功后才能去掉 8080 端口。

--cluster-cidr 指定 Cluster 中 Pod 的 CIDR 范围，该网段在各 Node 间必须路由可达(flanneld 保证)

--service-cluster-ip-range 参数指定 Cluster 中 Service 的 CIDR 范围，该网络在各 Node 间必须路由不可达，必须和 kube-apiserver 中的参数一致

--cluster-signing-* 指定的证书和私钥文件用来签名为 TLS BootStrap 创建的证书和私钥

--root-ca-file 用来对 kube-apiserver 证书进行校验，指定该参数后，才会在 Pod 容器的 ServiceAccount 中放置该 CA 证书文件

--leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程

4.2 创建 service 文件，并通过 systemd 启动服务

vim /usr/lib/systemd/system/kube-controller-manager.conf

```
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
EnvironmentFile=/etc/kubernetes/kube-controller-manager.conf
ExecStart=/usr/k8s/bin/kube-controller-manager    $ADDRESS $MASTER $CLUSTER_IP $CLUSTER_CIDR $TLS_FILE $ARGS
Restart=on-failure
RestartSec=5

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

systemctl daemon-reload 载入 service 文件

systemctl start kube-controller-manager && systemctl enable kube-controller-manager #启动并开机自启

systemctl status kube-controller-manager #查看状态

```
[root@master1 ~]# systemctl status kube-controller-manager.service
● kube-controller-manager.service - Kubernetes Controller Manager
   Loaded: loaded (/usr/lib/systemd/system/kube-controller-manager.service; disabled; ve
ndor preset: disabled)
   Active: active (running) since Mon 2019-02-18 14:21:57 CST; 18s ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
 Main PID: 16679 (kube-controller)
    Tasks: 6
   Memory: 13.9M
   CGroup: /system.slice/kube-controller-manager.service
           └─16679 /usr/k8s/bin/kube-controller-manager --address=127.0.0.1 --master=...

Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.328569   1667..."
Feb 18 14:22:14 master1 kube-controller-manager[16679]: W0218 14:22:14.328589   1667...d
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.328604   1667..."
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337588   1667...r
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337631   1667...r
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337716   1667...r
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337743   1667...r
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337917   1667...r
Feb 18 14:22:14 master1 kube-controller-manager[16679]: I0218 14:22:14.337944   1667...r
Feb 18 14:22:15 master1 kube-controller-manager[16679]: I0218 14:22:15.524942   1667...s
Hint: Some lines were ellipsized, use -l to show in full.
[root@master1 ~]#
```

5. 配置并启动 kube-scheduler

5.1 创建 kube-schedular 配置文件

vim /etc/kubernetes/kube-schedular.conf

```
ADDRSS="--address=127.0.0.1"
MASTER="--master=http://master:8080"
ARGS="--leader-elect=true --v=2"
```

说明：--address 值必须为 127.0.0.1，因为当前 kube-apiserver 期望 scheduler 和

controller-manager 在同一台机器

--master=http://${MASTER_URL}:8080：使用 http(非安全端口)与 kube-apiserver 通信，需要下面的 haproxy 启动成功后才能去掉 8080 端口

--leader-elect=true 部署多台机器组成的 master 集群时选举产生一处于工作状态的 kube-controller-manager 进程

5.2 创建 service 文件，通过 systemd 启动 kube-schedular

vim /usr/lib/systemd/system/kube-schedular.service

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

[Service]
EnvironmentFile=/etc/kubernetes/kube-schedular.conf
ExecStart=/usr/k8s/bin/kube-scheduler $ADDRESS $MASTER $ARGS
Restart=on-failure
RestartSec=5

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

systemctl daemon-reload #载入 service 文件

systemctl start kube-schedular && systemctl enable kube-schedular #启动并开机自启

systemctl statu kube-schedular#查看状态

```
[root@master1 ~]# systemctl status kube-schedular.service
● kube-schedular.service - Kubernetes Scheduler
   Loaded: loaded (/usr/lib/systemd/system/kube-schedular.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Mon 2019-02-18 14:29:32 CST; 6min ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
 Main PID: 16923 (kube-scheduler)
    Tasks: 6
   Memory: 10.2M
   CGroup: /system.slice/kube-schedular.service
           └─16923 /usr/k8s/bin/kube-scheduler --master=http://master:8080 --leader-e...

Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.063590   16923 flags...""
Feb 18 14:29:33 master1 kube-scheduler[16923]: W0218 14:29:33.063628   16923 server...P.
Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.066858   16923 server....5
Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.068739   16923 factor...r'
Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.068793   16923 factor...od
Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.074562   16923 server...51
Feb 18 14:29:33 master1 kube-scheduler[16923]: I0218 14:29:33.976331   16923 contro...er
Feb 18 14:29:34 master1 kube-scheduler[16923]: I0218 14:29:34.076575   16923 contro...er
Feb 18 14:29:34 master1 kube-scheduler[16923]: I0218 14:29:34.076679   16923 leader.....
Feb 18 14:29:34 master1 kube-scheduler[16923]: I0218 14:29:34.095002   16923 leader...er
Hint: Some lines were ellipsized, use -l to show in full.
[root@master1 ~]#
```

6. 验证 master 节点

kubectl get componentstatuses

```
[root@master1 ~]# kubectl get componentstatuses
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
etcd-1               Healthy   {"health":"true"}
etcd-2               Healthy   {"health":"true"}
controller-manager   Healthy   ok
etcd-0               Healthy   {"health":"true"}
[root@master1 ~]#
```

7. 在另外两台机器上按照同样的方式安装 kube-apiserver、kube-controller-manager、kube-scheduler。

七、配置 kube-apiserver 高可用

现在我们还是手动指定访问的 6443 和 8080 端口的，因为我们的域名 masterl 对应的 master01 节点直接通过 http 和 https 还不能访问，这里我们使用 haproxy 来代替请求。就是我们需要将 http 默认的 80 端口请求转发到 apiserver 的 8080 端口，将 https 默认的 443 端口请求转发到 apiserver 的 6443 端口，所以我们这里使用 haproxy 来做请求转发。通过 keepalived 实现高可用

1. 通过 haproxy 配置端口转发（三台 master 都要做，修改对应 ip 即可）

1.1 安装配置 haproxy

yum -y install haproxy

vim /etc/haproxy/haproxy.cfg

```
log                 127.0.0.1 local2
    chroot          /var/lib/haproxy
    pidfile         /var/run/haproxy.pid
    maxconn         4000
    user            haproxy
    group           haproxy
    daemon
    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
```

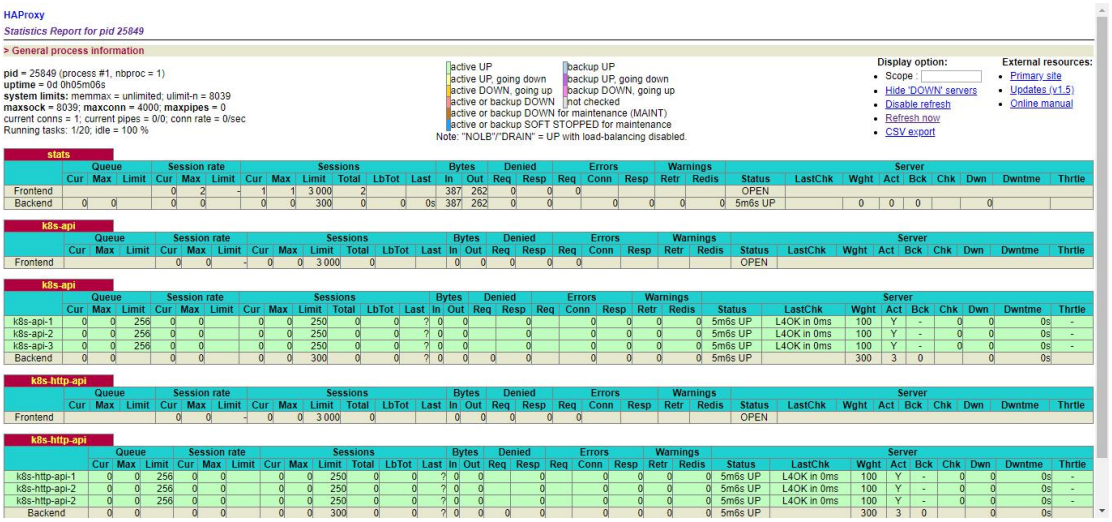
```
listen stats
    bind        *:9000
    mode        http
    stats       enable
    stats       hide-version
    stats       uri             /stats
    stats       refresh       30s
    stats       realm          Haproxy\ Statistics
    stats       auth            Admin:password
frontend k8s-api
        bind *:443
        mode tcp
        option tcplog
        tcp-request inspect-delay 5s
        tcp-request content accept if { req.ssl_hello_type 1 }
        default_backend k8s-api
backend k8s-api
        mode tcp
        option tcplog
        option tcp-check
        balance roundrobin
        default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256
weight 100
        server k8s-api-1 172.30.170.76:6443 check
        server k8s-api-2 172.30.170.77:6443 check
        server k8s-api-3 172.30.170.78:6443 check
frontend k8s-http-api
        bind *:80
        mode tcp
        option tcplog
        default_backend k8s-http-api
backend k8s-http-api
        mode tcp
        option tcplog
        option tcp-check
        balance roundrobin
        default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256
weight 100
        server k8s-http-api-1 172.30.170.76:8080 check
        server k8s-http-api-2 172.30.170.77:8080 check
        server k8s-http-api-2 172.30.170.78:8080 check
```

1.2 启动 haproxy，并检查状态

```
systemctl start haproxy && systemctlenable haproxy
ss -natulp | grep haproxy
```

```
[root@master1 kubernetes]# ss -natulp | grep haproxy
udp    UNCONN     0      0            *:53387                 *:*                   users:
("haproxy",pid=25849,fd=6),("haproxy",pid=25848,fd=6))
tcp    LISTEN     0      128          *:9000                  *:*                   users:
("haproxy",pid=25849,fd=5))
tcp    LISTEN     0      128          *:80                    *:*                   users:
("haproxy",pid=25849,fd=8))
tcp    LISTEN     0      128          *:443                   *:*                   users:
("haproxy",pid=25849,fd=7))
[root@master1 kubernetes]#
```

访问任意 masterip:9000/stats 查看 rs 状态：



看到如上访问页面我们就做好了 apiserver 的多活，下面我们使用 keepalived 配置 apiserver 的高可用

2. 安装配置 keepalived（三台都要配置，只要修改对应的 ip 和优先级即可，status 改成 BACKUP)

yum -y install keepalived

vim /etc/keepalived/keepalived.conf

```
! Configuration File for keepalived

global_defs {
   notification_email {
     acassen@firewall.loc
     failover@firewall.loc
     sysadmin@firewall.loc
   }
   notification_email_from Alexandre.Cassen@firewall.loc
   smtp_server 192.168.200.1
   smtp_connect_timeout 30
   router_id LVS_DEVEL
   vrrp_skip_check_adv_addr
   vrrp_garp_interval 0
   vrrp_gna_interval 0
}
```

```
vrrp_script check_haproxy {
    script "/usr/bin/killall -0 haproxy"
    interval 3
    weight -3
    user root     # 这里最好设置使用 root 用户，否则使用 keepalived 的自建用户是没有权限访问 haproxy 的，所以脚本总是执行失败的。
}
vrrp_instance haproxy-api {
    unicast_src_ip 172.30.170.76
    unicast_peer {
        172.30.170.77
        172.30.170.78
}
    state MASTER
    interface ens160
    virtual_router_id 51
    priority 101
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
}
virtual_ipaddress {
172.30.170.99
}
track_script {
check_haproxy
}
}
```

```
systemctl start keepalived && systemctl enable keepalived
systemctl statu keepalived
```

3. 测试 keealived 设置是否生效

停掉现在拿到 vip 的主机的的 haproxy 服务，看看 VIP 是否进行了相应的漂移。

到这里，我们就可以将上面的 6443 端口和 8080 端口去掉了，可以手动将 kubectl 生成的 config 文件(~/.kube/config)中的 server 地址 6443 端口去掉，另外 kube-controller-manager 和 kube-scheduler 的 –master 参数中的 8080 端口去掉了，然后分别重启这两个组件即可。

然后我们就可以将第一步在/etc/hosts 里面设置的域名对应的 IP 更改为我们的虚拟 IP 了

八、配置 kube-controller-manager 和 kube-scheduler 的高可用

Kubernetes 的管理层服务包括 kube-scheduler 和 kube-controller-manager。kube-scheduler 和 kube-controller-manager 使用一主多从的高可用方案，在同一时刻只允许一个服务处以具体的任务。Kubernetes 中实现了一套简单的选主逻辑，依赖 Etcd 实现 scheduler 和 controller-manager 的选主功能。如果 scheduler 和 controller-manager 在启动的时候设置了 leader-elect 参数，它们在启动后会先尝试获取 leader 节点身份，只有在获取 leader 节点身份后才可以执行具体的业务逻辑。它们分别会在 Etcd 中创建 kube-scheduler 和

kube-controller-manager 的 endpoint，endpoint 的信息中记录了当前的 leader 节点信息，以及记录的上次更新时间。leader 节点会定期更新 endpoint 的信息，维护自己的 leader 身份。每个从节点的服务都会定期检查 endpoint 的信息，如果 endpoint 的信息在时间范围内没有更新，它们会尝试更新自己为 leader 节点。scheduler 服务以及 controller-manager 服务之间不会进行通信，利用 Etcd 的强一致性，能够保证在分布式高并发情况下 leader 节点的全局唯一性。

当集群中的 leader 节点服务异常后，其它节点的服务会尝试更新自身为 leader 节点，当有多个节点同时更新 endpoint 时，由 Etcd 保证只有一个服务的更新请求能够成功。通过这种机制 sheduler 和 controller-manager 可以保证在 leader 节点宕机后其它的节点可以顺利选主，保证服务故障后快速恢复。当集群中的网络出现故障时对服务的选主影响不是很大，因为 scheduler 和 controller-manager 是依赖 Etcd 进行选主的，在网络故障后，可以和 Etcd 通信的主机依然可以按照之前的逻辑进行选主，就算集群被切分，Etcd 也可以保证同一时刻只有一个节点的服务处于 leader 状态。

我们已经在启动这两个服务时加上了相应的参数，无需做额外配置

九、部署 node 节点（以 node1 为例）

1. 下载 kubelet 和 kube-proxy 的相应版本的二进制包，并配置相关命令

wget https://dl.k8s.io/v1.8.2/kubernetes-server-linux-amd64.tar.gz

tar -xzvf kubernetes-server-linux-amd64.tar.gz

cd kubernetes

tar -xzvf kubernetes-src.tar.gz

sudo cp -r ./server/bin/{kube-proxy,kubelet} /usr/k8s/bin/ #将相关命令放到我们声明的 PATH 路径中

2. 创建 kubelet bootstapping kubeconfig 文件

2.1 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

2.2 置客户端认证参数

kubectl config set-credentials kubelet-bootstrap --token=${BOOTSTRAP_TOKEN} --kubeconfig=bootstrap.kubeconfig

2.3 设置上下文参数

kubectl config set-context default --cluster=kubernetes --user=kubelet-bootstrap --kubeconfig=bootstrap.kubeconfig

2.4 设置默认上下文

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

mv bootstrap.kubeconfig /etc/kubernetes/

说明：我们直接使用~/.kube/config 文件作为 kubelet 的 kubeconfig 文件的话，可以省略这部配置（我们选择直接使用 config 文件）

--embed-certs 为 true 时表示将 certificate-authority 证书写入到生成的 bootstrap.kubeconfig 文件中；

设置 kubelet 客户端认证参数时没有指定秘钥和证书，后续由 kube-apiserver 自动生成；

3. 创建 kubelet 的配置文件

vim /etc/kubernetes/kubelet.conf

```
ADDRESS="--address=172.30.170.79 --hostname-override=172.30.170.79"
KUBECONFIG="--experimental-bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig
```

--kubeconfig=/etc/kubernetes/config"
TLS="--cert-dir=/etc/kubernetes/ssl"
DNS="--cluster-dns=10.254.0.2 --cluster-domain='cluster.local.'"
ARGS="--network-plugin=cni --fail-swap-on=false --cgroup-driver=cgroupfs --hairpin-mode promiscuous-bridge --allow-privileged=true --serialize-image-pulls=false --logtostderr=true --v=2"

说明：--fail-swap-on 参数，这个一定要注意，Kubernetes 1.8 开始要求关闭系统的 Swap，如果不关闭，默认配置下 kubelet 将无法启动，也可以通过 kubelet 的启动参数 –fail-swap-on=false 来避免该问题

--cgroup-driver 参数，kubelet 用来维护主机的的 cgroups 的，默认是 cgroupfs，但是这个地方的值需要你根据 docker 的配置来确定（docker info |grep cgroup）

-address 不能设置为 127.0.0.1，否则后续 Pods 访问 kubelet 的 API 接口时会失败，因为 Pods 访问的 127.0.0.1 指向自己而不是 kubelet

如果设置了 --hostname-override 选项，则 kube-proxy 也需要设置该选项，否则会出现找不到 Node 的情况

--experimental-bootstrap-kubeconfig 指向 bootstrap kubeconfig 文件，kubelet 使用该文件中的用户名和 token 向 kube-apiserver 发送 TLS Bootstrapping 请求

管理员通过了 CSR 请求后，kubelet 自动在 --cert-dir 目录创建证书和私钥文件(kubelet-client.crt 和 kubelet-client.key)，然后写入 --kubeconfig 文件(自动创建 --kubeconfig 指定的文件)

建议在 --kubeconfig 配置文件中指定 kube-apiserver 地址，如果未指定 --api-servers 选项，则必须指定 --require-kubeconfig 选项后才从配置文件中读取 kue-apiserver 的地址，否则 kubelet 启动后将找不到 kube-apiserver (日志中提示未找到 API Server），kubectl get nodes 不会返回对应的 Node 信息

--cluster-dns 指定 kubedns 的 Service IP(可以先分配，后续创建 kubedns 服务时指定该 IP)，--cluster-domain 指定域名后缀，这两个参数同时指定后才会生效。

4. 创建 service 文件，并通过 systemd 启动

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
EnvironmentFile=/etc/kubernetes/kubelet.conf
ExecStart=/usr/k8s/bin/kubelet $ADDRESS $DNS $KUBECONFIG $ARGS $TLS
Restart=on-failure
RestartSec=5
[Install]
WantedBy=multi-user.target

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

```
[root@node1 ~]# systemctl sta
start   status
[root@node1 ~]# systemctl status kubelet
● kubelet.service - Kubernetes Kubelet
   Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2019-02-20 10:51:28 CST; 3h 31min ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
 Main PID: 22868 (kubelet)
    Tasks: 12
   Memory: 42.1M
   CGroup: /system.slice/kubelet.service
           └─22868 /usr/k8s/bin/kubelet --address=172.30.170.79 --hostname-override=172.30...

Feb 20 13:57:59 node1 kubelet[22868]: I0220 13:57:59.188238   22868 kubelet.go:2065] Fa...und
Feb 20 13:58:08 node1 kubelet[22868]: E0220 13:58:08.598574   22868 fsHandler.go:121] fail…ry
Feb 20 13:58:08 node1 kubelet[22868]: - exit status 1, rootInodeErr: cmd [find /var/lib/do…ry
Feb 20 13:58:08 node1 kubelet[22868]: ; err: exit status 1, extraDiskErr: du command faile…ry
Feb 20 13:58:08 node1 kubelet[22868]: - exit status 1
Feb 20 14:01:31 node1 kubelet[22868]: I0220 14:01:31.641270   22868 container_manager_l...ice
Feb 20 14:06:31 node1 kubelet[22868]: I0220 14:06:31.642214   22868 container_manager_l...ice
Feb 20 14:11:31 node1 kubelet[22868]: I0220 14:11:31.643199   22868 container_manager_l...ice
Feb 20 14:16:31 node1 kubelet[22868]: I0220 14:16:31.644260   22868 container_manager_l...ice
Feb 20 14:21:31 node1 kubelet[22868]: I0220 14:21:31.644896   22868 container_manager_l...ice
Hint: Some lines were ellipsized, use -l to show in full.
[root@node1 ~]#
```

5. 将 config 文件拷贝一份到我们配置文件指定的位置，查看效果。

cp ~/.kube/config /etc/kubernetes/

kubectl get nodes

```
[root@node1 ~]# kubectl get nodes
NAME            STATUS    ROLES     AGE       VERSION
172.30.170.79   Ready     <none>    1d        v1.10.5
```

此时我们就看到了 node 节点成功的加入到集群中了，实际上由于我们现在还饿米有配置集群的网络策略，所以 status 的状态应该时 notready，等配置好 calico 网络之后才会变成 ready。

6. 配置 kube-proxy

6.1 创建 kube-proxy 证书签名请求：

cd /etc/kubernetes/ssl

vim 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"
    }
  ]
}
```

说明：CN 指定该证书的 User 为 system:kube-proxy

kube-apiserver 预定义的 RoleBinding system:node-proxier 将 User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限

hosts 属性值为空列表。

6.2 生成 kube-proxy 客户端证书和私钥

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

```
[root@node2 ssl]# ls kube-proxy*
kube-proxy.csr  kube-proxy-csr.json  kube-proxy-key.pem  kube-proxy.pem
[root@node2 ssl]#
```

6.3 创建 kube-proxy kubeconfig 文件

cd /etc/kubernetes

a.设置集群参数

kubectl config set-cluster kubernetes --certificate-authority=/etc/kubernetes/ssl/ca.pem --embed-certs=true --server=https://master --kubeconfig=kube-proxy.kubeconfig

b.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

c.设置上下文参数

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

d.设置默认上下文

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

说明：

设置集群参数和客户端认证参数时 --embed-certs 都为 true，这会将 certificate-authority、client-certificate 和 client-key 指向的证书文件内容写入到生成的 kube-proxy.kubeconfig 文件中

kube-proxy.pem 证书中 CN 为 system:kube-proxy，kube-apiserver 预定义的 RoleBinding cluster-admin 将 User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限。

6.4 创建 kube-proxy 的配置文件

vim /etc/kubernetes/kube-proxy.conf

```
ADDRESS="--bind-address=172.30.170.79 --hostname-override=172.30.170.79"
CLUSTER_CIDR="--cluster-cidr=10.254.0.0/16"
KUBECONFIG="--kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig"
ARGS="--logtostderr=true --v=2"
```

说明：--hostname-override 参数值必须与 kubelet 的值一致，否则 kube-proxy 启动后会找不到该 Node，从而不会创建任何 iptables 规则

--cluster-cidr 必须与 kube-apiserver 的 --service-cluster-ip-range 选项值一致

kube-proxy 根据 --cluster-cidr 判断集群内部和外部流量，指定 --cluster-cidr 或 --masquerade-all 选项后 kube-proxy 才会对访问 Service IP 的请求做 SNAT

--kubeconfig 指定的配置文件嵌入了 kube-apiserver 的地址、用户名、证书、秘钥等请求和认证信息

预定义的 RoleBinding cluster-admin 将 User system:kube-proxy 与 Role system:node-proxier 绑定，该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限。

6.5 创建 service 文件，并通过 systemd 启动服务

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
EnvironmentFile=/etc/kubernetes/kube-proxy.conf
ExecStart=/usr/k8s/bin/kube-proxy $ADDRESS $CLUSTER_CIDR $KUBECONFIG $ARGS
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

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

systemctl daemon-reload

systemctl enable kube-proxy

sudo systemctl start kube-proxy

systemctl status kube-proxy

```
[root@node1 ~]# systemctl status kube-proxy
● kube-proxy.service - Kubernetes Kube-Proxy Server
   Loaded: loaded (/usr/lib/systemd/system/kube-proxy.service; enabled; vendor preset: disabl
ed)
   Active: active (running) since Wed 2019-02-20 10:56:23 CST; 3h 48min ago
     Docs: https://github.com/GoogleCloudPlatform/kubernetes
 Main PID: 24308 (kube-proxy)
    Tasks: 0
   Memory: 6.9M
   CGroup: /system.slice/kube-proxy.service
           ‣ 24308 /usr/k8s/bin/kube-proxy --bind-address=172.30.170.79 --hostname-overrid...

Feb 20 13:08:43 node1 kube-proxy[24308]: E0220 13:08:43.221877   24308 reflector.go:205...out
Feb 20 13:08:43 node1 kube-proxy[24308]: E0220 13:08:43.318443   24308 reflector.go:205...out
Feb 20 13:08:45 node1 kube-proxy[24308]: E0220 13:08:45.738411   24308 reflector.go:205...ope
Feb 20 13:20:16 node1 kube-proxy[24308]: E0220 13:20:16.773485   24308 streamwatcher.go...=""
Feb 20 13:20:16 node1 kube-proxy[24308]: E0220 13:20:16.773753   24308 streamwatcher.go...=""
Feb 20 13:20:16 node1 kube-proxy[24308]: W0220 13:20:16.937481   24308 reflector.go:341] k...
Feb 20 13:20:16 node1 kube-proxy[24308]: W0220 13:20:16.937905   24308 reflector.go:341] k...
Feb 20 13:20:27 node1 kube-proxy[24308]: E0220 13:20:27.940473   24308 reflector.go:205...out
Feb 20 13:20:27 node1 kube-proxy[24308]: E0220 13:20:27.940473   24308 reflector.go:205...out
Feb 20 13:20:32 node1 kube-proxy[24308]: E0220 13:20:32.154256   24308 reflector.go:205...ope
Hint: Some lines were ellipsized, use -l to show in full.
[root@node1 ~]#
```

7. node2 重复上面的步骤加入到集群中

查看集群节点：

kubectl get nodes -o wide （若还没配置 calico 网络，此时 status 应该是 notready）

```
[root@master1 kubernetes]# kubectl get nodes -o wide
NAME           STATUS    ROLES     AGE       VERSION   EXTERNAL-IP   OS-IMAGE                KERNEL-VERSION                CONTAINER-RUNTIME
172.30.170.79  Ready     <none>    1d        v1.10.5   <none>        CentOS Linux 7 (Core)   3.10.0-957.1.3.el7.x86_64   docker://17.3.3
172.30.170.80  Ready     <none>    1d        v1.10.5   <none>        CentOS Linux 7 (Core)   3.10.0-957.1.3.el7.x86_64   docker://17.3.3
[root@master1 kubernetes]#
```

建议：查看下我们的 serviceaccount 是否正常生成了，否则后面的插件部署和服务部署都会失败

kubectl get sa --all-namespaces

```
[root@master1 kubernetes]# kubectl get sa --all-namespaces
NAMESPACE     NAME                                 SECRETS   AGE
default       default                              1         1h
kube-public   default                              1         1h
```

显示出图中两个 default 即可。

8. 总结

至此我们就完成了 kubernetes 的集群高可用架构的部署，如果你想 master 节点也作为 node 节点运行 pod 的话只需要在 master 节点上按照上面步骤部署 kubelet 和 kube-proxy 即可，但是目前我们还没有部署集群的网络，所以还是不能正常的在集群中部署服务的，下面，我们就来部署 calico 网络

十、部署集群的 calico 网络

1. 环境以及镜像准备（所有 node 节点进行操作）

1.1 下载需要的镜像

docker pull calico/node:v3.1.4 && docker pull calico/cni:v3.1.4 &&docker pull calico/kube-controllers:v3.1.4 && docker pull mirrorgooglecontainers/pause-amd64:3.1

说明：前三个镜像是部署 calico 需要的，pause-amd 是 k8s 运行容器必须要的镜像

1.2 为下载的镜像打上标签

docker tag calico/node:v3.1.4 quay.io/calico/node:v3.1.4 && docker tag calico/cni:v3.1.4 quay.io/calico/cni:v3.1.4 && docker tag calico/kube-controllers:v3.1.4 quay.io/calico/kube-controllers:v3.1.4 && docker tag mirrorgooglecontainers/pause-amd64:3.1 k8s.gcr.io/pause-amd64:3.1

说明：我们通过 kubernetes 来安装 calico，需要将镜像的名字改成和 yaml 文件中要求的一样，所以关于 calico 的奖项都需要重新打赏标签，由于 kubernetes 只认 k8s.gcr.io/的镜像，但是这个镜像下载是需要翻墙的，所以我们从私有仓库下载到 pause 以后给他打上相应的标签以便使用。

2. 按照 calico 官网流程安装 calico（官网：https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/calico）

2.1 下载 rbac 文件并应用

wget https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/rbac.yaml

rbac.yaml 文件内容如下：

```
# Calico Version v3.1.4
# https://docs.projectcalico.org/v3.1/releases#v3.1.4

---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-kube-controllers
rules:
  - apiGroups:
    - ""
    - extensions
    resources:
```

```
      - pods
      - namespaces
      - networkpolicies
      - nodes
    verbs:
      - watch
      - list
  - apiGroups:
    - networking.k8s.io
    resources:
      - networkpolicies
    verbs:
      - watch
      - list
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-kube-controllers
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-kube-controllers
subjects:
- kind: ServiceAccount
  name: calico-kube-controllers
  namespace: kube-system

---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-node
rules:
  - apiGroups: [""]
    resources:
      - pods
      - nodes
    verbs:
      - get

---
```

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: calico-node
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-node
subjects:
- kind: ServiceAccount
  name: calico-node
  namespace: kube-system
```

kubectl apply -f rbac.yaml

kubectl get clusterroles --all-namespaces | grep calico

```
[root@node1 kubernetes]# kubectl get clusterroles  --all-namespaces | grep calico
calico-kube-controllers                                                4h
calico-node                                                            4h
[root@node1 kubernetes]#
```

kubectl get clusterrolebinding --all-namespaces | grep calico

```
[root@node1 kubernetes]# kubectl get clusterrolebinding  --all-namespaces | grep calico
calico-kube-controllers                                4h
calico-node                                            4h
```

2.2 下载 calico.yaml 文件

wget

https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/calico.yaml

vim calico.yaml

修改 ectd_endpoints 为自己的 etcd 集群地址：

```
 9 kind: ConfigMap
10 apiVersion: v1
11 metadata:
12   name: calico-config
13   namespace: kube-system
14 data:
15   # Configure this with the location of your etcd cluster.
16   etcd_endpoints: "http://172.30.170.77:2379,http://172.30.170.77:2379,http://172.30.170.
   78:2379"
17
```

修改 CALICO_IPV4POOL_CIDR 为我们设置的 serviceip 的网段。

```
150             - name: FELIX_DEFAULTENDPOINTTOHOSTACTION
151               value: "ACCEPT"
152             # The default IPv4 pool to create on startup if none exists. Pod IPs will be
153             # chosen from this range. Changing this value after installation will have
154             # no effect. This should fall within `--cluster-cidr`.
155             - name: CALICO_IPV4POOL_CIDR
156               value: "192.168.0.0/16"
157             - name: CALICO_IPV4POOL_IPIP
158               value: "Always"
```

2.2 应用 calico.yaml 文件并查看状态

kubectl apply -f calico.yaml

kubectl get po --all-namespaces

```
[root@master1 kubernetes]# kubectl get po --all-namespaces
NAMESPACE     NAME                                       READY     STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-94c6ddc99-85h7w    1/1       Running   0          2h
kube-system   calico-node-bjfc7                          2/2       Running   0          2h
kube-system   calico-node-kqdrj                          2/2       Running   0          2h
[root@master1 kubernetes]#
```

kubectl get ds --all-namespaces

```
[root@master1 kubernetes]# kubectl get ds --all-namespaces
NAMESPACE     NAME          DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
kube-system   calico-node   2         2         2         2            2           <none>          2h
[root@master1 kubernetes]#
```

kubectl get deploy --all-namespaces

```
[root@master1 kubernetes]# kubectl get deploy --all-namespaces
NAMESPACE     NAME                       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-system   calico-kube-controllers    1         1         1            1           2h
[root@master1 kubernetes]#
```

此文我们再查看 node 节点 status 就应该是 ready 了

kubectl get node

```
[root@master1 kubernetes]# kubectl get node
NAME            STATUS    ROLES     AGE       VERSION
172.30.170.79   Ready     <none>    1d        v1.10.5
172.30.170.80   Ready     <none>    1d        v1.10.5
[root@master1 kubernetes]#
```

至此，我们就完成了 calico 网络的部署。

此时我们的集群已经可以进行正常的工作了，下面我们再部署几个常用的插件

十一、kubedns 的部署

1. 环境和镜像准备

docker pull netonline/k8s-dns-kube-dns-amd64:1.14.10 #kubedns 容器：基于 skydns 实现；监视 k8s Service 资源并更新 DNS 记录；替换 etcd，使用 TreeCache 数据结构保存 DNS 记录并实现 SkyDNS 的 Backend 接口；接入 SkyDNS，对 dnsmasq 提供 DNS 查询服务。

docker pull netonline/k8s-dns-dnsmasq-nanny-amd64:1.14.10 #dnsmasq 容器：为集群提供 DNS 查询服务，即简易的 dns server；设置 kubedns 为 upstream；提供 DNS 缓存，降低 kubedns 负载，提高性能。

docker pull netonline/k8s-dns-sidecar-amd64:1.14.10 #sidecar 容器：监控健康模块，同时向外暴露 metrics 记录；定期检查 kubedns 和 dnsmasq 的健康状态；为 k8s 活性检测提供 HTTP API。

2. 下载创建 kube-dns 的 yaml 文件

wget -O kube-dns.yaml

https://raw.githubusercontent.com/kubernetes/kubernetes/v1.10.5/cluster/addons/dns/kube-dns.yaml.base

3. 修改 kube-dns.yaml 文件

vim kube-dns.yaml

修改 clusterIP 为 kubelet 配置文件中--cluster-dns 指定的 IP 地址

```yaml
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.254.0.2
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
```

修改 image 为我们下载好的镜像名，并修改 dns-domain 为我们在 kubelet 配置文件中 --cluster-domain 指定的 dns 域名（注意最后的“.”）

```yaml
        optional: true
    containers:
    - name: kubedns
      image: netonline/k8s-dns-kube-dns-amd64:1.14.10
      resources:
        # TODO: Set memory limits when we've profiled the container for large
```

```yaml
        timeoutSeconds: 5
      args:
      - --domain=cluster.local.
      - --dns-port=10053
      - --config-dir=/kube-dns-config
      - --v=2
```

```yaml
        mountPath: /kube-dns-config
    - name: dnsmasq
      image: netonline/k8s-dns-dnsmasq-nanny-amd64:1.14.10
      livenessProbe:
        httpGet:
          path: /healthcheck/dnsmasq
```

```yaml
      - --no-negcache
      - --log-facility=-
      - --server=/cluster.local./127.0.0.1#10053
      - --server=/in-addr.arpa/127.0.0.1#10053
      - --server=/ip6.arpa/127.0.0.1#10053
```

```yaml
      volumeMounts:
      - name: kube-dns-config
        mountPath: /etc/k8s/dns/dnsmasq-nanny
    - name: sidecar
      image: netonline/k8s-dns-sidecar-amd64:1.14.10
      livenessProbe:
```

```yaml
      args:
      - --v=2
      - --logtostderr
      - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local.,5,SRV
      - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,SRV
      ports:
```

4. 使用 kube-dns.yaml 文件创建 kube-dns

kubectl create -f kube-dns.yaml

```
[root@master1 kubernetes]# kubectl create -f kube-dns.yaml
service "kube-dns" created
serviceaccount "kube-dns" created
configmap "kube-dns" created
deployment.extensions "kube-dns" created
[root@master1 kubernetes]#
```

查看 kube-dns 生成的各组件状态：

kubectl get po --all-namespaces

```
[root@master1 kubernetes]# kubectl get po --all-namespaces
NAMESPACE     NAME                                      READY     STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-94c6ddc99-85h7w   1/1       Running   0          3h
kube-system   calico-node-bjfc7                         2/2       Running   0          3h
kube-system   calico-node-kqdrj                         2/2       Running   0          3h
kube-system   kube-dns-65d9f9cff9-ndvq5                 3/3       Running   0          2m
```

kubectl get deploy --all-namespaces

```
[root@master1 kubernetes]# kubectl get deploy --all-namespaces
NAMESPACE     NAME                      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-system   calico-kube-controllers   1         1         1            1           3h
kube-system   kube-dns                  1         1         1            1           1m
```

kubectl get svc --all-namespaces

```
[root@master1 kubernetes]# kubectl get svc --all-namespaces
NAMESPACE     NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
default       kubernetes   ClusterIP   10.254.0.1    <none>        443/TCP         2d
kube-system   kube-dns     ClusterIP   10.254.0.2    <none>        53/UDP,53/TCP   2m
[root@master1 kubernetes]#
```

5. 测试 dns 服务

创建任意 pod 进入查看 dns 地址

```
[root@master3 ~]# kubectl exec nginx-fzw5x  -n kube-system -it -- /bin/bash
root@nginx-fzw5x:/# cat /etc/resolv.conf
nameserver 10.254.0.2
search kube-system.svc.cluster.local. svc.cluster.local. cluster.local. vsphere.local
options ndots:5
```

十二、dashboard 的部署

1. 准备环境和镜像（所有 node 节点都需要下载）

docker pul lregistry.cn-hangzhou.aliyuncs.com/kubernete/kubernetes-dashboard-amd64 :v1.10.0

docker images

```
[root@master1 kubernetes]# docker images | grep dashboard
registry.cn-hangzhou.aliyuncs.com/kubernete/kubernetes-dashboard-amd64       v1.10.0             0dab2435c100        6 months ago        122 MB
[root@master1 kubernetes]#
```

2. 使用官网上下载的 dashboard 文件进行部署

2.1 下载官网的 yaml 部署文件

mkdir /etc/kubernetes/dashboard

cd /etc/kubernetes/dashboard

wget

https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml

2.3 修改 yaml 文件

vim kubernetes-dashboard.yaml

```
# ------------------- Dashboard Deployment ------------------- #

kind: Deployment
apiVersion: apps/v1beta2
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1                                   可以修改副本数量
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard                                  修改成我们下载的镜像
        image: registry.cn-hangzhou.aliyuncs.com/kubernete/kubernetes-dashboard-amd64:v1.10.0
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          - --heapster-host=http://heapster.kube-system:80   极其重要的参数，关系到dashboard能否连接heapster
          # Uncomment the following line to manually specify Kubernetes API server Host
          # If not specified, Dashboard will attempt to auto discover the API server and connect
```

特别说明：由于我们使用的镜像版本问题，不指定“--heapster-host=http://heapster.kube-system:80”，后面 heaspter 不能正常显示，这是版本的 bug，详情可以看：https://github.com/kubernetes/dashboard/pull/2181

```
# ------------------- Dashboard Service ------------------- #

kind: Service
apiVersion: v1
metadata:
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  type: NodePort
  ports:                                暴露服务，便于我们本地访问
    - port: 443
      targetPort: 8443
      nodePort: 30000
  selector:
    k8s-app: kubernetes-dashboard
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
```

3. 使用 yaml 文件部署 dashboard

kubectl apply -f . # 应用文件夹内的所有文件

4. 查看部署结果

kubectl get po --all-namespaces

```
[root@master1 dash-board]# kubectl get po --all-namespaces
NAMESPACE     NAME                                    READY     STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-94c6ddc99-85h7w   1/1       Running   0          20h
kube-system   calico-node-bjfc7                       2/2       Running   0          20h
kube-system   calico-node-kqdrj                       2/2       Running   0          20h
kube-system   kube-dns-65d9f9cff9-ndvq5               3/3       Running   0          17h
kube-system   kubernetes-dashboard-65f7b4f486-nhfzd   1/1       Running   0          1h
[root@master1 dash-board]# ^C
```

kubectl get svc --all-namespaces

```
[root@master1 dash-board]# kubectl get svc --all-namespaces
NAMESPACE     NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
default       kubernetes             ClusterIP   10.254.0.1      <none>        443/TCP          2d
kube-system   kube-dns               ClusterIP   10.254.0.2      <none>        53/UDP,53/TCP    17h
kube-system   kubernetes-dashboard   NodePort    10.254.126.88   <none>        443:30002/TCP    1h
[root@master1 dash-board]#
```

这里可以看到我们已经将 dashboard service 的 443 端口映射到 30002 端口了，那么我们去任意 node 节点查看下 30002 端口是否开启了呢：

ss -natulp | grep 30002

```
[root@node1 ~]# ss -natulp | grep 30002
tcp    LISTEN     0      128       :::30002                :::*                   users:(("kube-proxy",pid=24308,fd=9))
[root@node1 ~]#
```

同时我们也可以看到 node 节点时使用 kube-proxy 调度的端口。

5. 访问测试

在浏览器输入“http://<任意 node 节点 ip>:30002”进行访问，我使用的是 firefox，使用其他浏览器可能会出现不能访问的情况，如下图：

换一个浏览器试下，这个是浏览器的问题。

这里我们使用 token 进行访问，token 获取方式：

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

```
[root@master1 dash-board]# kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')
Name:         admin-user-token-vd5mr
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name=admin-user
              kubernetes.io/service-account.uid=126ef098-3574-11e9-982f-005056ba1b7c

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1359 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImtlYmVybmV0ZXMuaW8vc2Vyd
```

将这个 token 复制保存，以后都可以使用它来进行访问

至此我们就完成了 dashboard 插件的部署，由于缺少 Heapster 插件，当前 dashboard 不能展示 Pod、Nodes 的 CPU、内存等 metric 图形，下面我们就来部署 heapster 插件

十三、heapster 插件的部署

1. 环境和镜像准备

docker pul lregistry.cn-hangzhou.aliyuncs.com/google_containers/heapster-amd64:v1.5.3

docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/heapster-influxdb-amd64:v1.3.3

docker pull registry.cn-hangzhou.aliyuncs.com/google_containers/heapster-grafana-amd64:v4.4.3

2. 下载 heapster 的包

wget https://github.com/kubernetes/heapster/archive/v1.5.3.tar.gz

tar -xf v1.5.3.tar.gz

cd heapster-1.5.3/deploy/kube-config

mkdir /etc/kubernetes/heapster

mv influxdb/* rbac/* /etc/kubernetes/heapster

3. 修改相应的 yaml 文件

rbac 文件不需要修改，另外三个文件都只要把 image 修改成我们下载的版本即可

a. vim heapster.yaml

```
13    spec:
14      containers:
15      - name: grafana
16        image: registry.cn-hangzhou.aliyuncs.com/google_containers/heapster-grafana-amd64:v4.4.3
17        ports:
18        - containerPort: 3000
19          protocol: TCP
20        volumeMounts:
```

```
66    # You could also use NodePort to expose the service at a randomly-generated por
67    type: NodePort
68    ports:
69    - port: 80
70      targetPort: 3000
71      nodePort: 30003
72    selector:
73      k8s-app: grafana
```

暴露端口方便我们访问

b. vim heapster.yaml

```
    spec:
      serviceAccountName: heapster
      containers:
      - name: heapster
        image: registry.cn-hangzhou.aliyuncs.com/google_containers/heapster-amd64:v1.5.3
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes:https://kubernetes.default
        - --sink=influxdb:http://monitoring-influxdb.kube-system.svc:8086
```

c. vim influxdb.yaml

```
    spec:
      containers:
      - name: influxdb
        image: registry.cn-hangzhou.aliyuncs.com/google_containers/heapster-influxdb-amd64:v1.3.3
        volumeMounts:
        - mountPath: /data
          name: influxdb-storage
```

4. 应用文件并查看状态

kubectl apply -f .

kubectl get po -n kube-system

```
[root@master1 heapster]# kubectl get po -n kube-system
NAME                                    READY     STATUS    RESTARTS   AGE
calico-kube-controllers-94c6ddc99-85h7w   1/1       Running   0          1d
calico-node-bjfc7                       2/2       Running   0          1d
calico-node-kqdrj                       2/2       Running   0          1d
heapster-859b8c7677-tztk6               1/1       Running   0          15m
kube-dns-65d9f9cff9-ndvq5               3/3       Running   0          22h
kubernetes-dashboard-65f7b4f486-nhfzd   1/1       Running   0          7h
monitoring-grafana-5cf4fc6cbf-lstbl     1/1       Running   0          15m
monitoring-influxdb-65fd9f6cf-d8tjm     1/1       Running   0          15m
[root@master1 heapster]#
```

kubectl get deploy -n kube-system

```
[root@master1 heapster]# kubectl get deploy -n kube-system
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
calico-kube-controllers   1         1         1            1           1d
heapster               1         1         1            1           15m
kube-dns               1         1         1            1           22h
kubernetes-dashboard   1         1         1            1           7h
monitoring-grafana     1         1         1            1           15m
monitoring-influxdb    1         1         1            1           15m
[root@master1 heapster]#
```

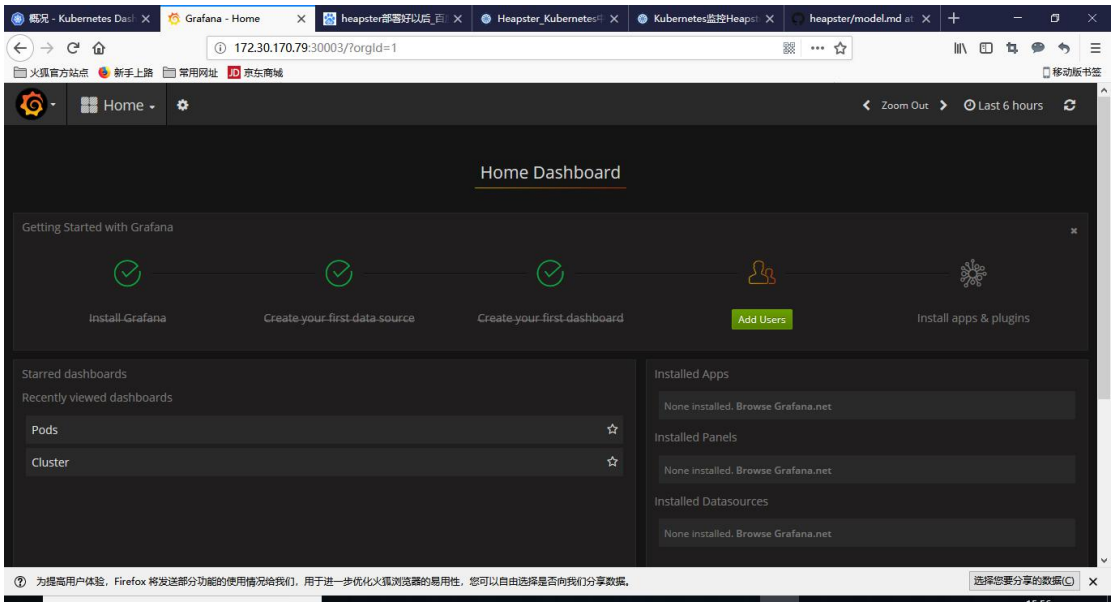
kubectl get svc -n kube-system

```
[root@master1 heapster]# kubectl get svc -n kube-system
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
heapster               ClusterIP   10.254.77.144    <none>        80/TCP           16m
kube-dns               ClusterIP   10.254.0.2       <none>        53/UDP,53/TCP    22h
kubernetes-dashboard   NodePort    10.254.126.88    <none>        443:30002/TCP    7h
monitoring-grafana     NodePort    10.254.76.223    <none>        80:30003/TCP     16m
monitoring-influxdb    ClusterIP   10.254.205.134   <none>        8086/TCP         16m
[root@master1 heapster]#
```

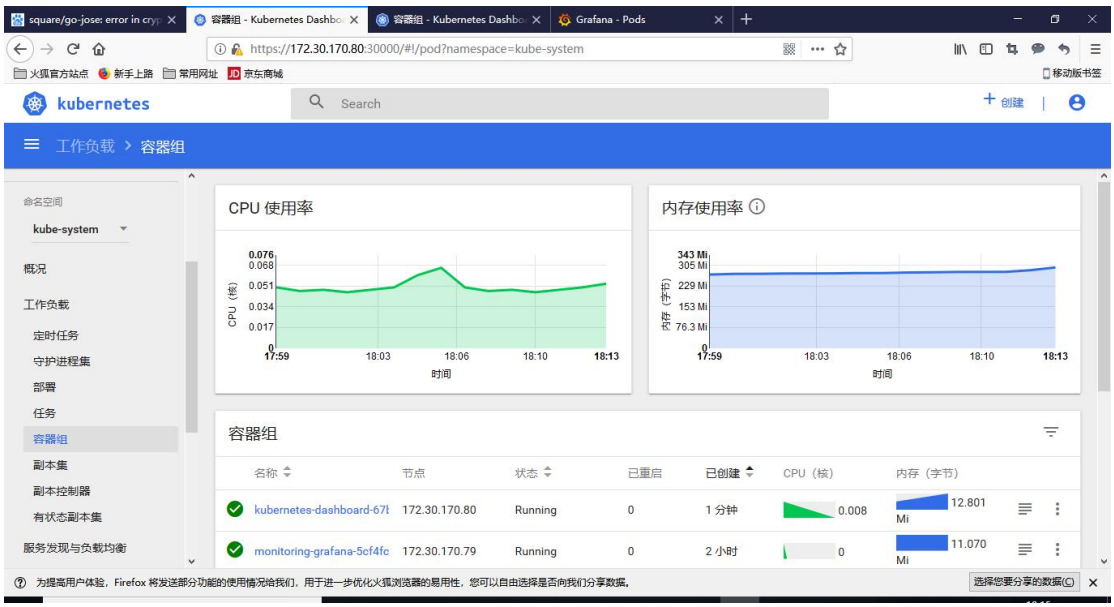
5. 访问测试

5.1 访问 grafana

地址栏输入“<任意 node 节点 ip>:30003”进入主页



此时 dashboard 也可以显示相应的 cpu 和内存使用了



十四、安装 ingress（使用 treafik）

参考 https://www.qikqiak.com/post/manual-install-high-available-kubernetes-cluster/#%E6%A3%80%E6%9F%A5%E6%89%A7%E8%A1%8C%E7%BB%93%E6%9E%9C 中安装 ingress 章节，或者我写的另外一份文档：k8s 使用 traefik ingress 暴露服务.docx