**使用kubeadm部署Kubernetes集群**

注意：请不要把目光仅仅放在部署上，要慢慢的了解其本质。

Kubernetes v1.13版本发布后，kubeadm才正式进入GA，可以生产使用。目前Kubernetes的对应镜像仓库，在国内阿里云也有了镜像站点，使用kubeadm部署Kubernetes集群变得简单并且容易了很多，本文使用kubeadm带领大家快速部署Kubernetes v1.13.3版本。

**实验环境准备**

在本书的实验环境的基础上，我们如下来分配角色：

|  |  |  |  |
| --- | --- | --- | --- |
| 主机名 | IP地址（NAT） | 最低配置 | 描述 |
| linux-node1.linuxhot.com | eth0:192.168.56.11 | 1CPU/1G内存 | Kubernets Master/Etcd节点 |
| linux-node2.linuxhot.com | eth0:192.168.56.12 | 1CPU/1G内存 | Kubernets Node节点 |
| linux-node3.linuxhot.com | eth0:192.168.56.13 | 1CPU/1G内存 | Kubernets Node节点 |
| Service网段 | 10.1.0.0/16 | | |
| Pod网段 | 10.1.0.0/16 | | |
| 备注 | 1.        如果有条件可以部署多个Kubernets node，实验效果更佳。  2.        所有虚拟机操作系统使用CentOS 7.6。 | | |

**部署Docker和kubeadm**

    首先需要在所有Kubernetes集群的节点中安装Docker和kubeadm。

**1.设置使用国内Yum源**

[root@linux-node1 ~]# cd /etc/yum.repos.d/

[root@linux-node1 yum.repos.d]# wget \

 https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

**2.安装指定的Docker版本**

由于kubeadm对Docker的版本是有要求的，需要安装与kubeadm匹配的版本。

[root@linux-node1 ~]# yum list docker-ce.x86\_64 --showduplicates | sort -r

安装Docker18.06版本

[root@linux-node1 ~]# yum -y install docker-ce-18.06.1.ce-3.el7

**3.启动后台进程**

[root@linux-node1 ~]# systemctl enable docker&& systemctl start docker

查看Docker版本

[root@linux-node1 ~]# docker --version

Docker version 18.06.1-ce, build e68fc7a

**4.设置kubernetes YUM仓库**

[root@linux-node1 ~]# vim /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

**5.安装软件包**

由于版本更新频繁，请指定对应的版本号，本文采用1.13.3版本，其它版本未经测试。

[root@linux-node1 ~]# yum install -y kubelet-1.13.3 kubeadm-1.13.3 kubectl-1.13.3 ipvsadm

**6.配置kubelet**

默认情况下，Kubelet不允许所在的主机存在交换分区，后期规划的时候，可以考虑在系统安装的时候不创建交换分区，针对已经存在交换分区的可以设置忽略禁止使用Swap的限制，不然无法启动Kubelet。

[root@linux-node1 ~]# vim /etc/sysconfig/kubelet

KUBELET\_EXTRA\_ARGS="--fail-swap-on=false"

**7.设置内核参数**

[root@linux-node1 ~]# cat <<EOF >  /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

EOF

使配置生效

[root@linux-node1 ~]# sysctl --system

**8.启动kubelet并设置开机启动**

    注意，此时kubelet是无法正常启动的，可以查看/var/log/messages有报错信息，等待执行初始化之后即可正常，为正常现象。

[root@linux-node1 ~]# systemctl enable kubelet&& systemctl start kubelet

*以上步骤请在Kubernetes的所有节点上执行，本实验环境是需要在linux-node1、linux-node2、linux-node3这三台机器上均安装Docker和kubeadm*

**初始化集群部署Master**

   在所有节点上安装完毕后，在linux-node1这台Master节点上进行集群的初始化工作。

**1.执行初始化操作**

[root@linux-node1 ~]# kubeadm init \

  --apiserver-advertise-address=192.168.56.11 \

  --image-repository registry.aliyuncs.com/google\_containers \

  --kubernetes-version v1.13.3 \

   --service-cidr=10.1.0.0/16\

  --pod-network-cidr=10.2.0.0/16

[init] Using Kubernetes version: v1.13.3

[preflight] Running pre-flight checks

error execution phase preflight: [preflight] Somefatal errors occurred:

      [ERRORNumCPU]: the number of available CPUs 1 is less than the required 2

      [ERRORSwap]: running with swap on is not supported. Please disable swap

[preflight] If you know what you are doing, you canmake a check non-fatal with `--ignore-preflight-errors=...`

先忽略报错，我们来看一下，初始化选项的意义：

* --apiserver-advertise-address：指定用 Master 的哪个IP地址与 Cluster的其他节点通信。
* --service-cidr：指定Service网络的范围，即负载均衡VIP使用的IP地址段。
* --pod-network-cidr：指定Pod网络的范围，即Pod的IP地址段。
* --image-repository：Kubenetes默认Registries地址是k8s.gcr.io，在国内并不能访问 gcr.io，在1.13版本中我们可以增加-image-repository参数，默认值是k8s.gcr.io，将其指定为阿里云镜像地址：registry.aliyuncs.com/google\_containers。
* --kubernetes-version=v1.13.3：指定要安装的版本号。
* --ignore-preflight-errors=：忽略运行时的错误，例如上面目前存在[ERROR NumCPU]和[ERROR Swap]，忽略这两个报错就是增加--ignore-preflight-errors=NumCPU 和--ignore-preflight-errors=Swap的配置即可。

再次执行初始化操作：

[root@linux-node1 ~]# kubeadm init \

  --apiserver-advertise-address=192.168.56.11 \

  --image-repository registry.aliyuncs.com/google\_containers \

  --kubernetes-version v1.13.3 \

  --service-cidr=10.1.0.0/16 \

  --pod-network-cidr=10.2.0.0/16 \

  --service-dns-domain=cluster.local \

  --ignore-preflight-errors=Swap \

  --ignore-preflight-errors=NumCPU

[init] Using Kubernetes version: v1.13.3

[preflight] Running pre-flight checks

       [WARNINGNumCPU]: the number of available CPUs 1 is less than the required 2

       [WARNINGSwap]: running with swap on is not supported. Please disable swap

[preflight] Pulling images required for setting upa Kubernetes cluster

[preflight] This might take a minute or two, dependingon the speed of your internet connection

[preflight] You can also perform this action inbeforehand using 'kubeadm config images pull'

执行完毕后，会在当前输出下停留，等待下载Kubernetes组件的Docker镜像。根据你的网络情况，可以持续1-5分钟，你也可以使用docker images查看下载的镜像。镜像下载完毕之后，就会进行初始操作：

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

[kubelet-start] Writing kubelet configuration tofile "/var/lib/kubelet/config.yaml"

[kubelet-start] Activating the kubelet service

[certs] Using certificateDir folder"/etc/kubernetes/pki"

[certs] Generating "ca" certificate andkey

[certs] Generating "apiserver"certificate and key

…

这里省略了所有输出，初始化操作主要经历了下面15个步骤，每个阶段均输出均使用[步骤名称]作为开头：

kubeadm init到底干了什么？

1. [init]：指定版本进行初始化操作
2. [preflight] ：初始化前的检查和下载所需要的Docker镜像文件
3. [kubelet-start] ：生成kubelet的配置文件”/var/lib/kubelet/config.yaml”，没有这个文件kubelet无法启动，所以初始化之前的kubelet实际上启动失败。
4. [certificates]：生成Kubernetes使用的证书，存放在/etc/kubernetes/pki目录中。
5. [kubeconfig] ：生成 KubeConfig 文件，存放在/etc/kubernetes目录中，组件之间通信需要使用对应文件。
6. [control-plane]：使用/etc/kubernetes/manifest目录下的YAML文件，安装Master 组件。
7. [etcd]：使用/etc/kubernetes/manifest/etcd.yaml安装Etcd服务。
8. [wait-control-plane]：等待control-plan部署的Master组件启动。
9. [apiclient]：检查Master组件服务状态。
10. [uploadconfig]：更新配置
11. [kubelet]：使用configMap配置kubelet。
12. [patchnode]：更新CNI信息到Node上，通过注释的方式记录。
13. [mark-control-plane]：为当前节点打标签，打了角色Master，和不可调度标签，这样默认就不会使用Master节点来运行Pod。
14. [bootstrap-token]：生成token记录下来，后边使用kubeadm join往集群中添加节点时会用到
15. [addons]：安装附加组件CoreDNS和kube-proxy

成功执行之后，你会看到下面的输出：

Your Kubernetes master has initializedsuccessfully!

To start using your cluster, you need to run thefollowing 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 runningthe following on each node

as root:

  kubeadmjoin 192.168.56.11:6443 --token 19fhhl.3mzkyk16tcgp6vga--discovery-token-ca-cert-hashsha256:76a88c38b673d3b2ac73e33127a809688cb3e58c533512ac6d92ecb66aa57a45

请根据上面输出的要求配置kubectl命令来访问集群。

**2.为kubectl准备Kubeconfig文件。**

kubectl默认会在执行的用户家目录下面的.kube目录下寻找config文件。这里是将在初始化时[kubeconfig]步骤生成的admin.conf拷贝到.kube/config。

[root@linux-node1 ~]# mkdir -p $HOME/.kube

[root@linux-node1 ~]# cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

[root@linux-node1 ~]# chown $(id -u):$(id -g)$HOME/.kube/config

在该配置文件中，记录了API Server的访问地址，所以后面直接执行kubectl命令就可以正常连接到API Server中。

**3.使用kubectl命令查看组件状态**

[root@linux-node1 ~]#  kubectl get cs

NAME                STATUS    MESSAGE              ERROR

scheduler               Healthy     ok

controller-manager        Healthy      ok

etcd-0                  Healthy     {"health":"true"}

**知识回顾：**为什么上面的输出没有显示API Server组件的状态

因为API Server是Kubernetes集群的入口，所有和Kubernetes集群的交互都必须经过API Server，kubectl命令也是连接到API Server上进心交互，所以如果能够正常使用kubectl执行命令，意味着API Server运行正常。

**4.使用kubectl获取Node信息**

目前只有一个节点，角色是Master，状态是NotReady。

[root@linux-node1 ~]# kubectl get node

NAME                       STATUS     ROLES   AGE   VERSION

linux-node1.linuxhot.com   NotReady  master   14m   v1.13.3

**部署网络插件canal**

   Master节点NotReady的原因就是因为没有使用任何的网络插件，此时Node和Master的连接还不正常。目前最流行的Kubernetes网络插件有Flannel、Calico、Canal，这里选择使用Canal。

  因为基础的Kubernetes集群已经配置完毕，后面的增加组件等操作，几乎都可以使用kubectl和一个YAML配置文件来完成。

**1.部署canal网络插件**

[root@linux-node1 ~]# kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/canal/rbac.yaml

[root@linux-node1 ~]# kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/canal/canal.yaml

**2.查看启动的Pod**

[root@linux-node1 ~]# kubectl get pods --all-namespaces

NAMESPACE    NAME                                              READY   STATUS              RESTARTS   AGE

kube-system  canal-rq5n5                                       0/3     ContainerCreating   0         109s

kube-system  coredns-78d4cf999f-5k4sg                           0/1     Pending             0          31m

kube-system  coredns-78d4cf999f-bnbgf                           0/1     Pending             0          31m

kube-system  etcd-linux-node1.linuxhot.com                     1/1     Running             0          30m

kube-system  kube-apiserver-linux-node1.linuxhot.com            1/1     Running             0          30m

kube-system  kube-controller-manager-linux-node1.linuxhot.com   1/1    Running             0          31m

kube-system  kube-proxy-sddlp                                   1/1     Running             0          31m

kube-system  kube-scheduler-linux-node1.linuxhot.com            1/1     Running             0          30m

可以看到此时CoreDNS处于Pending状态，需要等待网络插件canal的Pod状态变成Running之后CoreDNS也会正常。所有Pod的状态都变成Running之后，这个时候再次获取Node，会发现节点变成了Ready状态。

[root@linux-node1 ~]# kubectl get node

NAME                       STATUS   ROLES   AGE   VERSION

linux-node1.linuxhot.com   Ready   master   29m   v1.13.3

*kubeadm其实使用Kubernetes部署Kubernetes，这样就存在先有鸡还是先有蛋的问题，所以，我们首先手动部署了Docker和kubelet，然后kubeadm调用kubelet以静态Pod的方式部署了Kubernetes集群中的其它组件。静态Pod在后面的章节会讲到。*

**部署Node节点**

   Master节点部署完毕之后，就可以部署Node节点，首先请遵循部署Docker和kubeadm章节为Node节点部署安装好docker、kubeadm和kubelet，此过程这里不再重复列出。

**1.在Master节点输出增加节点的命令**

[root@linux-node1 ~]# kubeadm token create --print-join-command

kubeadm join 192.168.56.11:6443 --tokenisggqa.xjwsm3i6nex91d2x --discovery-token-ca-cert-hashsha256:718827895a9a5e63dfa9ff54e16ad6dc0c493139c9c573b67ad66968036cd569

    输出的“kubeadm join ...”直接在各个已经安装好Docker和kubeadm的节点上执行即可。

**2.在Node节点执行**

  注意如果节点有交换分区，需要增加--ignore-preflight-errors=Swap。

部署linux-node2

[root@linux-node2 ~]# kubeadm join 192.168.56.11:6443 --token isggqa.xjwsm3i6nex91d2x--discovery-token-ca-cert-hash sha256:718827895a9a5e63dfa9ff54e16ad6dc0c493139c9c573b67ad66968036cd569--ignore-preflight-errors=Swap

部署linux-node3

[root@linux-node3 ~]# kubeadm join 192.168.56.11:6443 --token isggqa.xjwsm3i6nex91d2x--discovery-token-ca-cert-hashsha256:718827895a9a5e63dfa9ff54e16ad6dc0c493139c9c573b67ad66968036cd569--ignore-preflight-errors=Swap

这个时候kubernetes会使用DaemonSet在所有节点上都部署canal和kube-proxy。部署完毕之后节点即部署完毕。DaemonSet的内容后面会讲解。

[root@linux-node1 ~]# kubectl get daemonset --all-namespaces

NAMESPACE    NAME         DESIRED   CURRENT  READY   UP-TO-DATE   AVAILABLE  NODE SELECTOR                 AGE

kube-system  canal        2         2         1      2            1           beta.kubernetes.io/os=linux   17m

kube-system  kube-proxy   2         2         2      2            2           <none>                        47m

待所有Pod全部启动完毕之后，节点就恢复Ready状态。

[root@linux-node1 ~]# kubectl get pod --all-namespaces

NAMESPACE    NAME                                              READY   STATUS    RESTARTS  AGE

kube-system  canal-lv92w                                       3/3     Running   0         8m45s

kube-system  canal-rq5n5                                       3/3     Running   0         23m

kube-system  coredns-78d4cf999f-5k4sg                           1/1     Running  0          53m

kube-system  coredns-78d4cf999f-bnbgf                           1/1     Running  0          53m

kube-system  etcd-linux-node1.linuxhot.com                      1/1     Running  0          52m

kube-system  kube-apiserver-linux-node1.linuxhot.com            1/1     Running  0          52m

kube-system  kube-controller-manager-linux-node1.linuxhot.com   1/1    Running   0          52m

kube-system  kube-proxy-sddlp                                   1/1     Running  0          53m

kube-system  kube-proxy-tw96b                                  1/1     Running  0          8m45s

kube-system  kube-scheduler-linux-node1.linuxhot.com            1/1     Running  0          52m

**查看所有节点**

[root@linux-node1 ~]# kubectl get node

NAME                       STATUS   ROLES   AGE     VERSION

linux-node1.linuxhot.com   Ready   master   49m     v1.13.2

linux-node2.linuxhot.com   Ready   <none>   4m48s   v1.13.2

**如何给Node加上Roles标签**

  使用kubectl getnode能够看到linux-node1.linuxhot.com的ROLES是master这个是在进行集群初始化的时候[mark-control-plane]进行标记的。

[mark-control-plane] Marking the nodelinux-node1.linuxhot.com as control-plane by adding the label"node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the nodelinux-node1.linuxhot.com as control-plane by adding the taints[node-role.kubernetes.io/master:NoSchedule]

1.查看节点的标签

[root@linux-node1 ~]# kubectl get nodes --show-labels

NAME                       STATUS   ROLES   AGE     VERSION   LABELS

linux-node1.linuxhot.com   Ready   master   48m     v1.13.3  beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=linux-node1.linuxhot.com,node-role.kubernetes.io/master=

linux-node2.linuxhot.com   Ready   <none>   7m13s   v1.13.3  beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=linux-node2.linuxhot.com

2.增加标签

[root@linux-node1 ~]# kubectl label nodes linux-node2.linuxhot.com node-role.kubernetes.io/node=

node/linux-node2.linuxhot.com labeled

3.查看效果

[root@linux-node1 ~]# kubectl get nodes

NAME                       STATUS   ROLES   AGE     VERSION

linux-node1.linuxhot.com   Ready   master   50m     v1.13.3

linux-node2.linuxhot.com   Ready   node     8m41s   v1.13.3

**测试Kubernetes集群**

   在上面的步骤中，我们创建了一个Kubernetes集群，1个Master和2个Node节点，在生产环境需要考虑Master的高可用，这里先不用考虑，后面会讲到。

**1.创建一个单Pod的Nginx应用**

[root@linux-node1 ~]# kubectl create deployment nginx --image=nginx:alpine

deployment.apps/nginx created

[root@linux-node1 ~]# kubectl get pod

NAME                     READY   STATUS              RESTARTS   AGE

nginx-54458cd494-9j7ql   0/1    ContainerCreating   0          10s

**2.查看Pod详细信息**

待Pod的状态为Running后，可以获取Pod的IP地址，这个IP地址是从Master节点初始化的--pod-network-cidr=10.2.0.0/16地址段中分配的。

[root@linux-node1 ~]# kubectl get pod -o wide

NAME                     READY   STATUS   RESTARTS   AGE   IP        NODE                      NOMINATED NODE   READINESS GATES

nginx-54458cd494-9j7ql   1/1    Running   0          59s  10.2.1.2  linux-node2.linuxhot.com  <none>          <none>

**3.测试Nginx访问**

[root@linux-node1 ~]# curl --head http://10.2.1.2

HTTP/1.1 200 OK

Server: nginx/1.15.8

Date: Sun, 13 Jan 2019 01:16:36 GMT

Content-Type: text/html

Content-Length: 612

Last-Modified: Wed, 26 Dec 2018 23:21:49 GMT

Connection: keep-alive

ETag: "5c240d0d-264"

Accept-Ranges: bytes

**4.测试扩容**

现在将Nginx应用的Pod副本数量拓展到2个节点

[root@linux-node1 ~]# kubectl scale deployment nginx --replicas=2

deployment.extensions/nginx scaled

[root@linux-node1 ~]# kubectl get pod

NAME                     READY   STATUS   RESTARTS   AGE

nginx-54458cd494-9j7ql   1/1    Running   0          2m13s

nginx-54458cd494-vnm4f   1/1    Running   0         5s

**5.为Nginx增加Service**

为Nginx增加Service，会创建一个Cluster IP，从Master初始化的--service-cidr=10.1.0.0/16地址段中进行分配， 并开启NodePort是在Node节点上进行端口映射，进行外部访问。

[root@linux-node1 ~]# kubectl expose deployment nginx --port=80 --type=NodePort

service/nginx exposed

[root@linux-node1 ~]# kubectl get service

NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE

kubernetes  ClusterIP   10.1.0.1       <none>        443/TCP        88m

nginx       NodePort    10.1.147.204   <none>        80:30599/TCP   67m

**6.测试Service的VIP**

[root@linux-node1 ~]# curl --headhttp://10.1.147.204/

HTTP/1.1 200 OK

Server: nginx/1.15.8

Date: Sun, 13 Jan 2019 01:26:21 GMT

Content-Type: text/html

Content-Length: 612

Last-Modified: Wed, 26 Dec 2018 23:21:49 GMT

Connection: keep-alive

ETag: "5c240d0d-264"

Accept-Ranges: bytes

**7.测试NodePort，外部访问。**

在第一篇文章介绍了Kubneretes的三种网络。NodePort就可以使用Node IP + Port来进行访问，而且是所有的Node都可以增加此端口进行访问。



**使用IPVS进行负载均衡**

在Kubernetes集群中Kube-Proxy组件负载均衡的功能，默认使用iptables，生产环境建议使用ipvs进行负载均衡。

**1.在所有节点启用ipvs模块**

[root@linux-node1 ~]# vim /etc/sysconfig/modules/ipvs.modules

#!/bin/bash

modprobe -- ip\_vs

modprobe -- ip\_vs\_rr

modprobe -- ip\_vs\_wrr

modprobe -- ip\_vs\_sh

modprobe -- nf\_conntrack\_ipv4

[root@linux-node1 ~]# chmod +x /etc/sysconfig/modules/ipvs.modules

[root@linux-node1 ~]# source /etc/sysconfig/modules/ipvs.modules

查看模块是否加载正常

[root@linux-node1 ~]# lsmod | grep -e ip\_vs -enf\_conntrack\_ipv4

ip\_vs\_sh               12688  0

ip\_vs\_wrr              12697  0

ip\_vs\_rr               12600  0

ip\_vs                 145497  6 ip\_vs\_rr,ip\_vs\_sh,ip\_vs\_wrr

nf\_conntrack\_ipv4      15053 15

nf\_defrag\_ipv4        12729  1 nf\_conntrack\_ipv4

nf\_conntrack          133095  7ip\_vs,nf\_nat,nf\_nat\_ipv4,xt\_conntrack,nf\_nat\_masquerade\_ipv4,nf\_conntrack\_netlink,nf\_conntrack\_ipv4

libcrc32c              12644  4 xfs,ip\_vs,nf\_nat,nf\_conntrack

**2.修改kube-proxy的配置**

  将mode修改为ipvs，如下所示：

[root@linux-node1~]# kubectl edit cm kube-proxy -n kube-system

…

kind: KubeProxyConfiguration

   metricsBindAddress: 127.0.0.1:10249

    mode: "ipvs"

   nodePortAddresses: null

   oomScoreAdj: -999

 …

对于Kubernetes来说，可以直接将这三个Pod删除之后，会自动重建。

[root@linux-node1 ~]# kubectl get pod -n kube-system | grep kube-proxy

kube-proxy-2wmnr                                 1/1     Running  0          31m

kube-proxy-pzn5h                                   1/1     Running  0          20m

kube-proxy-qhsb8                                   1/1    Running   0          20m

**3.批量删除并重建kube-proxy**

[root@linux-node1 ~]# kubectl get pod -n kube-system | grep kube-proxy | awk '{system("kubectl delete pod"$1" -n kube-system")}'

pod "kube-proxy-2wmnr" deleted

pod "kube-proxy-pzn5h" deleted

pod "kube-proxy-qhsb8" deleted

   由于你已经通过ConfigMap修改了kube-proxy的配置，所以后期增加的Node节点，会直接使用ipvs模式。

**4.测试ipvs**

使用ipvsadm测试，可以查看之前创建的Service已经使用LVS创建了集群。

[root@linux-node1 ~]# ipvsadm -Ln

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

  ->RemoteAddress:Port           ForwardWeight ActiveConn InActConn

TCP 10.1.0.1:443 rr

  ->192.168.56.11:6443           Masq    1     0          0

TCP 10.1.0.10:53 rr

  ->10.2.0.2:53                  Masq    1     0          0

  ->10.2.0.3:53                  Masq    1     0          0

TCP  10.1.147.204:80 rr

  ->10.2.1.2:80                  Masq    1     0          0

  ->10.2.2.2:80                  Masq    1     0          0

TCP  127.0.0.1:30599 rr

  ->10.2.1.2:80                  Masq    1     0          0

  ->10.2.2.2:80                  Masq    1     0          0

TCP  172.17.0.1:30599 rr

  ->10.2.1.2:80                  Masq    1     0          0

  -> 10.2.2.2:80                  Masq    1     0          0

TCP  192.168.56.11:30599 rr

  ->10.2.1.2:80                  Masq    1     0          0

  ->10.2.2.2:80                  Masq    1     0          0

TCP  10.2.0.0:30599 rr

  ->10.2.1.2:80                  Masq    1     0          0

  ->10.2.2.2:80                  Masq    1     0          0

UDP 10.1.0.10:53 rr

  ->10.2.0.2:53                  Masq    1     0          0

  ->10.2.0.3:53                  Masq    1     0          0

这一切看起来似乎不是十分完美，但是现在你已经拥有了一个Kubernetes集群，接下来就可以继续探索Kubernetes的世界了，后面会讲解如何部署生产高可用的K8S集群，而且一定要牢记Kubernetes的架构图。注意kube-scheduler和kube-controller-manager是不能直接访问etcd的。

