Kubernetes 1.15 安装

目录

1 软件版本

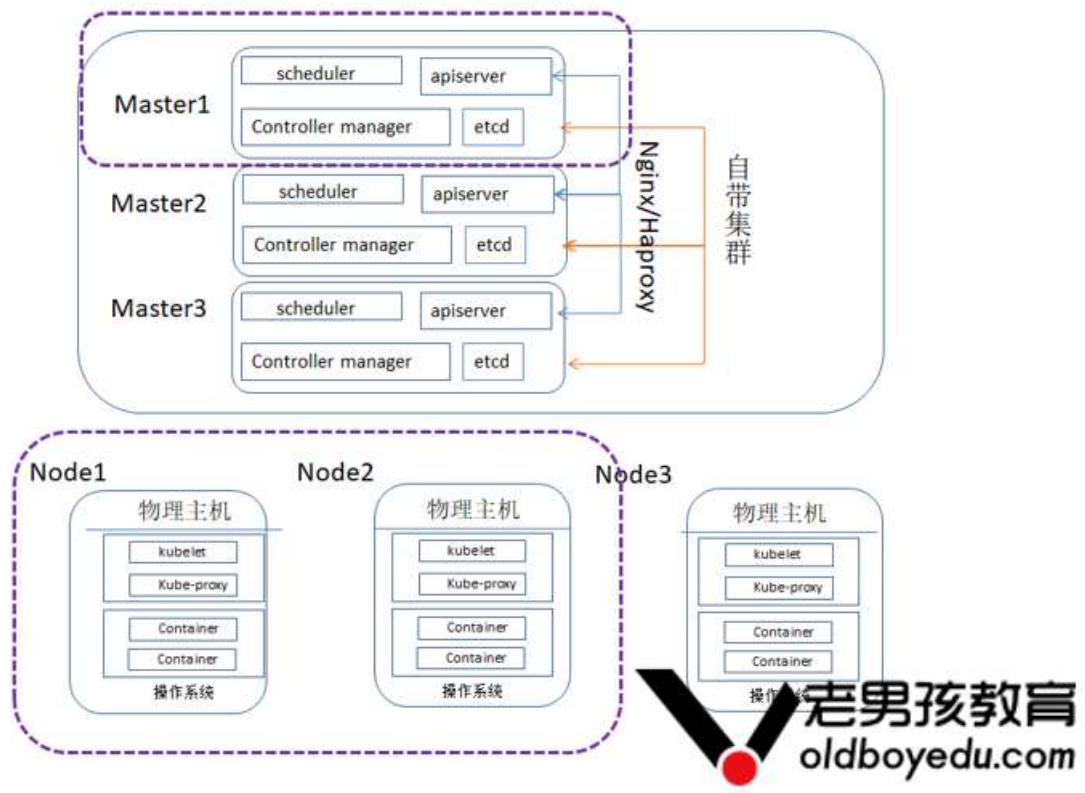
软件/系统	版本	备注
CentOS	7.5	
kubernetes-node-linux-amd64.tar.gz	1.15.1	
flannel	0.11	
etcd	3.3.10	

2 角色分配

Kubernetes 角色	分布节点	节点 IP
kube-apiserver	Master	192.168.91.18/19/20
kube-controller-manager	Master	192.168.91. 18/19/20
kube-scheduler	Master	192.168.91. 18/19/20
Etcd	Master	192.168.91. 18/19/20
kubelet	Node	192.168.91.21/22
kube-proxy	Node	192.168.91. 21/22
docker	Node	192.168.91. 21/22
flannel	Node	192.168.91. 21/22

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

```
[root@master-1 ~]# yum install net-tools vim wget lrzsz git –y
```

4.2 关闭防火墙与 Selinux

#所有节点

```
[root@master-1 ~]# systemctl stop firewalld
```

```
[root@master-1 ~]# systemctl disable firewalld
[root@master-1 ~]# sed -i "s/SELINUX=enforcing/SELINUX=disabled/g" /etc/selinux/config
[root@master-1 ~]# yum update -y
[root@master-1 ~]# reboot
```

4.3 设置时区

#所有节点

```
[root@master-1 ~]# \cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime -rf
```

4.4 关闭交换分区

#所有节点

```
[root@master-1 ~]# swapoff -a
[root@master-1 ~]# sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

4.5 设置系统时间同步

#所有节点

```
[root@master-1 ~]# yum install -y ntpdate
[root@master-1 ~]# ntpdate -u ntp.api.bz
[root@master-1 ~]# echo "*/5 * * * * ntpdate time7.aliyun.com >/dev/null 2>&1" >> /etc/crontab
[root@master-1 ~]# service crond restart
[root@master-1 ~]# chkconfig crond on
```

4.6 设置主机名

#所有节点

```
[root@master-1 ~]# cat > /etc/hosts <<EOF
127.0.0.1      localhost localhost.localdomain localhost4 localhost4.localdomain4
::1              localhost localhost.localdomain localhost6 localhost6.localdomain6
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)

```
[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=123456s
[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 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
```

4.9 Master 安装 Keepalived

```
[root@master-1 ~]# yum install -y keepalived
cat >/etc/keepalived/keepalived.conf <<EOL
global_defs {
    router_id KUB_LVS
}
vrrp_script CheckMaster {
      script "curl -k https://192.168.91.254:6443"
      interval 3
      timeout 9
      fall 2
      rise 2
}
vrrp_instance VI_1 {
      state MASTER
      interface ens32
      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 ens32
      }
      track_script {
            CheckMaster
      }
}
EOL
```

9.启动 keepalived

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

5 配置证书

5.1 下载自签名证书生成工具

#在分发机器 Master 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

```
[root@master-1 ~]# mkdir /root/etcd
[root@master-1 ~]# cd /root/etcd
```

5.2.1 CA 证书配置

```
[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

```
[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

```
[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

```
[root@master-1 ~]# cd /root/etcd/
```

#生成 ca 证书

```
[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

```
[root@master-1 ~]# 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.

```
[root@master-1 ~]# mkdir /root/kubernetes/
[root@master-1 ~]# cd /root/kubernetes/
```

5.3.1 配置 ca 文件

```
[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

```
[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

```
#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

```
[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

```
[root@master-1 ~]# cfssl gencert -initca ca-csr.json | cfssljson -bare ca –
```

生成 api-server 证书

```
[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

```
[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

```
[root@master-1 ~]# mkdir /soft
[root@master-1 ~]# 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

```
[root@master-1 ~]# mkdir -p /etc/etcd/{cfg,ssl}
[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
```

#参数说明:

ETCD_NAME 节点名称，如果有多个节点，那么每个节点要修改为本节点的名称。
ETCD_DATA_DIR 数据目录
ETCD_LISTEN_PEER_URLS 集群通信监听地址
ETCD_LISTEN_CLIENT_URLS 客户端访问监听地址
ETCD_INITIAL_ADVERTISE_PEER_URLS 集群通告地址
ETCD_ADVERTISE_CLIENT_URLS 客户端通告地址
ETCD_INITIAL_CLUSTER 集群节点地址,如果多个节点那么逗号分隔
ETCD_INITIAL_CLUSTER="master1=https://192.168.91.200:2380,master2=https://192.168.91.201:2380,master3=https://192.168.91.202:2380"
ETCD_INITIAL_CLUSTER_TOKEN 集群 Token
ETCD_INITIAL_CLUSTER_STATE 加入集群的当前状态，new 是新集群，existing 表示加入已有集群

6.2 创建 ETCD 的系统启动服务

```
[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

#此目录与之前的 ETCD 启动目录相一致

#如果有多个 Master 节点，那么需要复制到每个 Master

```
[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 ssh $i ls /etc/etcd/ssl;done
```

6.4 启动 etcd

```
[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
member bcef4c3b581e1d2e is healthy: got healthy result from https://192.168.91.18:2379
member d99a26304cec5ace is healthy: got healthy result from https://192.168.91.19:2379
member fc4e801f28271758 is healthy: got healthy result from https://192.168.91.20:2379
cluster is healthy
```

6.6 创建 Docker 所需分配 POD 网段

#向 etcd 写入集群 Pod 网段信息

#172.17.0.0/16 为 Kubernetes Pod 的 IP 地址段.

#网段必须与 kube-controller-manager 的 --cluster-cidr 参数值一致

```
[root@master-2 ~]# 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-2 etcd-v3.3.10-linux-amd64]# 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
{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}
```

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 https://download.docker.com/linux/centos/docker-ce.repo
[root@node-1 ~]# yum install 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
```

8 部署 Flannel

8.1 下载 Flannel 二进制包

#所有的节点,下载到 master-1

```
[root@ node -1 ~]# mkdir /soft && cd /soft
[root@ node -1 ~]# wget https://github.com/coreos/flannel/releases/download/v0.11.0/flannel-v0.11.0-linux-amd64.tar.gz
[root@ node -1 ~]# tar xvf flannel-v0.11.0-linux-amd64.tar.gz
[root@ node -1 ~]# mv flanneld mk-docker-opts.sh /usr/local/bin/
```

#复制 flanneld 到其他的所有节点

```
[root@ node -1 ~]# for i in master-2 master-3 node-1 node-2;do scp /usr/local/bin/flanneld $i:/usr/local/bin/;done
[root@ node -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

```
[root@node-1 ~]# mkdir -p /etc/flannel
[root@ node -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

```
[root@node-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
```

#启动脚本说明
#mk-docker-opts.sh 脚本将分配给 flanneld 的 Pod 子网网段信息写入 /run/flannel/docker 文件，后续 docker 启动时 使用这个文件中的环境变量配置 docker0 网桥；
#flanneld 使用系统缺省路由所在的接口与其它节点通信，对于有多个网络接口（如内网和公网）的节点，可以用 -iface 参数指定通信接口，如上面的 eth0 接口;

8.4 启动 Flannel

```
[root@node-1 ~]# service flanneld start
[root@node-1 ~]# chkconfig flanneld on
[root@node-2 ~]# service flanneld status
Redirecting to /bin/systemctl status flanneld.service
● flanneld.service - Flanneld overlay address etcd agent
   Loaded: loaded (/usr/lib/systemd/system/flanneld.service; disabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-04-05 14:35:51 CST; 7min ago
  Process: 11420 ExecStartPost=/usr/local/bin/mk-docker-opts.sh -k DOCKER_NETWORK_OPTIONS -d /run/flannel/subnet.env (code=exited, status=0/SUCCESS)
 Main PID: 11406 (flanneld)
    Tasks: 8
   Memory: 6.6M
   CGroup: /system.slice/flanneld.service
           └─11406 /usr/local/bin/flanneld --ip-masq -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...
```

#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 ~]# service flanneld stop
```

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

```
[root@node-1 ~]# systemctl daemon-reload
[root@node-1 ~]# service docker restart
```

#检查 IP 地址, docker 与 flanneld 是同一个网段

```
[root@node-1 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:7b:24:0a brd ff:ff:ff:ff:ff:ff
```

```
    inet 192.168.91.21/24 brd 192.168.91.255 scope global noprefixroute ens32
       valid_lft forever preferred_lft forever
    inet6 fe80::f8e9:2eba:8648:f6ad/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:1b:93:48:98 brd ff:ff:ff:ff:ff:ff
    inet 172.17.68.1/24 brd 172.17.68.255 scope global docker0
       valid_lft forever preferred_lft forever
4: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 8a:11:c3:08:83:48 brd ff:ff:ff:ff:ff:ff
    inet 172.17.68.0/32 scope global flannel.1
       valid_lft forever preferred_lft forever
    inet6 fe80::8811:c3ff:fe08:8348/64 scope link
       valid_lft forever preferred_lft forever
```

8.7 Node 节点验证是否可以访问其他节点 Docker0

#在每个 Node 节点 Ping 其他的节点，网段都是通的。

```
[root@master-1 soft]# ping   172.17.68.1
PING 172.17.68.1 (172.17.68.1) 56(84) bytes of data.
64 bytes from 172.17.68.1: icmp_seq=1 ttl=64 time=0.345 ms
64 bytes from 172.17.68.1: icmp_seq=2 ttl=64 time=0.325 ms
64 bytes from 172.17.68.1: icmp_seq=3 ttl=64 time=0.518 ms
```

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.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 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   soft]# 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]# service kube-apiserver start
[root@master-1    soft]# chkconfig kube-apiserver on
[root@master-1    soft]# service kube-apiserver status
[root@master-2 ~]# service kube-apiserver status
```

```
Redirecting to /bin/systemctl status kube-apiserver.service
● kube-apiserver.service - Kubernetes API Server
   Loaded: loaded (/usr/lib/systemd/system/kube-apiserver.service; disabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-04-05 15:12:09 CST; 523ms ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 13884 (kube-apiserver)
   CGroup: /system.slice/kube-apiserver.service
           └─13884 /usr/local/bin/kube-apiserver --logtostderr=true --v=4 --insecure-bind-address=0.0.0.0 --insecure-port=8080
--etcd-servers=https://192.168.91.18:2379,https://192.16...

Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939058   13884 flags.go:33] FLAG: --token-auth-file="/etc/kubernetes/cfg/token.csv"
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939062   13884 flags.go:33] FLAG: --v="4"
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939065   13884 flags.go:33] FLAG: --version="false"
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939077   13884 flags.go:33] FLAG: --vmodule=""
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939081   13884 flags.go:33] FLAG: --watch-cache="true"
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939085   13884 flags.go:33] FLAG: --watch-cache-sizes="[]"
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939103   13884 services.go:45] Setting service IP to "10.0.0.1" (read-write).
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939120   13884 server.go:560] external host was not specified, using 192.168.91.19
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.939129   13884 server.go:603] Initializing cache sizes based on 0MB limit
Apr 05 15:12:10 master-2 kube-apiserver[13884]: I0405 15:12:09.952964   13884 server.go:147] Version: v1.15.1
```

#查看加密的端口是否已经启动

```
[root@master-2 ~]# netstat -anltup | grep 6443
tcp        0      0 192.168.91.19:6443      0.0.0.0:*               LISTEN      14061/kube-apiserve
tcp        0      0 192.168.91.19:6443      192.168.91.19:36760     ESTABLISHED 14061/kube-apiserve
tcp        0      0 192.168.91.19:36760     192.168.91.19:6443      ESTABLISHED 14061/kube-apiserve
```

#查看加密的端口是否已经启动（node 节点）

```
[root@node-1 ~]# telnet 192.168.91.254 6443
Trying 192.168.91.254...
Connected to 192.168.91.254.
Escape character is '^]'.
```

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
```

#参数说明

--bind-address=0.0.0.0 启动绑定地址

--master 连接本地 apiserver(非加密端口)

--leader-elect=true：集群运行模式，启用选举功能；被选为 leader 的节点负责处理工作，其它节点为阻塞状态；

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    soft]# service kube-scheduler restart
[root@master-1    soft]# chkconfig kube-scheduler on
```

9.2.3 查看 Master 节点组件状态（任意一台 master）

```
[root@master-1 bin]# kubectl get cs
NAME                 STATUS    MESSAGE                         ERROR
controller-manager   Unhealthy Get http://127.0.0.1:10252/healthz: dial tcp 127.0.0.1:10252: connect: connection refused
scheduler            Healthy   ok
etcd-0               Healthy   {"health":"true"}
```

9.3 部署 kube-controller-manager

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

```
[root@master-1 bin]# 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

```
[root@master-1 bin]#    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
```

9.3.3 启动 kube-controller-manager 服务

```
[root@master-1 bin]#    chkconfig kube-controller-manager on
[root@master-1 bin]#    service kube-controller-manager start
[root@master-2 ~]# service kube-controller-manager status
Redirecting to /bin/systemctl status kube-controller-manager.service
● kube-controller-manager.service - Kubernetes Controller Manager
   Loaded: loaded (/usr/lib/systemd/system/kube-controller-manager.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-04-05 15:52:30 CST; 1s ago
     Docs: https://github.com/kubernetes/kubernetes
 Main PID: 16979 (kube-controller)
   CGroup: /system.slice/kube-controller-manager.service
           └─16979 /usr/local/bin/kube-controller-manager --logtostderr=true --v=4 --master=127.0.0.1:8080 --leader-elect=true --address=0.0.0.0
```

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
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
```

```
etcd-1                  Healthy   {"health":"true"}
etcd-0                  Healthy   {"health":"true"}
etcd-2                  Healthy   {"health":"true"}
```

10 部署 Node 节点组件

Node 节点需要部署的组件:
Kubelet
Kube-proxy
Flannel
Docker

10.1 部署 kubelet 组件

kublet 运行在每个 Node 节点上，接收 kube-apiserver 发送的请求，管理 Pod 容器，执行交互式命令，如 exec、run、logs 等;
kublet 启动时自动向 kube-apiserver 注册节点信息，内置的 cadvisor 统计和监控节点的资源使用情况;

10.1.1 从 Master 节点复制 Kubernetes 文件到 Node

#配置 Node 节点

```
[root@master-1 bin]#cd /soft
[root@master-1 bin]#scp kubernetes/server/bin/kubelet kubernetes/server/bin/kube-proxy node-1:/usr/local/bin/
[root@master-1 bin]#scp kubernetes/server/bin/kubelet kubernetes/server/bin/kube-proxy node-2:/usr/local/bin/
```

10.1.2 创建 kubelet bootstrap.kubeconfig 文件

Kubernetes 中 kubeconfig 文件配置文件用于访问集群信息，在开启了 TLS 的集群中，每次与集群交互时都需要身份认证，生产环境一般使用证书进行认证，其认证所需要的信息会放在 kubeconfig 文件中。

#Maste-1 节点

```
[root@master-1 bin]#mkdir /root/config
[root@master-1 bin]#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 bin]#ssh node-1 "mkdir -p /etc/kubernetes/{cfg,ssl}"
[root@master-1 bin]#ssh node-2 "mkdir -p /etc/kubernetes/{cfg,ssl}"
```

#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 bin]#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）

```
[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
```

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 bin]#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
```

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

```
[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 bin]#chkconfig kubelet on
[root@node-1 bin]#service kubelet start
[root@node-1 bin]#service kubelet status
```

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
NAME                                                   AGE     REQUESTOR           CONDITION
node-csr-4_tHtI9Y1ZOd1V3ZF5URGT7bWuRZWOizZYgeaBiAHOY   9m40s   kubelet-bootstrap   Pending
node-csr-bvq5buFKqAMvdJWOUjjP7hdez3xkQq5DPC4nNIL2vQs   9m37s   kubelet-bootstrap   Pending
```

10.2.1 批准请求

#Master 节点操作

```
[root@master-1 bin]#kubectl certificate approve node-csr-4_tHtI9Y1ZOd1V3ZF5URGT7bWuRZWOizZYgeaBiAHOY
```

10.3 节点重名处理

#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 查看节点状态

#所有的 Node 节点状态必须为 Ready （master）

```
[root@master-1 ~]# kubectl get nodes
NAME     STATUS   ROLES    AGE   VERSION
node-1   Ready    <none>   8s    v1.15.1
node-2   Ready    <none>   16s   v1.15.1
```

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

#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
```

10.4.2 创建 kube-proxy systemd unit 文件

```
[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 ~]#chkconfig kube-proxy on
[root@node-1 ~]#service kube-proxy start
[root@node-1 ~]#service kube-proxy status
```

11 运行 Demo 项目

```
[root@master-1 soft]# kubectl run nginx --image=nginx --replicas=2
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
deployment.apps/nginx created
[root@master-1 ~]#kubectl expose deployment nginx --port=88 --target-port=80 --type=NodePort
```

11.1 查看 pod

```
[root@master-1 cfg]# kubectl get pods
NAME                     READY   STATUS              RESTARTS   AGE
nginx-dbddb74b8-5zt7j    0/1     ContainerCreating   0          11s
nginx-dbddb74b8-grk7f    0/1     ContainerCreating   0          11s
```

11.2 查看 SVC

```
[root@master-1 cfg]# kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.0.0.1      <none>        443/TCP          45m
nginx        NodePort    10.0.0.93     <none>        88:43404/TCP     3s
```

11.3 访问 web

```
[root@master-1 cfg]# curl http://192.168.91.21:43404
```

11.4 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
```

12 部署 DNS

12.1 部署 coredns

```
[root@master-1 cfg]#    kubectl apply -f coredns.yaml
[root@master-1 dns]# kubectl get pod -A
[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
NAME                    TYPE              CLUSTER-IP      EXTERNAL-IP      PORT(S)                        AGE      SELECTOR
kube-dns                ClusterIP      10.0.0.254      <none>              53/UDP,53/TCP,9153/TCP      27s      k8s-app=kube-dns
```

12.3 验证 DNS 是否有效

12.3.1 删除之前创建的 nginx demo

```
[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
```

12.3.2 启动新容器

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

#出现错误

```
error: unable to upgrade connection: Forbidden (user=system:anonymous, verb=create, resource=nodes, subresource=proxy)
```

#解决方法

```
[root@master-1 nginx]# kubectl create clusterrolebinding system:anonymous --clusterrole=cluster-admin --user=system:anonymous
[root@master-1 dns]# kubectl delete pod    dnstools
[root@master-1 nginx]# kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools
```

12.3.2 创建 Nginx 容器

```
[root@master-1 ~]# kubectl run nginx --image=nginx --replicas=2
[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
NAME              TYPE              CLUSTER-IP      EXTERNAL-IP      PORT(S)              AGE
kubernetes      ClusterIP      10.0.0.1          <none>              443/TCP              158m
nginx              NodePort       10.0.0.55       <none>              88:35638/TCP      5s
```

12.3.4 测试解析 Nginx

#测试解析 nginx

```
dnstools# nslookup nginx
Server:                10.0.0.2
Address:               10.0.0.2#53

Name:      nginx.default.svc.cluster.local
Address: 10.0.0.55
```

12.3.5 容器的访问不区分命名空间

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

```
[root@master-1 ~]# kubectl run nginx --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.
[root@master-1 ~]# kubectl expose deployment nginx --port=99 --target-port=80 -n kube-system
```

13 部署 Dashboard

13.1 创建 Dashboard 证书

13.1.1 创建目录

```
[root@master-1 ~]# mkdir /certs
[root@master-1 ~]# cd /certs
```

13.1.2 创建命名空间

```
[root@master-1 ~]# kubectl create namespace kubernetes-dashboard
```

13.1.3 创建 key 文件

```
[root@master-1 ~]# openssl genrsa -out dashboard.key 2048
```

13.1.4 证书请求

```
[root@master-1 ~]# openssl req -days 36000 -new -out dashboard.csr -key dashboard.key -subj '/CN=dashboard-cert'
```

13.1.5 自签证书

```
[root@master-1 ~]# openssl x509 -req -in dashboard.csr -signkey dashboard.key -out dashboard.crt
```

13.1.6 创建 kubernetes-dashboard-certs 对象

```
[root@master-1 ~]# kubectl delete secrets kubernetes-dashboard-certs -n kubernetes-dashboard
[root@master-1 ~]# kubectl create secret generic kubernetes-dashboard-certs --from-file=/certs -n kubernetes-dashboard
```

13.1.7 查看系统中是否存在证书文件

```
[root@master-1 certs]# kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-745lc   kubernetes.io/service-account-token   3      19m
```

13.2 安装 Dashboard

13.2.1 创建目录

```
[root@master-1 certs]# mkdir /root/dashboard
[root@master-1 certs]# cd /root/dashboard/
```

13.2.2 下载证书文件

```
[root@master-1 certs]# wget https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta4/aio/deploy/recommended.yaml
```

13.2.3 修改 recommended 配置

由于证书问题，只能 firefox 浏览器才能打开，通过修改证书的方式，使得所有浏览器都能打开
下列内容全部注释

```
 50 #apiVersion: v1
 51 #kind: Secret
 52 #metadata:
 53 #    labels:
 54 #        k8s-app: kubernetes-dashboard
 55 #    name: kubernetes-dashboard-certs    #生成证书会用到该名字
 56 #    namespace: kubernetes-dashboard    #生成证书使用该命名空间
 57 #type: Opaque
```

13.2.3.1 增加 NodePort 配置

修改配置

```
 39 spec:
 40     type: NodePort                    #  设置为 NodePort
 41     ports:
 42       - port: 443
```

13.2.4 修改之后的 Dashboard 文件

```
[root@master-1 dashboard]# cat recommended.yaml
apiVersion: v1
kind: Namespace
metadata:
```

```
    name: kubernetes-dashboard

---

apiVersion: v1
kind: ServiceAccount
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard

---

kind: Service
apiVersion: v1
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard
spec:
    type: NodePort
    ports:
        - port: 443
            targetPort: 8443
    selector:
        k8s-app: kubernetes-dashboard

---

#apiVersion: v1
#kind: Secret
#metadata:
#    labels:
#        k8s-app: kubernetes-dashboard
#    name: kubernetes-dashboard-certs
#    namespace: kubernetes-dashboard
#type: Opaque

---

apiVersion: v1
kind: Secret
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard-csrf
    namespace: kubernetes-dashboard
type: Opaque
data:
    csrf: ""

---

apiVersion: v1
kind: Secret
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard-key-holder
    namespace: kubernetes-dashboard
type: Opaque

---
```

```
kind: ConfigMap
apiVersion: v1
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard-settings
    namespace: kubernetes-dashboard

---

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard
rules:
    # Allow Dashboard to get, update and delete Dashboard exclusive secrets.
    - apiGroups: [""]
        resources: ["secrets"]
        resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs", "kubernetes-dashboard-csrf"]
        verbs: ["get", "update", "delete"]
        # Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.
    - apiGroups: [""]
        resources: ["configmaps"]
        resourceNames: ["kubernetes-dashboard-settings"]
        verbs: ["get", "update"]
        # Allow Dashboard to get metrics.
    - apiGroups: [""]
        resources: ["services"]
        resourceNames: ["heapster", "dashboard-metrics-scraper"]
        verbs: ["proxy"]
    - apiGroups: [""]
        resources: ["services/proxy"]
        resourceNames: ["heapster", "http:heapster:", "https:heapster:", "dashboard-metrics-scraper", "http:dashboard-metrics-scraper"]
        verbs: ["get"]

---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
rules:
    # Allow Metrics Scraper to get metrics from the Metrics server
    - apiGroups: ["metrics.k8s.io"]
        resources: ["pods", "nodes"]
        verbs: ["get", "list", "watch"]

---

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard
roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
```

```
    name: kubernetes-dashboard
subjects:
    - kind: ServiceAccount
        name: kubernetes-dashboard
        namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard
roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: kubernetes-dashboard
subjects:
    - kind: ServiceAccount
        name: kubernetes-dashboard
        namespace: kubernetes-dashboard

---

kind: Deployment
apiVersion: apps/v1
metadata:
    labels:
        k8s-app: kubernetes-dashboard
    name: kubernetes-dashboard
    namespace: kubernetes-dashboard
spec:
    replicas: 1
    revisionHistoryLimit: 10
    selector:
        matchLabels:
            k8s-app: kubernetes-dashboard
    template:
        metadata:
            labels:
                k8s-app: kubernetes-dashboard
        spec:
            containers:
                - name: kubernetes-dashboard
                    image: kubernetesui/dashboard:v2.0.0-beta4
                    imagePullPolicy: Always
ports:
    - containerPort: 8443
        protocol: TCP
args:
    - --auto-generate-certificates
    - --namespace=kubernetes-dashboard
    # Uncomment the following line to manually specify Kubernetes API server Host
    # If not specified, Dashboard will attempt to auto discover the API server and connect
    # to it. Uncomment only if the default does not work.
    # - --apiserver-host=http://my-address:port
volumeMounts:
    - name: kubernetes-dashboard-certs
        mountPath: /certs
        # Create on-disk volume to store exec logs
    - mountPath: /tmp
        name: tmp-volume
livenessProbe:
    httpGet:
        scheme: HTTPS
```

```
                path: /
                port: 8443
              initialDelaySeconds: 30
              timeoutSeconds: 30
      volumes:
        - name: kubernetes-dashboard-certs
          secret:
            secretName: kubernetes-dashboard-certs
        - name: tmp-volume
          emptyDir: {}
      serviceAccountName: kubernetes-dashboard
      # Comment the following tolerations if Dashboard must not be deployed on master
      tolerations:
        - key: node-role.kubernetes.io/master
          effect: NoSchedule
```

13.2.5 应用 recommended.yaml

```
[root@master-1 ~]# kubectl apply -f recommended.yaml
```

13.2.6 查询创建结果

#获取到创建之后的 pod 的 Node

```
#获取 pod 节点
[root@master-1 ~]# kubectl get pod -A -o wide| grep kubernetes
kubernetes-dashboard      kubernetes-dashboard-6bb65fcc49-xm7fv         1/1         Running     0                2d2h    172.17.2.3        192.168.91.147
<none>           <none>

#获取 Nodepod 端口
[root@master-1 ~]# kubectl get svc -A | grep kubernetes
default                   kubernetes                  ClusterIP   10.0.0.1       <none>        443/TCP                  2d3h
kubernetes-dashboard      kubernetes-dashboard        NodePort    10.0.0.222     <none>        443:34501/TCP            2d2h
```

13.3 创建 Dashboard 访问账户

13.3.1 创建 SA

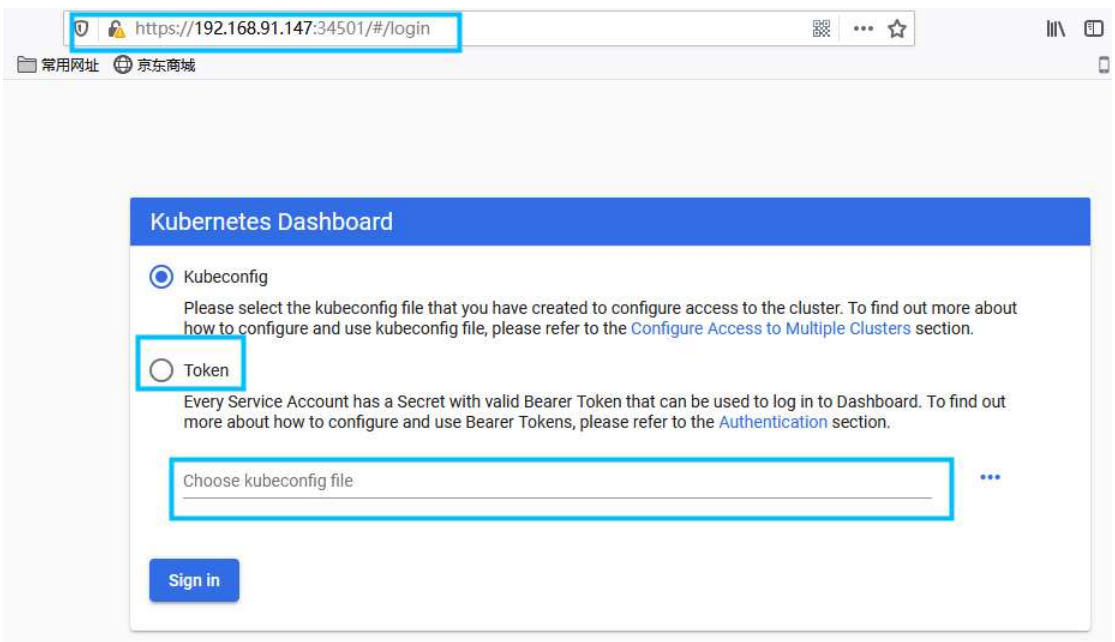
```
[root@master-1 ~]# kubectl create serviceaccount dashboard-admin -n kubernetes-dashboard
```

13.3.2 绑定集群管理员

```
[root@master-1 ~]# kubectl create clusterrolebinding    dashboard-cluster-admin --clusterrole=cluster-admin --serviceaccount=kubernetes-dashboard:dashboard-admin
```

13.4 访问 Dashboard

#使用火狐浏览器

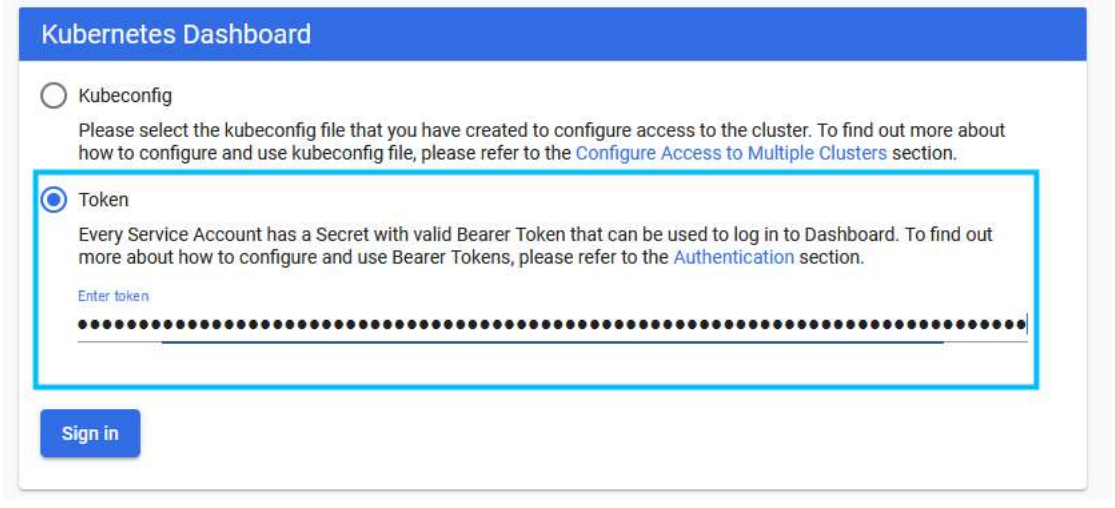


13.4.1 获取 Token

#复制 token 内容

```
[root@master-1 ~]# kubectl describe secrets    $(kubectl    get secrets -n kubernetes-dashboard | awk    '/dashboard-admin-token/{print $1}' ) -n kubernetes-dashboard |sed -n '/token:.*/p'
```

13.4.2 输入 token



14 部署 Ingress

#部署 Traefik 2.0 版本

14.1 创建 traefik-crd.yaml 文件

```
[root@master-1 ~]#    vim traefik-crd.yaml
## IngressRoute
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
   name: ingressroutes.traefik.containo.us
spec:
   scope: Namespaced
   group: traefik.containo.us
   version: v1alpha1
   names:
      kind: IngressRoute
      plural: ingressroutes
      singular: ingressroute
---
## IngressRouteTCP
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
   name: ingressroutetcps.traefik.containo.us
spec:
   scope: Namespaced
   group: traefik.containo.us
   version: v1alpha1
   names:
      kind: IngressRouteTCP
      plural: ingressroutetcps
      singular: ingressroutetcp
---
## Middleware
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
   name: middlewares.traefik.containo.us
spec:
   scope: Namespaced
   group: traefik.containo.us
   version: v1alpha1
   names:
      kind: Middleware
      plural: middlewares
      singular: middleware
---
apiVersion: apiextensions.k8s.io/v1beta1
```

```
kind: CustomResourceDefinition
metadata:
  name: tlsoptions.traefik.containo.us
spec:
  scope: Namespaced
  group: traefik.containo.us
  version: v1alpha1
  names:
    kind: TLSOption
    plural: tlsoptions
singular: tlsoption
```

14.1.1 创建 Traefik CRD 资源

```
[root@master-1 ~]#  kubectl apply -f traefik-crd.yaml
```

14.2 创建 Traefik RABC 文件

```
[root@master-1 ~]#  vi  traefik-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: kube-system
  name: traefik-ingress-controller
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
rules:
  - apiGroups: [""]
    resources: ["services","endpoints","secrets"]
    verbs: ["get","list","watch"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get","list","watch"]
  - apiGroups: ["extensions"]
    resources: ["ingresses/status"]
    verbs: ["update"]
  - apiGroups: ["traefik.containo.us"]
    resources: ["middlewares"]
    verbs: ["get","list","watch"]
  - apiGroups: ["traefik.containo.us"]
    resources: ["ingressroutes"]
    verbs: ["get","list","watch"]
  - apiGroups: ["traefik.containo.us"]
    resources: ["ingressroutetcps"]
    verbs: ["get","list","watch"]
  - apiGroups: ["traefik.containo.us"]
    resources: ["tlsoptions"]
    verbs: ["get","list","watch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: traefik-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-ingress-controller
subjects:
  - kind: ServiceAccount
    name: traefik-ingress-controller
namespace: kube-system
```

14.2.1 创建 RABC 资源

```
[root@master-1 ~]#  kubectl apply -f traefik-rbac.yaml
```

14.3 创建 Traefik ConfigMap

```
[root@master-1 ~]#    vi traefik-config.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: traefik-config
data:
  traefik.yaml: |-
    serversTransport:
      insecureSkipVerify: true
    api:
      insecure: true
      dashboard: true
      debug: true
    metrics:
      prometheus: ""
    entryPoints:
      web:
        address: ":80"
      websecure:
        address: ":443"
    providers:
      kubernetesCRD: ""
    log:
      filePath: ""
      level: error
      format: json
    accessLog:
      filePath: ""
      format: json
      bufferingSize: 0
      filters:
        retryAttempts: true
        minDuration: 20
      fields:
        defaultMode: keep
        names:
          ClientUsername: drop
        headers:
          defaultMode: keep
          names:
            User-Agent: redact
            Authorization: drop
            Content-Type: keep
```

14.3.1 创建 Traefik ConfigMap 资源配置

```
[root@master-1 ~]#    kubectl apply -f traefik-config.yaml -n kube-system
```

14.4 设置节点标签

#设置节点 label

```
[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
```

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

```
[root@master-1 ingress]# vi traefik-deploy.yaml
apiVersion: v1
kind: Service
metadata:
  name: traefik
```

```
spec:
    ports:
        - name: web
            port: 80
        - name: websecure
            port: 443
        - name: admin
            port: 8080
    selector:
        app: traefik
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
    name: traefik-ingress-controller
    labels:
        app: traefik
spec:
    selector:
        matchLabels:
            app: traefik
template:
    metadata:
        name: traefik
        labels:
            app: traefik
    spec:
        serviceAccountName: traefik-ingress-controller
        terminationGracePeriodSeconds: 1
        containers:
            - image: traefik:latest
                name: traefik-ingress-lb
                ports:
                    - name: web
                        containerPort: 80
                        hostPort: 80
                    - name: websecure
                        containerPort: 443
                        hostPort: 443
                    - name: admin
                        containerPort: 8080
                resources:
                    limits:
                        cpu: 2000m
                        memory: 1024Mi
                    requests:
                        cpu: 1000m
                        memory: 1024Mi
            securityContext:
                capabilities:
                    drop:
                        - ALL
                    add:
                        - NET_BIND_SERVICE
            args:
                - --configfile=/config/traefik.yaml
            volumeMounts:
                - mountPath: "/config"
                    name: "config"
    volumes:
        - name: config
            configMap:
                name: traefik-config
    tolerations:
        - operator: "Exists"
```

```
      nodeSelector:
        IngressProxy: "true"
```

14.5.1 部署 Traefik 资源

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

14.6 Traefik 路由配置

14.6.1 配置 Traefik Dashboard

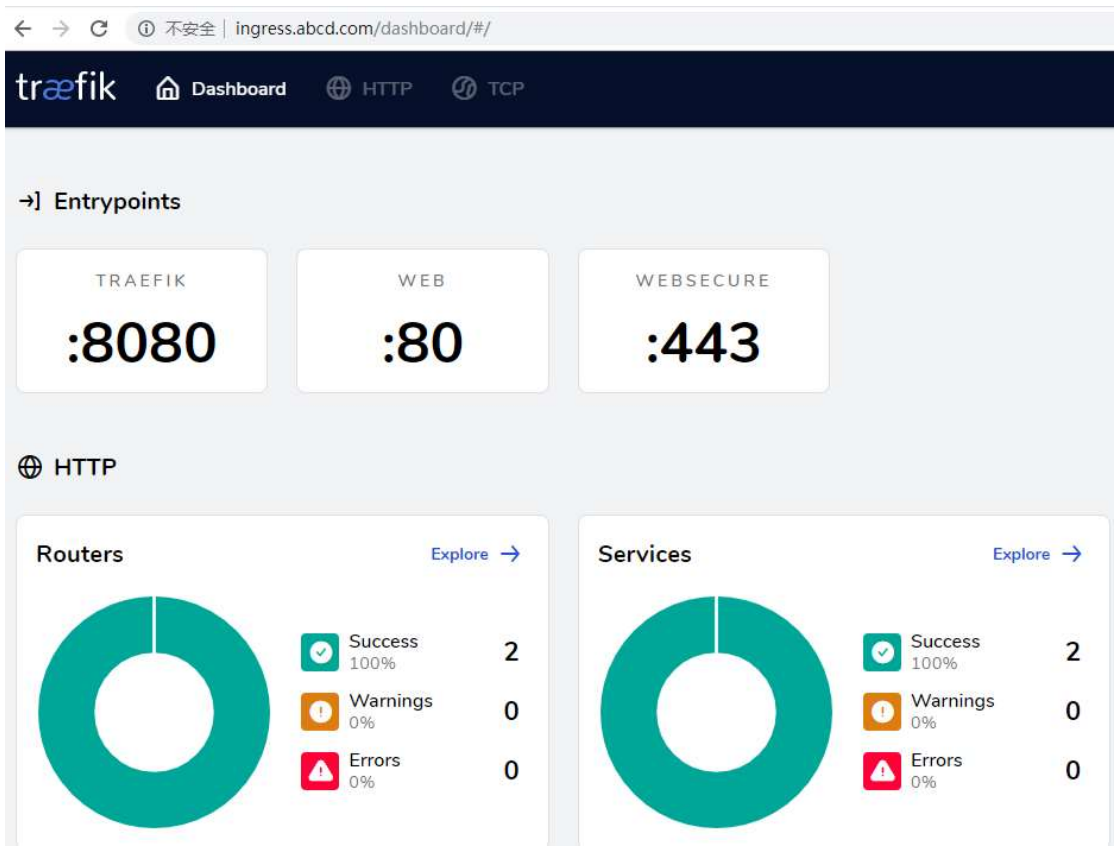
```
[root@master-1 ingress]#    vi traefik-dashboard-route.yaml
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: traefik-dashboard-route
spec:
  entryPoints:
    - web
  routes:
    - match: Host(`ingress. abcd.com`)
      kind: Rule
      services:
        - name: traefik
          port: 8080
```

14.6.2 客户端访问 Traefik Dashboard

14.6.2.1 绑定 Hosts 文件或者域名解析

192.168.91.146 ingress.abcd.com

14.6.2.2 访问 web



15 部署监控系统

#组件

#node-exporter alertmanager grafana kube-state-metrics Prometheus

#组件说明

MetricServer：是 kubernetes 集群资源使用情况的聚合器，收集数据给 kubernetes 集群内使用，如 kubectl,hpa,scheduler 等。

NodeExporter：用于各 node 的关键度量指标状态数据。

KubeStateMetrics：收集 kubernetes 集群内资源对象数据，制定告警规则。

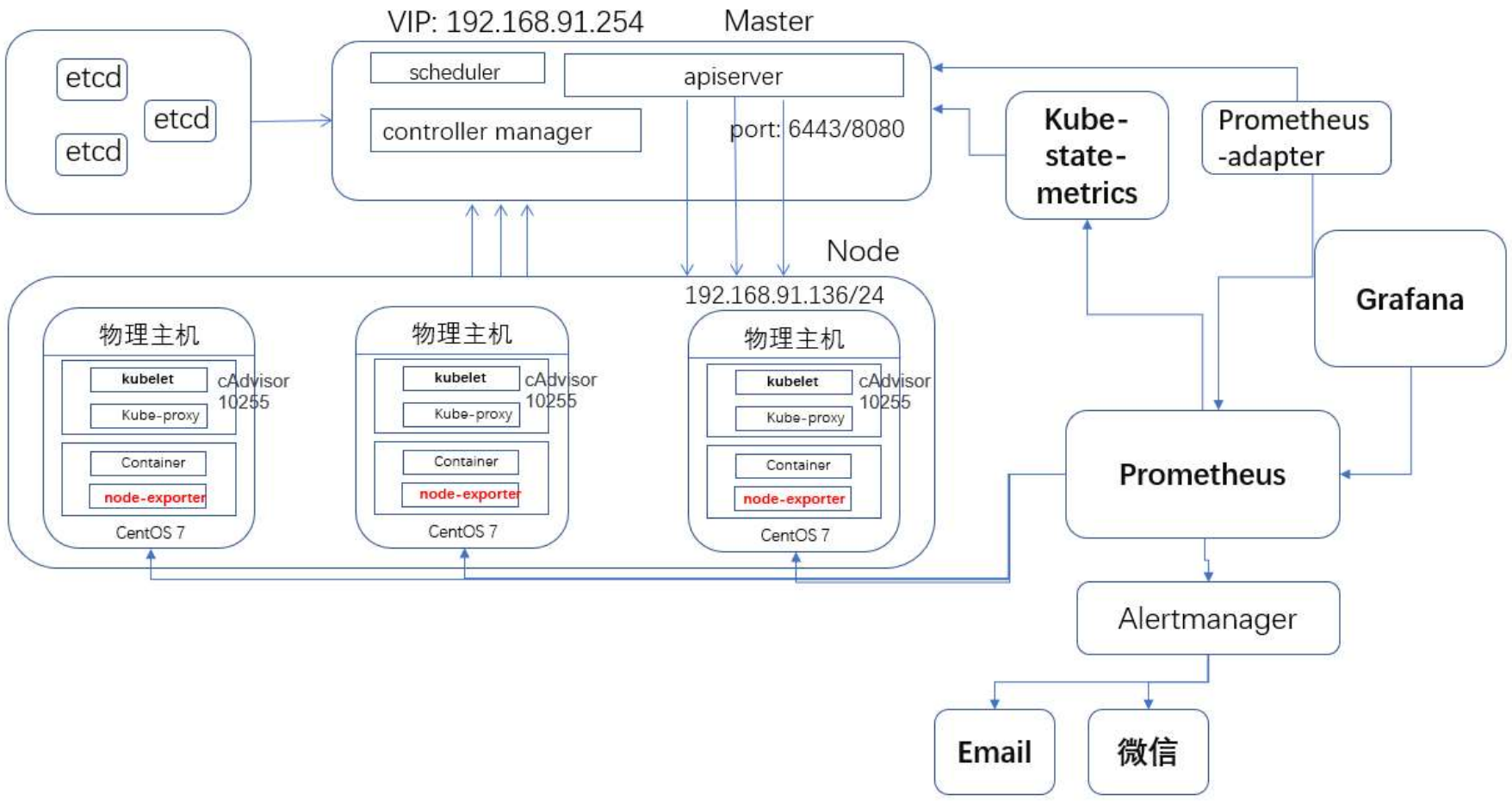
Prometheus -adapter: 自定义监控指标与容器指标

Prometheus：采用 pull 方式收集 apiserver，scheduler，controller-manager，kubelet 组件数据，通过 http 协议传输。

Grafana：是可视化数据统计和监控平台。

Alertmanager：实现短信或邮件报警。

读取数据流程:



15.1 安装 NFS 服务端

15.1.1 master 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    –r
```

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

```
[root@master-1 ~]# showmount -e 192.168.91.143
Export list for 192.168.91.143:
/ifs/kubernetes *
```

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 192.168.91.143
Export list for 192.168.91.143:
/ifs/kubernetes *
```

15.2.1 部署 PVC

Nfs 服务端地址需要修改

```
[root@master-1 ~]# 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
```

15.2.2 查看是否部署成功

```
[root@master-1 nfs]# kubectl get StorageClass
NAME                  PROVISIONER          AGE
managed-nfs-storage   fuseim.pri/ifs   33s
```

15.3 部署监控系统

#注意需要修改的配置文件

#修改 IP

ServiceMonitor/prometheus-EtcdService.yaml

ServiceMonitor/prometheus-KubeProxyService.yaml

ServiceMonitor/prometheus-kubeSchedulerService.yaml

```
[root@master-1 monitor]#  kubectl apply -f nfs/
[root@master-1 monitor]#  kubectl apply -f setup/
[root@master-1 monitor]#  kubectl apply -f alertmanager/
[root@master-1 monitor]#  kubectl apply -f node-exporter/
[root@master-1 monitor]#  kubectl apply -f kube-state-metrics/
[root@master-1 monitor]#  kubectl apply -f grafana/
[root@master-1 monitor]#  kubectl apply -f prometheus/
[root@master-1 monitor]#  kubectl apply -f serviceMonitor/
```

注意如果提示权限问题, 解决方法如下：

```
[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

```
#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
monitoring                  grafana-5dc77ff8cb-gwnqs                  1/1     Running   0          2d3h   172.17.64.10   192.168.91.146
```

15.3.2 获取 Grafana SVC

```
[root@master-1 ~]# kubectl get svc -A | grep grafana
monitoring                  grafana                      NodePort    10.0.0.57    <none>        3000:45523/TCP                  2d3h
```

15.3.3 登录 Grafana Dashboard

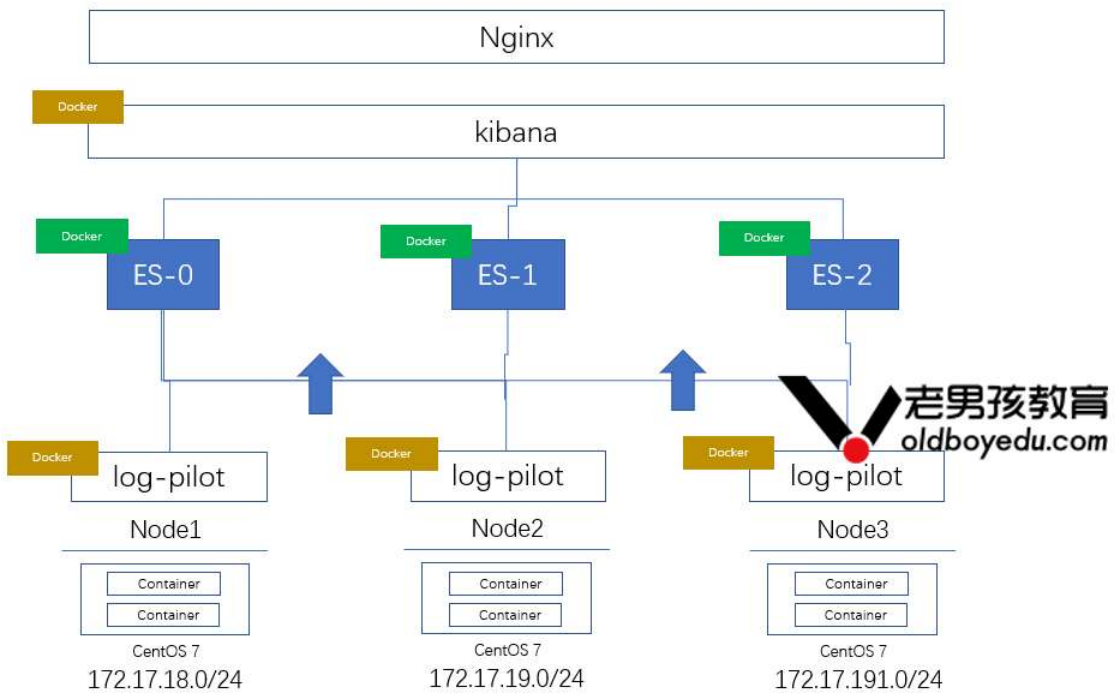
#用户与密码: admin/admin

15.3.4 选择资源

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

#设置 serviceAccount

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

8.7.1 配置权限

```
[root@master-1 logs]# cat es-rbac.yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: es-rbac
```

```
subjects:
   - kind: ServiceAccount
       name: admin
       namespace: kube-system
roleRef:
   kind: ClusterRole
   name: cluster-admin
   apiGroup: rbac.authorization.k8s.io
```

8.7.1 安装 Elasticsearch

```
[root@master-200 log]# docker pull registry.cn-hangzhou.aliyuncs.com/cqz/elasticsearch:5.5.1


[root@master-200 log]# wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/elasticsearch.yml (需要修改内存大小)
[root@master-200 log]# kubectl apply -f elasticsearch.yml
```

8.7.2 查看 ES 在 Kubernetes 中的状态

#最好有三个 ES 节点

```
[root@master-200 log]# kubectl get StatefulSet -n kube-system
NAME                                    READY   STATUS              RESTARTS   AGE
elasticsearch-0                         0/1     PodInitializing     0          91s
kubernetes-dashboard-69dcdb65fd-psnq9   1/1     Running             1          32h
```

8.7.3 查看 ES 状态

```
[root@master-200 log]#   kubectl exec -it   elasticsearch-0   bash   -n   kube-system
#执行检查命令:
#curl http://localhost:9200/_cat/health?v
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
```

8.7.4 安装 log-pilot

```
[root@master-200 log]#   wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/log-pilot.yml
[root@master-200 log]#   docker pull registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9-filebeat
#部署
[root@master-200 log]#   kubectl apply -f log-pilot.yml
```

8.7.5 安装 kibana

#注意修改命名空间

```
[root@master-200 log]#   wget https://acs-logging.oss-cn-hangzhou.aliyuncs.com/kibana.yml
#部署
[root@master-200 log]#   kubectl apply -f kibana.yml
```

8.7.6 访问 Kibana 界面

8.7.6.1 获取 Kibana 节点

```
[root@master-200 log]# kubectl get pods -o wide --all-namespaces
kube-system   kibana-9fd8b85c-cq8v4   1/1     Running   0          25s     172.17.70.7     192.168.91.203   <none>
```

8.7.6.2 获取 Kibana HostPort 节点

```
[root@master-200 log]# kubectl get svc --all-namespaces
NAMESPACE     NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                  AGE
kube-system   kibana       NodePort    10.0.0.126     <none>        80:39191/TCP             2m33s
```

#查看 ES 中是否有建立索引

```
elasticsearch@elasticsearch-0:/usr/share/elasticsearch$ curl 'localhost:9200/_cat/indices?v'
health status index                            uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   .kibana                          dVhG5gNUQk6cwdO_fsXk2w   1   1          1            0      6.2kb          3.1kb
green  open   passport-logs-2019.11.26 OTiUYHVrQe2DrxLg28VbBQ   5   1        964            0      2.3mb          1.3mb
```

8.7.6.3 访问 web 界面:

```
http://192.168.91.203:39191
```

8.7.6.4 案例一:运行容器收集日志

1. 创建 nginx yaml 文件

```
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
```

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

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
```

3. 绑定 hosts 或者 使用 services 访问
kubectl run -it --rm --restart=Never --image=infoblox/dnstools:latest dnstools

3.1 查看是否建立索引

```
[root@master-200 log]#kubectl get pods -n=kube-system
[root@master-200 log]#kubectl exec -it elasticsearch-0 /bin/bash -n kube-system
```

```
curl 'localhost:9200/_cat/indices?v'
```

4.注意多行日志收集

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