http://blog.51cto.com/ylw6006/2104031

apiserver的高可用也有三种基本思路：

一.是使用外部负载均衡器，不管是使用公有云提供的负载均衡器服务或是在私有云中使用LVS或者HaProxy自建负载均衡器都可以归到这一类。 负载均衡器是非常成熟的方案，在这里略过不做过多介绍。如何保证负载均衡器的高可用，则是选择这一方案需要考虑的新问题。

二.是在网络层做负载均衡。比如在Master节点上用BGP做ECMP，或者在Node节点上用iptables做NAT都可以实现。采用这一方案不需要额外的外部服务，但是对网络配置有一定的要求。

三.是在Node节点上使用反向代理对多个Master做负载均衡。这一方案同样不需要依赖外部的组件，但是当Master节点有增减时，如何动态配置Node节点上的负载均衡器成为了另外一个需要解决的问题。

对于 kube-apiserver，可以运行多个实例（本文档是 3 实例），但对其它组件需要提供统一的访问地址，该地址需要高可用。本文档使用 keepalived 和 haproxy 实现 kubeapiserver
VIP 高可用和负载均衡

集群模式和ha+keepalived的主要区别是什么呢?
ha+keepalived配置vip，实现了api唯一的访问地址和负载均衡。
集群模式没有配置vip。

cfssl
下载解压cfssl

```
curl -L https://pkg.cfssl.org/R1.2/cfssl_linux-amd64 -o cfssl
chmod +x cfssl
curl -L https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64 -o cfssljson
chmod +x cfssljson
curl -L https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64 -o cfssl-certinfo
chmod +x cfssl-certinfo
```

```
[root@kubernetes-build cfssl]# ls -l
总用量 18808
-rwxr-xr-x. 1 root root 10376657 4月  21 14:31 cfssl
-rwxr-xr-x. 1 root root  6595195 4月  21 14:34 cfssl-certinfo
```

-rwxr-xr-x. 1 root root 2277873 4月 21 14:32 cfssljson

初始化cfssl

```
mkdir cert
cd cert
../cfssl print-defaults config > config.json
../cfssl print-defaults csr > csr.json
```

[root@k8s-master2 pki]# cat ca-config.json

```
{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
          "client auth"
        ],
        "expiry": "8760h"
      }
    }
  }
}
```

[root@k8s-master2 pki]# cat ca-csr.json

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

```
    "ST": "Beijing",
    "L": "Beijing",
    "O": "k8s",
    "OU": "k8s"
  }]
}
```

生成ca-key.pem ca.pem证书文件

```
../cfssl gencert -initca ca-csr.json | ../cfssljson -bare ca
-rw-------. 1 root root 1675 4月  21 14:38 ca-key.pem
-rw-r--r--. 1 root root 1363 4月  21 14:38 ca.pem
```

```
[root@k8s-master2 pki]# cat server-csr.json
{
  "CN": "kubernetes",
  "hosts": [
    "127.0.0.1",
    "172.16.103.184",
    "172.16.103.245",
    "172.16.103.246",
    "172.16.102.19",
    "172.16.102.20",
    "172.16.102.100",
    "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": "CH",
```

```
    "ST": "Beijing",
    "L": "Beijing",
    "O": "k8s",
    "OU": "k8s"
  }]
}
../cfssl gencert -ca=ca.pem -ca-key=ca-key.pem --config=ca-config.json -
profile=kubernetes server-csr.json | ../cfssljson -bare server
../cfssl-certinfo -cert server.pem

[root@k8s-master1 kubernetes]# cat /usr/lib/systemd/system/kube-apiserver.service
[Unit]
Description=Kube API  Server
After=etcd.service
Wants=etcd.service

[Service]
Type=notify
EnvironmentFile=/etc/kubernetes/apiserver
ExecStart=/usr/bin/kube-apiserver $KUBE_API_ARGS
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target

[root@k8s-master1 kubernetes]# cat /etc/kubernetes/apiserver
KUBE_API_ARGS=
"--etcd-
servers=http://172.16.103.184:2379,http://172.16.103.245:2379,http://172.16.103.246:2379

 --insecure-bind-address=172.16.103.184
 --insecure-port=8080
 --advertise-address=172.16.103.184
 --service-cluster-ip-range=10.254.0.0/16
 --service-node-port-range=1-65535
 --admission-control=NamespaceLifecycle,ServiceAccount,LimitRanger,ResourceQuota
```

--client-ca-file=/etc/kubernetes/pki/ca.pem
--tls-cert-file=/etc/kubernetes/pki/server.pem
--tls-private-key-file=/etc/kubernetes/pki/server-key.pem
--apiserver-count=3 //负载均衡参数
--logtostderr=false
--log-dir=/home/k8s/log
--v=3"