【总结】利用kubeadm部署Kubernetes 1.13集群

宋亚伟
2019年4月3日
2019年3月20日

kubeadm是Kubernetes官方提供的用于快速安装Kubernetes集群的工具，伴随Kubernetes每个版本的发布都会同步更新，kubeadm会对集群配置方面的一些实践做调整，通过实验kubeadm可以学习到Kubernetes官方在集群配置上一些新的最佳实践。

最近发布的Kubernetes 1.13中，kubeadm的主要特性已经GA了，但还不包含高可用，不过说明kubeadm可在生产环境中使用的距离越来越近了。

Area	Maturity Level
Command line UX	GA
Implementation	GA
Config file API	beta
CoreDNS	GA

kubeadm alpha subcommands	alpha
High availability	alpha
DynamicKubeletConfig	alpha
Self-hosting	alpha

当然我们线上稳定运行的Kubernetes集群是使用ansible以二进制形式的部署的高可用集群，这里体验Kubernetes 1.13中的kubeadm是为了跟随官方对集群初始化和配置方面的最佳实践，进一步完善我们的ansible部署脚本。

1. 准备工作[master和node节点均执行]

1.1 系统初始化配置

关闭Selinux/firewalld/iptables

Close SELINUX

```
setenforce 0 \
&& sed -i 's/^SELINUX=.*$/SELINUX=disabled/' /etc/selinux/config \
&& getenforce

systemctl stop firewalld \
&& systemctl daemon-reload \
&& systemctl disable firewalld \
&& systemctl daemon-reload \
&& systemctl status firewalld

yum install -y iptables-services \
&& systemctl stop iptables \
&& systemctl disable iptables \
&& systemctl status iptables
```

设置hostname

```
hostnamectl set-hostname master01-77
```

设置repo源仓库地址

USTC CentOS镜像源[推荐]

1、备份

```
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup
mv /etc/yum.repos.d/epel.repo /etc/yum.repos.d/epel.repo-bak
```

2、下载新的CentOS-Base.repo 到/etc/yum.repos.d/

```
CentOS 7
wget -c https://lug.ustc.edu.cn/wiki/_export/code/mirrors/help/centos?codeblock=3 -O /etc/yum.repos.d/CentOS-Base.repo

yum install -y epel-release \
&& sed -e 's!^mirrorlist=!#mirrorlist=!g' \
         -e 's!^#baseurl=!baseurl=!g' \
         -e 's!//download\.fedoraproject\.org/pub!//mirrors.ustc.edu.cn!g' \
         -e 's!http://mirrors\.ustc!https://mirrors.ustc!g' \
         -i /etc/yum.repos.d/epel.repo /etc/yum.repos.d/epel-testing.repo
```

阿里云 CentOS镜像源

```
[ -e "/etc/yum.repos.d/CentOS-Base.repo" ] && mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo-bak

[ -e "/etc/yum.repos.d/epel.repo" ] && mv /etc/yum.repos.d/epel.repo /etc/yum.repos.d/epel.repo-bak

[ -e "/etc/yum.repos.d/epel-testing.repo" ] &&  mv /etc/yum.repos.d/epel-testing.repo /etc/yum.repos.d/epel-testing.repo.backup

wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo \
&& wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
```

3、之后运行yum makecache生成缓存

```
yum makecache
```

如何有效的清理yum缓存

```
yum clean all \
```

```
&& rm -rf /var/cache/yum/* \
&& yum makecache
```

设置时区Shanghai

```
rm -rf /etc/localtime \
&& ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

禁用ssh连接的DNS

```
sed -i 's/\#UseDNS yes/UseDNS no/' /etc/ssh/sshd_config \
&& sed -i 's/\#UseDNS no/UseDNS no/' /etc/ssh/sshd_config \
&& systemctl restart sshd
```

安装基础软件包

```
yum -y update \
&& yum -y groupinstall "Development Tools" \
&& yum -y install wget vim iftop iotop net-tools nmon telnet lsof iptraf nmap httpd
-tools lrzsz mlocate ntp ntpdate strace libpcap nethogs iptraf iftop nmon bridge-ut
ils bind-utils \
&& yum -y install python python-devel \
&& yum -y update
```

Set DNS

```
cat > /etc/resolv.conf << EOF
nameserver 61.139.2.69
nameserver 114.114.114.114
nameserver 8.8.8.8
EOF
```

/etc/security/limits.conf

```
[ -e /etc/security/limits.d/*nproc.conf ] && rename nproc.conf nproc.conf_bk /etc/s
ecurity/limits.d/*nproc.conf \
&& sed -i '/^# End of file/,$d' /etc/security/limits.conf \
&& cat >> /etc/security/limits.conf <<EOF
# End of file
```

```
* soft nproc 10240000
* hard nproc 10240000
* soft nofile 10240000
* hard nofile 10240000
EOF
```

```
sysctl -p
```

/etc/sysctl.conf

```
/etc/sysctl.conf
/etc/sysctl.d/99-sysctl.conf
#/usr/lib/sysctl.d/sysctl.conf


[ ! -e "/etc/sysctl.conf_bk" ] && /bin/mv /etc/sysctl.conf{,_bk} \
&& cat > /etc/sysctl.conf << EOF
fs.file-max=1000000
fs.nr_open=20480000
net.ipv4.tcp_max_tw_buckets = 180000
net.ipv4.tcp_sack = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_rmem = 4096 87380 4194304
net.ipv4.tcp_wmem = 4096 16384 4194304
net.ipv4.tcp_max_syn_backlog = 16384
net.core.netdev_max_backlog = 32768
net.core.somaxconn = 32768
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_fin_timeout = 20
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_syncookies = 1
#net.ipv4.tcp_tw_len = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_mem = 94500000 915000000 927000000
net.ipv4.tcp_max_orphans = 3276800
net.ipv4.ip_local_port_range = 1024 65000
#net.nf_conntrack_max = 6553500
#net.netfilter.nf_conntrack_max = 6553500
#net.netfilter.nf_conntrack_tcp_timeout_close_wait = 60
#net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 120
```

```
#net.netfilter.nf_conntrack_tcp_timeout_time_wait = 120
#net.netfilter.nf_conntrack_tcp_timeout_established = 3600
EOF
```

```
sysctl -p
```

locale

```
sed -i 's@LANG=.*$@LANG="en_US.UTF-8"@g' /etc/locale.conf
```

安装配置ntp

```
yum -y install ntp ntpdate

echo "*/10 * * * * /usr/sbin/ntpdate 202.112.10.36;/usr/sbin/hwclock -w" >> /var/sp
ool/cron/root \
&& ntpdate pool.ntp.org \
&& date
```

iptables -P FORWARD ACCEPT

```
iptables -P FORWARD ACCEPT
```

创建/etc/sysctl.d/k8s.conf文件，添加如下内容：

```
cat > /etc/sysctl.d/k8s.conf <<EOF

net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
vm.swappiness=0

EOF
```

执行命令使修改生效。

```
modprobe br_netfilter \
&& sysctl -p /etc/sysctl.d/k8s.conf
```

关闭交换分区

Kubernetes 1.8开始要求关闭系统的Swap，如果不关闭，默认配置下kubelet将无法启动。

```
swapoff -a
yes | cp /etc/fstab /etc/fstab_bak
cat /etc/fstab_bak |grep -v swap > /etc/fstab
```

执行`sysctl -p /etc/sysctl.d/k8s.conf`使修改生效。

```
echo "vm.swappiness=0"  >> /etc/sysctl.d/k8s.conf \
&& sysctl -p /etc/sysctl.d/k8s.conf
```

使用kubelet的启动参数`--fail-swap-on=false`去掉必须关闭Swap的限制。
修改/etc/sysconfig/kubelet，加入：

```
KUBELET_EXTRA_ARGS=--fail-swap-on=false
```

```
echo "KUBELET_EXTRA_ARGS=--fail-swap-on=false"  > /etc/sysconfig/kubelet
```

安装指定版本Docker[推荐]

安装必要的一些系统工具

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

添加软件源信息

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

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

yum install -y docker-ce-<VERSION STRING>

yum -y install docker-ce-18.06.3.ce-3.el7
```

```
sudo usermod -aG docker pcl
```

配置daemon.json文件

生产环境配置daemon.json

```
mkdir -p /etc/docker/ \
&& cat > /etc/docker/daemon.json << EOF
{
    "registry-mirrors":[
        "https://c6ai9izk.mirror.aliyuncs.com"
    ],
    "insecure-registries":[
        "repo.yooli.com:5858",
        "10.1.10.114:5080",
        "10.20.1.123:5080",
        "182.140.221.86:5080",
        "10.30.8.78:5080",
        "118.123.216.193:15080"
    ],
    "max-concurrent-downloads":3,
    "data-root":"/data/docker",
    "log-driver":"json-file",
    "log-opts":{
        "max-size":"100m",
        "max-file":"1"
    },
    "max-concurrent-uploads":5,
    "storage-driver":"overlay2"
}
EOF
```

启动检查Docker服务

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

运行hello-world验证docker安装

```
docker run hello-world
```

kube-proxy开启ipvs的前置条件

由于ipvs已经加入到了内核的主干，所以为kube-proxy开启ipvs的前提需要加载以下的内核模块：

```
ip_vs
ip_vs_rr
ip_vs_wrr
ip_vs_sh
nf_conntrack_ipv4
```

在所有的Kubernetes节点node1和node2上执行以下脚本:

```
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
```

```
chmod 755 /etc/sysconfig/modules/ipvs.modules \
&& bash /etc/sysconfig/modules/ipvs.modules \
&& lsmod | grep -e ip_vs -e nf_conntrack_ipv4
```

上面脚本创建了的 `/etc/sysconfig/modules/ipvs.modules` 文件，保证在节点重启后能自动加载所需模块。 使用 `lsmod | grep -e ip_vs -e nf_conntrack_ipv4` 命令查看是否已经正确加载所需的内核模块。

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

接下来还需要确保各个节点上已经安装了ipset软件包 `yum install ipset` 。 为了便于查看ipvs的代理规则，最好安装一下管理工具ipvsadm `yum install ipvsadm` 。

```
yum -y install ipset ipvsadm
```

如果以上前提条件如果不满足，则即使kube-proxy的配置开启了ipvs模式，也会退回到iptables

模式。

iptables filter表中FOWARD链的策略

确认一下iptables filter表中FOWARD链的默认策略(pllicy)为ACCEPT

```
iptables -nvL
Chain INPUT (policy ACCEPT 263 packets, 19209 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 DOCKER-ISOLATION-STAGE-1  all  --  *      *       0.0.0.0/0
0.0.0.0/0
    0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
       ctstate RELATED,ESTABLISHED
    0     0 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0
```

> Docker从1.13版本开始调整了默认的防火墙规则，禁用了iptables filter表中FOWARD链，这样会引起Kubernetes集群中跨Node的Pod无法通信。但这里通过安装docker 1806，发现默认策略又改回了ACCEPT，这个不知道是从哪个版本改回的，因为我们线上版本使用的1706还是需要手动调整这个策略的。

2. 使用kubeadm部署Kubernetes[master节点]

2.1 安装kubeadm和kubelet

配置 kubernetes.repo

```
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=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.al
iyun.com/kubernetes/yum/doc/rpm-package-key.gpg
```

```
EOF
```

安装kubeadm和kubelet

```
yum makecache fast \
&& yum install -y kubelet kubeadm kubectl ipvsadm \
&& systemctl enable kubelet
```

注意：这一步不能直接执行 systemctl start kubelet，否侧会报错，kubelet也起动不成功.

```
systemctl enable kubelet \
&& systemctl status kubelet
```

> - 从安装结果可以看出还安装了cri-tools, kubernetes-cni, socat三个依赖：
>   - 官方从Kubernetes 1.9开始就将cni依赖升级到了0.6.0版本，在当前1.12中仍然是这个版本
>   - socat是kubelet的依赖
>   - cri-tools是CRI(Container Runtime Interface)容器运行时接口的命令行工具

安装内核组件

```
rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm ;yum --en
ablerepo=elrepo-kernel install kernel-lt-devel kernel-lt -y

grub2-editenv list  # 查看内核修改结果
cat /boot/grub2/grub.cfg |grep "menuentry "
grub2-set-default 'CentOS Linux (4.4.177-1.el7.elrepo.x86_64) 7 (Core)'

uname -r   # 查看当前内核版本
# 查看所有可用内核
cat /boot/grub2/grub.cfg |grep "menuentry "

# 检查默认内核版本高于4.1，否则请调整默认启动参数
grub2-editenv list  # 查看内核修改结果

#重启以更换内核
reboot

rpm -qa |grep kernel-[0-9]  # 查看全部内核包
yum remove kernel-3.10.0-327.el7.x86_64  # 删除指定的无用内核

rpm -qa |grep kernel-[0-9]  # 查看全部内核包
```

```
cat /boot/grub2/grub.cfg |grep "menuentry "

# 确认内核版本
uname -a

# 确认内核高于4.1后，开启IPVS
cat > /etc/sysconfig/modules/ipvs.modules <<EOF
#!/bin/bash
ipvs_modules="ip_vs ip_vs_lc ip_vs_wlc ip_vs_rr ip_vs_wrr ip_vs_lblc ip_vs_lblcr ip
_vs_dh ip_vs_sh ip_vs_fo ip_vs_nq ip_vs_sed ip_vs_ftp nf_conntrack_ipv4"
for kernel_module in \${ipvs_modules}; do
 /sbin/modinfo -F filename \${kernel_module} > /dev/null 2>&1
 if [ $? -eq 0 ]; then
 /sbin/modprobe \${kernel_module}
 fi
done
EOF

chmod 755 /etc/sysconfig/modules/ipvs.modules && bash /etc/sysconfig/modules/ipvs.m
odules && lsmod | grep ip_vs
```

2.2 使用kubeadm init初始化集群

在各节点开机启动kubelet服务：

```
systemctl enable kubelet \
&& systemctl status kubelet
```

查看需要安装的镜像

`kubeadm config images list`

`kubeadm config images pull`

```
[root@master01-77 ~]# kubeadm config images list
k8s.gcr.io/kube-apiserver:v1.13.4
k8s.gcr.io/kube-controller-manager:v1.13.4
k8s.gcr.io/kube-scheduler:v1.13.4
k8s.gcr.io/kube-proxy:v1.13.4
k8s.gcr.io/pause:3.1
k8s.gcr.io/etcd:3.2.24
k8s.gcr.io/coredns:1.2.6
```

下载准备镜像

```
cat kubernetes-pull-aliyun-1.13.4.sh
```

```
#!/usr/bin/env bash

MY_REGISTRY=registry.cn-hangzhou.aliyuncs.com/openthings

## 拉取镜像
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-apiserver:v1.13.4
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-controller-manager:v1.13.4
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-scheduler:v1.13.4
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-proxy:v1.13.4
docker pull ${MY_REGISTRY}/k8s-gcr-io-etcd:3.2.24
docker pull ${MY_REGISTRY}/k8s-gcr-io-pause:3.1
docker pull ${MY_REGISTRY}/k8s-gcr-io-coredns:1.2.6


## 添加Tag
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-apiserver:v1.13.4 k8s.gcr.io/kube-apiserv
er:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-scheduler:v1.13.4 k8s.gcr.io/kube-schedul
er:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-controller-manager:v1.13.4 k8s.gcr.io/kub
e-controller-manager:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-proxy:v1.13.4 k8s.gcr.io/kube-proxy:v1.13
.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-etcd:3.2.24 k8s.gcr.io/etcd:3.2.24
docker tag ${MY_REGISTRY}/k8s-gcr-io-pause:3.1 k8s.gcr.io/pause:3.1
docker tag ${MY_REGISTRY}/k8s-gcr-io-coredns:1.2.6 k8s.gcr.io/coredns:1.2.6
```

kubeadm初始化集群

接下来使用kubeadm初始化集群，选择node1作为Master Node，在node1上执行下面的命令：

中立机房K8s集群规划网段
10.210.0.0/16

```
kubeadm init  \
   --kubernetes-version=v1.13.4 \
   --pod-network-cidr=10.210.0.0/16 \
```

```
    --apiserver-advertise-address=10.20.1.77 \
    --ignore-preflight-errors=Swap \
    --ignore-preflight-errors=NumCPU
```

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

kubeadm join 加入集群命令[node节点执行命令]

```
kubeadm join 10.20.1.77:6443 --token w948is.0lnpcawol3hn81l2 --discovery-token-ca-c
ert-hash sha256:62ced6be0aaa79157651e4fd337abbb974110382abbeff98b5813e7074fe4ec4
```

查看kubelet 日志

```
systemctl status kubelet
journalctl -xeu kubelet
```

kubernetes 查询基本命令

```
docker ps -a | grep kube | grep -v pause
docker logs 1f7e11d3c413

kubeadm version
uname -a
cat /etc/debian_version
docker version
```

根据输出的内容基本上可以看出手动初始化安装一个Kubernetes集群所需要的关键步骤。

> 其中有以下关键内容：
> * `[kubelet-start]` 生成kubelet的配置文件”/var/lib/kubelet/config.yaml”
> * `[certificates]`生成相关的各种证书
> * `[kubeconfig]`生成相关的kubeconfig文件
> * `[bootstraptoken]`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
> * 下面的命令是配置常规用户如何使用kubectl访问集群：

```
      mkdir -p $HOME/.kube
```

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

节点加入集群

- 最后给出了将节点加入集群的命令

```
kubeadm join 10.20.1.77:6443 --token w948is.0lnpcawol3hn81l2 --discovery-token-ca-c
ert-hash sha256:62ced6be0aaa79157651e4fd337abbb974110382abbeff98b5813e7074fe4ec4
```

查看一下集群状态：

```
kubectl get cs
```

```
[root@master01-77 ~]# kubectl get cs
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

确认个组件都处于healthy状态。

重置kubernetes集群

kubeadm reset

集群初始化如果遇到问题，可以使用下面的命令进行清理：

```
kubeadm reset
ifconfig cni0 down
ip link delete cni0
ifconfig flannel.1 down
ip link delete flannel.1
rm -rf /var/lib/cni/
```

3.安装基于Calico的Pod Network

3.1 安装Calico网络

Quickstart for Calico on Kubernetes
https://docs.projectcalico.org/v3.6/getting-started/kubernetes/

Install Calico with the following command.

```
wget -c https://docs.projectcalico.org/v3.6/getting-started/kubernetes/installation
/hosted/kubernetes-datastore/calico-networking/1.7/calico.yaml

修改calico.yaml


            # The default IPv4 pool to create on startup if none exists. Pod IPs wi
ll be
            # chosen from this range. Changing this value after installation will h
ave
            # no effect. This should fall within `--cluster-cidr`.
            - name: CALICO_IPV4POOL_CIDR
              value: "10.210.0.0/16"


kubectl apply -f calico.yaml
```

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

```
[root@master01-77 pkg]# kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS
   AGE
kube-system   calico-kube-controllers-644fcf8fbf-j77wx   1/1     Running   0
   41s
kube-system   calico-node-tn7s7                          1/1     Running   0
   41s
kube-system   coredns-86c58d9df4-7xvcx                   1/1     Running   0
   3m9s
kube-system   coredns-86c58d9df4-bsh6h                   1/1     Running   0
   3m9s
kube-system   etcd-master01-77                           1/1     Running   0
   2m19s
kube-system   kube-apiserver-master01-77                 1/1     Running   0
   2m18s
kube-system   kube-controller-manager-master01-77        1/1     Running   0
   2m13s
kube-system   kube-proxy-jfx7n                           1/1     Running   0
   3m9s
```

```
kube-system   kube-scheduler-master01-77                1/1     Running   0
   2m24s
```

使用`kubectl get pod --all-namespaces -o wide`确保所有的Pod都处于Running状态。

```
kubectl get pod --all-namespaces -o wide
```

```
[root@master01-77 ~]# kubectl get pod --all-namespaces -o wide
NAMESPACE     NAME                                      READY   STATUS    RESTARTS
   AGE     IP                NODE          NOMINATED NODE   READINESS GATES
kube-system   calico-kube-controllers-644fcf8fbf-j77wx   1/1     Running   0
   8m23s   192.168.104.194   master01-77   <none>           <none>
kube-system   calico-node-tn7s7                         1/1     Running   0
   8m23s   10.20.1.77        master01-77   <none>           <none>
kube-system   coredns-86c58d9df4-7xvcx                  1/1     Running   0
   10m     192.168.104.193   master01-77   <none>           <none>
kube-system   coredns-86c58d9df4-bsh6h                  1/1     Running   0
   10m     192.168.104.195   master01-77   <none>           <none>
kube-system   etcd-master01-77                          1/1     Running   0
   10m     10.20.1.77        master01-77   <none>           <none>
kube-system   kube-apiserver-master01-77                1/1     Running   0
   10m     10.20.1.77        master01-77   <none>           <none>
kube-system   kube-controller-manager-master01-77       1/1     Running   0
   9m55s   10.20.1.77        master01-77   <none>           <none>
kube-system   kube-proxy-jfx7n                          1/1     Running   0
   10m     10.20.1.77        master01-77   <none>           <none>
kube-system   kube-scheduler-master01-77                1/1     Running   0
   10m     10.20.1.77        master01-77   <none>           <none>
```

3.2 master node参与工作负载

使用kubeadm初始化的集群，出于安全考虑Pod不会被调度到Master Node上，也就是说Master Node不参与工作负载。这是因为当前的master节点node1被打上了http://node-role.kubernetes.io/master:NoSchedule的污点：

因为这里搭建的是测试环境，去掉这个污点使node1参与工作负载：

taint 节点
Remove the taints on the master so that you can schedule pods on it.

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

[root@master01-77 ~]# kubectl taint nodes --all node-role.kubernetes.io/master-node/master01-77 untainted

```
[root@master01-77 ~]# kubectl describe node master01-77 | grep Taints
Taints:             <none>
```

```
[root@master01-77 ~]# kubectl get nodes -o wide
NAME          STATUS   ROLES    AGE     VERSION   INTERNAL-IP    EXTERNAL-IP   OS-IM
AGE               KERNEL-VERSION               CONTAINER-RUNTIME
master01-77   Ready    master   9m30s   v1.13.4   10.20.1.77     <none>        CentO
S Linux 7 (Core)   3.10.0-957.10.1.el7.x86_64   docker://18.6.3
```

3.3 测试DNS

```
kubectl run curl --image=radial/busyboxplus:curl -it
kubectl run --generator=deployment/apps.v1beta1 is DEPRECATED and will be removed i
n a future version. Use kubectl create instead.
If you don't see a command prompt, try pressing enter.
[ root@curl-5cc7b478b6-r997p:/ ]$
```

进入后执行`nslookup kubernetes.default`确认解析正常:

```
[ root@curl-66959f6557-xdnpx:/ ]$ nslookup kubernetes.default
Server:    10.96.0.10
Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name:      kubernetes.default
Address 1: 10.96.0.1 kubernetes.default.svc.cluster.local
```

3.4 向Kubernetes集群中添加Node节点

下面我们将node2这个主机添加到Kubernetes集群中，因为我们同样在node2上的kubelet的启动参数中去掉了必须关闭swap的限制，所以同样需要`--ignore-preflight-errors=Swap`这个参数。 在node2上执行:

```
kubeadm join 10.20.1.77:6443 --token w948is.0lnpcawol3hn81l2 --discovery-token-ca-c
ert-hash sha256:62ced6be0aaa79157651e4fd337abbb974110382abbeff98b5813e7074fe4ec4 \
 --ignore-preflight-errors=Swap
```

```
[preflight] Running pre-flight checks
        [WARNING Swap]: running with swap on is not supported. Please disable swap
[discovery] Trying to connect to API Server "192.168.61.11:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://19
2.168.61.11:6443"
[discovery] Requesting info from "https://192.168.61.11:6443" again to validate TLS
 against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate valid
ates against pinned roots, will use API Server "192.168.61.11:6443"
[discovery] Successfully established connection with API Server "192.168.61.11:6443
"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm ku
beadm-config -oyaml'
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13"
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/kubel
et/kubeadm-flags.env"
[kubelet-start] 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 "node2" 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.
```

node2加入集群很是顺利，下面在master节点上执行命令查看集群中的节点：

```
kubectl get nodes
NAME    STATUS   ROLES    AGE    VERSION
node1   Ready    master   16m    v1.13.0
node2   Ready    <none>   4m5s   v1.13.0
```

3.5 如何从集群中移除Node

如果需要从集群中移除node2这个Node执行下面的命令：

在master节点上执行：

```
kubectl drain node2 --delete-local-data --force --ignore-daemonsets
kubectl delete node node2
```

在node2上执行：

```
kubeadm reset
ifconfig cni0 down
ip link delete cni0
ifconfig flannel.1 down
ip link delete flannel.1
rm -rf /var/lib/cni/
```

在node1上执行：

```
kubectl delete node node2
```

3.6 kube-proxy开启ipvs[所有节点]

修改ConfigMap的kube-system/kube-proxy中的config.conf，`mode: "ipvs"`：

```
kubectl edit cm kube-proxy -n kube-system
```

之后重启各个节点上的kube-proxy pod：

```
kubectl get pod -n kube-system | grep kube-proxy | awk '{system("kubectl delete pod
 "$1" -n kube-system")}'
```

```
kubectl get pod -n kube-system | grep kube-proxy
```

```
[root@master01-77 ~]# kubectl get pod -n kube-system | grep kube-proxy
kube-proxy-c294x                           1/1     Running   0          22s

[root@master01-77 ~]# kubectl logs kube-proxy-c294x -n kube-system
```

```
I0320 09:13:09.808804       1 server_others.go:189] Using ipvs Proxier.
W0320 09:13:09.809343       1 proxier.go:381] IPVS scheduler not specified, use rr
by default
I0320 09:13:09.809545       1 server_others.go:216] Tearing down inactive rules.
I0320 09:13:09.873192       1 server.go:483] Version: v1.13.4
I0320 09:13:09.897102       1 conntrack.go:52] Setting nf_conntrack_max to 524288
I0320 09:13:09.897383       1 config.go:102] Starting endpoints config controller
I0320 09:13:09.897406       1 controller_utils.go:1027] Waiting for caches to sync
for endpoints config controller
I0320 09:13:09.897435       1 config.go:202] Starting service config controller
I0320 09:13:09.897457       1 controller_utils.go:1027] Waiting for caches to sync
for service config controller
I0320 09:13:09.997607       1 controller_utils.go:1034] Caches are synced for servi
ce config controller
I0320 09:13:09.997611       1 controller_utils.go:1034] Caches are synced for endpo
ints config controller
```

日志中打印出了`Using ipvs Proxier`，说明ipvs模式已经开启。

4.Kubernetes常用组件部署

越来越多的公司和团队开始使用Helm这个Kubernetes的包管理器，我们也将使用Helm安装Kubernetes的常用组件。

4.1 Helm的安装

helm添加阿里云仓库
因官方仓库被墙，可以将官方仓库stable改成国内阿里云的镜像仓库

查看当前的仓库地址

```
helm repo list
```

修改为阿里云仓库地址

```
helm repo remove stable
helm repo add stable https://kubernetes.oss-cn-hangzhou.aliyuncs.com/charts
helm repo update
helm repo list
```

Helm由客户端命helm令行工具和服务端tiller组成，Helm的安装十分简单。 下载helm命令行工具到

```
wget https://storage.googleapis.com/kubernetes-helm/helm-v2.12.0-linux-amd64.tar.gz
tar -zxvf helm-v2.12.0-linux-amd64.tar.gz
cd linux-amd64/
cp helm /usr/local/bin/
```

为了安装服务端tiller，还需要在这台机器上配置好kubectl工具和kubeconfig文件，确保kubectl工具可以在这台机器上访问apiserver且正常使用。 这里的node1节点以及配置好了kubectl。

因为Kubernetes APIServer开启了RBAC访问控制，所以需要创建tiller使用的service account: tiller并分配合适的角色给它。 详细内容可以查看helm文档中的Role-based Access Control。 这里简单起见直接分配cluster-admin这个集群内置的ClusterRole给它。创建rbac-config.yaml文件：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

```
kubectl create -f rbac-config.yaml
serviceaccount/tiller created
clusterrolebinding.rbac.authorization.k8s.io/tiller created
```

接下来使用helm部署tiller:

```
helm init --service-account tiller --skip-refresh
Creating /root/.helm
Creating /root/.helm/repository
Creating /root/.helm/repository/cache
Creating /root/.helm/repository/local
Creating /root/.helm/plugins
Creating /root/.helm/starters
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Clu
ster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated
 users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_
helm/#securing-your-helm-installation
Happy Helming!
```

```
[root@master01-77 helm]# helm init --service-account tiller --skip-refresh
Creating /root/.helm
Creating /root/.helm/repository
Creating /root/.helm/repository/cache
Creating /root/.helm/repository/local
Creating /root/.helm/plugins
Creating /root/.helm/starters
Creating /root/.helm/cache/archive
Creating /root/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /root/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Clu
ster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated
 users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_
helm/#securing-your-helm-installation
```

```
Happy Helming!
You have new mail in /var/spool/mail/root
```

tiller默认被部署在k8s集群中的kube-system这个namespace下：

```
kubectl get pod -n kube-system -l app=helm
NAME                             READY   STATUS    RESTARTS   AGE
tiller-deploy-c4fd4cd68-dwkhv    1/1     Running   0          83s
```

```
[root@master01-77 helm]# kubectl get pod -n kube-system -l app=helm
NAME                             READY   STATUS    RESTARTS   AGE
tiller-deploy-5b7c66d59c-2pb9f   1/1     Running   0          5m40s
```

```
[root@master01-77 helm]# helm version
Client: &version.Version{SemVer:"v2.13.0", GitCommit:"79d07943b03aea2b76c12644b4b54
733bc5958d6", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.13.0", GitCommit:"79d07943b03aea2b76c12644b4b54
733bc5958d6", GitTreeState:"clean"}
```

> 注意由于某些原因需要网络可以访问gcr.io和kubernetes-charts.storage.googleapis.com，如果无法访问可以通过 `helm init --service-account tiller --tiller-image <your-docker-registry>/tiller:v2.11.0 --skip-refresh` 使用私有镜像仓库中的tiller镜像

4.2 使用Helm部署Nginx Ingress

为了便于将集群中的服务暴露到集群外部，从集群外部访问，接下来使用Helm将Nginx Ingress部署到Kubernetes上。 Nginx Ingress Controller被部署在Kubernetes的边缘节点上，关于Kubernetes边缘节点的高可用相关的内容可以查看我前面整理的Bare metal环境下Kubernetes Ingress边缘节点的高可用(基于IPVS)。

我们将node1(192.168.61.11)和node2(192.168.61.12)同时做为边缘节点，打上Label：

```
kubectl label node master01-77 node-role.kubernetes.io/edge=
```

```
kubectl label node node1 node-role.kubernetes.io/edge=
node/node1 labeled

kubectl label node node2 node-role.kubernetes.io/edge=
node/node2 labeled
```

```
kubectl get node
NAME    STATUS   ROLES         AGE   VERSION
node1   Ready    edge,master   24m   v1.13.0
node2   Ready    edge          11m   v1.13.0
```

```
[root@master01-77 dashboard]# kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
master01-77   Ready    master   22h   v1.13.4
[root@master01-77 dashboard]# kubectl label node master01-77 node-role.kubernetes.i
o/edge=
node/master01-77 labeled
[root@master01-77 dashboard]# kubectl get node
NAME          STATUS   ROLES         AGE   VERSION
master01-77   Ready    edge,master   22h   v1.13.4
```

stable/nginx-ingress chart的值文件ingress-nginx.yaml：

```
controller:
  replicaCount: 2
  service:
    externalIPs:
      - 10.20.1.77
  nodeSelector:
    node-role.kubernetes.io/edge: ''
  affinity:
    podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - nginx-ingress
            - key: component
              operator: In
              values:
              - controller
          topologyKey: kubernetes.io/hostname
  tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule

defaultBackend:
  nodeSelector:
```

```
    node-role.kubernetes.io/edge: ''
  tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
```

nginx ingress controller的副本数replicaCount为2，将被调度到node1和node2这两个边缘节点上。externalIPs指定的10.20.1.77为VIP，将绑定到 `kube-proxy` `kube-ipvs0` 网卡上。

```
docker pull mirrorgooglecontainers/defaultbackend:1.4
docker tag mirrorgooglecontainers/defaultbackend:1.4 k8s.gcr.io/defaultbackend:1.4
```

```
helm ls
helm del --purge nginx-ingress
```

```
helm repo update

helm install stable/nginx-ingress \
-n nginx-ingress \
--namespace ingress-nginx  \
-f ingress-nginx.yaml
```

```
kubectl get pod -n ingress-nginx -o wide
NAME                                             READY   STATUS    RESTARTS   AGE
  IP           NODE    NOMINATED NODE   READINESS GATES
nginx-ingress-controller-85f8597fc6-g2kcx        1/1     Running   0          5m2s
  10.244.1.3   node2   <none>           <none>
nginx-ingress-controller-85f8597fc6-g7pp5        1/1     Running   0          5m2s
  10.244.0.5   node1   <none>           <none>
nginx-ingress-default-backend-6dc6c46dcc-7plm8   1/1     Running   0          5m2s
  10.244.1.4   node2   <none>           <none>
```

```
[root@master01-77 ingress-nginx]# helm install stable/nginx-ingress -n nginx-ingres
s --namespace ingress-nginx  -f ingress-nginx.yaml
NAME:   nginx-ingress
LAST DEPLOYED: Thu Mar 21 15:21:28 2019
NAMESPACE: ingress-nginx
STATUS: DEPLOYED
```

```
RESOURCES:
==> v1/ConfigMap
NAME                       DATA  AGE
nginx-ingress-controller   1     1s

==> v1/Pod(related)
NAME                                            READY  STATUS             RESTARTS
AGE
nginx-ingress-controller-86649f6f4-78nqf        0/1    ContainerCreating  0
1s
nginx-ingress-default-backend-759456dbc-6dx28   0/1    ContainerCreating  0
1s

==> v1/Service
NAME                            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
                 AGE
nginx-ingress-controller        LoadBalancer   10.96.73.230    10.20.1.77    80:30287/T
CP,443:30421/TCP  1s
nginx-ingress-default-backend   ClusterIP      10.102.95.173   <none>        80/TCP
                 1s

==> v1beta1/Deployment
NAME                            READY  UP-TO-DATE  AVAILABLE  AGE
nginx-ingress-controller        0/1    1           0          1s
nginx-ingress-default-backend   0/1    1           0          1s

==> v1beta1/PodDisruptionBudget
NAME                            MIN AVAILABLE  MAX UNAVAILABLE  ALLOWED DISRUPTIONS
 AGE
nginx-ingress-controller        1              N/A              0
 1s
nginx-ingress-default-backend   1              N/A              0
 1s


NOTES:
The nginx-ingress controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace ingress-nginx get services
 -o wide -w nginx-ingress-controller'

An example Ingress that makes use of the controller:

  apiVersion: extensions/v1beta1
  kind: Ingress
  metadata:
    annotations:
```

```
      kubernetes.io/ingress.class: nginx
    name: example
    namespace: foo
  spec:
    rules:
      - host: www.example.com
        http:
          paths:
            - backend:
                serviceName: exampleService
                servicePort: 80
              path: /
    # This section is only required if TLS is to be enabled for the Ingress
    tls:
        - hosts:
            - www.example.com
          secretName: example-tls

If TLS is enabled for the Ingress, a Secret containing the certificate and key must
 also be provided:

  apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls
```

如果访问http://10.20.1.77返回default backend，则部署完成。

实际测试的结果是无法访问，于是怀疑`kube-proxy`出了问题，查看`kube-proxy`的日志，不停的刷下面的log：

```
I1208 07:59:28.902970       1 graceful_termination.go:160] Trying to delete rs: 10.
104.110.193:80/TCP/10.244.1.5:80
I1208 07:59:28.903037       1 graceful_termination.go:170] Deleting rs: 10.104.110.
193:80/TCP/10.244.1.5:80
I1208 07:59:28.903072       1 graceful_termination.go:160] Trying to delete rs: 10.
104.110.193:80/TCP/10.244.0.6:80
I1208 07:59:28.903105       1 graceful_termination.go:170] Deleting rs: 10.104.110.
193:80/TCP/10.244.0.6:80
I1208 07:59:28.903713       1 graceful_termination.go:160] Trying to delete rs: 10.
20.1.77:80/TCP/10.244.1.5:80
I1208 07:59:28.903764       1 graceful_termination.go:170] Deleting rs: 10.20.1.77:
```

```
80/TCP/10.244.1.5:80
I1208 07:59:28.903798       1 graceful_termination.go:160] Trying to delete rs: 10.
20.1.77:80/TCP/10.244.0.6:80
I1208 07:59:28.903824       1 graceful_termination.go:170] Deleting rs: 10.20.1.77:
80/TCP/10.244.0.6:80
I1208 07:59:28.904654       1 graceful_termination.go:160] Trying to delete rs: 10.
0.2.15:31698/TCP/10.244.0.6:80
I1208 07:59:28.904837       1 graceful_termination.go:170] Deleting rs: 10.0.2.15:3
1698/TCP/10.244.0.6:80
```

4.3 使用Helm部署dashboard

准备镜像

```
docker pull mirrorgooglecontainers/kubernetes-dashboard-amd64:v1.10.1
docker tag mirrorgooglecontainers/kubernetes-dashboard-amd64:v1.10.1 k8s.gcr.io/kub
ernetes-dashboard-amd64:v1.10.1
```

kubernetes-dashboard.yaml：

```
image:
  repository: k8s.gcr.io/kubernetes-dashboard-amd64
  tag: v1.10.1
ingress:
  enabled: true
  hosts:
    - k8s.frognew.com
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
  tls:
    - secretName: frognew-com-tls-secret
      hosts:
      - k8s.frognew.com
rbac:
  clusterAdminRole: true
```

```
helm install stable/kubernetes-dashboard \
-n kubernetes-dashboard \
--namespace kube-system  \
-f kubernetes-dashboard.yaml
```

```
kubectl -n kube-system get secret | grep kubernetes-dashboard-token
kubernetes-dashboard-token-pkm2s                 kubernetes.io/service-account-toke
n   3      3m7s

kubectl describe -n kube-system secret/kubernetes-dashboard-token-pkm2s
Name:         kubernetes-dashboard-token-pkm2s
Namespace:    kube-system
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: kubernetes-dashboard
              kubernetes.io/service-account.uid: 2f0781dd-156a-11e9-b0f0-080027bb7c
43

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1025 bytes
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2
NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsI
mt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2Fy
ZC10b2tlbi1wa20ycyIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5
hbWUiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2Vydm
ljZS1hY2NvdW50LnVpZCI6IjJmMDc4MWRkLTE1NmEtMTFlOS1iMGYwLTA4MDAyN2JiN2M0MyIsInN1YiI6I
nN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTprdWJlcm5ldGVzLWRhc2hib2FyZCJ9.24ad6Zg
ZMxdydpwlmYAiMxZ9VSIN7dDR7Q6-RLW0qC81ajXoQKHAyrEGpIonfld3gqbE0xO8nisskpmlkQra72-9X6
sBPoByqIKyTsO83BQlME2sfOJemWD0HqzwSCjvSQa0x-bUlq9HgH2vEXzpFuSS6Svi7RbfzLXlEuggNoC4M
fA4E2hF1OX_ml8iAKx-49y1BQQe5FGWyCyBSi1TD_-ZpVs44H5gIvsGK2kcvi0JT4oHXtWjjQBKLIWL7xxy
RCSE4HmUZT2StIHnOwlX7IEIB0oBX4mPg2_xNGnqwcu-8OERU9IoqAAE2cZa0v3b5O2LMcJPrcxrVOukvRI
umA
```

在dashboard的登录窗口使用上面的token登录。

5.总结

本次安装涉及到的Docker镜像：

```
# kubernetes
k8s.gcr.io/kube-apiserver:v1.13.2
k8s.gcr.io/kube-controller-manager:v1.13.2
k8s.gcr.io/kube-proxy:v1.13.2
```

```
k8s.gcr.io/kube-scheduler:v1.13.2
k8s.gcr.io/kube-proxy:v1.13.1
k8s.gcr.io/etcd:3.2.24
k8s.gcr.io/pause:3.1

# network and dns
quay.io/coreos/flannel:v0.10.1-amd64
k8s.gcr.io/coredns:1.2.6

# helm and tiller
gcr.io/kubernetes-helm/tiller:v2.12.0

# nginx ingress
quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.21.0
k8s.gcr.io/defaultbackend:1.4

# dashboard and metric-sever
k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
gcr.io/google_containers/metrics-server-amd64:v0.3.1
```

FAQ

/sbin/ldconfig: File /lib64/libapr-1.so.0 is empty, not checked.

```
/sbin/ldconfig: File /lib64/libapr-1.so.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libapr-1.so.0.4.8 is empty, not checked.
/sbin/ldconfig: File /lib64/libboost_system-mt.so.1.53.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libaprutil-1.so.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libaprutil-1.so.0.5.2 is empty, not checked.
/sbin/ldconfig: File /lib64/libboost_system.so.1.53.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libboost_thread-mt.so.1.53.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libpakchois.so.0 is empty, not checked.
/sbin/ldconfig: File /lib64/libpakchois.so.0.1.0 is empty, not checked.
```

解决方案：
yum reinstall apr
reinstall对应包即可。

sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-

ip6tables: No such file or directory

```
modprobe br_netfilter

sysctl -p
sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-ip6tables: No such file or
directory
sysctl: cannot stat /proc/sys/net/bridge/bridge-nf-call-iptables: No such file or d
irectory
```

解决办法：

```
modprobe br_netfilter

ls /proc/sys/net/bridge
bridge-nf-call-arptables bridge-nf-filter-pppoe-tagged
bridge-nf-call-ip6tables bridge-nf-filter-vlan-tagged
bridge-nf-call-iptables bridge-nf-pass-vlan-input-dev


sysctl -p
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

kubeadm systemd[1]: Started kubelet: The Kubernetes Node Agent.

2019年03月21日09:24:33

安装kubelet等组件

```
yum install -y kubelet kubeadm kubectl ipvsadm \
&& systemctl enable kubelet \
&& systemctl status kubelet
```

机房K8s集群规划网段
注意：这一步不能直接执行 systemctl start kubelet，否侧会报错，kubelet也起动不成功

```
7月 29 12:17:19 kubeadm systemd[1]: Started kubelet: The Kubernetes Node Agent.
7月 29 12:17:19 kubeadm systemd[1]: Starting kubelet: The Kubernetes Node Agent...
7月 29 12:17:20 kubeadm kubelet[32751]: F0729 12:17:20.026220   32751 server.go:190]
```

```
 failed to load Kubelet config file /var/lib/kubelet/config.yaml, error failed to r
ead kubelet config file "/var/lib/kubelet/config.yaml", error: open /var/lib/kubele
t/config.yaml: no such file or directory
```

END