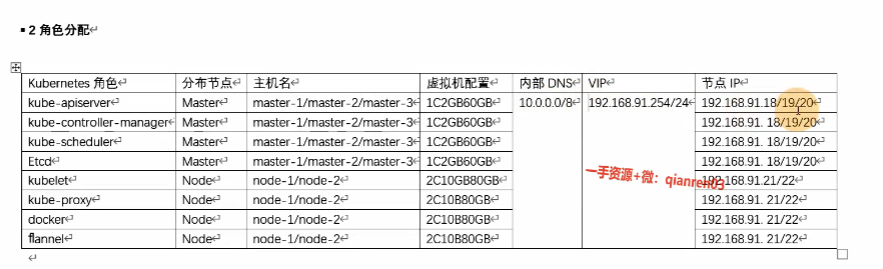
# 老男孩笔记-（二进制搭建k8s）

## 1、软件版本

|  |
| --- |
| 软件/系统 版本 备注  CentOS 7.9  kubernetes-node-linux-amd64.tar.gz 1.15.2  flannel 0.11  etcd 3.3.10 |

## 2、角色分配



Etc 存储

Apiserver 任务接口 6443

Scheduler 任务调度, pod调度

Control-manager 管理容器

Kubelet agent启动容器

Kube-proxy 启动端口

**Kubernetes 1.15 安装**

## 3、集群部署架构

本章节部署架构为: Master-1、Master-2、Master-3、 Node-1、 Node-2

## 4、系统初始化

### 4.1 初始化工具安装

#所有节点

yum install net-tools vim wget lrzsz git -y

### 4.2 关闭防火墙与Selinux

#所有节点

systemctl stop firewalld  
systemctl disable firewalld  
sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config  
reboot

### 4.3设置时区

#所有节点

\cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime -rf

### 4.4关闭交换分区

#所有节点

swapoff -a  
sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab

### 4.5设置系统时间同步

#所有节点

yum install -y ntpdate  
ntpdate -u ntp.api.bz  
echo "\*/5 \* \* \* \* ntpdate time7.aliyun.com >/dev/null 2>&1" >> /etc/crontab  
service crond restart  
chkconfig crond on

sudo tee /etc/chrony.conf << 'EOF'  
pool s1a.time.edu.cn iburst maxsources 1  
pool ntp.aliyun.com iburst maxsources 4  
keyfile /etc/chrony.keys  
driftfile /var/lib/chrony/chrony.drift  
logdir /var/log/chrony  
maxupdateskew 100.0  
rtcsync  
makestep 1.0 3  
EOF  
  
systemctl enable chronyd && systemctl restart chronyd && systemctl status chronyd -l  
sudo timedatectl set-local-rtc 0  
timedatectl set-ntp yes  
chronyc tracking   
hwclock --systohc  
sudo systemctl restart rsyslog.service crond.service

### 4.6 设置主机名

[root@master-1 ]#hostnamectl set-hostname master-1  
[root@master-2]#hostnamectl set-hostname master-2  
[root@master-3]#hostnamectl set-hostname master-3  
[root@node-1]#hostnamectl set-hostname node-1  
[root@node-2]#hostnamectl set-hostname node-2  
  
#所有节点  
[root@master-1 ~]# cat >> /etc/hosts <<EOF  
192.168.91.18 master-1  
192.168.91.19 master-2  
192.168.91.20 master-3  
192.168.91.21 node-1  
192.168.91.22 node-2  
EOF

### 4.7 设置免密码登录

#从任意Master节点分发配置到其他所有的节点(包括其他的Master与Node)  
#本例中从master-1分发  
[root@master-1 ~]# yum install -y expect  
[root@master-1 ~]# ssh-keygen -t rsa -P "" -f /root/.ssh/id\_rsa  
#密码更换  
[root@master-1 ~]# export mypass=123.com  
[root@master-1 ~]# name=(master-1 master-2 master-3 node-1 node-2)  
[root@master-1 ~]# for i in ${name[@]};do  
expect -c "  
spawn ssh-copy-id -i /root/.ssh/id\_rsa.pub root@$i  
 expect {  
 \"\*yes/no\*\" {send \"yes\r\"; exp\_continue}  
 \"\*password\*\" {send \"$mypass\r\"; exp\_continue}  
 \"\*Password\*\" {send \"$mypass\r\";}  
 }"  
done  
#连接测试  
[root@master-1 ~]#ssh master-2

### 4.8 升级内核版本

#!/bin/bash  
export PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin  
LOGDIR=/var/log/init\_record  
if [ ! -d ${LOGDIR} ];then mkdir -vp ${LOGDIR}; fi  
log\_err() {  
 printf "[$(date +'%Y-%m-%dT%H:%M:%S')]: \033[31mERROR: $@ \033[0m\n"  
}  
  
log\_info() {  
 printf "[$(date +'%Y-%m-%dT%H:%M:%S')]: \033[32mINFO: $@ \033[0m\n"  
}  
  
log\_warning() {  
 printf "[$(date +'%Y-%m-%dT%H:%M:%S')]: \033[33mWARNING: $@ \033[0m\n"  
}  
upgrade\_kernel() {  
 log\_name=upgrade\_kernel.log  
 succ\_num=$(grep successful ${LOGDIR}/${log\_name}|wc -l)  
 if [ ! -f ${LOGDIR}/${log\_name} -o ${succ\_num} -eq 0 ];then  
 cp -av /etc/grub2.cfg ${BACKUPDIR}/grub2.cfg.kernelupdate.bak |tee -a ${LOGDIR}/${log\_name}  
 log\_info "[\*] CentOS 操作系统内核升级(可选) "  
 rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org |tee -a ${LOGDIR}/${log\_name}  
 yum -y install https://www.elrepo.org/elrepo-release-7.el7.elrepo.noarch.rpm |tee -a ${LOGDIR}/${log\_name}  
 yum install -y yum-plugin-fastestmirror |tee -a ${LOGDIR}/${log\_name}  
 yum --disablerepo="\*" --enablerepo=elrepo-kernel repolist |tee -a ${LOGDIR}/${log\_name}  
 yum --disablerepo="\*" --enablerepo=elrepo-kernel list kernel\* |tee -a ${LOGDIR}/${log\_name}  
 # 内核安装，服务器里我们选择长期lt版本，安全稳定是我们最大的需求，除非有特殊的需求内核版本需求;  
 yum update -y --enablerepo=elrepo-kernel |tee -a ${LOGDIR}/${log\_name}  
 # 内核版本介绍, lt:longterm 的缩写长期维护版, ml:mainline 的缩写最新主线版本;  
 yum install -y --enablerepo=elrepo-kernel --skip-broken kernel-lt kernel-lt-devel kernel-lt-tools |tee -a ${LOGDIR}/${log\_name}  
 # yum -y --enablerepo=elrepo-kernel --skip-broken install kernel-ml.x86\_64 kernel-ml-devel.x86\_64 kernel-ml-tools.x86\_64  
 log\_warning "[\*] 当前 CentOS 操作系统可切换的内核内核版本" |tee -a ${LOGDIR}/${log\_name}  
 awk -F \' '$1=="menuentry " {print i++ " : " $2}' /etc/grub2.cfg |tee -a ${LOGDIR}/${log\_name}  
 sudo grub2-set-default 0 |tee -a ${LOGDIR}/${log\_name}  
 log\_info "[\*] 操作系统内核升级successful!,接下来重启！ " |tee -a ${LOGDIR}/${log\_name}  
 reboot  
 else  
 log\_info "[\*] 操作系统内核已升级!"  
 fi  
  
}  
  
upgrade\_kernel

### 4.9 优化内核参数

#所有节点  
[root@master-1 ~]# cat >/etc/sysctl.d/kubernetes.conf <<EOF  
net.bridge.bridge-nf-call-iptables=1  
net.bridge.bridge-nf-call-ip6tables=1  
net.ipv4.ip\_forward=1  
vm.swappiness=0  
fs.file-max=52706963  
fs.nr\_open=52706963  
EOF  
  
#应用内核配置  
[root@master-1 ~]# sysctl -p

### 4.10 ipvs

# 所有节点  
# 安装ipvs工具,方便以后操作ipvs，ipset，conntrack等  
yum install ipvsadm ipset sysstat conntrack libseccomp -y  
# 所有节点配置ipvs模块,执行以下命令，在内核4.19+版本改为nf\_conntrack， 4.18下改为nf\_conntrack\_ipv4  
modprobe -- ip\_vs  
modprobe -- ip\_vs\_rr  
modprobe -- ip\_vs\_wrr  
modprobe -- ip\_vs\_sh  
modprobe -- nf\_conntrack  
  
#修改ipvs配置，加入以下内容  
vi /etc/modules-load.d/ipvs.conf  
  
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  
ip\_vs\_sh  
nf\_conntrack  
ip\_tables  
ip\_set  
xt\_set  
ipt\_set  
ipt\_rpfilter  
ipt\_REJECT  
ipip  
  
# 执行命令  
systemctl enable --now systemd-modules-load.service #--now = enable+start  
  
#检测是否加载  
lsmod | grep -e ip\_vs -e nf\_conntrack

### 4.10 优化内核

## 所有节点  
cat <<EOF > /etc/sysctl.d/k8s.conf  
net.ipv4.ip\_forward = 1  
net.bridge.bridge-nf-call-iptables = 1  
net.bridge.bridge-nf-call-ip6tables = 1  
fs.may\_detach\_mounts = 1  
vm.overcommit\_memory=1  
net.ipv4.conf.all.route\_localnet = 1  
  
vm.panic\_on\_oom=0  
fs.inotify.max\_user\_watches=89100  
fs.file-max=52706963  
fs.nr\_open=52706963  
net.netfilter.nf\_conntrack\_max=2310720  
  
net.ipv4.tcp\_keepalive\_time = 600  
net.ipv4.tcp\_keepalive\_probes = 3  
net.ipv4.tcp\_keepalive\_intvl =15  
net.ipv4.tcp\_max\_tw\_buckets = 36000  
net.ipv4.tcp\_tw\_reuse = 1  
net.ipv4.tcp\_max\_orphans = 327680  
net.ipv4.tcp\_orphan\_retries = 3  
net.ipv4.tcp\_syncookies = 1  
net.ipv4.tcp\_max\_syn\_backlog = 16768  
net.ipv4.ip\_conntrack\_max = 65536  
net.ipv4.tcp\_timestamps = 0  
net.core.somaxconn = 16768  
EOF  
sysctl --system

# 所有节点配置完内核后，重启服务器，保证重启后内核依旧加载  
reboot  
lsmod | grep -e ip\_vs -e nf\_conntrack

### 4.11 高可用节点安装Keepalived

#192.168.91.18 192.168.91.19两台

|  |
| --- |
| [root@master-1 ~]# yum install -y keepalived  #注意修改网卡地址与SLAVE节点的优先级  [root@master-1 ~]# cat >/etc/keepalived/keepalived.conf <<EOL  global\_defs {  router\_id KUB\_LVS  }  vrrp\_script CheckMaster {  script "curl -k [https://192.168.91.254:6443"](https://192.168.91.254:6443%22)  interval 3  timeout 9  fall 2  rise 2  }  vrrp\_instance VI\_1 {  state MASTER  interface ens33  virtual\_router\_id 61  priority 100  advert\_int 1  nopreempt  authentication {  auth\_type PASS  auth\_pass 111111  }  virtual\_ipaddress {  192.168.91.254/24 dev ens33  }  track\_script {  CheckMaster  }  }  EOL  **#SLAVE**  #修改state为slave, priority 为 90 |

**启动keepalived**

[root@master-1 ~]# systemctl enable keepalived && systemctl restart keepalived

[root@master-1 ~]# service keepalived status

**所有节点ping虚拟IP地址192.168.91.254，通是正常的**

### 4.12 安装Nginx （先不做，跳过）

|  |
| --- |
| #192.168.91.11  #添加版本库  [root@harbor ~]# vim /etc/yum.repos.d/nginx.repo  [nginx]  name=nginx repo  baseurl=http://nginx.org/packages/centos/7/x86\_64/  gpgcheck=0  enabled=1  #安装  [root@harbor ~]# yum install nginx-1.12.2 -y  #删除默认页面  [root@harbor harbor]# rm /etc/nginx/conf.d/default.conf -rf  #编辑配置文件  [root@harbor harbor]# vim /etc/nginx/nginx.conf  #最后添加 http之外  stream {  log\_format main '$remote\_addr $upstream\_addr - [$time\_local] $status $upstream\_bytes\_sent';  access\_log /var/log/nginx/access.log main;  upstream apiserver {  server 192.168.91.18:6443;  server 192.168.91.19:6443;  server 192.168.91.20:6443;  }  server {  listen 192.168.91.254:6443;  proxy\_connect\_timeout 1s;  proxy\_timeout 2s;  proxy\_pass apiserver;  }  }  #启动服务  [root@harbor harbor]# chkconfig nginx on  [root@harbor harbor]# service nginx start |

## 5、配置证书

### 5.1 下载自签名证书生成工具

#在分发机器Master-1上操作  
[root@master-1 ~]# mkdir /soft && cd /soft  
[root@master-1 ~]# wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64  
[root@master-1 ~]# wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64  
[root@master-1 ~]# wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64  
[root@master-1 ~]# chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64  
[root@master-1 ~]# mv cfssl\_linux-amd64 /usr/local/bin/cfssl  
[root@master-1 ~]# mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson  
[root@master-1 ~]# mv cfssl-certinfo\_linux-amd64 /usr/bin/cfssl-certinfo

### 5.2 生成ETCD证书

#创建目录（Master-1）  
[root@master-1 ~]# mkdir /root/etcd && cd /root/etcd

#### 5.2.1 CA 证书配置（Master-1）

[root@master-1 ~]# cat << EOF | tee ca-config.json  
{  
 "signing": {  
 "default": {  
 "expiry": "87600h"  
 },  
 "profiles": {  
 "www": {  
 "expiry": "87600h",  
 "usages": [  
 "signing",  
 "key encipherment",  
 "server auth",  
 "client auth"  
 ]  
 }  
 }  
 }  
}  
EOF

#### 5.2.2 创建CA证书请求文件（Master-1）

[root@master-1 ~]# cat << EOF | tee ca-csr.json  
{  
 "CN": "etcd CA",  
 "key": {  
 "algo": "rsa",  
 "size": 2048  
 },  
 "names": [  
 {  
 "C": "CN",  
 "L": "Beijing",  
 "ST": "Beijing"  
 }  
 ]  
}  
EOF

#### 5.2.3 创建ETCD证书请求文件

#可以把所有的master IP 加入到csr文件中（Master-1）  
[root@master-1 ~]# cat << EOF | tee server-csr.json  
{  
 "CN": "etcd",  
 "hosts": [  
 "master-1",  
 "master-2",  
 "master-3",  
 "192.168.91.18",  
 "192.168.91.19",  
 "192.168.91.20"  
 ],  
 "key": {  
 "algo": "rsa",  
 "size": 2048  
 },  
 "names": [  
 {  
 "C": "CN",  
 "L": "Beijing",  
 "ST": "Beijing"  
 }  
 ]  
}  
EOF

**5.2.4 生成 ETCD CA 证书和ETCD公私钥（Master-1）**

|  |
| --- |
| [root@master-1 ~]# cd /root/etcd/  #生成ca证书（Master-1）  [root@master-1 ~]# cfssl gencert -initca ca-csr.json | cfssljson -bare ca – |

[root@master-1 etcd]# ll  
total 24  
-rw-r--r-- 1 root root 287 Apr 5 11:23 ca-config.json #ca 的配置文件  
-rw-r--r-- 1 root root 956 Apr 5 11:26 ca.csr #ca 证书生成文件  
-rw-r--r-- 1 root root 209 Apr 5 11:23 ca-csr.json #ca 证书请求文件  
-rw------- 1 root root 1679 Apr 5 11:26 ca-key.pem #ca 证书key  
-rw-r--r-- 1 root root 1265 Apr 5 11:26 ca.pem #ca 证书  
-rw-r--r-- 1 root root 338 Apr 5 11:26 server-csr.json

|  |
| --- |
| #生成etcd证书（Master-1）  [root@master-1 etcd]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=www server-csr.json | cfssljson -bare server |

[root@master-1 etcd]# ll  
total 36  
-rw-r--r-- 1 root root 287 Apr 5 11:23 ca-config.json  
-rw-r--r-- 1 root root 956 Apr 5 11:26 ca.csr  
-rw-r--r-- 1 root root 209 Apr 5 11:23 ca-csr.json  
-rw------- 1 root root 1679 Apr 5 11:26 ca-key.pem  
-rw-r--r-- 1 root root 1265 Apr 5 11:26 ca.pem  
-rw-r--r-- 1 root root 1054 Apr 5 11:31 server.csr  
-rw-r--r-- 1 root root 338 Apr 5 11:26 server-csr.json  
-rw------- 1 root root 1675 Apr 5 11:31 server-key.pem #etcd客户端使用  
-rw-r--r-- 1 root root 1379 Apr 5 11:31 server.pem

**5.3 创建 Kubernetes 相关证书**

|  |
| --- |
| #此证书用于Kubernetes节点直接的通信, 与之前的ETCD证书不同. （Master-1）  [root@master-1 ~]# mkdir /root/kubernetes/ && cd /root/kubernetes/ |

**5.3.1 配置ca 文件（Master-1）**

|  |
| --- |
| [root@master-1 ~]# cat << EOF | tee ca-config.json  {  "signing": {  "default": {  "expiry": "87600h"  },  "profiles": {  "kubernetes": {  "expiry": "87600h",  "usages": [  "signing",  "key encipherment",  "server auth",  "client auth"  ]  }  }  }  }  EOF |

**5.3.2 创建ca证书申请文件（Master-1）**

|  |
| --- |
| [root@master-1 ~]# cat << EOF | tee ca-csr.json  {  "CN": "kubernetes",  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "L": "Beijing",  "ST": "Beijing",  "O": "k8s",  "OU": "System"  }  ]  }  EOF |

**5.3.3 生成API SERVER证书申请文件（Master-1）**

**#注意要修改VIP的地址**

|  |
| --- |
| [root@master-1 ~]# cat << EOF | tee server-csr.json  {  "CN": "kubernetes",  "hosts": [  "10.0.0.1",  "127.0.0.1",  "10.0.0.2",  "192.168.91.18",  "192.168.91.19",  "192.168.91.20",  "192.168.91.21",  "192.168.91.22",  "192.168.91.254",  "master-1",  "master-2",  "master-3",  "node-1",  "node-2",  "kubernetes",  "kubernetes.default",  "kubernetes.default.svc",  "kubernetes.default.svc.cluster",  "kubernetes.default.svc.cluster.local"  ],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "L": "Beijing",  "ST": "Beijing",  "O": "k8s",  "OU": "System"  }  ]  }  EOF |

**5.3.4 创建 Kubernetes Proxy 证书申请文件（Master-1）**

|  |
| --- |
| [root@master-1 ~]# cat << EOF | tee kube-proxy-csr.json  {  "CN": "system:kube-proxy",  "hosts": [],  "key": {  "algo": "rsa",  "size": 2048  },  "names": [  {  "C": "CN",  "L": "Beijing",  "ST": "Beijing",  "O": "k8s",  "OU": "System"  }  ]  }  EOF |

**5.3.5 生成 kubernetes CA 证书和公私钥**

|  |
| --- |
| # 生成ca证书（Master-1）  [root@master-1 ~]# cfssl gencert -initca ca-csr.json | cfssljson -bare ca –    # 生成 api-server 证书（Master-1）  [root@master-1 ~]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server    # 生成 kube-proxy 证书（Master-1）  [root@master-1 ~]# cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json \  -profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy |

**6、部署ETCD**

|  |
| --- |
| #下载etcd二进制安装文件（所有master）  [root@master-1 ~]# mkdir -p /soft && cd /soft  [root@master-1~]#wget <https://github.com/etcd-io/etcd/releases/download/v3.3.10/etcd-v3.3.10-linux-amd64.tar.gz>  [root@master-1 ~]# tar -xvf etcd-v3.3.10-linux-amd64.tar.gz  [root@master-1 ~]# cd etcd-v3.3.10-linux-amd64/  [root@master-1 ~]# cp etcd etcdctl /usr/local/bin/ |

**6.1 编辑etcd配置文件（所有master）**

#注意修改每个节点的ETCD\_NAME

#注意修改每个节点的监听地址

|  |
| --- |
| [root@master-1 ~]# mkdir -p /etc/etcd/{cfg,ssl} （所有master）  [root@master-1 ~]# cat >/etc/etcd/cfg/etcd.conf<<EOFL  #[Member]  ETCD\_NAME="master-1"  ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"  ETCD\_LISTEN\_PEER\_URLS="https://192.168.91.18:2380"  ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.91.18:2379,http://192.168.91.18:2390"  #[Clustering]  ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.91.18:2380"  ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.91.18:2379"  ETCD\_INITIAL\_CLUSTER="master-1=https://192.168.91.18:2380,master-2=https://192.168.91.19:2380,master-3=https://192.168.91.20:2380"  ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  ETCD\_INITIAL\_CLUSTER\_STATE="new"  EOFL |

|  |
| --- |
| [root@master-2 ~]#cat >/etc/etcd/cfg/etcd.conf<<EOFL  #[Member]  ETCD\_NAME="master-2"  ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"  ETCD\_LISTEN\_PEER\_URLS="https://192.168.91.19:2380"  ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.91.19:2379,http://192.168.91.19:2390"  #[Clustering]  ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.91.19:2380"  ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.91.19:2379"  ETCD\_INITIAL\_CLUSTER="master-1=https://192.168.91.18:2380,master-2=https://192.168.91.19:2380,master-3=https://192.168.91.20:2380"  ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  ETCD\_INITIAL\_CLUSTER\_STATE="new"  EOFL |

|  |
| --- |
| [root@master-3 ~]#cat >/etc/etcd/cfg/etcd.conf<<EOFL  #[Member]  ETCD\_NAME="master-3"  ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"  ETCD\_LISTEN\_PEER\_URLS="https://192.168.91.20:2380"  ETCD\_LISTEN\_CLIENT\_URLS="https://192.168.91.20:2379,http://192.168.91.20:2390"  #[Clustering]  ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="https://192.168.91.20:2380"  ETCD\_ADVERTISE\_CLIENT\_URLS="https://192.168.91.20:2379"  ETCD\_INITIAL\_CLUSTER="master-1=https://192.168.91.18:2380,master-2=https://192.168.91.19:2380,master-3=https://192.168.91.20:2380"  ETCD\_INITIAL\_CLUSTER\_TOKEN="etcd-cluster"  ETCD\_INITIAL\_CLUSTER\_STATE="new"  EOFL |

**6.2创建ETCD的系统启动服务（所有master）**

**不用改，直接复制**

|  |
| --- |
| [root@master-1 ~]# cat > /usr/lib/systemd/system/etcd.service<<EOFL  [Unit]  Description=Etcd Server  After=network.target  After=network-online.target  Wants=network-online.target  [Service]  Type=notify  EnvironmentFile=/etc/etcd/cfg/etcd.conf  ExecStart=/usr/local/bin/etcd \  --name=\${ETCD\_NAME} \  --data-dir=\${ETCD\_DATA\_DIR} \  --listen-peer-urls=\${ETCD\_LISTEN\_PEER\_URLS} \  --listen-client-urls=\${ETCD\_LISTEN\_CLIENT\_URLS},http://127.0.0.1:2379 \  --advertise-client-urls=\${ETCD\_ADVERTISE\_CLIENT\_URLS} \  --initial-advertise-peer-urls=\${ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS} \  --initial-cluster=\${ETCD\_INITIAL\_CLUSTER} \  --initial-cluster-token=\${ETCD\_INITIAL\_CLUSTER\_TOKEN} \  --initial-cluster-state=new \  --cert-file=/etc/etcd/ssl/server.pem \  --key-file=/etc/etcd/ssl/server-key.pem \  --peer-cert-file=/etc/etcd/ssl/server.pem \  --peer-key-file=/etc/etcd/ssl/server-key.pem \  --trusted-ca-file=/etc/etcd/ssl/ca.pem \  --peer-trusted-ca-file=/etc/etcd/ssl/ca.pem  Restart=on-failure  LimitNOFILE=65536  [Install]  WantedBy=multi-user.target  EOFL |

**6.3 复制etcd证书到指定目录（master-1）**

|  |
| --- |
| [root@master-1 ~]# mkdir -p /etc/etcd/ssl/  [root@master-1 ~]# \cp /root/etcd/\*pem /etc/etcd/ssl/ -rf  #复制etcd证书到每个节点  [root@master-1 ~]# for i in master-2 master-3 node-1 node-2;do ssh $i mkdir -p /etc/etcd/{cfg,ssl};done  [root@master-1 ~]# for i in master-2 master-3 node-1 node-2;do scp /etc/etcd/ssl/\* $i:/etc/etcd/ssl/;done  [root@master-1 ~]# for i in master-2 master-3 node-1 node-2;do echo $i "------>"; ssh $i ls /etc/etcd/ssl;done |

**6.4 启动etcd (所有节点)**

|  |
| --- |
| [root@master-1 ~]# chkconfig etcd on  [root@master-1 ~]# service etcd start  [root@master-1 ~]# service etcd status |

**6.5 检查etcd 集群是否运行正常**

|  |
| --- |
| [root@master-1 ~]# etcdctl --ca-file=/etc/etcd/ssl/ca.pem --cert-file=/etc/etcd/ssl/server.pem \  --key-file=/etc/etcd/ssl/server-key.pem --endpoints="https://192.168.91.18:2379" cluster-health |

**6.6 创建Docker所需分配POD 网段 (任意master节点)**

|  |
| --- |
| [root@master-1]# etcdctl --ca-file=/etc/etcd/ssl/ca.pem \  --cert-file=/etc/etcd/ssl/server.pem --key-file=/etc/etcd/ssl/server-key.pem \  --endpoints="https://192.168.91.18:2379,https://192.168.91.19:2379,https://192.168.91.20:2379" \  set /coreos.com/network/config \  '{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}'  **#检查是否建立网段**  [root@master-1]# etcdctl \  --endpoints=https://192.168.91.18:2379,https://192.168.91.19:2379,https://192.168.91.20:2379 \  --ca-file=/etc/etcd/ssl/ca.pem \  --cert-file=/etc/etcd/ssl/server.pem \  --key-file=/etc/etcd/ssl/server-key.pem \  get /coreos.com/network/config |

**7、安装Docker**

|  |
| --- |
| **#在所有的Node节点安装**  #安装CE版本  [root@node-1 ~]# yum install -y yum-utils device-mapper-persistent-data lvm2  [root@node-1 ~]# yum-config-manager --add-repo <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>  [root@node-1 ~]# yum install -y docker-ce-19.03.6 docker-ce-cli-19.03.6 containerd.io |

**7.1 启动Docker服务**

|  |
| --- |
| [root@node-1 ~]# chkconfig docker on  [root@node-1 ~]# service docker start  [root@node-1 ~]# service docker status |

**7.2 配置镜像加速器(所有node节点)**

|  |
| --- |
| [root@node-1 ~]# mkdir -p /etc/docker  [root@node-1 ~]# tee /etc/docker/daemon.json <<-'EOF'  {  "registry-mirrors": ["https://lb3cvacp.mirror.aliyuncs.com"]  }  EOF  [root@node-1 ~]# systemctl daemon-reload  [root@ node-1 ~]# systemctl restart docker |

**8、部署Flannel**

**8.1 下载Flannel二进制包**

|  |
| --- |
| **#所有的节点,下载到master-1**  [root@master-1 ~]# mkdir /soft ; cd /soft  [root@master-1]#wget <https://github.com/coreos/flannel/releases/download/v0.11.0/flannel-v0.11.0-linux-amd64.tar.gz>  [root@master-1 ~]# tar xvf flannel-v0.11.0-linux-amd64.tar.gz  [root@master-1 ~]# mv flanneld mk-docker-opts.sh /usr/local/bin/  **#复制flanneld到其他的所有节点**  [root@master-1 ~]# for i in master-2 master-3 node-1 node-2;do scp /usr/local/bin/flanneld $i:/usr/local/bin/;done  [root@master-1~]# for i in master-2 master-3 node-1 node-2;do scp /usr/local/bin/mk-docker-opts.sh $i:/usr/local/bin/;done |

**8.2 配置Flannel (所有节点)**

|  |
| --- |
| [root@master-1 ~]# mkdir -p /etc/flannel  [root@master-1 ~]# cat > /etc/flannel/flannel.cfg<<EOF  FLANNEL\_OPTIONS="-etcd-endpoints=https://192.168.91.18:2379,https://192.168.91.19:2379,https://192.168.91.20:2379 -etcd-cafile=/etc/etcd/ssl/ca.pem -etcd-certfile=/etc/etcd/ssl/server.pem -etcd-keyfile=/etc/etcd/ssl/server-key.pem"  EOF  #多个ETCD: -etcd-endpoints=https://192.168.91.200:2379,https://192.168.91.201:2379,https://192.168.91.202:2379 |

**8.3 配置Flannel配置文件 (所有节点)**

|  |
| --- |
| [root@master-1 ~]# cat > /usr/lib/systemd/system/flanneld.service <<EOF  [Unit]  Description=Flanneld overlay address etcd agent  After=network-online.target network.target  Before=docker.service  [Service]  Type=notify  EnvironmentFile=/etc/flannel/flannel.cfg  ExecStart=/usr/local/bin/flanneld --ip-masq \$FLANNEL\_OPTIONS  ExecStartPost=/usr/local/bin/mk-docker-opts.sh -k DOCKER\_NETWORK\_OPTIONS -d /run/flannel/subnet.env  Restart=on-failure  [Install]  WantedBy=multi-user.target  EOF |

**8.4 启动Flannel(所有节点)**

|  |
| --- |
| [root@master-1 ~]#systemctl start flanneld  [root@master-1 ~]#systemctl enable flanneld  [root@master-1~]# systemctl status flanneld    #所有的节点都需要有172.17.0.0/16 网段IP  [root@master-1 soft]# ip a | grep flannel  3: flannel.1: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue state UNKNOWN group default  inet 172.17.41.0/32 scope global flannel.1  #node节点停止flanneld  [root@node-1 ~]# systemctl start flanneld |

**8.5 修改Docker启动文件（node节点）**

|  |
| --- |
| [root@node-1 ~]# cat >/usr/lib/systemd/system/docker.service<<EOFL  [Unit]  Description=Docker Application Container Engine  Documentation=https://docs.docker.com  After=network-online.target firewalld.service  Wants=network-online.target  [Service]  Type=notify  EnvironmentFile=/run/flannel/subnet.env  ExecStart=/usr/bin/dockerd \$DOCKER\_NETWORK\_OPTIONS  ExecReload=/bin/kill -s HUP \$MAINPID  LimitNOFILE=infinity  LimitNPROC=infinity  LimitCORE=infinity  TimeoutStartSec=0  Delegate=yes  KillMode=process  Restart=on-failure  StartLimitBurst=3  StartLimitInterval=60s  [Install]  WantedBy=multi-user.target  EOFL |

**8.6 重启Docker服务(node节点)**

|  |
| --- |
| [root@node-1 ~]# systemctl daemon-reload  [root@node-1 ~]# systemctl restart flanneld  [root@node-1 ~]#systemctl restart docker  #检查IP地址, docker 与flanneld 是同一个网段  [root@node-1 ~]# ip a    [root@node-2 ]#ip a |

**8.7 Node 节点验证是否可以访问其他节点Docker0**

#任意一台master Ping其他node节点, 网段都是通的。

|  |
| --- |
| [root@master-1 ]#ping 172.17.19.1 |

**9、安装Master 组件**

|  |
| --- |
| #Master端需要安装的组件如下:  kube-apiserver  kube-scheduler  kube-controller-manager |

**9.1 安装Api Server服务**

**9.1.1 下载Kubernetes二进制包(1.15.1)（master-1）**

|  |
| --- |
| [root@master-1 soft]# cd /soft  [root@master-1 soft]# tar xvf kubernetes-server-linux-amd64-v1.15.2.tar.gz  [root@master-1 soft]# cd kubernetes/server/bin/  [root@master-1 soft]# cp kube-scheduler kube-apiserver kube-controller-manager kubectl /usr/local/bin/ |

#复制执行文件到其他的master节点

[root@master-1 bin]# for i in master-2 master-3;do scp /usr/local/bin/kube\* $i:/usr/local/bin/;done

**9.1.2 配置Kubernetes证书**

|  |
| --- |
| #Kubernetes各个组件之间通信需要证书,需要复制个每个master节点（master-1）  [root@master-1 soft]#mkdir -p /etc/kubernetes/{cfg,ssl}  [root@master-1 soft]#cp /root/kubernetes/\*.pem /etc/kubernetes/ssl/  #复制到其他的节点  [root@master-1 soft]# for i in master-2 master-3 node-1 node-2;do ssh $i mkdir -p /etc/kubernetes/{cfg,ssl};done  [root@master-1 soft]# for i in master-2 master-3 node-1 node-2;do scp /etc/kubernetes/ssl/\* $i:/etc/kubernetes/ssl/;done  [root@master-1 bin]# for i in master-2 master-3 node-1 node-2;do echo $i "---------->"; ssh $i ls /etc/kubernetes/ssl;done |

**9.1.3忽略不做(跳过，直接9.1.4)**

**9.1.3 创建 TLS Bootstrapping Token**

|  |
| --- |
| # TLS bootstrapping 功能就是让 kubelet 先使用一个预定的低权限用户连接到 apiserver，  然后向 apiserver 申请证书，kubelet 的证书由 apiserver 动态签署  #Token可以是任意的包涵128 bit的字符串，可以使用安全的随机数发生器生成  [root@master-1]# head -c 16 /dev/urandom | od -An -t x | tr -d ' '  f89a76f197526a0d4bc2bf9c86e871c3 |

**9.1.4 编辑Token 文件(master-1)**

#f89a76f197526a0d4bc2bf9c86e871c3:随机字符串,自定义生成; kubelet-bootstrap:用户名; 10001:UID; system:kubelet-bootstrap：用户组

|  |
| --- |
| [root@master-1 soft]# vim /etc/kubernetes/cfg/token.csv  f89a76f197526a0d4bc2bf9c86e871c3,kubelet-bootstrap,10001,"system:kubelet-bootstrap"  **#复制到其他的master节点**  [root@master-1 bin]# for i in master-2 master-3;do scp /etc/kubernetes/cfg/token.csv $i:/etc/kubernetes/cfg/token.csv;done |

**9.1.5创建Apiserver配置文件(所有的master节点)**

#配置文件内容基本相同, 如果有多个节点, 那么需要修改IP地址即可

|  |
| --- |
| [root@master-1 soft]# cat >/etc/kubernetes/cfg/kube-apiserver.cfg <<EOFL  KUBE\_APISERVER\_OPTS="--logtostderr=true \  --v=4 \  --insecure-bind-address=0.0.0.0 \  --insecure-port=8080 \  --etcd-servers=https://192.168.91.18:2379,https://192.168.91.19:2379,https://192.168.91.20:2379 \  --bind-address=0.0.0.0 \  --secure-port=6443 \  --advertise-address=0.0.0.0 \  --allow-privileged=true \  --service-cluster-ip-range=10.0.0.0/24 \  --enable-admission-plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,ResourceQuota,NodeRestriction \  --authorization-mode=RBAC,Node \  --enable-bootstrap-token-auth \  --token-auth-file=/etc/kubernetes/cfg/token.csv \  --service-node-port-range=30000-50000 \  --tls-cert-file=/etc/kubernetes/ssl/server.pem \  --tls-private-key-file=/etc/kubernetes/ssl/server-key.pem \  --client-ca-file=/etc/kubernetes/ssl/ca.pem \  --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \  --etcd-cafile=/etc/etcd/ssl/ca.pem \  --etcd-certfile=/etc/etcd/ssl/server.pem \  --etcd-keyfile=/etc/etcd/ssl/server-key.pem"  EOFL |

#参数说明

--logtostderr 启用日志

---v 日志等级

--etcd-servers etcd 集群地址

--etcd-servers=https://192.168.91.200:2379,https://192.168.91.201:2379,https://192.168.91.202:2379

--bind-address 监听地址

--secure-port https 安全端口

--advertise-address 集群通告地址

--allow-privileged 启用授权

--service-cluster-ip-range Service 虚拟IP地址段

--enable-admission-plugins 准入控制模块

--authorization-mode 认证授权,启用RBAC授权

--enable-bootstrap-token-auth 启用TLS bootstrap功能

--token-auth-ﬁle token 文件

--service-node-port-range Service Node类型默认分配端口范围

**9.1.6 配置kube-apiserver 启动文件(所有的master节点)**

|  |
| --- |
| [root@master-1 soft]# cat >/usr/lib/systemd/system/kube-apiserver.service<<EOFL  [Unit]  Description=Kubernetes API Server  Documentation=https://github.com/kubernetes/kubernetes  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kube-apiserver.cfg  ExecStart=/usr/local/bin/kube-apiserver \$KUBE\_APISERVER\_OPTS  Restart=on-failure  [Install]  WantedBy=multi-user.target  EOFL |

**9.1.7 启动kube-apiserver服务**

|  |
| --- |
| [root@master-1 soft]# systemctl start kube-apiserver  [root@master-1 soft]# systemctl enable kube-apiserver  [root@master-1 soft]# systemctl status kube-apiserver    #查看加密的端口是否已经启动  [root@master-2 ~]# netstat -anltup | grep 6443    #查看加密的端口是否已经启动（node节点telnet）  [root@node-1 ~]# telnet 192.168.91.254 6443 |

**9.2 部署kube-scheduler 服务**

#创建kube-scheduler配置文件（所有的master节点）

|  |
| --- |
| [root@master-1 soft]# cat >/etc/kubernetes/cfg/kube-scheduler.cfg<<EOFL  KUBE\_SCHEDULER\_OPTS="--logtostderr=true --v=4 --bind-address=0.0.0.0 --master=127.0.0.1:8080 --leader-elect"  EOFL  #查看配置文件  [root@master-3 ~]# cat /etc/kubernetes/cfg/kube-scheduler.cfg |

**9.2.1 创建kube-scheduler 启动文件**

#创建kube-scheduler systemd unit 文件（所有的master节点）

|  |
| --- |
| [root@master-1 soft]# cat >/usr/lib/systemd/system/kube-scheduler.service<<EOFL  [Unit]  Description=Kubernetes Scheduler  Documentation=https://github.com/kubernetes/kubernetes  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kube-scheduler.cfg  ExecStart=/usr/local/bin/kube-scheduler \$KUBE\_SCHEDULER\_OPTS  Restart=on-failure  [Install]  WantedBy=multi-user.target  EOFL |

**9.2.2 启动kube-scheduler服务（所有的master节点）**

[root@master-1]#systemctl restart kube-scheduler

[root@master-1]#systemctl enable kube-scheduler

[root@master-1]#systemctl status kube-scheduler

**9.2.3查看Master节点组件状态（任意一台master）**

|  |
| --- |
| [root@master-1 bin]# kubectl get cs |

**9.3 部署kube-controller-manager**

**9.3.1创建kube-controller-manager配置文件(所有master节点)**

|  |
| --- |
| [root@master-1]# cat >/etc/kubernetes/cfg/kube-controller-manager.cfg<<EOFL  KUBE\_CONTROLLER\_MANAGER\_OPTS="--logtostderr=true \  --v=4 \  --master=127.0.0.1:8080 \  --leader-elect=true \  --address=0.0.0.0 \  --service-cluster-ip-range=10.0.0.0/24 \  --cluster-name=kubernetes \  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \  --root-ca-file=/etc/kubernetes/ssl/ca.pem \  --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem"  EOFL |

#参数说明

--master=127.0.0.1:8080 #指定Master地址

--leader-elect #竞争选举机制产生一个 leader 节点，其它节点为阻塞状态。

--service-cluster-ip-range #kubernetes service 指定的IP地址范围。

**9.3.2 创建kube-controller-manager 启动文件(所有master节点)**

[root@master-1]#

|  |
| --- |
| cat >/usr/lib/systemd/system/kube-controller-manager.service<<EOFL  [Unit]  Description=Kubernetes Controller Manager  Documentation=https://github.com/kubernetes/kubernetes  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kube-controller-manager.cfg  ExecStart=/usr/local/bin/kube-controller-manager \$KUBE\_CONTROLLER\_MANAGER\_OPTS  Restart=on-failure  [Install]  WantedBy=multi-user.target  EOFL |

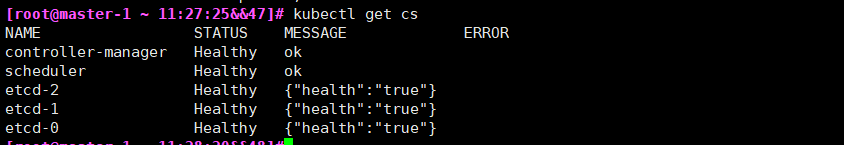
**7.3.3启动kube-controller-manager服务(所有master节点)**

|  |
| --- |
| [root@master-1]# systemctl enable kube-controller-manager  [root@master-1]# systemctl start kube-controller-manager  [root@master-1]# systemctl status kube-controller-manager |

**9.4 查看Master 节点组件状态**

#必须要在各个节点组件正常的情况下, 才去部署Node节点组件.（master节点）

[root@master-1 bin]# kubectl get cs



**10、部署Node节点组件**

**10.1部署 kubelet 组件**

**10.1.1 从Master节点复制Kubernetes 文件到Node**

#配置Node节点

|  |
| --- |
| [root@master-1 ]#cd /soft  [root@master-1]#for i in node-1 node-2;do scp kubernetes/server/bin/kubelet kubernetes/server/bin/kube-proxy $i:/usr/local/bin/;done |

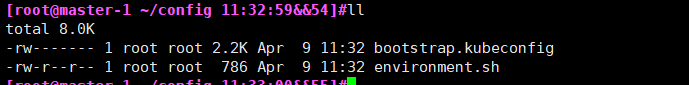
**10.1.2 创建kubelet bootstrap.kubeconfig 文件**

#Maste-1节点 192.168.91.254如果是其他地址就要改，如公有云地址

|  |
| --- |
| [root@master-1 bin]# mkdir /root/config ; cd /root/config  [root@master-1 bin]# cat >environment.sh<<EOFL  # 创建kubelet bootstrapping kubeconfig  BOOTSTRAP\_TOKEN=f89a76f197526a0d4bc2bf9c86e871c3  KUBE\_APISERVER="https://192.168.91.254:6443"  # 设置集群参数  kubectl config set-cluster kubernetes \  --certificate-authority=/etc/kubernetes/ssl/ca.pem \  --embed-certs=true \  --server=\${KUBE\_APISERVER} \  --kubeconfig=bootstrap.kubeconfig  # 设置客户端认证参数  kubectl config set-credentials kubelet-bootstrap \  --token=\${BOOTSTRAP\_TOKEN} \  --kubeconfig=bootstrap.kubeconfig  # 设置上下文参数  kubectl config set-context default \  --cluster=kubernetes \  --user=kubelet-bootstrap \  --kubeconfig=bootstrap.kubeconfig  # 设置默认上下文  kubectl config use-context default --kubeconfig=bootstrap.kubeconfig  #通过 bash environment.sh获取 bootstrap.kubeconfig 配置文件。  EOFL |

#执行脚本

[root@master-1 bin]# sh environment.sh



**10.1.3创建kube-proxy kubeconfig文件 （master-1）**

|  |
| --- |
| [root@master-1 bin]# cat >env\_proxy.sh<<EOF  # 创建kube-proxy kubeconfig文件  BOOTSTRAP\_TOKEN=f89a76f197526a0d4bc2bf9c86e871c3  KUBE\_APISERVER="https://192.168.91.254:6443"  kubectl config set-cluster kubernetes \  --certificate-authority=/etc/kubernetes/ssl/ca.pem \  --embed-certs=true \  --server=\${KUBE\_APISERVER} \  --kubeconfig=kube-proxy.kubeconfig  kubectl config set-credentials kube-proxy \  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \  --embed-certs=true \  --kubeconfig=kube-proxy.kubeconfig  kubectl config set-context default \  --cluster=kubernetes \  --user=kube-proxy \  --kubeconfig=kube-proxy.kubeconfig  kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig  EOF  #执行脚本  [root@master-1 bin]# sh env\_proxy.sh |

**10.1.4 复制kubeconfig文件与证书到所有Node节点**

|  |
| --- |
| #将bootstrap kubeconfig kube-proxy.kubeconfig 文件复制到所有Node节点  #**远程创建目录** (master-1)  [root@master-1]# for i in node-1 node-2;do ssh $i "mkdir -p /etc/kubernetes/{cfg,ssl}";done  #**复制证书文件ssl** (master-1)  [root@master-1 config]# for i in node-1 node-2;do scp /etc/kubernetes/ssl/\* $i:/etc/kubernetes/ssl/;done  #**复制kubeconfig文件** (master-1)  [root@master-1]# cd /root/config  [root@master-1 config]# for i in node-1 node-2;do scp -rp bootstrap.kubeconfig kube-proxy.kubeconfig $i:/etc/kubernetes/cfg/;done |

**10.1.5 创建kubelet参数配置文件**

**#不同的Node节点, 需要修改IP地址 （node节点操作）**

|  |
| --- |
| 注意节点IP不同要修改  [root@ node-1]#cat >/etc/kubernetes/cfg/kubelet.config<<EOF  kind: KubeletConfiguration  apiVersion: kubelet.config.k8s.io/v1beta1  address: 192.168.91.21  port: 10250  readOnlyPort: 10255  cgroupDriver: cgroupfs  clusterDNS: ["10.0.0.2"]  clusterDomain: cluster.local.  failSwapOn: false  authentication:  anonymous:  enabled: true  EOF  [root@ node-2]#cat >/etc/kubernetes/cfg/kubelet.config<<EOF  kind: KubeletConfiguration  apiVersion: kubelet.config.k8s.io/v1beta1  address: 192.168.91.22  port: 10250  readOnlyPort: 10255  cgroupDriver: cgroupfs  clusterDNS: ["10.0.0.2"]  clusterDomain: cluster.local.  failSwapOn: false  authentication:  anonymous:  enabled: true  EOF |

**10.1.6 创建kubelet配置文件**

#不同的Node节点, 需要修改IP地址

#/etc/kubernetes/cfg/kubelet.kubeconfig 文件自动生成

|  |
| --- |
| [root@node-1]#cat >/etc/kubernetes/cfg/kubelet<<EOF  KUBELET\_OPTS="--logtostderr=true \  --v=4 \  --hostname-override=192.168.91.21 \  --kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \  --bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \  --config=/etc/kubernetes/cfg/kubelet.config \  --cert-dir=/etc/kubernetes/ssl \  --pod-infra-container-image=docker.io/kubernetes/pause:latest"  EOF  [root@node-1]#cat >/etc/kubernetes/cfg/kubelet<<EOF  KUBELET\_OPTS="--logtostderr=true \  --v=4 \  --hostname-override=192.168.91.22 \  --kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \  --bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \  --config=/etc/kubernetes/cfg/kubelet.config \  --cert-dir=/etc/kubernetes/ssl \  --pod-infra-container-image=docker.io/kubernetes/pause:latest"  EOF |

**10.1.7 创建kubelet系统启动文件(node节点)**

|  |
| --- |
| [root@node-1 bin]#cat >/usr/lib/systemd/system/kubelet.service<<EOF  [Unit]  Description=Kubernetes Kubelet  After=docker.service  Requires=docker.service  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kubelet  ExecStart=/usr/local/bin/kubelet \$KUBELET\_OPTS  Restart=on-failure  KillMode=process  [Install]  WantedBy=multi-user.target  EOF |

**10.1.8 将kubelet-bootstrap用户绑定到系统集群角色**

#master-1节点操作（没有路径要求，就是创建个权限）

|  |
| --- |
| [root@master-1 bin]#kubectl create clusterrolebinding kubelet-bootstrap \  --clusterrole=system:node-bootstrapper \  --user=kubelet-bootstrap |

**10.1.9 启动kubelet服务（node节点）**

|  |
| --- |
| [root@node-1]#systemctl enable kubelet  [root@node-1]#systemctl start kubelet  [root@node-1]#systemctl status kubelet |

**10.2 服务端批准与查看CSR请求**

#查看CSR请求

#Maste-1节点操作

|  |
| --- |
| [root@master1 cfg]# kubectl get csr |

**10.2.1 批准请求**

|  |
| --- |
| #Master节点操作  [root@master-1 bin]#kubectl certificate approve node-csr-opN3A4mqx6e2AnIXcKOF8GGgs\_tMYmiMzLg12HRyCd0  [root@master-1 bin]#kubectl certificate approve node-csr-sWcxOcEKKR7WW3D9yFD-z-ZNnK8Nk4\_0SdcpPtuGqxc |

**10.2.3 查看节点状态**

|  |
| --- |
| #所有的Node节点状态必须为Ready （master查询）  [root@master-1]#kubectl get nodes |

**10.3 节点重名处理--没有重名，此步骤忽略跳过。只是个处理方式**

|  |
| --- |
| #如果出现节点重名, 可以先删除证书, 然后重新申请  #Master节点删除csr  [root@master-1 bin]# kubectl delete csr node-csr-U4v31mc3j\_xPq5n1rU2KdpyugqfFH\_0g1wOC66oiu04  #Node节点删除kubelet.kubeconfig  #客户端重启kubelet服务, 再重新申请证书  [root@node-1 bin]#rm -rf /etc/kubernetes/cfg/kubelet.kubeconfig |

**10.4 部署kube-proxy 组件**

# kube-proxy 运行在所有Node节点上, 监听Apiserver 中 Service 和 Endpoint 的变化情况，创建路由规则来进行服务负载均衡。

**10.4.1 创建kube-proxy配置文件（node节点操作）**

#注意修改hostname-override地址, 不同的节点则不同。

|  |
| --- |
| [root@node-1 ~]#cat >/etc/kubernetes/cfg/kube-proxy<<EOF  KUBE\_PROXY\_OPTS="--logtostderr=true \  --v=4 \  --metrics-bind-address=0.0.0.0 \  --hostname-override=192.168.91.21 \  --cluster-cidr=10.0.0.0/24 \  --kubeconfig=/etc/kubernetes/cfg/kube-proxy.kubeconfig"  EOF  [root@node-2 ~]#cat >/etc/kubernetes/cfg/kube-proxy<<EOF  KUBE\_PROXY\_OPTS="--logtostderr=true \  --v=4 \  --metrics-bind-address=0.0.0.0 \  --hostname-override=192.168.91.22 \  --cluster-cidr=10.0.0.0/24 \  --kubeconfig=/etc/kubernetes/cfg/kube-proxy.kubeconfig"  EOF |

**10.4.2 创建kube-proxy systemd unit 文件（node节点操作）**

|  |
| --- |
| [root@node-1 ~]#cat >/usr/lib/systemd/system/kube-proxy.service<<EOF  [Unit]  Description=Kubernetes Proxy  After=network.target  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kube-proxy  ExecStart=/usr/local/bin/kube-proxy \$KUBE\_PROXY\_OPTS  Restart=on-failure  [Install]  WantedBy=multi-user.target  EOF |

**10.4.3 启动kube-proxy 服务**

|  |
| --- |
| [root@node-1 ~]#systemctl enable kube-proxy  [root@node-1 ~]#systemctl start kube-proxy  [root@node-1 ~]#systemctl status kube-proxy |

**11、运行Demo项目(2个node节点先拉取镜像 docker pull nginx)**

|  |
| --- |
| [root@master-1 soft]# kubectl run nginx --image=nginx --replicas=2    #获取容器IP与运行节点  [root@master-1 ~]# kubectl get pods -o wide    #创建容器svc端口  [root@master-1 ~]#kubectl expose deployment nginx --port=88 --target-port=80 --type=NodePort  #查看容器状态  [root@master-1 ~]# kubectl describe pod nginx-7bb7cd8db5-g7ms2 |

**11.1 查看SVC**

|  |
| --- |
| [root@master-1 cfg]# kubectl get svc |

**11.2 访问web**

|  |
| --- |
| [root@master-1 cfg]# curl <http://192.168.91.21:43404> |



**11.3 删除项目**

|  |
| --- |
| [root@master-1 cfg]# kubectl delete deployment nginx  [root@master-1 cfg]# kubectl delete pods nginx  [root@master-1 cfg]# kubectl delete svc -l run=nginx  [root@master-1 cfg]# kubectl delete deployment.apps/nginx |

**11.5 服务启动顺序**

**11.5.1 启动Master节点**

[root@master-1 cfg]# service keepalived start

[root@master-1 cfg]# service etcd start

[root@master-1 cfg]# service kube-scheduler start

[root@master-1 cfg]# service kube-controller-manager start

[root@master-1 ~]# **service kube-apiserver restart**

[root@master-1 cfg]# **kubectl get cs**

**11.5.2 启动Node节点**

[root@node-1 cfg]# **service flanneld start**

[root@node-1 cfg]# **service docker start**

[root@node-1 cfg]# **service kubelet start**

[root@node-1 cfg]# **service kube-proxy start**

**11.5.3 停止Node节点**

[root@node-1 cfg]# **service kubelet stop**

[root@node-1 cfg]# **service kube-proxy stop**

[root@node-1 cfg]# **service docker stop**

[root@node-1 cfg]# **service flanneld stop**

**11.5.4 停止Master 节点**

[root@master-1 cfg]# **service kube-controller-manager stop**

[root@master-1 cfg]# **service kube-scheduler stop**

[root@master-1 cfg]# **service etcd stop**

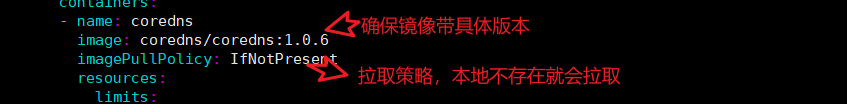
[root@master-1 cfg]# **service keepalived stop**

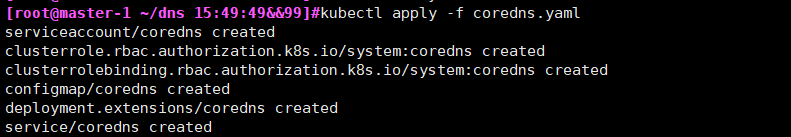
**12 部署DNS**

**12.1 部署coredns**

#2个node节点先拉取镜像

|  |
| --- |
| [root@node-1 ~ 22:53:32&&2]#**docker pull coredns/coredns:1.0.6**  [root@node-2 ~ 22:54:05&&2]#**docker pull coredns/coredns:1.0.6**  [root@master-1 cfg]# **mkdir /root/dns && cd /root/dns**  [root@master-1 cfg]# **kubectl apply -f coredns.yaml** |





|  |
| --- |
| #查询所有ns中的pod  [root@master-1 dns]# **kubectl get pod -A**  #查询指定ns中的pod  [root@master-1 dns]# **kubectl get pod -n kube-system**    #查看启动进程  [root@master-1 dns]# **kubectl describe pod coredns-66db855d4d-26bvw -n kube-system** |

**12.2 查看SVC**

|  |
| --- |
| [root@master1 kubernetes]# **kubectl get svc -o wide -n=kube-system** |

**12.3 验证DNS是否有效**

**12.3.1 删除之前创建的nginx demo**

|  |
| --- |
| [root@master-1]#**kubectl delete deployment nginx**  [root@master-1]#**kubectl delete pods nginx**  [root@master-1]#**kubectl delete svc -l run=nginx**  [root@master-1#**kubectl delete deployment.apps/nginx** |

**12.3.2 启动新容器**

[root@master-1]# **kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools**

|  |
| --- |
| #出现错误    Error from server (Forbidden): Forbidden (user=system:anonymous, verb=get, resource=nodes, subresource=proxy) ( pods/log dnstools)  **#解决方法**  [root@master-1]# **kubectl create clusterrolebinding system:anonymous --clusterrole=cluster-admin --user=system:anonymous** |

|  |
| --- |
| [root@master-1]# **kubectl delete pod dnstools**  [root@master-1]# **kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools** |

**12.3.2 创建Nginx 容器**

**另开一个master1窗口**

|  |
| --- |
| [root@master-1 ~]# **kubectl run nginx --image=nginx --replicas=2**  #创建svc (cluster IP)  # Create a service for an nginx deployment, which serves on port 88 and connects to the containers on port 80.  #template format is golang templates [http://golang.org/pkg/text/template/#pkg-overview].  --type='': Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'. |

[root@master-1 ~]# **kubectl expose deployment nginx --port=88 --target-port=80 --type=NodePort**

**12.3.3 查看SVC**

|  |
| --- |
| [root@master-1 ~]# **kubectl get svc** |

**12.3.4 测试解析Nginx**

#测试解析nginx

#dns 解析的名称是svc (service 名称, 非pod名称)

|  |
| --- |
| dnstools# **nslookup nginx(**回到创建新容器的窗口**)**    dnstools# **curl 10.0.0.243:88** |

**12.3.5 案例:容器的网络访问不区分命名空间（kubernetes ns）**

**#在default ns 可以访问到kube-system ns 服务nginx**

|  |
| --- |
| [root@master-1 ~]# **kubectl run nginx-n1 --image=nginx --replicas=1 -n kube-system**  # Create a service for an nginx deployment, which serves on port 99 and connects to the containers on port 80.  #查看容器状态（指定命名空间）  [root@master-3 ~]# **kubectl get pods -n kube-system** |

#查看容器状态（显示所有的命名空间）

|  |
| --- |
| [root@master-1 ~]# **kubectl get pod,svc -A** |

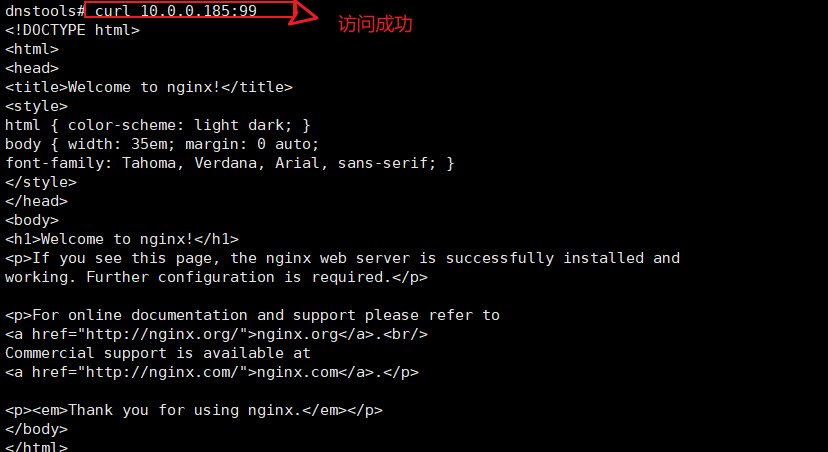
|  |
| --- |
| [root@master-1 ~]# **kubectl expose deployment nginx-n1 --port=99 --target-port=80 -n kube-system** |

**12.3.7 跨ns访问服务**

[root@master-1 dns]# **kubectl get svc -n kube-system | grep nginx-n1**



#访问服务



**12.3.8 #解析不成功**

|  |
| --- |
| dnstools# **nslookup nginx-n1**    #**解决方法**(默认解析为default空间)  dnstools# **nslookup nginx-n1.kube-system.svc.cluster.local** |

**13-1、部署Dashboard(原版)**

**13.1 下载文件**

|  |
| --- |
| [root@node-1]#**docker load -i kubernetes-dashboard-amd64\_v1.10.1.tar.gz**  [root@node-2 ]#**docker load -i kubernetes-dashboard-amd64\_v1.10.1.tar.gz** |

**13.1.1创建目录(master-1节点)**

|  |
| --- |
| [root@master-1 ~]# **mkdir /root/dashboard**  [root@master-1~]#**kubectl apply -f** [**https://raw.githubusercontent.com/kubernetes/dashboard/v1.8.1/src/deploy/recommended/kubernetes-dashboard.yaml**](https://raw.githubusercontent.com/kubernetes/dashboard/v1.8.1/src/deploy/recommended/kubernetes-dashboard.yaml) **（忽略）直接上传修改好的** |

**13.1.2 修改端口**

#修改为nodeport端口50000

#注意镜像地址无法下载, 使用另外的镜像替换

mirrorgooglecontainers/kubernetes-dashboard-amd64:v1.8.1

[root@master-1 ~]# **sed -i '/targetPort:/a\ \ \ \ \ \ nodePort: 30000\n\ \ type: NodePort' kubernetes-dashboard.1.10.yaml （忽略，直接上传修改好的）**

**13.1.3 部署**

|  |
| --- |
| [root@master-1 dashboard]# **kubectl apply -f kubernetes-dashboard.1.10.yaml** |

**13.1.4查看服务端口**

|  |
| --- |
| [root@master-1 dashboard]# **kubectl get services -n kube-system**    [root@master-1 dashboard]#**kubectl get pod -A** |

**13.1.5创建用户授权**

[root@master-1 dashboard]# **kubectl create serviceaccount dashboard-admin -n kube-system**



[root@master-1 dashboard]# **kubectl create clusterrolebinding \**

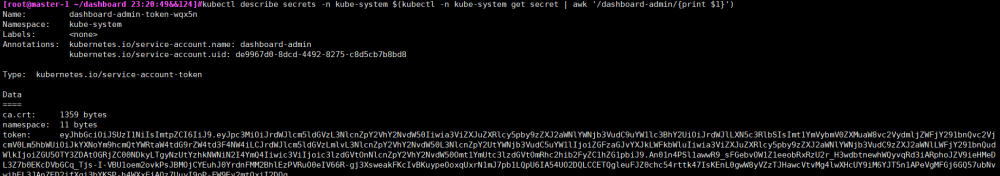
**dashboard-admin --clusterrole=cluster-admin --serviceaccount=kube-system:dashboard-admin**



**13.1.6获取Token**

#获取token

[root@master-1 ~]# **kubectl describe secrets -n kube-system $(kubectl -n kube-system get secret | awk '/dashboard-admin/{print $1}')**



**Yaml形式**

cat admin-user.yaml

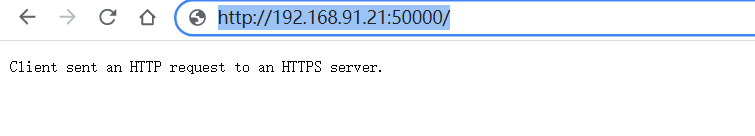
|  |
| --- |
| cat > admin-user.yaml << EOF  apiVersion: v1  kind: ServiceAccount  metadata:  name: admin-user  namespace: kubernetes-dashboard  ---  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRoleBinding  metadata:  name: admin-user  roleRef:  apiGroup: rbac.authorization.k8s.io  kind: ClusterRole  name: cluster-admin  subjects:  - kind: ServiceAccount  name: admin-user  namespace: kubernetes-dashboard  EOF |

#获取访问令牌

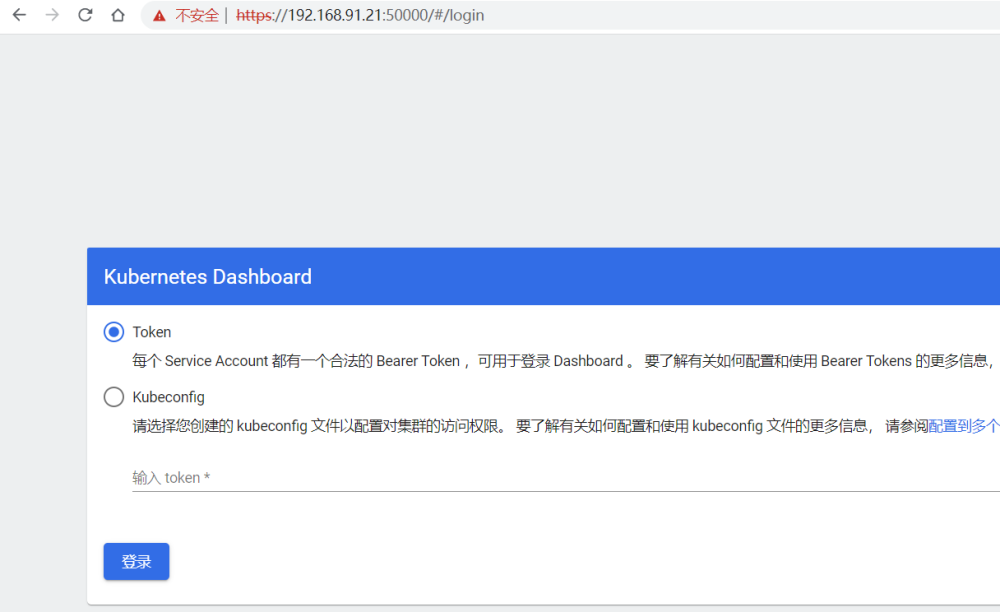
|  |
| --- |
| kubectl -n kubernetes-dashboard get secret $(kubectl -n kubernetes-dashboard get sa/admin-user -o jsonpath="{.secrets[0].name}") -o go-template="{{.data.token | base64decode}}" |

**13.1.7 登录系统**

#如果之前有安装过其他的版本的dashboard, 那么切换node节点IP访问



**改成https**



输入token

|  |
| --- |
| eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9..An01n4PSl1awwR9\_sFGebvOW1Z1eeobRxRzU2r\_H3wdbtnewhWQyvqRd3iARphoJZV9ieHMeDL3Z7b0EKcDVbGCq\_Tjs-I-VBU1oem2ovkPsJBMOjCYEuhJ0YrdnFMM2BhlEzPVRuO0eIV66R-gj3XsweakFKcIvBKuypeOoxqUxrN1mJ7pb1LQpU6IA54UO2DQLCCETQgleuFJZ0chc54rttk47IsKEnL0gwW8yVZzTJHawcVtvMg4lwXHcUY9iM6YJT5n1APeVgMFGj6GQ57ubNvwihEL3JAnZED2ifXgi3bYKSP-h4WXxFjAQz7UuyI9oP-FW9Ev2mtOxiI2DOg |

#创建虚拟机快照:Kubernetes Basic

**13-2、Dashboard部署（最新版）**

**13.1.1创建目录(master-1节点)**

|  |
| --- |
| [root@master-1 ~]# **mkdir /root/dashboard**  [root@master-1~]#**wget** [**https://raw.githubusercontent.com/kubernetes/dashboard/v2.5.1/aio/deploy/recommended.yaml**](https://raw.githubusercontent.com/kubernetes/dashboard/v2.5.1/aio/deploy/recommended.yaml) |

**13.1.2 修改端口**

[root@master-1 ~]# **sed -i '/targetPort:/a\ \ \ \ \ \ nodePort: 30000\n\ \ type: NodePort' recommended.yaml （忽略，直接上传修改好的）**



**13.1.3 部署**

|  |
| --- |
| [root@master-1]# **kubectl apply -f recommended.yaml**  **会报错**    删除加参数**--validate=fals**  [root@master-1]#**kubectl delete -f recommended.yaml**  [root@master-1]#**kubectl apply -f recommended.yaml --validate=false** |

**13.1.4查看服务端口**

|  |
| --- |
| [root@master-1 dashboard]# **kubectl get services -n kubernetes-dashboard**    [root@master-1 dashboard]#**kubectl get pod -A** |

**13.1.5-13.1.7部分与13-1一样不再写明**

**14、部署Ingress**

|  |
| --- |
| #服务反向代理  #部署Traefik 2.0版本  [root@node-1 ~ 00:03:44&&35]#**docker load -i traefik.2.0.5.tar.gz**  [root@node-2 ~ 00:03:44&&35]#**docker load -i traefik.2.0.5.tar.gz** |

**14.1创建 traefik-crd.yaml 文件 (master-1)**

[root@master-1 ~]# **mkdir /root/ingress && cd /root/ingress**

**14.1.1 创建Traefik CRD资源(master-1)**

|  |
| --- |
| [root@master-1 ~]# **kubectl create -f traefik-crd.yaml**    [root@master-1 ingress]# **kubectl get CustomResourceDefinition**    **创建没有指定命名空间** |

|  |
| --- |
| [root@master-1 ]#**kubectl get crd** |

**14.2 创建Traefik RABC文件(master-1)**

**14.2.1 创建RABC 资源**

|  |
| --- |
| [root@master-1 ingress]# **kubectl create -f traefik-rbac.yaml** |

**14.3 创建Traefik ConfigMap (master-1)**

**14.3.1 创建Traefik ConfigMap资源配置**

|  |
| --- |
| [root@master-1 ~]# **kubectl apply -f traefik-config.yaml -n kube-system**    [root@master-1]#**kubectl get configMap -n kube-system** |

**14.4 设置节点标签**

|  |
| --- |
| #设置节点label（**只做一个，不可能所有节点服务器都对外暴露端口**）  [root@master-1 ingress]# **kubectl label nodes 192.168.91.21 IngressProxy=true**    [root@master-1 ingress]# **kubectl label nodes 192.168.91.22 IngressProxy=true(**暂时不做) |

**14.4.1 查看节点标签**

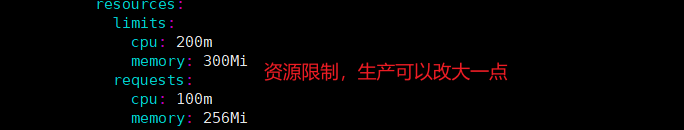
|  |
| --- |
| #检查是否成功  [root@master-1 ingress]# **kubectl get nodes --show-labels** |

**14.5 创建 traefik 部署文件**

#注意每个Node节点的80与443端口不能被占用

**（此处只做了一个节点node1,所以node1的80和443端口不能被占用）**

[root@node-1]#**netstat -antupl | grep -E "80|443"**



**14.5.1部署 Traefik 资源**

|  |
| --- |
| [root@master-1 ingress]# **kubectl apply -f traefik-deploy.yaml -n kube-system** |

**#查看运行状态**

|  |
| --- |
| [root@master-1 ingress]# **kubectl get DaemonSet -A** |

|  |
| --- |
| 错误问题:  #解决方法  #命名空间  Kubectl apply -f traefik-default-rbac.yaml  #错误  #解决方法  [root@master-1 ingress]# kubectl apply -f traefik-config.yaml  正常显示 |

**14.6 Traefik 路由配置**

**14.6.1 配置Traefik Dashboard**

**#创建Ingress (traefik)**

[root@master-1 ingress]# **kubectl apply -f traefik-dashboard-route.yaml**



[root@master-1]#**kubectl get ingressroute -A**



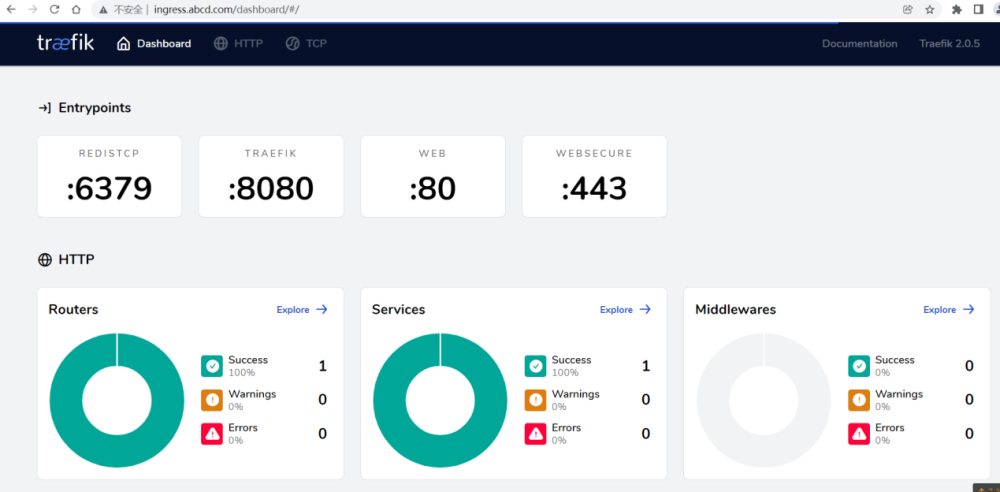
**14.6.2 客户端访问Traefik Dashboard**

**14.6.2.1 绑定物理主机Hosts文件或者域名解析**

|  |
| --- |
| C:\Windows\System32\drivers\etc  192.168.91.21 ingress.abcd.com (解析到设置标签的node) |

**14.6.2.2 访问web**

[**http://ingress.abcd.com**](http://ingress.abcd.com)



**14.7 部署访问服务(http)**

#创建nginx服务

[root@master-1 ingress]# **kubectl run nginx-ingress-demo1 --image=nginx --replicas=1 -n kube-system**

[root@master-1 ingress]# **kubectl expose deployment nginx-ingress-demo1 --port=1099 --target-port=80 -n kube-system**

#创建nginx路由服务

vim nginx-ingress-demo-route1.yaml

apiVersion: traefik.containo.us/v1alpha1

kind: IngressRoute

metadata:

name: traefik-nginx-demo-route1

namespace: kube-system

spec:

entryPoints:

- web

routes:

- match: Host(`nginx11.abcd.com`)

kind: Rule

services:

- name: nginx-ingress-demo1

port: 1099

#创建

[root@master-1 ingress]# kubectl apply -f nginx-ingress-demo-route1.yaml

[root@master-1 ingress]# kubectl get IngressRoute -A

NAMESPACE NAME AGE

default traefik-dashboard-route 48m

kube-system traefik-nginx-demo-route 68s

#访问

#绑定hosts (物理机器)

192.168.91.21 nginx11.abcd.com

**14.8 创建https服务**

#代理dashboard https 服务

# 创建自签名证书

[root@master-1 ingress]# cd /root/ingress

[root@master-1 ingress]# openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout tls.key -out tls.crt -subj "/CN=cloud.abcd.com"

#将证书存储到 Kubernetes Secret中

[root@master-1 ingress]# kubectl create secret tls dashboard-tls --key=tls.key --cert=tls.crt -n kube-system

#查看系统secret

[root@master-1 ingress]# kubectl get secret

NAME TYPE DATA AGE

default-token-l77nw kubernetes.io/service-account-token 3 6d22h

traefik-ingress-controller-token-pdbhn kubernetes.io/service-account-token 3 132m

#创建路由文件

#先查询kuberbentes dashboard 的命名空间

[root@master-1 ingress]# cat kubernetes-dashboard-route.yaml

#注意命名空间

apiVersion: traefik.containo.us/v1alpha1

kind: IngressRoute

metadata:

name: kubernetes-dashboard-route

namespace: kubernetes-dashboard

spec:

entryPoints:

- websecure

tls:

secretName: dashboard-tls

routes:

- match: Host(`cloud.abcd.com`)

kind: Rule

services:

- name: kubernetes-dashboard

port: 443

#创建 Kubernetes Dashboard 路由规则对象

[root@master-1 ingress]# kubectl create ns kubernetes-dashboard

[root@master-1 ingress]# kubectl apply -f kubernetes-dashboard-route.yaml

#查看创建的路由

[root@master-1 ingress]# kubectl get IngressRoute -A

NAMESPACE NAME AGE

default traefik-dashboard-route 125m

kube-system traefik-nginx-demo-route 77m

kube-system traefik-nginx-demo-route1 3m5s

kubernetes-dashboard kubernetes-dashboard-route 13s

#绑定hosts 访问

192.168.91.21 cloud.abcd.com

配置完成后，打开浏览器输入地址：<https://cloud.abcd.com>打开 Dashboard Dashboard。

**14.9 TCP服务访问**

**14.9.1配置redis文件**

|  |
| --- |
| [root@node-1 ~ 10:01:45&&2]#**docker pull redis**  [root@node-2 ~ 10:01:45&&2]#**docker pull redis** |

**14.9.2部署redis**

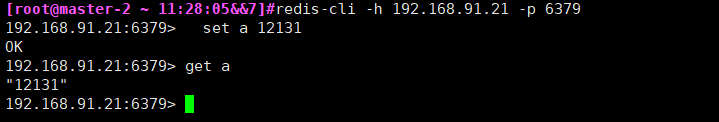
|  |
| --- |
| [root@master-1 ingress]# **kubectl apply -f redis-tcp-deploy.yaml** |

**14.9.3配置路由**

|  |
| --- |
| **#部署路由**  [root@master-1 ingress]# **kubectl apply -f traefik-redis-tcp-route.yaml**  **#查看界面**  <http://ingress.abcd.com/dashboard/#/tcp/routers> |

**14.9.4绑定任意主机名到node节点访问**

|  |
| --- |
| #192.168.91.21 redis.cc.com  [root@master-2 ~]# **redis-cli -h redis.cc.com -p 6379**  redis.cc.com:6379> set a 12131  OK  redis.cc.com:6379> get a  "12131" |



**14.10-1 （原版）部署Ingress Controler**

**使用的镜像image: quay.io/jcmoraisjr/haproxy-ingress**

**14.10.1导入镜像(所有node)**

[root@master-1 ~]# **docker load -i haproxy-ingress.tar.gz**

**14.10.2设置label**

[root@master-1 ~]# **kubectl label node 192.168.91.22 role=ingress-controller**

**14.10.3创建Ingress**

[root@master-1 ~]#**mkdir -p /root/haproxy && cd /root/haproxy**

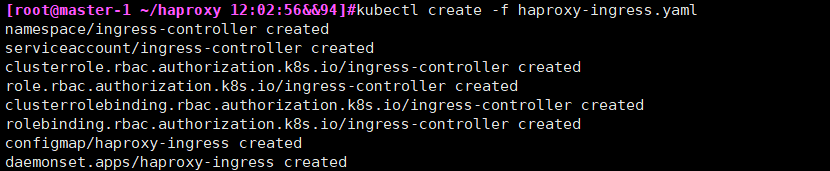
[root@master-1~]#**wget** [**https://haproxy-ingress.github.io/resources/haproxy-ingress.yaml**](https://haproxy-ingress.github.io/resources/haproxy-ingress.yaml)

[root@master-1~]vim **haproxy-ingress.yaml**



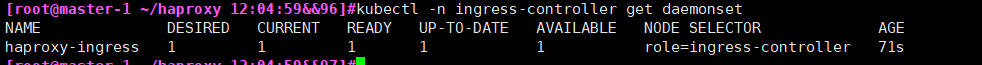
|  |
| --- |
| imagePullPolicy: IfNotPresent |

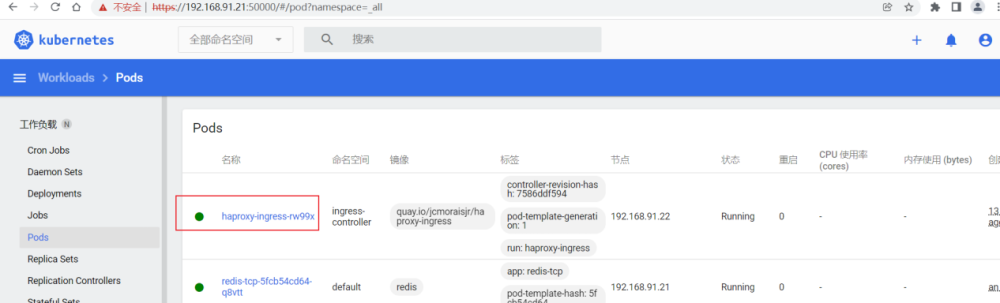
[root@master-1 ~]#**kubectl create -f haproxy-ingress.yaml**



**14.10.4查看ingress controller**

[root@master-1 haproxy]# **kubectl -n ingress-controller get daemonset**





**14.10.5部署Nginx访问**

[root@master-1 haproxy]# **kubectl create deployment nginx-http --image nginx:alpine**

[root@master-1 haproxy]# **kubectl expose deployment nginx-http --port=80**

#创建ingress

[root@master-1 haproxy]# **vi nginx.ingress.yaml**

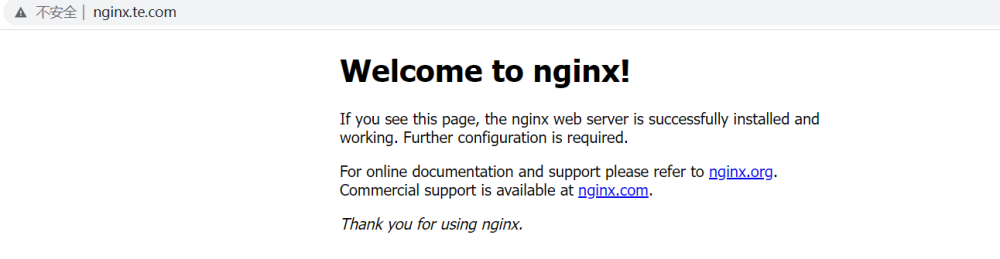
|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: nginx  spec:  rules:  - host: nginx.te.com  http:  paths:  - backend:  serviceName: nginx-http  servicePort: 80  path: / |

[root@master-1]#**kubectl apply -f nginx.ingress.yaml**

#客户端访问

#192.168.91.22 nginx.te.com

[root@master-1]#**curl nginx.te.com**



**14.10-2 （新版）部署Ingress Controler-**

**14.10.1 安装helm，HAProxy Ingress 需要版本 3**

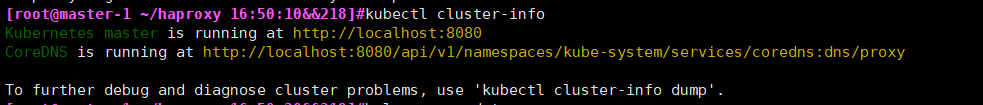
|  |
| --- |
| [root@master-1]#**tar xf helm-v3.8.1-linux-amd64.tar.gz**  [root@master-1]#**mv helm /usr/bin/** |

**14.10.2 添加 HAProxy Ingress 的 Helm 存储库**

|  |
| --- |
| [root@master-1]# **helm repo add haproxy-ingress** [**https://haproxy-ingress.github.io/charts**](https://haproxy-ingress.github.io/charts)  [root@master-1]#**helm repo update**  [root@master-1]#**helm repo list** |

**14.10.3检查 kubeconfig 是否指向正确的集群：**

[root@master-1]#**kubectl cluster-info**



**14.10.4下载模板**

**由于使用的kubernetes版本为v1.15.2，所以只能使用release-0.10版本的**

|  |
| --- |
| [root@master-1]#[**https://github.com/haproxy-ingress/charts/blob/release-0.10/haproxy-ingress/values.yaml**](https://github.com/haproxy-ingress/charts/blob/release-0.10/haproxy-ingress/values.yaml)  [root@master-1]#**mv values.yaml haproxy-ingress-values.yaml**  [root@master-1]#**vim haproxy-ingress-values.yaml** |

**14.10.5安装**

|  |
| --- |
| [root@master-1]#**helm install haproxy-ingress haproxy-ingress/haproxy-ingress\ --create-namespace --namespace ingress-controller\ --version 0.10.14\ -f haproxy-ingress-values.yaml** |

<https://www.modb.pro/db/239040>

<https://haproxy-ingress.github.io/v0.10/docs/getting-started/>

**14.10.6部署Nginx访问**

**同原版14.10.5部署Nginx访问步骤一致**

**14.11 查看配置文件**

|  |
| --- |
| [root@master-1 haproxy]# **kubectl exec -it haproxy-ingress-hchqj -n ingress-controller /bin/sh**  / # cat /etc/haproxy/haproxy.cfg  #查看转发配置  / # cat /etc/haproxy/maps/\_global\_http\_front.map |

**14.12基于Ingress Controller 实现蓝绿发布**

**#v1为绿色环境, v2为蓝色环境**

**14.12.1 创建第一个版本**

[root@master-1 haproxy]# **kubectl run blue \**

**--image=jcmoraisjr/whoami \**

**--port=8000 --labels=run=bluegreen,group=blue**

**#创建第二个版本**

[root@master-1 haproxy]# **kubectl run green \**

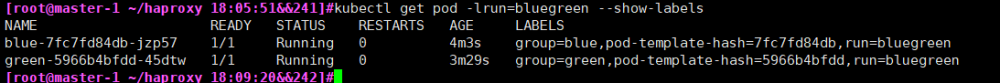
**--image=jcmoraisjr/whoami \**

**--port=8000 --labels=run=bluegreen,group=green**



**14.12.2 获取pod**

[root@master-1 haproxy]# **kubectl get pod -lrun=bluegreen --show-labels**



**14.12.3 为svc 添加标签（从deployment blue创建）**

[root@master-1 haproxy]# **kubectl expose deploy blue --name bluegreen --selector=run=bluegreen**



**14.12.4 获取SVC标签**

[root@master-1 haproxy]# **kubectl get svc bluegreen -otemplate --template '{{.spec.selector}}'**

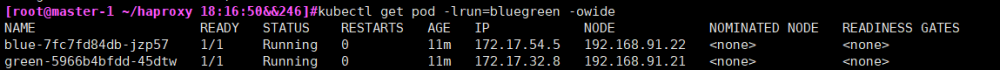
**14.12.5 获取ENDPOINTS（IP）**

[root@master-1 haproxy]# **kubectl get ep bluegreen**



**14.12.6 获取节点信息**

[root@master-1 haproxy]# **kubectl get pod -lrun=bluegreen -owide**



**14.12.7 创建ingress**

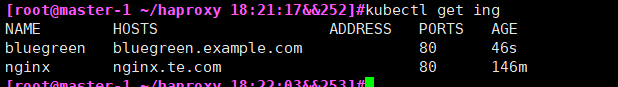
[root@master-1 haproxy]#

|  |
| --- |
| **cat > bluegreen.yaml << EOF**  apiVersion: networking.k8s.io/v1beta1  kind: Ingress  metadata:  annotations:  ingress.kubernetes.io/balance-algorithm: roundrobin  ingress.kubernetes.io/blue-green-deploy: group=blue=1,group=green=1  ingress.kubernetes.io/blue-green-mode: pod  ingress.kubernetes.io/ssl-redirect: "false"  name: bluegreen  spec:  rules:  - host: bluegreen.example.com  http:  paths:  - backend:  serviceName: bluegreen  servicePort: 8000  path: /  **EOF** |

[root@master-1 haproxy]#**kubectl apply -f bluegreen.yaml**

**14.12.8 获取ingress**

[root@master-1 haproxy]# **kubectl get ing**



**14.12.9 客户端测试**

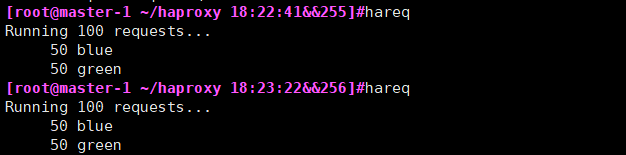
[root@master-1 haproxy]# **IP=192.168.91.22**

[root@master-1 haproxy]#**alias hareq='echo Running 100 requests...; for i in `seq 1 100`; do**

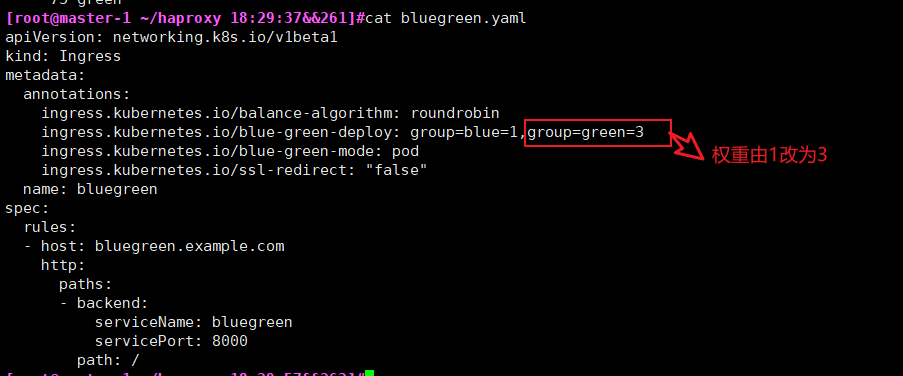
**curl -fsS $IP -H "Host: bluegreen.example.com" | cut -d- -f1**

**done | sort | uniq -c'**

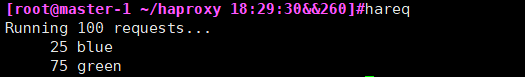
[root@master-1 ~/haproxy 18:22:41&&255]**#hareq**



**模式权重1：1 （50 blue,50 green）**



[root@master-1 ]#**kubectl apply -f bluegreen.yaml**



|  |
| --- |
| #测试第一种模式  BG Mode: pod  BG Balance: blue=1, green=1  Replicas: blue=1, green=1  #执行命令  [root@master-1 haproxy]# hareq  Running 100 requests...  50 blue  50 green  #测试第二种模式  BG Mode: pod  BG Balance: blue=1, green=1  Replicas: blue=1, green=3  [root@master-1 haproxy]# hareq  Running 100 requests...  25 blue  75 green  #测试第三种模式  BG Mode: deploy  BG Balance: blue=1, green=2  Replicas: blue=1, green=6  [root@master-1 haproxy]# hareq  Running 100 requests...  33 blue  67 green |

**15部署监控系统**

**15.0 监控内容**

|  |
| --- |
| 性能指标(如：CPU、Memory、Load、磁盘、网络等）  1.容器、Pod相关的性能指标数据  2.主机节点相关的性能指标数据  3.容器的网络性能，如http、tcp等数据  状态指标  1.kubernetes资源对象（Deployment、Daemonset、Pod等）的运行状态指标  2. kubernetes平台组件（如kube-apiserver、kube-scheduler、etcd等）的运行状态指标 |

**15.1 安装NFS服务端**

# Prometheus 与 Grafana 存储使用

15.1.1 master-1节点安装nfs

[root@master-1 ~]# **yum -y install nfs-utils**

15.1.2创建nfs目录

[root@master-1 ~]# **mkdir -p /ifs/kubernetes**

15.1.3修改权限

[root@master-1 ~]# **chmod -R 777 /ifs/kubernetes**

15.1.4编辑export文件

[root@master-1 ~]# **vim /etc/exports**

/ifs/kubernetes \*(rw,no\_root\_squash,sync)

15.1.5 修改配置启动文件

#修改配置文件

[root@master-1 ~]# **cat >/etc/systemd/system/sockets.target.wants/rpcbind.socket<<EOFL**

[Unit]

Description=RPCbind Server Activation Socket

[Socket]

ListenStream=/var/run/rpcbind.sock

ListenStream=0.0.0.0:111

ListenDatagram=0.0.0.0:111

[Install]

WantedBy=sockets.target

**EOFL**

15.1.6 配置生效

[root@master-1 ~]# exportfs -f

15.1.7 启动rpcbind、nfs服务

[root@master-1 ~]# **systemctl restart rpcbind**

[root@master-1 ~]# **systemctl enable rpcbind**

[root@master-1 ~]# **systemctl restart nfs**

[root@master-1 ~]# **systemctl enable nfs**

15.1.8 showmount测试(master-1)

[root@master-1 ~]# **showmount -e 192.168.91.18**



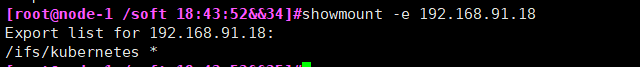
15.1.9 所有node节点安装客户端

[root@master-1 ~]# **yum -y install nfs-utils**

15.2.0 所有的Node检查

#所有的节点是否可以挂载, 必须要可以看到, 才能挂载成功.

[root@node-1 ~]# **showmount -e 192.168.91.18**



**15.2监控方案**

读取数据流程:

**15.2.0 node节点拉取镜像**

|  |
| --- |
| [root@node-1 ~]# **docker pull quay.io/external\_storage/nfs-client-provisioner** |

**15.2.1 部署PVC （master1操作）**

Nfs服务端地址需要修改

|  |
| --- |
| [root@master-1 nfs]# **kubectl apply -f nfs-class.yaml** |

|  |
| --- |
| #注意修改NFS IP地址nfs-deployment.yaml  [root@master-1 ~]# **kubectl apply -f nfs-deployment.yaml** |

|  |
| --- |
| [root@master-1 ~]# **kubectl apply -f nfs-rabc.yaml** |

#查看nfs pod状态

|  |
| --- |
| [root@master-1 nfs]# **kubectl get pods** |

**15.2.2 查看是否部署成功（master1操作）**

[root@master-1 nfs]# **kubectl get StorageClass**



**15.2.3 登录页面查看**



**15.3 部署监控系统**

**15.3.0.1注意需要修改的配置文件（master1操作）**

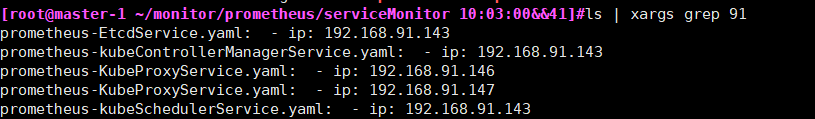
#**修改IP为18,19,20为etcd节点Ip。 21,22为节点Ip**

[root@master-1]#**mkdir /root/monitor && cd /root/monitor**

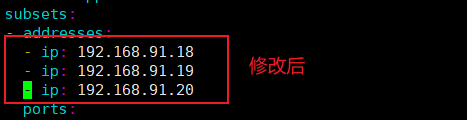
[root@master-1]# **unzip prometheus.zip** (上传prometheus.zip到/root/monitor)

[root@master-1 serviceMonitor]# **cd serviceMonitor**

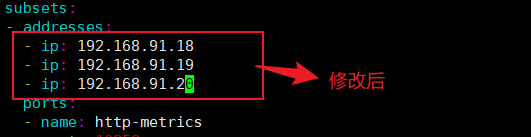
[root@master-1 serviceMonitor]# **ls | xargs grep 91** (/root/monitor/prometheus/serviceMonitor)



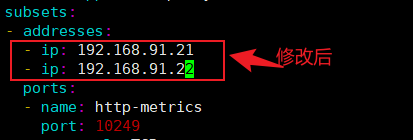
[root@master-1]#**vim prometheus-EtcdService.yaml**



[root@master-1]# **vim prometheus-kubeControllerManagerService.yaml**



[root@master-1]# **vim prometheus-KubeProxyService.yaml**

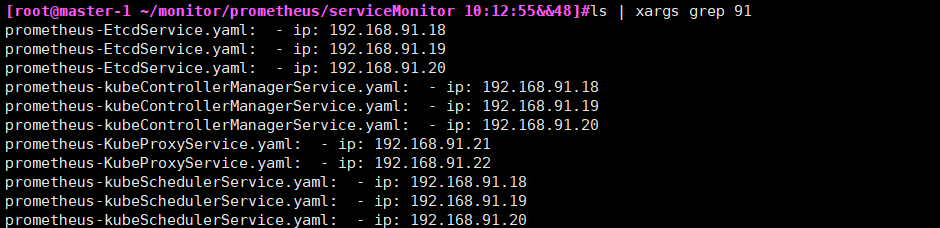


[root@master-1]# **vim prometheus-kubeSchedulerService.yaml**



最后确认下

[root@master-1]#**ls | xargs grep 91**



**15.3.0.2先下载镜像**（node所有节点）

|  |
| --- |
| [root@node-1]#**cat > monitor.sh <<EOF**  #/bin/bash  docker pull prom/alertmanager:v0.18.0  docker pull quay.io/coreos/configmap-reload:v0.0.1  docker pull grafana/grafana:6.4.3  docker pull quay.io/coreos/k8s-prometheus-adapter-amd64:v0.5.0  docker pull quay.io/coreos/kube-rbac-proxy:v0.4.1  docker pull quay.io/coreos/kube-state-metrics:v1.8.0  docker pull prom/node-exporter:v0.18.1  docker pull prom/prometheus:v2.11.0  docker pull quay.io/coreos/prometheus-config-reloader:v0.34.0  docker pull quay.io/coreos/prometheus-operator:v0.34.0  **EOF** |

[root@node-1]# **sh monitor.sh**

**15.3.0.3创建权限与alertmanager服务 (/root/monitor/prometheus)**

[root@master-1 monitor]# **kubectl apply -f setup/**

|  |
| --- |
| #查看dashboard |

[root@master-1 monitor]# **kubectl apply -f alertmanager/**

|  |
| --- |
| #查看dashboard |

[root@master-1]#**kubectl apply -f node-exporter/**

|  |
| --- |
|  |

[root@master-1 ]#**kubectl apply -f kube-state-metrics/**

|  |
| --- |
|  |

[root@master-1]#**kubectl apply -f grafana/**

|  |
| --- |
|  |

[root@master-1 ~/prometheus/prometheus 14:31:03&&9]#**kubectl apply -f prometheus/**

|  |
| --- |
|  |

[root@master-1 ]#**kubectl apply -f serviceMonitor/**

|  |
| --- |
|  |

**15.3.0.4注意如果提示权限问题, 解决方法如下：**

|  |
| --- |
| #如果没有错误略过  [root@master-1 monitor]# **kubectl create serviceaccount kube-state-metrics -n monitoring**  [root@master-1 monitor]# **kubectl create serviceaccount grafana -n monitoring**  [root@master-1 monitor]#  **kubectl create serviceaccount prometheus-k8s -n monitoring**  #创建权限文件  #如果没有错误略过  #kube-state-metrics  [root@master-1 kube-state-metrics]# cat kube-state-metrics-rabc.yaml  apiVersion: rbac.authorization.k8s.io/v1beta1  kind: ClusterRoleBinding  metadata:  name: kube-state-metrics-rbac  subjects:  - kind: ServiceAccount  name: kube-state-metrics  namespace: monitoring  roleRef:  kind: ClusterRole  name: cluster-admin  apiGroup: rbac.authorization.k8s.io  # grafana  [root@master-1 grafana]# cat grafana-rabc.yaml  apiVersion: rbac.authorization.k8s.io/v1beta1  kind: ClusterRoleBinding  metadata:  name: grafana-rbac  subjects:  - kind: ServiceAccount  name: grafana  namespace: monitoring  roleRef:  kind: ClusterRole  name: cluster-admin  apiGroup: rbac.authorization.k8s.io  # prometheus  [root@master-1 grafana]# **cat prometheus-rabc.yaml**  apiVersion: rbac.authorization.k8s.io/v1beta1  kind: ClusterRoleBinding  metadata:  name: prometheus-rbac  subjects:  - kind: ServiceAccount  name: prometheus-k8s  namespace: monitoring  roleRef:  kind: ClusterRole  name: cluster-admin  apiGroup: rbac.authorization.k8s.io |

**15.3.1 获取Grafana Pod**

[root@master-1 ~]# **kubectl get pod -A -o wide| grep grafana**



**15.3.2 获取Grafana SVC**

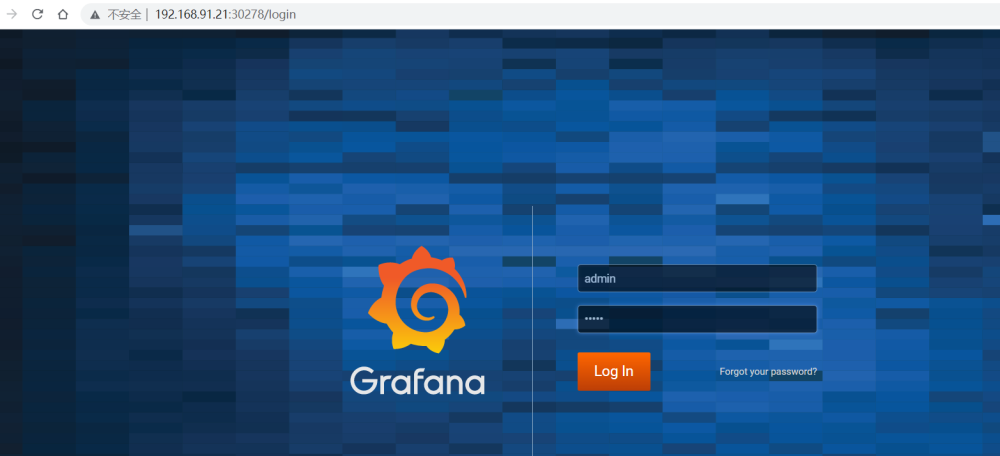
[root@master-1 ~]# **kubectl get svc -A | grep grafana**

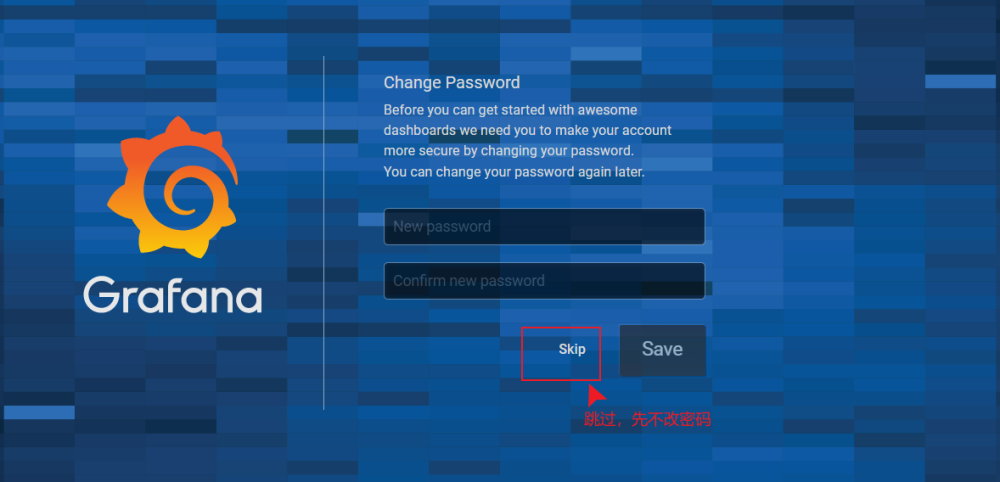


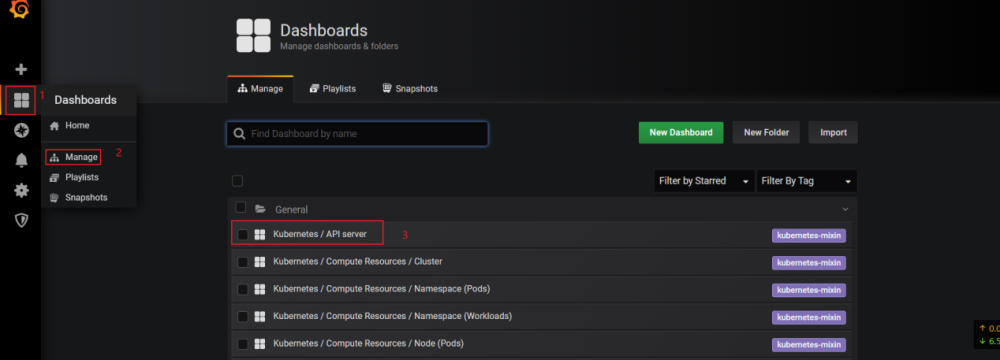
**15.3.3 登录Grafana Dashboard**

[**http://192.168.91.21:30278/login**](http://192.168.91.21:30278/login) **(任意一节点IP+端口访问)**

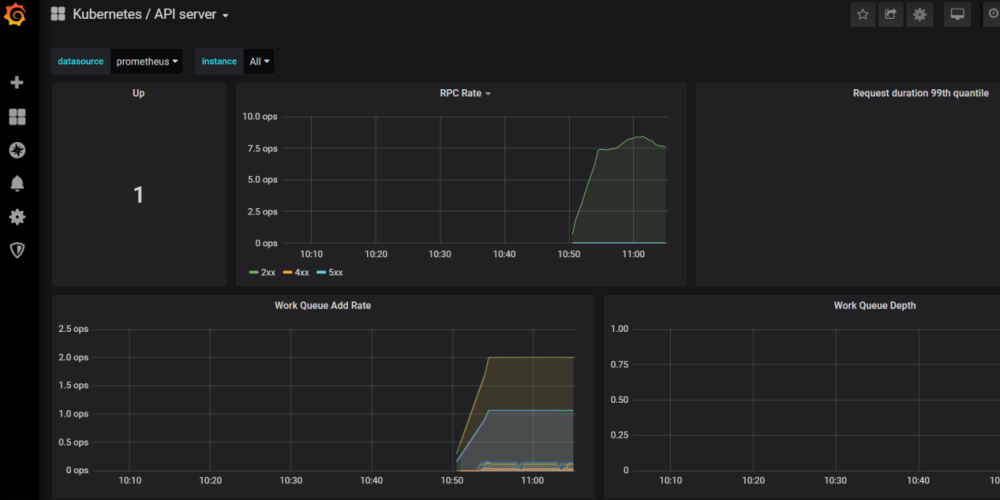
#用户与密码: admin/admin



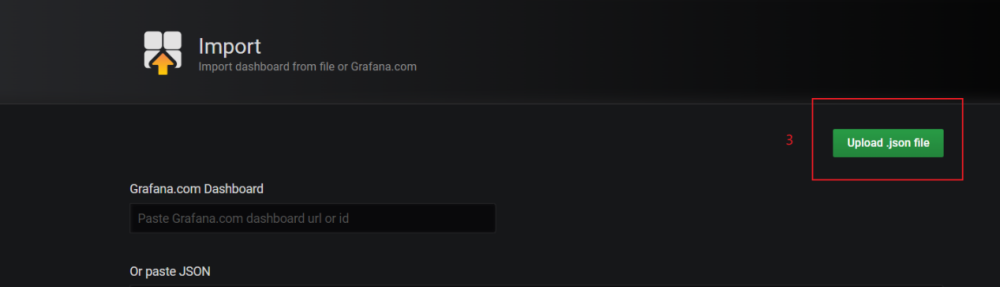
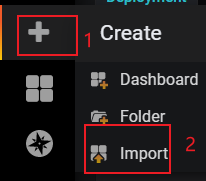




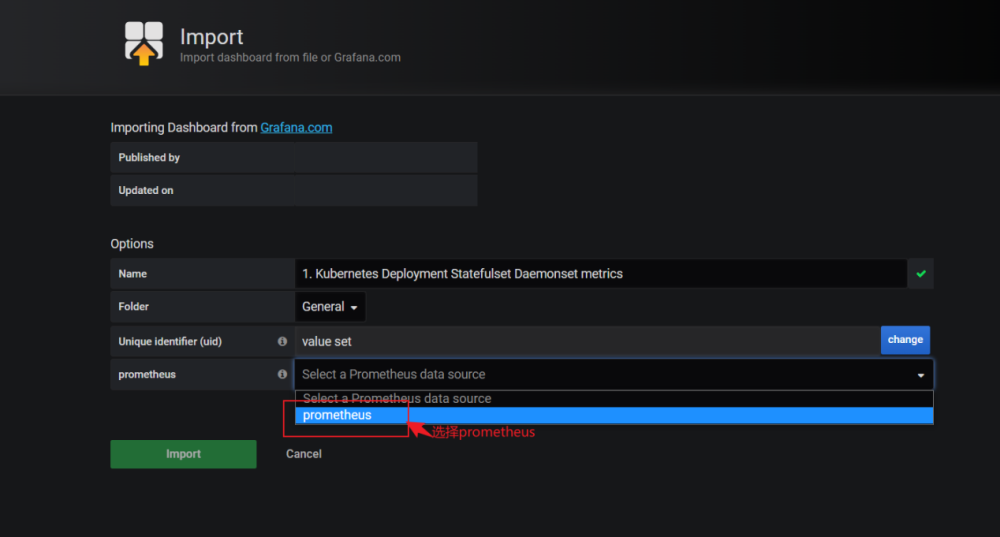
查看下图就会有数据了



导入模版



导入模版名为 **KCE-deployment-statefulset-daemonset-metrics 资源监控**



导入模版名为**KCE-Node节点-监控-1566546251543**



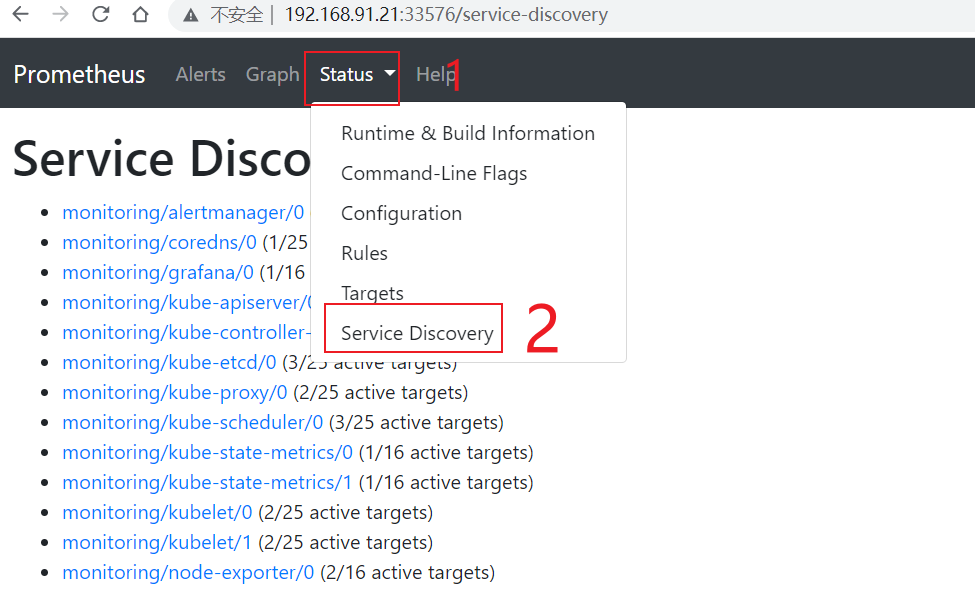
**15.3.4 选择资源**

#获取prometheus 地址

[root@master-1 prometheus]# **kubectl get svc -A | grep prometheus-k8s**

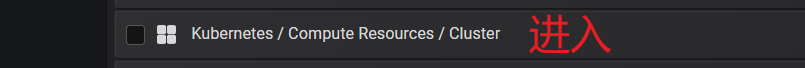


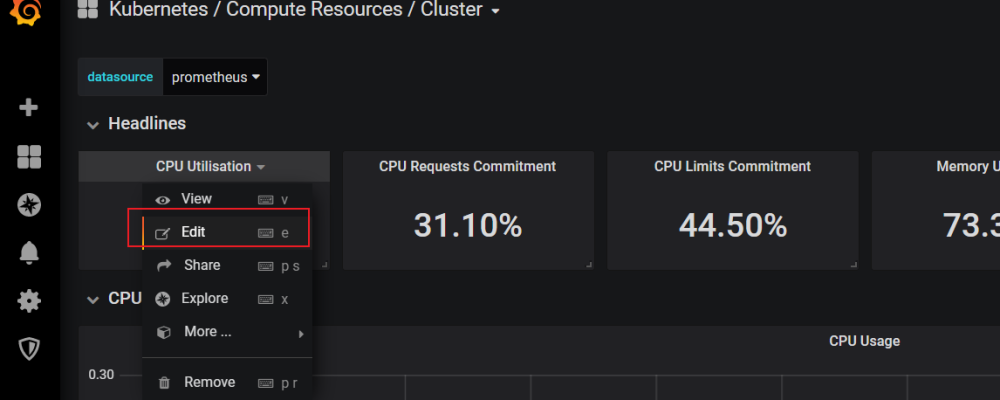
<http://192.168.91.21:33576/>



**验证是否有收集到数据**

举例

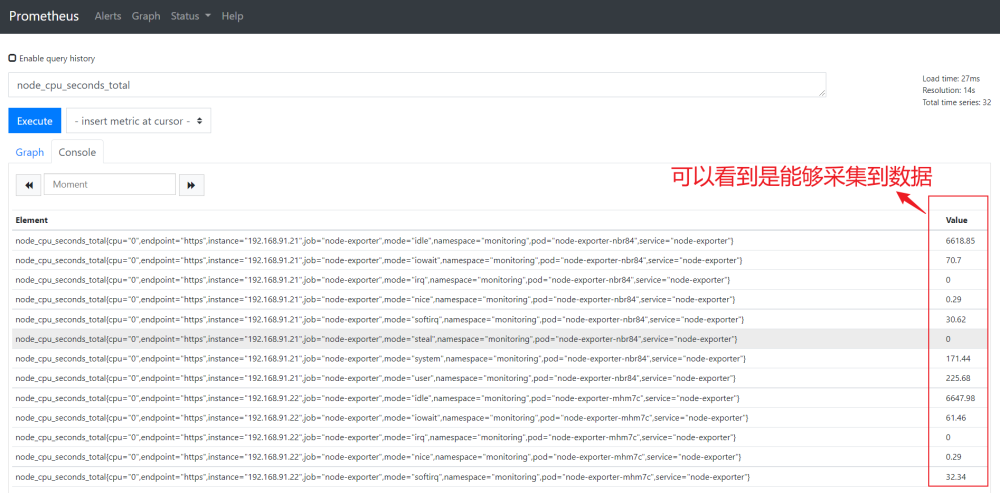






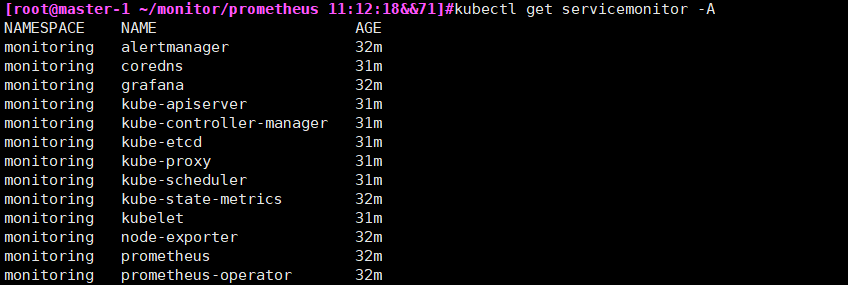
再到[**http://192.168.91.21:3357/**](http://192.168.91.21:3357/)查看





#查看servicemonitor

[root@master-1]# **kubectl get servicemonitor -A**



<http://192.168.91.21:31626/targets>

**15.3.5配置邮件报警**

|  |
| --- |
| [root@master-1 ~/prometheus/prometheus/alertmanager 15:06:58&&44]#cat mail\_alert/alertmanager.yaml  global:  resolve\_timeout: 5m  smtp\_smarthost: 'smtp.126.com:25'  smtp\_from: 'soulofsky1990@126.com'  smtp\_auth\_username: 'soulofsky1990@126.com'  smtp\_auth\_password: '123.com'  smtp\_hello: '126.com'  smtp\_require\_tls: false  route:  group\_by: ['job', 'severity']  group\_wait: 5s  group\_interval: 5m  repeat\_interval: 12h  receiver: default  receivers:  - name: 'default'  email\_configs:  - to: '15280808129@139.com'  send\_resolved: true  [root@master-1 ~/prometheus/prometheus/alertmanager 15:19:18&&58]#kubectl delete secret alertmanager-main -n monitoring  secret "alertmanager-main" deleted  [root@master-1 ~/prometheus/prometheus/alertmanager 15:20:03&&61]#kubectl create secret generic alertmanager-main --from-file=./mail\_alert/alertmanager.yaml -n monitoring  secret/alertmanager-main created  root@master-1 ~/prometheus/prometheus/alertmanager 15:05:37&&40]#kubectl delete -f alertmanager-alertmanager.yaml  alertmanager.monitoring.coreos.com "main" deleted  [root@master-1 ~/prometheus/prometheus/alertmanager 15:05:53&&41]#kubectl apply -f alertmanager-alertmanager.yaml  alertmanager.monitoring.coreos.com/main created  [root@master-1 ~/prometheus/prometheus/alertmanager 15:22:34&&66]#kubectl get svc alertmanager-main -n monitoring -o yaml |sed "s/ClusterIP/NodePort/g"|kubectl apply -f -  service/alertmanager-main configured  [root@master-1 ~/prometheus/prometheus/alertmanager 15:23:12&&67]#kubectl get svc alertmanager-main -n monitoring  NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  alertmanager-main NodePort 10.0.0.41 <none> 9093:36557/TCP 3h21m |

**15.3.6添加Ingress 到监控系统**

**15.3.6.1添加serviesMonitor 监控 （这里开始）**

#创建监控

[root@master-1 ~/ingress 15:28:26&&73]#**kubectl apply -f traefik-serviceMonitor.yaml**



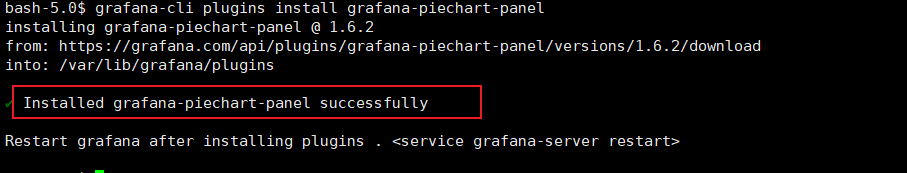
**15.3.6.2安装插件grafana-piechart-panel**

[root@master-1]# **kubectl get pods -A|grep grafana**



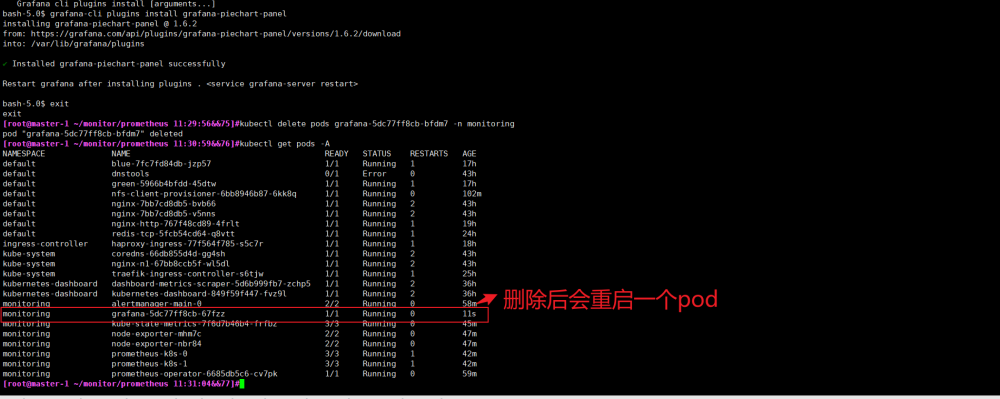
[root@master-1]# **kubectl exec -it -n monitoring grafana-5dc77ff8cb-bfdm7 /bin/bash**

bash-5.0$ **grafana-cli plugins install grafana-piechart-panel**



删除pod

[root@master-1]# **kubectl delete pods grafana-5dc77ff8cb-bfdm7 -n monitoring**



**15.3.6.3导入模版文件**

**Traefik 2-1587191399741.json**

**15.3.6.4访问nginx pod**

|  |
| --- |
| [root@master-1 ingress]# **kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools** |

|  |
| --- |
| [root@master-1 ingress]#**cat > nginx-route.yaml <<EOF**  apiVersion: traefik.containo.us/v1alpha1  kind: IngressRoute  metadata:  name: traefik-nginx11  spec:  entryPoints:  - web  routes:  - match: Host(`nginx11.abcd.com`)  kind: Rule  services:  - name: nginx-http  port: 80  **EOF** |

|  |
| --- |
| [root@master-1 ingress]#**kubectl apply -f nginx-route.yaml** |

**#绑定hosts**

[root@master-1 ]#**echo “192.168.91.21 nginx11.abcd.com” >> /etc/hosts**

#通过Ingress 访问服务

#展示数据

#需要访问Ingress 路由的服务, 才会有数据展示

[root@master-1 ]#**for i in `seq 1 100`;do curl nginx11.abcd.com;done (http 200)**

[root@master-1]#**for i in `seq 1 100`;do curl nginx11.abcd.com/werwerewr;done (http 404)**

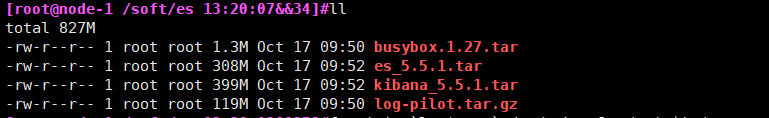


**16 容器日志收集方案**

|  |
| --- |
| ※ 把log-agent打包至业务镜像  ※ 日志落地至物理节点  ※ 每个物理节点启动日志容器  本例中在每个node 节点部署一个pod 收集日志 |

**17 安装日志组件**

**17.1上传镜像到node节点**



[root@node-1]#**for i in `ls \*`;do docker load -i $i;done**

#设置serviceAccount

[root@master-1 java]# **kubectl create serviceaccount admin -n kube-system**

**17.2 配置权限**

[root@master-1 logs]# **cat es-rbac.yaml**

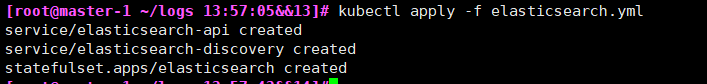
#创建权限

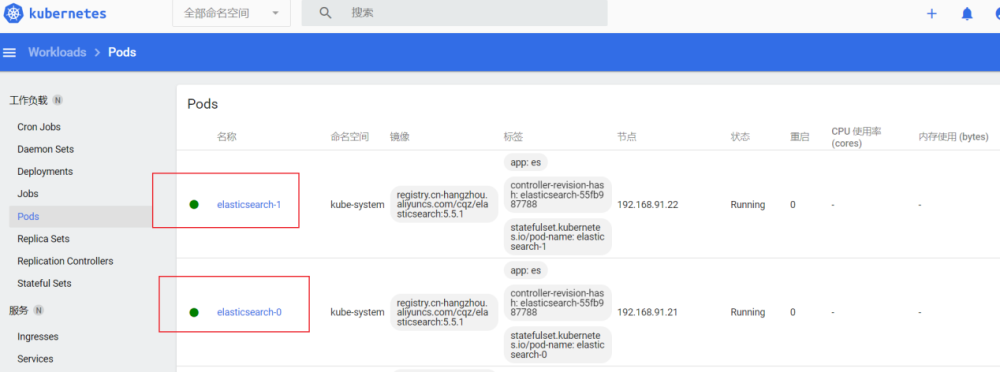
[root@master-1 logs]# **kubectl apply -f es-rbac.yaml**

**17.3 安装Elasticsearch**

|  |
| --- |
| **跳过，前面已导入**  [root@node-1]# docker pull registry.cn-hangzhou.aliyuncs.com/cqz/elasticsearch:5.5.1  [root@master-1]# wget <https://acs-logging.oss-cn-hangzhou.aliyuncs.com/elasticsearch.yml> (需要修改内存大小)  #导入的方式加载容器（所有的Node节点）  [root@node-1]# docker load < es\_5.5.1.tar  [root@node-1]# docker load < kibana\_5.5.1.tar  [root@node-1]# docker load < log-pilot.tar.gz  [root@node-1]# docker tag repo.hostscc.com/elk/elasticsearch:5.5.1 registry.cn-hangzhou.aliyuncs.com/cqz/elasticsearch:5.5.1 |

[root@master-1]# **kubectl apply -f elasticsearch.yml**





#查看节点状态

[root@master-1 logs]# **kubectl describe StatefulSet -A**

**17.4 查看ES 在Kubernetes中的状态**

#最好有三个ES 节点

[root@master-1 logs]# **kubectl get StatefulSet -n kube-system**

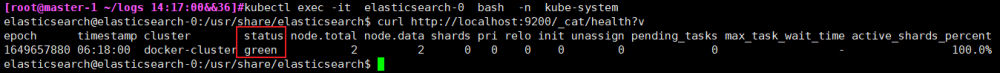


**17.5 查看ES 状态**

[root@master-1]#  **kubectl exec -it elasticsearch-0 bash -n kube-system**

#执行检查命令:

elasticsearch@elasticsearch-0:/usr/share/elasticsearch$**curl http://localhost:9200/\_cat/health?v**



**一定要是green状态**

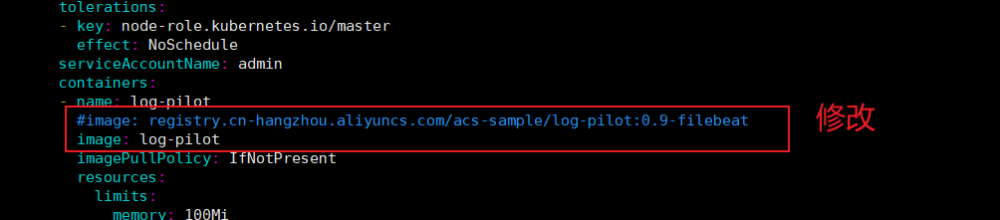
报错情况下

|  |
| --- |
| elasticsearch@elasticsearch-0: $ **curl http://localhost:9200/\_cat/health?v**  epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent  1574791667 18:07:47 docker-cluster green 2 2 0 0 0 0 0 0 - 100.0%  error: unable to upgrade connection: Forbidden (user=system:anonymous, verb=create, resource=nodes, subresource=proxy)  #解决方法:  [root@master-200 log]# **kubectl create clusterrolebinding system:anonymous --clusterrole=cluster-admin --user=system:anonymous** |

**17.6 安装 log-pilot**

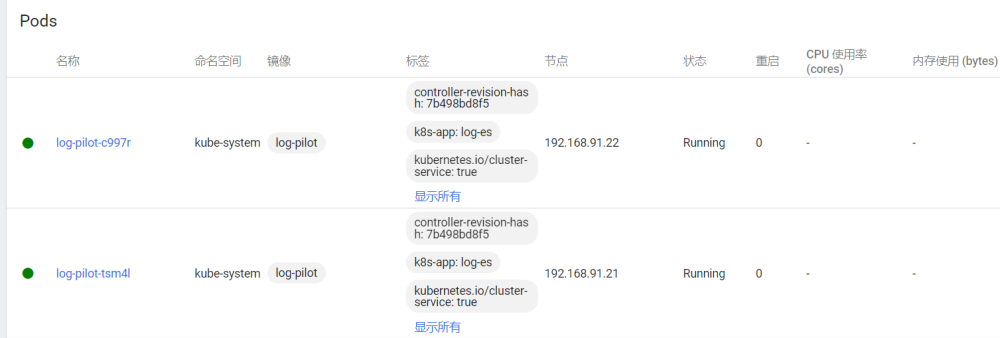
|  |
| --- |
| **跳过，前面已导入**  [root@master-1]# wget <https://acs-logging.oss-cn-hangzhou.aliyuncs.com/log-pilot.yml>  [root@node-1]# docker pull registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9-filebeat  #所有的Node节点  [root@node-1]# docker tag log-pilot:latest registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9-filebeat |

[root@master-1]#**vim log-pilot-2.0.yml**



#部署

[root@master-1]# **kubectl apply -f log-pilot-2.0.yml**



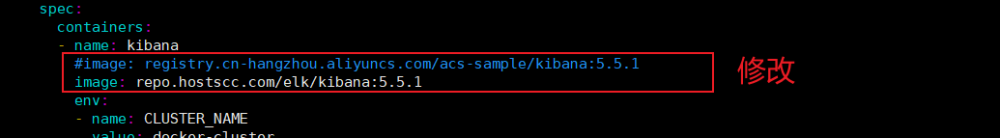
**17.7 安装kibana**

#注意修改命名空间

|  |
| --- |
| **跳过，前面已导入**  [root@master-1]# wget <https://acs-logging.oss-cn-hangzhou.aliyuncs.com/kibana.yml>  #所有节点  [root@node-1 ~]# docker tag repo.hostscc.com/elk/kibana:5.5.1 registry.cn-hangzhou.aliyuncs.com/acs-sample/kibana:5.5.1 |

#部署

[root@master-1]# **vim kibana.yml**



[root@master-1]# **kubectl apply -f kibana.yml**



**17.8 访问Kibana 界面**

**17.8.1 获取Kibana节点**

[root@master-1 logs]# **kubectl get pods -o wide --all-namespaces | grep kibana**



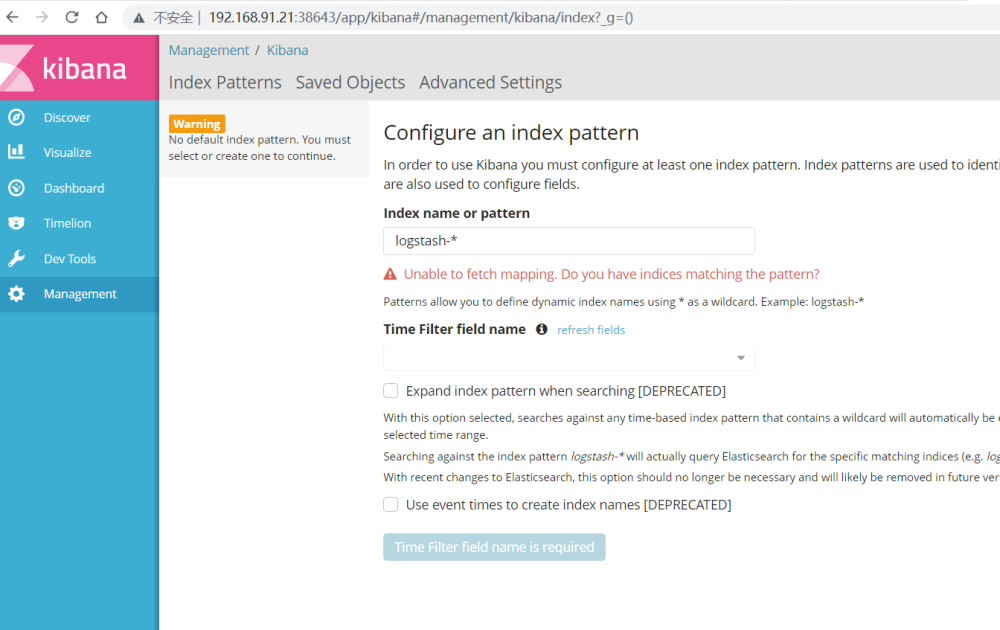
**17.8.2 获取Kibana HostPort 节点**

[root@master-1 logs]# **kubectl get svc --all-namespaces | grep kibana**



**17.8.3 访问web界面:**

<http://192.168.91.21:38643>



**17.8.4 案例一:运行容器收集日志**

**17.8.4.1 创建nginx yaml 文件**

#yaml 格式文件校验网址: <http://www.bejson.com/validators/yaml/>

[root@master-1 ~]# **mkdir /root/nginx && cd /root/nginx**

|  |
| --- |
| [root@master-1 nginx]# **cat > nginx-demo.yaml <<EOF**  apiVersion: apps/v1beta2  kind: Deployment  metadata:  name: nginx-demo  spec:  selector:  matchLabels:  app: nginx-demo  replicas: 1  template:  metadata:  labels:  app: nginx-demo  spec:  containers:  - name: nginx  image: nginx  imagePullPolicy: IfNotPresent  env:  - name: aliyun\_logs\_nginx  value: "stdout"  ---  apiVersion: v1  kind: Service  metadata:  name: nginx-demo-svc  spec:  selector:  app: nginx-demo  ports:  - port: 80  targetPort: 80  **EOF** |

#注意Yaml空格行

#创建pod

[root@master-1 nginx]# **kubectl apply -f nginx-demo.yaml**



#检查demo 状态

[root@master-1 nginx]# **kubectl get svc,pods**

|  |
| --- |
| aliyun\_logs\_catalina=stdout表示要收集容器的 stdout 日志。  aliyun\_logs\_access=/usr/local/tomcat/logs/catalina.\*.log 表示要收集容器内 /usr/local/tomcat/logs/ 目录下所有名字匹配 catalina.\*.log 的文件日志。  Log-Pilot 可以依据环境变量 aliyun\_logs\_$name = $path 动态地生成日志采集配置文件 |

**17.8.4.2 创建Nginx Ingress**

[root@master-1 java]# **cat nginx-route.yaml**

|  |
| --- |
| apiVersion: traefik.containo.us/v1alpha1  kind: IngressRoute  metadata:  name: nginx-demo-route  spec:  entryPoints:  - web  routes:  - match: Host(`nginx.cc.com`)  kind: Rule  services:  - name: nginx-demo-svc  port: 80 |

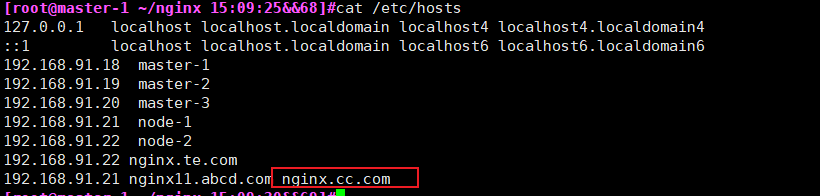
#创建路由

[root@master-1 nginx]# **kubectl apply -f nginx-route.yaml**

3.使用services 访问

[root@master-1 nginx]#**kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools**

4. 绑定主机hosts



5．访问界面

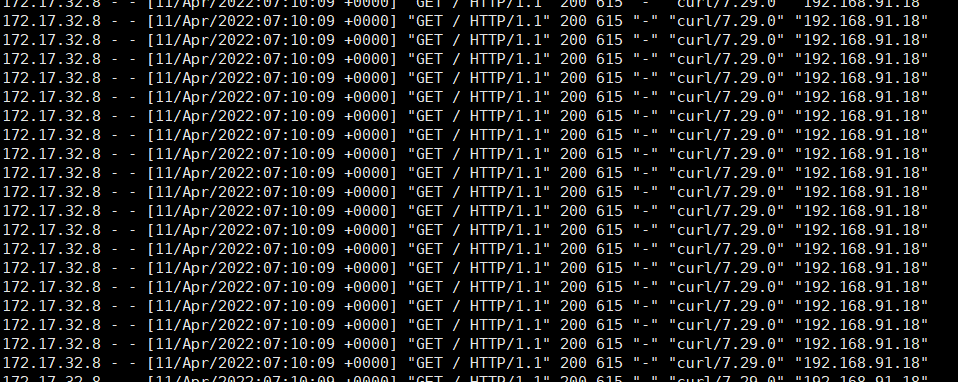
[root@master-1 nginx]# **for i in `seq 1 100`;do curl nginx.cc.com;done**

1. 查看容器日志

[root@master-1 nginx]#**kubectl get pods**

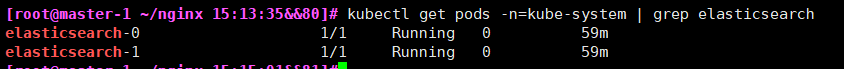


[root@master-1 nginx]# **kubectl logs -f nginx-demo-68749b58dc-b6l7k**



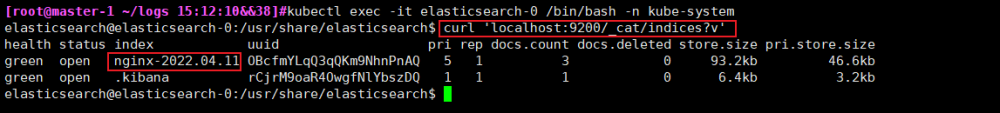
7查看是否建立索引

[root@master-1 nginx]# **kubectl get pods -n=kube-system | grep elasticsearch**

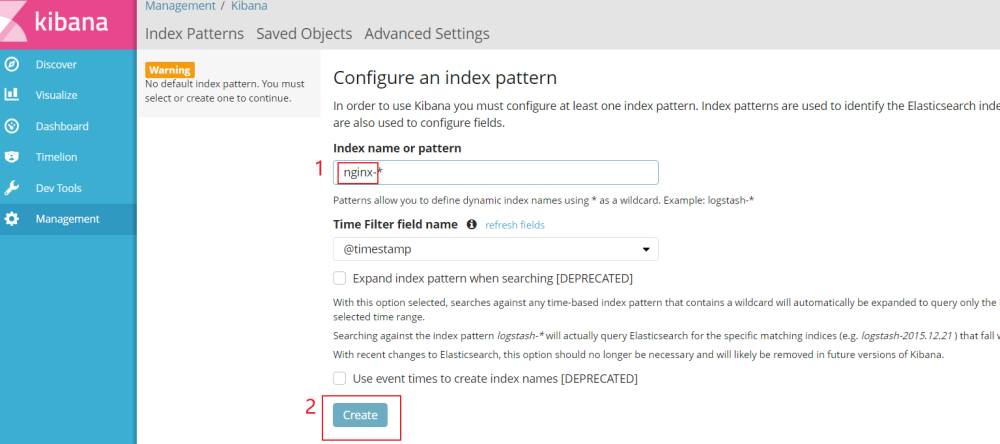


[root@master-1 nginx]# **kubectl exec -it elasticsearch-0 /bin/bash -n kube-system**

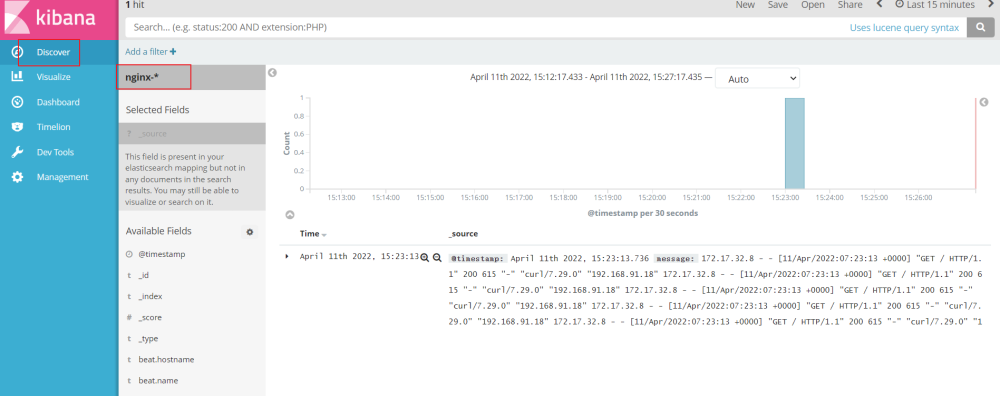
elasticsearch@elasticsearch-0:/usr/share/elasticsearch**$ curl 'localhost:9200/\_cat/indices?v'**



8 在kibana 中写入索引的地址



#查看访问日志



9.注意多行日志收集（JAVA）

参考：<https://www.iyunw.cn/archives/k8s-tong-guo-log-pilot-cai-ji-ying-yong-ri-zhi-ding-zhi-hua-tomcat-duo-xing/>

**18自动签发证书配置**

**此步骤做了就可以跳过10.2.1 批准请求,也就是10.1.9做完之后，10.2.1就可以不做操作了。**

**18.1 编辑token.csv**

**master-1操作**

[root@master-1]#**vim /etc/kubernetes/cfg/token.csv**

f89a76f197526a0d4bc2bf9c86e871c3,kubelet-bootstrap,10001,"system:bootstrappers"



**18.2 拷贝token到node节点**

**master-1操作**

|  |
| --- |
| [root@master-1]#**scp /etc/kubernetes/cfg/token.csv master-2:/etc/kubernetes/cfg/token.csv**  [root@master-1]#**scp /etc/kubernetes/cfg/token.csv master-3:/etc/kubernetes/cfg/token.csv** |

**18.3 重启kube-apiserver**

|  |
| --- |
| **service kube-apiserver restart** （所有master节点） |

**18.4 配置controller-manager**

**（所有master节点）**

|  |
| --- |
| cat >/etc/kubernetes/cfg/kube-controller-manager.cfg<<EOFL  KUBE\_CONTROLLER\_MANAGER\_OPTS="--logtostderr=true \  --v=4 \  --master=127.0.0.1:8080 \  --leader-elect=true \  --address=0.0.0.0 \  --service-cluster-ip-range=10.0.0.0/24 \  --cluster-name=kubernetes \  --feature-gates=RotateKubeletServerCertificate=true \  --experimental-cluster-signing-duration=87600h0m0s \  --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \  --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \  --root-ca-file=/etc/kubernetes/ssl/ca.pem \  --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem"  EOFL |

**18.5 重启kube-controller-manager**

**（所有master节点）**

**service kube-controller-manager restart**

**18.6 创建证书文件**

**18.6.1 master操作**

**master-1操作**

[root@master-1]#**mkdir /root/csr && cd /root/csr**

[root@master-1]#**vim tls-instructs-csr.yaml**

|  |
| --- |
| kind: ClusterRole  apiVersion: rbac.authorization.k8s.io/v1  metadata:  name: system:certificates.k8s.io:certificatesigningrequests:selfnodeserver  rules:  - apiGroups: ["certificates.k8s.io"]  resources: ["certificatesigningrequests/selfnodeserver"]  verbs: ["create"] |

[root@master-1]#**kubectl apply -f tls-instructs-csr.yaml**

**master-1操作**

#自动批准 kubelet-bootstrap 用户 TLS bootstrapping 首次申请证书的 CSR 请求

[root@master-1]#**kubectl create clusterrolebinding node-client-auto-approve-csr --clusterrole=system:certificates.k8s.io:certificatesigningrequests:nodeclient --user=kubelet-bootstrap**

**master-1操作**

#自动批准 system:nodes 组用户更新 kubelet 自身与 apiserver 通讯证书的 CSR 请求

[root@master-1]#**kubectl create clusterrolebinding node-client-auto-renew-crt --clusterrole=system:certificates.k8s.io:certificatesigningrequests:selfnodeclient --group=system:nodes**

**master-1操作**

#自动批准 system:nodes 组用户更新 kubelet 10250 api 端口证书的 CSR 请求

[root@master-1]#**kubectl create clusterrolebinding node-server-auto-renew-crt --clusterrole=system:certificates.k8s.io:certificatesigningrequests:selfnodeserver --group=system:nodes**

**18.6.2 客户端操作**

**客户端删除现在的证书（所有node节点**）

**rm -f /etc/kubernetes/ssl/kubelet-client-current.pem /etc/kubernetes/ssl/kubelet-client\*pem /etc/kubernetes/ssl/kubelet.key kubelet.crt**

客户端实现自动更新（所有node节点，**不同节点注意修改IP**）

**node1节点操作**

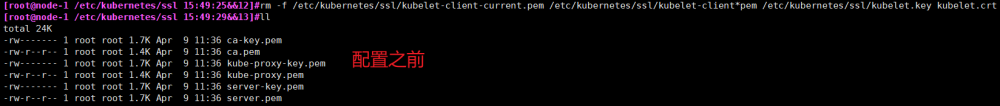
|  |
| --- |
| cat >/etc/kubernetes/cfg/kubelet<<EOF  KUBELET\_OPTS="--logtostderr=true \  --v=4 \  --hostname-override=192.168.91.21 \  --kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \  --bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \  --config=/etc/kubernetes/cfg/kubelet.config \  --feature-gates=RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true \  --rotate-certificates \  --cert-dir=/etc/kubernetes/ssl \  --pod-infra-container-image=docker.io/kubernetes/pause:latest"  EOF |

**node2节点操作**

|  |
| --- |
| cat >/etc/kubernetes/cfg/kubelet<<EOF  KUBELET\_OPTS="--logtostderr=true \  --v=4 \  --hostname-override=192.168.91.22 \  --kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \  --bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \  --config=/etc/kubernetes/cfg/kubelet.config \  --feature-gates=RotateKubeletClientCertificate=true,RotateKubeletServerCertificate=true \  --rotate-certificates \  --cert-dir=/etc/kubernetes/ssl \  --pod-infra-container-image=docker.io/kubernetes/pause:latest"  EOF |

**所有node节点**

**service kubelet restart**

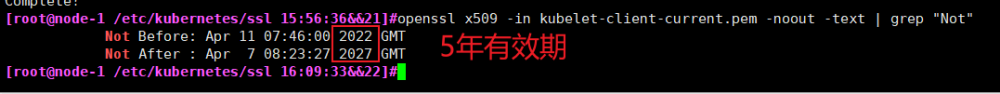




#查看kubelet.crt 查看（**node节点操作**）

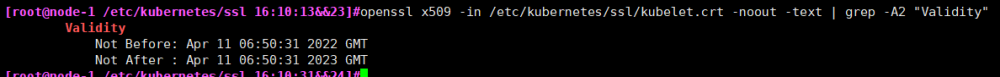
#默认一年

openssl x509 -in kubelet-client-current.pem -noout -text | grep "Not"



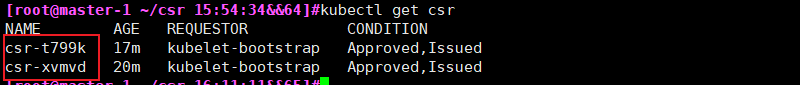
openssl x509 -in server.crt -text

openssl x509 -in /etc/kubernetes/ssl/kubelet.crt -noout -text | grep -A2 "Validity"



#服务端状态

[root@master-1 config]# kubectl get csr



|  |
| --- |
| #-----------------------参考  #------------  Below is a list of K8s (1.16) internal files (on each master node) which include certificates.  /etc/kubernetes/admin.conf  /etc/kubernetes/controller-manager.conf  /etc/kubernetes/scheduler.conf  /etc/kubernetes/pki/apiserver.crt  /etc/kubernetes/pki/apiserver-kubelet-client.crt  /etc/kubernetes/pki/apiserver-etcd-client.crt  /etc/kubernetes/pki/ca.crt  /etc/kubernetes/pki/etcd/healthcheck-client.crt  /etc/kubernetes/pki/etcd/peer.crt  /etc/kubernetes/pki/etcd/server.crt  /etc/kubernetes/pki/front-proxy-ca.crt  /etc/kubernetes/pki/front-proxy-client.crt  /var/lib/kubelet/pki/kubelet.crt  /var/lib/kubelet/pki/kubelet-client-current.pem  There are also some certificates on each worker node, mainly used by kubelet.  /etc/kubernetes/kubelet.conf  /etc/kubernetes/pki/ca.crt  /var/lib/kubelet/pki/kubelet.crt  /var/lib/kubelet/pki/kubelet-client-current.pem |

**19 Kubernetes NodeAllocatable**

**本例只做node1节点**

**19.1背景**

|  |
| --- |
| Kubernetes的节点可以按照Capacity资源进行调度. 在默认情况下pod能够使用(Worker)节点全部可用资源容量.  那么由此会带来一系列问题,因为每个Worker节点基本运行了系统程序以及Kubernetes的守护进程.  除非为这些守护进程留出系统资源,否则系统资源将与pod争夺资源并导致节点不可用. |

**19.2管理手段**

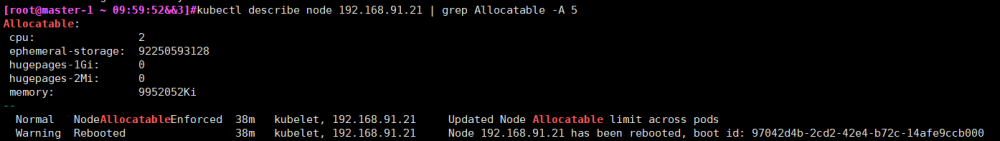
|  |
| --- |
| kubelet使用了一个Node Allocatable的特性,有利于为系统守护进程预留计算资源.  Kubernetes 推荐集群管理员按照每个节点上的工作负载情况,适当的配置 Node Allocatable. |

**19.3运行条件**

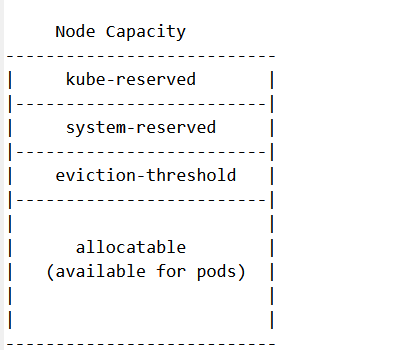
|  |
| --- |
| Kubernetes 服务器版本必须不低于版本 1.8 |

**19.4 查看节点的Capacity(容量)和Allocatable(可分配)**

[root@master-1 ~]# **kubectl describe node 192.168.91.21 | grep Allocatable -A 5**



#可分配资源和资源预留之间的关系



|  |
| --- |
| Kubelet Node Allocatable 用来为 Kube 组件和 System 进程预留资源,从而保证当节点出现满负荷时,  也能保证 Kube 和 System 进程有足够的资源。  目前支持 cpu, memory, ephemeral-storage 三种资源预留。  Node Capacity 是节点的所有硬件资源,  kube-reserved 是给 kube 组件预留的资源,  system-reserved 是给系统进程预留的资源,  eviction-threshold 是 kubelet 驱逐的阈值设定,  allocatable 才是真正调度器调度 Pod 时的参考值(保证节点上所有 Pods 的 request 资源不超过Allocatable)  节点可分配资源的计算方式为:  Node Allocatable Resource = Node Capacity - Kube-reserved - system-reserved - eviction-threshold |

**19.5配置**

**19.5.1 kubelet 配置为cgroupfs驱动(node-1)**

|  |
| --- |
| [root@ node-1 bin]# **cat >/etc/kubernetes/cfg/kubelet.config<<EOF**  kind: KubeletConfiguration  apiVersion: kubelet.config.k8s.io/v1beta1  address: 192.168.91.21  port: 10250  readOnlyPort: 10255  cgroupDriver: cgroupfs  clusterDNS: ["10.0.0.2"]  clusterDomain: cluster.local.  failSwapOn: false  authentication:  anonymous:  enabled: true  **EOF** |

**19.5.2启动配置参数**

|  |
| --- |
| --eviction-hard=memory.available<1024Mi,nodefs.available<10%,nodefs.inodesFree<5% \  --system-reserved=cpu=0.5,memory=1G \  --kube-reserved=cpu=0.5,memory=1G \  --kube-reserved-cgroup=/system.slice/kubelet.service \  --system-reserved-cgroup=/system.slice \  --enforce-node-allocatable=pods,kube-reserved,system-reserved  [root@node-1 bin]# **cat >/etc/kubernetes/cfg/kubelet<<EOF**  KUBELET\_OPTS="--logtostderr=true \\  --v=4 \\  --hostname-override=192.168.91.21 \  --eviction-hard=memory.available<1024Mi,nodefs.available<10%,nodefs.inodesFree<5% \\  --system-reserved=cpu=0.5,memory=1G \\  --kube-reserved=cpu=0.5,memory=1G \\  --kube-reserved-cgroup=/system.slice/kubelet.service \\  --system-reserved-cgroup=/system.slice \\  --enforce-node-allocatable=pods,kube-reserved,system-reserved \\  --kubeconfig=/etc/kubernetes/cfg/kubelet.kubeconfig \\  --bootstrap-kubeconfig=/etc/kubernetes/cfg/bootstrap.kubeconfig \\  --config=/etc/kubernetes/cfg/kubelet.config \\  --cert-dir=/etc/kubernetes/ssl \\  --pod-infra-container-image=docker.io/kubernetes/pause:latest"  **EOF**  #配置参数说明  --enforce-node-allocatable，默认为pods，要为kube组件和System进程预留资源，则需要设置为pods,kube-reserved,system-reserve。  --cgroups-per-qos，Enabling QoS and Pod level cgroups，默认开启。开启后，kubelet会将管理所有workload Pods的cgroups。  --cgroup-driver，默认为cgroupfs，另一可选项为systemd。取决于容器运行时使用的cgroup driver，kubelet与其保持一致。比如你配置docker使用systemd cgroup driver，那么kubelet也需要配置--cgroup-driver=systemd。  --kube-reserved,用于配置为kube组件（kubelet,kube-proxy,dockerd等）预留的资源量，比如--kube-reserved=cpu=1000m,memory=8Gi，ephemeral-storage=16Gi。  --kube-reserved-cgroup，如果你设置了--kube-reserved，那么一定要设置对应的cgroup，并且该cgroup目录要事先创建好，否则kubelet将不会自动创建导致kubelet启动失败。比如设置为kube-reserved-cgroup=/kubelet.service 。如果不设置此项，上面的--kube-reserved也不会生效。  --system-reserved，用于配置为System进程预留的资源量，比如--system-reserved=cpu=500m,memory=4Gi,ephemeral-storage=4Gi。  --system-reserved-cgroup，如果你设置了--system-reserved，那么一定要设置对应的cgroup，并且该cgroup目录要事先创建好，否则kubelet将不会自动创建导致kubelet启动失败。比如设置为system-reserved-cgroup=/system.slice。如果不设置此项的话，上面的--system-reserved也不会生效。  --eviction-hard，用来配置kubelet的hard eviction条件，只支持memory和ephemeral-storage两种不可压缩资源。当出现MemoryPressure时，Scheduler不会调度新的Best-Effort QoS Pods到此节点。当出现DiskPressure时，Scheduler不会调度任何新Pods到此节点。 |

|  |
| --- |
| #示例场景,说明节点分配计算方式的示例：  节点拥有 32Gi 内存，16 核 CPU 和 100Gi 存储  --kube-reserved 设置为 cpu=1,memory=2Gi,ephemeral-storage=1Gi  --system-reserved 设置为 cpu=500m,memory=1Gi,ephemeral-storage=1Gi  --eviction-hard 设置为 memory.available<500Mi,nodefs.available<10%  在这个场景下，Allocatable 将会是 14.5 CPUs、28.5Gi 内存以及 88Gi 本地存储。 调度器保证这个节点上的所有 pod 请求的内存总量不超过 28.5Gi，存储不超过 88Gi。 当 pod 的内存使用总量超过 28.5Gi 或者磁盘使用总量超过 88Gi 时，Kubelet 将会驱逐它们。 如果节点上的所有进程都尽可能多的使用 CPU，则 pod 加起来不能使用超过 14.5 CPUs 的资源。  当没有执行 kube-reserved 和/或 system-reserved 且系统守护进程使用量超过其预留时， 如果节点内存用量高于 31.5Gi 或存储大于 90Gi，kubelet 将会驱逐 pod |

**19.6修改kubelet启动配置服务(node-1)**

**#增加ExecStartPre项**

|  |
| --- |
| [root@node-1 bin]#**cat >/usr/lib/systemd/system/kubelet.service<<EOF**  [Unit]  Description=Kubernetes Kubelet  After=docker.service  Requires=docker.service  [Service]  EnvironmentFile=/etc/kubernetes/cfg/kubelet  ExecStartPre=-/bin/mkdir -p /sys/fs/cgroup/cpuset/system.slice/kubelet.service /sys/fs/cgroup/hugetlb/system.slice/kubelet.service  ExecStart=/usr/local/bin/kubelet \$KUBELET\_OPTS  Restart=on-failure  KillMode=process  [Install]  WantedBy=multi-user.target  **EOF** |

**#重启服务(node-1)**

[root@node-1 ~]# **systemctl daemon-reload**

[root@node-1 ~]# **service kubelet restart**

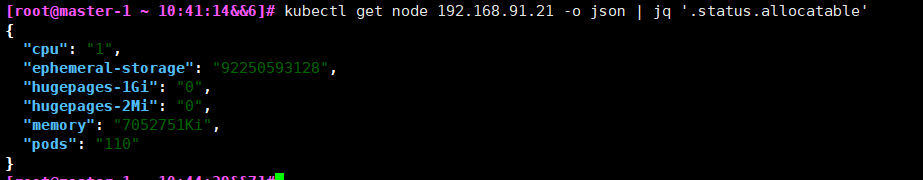
[root@node-1 ~]# **service kubelet status**

**#查看限制(master-1)**

[root@master-1 ~]#**yum install epel-relaese -y**

[root@master-1 ~]#**yum install jq -y**

[root@master-1 ~]# **kubectl get node 192.168.91.21 -o json | jq '.status.allocatable'**



**20 localdns\_install**

NodeLocal DNS (缓存)

**20.1为什么需要本地 DNS 缓存**

|  |
| --- |
| 减轻集群 DNS 解析压力，提高 DNS 性能  避免 netfilter 做 DNAT 导致 conntrack 冲突引发 DNS 5 秒延时  镜像底层库 DNS 解析行为默认使用 UDP 在同一个 socket 并发 A 和 AAAA 记录请求，  由于 UDP 无状态，两个请求可能会并发创建 conntrack 表项，  如果最终 DNAT 成同一个集群 DNS 的 Pod IP 就会导致 conntrack 冲突，  由于 conntrack 的创建和插入是不加锁的，最终后面插入的 conntrack 表项就会被丢弃，  从而请求超时，默认 5s 后重试，造成现象就是 DNS 5 秒延时;  底层库是 glibc 的容器镜像可以通过配 resolv.conf 参数来控制 DNS 解析行为，  不用 TCP 或者避免相同五元组并发(使用串行解析 A 和 AAAA 避免并发或者使用不同 socket 发请求避免相同源端口)，  但像基于 alpine 镜像的容器由于底层库是 musl libc，  不支持这些 resolv.conf 参数，也就无法规避，所以最佳方案还是使用本地 DNS 缓存。 |

**20.2原理**

|  |
| --- |
| 本地 DNS 缓存以 DaemonSet 方式在每个节点部署一个使用 hostNetwork 的 Pod，  创建一个网卡绑上本地 DNS 的 IP，本机的 Pod 的 DNS 请求路由到本地 DNS，  然后取缓存或者继续使用 TCP 请求上游集群 DNS 解析。 |

**20.3 IPtables 模式下部署方法**

IPVS 模式下需要修改 kubelet 参数

有两点需要注意下:

|  |
| --- |
| ipvs 模式下需要改 kubelet --cluster-dns 参数，指向一个非 kube-dns service 的 IP，通常用 169.254.20.10，Daemonset 会在每个节点创建一个网卡绑这个 IP，Pod 向本节点这个 IP 发 DNS 请求，本机 DNS 再代理到上游集群 DNS  iptables 模式下不需要改 kubelet --cluster-dns 参数，Pod 还是向原来的集群 DNS 请求，节点上有这个 IP 监听，被本机拦截，再请求集群上游 DNS (使用集群 DNS 的另一个 CLUSTER IP，来自事先创建好的 Service，跟原集群 DNS 的 Service 有相同的 selector 和 endpoint)  ipvs 模式下必须修改 kubelet 参数的原因是：如果不修改，DaemonSet Pod 在本机创建了网卡，会绑跟集群 DNS 的 CLUSTER IP，  但 kube-ipvs0 这个 dummy interface 上也会绑这个 IP (这是 ipvs 的机制，为了能让报文到达 INPUT 链被 ipvs 处理)，  所以 Pod 请求集群 DNS 的报文最终还是会被 ipvs 处理, DNAT 成集群 DNS 的 Pod IP，  最终路由到集群 DNS，相当于本机 DNS 就没有作用了。 |

[root@master-1 ~]# **kubectl -n kube-system get svc coredns**



**20.3.1 DNS配置**

|  |
| --- |
| [root@master-1 ~]# **kubectl -n kube-system get cm coredns -o yaml** |

**20.3.2导入镜像**

|  |
| --- |
| [root@node-1 ~]#**docker load -i k8s-dns-node-cache-1.21.1.tar**  [root@node-1 ~]#**docker tag 5bae806f8f12 k8s.gcr.io/dns/k8s-dns-node-cache:1.21.1** |

**20.3.3安装nodedns**

|  |
| --- |
| [root@master-1 ~]# **mkdir /root/nodedns**  [root@master-1 nodedns]# **yum install dos2unix -y**  [root@master-1 nodedns]# **dos2unix install\_localdns.sh** |

<https://github.com/kubernetes/kubernetes/raw/master/cluster/addons/dns/nodelocaldns/nodelocaldns.yaml>

**20.3.4查看dns**

|  |
| --- |
| [root@master-2 ~]# **kubectl get pods -n kube-system -o wide | grep node-local**    定制业务容器dnsConfig 为了使业务容器能够使用nodelocaldns，需要将nameserver配置为169.254.20.10,而不是ClusterDNS。定制dnsConfig有以下几点需要注意到： dnsPolicy: None。不使用ClusterDNS。 配置searches，保证集群内部域名能够被正常解析。 适当降低ndots值。当前ACK集群ndots值默认为5,降低ndots值有利于加速集群外部域名访问。如果业务容器没有使用带多个dots的集群内部域名，建议将值设为2。 |

**20.3.5测DNS**

|  |
| --- |
| [root@master-1 nginx]# **kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools**  dnstools# **nslookup -qt=A WWW.BAIDU.COM 169.254.20.10**    dnstools# **nslookup -qt=A grafana.monitoring 169.254.20.10**    dnstools# **nslookup baidu.com** |
|  |

**20.3.6修改DNS地址(所有node节点，注意修改节点IP)**

|  |
| --- |
| [root@node-1]#**cat >/etc/kubernetes/cfg/kubelet.config<<EOF**  kind: KubeletConfiguration  apiVersion: kubelet.config.k8s.io/v1beta1  address: 192.168.91.21  port: 10250  readOnlyPort: 10255  cgroupDriver: cgroupfs  clusterDNS: ["169.254.20.10"]  clusterDomain: cluster.local.  failSwapOn: false  authentication:  anonymous:  enabled: true  **EOF**  **#重启服务**  [root@node-1 ]#**service kubelet restart**  [root@node-1]#**service kubelet status** |

**20.3.7启动一个新容器**

|  |
| --- |
| [root@master-1]# **kubectl run nginx-dns --image=nginx --replicas=1**      **#DNS 地址修改**  [root@master-1]# **kubectl exec -i -t nginx-dns-54bdc96bb5-cclmt -- cat /etc/resolv.conf**    上图可看出dns地址已修改。 |