Kubernetes二进制高可用方案

目录

文档信息

文档作者	房佳亮
文档版本	**Version1.0**
文档版权	内部资料禁止传播
文档归类	**Kubernetes 项目实战训练营**
系统环境	**CentOS-7.X-x86_64**
作者邮箱	**crushlinux@163.com**
修订信息	**2021-04-02**
技术交流	

文档约定

[绿色背景]　知识重点
[红色背景]　错误警告
[黄色背景]　注意事项

执行命令

1、Kubernetes 高可用概述

Kubernetes 高可用是保证 Master 节点中的 API Server 服务的高可用。API Server 提供了 Kubernetes 各类资源对象增删改查的唯一访问入口，是整个 Kubernetes 系统的数据总线和数据中心。采用负载均衡（Load Balance）连接两个 Master 节点可以提供稳定容器云业务。本章主要学习 Kubernetes 高可用的部署方法。

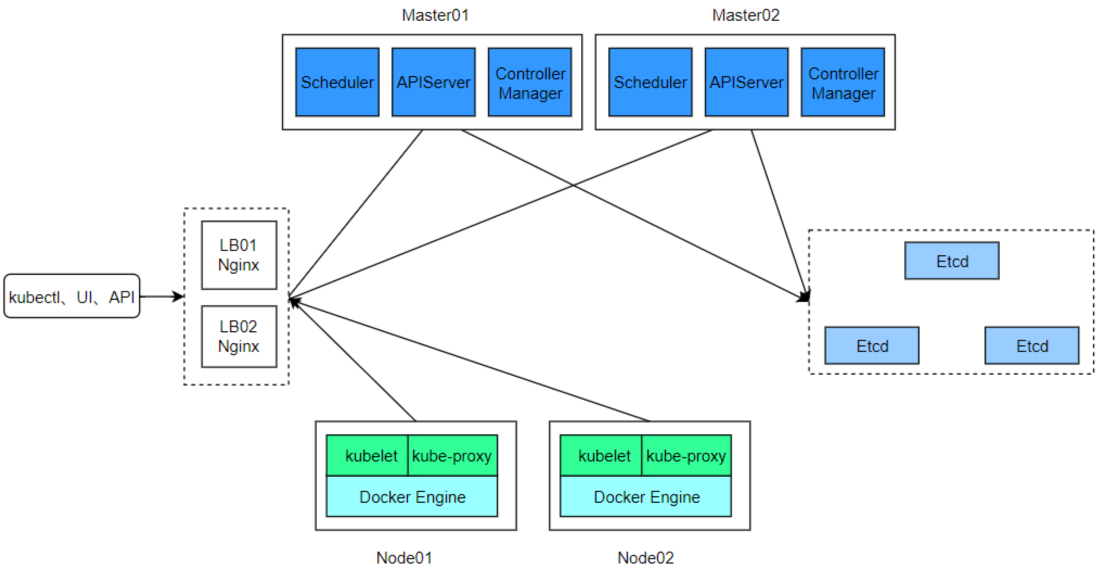
1.1、Kubernetes 高可用主机分配

主机名	IP 地址	操作系统	主要软件
k8s-master01	192.168.200.111	CentOS7.x	Etcd+Kubernetes
k8s-master02	192.168.200.112	CentOS7.x	Etcd+Kubernetes
k8s-node01	192.168.200.113	CentOS7.x	Etcd+Kubernetes+Flannel+Docker
k8s-node02	192.168.200.114	CentOS7.x	Etcd+Kubernetes+Flannel+Docker
k8s-lb01	192.168.200.115	CentOS7.x	Nginx+Keepalived

k8s-lb02	192.168.200.116	CentOS7.x	Nginx+Keepalived

LB 群集 VIP 地址为 192.168.200.200。

1.2、Kubernetes 高可用架构拓扑



2、高可用架构部署

2.1、基础环境配置

（1）配置基础网络信息

为所有主机配置 IP 地址、网关、DNS（建议配置阿里云的 223.5.5.5）等基础网络信息。建议主机设置为静态 IP 地址，避免因为 IP 地址变化出现群集中无法连接 API Server 的现象，导致 Kubernetes 群集不可用。

（2）配置主机名与地址解析记录

为所有主机配置主机名并添加地址解析记录，下面以 k8s-master01 主机为例进行操作演示。

```
[root@localhost ~]# hostname k8s-master01
[root@localhost ~]# bash

[root@k8s-master01 ~]# cat << EOF >> /etc/hosts
192.168.200.111 k8s-master01
192.168.200.112 k8s-master02
192.168.200.113 k8s-node01
192.168.200.114 k8s-node02
192.168.200.115 k8s-lb01
```

```
192.168.200.116 k8s-lb02
EOF
```

（3）禁用防火墙与 Selinux

```
[root@k8s-master01 ~]# iptables -F
[root@k8s-master01 ~]# systemctl stop firewalld
[root@k8s-master01 ~]# systemctl disable firewalld
[root@k8s-master01 ~]# setenforce 0
[root@k8s-master01 ~]# sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

2.2、部署集群证书

在 k8s-master01 主机上创建的目录“/k8s”，并将准备好的脚本文件 etcd-cert.sh 和 etcd.sh 上传至/k8s 目录中。其中 etcd-cert.sh 脚本是 Etcd 证书创建的脚本；etcd.sh 脚本是 Etcd 服务脚本，包含配置文件及启动脚本。

```
[root@k8s-master01 ~]# mkdir /k8s
[root@k8s-master01 ~]# cd /k8s
[root@k8s-master01 k8s]# ls
etcd-cert.sh    etcd.sh
```

创建目录/k8s/etcd-cert，证书全部存放至该目录中，方便管理。

```
[root@k8s-master01 k8s]# mkdir /k8s/etcd-cert
[root@k8s-master01 k8s]# mv /k8s/etcd-cert.sh /k8s/etcd-cert
```

上传 cfssl、cfssl-certinfo、cfssljson 软件包。部署到/usr/local/bin 目录下并配置执行权限

```
[root@k8s-master01 k8s]# ls
cfssl    cfssl-certinfo    cfssljson    etcd-cert    etcd.sh
[root@k8s-master01 k8s]# mv cfssl* /usr/local/bin/
[root@k8s-master01 k8s]# chmod +x /usr/local/bin/cfssl*
[root@k8s-master01 k8s]# ls -l /usr/local/bin/cfssl*
-rwxr-xr-x 1 root root 10376657 7 月    21 2020 /usr/local/bin/cfssl
-rwxr-xr-x 1 root root    6595195 7 月    21 2020 /usr/local/bin/cfssl-certinfo
-rwxr-xr-x 1 root root    2277873 7 月    21 2020 /usr/local/bin/cfssljson
```

创建 CA 和 Server 证书

```
[root@k8s-master01 k8s]# cd /k8s/etcd-cert/
[root@k8s-master01 etcd-cert]# cat etcd-cert.sh
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "www": {
         "expiry": "87600h",
         "usages": [
            "signing",
            "key encipherment",
            "server auth",
            "client auth"
        ]
      }
    }
  }
}
EOF

cat > ca-csr.json <<EOF
{
    "CN": "etcd CA",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "L": "Beijing",
            "ST": "Beijing"
        }
    ]
}
EOF

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

#----------------------

cat > server-csr.json <<EOF
{
    "CN": "etcd",
    "hosts": [
    "192.168.200.111",
    "192.168.200.112",
    "192.168.200.113",
    "192.168.200.114"
```

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

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=www server-
csr.json | cfssljson -bare server

[root@k8s-master01 etcd-cert]# bash etcd-cert.sh
2021/01/21 23:47:32 [INFO] generating a new CA key and certificate from CSR
2021/01/21 23:47:32 [INFO] generate received request
2021/01/21 23:47:32 [INFO] received CSR
2021/01/21 23:47:32 [INFO] generating key: rsa-2048
2021/01/21 23:47:32 [INFO] encoded CSR
2021/01/21 23:47:32 [INFO] signed certificate with serial number
116674972491031521074903654614645798533990711359
2021/01/21 23:47:32 [INFO] generate received request
2021/01/21 23:47:32 [INFO] received CSR
2021/01/21 23:47:32 [INFO] generating key: rsa-2048
2021/01/21 23:47:33 [INFO] encoded CSR
2021/01/21 23:47:33 [INFO] signed certificate with serial number
450967638080328676198204931089702412865625644325
2021/01/21 23:47:33 [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@k8s-master01 etcd-cert]# ls
ca-config.json   ca.csr   ca-csr.json   ca-key.pem   ca.pem   etcd-cert.sh   server.csr   server-
csr.json   server-key.pem   server.pem
```

2.3、部署 Etcd 集群

2.3.1、准备 Etcd 相关工具与启动所需证书

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

上传 etcd-v3.3.18-linux-amd64.tar.gz 软件包

```
[root@k8s-master01 k8s]# tar xf etcd-v3.3.18-linux-amd64.tar.gz
[root@k8s-master01 k8s]# ls etcd-v3.3.18-linux-amd64
Documentation    etcdctl                    README.md
etcd                   README-etcdctl.md    READMEv2-etcdctl.md
[root@k8s-master01 k8s]# mkdir /opt/etcd/{cfg,bin,ssl} -p
[root@k8s-master01 k8s]# cd etcd-v3.3.18-linux-amd64/
[root@k8s-master01 etcd-v3.3.18-linux-amd64]# mv etcd etcdctl /opt/etcd/bin/
[root@k8s-master01 etcd-v3.3.18-linux-amd64]# cp /k8s/etcd-cert/*.pem /opt/etcd/ssl/
[root@k8s-master01 etcd-v3.3.18-linux-amd64]# ls /opt/etcd/ssl/
ca-key.pem    ca.pem    server-key.pem    server.pem
```

2.3.2、部署 Etcd 集群

```
[root@k8s-master01 etcd-v3.3.18-linux-amd64]# cd /k8s/
[root@k8s-master01 k8s]# bash etcd.sh etcd01 192.168.200.111
etcd02=https://192.168.200.112:2380,etcd03=https://192.168.200.113:2380,etcd04=https://1
92.168.200.114:2380
Created symlink from /etc/systemd/system/multi-user.target.wants/etcd.service to
/usr/lib/systemd/system/etcd.service.
```

执行时会卡在启动 etcd 服务上，实际已经启动 Ctrl+C 终止就行。

```
[root@k8s-master01 k8s]# scp -r /opt/etcd/ root@k8s-master02:/opt/
[root@k8s-master01 k8s]# scp -r /opt/etcd/ root@k8s-node01:/opt/
[root@k8s-master01 k8s]# scp -r /opt/etcd/ root@k8s-node02:/opt/

[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/etcd.service root@k8s-
master02:/usr/lib/systemd/system/
[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/etcd.service root@k8s-
node01:/usr/lib/systemd/system/
[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/etcd.service root@k8s-
node02:/usr/lib/systemd/system/
```

```
[root@k8s-master02 ~]# cat /opt/etcd/cfg/etcd
#[Member]
```

```
ETCD_NAME="etcd02"
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="https://192.168.200.112:2380"
ETCD_LISTEN_CLIENT_URLS="https://192.168.200.112:2379"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.200.112:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.200.112:2379"
ETCD_INITIAL_CLUSTER="etcd01=https://192.168.200.111:2380,etcd02=https://192.168.200.112:2380,etcd03=https://192.168.200.113:2380,etcd04=https://192.168.200.114:2380"ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

```
[root@k8s-node01 ~]# cat /opt/etcd/cfg/etcd
#[Member]
ETCD_NAME="etcd03"
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="https://192.168.200.113:2380"
ETCD_LISTEN_CLIENT_URLS="https://192.168.200.113:2379"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.200.113:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.200.113:2379"
ETCD_INITIAL_CLUSTER="etcd01=https://192.168.200.111:2380,etcd02=https://192.168.200.112:2380,etcd03=https://192.168.200.113:2380,etcd04=https://192.168.200.114:2380"ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

```
[root@k8s-node02 ~]# cat /opt/etcd/cfg/etcd
#[Member]
ETCD_NAME="etcd04"
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="https://192.168.200.114:2380"
ETCD_LISTEN_CLIENT_URLS="https://192.168.200.114:2379"

#[Clustering]
ETCD_INITIAL_ADVERTISE_PEER_URLS="https://192.168.200.114:2380"
ETCD_ADVERTISE_CLIENT_URLS="https://192.168.200.114:2379"
ETCD_INITIAL_CLUSTER="etcd01=https://192.168.200.111:2380,etcd02=https://192.168.200.112:2380,etcd03=https://192.168.200.113:2380,etcd04=https://192.168.200.114:2380"ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

master01、master02、node01、node02 这 4 台主机上均执行以下操作

```
[root@k8s-master01 k8s]# systemctl daemon-reload
[root@k8s-master01 k8s]# systemctl restart etcd
[root@k8s-master01 k8s]# systemctl enable etcd
```

2.4、部署 APIServer 组件

2.4.1、创建所需证书

上传并解压 master.zip 包后会生成三个脚本：apiserver.sh、controller-manager.sh、及 scheduler.sh。为脚本文件添加执行权限，后面每一个服务的启动都要依赖于这三个脚本。

```
[root@k8s-master01 ~]# cd /k8s/
[root@k8s-master01 k8s]# unzip master.zip
Archive:  master.zip
  inflating: apiserver.sh
  inflating: controller-manager.sh
  inflating: scheduler.sh
[root@k8s-master01 k8s]# chmod +x *.sh
```

创建/k8s/k8s-cert 目录，作为证书自签的工作目录，将所有证书都生成到此目录中。在/k8s/k8s-cert 目录中创建证书生成脚本 k8s-cert.sh，脚本内容如下所示。执行 k8s-cert.sh 脚本即可生成 CA 证书、服务器端的私钥、admin 证书、proxy 代理端证书。

```
[root@k8s-master01 k8s]# mkdir /k8s/k8s-cert
[root@k8s-master01 k8s]# cd /k8s/k8s-cert

[root@k8s-master01 k8s-cert]# vim k8s-cert.sh
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "kubernetes": {
         "expiry": "87600h",
         "usages": [
            "signing",
            "key encipherment",
            "server auth",
            "client auth"
        ]
      }
    }
```

```
   }
}
EOF

cat > ca-csr.json <<EOF
{
        "CN": "kubernetes",
        "key": {
                "algo": "rsa",
                "size": 2048
        },
        "names": [
                {
                        "C": "CN",
                        "L": "Beijing",
                        "ST": "Beijing",
                        "O": "k8s",
                        "OU": "System"
                }
        ]
}
EOF

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

#----------------------

cat > server-csr.json <<EOF
{
        "CN": "kubernetes",
        "hosts": [
            "10.0.0.1",
            "127.0.0.1",
            "192.168.200.111",
            "192.168.200.112",
            "192.168.200.113",
            "192.168.200.114",
            "192.168.200.200",
            "kubernetes",
            "kubernetes.default",
            "kubernetes.default.svc",
            "kubernetes.default.svc.cluster",
            "kubernetes.default.svc.cluster.local"
        ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "L": "BeiJing",
            "ST": "BeiJing",
            "O": "k8s",
            "OU": "System"
        }
    ]
}
EOF

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

#----------------------

cat > admin-csr.json <<EOF
{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "BeiJing",
      "ST": "BeiJing",
      "O": "system:masters",
      "OU": "System"
    }
  ]
}
EOF

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

```
#----------------------

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

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

执行 k8s-cert.sh 脚本会生成 8 张证书。

```
[root@k8s-master01 k8s-cert]# bash k8s-cert.sh
2021/01/22 00:12:59 [INFO] generating a new CA key and certificate from CSR
2021/01/22 00:12:59 [INFO] generate received request
2021/01/22 00:12:59 [INFO] received CSR
2021/01/22 00:12:59 [INFO] generating key: rsa-2048
2021/01/22 00:13:00 [INFO] encoded CSR
2021/01/22 00:13:00 [INFO] signed certificate with serial number
619453345084307609137096056260289738127015485732
2021/01/22 00:13:00 [INFO] generate received request
2021/01/22 00:13:00 [INFO] received CSR
2021/01/22 00:13:00 [INFO] generating key: rsa-2048
2021/01/22 00:13:00 [INFO] encoded CSR
2021/01/22 00:13:00 [INFO] signed certificate with serial number
234629517018162017464882890509227737282750877254
2021/01/22 00:13:00 [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").
2021/01/22 00:13:00 [INFO] generate received request
2021/01/22 00:13:00 [INFO] received CSR
2021/01/22 00:13:00 [INFO] generating key: rsa-2048
2021/01/22 00:13:00 [INFO] encoded CSR
2021/01/22 00:13:00 [INFO] signed certificate with serial number
714372928223821030618976615726024042558311606716
2021/01/22 00:13:00 [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").
2021/01/22 00:13:00 [INFO] generate received request
2021/01/22 00:13:00 [INFO] received CSR
2021/01/22 00:13:00 [INFO] generating key: rsa-2048
2021/01/22 00:13:01 [INFO] encoded CSR
2021/01/22 00:13:01 [INFO] signed certificate with serial number
425118236835992989931843838578972299860432626484
2021/01/22 00:13:01 [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@k8s-master01 k8s-cert]# ls *.pem
admin-key.pem  admin.pem  ca-key.pem  ca.pem  kube-proxy-key.pem  kube-proxy.pem
server-key.pem  server.pem
[root@k8s-master01 k8s-cert]# ls *.pem | wc -l
8
```

证书生成以后，需要将其中的 CA 与 Server 相关证书拷贝到 Kubernetes 的工作目录。创建/opt/kubernetes/{cfg,bin,ssl}目录 ，分别用于存放配置文件、可执行文件以及证书文件。

```
[root@k8s-master01 ~]# mkdir /opt/kubernetes/{cfg,bin,ssl} -p
[root@k8s-master01 ~]# cd /k8s/k8s-cert/
[root@k8s-master01 k8s-cert]# cp ca*pem server*pem /opt/kubernetes/ssl/
[root@k8s-master01 k8s-cert]# ls /opt/kubernetes/ssl/
ca-key.pem  ca.pem  server-key.pem  server.pem
```

2.4.2、部署 APIServer 组件

上传并解压 Kubernetes 软件压缩包，将压缩包中的 kube-apiserver、kubectl、kube-controller-manager 与 kube-scheduler 组件的脚本文件拷贝到/opt/kubernetes/bin/目录下。

```
[root@k8s-master01 ~]# cd /k8s/
[root@k8s-master01 k8s]# tar xf kubernetes-server-linux-amd64.tar.gz
[root@k8s-master01 k8s]# cd kubernetes/server/bin/
[root@k8s-master01 bin]# cp kube-apiserver kubectl kube-controller-manager kube-scheduler
/opt/kubernetes/bin/
[root@k8s-master01 bin]# ls /opt/kubernetes/bin/
kube-apiserver   kube-controller-manager   kubectl   kube-scheduler
```

在/opt/kubernetes/cfg/目录中创建名为 token.csv 的 token 文件，其本质就是创建一个用户角色，可以理解为管理性的角色。Node 节点加入到群集当中也是通过这个角色去控制。但是，在此之前需要通过 head 命令生成随机序列号作为 token 令牌。token 文件的主要内容如下所示，其中：

- b9c8ff663c989468eae4c42ed6577b55 为 token 令牌；
- Kubelet-bootstrap 为角色名；
- 100001 为角色 ID；
- "system:kubelet-bootstrap"为绑定的超级用户权限。

```
[root@k8s-master01 bin]# head -c 16 /dev/urandom | od -An -t x | tr -d ' '
b9c8ff663c989468eae4c42ed6577b55

[root@k8s-master01 bin]# vim /opt/kubernetes/cfg/token.csv
b9c8ff663c989468eae4c42ed6577b55,kubelet-bootstrap,10001,"system:kubelet-bootstrap"
```

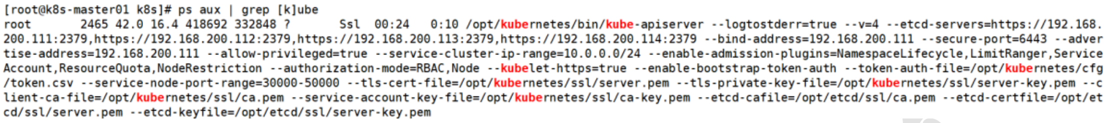
将 k8s-master01 主机/opt/kubernetes/目录下的所有文件拷贝到 k8s-master02 主机中。

```
[root@k8s-master01 bin]# ls -R /opt/kubernetes/
/opt/kubernetes/:
bin   cfg   ssl

/opt/kubernetes/bin:
kube-apiserver   kube-controller-manager   kubectl   kube-scheduler

/opt/kubernetes/cfg:
token.csv

/opt/kubernetes/ssl:
ca-key.pem   ca.pem   server-key.pem   server.pem
[root@k8s-master01 bin]# scp -r /opt/kubernetes/ root@k8s-master02:/opt
```

运行 apiserver.sh 脚本，运行脚本需要填写两个位置参数。第一个位置参数是本地的 IP

地址，第二个位置参数是 API Server 群集列表。

```
[root@k8s-master01 ~]# cd /k8s/
[root@k8s-master01 k8s]# bash apiserver.sh 192.168.200.111
https://192.168.200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:2379,https://192.168.200.114:2379
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-apiserver.service to
/usr/lib/systemd/system/kube-apiserver.service.
[root@k8s-master01 k8s]# ps aux | grep [k]ube
```



```
[root@k8s-master01 k8s]# ps aux | grep [k]ube
root       2465 42.0 16.4 418692 332848 ?      Ssl  00:24   0:10 /opt/kubernetes/bin/kube-apiserver --logtostderr=true --v=4 --etcd-servers=https://192.168.
200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:2379,https://192.168.200.114:2379 --bind-address=192.168.200.111 --secure-port=6443 --adver
tise-address=192.168.200.111 --allow-privileged=true --service-cluster-ip-range=10.0.0.0/24 --enable-admission-plugins=NamespaceLifecycle,LimitRanger,Service
Account,ResourceQuota,NodeRestriction --authorization-mode=RBAC,Node --kubelet-https=true --enable-bootstrap-token-auth --token-auth-file=/opt/kubernetes/cfg
/token.csv --service-node-port-range=30000-50000 --tls-cert-file=/opt/kubernetes/ssl/server.pem --tls-private-key-file=/opt/kubernetes/ssl/server-key.pem --c
lient-ca-file=/opt/kubernetes/ssl/ca.pem --service-account-key-file=/opt/kubernetes/ssl/ca-key.pem --etcd-cafile=/opt/etcd/ssl/ca.pem --etcd-certfile=/opt/et
cd/ssl/server.pem --etcd-keyfile=/opt/etcd/ssl/server-key.pem
```

查看 k8s-master01 节点的 6443 安全端口以及 https 的 8080 端口是否启动。

```
[root@k8s-master01 k8s]# netstat -anpt | grep -E "6443|8080"
tcp        0      0 192.168.200.111:6443    0.0.0.0:*               LISTEN
2465/kube-apiserver
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
2465/kube-apiserver
tcp        0      0 192.168.200.111:6443    192.168.200.111:33152   ESTABLISHED
2465/kube-apiserver
tcp        0      0 192.168.200.111:33152   192.168.200.111:6443    ESTABLISHED
2465/kube-apiserver
```

将/opt/kubernetes/cfg/工作目录下的 kube-apiserver 配置文件及其 token.csv 令牌文件拷贝到 k8s-master02 主机上。在 k8s-master02 主机上修改 kube-apiserver 配置文件，将 bind-address、advertise-address 地址修改为本机地址。

```
[root@k8s-master01 k8s]# scp /opt/kubernetes/cfg/* root@k8s-
master02:/opt/kubernetes/cfg/
```

k8s-master02 主机操作：

```
[root@k8s-master02 k8s]# vim /opt/kubernetes/cfg/kube-apiserver

KUBE_APISERVER_OPTS="--logtostderr=true \
--v=4 \
--etcd-
servers=https://192.168.200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:
2379,https://192.168.200.114:2379 \
--bind-address=192.168.200.112 \
--secure-port=6443 \
--advertise-address=192.168.200.112 \
--allow-privileged=true \
--service-cluster-ip-range=10.0.0.0/24 \
--enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,ResourceQuota,NodeRestriction \
--authorization-mode=RBAC,Node \
--kubelet-https=true \
--enable-bootstrap-token-auth \
--token-auth-file=/opt/kubernetes/cfg/token.csv \
--service-node-port-range=30000-50000 \
--tls-cert-file=/opt/kubernetes/ssl/server.pem    \
--tls-private-key-file=/opt/kubernetes/ssl/server-key.pem \
--client-ca-file=/opt/kubernetes/ssl/ca.pem \
--service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \
--etcd-cafile=/opt/etcd/ssl/ca.pem \
--etcd-certfile=/opt/etcd/ssl/server.pem \
--etcd-keyfile=/opt/etcd/ssl/server-key.pem"
```

将 k8s-master01 节点的 kube-apiserver.service 启动脚本拷贝到 k8s-master02 节点的 /usr/lib/systemd/system 目录下，并且在 k8s-master02 启动 API Server，并且查看端口信息。

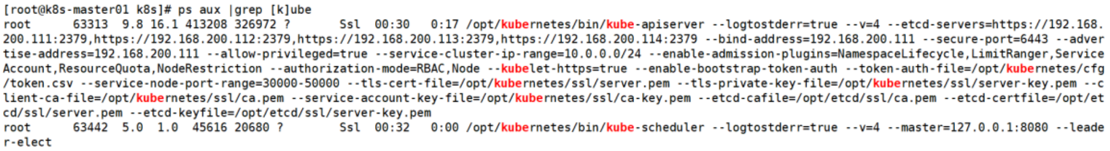
```
[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/kube-apiserver.service root@k8s-master02:/usr/lib/systemd/system/
```

k8s-master02 主机操作：

```
[root@k8s-master02 ~]# systemctl start kube-apiserver && systemctl enable kube-apiserver
[root@k8s-master02 ~]# netstat -anpt | grep -E "6443|8080"
tcp        0      0 192.168.200.112:6443    0.0.0.0:*               LISTEN
2260/kube-apiserver
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
2260/kube-apiserver
```

2.5、部署 Scheduler 组件

```
[root@k8s-master01 ~]# cd /k8s/
[root@k8s-master01 k8s]# ./scheduler.sh 127.0.0.1
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-scheduler.service to /usr/lib/systemd/system/kube-scheduler.service.
[root@k8s-master01 k8s]# ps aux |grep [k]ube
```



```
[root@k8s-master01 k8s]# ps aux |grep [k]ube
root      63313  9.8 16.1 413208 326972 ?      Ssl  00:30   0:17 /opt/kubernetes/bin/kube-apiserver --logtostderr=true --v=4 --etcd-servers=https://192.168.
200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:2379,https://192.168.200.114:2379 --bind-address=192.168.200.111 --secure-port=6443 --adver
tise-address=192.168.200.111 --allow-privileged=true --service-cluster-ip-range=10.0.0.0/24 --enable-admission-plugins=NamespaceLifecycle,LimitRanger,Service
Account,ResourceQuota,NodeRestriction --authorization-mode=RBAC,Node --kubelet-https=true --enable-bootstrap-token-auth --token-auth-file=/opt/kubernetes/cfg
/token.csv --service-node-port-range=30000-50000 --tls-cert-file=/opt/kubernetes/ssl/server.pem --tls-private-key-file=/opt/kubernetes/ssl/server-key.pem --c
lient-ca-file=/opt/kubernetes/ssl/ca.pem --service-account-key-file=/opt/kubernetes/ssl/ca-key.pem --etcd-cafile=/opt/etcd/ssl/ca.pem --etcd-certfile=/opt/et
cd/ssl/server.pem --etcd-keyfile=/opt/etcd/ssl/server-key.pem
root      63442  5.0  1.0  45616 20680 ?       Ssl  00:32   0:00 /opt/kubernetes/bin/kube-scheduler --logtostderr=true --v=4 --master=127.0.0.1:8080 --leade
r-elect
```

将 k8s-master01 节点的 kube-scheduler 配置文件与 kube-scheduler.service 启动脚本拷贝到 k8s-master02 节点上，并且在 k8s-master02 启动 Scheduler。

```
[root@k8s-master01 k8s]# scp /opt/kubernetes/cfg/kube-scheduler root@k8s-master02:/opt/kubernetes/cfg/
```

```
[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/kube-scheduler.service root@k8s-master02:/usr/lib/systemd/system/
```

k8s-master02主机操作：

```
[root@k8s-master02 ~]# systemctl start kube-scheduler
[root@k8s-master02 ~]# systemctl enable kube-scheduler
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-scheduler.service to /usr/lib/systemd/system/kube-scheduler.service.
```

2.6、部署 Controller-Manager 组件

在k8s-master01节点，启动Controller-Manager服务。

```
[root@k8s-master01 k8s]# ./controller-manager.sh 127.0.0.1
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-controller-manager.service to /usr/lib/systemd/system/kube-controller-manager.service.
```

将k8s-master01节点的kube-controller-manager配置文件和controller-manager.service启动脚本拷贝到k8s-master02节点的/opt/kubernetes/cfg目录下，并且在k8s-master02节点上启动Controller-Manager。

```
[root@k8s-master01 k8s]# scp /opt/kubernetes/cfg/kube-controller-manager root@k8s-master02:/opt/kubernetes/cfg/
[root@k8s-master01 k8s]# scp /usr/lib/systemd/system/kube-controller-manager.service root@k8s-master02:/usr/lib/systemd/system/
```

k8s-master02主机操作：

```
[root@k8s-master02 ~]# systemctl start kube-controller-manager
[root@k8s-master02 ~]# systemctl enable kube-controller-manager
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-controller-manager.service to /usr/lib/systemd/system/kube-controller-manager.service.
```

在k8s-master01和k8s-master02节点上，查看各组件状态。

```
[root@k8s-master01 k8s]# /opt/kubernetes/bin/kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-2               Healthy   {"health":"true"}
etcd-0               Healthy   {"health":"true"}
etcd-1               Healthy   {"health":"true"}
etcd-3               Healthy   {"health":"true"}
```

```
[root@k8s-master02 ~]# /opt/kubernetes/bin/kubectl get cs
NAME                 STATUS    MESSAGE             ERROR
controller-manager   Healthy   ok
```

```
scheduler            Healthy    ok
etcd-1               Healthy    {"health":"true"}
etcd-0               Healthy    {"health":"true"}
etcd-3               Healthy    {"health":"true"}
etcd-2               Healthy    {"health":"true"}
```

2.7、部署 Docker 环境

在两台 node 节点上均需要操作，以 k8s-node01 主机为例：
安装 docker-ce

```
[root@k8s-node01 ~]# wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-7.repo

[root@k8s-node01 ~]# yum -y install yum-utils device-mapper-persistent-data lvm2
[root@k8s-node01 ~]# yum-config-manager --add-repo http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo

[root@k8s-node01 ~]# ls /etc/yum.repos.d/
backup   Centos-aliyun.repo   CentOS-Media.repo   docker-ce.repo

[root@k8s-node01 ~]# yum -y install docker-ce
[root@k8s-node01 ~]# systemctl start docker
[root@k8s-node01 ~]# systemctl enable docker
```

阿里云镜像加速器

```
[root@k8s-node01 ~]# cat << END > /etc/docker/daemon.json
{
        "registry-mirrors":[ "https://nyakyfun.mirror.aliyuncs.com" ]
}
END
[root@k8s-node01 ~]# systemctl daemon-reload
[root@k8s-node01 ~]# systemctl restart docker
[root@k8s-node01 ~]# docker version
```

2.8、部署 Flannel 网络组件

虽然在两台 node 节点上安装了 Docker，但是 Docker 运行的容器还需要网络组件 Flannel 的支持来实现彼此之间互联互通。

首先需要将分配的子网段写入到 Etcd 中，以便 Flannel 使用。网络中涉及到的路由如何转发、源目地址如何封装等信息均存储到 Etcd 中。

通过执行以下的 etcdctl 命令，定义以逗号进行分割列出群集中的 IP 地址,set 指定网络中的配置，对应的参数 etcd 是一个键值对，设置网段为 172.17.0.0/16，类型是 vxlan。

执行完后，查看两台 node 节点的 docker0 地址，即 docker 网关的地址是否为

172.17.0.0/16 网段的地址。

```
[root@k8s-master01 ~]# cd /k8s/etcd-cert/
[root@k8s-master01 etcd-cert]# /opt/etcd/bin/etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="https://192.168.200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:2379,https://192.168.200.114:2379" set /coreos.com/network/config '{"Network": "172.17.0.0/16","Backend": {"Type": "vxlan"}}'

{"Network": "172.17.0.0/16","Backend": {"Type": "vxlan"}}
```

```
[root@k8s-node01 ~]# ifconfig docker0
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:57:0b:ea:7a  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

查看写入的网络信息。

```
[root@k8s-master01 etcd-cert]# /opt/etcd/bin/etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints=https://192.168.200.111:2379,https://192.168.200.112:2379,https://192.168.200.113:2379,https://192.168.200.114:2379 set /coreos.com/network/config '{"Network": "172.17.0.0/16","Backend":{"Type": "vxlan"}}' get /coreos.com/network/config

{"Network": "172.17.0.0/16","Backend":{"Type": "vxlan"}}
```

将 flannel-v0.10.0-linux-amd64.tar.gz 软件包上传两个 node 节点服务器，并进行解压缩。在两台 node 节点上均需要操作

```
[root@k8s-node01 ~]# tar xf flannel-v0.12.0-linux-amd64.tar.gz
```

在两台 node 节点上创建 k8s 工作目录。将 flanneld 脚本和 mk-docker-opts.sh 脚本剪切至 k8s 工作目录中。

```
[root@k8s-node01 ~]# mkdir /opt/kubernetes/{cfg,bin,ssl} -p
[root@k8s-node01 ~]# mv mk-docker-opts.sh flanneld /opt/kubernetes/bin/
```

将准备好的 flannel.sh 脚本拖至到两台 node 节点上，用以启动 Flannel 服务和创建配置文件。其中：指定配置文件路径/opt/kubernetes/cfg/flanneld，Etcd 的终端地址以及需要认证的证书秘钥文件；指定启动脚本路径/usr/lib/systemd/system/flanneld.service，添加至自定义系统服务中，交由系统统一管理。

以 k8s-node01 为例：

```
[root@k8s-node01 ~]# cat flannel.sh
#!/bin/bash

ETCD_ENDPOINTS=${1:-"http://127.0.0.1:2379"}

cat <<EOF >/opt/kubernetes/cfg/flanneld

FLANNEL_OPTIONS="--etcd-endpoints=${ETCD_ENDPOINTS} \
-etcd-cafile=/opt/etcd/ssl/ca.pem \
-etcd-certfile=/opt/etcd/ssl/server.pem \
-etcd-keyfile=/opt/etcd/ssl/server-key.pem"

EOF

cat <<EOF >/usr/lib/systemd/system/flanneld.service
[Unit]
Description=Flanneld overlay address etcd agent
After=network-online.target network.target
Before=docker.service

[Service]
Type=notify
EnvironmentFile=/opt/kubernetes/cfg/flanneld
ExecStart=/opt/kubernetes/bin/flanneld --ip-masq \$FLANNEL_OPTIONS
ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -d
/run/flannel/subnet.env
Restart=on-failure

[Install]
WantedBy=multi-user.target

EOF

systemctl daemon-reload
systemctl enable flanneld
systemctl restart flannel

[root@k8s-node01 ~]# bash flannel.sh
https://192.168.200.111:2379,https://192.168.200.112:2379,https://192.168.200.113,https://
192.168.200.114:2379
Created symlink from /etc/systemd/system/multi-user.target.wants/flanneld.service to
/usr/lib/systemd/system/flanneld.service.
```

两台 node 节点配置 Docker 连接 Flannel。docker.service 需要借助 Flannel 进行通信，

需要修改 docker.service。添加 EnvironmentFile=/run/flannel/subnet.env，借助 Flannel 的子网进行通信以及添加$DOCKER_NETWORK_OPTIONS 网络参数。以上两个参数均是官网要求。下面以 k8s-node01 主机为例进行操作演示。

```
[root@k8s-node02 ~]# vim /usr/lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

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

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd 229.
# Both the old, and new location are accepted by systemd 229 and up, so using the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.
# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not support it.
# Only systemd 226 and above support this option.
```

```
TasksMax=infinity

# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes

# kill only the docker process, not all processes in the cgroup
KillMode=process

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

在两台 node 节点上查看使用的子网地址分别为 172.17.13.1/24 和 172.17.81.1/24。bip 是指定启动时的子网。

```
[root@k8s-node01 ~]# cat /run/flannel/subnet.env
DOCKER_OPT_BIP="--bip=172.17.13.1/24"
DOCKER_OPT_IPMASQ="--ip-masq=false"
DOCKER_OPT_MTU="--mtu=1450"
DOCKER_NETWORK_OPTIONS=" --bip=172.17.13.1/24 --ip-masq=false --mtu=1450"
```

```
[root@k8s-node02 ~]# cat /run/flannel/subnet.env
DOCKER_OPT_BIP="--bip=172.17.81.1/24"
DOCKER_OPT_IPMASQ="--ip-masq=false"
DOCKER_OPT_MTU="--mtu=1450"
DOCKER_NETWORK_OPTIONS=" --bip=172.17.81.1/24 --ip-masq=false --mtu=1450"
```

在两台 node 节点上修改完启动脚本之后，需要重新启动 Docker 服务。分别查看两台 node 节点的 docker0 网卡信息。

```
[root@k8s-node01 ~]# systemctl daemon-reload && systemctl restart docker
[root@k8s-node01 ~]# ip add s docker0
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
group default
link/ether 02:42:57:0b:ea:7a brd ff:ff:ff:ff:ff:ff
    inet 172.17.13.1/24 brd 172.17.13.255 scope global docker0
       valid_lft forever preferred_lft forever
```

```
[root@k8s-node02 ~]# systemctl daemon-reload && systemctl restart docker
[root@k8s-node02 ~]# ip add s docker0
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
group default
link/ether 02:42:7d:ad:bf:51 brd ff:ff:ff:ff:ff:ff
    inet 172.17.81.1/24 brd 172.17.81.255 scope global docker0
       valid_lft forever preferred_lft forever
```

在两台 node 节点上分别运行 busybox 容器。（busybox 是一个集成了三百多个常用 Linux 命令和工具的软件工具箱，在本案例中用于测试）。

进入容器内部查看 k8s-node01 节点的地址是 172.17.88.2；k8s-node02 节点的地址是 172.17.44.2。与/run/flannel/subnet.env 文件中看到的子网信息处于同一个网段。

接着再通过 ping 命令测试，如果 k8s-node02 容器能 ping 通 k8s-node01 容器的 IP 地址就代表两个独立的容器可以互通，说明 Flannel 组件搭建成功。

```
[root@k8s-node01 ~]# docker pull busybox
[root@k8s-node01 ~]# docker run -it busybox /bin/sh
/ # ipaddr show eth0
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue
      link/ether 02:42:ac:11:0d:02 brd ff:ff:ff:ff:ff:ff
     inet 172.17.13.2/24 brd 172.17.13.255 scope global eth0
        valid_lft forever preferred_lft forever
```

```
[root@k8s-node02 ~]# docker pull busybox
[root@k8s-node02 ~]# docker run -it busybox /bin/sh
/ # ipaddr show eth0
7: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue
      link/ether 02:42:ac:11:51:02 brd ff:ff:ff:ff:ff:ff
     inet 172.17.81.2/24 brd 172.17.81.255 scope global eth0
        valid_lft forever preferred_lft forever
/ # ping -c 4 172.17.13.2
PING 172.17.13.2 (172.17.13.2): 56 data bytes
64 bytes from 172.17.13.2: seq=0 ttl=62 time=0.729 ms
64 bytes from 172.17.13.2: seq=1 ttl=62 time=1.673 ms
64 bytes from 172.17.13.2: seq=2 ttl=62 time=1.099 ms
64 bytes from 172.17.13.2: seq=3 ttl=62 time=0.745 ms

--- 172.17.13.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.729/1.061/1.673 ms
```

2.9、部署 kubeconfig 配置

在 k8s-master01 节点上将 kubelet 和 kube-proxy 执行脚本拷贝到两台 node 节点上。

```
[root@k8s-master01 ~]# cd /k8s/kubernetes/server/bin/
[root@k8s-master01 bin]# scp kubelet kube-proxy root@k8s-node01:/opt/kubernetes/bin/
[root@k8s-master01 bin]# scp kubelet kube-proxy root@k8s-node02:/opt/kubernetes/bin/
```

将 node.zip 上传至两台 node 节点，并解压 node.zip，可获得 proxy.sh 和 kubelet.sh 两个执行脚本。以 k8s-node01 为例进行操作演示。

```
[root@k8s-node01 ~]# unzip node.zip
Archive:   node.zip
```

```
   inflating: proxy.sh
   inflating: kubelet.sh

[root@k8s-node02 ~]# unzip node.zip
Archive:   node.zip
   inflating: proxy.sh
   inflating: kubelet.sh
```

在 k8s-master01 节点上创建 kubeconfig 工作目录。将 kubeconfig.sh 脚本上传至当前目录 k8s/kubeconfig/ 下，此脚本中包含有创建 TLS Bootstrapping Token、创建 kubeletbootstrapping kubeconfig、设置集群参数、设置客户端认证参数、设置上下文参数、设置默认上下文、创建 kube-proxy kubeconfig 文件。

查看序列号将其拷贝到客户端认证参数。更新 kubeconfig.sh 脚本的 token 值。

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

[root@k8s-master01 kubeconfig]# cat /opt/kubernetes/cfg/token.csv
b9c8ff663c989468eae4c42ed6577b55,kubelet-bootstrap,10001,"system:kubelet-bootstrap"

[root@k8s-master01 kubeconfig]# vim kubeconfig.sh
# 创建 TLS Bootstrapping Token
#BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
BOOTSTRAP_TOKEN=b9c8ff663c989468eae4c42ed6577b55
```

为了便于识别在 k8s-master01 和 k8s-master02 节点上声明路径 export PATH=$PATH:/opt/kubernetes/bin/到环境变量中。

```
[root@k8s-master01 ~]# echo "export PATH=$PATH:/opt/kubernetes/bin/" >> /etc/profile
[root@k8s-master01 ~]# source /etc/profile

[root@k8s-master02 ~]# echo "export PATH=$PATH:/opt/kubernetes/bin/" >> /etc/profile
[root@k8s-master02 ~]# source /etc/profile
```

将 kubeconfig.sh 重命名为 kubeconfig，执行 kubeconfig 脚本。使用 bash 执行 kubeconfig，第一个参数是当前 APIServer 的 IP，它会写入整个配置当中；第二个参数指定证书 kubenets 的证书位置。执行完成以后会生成 bootstrap.kubeconfig 和 kube-proxy.kubeconfig 两个文件。

```
[root@k8s-master01 ~]# cd /k8s/kubeconfig/
[root@k8s-master01 kubeconfig]# mv kubeconfig.sh kubeconfig
[root@k8s-master01 kubeconfig]# bash kubeconfig 192.168.200.111 /k8s/k8s-cert/
Cluster "kubernetes" set.
User "kubelet-bootstrap" set.
Context "default" created.
Switched to context "default".
```

```
Cluster "kubernetes" set.
User "kube-proxy" set.
Context "default" created.
Switched to context "default".

[root@k8s-master01 kubeconfig]# ls
bootstrap.kubeconfig   kubeconfig   kube-proxy.kubeconfig   token.csv
```

将 bootstrap.kubeconfig 和 kube-proxy.kubeconfig 文件拷贝到两台 node 节点上。

```
[root@k8s-master01 kubeconfig]# scp bootstrap.kubeconfig kube-proxy.kubeconfig root@k8s-node01:/opt/kubernetes/cfg/
[root@k8s-master01 kubeconfig]# scp bootstrap.kubeconfig kube-proxy.kubeconfig root@k8s-node02:/opt/kubernetes/cfg/
```

创建 bootstrap 角色，并赋予权限。用于连接 API Server 请求签名（关键）。查看 k8s-node01 节点的 bootstrap.kubeconfig。Kubelet 在启动的时候如果想加入群集中，需要请求申请 API Server 请求签名。kubeconfig 的作用是指明如果想要加入群集，需要通过哪一个地址、端口才能申请到所需要的证书。

```
[root@k8s-master01 kubeconfig]# kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap
clusterrolebinding.rbac.authorization.k8s.io/kubelet-bootstrap created
```

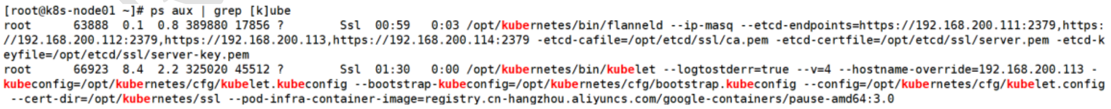
```
[root@k8s-node01 ~]# cat /opt/kubernetes/cfg/bootstrap.kubeconfig
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR2akNDQXFhZ0F3SUJBZ0lVYklFL3FWeUd4d0
FmZ3VodXdqT2x2aGMxSVNRd0RRWUpLb1pJaHZjTkFRRUwKQlFBd1pURUxNQWtHQTFVRUJoT
UNRMDR4RURBT0JnTlZCQWdUQjBKbGFXcHBibWN4RURBT0JnTlZCQWNUQjBKbAphV3BwYm1j
eEREQUtCZ05WQkFvVEEyczRjekVQTUEwR0ExVUVDeE1HVTNsemRHVnRNUk13RVFZRFZRUURF
d3ByCmRXSmxjbTVsZEdWek1CNFhEVEl4TURFeU1URTJNRGd3TUZvWERUSTJNREV5TURFMk1EZ
3dNRm93WlRFTE1Ba0cKQTFVRUJoTUNRMDR4RURBT0JnTlZCQWdUQjBKbGFXcHBibWN4RURB
T0JnTlZCQWNUQjBKbGFXcHBibWN4RERBSwpCZ05WQkFvVEEyczRjekVQTUEwR0ExVUVDeE1H
VTNsemRHVnRNUk13RVFZRFZRUURFd3ByZFdKbGNtNWxkR1Z6Ck1JSUJJakFOQmdrcWhraUc5d
zBCQVFFRkFBT0NBUThBTUlJQkNnS0NBUUVBcnc3Ujh3SEFvNUJRYXpwZEdYSW0KQVpPMWxJN2
FOOUhGdzA4THFZZXI3NnlXUWV1SFRzclNXRDZpUjZ5dzUrQmFlMVJoRHJYZ3l2Z2hSd0N6bmZoN
QpRYVhnMkhsM0dGMm1MMWhKazZOdWI5L3BjMldRa2dBM2F5b0pFWjRXazU4TGdNbmJ5Z2
Riclpna1dDc3RVdWNvCm1oNnNrZStCQXkvNXJjeDM5NXBWd0VvZUVQV0NjL1pXR0E0RDlYZUR
hRnhsS1lkN2QyMmdsUTNNQnp6V0g3K3gKeHF0UDhBNGh3WHJHcG5xVGFlUW0vQkFPeGdDY
WZsWm9Cek45MTZsWC94NElMVEhlRVlxcDVJb2xxQREpxbkZZcwpzY3ZOSHhlamRKTEdWRW9n
MFgwaU1KaFZyUGc0ZmNrWnJ3Q1VkMWZuQXZsQ0xITFNINHAzT2pVRkd3M09aWEFqCmd3SU
RBUUFCbzJJd1pEQU9CZ05WSFE4QkFmOEVCQU1DQVFZd0VnWURWUjBUQVFIL0JBZ3dCZ0VCL3
dJQkFqQWQKQmdOVkhRNEVGZ1FVMW9uVDF6QmxFenpuTGZFQVRBaHM3VlNabm9Zd0h3WU
```

```
RWUjBqQkJnd0ZvQVUxb25UMXpCbApFenpuTGZFQVRBaHM3VlNabm9Zd0RRWUpLb1pJaHZjTk
FRRUxCUUFEZ2dFQkFKWldXbU1xaGc5aDk4cFJvK0F5CkNyMDZ3Rmg1ZDN2MjhObkJJcmdyZ1Qy
RGt6bTRVL3p3eFdaaGk1SitYSlNuNk1aUDFLYTFHeUd1Q21RZE9HUysKRTh4dWQweTUyNjdhbTF
0cy8xMW5SS3JISmVGbkFZZ3dTcnpmZU1CMEVWa3hMK1hYd2JnMjR0M09acTZaR0tXOQpMSzV
WTDNYZnp3ajNiS2lRektJTXgyYk1EVXZpWklya1k0dDZaNENrRjFDSG9BRlFObkFjS0ZEcE1WaDFlN
Vp2CjdwMUFLL2VTblRkSE5oL01UUU44c2dpNWFtdmdMa2s2aVdRWEg3Q242MXdlMFJka3pNO
FFBZFJ2R0VDVDBrSDQKYmVEbWtXcFViVmg2U3hJbEdQbnBJVTFDSkRUMGJzQ3hyR3B5U2c2K0J
WQTExb3JYc1NBYU81dSs5ZllRSmlFVworYUU9Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
server: https://192.168.200.111:6443
   name: kubernetes
contexts:
- context:
      cluster: kubernetes
      user: kubelet-bootstrap
   name: default
current-context: default
kind: Config
preferences: {}
users:
- name: kubelet-bootstrap
   user:
token: b9c8ff663c989468eae4c42ed6577b55
```

2.10、部署 Kubelet 组件

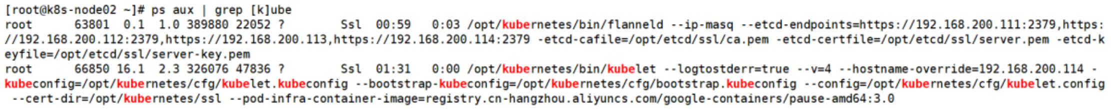
在两台 node 节点上，执行 kubelet 脚本，并通过 ps 命令查看服务启动情况。kublet 启动之后会自动联系 API Server 进行证书申请。在 k8s-master01 节点上通过 get csr 命令查看是否收到请求申请。当看到处于 Pending 状态时，即为等待集群给该节点颁发证书。

```
[root@k8s-node01 ~]# bash kubelet.sh 192.168.200.113
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to
/usr/lib/systemd/system/kubelet.service.
[root@k8s-node01 ~]# ps aux | grep [k]ube
```



```
[root@k8s-node01 ~]# ps aux | grep [k]ube
root      63888  0.1  0.8 389880 17856 ?        Ssl  00:59   0:03 /opt/kubernetes/bin/flanneld --ip-masq --etcd-endpoints=https://192.168.200.111:2379,https:
//192.168.200.112:2379,https://192.168.200.113,https://192.168.200.114:2379 -etcd-cafile=/opt/etcd/ssl/ca.pem -etcd-certfile=/opt/etcd/ssl/server.pem -etcd-k
eyfile=/opt/etcd/ssl/server-key.pem
root      66923  8.4  2.2 325020 45512 ?        Ssl  01:30   0:00 /opt/kubernetes/bin/kubelet --logtostderr=true --v=4 --hostname-override=192.168.200.113 -
kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig --bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig --config=/opt/kubernetes/cfg/kubelet.config
 --cert-dir=/opt/kubernetes/ssl --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google-containers/pause-amd64:3.0
```

```
[root@k8s-node02 ~]# bash kubelet.sh 192.168.200.114
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to
/usr/lib/systemd/system/kubelet.service.
[root@k8s-node02 ~]# ps aux | grep [k]ube
```



```
[root@k8s-node02 ~]# ps aux | grep [k]ube
root      63801  0.1  1.0 389880 22052 ?        Ssl  00:59   0:03 /opt/kubernetes/bin/flanneld --ip-masq --etcd-endpoints=https://192.168.200.111:2379,https:
//192.168.200.112:2379,https://192.168.200.113,https://192.168.200.114:2379 -etcd-cafile=/opt/etcd/ssl/ca.pem -etcd-certfile=/opt/etcd/ssl/server.pem -etcd-k
eyfile=/opt/etcd/ssl/server-key.pem
root      66850 16.1  2.3 326076 47836 ?        Ssl  01:31   0:00 /opt/kubernetes/bin/kubelet --logtostderr=true --v=4 --hostname-override=192.168.200.114 -
kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig --bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig --config=/opt/kubernetes/cfg/kubelet.config
 --cert-dir=/opt/kubernetes/ssl --pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google-containers/pause-amd64:3.0
```

```
[root@k8s-master01 kubeconfig]# kubectl get csr
NAME                                                   AGE     REQUESTOR
CONDITION
node-csr-dkiyiMBhzYUBHQBbPlLzxgPS6u-RwRDyh2soUyVffkM   113s    kubelet-bootstrap
Pending
node-csr-xLvZIWzh5_CMj4da9orMopr3pdgu7PivQADP2QZowA8   66s     kubelet-bootstrap
Pending
```

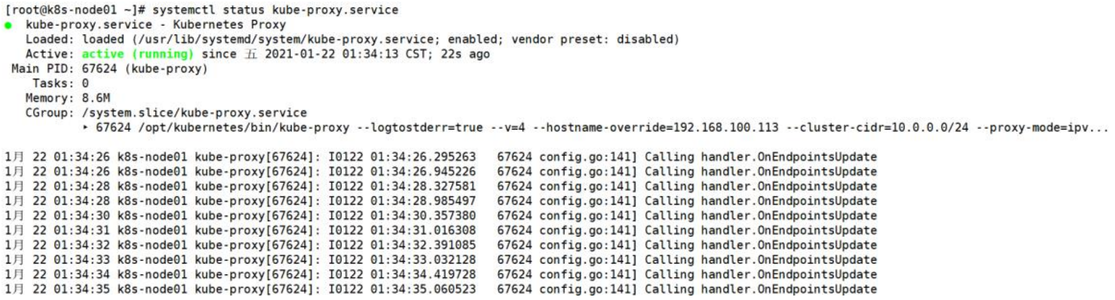
k8s-master01 节点颁发证书给两台 node 节点。通过 get csr 命令可以查看到证书已经颁发。使用 get node 查看，两台 node 节点都已经加入到了群集中。

```
[root@k8s-master01 kubeconfig]# kubectl certificate approve node-csr-
dkiyiMBhzYUBHQBbPlLzxgPS6u-RwRDyh2soUyVffkM
certificatesigningrequest.certificates.k8s.io/node-csr-dkiyiMBhzYUBHQBbPlLzxgPS6u-
RwRDyh2soUyVffkM approved
[root@k8s-master01 kubeconfig]# kubectl certificate approve node-csr-
xLvZIWzh5_CMj4da9orMopr3pdgu7PivQADP2QZowA8
certificatesigningrequest.certificates.k8s.io/node-csr-
xLvZIWzh5_CMj4da9orMopr3pdgu7PivQADP2QZowA8 approved

[root@k8s-master01 kubeconfig]# kubectl get csr
NAME                                                   AGE      REQUESTOR
CONDITION
node-csr-dkiyiMBhzYUBHQBbPlLzxgPS6u-RwRDyh2soUyVffkM   3m16s    kubelet-bootstrap
Approved,Issued
node-csr-xLvZIWzh5_CMj4da9orMopr3pdgu7PivQADP2QZowA8   2m29s    kubelet-
bootstrap    Approved,Issued
```

2.11、部署 Kube-Proxy 组件

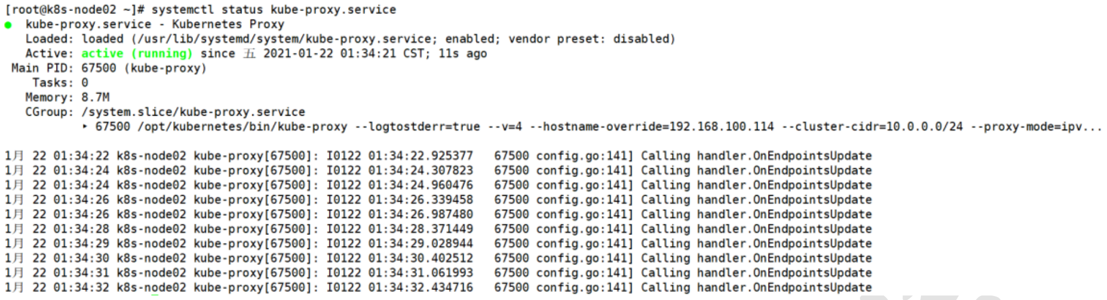
在两台 node 节点上执行 proxy.sh 脚本。

```
[root@k8s-node01 ~]# bash proxy.sh 192.168.100.113
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-proxy.service to
/usr/lib/systemd/system/kube-proxy.service.
[root@k8s-node01 ~]# systemctl status kube-proxy.service
```



```
[root@k8s-node01 ~]# systemctl status kube-proxy.service
● kube-proxy.service - Kubernetes Proxy
   Loaded: loaded (/usr/lib/systemd/system/kube-proxy.service; enabled; vendor preset: disabled)
   Active: active (running) since 五 2021-01-22 01:34:13 CST; 22s ago
 Main PID: 67624 (kube-proxy)
    Tasks: 0
   Memory: 8.6M
   CGroup: /system.slice/kube-proxy.service
           ‣ 67624 /opt/kubernetes/bin/kube-proxy --logtostderr=true --v=4 --hostname-override=192.168.100.113 --cluster-cidr=10.0.0.0/24 --proxy-mode=ipv...

1月 22 01:34:26 k8s-node01 kube-proxy[67624]: I0122 01:34:26.295263   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:26 k8s-node01 kube-proxy[67624]: I0122 01:34:26.945226   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:28 k8s-node01 kube-proxy[67624]: I0122 01:34:28.327581   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:28 k8s-node01 kube-proxy[67624]: I0122 01:34:28.985497   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:30 k8s-node01 kube-proxy[67624]: I0122 01:34:30.357380   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:31 k8s-node01 kube-proxy[67624]: I0122 01:34:31.016308   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:32 k8s-node01 kube-proxy[67624]: I0122 01:34:32.391085   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:33 k8s-node01 kube-proxy[67624]: I0122 01:34:33.032128   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:34 k8s-node01 kube-proxy[67624]: I0122 01:34:34.419728   67624 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:35 k8s-node01 kube-proxy[67624]: I0122 01:34:35.060523   67624 config.go:141] Calling handler.OnEndpointsUpdate
```

```
[root@k8s-node02 ~]# bash proxy.sh 192.168.100.114
Created symlink from /etc/systemd/system/multi-user.target.wants/kube-proxy.service to
/usr/lib/systemd/system/kube-proxy.service.
[root@k8s-node02 ~]# systemctl status kube-proxy.service
```



```
[root@k8s-node02 ~]# systemctl status kube-proxy.service
● kube-proxy.service - Kubernetes Proxy
   Loaded: loaded (/usr/lib/systemd/system/kube-proxy.service; enabled; vendor preset: disabled)
   Active: active (running) since 五 2021-01-22 01:34:21 CST; 11s ago
 Main PID: 67500 (kube-proxy)
    Tasks: 0
   Memory: 8.7M
   CGroup: /system.slice/kube-proxy.service
           ‣ 67500 /opt/kubernetes/bin/kube-proxy --logtostderr=true --v=4 --hostname-override=192.168.100.114 --cluster-cidr=10.0.0.0/24 --proxy-mode=ipv...

1月 22 01:34:22 k8s-node02 kube-proxy[67500]: I0122 01:34:22.925377   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:24 k8s-node02 kube-proxy[67500]: I0122 01:34:24.307823   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:24 k8s-node02 kube-proxy[67500]: I0122 01:34:24.960476   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:26 k8s-node02 kube-proxy[67500]: I0122 01:34:26.339458   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:26 k8s-node02 kube-proxy[67500]: I0122 01:34:26.987480   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:28 k8s-node02 kube-proxy[67500]: I0122 01:34:28.371449   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:29 k8s-node02 kube-proxy[67500]: I0122 01:34:29.028944   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:30 k8s-node02 kube-proxy[67500]: I0122 01:34:30.402512   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:31 k8s-node02 kube-proxy[67500]: I0122 01:34:31.061993   67500 config.go:141] Calling handler.OnEndpointsUpdate
1月 22 01:34:32 k8s-node02 kube-proxy[67500]: I0122 01:34:32.434716   67500 config.go:141] Calling handler.OnEndpointsUpdate
```

2.12、部署 Nginx 反向代理

在 NodePort 基础上，Kubernetes 可以请求底层云平台创建一个负载均衡器，将每个 Node 作为后端，进行服务分发。该模式需要底层云平台（例如 GCE）支持。

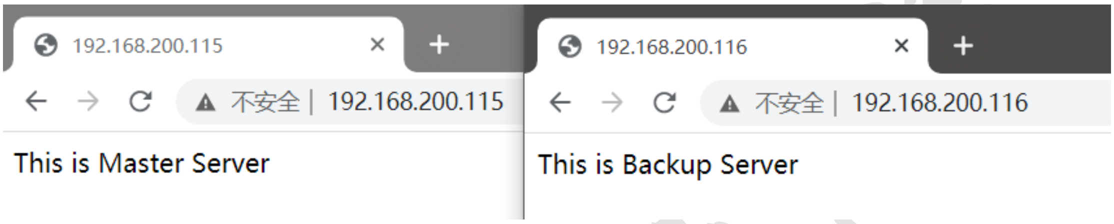
安装配置 Nginx 服务，lb01、lb02 主机上均执行以下操作，以 lb01 节点为例

```
[root@k8s-lb01 ~]# rpm -ivh epel-release-latest-7.noarch.rpm
[root@k8s-lb01 ~]# yum -y install nginx
[root@k8s-lb01 ~]# vim /etc/nginx/nginx.conf    //配置四层转发负载均衡
events {
        worker_connections 1024;
}

stream {
        log_format main '$remote_addr $upstream_addr - [$time_local] $status
$upstream_bytes_sent';
        access_log /var/log/nginx/k8s-access.log main;
        upstream k8s-apiserver {
                server 192.168.200.111:6443;
                server 192.168.200.112:6443;
        }
        server {
                listen 6443;
                proxy_pass k8s-apiserver;
        }
}



http {
log_format    main    '$remote_addr - $remote_user [$time_local] "$request" '
```

```
[root@k8s-lb01 ~]# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
[root@k8s-lb02 ~]# systemctl start nginx && systemctl enable nginx
Created symlink from /etc/systemd/system/multi-user.target.wants/nginx.service to
/usr/lib/systemd/system/nginx.service.
```

修改两台 Nginx 节点的首页，以示区分，并且浏览器中访问两台 LB 节点

```
[root@k8s-lb01 ~]# echo "This is Master Server" > /usr/share/nginx/html/index.html
[root@k8s-lb02 ~]# echo "This is Backup Server" > /usr/share/nginx/html/index.html
```



This is Master Server

This is Backup Server

2.13、部署 Keepalived

```
[root@k8s-lb01 ~]# yum -y install keepalived
[root@k8s-lb01 ~]# vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
   router_id LVS_DEVEL
}

vrrp_script check_nginx {
    script "/etc/nginx/check_nginx.sh"
}

vrrp_instance VI_1 {
    state MASTER
    interface ens32
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
```

```
                192.168.200.200
        }
        track_script {
                check_nginx
        }
}

[root@k8s-lb01 ~]# scp /etc/keepalived/keepalived.conf 192.168.200.116:/etc/keepalived/
```

```
[root@k8s-lb02 ~]# vim /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    router_id LVS_DEVEL
}

vrrp_script check_nginx {
        script "/etc/nginx/check_nginx.sh"
}

vrrp_instance VI_1 {
        state BACKUP
        interface ens32
        virtual_router_id 51
        priority 90
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass 1111
        }
        virtual_ipaddress {
                192.168.200.200
        }
        track_script {
                check_nginx
        }
}
```

在两台 LB 节点上创建触发脚本。统计数据进行比对，值为 0 的时候，关闭 Keepalived 服务。

lb01、lb02 主机上均执行以下操作

```
[root@k8s-lb01 ~]# vim /etc/nginx/check_nginx.sh
count=$(ps -ef |grep nginx |egrep -cv "grep|$$")
if [ "$count" -eq 0 ];then
```

```
    systemctl stop keepalived
fi
[root@k8s-lb01 ~]# chmod +x /etc/nginx/check_nginx.sh
[root@k8s-lb01 ~]# systemctl start keepalived && systemctl enable keepalived
```

查看网卡信息，可以看到 k8s-lb01 节点上有漂移地址 192.168.200.200/24，而 k8s-lb02 节点上没有漂移地址。

```
[root@k8s-lb01 ~]# ip add s ens32
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 00:0c:29:50:1b:42 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.115/24 brd 192.168.200.255 scope global ens32
       valid_lft forever preferred_lft forever
    inet 192.168.200.200/32 scope global ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe50:1b42/64 scope link
       valid_lft forever preferred_lft forever
```

```
[root@k8s-lb02 ~]# ip add s ens32
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 00:0c:29:19:c4:ac brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.116/24 brd 192.168.200.255 scope global ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe19:c4ac/64 scope link
       valid_lft forever preferred_lft forever
```

验证故障转移切换：首先将 k8s-lb01 节点上的 Nginx 服务关闭，查看 IP 信息可以看出 k8s-lb01 的漂移 IP 已经不存在，Keepalived 服务也关闭了；查看 k8s-lb02 的 IP 信息，漂移 IP 地址已经绑定在 k8s-lb02 节点上。此时再将 k8s-lb01 的 Nginx 与 Keepalived 服务开启，漂移 IP 地址就会重新 k8s-lb01 节点上。

```
[root@k8s-lb01 ~]# systemctl stop nginx
[root@k8s-lb01 ~]# ps aux |grep [k]eepalived

[root@k8s-lb02 ~]# ip add s ens32
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 00:0c:29:19:c4:ac brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.116/24 brd 192.168.200.255 scope global ens32
       valid_lft forever preferred_lft forever
    inet 192.168.200.200/32 scope global ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe19:c4ac/64 scope link
       valid_lft forever preferred_lft forever
```

故障恢复测试

```
[root@k8s-lb01 ~]# systemctl start nginx
[root@k8s-lb01 ~]# systemctl start keepalived
[root@k8s-lb01 ~]# ip add s ens32
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 00:0c:29:50:1b:42 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.115/24 brd 192.168.200.255 scope global ens32
       valid_lft forever preferred_lft forever
    inet 192.168.200.200/32 scope global ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe50:1b42/64 scope link
       valid_lft forever preferred_lft forever
```

```
[root@k8s-lb02 ~]# ip add s ens32
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group
default qlen 1000
    link/ether 00:0c:29:19:c4:ac brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.116/24 brd 192.168.200.255 scope global ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe19:c4ac/64 scope link
       valid_lft forever preferred_lft forever
```

修改两台 node 节点上的 bootstrap.kubeconfig、kubelet.kubeconfig 和 kube-proxy.kubeconfig 配置文件，这三个文件中指向 API Server 的 IP 地址，将此地址更新为 VIP 地址。

node01、node02 主机上均执行以下操作

```
[root@k8s-node01 ~]# cd /opt/kubernetes/cfg/
[root@k8s-node01 cfg]# vim bootstrap.kubeconfig
....... //省略部分内容
server: https://192.168.100.200:6443
....... //省略部分内容

[root@k8s-node01 cfg]# vim kubelet.kubeconfig
....... //省略部分内容
server: https://192.168.100.200:6443
....... //省略部分内容

[root@k8s-node01 cfg]# vim kube-proxy.kubeconfig
....... //省略部分内容
server: https://192.168.100.200:6443
....... //省略部分内容
```

```
[root@k8s-node01 cfg]# grep 200 *
bootstrap.kubeconfig: server: https://192.168.200.200:6443
kubelet.kubeconfig: server: https://192.168.200.200:6443
kube-proxy.kubeconfig: server: https://192.168.200.200:6443
```

重启两台 node 节点相关服务。node01、node02 主机上均执行以下操作

```
[root@k8s-node01 cfg]# systemctl restart kubelet
[root@k8s-node01 cfg]# systemctl restart kube-proxy
```

k8s-lb01 节点上动态查看 Nginx 的访问日志。从日志中可以看出了负载均衡已经实现。

```
[root@k8s-lb01 ~]# tail -f /var/log/nginx/k8s-access.log
192.168.200.113 192.168.200.112:6443 - [22/Jan/2021:02:06:00 +0800] 200 1121
192.168.200.113 192.168.200.112:6443 - [22/Jan/2021:02:06:00 +0800] 200 1120
192.168.200.114 192.168.200.111:6443 - [22/Jan/2021:02:06:02 +0800] 200 1122
192.168.200.114 192.168.200.111:6443 - [22/Jan/2021:02:06:02 +0800] 200 1122
```

2.14、部署测试应用

在 k8s-master01 节点上创建 Pod，使用的镜像是 Nginx。

```
[root@k8s-node01 ~]# docker pull nginx
[root@k8s-node02 ~]# docker pull nginx
```

```
[root@k8s-master01 ~]# kubectl run nginx --image=nginx
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and will be removed in a
future version. Use kubectl create instead.
deployment.apps/nginx created

[root@k8s-master01 ~]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-8dff4969c-fnlj4    1/1     Running   0          102s
```

开启查看日志权限。

```
[root@k8s-master01 ~]# kubectl create clusterrolebinding cluster-system-anonymous --
clusterrole=cluster-admin --user=system:anonymous
clusterrolebinding.rbac.authorization.k8s.io/cluster-system-anonymous created
```

通过-o wide 参数，输出整个网络状态。可以查看此容器的 IP 是 172.17.44.2，容器是放在 IP 地址为 192.168.100.54 的 node 节点中。

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE    IP
NODE               NOMINATED NODE
nginx-8dff4969c-fnlj4    1/1     Running   0          118s   172.17.13.2
```

```
192.168.200.113      <none>
```

```
[root@k8s-node01 ~]# ip add s flannel.1
6: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state
UNKNOWN group default
    link/ether 32:dd:04:6b:eb:64 brd ff:ff:ff:ff:ff:ff
    inet 172.17.13.0/32 scope global flannel.1
       valid_lft forever preferred_lft forever
    inet6 fe80::30dd:4ff:fe6b:eb64/64 scope link
       valid_lft forever preferred_lft foreve
```
r

使用 curl 访问 Pod 容器地址 172.17.13.2。访问日志会产生信息，回到 k8s-master01 节点中查看日志信息。并且查看容器。其他的 node 节点也能访问到。

```
[root@k8s-node01 ~]# curl 172.17.13.2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

查看日志输出

```
[root@k8s-master01 ~]# kubectl logs nginx-8dff4969c-fnlj4
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
```

```
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
172.17.13.1 - - [21/Jan/2021:18:19:59 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.29.0" "-"
172.17.13.1 - - [21/Jan/2021:18:20:40 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.29.0" "-"
```