**kubernetes 1.12.1**

**文章来源：**[**https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html**](https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html) **致谢**

**CHANGELOG**

1. docker 版本 更新为 1.11.1,1.12.1,1.13.1,17.03,17.06,17.09,18.06

2. etcd 版本 更新为 v3.2.24 , 并且在1.13.x 版本中删除 etcd v2 支持

3. [ POD 自动伸缩 ] kubectl autoscale deployment nginx --min=2 --max=10

kubectl get horizontalpodautoscalers 查看伸缩的服务

获取 伸缩的指标 需要部署 metrics-server 到集群中

https://github.com/kubernetes-incubator/metrics-server

4. 已知问题BUG:

kubelet: E1009 17:24:03.820462 10457 azure\_dd.go:147] failed to get azure cloud in GetVolumeLimits

**kubernetes 1.12.1**

**环境说明**

基于 二进制 文件部署 本地化 kube-apiserver, kube-controller-manager , kube-scheduler 我这边配置 既是 master 也是 nodes

这里配置2个Master 1个node,K8S-1 只做 Master, K8S-2 既是 Master 也是 Node, k8s-3 只做单纯 Node

k8s-1: 172.16.10.21

k8s-2: 172.16.10.22

k8s-3: 172.16.10.23

**初始化环境**

hostnamectl --static set-hostname hostname

k8s-1: 172.16.10.21

k8s-2: 172.16.10.22

k8s-3: 172.16.10.23

#编辑 /etc/hosts 文件，配置hostname 通信

vi /etc/hosts

172.16.10.21 k8s-1

172.16.10.22 k8s-2

172.16.10.23 k8s-3

**创建 验证**

这里使用 CloudFlare 的 PKI 工具集 cfssl 来生成 Certificate Authority (CA) 证书和秘钥文件。

**安装 cfssl**

mkdir -p /opt/local/cfssl

cd /opt/local/cfssl

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

mv cfssl\_linux-amd64 cfssl

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

mv cfssljson\_linux-amd64 cfssljson

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

mv cfssl-certinfo\_linux-amd64 cfssl-certinfo

chmod +x \*

**创建 CA 证书配置**

mkdir /opt/ssl

cd /opt/ssl

# config.json 文件

vi config.json

{

"signing": {

"default": {

"expiry": "87600h"

},

"profiles": {

"kubernetes": {

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

],

"expiry": "87600h"

}

}

}

}

# csr.json 文件

vi csr.json

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShenZhen",

"L": "ShenZhen",

"O": "k8s",

"OU": "System"

}

]

}

**生成 CA 证书和私钥**

cd /opt/ssl/

/opt/local/cfssl/cfssl gencert -initca csr.json | /opt/local/cfssl/cfssljson -bare ca

[root@k8s-1 ssl]# ls -lt

总用量 20

-rw-r--r-- 1 root root 1005 7月 3 17:26 ca.csr

-rw------- 1 root root 1675 7月 3 17:26 ca-key.pem

-rw-r--r-- 1 root root 1363 7月 3 17:26 ca.pem

-rw-r--r-- 1 root root 210 7月 3 17:24 csr.json

-rw-r--r-- 1 root root 292 7月 3 17:23 config.json

**分发证书**

# 创建证书目录

mkdir -p /etc/kubernetes/ssl

# 拷贝所有文件到目录下

cp \*.pem /etc/kubernetes/ssl

cp ca.csr /etc/kubernetes/ssl

# 这里要将文件拷贝到所有的k8s 机器上

scp \*.pem \*.csr 172.16.10.22:/etc/kubernetes/ssl/

scp \*.pem \*.csr 172.16.10.23:/etc/kubernetes/ssl/

**安装 docker**

官方最新版本 docker 为 18.06.1 , 官方验证最高版本支持到 18.06.0

# 导入 yum 源

# 安装 yum-config-manager

yum -y install yum-utils

# 导入

yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

# 更新 repo

yum makecache

# 查看yum 版本

yum list docker-ce.x86\_64 --showduplicates |sort -r

# 安装指定版本 docker-ce 18.06 被 docker-ce-selinux 依赖, 不能直接yum 安装 docker-ce-selinux

wget https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/docker-ce-18.06.0.ce-3.el7.x86\_64.rpm

rpm -ivh docker-ce-18.06.0.ce-3.el7.x86\_64.rpm

yum -y install docker-ce-18.06.0.ce

# 查看安装

docker version

Client:

Version: 18.06.0-ce

API version: 1.38

Go version: go1.10.3

Git commit: 0ffa825

Built: Wed Jul 18 19:08:18 2018

OS/Arch: linux/amd64

**更改docker 配置**

# 添加配置

vi /etc/systemd/system/docker.service

[Unit]

Description=Docker Application Container Engine

Documentation=http://docs.docker.com

After=network.target docker-storage-setup.service

Wants=docker-storage-setup.service

[Service]

Type=notify

Environment=GOTRACEBACK=crash

ExecReload=/bin/kill -s HUP $MAINPID

Delegate=yes

KillMode=process

ExecStart=/usr/bin/dockerd \

$DOCKER\_OPTS \

$DOCKER\_STORAGE\_OPTIONS \

$DOCKER\_NETWORK\_OPTIONS \

$DOCKER\_DNS\_OPTIONS \

$INSECURE\_REGISTRY

LimitNOFILE=1048576

LimitNPROC=1048576

LimitCORE=infinity

TimeoutStartSec=1min

Restart=on-abnormal

[Install]

WantedBy=multi-user.target

# 修改其他配置

# 低版本内核， kernel 3.10.x 配置使用 overlay2

vi /etc/docker/daemon.json

{

"storage-driver": "overlay2",

"storage-opts": [

"overlay2.override\_kernel\_check=true"

]

}

mkdir -p /etc/systemd/system/docker.service.d/

vi /etc/systemd/system/docker.service.d/docker-options.conf

# 添加如下 : (注意 environment 必须在同一行，如果出现换行会无法加载)

# docker 版本 17.03.2 之前配置为 --graph=/opt/docker

# docker 版本 17.04.x 之后配置为 --data-root=/opt/docker

[Service]

Environment="DOCKER\_OPTS=--insecure-registry=10.254.0.0/16 \

--registry-mirror=http://b438f72b.m.daocloud.io \

--data-root=/opt/docker --log-opt max-size=50m --log-opt max-file=5"

vi /etc/systemd/system/docker.service.d/docker-dns.conf

# 添加如下 :

[Service]

Environment="DOCKER\_DNS\_OPTIONS=\

--dns 10.254.0.2 --dns 114.114.114.114 \

--dns-search default.svc.cluster.local --dns-search svc.cluster.local \

--dns-opt ndots:2 --dns-opt timeout:2 --dns-opt attempts:2"

# 重新读取配置，启动 docker

systemctl daemon-reload

systemctl start docker

systemctl enable docker

# 如果报错 请使用

journalctl -f -t docker 和 journalctl -u docker 来定位问题

**etcd 集群**

etcd 是k8s集群最重要的组件， etcd 挂了，集群就挂了， 1.12.1 etcd 支持最新版本为 v3.2.24

**安装 etcd**

官方地址 https://github.com/coreos/etcd/releases

# 下载 二进制文件

wget https://github.com/coreos/etcd/releases/download/v3.2.24/etcd-v3.2.24-linux-amd64.tar.gz

tar zxvf etcd-v3.2.24-linux-amd64.tar.gz

cd etcd-v3.2.24-linux-amd64

mv etcd etcdctl /usr/bin/

**创建 etcd 证书**

etcd 证书这里，默认配置三个，后续如果需要增加，更多的 etcd 节点 这里的认证IP 请多预留几个，以备后续添加能通过认证，不需要重新签发

cd /opt/ssl/

vi etcd-csr.json

{

"CN": "etcd",

"hosts": [

"127.0.0.1",

"172.16.10.21",

"172.16.10.22",

"172.16.10.23"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShenZhen",

"L": "ShenZhen",

"O": "k8s",

"OU": "System"

}

]

}

# 生成 etcd 密钥

/opt/local/cfssl/cfssl gencert -ca=/opt/ssl/ca.pem \

-ca-key=/opt/ssl/ca-key.pem \

-config=/opt/ssl/config.json \

-profile=kubernetes etcd-csr.json | /opt/local/cfssl/cfssljson -bare etcd

# 查看生成

[root@k8s-1 ssl]# ls etcd\*

etcd.csr etcd-csr.json etcd-key.pem etcd.pem

# 检查证书

[root@k8s-1 ssl]# /opt/local/cfssl/cfssl-certinfo -cert etcd.pem

# 拷贝到etcd服务器

# etcd-1

cp etcd\*.pem /etc/kubernetes/ssl/

# etcd-2

scp etcd\*.pem 172.16.10.22:/etc/kubernetes/ssl/

# etcd-3

scp etcd\*.pem 172.16.10.23:/etc/kubernetes/ssl/

# 如果 etcd 非 root 用户，读取证书会提示没权限

chmod 644 /etc/kubernetes/ssl/etcd-key.pem

**修改 etcd 配置**

由于 etcd 是最重要的组件，所以 –data-dir 请配置到其他路径中

# 创建 etcd data 目录， 并授权

useradd etcd

mkdir -p /opt/etcd

chown -R etcd:etcd /opt/etcd

# etcd-1

vi /etc/systemd/system/etcd.service

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

WorkingDirectory=/opt/etcd/

User=etcd

# set GOMAXPROCS to number of processors

ExecStart=/usr/bin/etcd \

--name=etcd1 \

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

--peer-cert-file=/etc/kubernetes/ssl/etcd.pem \

--peer-key-file=/etc/kubernetes/ssl/etcd-key.pem \

--trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--initial-advertise-peer-urls=https://172.16.10.21:2380 \

--listen-peer-urls=https://172.16.10.21:2380 \

--listen-client-urls=https://172.16.10.21:2379,http://127.0.0.1:2379 \

--advertise-client-urls=https://172.16.10.21:2379 \

--initial-cluster-token=k8s-etcd-cluster \

--initial-cluster=etcd1=https://172.16.10.21:2380,etcd2=https://172.16.10.22:2380,etcd3=https://172.16.10.23:2380 \

--initial-cluster-state=new \

--data-dir=/opt/etcd/

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

# etcd-2

vi /etc/systemd/system/etcd.service

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

WorkingDirectory=/opt/etcd/

User=etcd

# set GOMAXPROCS to number of processors

ExecStart=/usr/bin/etcd \

--name=etcd2 \

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

--peer-cert-file=/etc/kubernetes/ssl/etcd.pem \

--peer-key-file=/etc/kubernetes/ssl/etcd-key.pem \

--trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--initial-advertise-peer-urls=https://172.16.10.22:2380 \

--listen-peer-urls=https://172.16.10.22:2380 \

--listen-client-urls=https://172.16.10.22:2379,http://127.0.0.1:2379 \

--advertise-client-urls=https://172.16.10.22:2379 \

--initial-cluster-token=k8s-etcd-cluster \

--initial-cluster=etcd1=https://172.16.10.21:2380,etcd2=https://172.16.10.22:2380,etcd3=https://172.16.10.23:2380 \

--initial-cluster-state=new \

--data-dir=/opt/etcd

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

# etcd-3

vi /etc/systemd/system/etcd.service

[Unit]

Description=Etcd Server

After=network.target

After=network-online.target

Wants=network-online.target

[Service]

Type=notify

WorkingDirectory=/opt/etcd/

User=etcd

# set GOMAXPROCS to number of processors

ExecStart=/usr/bin/etcd \

--name=etcd3 \

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

--peer-cert-file=/etc/kubernetes/ssl/etcd.pem \

--peer-key-file=/etc/kubernetes/ssl/etcd-key.pem \

--trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \

--initial-advertise-peer-urls=https://172.16.10.23:2380 \

--listen-peer-urls=https://172.16.10.23:2380 \

--listen-client-urls=https://172.16.10.23:2379,http://127.0.0.1:2379 \

--advertise-client-urls=https://172.16.10.23:2379 \

--initial-cluster-token=k8s-etcd-cluster \

--initial-cluster=etcd1=https://172.16.10.21:2380,etcd2=https://172.16.10.22:2380,etcd3=https://172.16.10.23:2380 \

--initial-cluster-state=new \

--data-dir=/opt/etcd/

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

**启动 etcd**

分别启动 所有节点的 etcd 服务

systemctl daemon-reload

systemctl enable etcd

systemctl start etcd

systemctl status etcd

# 如果报错 请使用

journalctl -f -t etcd 和 journalctl -u etcd 来定位问题

**验证 etcd 集群状态**

查看 etcd 集群状态：

etcdctl --endpoints=https://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379\

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--ca-file=/etc/kubernetes/ssl/ca.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

cluster-health

member 35eefb8e7cc93b53 is healthy: got healthy result from https://172.16.10.23:2379

member 4576ff5ed626a66b is healthy: got healthy result from https://172.16.10.21:2379

member bf3bd651ec832339 is healthy: got healthy result from https://172.16.10.22:2379

cluster is healthy

查看 etcd 集群成员：

etcdctl --endpoints=https://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379\

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--ca-file=/etc/kubernetes/ssl/ca.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

member list

35eefb8e7cc93b53: name=etcd3 peerURLs=https://172.16.10.23:2380 clientURLs=https://172.16.10.23:2379 isLeader=false

4576ff5ed626a66b: name=etcd1 peerURLs=https://172.16.10.21:2380 clientURLs=https://172.16.10.21:2379 isLeader=true

bf3bd651ec832339: name=etcd2 peerURLs=https://172.16.10.22:2380 clientURLs=https://172.16.10.22:2379 isLeader=false

**配置 Kubernetes 集群**

kubectl 安装在所有需要进行操作的机器上

**Master and Node**

Master 需要部署 kube-apiserver , kube-scheduler , kube-controller-manager 这三个组件。 kube-scheduler 作用是调度pods分配到那个node里，简单来说就是资源调度。

kube-controller-manager 作用是 对 deployment controller , replication controller, endpoints controller, namespace controller, and serviceaccounts controller等等的循环控制，与kube-apiserver交互。

**安装组件**

# 从github 上下载版本

cd /tmp

wget https://dl.k8s.io/v1.12.1/kubernetes-server-linux-amd64.tar.gz

tar -xzvf kubernetes-server-linux-amd64.tar.gz

cd kubernetes

cp -r server/bin/{kube-apiserver,kube-controller-manager,kube-scheduler,kubectl,kubelet,kubeadm} /usr/local/bin/

scp server/bin/{kube-apiserver,kube-controller-manager,kube-scheduler,kubectl,kube-proxy,kubelet,kubeadm} 172.16.10.22:/usr/local/bin/

scp server/bin/{kube-proxy,kubelet} 172.16.10.23:/usr/local/bin/

**创建 admin 证书**

kubectl 与 kube-apiserver 的安全端口通信，需要为安全通信提供 TLS 证书和秘钥。

cd /opt/ssl/

vi admin-csr.json

{

"CN": "admin",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShenZhen",

"L": "ShenZhen",

"O": "system:masters",

"OU": "System"

}

]

}

# 生成 admin 证书和私钥

cd /opt/ssl/

/opt/local/cfssl/cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \

-ca-key=/etc/kubernetes/ssl/ca-key.pem \

-config=/opt/ssl/config.json \

-profile=kubernetes admin-csr.json | /opt/local/cfssl/cfssljson -bare admin

# 查看生成

[root@k8s-1 ssl]# ls admin\*

admin.csr admin-csr.json admin-key.pem admin.pem

cp admin\*.pem /etc/kubernetes/ssl/

scp admin\*.pem 172.16.10.22:/etc/kubernetes/ssl/

**生成 kubernetes 配置文件**

生成证书相关的配置文件存储与 /root/.kube 目录中

# 配置 kubernetes 集群

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=https://127.0.0.1:6443

# 配置 客户端认证

kubectl config set-credentials admin \

--client-certificate=/etc/kubernetes/ssl/admin.pem \

--embed-certs=true \

--client-key=/etc/kubernetes/ssl/admin-key.pem

kubectl config set-context kubernetes \

--cluster=kubernetes \

--user=admin

kubectl config use-context kubernetes

**创建 kubernetes 证书**

cd /opt/ssl

vi kubernetes-csr.json

{

"CN": "kubernetes",

"hosts": [

"127.0.0.1",

"172.16.10.21",

"172.16.10.22",

"172.16.10.23",

"10.254.0.1",

"kubernetes",

"kubernetes.default",

"kubernetes.default.svc",

"kubernetes.default.svc.cluster",

"kubernetes.default.svc.cluster.local"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShenZhen",

"L": "ShenZhen",

"O": "k8s",

"OU": "System"

}

]

}

## 这里 hosts 字段中 三个 IP 分别为 127.0.0.1 本机， 172.16.10.21 和 172.16.10.22 为 Master 的IP，多个Master需要写多个。 10.254.0.1 为 kubernetes SVC 的 IP， 一般是 部署网络的第一个IP , 如: 10.254.0.1 ， 在启动完成后，我们使用 kubectl get svc ， 就可以查看到

**生成 kubernetes 证书和私钥**

/opt/local/cfssl/cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \

-ca-key=/etc/kubernetes/ssl/ca-key.pem \

-config=/opt/ssl/config.json \

-profile=kubernetes kubernetes-csr.json | /opt/local/cfssl/cfssljson -bare kubernetes

# 查看生成

[root@k8s-1 ssl]# ls -lt kubernetes\*

-rw-r--r-- 1 root root 1261 11月 16 15:12 kubernetes.csr

-rw------- 1 root root 1679 11月 16 15:12 kubernetes-key.pem

-rw-r--r-- 1 root root 1635 11月 16 15:12 kubernetes.pem

-rw-r--r-- 1 root root 475 11月 16 15:12 kubernetes-csr.json

# 拷贝到目录

cp kubernetes\*.pem /etc/kubernetes/ssl/

scp kubernetes\*.pem 172.16.10.22:/etc/kubernetes/ssl/

**配置 kube-apiserver**

kubelet 首次启动时向 kube-apiserver 发送 TLS Bootstrapping 请求，kube-apiserver 验证 kubelet 请求中的 token 是否与它配置的 token 一致，如果一致则自动为 kubelet生成证书和秘钥。

# 生成 token

[root@k8s-1 ssl]# head -c 16 /dev/urandom | od -An -t x | tr -d ' '

22a762c6fd1e636c3b1c7248980e4b93

# 创建 encryption-config.yaml 配置

cat > encryption-config.yaml <<EOF

kind: EncryptionConfig

apiVersion: v1

resources:

- resources:

- secrets

providers:

- aescbc:

keys:

- name: key1

secret: 40179b02a8f6da07d90392ae966f7749

- identity: {}

EOF

# 拷贝

cp encryption-config.yaml /etc/kubernetes/

scp encryption-config.yaml 172.16.10.22:/etc/kubernetes/

# 生成高级审核配置文件

> 官方说明 https://kubernetes.io/docs/tasks/debug-application-cluster/audit/

>

> 如下为最低限度的日志审核

cd /etc/kubernetes

cat >> audit-policy.yaml <<EOF

# Log all requests at the Metadata level.

apiVersion: audit.k8s.io/v1beta1

kind: Policy

rules:

- level: Metadata

EOF

# 拷贝

scp audit-policy.yaml 172.16.10.22:/etc/kubernetes/

**创建 kube-apiserver.service 文件**

# 自定义 系统 service 文件一般存于 /etc/systemd/system/ 下

# 配置为 各自的本地 IP

vi /etc/systemd/system/kube-apiserver.service

[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,DefaultStorageClass,ResourceQuota,NodeRestriction \

--anonymous-auth=false \

--experimental-encryption-provider-config=/etc/kubernetes/encryption-config.yaml \

--advertise-address=172.16.10.21 \

--allow-privileged=true \

--apiserver-count=3 \

--audit-policy-file=/etc/kubernetes/audit-policy.yaml \

--audit-log-maxage=30 \

--audit-log-maxbackup=3 \

--audit-log-maxsize=100 \

--audit-log-path=/var/log/kubernetes/audit.log \

--authorization-mode=Node,RBAC \

--bind-address=0.0.0.0 \

--secure-port=6443 \

--client-ca-file=/etc/kubernetes/ssl/ca.pem \

--kubelet-client-certificate=/etc/kubernetes/ssl/kubernetes.pem \

--kubelet-client-key=/etc/kubernetes/ssl/kubernetes-key.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://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379 \

--event-ttl=1h \

--kubelet-https=true \

--insecure-bind-address=127.0.0.1 \

--insecure-port=8080 \

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

--service-cluster-ip-range=10.254.0.0/18 \

--service-node-port-range=30000-32000 \

--tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem \

--tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \

--enable-bootstrap-token-auth \

--v=1

Restart=on-failure

RestartSec=5

Type=notify

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

# --experimental-encryption-provider-config ，替代之前 token.csv 文件

# 这里面要注意的是 --service-node-port-range=30000-32000

# 这个地方是 映射外部端口时 的端口范围，随机映射也在这个范围内映射，指定映射端口必须也在这个范围内。

**启动 kube-apiserver**

systemctl daemon-reload

systemctl enable kube-apiserver

systemctl start kube-apiserver

systemctl status kube-apiserver

# 如果报错 请使用

journalctl -f -t kube-apiserver 和 journalctl -u kube-apiserver 来定位问题

**配置 kube-controller-manager**

新增几个配置，用于自动 续期证书 –feature-gates=RotateKubeletServerCertificate=true

–experimental-cluster-signing-duration=86700h0m0s

# 创建 kube-controller-manager.service 文件

vi /etc/systemd/system/kube-controller-manager.service

[Unit]

Description=Kubernetes Controller Manager

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]

ExecStart=/usr/local/bin/kube-controller-manager \

--address=0.0.0.0 \

--master=http://127.0.0.1:8080 \

--allocate-node-cidrs=true \

--service-cluster-ip-range=10.254.0.0/18 \

--cluster-cidr=10.254.64.0/18 \

--cluster-signing-cert-file=/etc/kubernetes/ssl/ca.pem \

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

--feature-gates=RotateKubeletServerCertificate=true \

--controllers=\*,tokencleaner,bootstrapsigner \

--experimental-cluster-signing-duration=86700h0m0s \

--cluster-name=kubernetes \

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

--root-ca-file=/etc/kubernetes/ssl/ca.pem \

--leader-elect=true \

--node-monitor-grace-period=40s \

--node-monitor-period=5s \

--pod-eviction-timeout=5m0s \

--v=2

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

**启动 kube-controller-manager**

systemctl daemon-reload

systemctl enable kube-controller-manager

systemctl start kube-controller-manager

systemctl status kube-controller-manager

# 如果报错 请使用

journalctl -f -t kube-controller-manager 和 journalctl -u kube-controller-manager 来定位问题

**配置 kube-scheduler**

# 创建 kube-cheduler.service 文件

vi /etc/systemd/system/kube-scheduler.service

[Unit]

Description=Kubernetes Scheduler

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]

ExecStart=/usr/local/bin/kube-scheduler \

--address=0.0.0.0 \

--master=http://127.0.0.1:8080 \

--leader-elect=true \

--v=1

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

**启动 kube-scheduler**

systemctl daemon-reload

systemctl enable kube-scheduler

systemctl start kube-scheduler

systemctl status kube-scheduler

**验证 Master 节点**

[root@k8s-1 ~]# kubectl get componentstatuses

NAME STATUS MESSAGE ERROR

controller-manager Healthy ok

scheduler Healthy ok

etcd-2 Healthy {"health": "true"}

etcd-0 Healthy {"health": "true"}

etcd-1 Healthy {"health": "true"}

[root@k8s-2 ~]# kubectl get componentstatuses

NAME STATUS MESSAGE ERROR

controller-manager Healthy ok

scheduler Healthy ok

etcd-2 Healthy {"health": "true"}

etcd-0 Healthy {"health": "true"}

etcd-1 Healthy {"health": "true"}

**配置 kubelet 认证**

kubelet 授权 kube-apiserver 的一些操作 exec run logs 等

# RBAC 只需创建一次就可以

kubectl create clusterrolebinding kube-apiserver:kubelet-apis --clusterrole=system:kubelet-api-admin --user kubernetes

**创建 bootstrap kubeconfig 文件**

注意: token 生效时间为 1day , 超过时间未创建自动失效，需要重新创建 token

# 创建 集群所有 kubelet 的 token

[root@k8s-1 kubernetes]# kubeadm token create --description kubelet-bootstrap-token --groups system:bootstrappers:k8s-1 --kubeconfig ~/.kube/config

I1009 10:39:16.623409 3117 version.go:89] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable-1.txt": Get https://storage.googleapis.com/kubernetes-release/release/stable-1.txt: dial tcp 172.217.25.16:443: connect: connection timed out

I1009 10:39:16.623486 3117 version.go:94] falling back to the local client version: v1.12.1

ado3mb.00vde0vkgvfbpz30

[root@k8s-1 kubernetes]# kubeadm token create --description kubelet-bootstrap-token --groups system:bootstrappers:k8s-2 --kubeconfig ~/.kube/config

I1009 10:40:14.199418 3126 version.go:89] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable-1.txt": Get https://storage.googleapis.com/kubernetes-release/release/stable-1.txt: dial tcp 172.217.25.16:443: connect: connection timed out

I1009 10:40:14.199487 3126 version.go:94] falling back to the local client version: v1.12.1

6xkesn.bmym9293ty2r1umr

[root@k8s-1 kubernetes]# kubeadm token create --description kubelet-bootstrap-token --groups system:bootstrappers:k8s-3 --kubeconfig ~/.kube/config

I1009 10:40:42.919424 3136 version.go:89] could not fetch a Kubernetes version from the internet: unable to get URL "https://dl.k8s.io/release/stable-1.txt": Get https://storage.googleapis.com/kubernetes-release/release/stable-1.txt: dial tcp 172.217.25.16:443: connect: connection timed out

I1009 10:40:42.919501 3136 version.go:94] falling back to the local client version: v1.12.1

6jj682.j7wgboa50f6agith

# 查看生成的 token

[root@k8s-1 kubernetes]# kubeadm token list --kubeconfig ~/.kube/config

TOKEN TTL EXPIRES USAGES DESCRIPTION EXTRA GROUPS

6jj682.j7wgboa50f6agith 23h 2018-10-10T10:40:42+08:00 authentication,signing kubelet-bootstrap-token system:bootstrappers:k8s-1

6xkesn.bmym9293ty2r1umr 23h 2018-10-10T10:40:14+08:00 authentication,signing kubelet-bootstrap-token system:bootstrappers:k8s-2

ado3mb.00vde0vkgvfbpz30 23h 2018-10-10T10:39:16+08:00 authentication,signing kubelet-bootstrap-token system:bootstrappers:k8s-3

以下为了区分 会先生成 node 名称加 bootstrap.kubeconfig

生成 k8s-1

# 生成 64 的 bootstrap.kubeconfig

# 配置集群参数

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=https://127.0.0.1:6443 \

--kubeconfig=k8s-1-bootstrap.kubeconfig

# 配置客户端认证

kubectl config set-credentials kubelet-bootstrap \

--token=6jj682.j7wgboa50f6agith \

--kubeconfig=k8s-1-bootstrap.kubeconfig

# 配置关联

kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=k8s-1-bootstrap.kubeconfig

# 配置默认关联

kubectl config use-context default --kubeconfig=k8s-1-bootstrap.kubeconfig

# 拷贝生成的 k8s-1-bootstrap.kubeconfig 文件

mv k8s-1-bootstrap.kubeconfig /etc/kubernetes/bootstrap.kubeconfig

生成 k8s-2

# 生成 65 的 bootstrap.kubeconfig

# 配置集群参数

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=https://127.0.0.1:6443 \

--kubeconfig=k8s-2-bootstrap.kubeconfig

# 配置客户端认证

kubectl config set-credentials kubelet-bootstrap \

--token=1ua4d4.9bluufy3esw4lch6 \

--kubeconfig=k8s-2-bootstrap.kubeconfig

# 配置关联

kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=k8s-2-bootstrap.kubeconfig

# 配置默认关联

kubectl config use-context default --kubeconfig=k8s-2-bootstrap.kubeconfig

# 拷贝生成的 k8s-2-bootstrap.kubeconfig 文件

scp k8s-2-bootstrap.kubeconfig 172.16.10.22:/etc/kubernetes/bootstrap.kubeconfig

生成 k8s-3

# 生成 66 的 bootstrap.kubeconfig

# 配置集群参数

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=https://127.0.0.1:6443 \

--kubeconfig=k8s-3-bootstrap.kubeconfig

# 配置客户端认证

kubectl config set-credentials kubelet-bootstrap \

--token=r8llj2.itme3y54ok531ops \

--kubeconfig=k8s-3-bootstrap.kubeconfig

# 配置关联

kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=k8s-3-bootstrap.kubeconfig

# 配置默认关联

kubectl config use-context default --kubeconfig=k8s-3-bootstrap.kubeconfig

# 拷贝生成的 k8s-3-bootstrap.kubeconfig 文件

scp k8s-3-bootstrap.kubeconfig 172.16.10.23:/etc/kubernetes/bootstrap.kubeconfig

# 配置 bootstrap RBAC 权限

kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --group=system:bootstrappers

# 否则报如下错误

failed to run Kubelet: cannot create certificate signing request: certificatesigningrequests.certificates.k8s.io is forbidden: User "system:bootstrap:1jezb7" cannot create certificatesigningrequests.certificates.k8s.io at the cluster scope

**创建自动批准相关 CSR 请求的 ClusterRole**

vi /etc/kubernetes/tls-instructs-csr.yaml

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: system:certificates.k8s.io:certificatesigningrequests:selfnodeserver

rules:

- apiGroups: ["certificates.k8s.io"]

resources: ["certificatesigningrequests/selfnodeserver"]

verbs: ["create"]

# 导入 yaml 文件

[root@k8s-1 opt]# kubectl apply -f /etc/kubernetes/tls-instructs-csr.yaml

clusterrole.rbac.authorization.k8s.io "system:certificates.k8s.io:certificatesigningrequests:selfnodeserver" created

# 查看

[root@k8s-1 opt]# kubectl describe ClusterRole/system:certificates.k8s.io:certificatesigningrequests:selfnodeserver

Name: system:certificates.k8s.io:certificatesigningrequests:selfnodeserver

Labels: <none>

Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"rbac.authorization.k8s.io/v1","kind":"ClusterRole","metadata":{"annotations":{},"name":"system:certificates.k8s.io:certificatesigningreq...

PolicyRule:

Resources Non-Resource URLs Resource Names Verbs

--------- ----------------- -------------- -----

certificatesigningrequests.certificates.k8s.io/selfnodeserver [] [] [create]

# 将 ClusterRole 绑定到适当的用户组

# 自动批准 system:bootstrappers 组用户 TLS bootstrapping 首次申请证书的 CSR 请求

kubectl create clusterrolebinding node-client-auto-approve-csr --clusterrole=system:certificates.k8s.io:certificatesigningrequests:nodeclient --group=system:bootstrappers

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

kubectl create clusterrolebinding node-client-auto-renew-crt --clusterrole=system:certificates.k8s.io:certificatesigningrequests:selfnodeclient --group=system:nodes

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

kubectl create clusterrolebinding node-server-auto-renew-crt --clusterrole=system:certificates.k8s.io:certificatesigningrequests:selfnodeserver --group=system:nodes

**创建 kubelet.service 文件**

关于 kubectl get node 中的 ROLES 的标签

单 Master 打标签 kubectl label node k8s-1 node-role.kubernetes.io/master=””

这里需要将 单Master 更改为 NoSchedule

更新标签命令为 kubectl taint nodes k8s-1 node-role.kubernetes.io/master=:NoSchedule

既 Master 又是 node 打标签 kubectl label node k8s-2 node-role.kubernetes.io/master=””

单 Node 打标签 kubectl label node k8s-3 node-role.kubernetes.io/node=””

关于删除 label 可使用 - 号相连 如: kubectl label nodes k8s-2 node-role.kubernetes.io/node-

**动态 kubelet 配置**

官方说明 https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

https://github.com/kubernetes/kubernetes/blob/release-1.12/pkg/kubelet/apis/config/types.go

# 创建 kubelet 目录

mkdir -p /var/lib/kubelet

vi /etc/systemd/system/kubelet.service

[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 \

--hostname-override=k8s-1 \

--pod-infra-container-image=jicki/pause-amd64:3.1 \

--bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \

--kubeconfig=/etc/kubernetes/kubelet.kubeconfig \

--config=/etc/kubernetes/kubelet.config.json \

--cert-dir=/etc/kubernetes/ssl \

--logtostderr=true \

--v=2

[Install]

WantedBy=multi-user.target

# 创建 kubelet config 配置文件

vi /etc/kubernetes/kubelet.config.json

{

"kind": "KubeletConfiguration",

"apiVersion": "kubelet.config.k8s.io/v1beta1",

"authentication": {

"x509": {

"clientCAFile": "/etc/kubernetes/ssl/ca.pem"

},

"webhook": {

"enabled": true,

"cacheTTL": "2m0s"

},

"anonymous": {

"enabled": false

}

},

"authorization": {

"mode": "Webhook",

"webhook": {

"cacheAuthorizedTTL": "5m0s",

"cacheUnauthorizedTTL": "30s"

}

},

"address": "172.16.10.21",

"port": 10250,

"readOnlyPort": 0,

"cgroupDriver": "cgroupfs",

"hairpinMode": "promiscuous-bridge",

"serializeImagePulls": false,

"RotateCertificates": true,

"featureGates": {

"RotateKubeletClientCertificate": true,

"RotateKubeletServerCertificate": true

},

"MaxPods": "512",

"failSwapOn": false,

"containerLogMaxSize": "10Mi",

"containerLogMaxFiles": 5,

"clusterDomain": "cluster.local.",

"clusterDNS": ["10.254.0.2"]

}

# 如上配置:

k8s-1 本机hostname

10.254.0.2 预分配的 dns 地址

cluster.local. 为 kubernetes 集群的 domain

jicki/pause-amd64:3.1 这个是 pod 的基础镜像，既 gcr 的 gcr.io/google\_containers/pause-amd64:3.1 镜像， 下载下来修改为自己的仓库中的比较快。

"clusterDNS": ["10.254.0.2"] 可配置多个 dns地址，逗号可开, 可配置宿主机dns.

**启动 kubelet**

systemctl daemon-reload

systemctl enable kubelet

systemctl start kubelet

systemctl status kubelet

# 如果报错 请使用

journalctl -f -t kubelet 和 journalctl -u kubelet 来定位问题

**验证 nodes**

[root@k8s-1 ~]# kubectl get nodes

NAME STATUS ROLES AGE VERSION

k8s-1 Ready master 17h v1.12.1

**查看 kubelet 生成文件**

[root@k8s-1 ~]# ls -lt /etc/kubernetes/ssl/kubelet-\*

-rw------- 1 root root 1374 4月 23 11:55 /etc/kubernetes/ssl/kubelet-server-2018-04-23-11-55-38.pem

lrwxrwxrwx 1 root root 58 4月 23 11:55 /etc/kubernetes/ssl/kubelet-server-current.pem -> /etc/kubernetes/ssl/kubelet-server-2018-04-23-11-55-38.pem

-rw-r--r-- 1 root root 1050 4月 23 11:55 /etc/kubernetes/ssl/kubelet-client.crt

-rw------- 1 root root 227 4月 23 11:55 /etc/kubernetes/ssl/kubelet-client.key

**配置 kube-proxy**

**创建 kube-proxy 证书**

# 证书方面由于我们node端没有装 cfssl

# 我们回到 master 端 机器 去配置证书，然后拷贝过来

[root@k8s-1 ~]# cd /opt/ssl

vi kube-proxy-csr.json

{

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

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "ShenZhen",

"L": "ShenZhen",

"O": "k8s",

"OU": "System"

}

]

}

**生成 kube-proxy 证书和私钥**

/opt/local/cfssl/cfssl gencert -ca=/etc/kubernetes/ssl/ca.pem \

-ca-key=/etc/kubernetes/ssl/ca-key.pem \

-config=/opt/ssl/config.json \

-profile=kubernetes kube-proxy-csr.json | /opt/local/cfssl/cfssljson -bare kube-proxy

# 查看生成

ls kube-proxy\*

kube-proxy.csr kube-proxy-csr.json kube-proxy-key.pem kube-proxy.pem

# 拷贝到目录

cp kube-proxy\* /etc/kubernetes/ssl/

scp kube-proxy\* 172.16.10.22:/etc/kubernetes/ssl/

scp kube-proxy\* 172.16.10.23:/etc/kubernetes/ssl/

**创建 kube-proxy kubeconfig 文件**

# 配置集群

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=https://127.0.0.1:6443 \

--kubeconfig=kube-proxy.kubeconfig

# 配置客户端认证

kubectl config set-credentials kube-proxy \

--client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \

--client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \

--embed-certs=true \

--kubeconfig=kube-proxy.kubeconfig

# 配置关联

kubectl config set-context default \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=kube-proxy.kubeconfig

# 配置默认关联

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

# 拷贝到需要的 node 端里

scp kube-proxy.kubeconfig 172.16.10.22:/etc/kubernetes/

scp kube-proxy.kubeconfig 172.16.10.23:/etc/kubernetes/

**创建 kube-proxy.service 文件**

1.10 官方 ipvs 已经是默认的配置 –masquerade-all 必须添加这项配置，否则 创建 svc 在 ipvs 不会添加规则

打开 ipvs 需要安装 ipvsadm ipset conntrack 软件， 在 node 中安装 yum install ipset ipvsadm conntrack-tools.x86\_64 -y

yaml 配置文件中的 参数如下:

https://github.com/kubernetes/kubernetes/blob/master/pkg/proxy/apis/config/types.go

cd /etc/kubernetes/

vi kube-proxy.config.yaml

apiVersion: kubeproxy.config.k8s.io/v1alpha1

bindAddress: 172.16.10.23

clientConnection:

kubeconfig: /etc/kubernetes/kube-proxy.kubeconfig

clusterCIDR: 10.254.64.0/18

healthzBindAddress: 172.16.10.23:10256

hostnameOverride: k8s-3

kind: KubeProxyConfiguration

metricsBindAddress: 172.16.10.23:10249

mode: "ipvs"

# 创建 kube-proxy 目录

mkdir -p /var/lib/kube-proxy

vi /etc/systemd/system/kube-proxy.service

[Unit]

Description=Kubernetes Kube-Proxy Server

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

ExecStart=/usr/local/bin/kube-proxy \

--config=/etc/kubernetes/kube-proxy.config.yaml \

--logtostderr=true \

--v=1

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

**启动 kube-proxy**

systemctl daemon-reload

systemctl enable kube-proxy

systemctl start kube-proxy

systemctl status kube-proxy

# 如果报错 请使用

journalctl -f -t kube-proxy 和 journalctl -u kube-proxy 来定位问题

# 检查 ipvs

[root@k8s-2 ~]# ipvsadm -L -n

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 10.254.0.1:443 rr persistent 10800

-> 172.16.10.21:6443 Masq 1 0 0

-> 172.16.10.22:6443 Masq 1 0 0

# 如果报错 请使用

journalctl -f -t kube-proxy 和 journalctl -u kube-proxy 来定位问题

**至此 Master 端 与 Master and Node 端的安装完毕**

**Node 端**

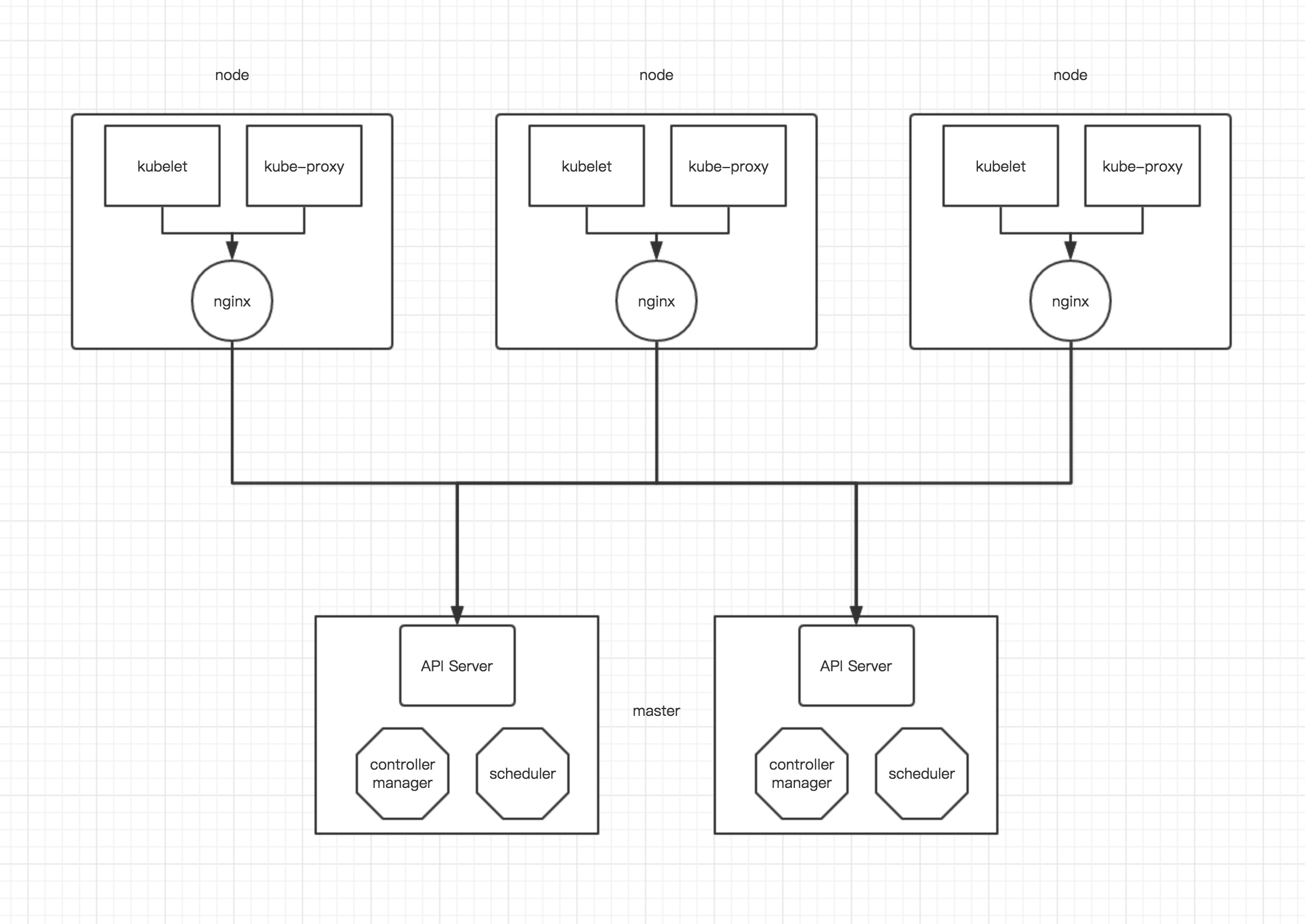
单 Node 部分 需要部署的组件有 docker calico kubelet kube-proxy 这几个组件。 Node 节点 基于 Nginx 负载 API 做 Master HA

# master 之间除 api server 以外其他组件通过 etcd 选举，api server 默认不作处理；

在每个 node 上启动一个 nginx，每个 nginx 反向代理所有 api server;

node 上 kubelet、kube-proxy 连接本地的 nginx 代理端口;

当 nginx 发现无法连接后端时会自动踢掉出问题的 api server，从而实现 api server 的 HA;



**发布证书**

# ALL node

mkdir -p /etc/kubernetes/ssl/

scp ca.pem kube-proxy.pem kube-proxy-key.pem node-\*:/etc/kubernetes/ssl/

**创建Nginx 代理**

在每个 node 都必须创建一个 Nginx 代理， 这里特别注意， 当 Master 也做为 Node 的时候 不需要配置 Nginx-proxy

# 创建配置目录

mkdir -p /etc/nginx

# 写入代理配置

cat << EOF >> /etc/nginx/nginx.conf

error\_log stderr notice;

worker\_processes auto;

events {

multi\_accept on;

use epoll;

worker\_connections 1024;

}

stream {

upstream kube\_apiserver {

least\_conn;

server 172.16.10.21:6443;

server 172.16.10.22:6443;

}

server {

listen 0.0.0.0:6443;

proxy\_pass kube\_apiserver;

proxy\_timeout 10m;

proxy\_connect\_timeout 1s;

}

}

EOF

# 更新权限

chmod +r /etc/nginx/nginx.conf

# 配置 Nginx 基于 docker 进程，然后配置 systemd 来启动

cat << EOF >> /etc/systemd/system/nginx-proxy.service

[Unit]

Description=kubernetes apiserver docker wrapper

Wants=docker.socket

After=docker.service

[Service]

User=root

PermissionsStartOnly=true

ExecStart=/usr/bin/docker run -p 127.0.0.1:6443:6443 \\

-v /etc/nginx:/etc/nginx \\

--name nginx-proxy \\

--net=host \\

--restart=on-failure:5 \\

--memory=512M \\

nginx:1.13.7-alpine

ExecStartPre=-/usr/bin/docker rm -f nginx-proxy

ExecStop=/usr/bin/docker stop nginx-proxy

Restart=always

RestartSec=15s

TimeoutStartSec=30s

[Install]

WantedBy=multi-user.target

EOF

# 启动 Nginx

systemctl daemon-reload

systemctl start nginx-proxy

systemctl enable nginx-proxy

systemctl status nginx-proxy

**配置 Kubelet.service 文件**

**systemd kubelet 配置**

**动态 kubelet 配置**

官方说明 https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ https://kubernetes.io/docs/tasks/administer-cluster/reconfigure-kubelet/

https://github.com/kubernetes/kubernetes/blob/release-1.12/pkg/kubelet/apis/config/types.go

# 创建 kubelet 目录

vi /etc/systemd/system/kubelet.service

[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 \

--hostname-override=k8s-1 \

--pod-infra-container-image=jicki/pause-amd64:3.1 \

--bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig \

--kubeconfig=/etc/kubernetes/kubelet.kubeconfig \

--config=/etc/kubernetes/kubelet.config.json \

--cert-dir=/etc/kubernetes/ssl \

--logtostderr=true \

--v=2

[Install]

WantedBy=multi-user.target

# 创建 kubelet config 配置文件

vi /etc/kubernetes/kubelet.config.json

{

"kind": "KubeletConfiguration",

"apiVersion": "kubelet.config.k8s.io/v1beta1",

"authentication": {

"x509": {

"clientCAFile": "/etc/kubernetes/ssl/ca.pem"

},

"webhook": {

"enabled": true,

"cacheTTL": "2m0s"

},

"anonymous": {

"enabled": false

}

},

"authorization": {

"mode": "Webhook",

"webhook": {

"cacheAuthorizedTTL": "5m0s",

"cacheUnauthorizedTTL": "30s"

}

},

"address": "172.16.10.23",

"port": 10250,

"readOnlyPort": 0,

"cgroupDriver": "cgroupfs",

"hairpinMode": "promiscuous-bridge",

"serializeImagePulls": false,

"featureGates": {

"RotateKubeletClientCertificate": true,

"RotateKubeletServerCertificate": true

},

"MaxPods": "512",

"failSwapOn": false,

"containerLogMaxSize": "10Mi",

"containerLogMaxFiles": 5,

"clusterDomain": "cluster.local.",

"clusterDNS": ["10.254.0.2"]

}

# 启动 kubelet

systemctl daemon-reload

systemctl enable kubelet

systemctl start kubelet

systemctl status kubelet

**配置 kube-proxy.service**

cd /etc/kubernetes/

vi kube-proxy.config.yaml

apiVersion: kubeproxy.config.k8s.io/v1alpha1

bindAddress: 172.16.10.23

clientConnection:

kubeconfig: /etc/kubernetes/kube-proxy.kubeconfig

clusterCIDR: 10.254.64.0/18

healthzBindAddress: 172.16.10.23:10256

hostnameOverride: k8s-3

kind: KubeProxyConfiguration

metricsBindAddress: 172.16.10.23:10249

mode: "ipvs"

# 创建 kube-proxy 目录

vi /etc/systemd/system/kube-proxy.service

[Unit]

Description=Kubernetes Kube-Proxy Server

Documentation=https://github.com/GoogleCloudPlatform/kubernetes

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

ExecStart=/usr/local/bin/kube-proxy \

--config=/etc/kubernetes/kube-proxy.config.yaml \

--logtostderr=true \

--v=1

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

# 启动

systemctl daemon-reload

systemctl enable kube-proxy

systemctl start kube-proxy

systemctl status kube-proxy

**配置 Flannel 网络**

公有云如 阿里云 华为云 可能无法使用 flannel 的 host-gw 模式，请使用 vxlan 或 calico 网络

flannel 网络只部署在 kube-proxy 相关机器

个人 百度盘 下载 https://pan.baidu.com/s/1\_A3zzurG5vV40-FnyA8uWg

rpm -ivh flannel-0.10.0-1.x86\_64.rpm

# 配置 flannel

# 由于我们docker更改了 docker.service.d 的路径

# 所以这里把 flannel.conf 的配置拷贝到 这个目录去

mv /usr/lib/systemd/system/docker.service.d/flannel.conf /etc/systemd/system/docker.service.d

# 配置 flannel 网段

etcdctl --endpoints=https://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379\

--cert-file=/etc/kubernetes/ssl/etcd.pem \

--ca-file=/etc/kubernetes/ssl/ca.pem \

--key-file=/etc/kubernetes/ssl/etcd-key.pem \

set /flannel/network/config \ '{"Network":"10.254.64.0/18","SubnetLen":24,"Backend":{"Type":"host-gw"}}'

# 修改 flanneld 配置

vi /etc/sysconfig/flanneld

# Flanneld configuration options

# etcd 地址

FLANNEL\_ETCD\_ENDPOINTS="https://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379"

# 配置为上面的路径 flannel/network

FLANNEL\_ETCD\_PREFIX="/flannel/network"

# 其他的配置，可查看 flanneld --help,这里添加了 etcd ssl 认证

FLANNEL\_OPTIONS="-ip-masq=true -etcd-cafile=/etc/kubernetes/ssl/ca.pem -etcd-certfile=/etc/kubernetes/ssl/etcd.pem -etcd-keyfile=/etc/kubernetes/ssl/etcd-key.pem -iface=em1"

# 启动 flannel

systemctl daemon-reload

systemctl enable flanneld

systemctl start flanneld

systemctl status flanneld

# 如果报错 请使用

journalctl -f -t flanneld 和 journalctl -u flanneld 来定位问题

# 配置完毕，重启 docker

systemctl daemon-reload

systemctl enable docker

systemctl restart docker

systemctl status docker

# 重启 kubelet

systemctl daemon-reload

systemctl restart kubelet

systemctl status kubelet

# 验证 网络

ifconfig 查看 docker0 网络 是否已经更改为配置IP网段

**配置 Calico 网络**

官方文档 https://docs.projectcalico.org/v3.2/introduction

**下载 Calico yaml**

# 下载 yaml 文件

wget http://docs.projectcalico.org/v3.2/getting-started/kubernetes/installation/hosted/calico.yaml

wget http://docs.projectcalico.org/v3.2/getting-started/kubernetes/installation/rbac.yaml

**下载镜像**

# 下载 镜像

# 国外镜像 有墙

quay.io/calico/node:v3.2.3

quay.io/calico/cni:v3.2.3

quay.io/calico/kube-controllers:v3.2.3

# 国内镜像

jicki/node:v3.1.3

jicki/cni:v3.1.3

jicki/kube-controllers:v3.1.3

# 替换镜像

sed -i 's/quay\.io\/calico/jicki/g' calico.yaml

**修改配置**

vi calico.yaml

# 注意修改如下选项:

# etcd 地址

etcd\_endpoints: "https://172.16.10.21:2379,https://172.16.10.22:2379,https://172.16.10.23:2379"

# etcd 证书路径

# If you're using TLS enabled etcd uncomment the following.

# You must also populate the Secret below with these files.

etcd\_ca: "/calico-secrets/etcd-ca"

etcd\_cert: "/calico-secrets/etcd-cert"

etcd\_key: "/calico-secrets/etcd-key"

# etcd 证书 base64 地址 (执行里面的命令生成的证书 base64 码，填入里面)

data:

etcd-key: (cat /etc/kubernetes/ssl/etcd-key.pem | base64 | tr -d '\n')

etcd-cert: (cat /etc/kubernetes/ssl/etcd.pem | base64 | tr -d '\n')

etcd-ca: (cat /etc/kubernetes/ssl/ca.pem | base64 | tr -d '\n')

# 修改 pods 分配的 IP 段

- name: CALICO\_IPV4POOL\_CIDR

value: "10.254.64.0/18"

# 导入 yaml 文件

[root@k8s-1 ~]# kubectl apply -f .

configmap/calico-config created

secret/calico-etcd-secrets created

daemonset.extensions/calico-node created

deployment.extensions/calico-kube-controllers created

serviceaccount/calico-kube-controllers created

serviceaccount/calico-node created

clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created

clusterrole.rbac.authorization.k8s.io/calico-node created

clusterrolebinding.rbac.authorization.k8s.io/calico-node created

# 查看服务

[root@k8s-1 ~]# kubectl get pods -n kube-system

NAME READY STATUS RESTARTS AGE

calico-kube-controllers-79cfd7887-nbgdv 1/1 Running 0 1m

calico-node-9mkrt 2/2 Running 0 1m

calico-node-dzf4c 2/2 Running 0 1m

calico-node-gdxnn 2/2 Running 0 1m

**修改 kubelet 配置**

# kubelet 需要增加 cni 插件 --network-plugin=cni

vi /etc/systemd/system/kubelet.service

--network-plugin=cni \

# 重新加载配置

systemctl daemon-reload

systemctl restart kubelet.service

systemctl status kubelet.service

**检查网络**

# 查看 node 中网络状况

[root@k8s-1 ~]# ifconfig

tunl0: flags=193<UP,RUNNING,NOARP> mtu 1440

inet 10.254.95.64 netmask 255.255.255.255

tunnel txqueuelen 1 (IPIP Tunnel)

RX packets 2 bytes 168 (168.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 2 bytes 168 (168.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@k8s-2 ~]# ifconfig

tunl0: flags=193<UP,RUNNING,NOARP> mtu 1440

inet 10.254.116.128 netmask 255.255.255.255

tunnel txqueuelen 1 (IPIP Tunnel)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@k8s-3 ~]# ifconfig

tunl0: flags=193<UP,RUNNING,NOARP> mtu 1440

inet 10.254.70.64 netmask 255.255.255.255

tunnel txqueuelen 1 (IPIP Tunnel)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

**安装 calicoctl**

calicoctl 是 calico 网络的管理客户端, 只需要在一台 node 里配置既可。

# 下载 二进制文件

curl -O -L https://github.com/projectcalico/calicoctl/releases/download/v3.2.3/calicoctl

mv calicoctl /usr/local/bin/

chmod +x /usr/local/bin/calicoctl

# 创建 calicoctl.cfg 配置文件

mkdir /etc/calico

vi /etc/calico/calicoctl.cfg

apiVersion: projectcalico.org/v3

kind: CalicoAPIConfig

metadata:

spec:

datastoreType: "kubernetes"

kubeconfig: "/root/.kube/config"

# 查看 calico 状态

[root@k8s-1 ~]# calicoctl node status

Calico process is running.

IPv4 BGP status

+----------------+-------------------+-------+----------+-------------+

| PEER ADDRESS | PEER TYPE | STATE | SINCE | INFO |

+----------------+-------------------+-------+----------+-------------+

| 172.16.10.22 | node-to-node mesh | up | 01:10:35 | Established |

| 172.16.10.23 | node-to-node mesh | up | 01:10:35 | Established |

+----------------+-------------------+-------+----------+-------------+

IPv6 BGP status

No IPv6 peers found.

[root@k8s-1 ~]# calicoctl get node

NAME

k8s-1

k8s-2

k8s-3

**测试集群**

# 创建一个 nginx deplyment

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-dm

spec:

replicas: 2

template:

metadata:

labels:

name: nginx

spec:

containers:

- name: nginx

image: nginx:alpine

imagePullPolicy: IfNotPresent

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: nginx-svc

spec:

ports:

- port: 80

targetPort: 80

protocol: TCP

selector:

name: nginx

[root@k8s-1 ~]# kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE

nginx-dm-84f8f49555-dzpm9 1/1 Running 0 6s 10.254.90.2 k8s-2

nginx-dm-84f8f49555-qbnvv 1/1 Running 0 6s 10.254.66.2 k8s-master-66

[root@k8s-1 ~]# kubectl get svc -o wide

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR

kubernetes ClusterIP 10.254.0.1 <none> 443/TCP 2h <none>

nginx-svc ClusterIP 10.254.41.39 <none> 80/TCP 1m

# 在 安装了 网络的节点 里 curl

[root@k8s-1 ~]# curl 10.254.51.137

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

<style>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

}

</style>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully installed and

working. Further configuration is required.</p>

<p>For online documentation and support please refer to

<a href="http://nginx.org/">nginx.org</a>.<br/>

Commercial support is available at

<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>

</body>

</html>

# 查看 ipvs 规则

[root@k8s-2 ~]# ipvsadm -L -n

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 10.254.0.1:443 rr persistent 10800

-> 172.16.10.21:6443 Masq 1 0 0

-> 172.16.10.22:6443 Masq 1 0 0

TCP 10.254.41.39:80 rr

-> 10.254.66.2:80 Masq 1 0 0

-> 10.254.90.2:80 Masq 1 0 1

**配置 CoreDNS**

官方 地址 https://coredns.io

**下载 yaml 文件**

wget https://raw.githubusercontent.com/coredns/deployment/master/kubernetes/coredns.yaml.sed

mv coredns.yaml.sed coredns.yaml

1.2.x 版本中 Corefile 部分更新了点东西，使用如下替换整个 Corefile 部分

# vi coredns.yaml

...

data:

Corefile: |

.:53 {

errors

health

kubernetes cluster.local 10.254.0.0/18 {

pods insecure

upstream

fallthrough in-addr.arpa ip6.arpa

}

prometheus :9153

proxy . /etc/resolv.conf

cache 30

}

...

clusterIP: 10.254.0.2

# 配置说明

# 这里 kubernetes cluster.local 为 创建 svc 的 IP 段

kubernetes cluster.local 10.254.0.0/18

# clusterIP 为 指定 DNS 的 IP

clusterIP: 10.254.0.2

**导入 yaml 文件**

# 导入

[root@k8s-1 coredns]# kubectl apply -f coredns.yaml

serviceaccount/coredns created

clusterrole.rbac.authorization.k8s.io/system:coredns created

clusterrolebinding.rbac.authorization.k8s.io/system:coredns created

configmap/coredns created

deployment.extensions/coredns created

service/kube-dns created

**查看 coredns 服务**

[root@k8s-1 coredns]# kubectl get pod,svc -n kube-system

NAME READY STATUS RESTARTS AGE

pod/coredns-6975654877-nzhgr 1/1 Running 0 23s

pod/coredns-6975654877-qn4bp 1/1 Running 0 23s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kube-dns ClusterIP 10.254.0.2 <none> 53/UDP,53/TCP 23s

**检查日志**

[root@k8s-1 coredns]# kubectl logs -n kube-system pod/coredns-6975654877-nzhgr

.:53

2018/08/09 02:11:11 [INFO] CoreDNS-1.2.0

2018/08/09 02:11:11 [INFO] linux/amd64, go1.10.3, 2e322f6

CoreDNS-1.2.0

linux/amd64, go1.10.3, 2e322f6

2018/08/09 02:11:11 [INFO] plugin/reload: Running configuration MD5 = 271feea1e1cf54e66a65c7ffcf2b89ad

**验证 dns 服务**

在验证 dns 之前，在 dns 未部署之前创建的 pod 与 deployment 等，都必须删除，重新部署，否则无法解析

# 创建一个 pods 来测试一下 dns

apiVersion: v1

kind: Pod

metadata:

name: alpine

spec:

containers:

- name: alpine

image: alpine

command:

- sleep

- "3600"

# 查看 创建的服务

[root@k8s-1 yaml]# kubectl get pods,svc

NAME READY STATUS RESTARTS AGE

po/alpine 1/1 Running 0 19s

po/nginx-dm-84f8f49555-tmqzm 1/1 Running 0 23s

po/nginx-dm-84f8f49555-wdk67 1/1 Running 0 23s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

svc/kubernetes ClusterIP 10.254.0.1 <none> 443/TCP 5h

svc/nginx-svc ClusterIP 10.254.40.179 <none> 80/TCP 23s

# 测试

[root@k8s-1 ~]# kubectl exec -it alpine nslookup nginx-svc

nslookup: can't resolve '(null)': Name does not resolve

Name: nginx-svc

Address 1: 10.254.40.179 nginx-svc.default.svc.cluster.local

[root@k8s-1 yaml]# kubectl exec -it alpine nslookup kubernetes

nslookup: can't resolve '(null)': Name does not resolve

Name: kubernetes

Address 1: 10.254.0.1 kubernetes.default.svc.cluster.local

**部署 DNS 自动伸缩**

按照 node 数量 自动伸缩 dns 数量

vi dns-auto-scaling.yaml

kind: ServiceAccount

apiVersion: v1

metadata:

name: kube-dns-autoscaler

namespace: kube-system

labels:

addonmanager.kubernetes.io/mode: Reconcile

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: system:kube-dns-autoscaler

labels:

addonmanager.kubernetes.io/mode: Reconcile

rules:

- apiGroups: [""]

resources: ["nodes"]

verbs: ["list"]

- apiGroups: [""]

resources: ["replicationcontrollers/scale"]

verbs: ["get", "update"]

- apiGroups: ["extensions"]

resources: ["deployments/scale", "replicasets/scale"]

verbs: ["get", "update"]

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["get", "create"]

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: system:kube-dns-autoscaler

labels:

addonmanager.kubernetes.io/mode: Reconcile

subjects:

- kind: ServiceAccount

name: kube-dns-autoscaler

namespace: kube-system

roleRef:

kind: ClusterRole

name: system:kube-dns-autoscaler

apiGroup: rbac.authorization.k8s.io

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: kube-dns-autoscaler

namespace: kube-system

labels:

k8s-app: kube-dns-autoscaler

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

spec:

selector:

matchLabels:

k8s-app: kube-dns-autoscaler

template:

metadata:

labels:

k8s-app: kube-dns-autoscaler

annotations:

scheduler.alpha.kubernetes.io/critical-pod: ''

spec:

priorityClassName: system-cluster-critical

containers:

- name: autoscaler

image: jicki/cluster-proportional-autoscaler-amd64:1.1.2-r2

resources:

requests:

cpu: "20m"

memory: "10Mi"

command:

- /cluster-proportional-autoscaler

- --namespace=kube-system

- --configmap=kube-dns-autoscaler

- --target=Deployment/coredns

- --default-params={"linear":{"coresPerReplica":256,"nodesPerReplica":16,"preventSinglePointFailure":true}}

- --logtostderr=true

- --v=2

tolerations:

- key: "CriticalAddonsOnly"

operator: "Exists"

serviceAccountName: kube-dns-autoscaler

# 导入文件

[root@k8s-1 coredns]# kubectl apply -f dns-auto-scaling.yaml

serviceaccount/kube-dns-autoscaler created

clusterrole.rbac.authorization.k8s.io/system:kube-dns-autoscaler created

clusterrolebinding.rbac.authorization.k8s.io/system:kube-dns-autoscaler created

deployment.apps/kube-dns-autoscaler created

**部署 Ingress 与 Dashboard**

**部署 heapster**

官方 dashboard 的github https://github.com/kubernetes/dashboard

官方 heapster 的github https://github.com/kubernetes/heapster

**下载 heapster 相关 yaml 文件**

wget https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/grafana.yaml

wget https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/influxdb.yaml

wget https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/influxdb/heapster.yaml

wget https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/rbac/heapster-rbac.yaml

**下载 heapster 镜像下载**

# 官方镜像

k8s.gcr.io/heapster-grafana-amd64:v4.4.3

k8s.gcr.io/heapster-amd64:v1.5.3

k8s.gcr.io/heapster-influxdb-amd64:v1.3.3

# 个人的镜像

jicki/heapster-grafana-amd64:v4.4.3

jicki/heapster-amd64:v1.5.3

jicki/heapster-influxdb-amd64:v1.3.3

# 替换所有yaml 镜像地址

sed -i 's/k8s\.gcr\.io/jicki/g' \*.yaml

**修改 yaml 文件**

# heapster.yaml 文件

#### 修改如下部分 #####

因为 kubelet 启用了 https 所以如下配置需要增加 https 端口

- --source=kubernetes:https://kubernetes.default

修改为

- --source=kubernetes:https://kubernetes.default?kubeletHttps=true&kubeletPort=10250&insecure=true

# heapster-rbac.yaml 文件

#### 修改为部分 #####

将 serviceAccount kube-system:heapster 与 ClusterRole system:kubelet-api-admin 绑定，授予它调用 kubelet API 的权限；

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: heapster

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:heapster

subjects:

- kind: ServiceAccount

name: heapster

namespace: kube-system

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: heapster-kubelet-api

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:kubelet-api-admin

subjects:

- kind: ServiceAccount

name: heapster

namespace: kube-system

# 导入所有的文件

[root@k8s-1 heapster]# kubectl apply -f .

deployment.extensions/monitoring-grafana created

service/monitoring-grafana created

clusterrolebinding.rbac.authorization.k8s.io/heapster created

serviceaccount/heapster created

deployment.extensions/heapster created

service/heapster created

deployment.extensions/monitoring-influxdb created

service/monitoring-influxdb created

# 查看运行

[root@k8s-1 heapster]# kubectl get pods -n kube-system | grep -E 'heapster|monitoring'

heapster-545d9555d4-lm5fs 1/1 Running 0 1m

monitoring-grafana-59b4f6d8b7-ft2gv 1/1 Running 0 1m

monitoring-influxdb-f6bcc9795-9zjnl 1/1 Running 0 1m

**部署 dashboard**

**下载 dashboard 镜像**

# 官方镜像

k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0

# 个人的镜像

jicki/kubernetes-dashboard-amd64:v1.10.0

**下载 yaml 文件**

curl -O https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml

**导入 yaml**

# 替换所有的 images

sed -i 's/k8s\.gcr\.io/jicki/g' \*

# 导入文件

[root@k8s-1 dashboard]# kubectl apply -f kubernetes-dashboard.yaml

secret "kubernetes-dashboard-certs" created

serviceaccount "kubernetes-dashboard" created

role "kubernetes-dashboard-minimal" created

rolebinding "kubernetes-dashboard-minimal" created

deployment "kubernetes-dashboard" created

service "kubernetes-dashboard" created

[root@k8s-1 ~]# kubectl get pods,svc -n kube-system

NAME READY STATUS RESTARTS AGE

po/coredns-5984fb8cbb-77dl4 1/1 Running 0 3h

po/coredns-5984fb8cbb-9hdwt 1/1 Running 0 3h

po/kubernetes-dashboard-78bcdc4d64-x6fhq 1/1 Running 0 14s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

svc/kube-dns ClusterIP 10.254.0.2 <none> 53/UDP,53/TCP 3h

svc/kubernetes-dashboard ClusterIP 10.254.18.143 <none> 443/TCP 14s

**部署 Nginx Ingress**

Kubernetes 暴露服务的方式目前只有三种：LoadBlancer Service、NodePort Service、Ingress； 什么是 Ingress ? Ingress 就是利用 Nginx Haproxy 等负载均衡工具来暴露 Kubernetes 服务。

官方 Nginx Ingress github: https://github.com/kubernetes/ingress-nginx/

**配置 调度 node**

# ingress 有多种方式 1. deployment 自由调度 replicas

2. daemonset 全局调度 分配到所有node里

# deployment 自由调度过程中，由于我们需要 约束 controller 调度到指定的 node 中，所以需要对 node 进行 label 标签

# 默认如下:

[root@k8s-1 ingress]# kubectl get nodes

NAME STATUS ROLES AGE VERSION

k8s-1 Ready master 1d v1.12.1

k8s-2 Ready master 1d v1.12.1

k8s-3 Ready node 1d v1.12.1

# 对 65 与 66 打上 label

[root@k8s-1 ingress]# kubectl label nodes k8s-2 ingress=proxy

node "k8s-2" labeled

[root@k8s-1 ingress]# kubectl label nodes k8s-3 ingress=proxy

node "k8s-3" labeled

# 打完标签以后

[root@k8s-1 ingress]# kubectl get nodes --show-labels

NAME STATUS ROLES AGE VERSION LABELS

k8s-1 Ready,SchedulingDisabled <none> 32m v1.12.1 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/hostname=k8s-1

k8s-2 Ready <none> 17m v1.12.1 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,ingress=proxy,kubernetes.io/hostname=k8s-2

k8s-3 Ready <none> 4m v1.12.1 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,ingress=proxy,kubernetes.io/hostname=k8s-3

# 下载镜像

# 官方镜像

quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.20.0

# 国内镜像

jicki/nginx-ingress-controller:0.20.0

# 下载 yaml 文件

# 部署 Nginx backend , Nginx backend 用于统一转发 没有的域名 到指定页面。

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

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

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

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

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

# 部署 Ingress RBAC 认证

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

# 部署 Ingress Controller 组件

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

# tcp-service 与 udp-service, 由于 ingress 不支持 tcp 与 udp 的转发，所以这里配置了两个基于 tcp 与 udp 的 service ,通过 --tcp-services-configmap 与 --udp-services-configmap 来配置 tcp 与 udp 的转发服务

# tcp 例子

apiVersion: v1

kind: ConfigMap

metadata:

name: tcp-services

namespace: ingress-nginx

data:

9000: "default/tomcat:8080"

# 以上配置， 转发 tomcat:8080 端口 到 ingress 节点的 9000 端口中

# udp 例子

apiVersion: v1

kind: ConfigMap

metadata:

name: udp-services

namespace: ingress-nginx

data:

53: "kube-system/kube-dns:53"

# 替换所有的 images

sed -i 's/gcr\.io\/google\_containers/jicki/g' \*

sed -i 's/quay\.io\/kubernetes-ingress-controller/jicki/g' \*

# 上面 对 两个 node 打了 label 所以配置 replicas: 2

# 修改 yaml 文件 增加 rbac 认证 , hostNetwork 还有 nodeSelector, 第二个 spec 下 增加。

vi with-rbac.yaml

spec:

replicas: 2

....

spec:

serviceAccountName: nginx-ingress-serviceaccount

hostNetwork: true

nodeSelector:

ingress: proxy

....

# 这里添加一个 other 端口做为后续tcp转发

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

- name: other

containerPort: 8888

# 导入 yaml 文件

[root@k8s-1 ingress]# kubectl apply -f namespace.yaml

namespace "ingress-nginx" created

[root@k8s-1 nginx-ingress]# kubectl apply -f .

configmap "nginx-configuration" created

deployment "default-http-backend" created

service "default-http-backend" created

namespace "ingress-nginx" configured

serviceaccount "nginx-ingress-serviceaccount" created

clusterrole "nginx-ingress-clusterrole" created

role "nginx-ingress-role" created

rolebinding "nginx-ingress-role-nisa-binding" created

clusterrolebinding "nginx-ingress-clusterrole-nisa-binding" created

configmap "tcp-services" created

configmap "udp-services" created

deployment "nginx-ingress-controller" created

# 查看服务，可以看到这两个 pods 被分别调度到 65 与 66 中

[root@k8s-1 ingress]# kubectl get pods -n ingress-nginx -o wide

NAME READY STATUS RESTARTS AGE IP NODE

nginx-ingress-controller-8476958f94-8fh5h 1/1 Running 0 5m 172.16.10.23 k8s-3

nginx-ingress-controller-8476958f94-qfhhp 1/1 Running 0 5m 172.16.10.22 k8s-2

# 查看我们原有的 svc

[root@k8s-1 ingress]# kubectl get pods

NAME READY STATUS RESTARTS AGE

alpine 1/1 Running 0 24m

nginx-dm-84f8f49555-tmqzm 1/1 Running 0 24m

nginx-dm-84f8f49555-wdk67 1/1 Running 0 24m

# 创建一个 基于 nginx-dm 的 ingress

vi nginx-ingress.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginx-ingress

spec:

rules:

- host: nginx.jicki.me

http:

paths:

- backend:

serviceName: nginx-svc

servicePort: 80

# 查看服务

[root@k8s-1 ingress]# kubectl get ingress

NAME HOSTS ADDRESS PORTS AGE

nginx-ingress nginx.jicki.me 80 6s

# 测试访问

[root@k8s-1 ingress]# curl nginx.jicki.me

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

<style>

body {

width: 35em;

margin: 0 auto;

font-family: Tahoma, Verdana, Arial, sans-serif;

}

</style>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully installed and

working. Further configuration is required.</p>

<p>For online documentation and support please refer to

<a href="http://nginx.org/">nginx.org</a>.<br/>

Commercial support is available at

<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>

</body>

</html>

# 创建一个基于 dashboard 的 https 的 ingress

# 新版本的 dashboard 默认就是 ssl ,所以这里使用 tcp 代理到 443 端口

# 查看 dashboard svc

[root@k8s-1 dashboard]# kubectl get svc -n kube-system

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kube-dns ClusterIP 10.254.0.2 <none> 53/UDP,53/TCP 4h

kubernetes-dashboard ClusterIP 10.254.18.143 <none> 443/TCP 57m

# 修改 tcp-services-configmap.yaml 文件

vi tcp-services-configmap.yaml

kind: ConfigMap

apiVersion: v1

metadata:

name: tcp-services

namespace: ingress-nginx

data:

8888: "kube-system/kubernetes-dashboard:443"

# 导入文件

[root@k8s-1 dashboard]# kubectl apply -f tcp-services-configmap.yaml

configmap "tcp-services" created

# 查看服务

[root@k8s-1 dashboard]# kubectl get configmap/tcp-services -n ingress-nginx

NAME DATA AGE

tcp-services 1 11m

[root@k8s-1 dashboard]# kubectl describe configmap/tcp-services -n ingress-nginx

Name: tcp-services

Namespace: ingress-nginx

Labels: <none>

Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","data":{"8888":"kube-system/kubernetes-dashboard:443"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"tcp-services","namesp...

Data

====

8888:

----

kube-system/kubernetes-dashboard:443

Events: <none>

# 测试访问

[root@k8s-1 dashboard]# curl -I -k https://dashboard.jicki.me:8888

HTTP/1.1 200 OK

Accept-Ranges: bytes

Cache-Control: no-store

Content-Length: 990

Content-Type: text/html; charset=utf-8

Last-Modified: Mon, 15 Jan 2018 13:10:36 GMT

Date: Tue, 23 Jan 2018 09:12:08 GMT

# 配置一个基于域名的 https , ingress

# 创建一个 基于 自身域名的 证书

openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -keyout dashboard.jicki.me-key.key -out dashboard.jicki.me.pem -subj "/CN=dashboard.jicki.me"

# 导入 域名的证书 到 集群 的 secret 中

kubectl create secret tls dashboard-secret --namespace=kube-system --cert dashboard.jicki.me.pem --key dashboard.jicki.me-key.key

# 查看 secret

[root@k8s-1 dashboard]# kubectl get secret -n kube-system

NAME TYPE DATA AGE

dashboard-secret kubernetes.io/tls 2 1m

# 创建一个 ingress

vi dashboard-ingress.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: kubernetes-dashboard

namespace: kube-system

annotations:

ingress.kubernetes.io/ssl-passthrough: "true"

nginx.ingress.kubernetes.io/secure-backends: "true"

spec:

tls:

- hosts:

- dashboard.jicki.me

secretName: dashboard-secret

rules:

- host: dashboard.jicki.me

http:

paths:

- path: /

backend:

serviceName: kubernetes-dashboard

servicePort: 443

# 测试访问

[root@k8s-1 dashboard]# curl -I -k https://dashboard.jicki.me

HTTP/1.1 200 OK

Server: nginx/1.13.12

Date: Wed, 11 Jul 2018 07:19:03 GMT

Content-Type: text/html; charset=utf-8

Content-Length: 990

Connection: keep-alive

Vary: Accept-Encoding

Accept-Ranges: bytes

Cache-Control: no-store

Last-Modified: Tue, 13 Feb 2018 11:17:03 GMT

Strict-Transport-Security: max-age=15724800; includeSubDomains

# 登录认证

# 首先创建一个 dashboard rbac 超级用户

vi dashboard-admin-rbac.yaml

---

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-admin

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: kubernetes-dashboard-admin

labels:

k8s-app: kubernetes-dashboard

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard-admin

namespace: kube-system

# 导入文件

[root@k8s-1 dashboard]# kubectl apply -f dashboard-admin-rbac.yaml

serviceaccount "kubernetes-dashboard-admin" created

clusterrolebinding "kubernetes-dashboard-admin" created

# 查看超级用户的 token 名称

[root@k8s-1 dashboard]# kubectl -n kube-system get secret | grep kubernetes-dashboard-admin

kubernetes-dashboard-admin-token-mnhdz kubernetes.io/service-account-token 3 1m

# 查看 token 部分

[root@k8s-1 dashboard]# kubectl describe -n kube-system secret/kubernetes-dashboard-admin-token-mnhdz

Name: kubernetes-dashboard-admin-token-mnhdz

Namespace: kube-system

Labels: <none>

Annotations: kubernetes.io/service-account.name=kubernetes-dashboard-admin

kubernetes.io/service-account.uid=dc14511d-0020-11e8-b47b-44a8420b9988

Type: kubernetes.io/service-account-token

Data

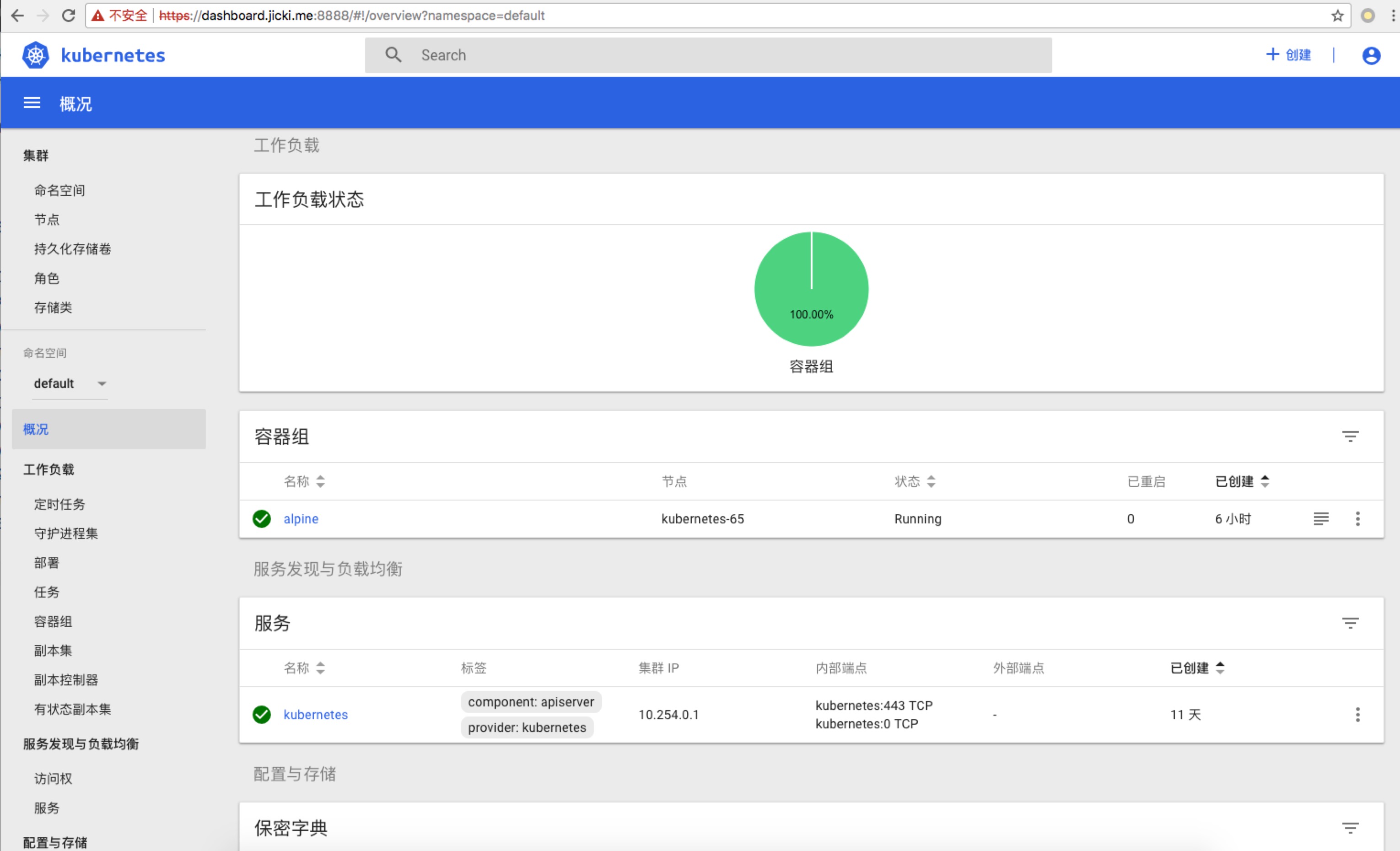
====

ca.crt: 1363 bytes

namespace: 11 bytes

token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9..Vg7vYBIaBICYFCX\_XORvoUjkYAKdQoAuT2sy8o4y8Z6DmMaCQXijOBGCWsS40-n\_qiBhlrSwLeN0RvjCOfLmcH4gUSjPBkSmc-S6SHh09ErzrHjCQSblCCZgXjyyse2w1LwWw87CiAiwHCb0Jm7r0lhm4DjhXeLpUhdXoqOltHlBoJqxzDwb9qKgtY-nsQ2Y9dhV405GeqB9RLOxSKHWx6K1lXP\_0tLUGgIatJx6f-EMurFbmODJfex9mT2LTq9pblblegw9EG9j2IhfHQSnwR8hPMT3Tku-XEf3vtV-1eFqetZHRJHS23machhvSvuppFjmPAd\_ID3eETBt7ncNmQ

# 登录 web ui 选择 令牌登录



**部署 monitoring**

**k8s 运维相关**

**基础维护**

# 当需要对主机进行维护升级时，首先将节点主机设置成不可调度模式：

kubectl cordon［nodeid］

# 然后需要将主机上正在运行的容器驱赶到其它可用节点：

kubectl drain ［nodeid］

# 给予900秒宽限期优雅的调度

kubectl drain node1.k8s.novalocal --grace-period=120

# 当容器迁移完毕后，运维人员可以对该主机进行操作，配置升级性能参数调优等等。当对主机的维护操作完毕后， 再将主机设置成可调度模式：

kubectl uncordon [nodeid]

**Other**

**特殊 env**

# yaml 中的一些 特殊 env

env:

- name: MY\_POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: MY\_POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

- name: MY\_POD\_IP

valueFrom:

fieldRef:

fieldPath: status.podIP

SHARE

* [Share on Facebook](https://www.facebook.com/sharer/sharer.php?u=https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html)
* [Tweet](https://twitter.com/intent/tweet?source=https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html&text=kubernetes+1.12.1%20%7C%20%E5%B0%8F%E7%82%92%E8%82%89:%20https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html)
* [Share on Google+](https://plus.google.com/share?url=https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html)
* [Share on Reddit](http://www.reddit.com/submit?url=https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html&title=kubernetes+1.12.1%20%7C%20%E5%B0%8F%E7%82%92%E8%82%89)
* [Email](mailto:?subject=kubernetes+1.12.1%20%7C%20%E5%B0%8F%E7%82%92%E8%82%89&body=:%20https://jicki.me/kubernetes/2018/10/09/kubernetes-1.12.1.html)

PREVIOUS POST

[kubernetes 1.11.2](https://jicki.me/kubernetes/2018/08/10/kubernetes-1.11.2.html)

Powered by [jicki](https://jicki.me/) with [Github](https://github.com/jicki)