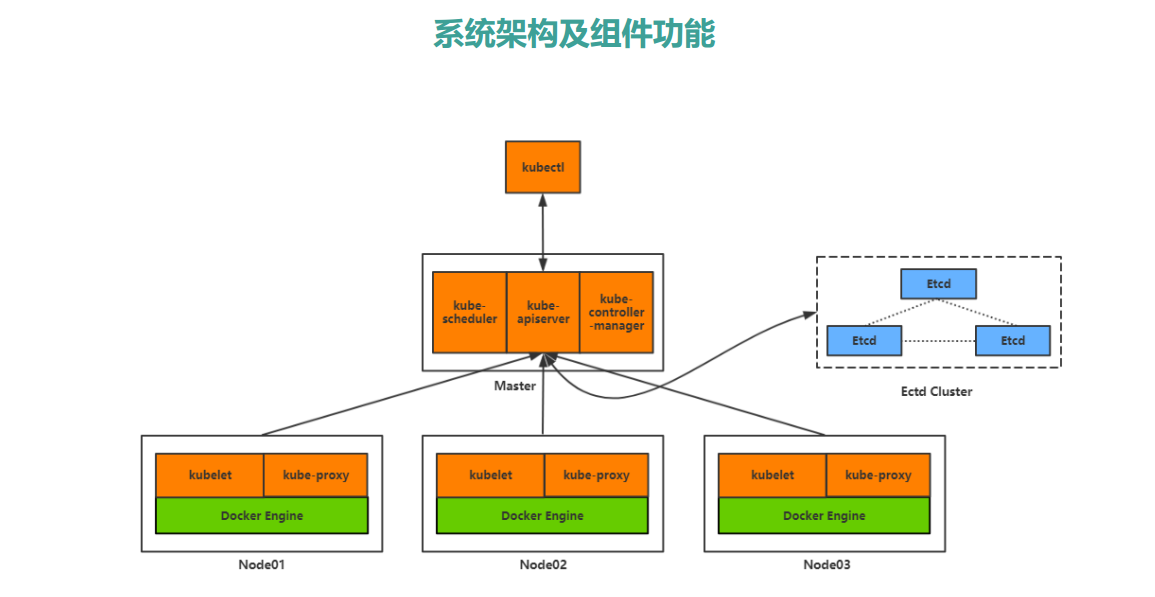
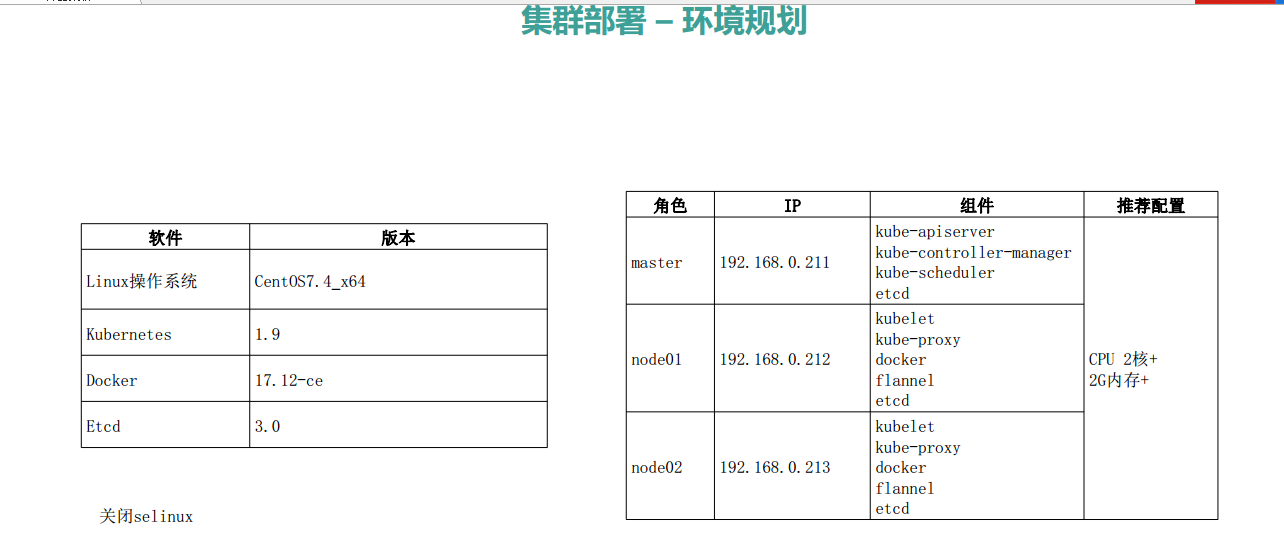
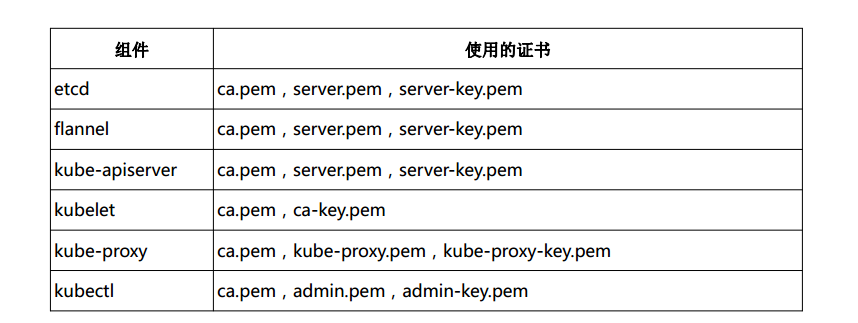
1. k8s的集群架构图



二．机器规划和组件



三．自签TLS证书



1.安装cfssl生成ca证书 和公私钥

wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo\_linux-amd64

chmod +x cfssl\_linux-amd64 cfssljson\_linux-amd64 cfssl-certinfo\_linux-amd64

mv cfssl\_linux-amd64 /usr/local/bin/cfssl

mv cfssljson\_linux-amd64 /usr/local/bin/cfssljson

mv cfssl-certinfo\_linux-amd64 /usr/bin/cfssl-certinfo

1.生成模板文件

cfssl print-defaults config > config.json

2.请求颁发证书的文件模板

cfssl print-defaults csr > csr.json

3.生成ca证书配置

cat > ca-config.json <<EOF

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"expiry": "87600h",

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

]

}

}

}

}

EOF

4.生成ca证书

cat > ca-csr.json <<EOF

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "Beijing",

"ST": "Beijing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

5. 生成ca-key.pem 和ca.pem 证书文件

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

6.用于server的http通信的证书

cat > server-csr.json <<EOF

{

"CN": "kubernetes",

"hosts": [

"127.0.0.1",

这里填自己集群的ip地址

"192.168.0.211",

"192.168.0.212",

"192.168.0.213",

"10.10.10.1",

"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

生成证书：cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes server-csr.json | cfssljson -bare server

7.生成集群管理员证书

cat > admin-csr.json <<EOF

{

"CN": "admin",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "system:masters",

"OU": "System"

}

]

}

EOF

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes admin-csr.json | cfssljson -bare admin

8.生成kube-proxy 证书 提供网络策略

cat > kube-proxy-csr.json <<EOF

{

"CN": "system:kube-proxy",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"L": "BeiJing",

"ST": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

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

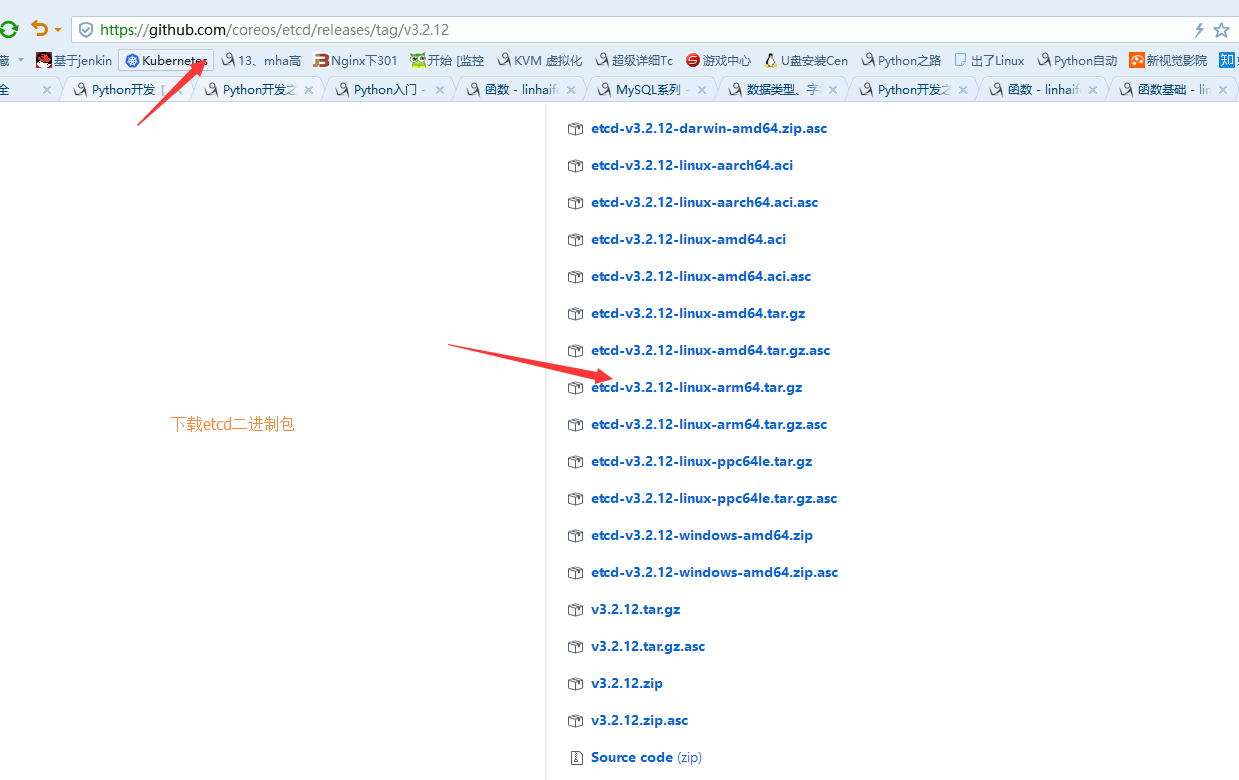
9.将pem证书保留

ls | grep –v pem | xargs –I rm {}



四 搭建etcd集群

1.下载etcd二进制包



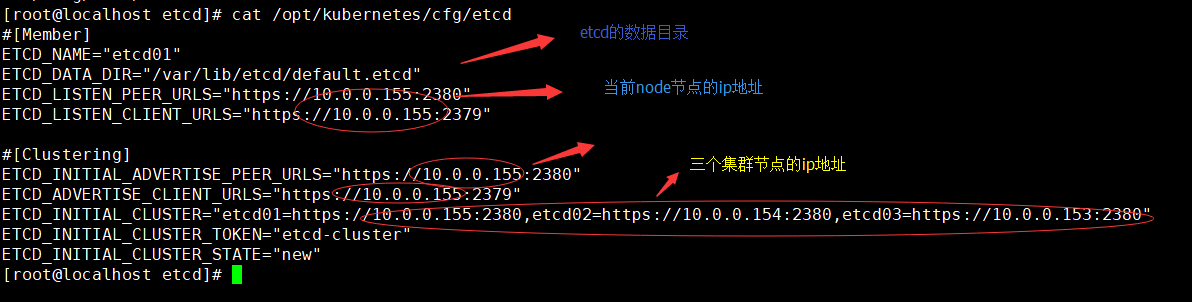
2.建议k8s集群的组件目录和配置文件存放位置

mkdir –p /opt/kubernetes{bin,cfg,ssl}

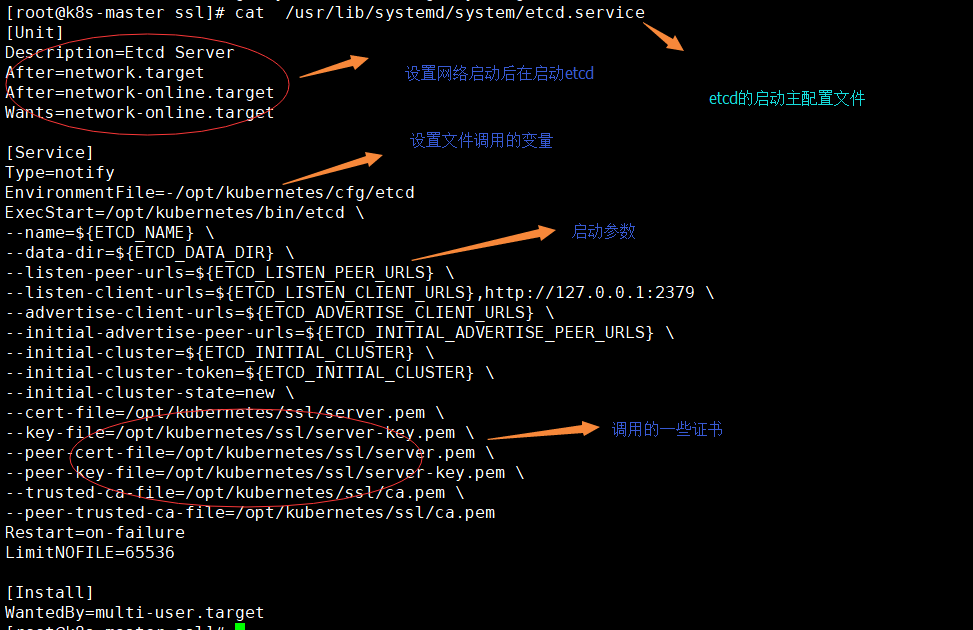
tar –xf etcd-v3.2.12-linux-amd64

mv etcd-v3.2.12-linux-amd64/etcd /opt/kubernetes/bin/

3.生成主配置文件



4.ectd启动文件



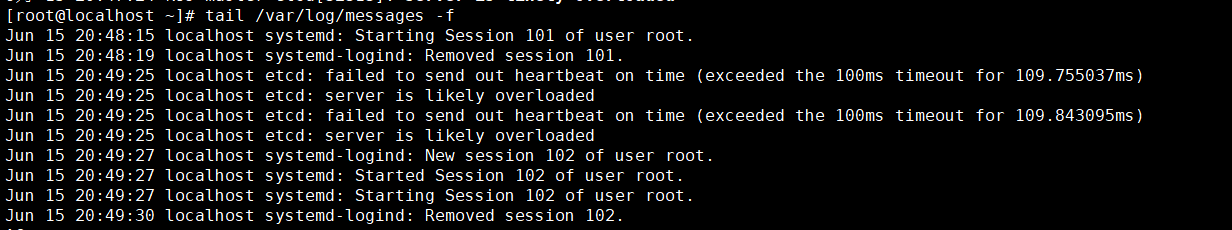
5.将ssl下面的证书移动到kubernetes的ssl下面

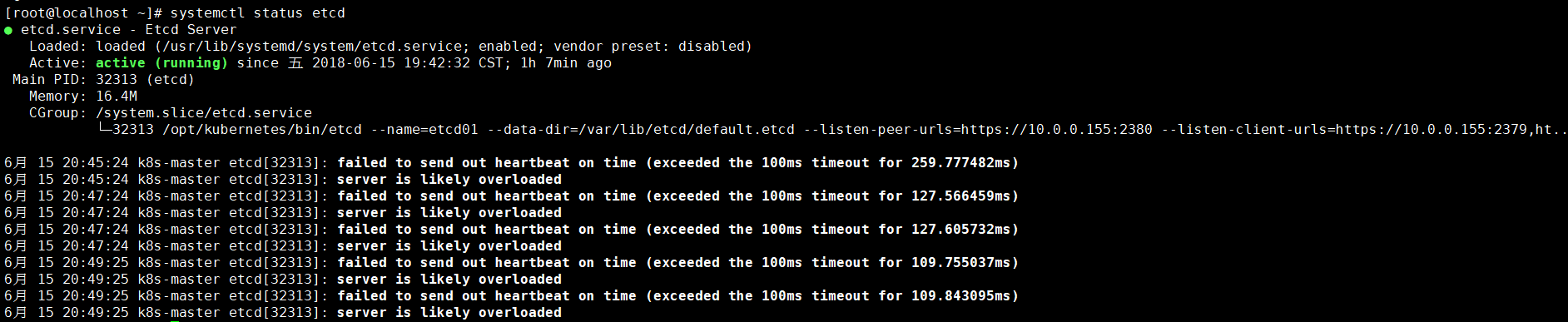
Cp ssl /server\*.pem ssl/ca\*.pem /opt/kubernetes/ssl/

6.启动etcd

Systemctl start etcd

7.查看日志





8.配置其他两个节点配置和ssh免密钥登录

ssh-keygen

ssh-copy-id [root@10.0.0.154](mailto:root@10.0.0.154)

ssh-copy-id [root@10.0.0.153](mailto:root@10.0.0.153)

scp -r /opt/kubernetes/cfg/etcd [root@10.0.0.154:/opt/kubernetes/](mailto:root@10.0.0.154:/opt/kubernetes/)

scp -r /opt/kubernetes/ssl/ [root@10.0.0.154:/opt/kubernetes/ssl/](mailto:root@10.0.0.154:/opt/kubernetes/ssl/)

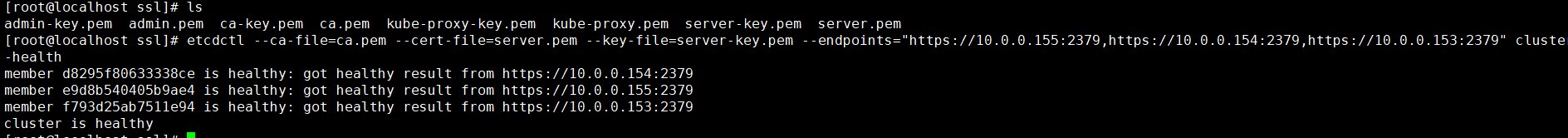
scp -r /opt/kubernetes/bin/ [root@10.0.0.153:/opt/kubernetes/bin/](mailto:root@10.0.0.153:/opt/kubernetes/bin/)

scp –r /usr/lib/systemd/system/etcd.service root@10.0.0.154:/usr/lib/systemd/system/

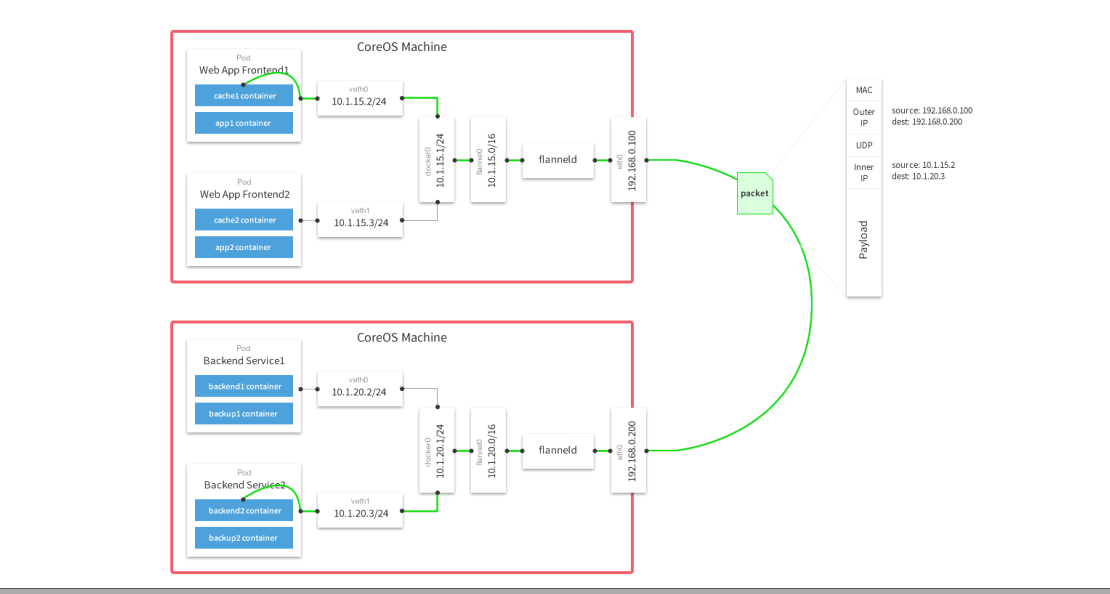
8.验证集群正常

echo “PATH=$PATH:/opt/kubernetes/bin” >> /etc/profile

etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="https://10.0.0.155:2379,https://10.0.0.154:2379,https://10.0.0.153:2379" cluster-health



五.部署Flannel网络



1)在master写入分配的子网段到etcd，供flanneld使用

/opt/kubernetes/bin/etcdctl \

> --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem \

> --endpoints="https://10.0.0.155:2379,https://10.0.0.154:2379,https://10.0.0.153:2379" \

> set /coreos.com/network/config '{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}'

{ "Network": "172.17.0.0/16", "Backend": {"Type": "vxlan"}}

2)下载flannel二进制包

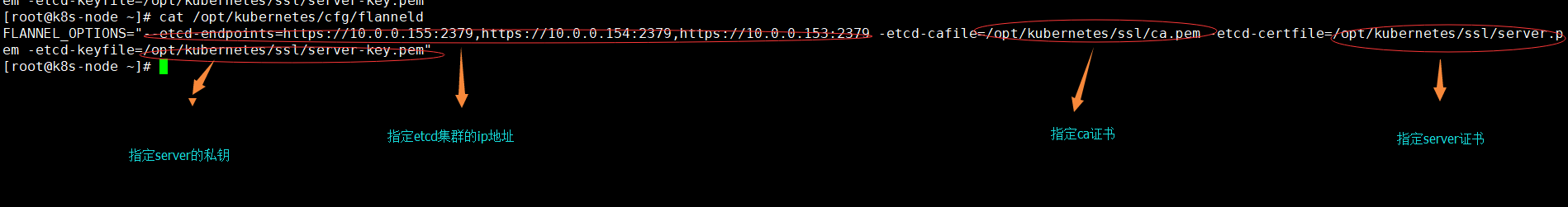
wget <https://github.com/coreos/flannel/releases/download/v0.9.1/flannel-v0.9.1-linux-amd64.tar.gz>

3) 在两个node节点配置Flannel

tar -xf flannel-v0.9.1-linux-amd64.tar.gz

scp flanneld mk-docker-opts.sh [root@10.0.0.153:/opt/kubernetes/bin/](mailto:root@10.0.0.153:/opt/kubernetes/bin/)

scp flannel [root@10.0.0.153:/opt/kubernetes/bin/](mailto:root@10.0.0.153:/opt/kubernetes/bin/)



4) systemd管理Flannel

cat <<EOF >/usr/lib/systemd/system/flanneld.service

[Unit]

Description=Flanneld overlay address etcd agent

After=network-online.target network.target

Before=docker.service

[Service]

Type=notify

EnvironmentFile=/opt/kubernetes/cfg/flanneld

ExecStart=/opt/kubernetes/bin/flanneld --ip-masq \$FLANNEL\_OPTIONS

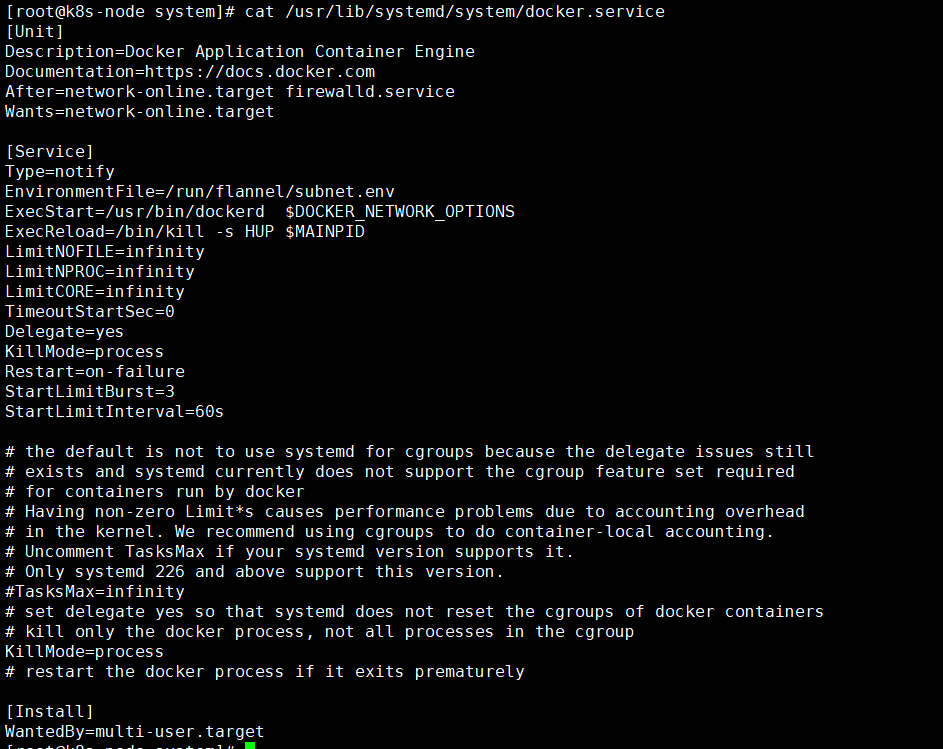
ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -k DOCKER\_NETWORK\_OPTIONS -d /run/flannel/subnet.env

Restart=on-failure

[Install]

WantedBy=multi-user.target

5) 配置Docker启动指定子网段



重启docker服务

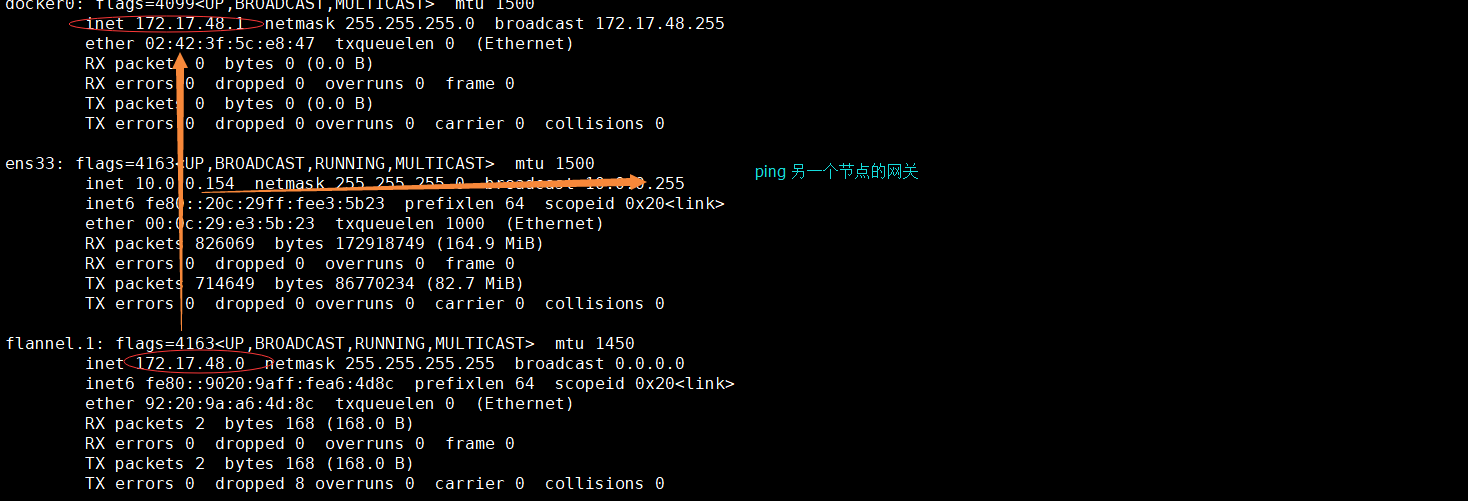
systemctl daemon-reload

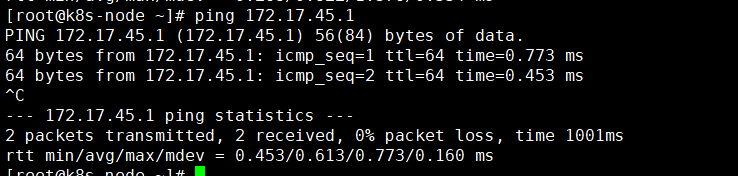
systemctl enable flanneld

systemctl restart flanneld

systemctl restart docker

6)测试flannel网络

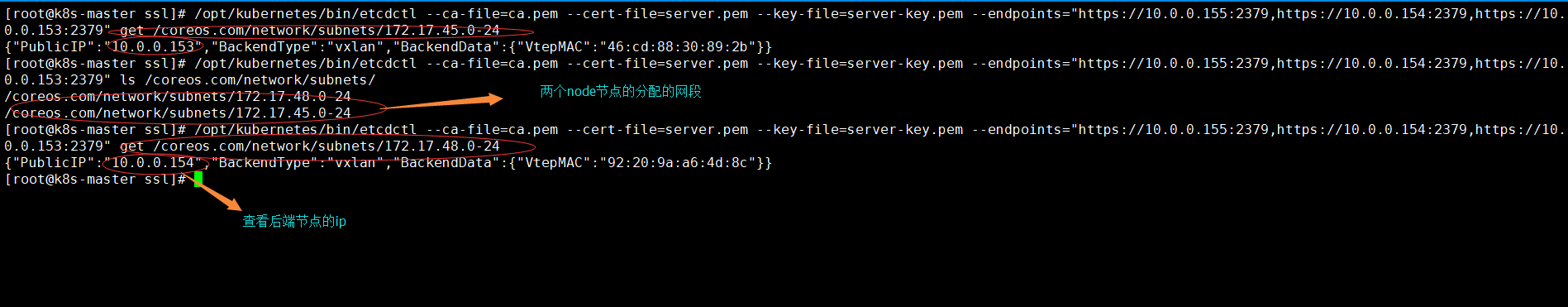




7)查看分配的网络

/opt/kubernetes/bin/etcdctl --ca-file=ca.pem --cert-file=server.pem --key-file=server-key.pem --endpoints="https://10.0.0.155:2379,https://10.0.0.154:2379,https://10.0.0.153:2379" ls /coreos.com/network/subnets

8)获取节点的ip



六 创建node节点的kubeconfig文件 使用kubelet为node节点颁发证书

1. 创建TLS Bootstrapping Token 引导kubelet生成证书

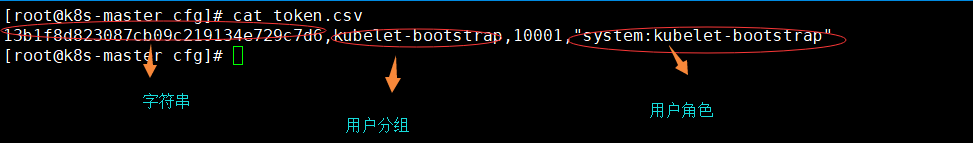
export BOOTSTRAP\_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')

cat > token.csv <<EOF

${BOOTSTRAP\_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"

EOF

2 使用用户组授权node节点



3. 创建kubelet bootstrapping kubeconfig

export KUBE\_APISERVER="https://192.168.0.211:6443"

# 设置集群参数

kubectl config set-cluster kubernetes \

--certificate-authority=./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

4. 创建kube-proxy kubeconfig

#设置集群参数

kubectl config set-cluster kubernetes \

--certificate-authority=./ca.pem \

--embed-certs=true \

--server=${KUBE\_APISERVER} \

--kubeconfig=kube-proxy.kubeconfig

# 设置客户端认证参数

kubectl config set-credentials kube-proxy \

--client-certificate=./kube-proxy.pem \

--client-key=./kube-proxy-key.pem \

--embed-certs=true \

--kubeconfig=kube-proxy.kubeconfig

# 设置上下文参数

kubectl config set-context default \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=kube-proxy.kubeconfig

# 设置默认上下文

kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig

七 部署master节点 三个组件 apiserver

1. 将master的二进制包移动到bin下面

mv kube-controller-manager kube-scheduler kube-apiserver /opt/kubernetes/bin

1. 部署apiserver 配置文件

KUBE\_APISERVER\_OPTS="--logtostderr=true \ #定义变量其实错误日志

--v=4 \

--etcd-servers=https://10.0.0.155:2379,https://10.0.0.154:2379,https://10.0.0.153:2379 \

#指定etcd的集群的server地址

--insecure-bind-address=127.0.0.1 \ #非安全的端口绑定本地 的ip不对外

--bind-address=10.0.0.155 \ #绑定的对外端口的ip

--insecure-port=8080 \ #指定非安全端口

--secure-port=6443 \ # 安全端口

--advertise-address=10.0.0.155 \ #通告的地址

--allow-privileged=true \ #启用容器集群的授权

--service-cluster-ip-range=10.10.10.0/24 \ #分配service集群的ip 的网段

--admission-control=NamespaceLifecycle,LimitRanger,SecurityContextDeny,ServiceAccount,ResourceQuota,NodeRestriction --authorization-mode=RBAC,Node \ #准入的模块 RBAC 作于认证

--kubelet-https=true \ #启用https访问

--enable-bootstrap-token-auth \ #启用node节点使用boostrap引导证书生成

--token-auth-file=/opt/kubernetes/cfg/token.csv \ #指定token文件

--service-node-port-range=30000-50000 \ #指定node节点端口范围

--tls-cert-file=/opt/kubernetes/ssl/server.pem \ #server证书

--tls-private-key-file=/opt/kubernetes/ssl/server-key.pem \ # server私钥

--client-ca-file=/opt/kubernetes/ssl/ca.pem \ #ca 证书

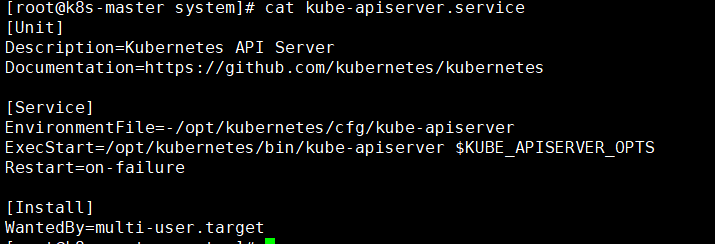
--service-account-key-file=/opt/kubernetes/ssl/ca-key.pem \ #ca key

--etcd-cafile=/opt/kubernetes/ssl/ca.pem \

--etcd-certfile=/opt/kubernetes/ssl/server.pem \ ##etcd证书

--etcd-keyfile=/opt/kubernetes/ssl/server-key.pem"

3.启动service的脚本



4．部署controller-manager、配置文件

KUBE\_CONTROLLER\_MANAGER\_OPTS="--logtostderr=true \ #启动错误日志级别

--v=4 \

--master=127.0.0.1:8080 \ #apiserver地址

--leader-elect=true \ #多个组件高可用做选举

--address=127.0.0.1 \ #连接地址

--service-cluster-ip-range=10.10.10.0/24 \ #虚拟vip

--cluster-name=kubernetes \ #集群名字

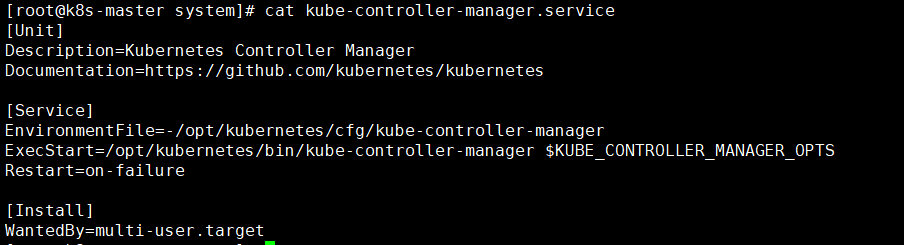
--cluster-signing-cert-file=/opt/kubernetes/ssl/ca.pem \ #ca 私钥 根 证书

--cluster-signing-key-file=/opt/kubernetes/ssl/ca-key.pem \

--service-account-private-key-file=/opt/kubernetes/ssl/ca-key.pem \

--root-ca-file=/opt/kubernetes/ssl/ca.pem"

5. controller-manager service文件



6 部署scheduler组件 配置文件

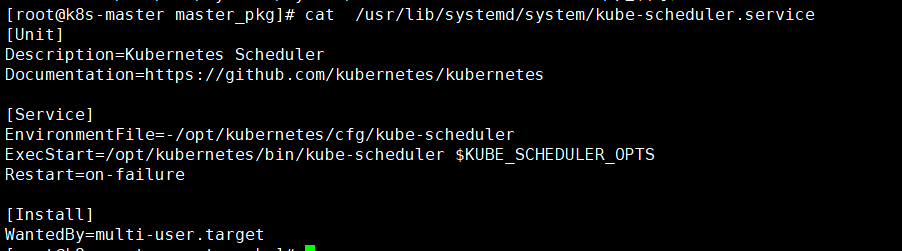
KUBE\_SCHEDULER\_OPTS="--logtostderr=true \ #日志级别

--v=4 \

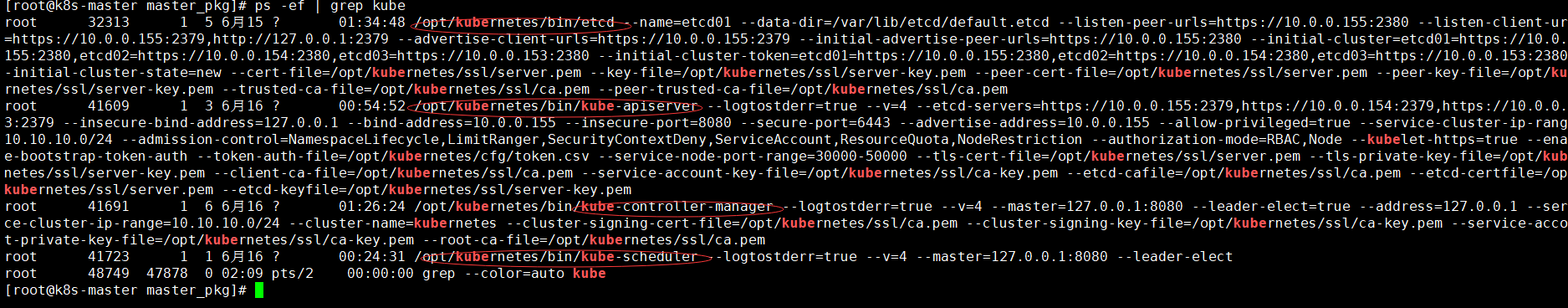
--master=127.0.0.1:8080 \

--leader-elect" #多个组件选举

7.scheduler的service文件



8.查看启动



9.查看节点组件状态

kubectl get componentstatus

八 部署node节点

1.将kubeconfig文件拷贝到节点上

scp \*kubeconfig [root@10.0.0.154:/opt/kubernetes/cfg/](mailto:root@10.0.0.154:/opt/kubernetes/cfg/)

scp \*kubeconfig [root@10.0.0.153:/opt/kubernetes/cfg/](mailto:root@10.0.0.153:/opt/kubernetes/cfg/)

2.部署node.zip 节点包

mv kubelet kube-proxy /opt/kubernetes/bin/

chmod +x /op/kubernetes/bin/\*

1. 部署kubelet 配置文件

KUBELET\_OPTS="--logtostderr=true \ # 日志级别

--v=4 \

--address=10.0.0.154 \ #当前监听的地址

--hostname-override=10.0.0.154 \ #当前节点的名字

--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \ #证书的引导自动生成的配置文件

--experimental-bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \ #用于引导生成证书的文件

--cert-dir=/opt/kubernetes/ssl \ #自动生成证书的位置

--allow-privileged=true \ # #启动授权

--cluster-dns=10.10.10.2 \ #dns地址

--cluster-domain=cluster.local \ #dns域

--fail-swap-on=false \ #关闭swap

--pod-infra-container-image=registry.cn-hangzhou.aliyuncs.com/google-containers/pause-amd64:3.0" #指定的国内pod的阿里云镜像

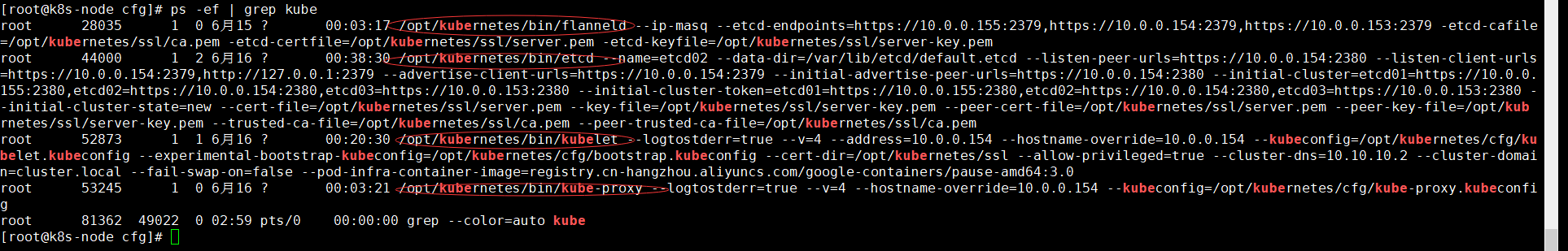
1. 部署proxy

KUBE\_PROXY\_OPTS="--logtostderr=true --v=4 #日志级别

--hostname-override=10.0.0.154 #指定kubelet地址一样

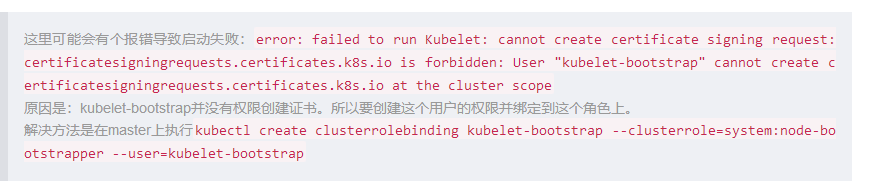
--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig" #指定kube-proxy配置文件 访问集群信息

5.查看服务启动状态



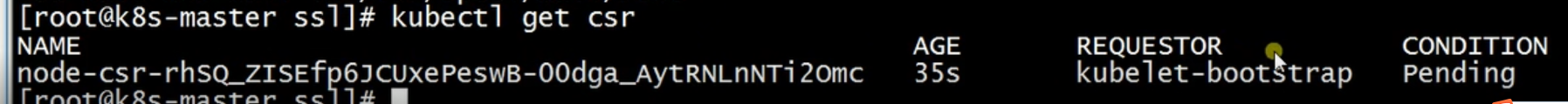
6启动kubelet prox 会没有权限访问 看到日志 /var/log/message

需要在master创建角色绑定权限



7.证书请求允许

Kubectl get csr 查看证书请求信息

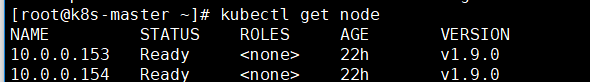


允许证书颁发

kubectl certificate approve node-csr-qpqmTyx7LNuU3kSc\_xJ\_J98oYn1GBbPi64ScF18EweI 允许证书

8.查看节点状态 准备状态

Kubectl get node



9. 查看为节点自动签发的证书加入集群



10.部署 第二个节点 直接从第一个拷贝改ip地址

scp –r /opt/kubernetes/bin [root@10.0.0.153:/opt/kubernetes/](mailto:root@10.0.0.153:/opt/kubernetes/)

scp –r /opt/kubernetes/cfg [root@10.0.0.153:/opt/kubernetes/](mailto:root@10.0.0.153:/opt/kubernetes/)

scp /usr/lib/systemd/system/kubelet.service [root@10.0.0.153:/usr/lib/systemd/system](mailto:root@10.0.0.153:/usr/lib/systemd/system)

scp /usr/lib/systemd/system/kube-proxy.service [root@10.0.0.153:/usr/lib/systemd/system](mailto:root@10.0.0.153:/usr/lib/systemd/system)

九 配置kubectl客户端使用https工具连接集群

1. node需要三个证书

2.设置集群项中名为kubernetes的apiserver地址与根证书

kubectl config set-cluster kubernetes --server=https://10.0.0.155:6443 --certificate-authority=ca.pem

设置用户项中cluster-admin用户证书认证字段

kubectl config set-credentials cluster-admin --certificate-authority=ca.pem --client-key=admin-key.pem --client-certificate=admin.pem

设置环境项中名为default的默认集群和用户

kubectl config set-context default --cluster=kubernetes --user=cluster-admin

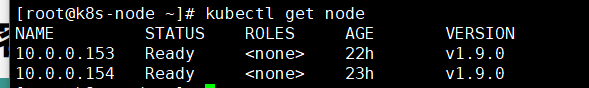
设置默认环境项为default

kubectl config use-context default



3 kubectl 测试访问

kubectl get node

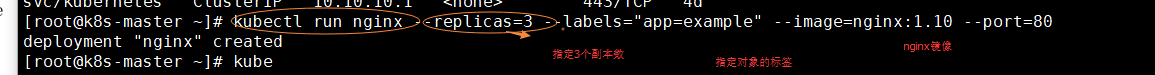


十．Kubectl管理工具命令

kubectl get all 查看所有节点

kubectl delete deploy/nginx 删除deploy镜像

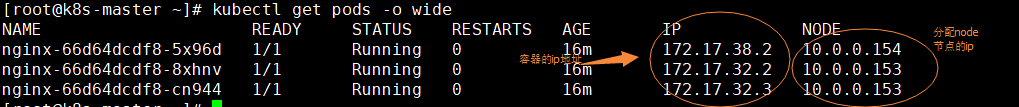
kubectl delete svc/nginx 删除发布镜像



kubectl describe po/nginx-66d64dcdf8-5x96d 查看pod 的详细信息



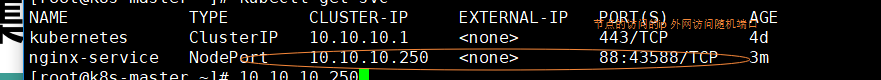
kubectl get pods --show-labels 查看pod标签

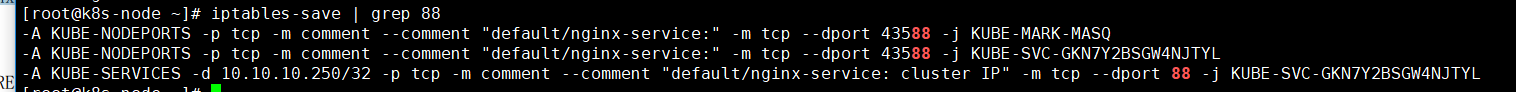


kubectl get deploy -o wide 查看当前列出的哪些镜像

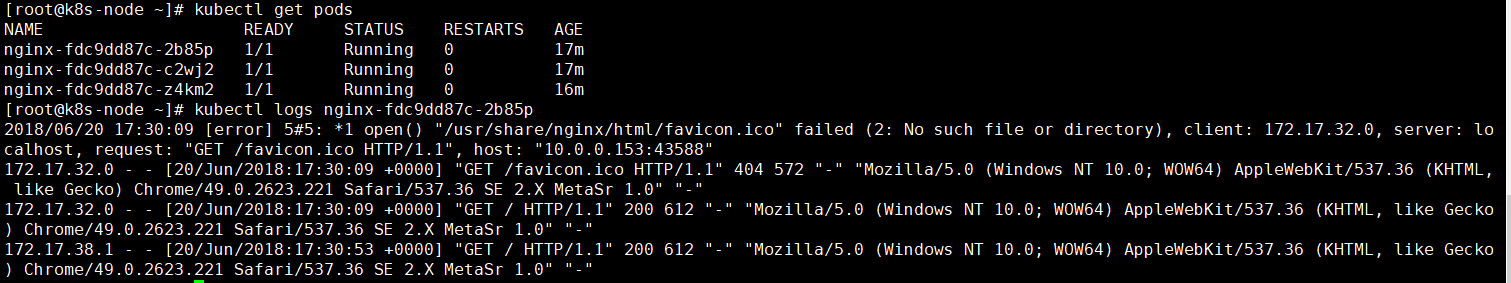
kubectl expose deployment nginx --port=88 --type=NodePort --target-port=80 --name=nginx-service 发布deployment nginx 负载的88端口 容器端口80 服务名

kubectl get svc 查看服务





kubectl describe deploy/nginx





kubectl set image deployment/nginx nginx=nginx:1.11 更新images

kubectl describe pod nginx-fdc9dd87c-2b85p 看到pod的信息

kubectl rollout status deploy/nginx 查看发布状态

kubectl rollout history deploy/nginx 查看发布的历史

kubectl rollout undo deployment/nginx 回滚

kubectl scale deployment nginx --replicas=10 扩容pod数

十一 yaml配置文件

Kubectl api-versions 查看api版本

创建nginx deloyment

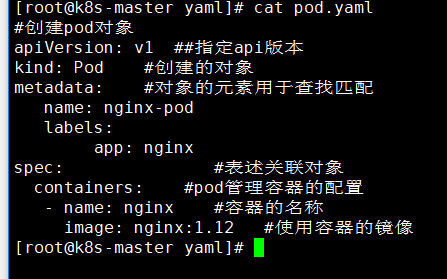


创建service发布



十二 pod配置

1. 创建pod对象



2.pod的基本管理

kubectl create -f pod.yaml 创建pod

kubectl get pods 查看pod

kubectl describe pod nginx-pod 描述

kubectl delete -f pod.yaml 删除

kubectl replace -f pod.yaml –force 重载

kubectl apply –f pod.yaml 更新资源

kubectl delete pod nginx-pod 删除pod

kebectl delete –f podz.yaml 删除pod资源

kubectl get pods –n kube-system 指定命名空间查看pod

kubectl get pods nginx-pod –o wide 查看当前pod节点分布哪个节点和ip

kubectl logs kubectl logs nginx-pod-probe 查看容器日志

1. pod的资源限制

apiVersion: v1

kind: Pod

metadata:

name: nginx-podz

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

resources: #-->resoures限制资源

requests:

memory: "64Mi" ---->指定内存64Mi

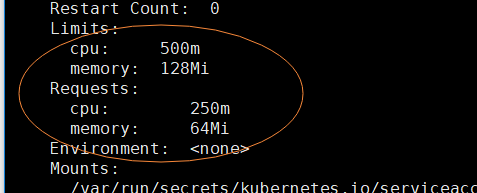
cpu: "250m" -🡪 cpu 资源

limits:

memory: "128Mi" 最大内存

cpu: "500m" cpu资源

用kubectl describe pod nginx-podz 查看pod详细资源



1. pod的调度约束

3.1使用pod.spec.nodeNam 强制的约束pod调度到执行的node节点上

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-nodename

labels:

app: nginx

spec:

nodeName: 10.0.0.154

nodeSelector:

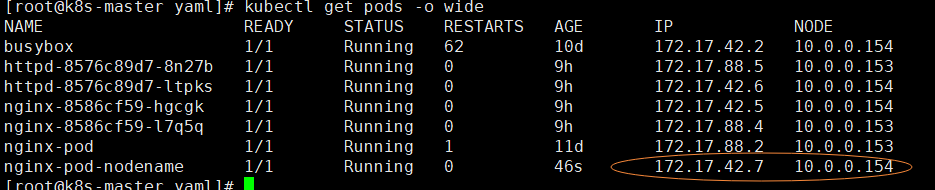
# env\_role: dev

containers:

- name: nginx

image: nginx

查看分布在哪个节点上



* 1. 使用pod.spec.nodeSelector 通过lable-selector机制选择节点

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-nodename

labels:

app: nginx

spec:

#nodeName: 10.0.0.154

nodeSelector:

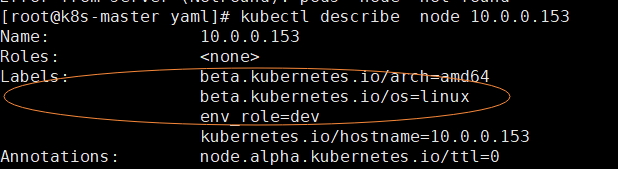
env\_role: dev

containers:

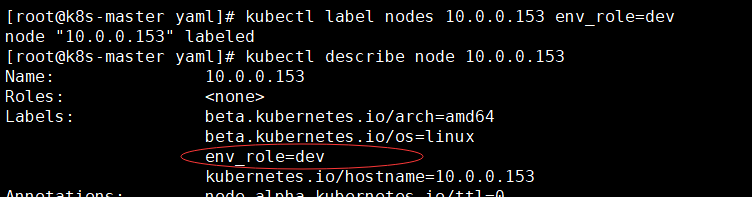
- name: nginx

image: nginx

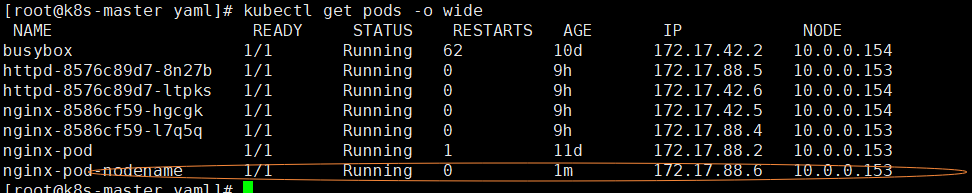
kubectl describe node 10.0.0.153 查看节点标签



kubectl label nodes 10.0.0.153 env\_role=dev 创建节点标签



查看节点分布



1. pod的重启策略

三种重启策略：

Always：当容器停止，总是重建容器， 默认策略。

OnFailure：当容器异常退出（退出状态码非0）时，才重启容器。

Never：当容器终止退出，从不重

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

restartPolicy: OnFailure

1. pod的健康检查 使用probe机制 两种类型：

livenessProbe

如果检查失败 会杀死容器 根据pod的restartPolicy 来操作 会重新重建容器

readinessProbe

如果检查失败 k8s 会把pod从service endpoint中剔除

Probe支持三种检查方法

HttpGet 发送http请求 返回状态码

Exec 执行shell 命令返回状态码

tcpSocket 发起TCP Socket 建立成功

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-probe

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

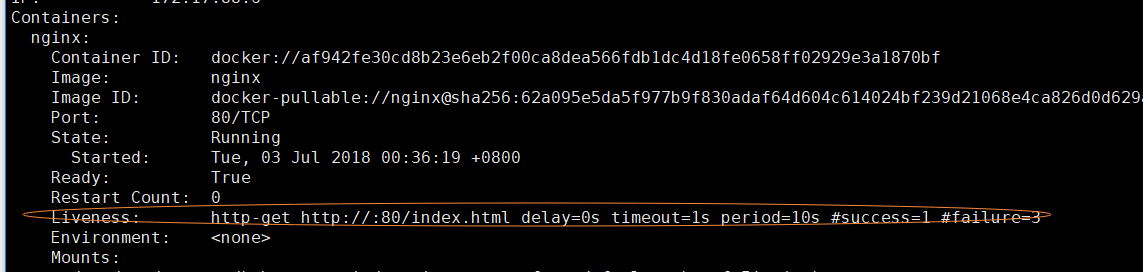
- containerPort: 80

livenessProbe:

httpGet:

path: /index.html

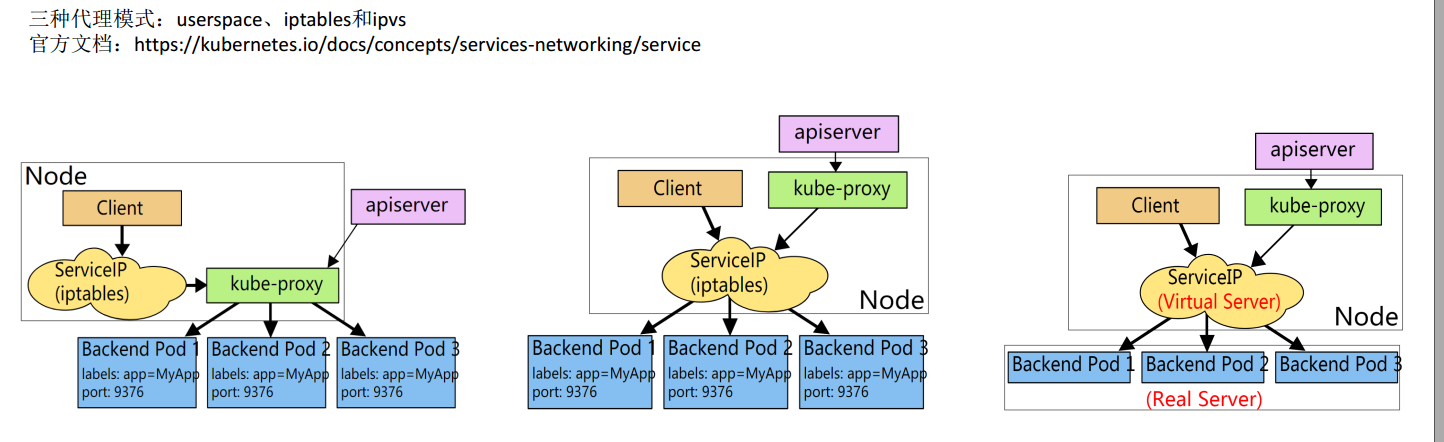
port: 80



获取帮助 kubectl explain pods.spec.containers

查看官方文档 <https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes>

1. service 转发代理pod 负载均衡



1.Client🡪ClusterIP🡪iptables🡪kube-proxy🡪node-backend-pod到节点

2.client🡪 iptables🡪node-backend-pod节点 ##iptable 是内核态

3.client-> clusterIp ->> ipvs转发—>node-pod

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector: 选择器

app: MyApp 标签选择器匹配后端应用

ports:

- name: http

protocol: TCP

port: 80 暴露service端口

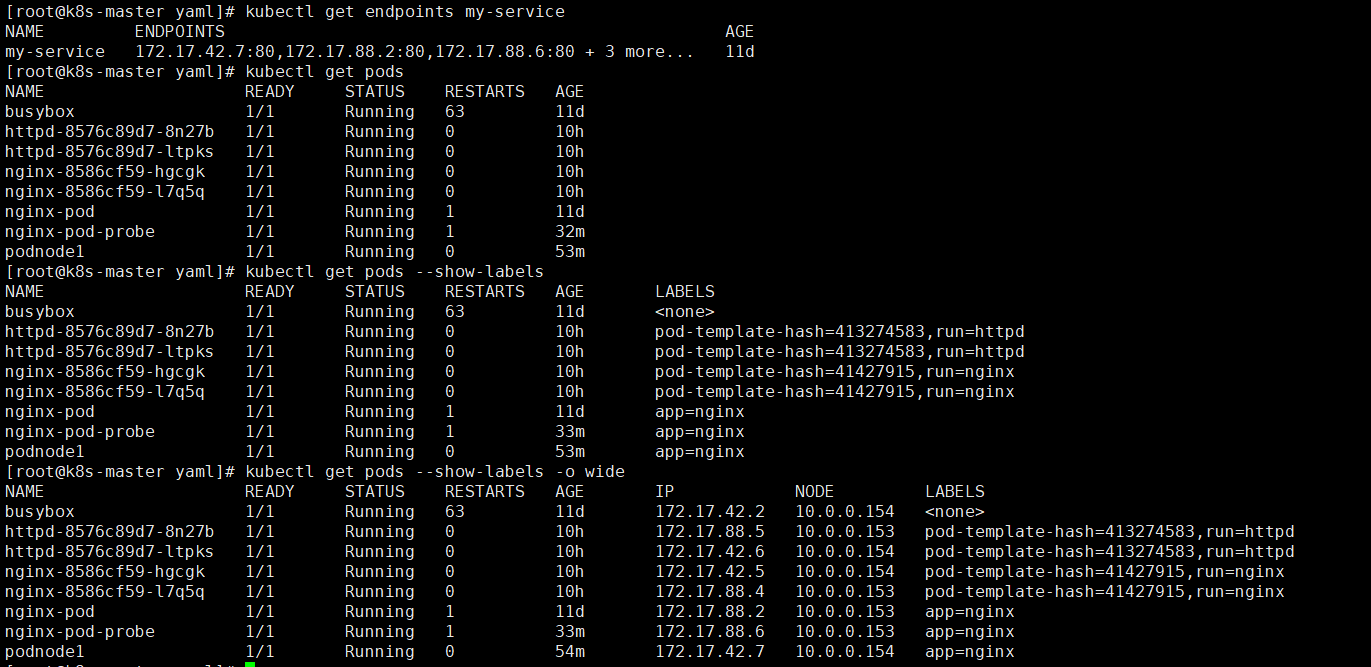
targetPort: 9376 容器端口80

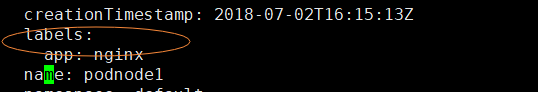
- name: https

protocol: TCP

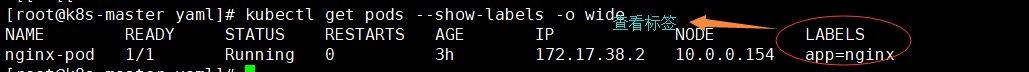
port: 443

targetPort: 9377

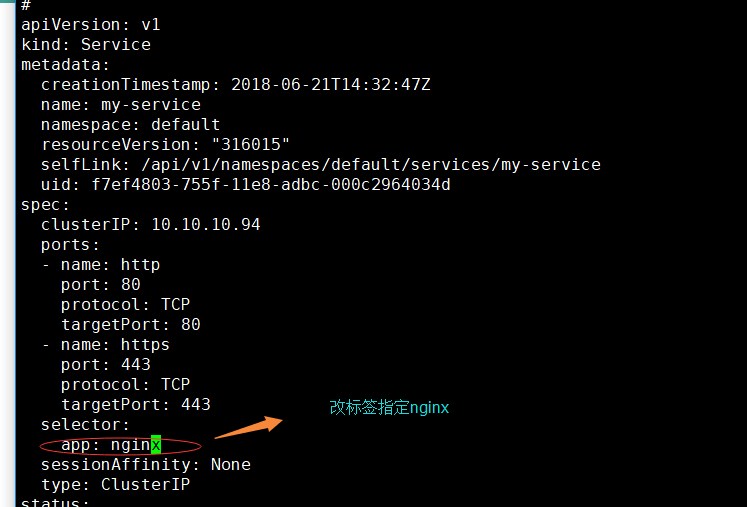


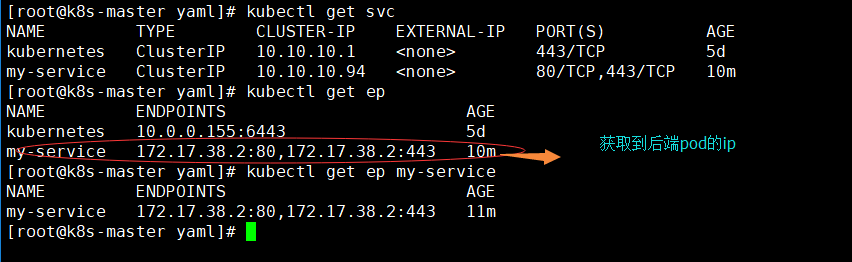


创建一个项目deployment和service是通过标签做相关关联发布



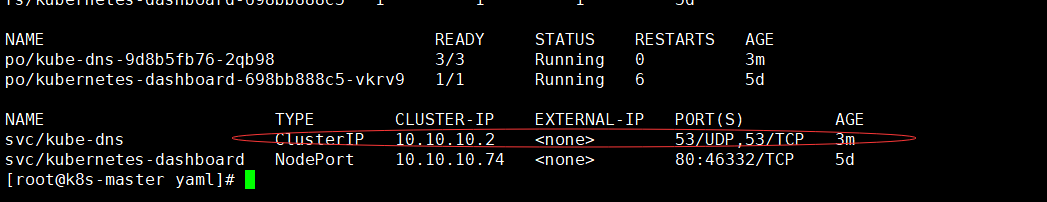
kubectl edit svc/my-service 编辑svc

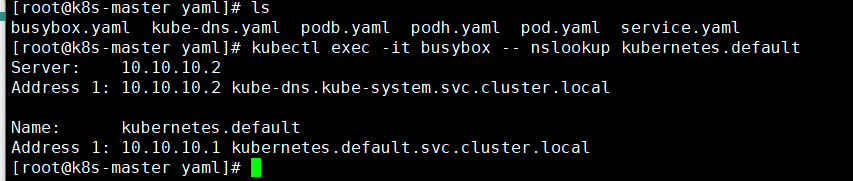




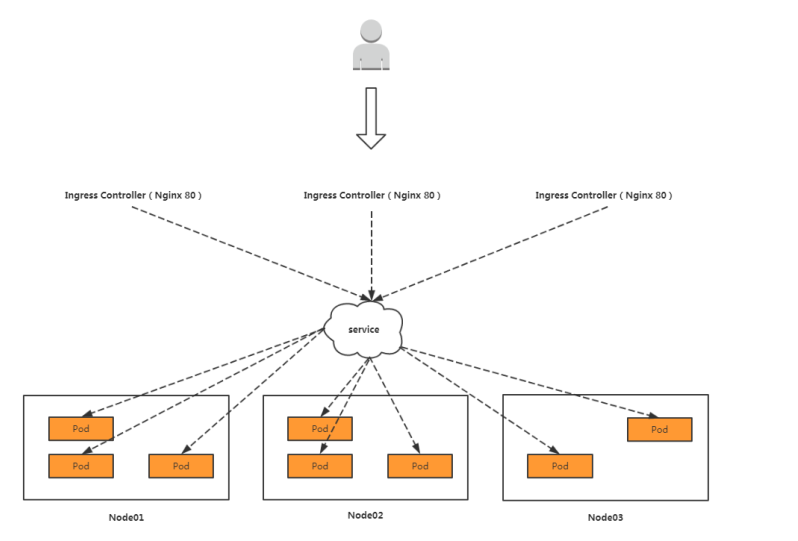


kubectl get all -n kube-system 查看kubedns 命名空间





十四 ingress 七层代理



kubectl create -f namespace.yaml 创建命名空间

kubectl create -f default-backend.yaml 创建Deployment对象

kubectl create -f tcp-services-configmap.yaml

kubectl create -f udp-services-configmap.yaml

kubectl create -f rbac.yaml

kubectl create -f deployment.yaml

kubectl delete po/nginx-glusterfs-b86fb66b9-zbjvt --grace-period=0 –force 强制删除pod

kubectl get pods -n ingress-nginx

1. curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/namespace.yaml \

| kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/default-backend.yaml \

| kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/tcp-services-configmap.yaml \

| kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/udp-services-configmap.yaml \

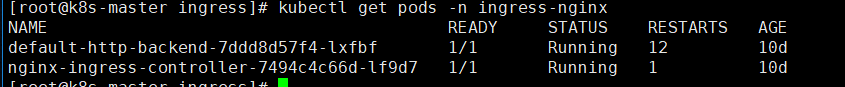
| kubectl apply -f -

curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/rbac.yaml \

| kubectl apply -f -

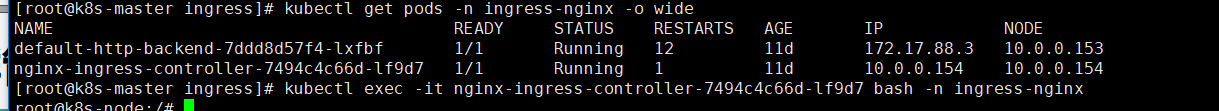
curl https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/with-rbac.yaml \

| kubectl apply –f

2. 

kubectl get ingress

kubectl get pods -n ingress-nginx -o wide



kubectl get all -n ingress-nginx



cfssl print-defaults csr > ca.csr.json

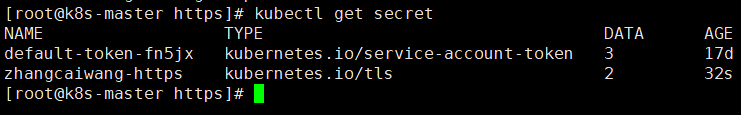
cfssl print-defaults config > ca-config.json

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

cfssl print-defaults csr > service.json

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem --config=ca-config.json --profile=www service.json | cfssljson -bare server

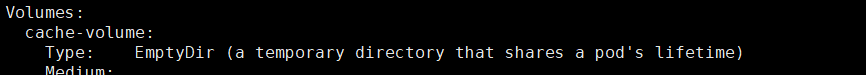
kubectl create secret tls zhangcaiwang-https --key server-key.pem --cert server.pem



kubectl get ing

kubectl exec nginx-ingress-controller-7494c4c66d-lf9d7 -it cat /etc/nginx/nginx.conf -n ingress-nginx





K8s结合glusterfs存储搭建分布式存储

1.创建endpoints

kubectl create -f glusterfs-endpoints.json

[root@k8s-master volume]# cat glusterfs-endpoints.json

{

"kind": "Endpoints",

"apiVersion": "v1",

"metadata": {

"name": "glusterfs-cluster"

},

"subsets": [

{

"addresses": [

{

"ip": "192.168.0.214"

}

],

"ports": [

{

"port": 1

}

]

},

{

"addresses": [

{

"ip": "192.168.0.215"

}

],

"ports": [

{

"port": 1

}

]

}

]

}

2.创建service 发布

[root@k8s-master volume]# cat glusterfs-service.json

{

"kind": "Service",

"apiVersion": "v1",

"metadata": {

"name": "glusterfs-cluster"

},

"spec": {

"ports": [

{"port": 1}

]

}

}

Kubectl create –f glusterfs-service.json

创建 Deployment挂载发布

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

volumeMounts:

- name: glusterfsvol

mountPath: /usr/share/nginx/html

ports:

- containerPort: 80

volumes:

- name: glusterfsvol

glusterfs:

endpoints: glusterfs-cluster

path: volume1

readOnly: false

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

selector:

app: nginx

ports:

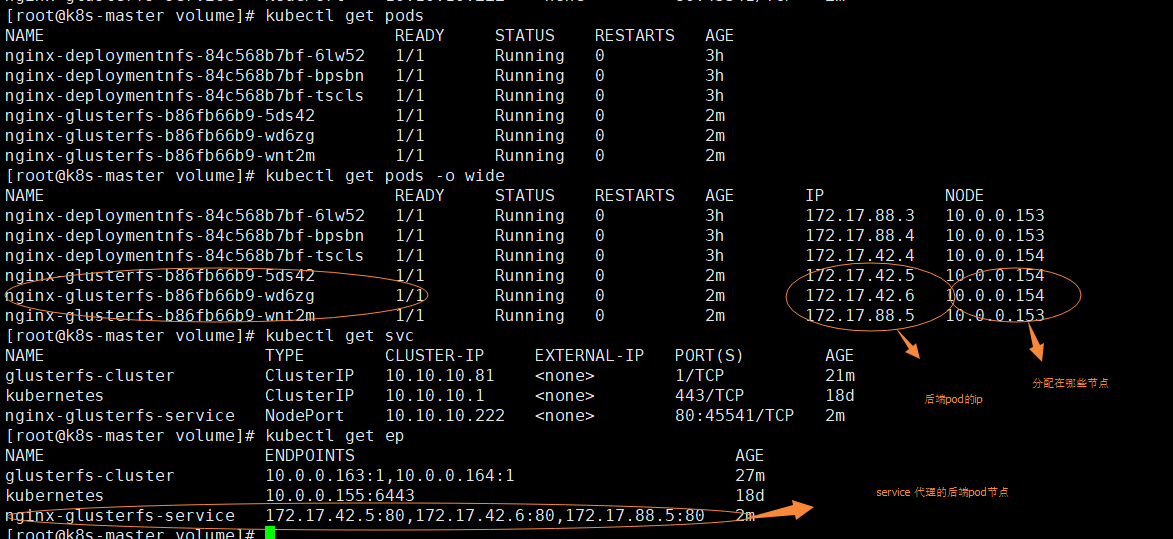
- name: http

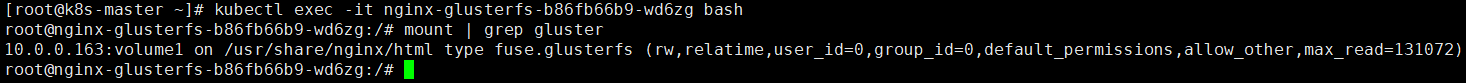
port: 80

protocol: TCP

targetPort: 80

type: NodePort





Gluster fs搭建

1. yum install -y glusterfs glusterfs-server glusterfs-fuse glusterfs-rdma
2. systemctl enable glusterd.service
3. systemctl start glusterd.service
4. gluster peer probe glusterfs02
5. gluster peer status
6. mkdir -p /opt/glusterfs/data
7. gluster volume create volume1 replica 2 transport tcp glusterfs01:/opt/glusterfs/data glusterfs02:/opt/glusterfs/data force
8. gluster volume start volume1
9. gluster volume info

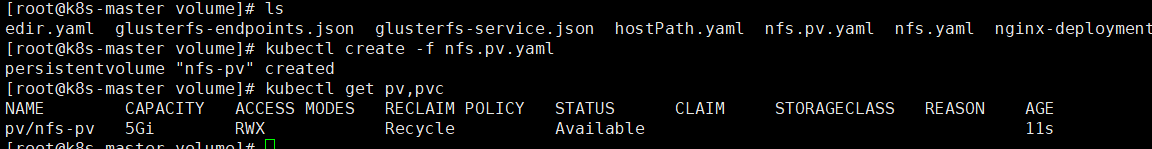
客户端挂载：

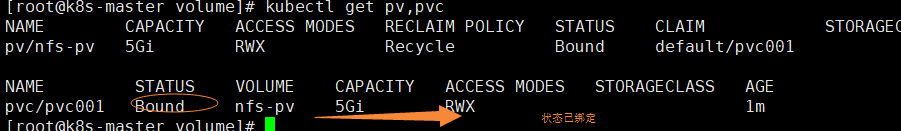
1. yum install -y glusterfs glusterfs-fuse
2. mount -t glusterfs glusterfs01:/volume1 /mnt 指定volume1卷名

持久卷 pv 持久卷 对存储抽象实现 使得存储作为集群中的资源

Pvc: 消费pv的资源

Pod申请PVC作为卷来使用，集群通过PVC查找绑定的PV，并Mount给Pod。





普通Volume和使用它的Pod之间是一种静态绑定关系，在定义Pod的文件里，同时定义了它使用的Volume。Volume 是Pod的附属品，我们无法单独创建一个Volume，因为它不是一个独立的K8S资源对象。

而Persistent Volume 简称PV是一个K8S资源对象，所以我们可以单独创建一个PV。它不和Pod直接发生关系，而是通过Persistent Volume Claim，简称PVC来实现动态绑定。Pod定义里指定的是PVC，然后PVC会根据Pod的要求去自动绑定合适的PV给Pod使用。

一个PV创建完后状态会变成Available，等待被PVC绑定。一旦被PVC邦定，PV的状态会变成Bound，就可以被定义了相应PVC的Pod使用。Pod使用完后会释放PV，PV的状态变成Released。变成Released的PV会根据定义的回收策略做相应的回收工作。有三种回收策略，Retain、Delete 和 Recycle。Retain就是保留现场，K8S什么也不做，等待用户手动去处理PV里的数据，处理完后，再手动删除PV。Delete 策略，K8S会自动删除该PV及里面的数据。Recycle方式，K8S会将PV里的数据删除，然后把PV的状态变成Available，又可以被新的PVC绑定使用。

在实际使用场景里，PV的创建和使用通常不是同一个人。这里有一个典型的应用场景：管理员创建一个PV池，开发人员创建Pod和PVC，PVC里定义了Pod所需存储的大小和访问模式，然后PVC会到PV池里自动匹配最合适的PV给Pod使用。

https://blog.csdn.net/liukuan73/article/details/60089305