Kubeadm创建高可用Kubernetes v1.10.0集群

# 节点规划

|  |  |  |
| --- | --- | --- |
| IP | 主机名 | Role |
| 192.168.40.23 | master01 | Master、keepalived |
| 192.168.40.24 | master02 | Master、keepalived |
| 192.168.40.25 | master03 | Master、keepalived |
| 192.168.40.22 | None | VIP |

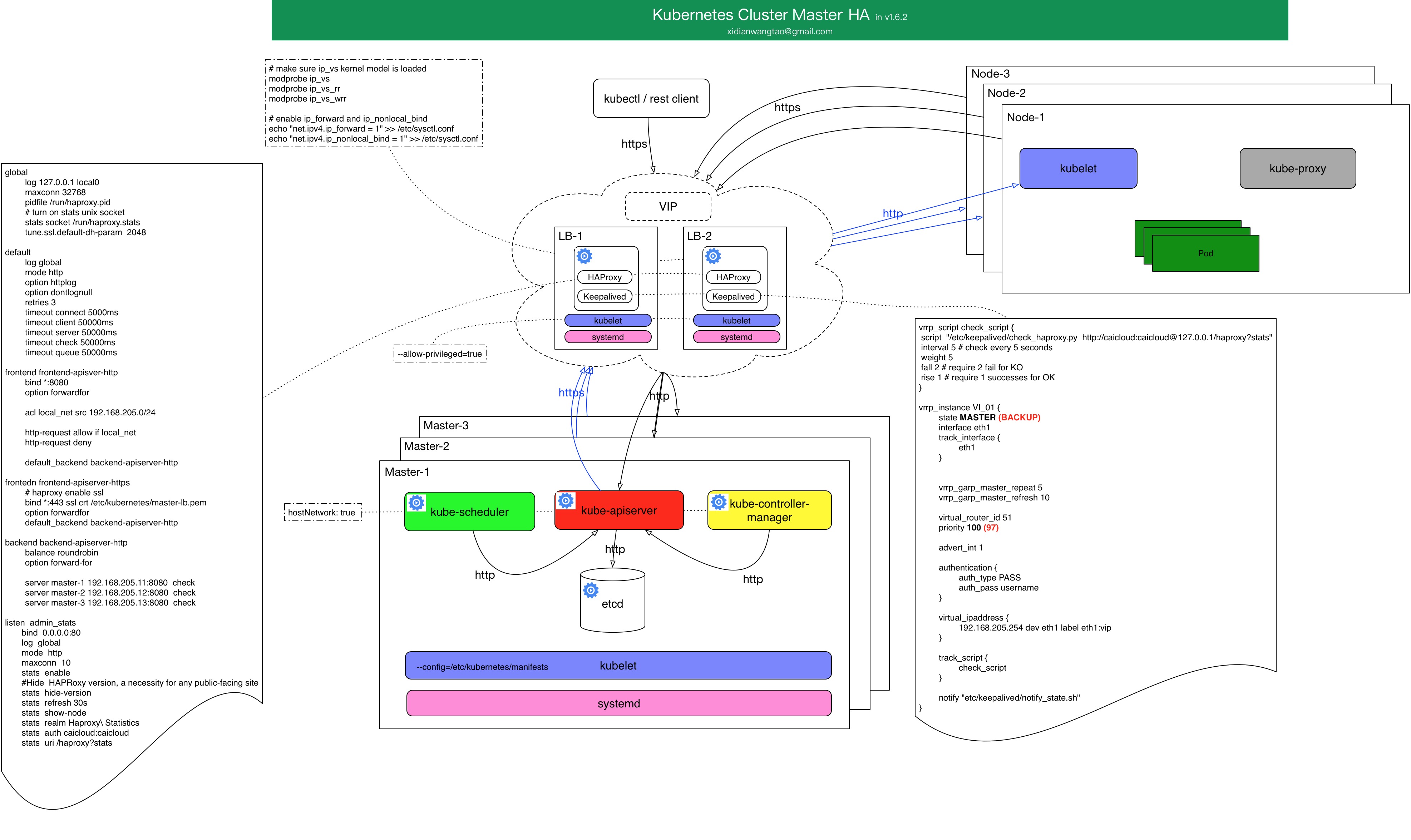
# 软件信息

OS:：CentOS 18.04

Docker：17.03.0-ce

k8s：v1.10

# 高可用架构图



## 高可用最重要的两个组件

etcd：分布式键值存储、k8s集群数据中心

kube-apiserver：集群的唯一入口，各组件通信枢纽。apiserver本身无状态，因此分布式很容易

## 其它核心组件

controller-manager和scheduler也可以部署多个，但只有一个处于活跃状态，以保证数据一致性。因为它们会改变集群状态。

集群各组件都是松耦合的，如何高可用就有很多种方式了。

kube-apiserver有多个，那么apiserver客户端应该连接哪个了，因此就在apiserver前面加个传统的类似于haproxy+keepalived方案漂个VIP出来，apiserver客户端，比如kubelet、kube-proxy连接此VIP。

# 安装前准备

## k8s各节点SSH免密登录

master01

#ssh-keygen –t rsa

#ssh-copy-id –i root@master02

#ssh-copy-id –i root@master03

master02

#ssh-keygen –t rsa

#ssh-copy-id –i root@master01

#ssh-copy-id –i root@master03

master03

#ssh-keygen –t rsa

#ssh-copy-id –i root@master01

#ssh-copy-id –i root@master02

## 时间同步

master01

#yum –y install ntpdate

#ntpdate ntp.pool.org

master02

#yum –y install ntpdate

#ntpdate ntp.pool.org

master03

#yum –y install ntpdate

#ntpdate ntp.pool.org

## 关闭swap

各Node必须关闭swap,否则kubelet启动失败

#swapoff –a

注释掉/etc/fstab里面swap的自动挂载

## 添加hosts解析

各节点主机名和IP加入/etc/hosts解析

192.168.40.23 master01

192.168.40.24 master02

192.168.40.25 master03

## kubeadm创建高可用集群有两种方法

etcd集群由kubeadm配置并运行于pod，启动在Master节点之上。

etcd集群单独部署。

## 拷贝证书到所有master节点对应目录

# scp -r /etc/etcd 192.168.40.23:/etc/

# scp -r /etc/etcd 192.168.40.24:/etc/

# scp -r /etc/etcd 192.168.40.25:/etc/

## 安装Docker

所有节点都需要安装

$uname -r

### 确保yum包最新

使用root 权限登录 Centos。确保 yum 包更新到最新。

$sudo yum update

### 卸载旧版本(如果安装过旧版本的话)

$sudo rpm -qa|grep docker

$sudo yum -y remove docker docker-common docker-selinux docker-engine

### 安装需要的软件包

yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

$sudo yum -y install yum-utils device-mapper-persistent-data lvm2

### 设置docker yum源

$sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

### 查看docker版本

可以查看所有仓库中所有docker版本，并选择特定版本安装

$yum list docker-ce --showduplicates | sort -r

### 开始安装

$sudo yum install docker-ce-17.03.0.ce

$vi /usr/lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd --storage-opt overlay2.override\_kernel\_check=1 --insecure-registry 192.168.80.95:6000

### 启动并加入开机启动

$ sudo systemctl start docker

$ sudo systemctl enable docker

### 验证安装

有client和service两部分表示docker安装启动都成功了

$ docker version

# 安装kubeadm集群

## 安装管理工具

安装kubeadm kubelete kubectl工具

注意：所有节点都需要安装

$ yum -y install kubeadm-1.10.0 kubelet-1.10.0 kubectl-1.10.0

### 调整cgroup driver

$ vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

Environment="KUBELET\_CGROUP\_ARGS=--cgroup-driver=cgroupfs"

### 设置kubelet服务开机自启

systemctl enable kubelet

注意：必须设置Kubelet开机自启动，才能让k8s集群各组件在系统重启后自动运行

## 集群初始化

### 镜像拉取

#### 登录nexus docker镜像仓库

$ docker login 192.168.80.95:6000

#### 自动拉取和tag镜像

三台master都需要拉取

$ vi k8s.gcr.io.sh

#!/bin/bash

set -o errexit

set -o nounset

set -o pipefail

KUBE\_VERSION=v1.10.0

KUBE\_PAUSE\_VERSION=3.1

#ETCD\_VERSION=3.1.12

DNS\_VERSION=1.14.8

FLANNEL\_VERSION=v0.10.0

HEAPSTER\_VERSION=v1.5.3

INFLUXDB\_VERSION=v1.3.3

GRAFANA\_VERSION=v4.4.3

DASHBOARD\_VERSION=v1.8.3

LOCAL\_URL=192.168.80.95:6000/k8s

images=(kube-proxy-amd64:${KUBE\_VERSION}

kube-scheduler-amd64:${KUBE\_VERSION}

kube-controller-manager-amd64:${KUBE\_VERSION}

kube-apiserver-amd64:${KUBE\_VERSION}

pause-amd64:${KUBE\_PAUSE\_VERSION}

#etcd-amd64:${ETCD\_VERSION}

flannel:${FLANNEL\_VERSION}-amd64

k8s-dns-sidecar-amd64:${DNS\_VERSION}

k8s-dns-kube-dns-amd64:${DNS\_VERSION}

k8s-dns-dnsmasq-nanny-amd64:${DNS\_VERSION}

heapster-amd64:${HEAPSTER\_VERSION}

kubernetes-dashboard-amd64:${DASHBOARD\_VERSION}

heapster-influxdb-amd64:${INFLUXDB\_VERSION}

heapster-grafana-amd64:${GRAFANA\_VERSION}

)

for imageName in ${images[@]} ; do

docker pull $LOCAL\_URL/$imageName

docker tag $LOCAL\_URL/$imageName k8s.gcr.io/$imageName

docker rmi $LOCAL\_URL/$imageName

done

### master执行集群初始化

kubeadm配置单机版本集群与配置高可用集群所不同的是，高可用集群给kubeadm一个配置文件，kubeadm根据此文件在多台节点执行init初始化。

#### 编写kubeadm配置文件

/kubeadm-config#vi kubeadm-config.yaml

apiVersion: kubeadm.k8s.io/v1alpha1

kind: MasterConfiguration

kubernetesVersion: v1.10.0

networking:

podSubnet: 10.244.0.0/16

apiServerCertSANs:

- master01.local.com

- master02.local.com

- master03.local.com

- 192.168.40.24

- 192.168.40.23

- 192.168.40.22

- 192.168.40.28

- 127.0.0.1

etcd:

endpoints:

- https://192.168.40.3:2379

- https://192.168.40.4:2379

- https://192.168.40.5:2379

caFile: /etc/etcd/ssl/ca.pem

certFile: /etc/etcd/ssl/etcd.pem

keyFile: /etc/etcd/ssl/etcd-key.pem

配置释析：

版本v1.12的api版本已提升为kubeadm.k8s.io/v1alpha3，kind已变成ClusterConfiguration，v1.10和v1.11的api版本是kubeadm.k8s.io/v1alpha1，kind是MasterConfiguration

podSubnet：自定义pod网段。

apiServerCertSANs：填写所有kube-apiserver节点的hostname、IP、VIP

etcd：external表示使用外部etcd集群，后面写上etcd节点IP、证书位置。

如果etcd集群由kubeadm配置，则应该写local，加上自定义的启动参数。

token：可以不指定，使用指令 kubeadm token generate 生成。

#### master01执行init

#确保swap已关闭

/kubeadm-config# kubeadm init --config kubeadm-config.yaml

最后输出如下信息：

……

Your Kubernetes master 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 machines by running the following on each node

as root:

#记录下面这句，在其它Node加入时用到。

kubeadm join 192.168.40.24:6443 --token w79yp6.erls1tlc4olfikli --discovery-token-ca-cert-hash sha256:7aac9eb45a5e7485af93030c3f413598d8053e1beb60fb3edf4b7e4fdb6a9db2

##### 根据初始化完成后的提示执行

# 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

##### 查看Master01核心组件运行为Pod

# kubectl get pod -n kube-system -o wide

##### 拷贝生成的pki目录到各master节点

# scp -r /etc/kubernetes/pki root@192.168.40.23:/etc/kubernetes/

# scp -r /etc/kubernetes/pki root@192.168.40.22:/etc/kubernetes/

##### 拷贝kubeadm的配置文件到master02和master03

master01部署完成了，接下来的maser02和master03，无论后面有多少个Master都使用相同的kubeadm-config.yaml进行初始化

/# scp kubeadm-config.yaml root@192.168.40.23:~/

/# scp kubeadm-config.yaml root@192.168.40.22:~/

#### master02执行kubeadm init

# kubeadm init --config kubeadm-config.yaml

……

Your Kubernetes master 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 machines by running the following on each node

as root:

#记录下面这句，在其它Node加入时用到。

kubeadm join 192.168.40.23:6443 --token w79yp6.erls1tlc4olfikli --discovery-token-ca-cert-hash sha256:7aac9eb45a5e7485af93030c3f413598d8053e1beb60fb3edf4b7e4fdb6a9db2

##### 查看Master01核心组件运行为Pod

# kubectl get pod -n kube-system -o wide

#### master03执行kubeadm init

# kubeadm init --config kubeadm-config.yaml

……

Your Kubernetes master 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 machines by running the following on each node

as root:

#记录下面这句，在其它Node加入时用到。

kubeadm join 192.168.40.22:6443 --token w79yp6.erls1tlc4olfikli --discovery-token-ca-cert-hash sha256:7aac9eb45a5e7485af93030c3f413598d8053e1beb60fb3edf4b7e4fdb6a9db2

##### 查看Master01核心组件运行为Pod

# kubectl get pod -n kube-system -o wide

### 最后查看Node状况

# kubectl get node

NAME STATUS ROLES AGE VERSION

k8s-master01 NotReady master 31m v1.10.0

k8s-master02 NotReady master 15m v1.10.0

k8s-master03 NotReady master 6m52s v1.10.0

因为还没有部署网络，所有状态都是NotReady

### 查看各组件运行状态

# 核心组件已正常running

# kubectl get pod -n kube-system -o wide

去除所有master上的taint(污点)，让master也可被调度：

# kubectl taint nodes --all node-role.kubernetes.io/master-

node/k8s-master01 untainted

node/k8s-master02 untainted

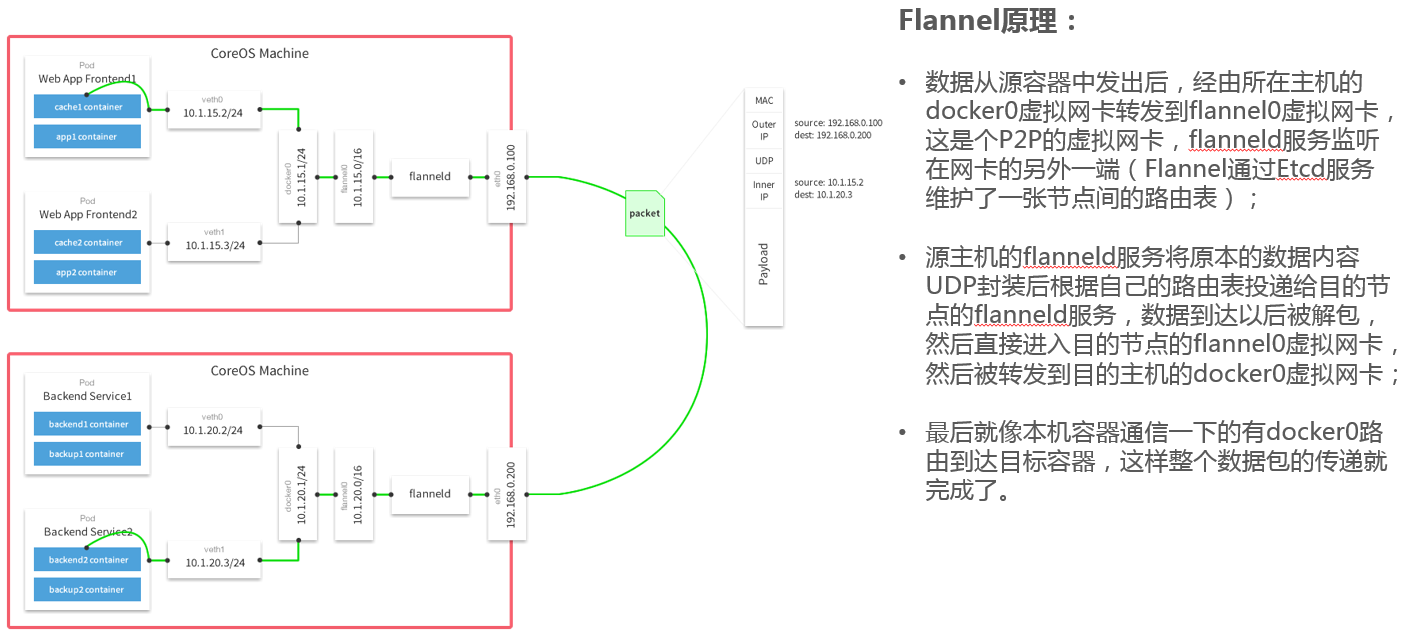
node/k8s-master03 untainted

所有节点是"NotReady"状态，需要安装CNI插件

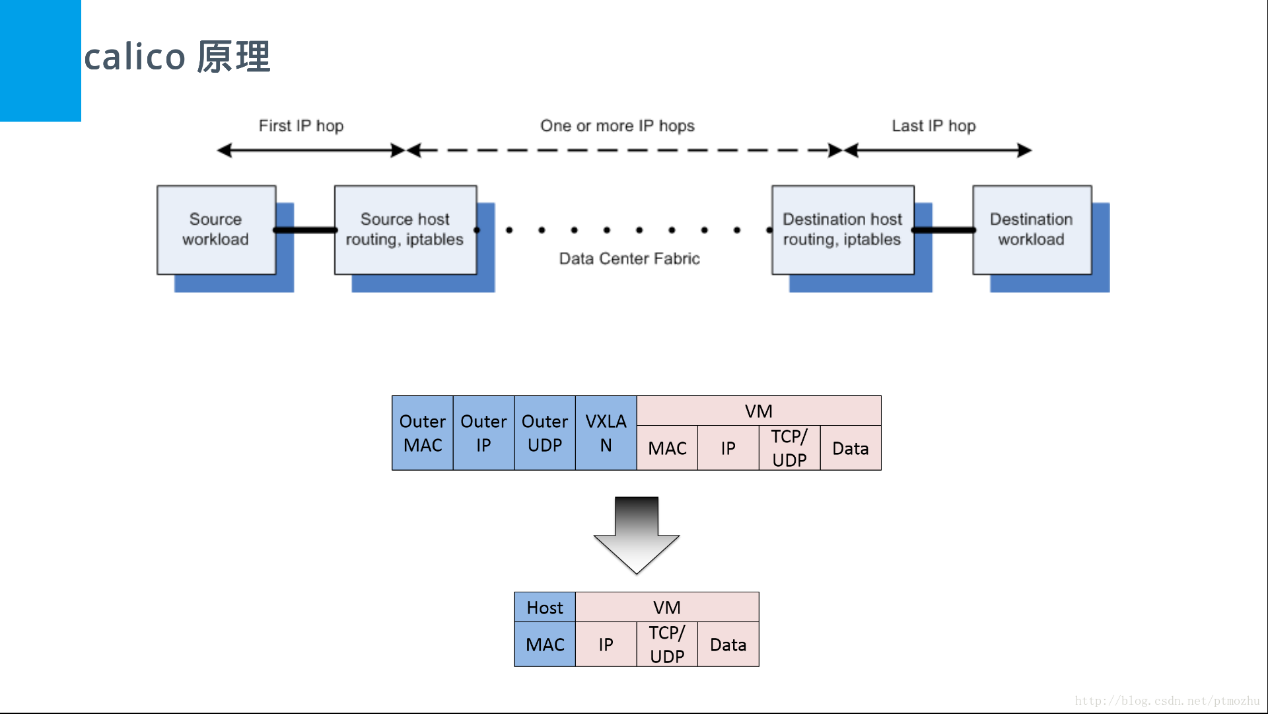
## 安装CNI插件

如下两种插件二选一即可

隧道方案：在IAAS层网络中应用较多，但是不适合大规模集群，因为随着节点的增多，复杂度会增加，出现网络问题不易排查。代表有flannel。



路由方案：从2层或者3层实现容器隔离和跨主机通信的，代表有calico



### 安装Calico网络插件（可选）

# kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml

configmap/calico-config created

daemonset.extensions/calico-etcd created

service/calico-etcd created

daemonset.extensions/calico-node created

deployment.extensions/calico-kube-controllers created

clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created

clusterrole.rbac.authorization.k8s.io/calico-cni-plugin created

serviceaccount/calico-cni-plugin created

clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created

serviceaccount/calico-kube-controllers created

### 安装flannel网络插件（可选）

kubectl apply -f https://github.com/coreos/flannel/blob/master/Documentation/kube-flannel.yml

mkdir -pv /etc/cni/net.d

cat << EOF > /etc/cni/net.d/10-flannel.conf

{

"name": "cbr0",

"type": "flannel",

"delegate": {

"isDefaultGateway": true

}

}

EOF

mkdir /usr/share/oci-umount/pci-umount.d -pv

mkdir /run/flannel

cat <<EOF> /run/flannel/subnet.env

FLANNEL\_NETWORK=10.244.0.0/16

FLANNEL\_SUBNET=10.244.2.0/24

FLANNEL\_MTU=1450

FLANNEL\_IPMASQ=true

EOF

rm -rf /var/lib/cni/flannel/\*

rm -rf /var/lib/cni/networks/cbr0/\*

ip link delete cni0

ip link delete flannel.1

rm -rf /var/lib/cni/networks/cni0/\*

## 再次查看Node状态

# kubectl get node

NAME STATUS ROLES AGE VERSION

k8s-master01 Ready master 39m v1.10.0

k8s-master02 Ready master 24m v1.10.0

k8s-master03 Ready master 15m v1.10.0

各master上所有组件已正常：

# kubectl get pod -n kube-system -o wide

# 部署Node

在所有node节点上使用kubeadm join进行加入kubernetes集群操作，这里统一使用k8s-master01的apiserver地址来加入集群

## node01加入集群

# kubeadm join 192.168.40.23:6443 --token w79yp6.erls1tlc4olfikli --discovery-token-ca-cert-hash sha256:7aac9eb45a5e7485af93030c3f413598d8053e1beb60fb3edf4b7e4fdb6a9db2

输出如下信息：

[preflight] running pre-flight checks

[WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be used, because the following required kernel modules are not loaded: [ip\_vs\_rr ip\_vs\_wrr ip\_vs\_sh] or no builtin kernel ipvs support: map[ip\_vs\_rr:{} ip\_vs\_wrr:{} ip\_vs\_sh:{} nf\_conntrack\_ipv4:{} ip\_vs:{}]

you can solve this problem with following methods:

1. Run 'modprobe -- ' to load missing kernel modules;

2. Provide the missing builtin kernel ipvs support

[WARNING Service-Kubelet]: kubelet service is not enabled, please run 'systemctl enable kubelet.service'

[discovery] Trying to connect to API Server "10.3.1.20:6443"

[discovery] Created cluster-info discovery client, requesting info from "https://10.3.1.20:6443"

[discovery] Requesting info from "https://10.3.1.20:6443" again to validate TLS against the pinned public key

[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinned roots, will use API Server "10.3.1.20:6443"

[discovery] Successfully established connection with API Server "10.3.1.20:6443"

[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12" ConfigMap in the kube-system namespace

[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[preflight] Activating the kubelet service

[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...

[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "k8s-node01" as an annotation

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 master to see this node join the cluster.

## 查看Node运行的组件

# kubectl get pod -n kube-system -o wide |grep node01

calico-node-hsg4w 2/2 Running 2 47m 10.3.1.63 k8s-node01 <none>

kube-proxy-xn795 1/1 Running 0 47m 10.3.1.63 k8s-node01 <none>

## 查看现在的Node状态

#现在有四个Node，全部Ready

# kubectl get node

NAME STATUS ROLES AGE VERSION

k8s-master01 Ready master 132m v1.10.0

k8s-master02 Ready master 117m v1.10.0

k8s-master03 Ready master 108m v1.10.0

k8s-node01 Ready <none> 52m v1.10.0

# 部署keepalived

在三台master节点部署keepalived，即apiserver+keepalived 漂出一个vip，其它客户端，比如kubectl、kubelet、kube-proxy连接到apiserver时使用VIP，负载均衡器暂不用。

## 安装keepalived

### 首先安装依赖包

# yum install -y libnl\*

# yum install -y libnfnetlink-devel zlib zlib-devel gcc gcc-c++ openssl openssl-devel

2、下载并解压Keepalived

# wget http://www.keepalived.org/software/keepalived-1.3.5.tar.gz

# tar xvf keepalived-1.3.5.tar.gz

### 编译安装

# cd keepalived-1.3.5

[root@bogon keepalived-1.3.5]# ./configure --prefix=/usr/local/keepalived

# make&&make install

# cp /usr/local/keepalived/sbin/keepalived /usr/sbin/

# cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/

# cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/

# cp keepalived-1.3.6/keepalived/etc/init.d/keepalived /etc/init.d/

# chmod 755 /etc/init.d/keepalived

### 设置systemd服务器控制文件

[Unit]

Description=LVS and VRRP High Availability Monitor

After=syslog.target network.target remote-fs.target nss-lookup.target network-online.target

[Service]

Type=forking

PIDFile=/var/run/keepalived.pid

KillMode=process

EnvironmentFile=-/etc/keepalived/keepalived.conf

ExecStart=/usr/local/keepalived/sbin/keepalived -D

ExecReload=/bin/kill -HUP $MAINPID

ExecStop=/bin/kill -QUIT $MAINPID

PrivateTmp=true

[Install]

WantedBy=multi-user.target

### 编写keepalived配置文件

把此配置文件复制到其余的master，修改下优先级，设置为slave，最后漂出一个VIP 192.168.40.22，在前面创建证书时已包含该IP。

#MASTER节点

cat /etc/keepalived/keepalived.conf

! Configuration File for keepalived

global\_defs {

notification\_email {

root@loalhost

}

notification\_email\_from Alexandre.Cassen@firewall.loc

smtp\_server 127.0.0.1

smtp\_connect\_timeout 30

router\_id KEP

}

vrrp\_script chk\_k8s {

script "killall -0 kube-apiserver"

interval 1

weight -5

}

vrrp\_instance VI\_1 {

state MASTER

interface eth0

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.40.22

}

track\_script {

chk\_k8s

}

notify\_master "/data/service/keepalived/notify.sh master"

notify\_backup "/data/service/keepalived/notify.sh backup"

notify\_fault "/data/service/keepalived/notify.sh fault"

}

### keepalived服务控制

systemctl enable keepalived.service #设置开机自动启动

systemctl disable keepalived.service #取消开机自动启动

systemctl start keepalived.service #启动服务

systemctl restart keepalived.service #重启服务

systemctl stop keepalived.service #停止服务

systemctl status keepalived.service #查看服务状态

### 修改客户端配置

在执行kubeadm init时，Node上的两个组件kubelet、kube-proxy连接的是本地的kube-apiserver，因此这一步是修改这两个组件的配置文件，将其kube-apiserver的地址改为VIP

### kubelet配置修改

配置文件中"server"字段指向apiserver的URL

修改每个节点上的kubelet服务

$ sed -i "s/10.3.1.63:6443/10.3.1.29:6443/g" /etc/kubernetes/bootstrap-kubelet.conf

$ sed -i "s/10.3.1.63.6443/10.3.1.29:6443/g" /etc/kubernetes/kubelet.conf

重启kubelet

$ systemctl restart docker kubelet

kube-proxy配置修改

kube-proxy的配置文件保存于kube-system空间中的ConfigMap "kube-proxy"。

Master节点上修改

kubectl edit configmap kube-proxy -n kube-system

server: https://10.3.1.29:6443

修改完后删除并自动重启kube-proxy

$ kubectl delete pod -n kube-system kube-proxy-XXXXX

kubeadm HA功能目前仍处于v1alpha状态，慎用于生产环境，详细部署文档还可以参考官方文档。