# Kubeadm 构建高可用 Kubernetes 集群

## 实验目标

本实验的目标是基于 Kubeadm 搭建高可用生产的 Kubernetes 集群环境…

## 实验环境

OS: Centos 7.8 Kubernetes: v1.18.10

## Step1. 准备虚拟机测试环境

本次基于 Centos 7.8, 内核版本 3.10.0-1127.el7.x86\_64 基准环境部署

使用 Vagrant 配置 9 台虚拟机环境..

* VIP: 192.168.20.150
  + keepalived/haproxy
  + kube-vip
* 三台 etcd 集群独立部署
* 三台 master HA 高可用部署
* 两台 node 节点

VIP 配置有两个方案，分别是 keepalived/haproxy 或 kube-vip， 对比而言 keepalived/haproxy 使用更广一些，而 kube-vip 配置更简单一些..

# -\*- mode: ruby -\*-  
# # vi: set ft=ruby :  
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!  
  
# VIP: 192.168.20.150  
VAGRANTFILE\_API\_VERSION = "2"  
  
boxes = [  
 {  
 :name => "etcd-1",  
 :eth1 => "192.168.20.151",  
 :mem => "2048",  
 :cpu => "1"  
 },  
 {  
 :name => "etcd-2",  
 :eth1 => "192.168.20.152",  
 :mem => "2048",  
 :cpu => "1"  
 },  
 {  
 :name => "etcd-3",  
 :eth1 => "192.168.20.153",  
 :mem => "2048",  
 :cpu => "1"  
 },  
 {  
 :name => "master-1",  
 :eth1 => "192.168.20.154",  
 :mem => "4096",  
 :cpu => "2"  
 },  
 {  
 :name => "master-2",  
 :eth1 => "192.168.20.155",  
 :mem => "4096",  
 :cpu => "2"  
 },  
 {  
 :name => "master-3",  
 :eth1 => "192.168.20.156",  
 :mem => "4096",  
 :cpu => "2"  
 },  
 {  
 :name => "node-1",  
 :eth1 => "192.168.20.157",  
 :mem => "4096",  
 :cpu => "2"  
 },  
 {  
 :name => "node-2",  
 :eth1 => "192.168.20.158",  
 :mem => "4096",  
 :cpu => "2"  
 },  
  
]  
  
Vagrant.configure(VAGRANTFILE\_API\_VERSION) do |config|  
 config.vm.box = "centos/7"  
 # Turn off shared folders  
 #config.vm.synced\_folder ".", "/vagrant", id: "vagrant-root", disabled: true  
 #config.vm.synced\_folder "~/works/codelab/cka/files", "/files"  
 # config.ssh.private\_key\_path = "~/.ssh/id\_rsa"  
 # config.ssh.forward\_agent = true  
  
 boxes.each do |opts|  
 config.vm.define opts[:name] do |config|  
 config.vm.hostname = opts[:name]  
 config.ssh.insert\_key = true  
 # config.ssh.username = 'vagrant'  
 # config.ssh.password = "vagrant"  
 # config.vm.provision "shell", inline: $script  
 config.vm.provider "virtualbox" do |v|  
 # v.gui = true  
 v.customize ["modifyvm", :id, "--memory", opts[:mem]]  
 v.customize ["modifyvm", :id, "--cpus", opts[:cpu]]  
 end  
 # config.vm.network :public\_network  
 config.vm.network "private\_network", ip: opts[:eth1], auto\_config: true  
 end  
 end  
end

启动虚拟机

# vagrant up  
# 查看机器的启动状态...  
# vagrant status  
Current machine states:  
  
etcd-1 running (virtualbox)  
etcd-2 running (virtualbox)  
etcd-3 running (virtualbox)  
master-1 running (virtualbox)  
master-2 running (virtualbox)  
master-3 running (virtualbox)  
node-1 running (virtualbox)  
node-2 running (virtualbox)  
  
This environment represents multiple VMs. The VMs are all listed  
above with their current state. For more information about a specific  
VM, run `vagrant status NAME`.

## Step2. 准备软件基础环境

需要在所有节点执行如下操作，并重启机器

* install [Container RunTime](https://kubernetes.io/docs/setup/production-environment/container-runtimes/)
* install kubelet/kubectl/kubeadm
* close selinux & close swap

下面是 prek8s.sh 代码：

#!/bin/bash  
sudo yum install -y yum-utils device-mapper-persistent-data lvm2  
sudo yum-config-manager --add-repo \  
 https://download.docker.com/linux/centos/docker-ce.repo  
sudo yum update -y && sudo yum install -y \  
 containerd.io-1.2.13 \  
 docker-ce-19.03.11 \  
 docker-ce-cli-19.03.11  
sudo mkdir /etc/docker  
# Set up the Docker daemon  
cat <<EOF | sudo tee /etc/docker/daemon.json  
{  
 "exec-opts": ["native.cgroupdriver=systemd"],  
 "log-driver": "json-file",  
 "log-opts": {  
 "max-size": "100m"  
 },  
 "storage-driver": "overlay2",  
 "storage-opts": [  
 "overlay2.override\_kernel\_check=true"  
 ]  
}  
EOF  
sudo mkdir -p /etc/systemd/system/docker.service.d  
  
# 安装指定版本的 kubelet/kubectl/kubeadm  
cat <<EOF > /etc/yum.repos.d/kubernetes.repo  
[kubernetes]  
name=Kubernetes  
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64  
enabled=1  
gpgcheck=1  
repo\_gpgcheck=1  
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
EOF  
sudo yum autoremove -y kubelet kubeadm kubectl --disableexcludes=kubernetes   
sudo yum install -y kubelet-1.18.10-0 kubeadm-1.18.10-0 kubectl-1.18.10-0 --disableexcludes=kubernetes   
# 设置开机启动  
sudo systemctl daemon-reload && sudo systemctl enable docker && sudo systemctl restart docker  
sudo systemctl enable --now kubelet  
  
# echo "1" > /proc/sys/net/bridge/bridge-nf-call-iptables  
cat <<EOF > /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
EOF  
sudo sysctl --system  
  
sudo sed -i '/swap/d' /etc/fstab  
sudo swapoff -a  
  
# 配置系统环境  
echo "export LC\_ALL=en\_US.UTF-8" >> /etc/profile  
source /etc/profile  
  
# 将 SELinux 禁用  
setenforce 0  
sed -i 's/^SELINUX=enforcing$/SELINUX=disabled/' /etc/selinux/config

接下来，先登陆所有节点，将 sshd 配置为可以密码登陆..

# 配置 SSH 允许密码登陆  
sed -i 's/^PasswordAuthentication no$/PasswordAuthentication yes/' /etc/ssh/sshd\_config  
systemctl restart sshd

考虑到需要对所有节点安装，进入 etcd-1 节点，对所有节点执行 ansible playbook 进行批量操作..

vagrant ssh etcd-1  
# 生成密钥  
ssh-keygen  
# 拷贝密钥到所有节点，配置免密登陆..  
ssh-copy-id vagrant@192.168.20.x  
# 安装 ansible  
yum install epel-release -y && yum install ansible -y

下面是 prek8s.yaml 的 playbook 配置..

---  
- hosts:   
 - k8s\_node  
 remote\_user: vagrant  
 become: true  
 tasks:  
 - name: copy script  
 copy: src="prek8s.sh" mode="0744" dest="/home/vagrant/prek8s.sh"  
  
 - name: exec prek8s  
 command: /home/vagrant/prek8s.sh  
  
 - name: reboot machine  
 reboot:  
 reboot\_timeout: 600  
 test\_command: whoami

配置主机文件 prek8s

[all:vars]  
ansible\_port=22  
ansible\_user=vagrant  
ansible\_ssh\_pass=vagrant  
ansible\_become\_pass=vagrant  
  
  
[k8s\_node]  
# etcd-cluster  
192.168.20.151  
192.168.20.152  
192.168.20.153  
# k8s-master  
192.168.20.154  
192.168.20.155  
192.168.20.156  
# k8s-node  
192.168.20.157  
192.168.20.158

执行 ansible 进行批量安装…

andible-playbook -i prek8s prek8s.yaml

检查安装结果…

[root@etcd-1 vagrant]# docker --version  
Docker version 19.03.11, build 42e35e61f3  
[root@etcd-1 vagrant]# kubelet --version  
Kubernetes v1.18.10

## Step3. 安装 etcd 集群

一般来说，在一个节点上生成所有的证书并且只分发这些必要的文件到其他节点..

* 前置条件：
  + 三个可以通过 2379/2380 端口互通的主机，此端口也可以 kubeadm 通过配置文件自定义
  + 每个主机必须安装 docker/kubelet/kubadm
  + 可以通过 scp 在主机之间复制文件

1. 将 kubelet 配置为 etcd 的服务管理器

由于 etcd 是首先创建，因此必须通过创建具有更高优先级的新文件来覆盖 kubdadm 提供的 kubelet 单元文件

vagrant ssh etcd-1  
# install kubelet/kubectl/kubeadm 此软件包已经在前面的步骤通过 ansible 脚本安装完成...  
# yum install -y kubelet-1.18.10-0 kubeadm-1.18.10-0 kubectl-1.18.10-0 --disableexcludes=kubernetes  
  
mkdir -p /etc/systemd/system/kubelet.service.d/  
cat << EOF > /etc/systemd/system/kubelet.service.d/20-etcd-service-manager.conf  
[Service]  
ExecStart=  
# Replace "systemd" with the cgroup driver of your container runtime. The default value in the kubelet is "cgroupfs".  
ExecStart=/usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests --cgroup-driver=systemd  
Restart=always  
EOF  
  
systemctl daemon-reload  
systemctl restart kubelet

查看 kubelet， 保证 kubelet 已正常启动…

[root@etcd-1 vagrant]# systemctl status kubelet  
● kubelet.service - kubelet: The Kubernetes Node Agent  
 Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; vendor preset: disabled)  
 Drop-In: /usr/lib/systemd/system/kubelet.service.d  
 └─10-kubeadm.conf  
 /etc/systemd/system/kubelet.service.d  
 └─20-etcd-service-manager.conf  
 Active: active (running) since Mon 2020-11-09 03:11:18 UTC; 1min 32s ago  
 Docs: https://kubernetes.io/docs/  
 Main PID: 1324 (kubelet)  
 Tasks: 13  
 Memory: 38.1M  
 CGroup: /system.slice/kubelet.service  
 └─1324 /usr/bin/kubelet --address=127.0.0.1 --pod-manifest-path=/etc/kubernetes/manifests --cgroup-driver=systemd

注: 对 etcd-1/etcd-2/etd-3 三个节点分别执行上述命令，保证 kubelet 服务正常启动..

1. kubeadm 创建配置文件

以下脚本为每个将要运行 etcd 成员的主机生成一个 kubeadm 配置文件

# 使用 IP 或可解析的主机名替换 HOST0、HOST1 和 HOST2  
export HOST0=192.168.20.151  
export HOST1=192.168.20.152  
export HOST2=192.168.20.153  
  
# 创建临时目录来存储将被分发到其它主机上的文件  
mkdir -p /tmp/${HOST0}/ /tmp/${HOST1}/ /tmp/${HOST2}/  
  
ETCDHOSTS=(${HOST0} ${HOST1} ${HOST2})  
NAMES=("infra0" "infra1" "infra2")  
  
for i in "${!ETCDHOSTS[@]}"; do  
HOST=${ETCDHOSTS[$i]}  
NAME=${NAMES[$i]}  
cat << EOF > /tmp/${HOST}/kubeadmcfg.yaml  
apiVersion: "kubeadm.k8s.io/v1beta2"  
kind: ClusterConfiguration  
etcd:  
 local:  
 serverCertSANs:  
 - "${HOST}"  
 peerCertSANs:  
 - "${HOST}"  
 extraArgs:  
 initial-cluster: infra0=https://${ETCDHOSTS[0]}:2380,infra1=https://${ETCDHOSTS[1]}:2380,infra2=https://${ETCDHOSTS[2]}:2380  
 initial-cluster-state: new  
 name: ${NAME}  
 listen-peer-urls: https://${HOST}:2380  
 listen-client-urls: https://${HOST}:2379  
 advertise-client-urls: https://${HOST}:2379  
 initial-advertise-peer-urls: https://${HOST}:2380  
EOF  
done

输出的文件如下：

[root@etcd-1 vagrant]# tree -L 3 /tmp  
/tmp  
├── 192.168.20.151  
│   └── kubeadmcfg.yaml  
├── 192.168.20.152  
│   └── kubeadmcfg.yaml  
├── 192.168.20.153  
│   └── kubeadmcfg.yaml

1. 生成证书颁发机构

如果您已经拥有 CA，那么唯一的操作是复制 CA 的 crt 和 key 文件到 etc/kubernetes/pki/etcd/ca.crt 和 /etc/kubernetes/pki/etcd/ca.key。复制完这些文件后继续下一步，“为每个成员创建证书”。

如果您还没有 CA，则在 $HOST0（您为 kubeadm 生成配置文件的位置）上运行此命令。

[root@etcd-1 vagrant]# kubeadm init phase certs etcd-ca --kubernetes-version v1.18.10  
W1109 03:17:35.165974 2460 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]  
[certs] Generating "etcd/ca" certificate and key

创建了如下两个文件

* /etc/kubernetes/pki/etcd/ca.crt
* /etc/kubernetes/pki/etcd/ca.key

1. 为每个成员创建证书

kubeadm init phase certs etcd-server --config=/tmp/${HOST2}/kubeadmcfg.yaml  
kubeadm init phase certs etcd-peer --config=/tmp/${HOST2}/kubeadmcfg.yaml   
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST2}/kubeadmcfg.yaml   
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST2}/kubeadmcfg.yaml   
cp -R /etc/kubernetes/pki /tmp/${HOST2}/  
# 清理不可重复使用的证书  
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete  
  
kubeadm init phase certs etcd-server --config=/tmp/${HOST1}/kubeadmcfg.yaml  
kubeadm init phase certs etcd-peer --config=/tmp/${HOST1}/kubeadmcfg.yaml  
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST1}/kubeadmcfg.yaml  
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST1}/kubeadmcfg.yaml  
cp -R /etc/kubernetes/pki /tmp/${HOST1}/  
find /etc/kubernetes/pki -not -name ca.crt -not -name ca.key -type f -delete  
  
kubeadm init phase certs etcd-server --config=/tmp/${HOST0}/kubeadmcfg.yaml  
kubeadm init phase certs etcd-peer --config=/tmp/${HOST0}/kubeadmcfg.yaml  
kubeadm init phase certs etcd-healthcheck-client --config=/tmp/${HOST0}/kubeadmcfg.yaml  
kubeadm init phase certs apiserver-etcd-client --config=/tmp/${HOST0}/kubeadmcfg.yaml  
# 不需要移动 certs 因为它们是给 HOST0 使用的  
  
# 清理不应从此主机复制的证书  
find /tmp/${HOST2} -name ca.key -type f -delete  
find /tmp/${HOST1} -name ca.key -type f -delete

1. 复制证书和 kubeadm 配置

证书已生成，现在必须将它们移动到对应的主机。

scp -r /tmp/${HOST1}/\* vagrant@${HOST1}:  
scp -r /tmp/${HOST2}/\* vagrant@${HOST2}:  
  
USER@HOST $ sudo -Es  
root@HOST $ chown -R root:root pki  
root@HOST $ mv pki /etc/kubernetes/

1. 确保已经所有预期的文件都存在

[root@etcd-1 vagrant]# ls /tmp/192.168.20.151/  
kubeadmcfg.yaml  
[root@etcd-1 vagrant]# tree -L 3 /etc/kubernetes/  
/etc/kubernetes/  
├── manifests  
└── pki  
 ├── apiserver-etcd-client.crt  
 ├── apiserver-etcd-client.key  
 └── etcd  
 ├── ca.crt  
 ├── ca.key  
 ├── healthcheck-client.crt  
 ├── healthcheck-client.key  
 ├── peer.crt  
 ├── peer.key  
 ├── server.crt  
 └── server.key  
  
3 directories, 10 files  
  
[root@etcd-2 vagrant]# ls  
kubeadmcfg.yaml prek8s.sh  
[root@etcd-2 vagrant]# tree -L 3 /etc/kubernetes/  
/etc/kubernetes/  
├── manifests  
└── pki  
 ├── apiserver-etcd-client.crt  
 ├── apiserver-etcd-client.key  
 └── etcd  
 ├── ca.crt  
 ├── healthcheck-client.crt  
 ├── healthcheck-client.key  
 ├── peer.crt  
 ├── peer.key  
 ├── server.crt  
 └── server.key  
  
3 directories, 9 files  
  
[root@etcd-3 vagrant]# ls  
kubeadmcfg.yaml prek8s.sh  
[root@etcd-3 vagrant]# tree -L 3 /etc/kubernetes/  
/etc/kubernetes/  
├── manifests  
└── pki  
 ├── apiserver-etcd-client.crt  
 ├── apiserver-etcd-client.key  
 └── etcd  
 ├── ca.crt  
 ├── healthcheck-client.crt  
 ├── healthcheck-client.key  
 ├── peer.crt  
 ├── peer.key  
 ├── server.crt  
 └── server.key  
  
3 directories, 9 files

1. 创建静态 Pod 清单

既然证书和配置已经就绪，是时候去创建清单了。在每台主机上运行 kubeadm 命令来生成 etcd 使用的静态清单。

root@HOST0 $ kubeadm init phase etcd local --config=/tmp/${HOST0}/kubeadmcfg.yaml  
root@HOST1 $ kubeadm init phase etcd local --config=/home/vagrant/kubeadmcfg.yaml  
root@HOST2 $ kubeadm init phase etcd local --config=/home/vagrant/kubeadmcfg.yaml

1. 可选：检查群集运行状况

export ETCD\_TAG=3.4.3-0  
  
docker run --rm -it \  
--net host \  
-v /etc/kubernetes:/etc/kubernetes k8s.gcr.io/etcd:${ETCD\_TAG} etcdctl \  
--cert /etc/kubernetes/pki/etcd/peer.crt \  
--key /etc/kubernetes/pki/etcd/peer.key \  
--cacert /etc/kubernetes/pki/etcd/ca.crt \  
--endpoints https://${HOST0}:2379 endpoint health --cluster  
...  
https://192.168.20.151:2379 is healthy: successfully committed proposal: took = 9.856771ms  
https://192.168.20.153:2379 is healthy: successfully committed proposal: took = 17.322191ms  
https://192.168.20.152:2379 is healthy: successfully committed proposal: took = 17.814217ms

* 将 ${ETCD\_TAG} 设置为你的 etcd 镜像的版本标签，例如 3.4.3-0。要查看 kubeadm 使用的 etcd 镜像和标签，请执行 kubeadm config images list –kubernetes-version ${K8S\_VERSION}，其中 ${K8S\_VERSION} 是 v1.18.10 作为例子。
* 将 ${HOST0} 设置为要测试的主机的 IP 地址

一旦拥有了一个正常工作的 3 成员的 etcd 集群，就可以基于 使用 kubeadm 的外部 etcd 方法， 继续部署一个高可用的控制平面。

## Step4. 准备 VIP 负载均衡器

本次使用 VIP: 192.168.20.150 使用端口 6443..

对应的三个 Master 节点 IP - 192.168.20.154 - 192.168.20.155 - 192.168.20.156

负载均衡虚拟 VIP 主要解决 Kubernetes Master 节点高可用性问题。

常见 keepalived 和 haproxy的组合已经存在了很时间，并且可以用于生产环境。需要注意的是 Keepalived 需要两个配置文件，分别是服务配置和运行状态检查脚本， haproxy 需要一个配置文件，所以要提供 VIP 需要 keepalived 和 haproxy 进行配合才能完成(由于太常见，网上的资源也比较丰富，这里不打算继续介绍)。

作为传统 keepalived 和 haproxy 的替代方法，社区提供了一个 kube-vip 的服务实现，与 keepalived 一样，协商 VIP 也必须位于同一 IP 子网中，同样 与 haproxy 一样，也是基于流的负载均衡，允许 TLS 终止其背后的 apiserver 实例。

考虑到此方案比较简单易用，本实验的负载均衡器选择 kube-vip…

编写 master-1 节点的负载均衡器配置..

mkdir -p /etc/kube-vip  
cat <<EOF > /etc/kube-vip/config.yaml  
localPeer:  
 id: master-1  
 address: 192.168.20.154  
 port: 10000  
remotePeers:  
- id: master-2  
 address: 192.168.20.155  
 port: 10000  
- id: master-3  
 address: 192.168.20.156  
 port: 10000  
vip: 192.168.20.150  
gratuitousARP: true  
singleNode: false  
startAsLeader: true  
interface: eth1  
loadBalancers:  
- name: API Server Load Balancer  
 type: tcp  
 port: 6443  
 bindToVip: false  
 backends:  
 - port: 6443  
 address: 192.168.20.155  
 - port: 6443  
 address: 192.168.20.156  
EOF

由于使用 vagrant 生成的虚拟机 IP 配置在 eth1 网卡上，这里要注意指定网卡名，另外 startAsLeader 参数指定默认 VIP 绑定的节点，这里指定为 etcd-1 节点，其后两个节点黑默认值为 false。

编写 master-2 节点的负载均衡器配置..

mkdir -p /etc/kube-vip  
cat <<EOF > /etc/kube-vip/config.yaml  
localPeer:  
 id: master-2  
 address: 192.168.20.155  
 port: 10000  
remotePeers:  
- id: master-1  
 address: 192.168.20.154  
 port: 10000  
- id: master-3  
 address: 192.168.20.156  
 port: 10000  
vip: 192.168.20.150  
gratuitousARP: true  
singleNode: false  
startAsLeader: false  
interface: eth1  
loadBalancers:  
- name: API Server Load Balancer  
 type: tcp  
 port: 6443  
 bindToVip: false  
 backends:  
 - port: 6443  
 address: 192.168.20.154  
 - port: 6443  
 address: 192.168.20.156  
EOF

编写 master-3 节点的负载均衡器配置..

mkdir -p /etc/kube-vip  
cat <<EOF > /etc/kube-vip/config.yaml  
localPeer:  
 id: master-3  
 address: 192.168.20.156  
 port: 10000  
remotePeers:  
- id: master-1  
 address: 192.168.20.154  
 port: 10000  
- id: master-2  
 address: 192.168.20.155  
 port: 10000  
vip: 192.168.20.150  
gratuitousARP: true  
singleNode: false  
startAsLeader: false  
interface: eth1  
loadBalancers:  
- name: API Server Load Balancer  
 type: tcp  
 port: 6443  
 bindToVip: false  
 backends:  
 - port: 6443  
 address: 192.168.20.154  
 - port: 6443  
 address: 192.168.20.155  
EOF

注意: 由于 kubeadm init/join 还未运行，默认的 kubelet 服务是无法启动的，此时生成的静态 POD 配置文件是无法运行的。

这里选择使用 Kubernetes 静态容器的方式部署, 对应的配置文件如下:

# /etc/kubernetes/manifests/kube-vip.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
 creationTimestamp: null  
 name: kube-vip  
 namespace: kube-system  
spec:  
 containers:  
 - command:  
 - /kube-vip  
 - start  
 - -c  
 - /vip.yaml  
 image: 'plndr/kube-vip:0.1.1'  
 name: kube-vip  
 resources: {}  
 securityContext:  
 capabilities:  
 add:  
 - NET\_ADMIN  
 - SYS\_TIME  
 volumeMounts:  
 - mountPath: /vip.yaml  
 name: config  
 hostNetwork: true  
 volumes:  
 - hostPath:  
 path: /etc/kube-vip/config.yaml  
 name: config  
status: {}

也可以运行容器生成此配置..

# docker run -it --rm plndr/kube-vip:0.1.1 /kube-vip sample manifest \  
 | sed "s|plndr/kube-vip:'|plndr/kube-vip:0.1.1'|" \  
 | sudo tee /etc/kubernetes/manifests/kube-vip.yaml  
# 执行上述命令，生成配置文件   
# /etc/kubernetes/manifests/kube-vip.yaml

## Step5. 安装 Kubernetes 控制平面

使用 nc 工具对 VIP 负载均衡器测试连接，发现服务无法连接，这是因为 kubelet 服务配置无效并没有启动，也没有对应的 Kubernetes apiserver 服务，因此是正常现象。

#nc -v LOAD\_BALANCER\_IP PORT  
nc -v 192.168.20.150 6443

注：kubelet 要正常启动，需要 kubeadm init/join 命令生成对应的配置才可以正常提供服务…

由于使用的是外部的 etcd 集群，需要配置 kubeadm-config.yaml

cat <<EOF > kubeadm-config.yaml  
apiVersion: kubeadm.k8s.io/v1beta2  
kind: ClusterConfiguration  
kubernetesVersion: v1.18.10  
image-repository: registry.cn-hangzhou.aliyuncs.com/google\_containers  
controlPlaneEndpoint: "192.168.20.150:6443"  
etcd:  
 external:  
 endpoints:  
 - https://192.168.20.151:2379  
 - https://192.168.20.152:2379  
 - https://192.168.20.153:2379  
 caFile: /etc/kubernetes/pki/etcd/ca.crt  
 certFile: /etc/kubernetes/pki/apiserver-etcd-client.crt  
 keyFile: /etc/kubernetes/pki/apiserver-etcd-client.key  
EOF

* kubernetesVersion: 指定 kubernetes 版本..
* image-repository: 这里指定镜像仓库的软件源
* controlPlaneEndpoint: 指定虚拟 VIP 与服务端口
* etcd:external:endpoints: 指定 etcd 集群的连接地址
* etcd:external:caFile: 指定 etcd 集群的证书

由于使用了外部的 ETCD，需要将外部的 etcd 根证书与etcd 证书拷贝到三个 master 节点..

# 进入 etcd-1 节点执行  
vagrant ssh etcd-1  
export CONTROL\_PLANE="vagrant@192.168.20.154"  
scp /etc/kubernetes/pki/etcd/ca.crt "${CONTROL\_PLANE}":  
scp /etc/kubernetes/pki/apiserver-etcd-client.crt "${CONTROL\_PLANE}":  
scp /etc/kubernetes/pki/apiserver-etcd-client.key "${CONTROL\_PLANE}":  
  
# 进入 master-1 节点  
vagrant ssh master-1  
mkdir -p /etc/kubernetes/pki/etcd  
cp ca.crt /etc/kubernetes/pki/etcd/  
cp apiserver-etcd-client.\* /etc/kubernetes/pki/

注: 需要将 etcd 证书拷贝到三个 Master 节点…

登陆第一个 Master 节点，执行如下命令, 开始安装控制平面。

vagrant ssh master-1  
kubeadm init --config kubeadm-config.yaml --upload-certs

* –upload-certs 标志用来将在所有控制平面实例之间的共享证书上传到集群, 当 –upload-certs 与 kubeadm init 一起使用时，主控制平面的证书被加密并上传到 kubeadm-certs 密钥中

注: 如果不使用外部 etcd 的部署方式，下面是对应的等效命令，由于 vagrant 使用 eth0 的网关，所以此种部署方式，使用的 etcd 基于 10.0.2.15 部署，造成其他节点的 etcd 无法部署成功，可以改变默认网关解决..

# kubeadm init --image-repository registry.cn-hangzhou.aliyuncs.com/google\_containers --kubernetes-version v1.18.10 --control-plane-endpoint "192.168.20.150:6443" --pod-network-cidr=172.20.0.0/16 --service-cidr=10.32.0.0/24 --apiserver-advertise-address 192.168.20.154 --upload-certs

下面是对应的输出：

[root@master-1 vagrant]# kubeadm init --config kubeadm-config.yaml --upload-certs  
W1109 03:34:11.838552 1059 strict.go:54] error unmarshaling configuration schema.GroupVersionKind{Group:"kubeadm.k8s.io", Version:"v1beta2", Kind:"ClusterConfiguration"}: error unmarshaling JSON: while decoding JSON: json: unknown field "image-repository"  
W1109 03:34:11.841796 1059 configset.go:202] WARNING: kubeadm cannot validate component configs for API groups [kubelet.config.k8s.io kubeproxy.config.k8s.io]  
[init] Using Kubernetes version: v1.18.10  
[preflight] Running pre-flight checks  
[preflight] Pulling images required for setting up a Kubernetes cluster  
[preflight] This might take a minute or two, depending on the speed of your internet connection  
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Starting the kubelet  
[certs] Using certificateDir folder "/etc/kubernetes/pki"  
[certs] Generating "ca" certificate and key  
[certs] Generating "apiserver" certificate and key  
[certs] apiserver serving cert is signed for DNS names [master-1 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.0.2.15 192.168.20.150]  
[certs] Generating "apiserver-kubelet-client" certificate and key  
[certs] Generating "front-proxy-ca" certificate and key  
[certs] Generating "front-proxy-client" certificate and key  
[certs] External etcd mode: Skipping etcd/ca certificate authority generation  
[certs] External etcd mode: Skipping etcd/server certificate generation  
[certs] External etcd mode: Skipping etcd/peer certificate generation  
[certs] External etcd mode: Skipping etcd/healthcheck-client certificate generation  
[certs] External etcd mode: Skipping apiserver-etcd-client certificate generation  
[certs] Generating "sa" key and public key  
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"  
[kubeconfig] Writing "admin.conf" kubeconfig file  
[kubeconfig] Writing "kubelet.conf" kubeconfig file  
[kubeconfig] Writing "controller-manager.conf" kubeconfig file  
[kubeconfig] Writing "scheduler.conf" kubeconfig file  
[control-plane] Using manifest folder "/etc/kubernetes/manifests"  
[control-plane] Creating static Pod manifest for "kube-apiserver"  
[control-plane] Creating static Pod manifest for "kube-controller-manager"  
W1109 03:35:09.462062 1059 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[control-plane] Creating static Pod manifest for "kube-scheduler"  
W1109 03:35:09.462787 1059 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s  
[apiclient] All control plane components are healthy after 17.013461 seconds  
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace  
[kubelet] Creating a ConfigMap "kubelet-config-1.18" in namespace kube-system with the configuration for the kubelets in the cluster  
[upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace  
[upload-certs] Using certificate key:  
b25f558da314047ac6647cd8d4ab193153020f8fd83ec08c8db5e686db3a1e16  
[mark-control-plane] Marking the node master-1 as control-plane by adding the label "node-role.kubernetes.io/master=''"  
[mark-control-plane] Marking the node master-1 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]  
[bootstrap-token] Using token: tw47nq.ddzmm29cyv7fwr30  
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles  
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to get nodes  
[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials  
[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token  
[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster  
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace  
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key  
[addons] Applied essential addon: CoreDNS  
[addons] Applied essential addon: kube-proxy  
  
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  
  
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 the control-plane node running the following command on each as root:  
  
 kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 \  
 --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f \  
 --control-plane --certificate-key b25f558da314047ac6647cd8d4ab193153020f8fd83ec08c8db5e686db3a1e16  
  
Please note that the certificate-key gives access to cluster sensitive data, keep it secret!  
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use  
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.  
  
Then you can join any number of worker nodes by running the following on each as root:  
  
kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 \  
 --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f  
  
[root@master-1 vagrant]# exit  
exit  
[root@master-1 vagrant]# mkdir -p $HOME/.kube  
[root@master-1 vagrant]# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
[root@master-1 vagrant]# sudo chown $(id -u):$(id -g) $HOME/.kube/config  
[root@master-1 vagrant]# kubectl get no  
NAME STATUS ROLES AGE VERSION  
master-1 NotReady master 41s v1.18.10

在 Master-2 节点执行控制平台添加，发现 kube-vip.yaml 静态POD 影响了 kubeadm 的执行，因此，这里先移除，后面在加上…

[root@master-2 vagrant]# kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f --control-plane --certificate-key b25f558da314047ac6647cd8d4ab193153020f8fd83ec08c8db5e686db3a1e16  
[preflight] Running pre-flight checks  
error execution phase preflight: [preflight] Some fatal errors occurred:  
 [ERROR DirAvailable--etc-kubernetes-manifests]: /etc/kubernetes/manifests is not empty  
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`  
To see the stack trace of this error execute with --v=5 or higher  
[root@master-2 vagrant]# ls /etc/kubernetes/manifests  
kube-vip.yaml  
[root@master-2 vagrant]# rm -rf /etc/kubernetes/manifests/kube-vip.yaml

[root@master-2 vagrant]# kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f --control-plane --certificate-key b25f558da314047ac6647cd8d4ab193153020f8fd83ec08c8db5e686db3a1e16  
[preflight] Running pre-flight checks  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'  
[preflight] Running pre-flight checks before initializing the new control plane instance  
[preflight] Pulling images required for setting up a Kubernetes cluster  
[preflight] This might take a minute or two, depending on the speed of your internet connection  
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'  
[download-certs] Downloading the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace  
[certs] Using certificateDir folder "/etc/kubernetes/pki"  
[certs] Generating "apiserver-kubelet-client" certificate and key  
[certs] Generating "apiserver" certificate and key  
[certs] apiserver serving cert is signed for DNS names [master-2 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.0.2.15 192.168.20.150]  
[certs] Generating "front-proxy-client" certificate and key  
[certs] Valid certificates and keys now exist in "/etc/kubernetes/pki"  
[certs] Using the existing "sa" key  
[kubeconfig] Generating kubeconfig files  
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"  
[kubeconfig] Writing "admin.conf" kubeconfig file  
[kubeconfig] Writing "controller-manager.conf" kubeconfig file  
[kubeconfig] Writing "scheduler.conf" kubeconfig file  
[control-plane] Using manifest folder "/etc/kubernetes/manifests"  
[control-plane] Creating static Pod manifest for "kube-apiserver"  
W1109 03:41:11.731541 1885 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[control-plane] Creating static Pod manifest for "kube-controller-manager"  
W1109 03:41:11.736914 1885 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[control-plane] Creating static Pod manifest for "kube-scheduler"  
W1109 03:41:11.738352 1885 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[check-etcd] Skipping etcd check in external mode  
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in the kube-system namespace  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Starting the kubelet  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
[control-plane-join] using external etcd - no local stacked instance added  
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace  
[mark-control-plane] Marking the node master-2 as control-plane by adding the label "node-role.kubernetes.io/master=''"  
[mark-control-plane] Marking the node master-2 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]  
  
This node has joined the cluster and a new control plane instance was created:  
  
\* Certificate signing request was sent to apiserver and approval was received.  
\* The Kubelet was informed of the new secure connection details.  
\* Control plane (master) label and taint were applied to the new node.  
\* The Kubernetes control plane instances scaled up.  
  
  
To start administering your cluster from this node, 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  
  
Run 'kubectl get nodes' to see this node join the cluster.  
[root@master-2 vagrant]# exit  
exit  
[vagrant@master-2 ~]$ mkdir -p $HOME/.kube  
[vagrant@master-2 ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
[vagrant@master-2 ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config  
[vagrant@master-2 ~]$ kubectl get no  
NAME STATUS ROLES AGE VERSION  
master-1 NotReady master 6m46s v1.18.10  
master-2 NotReady master 52s v1.18.10

接下来继续创建第三个控制平面 master-3

[root@master-3 vagrant]# rm -rf /etc/kubernetes/manifests/kube-vip.yaml  
[root@master-3 vagrant]# kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 \  
> --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f \  
> --control-plane --certificate-key b25f558da314047ac6647cd8d4ab193153020f8fd83ec08c8db5e686db3a1e16  
[preflight] Running pre-flight checks  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'  
[preflight] Running pre-flight checks before initializing the new control plane instance  
[preflight] Pulling images required for setting up a Kubernetes cluster  
[preflight] This might take a minute or two, depending on the speed of your internet connection  
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'  
[download-certs] Downloading the certificates in Secret "kubeadm-certs" in the "kube-system" Namespace  
[certs] Using certificateDir folder "/etc/kubernetes/pki"  
[certs] Generating "front-proxy-client" certificate and key  
[certs] Generating "apiserver" certificate and key  
[certs] apiserver serving cert is signed for DNS names [master-3 kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 10.0.2.15 192.168.20.150]  
[certs] Generating "apiserver-kubelet-client" certificate and key  
[certs] Valid certificates and keys now exist in "/etc/kubernetes/pki"  
[certs] Using the existing "sa" key  
[kubeconfig] Generating kubeconfig files  
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"  
[kubeconfig] Writing "admin.conf" kubeconfig file  
[kubeconfig] Writing "controller-manager.conf" kubeconfig file  
[kubeconfig] Writing "scheduler.conf" kubeconfig file  
[control-plane] Using manifest folder "/etc/kubernetes/manifests"  
[control-plane] Creating static Pod manifest for "kube-apiserver"  
W1109 03:43:26.504145 1133 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[control-plane] Creating static Pod manifest for "kube-controller-manager"  
W1109 03:43:26.509944 1133 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[control-plane] Creating static Pod manifest for "kube-scheduler"  
W1109 03:43:26.510738 1133 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"  
[check-etcd] Skipping etcd check in external mode  
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in the kube-system namespace  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Starting the kubelet  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
[control-plane-join] using external etcd - no local stacked instance added  
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace  
[mark-control-plane] Marking the node master-3 as control-plane by adding the label "node-role.kubernetes.io/master=''"  
[mark-control-plane] Marking the node master-3 as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]  
  
This node has joined the cluster and a new control plane instance was created:  
  
\* Certificate signing request was sent to apiserver and approval was received.  
\* The Kubelet was informed of the new secure connection details.  
\* Control plane (master) label and taint were applied to the new node.  
\* The Kubernetes control plane instances scaled up.  
  
  
To start administering your cluster from this node, 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  
  
Run 'kubectl get nodes' to see this node join the cluster.  
  
[root@master-3 vagrant]# exit  
exit  
[vagrant@master-3 ~]$  
[vagrant@master-3 ~]$ mkdir -p $HOME/.kube  
[vagrant@master-3 ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
[vagrant@master-3 ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config  
[vagrant@master-3 ~]$ kubectl get no  
NAME STATUS ROLES AGE VERSION  
master-1 NotReady master 8m42s v1.18.10  
master-2 NotReady master 2m48s v1.18.10  
master-3 NotReady master 34s v1.18.10

至此完成三个控制平面的创建， 接下来将 master-2/master-3 节点添加 kube-vip 静态 pod 配置文件…

vagrant ssh master-2  
# docker run -it --rm plndr/kube-vip:0.1.1 /kube-vip sample manifest \  
 | sed "s|plndr/kube-vip:'|plndr/kube-vip:0.1.1'|" \  
 | sudo tee /etc/kubernetes/manifests/kube-vip.yaml  
# 执行上述命令，生成配置文件   
# /etc/kubernetes/manifests/kube-vip.yaml

## Step6. 安装 Kubernetes 工作平面

登陆 node-1 节点创建工作节点..

[root@node-1 vagrant]# kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 \  
> --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f  
W1109 03:46:48.238647 1011 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane settings will be ignored when control-plane flag is not set.  
[preflight] Running pre-flight checks  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'  
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in the kube-system namespace  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Starting the kubelet  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
  
This node has joined the cluster:  
\* Certificate signing request was sent to apiserver and a response was received.  
\* The Kubelet was informed of the new secure connection details.  
  
Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

登陆 node-2 节点创建工作节点..

[root@node-2 vagrant]# kubeadm join 192.168.20.150:6443 --token tw47nq.ddzmm29cyv7fwr30 \  
> --discovery-token-ca-cert-hash sha256:940c54c02d4e058b983476692c2387ff521fff75b4db5463423e5469a52c7f3f  
W1109 03:46:51.852344 1016 join.go:346] [preflight] WARNING: JoinControlPane.controlPlane settings will be ignored when control-plane flag is not set.  
[preflight] Running pre-flight checks  
[preflight] Reading configuration from the cluster...  
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'  
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.18" ConfigMap in the kube-system namespace  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"  
[kubelet-start] Starting the kubelet  
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...  
  
This node has joined the cluster:  
\* Certificate signing request was sent to apiserver and a response was received.  
\* The Kubelet was informed of the new secure connection details.  
  
Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

登陆 master-1 节点查看节点工作状态：

[vagrant@master-1 ~]$ kubectl get no  
NAME STATUS ROLES AGE VERSION  
master-1 NotReady master 13m v1.18.10  
master-2 NotReady master 7m43s v1.18.10  
master-3 NotReady master 5m29s v1.18.10  
node-1 NotReady <none> 2m6s v1.18.10  
node-2 NotReady <none> 2m6s v1.18.10

查检节点发现 API Server 默认以默认网关所在网卡进行服务发现绑定，所以这里的 INTERNAL-IP 地址都是 10.0.2.15， 需要修改 apiserver 的配置..

[root@master-1 manifests]# kubectl get no -o wide  
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME  
master-1 Ready master 122m v1.18.10 10.0.2.15 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
master-2 Ready master 116m v1.18.10 10.0.2.15 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
master-3 Ready master 114m v1.18.10 10.0.2.15 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
node-1 Ready <none> 111m v1.18.10 10.0.2.15 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
node-2 Ready <none> 111m v1.18.10 10.0.2.15 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11

改动方法如下：

# /etc/kubernetes/manifests/kube-apiserver.yaml  
# --advertise-address=10.0.2.15  
# 1. 变更为各个 master 节点的 IP  
sed -i 's/10.0.2.15/192.168.20.154/g' /etc/kubernetes/manifests/kube-apiserver.yaml

对于 master-1/master-2/master-3 三个节点更新 Kubernetes 服务发现 IP..

# 2. 修改 kubelet 配置 kubelet --node-ip 192.168.20.154  
sed -i 's#ExecStart=/usr/bin/kubelet#ExecStart=/usr/bin/kubelet --node-ip 192.168.20.154#g' /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf  
systemctl daemon-reload && systemctl restart kubelet

对于所有的节点都要改新对应的 kubelet 参数, 以确认使用 eth1 网卡..

[vagrant@master-3 ~]$ kubectl get no -o wide  
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME  
master-1 Ready master 138m v1.18.10 192.168.20.154 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
master-2 Ready master 132m v1.18.10 192.168.20.155 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
master-3 Ready master 129m v1.18.10 192.168.20.156 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
node-1 Ready <none> 126m v1.18.10 192.168.20.157 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
node-2 Ready <none> 126m v1.18.10 192.168.20.158 <none> CentOS Linux 7 (Core) 3.10.0-1127.el7.x86\_64 docker://19.3.11  
[root@master-1 manifests]# kubectl get po -o wide -A  
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  
kube-system coredns-66bff467f8-22zb4 0/1 ContainerCreating 0 40m <none> node-1 <none> <none>  
kube-system coredns-66bff467f8-pnm7p 0/1 ContainerCreating 0 40m <none> node-2 <none> <none>  
kube-system kube-apiserver-master-1 1/1 Running 0 28m 192.168.20.154 master-1 <none> <none>  
kube-system kube-apiserver-master-2 1/1 Running 0 35m 192.168.20.155 master-2 <none> <none>  
kube-system kube-apiserver-master-3 1/1 Running 0 34m 192.168.20.156 master-3 <none> <none>  
kube-system kube-controller-manager-master-1 1/1 Running 1 40m 192.168.20.154 master-1 <none> <none>  
kube-system kube-controller-manager-master-2 1/1 Running 0 40m 192.168.20.155 master-2 <none> <none>  
kube-system kube-controller-manager-master-3 1/1 Running 0 40m 192.168.20.156 master-3 <none> <none>  
kube-system kube-proxy-56lq7 1/1 Running 0 39m 192.168.20.157 node-1 <none> <none>  
kube-system kube-proxy-f7djn 1/1 Running 0 21s 192.168.20.158 node-2 <none> <none>  
kube-system kube-proxy-hvss2 1/1 Running 0 39m 192.168.20.154 master-1 <none> <none>  
kube-system kube-proxy-lwzbh 1/1 Running 0 40m 192.168.20.155 master-2 <none> <none>  
kube-system kube-proxy-mzdpz 1/1 Running 0 39m 192.168.20.156 master-3 <none> <none>  
kube-system kube-scheduler-master-1 1/1 Running 1 40m 192.168.20.154 master-1 <none> <none>  
kube-system kube-scheduler-master-2 1/1 Running 0 40m 192.168.20.155 master-2 <none> <none>  
kube-system kube-scheduler-master-3 1/1 Running 0 40m 192.168.20.156 master-3 <none> <none>  
kube-system kube-vip-master-1 1/1 Running 0 40m 192.168.20.154 master-1 <none> <none>  
kube-system kube-vip-master-2 1/1 Running 0 40m 192.168.20.155 master-2 <none> <none>  
kube-system kube-vip-master-3 1/1 Running 0 40m 192.168.20.156 master-3 <none> <none>

## Step7. 配置容器网络环境

Kubernetes 的网络方案很多，我们这里使用 canal 做为本次实验的容器网络.. 最后我们还是以传统的 Haproxy+keepalived 方式使用我们的 VIP

[vagrant@master-1 ~]$ kubectl get po -o wide -A  
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  
kube-system canal-95h6s 3/3 Running 0 7m34s 192.168.20.158 node-2 <none> <none>  
kube-system canal-96qm6 3/3 Running 0 7m35s 192.168.20.156 master-3 <none> <none>  
kube-system canal-r8kmj 3/3 Running 0 7m35s 192.168.20.154 master-1 <none> <none>  
kube-system canal-r9x9r 3/3 Running 0 7m35s 192.168.20.155 master-2 <none> <none>  
kube-system canal-tf2nk 3/3 Running 0 7m35s 192.168.20.157 node-1 <none> <none>  
kube-system haproxy-master-1 1/1 Running 4 7m34s 192.168.20.154 master-1 <none> <none>  
kube-system haproxy-master-2 1/1 Running 4 7m34s 192.168.20.155 master-2 <none> <none>  
kube-system haproxy-master-3 1/1 Running 5 7m34s 192.168.20.156 master-3 <none> <none>  
kube-system keepalived-master-1 1/1 Running 0 7m34s 192.168.20.154 master-1 <none> <none>  
kube-system keepalived-master-2 1/1 Running 0 7m33s 192.168.20.155 master-2 <none> <none>  
kube-system keepalived-master-3 1/1 Running 0 7m33s 192.168.20.156 master-3 <none> <none>  
kube-system kube-apiserver-master-1 1/1 Running 6 7m33s 192.168.20.154 master-1 <none> <none>  
kube-system kube-apiserver-master-2 1/1 Running 0 7m33s 192.168.20.155 master-2 <none> <none>  
kube-system kube-apiserver-master-3 1/1 Running 7 7m33s 192.168.20.156 master-3 <none> <none>  
kube-system kube-controller-manager-master-1 1/1 Running 2 7m33s 192.168.20.154 master-1 <none> <none>  
kube-system kube-controller-manager-master-2 1/1 Running 1 7m33s 192.168.20.155 master-2 <none> <none>  
kube-system kube-controller-manager-master-3 1/1 Running 3 7m33s 192.168.20.156 master-3 <none> <none>  
kube-system kube-proxy-2gxxk 1/1 Running 0 7m21s 192.168.20.157 node-1 <none> <none>  
kube-system kube-proxy-gk8qf 1/1 Running 0 7m30s 192.168.20.156 master-3 <none> <none>  
kube-system kube-proxy-qd58x 1/1 Running 0 7m22s 192.168.20.154 master-1 <none> <none>  
kube-system kube-proxy-rcc57 1/1 Running 0 7m26s 192.168.20.155 master-2 <none> <none>  
kube-system kube-proxy-zk694 1/1 Running 0 7m28s 192.168.20.158 node-2 <none> <none>  
kube-system kube-scheduler-master-1 1/1 Running 2 7m33s 192.168.20.154 master-1 <none> <none>  
kube-system kube-scheduler-master-2 1/1 Running 1 7m33s 192.168.20.155 master-2 <none> <none>  
kube-system kube-scheduler-master-3 1/1 Running 3 7m32s 192.168.20.156 master-3 <none> <none>

至此完成高可用实验…