<https://www.bilibili.com/video/BV1QZ4y1V7M5?p=12>

设置时间：timedatectl set-timezone Asia/Shanghai

master：192.168.1.100

node1：192.168.1.101

node2：192.168.1.102

1、vim /etc/sysconfig/network-scripts/ifcfg-ens33

TYPE="Ethernet"

PROXY\_METHOD="none"

BROWSER\_ONLY="no"

BOOTPROTO="static"

DEFROUTE="yes"

GATEWAY="192.168.1.1"

IPADDR="192.168.1.100"

IPV4\_FAILURE\_FATAL="no"

IPV6INIT="yes"

IPV6\_AUTOCONF="yes"

IPV6\_DEFROUTE="yes"

IPV6\_FAILURE\_FATAL="no"

IPV6\_ADDR\_GEN\_MODE="stable-privacy"

NAME="ens33"

UUID="280efcdf-7488-416f-b59c-f10fd5357abb"

DEVICE="ens33"

DNS1=8.8.8.8

DNS2=8.8.4.4

ONBOOT="yes"

2、service network restart

3、wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo

4、关闭selinux

setenforce 0

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

1. 关闭swap分区或禁用swap文件

# 注释掉关于swap分区的行

swapoff -a

yes | cp /etc/fstab /etc/fstab\_bak

cat /etc/fstab\_bak |grep -v swap > /etc/fstab

6、关闭防火墙

systemctl stop firewalld

systemctl disable firewalld

7、

systemctl start NetworkManager.service

systemctl disable NetworkManager.service

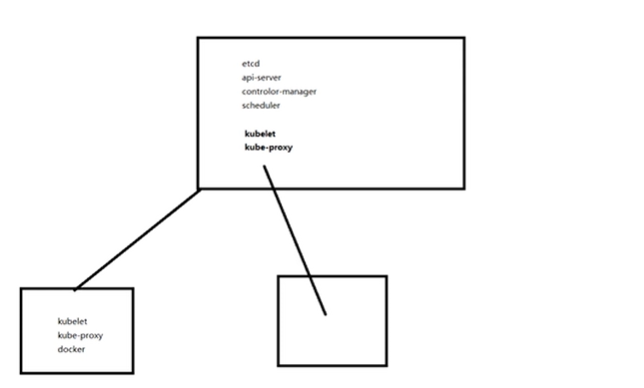
8、

systemctl stop postfix.service

systemctl disable postfix.service

9、

yum install -y net-tools vim lrzsz wget tree screen lsof tcpdump



10、

vim /etc/hosts

192.168.1.100 k8s-master

192.168.1.101 k8s-node1

192.168.1.102 k8s-node2

11、

scp -rp /etc/hosts 192.168.1.101:/etc/hosts

scp -rp /etc/hosts 192.168.1.102:/etc/hosts

1. 先安装master节点：

(1)、装etcd

Yum install etcd -y

vim /etc/etcd/etcd.conf

修改第6行:ETCD\_LISTEN\_CLIENT\_URLS="http://0.0.0.0:2379"

修改第21行: ETCD\_ADVERTISE\_CLIENT\_URLS="http://192.168.1.100:2379"

systemctl start etcd.service

systemctl enable etcd.service

netstat -nltp 监听两个端口 第一个：0.0.0.0:2379 2379是对外提供服务用的,往etcd里面写数据用的这个端口

第二个：127.0.0.1:2380 etcd集群之间相互数据同步用2380

测试：

etcdctl set testdir/testkey0 0

etcdctl get testdir/testkey0

etcdctl -C http://192.168.1.100:2379 cluster-health

配置yum源(自带的kubernetes版本太低)

rm -rf /etc/yum.repos.d/\*  
 cd /etc/yum.repo/  
 wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo  
 wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo

vim k8s.repo

[kubernetes]

name=Kubernetes

baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64/

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

下一步：

yum clean all && yum makecache

安装master节点

yum install kubernetes-master.x86\_64 -y

[root@holly ~]# ll /etc/kubernetes/

total 16

-rw-r--r-- 1 root root 767 Jul 3 2017 apiserver

-rw-r--r-- 1 root root 655 Jul 3 2017 config

-rw-r--r-- 1 root root 189 Jul 3 2017 controller-manager

-rw-r--r-- 1 root root 111 Jul 3 2017 scheduler

vim /etc/kubernetes/apiserver

配置8、11、14、17行

8 、KUBE\_API\_ADDRESS="--insecure-bind-address=0.0.0.0"

11、 KUBE\_API\_PORT="--port=8080"

14、 KUBELET\_PORT="--kubelet-port=10250"

17 、KUBE\_ETCD\_SERVERS="--etcd-servers=http://192.168.1.100:2379"

最后一行KUBE\_ADMISSION\_CONTROL 去掉ServiceAccount

vim /etc/kubernetes/config

配置最后一行，controller-manager, scheduler、proxy同用一个配置文件config

# How the controller-manager, scheduler, and proxy find the apiserver

KUBE\_MASTER="--master=http://192.168.1.100:8080"

systemctl start kube-apiserver.service

systemctl start kube-scheduler.service

systemctl start kube-controller-manager.service

systemctl enable kube-apiserver.service

systemctl enable kube-scheduler.service

systemctl enable kube-controller-manager.service

检测健康状态

[root@holly kubernetes]# kubectl get componentstatus

NAME STATUS MESSAGE ERROR

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

scheduler Healthy ok

controller-manager Healthy ok

13、安装node节点。

Master节点也做node节点，先安装master节点上的

yum install kubernetes-node.x86\_64 -y

vim /etc/kubernetes/kubelet

修改第5、8、11、14行。在master节点上还要修改config文件，但是在安装master的时候已经修改了，后面配置的参数与之相对应，api-server在master上,写master地址

1. KUBELET\_ADDRESS="--address=192.168.1.100"

8、 KUBELET\_PORT="--port=10250"

11、KUBELET\_HOSTNAME="--hostname-override=k8s-master" //k8s-master是主机名

14、KUBELET\_API\_SERVER="--api-servers=http://192.168.1.100:8080"

启动kubelet和kube-proxy。启动kubelet会自动启动docker

[root@holly kubernetes]# systemctl start kubelet.service

[root@holly kubernetes]# systemctl status docker

[root@holly kubernetes]# systemctl enable kubelet.service

[root@holly kubernetes]# systemctl start kube-proxy.service

[root@holly kubernetes]# systemctl enable kube-proxy.service

[root@holly kubernetes]# kubectl get nodes

NAME STATUS AGE6

k8s-master Ready 6m

配置node1和node2

vim /etc/kubernetes/config

最后一行：KUBE\_MASTER="--master=http://192.168.1.100:8080"

vim /etc/kubernetes/kubelet

修改第5、8、11、14行。在master节点上还要修改config文件，但是在安装master的时候已经修改了，后面配置的参数与之相对应，api-server在master上,写master地址

1. KUBELET\_ADDRESS="--address=192.168.1.101" //node2：192.168.1.101

8、 KUBELET\_PORT="--port=10250"

11、KUBELET\_HOSTNAME="--hostname-override=k8s-node1"//node2：k8s-node2都是主机名，修改主机名：vi /etc/hostname或者hostnamectl set-hostname主机名 重启

14、KUBELET\_API\_SERVER="--api-servers=http://192.168.1.100:8080"

启动kubelet和kube-proxy。启动kubelet会自动启动docker

[root@holly kubernetes]# systemctl start kubelet.service

[root@holly kubernetes]# systemctl status docker

[root@holly kubernetes]# systemctl enable kubelet.service

[root@holly kubernetes]# systemctl start kube-proxy.service

[root@holly kubernetes]# systemctl enable kube-proxy.service

删除node节点

yum -y install kubeadm-1.15.1

14、安装网络服务，让不同节点上的docker能通讯

三台都安装

yum install flannel -y

配置master节点：

vim /etc/sysconfig/flanneld

FLANNEL\_ETCD\_ENDPOINTS="http://192.168.1.100:2379" //flannel连接etcd的地址

还要配置一个key,是flannel查询etcd用的，每个容器地址的分配都存储在etcd中

FLANNEL\_ETCD\_PREFIX="/atomic.io/network" //默认这个key 不改

需要在etcd里面创建这个key，k8s集群中ip地址的范围【172.17.0.0/16是ifconfig内网】

etcdctl set /atomic.io/network/config '{"Network":"172.17.0.0/16"}'

systemctl start flanneld.service

systemctl enable flanneld.service

systemctl restart docker

Ifconfig //重启后生效，docker网段跟它一致了

配置node节点：（node1和node2都一样）

vim /etc/sysconfig/flanneld

FLANNEL\_ETCD\_ENDPOINTS="http://192.168.1.100:2379"

systemctl start flanneld.service

systemctl enable flanneld.service

systemctl restart docker

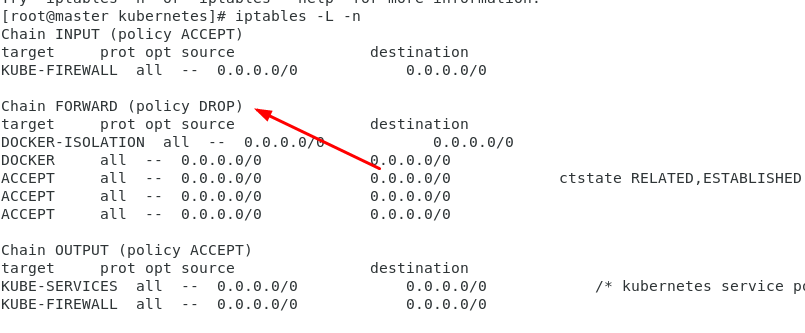
测试容器之间是否通讯[三台同时操作]：上传镜像包： docker-busybox.tar.gz 然后

docker load -i docker-busybox.tar.gz

docker image 查看镜像

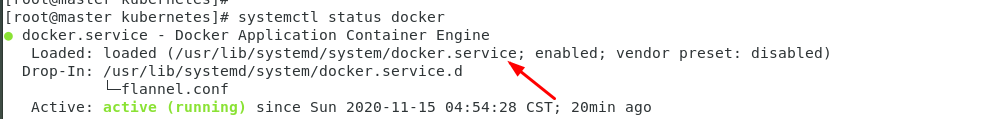
docker run -it busybox 进入容器 然后ifconfig

相互ping一下，ping不通，因为装的docker版本是1.13, 防火墙规则调了，



三台都执行：

iptables -P FORWARD ACCEPT 但是断电后会重新加，解决方法：



[root@node1 kubernetes]# which iptables

/usr/sbin/iptables

vim /usr/lib/systemd/system/docker.service //加在18行

ExecStartPost=/usr/sbin/iptables -P FORWARD ACCEPT

[root@node1 kubernetes]# systemctl daemon-reload

在master上

K8s是通过创建pod来运行容器

cd ~

mkdir pod

mkdir k8s

cd pod

vi nginx\_pod.yaml

kubectl create -f nginx\_pod.yaml

vim nginx\_pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx

labels:

app: web

spec:

containers:

- name: nginx

image: nginx:1.13

ports:

- containerPort: 80

kubectl get node

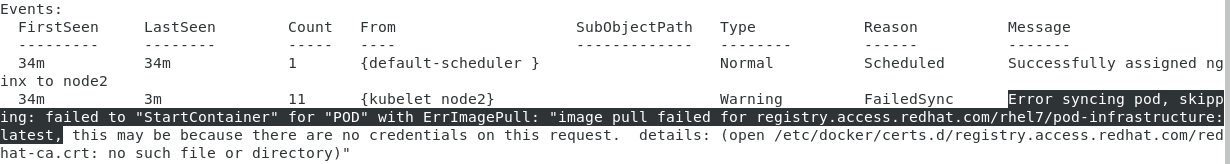
kubectl create -f nginx\_pod.yaml

kubectl get pod nginx

kubectl get componentstatus

kubectl get pod nginx -o wide

kubectl describe pod nginx



解决办法：【原因是证书错误，pod调度到node2上，查看kubelet配置文件：# pod infrastructure container KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest" 这里配置的问题，在node2上修改为：】

Node2修改

vim /etc/kubernetes/kubelet【pull 一个pod的基础镜像 pod-infrastructure】

# pod infrastructure container

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=docker.io/tianyebj/pod-infrastructure:latest"

systemctl restart kubelet.service

万一镜像下载不下来，需要配置docker加速

vim /etc/sysconfig/docker //注释OPTIONS

OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false --registry-mirror=https://registry.docker-cn.com --insecure-registry=192.168.1.100:5000'

systemctl restart docker

Master上：

kubectl delete pod nginx

kubectl get pod -o wide

kubectl create -f nginx\_pod.yaml

kubectl get pod -o wide

做私有仓库

Master上

docker pull registry

docker run -d -p 5000:5000 --restart=always --name registry -v /opt/myregistry:/var/lib/registry registry

安装了Node都需要改，意味着3个都改

vim /etc/sysconfig/docker //注释OPTIONS

OPTIONS='--selinux-enabled --log-driver=journald --signature-verification=false --registry-mirror=https://registry.docker-cn.com --insecure-registry=192.168.1.100:5000'

systemctl restart docker

docker tag docker.io/tianyebj/pod-infrastructure:latest 192.168.1.100:5000/pod-infrastructure:latest

docker push 192.168.1.100:5000/pod-infrastructure:latest

为了能pull镜像 需要改

vim /etc/kubernetes/kubelet

KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=192.168.1.100:5000/pod-infrastructure:latest"

systemctl restart kubelet

docker tag docker.io/nginx:1.13 192.168.1.100:5000/nginx:1.13

docker push 192.168.1.100:5000/nginx:1.13

master上

vim /root/pod/nginx\_1.14.yaml

apiVersion: v1

kind: Pod

metadata:

name: test

labels:

app: web

spec:

containers:

- name: nginx

image: 192.168.1.100:5000/nginx:1.13

ports:

- containerPort: 80

//kubectl create -f nginx\_1.14.yaml 更新资源

[root@master pod]# kubectl create -f nginx\_1.14.yaml

pod "test" created

[root@master pod]# [root@master pod]# kubectl get pod

NAME READY STATUS RESTARTS AGE

nginx 1/1 Running 5 21h

test 1/1 Running 0 3s

如果status哪里显示错误，检查5000端口是否开启，如果没有，docker ps -a

docker rm 镜像id 然后重启仓库：docker run -d -p 5000:5000 --restart=always --name registry -v /opt/myregistry:/var/lib/registry registry

[root@node1 ~]# docker ps

PORTS NAMES

efe552570a92 nginx:1.13

9286644985c6 192.168.1.100:5000/pod-infrastructure:latest

[root@node1 ~]# docker inspect efe552570a92|grep -i ipaddress

"SecondaryIPAddresses": null,

"IPAddress": "",

[root@node1 ~]# docker inspect efe552570a92|grep -i network

"NetworkMode": "container:9286644985c6215b0a0faa350afca83b04f54967cf222b300299596a70102103",

"NetworkSettings": {

"Networks": {}

POD容器和nginx共用一个ip地址，k8s中创建了一个pod资源，控制docker启动2个容器，业务容器nginx，基础容器pod？ K8s 功能走pod容器，一般最多启动四个业务程序，1个基础pod容器

强制删除pod

kubectl delete pod pod名字 --force --grace-period=0

Replication Controller得作用

应用通过k8s跑起来后，k8s要保证这个容器持续运行，就是高可用，RC就能确保容器得高可用，rc得原理是 他会一直监控k8s得容器，也就是pod资源得运行状态 ，如果发现有异常了，rc就会控制，k8s在其他得node节点上启动一个新得pod，来保业务得高可用运行。rc还提供了滚动升级，升级回滚，或者发布新版本

template里面没声明名字，因为两个pod名字不能重复得，rc会随机给他生成名字

vim /rc/nginx-rc.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: myweb

spec:

replicas: 2

selector:

app: myweb

template:

metadata:

labels:

app: myweb

spec:

containers:

- name: myweb

image: 192.168.1.100:5000/nginx:1.13

ports:

- containerPort: 80

[root@master rc]# kubectl create -f nginx-rc.yaml

replicationcontroller "myweb" created

[root@master rc]# kubectl get rc

NAME DESIRED CURRENT READY AGE

myweb 2 2 2 1m

[root@master rc]# kubectl get pod

NAME READY STATUS RESTARTS AGE

myweb-jlgtt 1/1 Running 0 2m

myweb-w2s2j 1/1 Running 0 2m

nginx 1/1 Running 0 1d

test 1/1 Running 0 22h

//删除个pod，他会里面启动一个新得，因为定义了2个rc，他会时刻监控pod得状态，少了就启用新得，多了就删除，始终保证配置文件里面的数量2

[root@master rc]# kubectl delete pod myweb-jlgtt

pod "myweb-jlgtt" deleted

[root@master rc]# kubectl get pod

NAME READY STATUS RESTARTS AGE

myweb-w2s2j 1/1 Running 0  **3m**

**myweb-zjgfg** 1/1 Running 0 **2s**

**nginx** 1/1 Running 0 **1d**

test 1/1 Running 0 22h

rc定义的选择器selector的标签是lables，每个pod启动他会自动加上标签myweb，rc就会根据标签来选择pod是归我管理的

[root@master rc]# kubectl edit pod nginx

apiVersion: v1

kind: Pod

metadata:

creationTimestamp: 2020-11-15T06:53:45Z

**labels:**

**app: myweb**

name: nginx

........

pod "nginx" edited

[root@master rc]# kubectl get pod

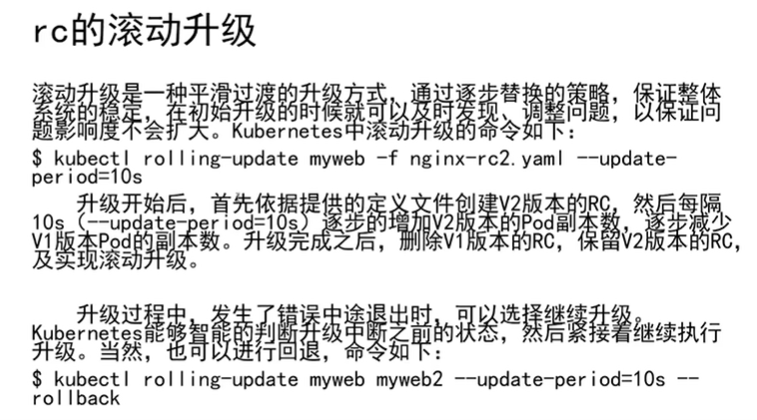
NAME READY STATUS RESTARTS AGE

**myweb-w2s2j** 1/1 Running 0 **15m**

**nginx**  1/1 Running 0 **1d**

test 1/1 Running 0 22h

**rc发现了有三个pod，因为三个的label都是myweb，他会干掉一个，因为配置文件里面是2个。那么他会删除最年轻的，启动时间最少的那个pod是 myweb-zjgfg。Rc是通过标签来关联pod**



[root@master rc]# cp nginx-rc.yaml nginx-rc2.yaml

:%s#myweb#myweb2#g 把标签myweb全部改成myweb2，

image: 192.168.1.100:5000/nginx:1.15 镜像版本从1.13改成1.15

[root@node1 ~]# docker tag docker.io/nginx:1.13 192.168.1.100:5000/nginx:1.15

[root@node1 ~]# docker push 192.168.1.100:5000/nginx:1.15

**升级版本**

**[root@master rc]# kubectl rolling-update myweb -f nginx-rc2.yaml --update-period=30s**

Created mysweb2

Scaling up mysweb2 from 0 to 2, scaling down myweb from 2 to 0 (keep 2 pods available, don't exceed 3 pods)

Scaling mysweb2 up to 1

Scaling myweb down to 1

Scaling mysweb2 up to 2

Scaling myweb down to 0

Update succeeded. Deleting myweb

replicationcontroller "myweb" rolling updated to "mysweb2"

**[root@master rc]# kubectl get pod**

**NAME READY STATUS RESTARTS AGE**

**mysweb2-1tkp2 1/1 Running 0 1m**

**mysweb2-n7tht 1/1 Running 0 1m**

**test 1/1 Running 0 23h**

**一旦myweb2存活了30秒后就把之前的删掉。创建了2个myweb2**

**回滚设置**

**[root@master rc]# kubectl rolling-update myweb2 -f nginx-rc.yaml --update-period=10s**

Created myweb

Scaling up myweb from 0 to 2, scaling down mysweb2 from 2 to 0 (keep 2 pods available, don't exceed 3 pods)

Scaling myweb up to 1

Scaling mysweb2 down to 1

Scaling myweb up to 2

Scaling mysweb2 down to 0

Update succeeded. Deleting mysweb2

replicationcontroller "mysweb2" rolling updated to "myweb"

**[root@master rc]# kubectl get pod**

NAME READY STATUS RESTARTS AGE

**myweb-9rrx0**  1/1 Running 0 27s

**myweb-gxj7d** 1/1 Running 0 17s

test 1/1 Running 0 23h

如果在升级的过程中报错了

**[root@master rc]# kubectl rolling-update myweb -f nginx-rc2.yaml --update-period=50s**

比如myweb升级到myweb2，升级中报错了，ctrl+z中断升级

kubectl get pod两个myweb 一个myweb2.然后需要回滚

**[root@master rc]# kubectl rolling-update myweb myweb2 --rollback**

**会自动把myweb2删掉**

**为什么k8s要引入service资源**

**想要被外界访问。docker容器中做端口映射才能被访问，那么运行在k8s中的容器，为什么不能之间为它做端口映射？**

**pod用rc实现了高可用，假如pod死掉了，他会去启动一个新得pod，那新得pod的ip地址就不一样了，所以k8s引入了一个新的资源service。**

**他创建了clusterIP，ip是固定的。pod启动起来了会自动加入clusterIP，service提供了一个负载均衡的功能。rr轮询。k8s内部才能访问。**

**podip cluster ip nodeip**

[root@master rc]# kubectl get all

NAME DESIRED CURRENT READY AGE

rc/myweb 2 2 2 57m

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

svc/kubernetes **10.254.0.1**  <none> 443/TCP 2d

NAME READY STATUS RESTARTS AGE

po/myweb-9rrx0 1/1 Running 0 57m

po/myweb-gxj7d 1/1 Running 0 57m

po/test 1/1 Running 0 1d

创建service资源

[root@master svc]# vim /svc/nginx-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: myweb

spec:

type: NodePort

ports:

- port: 80 //**clusterIP端口**

nodePort: 30000 //**node宿主机端口**

targetPort: 80 //**pod端口**

selector:

app: myweb

[root@master svc]# kubectl create -f nginx-svc.yaml

service "myweb" created

[root@master svc]# kubectl describe svc myweb

Name: myweb

Namespace: default

Labels: <none>

Selector: app=myweb

Type: NodePort

IP: 10.254.244.32

Port: <unset> 80/TCP //**clusterIP端口**

NodePort: <unset> 30000/TCP //**宿主机端口**

Endpoints: 172.17.20.4:80,172.17.74.2:80 //pod的80

Session Affinity: None

No events.

**端口服务是由kube-proxy.service 创建的，每个node节点上都有.现在使用任意node的ip加30000端口可以访问页面。例如：192.168.1.101:30000,出来nginx欢迎页面**

**增加副本，service服务的自动发现，pod起来了会自动加入到service，然后接受调度**

[root@master svc]# kubectl scale rc myweb --replicas=3

replicationcontroller "myweb" scaled

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

NAME READY STATUS RESTARTS AGE IP NODE

myweb-9v1gr 1/1 Running 0 3s 172.17.74.3 node1

myweb-g6fvr 1/1 Running 0 7m 172.17.20.3 master

myweb-znnbc 1/1 Running 0 7m 172.17.89.3 node2

test 1/1 Running 0 1d 172.17.89.2 node2

[root@master svc]# kubectl describe svc myweb

Name: myweb

Namespace: default

Labels: <none>

Selector: app=myweb

Type: NodePort

IP: 10.254.244.32

Port: <unset> 80/TCP

NodePort: <unset> 30000/TCP

Endpoints: 172.17.20.3:80,172.17.74.3:80,172.17.89.3:80

Session Affinity: None

No events.

**Service的负载均衡**

[root@master svc]# echo "node1" >index.html

[root@master svc]# kubectl cp index.html myweb-9v1gr:/usr/share/nginx/html/index.html

Error from server: error dialing backend: dial tcp 192.168.1.101:10250: getsockopt: no route to host //检查每个node节点上 防火墙是否关闭了。

[root@master svc]# echo "node2" >index.html

[root@master svc]# kubectl cp index.html myweb-9v1gr:/usr/share/nginx/html/index.html

[root@master svc]# echo "node3" >index.html

[root@master svc]# kubectl cp index.html myweb-9v1gr:/usr/share/nginx/html/index.html

**现在使用任意node的ip加30000端口可以访问页面。例如：192.168.1.101:30000,出来node1，等一会儿他会重新调度到node2或者master，若调度到node2，显示node2.所以实现了负载均衡**

NodePort的范围是30000-32367，但是可以改。在master上修改

[root@master svc]# vim /etc/kubernetes/apiserver

KUBE\_API\_ARGS="--service-node-port-range=10000-60000"

//apiservice的参数

[root@master svc]# vim /usr/lib/systemd/system/kube-apiserver.service

systemctl restart kube-apiserver.service

修改svc.yaml

vim /root/svc/nginx-svc.yaml

**nodePort: 20000**

【注：KUBE\_SERVICE\_ADDRESSES="--service-cluster-ip-range=10.254.0.0/16" cluster-ip也可以在/etc/kubernetes/apiserve中修改，然后重启apiservice】

[root@master svc]# kubectl apply -f nginx-svc.yaml

service "myweb" configured

[root@master svc]# kubectl describe svc myweb

......

Port: <unset> 80/TCP

NodePort: <unset> **20000/TCP**

Endpoints: 172.17.20.3:80,172.17.74.3:80,172.17.89.3:80

Session Affinity: None

No events.

例如：

[root@master svc]# curl 192.168.1.100:20000

node3

[root@master svc]# curl 192.168.1.100:20000

node2

**Deployment资源**

deployment也是保证pod高可用的一种方式。已经有了RC，为啥还需要引入deployment

因为他解决了RC的一个缺点

[root@master rc]# kubectl get all -o wide

NAME DESIRED CURRENT READY AGE CONTAINER(S) IMAGE(S) SELECTOR

rc/myweb 3 3 3 2h myweb **192.168.1.100:5000/nginx:1.13** app=myweb

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

svc/kubernetes 10.254.0.1 <none> 443/TCP 2d <none>

svc/myweb 10.254.244.32 <nodes> 80:20000/TCP 1h app=myweb

......

[root@master rc]# kubectl rolling-update myweb -f nginx-rc2.yaml --update-period=1s

....

replicationcontroller "myweb" rolling updated to "mysweb2"

...

[root@master rc]# curl 192.168.1.100:20000 //升级后就不能访问了

因为升级后**rc的标签是mysweb2 ，而svc的标签还是myweb**

[root@master rc]# kubectl get all -o wide

NAME DESIRED CURRENT READY AGE CONTAINER(S) IMAGE(S) SELECTOR

rc/mysweb2 2 2 2 7m  **mysweb2**  **192.168.1.100:5000/nginx:1.15** **app=mysweb2**

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

svc/kubernetes 10.254.0.1 <none> 443/TCP 2d <none>

svc/myweb 10.254.244.32 <nodes> 80:20000/TCP 1h **app=myweb**

....

[root@master rc]# kubectl describe svc myweb

。。。

Selector: app=myweb

Type: NodePort

IP: 10.254.244.32

Port: <unset> 80/TCP

NodePort: <unset> 20000/TCP

**Endpoints: <none> //这里没关联的node节点**

Session Affinity: None

No events.

//修改svc的标签为myweb2

[root@master rc]# kubectl edit svc

selector:

**app: myweb2**

[root@master rc]# kubectl describe svc myweb //这样就有node节点，也能访问的通了

删除rc和svc

[root@master rc]# kubectl delete rc myweb2

[root@master rc]# kubectl delete svc myweb2

[root@master ~]# vim /root/deployment/nginx-deploy.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-deployment

spec:

replicas: 3

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: 192.168.1.100:5000/nginx:1.13

ports:

- containerPort: 80

[root@master deployment]# kubectl create -f nginx-deploy.yaml

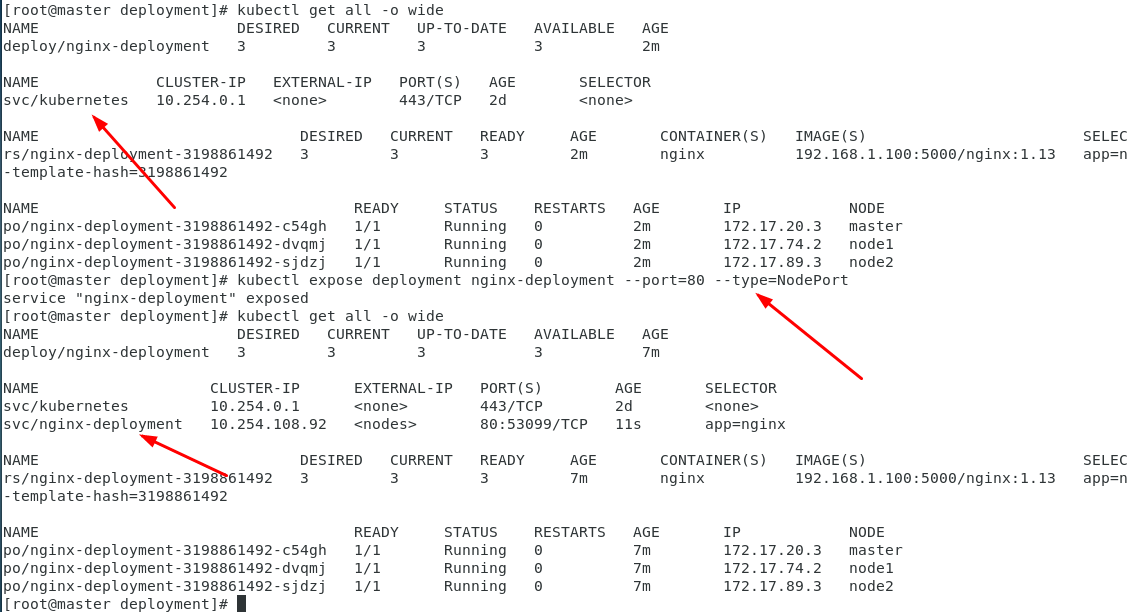
deployment "nginx-deployment" created

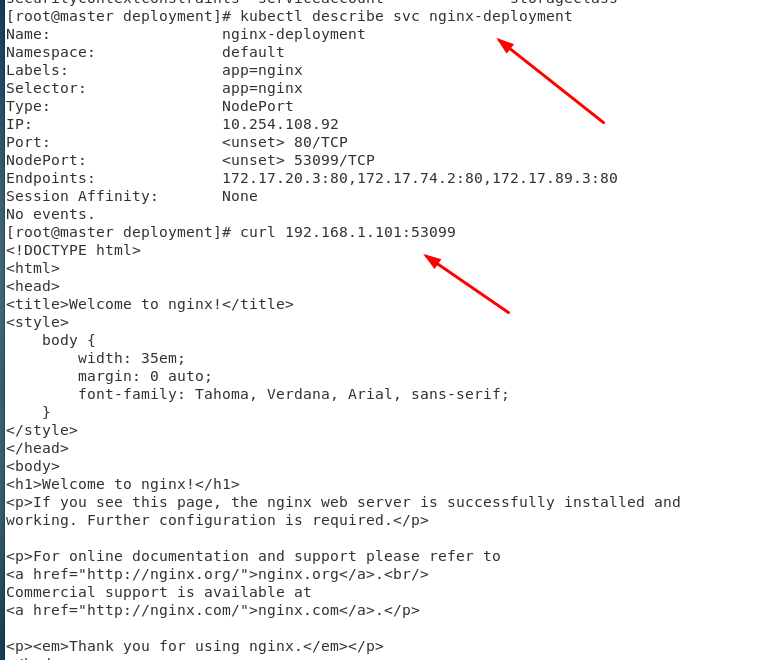
deployment先去启动rs，rs去启动pod，rs有rc的90%的功能

关联service

kubectl expose deployment nginx-deployment --port=80 --type=NodePort

expose 做端口映射，nginx-deployment是deployment名字，cluster-ip是80，端口类型，用node端口





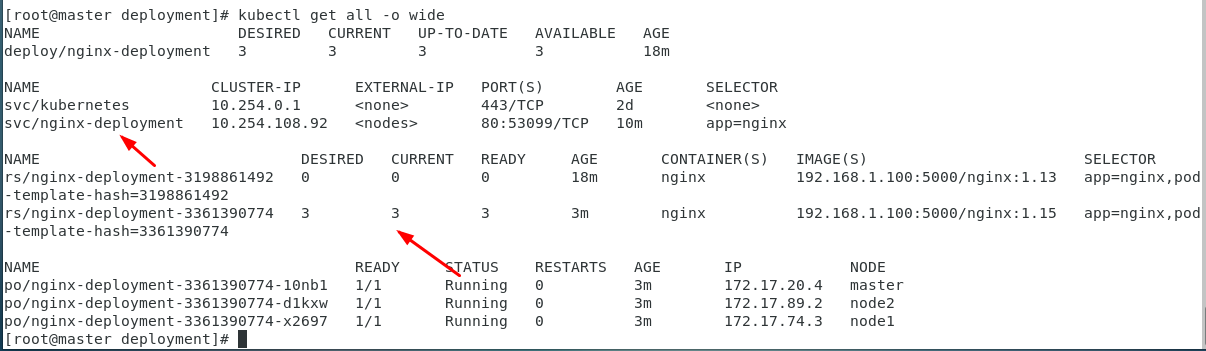
[root@master deployment]# kubectl edit deployment nginx-deployment

升级版本1.15,

spec:

containers:

- image: 192.168.1.100:5000/nginx:1.15



回滚到1.13版本。秒回滚

kubectl rollout undo deployment nginx-deployment