# Table of Contents


Introduction 1.1

k8sv1.10.0安装文档 1.2

- k8s v.1.10.0 二进制安装 1.2.1
- calico网络设置 1.2.2
- kubedns配置 1.2.3
- kube-dashboard 1.2.4
- nginx-ingress 1.2.5
- kube-route 1.2.6
- node部署初始化脚本 1.2.7
- node批量部署的api账户一些自动策略设置 1.2.8
- haproxy + keepalived 配置ingress-nginx 的前端代理 1.2.9

## k8s 安装文档

### 一些简介

```
本文档采用k8s v1.10.0二进制的集群部署方式，主要更改 使用kube-router 代理kube-proxy,使用ingress-nginx做边缘负载,使用haproxy+heartbeat实现高可用
```


### 一. k8s 软件版本选择

```
 系统版本 centos 7 64
docker 1.13.1
k8s 系列版本 v1.10.0
ip地址段划分:
  cluster-pod—ip: 10.228.0.0/16  #pod ip   --cluster-cidr
  calico—subnet-range: 10.229.0.0/16 # service ip  --service-cluster-ip-range
  vip:  10.255.72.199
```

### 二. 系统环境

| ip | hostname | 角色 |
| --- | --- | --- |
| 10.255.72.189 | FJR-bt-kvm-72-189 | apiserver,controller,scheduler,kubelet,kube-proxy,etcd |
| 10.255.72.190 | FJR-bt-kvm-72-190 | apiserver,controller,scheduler,kubelet,kube-proxy,etcd |
| 10.255.72.191 | FJR-bt-kvm-72-191 | apiserver,controller,scheduler,kubelet,kube-proxy,etcd |
| 10.255.72.192 | FJR-bt-kvm-72-192 | kubelet,kube-proxy |

### 三. 系统需求

- **1. host解析:**

```
10.255.72.189   kube-master01
10.255.72.190   kube-master02
10.255.72.191   kube-master03
10.255.72.192   kube-minon02
10.255.72.189   etcd01
10.255.72.190   etcd02
10.255.72.191   etcd03
```

- **2. 关闭系统selinux:**

  /etc/selinux/config `SELINUX=disabled` `setenforce 0`

- **3. 关闭系统swap:**

  swapoff -a 注释掉 `/etc/fstab` 下的所有的swap配置

- **4. 添加内核参数:**

```
echo "
net.ipv4.ip_forward = 1
```

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
" > /etc/sysctl.d/k8s.conf
sysctl -p
```

- **5. 系统更新到最新**

```
yum update -y && yum upgrade -y
```

- **6. 安装docker**

```
yum install docker -y
```

**四.安装etcd+tls集群环境**

| 角色 | ip地址 | 配置主机名 | 客户端端口 | 集群端口 |
| --- | --- | --- | --- | --- |
| etcd01 | 10.255.72.189 | etcd01 | 2379 | 2380 |
| etcd02 | 10.255.72.190 | etcd02 | 2379 | 2380 |
| etcd03 | 10.255.72.191 | etcd03 | 2379 | 2380 |

- **1. 安装cfssl工具:**

```
go get -u github.com/cloudflare/cfssl/cmd/...
cp go/bin/cfssl* /usr/local/bin/
chmod +x /usr/local/bin/cfssl
```

- **2. 创建etcd tls证书:**

```
cd /root/ ; mkdir ssl && cd ssl/
```

> ca-config.json

```
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
      "fengjr": {
        "usages": [
            "signing",
            "key encipherment",
            "server auth",
            "client auth"
        ],
```

```
        "expiry": "87600h"
      }
    }
  }
}
```

ca-csr.json

```
{
  "CN": "fengjr",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "fengjr",
      "OU": "cloudnative"
    }
  ]
}
```

**创建ca证书**

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca
```

etcd-csr.json

```
{
    "CN": "fengjr",
    "hosts": [
      "127.0.0.1",
      "10.255.72.189",
      "10.255.72.190",
      "10.255.72.191",
      "etcd01",
      "etcd02",
      "etcd03"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
```

```
            "C": "CN",
            "ST": "BeiJing",
            "L": "BeiJing",
            "O": "fengjr",
            "OU": "cloudnative"
        }
    ]
}
```

**创建etcd证书**

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fen
gjr etcd-csr.json | cfssljson -bare etcd
```

**查看证书的信息**

```
cfssl-certinfo -cert etcd.pem
```

- **3. 二进制文件下载**

```
curl -LO https://github.com/coreos/etcd/releases/download/v3.2.9/etcd-v3.2.9-li
nux-amd64.tar.gz
tar zxvf etcd-v3.2.9-linux-amd64.tar.gz
cd etcd-v3.2.9-linux-amd64 && mv bin/etcd* /usr/bin/ && chmod +x /usr/bin/etcd*
#拷贝配置文件
mkdir -p /etc/etcd/ssl && cp /root/ssl/etcd*.pem /etc/etcd/ssl/
cp /root/ssl/ca.pem /etc/etcd/ssl/
```

- **4. 系统service文件**

export nodeip=10.255.72.189 ~~export nodeip=10.255.72.190~~ ~~export nodeip=10.255.72.191~~

```
  echo "
  [Unit]
  Description=etcd server
  After=network.target
  After=network-online.target
  Wants=network-online.target

  [Service]
  Type=notify
  WorkingDirectory=/var/lib/etcd/
  EnvironmentFile=-/etc/etcd/etcd.conf
  ExecStart=/usr/bin/etcd \
    --name etcd01 \
    --cert-file=/etc/etcd/ssl/etcd.pem \
    --key-file=/etc/etcd/ssl/etcd-key.pem \
    --peer-cert-file=/etc/etcd/ssl/etcd.pem \
```

```
    --peer-key-file=/etc/etcd/ssl/etcd-key.pem \
    --trusted-ca-file=/etc/etcd/ssl/ca.pem \
    --peer-trusted-ca-file=/etc/etcd/ssl/ca.pem \
    --initial-advertise-peer-urls https://${nodeip}:2380 \
    --listen-peer-urls https://${nodeip}:2380 \
    --listen-client-urls https://${nodeip}:2379,https://127.0.0.1:2379 \
    --advertise-client-urls https://${nodeip}:2379 \
    --initial-cluster-token etcd-cluster-1 \
    --initial-cluster etcd01=https://10.255.72.189:2380,etcd02=https://10.255.7
2.190:2380,etcd03=https://10.255.72.191:2380 \
    --initial-cluster-state new \
    --data-dir=/var/lib/etcd
  Restart=on-failure
  RestartSec=5
  LimitNOFILE=65536

  [Install]
  WantedBy=multi-user.target
  " >/usr/lib/systemd/system/etcd.service
```

- **5. 启动etcd 集群**

```
  systemctl daemon-reload
  systemctl start etcd.service
  systemctl status etcd.service
```

- **6. 查看集群健康状态**

```
etcdctl \
--ca-file=/etc/etcd/ssl/ca.pem \
--cert-file=/etc/etcd/ssl/etcd.pem \
--key-file=/etc/etcd/ssl/etcd-key.pem \
--endpoints=https://10.255.72.189:2379,https://10.255.72.190:2379,https://10.25
5.72.191:2379 cluster-health
```

**输出结果为:**

```
member 11561ab2384b6c6d is healthy: got healthy result from https://10.255.72.1
91:2379
member 5b2cf22137bb72af is healthy: got healthy result from https://10.255.72.1
90:2379
member c88f6852e21dbb31 is healthy: got healthy result from https://10.255.72.1
89:2379
cluster is healthy
```

**五. 在10.255.72.{189,190,191}下载kubernetes二进制文件,并复制到/usr/local/bin/下,在10.255.72.192上只需要kubelet二进制文件**

```
curl - LO https://dl.k8s.io/v1.10.0/kubernetes-server-linux-amd64.tar.gz
tar zxvf kubernetes-server-linux-amd64.tar.gz
cd kubernetes/server/bin/
cp kube-apiserver kube-controller-manager kube-scheduler kubelet /usr/local/bin/
```

### 六.配置kube-apiserver集群

| 角色 | ip地址 |
| --- | --- |
| kube-apiserver01 | 10.255.72.189 |
| kube-apiserver02 | 10.255.72.190 |
| kube-apiserver03 | 10.255.72.191 |

- **1.生成tls证书**

```
cd /root/ssl/ && cat apiserver-csr.json
```

> apiserver-csr.json

```
{
 "CN": "kubernetes",
 "hosts": [
   "127.0.0.1",
   "10.255.72.189",
   "10.255.72.190",
   "10.255.72.191",
   "10.255.72.192",
   "10.255.72.199",
   "10.228.0.1",
   "kubernetes",
   "kubernetes.default",
   "kubernetes.default.svc",
   "kubernetes.default.svc.cluster",
   "kubernetes.default.svc.cluster.local"
 ],
 "key": {
     "algo": "rsa",
     "size": 2048
 },
 "names": [
     {
         "C": "CN",
         "ST": "BeiJing",
         "L": "BeiJing",
         "O": "k8s",
         "OU": "cloudnative"
     }
 ]
```

```
}
```

**生成证书的命令：**

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fengjr
apiserver-csr.json | cfssljson -bare apiserver
```

- **2.配置kube-apiserver证书文件**

```
mkdir -p /etc/kubernetes/ssl/
cd /root/ssl/
cp apiserver-key.pem apiserver.pem ca.pem ca-key.pem /etc/kubernetes/ssl
```

  - **3.生成token文件**

```
cd /etc/kubernetes/
export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
cat > token.csv < <EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"
EOF
```

- **4.配置kube-apiserver的service文件**

export nodeip=10.255.72.189 ~~export nodeip=10.255.72.190~~ ~~export nodeip=10.255.72.191~~

```
echo "
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
User=root
ExecStart=/usr/local/bin/kube-apiserver \
 --admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStora
geClass,ResourceQuota \
 --advertise-address=${nodeip} \
 --allow-privileged=true \
 --apiserver-count=3 \
 --audit-log-maxage=30 \
 --audit-log-maxbackup=3 \
 --audit-log-maxsize=100 \
 --audit-log-path=/var/lib/audit.log \
 --authorization-mode=Node,RBAC \
 --bind-address=${nodeip}  \
 --secure-port=6442 \
 --client-ca-file=/etc/kubernetes/ssl/ca.pem \
 --enable-swagger-ui=true \
 --etcd-cafile=/etc/kubernetes/ssl/ca.pem \
 --etcd-certfile=/etc/kubernetes/ssl/etcd.pem \
```

```
 --etcd-keyfile=/etc/kubernetes/ssl/etcd-key.pem \
 --etcd-servers=https://10.255.72.189:2379,https://10.255.72.190:2379,https://1
0.255.72.191:2379 \
 --event-ttl=1h \
 --kubelet-https=true \
 --insecure-bind-address=${nodeip}  \
 --runtime-config=rbac.authorization.k8s.io/v1alpha1 \
 --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem \
 --service-cluster-ip-range=10.228.0.0/16 \
 --service-node-port-range=30000-37000 \
 --tls-cert-file=/etc/kubernetes/ssl/apiserver.pem \
 --tls-private-key-file=/etc/kubernetes/ssl/apiserver-key.pem \
 --enable-bootstrap-token-auth \
 --token-auth-file=/etc/kubernetes/token.csv \
 --v=2
Restart=on-failure
RestartSec=5
Type=notify
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
">/usr/bin/systemd/system/kube-apiserver.service
```

- **5. 启动并查看apiserver的集群状态**

```
systemctl daemon-reload
systemctl start kube-apiserver.service
systemctl status kube-apiserver.service
```

- **6. 创建.kube/config**

  **创建admin证书**

  > admin-csr.json

```
{
    "CN": "kubernetes-admin",
    "hosts": [
            "10.255.72.189",
            "10.255.72.190",
            "10.255.72.191",
            "10.255.72.192"
    ],
    "key": {
      "algo": "rsa",
      "size": 2048
    },
    "names": [
      {
```

```
        "C": "CN",
        "ST": "BeiJing",
        "L": "BeiJing",
        "O": "system:masters",
        "OU": "cloudnative"
      }
    ]
}
```

**生成证书**

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fen
gjr admin-csr.json | cfssljson -bare admin
cp /root/ssl/admin.pem /root/ssl/admin-key.pem /etc/kubernetes/ssl/
```

> 生成配置文件

```
cd /etc/kubernetes
export KUBE_APISERVER="https://10.255.72.199:6443"

# set-cluster
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=admin.conf

# set-credentials
kubectl config set-credentials kubernetes-admin \
  --client-certificate=/etc/kubernetes/ssl/admin.pem \
  --embed-certs=true \
  --client-key=/etc/kubernetes/ssl/admin-key.pem \
  --kubeconfig=admin.conf

# set-context
kubectl config set-context kubernetes-admin@kubernetes \
  --cluster=kubernetes \
  --user=kubernetes-admin \
  --kubeconfig=admin.conf

# set default context
kubectl config use-context kubernetes-admin@kubernetes --kubeconfig=admin.conf

# mv
cp admin.conf ~/.kube/config
```

```
kubectl get cs
```

- **7. 配置haproxy+keepalive 搭建高可用**

```
yum install -y keepalived haproxy
```

> /etc/haproxy/haproxy.conf

```
global
      daemon
      maxconn 40960
      stats socket /etc/haproxy/haproxy.stats level operator
      stats timeout 2m
      log 127.0.0.1 local0

defaults
        log global
        mode http
        retries 3
        option redispatch
        timeout connect 60000ms
        timeout client 60000ms
        timeout server 60000ms

listen  kube-api *:6443 #代理配置
        mode tcp
        maxconn 4000
        balance roundrobin
        server  FJR-bt-kvm-72-189 10.255.72.189:6442 check inter 1500 fall 1 ri
se 2
        server  FJR-bt-kvm-72-190 10.255.72.190:6442 check inter 1500 fall 1 ri
se 2
        server  FJR-bt-kvm-72-191 10.255.72.191:6442 check inter 1500 fall 1 ri
se 2

listen stats *:7744 #status 状态配置
       mode http
       option httpclose
       balance roundrobin
       stats uri /
       stats realm Haproxy\ Statistics
       stats auth admin:nop@ss.1
       stats admin if TRUE
```

> /etc/keepalived/keepalived.conf

```
global_defs {
}
vrrp_script chk_haproxy {
        script "/etc/keepalived/check_haproxy.sh"
        interval 2
```

```
        weight 2
        }
vrrp_instance VIP_1 {
          state BACKUP #配合不抢占一起配置。
          interface eth0
          virtual_router_id 50 #统一标示为一组广播的消息
          priority 100 #权重,只影响vip的抢占优先级,每节点设置不一样,这里每节点加50权重。
          advert_int 1
          nopreempt #不开启抢占,节省ip替换的耗费
          authentication {
            auth_type PASS
            auth_pass 321654
          }
  track_interface {
          eth0
  }
  virtual_ipaddress {
          10.255.72.199
  }
  track_script {
          chk_haproxy
  }
}
```

> /etc/keepalived/check_haproxy.sh

```
#!/bin/bash
A=`ps -C haproxy --no-header |wc -l`
if [ $A -eq 0 ];then
  service haproxy start
  sleep 3
  if [ `ps -C haproxy --no-header |wc -l` -eq 0 ];then
    service keepalived stop
  fi
fi
```

> 启动服务:

```
service haproxy start
service keepalived start
ip a
netstat -lnpt
```

- **8. 配置kube-apiserver的log归集**

```
vim /etc/rsyslog.conf
#添加配置 在messges log的定义之前。其他应用也一样。更改appname即可
$template app-template,"/var/log/%app-name%/%app-name%_%$YEAR%-%$MONTH%-%$DAY%.
```

```
log"
if ( $app-name == "kube-apiserver") then {
                action(type="omfile" DynaFile="app-template")
                stop
}
```

### 七.配置kube-controller-manager

| ip | 角色 |
| --- | --- |
| 10.255.72.189 | controller01 |
| 10.255.72.190 | controller02 |
| 10.255.72.191 | controller03 |

- **1. 配置tls证书**

> controller-manager-csr.json

```
{
"CN": "system:kube-controller-manager",
"hosts": [
        "10.255.72.189",
        "10.255.72.190",
        "10.255.72.191",
        "10.255.72.192"
],
"key": {
  "algo": "rsa",
  "size": 2048
},
"names": [
  {
    "C": "CN",
    "ST": "BeiJing",
    "L": "BeiJing",
    "O": "system:kube-controller-manager",
    "OU": "cloudnative"
  }
]
}
```

**生成证书**

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fen
gjr controller-manager-csr.json | cfssljson -bare controller-manager
```

- **2. 移动证书文件**

```
cp controller-manager-key.pem controller-manager.pem /etc/kubernetes/ssl/
scp controller-manager-key.pem controller-manager.pem  root@10.255.72.190:/etc/
kubernetes/ssl/
scp controller-manager-key.pem controller-manager.pem  root@10.255.72.191:/etc/
kubernetes/ssl/
```

- **3. 生成controller-manager的kubeconfig文件controller-manager.conf**

```
  cd /etc/kubernetes
  export KUBE_APISERVER="https://10.255.72.199:6443"

  # set-cluster
  kubectl config set-cluster kubernetes \
    --certificate-authority=/etc/kubernetes/ssl/ca.pem \
    --embed-certs=true \
    --server=${KUBE_APISERVER} \
    --kubeconfig=controller-manager.conf

  # set-credentials
  kubectl config set-credentials system:kube-controller-manager \
    --client-certificate=/etc/kubernetes/ssl/controller-manager.pem \
    --embed-certs=true \
    --client-key=/etc/kubernetes/ssl/controller-manager-key.pem \
    --kubeconfig=controller-manager.conf
f
  # set-context
  kubectl config set-context system:kube-controller-manager@kubernetes \
    --cluster=kubernetes \
    --user=system:kube-controller-manager \
    --kubeconfig=controller-manager.conf

  # set default context
  kubectl config use-context system:kube-controller-manager@kubernetes --kubeconfig
=controller-manager.con
```

- **4.配置controller-manager的service文件**

  每节点配置一样

```
  echo "
  [Unit]
  Description=Kubernetes Controller Manager
  Documentation=https://github.com/GoogleCloudPlatform/kubernetes

  [Service]
  ExecStart=/usr/local/bin/kube-controller-manager \
            --master=https://10.255.72.199:6443 \
            --kubeconfig=/etc/kubernetes/controller-manager.conf \
            --cluster-name=kubernetes \
```

```
            --cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \
            --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem \
            --service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem \
            --root-ca-file=/etc/kubernetes/ssl/ca.pem \
            --service-cluster-ip-range=10.229.0.0/16 \
            --cluster-cidr=10.228.0.0/16 \
            --leader-elect=true \
            --controllers=*
LimitNOFILE=65536
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target" > /usr/lib/systemd/system/kube-controller-manager.s
ervice
```

- **5.启动服务**

```
systemctl daemon-reload
systemctl start kube-controller-manager
```

**八.配置scheduler服务集群**

| ip | 角色 |
| --- | --- |
| 10.255.72.189 | scheduler01 |
| 10.255.72.190 | scheduler02 |
| 10.255.72.191 | scheduler03 |

- **1. 配置tls证书**

> scheduler-csr.json

```
{
  "CN": "system:kube-scheduler",
  "hosts": [
          "10.255.72.189",
          "10.255.72.190",
          "10.255.72.191",
          "10.255.72.192"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
```

```
      "L": "BeiJing",
      "O": "system:kube-scheduler",
      "OU": "cloudnative"
    }
  ]
}
```

> 生成证书文件

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fen
gjr scheduler-csr.json | cfssljson -bare scheduler
```

- **2.移动证书文件**

```
cp scheduler-key.pem scheduler.pem /etc/kubernetes/ssl/
scp scheduler-key.pem scheduler.pem  root@10.255.72.190:/etc/kubernetes/ssl/
scp scheduler-key.pem scheduler.pem  root@10.255.72.191:/etc/kubernetes/ssl/
```

- **3.生成kube-scheduler的kubeconfig文件scheduler.conf**

```
cd /etc/kubernetes
export KUBE_APISERVER="https://10.255.72.199:6443"

# set-cluster
kubectl config set-cluster kubernetes \
--certificate-authority=/etc/kubernetes/ssl/ca.pem \
--embed-certs=true \
--server=${KUBE_APISERVER} \
--kubeconfig=scheduler.conf

# set-credentials
kubectl config set-credentials system:kube-scheduler \
--client-certificate=/etc/kubernetes/ssl/scheduler.pem \
--embed-certs=true \
--client-key=/etc/kubernetes/ssl/scheduler-key.pem \
--kubeconfig=scheduler.conf

# set-context
kubectl config set-context system:kube-scheduler@kubernetes \
--cluster=kubernetes \
--user=system:kube-scheduler \
--kubeconfig=scheduler.conf

# set default context
kubectl config use-context system:kube-scheduler@kubernetes --kubeconfig=schedu
ler.conf
```

- **4.创建scheduler的service文件**

~~每节点配置一样~~

> /usr/lib/systemd/system/kube-scheduler.service

```
[Unit]
Description=Kubernetes Scheduler
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
ExecStart=/usr/local/bin/kube-scheduler \
--master=https://10.255.72.199:6443 \
--kubeconfig=/etc/kubernetes/scheduler.conf \
--leader-elect=true \
--v=2
LimitNOFILE=65536
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

- **5.启动服务集群**

```
systemctl daemon-reload
systemctl start kube-scheduler.service
```

**九.部署kubelet客户端节点**

~~以10.255.72.189节点为例~~

- **1.生成kubelet客户端证书**

> kubelet-csr-189.json

```
{
"CN": "system:node:kube-master01",
"hosts": [
      "kube-master01",
      "10.255.72.189"
],
"key": {
  "algo": "rsa",
  "size": 2048
},
"names": [
  {
    "C": "CN",
    "ST": "BeiJing",
    "L": "BeiJing",
  "O": "system:nodes",
```

```
    "OU": "cloudnative"
  }
]
}
```

> 生成证书文件

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fen
gjr kubelet-csr-189.json | cfssljson -bare kubelet-189
cp kubelet-189.pem  /etc/kubernetes/ssl/kubelet.pem
cp kubectl-189-key.pem /etc/kubernetes/ssl/kubelet-key.pem
```

- **2.创建 kubelet bootstrapping kubeconfig 文件**

```
cd /etc/kubernetes
export KUBE_APISERVER="https://10.255.72.199:6443"

# 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
```

- **3. 创建kubelet申请证书的权限用户**

```
cd /etc/kubernetes
kubectl create clusterrolebinding kubelet-bootstrap \
--clusterrole=system:node-bootstrapper \
--user=kubelet-bootstrap
```

- **--user=kubelet-bootstrap 是在 /etc/kubernetes/token.csv 文件中指定的用户名，同时也写入了 /etc/kubernetes/bootstrap.kubeconfig 文件**

- **4. 创建kubelet的service文件**

  **创建必要的目录，不然服务启动会失败。**

```
mkdir -p /etc/cni/net.d
mkdir -p /opt/cni/bin/
mkdir -p  /var/lib/kubelet
```

```
[Unit]
Description=Kubernetes Kubelet
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
ExecStart=/usr/local/bin/kubelet \
  --runtime-cgroups=/systemd/system.slice \
  --kubelet-cgroups=/systemd/system.slice \
  --cgroup-driver=systemd \
  --address=10.255.72.189 \
  --hostname-override=kube-master01 \
  --pod-infra-container-image=hub-dev.fengjr.com/k8s/pause-amd64:3.0 \ #公司自建的
仓库 搭建参考 harbor[https://github.com/vmware/harbor]
  --bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \
  --kubeconfig=/etc/kubernetes/kubelet.kubeconfig \
  --pod-manifest-path=/etc/kubernetes/manifests \
  --cert-dir=/etc/kubernetes/ssl \
  --cluster-dns=10.229.0.10 \
  --cluster-domain=cluster.local. \
  --allow-privileged=true \
  --rotate-certificates=true \
  --fail-swap-on=false \
  --serialize-image-pulls=false \
  --network-plugin=cni \
  --cni-conf-dir=/etc/cni/net.d \
  --cni-bin-dir=/opt/cni/bin \
  --max-pods=110 \
  --v=2
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

- **4. 启动服务**

```
systemctl daemon-reload
systemctl start kubelet.service
```

- **5.证书的查看和批准**

```
kubectl get csr
kubectl certificate approve $csr
kubectl get nodes #查看加入的节点
```

## 十.部署kube-proxy

- **1.配置ssl证书**

  kube-proxy-csr.json

```
{
  "CN": "system:kube-proxy",
  "hosts": [
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "BeiJing",
      "L": "BeiJing",
      "O": "system:kube-proxy",
      "OU": "cloudnative"
    }
  ]
}
```

- **2.生成证书文件**

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=fengjr
kube-proxy-csr.json | cfssljson -bare kube-proxy
```

- **3.生成kubeconfig文件kube-proxy.conf：**

```
cd /etc/kubernetes
export KUBE_APISERVER="https://10.255.72.199:6443"
# set-cluster
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.conf
# set-credentials
kubectl config set-credentials system:kube-proxy \
```

```
  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \
  --embed-certs=true \
  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \
  --kubeconfig=kube-proxy.conf
# set-context
kubectl config set-context system:kube-proxy@kubernetes \
  --cluster=kubernetes \
  --user=system:kube-proxy \
  --kubeconfig=kube-proxy.conf
# set default context
kubectl config use-context system:kube-proxy@kubernetes --kubeconfig=kube-proxy.con
f
```

- **4.配置kube-proxy的系统service 文件**

**注意： 创建/var/lib/kube-proxy 目录，如果不存在 kube-proxy会起不来**

```
mkdir /var/lib/kube-proxy
```

> /usr/lib/systemd/system/kube-proxy.service

```
[Unit]
Description=kube-proxy
After=network.target

[Service]
WorkingDirectory=/var/lib/kube-proxy
EnvironmentFile=-/etc/kubernetes/kube-proxy
ExecStart=/usr/local/bin/kube-proxy \
    --logtostderr=true \
    --v=0 \
    --bind-address=10.255.72.189 \
    --kubeconfig=/etc/kubernetes/kube-proxy.kubeconfig \
    --cluster-cidr=10.229.0.0/16

Restart=on-failure

[Install]
WantedBy=multi-user.target
```

```
systemctl enable kube-proxy.service
systemctl start kube-proxy.service
```

### 十一.部署calico 网络插件

这里采用daemonset的方式部署:

> calico.yaml

```
# Calico Version v2.3.0
# http://docs.projectcalico.org/v2.3/releases#v2.3.0
# This manifest includes the following component versions:
#   calico/node:v1.3.0
#   calico/cni:v1.9.1
#   calico/kube-policy-controller:v0.6.0

# This ConfigMap is used to configure a self-hosted Calico installation.
kind: ConfigMap
apiVersion: v1
metadata:
  name: calico-config
  namespace: kube-system
data:
  # The location of your etcd cluster.  This uses the Service clusterIP
  # defined below.
  etcd_endpoints: "http://10.228.0.222:6666"

  # Configure the Calico backend to use.
  calico_backend: "bird"

  # The CNI network configuration to install on each node.
  cni_network_config: |-
    {
        "name": "k8s-pod-network",
        "cniVersion": "0.1.0",
        "type": "calico",
        "etcd_endpoints": "__ETCD_ENDPOINTS__",
        "log_level": "info",
        "ipam": {
            "type": "calico-ipam"
        },
        "policy": {
            "type": "k8s",
             "k8s_api_root": "https://__KUBERNETES_SERVICE_HOST__:__KUBERNETES_SERVICE_PORT__",
             "k8s_auth_token": "__SERVICEACCOUNT_TOKEN__"
        },
        "kubernetes": {
            "kubeconfig": "/etc/cni/net.d/__KUBECONFIG_FILENAME__"
        }
    }


# This manifest installs the Service which gets traffic to the Calico
# etcd.
apiVersion: v1
kind: Service
metadata:
  labels:
```

```
    k8s-app: calico-etcd
  name: calico-etcd
  namespace: kube-system
spec:
  # Select the calico-etcd pod running on the master.
  selector:
    k8s-app: calico-etcd
  # This ClusterIP needs to be known in advance, since we cannot rely
  # on DNS to get access to etcd.
  clusterIP: 10.228.0.222
  ports:
    - port: 6666

---

# This manifest installs the calico/node container, as well
# as the Calico CNI plugins and network config on
# each master and worker node in a Kubernetes cluster.
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: calico-node
  namespace: kube-system
  labels:
    k8s-app: calico-node
spec:
  selector:
    matchLabels:
      k8s-app: calico-node
  template:
    metadata:
      labels:
        k8s-app: calico-node
      annotations:
        # Mark this pod as a critical add-on; when enabled, the critical add-on sch
eduler
        # reserves resources for critical add-on pods so that they can be reschedul
ed after
        # a failure.  This annotation works in tandem with the toleration below.
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      hostNetwork: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      # Allow this pod to be rescheduled while the node is in "critical add-ons onl
y" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      - key: CriticalAddonsOnly
        operator: Exists
      serviceAccountName: calico-cni-plugin
```

```
      containers:
        # Runs calico/node container on each Kubernetes node.  This
        # container programs network policy and routes on each
        # host.
        - name: calico-node
          image: harbor.fengjr.com/k8s/node:v1.3.0
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # Enable BGP.  Disable to enforce policy only.
            - name: CALICO_NETWORKING_BACKEND
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: calico_backend
            # Disable file logging so `kubectl logs` works.
            - name: CALICO_DISABLE_FILE_LOGGING
              value: "true"
            # Set Felix endpoint to host default action to ACCEPT.
            - name: FELIX_DEFAULTENDPOINTTOHOSTACTION
              value: "ACCEPT"
            # Configure the IP Pool from which Pod IPs will be chosen.
            - name: CALICO_IPV4POOL_CIDR
              value: "10.229.0.0/16"
            - name: CALICO_IPV4POOL_IPIP
              value: "always"
            # Disable IPv6 on Kubernetes.
            - name: FELIX_IPV6SUPPORT
              value: "false"
            # Set Felix logging to "info"
            - name: FELIX_LOGSEVERITYSCREEN
              value: "info"
            # Auto-detect the BGP IP address.
            - name: IP
              value: ""
          securityContext:
            privileged: true
          resources:
            requests:
              cpu: 300m
          volumeMounts:
            - mountPath: /lib/modules
              name: lib-modules
              readOnly: true
            - mountPath: /var/run/calico
              name: var-run-calico
              readOnly: false
```

```
        # This container installs the Calico CNI binaries
        # and CNI network config file on each node.
        - name: install-cni
          image: harbor.fengjr.com/k8s/cni:v1.9.1
          command: ["/install-cni.sh"]
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # The CNI network config to install on each node.
            - name: CNI_NETWORK_CONFIG
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: cni_network_config
          volumeMounts:
            - mountPath: /host/opt/cni/bin
              name: cni-bin-dir
            - mountPath: /host/etc/cni/net.d
              name: cni-net-dir
      volumes:
        # Used by calico/node.
        - name: lib-modules
          hostPath:
            path: /lib/modules
        - name: var-run-calico
          hostPath:
            path: /var/run/calico
        # Used to install CNI.
        - name: cni-bin-dir
          hostPath:
            path: /opt/cni/bin
        - name: cni-net-dir
          hostPath:
            path: /etc/cni/net.d

---

# This manifest deploys the Calico policy controller on Kubernetes.
# See https://github.com/projectcalico/k8s-policy
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: calico-policy-controller
  namespace: kube-system
  labels:
    k8s-app: calico-policy
spec:
```

```
  # The policy controller can only have a single active instance.
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      name: calico-policy-controller
      namespace: kube-system
      labels:
        k8s-app: calico-policy-controller
      annotations:
        # Mark this pod as a critical add-on; when enabled, the critical add-on sch
eduler
        # reserves resources for critical add-on pods so that they can be reschedul
ed after
        # a failure.  This annotation works in tandem with the toleration below.
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      # The policy controller must run in the host network namespace so that
      # it isn't governed by policy that would prevent it from working.
      hostNetwork: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      # Allow this pod to be rescheduled while the node is in "critical add-ons onl
y" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      - key: CriticalAddonsOnly
        operator: Exists
      serviceAccountName: calico-policy-controller
      containers:
        - name: calico-policy-controller
          image: harbor.fengjr.com/k8s/kube-policy-controller:v0.6.0
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # The location of the Kubernetes API.  Use the default Kubernetes
            # service for API access.
            - name: K8S_API
            #  value: "https://kubernetes.default:443"
              value: "https://10.255.72.199:6443"
            # Since we're running in the host namespace and might not have KubeDNS
            # access, configure the container's /etc/hosts to resolve
            # kubernetes.default to the correct service clusterIP.
            - name: CONFIGURE_ETC_HOSTS
              value: "true"
---
```

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: calico-cni-plugin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-cni-plugin
subjects:
- kind: ServiceAccount
  name: calico-cni-plugin
  namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-cni-plugin
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources:
      - pods
      - nodes
    verbs:
      - get
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: calico-cni-plugin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: calico-policy-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-policy-controller
subjects:
- kind: ServiceAccount
  name: calico-policy-controller
  namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-policy-controller
  namespace: kube-system
rules:
```

```
    - apiGroups:
      - ""
      - extensions
      resources:
        - pods
        - namespaces
        - networkpolicies
      verbs:
        - watch
        - list
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: calico-policy-controller
  namespace: kube-system
```

～～注意：需要更改的地方～～

- **1.** `etcd_endpoints: "http://10.228.0.222:6666"` 第一次在这个yaml文件里定义 是创建的calico需要的etcd集群的ip地址和端口
- **2. 这里写apiserver的集群ip地址，也就是haproxy+keepalived 搭建的代理的vip和端口**

```
- name: K8S_API
   value: "https://10.255.72.199:6443"
```

- **3.calico-node的service定义的区域里需要再次更改etcd的clusterip 同1**

```
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: calico-etcd
  name: calico-etcd
  namespace: kube-system
spec:
  # Select the calico-etcd pod running on the master.
  selector:
    k8s-app: calico-etcd
clusterIP: 10.228.0.222
  ports:
    - port: 6666
```

- **4. 需要更改docker镜像的下载地址，贴出来的yaml文件里的地址为局域网的内部私有仓库地址，需要自己搭建docker私仓，这里不再展开** 类似这种的都得改 也可以使用开源的国内的源 `harbor.fengjr.com/k8s/kube-policy-controller:v0.6.0`

接上文 K8S V.1.10.0 二进制安装

**1.calicoctl的下载和配置 这里的配置calicoctl版本使用的是v1.2.0-6-gd1c370c** 下载地址为：calicoctl.bin 如果使用最新版的calicoctl 使用方法参考：https://docs.projectcalico.org/v3.1/usage/configuration/bgp

```
chmod +x  calicoctl
ETCD_ENDPOINTS=http://localhost:2379
**这里假定交换机的asnumber是65000 asnumber需要和网络那边协调，共同配置，所以这里只贴出主机层面的配置参考**

2.设置as号
calicoctl config set asNumber 65000

3.与附近机器建立per开关
calicoctl config get nodeToNodeMesh
calicoctl config set nodeToNodeMesh on
calicoctl config set nodeToNodeMesh off
```

4.与交换机10.255.72.254的asNumber: 65000建立bgp通信 10.255.72.254 是我这里的交换机对接节点，具体情况实际参考自己的网络环境。

> calico-bgp-netberhood.json
>
> ```
> apiVersion: v1
> kind: bgpPeer
> metadata:
>   peerIP: 10.255.72.254
>   scope: global
> spec:
>   asNumber: 65000
> ```
>
> `calicoctl create -f calico-bgp-netberhood.json`

5.关闭pod网络访问外部的NAT,并宣告calico本身的网段

> calico-disable-nat-subnet.json

```
apiVersion: v1
kind: ipPool
metadata:
  cidr: 10.229.0.0/16
spec:
  ipip:
    enabled: false
  nat-outgoing: false
  disabled: false
```

`calicoctl create -f calico-disable-nat-subnet.json`

1. 查看node状态： calicoctl node status

**7. 部署calico-node**

```
# Calico Version v2.3.0
# http://docs.projectcalico.org/v2.3/releases#v2.3.0
# This manifest includes the following component versions:
#   calico/node:v1.3.0
#   calico/cni:v1.9.1
#   calico/kube-policy-controller:v0.6.0

# This ConfigMap is used to configure a self-hosted Calico installation.
kind: ConfigMap
apiVersion: v1
metadata:
  name: calico-config
  namespace: kube-system
data:
  # The location of your etcd cluster.  This uses the Service clusterIP
  # defined below.
  etcd_endpoints: "http://10.255.72.190:6666" #这个是nodepod的ip
  # Configure the Calico backend to use.
  calico_backend: "bird"

  # The CNI network configuration to install on each node.
  cni_network_config: |-
    {
        "name": "k8s-pod-network",
        "cniVersion": "0.1.0",
        "type": "calico",
        "etcd_endpoints": "__ETCD_ENDPOINTS__",
        "log_level": "info",
        "ipam": {
            "type": "calico-ipam"
        },
        "policy": {
            "type": "k8s",
             "k8s_api_root": "https://__KUBERNETES_SERVICE_HOST__:__KUBERNETES_SERVICE_PORT__",
             "k8s_auth_token": "__SERVICEACCOUNT_TOKEN__"
        },
        "kubernetes": {
            "kubeconfig": "/etc/cni/net.d/__KUBECONFIG_FILENAME__"
        }
    }

---

# This manifest installs the Calico etcd on the kubeadm master.  This uses a DaemonSet
# to force it to run on the master even when the master isn't schedulable, and uses
```

```
# nodeSelector to ensure it only runs on the master.
apiVersion: extensions/v1beta1
#kind: DaemonSet
kind: Deployment
metadata:
  name: calico-etcd
  namespace: kube-system
  labels:
    k8s-app: calico-etcd
spec:
  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: calico-etcd
      annotations:
        # Mark this pod as a critical add-on; when enabled, the critical add-on sch
eduler
        # reserves resources for critical add-on pods so that they can be reschedul
ed after
        # a failure.  This annotation works in tandem with the toleration below.
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: calico-etcd
                  operator: In
                  values: ["calico-etcd"]
      # Only run this pod on the master.
      #tolerations:
      #- key: node-role.kubernetes.io/master
      #  effect: NoSchedule
      # Allow this pod to be rescheduled while the node is in "critical add-ons onl
y" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      #- key: CriticalAddonsOnly
      #  operator: Exists
      #nodeSelector:
      #  node-role.kubernetes.io/master: ""
      hostNetwork: true
      containers:
        - name: calico-etcd
          image: hub-dev.fengjr.com/k8s/etcd:2.2.1
          env:
            - name: CALICO_ETCD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
```

```
          command: ["/bin/sh","-c"]
          args: ["/usr/local/bin/etcd --name=calico --data-dir=/var/etcd/calico-dat
a --advertise-client-urls=http://$CALICO_ETCD_IP:6666 --listen-client-urls=http://0
.0.0.0:6666 --listen-peer-urls=http://0.0.0.0:6667"]
          volumeMounts:
            - name: var-etcd
              mountPath: /var/etcd
      volumes:
        - name: var-etcd
          hostPath:
            path: /var/etcd

---

# This manifest installs the Service which gets traffic to the Calico
# etcd.
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: calico-etcd
  name: calico-etcd
  namespace: kube-system
spec:
  # Select the calico-etcd pod running on the master.
  selector:
    k8s-app: calico-etcd
  # This ClusterIP needs to be known in advance, since we cannot rely
  # on DNS to get access to etcd.
  clusterIP: 10.228.0.222
  ports:
    - port: 6666

---

# This manifest installs the calico/node container, as well
# as the Calico CNI plugins and network config on
# each master and worker node in a Kubernetes cluster.
kind: DaemonSet
apiVersion: extensions/v1beta1
metadata:
  name: calico-node
  namespace: kube-system
  labels:
    k8s-app: calico-node
spec:
  selector:
    matchLabels:
      k8s-app: calico-node
  template:
    metadata:
```

```
      labels:
        k8s-app: calico-node
      annotations:
        # Mark this pod as a critical add-on; when enabled, the critical add-on sch
eduler
        # reserves resources for critical add-on pods so that they can be reschedul
ed after
        # a failure.  This annotation works in tandem with the toleration below.
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      hostNetwork: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      # Allow this pod to be rescheduled while the node is in "critical add-ons onl
y" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      - key: CriticalAddonsOnly
        operator: Exists
      serviceAccountName: calico-cni-plugin
      containers:
        # Runs calico/node container on each Kubernetes node.  This
        # container programs network policy and routes on each
        # host.
        - name: calico-node
          image: hub-dev.fengjr.com/k8s/node:v1.3.0
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # Enable BGP.  Disable to enforce policy only.
            - name: CALICO_NETWORKING_BACKEND
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: calico_backend
            # Disable file logging so `kubectl logs` works.
            - name: CALICO_DISABLE_FILE_LOGGING
              value: "true"
            # Set Felix endpoint to host default action to ACCEPT.
            - name: FELIX_DEFAULTENDPOINTTOHOSTACTION
              value: "ACCEPT"
            # Configure the IP Pool from which Pod IPs will be chosen.
            - name: CALICO_IPV4POOL_CIDR
              value: "10.229.0.0/16"
            - name: CALICO_IPV4POOL_IPIP
              value: "always"
            # Disable IPv6 on Kubernetes.
```

```
            - name: FELIX_IPV6SUPPORT
              value: "false"
            # Set Felix logging to "info"
            - name: FELIX_LOGSEVERITYSCREEN
              value: "info"
            # Auto-detect the BGP IP address.
            - name: IP
              value: ""
          securityContext:
            privileged: true
          resources:
            requests:
              cpu: 300m
          volumeMounts:
            - mountPath: /lib/modules
              name: lib-modules
              readOnly: true
            - mountPath: /var/run/calico
              name: var-run-calico
              readOnly: false
        # This container installs the Calico CNI binaries
        # and CNI network config file on each node.
        - name: install-cni
          image: hub-dev.fengjr.com/k8s/cni:v1.9.1
          command: ["/install-cni.sh"]
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # The CNI network config to install on each node.
            - name: CNI_NETWORK_CONFIG
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: cni_network_config
          volumeMounts:
            - mountPath: /host/opt/cni/bin
              name: cni-bin-dir
            - mountPath: /host/etc/cni/net.d
              name: cni-net-dir
      volumes:
        # Used by calico/node.
        - name: lib-modules
          hostPath:
            path: /lib/modules
        - name: var-run-calico
          hostPath:
            path: /var/run/calico
```

```
        # Used to install CNI.
        - name: cni-bin-dir
          hostPath:
            path: /opt/cni/bin
        - name: cni-net-dir
          hostPath:
            path: /etc/cni/net.d

---

# This manifest deploys the Calico policy controller on Kubernetes.
# See https://github.com/projectcalico/k8s-policy
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: calico-policy-controller
  namespace: kube-system
  labels:
    k8s-app: calico-policy
spec:
  # The policy controller can only have a single active instance.
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      name: calico-policy-controller
      namespace: kube-system
      labels:
        k8s-app: calico-policy-controller
      annotations:
        # Mark this pod as a critical add-on; when enabled, the critical add-on sch
eduler
        # reserves resources for critical add-on pods so that they can be reschedul
ed after
        # a failure.  This annotation works in tandem with the toleration below.
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      # The policy controller must run in the host network namespace so that
      # it isn't governed by policy that would prevent it from working.
      hostNetwork: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      # Allow this pod to be rescheduled while the node is in "critical add-ons onl
y" mode.
      # This, along with the annotation above marks this pod as a critical add-on.
      - key: CriticalAddonsOnly
        operator: Exists
      serviceAccountName: calico-policy-controller
      containers:
```

```
        - name: calico-policy-controller
          image: hub-dev.fengjr.com/k8s/kube-policy-controller:v0.6.0
          env:
            # The location of the Calico etcd cluster.
            - name: ETCD_ENDPOINTS
              valueFrom:
                configMapKeyRef:
                  name: calico-config
                  key: etcd_endpoints
            # The location of the Kubernetes API.  Use the default Kubernetes
            # service for API access.
            - name: K8S_API
            #  value: "https://kubernetes.default:443"
              value: "https://10.255.72.199:6443"
            # Since we're running in the host namespace and might not have KubeDNS
            # access, configure the container's /etc/hosts to resolve
            # kubernetes.default to the correct service clusterIP.
            - name: CONFIGURE_ETC_HOSTS
              value: "true"
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: calico-cni-plugin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-cni-plugin
subjects:
- kind: ServiceAccount
  name: calico-cni-plugin
  namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-cni-plugin
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources:
      - pods
      - nodes
    verbs:
      - get
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: calico-cni-plugin
  namespace: kube-system
```

```
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: calico-policy-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: calico-policy-controller
subjects:
- kind: ServiceAccount
  name: calico-policy-controller
  namespace: kube-system
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: calico-policy-controller
  namespace: kube-system
rules:
  - apiGroups:
    - ""
    - extensions
    resources:
      - pods
      - namespaces
      - networkpolicies
    verbs:
      - watch
      - list
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: calico-policy-controller
  namespace: kube-system
```

**8. 给node节点打标签**

```
kubectl label node kube-master02 calico-etcd=calico-etcd #这一步配合calico.yaml里的node节点亲和
```

就是这个配置

```
     affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
```

```
                - key: calico-etcd
                  operator: In
                  values: ["calico-etcd"]
```

### 1.kubedns-cm.yaml

> configmap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
```

### 2.kubedns-controller.yaml

> controller deployment

```
# Copyright 2016 The Kubernetes Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Should keep target in cluster/addons/dns-horizontal-autoscaler/dns-horizontal-aut
oscaler.yaml
# in sync with this file.

# __MACHINE_GENERATED_WARNING__

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  # replicas: not specified here:
  # 1. In order to make Addon Manager do not reconcile this replicas parameter.
  # 2. Default is 1.
  # 3. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
```

```
  strategy:
    rollingUpdate:
      maxSurge: 10%
      maxUnavailable: 0
  selector:
    matchLabels:
      k8s-app: kube-dns
  template:
    metadata:
      labels:
        k8s-app: kube-dns
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      volumes:
      - name: kube-dns-config
        configMap:
          name: kube-dns
          optional: true
      containers:
      - name: kubedns
        image: hub-dev.fengjr.com/k8s/k8s-dns-kube-dns-amd64:1.14.5
        resources:
          # TODO: Set memory limits when we've profiled the container for large
          # clusters, then set request = limit to keep this container in
          # guaranteed class. Currently, this container falls into the
          # "burstable" category so the kubelet doesn't backoff from restarting it.
          limits:
            memory: 170Mi
          requests:
            cpu: 100m
            memory: 70Mi
        livenessProbe:
          httpGet:
            path: /healthcheck/kubedns
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        readinessProbe:
          httpGet:
            path: /readiness
            port: 8081
            scheme: HTTP
          # we poll on pod startup for the Kubernetes master service and
          # only setup the /readiness HTTP server once that's available.
```

```
          initialDelaySeconds: 3
          timeoutSeconds: 5
        args:
        - --domain=cluster.local.
        - --dns-port=10053
        - --config-dir=/kube-dns-config
        - --v=2
        #__PILLAR__FEDERATIONS__DOMAIN__MAP__
        env:
        - name: PROMETHEUS_PORT
          value: "10055"
        ports:
        - containerPort: 10053
          name: dns-local
          protocol: UDP
        - containerPort: 10053
          name: dns-tcp-local
          protocol: TCP
        - containerPort: 10055
          name: metrics
          protocol: TCP
        volumeMounts:
        - name: kube-dns-config
          mountPath: /kube-dns-config
      - name: dnsmasq
        image: hub-dev.fengjr.com/k8s/k8s-dns-dnsmasq-nanny-amd64:1.14.5
        livenessProbe:
          httpGet:
            path: /healthcheck/dnsmasq
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        args:
        - -v=2
        - -logtostderr
        - -configDir=/etc/k8s/dns/dnsmasq-nanny
        - -restartDnsmasq=true
        - --
        - -k
        - --cache-size=1000
        - --log-facility=-
        - --server=/cluster.local./127.0.0.1#10053
        - --server=/in-addr.arpa/127.0.0.1#10053
        - --server=/ip6.arpa/127.0.0.1#10053
        ports:
        - containerPort: 53
          name: dns
          protocol: UDP
```

```
        - containerPort: 53
          name: dns-tcp
          protocol: TCP
        # see: https://github.com/kubernetes/kubernetes/issues/29055 for details
        resources:
          requests:
            cpu: 150m
            memory: 20Mi
        volumeMounts:
        - name: kube-dns-config
          mountPath: /etc/k8s/dns/dnsmasq-nanny
      - name: sidecar
        image: hub-dev.fengjr.com/k8s/k8s-dns-sidecar-amd64:1.14.8
        livenessProbe:
          httpGet:
            path: /metrics
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        args:
        - --v=2
        - --logtostderr
        - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local.,5,A
        - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,A
        ports:
        - containerPort: 10054
          name: metrics
          protocol: TCP
        resources:
          requests:
            memory: 20Mi
            cpu: 10m
      dnsPolicy: Default  # Don't use cluster DNS.
      serviceAccountName: kube-dns
```

### 3.kubedns-sa.yaml

> account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
```

### 4.kubedns-svc.yaml

> service

```
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.228.0.10 #这里是在之前二进制安装的时候在kubelet service里定义的dns的地址
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
```

### 5.应用yaml文件创建kubedns微服务

```
kubectl apply -f kubedns-cm.yaml  kubedns-controller.yaml  kubedns-sa.yaml  kubedns
-svc.yaml
```

### 1. kube-dashboard.yaml

```
# Copyright 2017 The Kubernetes Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Configuration to deploy release version of the Dashboard UI compatible with
# Kubernetes 1.8.
#
# Example usage: kubectl create -f <this_file>

# ------------------- Dashboard Secret ------------------- #

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

---
# ------------------- Dashboard Service Account ------------------- #

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

---
# ------------------- Dashboard Role & Role Binding ------------------- #

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: kubernetes-dashboard-minimal
```

```
  namespace: kube-system
rules:
  # Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create"]
  # Allow Dashboard to create 'kubernetes-dashboard-settings' config map.
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["create"]
  # Allow Dashboard to get, update and delete Dashboard exclusive secrets.
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs"]
  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 from heapster.
- apiGroups: [""]
  resources: ["services"]
  resourceNames: ["heapster"]
  verbs: ["proxy"]
- apiGroups: [""]
  resources: ["services/proxy"]
  resourceNames: ["heapster", "http:heapster:", "https:heapster:"]
  verbs: ["get"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubernetes-dashboard-minimal
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubernetes-dashboard-minimal
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system

---
# ------------------- Dashboard Deployment ------------------- #

kind: Deployment
apiVersion: apps/v1beta2
metadata:
```

```
  labels:
    k8s-app: kubernetes-dashboard
  name: kubernetes-dashboard
  namespace: kube-system
spec:
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
    spec:
      containers:
      - name: kubernetes-dashboard
        image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.8.3
        ports:
        - containerPort: 8443
          protocol: TCP
        args:
          - --auto-generate-certificates
          # Uncomment the following line to manually specify Kubernetes API server
Host
          # If not specified, Dashboard will attempt to auto discover the API serve
r 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 mast
er
```

```
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule

---
# ------------------- Dashboard Service ------------------- #

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

**2.配置kube-dashboard**

```
kubectl -n kube-system create sa dashboard
kubectl create clusterrolebinding dashboard --clusterrole cluster-admin --serviceac
count=kube-system:dashboard
kubectl apply -f kube-dashboard.yaml
SECRET=$(kubectl -n kube-system get sa dashboard -o yaml | awk '/dashboard-token/ {
print $3}')
kubectl -n kube-system describe secrets ${SECRET} | awk '/token:/{print $2}'
```

~~注意:~~ dashboard的镜像版本必须>=kubernetes-dashboard-amd64:v1.8.3

**1.镜像准备** 使用能翻墙的docker 将需要的镜像pull下来导出。

```
翻墙参考：https://github.com/getlantern/lantern/releases/tag/latest
使用的nginx-ingress的安装指导为：https://kubernetes.github.io/ingress-nginx/deploy/
需要用到yaml文件为： https://raw.githubusercontent.com/kubernetes/ingress-nginx/master
/deploy/mandatory.yaml
需要用到的images为：
gcr.io/google_containers/defaultbackend:1.4
quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.15.0
```


```
docker pull gcr.io/google_containers/defaultbackend:1.4
docker pull quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.15.0
docker images #获取镜像的id
docker save -o defaultbackend:1.4.tar 846921f0fe0e
docker save -o nginx-ingress-controller:0.15.0.tar c46bc3e1b53c
```

将导入的tar包传到所有的k8s node节点上

```
docker load < defaultbackend:1.4.tar
#此时的docker镜像没有名字和版本号 需要打tag 获取到导入的镜像的id号
docker tag 846921f0fe0e gcr.io/google_containers/defaultbackend:1.4
docker load < nginx-ingress-controller:0.15.0.tar
docker tag c46bc3e1b53c quay.io/kubernetes-ingress-controller/nginx-ingress-control
ler:0.15.0
```

**2. 修改yaml文件并修改apply**

> mandatory.yaml #官方文件: https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml 在这个文件的基础上做了一些修改和删除，这个文件是创建一个ingress-nginx的namespaces，创建相应的rabc权限，创建default-http-backend 的部署，同时 我删除了ingress-nginx的部署，将它单独摘出来做配置。 yaml文件是强格式的，缩进不对也会报错。
>
> ``` yaml

apiVersion: v1 kind: Namespace metadata:

# name: ingress-nginx

apiVersion: extensions/v1beta1 kind: Deployment metadata: name: default-http-backend labels: app: default-http-backend namespace: ingress-nginx spec: replicas: 1 selector: matchLabels: app: default-http-backend template: metadata: labels: app: default-http-backend spec: terminationGracePeriodSeconds: 60 containers:

```
  - name: default-http-backend
    # Any image is permissible as long as:
    # 1. It serves a 404 page at /
    # 2. It serves 200 on a /healthz endpoint
    image: gcr.io/google_containers/defaultbackend:1.4
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        scheme: HTTP
      initialDelaySeconds: 30
      timeoutSeconds: 5
    ports:
    - containerPort: 8080
    resources:
      limits:
        cpu: 10m
        memory: 20Mi
      requests:
        cpu: 10m
        memory: 20Mi
```

---

apiVersion: v1 kind: Service metadata: name: default-http-backend namespace: ingress-nginx labels: app: default-http-backend spec: ports:

- port: 80 targetPort: 8080 selector: app: default-http-backend

---

kind: ConfigMap apiVersion: v1 metadata: name: nginx-configuration namespace: ingress-nginx labels:

```
app: ingress-nginx
```

---

kind: ConfigMap apiVersion: v1 metadata: name: tcp-services

## namespace: ingress-nginx

kind: ConfigMap apiVersion: v1 metadata: name: udp-services

## namespace: ingress-nginx

apiVersion: v1 kind: ServiceAccount metadata: name: nginx-ingress-serviceaccount namespace: ingress-nginx

apiVersion: rbac.authorization.k8s.io/v1beta1 kind: ClusterRole metadata: name: nginx-ingress-clusterrole rules:

- apiGroups:
  - "" resources:
  - configmaps
  - endpoints
  - nodes
  - pods
  - secrets verbs:
  - list
  - watch
- apiGroups:
  - "" resources:
  - nodes verbs:
  - get
- apiGroups:
  - "" resources:
  - services verbs:
  - get
  - list
  - watch
- apiGroups:
  - "extensions" resources:
  - ingresses verbs:
  - get
  - list
  - watch
- apiGroups:
  - "" resources:
    - events verbs:
    - create
    - patch
- apiGroups:
  - "extensions" resources:
  - ingresses/status verbs:
  - update

apiVersion: rbac.authorization.k8s.io/v1beta1 kind: Role metadata: name: nginx-ingress-role namespace: ingress-nginx rules:

- apiGroups:

- "" resources:
- configmaps
- pods
- secrets
- namespaces verbs:
- get

- apiGroups:
  - "" resources:
  - configmaps resourceNames:

    ## Defaults to "-"

    ## Here: "-"

    ## This has to be adapted if you change either parameter

    ## when launching the nginx-ingress-controller.

  - "ingress-controller-leader-nginx" verbs:
  - get
  - update
- apiGroups:
  - "" resources:
  - configmaps verbs:
  - create
- apiGroups:
  - "" resources:
  - endpoints verbs:
  - get

---

apiVersion: rbac.authorization.k8s.io/v1beta1 kind: RoleBinding metadata: name: nginx-ingress-role-nisa-binding namespace: ingress-nginx roleRef: apiGroup: rbac.authorization.k8s.io kind: Role name: nginx-ingress-role subjects:

- kind: ServiceAccount name: nginx-ingress-serviceaccount namespace: ingress-nginx

---

apiVersion: rbac.authorization.k8s.io/v1beta1 kind: ClusterRoleBinding metadata: name: nginx-ingress-clusterrole-nisa-binding roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: nginx-ingress-clusterrole subjects:

- kind: ServiceAccount name: nginx-ingress-serviceaccount namespace: ingress-nginx

```
> ingress-nginx-deploy.yaml

``` yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ingress-nginx
  template:
    metadata:
      labels:
        app: ingress-nginx
      annotations:
        prometheus.io/port: '10254'
        prometheus.io/scrape: 'true'
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: role
                  operator: In
                  values: ["ingress-nginx"]
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      serviceAccountName: nginx-ingress-serviceaccount
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.1
5.0
          args:
            - /nginx-ingress-controller
            - --default-backend-service=$(POD_NAMESPACE)/default-http-backend
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
            - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
            - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx
```
```

```
            - --annotations-prefix=nginx.ingress.kubernetes.io
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          ports:
          - name: http
            containerPort: 80
          - name: https
            containerPort: 443
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            initialDelaySeconds: 10
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1
          securityContext:
            runAsNonRoot: false
```

相对于官方,这里做了两个方面的改动:

- **1.让部署集部署的pod 使用宿主node节点的ip**

```
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet #这一项是确保创建的pod里的dns是k8s的dns
```

- **2.选择具有某一类标签的node节点运行ingress**

```
  spec:
      affinity:
        nodeAffinity:
```

```
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: role
                  operator: In
                  values: ["ingress-nginx"]
```

- **3.给需要部署ingress的node节点打标签**

```
kubectl label node kube-master01 role=ingress-nginx
kubectl label node kube-master02 role=ingress-nginx
kubectl label node kube-master03 role=ingress-nginx
```

~~给node删除标签~~

```
kubectl label node kube-master01 role -
```

~~查看标签~~

```
kubectl get nodes -l role=ingress-nginx
```

这里提供两个yaml文件下载： ingress-nginx-deploy.yaml mandatory.yaml

**3.搭建一个简易的服务**

- **1.创建证书并导入到k8s中**

```
openssl req -x509 -nodes -days 36500 -newkey rsa:2048 -keyout /tmp/tls.key -out /tm
p/tls.crt -subj "/CN=bbs.test.com"
kubectl create secret tls bbs-secret --key /tmp/tls.key --cert /tmp/tls.crt
```

- **2. 创建一个简易的服务这里以一个nginx为后端服务**

```
kubectl run bbs --image=nginx--replicas=1 --port=8080
kubectl expose deployment bbs --port=80 --target-port=8080 --name=bbs
```

- **3.配置nginx-ingress代理规则**

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: bbs
  namespace: default
spec:
  tls:
  - hosts:
```

```
    - bbs.test.com
    secretName: bbs-secret
  rules:
  - host: bbs.test.com
    http:
      paths:
      - backend:
          serviceName: bbs
          servicePort: 80
        path: /
```

- **4.配置host解析**

```
kubectl get pods -n ingress-nginx
kubectl describe ingress-nginx-xxxx -n ingres-nginx #获取nginx-ingress的pod的ip地址
#在自己的电脑上配置hosts解析
echo "pods-ip  bbs.test.com" >>/etc/hosts #windows为  :c:/windows/system32/drivers/etc/hosts  #mac 为: /private/etc/hosts
```

- **5. 访问https://bbs.test.com ,接下来需要做ingress-nginx的客户端node节点绑定和前端的负载均衡**
- **6.配置ingress-nginx的nginx_status**

```
curl -LO https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/docs/examples/customization/custom-vts-metrics-prometheus/nginx-vts-metrics-conf.yaml
kubectl apply -f nginx-vts-metrics-conf.yaml
```

- **7. 然后访问 http://nginx_ingress_pods_ip:18080/nginx_status/**

### 1.清理kube-proxy

```
service kube-proxy stop
kube-proxy --cleanup
kube-proxy --cleanup-ipvs
```

### 2.创建kube-route rbac

~~使用之前的kube-proxy的证书,增加点权限就可以了~~

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: kube-router
  namespace: kube-system
rules:
  - apiGroups:
    - ""
    resources:
      - namespaces
      - pods
      - services
      - nodes
      - endpoints
    verbs:
      - list
      - get
      - watch
  - apiGroups:
    - "networking.k8s.io"
    resources:
      - networkpolicies
    verbs:
      - list
      - get
      - watch
  - apiGroups:
    - extensions
    resources:
      - networkpolicies
    verbs:
      - get
      - list
      - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: kube-router
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-router
subjects:
- kind: User
  name: system:kube-proxy
  namespace: kube-system
```

**3.kube-route.yaml**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kube-router-cfg
  namespace: kube-system
  labels:
    tier: node
    k8s-app: kube-router
data:
  cni-conf.json: |
    {
      "name":"kubernetes",
      "type":"bridge",
      "bridge":"kube-bridge",
      "isDefaultGateway":true,
      "ipam": {
        "type":"host-local"
      }
    }
---
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: kube-router
  namespace: kube-system
  labels:
    k8s-app: kube-router
spec:
  template:
    metadata:
      labels:
        k8s-app: kube-router
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      containers:
      - name: kube-router
        image: hub-dev.fengjr.com/k8s/kube-router
        args: ["--masquerade-all", "--run-router=false", "--run-firewall=false", "--run-service-proxy=true", "--kubeconfig=/var/lib/kube-router/kubeconfig"]
```

```
        securityContext:
          privileged: true
        imagePullPolicy: Always
        env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        volumeMounts:
        - name: lib-modules
          mountPath: /lib/modules
          readOnly: true
        - name: cni-conf-dir
          mountPath: /etc/cni/net.d
        - name: kubeconfig
          mountPath: /var/lib/kube-router/kubeconfig
          readOnly: true
      hostNetwork: true
      dnsPolicy: ClusterFirst
      tolerations:
      - key: CriticalAddonsOnly
        operator: Exists
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
        operator: Exists
      volumes:
      - name: lib-modules
        hostPath:
          path: /lib/modules
      - name: cni-conf-dir
        hostPath:
          path: /etc/cni/net.d
      - name: kube-router-cfg
        configMap:
          name: kube-router-cfg
      - name: kubeconfig
        hostPath:
          path: /etc/kubernetes/kube-proxy.kubeconfig
```

### 1. init_node_v1.sh

```
#!/bin/sh

# First Boot of init

timestamp() {
    date +'%Y-%m-%d %H:%M:%S'
}
die (){
    ts=`timestamp`
    echo "$ts ERR: $@"
    exit 1
}
log (){
    ts=`timestamp`
    echo "$ts INFO: $@"
}
warn (){
    ts=`timestamp`
    echo "$ts WARN: $@"
}
run() {
    log "$@"
    $@
}
get_ip() {
    #L_IP=`/usr/sbin/ifconfig | grep -P -o "(?< =inet)(.*)(?=netmask)" | grep -v "1
27.0.0.1" `
    L_IP=`hostname -I| cut -d" " -f 1`
    echo $L_IP
    [ -z "$L_IP" ] && L_IP=null && die "获取本机IP失败!!!"
}

log "====================================="
log "====**   Activating Begin     **===="
log "====================================="
selinux_status=$(/usr/sbin/getenforce)
if [ $selinux_status == "Disabled" ] ;then echo "selinux_status is disabled,continu
e...";else echo "selinux status is ${selinux_status},install will breakup...";exit
0 ;fi
cd /tmp
get_ip

URL_BASE='http://10.255.57.7:8090/upload/k8s'

MANIFESTS_DIR='/etc/kubernetes/manifests'
SSL_DIR='/etc/kubernetes/ssl'

[ -d "$MANIFESTS_DIR" ]  || run "mkdir -p $MANIFESTS_DIR"
```

```
[ -d "$SSL_DIR" ] || run "mkdir -p $SSL_DIR "
run " mkdir -p /etc/cni/net.d  /opt/cni/bin/ /var/lib/kubelet "
if ! grep "hub-dev.fengjr.com" /etc/hosts ; then  echo "10.255.57.7 hub-dev.fengjr.
com" >> /etc/hosts; fi
log "Update os software..."
run "yum update -y -q"
if [ $? == 0 ];then echo "Update sucess...";else echo "Update failed,install will b
reakup...";exit 0 ;fi
rpm -qa |grep -q 'docker-engine-1.13.1'
if [ $? -ne 0 ];then
    rm -rf /tmp/docker_install_pkg*
    log "Download docker_install_pkg..."
    run "wget -q --timeout=30 -O /tmp/docker_install_pkg.zip $URL_BASE/docker_insta
ll_pkg.zip" \
        || die "Download failed on docker_install_pkg.zip !!!"
    run "unzip -qq /tmp/docker_install_pkg.zip -d /tmp/ " \
        || die "Unzip failed on docker_install_pkg.zip !!!"
    log "Install docker1.31.1..."
    run "rpm -ivh /tmp/docker_install_pkg/*.rpm  --nodeps" \
        || die "Install pkg failed on docker_install_pkg.zip !!!"
else
   log "Docker-versio docker-1.31.1 is already installed"
fi
log "Download docker.service config..."
run "wget -q --timeout=5 -O /usr/lib/systemd/system/docker.service $URL_BASE/docker
.service" \
    || die "Download config on docker.service !!!"

log "Download sysctl.conf config..."
run "wget -q --timeout=5 -O /etc/sysctl.conf $URL_BASE/sysctl.conf" \
    || die "Download config on sysctl.conf !!!"

log "Download kubelet pkg and config..."
run "wget -q --timeout=60 -O /usr/local/bin/kubelet $URL_BASE/kubelet" \
    || die "Download pkg on kubelet !!!"
run "chmod +x /usr/local/bin/kubelet"
run "wget -q --timeout=5 -O /etc/kubernetes/bootstrap.kubeconfig $URL_BASE/bootstra
p.kubeconfig" \
    || die "Download config on bootstrap.kubeconfig !!!"
run "wget -q --timeout=5 -O /usr/lib/systemd/system/kubelet.service $URL_BASE/kubel
et.service" \
    || die "Download config on kubelet.service !!!"
run "sed -i s#xxx.xxx.xxx.xxx#$L_IP#g /usr/lib/systemd/system/kubelet.service" \
    || die "Fix config on kubelet.service !!!"

log "Download kube-proxy.kubeconfig configfile..."
run "wget -q --timeout=5 -O /etc/kubernetes/kube-proxy.kubeconfig $URL_BASE/kube-pr
oxy.kubeconfig" \
    || die "Download config on kube-proxy.kubeconfig !!!"

log "Download ipvsadm pkg..."
```

```
run "wget -q --timeout=60 -O /usr/local/bin/ipvsadm $URL_BASE/ipvsadm" \
    || die "Download pkg on ipvsadm  !!!"
run "chmod +x /usr/local/bin/ipvsadm"

log "Start docker server ..."
run "systemctl enable docker"
run "systemctl start docker" \
    || die "Start docker server faild ... !!!"

log "Start kubelet server ..."
run "systemctl enable kubelet"
run "systemctl start kubelet" \
    || die "Start kubelet server faild ... !!!"


log "====================================="
log "====**    Activating End      **===="
log "====================================="
```

参考链接： https://kubernetes.io/docs/admin/kubelet-tls-bootstrapping/#approval-controller
https://mritd.me/2018/01/07/kubernetes-tls-bootstrapping-note/

**1.命令行**

```
# 自动批准 kubelet 的首次 CSR 请求(用于与 apiserver 通讯的证书)
kubectl create clusterrolebinding node-client-auto-approve-csr --clusterrole=approv
e-node-client-csr --group=system:bootstrappers

#这里有一个注意的点： --group=system:bootstrappers 这里的参数 要和token.csv里的最后的用户一
致 。 比如 我之前配置api-server的时候 参数--token-auth-file=/etc/kubernetes/token.csv 这
个文件里 最后些的是 "system:kubelet-bootstrap" 那我这里的命令 --group=system:kubelet-boo
tstrap

# 自动批准 kubelet 后续 renew 用于与 apiserver 通讯证书的 CSR 请求
kubectl create clusterrolebinding node-client-auto-renew-crt --clusterrole=approve-
node-client-renewal-csr --group=system:nodes

# 自动批准 kubelet 发起的用于 10250 端口鉴权证书的 CSR 请求(包括后续 renew)
kubectl create clusterrolebinding node-server-auto-renew-crt --clusterrole=approve-
node-server-renewal-csr --group=system:nodes
```

**2.yaml文件**

> controller-manager-auto-appove.yaml

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-client-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/nodeclient"]
  verbs: ["create"]
---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-client-renewal-csr
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeclient"]
  verbs: ["create"]
---

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: approve-node-server-renewal-csr
```

```
rules:
- apiGroups: ["certificates.k8s.io"]
  resources: ["certificatesigningrequests/selfnodeserver"]
  verbs: ["create"]
```

> kubectl create -f controller-manager-auto-appove.yaml

**2018-05-25 13:49:32 星期五**

参考链接：
https://github.com/kubernetes/contrib/tree/master/ingress/controllers/nginx/examples/proxy-protocol

~~有一定需要确定的是ingress-nginx需要通过nodeselect或者是节点亲和设置，将它固定部署在某一些node节点上，然后在haproxy上进行代理配置~~

**1. haproxy配置**

```
global
        daemon
        maxconn 40960
        stats socket /etc/haproxy/haproxy.stats level operator
        stats timeout 2m
        log 127.0.0.1 local0

defaults
        log global
        mode http
        retries 3
        option redispatch
        timeout connect 60000ms
        timeout client 60000ms
        timeout server 60000ms

##################
listen  nginx-ingress *:80
        mode tcp
        maxconn 4000
        balance roundrobin
        server  FJR-bt-kvm-72-189 10.255.72.189:80 check-send-proxy inter 10s send-
proxy
        server  FJR-bt-kvm-72-190 10.255.72.190:80 check-send-proxy inter 10s send-
proxy
        server  FJR-bt-kvm-72-191 10.255.72.191:80 check-send-proxy inter 10s send-
proxy
##################
listen  nginx-ingress *:443
        mode tcp
        maxconn 4000
        balance roundrobin
        server  FJR-bt-kvm-72-189 10.255.72.189:443 check-send-proxy inter 10s send
-proxy
        server  FJR-bt-kvm-72-190 10.255.72.190:443 check-send-proxy inter 10s send
-proxy
        server  FJR-bt-kvm-72-191 10.255.72.191:443 check-send-proxy inter 10s send
-proxy
##################

listen stats *:7744
```

```
        mode http
        option httpclose
        balance roundrobin
        stats uri /
        stats realm Haproxy\ Statistics
        stats auth admin:admin
        stats admin if TRUE
#################
```

**2.keepalived配置**

```
global_defs {
}
vrrp_script chk_haproxy {
        script "/etc/keepalived/check_haproxy.sh"
        interval 2
        weight 2
        }
vrrp_instance VIP_1 {
          state BACKUP
          interface eth0
          virtual_router_id 100
          priority 100
          advert_int 1
          nopreempt
          authentication {
            auth_type PASS
            auth_pass 968743
          }
  track_interface {
          eth0
  }
  virtual_ipaddress {
          10.255.72.207
  }
  track_script {
          chk_haproxy
  }
}
```

- /etc/keepalived/check_haproxy.sh

```
#!/bin/bash
A=$(netstat -lnpt |grep haproxy  | grep -E "80|443" | wc -l)
if [ $A -ne 2 ];then
  service haproxy start
sleep 3
if [ `netstat -lnpt |grep haproxy  | grep -E "80|443" | wc -l` -ne 2 ];then
  service keepalived stop
```

```
fi
fi
```