**《部署一套完整的企业级K8s集群》**

**v1.23，kubeadm方式**

|  |  |
| --- | --- |
| 作者信息 | 李振良（阿良），微信：k8init |
| DevOps实战学院 | [http://www.aliangedu.cn](http://www.aliangedu.cn/) |
| 说明 | 该文档有导航窗格，方便阅读，如果左侧没有显示，请检查word是否启用。  转载请注明作者，拒绝不道德行为！ |
| 最后更新时间 | 2022-04-08 |



## 一、前置知识点

### 1.1 生产环境可部署Kubernetes集群的两种方式

目前生产部署Kubernetes集群主要有两种方式：

* **kubeadm**

Kubeadm是一个K8s部署工具，提供kubeadm init和kubeadm join，用于快速部署Kubernetes集群。

* **二进制包**

从github下载发行版的二进制包，手动部署每个组件，组成Kubernetes集群。

这里采用kubeadm搭建集群。

kubeadm工具功能：

* **kubeadm init：**初始化一个Master节点
* **kubeadm join：**将工作节点加入集群
* **kubeadm upgrade：**升级K8s版本
* **kubeadm token：**管理 kubeadm join 使用的令牌
* **kubeadm reset：**清空 kubeadm init 或者 kubeadm join 对主机所做的任何更改
* **kubeadm version：**打印 kubeadm 版本
* **kubeadm alpha：**预览可用的新功能

### 1.2 准备环境

服务器要求：

* 建议最小硬件配置：2核CPU、2G内存、30G硬盘
* 服务器最好可以访问外网，会有从网上拉取镜像需求，如果服务器不能上网，需要提前下载对应镜像并导入节点

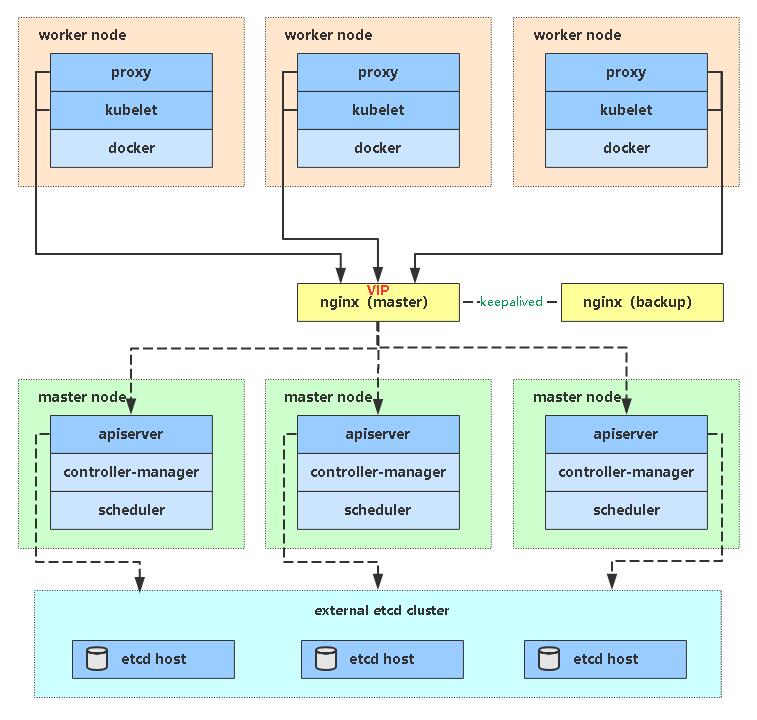
软件环境：

|  |  |
| --- | --- |
| **软件** | **版本** |
| 操作系统 | CentOS7.8\_x64 （mini） |
| Docker | 19-ce |
| Kubernetes | 1.23 |

服务器整体规划：

|  |  |  |
| --- | --- | --- |
| **角色** | **IP** | **其他单装组件** |
| k8s-master1 | 192.168.31.71 | docker，etcd，nginx，keepalived |
| k8s-master2 | 192.168.31.72 | docker，etcd，nginx，keepalived |
| k8s-node1 | 192.168.31.73 | docker，etcd |
| 负载均衡器对外IP | 192.168.31.88 (VIP) |  |

架构图：



### 1.3 操作系统初始化配置

# 关闭防火墙  
systemctl stop firewalld  
systemctl disable firewalld  
  
# 关闭selinux  
sed -i 's/enforcing/disabled/' /etc/selinux/config # 永久  
setenforce 0 # 临时  
  
# 关闭swap  
swapoff -a # 临时  
sed -ri 's/.\*swap.\*/#&/' /etc/fstab # 永久  
  
# 根据规划设置主机名  
hostnamectl set-hostname <hostname>  
  
# 在master添加hosts  
cat >> /etc/hosts << EOF  
192.168.31.71 k8s-master1  
192.168.31.72 k8s-master2  
192.168.31.73 k8s-node1  
EOF  
  
# 将桥接的IPv4流量传递到iptables的链  
cat > /etc/sysctl.d/k8s.conf << EOF  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
EOF  
sysctl --system # 生效  
  
# 时间同步  
yum install ntpdate -y  
ntpdate time.windows.com

## 二、部署Nginx+Keepalived高可用负载均衡器

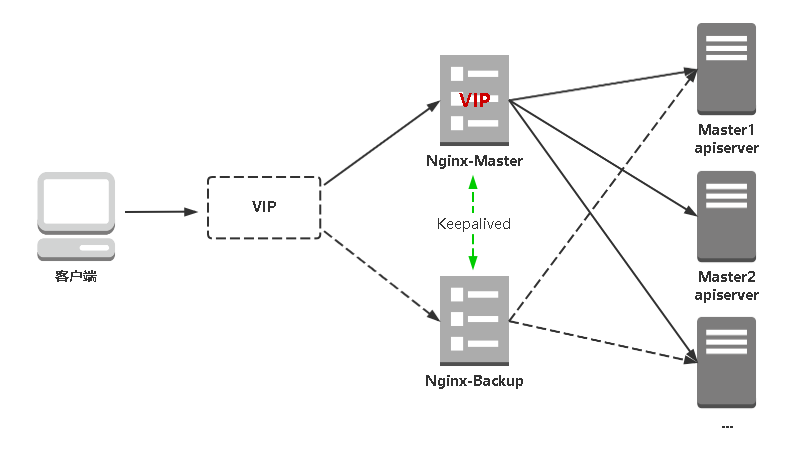
Kubernetes作为容器集群系统，通过健康检查+重启策略实现了Pod故障自我修复能力，通过调度算法实现将Pod分布式部署，并保持预期副本数，根据Node失效状态自动在其他Node拉起Pod，实现了应用层的高可用性。

针对Kubernetes集群，高可用性还应包含以下两个层面的考虑：Etcd数据库的高可用性和Kubernetes Master组件的高可用性。 而kubeadm搭建的K8s集群，Etcd只起了一个，存在单点，所以我们这里会独立搭建一个Etcd集群。

Master节点扮演着总控中心的角色，通过不断与工作节点上的Kubelet和kube-proxy进行通信来维护整个集群的健康工作状态。如果Master节点故障，将无法使用kubectl工具或者API做任何集群管理。

Master节点主要有三个服务kube-apiserver、kube-controller-manager和kube-scheduler，其中kube-controller-manager和kube-scheduler组件自身通过选举机制已经实现了高可用，所以Master高可用主要针对kube-apiserver组件，而该组件是以HTTP API提供服务，因此对他高可用与Web服务器类似，增加负载均衡器对其负载均衡即可，并且可水平扩容。

kube-apiserver高可用架构图：



* Nginx是一个主流Web服务和反向代理服务器，这里用四层实现对apiserver实现负载均衡。
* Keepalived是一个主流高可用软件，基于VIP绑定实现服务器双机热备，在上述拓扑中，Keepalived主要根据Nginx运行状态判断是否需要故障转移（偏移VIP），例如当Nginx主节点挂掉，VIP会自动绑定在Nginx备节点，从而保证VIP一直可用，实现Nginx高可用。

注：为了节省机器，这里与K8s master节点机器复用。也可以独立于k8s集群之外部署，只要nginx与apiserver能通信就行。

### 2.1 安装软件包（主/备）

yum install epel-release -y  
 yum install nginx keepalived -y

### 2.2 Nginx配置文件（主/备一样）

cat > /etc/nginx/nginx.conf << "EOF"  
user nginx;  
worker\_processes auto;  
error\_log /var/log/nginx/error.log;  
pid /run/nginx.pid;  
  
include /usr/share/nginx/modules/\*.conf;  
  
events {  
 worker\_connections 1024;  
}  
  
# 四层负载均衡，为两台Master apiserver组件提供负载均衡  
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 192.168.31.71:6443; # Master1 APISERVER IP:PORT  
 server 192.168.31.72:6443; # Master2 APISERVER IP:PORT  
 }  
   
 server {  
 listen 16443; # 由于nginx与master节点复用，这个监听端口不能是6443，否则会冲突  
 proxy\_pass k8s-apiserver;  
 }  
}  
  
http {  
 log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '  
 '$status $body\_bytes\_sent "$http\_referer" '  
 '"$http\_user\_agent" "$http\_x\_forwarded\_for"';  
  
 access\_log /var/log/nginx/access.log main;  
  
 sendfile on;  
 tcp\_nopush on;  
 tcp\_nodelay on;  
 keepalive\_timeout 65;  
 types\_hash\_max\_size 2048;  
  
 include /etc/nginx/mime.types;  
 default\_type application/octet-stream;  
}  
EOF

### 2.3 keepalived配置文件（Nginx Master）

cat > /etc/keepalived/keepalived.conf << EOF  
global\_defs {   
 notification\_email {   
 acassen@firewall.loc   
 failover@firewall.loc   
 sysadmin@firewall.loc   
 }   
 notification\_email\_from Alexandre.Cassen@firewall.loc   
 smtp\_server 127.0.0.1   
 smtp\_connect\_timeout 30   
 router\_id NGINX\_MASTER  
}   
  
vrrp\_script check\_nginx {  
 script "/etc/keepalived/check\_nginx.sh" # 判断返回状态码  
}  
  
vrrp\_instance VI\_1 {   
 state MASTER   
 interface ens33 # 修改为实际网卡名  
 virtual\_router\_id 51 # VRRP 路由 ID实例，每个实例是唯一的   
 priority 100 # 优先级，备服务器设置 90   
 advert\_int 1 # 指定VRRP 心跳包通告间隔时间，默认1秒   
 authentication {   
 auth\_type PASS   
 auth\_pass 1111   
 }   
 # 虚拟IP  
 virtual\_ipaddress {   
 192.168.31.88/24  
 }   
 track\_script {  
 check\_nginx  
 }   
}  
EOF

* vrrp\_script：指定检查nginx工作状态脚本（根据nginx状态判断是否故障转移）
* virtual\_ipaddress：虚拟IP（VIP）

准备上述配置文件中检查nginx运行状态的脚本：

cat > /etc/keepalived/check\_nginx.sh << "EOF"  
#!/bin/bash  
code=$(curl -k https://127.0.0.1:16443/version -s -o /dev/null -w %{http\_code})  
  
if [ "$code" -ne 200 ];then  
 exit 1  
else  
 exit 0  
fi  
EOF  
chmod +x /etc/keepalived/check\_nginx.sh

### 2.4 keepalived配置文件（Nginx Backup）

cat > /etc/keepalived/keepalived.conf << EOF  
global\_defs {   
 notification\_email {   
 acassen@firewall.loc   
 failover@firewall.loc   
 sysadmin@firewall.loc   
 }   
 notification\_email\_from Alexandre.Cassen@firewall.loc   
 smtp\_server 127.0.0.1   
 smtp\_connect\_timeout 30   
 router\_id NGINX\_BACKUP  
}   
  
vrrp\_script check\_nginx {  
 script "/etc/keepalived/check\_nginx.sh"  
}  
  
vrrp\_instance VI\_1 {   
 state BACKUP   
 interface ens33  
 virtual\_router\_id 51 # VRRP 路由 ID实例，每个实例是唯一的   
 priority 90  
 advert\_int 1  
 authentication {   
 auth\_type PASS   
 auth\_pass 1111   
 }   
 virtual\_ipaddress {   
 192.168.31.88/24  
 }   
 track\_script {  
 check\_nginx  
 }   
}  
EOF

准备上述配置文件中检查nginx运行状态的脚本：

cat > /etc/keepalived/check\_nginx.sh << "EOF"  
#!/bin/bash  
code=$(curl -k https://127.0.0.1:16443/version -s -o /dev/null -w %{http\_code})  
  
if [ "$code" -ne 200 ];then  
 exit 1  
else  
 exit 0  
fi  
EOF  
chmod +x /etc/keepalived/check\_nginx.sh

注：keepalived根据脚本返回状态码（0为工作正常，非0不正常）判断是否故障转移。

### 2.5 启动并设置开机启动

systemctl daemon-reload  
systemctl start nginx  
systemctl start keepalived  
systemctl enable nginx  
systemctl enable keepalived

### 2.6 查看keepalived工作状态

ip addr  
1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
 inet 127.0.0.1/8 scope host lo  
 valid\_lft forever preferred\_lft forever  
 inet6 ::1/128 scope host   
 valid\_lft forever preferred\_lft forever  
2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000  
 link/ether 00:0c:29:04:f7:2c brd ff:ff:ff:ff:ff:ff  
 inet 192.168.31.80/24 brd 192.168.31.255 scope global noprefixroute ens33  
 valid\_lft forever preferred\_lft forever  
 inet 192.168.31.88/24 scope global secondary ens33  
 valid\_lft forever preferred\_lft forever  
 inet6 fe80::20c:29ff:fe04:f72c/64 scope link   
 valid\_lft forever preferred\_lft forever

可以看到，在ens33网卡绑定了192.168.31.88 虚拟IP，说明工作正常。

### 2.7 Nginx+Keepalived高可用测试

关闭主节点Nginx，测试VIP是否漂移到备节点服务器。

在Nginx Master执行 pkill nginx  
在Nginx Backup，ip addr命令查看已成功绑定VIP。

## 三、部署Etcd集群

如果你在学习中遇到问题或者文档有误可联系阿良~ 微信: k8init

Etcd 是一个分布式键值存储系统，Kubernetes使用Etcd进行数据存储，kubeadm搭建默认情况下只启动一个Etcd Pod，存在单点故障，生产环境强烈不建议，所以我们这里使用3台服务器组建集群，可容忍1台机器故障，当然，你也可以使用5台组建集群，可容忍2台机器故障。

|  |  |
| --- | --- |
| **节点名称** | **IP** |
| etcd-1 | 192.168.31.71 |
| etcd-2 | 192.168.31.72 |
| etcd-3 | 192.168.31.73 |

注：为了节省机器，这里与K8s节点机器复用。也可以独立于k8s集群之外部署，只要apiserver能连接到就行。

### 3.1 准备cfssl证书生成工具

cfssl是一个开源的证书管理工具，使用json文件生成证书，相比openssl更方便使用。

找任意一台服务器操作，这里用Master节点。

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  
chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64  
mv cfssl\_linux-amd64 /usr/local/bin/cfssl  
mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson  
mv cfssl-certinfo\_linux-amd64 /usr/bin/cfssl-certinfo

### 3.2 生成Etcd证书

#### 1. 自签证书颁发机构（CA）

创建工作目录：

mkdir -p ~/etcd\_tls  
cd ~/etcd\_tls

自签CA：

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

生成证书：

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

会生成ca.pem和ca-key.pem文件。

#### 2. 使用自签CA签发Etcd HTTPS证书

创建证书申请文件：

cat > server-csr.json << EOF  
{  
 "CN": "etcd",  
 "hosts": [  
 "192.168.31.71",  
 "192.168.31.72",  
 "192.168.31.73"  
 ],  
 "key": {  
 "algo": "rsa",  
 "size": 2048  
 },  
 "names": [  
 {  
 "C": "CN",  
 "L": "BeiJing",  
 "ST": "BeiJing"  
 }  
 ]  
}  
EOF

注：上述文件hosts字段中IP为所有etcd节点的集群内部通信IP，一个都不能少！为了方便后期扩容可以多写几个预留的IP。

生成证书：

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

会生成server.pem和server-key.pem文件。

### 3.3 从Github下载二进制文件

下载地址：<https://github.com/etcd-io/etcd/releases/download/v3.4.9/etcd-v3.4.9-linux-amd64.tar.gz>

### 3.4 部署Etcd集群

以下在节点1上操作，为简化操作，待会将节点1生成的所有文件拷贝到节点2和节点3。

#### 1. 创建工作目录并解压二进制包

mkdir /opt/etcd/{bin,cfg,ssl} -p  
tar zxvf etcd-v3.4.9-linux-amd64.tar.gz  
mv etcd-v3.4.9-linux-amd64/{etcd,etcdctl} /opt/etcd/bin/

#### 2. 创建etcd配置文件

cat > /opt/etcd/cfg/etcd.conf << EOF  
#[Member]  
ETCD\_NAME="etcd-1"  
ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"  
ETCD\_LISTEN\_PEER\_URLS="https://192.168.31.71:2380"  
ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.31.71:2379"  
  
#[Clustering]  
ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.31.71:2380"  
ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.31.71:2379"  
ETCD\_INITIAL\_CLUSTER="etcd-1=https://192.168.31.71:2380,etcd-2=https://192.168.31.72:2380,etcd-3=https://192.168.31.73:2380"  
ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  
ETCD\_INITIAL\_CLUSTER\_STATE="new"  
EOF

* ETCD\_NAME：节点名称，集群中唯一
* ETCD\_DATA\_DIR：数据目录
* ETCD\_LISTEN\_PEER\_URLS：集群通信监听地址
* ETCD\_LISTEN\_CLIENT\_URLS：客户端访问监听地址
* ETCD\_INITIAL\_ADVERTISEPEERURLS：集群通告地址
* ETCD\_ADVERTISE\_CLIENT\_URLS：客户端通告地址
* ETCD\_INITIAL\_CLUSTER：集群节点地址
* ETCD\_INITIAL\_CLUSTER\_TOKEN：集群Token
* ETCD\_INITIAL\_CLUSTER\_STATE：加入集群的当前状态，new是新集群，existing表示加入已有集群

#### 3. systemd管理etcd

cat > /usr/lib/systemd/system/etcd.service << EOF  
[Unit]  
Description=Etcd Server  
After=network.target  
After=network-online.target  
Wants=network-online.target  
  
[Service]  
Type=notify  
EnvironmentFile=/opt/etcd/cfg/etcd.conf  
ExecStart=/opt/etcd/bin/etcd \  
--cert-file=/opt/etcd/ssl/server.pem \  
--key-file=/opt/etcd/ssl/server-key.pem \

--trusted-ca-file=/opt/etcd/ssl/ca.pem \  
--peer-cert-file=/opt/etcd/ssl/server.pem \  
--peer-key-file=/opt/etcd/ssl/server-key.pem \  
--peer-trusted-ca-file=/opt/etcd/ssl/ca.pem \  
--logger=zap  
Restart=on-failure  
LimitNOFILE=65536  
  
[Install]  
WantedBy=multi-user.target  
EOF

#### 4. 拷贝刚才生成的证书

把刚才生成的证书拷贝到配置文件中的路径：

cp ~/etcd\_tls/ca\*pem ~/etcd\_tls/server\*pem /opt/etcd/ssl/

#### 5. 启动并设置开机启动

systemctl daemon-reload  
systemctl start etcd  
systemctl enable etcd

#### 6. 将上面节点1所有生成的文件拷贝到节点2和节点3

scp -r /opt/etcd/ root@192.168.31.72:/opt/  
scp /usr/lib/systemd/system/etcd.service root@192.168.31.72:/usr/lib/systemd/system/

scp -r /opt/etcd/ root@192.168.31.73:/opt/  
scp /usr/lib/systemd/system/etcd.service root@192.168.31.73:/usr/lib/systemd/system/

然后在节点2和节点3分别修改etcd.conf配置文件中的节点名称和当前服务器IP：

vi /opt/etcd/cfg/etcd.conf  
#[Member]  
ETCD\_NAME="etcd-1" # 修改此处，节点2改为etcd-2，节点3改为etcd-3  
ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"  
ETCD\_LISTEN\_PEER\_URLS="https://192.168.31.71:2380" # 修改此处为当前服务器IP  
ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.31.71:2379" # 修改此处为当前服务器IP  
  
#[Clustering]  
ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.31.71:2380" # 修改此处为当前服务器IP  
ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.31.71:2379" # 修改此处为当前服务器IP  
ETCD\_INITIAL\_CLUSTER="etcd-1=https://192.168.31.71:2380,etcd-2=https://192.168.31.72:2380,etcd-3=https://192.168.31.73:2380"  
ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  
ETCD\_INITIAL\_CLUSTER\_STATE="new"

最后启动etcd并设置开机启动，同上。

#### 7. 查看集群状态

ETCDCTL\_API=3 /opt/etcd/bin/etcdctl --cacert=/opt/etcd/ssl/ca.pem --cert=/opt/etcd/ssl/server.pem --key=/opt/etcd/ssl/server-key.pem --endpoints="https://192.168.31.71:2379,https://192.168.31.72:2379,https://192.168.31.73:2379" endpoint health --write-out=table  
  
+----------------------------+--------+-------------+-------+  
| ENDPOINT | HEALTH | TOOK | ERROR |  
+----------------------------+--------+-------------+-------+  
| https://192.168.31.71:2379 | true | 10.301506ms | |  
| https://192.168.31.73:2379 | true | 12.87467ms | |  
| https://192.168.31.72:2379 | true | 13.225954ms | |  
+----------------------------+--------+-------------+-------+

如果输出上面信息，就说明集群部署成功。

如果有问题第一步先看日志：/var/log/message 或 journalctl -u etcd

## 四、安装Docker/kubeadm/kubelet【所有节点】

这里使用Docker作为容器引擎，也可以换成别的，例如containerd

### 4.1 安装Docker

wget https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo -O /etc/yum.repos.d/docker-ce.repo  
yum -y install docker-ce  
systemctl enable docker && systemctl start docker

配置镜像下载加速器：

cat > /etc/docker/daemon.json << EOF

{

"registry-mirrors": ["https://b9pmyelo.mirror.aliyuncs.com"],

"exec-opts": ["native.cgroupdriver=systemd"]

}  
EOF

systemctl restart docker  
docker info

### 4.2 添加阿里云YUM软件源

cat > /etc/yum.repos.d/kubernetes.repo << EOF  
[kubernetes]  
name=Kubernetes  
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64  
enabled=1  
gpgcheck=0  
repo\_gpgcheck=0  
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg  
EOF

### 4.3 安装kubeadm，kubelet和kubectl

由于版本更新频繁，这里指定版本号部署：

yum install -y kubelet-1.23.0 kubeadm-1.23.0 kubectl-1.23.0  
systemctl enable kubelet

## 五、部署Kubernetes Master

如果你在学习中遇到问题或者文档有误可联系阿良~ 微信: k8init

### 5.1 初始化Master1

生成初始化配置文件：

cat > kubeadm-config.yaml << EOF  
apiVersion: kubeadm.k8s.io/v1beta2  
bootstrapTokens:  
- groups:  
 - system:bootstrappers:kubeadm:default-node-token  
 token: 9037x2.tcaqnpaqkra9vsbw  
 ttl: 24h0m0s  
 usages:  
 - signing  
 - authentication  
kind: InitConfiguration  
localAPIEndpoint:  
 advertiseAddress: 192.168.31.71  
 bindPort: 6443  
nodeRegistration:  
 criSocket: /var/run/dockershim.sock  
 name: k8s-master1  
 taints:  
 - effect: NoSchedule  
 key: node-role.kubernetes.io/master  
---  
apiServer:  
 certSANs: # 包含所有Master/LB/VIP IP，一个都不能少！为了方便后期扩容可以多写几个预留的IP。  
 - k8s-master1  
 - k8s-master2  
 - 192.168.31.71  
 - 192.168.31.72  
 - 192.168.31.73

- 192.168.31.88  
 - 127.0.0.1  
 extraArgs:  
 authorization-mode: Node,RBAC  
 timeoutForControlPlane: 4m0s  
apiVersion: kubeadm.k8s.io/v1beta2  
certificatesDir: /etc/kubernetes/pki  
clusterName: kubernetes  
controlPlaneEndpoint: 192.168.31.88:16443 # 负载均衡虚拟IP（VIP）和端口  
controllerManager: {}  
dns:  
 type: CoreDNS  
etcd:  
 external: # 使用外部etcd  
 endpoints:  
 - https://192.168.31.71:2379 # etcd集群3个节点  
 - https://192.168.31.72:2379  
 - https://192.168.31.73:2379  
 caFile: /opt/etcd/ssl/ca.pem # 连接etcd所需证书  
 certFile: /opt/etcd/ssl/server.pem  
 keyFile: /opt/etcd/ssl/server-key.pem  
imageRepository: registry.aliyuncs.com/google\_containers # 由于默认拉取镜像地址k8s.gcr.io国内无法访问，这里指定阿里云镜像仓库地址  
kind: ClusterConfiguration  
kubernetesVersion: v1.23.0 # K8s版本，与上面安装的一致  
networking:  
 dnsDomain: cluster.local  
 podSubnet: 10.244.0.0/16 # Pod网络，与下面部署的CNI网络组件yaml中保持一致  
 serviceSubnet: 10.96.0.0/12 # 集群内部虚拟网络，Pod统一访问入口  
scheduler: {}  
EOF

或者使用配置文件引导：

kubeadm init --config kubeadm-config.yaml  
...  
Your Kubernetes control-plane has initialized successfully!  
  
To start using your cluster, you need to run the following as a regular user:  
  
 mkdir -p $HOME/.kube  
 sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
 sudo chown $(id -u):$(id -g) $HOME/.kube/config  
  
Alternatively, if you are the root user, you can run:  
  
 export KUBECONFIG=/etc/kubernetes/admin.conf  
  
You should now deploy a pod network to the cluster.  
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:  
 https://kubernetes.io/docs/concepts/cluster-administration/addons/  
  
You can now join any number of control-plane nodes by copying certificate authorities  
and service account keys on each node and then running the following as root:  
  
 kubeadm join 192.168.31.88:16443 --token 9037x2.tcaqnpaqkra9vsbw \  
 --discovery-token-ca-cert-hash sha256:b1e726042cdd5df3ce62e60a2f86168cd2e64bff856e061e465df10cd36295b8 \  
 --control-plane   
  
Then you can join any number of worker nodes by running the following on each as root:  
  
kubeadm join 192.168.31.88:16443 --token 9037x2.tcaqnpaqkra9vsbw \  
 --discovery-token-ca-cert-hash sha256:b1e726042cdd5df3ce62e60a2f86168cd2e64bff856e061e465df10cd36295b8

初始化完成后，会有两个join的命令，带有 --control-plane 是用于加入组建多master集群的，不带的是加入节点的。

拷贝kubectl使用的连接k8s认证文件到默认路径：

mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl get node  
NAME STATUS ROLES AGE VERSION  
k8s-master1 NotReady control-plane,master 6m42s v1.23.0

### 5.2 初始化Master2

将Master1节点生成的证书拷贝到Master2：

scp -r /etc/kubernetes/pki/ 192.168.31.72:/etc/kubernetes/

复制加入master join命令在master2执行：

kubeadm join 192.168.31.88:16443 --token 9037x2.tcaqnpaqkra9vsbw \  
 --discovery-token-ca-cert-hash sha256:b1e726042cdd5df3ce62e60a2f86168cd2e64bff856e061e465df10cd36295b8 \  
 --control-plane

拷贝kubectl使用的连接k8s认证文件到默认路径：

mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl get node  
NAME STATUS ROLES AGE VERSION  
k8s-master1 NotReady control-plane,master 28m v1.23.0  
k8s-master2 NotReady control-plane,master 2m12s v1.23.0

注：由于网络插件还没有部署，还没有准备就绪 NotReady

### 5.3 访问负载均衡器测试

找K8s集群中任意一个节点，使用curl查看K8s版本测试，使用VIP访问：

curl -k https://192.168.31.88:16443/version  
{  
 "major": "1",  
 "minor": "20",  
 "gitVersion": "v1.23.0",  
 "gitCommit": "e87da0bd6e03ec3fea7933c4b5263d151aafd07c",  
 "gitTreeState": "clean",  
 "buildDate": "2021-02-18T16:03:00Z",  
 "goVersion": "go1.15.8",  
 "compiler": "gc",  
 "platform": "linux/amd64"

}

可以正确获取到K8s版本信息，说明负载均衡器搭建正常。该请求数据流程：**curl -> vip(nginx) -> apiserver**

通过查看Nginx日志也可以看到转发apiserver IP：

tail /var/log/nginx/k8s-access.log -f  
192.168.31.71 192.168.31.71:6443 - [02/Apr/2021:19:17:57 +0800] 200 423  
192.168.31.71 192.168.31.72:6443 - [02/Apr/2021:19:18:50 +0800] 200 423

## 六、加入Kubernetes Node

在192.168.31.73（Node）执行。

向集群添加新节点，执行在kubeadm init输出的kubeadm join命令：

kubeadm join 192.168.31.88:16443 --token 9037x2.tcaqnpaqkra9vsbw \  
 --discovery-token-ca-cert-hash sha256:e6a724bb7ef8bb363762fbaa088f6eb5975e0c654db038560199a7063735a697

后续其他节点也是这样加入。

注：默认token有效期为24小时，当过期之后，该token就不可用了。这时就需要重新创建token，可以直接使用命令快捷生成：kubeadm token create --print-join-command

## 七、部署网络组件

Calico是一个纯三层的数据中心网络方案，是目前Kubernetes主流的网络方案。

部署Calico：

kubectl apply -f calico.yaml  
kubectl get pods -n kube-system

等Calico Pod都Running，节点也会准备就绪：

kubectl get node  
NAME STATUS ROLES AGE VERSION  
k8s-master1 Ready control-plane,master 50m v1.23.0  
k8s-master2 Ready control-plane,master 24m v1.23.0  
k8s-node1 Ready <none> 20m v1.23.0

## 八、部署 Dashboard

Dashboard是官方提供的一个UI，可用于基本管理K8s资源。

kubectl apply -f kubernetes-dashboard.yaml  
# 查看部署  
kubectl get pods -n kubernetes-dashboard

访问地址：<https://NodeIP:30001>

创建service account并绑定默认cluster-admin管理员集群角色：

kubectl create serviceaccount dashboard-admin -n kube-system  
kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin  
kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret | awk '/dashboard-admin/{print $1}')

使用输出的token登录Dashboard。

