单节点安装kubernetes

安装网络插件 Calico

集群环境

IP	Hostname	role	CPU	Memory
172.16.126.137	masterr-001	master	2	8G
172.16.126.138	node-001	node	2	8G

- Kubeadm 最低配置2个cpu，2个内存
- 所有操作全部必须用root用户

环境配置（每台机器）

修改hosts

```
$ cat >>/etc/hosts << EOF
172.16.126.137 master-001
172.16.126.138 worker-001
EOF
```

配置hostname

- master节点

  ```
  $ hostnamectl set-hostname master-001
  ```

- node节点

  ```
  $ hostnamectl set-hostname node-001
  ```

关闭防火墙

```
$ systemctl stop firewalld && systemctl disable firewalld
```

重置iptables

```
$ iptables -F && iptables -X && iptables -F -t nat && iptables -X -t nat
&& iptables -P FORWARD ACCEPT
```

关闭swap

- Kubernetes 建议关闭系统Swap,在 `所有机器` 使用以下指令关闭swap并注释掉 `/etc/fstab` 中swap的行，不想关闭可以不执行，后面会应对的配置选项：

```
$ swapoff -a && sysctl -w vm.swappiness=0
$ sed -ri '/^[^#]*swap/s@^@#@' /etc/fstab
```

关闭seliunx

- 永久关闭

  ```
  $ vim /etc/sysconfig/selinux
  $ SELINUX=enforcing 改为 SELINUX=disabled
  ```

- 临时关闭selinux

  不建议临时关闭，防止机器重启失效

  ```
  $ setenforce 0        #临时关闭selinux
  $ sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
  ```

  查看状态

  ```
  $ sestatus    #查看selinux状态
  ```

关闭 dnsmasq (可选)

- linux 系统开启了 dnsmasq 后(如 GUI 环境)，将系统 DNS Server 设置为 127.0.0.1，这会导致 docker 容器无法解析域名，需要关闭它

  ```
  $ systemctl stop dnsmasq
  $ systemctl disable --now dnsmasq
  ```

安装基础依赖和工具

安装红帽源

EPEL源-是什么?为什么安装?

EPEL (Extra Packages for Enterprise Linux)是基于Fedora的一个项目，为“红帽系”的操作系统提供额外的软件包，适用于RHEL、CentOS和Scientific Linux.

首先需要安装一个叫”epel-**release**”的软件包，这个软件包会自动配置yum的软件仓库。当然你也可以不安装这个包，自己配置软件仓库也是一样的。

```
yum install epel-release
```

安装常用依赖工具

```
yum install -y \
    curl \
    wget \
    git \
    conntrack-tools \
    psmisc \
    nfs-utils \
    jq \
    socat \
    bash-completion \
    ipset \
    ipvsadm \
    conntrack \
    libseccomp \
    net-tools \
    crontabs \
    sysstat \
    unzip \
    iftop \
    nload \
    strace \
    bind-utils \
    tcpdump \
    telnet \
    lsof \
    htop
```

安装ipvs

集群kube-proxy想使用ipvs模式，所以要安装

所有机器开机加载内核模块，并设置开机自动加载

```
$ cat > /etc/sysconfig/modules/ipvs.modules <<EOF
#!/bin/bash
modprobe -- ip_vs
modprobe -- ip_vs_rr
modprobe -- ip_vs_wrr
modprobe -- ip_vs_sh
modprobe -- nf_conntrack_ipv4
EOF
```

然后执行脚本

```
$ chmod 755 /etc/sysconfig/modules/ipvs.modules && bash
/etc/sysconfig/modules/ipvs.modules
```

确认内核模块加载

```
$ lsmod | grep -e ip_vs -e nf_conntrack_ipv4
```

配置内核参数

`所有机器` 需要设定 `/etc/sysctl.d/k8s.conf` 的系统参数，目前对ipv6支持不怎么好，所以里面也关闭ipv6了。

```
$ cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
net.ipv4.neigh.default.gc_stale_time = 120
net.ipv4.conf.all.rp_filter = 0
net.ipv4.conf.default.rp_filter = 0
net.ipv4.conf.default.arp_announce = 2
net.ipv4.conf.lo.arp_announce = 2
net.ipv4.conf.all.arp_announce = 2
net.ipv4.ip_forward = 1
net.ipv4.tcp_max_tw_buckets = 5000
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_max_syn_backlog = 1024
net.ipv4.tcp_synack_retries = 2
# 要求iptables不对bridge的数据进行处理
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-arptables = 1
net.netfilter.nf_conntrack_max = 2310720
fs.inotify.max_user_watches=89100
```

```
fs.may_detach_mounts = 1
fs.file-max = 52706963
fs.nr_open = 52706963
vm.overcommit_memory=1
vm.panic_on_oom=0
EOF

$ sysctl --system
```

最后重启机器

```
reboot
```

安装docker（每台机器）

安装必要的一些系统工具

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```

添加软件阿里源信息

```
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

更新缓存

```
yum makecache fast
```

安装docker

```
 yum -y install docker-ce
```

设置daemon

因为kubelet的启动环境变量要与docker的cgroup-driver驱动相同,以下是官方推荐处理方式（现在新版二进制kubelet就是 `cgroup` 了）

由于国内拉取镜像较慢，配置文件最后追加了阿里云镜像加速配置。

```
mkdir -p /etc/docker &&
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ],
  "registry-mirrors": ["http://f1361db2.m.daocloud.io"]
}
EOF
```

设置docker 开机启动

```
systemctl restart docker && systemctl enable docker && systemctl status docker
```

Kubeadm部署

安装kubeadm源

默认源在国外会无法安装，我们使用国内的镜像源， `所有机器` 都要操作

```
cat <<EOF >/etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
EOF
```

master部分

k8s的node就是kubelet+cri(一般是docker)，kubectl是一个agent读取kubeconfig去访问kube-apiserver来操作集群，kubeadm是部署，所以master节点需要安装三个，node一般不需要kubectl
安装相关软件

```
yum install -y \
    kubeadm-1.16.3 \
    kubectl-1.16.3 \
    kubelet-1.16.3 \
    --disableexcludes=kubernetes && \
    systemctl enable kubelet
```

node

```
yum install -y \
    kubeadm-1.16.3 \
    kubelet-1.16.3 \
    --disableexcludes=kubernetes && \
    systemctl enable kubelet
```

修改kubeadm配置

安装 kubernetes 主要是安装它的各个镜像，而 kubeadm 已经为我们集成好了运行 kubernetes 所需的基本镜像。但由于国内的网络原因，在搭建环境时，无法拉取到这些镜像。此时我们只需要修改为阿里云提供的镜像服务即可解决该问题。

- 导出修改配置

```
kubeadm config print init-defaults > initconfig.yaml
```

- 修改成下面这样

  主要修改地方

  1. 修改为主节点 IP advertiseAddress: 192.168.141.130
  2. 国内不能访问 Google，修改为阿里云源 imageRepository: registry.aliyuncs.com/google_containers
  3. 配置成 Calico 的默认网段 podSubnet: 10.244.0.0/16
  4. 开启 IPVS 模式

  apiVersion: kubeproxy.config.k8s.io/v1alpha1
  kind: KubeProxyConfiguration
  featureGates:
   SupportIPVSProxyMode: true

mode: ipvs

```
apiVersion: kubeadm.k8s.io/v1beta2
bootstrapTokens:
- groups:
  - system:bootstrappers:kubeadm:default-node-token
  token: abcdef.0123456789abcdef
  ttl: 24h0m0s
  usages:
  - signing
  - authentication
kind: InitConfiguration
localAPIEndpoint:
# 修改为主节点 IP
  advertiseAddress: 172.16.126.132
  bindPort: 6443
nodeRegistration:
  criSocket: /var/run/dockershim.sock
  name: master-001
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
---
apiServer:
  timeoutForControlPlane: 4m0s
apiVersion: kubeadm.k8s.io/v1beta2
certificatesDir: /etc/kubernetes/pki
clusterName: kubernetes
# controlPlaneEndpoint: "172.16.120.200:6444" # 单个master的话写master
的ip或者不写
controllerManager: {}
dns:
  type: CoreDNS
etcd:
  local:
    dataDir: /var/lib/etcd
    #国内不能访问 Google，修改为阿里云源
imageRepository: registry.aliyuncs.com/google_containers
kind: ClusterConfiguration
kubernetesVersion: v1.16.0
networking:
  dnsDomain: cluster.local
  # 配置成 Calico 的默认网段
    podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
```

```
scheduler: {}
---
# 开启 IPVS 模式
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
featureGates:
  SupportIPVSProxyMode: true
mode: ipvs
```

检查文件是否错误

- 检查文件是否错误，忽略 `warning` ，错误的话会抛出error，没错则会输出到包含字符串 `kubeadm join xxx` 啥的

```
kubeadm init --config initconfig.yaml --dry-run
```

- 检查镜像是否正确

```
kubeadm config images list --config initconfig.yaml
```

- 预先拉取镜像

```
kubeadm config images pull --config initconfig.yaml # 下面是输出
[config/images] Pulled gcr.azk8s.cn/google_containers/kube-
apiserver:v1.16.3
[config/images] Pulled gcr.azk8s.cn/google_containers/kube-
controller-manager:v1.16.3
[config/images] Pulled gcr.azk8s.cn/google_containers/kube-
scheduler:v1.16.3
[config/images] Pulled gcr.azk8s.cn/google_containers/kube-
proxy:v1.16.3
[config/images] Pulled gcr.azk8s.cn/google_containers/pause:3.1
[config/images] Pulled quay.azk8s.cn/coreos/etcd:v3.3.17
[config/images] Pulled coredns/coredns:1.6.3
```

kubeadm init

下面init只在第一个 `master` 上面操作

注意：--experimental-upload-certs 参数的意思为将相关的证书直接上传到etcd中保存，这样省去我们手动分发证书的过程
注意在v1.15+版本中，已经变成正式参数，不再是实验性质，之前的版本请使用 --experimental-upload-certs

```
kubeadm init --config initconfig.yaml --upload-certs
```

> - 如果超时了看看是不是kubelet没起来，调试见 **https://github.com/zhangguanzhang/Kubernetes-ansible/wiki/systemctl-running-debug**
> - 注意：如果安装 kubernetes 版本和下载的镜像版本不统一则会出现 `timed out waiting for the condition` 错误。中途失败或是想修改配置可以使用 `kubeadm reset` 命令重置配置，再做初始化操作即可。

记住init后打印的token，复制kubectl的kubeconfig，kubectl的kubeconfig路径默认是 `~/.kube/config`

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

init的yaml信息实际上会存在集群的configmap里，我们可以随时查看，该yaml在其他node和master join的时候会使用到

```
kubectl -n kube-system get cm kubeadm-config -o yaml
```

验证是否成功

```
kubectl get node

# 能够打印出节点信息即表示成功
NAME                STATUS     ROLES    AGE     VERSION
kubernetes-master   NotReady   master   8m40s   v1.14.1
```

至此主节点配置完成

将 node加入到集群

```
kubeadm join 192.168.141.130:6443 --token abcdef.0123456789abcdef --
discovery-token-ca-cert-hash
sha256:cab7c86212535adde6b8d1c7415e81847715cfc8629bb1d270b601744d662515
```

> 安装成功将看到如下信息
> [preflight] Running pre-flight checks
> [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at **https://kubernetes.io/docs/setup/cri/**
> [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-1.14" 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] Activating the kubelet service
> [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.

token忘记的话可以 `kubeadm token list` 查看，可以通过 `kubeadm token create` 创建
sha256的值可以通过下列命令获取

验证是否成功

回到 master 服务器

```
kubectl get nodes

# 可以看到 slave 成功加入 master
NAME                STATUS     ROLES    AGE   VERSION
kubernetes-master   NotReady   master   9h    v1.14.1
kubernetes-slave1   NotReady   <none>   22s   v1.14.1
```

> 如果 node 节点加入 master 时配置有问题可以在 slave 节点上使用 kubeadm reset 重置配置再使用 kubeadm join 命令重新加入即可。希望在 master 节点删除 node ， 可以使用 kubeadm delete nodes 删除。

查看 pod 状态

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

NAME                                       READY   STATUS    RESTARTS
AGE   IP                NODE                NOMINATED NODE   READINESS
GATES
coredns-8686dcc4fd-gwrmb                   0/1     Pending   0
9h    <none>            <none>              <none>           <none>
coredns-8686dcc4fd-j6gfk                   0/1     Pending   0
9h    <none>            <none>              <none>           <none>
etcd-kubernetes-master                     1/1     Running   1
9h    192.168.141.130   kubernetes-master   <none>           <none>
kube-apiserver-kubernetes-master           1/1     Running   1
9h    192.168.141.130   kubernetes-master   <none>           <none>
kube-controller-manager-kubernetes-master  1/1     Running   1
9h    192.168.141.130   kubernetes-master   <none>           <none>
kube-proxy-496dr                           1/1     Running   0
17m   192.168.141.131   kubernetes-slave1   <none>           <none>
kube-proxy-rsnb6                           1/1     Running   1
9h    192.168.141.130   kubernetes-master   <none>           <none>
kube-scheduler-kubernetes-master           1/1     Running   1
9h    192.168.141.130   kubernetes-master   <none>           <none>
```

由此可以看出 coredns 尚未运行，此时我们还需要安装网络插件。

配置网络

什么是 CNI

CNI(Container Network Interface) 是一个标准的，通用的接口。在容器平台，Docker，Kubernetes，Mesos 容器网络解决方案 flannel，calico，weave。只要提供一个标准的接口，就能为同样满足该协议的所有容器平台提供网络功能，而 CNI 正是这样的一个标准接口协议。

什么是 Calico

Calico 为容器和虚拟机提供了安全的网络连接解决方案，并经过了大规模生产验证（在公有云和跨数千个集群节点中），可与 Kubernetes，OpenShift，Docker，Mesos，DC / OS 和 OpenStack 集成。

Calico 还提供网络安全规则的动态实施。使用 Calico 的简单策略语言，您可以实现对容器，虚拟机工作负载和裸机主机端点之间通信的细粒度控制。

安装网络插件 Calico

在 Master 节点操作即可

```
kubectl apply -f
https://docs.projectcalico.org/v3.11/manifests/calico.yaml

# 安装时显示如下输出
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
```

```
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcali
co.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers
created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.extensions/calico-node created
serviceaccount/calico-node created
deployment.extensions/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

确认安装是否成功

```
watch kubectl get pods --all-namespaces

# 需要等待所有状态为 Running，注意时间可能较久，3 - 5 分钟的样子
Every 2.0s: kubectl get pods --all-namespaces

kubernetes-master: Fri May 10 18:16:51 2019

NAMESPACE     NAME                                    READY   STATUS
   RESTARTS   AGE
kube-system   calico-kube-controllers-8646dd497f-g2lln    1/1     Running
  0          50m
kube-system   calico-node-8jrtp                       1/1     Running
  0          50m
kube-system   coredns-8686dcc4fd-mhwfn                1/1     Running
  0          51m
kube-system   coredns-8686dcc4fd-xsxwk                1/1     Running
  0          51m
kube-system   etcd-kubernetes-master                  1/1     Running
  0          50m
kube-system   kube-apiserver-kubernetes-master        1/1     Running
  0          51m
kube-system   kube-controller-manager-kubernetes-master   1/1     Running
  0          51m
kube-system   kube-proxy-p8mdw                        1/1     Running
  0          51m
kube-system   kube-scheduler-kubernetes-master        1/1     Running
  0          51m
```

至此基本环境已部署完毕。

作者：张成基 🥳

时间：2020/01/06