目 录

[基于Flannel网络模式构建Kubernetes高可用容器云 - v1.10.5 1](#_Toc518566029)

[第1章 Kubernetes集群规划 1](#_Toc518566030)

[1.1 集群架构 1](#_Toc518566031)

[1.1.1 总架构图 1](#_Toc518566032)

[1.1.2 LB节点逻辑图 1](#_Toc518566033)

[1.2 组件介绍 1](#_Toc518566034)

[1.2.1 Master组件 1](#_Toc518566035)

[1.2.2 Node组件 1](#_Toc518566036)

[1.2.3 第三方服务 1](#_Toc518566037)

[1.3 集群环境规划 1](#_Toc518566038)

[1.3.1 软件版本 1](#_Toc518566039)

[1.3.2 节点信息 1](#_Toc518566040)

[1.3.3 系统优化 1](#_Toc518566041)

[1.4 基础环境准备 1](#_Toc518566042)

[1.4.1 Kernel 和swap 1](#_Toc518566043)

[1.4.2 免密钥互信 1](#_Toc518566044)

[1.4.3 同步hosts文件 1](#_Toc518566045)

[1.4.4 安装Docker 1](#_Toc518566046)

[1.4.5 准备部署目录 1](#_Toc518566047)

[1.4.6 准备软件包 1](#_Toc518566048)

[1.4.7 安装 CFSSL 1](#_Toc518566049)

[1.5 基础证书生成 1](#_Toc518566050)

[1.5.1 初始化cfssl 1](#_Toc518566051)

[1.5.2 生成证书 1](#_Toc518566052)

[1.5.3 分发证书 1](#_Toc518566053)

[第2章 部署LB节点 1](#_Toc518566054)

[2.1 安装Keepalived 1](#_Toc518566055)

[2.2 安装Nginx 1](#_Toc518566056)

[第3章 部署ETCD节点 1](#_Toc518566057)

[3.1 准备etcd软件包 1](#_Toc518566058)

[3.2 创建etcd 证书签名请求 1](#_Toc518566059)

[3.3 生成etcd 证书和私钥 1](#_Toc518566060)

[3.4 设置ETCD配置文件 1](#_Toc518566061)

[3.5 创建ETCD系统服务 1](#_Toc518566062)

[3.6 重新加载系统服务 1](#_Toc518566063)

[3.7 验证集群 1](#_Toc518566064)

[第4章 部署Kubernetes Master节点 1](#_Toc518566065)

[4.1 准备软件包 1](#_Toc518566066)

[4.2 部署Kubernetes API 1](#_Toc518566067)

[4.2.1 创建生成CSR的 JSON 配置文件 1](#_Toc518566068)

[4.2.2 生成 kubernetes 证书和私钥 1](#_Toc518566069)

[4.2.3 创建API Server系统服务 1](#_Toc518566070)

[4.2.4 启动API Server服务 1](#_Toc518566071)

[4.3 部署Controller Manager 1](#_Toc518566072)

[4.3.1 启动Controller Manager 1](#_Toc518566073)

[4.4 部署Kubernetes Scheduler 1](#_Toc518566074)

[4.4.1 启动Kubernetes Scheduler 1](#_Toc518566075)

[4.5 部署kubectl 1](#_Toc518566076)

[4.5.1 创建并生成admin证书 1](#_Toc518566077)

[4.5.2 设置集群参数 1](#_Toc518566078)

[4.5.3 使用kubectl工具 1](#_Toc518566079)

[第5章 部署Kubernetes Node节点 1](#_Toc518566080)

[5.1 部署准备 1](#_Toc518566081)

[5.1.1 拷贝node节点软件 1](#_Toc518566082)

[5.1.2 CNI插件 1](#_Toc518566083)

[5.2 部署kubelet 1](#_Toc518566084)

[5.2.1 创建证书 1](#_Toc518566085)

[5.2.2 创建kubelet系统服务 1](#_Toc518566086)

[5.2.3 启动Kubelet 1](#_Toc518566087)

[5.2.4 授权CSR请求 1](#_Toc518566088)

[5.3 部署Kubernetes Proxy 1](#_Toc518566089)

[5.3.1 创建kube-proxy证书 1](#_Toc518566090)

[5.3.2 创建kube-proxy配置文件 1](#_Toc518566091)

[5.3.3 创建kube-proxy系统服务 1](#_Toc518566092)

[5.3.4 启动Kubernetes Proxy 1](#_Toc518566093)

[5.3.5 验证kube-proxy安装 1](#_Toc518566094)

[第6章 部署Flannel网络 1](#_Toc518566095)

[6.1 生成flannel证书 1](#_Toc518566096)

[6.2 准备软件包 1](#_Toc518566097)

[6.3 部署Flannel 1](#_Toc518566098)

[6.4 Flannel CNI集成 1](#_Toc518566099)

[6.5 配置Docker使用Flannel 1](#_Toc518566100)

[6.6 测试 1](#_Toc518566101)

[第7章 部署Kubernetes 工具 1](#_Toc518566102)

[7.1 Kubernetes Dashboard 1](#_Toc518566103)

[7.1.1 创建Dashboard yaml文件 1](#_Toc518566104)

[7.1.2 执行创建 1](#_Toc518566105)

[7.1.3 web页面访问 1](#_Toc518566106)

[7.2 Heapster 1](#_Toc518566107)

[7.3 Ingress Controller 1](#_Toc518566108)

[7.4 Helm Tiller Server 1](#_Toc518566109)

[7.5 Prometheus 1](#_Toc518566110)

基于Flannel网络模式构建Kubernetes高可用容器云 - v1.10.5

# Kubernetes集群规划

## 集群架构

### 总架构图



### LB节点逻辑图



## 组件介绍

### Master组件

|  |  |
| --- | --- |
| **组件名称** | **说明** |
| kube-apiserver | Kubernetes API，集群的统一入口，各组件协调者，以HTTP API提供接口服务，所有对象资源的增删改查和监听操作都交给APIServer处理后再提交给Etcd存储。 |
| kube-controller-manager | 处理集群中常规后台任务，一个资源对应一个控制器，而ControllerManager就是负责管理这些控制器的。 |
| kube-scheduler | 根据调度算法为新创建的Pod选择一个Node节点。 |

### Node组件

|  |  |
| --- | --- |
| **组件名称** | **说明** |
| kubelet | kubelet是Master在Node节点上的Agent，管理本机运行容器的生命周期，比如创建容器、Pod挂载数据卷、下载secret、获取容器和节点状态等工作。kubelet将每个Pod转换成一组容器。 |
| kube-proxy | 在Node节点上实现Pod网络代理，维护网络规则和四层负载均衡工作。 |
| docker或rocket/rkt | 运行容器。 |

### 第三方服务

|  |  |
| --- | --- |
| **组件名称** | **说明** |
| etcd | 分布式键值存储系统。用于保持集群状态，比如Pod、Service等对象信息。 |

## 集群环境规划

### 软件版本

|  |  |
| --- | --- |
| **软件名称** | **版本** |
| Linux操作系统 | CentOS 7.4 x86\_64 |
| keepalived | 1.3.5 |
| nginx | 1.12.2 |
| cfssl | 1.2 |
| Kubernetes | 1.10.5 |
| Docker | 17.03.2 |
| Etcd | 3.3.8 |
| Flannel | 0.10 |
| CNI | 0.6.0 |

### 节点信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **主机名** | **IP** | **VIP** | **组件** | **硬件配置** |
| LB-1 | 192.168.56.97 | 192.168.56.99 | keepalived nginx | **1c\*2g** |
| LB-2 | 192.168.56.98 |
| master-1 | 192.168.56.101 |  | kube-apiserver  kube-controller-manager  kube-scheduler 、kubectl、etcd |
| master-2 | 192.168.56.102 |  |
| master-3 | 192.168.56.103 |  |
| node-1 | 192.168.56.104 |  | kubelet 、 kube-proxy  docker、flannel |
| node-2 | 192.168.56.105 |  |
| node-3 | 192.168.56.106 |  |

### 系统优化

我们使用最新的（截止2018.05.08），Centos7.4迷你版本（CentOS-7-x86\_64-Minimal-1708.iso）

1.指定本地yum源

mkdir -p /etc/yum.repos.d/bak

mv /etc/yum.repos.d/\* /etc/yum.repos.d/bak/ > /dev/null 2>&1

tee > /etc/yum.repos.d/centos7.repo <<-'EOF'

[CentOS-7-x86\_64-base]

name=CentOS-7-x86\_64-base

baseurl=http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-base

enabled=1

priority=99

gpgcheck=0

[CentOS-7-x86\_64-epel]

name=CentOS-7-x86\_64-epel

baseurl=http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-epel

enabled=1

priority=99

gpgcheck=0

[CentOS-7-x86\_64-extras]

name=CentOS-7-x86\_64-extras

baseurl=http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-extras

enabled=1

priority=99

gpgcheck=0

[CentOS-7-x86\_64-updates]

name=CentOS-7-x86\_64-updates

baseurl=http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-updates

enabled=1

priority=99

gpgcheck=0

EOF

tee > cat /etc/yum.repos.d/docker-ce.repo <<-'EOF'

[CentOS-7-x86\_64-Docker-ce]

name=CentOS-7-x86\_64-Docker-ce

baseurl=http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-Docker-ce

enabled=1

priority=99

gpgcheck=0

EOF

2.安装常用包组

因为是迷你安装，好多软件包都没有需要我们手动安装

yum clean all && yum repolist

yum install lrzsz nmap tree dos2unix nc htop wget vim bash-completion screen lsof net-tools sshpass -y

yum groupinstall "Compatibility libraries" "Base" "Development tools" "debugging Tools" "Dial-up Networking Support" -y --exclude=bash

reboot

3使用chrony服务同步时间（centos7）

yum install chrony -y

systemctl enable chronyd.service

systemctl start chronyd.service

4.闭防火墙相关服务（生产中按需操作）

systemctl disable firewalld

systemctl disable NetworkManager

systemctl disable postfix

systemctl disable kdump.service

systemctl disable NetworkManager-wait-online.service

sed -i 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config

grep SELINUX=disabled /etc/selinux/config

setenforce 0

systemctl list-unit-files -t service | grep enable

5.调整文件描述符

echo '\* - nofile 100000 ' >>/etc/security/limits.conf

6.grep高亮

echo "alias grep='grep --color=auto'" >> /etc/profile

echo "alias egrep='egrep --color=auto'" >> /etc/profile

tail -2 /etc/profile

source /etc/profile

## 基础环境准备

### Kernel 和swap

1.所有节点配置内核属性

cat > /etc/sysctl.d/k8s.conf <<-'EOF'

net.ipv4.ip\_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sysctl -p /etc/sysctl.d/k8s.conf

2.swap交换分区设置

所有节点 Kubernetes v1.8+ 要求关闭 Swap，否则 kubelet 无法正常启动

记得/etc/fstab也要注解掉SWAP挂载

for NODE in master-2 master-3 node-1 node-2 node-3; do

echo "--- $NODE ---"

scp /etc/sysctl.d/k8s.conf $NODE:/etc/sysctl.d/

ssh $NODE 'sysctl -p /etc/sysctl.d/k8s.conf'

ssh $NODE 'swapoff -a && sysctl -w vm.swappiness=0'

done

### 免密钥互信

1.ssh加速设置

sed -i 's@#UseDNS yes@UseDNS no@g' /etc/ssh/sshd\_config

systemctl restart sshd

2.查看脚本内容

tee > configure\_ssh\_without\_pass.sh <<-'EOF'

#!/bin/bash

ssh-keygen -t rsa -b 2048 -N "" -f $HOME/.ssh/id\_rsa

cat $HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized\_keys

chmod 600 $HOME/.ssh/authorized\_keys

for ip in 97 98 101 102 103 104 105 106; do

echo "------ $ip ------"

sshpass -p '123456' rsync -av -e 'ssh -o StrictHostKeyChecking=no' $HOME/.ssh/authorized\_keys root@192.168.56.$ip:$HOME/.ssh/

# rsync -av -e 'ssh -o StrictHostKeyChecking=no' $HOME/.ssh/authorized\_keys root@192.168.56.$ip:$HOME/.ssh/

done

EOF

3.在所有节点执行

for ip in 97 98 101 102 103 104 105 106; do

echo "------ $ip ------"

scp configure\_ssh\_without\_pass.sh 192.168.56.$ip:/root/

ssh 192.168.56.$ip 'sh configure\_ssh\_without\_pass.sh'

done

### 同步hosts文件

1. 设定/etc/hosts解析到所有集群主机

tee > /etc/hosts <<-'EOF'

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4 centos74

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6 centos74

#lb

192.168.56.97 lb-1

192.168.56.98 lb-2

#kubernetes master

192.168.56.101 master-1

192.168.56.102 master-2

192.168.56.103 master-3

#kubernetes node

192.168.56.104 node-1

192.168.56.105 node-2

192.168.56.106 node-3

EOF

2.拷贝到所有节点

for ip in 97 98 101 102 103 104 105 106; do

echo "------ $ip ------"

scp /etc/hosts root@192.168.56.$ip:/etc/hosts

done

### 安装Docker

所有node节点安装Docker服务，并设置开机自启动

cd /tmp

RPM='http://192.168.56.253/cobbler/repo\_mirror/CentOS-7-x86\_64-Docker-ce/Packages'

wget $RPM/docker-ce-17.03.2.ce-1.el7.centos.x86\_64.rpm

wget $RPM/docker-ce-selinux-17.03.2.ce-1.el7.centos.noarch.rpm

yum localinstall docker-ce\*.rpm -y

systemctl restart docker && systemctl enable docker

### 准备部署目录

所有节点需要统一操作

1.创建kubernetes安装目录

mkdir -p /opt/kubernetes/{cfg,bin,ssl,log}

2.修改.bash\_profile，增加粗体部分

# .bash\_profile

# Get the aliases and functions

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:**/opt/kubernetes/bin**

export PATH

3.生效环境变量

source .bash\_profile

### 准备软件包

### 安装 CFSSL

1.下载或上传软件包

[root@master-1 ~]# cd /usr/local/src/

[root@master-1 src]# wget https://pkg.cfssl.org/R1.2/cfssl\_linux-amd64

[root@master-1 src]# wget https://pkg.cfssl.org/R1.2/cfssljson\_linux-amd64

[root@master-1 src]# chmod +x cfssl\*

[root@master-1 src]# mv cfssljson\_linux-amd64 cfssljson

[root@master-1 src]# mv cfssl\_linux-amd64 cfssl

2.复制cfssl到其他master节点

for NODE in master-1 master-2 master-3; do

echo "------ $NODE ------"

for FILE in cfssljson cfssl; do

scp /usr/local/src/$FILE $NODE:/opt/kubernetes/bin/

done

done

3.node节点上传需要的容器

docker load --input pause-amd64-3.1-images.tar.gz

docker load --input kubernetes-1.10.5-images.tar.gz

## 基础证书生成

### 初始化cfssl

[root@master-1 src]# mkdir ssl && cd ssl

[root@master-1 ssl]# cfssl print-defaults config > config.json

[root@master-1 ssl]# cfssl print-defaults csr > csr.json

### 生成证书

1.创建用来生成 CA 文件的 JSON 配置文件

[root@master-1 ssl]# tee > ca-config.json <<-'EOF'

{

"signing": {

"default": {

"expiry": "8760h"

},

"profiles": {

"kubernetes": {

"usages": [

"signing",

"key encipherment",

"server auth",

"client auth"

],

"expiry": "8760h"

}

}

}

}

EOF

2.创建用来生成 CA 证书签名请求（CSR）的 JSON 配置文件

[root@master-1 ssl]# tee > ca-csr.json <<-'EOF'

{

"CN": "kubernetes",

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

3.生成CA证书（ca.pem）和密钥（ca-key.pem）

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

[root@master-1 ssl]# ll ca\*

-rw-r--r-- 1 root root 290 Jun 29 23:24 ca-config.json

-rw-r--r-- 1 root root 1001 Jun 29 23:25 ca.csr

-rw-r--r-- 1 root root 208 Jun 29 23:24 ca-csr.json

-rw------- 1 root root 1675 Jun 29 23:25 ca-key.pem

-rw-r--r-- 1 root root 1359 Jun 29 23:25 ca.pem

### 分发证书

这些是最基本的，master和node都需要。

for NODE in master-1 master-2 master-3 node-1 node-2 node-3; do

echo "------ $NODE ------"

for FILE in ca.csr ca.pem ca-key.pem ca-config.json; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

# 部署LB节点

## 安装Keepalived

1.LB所有节点安装

yum install keepalived -y

2.配置keepalived

注意state和priority参数参数不能一样

tee > /etc/keepalived/keepalived.conf <<-'EOF'

! Configuration File for keepalived

global\_defs {

router\_id Kubernetes\_LB

}

vrrp\_instance K8S\_LB {

state MASTER

# state BACKUP

interface eth0

virtual\_router\_id 51

priority 200

# priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

virtual\_ipaddress {

192.168.56.99

}

}

EOF

3.启动keepalived

systemctl start keepalived.service

systemctl status keepalived.service

systemctl enable keepalived.service

4.检查VIP是否存在

[root@ib-1 ~]# ip a | grep eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP qlen 1000

inet 192.168.56.97/24 brd 192.168.56.255 scope global eth0

inet 192.168.56.99/32 scope global eth0

[root@ib-2 ~]# ip a| grep eth0

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP qlen 1000

inet 192.168.56.98/24 brd 192.168.56.255 scope global eth0

## 安装Nginx

1.所有LB节点安装nginx

yum install nginx -y

2.配置nginx为反向代理kubernetes的apiserver

tee > cat /etc/nginx/nginx.conf <<-'EOF'

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log;

pid /run/nginx.pid;

# Load dynamic modules. See /usr/share/nginx/README.dynamic.

include /usr/share/nginx/modules/\*.conf;

events {

worker\_connections 1024;

}

stream {

log\_format main '$remote\_addr $upstream\_addr - [$time\_local] $status $upstream\_bytes\_sent';

access\_log /var/log/nginx/k8s-access.log main;

upstream kube\_apiserver {

least\_conn;

server 192.168.56.101:6443 weight=1;

server 192.168.56.102:6443 weight=1;

server 192.168.56.103:6443 weight=1;

}

server {

listen 0.0.0.0:6443;

proxy\_pass kube\_apiserver;

proxy\_timeout 10m;

proxy\_connect\_timeout 1s;

}

}

EOF

3.启动nginx服务

systemctl start nginx.service

systemctl status nginx.service

systemctl enable nginx.service

# 部署ETCD节点

## 准备etcd软件包

1.上传或下载etcd软件包

[root@master-1 ssl]# cd ../

[root@master-1 src]# wget https://github.com/coreos/etcd/releases/download/v3.3.8/etcd-v3.3.8-linux-amd64.tar.gz

2.解压etcd软件包

[root@master-1 src]# tar xf etcd-v3.3.8-linux-amd64.tar.gz && cd etcd-v3.3.8-linux-amd64/

3.拷贝etcd命令到其他master节点

for NODE in master-1 master-2 master-3; do

echo "------ $NODE ------"

for FILE in etcd etcdctl; do

scp /usr/local/src/etcd-v3.3.8-linux-amd64/$FILE $NODE:/opt/kubernetes/bin/

done

done

## 创建etcd 证书签名请求

1.注意json配置文件中的IP地址

[root@master-1 etcd-v3.2.18-linux-amd64]# cd ../ssl/

[root@master-1 ssl]# tee > etcd-csr.json <<-'EOF'

{

"CN": "etcd",

"hosts": [

"127.0.0.1",

**"192.168.56.101",**

**"192.168.56.102",**

**"192.168.56.103"**

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

## 生成etcd 证书和私钥

1.执行生成命令

[root@master-1 ssl]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \

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

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

-profile=kubernetes etcd-csr.json | cfssljson -bare etcd

2.会生成以下证书文件

[root@master-1 ssl]# ll etcd\*

-rw-r--r-- 1 root root 1062 Jun 29 23:30 etcd.csr

-rw-r--r-- 1 root root 302 Jun 29 23:29 etcd-csr.json

-rw------- 1 root root 1675 Jun 29 23:30 etcd-key.pem

-rw-r--r-- 1 root root 1436 Jun 29 23:30 etcd.pem

3.拷贝证书到etcd节点

for NODE in master-1 master-2 master-3; do

echo "------ $NODE ------"

for FILE in etcd.csr etcd-csr.json etcd-key.pem etcd.pem; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

## 设置ETCD配置文件

注意粗体部分需要适当修改为具体环境

1.增加etcd-1配置文件

[root@master-1 ssl]# tee > /opt/kubernetes/cfg/etcd.conf <<-'EOF'

#[member]

**ETCD\_NAME**="etcd-1"

ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

#ETCD\_SNAPSHOT\_COUNTER="10000"

#ETCD\_HEARTBEAT\_INTERVAL="100"

#ETCD\_ELECTION\_TIMEOUT="1000"

**ETCD\_LISTEN\_PEER\_URLS**="https://192.168.56.101:2380"

**ETCD\_LISTEN\_CLIENT\_URLS**="https://192.168.56.101:2379,https://127.0.0.1:2379"

#ETCD\_MAX\_SNAPSHOTS="5"

#ETCD\_MAX\_WALS="5"

#ETCD\_CORS=""

#[cluster]

**ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS**="https://192.168.56.101:2380"

# if you use different ETCD\_NAME (e.g. test),

# set ETCD\_INITIAL\_CLUSTER value for this name, i.e. "test=http://..."

ETCD\_INITIAL\_CLUSTER="etcd-1=https://192.168.56.101:2380,etcd-2=https://192.168.56.102:2380,etcd-3=https://192.168.56.103:2380"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

ETCD\_INITIAL\_CLUSTER\_TOKEN="k8s-etcd-cluster"

**ETCD\_ADVERTISE\_CLIENT\_URLS**="https://192.168.56.101:2379"

#[security]

CLIENT\_CERT\_AUTH="true"

ETCD\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

PEER\_CLIENT\_CERT\_AUTH="true"

ETCD\_PEER\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_PEER\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_PEER\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

EOF

2.增加etcd-2配置文件

[root@master-2 ~]# tee > /opt/kubernetes/cfg/etcd.conf <<-'EOF'

#[member]

**ETCD\_NAME**="etcd-2"

ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

#ETCD\_SNAPSHOT\_COUNTER="10000"

#ETCD\_HEARTBEAT\_INTERVAL="100"

#ETCD\_ELECTION\_TIMEOUT="1000"

**ETCD\_LISTEN\_PEER\_URLS**="https://192.168.56.102:2380"

**ETCD\_LISTEN\_CLIENT\_URLS**="https://192.168.56.102:2379,https://127.0.0.1:2379"

#ETCD\_MAX\_SNAPSHOTS="5"

#ETCD\_MAX\_WALS="5"

#ETCD\_CORS=""

#[cluster]

**ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS**="https://192.168.56.102:2380"

# if you use different ETCD\_NAME (e.g. test),

# set ETCD\_INITIAL\_CLUSTER value for this name, i.e. "test=http://..."

**ETCD\_INITIAL\_CLUSTER**="etcd-1=https://192.168.56.101:2380,etcd-2=https://192.168.56.102:2380,etcd-3=https://192.168.56.103:2380"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

ETCD\_INITIAL\_CLUSTER\_TOKEN="k8s-etcd-cluster"

**ETCD\_ADVERTISE\_CLIENT\_URLS**="https://192.168.56.102:2379"

#[security]

CLIENT\_CERT\_AUTH="true"

ETCD\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

PEER\_CLIENT\_CERT\_AUTH="true"

ETCD\_PEER\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_PEER\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_PEER\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

EOF

3.增加etcd-3配置文件

[root@master-3 ~]# tee > /opt/kubernetes/cfg/etcd.conf <<-'EOF'

#[member]

**ETCD\_NAME**="etcd-3"

ETCD\_DATA\_DIR="/var/lib/etcd/default.etcd"

#ETCD\_SNAPSHOT\_COUNTER="10000"

#ETCD\_HEARTBEAT\_INTERVAL="100"

#ETCD\_ELECTION\_TIMEOUT="1000"

**ETCD\_LISTEN\_PEER\_URLS**="https://192.168.56.103:2380"

**ETCD\_LISTEN\_CLIENT\_URLS**="https://192.168.56.103:2379,https://127.0.0.1:2379"

#ETCD\_MAX\_SNAPSHOTS="5"

#ETCD\_MAX\_WALS="5"

#ETCD\_CORS=""

#[cluster]

**ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS**="https://192.168.56.103:2380"

# if you use different ETCD\_NAME (e.g. test),

# set ETCD\_INITIAL\_CLUSTER value for this name, i.e. "test=http://..."

**ETCD\_INITIAL\_CLUSTER**="etcd-1=https://192.168.56.101:2380,etcd-2=https://192.168.56.102:2380,etcd-3=https://192.168.56.103:2380"

ETCD\_INITIAL\_CLUSTER\_STATE="new"

ETCD\_INITIAL\_CLUSTER\_TOKEN="k8s-etcd-cluster"

**ETCD\_ADVERTISE\_CLIENT\_URLS**="https://192.168.56.103:2379"

#[security]

CLIENT\_CERT\_AUTH="true"

ETCD\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

PEER\_CLIENT\_CERT\_AUTH="true"

ETCD\_PEER\_CA\_FILE="/opt/kubernetes/ssl/ca.pem"

ETCD\_PEER\_CERT\_FILE="/opt/kubernetes/ssl/etcd.pem"

ETCD\_PEER\_KEY\_FILE="/opt/kubernetes/ssl/etcd-key.pem"

EOF

## 创建ETCD系统服务

1.增加etcd-1配置文件

[root@master-1 ~]# tee > /usr/lib/systemd/system/etcd.service <<-'EOF'

[Unit]

Description=Etcd Server

After=network.target

[Service]

Type=simple

WorkingDirectory=/var/lib/etcd

EnvironmentFile=-/opt/kubernetes/cfg/etcd.conf

# set GOMAXPROCS to number of processors

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /opt/kubernetes/bin/etcd"

Type=notify

[Install]

WantedBy=multi-user.target

EOF

2.拷贝到其他master节点

for NODE in master-2 master-3; do

echo "------ $NODE ------"

scp /usr/lib/systemd/system/etcd.service $NODE:/usr/lib/systemd/system/

done

## 重新加载系统服务

etcd节点必须2个以上才能启动，在所有节点上创建etcd存储目录并启动etcd

mkdir -p /var/lib/etcd

systemctl daemon-reload

systemctl start etcd

systemctl status etcd

systemctl enable etcd

## 验证集群

当etcd集群全部为healthy才能继续操作

[root@master-1 ~]# etcdctl --endpoints=https://192.168.56.101:2379 \

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

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

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

cluster-health

member 12ee9fd789572d33 is healthy: got healthy result from https://192.168.56.101:2379

member 81e815708872a583 is healthy: got healthy result from https://192.168.56.103:2379

member b65a08cbe628137f is healthy: got healthy result from https://192.168.56.102:2379

cluster is healthy

# 部署Kubernetes Master节点

## 准备软件包

1.本地上传或者官网下载Kubernetes v1.10.5的二进制安装包

<https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.10.md>

[root@master-1 ~]# cd /usr/local/src

[root@master-1 src]# wget https://dl.k8s.io/v1.10.5/kubernetes-client-linux-amd64.tar.gz

[root@master-1 src]# wget https://dl.k8s.io/v1.10.5/kubernetes-server-linux-amd64.tar.gz

[root@master-1 src]# wget https://dl.k8s.io/v1.10.5/kubernetes-node-linux-amd64.tar.gz

2.解压如下软件包，组成kubernetes v1.10.5安装包

[root@master-1 src]# tar zxf kubernetes.tar.gz

[root@master-1 src]# tar zxf kubernetes-server-linux-amd64.tar.gz

[root@master-1 src]# tar zxf kubernetes-client-linux-amd64.tar.gz

[root@master-1 src]# tar zxf kubernetes-node-linux-amd64.tar.gz

3.拷贝到其他master节点

for NODE in master-1 master-2 master-3; do

echo "--- $NODE ---"

for FILE in kube-apiserver kube-controller-manager kube-scheduler kubectl; do

scp /usr/local/src/kubernetes/server/bin/$FILE $NODE:/opt/kubernetes/bin/

done

done

## 部署Kubernetes API

### 创建生成CSR的 JSON 配置文件

注意IP地址，需要指定master节点的ip

[root@master-1 bin]# cd /usr/local/src/ssl/

[root@master-1 ssl]# tee > kubernetes-csr.json <<-'EOF'

{

"CN": "kubernetes",

"hosts": [

**"127.0.0.1",**

**"192.168.56.99"**,

**"10.1.0.1",**

"kubernetes",

"kubernetes.default",

"kubernetes.default.svc",

"kubernetes.default.svc.cluster",

"kubernetes.default.svc.cluster.local"

],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

### 生成 kubernetes 证书和私钥

1.生成证书

[root@master-1 ssl]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \

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

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

-profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes

2.拷贝到其他master节点

for NODE in master-1 master-2 master-3; do

echo "--- $NODE ---"

for FILE in kubernetes-key.pem kubernetes.pem; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

3.创建 kube-apiserver 使用的客户端 token 文件

[root@master-1 ssl]# cd

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

a96360546b9e70743cec635ba9d039b4

[root@master-1 ~]# tee > /opt/kubernetes/ssl/bootstrap-token.csv <<EOF

a96360546b9e70743cec635ba9d039b4,kubelet-bootstrap,10001,"system:kubelet-bootstrap"

EOF

4.创建基础用户名/密码认证配置

[root@master-1 ~]# tee > /opt/kubernetes/ssl/basic-auth.csv <<-'EOF'

admin,admin,1

readonly,readonly,2

EOF

5.拷贝到其他master节点

for NODE in master-2 master-3; do

echo "------ $NODE ------"

for FILE in bootstrap-token.csv basic-auth.csv; do

scp /opt/kubernetes/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

### 创建API Server系统服务

1.创建master-1的api server系统服务

[root@master-1 ~]# tee > /usr/lib/systemd/system/kube-apiserver.service <<-'EOF'

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/opt/kubernetes/bin/kube-apiserver \

--admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota,NodeRestriction \

**--bind-address**=192.168.56.101 \

--insecure-bind-address=127.0.0.1 \

--authorization-mode=Node,RBAC \

--runtime-config=rbac.authorization.k8s.io/v1 \

--kubelet-https=true \

--anonymous-auth=false \

--basic-auth-file=/opt/kubernetes/ssl/basic-auth.csv \

--enable-bootstrap-token-auth \

--token-auth-file=/opt/kubernetes/ssl/bootstrap-token.csv \

--service-cluster-ip-range=10.1.0.0/16 \

--service-node-port-range=20000-40000 \

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

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

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

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

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

--etcd-certfile=/opt/kubernetes/ssl/kubernetes.pem \

--etcd-keyfile=/opt/kubernetes/ssl/kubernetes-key.pem \

**--etcd-servers**=https://192.168.56.101:2379,https://192.168.56.102:2379,https://192.168.56.103:2379 \

--enable-swagger-ui=true \

--allow-privileged=true \

--audit-log-maxage=30 \

--audit-log-maxbackup=3 \

--audit-log-maxsize=100 \

--audit-log-path=/opt/kubernetes/log/api-audit.log \

--event-ttl=1h \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

Type=notify

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

2.创建master-2的api server系统服务

[root@master-2 ~]# tee > /usr/lib/systemd/system/kube-apiserver.service <<-'EOF'

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/opt/kubernetes/bin/kube-apiserver \

--admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota,NodeRestriction \

**--bind-address**=192.168.56.102 \

--insecure-bind-address=127.0.0.1 \

--authorization-mode=Node,RBAC \

--runtime-config=rbac.authorization.k8s.io/v1 \

--kubelet-https=true \

--anonymous-auth=false \

--basic-auth-file=/opt/kubernetes/ssl/basic-auth.csv \

--enable-bootstrap-token-auth \

--token-auth-file=/opt/kubernetes/ssl/bootstrap-token.csv \

--service-cluster-ip-range=10.1.0.0/16 \

--service-node-port-range=20000-40000 \

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

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

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

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

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

--etcd-certfile=/opt/kubernetes/ssl/kubernetes.pem \

--etcd-keyfile=/opt/kubernetes/ssl/kubernetes-key.pem \

**--etcd-servers**=https://192.168.56.101:2379,https://192.168.56.102:2379,https://192.168.56.103:2379 \

--enable-swagger-ui=true \

--allow-privileged=true \

--audit-log-maxage=30 \

--audit-log-maxbackup=3 \

--audit-log-maxsize=100 \

--audit-log-path=/opt/kubernetes/log/api-audit.log \

--event-ttl=1h \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

Type=notify

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

3.创建master-3的api server系统服务

[root@master-3 ~]# tee > /usr/lib/systemd/system/kube-apiserver.service <<-'EOF'

[Unit]

Description=Kubernetes API Server

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

After=network.target

[Service]

ExecStart=/opt/kubernetes/bin/kube-apiserver \

--admission-control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,ResourceQuota,NodeRestriction \

**--bind-address**=192.168.56.103 \

--insecure-bind-address=127.0.0.1 \

--authorization-mode=Node,RBAC \

--runtime-config=rbac.authorization.k8s.io/v1 \

--kubelet-https=true \

--anonymous-auth=false \

--basic-auth-file=/opt/kubernetes/ssl/basic-auth.csv \

--enable-bootstrap-token-auth \

--token-auth-file=/opt/kubernetes/ssl/bootstrap-token.csv \

--service-cluster-ip-range=10.1.0.0/16 \

--service-node-port-range=20000-40000 \

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

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

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

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

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

--etcd-certfile=/opt/kubernetes/ssl/kubernetes.pem \

--etcd-keyfile=/opt/kubernetes/ssl/kubernetes-key.pem \

**--etcd-servers**=https://192.168.56.101:2379,https://192.168.56.102:2379,https://192.168.56.103:2379 \

--enable-swagger-ui=true \

--allow-privileged=true \

--audit-log-maxage=30 \

--audit-log-maxbackup=3 \

--audit-log-maxsize=100 \

--audit-log-path=/opt/kubernetes/log/api-audit.log \

--event-ttl=1h \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

Type=notify

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

### 启动API Server服务

systemctl daemon-reload

systemctl start kube-apiserver

systemctl status kube-apiserver

systemctl enable kube-apiserver

## 部署Controller Manager

1.创建系统服务

[root@master-1 ~]# tee > /usr/lib/systemd/system/kube-controller-manager.service <<-'EOF'

[Unit]

Description=Kubernetes Controller Manager

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

[Service]

ExecStart=/opt/kubernetes/bin/kube-controller-manager \

--address=127.0.0.1 \

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

--allocate-node-cidrs=true \

--service-cluster-ip-range=10.1.0.0/16 \

--cluster-cidr=10.2.0.0/16 \

--cluster-name=kubernetes \

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

--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 \

--leader-elect=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

EOF

2.拷贝到其他master节点

for NODE in master-2 master-3; do

echo "------ $NODE ------"

scp /usr/lib/systemd/system/kube-controller-manager.service $NODE:/usr/lib/systemd/system/

done

### 启动Controller Manager

systemctl daemon-reload

systemctl start kube-controller-manager

systemctl status kube-controller-manager

systemctl enable kube-controller-manager

## 部署Kubernetes Scheduler

[root@master01 ~]# tee > /usr/lib/systemd/system/kube-scheduler.service <<-'EOF'

[Unit]

Description=Kubernetes Scheduler

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

[Service]

ExecStart=/opt/kubernetes/bin/kube-scheduler \

--address=127.0.0.1 \

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

--leader-elect=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

EOF

2.拷贝kubernetes manage 到其他master节点

for NODE in master-2 master-3; do

echo "------ $NODE ------"

scp /usr/lib/systemd/system/kube-scheduler.service $NODE:/usr/lib/systemd/system/

done

### 启动Kubernetes Scheduler

所有master启动

systemctl daemon-reload

systemctl start kube-scheduler

systemctl status kube-scheduler

systemctl enable kube-scheduler

## 部署kubectl

### 创建并生成admin证书

1.创建 admin 证书签名请求

[root@master-1 ~]# cd /usr/local/src/ssl/

[root@master-1 ssl]# tee > admin-csr.json <<-'EOF'

{

"CN": "admin",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "system:masters",

"OU": "System"

}

]

}

EOF

2.生成 admin 证书和私钥：

[root@master-1 ssl]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \

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

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

-profile=kubernetes admin-csr.json | cfssljson -bare admin

3.查看生成的admin证书

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

-rw-r--r-- 1 root root 1009 Jun 30 00:10 admin.csr

-rw-r--r-- 1 root root 229 Jun 30 00:09 admin-csr.json

-rw------- 1 root root 1.7K Jun 30 00:10 admin-key.pem

-rw-r--r-- 1 root root 1.4K Jun 30 00:10 admin.pem

4.拷贝admin证书到其他master节点

for NODE in master-1 master-2 master-3; do

echo "------ $NODE ------"

for FILE in admin-key.pem admin.pem; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

### 设置集群参数

1.这个是指定的haproxy的端口

[root@master-1 ssl]# kubectl config set-cluster kubernetes \

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

--embed-certs=true \

--server=https://192.168.56.99:6443

2.设置客户端认证参数

[root@master-1 ssl]# kubectl config set-credentials admin \

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

--embed-certs=true \

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

3.设置上下文参数

[root@master-1 ssl]# kubectl config set-context kubernetes \

--cluster=kubernetes \

--user=admin

4.设置默认上下文

[root@master-1 ssl]# kubectl config use-context kubernetes

### 使用kubectl工具

0.拷贝amdin授权到master节点

for NODE in master-2 master-3; do

echo "------ $NODE ------"

scp -r /root/.kube $NODE:/root/

done

1.在master节点测试

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

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

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

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

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

2.拷贝admin授权到非集群节点测试

[root@master-1 ~]# scp -r .kube root@192.168.56.4:/root

[root@master-1 ~]# scp /usr/local/src/kubernetes/node/bin/kubectl root@192.168.56.4:/usr/local/bin/

3.测试

[root@centos72 ~]# kubectl get cs

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

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

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

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

4.LB节点查看日志

[root@ib-1 ~]# cat /var/log/nginx/k8s-access.log

192.168.56.4 192.168.56.101:6443 - [30/Jun/2018:00:40:44 +0800] 200 2023

# 部署Kubernetes Node节点

## 部署准备

### 拷贝node节点软件

1.将kubelet和kube-proxy软件包从master节点复制到node节点中去。

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

for FILE in kubelet kube-proxy; do

scp /usr/local/src/kubernetes/node/bin/$FILE $NODE:/opt/kubernetes/bin/

done

done

### CNI插件

1.所有node节点配置文件

mkdir -p /etc/cni/net.d

tee > /etc/cni/net.d/10-default.conf <<-'EOF'

{

"name": "flannel",

"type": "flannel",

"delegate": {

"bridge": "docker0",

"isDefaultGateway": true,

"mtu": 1400

}

}

EOF

2.node-1节点上传或下载CNI插件工具

https://github.com/containernetworking/cni/releases/download/

mkdir -p /opt/kubernetes/bin/cni

cd /opt/kubernetes/bin/cni

tar xf cni-plugins-amd64-v0.6.0.tgz

rm -f cni-plugins-amd64-v0.6.0.tgz

3.拷贝到其他节点

for NODE in node-2 node-3; do

echo "------ $NODE ------"

scp -r /opt/kubernetes/bin/cni $NODE:/opt/kubernetes/bin/

done

## 部署kubelet

### 创建证书

1.在master节点创建角色绑定

[root@master-1 ~]# cd /usr/local/src/ssl/

[root@master-1 ssl]# kubectl create clusterrolebinding kubelet-bootstrap \

--clusterrole=system:node-bootstrapper \

--user=kubelet-bootstrap

2.创建 kubelet bootstrapping kubeconfig 文件 设置集群参数

[root@master-1 ssl]# kubectl config set-cluster kubernetes \

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

--embed-certs=true \

--server=https://192.168.56.99:6443 \

--kubeconfig=bootstrap.kubeconfig

3.设置客户端认证参数

这个token用api-server的token要保持一样

[root@master-1 ssl]# kubectl config set-credentials kubelet-bootstrap \

--token=a96360546b9e70743cec635ba9d039b4 \

--kubeconfig=bootstrap.kubeconfig

4.设置上下文参数

[root@master-1 ssl]# kubectl config set-context default \

--cluster=kubernetes \

--user=kubelet-bootstrap \

--kubeconfig=bootstrap.kubeconfig

5.选择默认上下文

[root@master-1 ssl]# kubectl config use-context default --kubeconfig=bootstrap.kubeconfig

6.拷贝bootstrap.kubeconfig 到node节点

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

scp /usr/local/src/ssl/bootstrap.kubeconfig $NODE:/opt/kubernetes/cfg/

done

### 创建kubelet系统服务

1.创建node-1的kubelet服务配置

[root@node-1 ~]# tee > /usr/lib/systemd/system/kubelet.service <<-'EOF'

[Unit]

Description=Kubernetes Kubelet

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

After=docker.service

Requires=docker.service

[Service]

WorkingDirectory=/var/lib/kubelet

ExecStart=/opt/kubernetes/bin/kubelet \

**--address**=192.168.56.104 \

**--hostname-override**=192.168.56.104 \

--pod-infra-container-image=k8s.gcr.io/pause-amd64:3.1 \

--experimental-bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \

--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \

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

**--network-plugin**=cni \

--cni-conf-dir=/etc/cni/net.d \

--cni-bin-dir=/opt/kubernetes/bin/cni \

**--cluster-dns**=10.1.0.2 \

--cluster-domain=cluster.local \

--hairpin-mode hairpin-veth \

--allow-privileged=true \

--fail-swap-on=false \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

EOF

2.创建node-2的kubelet服务配置

[root@node-2 ~]# tee > /usr/lib/systemd/system/kubelet.service <<-'EOF'

[Unit]

Description=Kubernetes Kubelet

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

After=docker.service

Requires=docker.service

[Service]

WorkingDirectory=/var/lib/kubelet

ExecStart=/opt/kubernetes/bin/kubelet \

**--address**=192.168.56.105 \

**--hostname-override**=192.168.56.105 \

--pod-infra-container-image=k8s.gcr.io/pause-amd64:3.1 \

--experimental-bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \

--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \

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

**--network-plugin**=cni \

--cni-conf-dir=/etc/cni/net.d \

--cni-bin-dir=/opt/kubernetes/bin/cni \

**--cluster-dns**=10.1.0.2 \

--cluster-domain=cluster.local \

--hairpin-mode hairpin-veth \

--allow-privileged=true \

--fail-swap-on=false \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

EOF

3.创建node-3的kubelet服务配置

[root@node-3 ~]# tee > /usr/lib/systemd/system/kubelet.service <<-'EOF'

[Unit]

Description=Kubernetes Kubelet

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

After=docker.service

Requires=docker.service

[Service]

WorkingDirectory=/var/lib/kubelet

ExecStart=/opt/kubernetes/bin/kubelet \

**--address**=192.168.56.106 \

**--hostname-override**=192.168.56.106 \

--pod-infra-container-image=k8s.gcr.io/pause-amd64:3.1 \

--experimental-bootstrap-kubeconfig=/opt/kubernetes/cfg/bootstrap.kubeconfig \

--kubeconfig=/opt/kubernetes/cfg/kubelet.kubeconfig \

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

**--network-plugin**=cni \

--cni-conf-dir=/etc/cni/net.d \

--cni-bin-dir=/opt/kubernetes/bin/cni \

**--cluster-dns**=10.1.0.2 \

--cluster-domain=cluster.local \

--hairpin-mode hairpin-veth \

--allow-privileged=true \

--fail-swap-on=false \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

EOF

### 启动Kubelet

1.在所有node节点启动kubelet

mkdir -p /var/lib/kubelet

systemctl daemon-reload

systemctl start kubelet

systemctl status kubelet

systemctl enable kubelet

### 授权CSR请求

1. 查看csr请求 注意是在master或有kubectl命令及amdin权限上的机器执行。

[root@centos72 ~]# kubectl get csr

NAME AGE REQUESTOR CONDITION

node-csr-5I0JTXApQPTWuBtZRvjPty3Dxhz0Pm9UC7i8-wSIA3M 6s kubelet-bootstrap Pending

node-csr-Jt\_srppP2dyhtIN\_RMYxAZX\_Wh9yl4u7j9inTnt9iuQ 11s kubelet-bootstrap Pending

node-csr-mztVBII0hL74kouVOr295lRgW0-3mDCPSpYL8qVJSnc 9s kubelet-bootstrap Pending

2.批准kubelet 的 TLS 证书请求

[root@centos72 ~]# kubectl get csr|grep 'Pending' | awk 'NR>0{print $1}'| xargs kubectl certificate approve

certificatesigningrequest.certificates.k8s.io "node-csr-5I0JTXApQPTWuBtZRvjPty3Dxhz0Pm9UC7i8-wSIA3M" approved

certificatesigningrequest.certificates.k8s.io "node-csr-Jt\_srppP2dyhtIN\_RMYxAZX\_Wh9yl4u7j9inTnt9iuQ" approved

certificatesigningrequest.certificates.k8s.io "node-csr-mztVBII0hL74kouVOr295lRgW0-3mDCPSpYL8qVJSnc" approved

3.再次查看csr请求，执行完毕后，查看节点状态已经是Ready的状态了

[root@centos72 ~]# kubectl get csr

NAME AGE REQUESTOR CONDITION

node-csr-5I0JTXApQPTWuBtZRvjPty3Dxhz0Pm9UC7i8-wSIA3M 23s kubelet-bootstrap Approved,Issued

node-csr-Jt\_srppP2dyhtIN\_RMYxAZX\_Wh9yl4u7j9inTnt9iuQ 28s kubelet-bootstrap Approved,Issued

node-csr-mztVBII0hL74kouVOr295lRgW0-3mDCPSpYL8qVJSnc 26s kubelet-bootstrap Approved,Issued

4.查看node状态

[root@centos72 ~]# kubectl get node

NAME STATUS ROLES AGE VERSION

192.168.56.104 Ready <none> 1m v1.10.5

192.168.56.105 Ready <none> 1m v1.10.5

192.168.56.106 Ready <none> 1m v1.10.5

## 部署Kubernetes Proxy

1.配置kube-proxy使用LVS，需要在node节点执行

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

ssh $NODE ' yum install -y ipvsadm ipset conntrack'

done

### 创建kube-proxy证书

1.证书请求

[root@master-1 ~]# cd /usr/local/src/ssl/

[root@master-1 ssl]# tee > kube-proxy-csr.json <<-'EOF'

{

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

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

2.生成证书

[root@master-1 ssl]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \

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

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

-profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy

3.分发证书到所有节点

for NODE in master-1 master-2 master-3 node-1 node-2 node-3; do

echo "------ $NODE ------"

for FILE in kube-proxy-key.pem kube-proxy.pem; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

### 创建kube-proxy配置文件

1.创建proxy.kubeconfig文件，需要在master节点上操作

[root@master-1 ssl]# kubectl config set-cluster kubernetes \

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

--embed-certs=true \

--server=https://192.168.56.99:6443 \

--kubeconfig=kube-proxy.kubeconfig

[root@master-1 ssl]# kubectl config set-credentials kube-proxy \

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

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

--embed-certs=true \

--kubeconfig=kube-proxy.kubeconfig

[root@master-1 ssl]# kubectl config set-context default \

--cluster=kubernetes \

--user=kube-proxy \

--kubeconfig=kube-proxy.kubeconfig

[root@master-1 ssl]# kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig

2.分发kubeconfig配置文件

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

scp /usr/local/src/ssl/kube-proxy.kubeconfig $NODE:/opt/kubernetes/cfg/

done

### 创建kube-proxy系统服务

1.创建node-1的kube-proxy系统服务

[root@node-1 ~]# tee > /usr/lib/systemd/system/kube-proxy.service <<-'EOF'

[Unit]

Description=Kubernetes Kube-Proxy Server

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

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

ExecStart=/opt/kubernetes/bin/kube-proxy \

**--bind-address**=192.168.56.104 \

**--hostname-override**=192.168.56.104 \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig \

--masquerade-all \

--feature-gates=SupportIPVSProxyMode=true \

--proxy-mode=ipvs \

--ipvs-min-sync-period=5s \

--ipvs-sync-period=5s \

--ipvs-scheduler=rr \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

2.创建node-2的kube-proxy系统服务

[root@node-2 ~]# tee > /usr/lib/systemd/system/kube-proxy.service <<-'EOF'

[Unit]

Description=Kubernetes Kube-Proxy Server

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

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

ExecStart=/opt/kubernetes/bin/kube-proxy \

**--bind-address**=192.168.56.105 \

**--hostname-override**=192.168.56.105 \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig \

--masquerade-all \

--feature-gates=SupportIPVSProxyMode=true \

--proxy-mode=ipvs \

--ipvs-min-sync-period=5s \

--ipvs-sync-period=5s \

--ipvs-scheduler=rr \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

3.创建node-3的kube-proxy系统服务

[root@node-3 ~]# tee > /usr/lib/systemd/system/kube-proxy.service <<-'EOF'

[Unit]

Description=Kubernetes Kube-Proxy Server

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

After=network.target

[Service]

WorkingDirectory=/var/lib/kube-proxy

ExecStart=/opt/kubernetes/bin/kube-proxy \

**--bind-address**=192.168.56.106 \

**--hostname-override**=192.168.56.106 \

--kubeconfig=/opt/kubernetes/cfg/kube-proxy.kubeconfig \

--masquerade-all \

--feature-gates=SupportIPVSProxyMode=true \

--proxy-mode=ipvs \

--ipvs-min-sync-period=5s \

--ipvs-sync-period=5s \

--ipvs-scheduler=rr \

--logtostderr=true \

--v=2 \

--logtostderr=false \

--log-dir=/opt/kubernetes/log

Restart=on-failure

RestartSec=5

LimitNOFILE=65536

[Install]

WantedBy=multi-user.target

EOF

### 启动Kubernetes Proxy

1.全部node节点启动kube-proxy

mkdir -p /var/lib/kube-proxy

systemctl daemon-reload

systemctl start kube-proxy

systemctl status kube-proxy

systemctl enable kube-proxy

### 验证kube-proxy安装

1.以node-1节点为例检查LVS状态

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

ssh $NODE 'ipvsadm -Ln '

done

------ node-1 ------

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 10.1.0.1:443 rr persistent 10800

-> 192.168.56.102:6443 Masq 1 0 0

------ node-2 ------

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 10.1.0.1:443 rr persistent 10800

-> 192.168.56.102:6443 Masq 1 0 0

------ node-3 ------

IP Virtual Server version 1.2.1 (size=4096)

Prot LocalAddress:Port Scheduler Flags

-> RemoteAddress:Port Forward Weight ActiveConn InActConn

TCP 10.1.0.1:443 rr persistent 10800

-> 192.168.56.101:6443 Masq 1 0 0

2.如果你在node节点都安装了kubelet和proxy服务，使用下面的命令可以检查状态：

[root@centos72 ~]# kubectl get node

NAME STATUS ROLES AGE VERSION

192.168.56.104 Ready <none> 7m v1.10.5

192.168.56.105 Ready <none> 7m v1.10.5

192.168.56.106 Ready <none> 7m v1.10.5

# 部署Flannel网络

## 生成flannel证书

1.为Flannel生成证书，在master节点操作

[root@master-1 ~]# cd /usr/local/src/ssl/

[root@master-1 ssl]# tee > flanneld-csr.json <<-'EOF'

{

"CN": "flanneld",

"hosts": [],

"key": {

"algo": "rsa",

"size": 2048

},

"names": [

{

"C": "CN",

"ST": "BeiJing",

"L": "BeiJing",

"O": "k8s",

"OU": "System"

}

]

}

EOF

2.生成证书

[root@master-1 ssl]# cfssl gencert -ca=/opt/kubernetes/ssl/ca.pem \

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

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

-profile=kubernetes flanneld-csr.json | cfssljson -bare flanneld

3.分发证书

for NODE in master-1 master-2 master-3 node-1 node-2 node-3; do

echo "------ $NODE ------"

for FILE in flanneld-key.pem flanneld.pem; do

scp /usr/local/src/ssl/$FILE $NODE:/opt/kubernetes/ssl/

done

done

## 准备软件包

1.复制对应脚本到/opt/kubernetes/bin目录下,master操作

[root@master-1 ~]# cd /usr/local/src/kubernetes && tar xf kubernetes-src.tar.gz

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

scp /usr/local/src/kubernetes/cluster/centos/node/bin/remove-docker0.sh $NODE:/opt/kubernetes/bin/

done

2.下载或上传Flannel软件包

[root@node-1 ~]# cd /usr/local/src/

[root@node-1 src]# wget https://github.com/coreos/flannel/releases/download/v0.10.0/flannel-v0.10.0-linux-amd64.tar.gz

[root@node-1 src]# tar zxf flannel-v0.10.0-linux-amd64.tar.gz && cd flannel-v0.10.0-linux-amd64

[root@node-1 flannel-v0.10.0-linux-amd64]# chmod +x \*

3.复制到node节点

for NODE in node-1 node-2 node-3; do

echo "------ $NODE ------"

for FILE in flanneld mk-docker-opts.sh; do

scp /usr/local/src/$FILE $NODE:/opt/kubernetes/bin/

done

done

## 部署Flannel

1.增加flannel配置文件

[root@node-1 ~]# tee > /opt/kubernetes/cfg/flannel <<-'EOF'

FLANNEL\_ETCD="-etcd-endpoints=https://192.168.56.101:2379,https://192.168.56.102:2379,https://192.168.56.103:2379"

FLANNEL\_ETCD\_KEY="-etcd-prefix=/kubernetes/network"

FLANNEL\_ETCD\_CAFILE="--etcd-cafile=/opt/kubernetes/ssl/ca.pem"

FLANNEL\_ETCD\_CERTFILE="--etcd-certfile=/opt/kubernetes/ssl/flanneld.pem"

FLANNEL\_ETCD\_KEYFILE="--etcd-keyfile=/opt/kubernetes/ssl/flanneld-key.pem"

EOF

2.复制配置到其它节点上

for NODE in node-2 node-3; do

echo "------ $NODE ------"

scp /opt/kubernetes/cfg/flannel $NODE:/opt/kubernetes/cfg/

done

3.部署Flannel系统服务

[root@node-1 ~]# tee > /usr/lib/systemd/system/flannel.service <<-'EOF'

[Unit]

Description=Flanneld overlay address etcd agent

After=network.target

Before=docker.service

[Service]

EnvironmentFile=-/opt/kubernetes/cfg/flannel

ExecStartPre=/opt/kubernetes/bin/remove-docker0.sh

ExecStart=/opt/kubernetes/bin/flanneld ${FLANNEL\_ETCD} ${FLANNEL\_ETCD\_KEY} ${FLANNEL\_ETCD\_CAFILE} ${FLANNEL\_ETCD\_CERTFILE} ${FLANNEL\_ETCD\_KEYFILE}

ExecStartPost=/opt/kubernetes/bin/mk-docker-opts.sh -d /run/flannel/docker

Type=notify

[Install]

WantedBy=multi-user.target

RequiredBy=docker.service

EOF

4.复制系统服务脚本到其它节点上

for NODE in node-2 node-3; do

echo "------ $NODE ------"

scp /usr/lib/systemd/system/flannel.service $NODE:/usr/lib/systemd/system/

done

## Flannel CNI集成

<https://github.com/containernetworking/plugins/releases>

1.master创建Etcd的key

[root@master-1 ~]# /opt/kubernetes/bin/etcdctl \

--ca-file /opt/kubernetes/ssl/ca.pem \

--cert-file /opt/kubernetes/ssl/flanneld.pem \

--key-file /opt/kubernetes/ssl/flanneld-key.pem \

--no-sync -C https://192.168.56.101:2379,https://192.168.56.102:2379,https://192.168.56.103:2379 \

mk /kubernetes/network/config '{ "Network": "10.2.0.0/16", "Backend": { "Type": "vxlan", "VNI": 1 }}'

2.node节点启动flannel

systemctl daemon-reload

systemctl start flannel

systemctl status flannel

systemctl enable flannel

## 配置Docker使用Flannel

1.编辑docker系统服务

[root@node01 ~]# vim /usr/lib/systemd/system/docker.service

[Unit] #在Unit下面修改After和增加Requires

After=network-online.target firewalld.service **flannel.service**

Wants=network-online.target

**Requires=flannel.service**

[Service] #增加EnvironmentFile=-/run/flannel/docker

Type=notify

**EnvironmentFile=-/run/flannel/docker**

ExecStart=/usr/bin/dockerd **$DOCKER\_OPTS**

2.拷贝到node节点

for NODE in node-2 node-3; do

echo "------ $NODE ------"

scp /usr/lib/systemd/system/docker.service $NODE:/usr/lib/systemd/system/

done

3.重启Docker

systemctl daemon-reload

systemctl restart docker

systemctl status docker

systemctl enable docker

## 测试

1.创建一个测试用的deployment

[root@master-1 ~]# kubectl run net-test --image=alpine --replicas=2 sleep 360000

deployment.apps "net-test" created

2.查看获取IP情况

[root@master-1 ~]# kubectl get pod -o wide

NAME READY STATUS RESTARTS AGE IP NODE

net-test-5767cb94df-zcprt 1/1 Running 0 59s 10.2.54.2 192.168.56.242

net-test-5767cb94df-zkr2j 1/1 Running 0 59s 10.2.54.3 192.168.56.242

3.node节点测试联通性

[root@node-1 ~]# ping 10.2.54.2 -c 1

PING 10.2.54.2 (10.2.54.2) 56(84) bytes of data.

64 bytes from 10.2.54.2: icmp\_seq=1 ttl=63 time=0.354 ms

--- 10.2.54.2 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.354/0.354/0.354/0.000 ms

[root@node02 ~]# ping 10.2.54.3 -c 1

PING 10.2.54.3 (10.2.54.3) 56(84) bytes of data.

64 bytes from 10.2.54.3: icmp\_seq=1 ttl=64 time=0.062 ms

--- 10.2.54.3 ping statistics ---

1 packets transmitted, 1 received, 0% packet loss, time 0ms

rtt min/avg/max/mdev = 0.062/0.062/0.062/0.000 ms

# 部署Kubernetes 工具

## Kubernetes Dashboard

### 创建Dashboard yaml文件

1. dashboard-rbac.yaml

[root@master01 UI]# **cat > dashboard-rbac.yaml <<-'EOF'**

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

addonmanager.kubernetes.io/mode: Reconcile

name: kubernetes-dashboard

namespace: kube-system

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

labels:

k8s-app: kubernetes-dashboard

addonmanager.kubernetes.io/mode: Reconcile

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kube-system

EOF

2 dashboard-deployment.yaml

[root@master01 UI]# tee > cat dashboard-deployment.yaml <<-'EOF'

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: kubernetes-dashboard

namespace: kube-system

labels:

k8s-app: kubernetes-dashboard

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

spec:

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

annotations:

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

spec:

serviceAccountName: kubernetes-dashboard

containers:

- name: kubernetes-dashboard

image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.8.3

resources:

limits:

cpu: 100m

memory: 300Mi

requests:

cpu: 100m

memory: 100Mi

ports:

- containerPort: 9090

protocol: TCP

livenessProbe:

httpGet:

scheme: HTTP

path: /

port: 9090

initialDelaySeconds: 30

timeoutSeconds: 30

tolerations:

- key: "CriticalAddonsOnly"

operator: "Exists"

EOF

3dashboard-service.yaml

[root@master01 UI]# **cat > dashboard-service.yaml <<-'EOF'**

apiVersion: v1

kind: Service

metadata:

name: kubernetes-dashboard

namespace: kube-system

labels:

k8s-app: kubernetes-dashboard

kubernetes.io/cluster-service: "true"

addonmanager.kubernetes.io/mode: Reconcile

spec:

type: NodePort

selector:

k8s-app: kubernetes-dashboard

ports:

- port: 80

targetPort: 9090

EOF

### 执行创建

1.执行创建的命令

kubectl create -f dashboard-rbac.yaml

kubectl create -f dashboard-deployment.yaml

kubectl create -f dashboard-service.yaml

2.查看dashboard的状态

[root@centos72 ~]#

[root@centos72 ~]# kubectl get all -n kube-system

NAME READY STATUS RESTARTS AGE

pod/kubernetes-dashboard-679b8b8457-5jhkj 1/1 Running 0 3m

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

service/kubernetes-dashboard NodePort 10.1.71.161 <none> 80:**30768**/TCP 3m

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

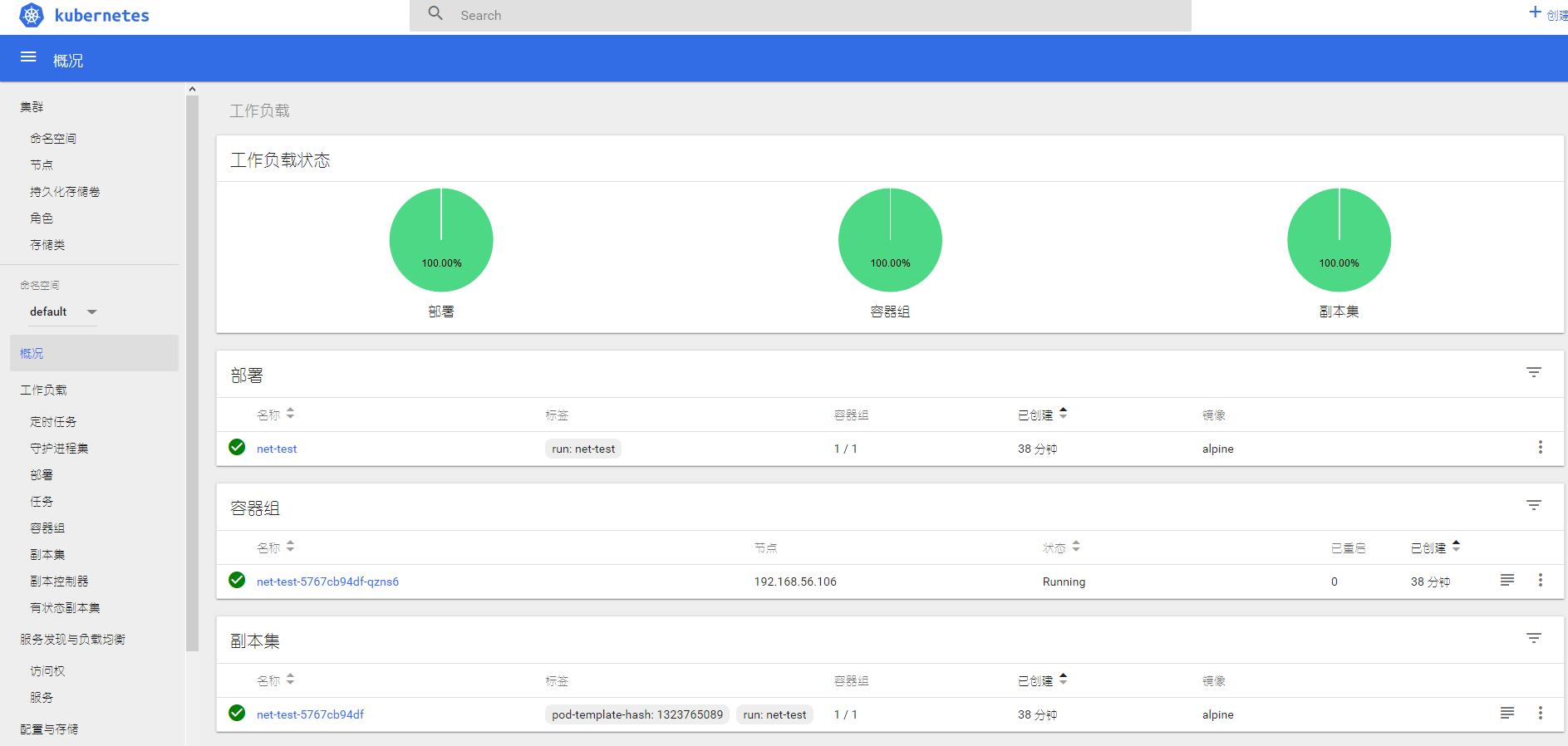
deployment.apps/kubernetes-dashboard 1 1 1 1 3m

NAME DESIRED CURRENT READY AGE

replicaset.apps/kubernetes-dashboard-679b8b8457 1 1 1 3m

### web页面访问

<http://192.168.56.106:30768>



## Heapster

Heapster 是 Kubernetes 社区维护的容器集群监控与效能分析工具。Heapster 会从 Kubernetes apiserver 取得所有 Node 信息，然后再通过这些 Node 来取得 kubelet 上的资料，最后再将所有收集到资料送到 Heapster 的后台储存 InfluxDB，最后利用 Grafana 来抓取 InfluxDB 的资料源来进行视觉化。

kubectl apply -f "https://kairen.github.io/files/manual-v1.10/addon/kube-monitor.yml.conf"

$ kubectl -n kube-system get po,svc

## Ingress Controller

Ingress是利用 Nginx 或 HAProxy 等负载平衡器来曝露集群内服务的元件，Ingress 主要通过设定 Ingress 规格来定义 Domain Name 映射 Kubernetes 内部 Service，这种方式可以避免掉使用过多的 NodePort 问题。

## Helm Tiller Server

Helm 是 Kubernetes Chart 的管理工具，Kubernetes Chart 是一套预先组态的 Kubernetes 资源套件。其中Tiller Server主要负责接收来至 Client 的指令，并通过 kube-apiserver 与 Kubernetes 集群做沟通，根据 Chart 定义的内容，来产生与管理各种对应 API 物件的 Kubernetes 部署文档(又称为 Release)。

## Prometheus