**Kubernetes安装部署操作手册**

|  |  |
| --- | --- |
| **作者姓名：** | **王杉** |
| **项目组：** | **联海创智** |
| **制作日期：** | **2019年01月07日** |

**Kubernetes安装部署操作手册**

**摘要**

Kubernetes单词起源于希腊语, 是“舵手”或者“领航员”的意思，是“管理者”和“控制论”的根源。 K8s是把用8代替8个字符“ubernete”而成的缩写。首先，他是一个全新的基于容器技术的分布式架构领先方案。Kubernetes(k8s)是Google开源的容器集群管理系统（谷歌内部:Borg）。在Docker技术的基础上，为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。

　　Kubernetes是一个完备的分布式系统支撑平台，具有完备的集群管理能力，多扩多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制、內建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制以及多粒度的资源配额管理能力。

**关键词：**Kubernetes；自我修复；滚动升级；动态伸缩

**目录**

总页数：19页

[1. 初识kubernetes 1](#_Toc26569)

[1.1kubernetes简介 1](#_Toc21070)

[1.2kubernetes特征 1](#_Toc27669)

[1.3核心组件 1](#_Toc14636)

[1.4拓扑图 1](#_Toc20309)

[2. 集群 2](#_Toc28287)

[2.1环境介绍 2](#_Toc7742)

[2.2安装必要的组件 2](#_Toc4286)

[2.2.1安装docker 2](#_Toc30647)

[2.2.2安装集群组件 3](#_Toc31827)

[2.2使用kubeadm初始化集群 4](#_Toc8986)

[2.3加入集群 4](#_Toc11616)

[3. 验证 5](#_Toc32111)

[3.1验证集群节点 5](#_Toc16531)

[3.2验证pod信息 5](#_Toc7519)

[4. 部署 5](#_Toc5058)

[4.1 deployment 5](#_Toc19450)

[4.2 Service 7](#_Toc19482)

[5. 升级 8](#_Toc8332)

[5.1查看当前镜像版本 8](#_Toc30513)

[5.2更新镜像 8](#_Toc16388)

[6. 回滚 9](#_Toc25911)

[6.1查看历史版本 9](#_Toc1221)

[6.2回滚版本 9](#_Toc3608)

[7. Dashborad 10](#_Toc14810)

[7.1镜像准备 10](#_Toc8275)

[7.2编写dashborad.yaml 10](#_Toc11486)

[7.3创建和修改 15](#_Toc4039)

[7.4创建ServiceAccount 16](#_Toc13342)

[7.5登录UI 17](#_Toc27604)

# 初识kubernetes

## 1.1kubernetes简介

Kubernetes是一个全新的基于容器技术的分布式架构领先方案。是Google内部集群管理系统Borg的一个开源版本。直到2015年4月，随着论文发布，才被众人熟知。Kubernetes是一个开放的开发平台。不局限于任何一种语言，没有限定任何编程接口。是一个完备的分布式系统支撑平台。它构建在docker之上，提供应用部署、维护、扩展机制等功能，利用Kubernetes能方便地管理跨机器运行容器化的应用。

## 1.2kubernetes特征

1. 自主的管理容器，保证云平台中的容器按照用户的期望状态运行着。
2. 自动扩容，弹性伸缩。
3. 自动监控，删除出故障的应用。
4. 容器编排成组，并提供容器间的负载均衡。
5. 自我调度，自我管理。

## 1.3核心组件

kubectl:客户端命令行工具，作为整个系统的操作入口。

kube-apiserver:以REST API服务形式提供接口，作为整个系统的控制入口。

kube-controller-manager:执行整个系统的后台任务，包括节点状态状况、Pod个数、Pods和Service的关联等。

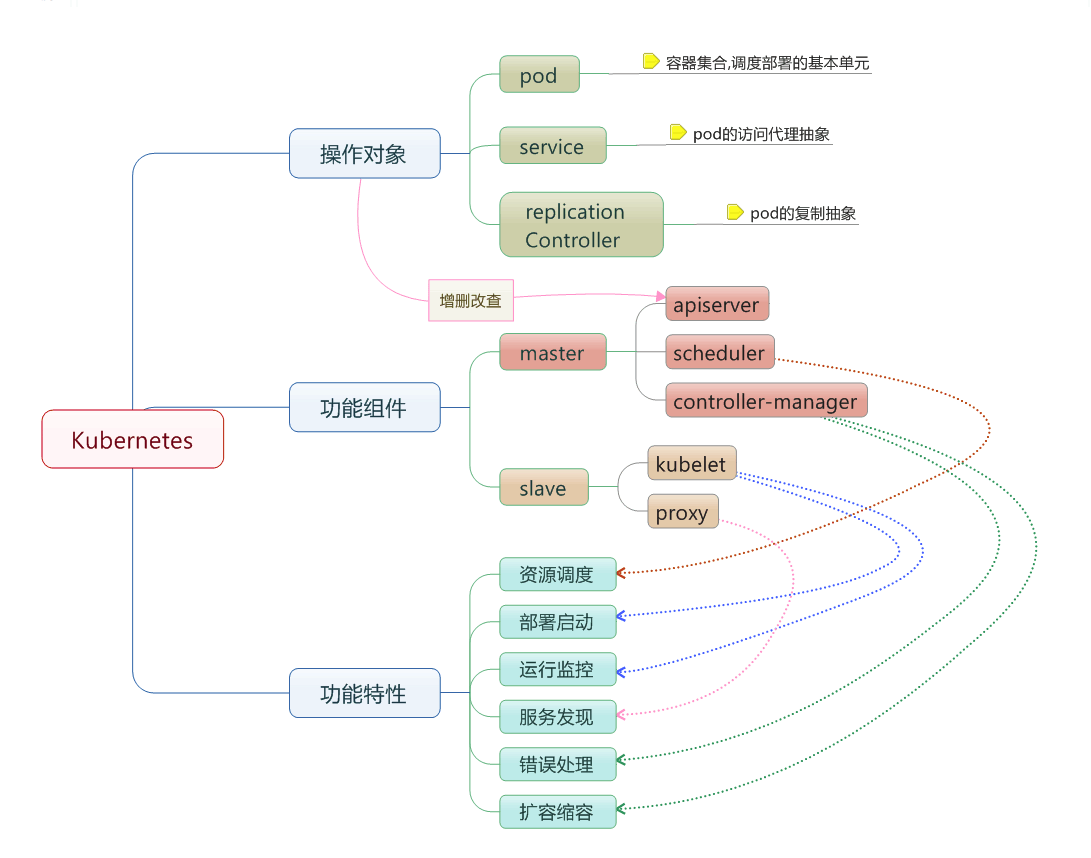
kube-scheduler:负责节点资源管理，接收来自kube-apiserver创建Pods任务，并分配到某个节点。

etcd:负责节点间的服务发现和配置共享。

kube-proxy:运行在每个计算节点上，负责Pod网络代理。定时从etcd获取到service信息来做相应的策略。

kubelet:运行在每个计算节点上，作为agent，接收分配该节点的Pods任务及管理容器，周期性获取容器状态，反馈给kube-apiserver。

## 1.4拓扑图



# 集群

## 2.1环境介绍

192.168.2.5 k8s-master 4G 2核

192.168.2.6 k8s-node 4G 2核

## 2.2安装必要的组件

### 2.2.1安装docker

在每个节点执行

1.添加阿里yum源

sudo yum-config-manager --add-repo <http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo>

2.生成缓存

yum makecache fast

安装docker-ce

yum -y install docker-ce

3.启动docker并设置为开机启动

systemctl start docker && systemctl enable docker

### 2.2.2安装集群组件

每个节点执行

1. 修改hosts文件

vim /etc/hosts 添加以下信息

192.168.2.5 k8s-master

182.168.2.6 k8s-node

2.分别在不同主机上设置hostname

hostnamectl --static set-hostname k8s-master

hostnamectl --static set-hostname k8s-node

3.关掉 selinux

setenforce 0

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

4.关掉防火墙

systemctl stop firewalld

systemctl disable firewalld

5.关闭 swap

swapoff -a

sed -i 's/.\*swap.\*/#&/' /etc/fstab

6.配置转发参数

cat <<EOF > /etc/sysctl.d/k8s.conf net.bridge.bridge-nf-call-ip6tables = 1 net.bridge.bridge-nf-call-iptables = 1 EOF

sysctl --system

7.配置国内yum源并生成缓存

cat <<EOF > /etc/yum.repos.d/kubernetes.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 EOF

yum makecache fast

8.安装kubeadm,kubectl,kubelet,ntpdate

yum install -y kubeadm kubectl kubelet ntpdate

systemctl start kubelet && systemctl enable kubelet

## 2.2使用kubeadm初始化集群

在master上面执行一下命令，初始化集群信息

kubeadm init --image-repository registry.aliyuncs.com/google\_containers --pod-network-cidr=10.244.0.0/16 --service-cidr=10.96.0.0/12

等待初始化完成后，系统提示我们执行一下三步：

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

并且初始化完成后，我们将token拷贝下来保存，后来加入node使用

kubeadm join 192.168.2.5:6443 --token 6pq12p.qolwnkesr8nu3ger --discovery-token-ca-cert-hash sha256:eb1dcad8d31fd03c4cf9ab0476c360da13b2204e01885ca8f98eafd43345ee8d

加入flannel:

kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>

注意：

这里有一个镜像拉取不下来我们手动拉取国内，并给改镜像打一个标签

docker pull registry.cn-hangzhou.aliyuncs.com/kubernetes\_containers/flannel:v0.10.0-amd64

docker tag registry.cn-hangzhou.aliyuncs.com/kubernetes\_containers/flannel:v0.10.0-amd64 quay.io/coreos/flannel:v0.10.0-amd64

## 2.3加入集群

在node上面执行之前生成的token

kubeadm join 192.168.2.5:6443 --token 6pq12p.qolwnkesr8nu3ger --discovery-token-ca-cert-hash sha256:eb1dcad8d31fd03c4cf9ab0476c360da13b2204e01885ca8f98eafd43345ee8d

# 验证

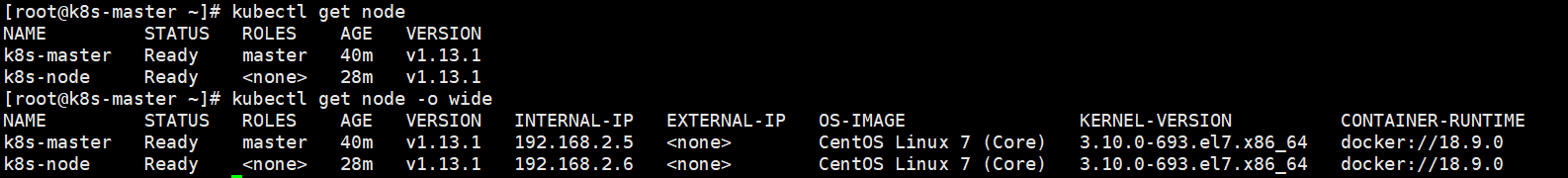
## 3.1验证集群节点

上面执行完命令后，集群不会马上变成ready状态，因为系统需要去下载docker镜像，稍等片刻后我们可以执行一下命令验证集群状态。

Kubectl get node

或者

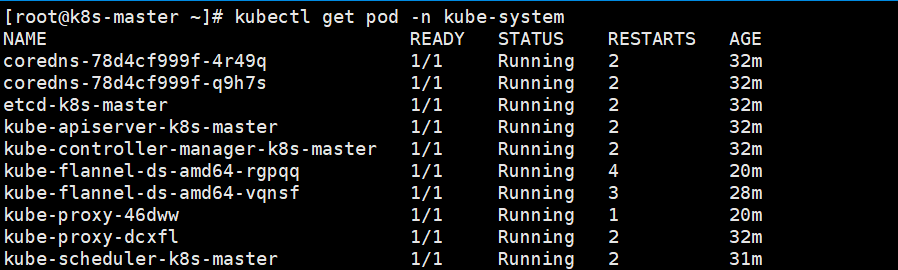
Kubectl get node -o wide显示更新信息



当所有节点都变成ready后，表示集群搭建完成。

## 3.2验证pod信息

kubectl get pod -n kube-system

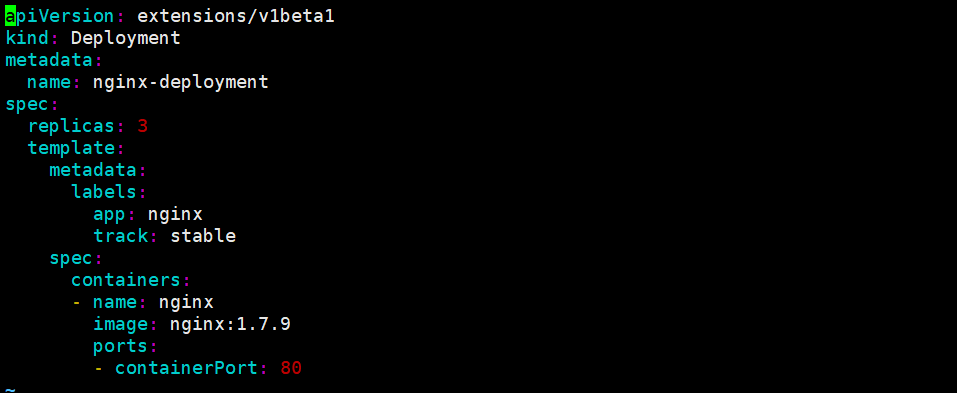


当所有pod的ready都变成1/1就表示集群搭建成功。

# 部署

## 4.1 deployment

一次deployment为一次部署，如图Deployment调用ReplicaSet创建多个Pod副本。而ReplicaSet不需要我们去管理，所以我们只需要创建一个deployment即可。我们编写一个nginx-deployment.yaml。



apiVersion: 组名/版本号

Kind: 类型

Metadata: 元数据，下面name表示当前deployment名称为nginx-deployment

Spec: 规格

Replicas:3表示副本集3个

Spec.template.metadata.labels 自定义标签，一般配合selector

spec.template.containers:可以有多个容器

- name 表示容器[1]的名字

Image: 使用镜像nginx:1.7.9

containerPort: 容器端口80

执行kubectl apply -f nginx-deployment.yaml即可创建deployment

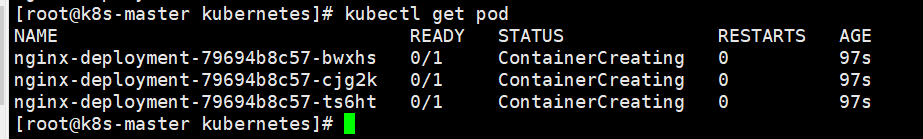
查看deployment:

kubectl get deploy/deployment/deployments

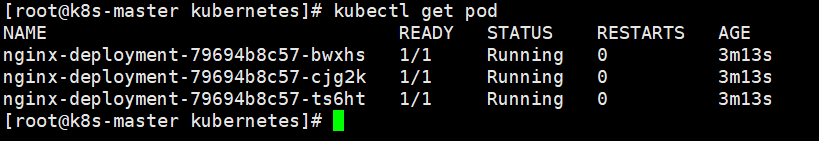
查看pod：

Kubectl get pod

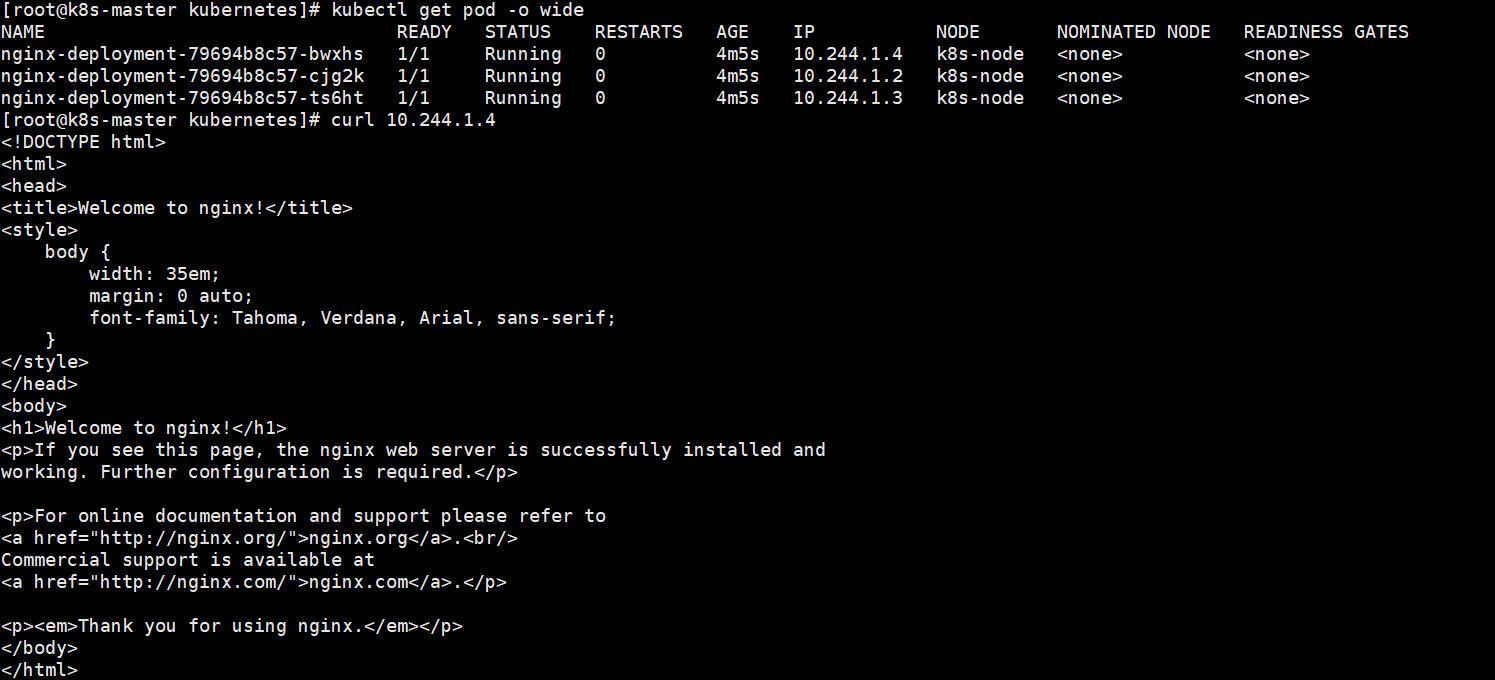
此时pod显示正在创建，因为镜像需要去pull下来。



等待一段时间后，我们的pod已经跑起来了。



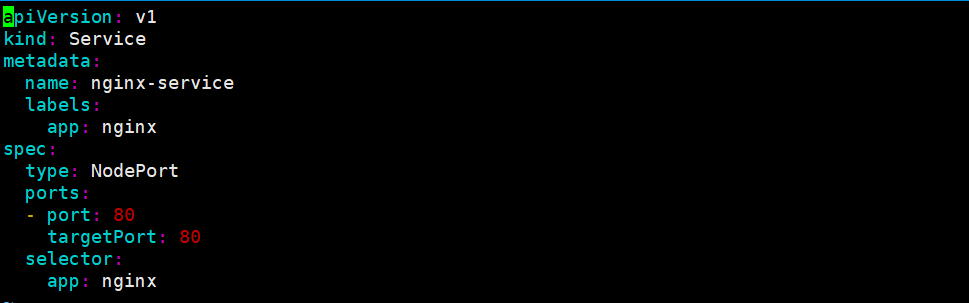
此时我们使用-o wide查看ip,此时的应用只有集群内部能访问，外部暂时不能访问。使用一个ip在任何一个节点都可以访问。



## 4.2 Service

Service定义了一个服务的访问入口地址，前端的应用通过这个入口地址访问其背后的一组由Pod副本组成的集群实例，来自外部的访问请求被负载均衡到后端的各个容器应用上。Service与其后端Pod副本集群之间则是通过Label Selector来实现对接的.

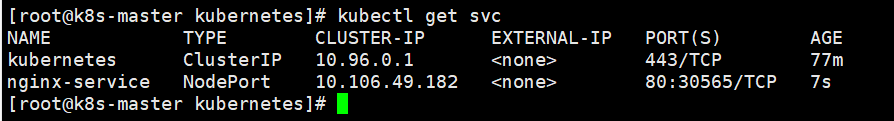
nginx-svc.yaml



这里我们讲一下selector,选择的是之前deployment，labels定义的标签。

创建Service:kubectl apply -f nginx-svc.yaml

查看service:kubectl get svc/service



因为我们的type指定的是NodePort所以我们现在可以直接在浏览器使用任意节点ip+30565访问。并且我们指定的pod有3个，他每次会根据负载策略去访问三个pod中的任意一个。



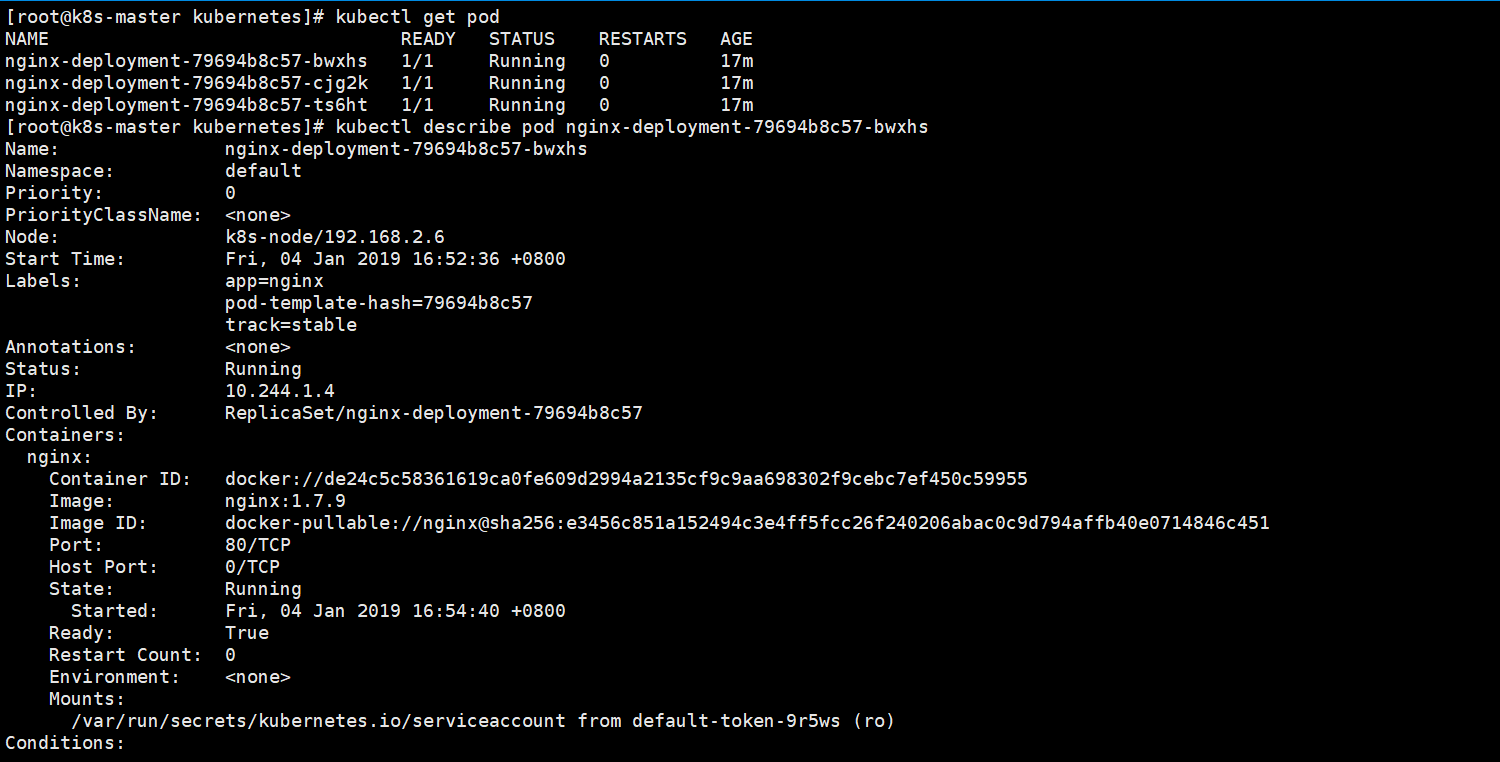
# 升级

## 5.1查看当前镜像版本

Kubectl get pod

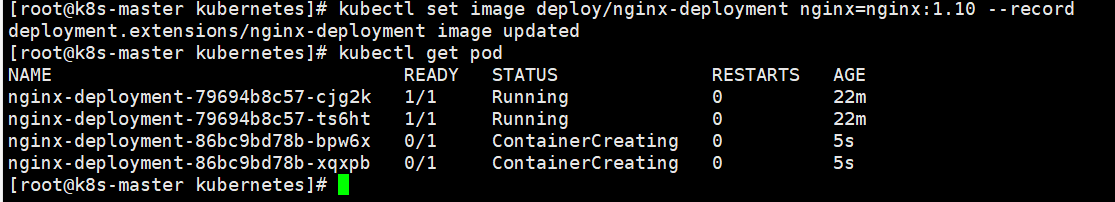
Kubectl describe pod podName

看了看到image: nginx:1.7.9

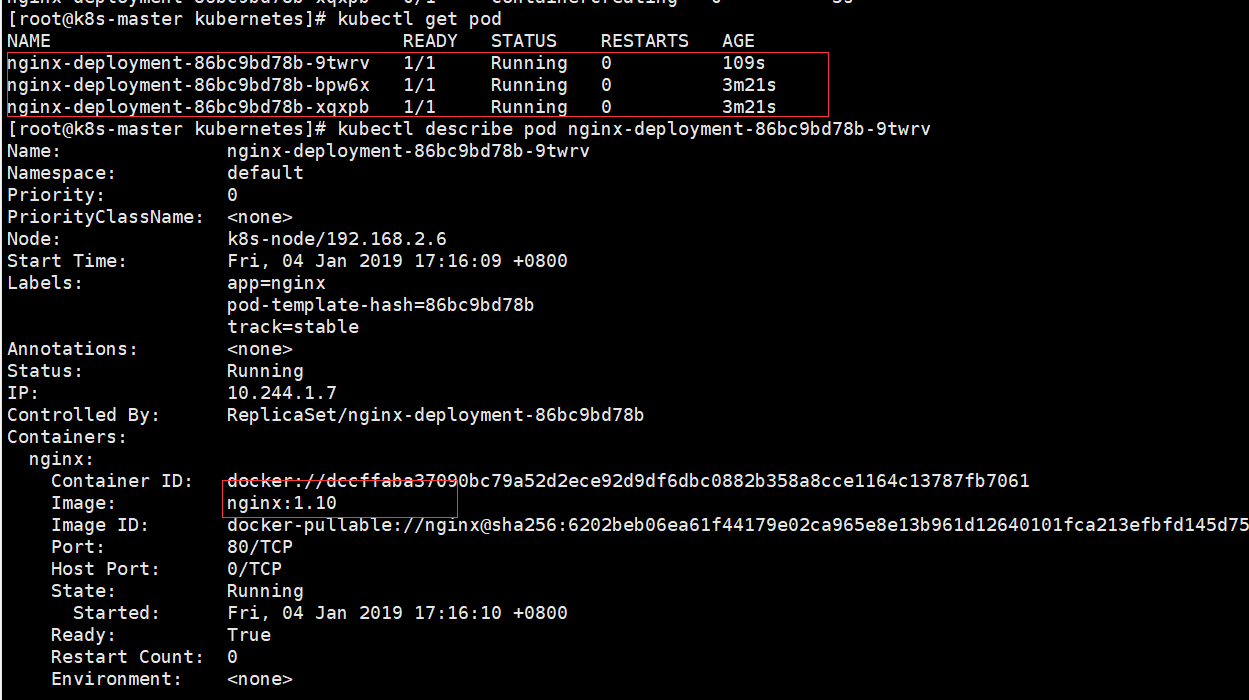


## 5.2更新镜像

Kubectl set image deploy/nginx-deployment nginx=nginx:1.10 --record



Kubectl get pod 可以看到之前的pod并不会被马上终止，而是一步步新运行一个，终止一个，达到灰度发布，不重启服务器更新的目的。稍等片刻后我们就可以看到之前的3个pod全部被替换。

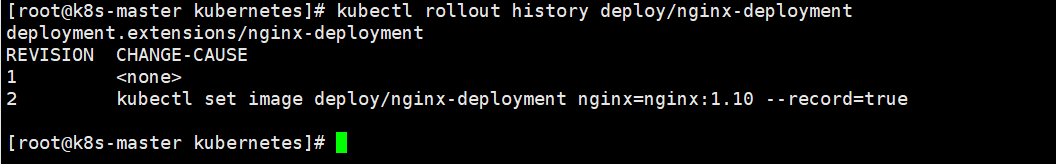


紧着我们查看pod信息，运行正常，版本也成功更新到1.10

# 回滚

## 6.1查看历史版本

Kubectl rollout history deploy/nginx-deployment



可以看到以上有1和2两个版本，1表示之前的，2表示当前的，可能有2,3,4那么最新的就表示当前版本。接着我们回滚一下。

## 6.2回滚版本

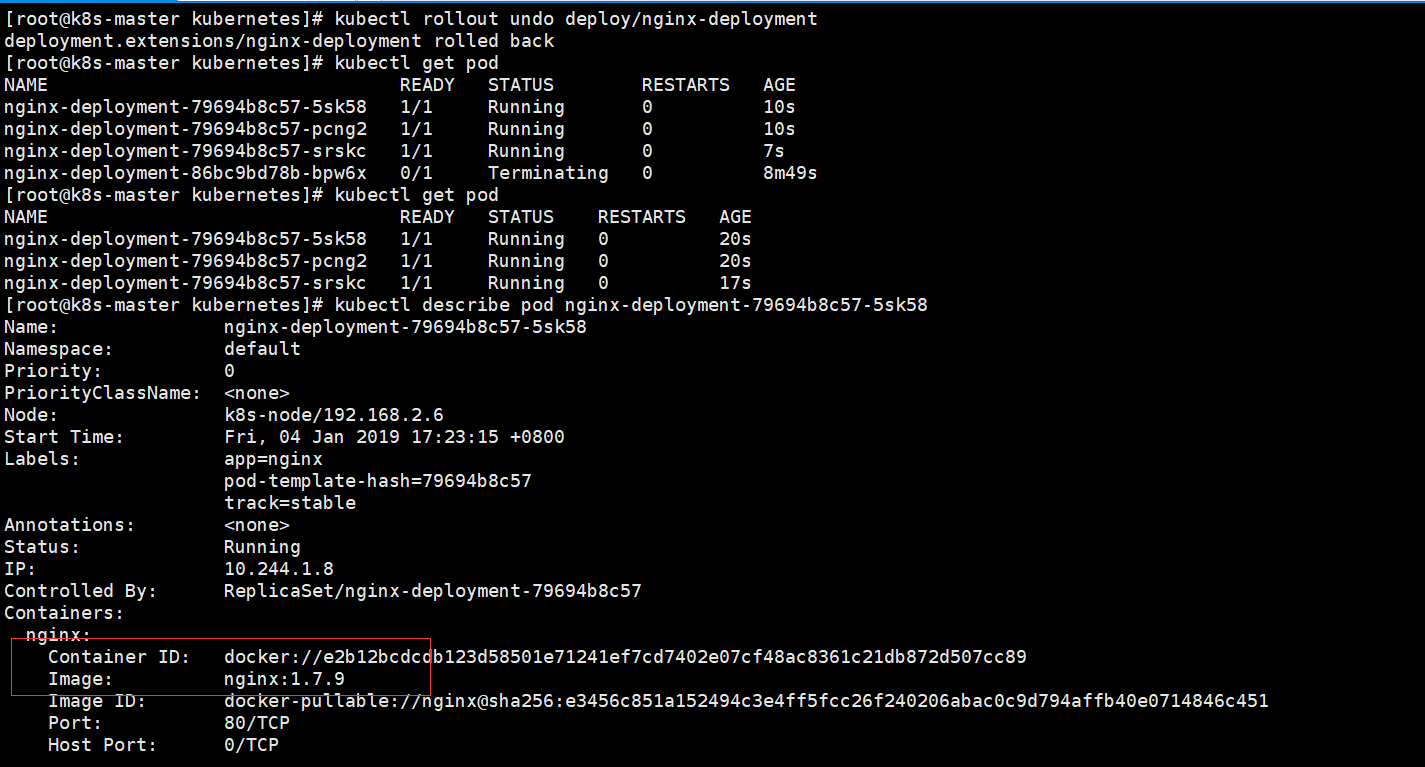
回滚到上一个版本：

Kubectl rollout undo deploy/nginx-deployment

回滚到指定版本：

Kubectl rollout undo deploy/nginx-deployment --to-revision=2

我们回滚到上一个版本试试，可以看到我们镜像版本已经回退到之前的1.7.9了。也是逐个替换，滚动更新。



# Dashborad

## 7.1镜像准备

因为k8s镜像都是需要从国外下载，我们需要先准备国内镜像。

注意：每个节点，包括master和node节点。

vim dashborad-image.sh

#/bin/bash  
 docker pull registry.cn-shanghai.aliyuncs.com/qubit/kubernetes-dashboard-amd64:v1.10.0  
 docker tag registry.cn-shanghai.aliyuncs.com/qubit/kubernetes-dashboard-amd64:v1.10.0 k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1  
 docker image rm registry.cn-shanghai.aliyuncs.com/qubit/kubernetes-dashboard-amd64:v1.10.0

chmod +x image.sh ; sh image.sh

## 7.2编写dashborad.yaml

# Copyright 2017 The Kubernetes Authors.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

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

apiVersion: v1

kind: Secret

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard-certs

namespace: kube-system

type: Opaque

---

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

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

---

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

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

rules:

# Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.

- apiGroups: [""]

resources: ["secrets"]

verbs: ["create"]

# Allow Dashboard to create 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

verbs: ["create"]

# Allow Dashboard to get, update and delete Dashboard exclusive secrets.

- apiGroups: [""]

resources: ["secrets"]

resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-certs"]

verbs: ["get", "update", "delete"]

# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config map.

- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

# Allow Dashboard to get metrics from heapster.

- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:"]

verbs: ["get"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: kubernetes-dashboard-minimal

namespace: kube-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: kubernetes-dashboard-minimal

subjects:

- kind: ServiceAccount

name: kubernetes-dashboard

namespace: kube-system

---

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

kind: Deployment

apiVersion: apps/v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

k8s-app: kubernetes-dashboard

template:

metadata:

labels:

k8s-app: kubernetes-dashboard

spec:

containers:

- name: kubernetes-dashboard

image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.1

ports:

- containerPort: 8443

protocol: TCP

args:

- --auto-generate-certificates

# Uncomment the following line to manually specify Kubernetes API server Host

# If not specified, Dashboard will attempt to auto discover the API server and connect

# to it. Uncomment only if the default does not work.

# - --apiserver-host=http://my-address:port

volumeMounts:

- name: kubernetes-dashboard-certs

mountPath: /certs

# Create on-disk volume to store exec logs

- mountPath: /tmp

name: tmp-volume

livenessProbe:

httpGet:

scheme: HTTPS

path: /

port: 8443

initialDelaySeconds: 30

timeoutSeconds: 30

volumes:

- name: kubernetes-dashboard-certs

secret:

secretName: kubernetes-dashboard-certs

- name: tmp-volume

emptyDir: {}

serviceAccountName: kubernetes-dashboard

# Comment the following tolerations if Dashboard must not be deployed on master

tolerations:

- key: node-role.kubernetes.io/master

effect: NoSchedule

---

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

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

ports:

- port: 443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboard

## 7.3创建和修改

创建：Kubectl create -f dashborad.yaml

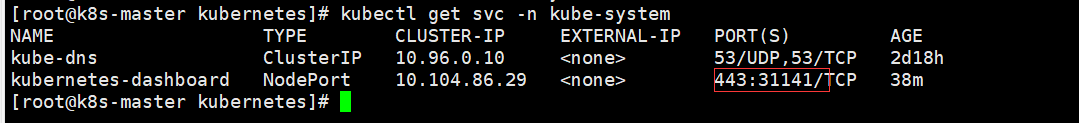
因为创建出来的service默认是ClusterIP,只能集群内访问，修改一下类型

需要改成NodePort。

kubectl patch svc kubernetes-dashboard -p '{"spec":{"type":"NodePort"}}' -n kube-system

查看一下service

Kubectl get svc -n kube-system



这里我们对外暴露的端口是31141，此时我们可以直接用https://master\_ip:31141来访问，这个端口是随机的默认是30000-32767

注意：我们只能用firefox浏览器

此时还不能直接访问，我们还需要添加一个授权管理的用户。

## 7.4创建ServiceAccount

vim admin-user.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

labels:

k8s-app: kubernetes-dashboard

name: admin

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: admin

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: admin

namespace: kube-system

创建：kubectl create -f admin-user.yaml

查看svc:kubectl describe serviceaccount admin -n kube-system

此时会有一个tokens的描述

查看token:kubectl describe secret tokens描述 -n kube-system

此时我们拷贝生成的token。

## 7.5登录UI

此时输入https://master\_ip:31141

此时我们选择高级，添加例外，然后选择token，将上面的token黏贴后即可登录。

