

全国职业院校技能大赛改革试点赛云计算赛项

国基北盛云开发平台软件（容器云平台）

用户手册

V1.0

2020年10月

目 录

[简介 4](#_Toc54858177)

[1、PAAS平台基础架构与环境说明 5](#_Toc54858178)

[1.1、Docker简介及基本组件 5](#_Toc54858179)

[1.Docker简介 5](#_Toc54858180)

[2.Docker平台组成 6](#_Toc54858181)

[1.2、Kubernetes简介及基本组件 8](#_Toc54858182)

[1.Kubernetes简介 8](#_Toc54858183)

[2.Kubernetes集群平台组成结构 9](#_Toc54858184)

[2、Kubernetes集群部署 11](#_Toc54858185)

[2.1、系统说明 11](#_Toc54858186)

[2.2、网络架构及硬件配置 11](#_Toc54858187)

[2.3、基础环境部署 11](#_Toc54858188)

[（1）修改主机名并配置映射 11](#_Toc54858189)

[（2）关闭SELinux和防火墙 12](#_Toc54858190)

[（3）删除iptables防火墙规则 12](#_Toc54858191)

[（4）配置yum源 12](#_Toc54858192)

[2.4、部署Harbor仓库 13](#_Toc54858193)

[（1）安装Docker 13](#_Toc54858194)

[（2）安装docker-compose 14](#_Toc54858195)

[（3）部署Harbor 14](#_Toc54858196)

[（4）上传镜像 15](#_Toc54858197)

[2.5、部署Kubernetes集群 16](#_Toc54858198)

[（1）安装Kubeadm 16](#_Toc54858199)

[（2）初始化集群 16](#_Toc54858200)

[（3）安装Dashboard 18](#_Toc54858201)

[（4）删除污点 22](#_Toc54858202)

[2.6、node节点加入集群 22](#_Toc54858203)

[（1）node节点加入集群 22](#_Toc54858204)

[（2）查看节点信息 22](#_Toc54858205)

[3、Kubernetes的基本使用 24](#_Toc54858206)

[3.1、基本操作 24](#_Toc54858207)

[（1）kubectl语法 24](#_Toc54858208)

[（2）创建资源对象 26](#_Toc54858209)

[（3）查看资源对象 26](#_Toc54858210)

[（4）描述资源对象 26](#_Toc54858211)

[（5）删除资源对象 26](#_Toc54858212)

[（6）容器操作 27](#_Toc54858213)

[（7）Pod的扩容与缩容 27](#_Toc54858214)

[（8）Pod的滚动升级 27](#_Toc54858215)

[3.2、运行容器 27](#_Toc54858216)

[（1）运行Nginx应用 27](#_Toc54858217)

[（2）查看Pods 28](#_Toc54858218)

[（3）开放端口 28](#_Toc54858219)

[（4）测试Nginx应用 28](#_Toc54858220)

[3.3、Kubernetes运维 29](#_Toc54858221)

[（1）Node的隔离与恢复 29](#_Toc54858222)

[（2）Pod动态扩容和缩放 29](#_Toc54858223)

[（3）将Pod调度到指定的Node 30](#_Toc54858224)

[（4）应用滚动升级 32](#_Toc54858225)

[3.4、部署博客系统 39](#_Toc54858226)

[（1）配置NFS服务 39](#_Toc54858227)

[（2）配置NFS PV持久化存储 40](#_Toc54858228)

[（3）部署应用 41](#_Toc54858229)

[（4）验证应用 46](#_Toc54858230)

[（5）访问应用 47](#_Toc54858231)

# 简介

chinaskills\_cloud\_paas.iso镜像包含安装Docker-ce服务的软件包、部署Harbor2.1.0和Kubernetes1.18.1容器云平台所需的镜像以及nginx、wordpress、jenkins等应用镜像。用户通过Chinaskill\_Cloud\_PaaS.iso镜像包中的文件可安装 Docker-ce服务及构建Kubernetes.集群，并能通过提供的ngimx.、wordpress、jenkins等容器镜像部署相应的应用。

chinaskills\_cloud\_paas.iso包含的具体内容如下：

|  |  |  |
| --- | --- | --- |
| 编号 | 软件包 | 详细信息 |
| 1 | yaml | 用于部署Dashboard与flannel网络的yaml模板文件 |
| 2 | plugins | 持续集成所需的离线插件 |
| 3 | kubernetes-repo | 用于安装kubernetes容器云平台的基础软件包 |
| 4 | images | 提供K8S平台、容器仓库、持续集成、LNMP环境需要用到的所有容器镜像；提供Wordpress、nginx、mysql、php、owncloud等服务与应用的容器镜像包 |
| 5 | harbor | 提供harbor仓库所需要的的离线安装包 |
| 6 | docker-compose | 提供docker-compose编排服务 |
| 7 | ChinaskillProject | 持续集成案例所需的代码包 |
| 8 | ChinaskillMall | 容器化部署商城所需的软件包 |
| 9 | k8s\_master\_install.sh  k8s\_node\_install.sh | 提供K8S容器云平台中master节点和node节点的安装脚本 |
| 10 | k8s\_image\_push.sh | 提供镜像的上传脚本 |
| 11 | k8s\_harbor\_install.sh | 提供harbor容器仓库的安装脚本 |

# 1、PAAS平台基础架构与环境说明

## 1.1、Docker简介及基本组件

### 1.Docker简介

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的Linux机器上，也可以实现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口。

一个完整的Docker有以下几个部分组成：

* Docker Client客户端
* Docker Daemon守护进程
* Docker Image镜像
* Docker Container容器

Docker引擎Docker Engine是C/S架构，主要有以下部件组成：

服务器（Docker daemon）：后台运行的Docker daemon进程。Daemon进程用于管理Docker对象，包括镜像（images）、容器（containers）、网络（networks）、数据卷（data volumes）。

REST接口：同daemon交互的REST API接口。

客户端（Docker client）：命令行（CLI）交互客户端。客户端使用REST API接口同Docker daemon进行访问。Docker服务的架构图如图1.1所示。

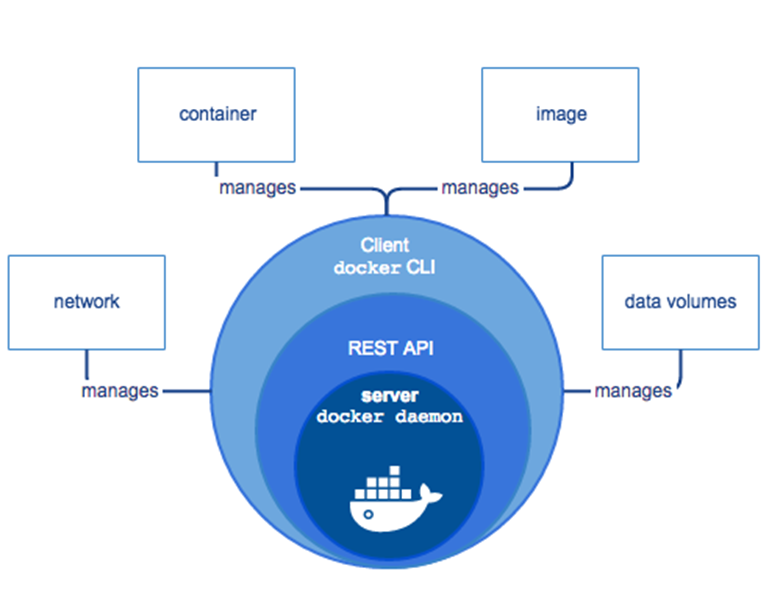


图1.1 Docker服务的架构图

### 2.Docker平台组成

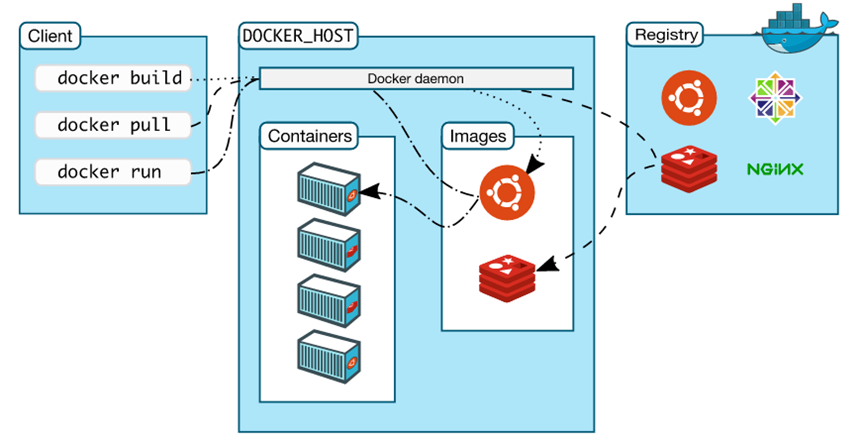


图1.2 Docker服务组成图

运行一个Docker服务，组成包括Docker daemon服务器、Docker Client客户端、Docker Image镜像、Docker Registry库、Docker Contrainer容器，如图1.2所示。

（1）Docker镜像：

是一个只读模板，用于创建Docker容器，由Dockerfile文本描述镜像的内容。镜像定义类似“面对对象的类”，从一个基础镜像（Base Image）开始。构建一个镜像实际就是安装、配置和运行的过程，Docker镜像基于UnionFS把以上过程进行分层（Layer）存储，这样更新镜像可以只更新变化的层。Docker的描述文件为Dockerfile，Dockerfile是一个文本文件，基本指令包括：

FROM：定义基础镜像。

MAINTAINER ：作者或维护者。

RUN：运行linux 命令。

ADD：增加文件或目录。

EVN：定义环境变量。

CMD：运行进程。

（2）Docker容器：

是一个镜像的运行实例。容器有镜像创建，运行过程例如：

运行ubuntu操作系统镜像，-I 前台交互模型，运行命令为/bin/bash

$ docker run -i -t ubuntu /bin/bash

运行过程如下：

拉取（pull）镜像，Docker Engine 检查ubuntu 镜像是否存在，如果本地已经存在，使用该镜像创建容器，如果不存在，Docker Engine从镜像库拉镜像。

使用该镜像创建新容器。

分配文件系统，挂载一个读写层，在读写层加载镜像。

分配网络/网桥接口，创建一个网络接口，让容器和主机通信。

从可用的IP池选择IP地址，分配给容器。

执行命令/bin/bash。

捕获和提供执行结果。

（3）Docker 仓库：

Docker仓库是Docker镜像库。Docker Registry也是一个容器。Docker Hub是Docker公司提供的互联网公共镜像仓库。可以构建自己本地的镜像仓库，国内有些公司也构建了镜像仓库。包括阿里云、新浪等。Docker 集群服务：Docker集群服务运行承租的Docker节点一起工作。目前支持swarm模式。

一个 Docker Registry 节点中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。

一般而言，一个仓库包含的是同一个软件的不同版本的镜像，而标签则用于对应于软件的的不同版本。可以通过 <仓库名>:<标签> 的格式来指定具体是哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

以 [Ubuntu 镜像](https://hub.docker.com/_/ubuntu/" \t "_blank) 为例，ubuntu 是仓库的名字，其内包含有不同的版本标签，如，14.04, 16.04。可以通过 ubuntu:14.04，或者 ubuntu:16.04 来具体指定所需哪个版本的镜像。如果忽略了标签，比如ubuntu，那将视为 ubuntu:latest。

## 1.2、Kubernetes简介及基本组件

### 1.Kubernetes简介

Kubernetes是Google开源的容器集群管理系统，其提供应用部署、维护、 扩展机制等功能，如图1.3所示。利用Kubernetes能方便地管理跨机器运行容器化的应用，其主要功能总结如下：

1) 使用Docker对应用程序包装(package)、实例化(instantiate)、运行(run)。

2) 以集群的方式运行、管理跨机器的容器。

3) 解决Docker跨机器容器之间的通讯问题。

4) Kubernetes的自我修复机制使得容器集群总是运行在用户期望的状态。

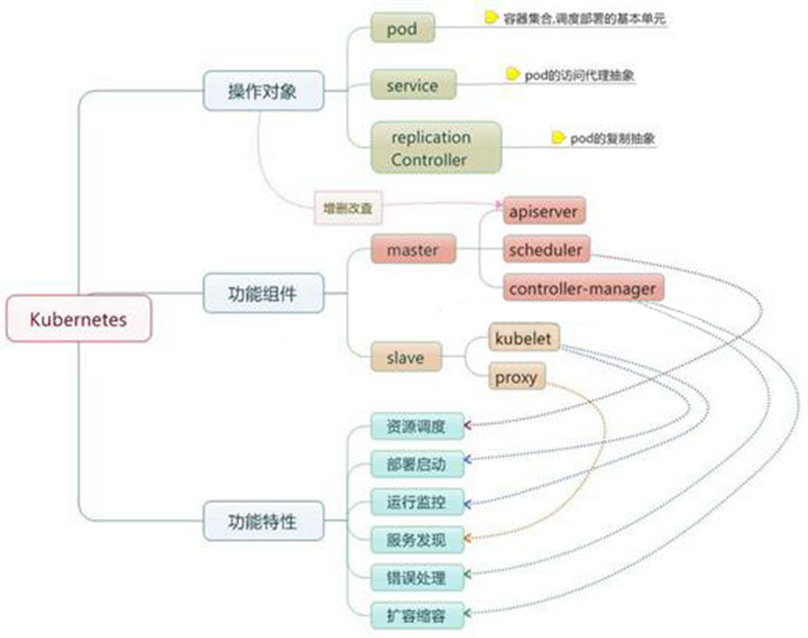


图1.3 Kubernetes的功能结构

### 2.Kubernetes集群平台组成结构

Kubernetes总体包含两种角色，一个是Master节点，负责集群调度、对外接口、访问控制、对象的生命周期维护等工作；另一个是Node节点，负责维护容器的生命周期，例如创建、删除、停止Docker容器，负责容器的服务抽象和负载均衡等工作。其中Master节点上，运行着三个核心组件：API Server, Scheduler, Controller Mananger。Node节点上运行两个核心组件：Kubelet， Kube-Proxy。API Server提供Kubernetes集群访问的统一接口，Scheduler, Controller Manager, Kubelet, Kube-Proxy等组件都通过API Server进行通信，API Server将Pod, Service, Replication Controller, Daemonset等对象存储在ETCD集群中。ETCD是CoreOS开发的高效、稳定的强一致性Key-Value数据库，ETCD本身可以搭建成集群对外服务，它负责存储Kubernetes所有对象的生命周期，是Kubernetes的最核心的组件。各个组件之间的关系详情如图1-4所示。

下面先大概介绍一下Kubernetes的核心组件的功能：

API Server: 提供了资源对象的唯一操作入口，其他所有的组件都必须通过它提供的API来操作资源对象。它以RESTful风格的API对外提供接口。所有Kubernetes资源对象的生命周期维护都是通过调用API Server的接口来完成，例如，用户通过kubectl创建一个Pod，即是通过调用API Server的接口创建一个Pod对象，并储存在ETCD集群中。

Controller Manager: 集群内部的管理控制中心，主要目的是实现Kubernetes集群的故障检测和自动恢复等工作。它包含两个核心组件：Node Controller和Replication Controller。其中Node Controller负责计算节点的加入和退出，可以通过Node Controller实现计算节点的扩容和缩容。Replication Controller用于Kubernetes资源对象RC的管理，应用的扩容、缩容以及滚动升级都是有Replication Controller来实现。

Scheduler: 集群中的调度器，负责Pod在集群的中的调度和分配。

Kubelet: 负责本Node节点上的Pod的创建、修改、监控、删除等Pod的全生命周期管理，Kubelet实时向API Server发送所在计算节点（Node）的信息。

Kube-Proxy: 实现Service的抽象，为一组Pod抽象的服务（Service）提供统一接口并提供负载均衡功能。

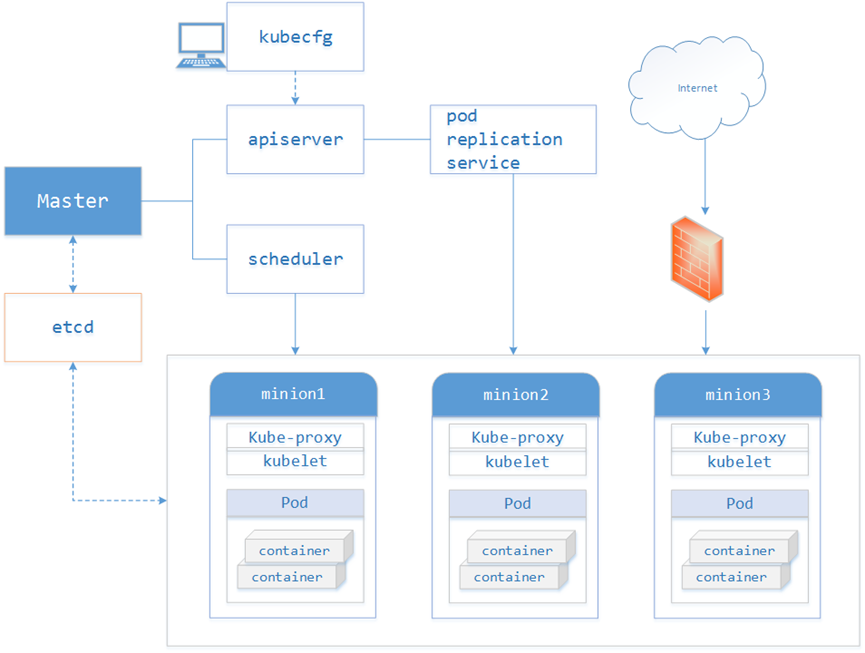


图1.4 kubernetes集群服务各组件关系

# 2、Kubernetes集群部署

## 2.1、系统说明

安装运行环境系统要求为CentOS7.5，内核版本不低于3.10。

Docker版本为docker-ce-19.03.13。

Kubernetes版本为1.18.1。

## 2.2、网络架构及硬件配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点角色 | 主机名 | 内存 | 硬盘 | IP地址 |
| Master Node | master | 12G | 100G | 10.18.4.10 |
| Worker Node | node | 8G | 100G | 10.18.4.11 |
| Harbor | master | 12G | 100G | 10.18.4.10 |

## 2.3、基础环境部署

本次实验环境采用master + node两个节点部署，node节点可以根据需求自主扩展。

### （1）修改主机名并配置映射

所有节点修改主机名并配置映射。

master节点：

# hostnamectl set-hostname master

# cat /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

10.18.4.10 master

10.18.4.11 node

node节点：

# hostnamectl set-hostname node

# cat /etc/hosts

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

10.18.4.10 master

10.18.4.11 node

### （2）关闭SELinux和防火墙

所有节点关闭SELinux：

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

# setenforce 0

所有节点关闭防火墙：

# systemctl stop firewalld.service

# systemctl disable firewalld.service

### （3）删除iptables防火墙规则

所有节点清除iptables规则：

# iptables –F

# iptables –X

# iptables –Z

# /usr/sbin/iptables-save

### （4）配置yum源

将Chinaskill\_Cloud\_PaaS.iso镜像包上传至master节点。

master节点：

# mount -o loop Chinaskill\_Cloud\_PaaS.iso /mnt/

# cp -rfv /mnt/\* /opt/

# umount /mnt/

# mv /etc/yum.repos.d/CentOS-\* /home

# cat << EOF >/etc/yum.repos.d/local.repo

[k8s]

name=k8s

baseurl=file:///opt/kubernetes-repo

gpgcheck=0

enabled=1

EOF

master节点安装ftp服务器：

# yum install -y vsftpd

# vi /etc/vsftpd/vsftpd.conf

anon\_root=/opt

# systemctl start vsftpd && systemctl enable vsftpd

node节点：

# mv /etc/yum.repos.d/CentOS-\* /home

# cat << EOF >/etc/yum.repos.d/local.repo

[k8s]

name=k8s

baseurl=ftp://master/kubernetes-repo

gpgcheck=0

enabled=1

EOF

## 2.4、部署Harbor仓库

在master节点执行脚本k8s\_harbor\_install.sh即可完成harbor仓库的搭建，具体步骤参考如下（1）-（3）步骤。

### （1）安装Docker

Kubernetes默认的容器运行时仍然是Docker，使用的是kubelet中内置dockershim CRI实现。

所有节点安装Docker-ce：

# yum install -y yum-utils device-mapper-persistent-data lvm2

# yum install -y docker-ce

启动Docker：

# systemctl start docker

# systemctl enable docker

调整部分docker参数：

# tee /etc/docker/daemon.json <<-'EOF'

{

"insecure-registries" : ["0.0.0.0/0"],

"registry-mirrors": ["https://5twf62k1.mirror.aliyuncs.com"],

"exec-opts": ["native.cgroupdriver=systemd"]

}

EOF

# systemctl restart docker

### （2）安装docker-compose

在master节点安装docker-conpose：

# cp -rfv /opt/docker-compose/v1.25.5-docker-compose-Linux-x86\_64 /usr/local/bin/docker-compose

# chmod +x /usr/local/bin/docker-compose

# docker-compose version

docker-compose version 1.25.5, build 8a1c60f6

docker-py version: 4.1.0

CPython version: 3.7.5

OpenSSL version: OpenSSL 1.1.0l 10 Sep 2019

### （3）部署Harbor

导入镜像：

# docker load -i /opt/images/Kubernetes.tar

解压安装包：

# cd harbor/

# tar -zxvf harbor-offline-installer-v2.1.0.tgz

# cd harbor

修改Harbor配置信息：

# cp harbor.yml.tmpl harbor.yml

# vi harbor.yml

hostname: 10.18.4.10 # 将域名修改为本机IP

harbor\_admin\_password: Harbor12345

#https: # 禁用https

# https port for harbor, default is 443

# port: 443

# The path of cert and key files for nginx

# certificate: /your/certificate/path

# private\_key: /your/private/key/path

启动Harbor：

# ./prepare

# ./install.sh --with-clair

登录Harbor仓库（http://10.18.4.10），如图2.4-1所示：

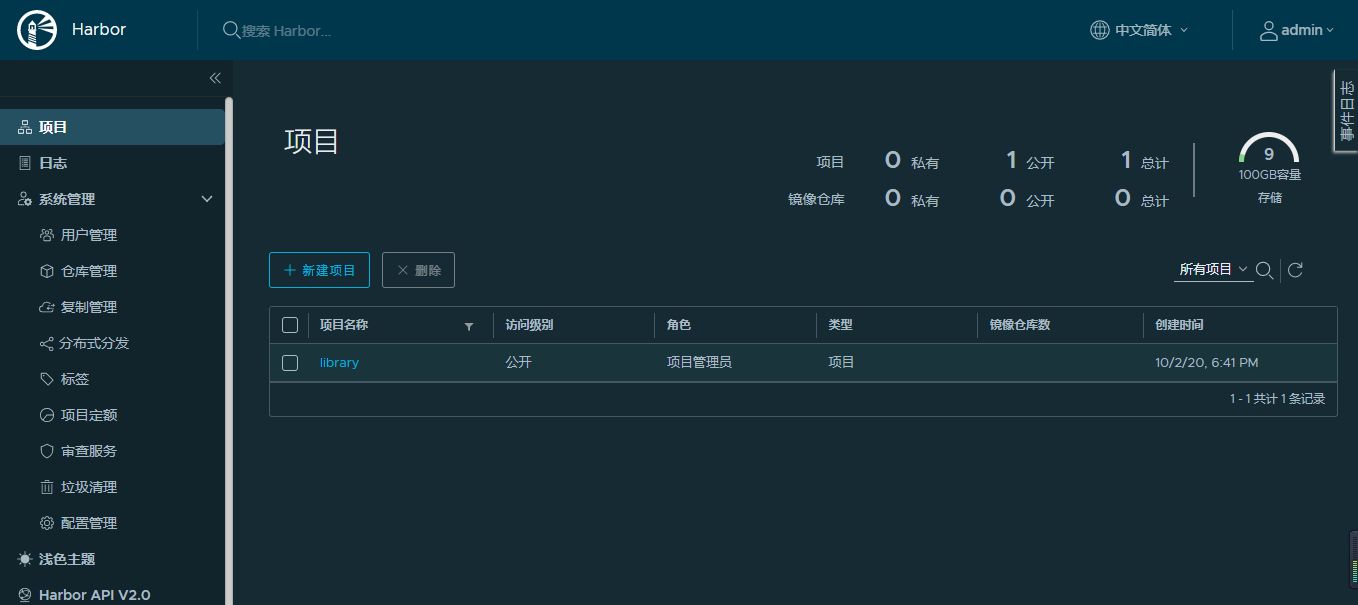


图2.4-1 Habor首页

启动、停止Harbor：

# docker-compose up -d

# docker-compose stop

# docker-compose restart

### （4）上传镜像

将镜像导入Harbor仓库：

# cd /opt/images/

# ./k8s\_image\_push.sh

输入镜像仓库地址(不加http/https): 10.18.4.10

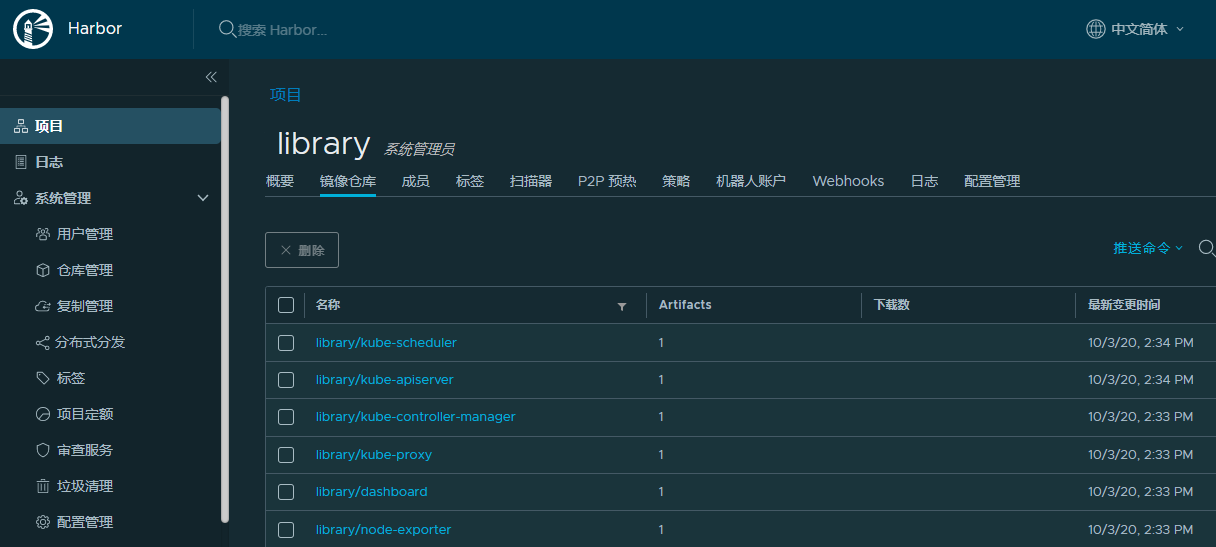
输入镜像仓库用户名: admin

输入镜像仓库用户密码: Harbor12345

您设置的仓库地址为: 10.24.2.32,用户名: admin,密码: xxx

是否确认(Y/N): Y

进入Harbor仓库library项目查看镜像列表，如图2.4-2所示。



## 2.5、部署Kubernetes集群

在master节点执行脚本k8s\_master\_install.sh即可完成K8S集群的部署，具体步骤参考如下（1）-（4）步骤。

### （1）安装Kubeadm

所有节点安装Kubeadm工具：

# yum -y install kubeadm-1.18.1 kubectl-1.18.1 kubelet-1.18.1

# systemctl enable kubelet && systemctl start kubelet

### （2）初始化集群

在master节点初始化集群：

# kubeadm init --kubernetes-version=1.18.1 --apiserver-advertise-address=10.18.4.10 --image-repository registry.aliyuncs.com/google\_containers --pod-network-cidr=10.244.0.0/16

[init] Using Kubernetes version: v1.18.1

[preflight] Running pre-flight checks

[preflight] Pulling images required for setting up a Kubernetes cluster

..................

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

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

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

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.18.4.10:6443 --token cxtb79.mqg7drycn5s82hhc \

--discovery-token-ca-cert-hash sha256:d7465b10f81ecb32ca30459efc1e0efe4f22bfbddc0c17d9b691f611082f415c

初始化完成后执行：

# mkdir -p $HOME/.kube

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

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

查看集群状态：

# kubectl get cs

NAME STATUS MESSAGE ERROR

scheduler Healthy ok

controller-manager Healthy ok

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

查看节点状态：

# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master NotReady master 2m57s v1.18.1

可以发现master处于notready状态，这是正常的，因为还没有网络插件，接下来安装网络后就变为正常了：

# kubectl apply -f yaml/kube-flannel.yaml

podsecuritypolicy.policy/psp.flannel.unprivileged created

clusterrole.rbac.authorization.k8s.io/flannel created

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

serviceaccount/flannel created

configmap/kube-flannel-cfg created

daemonset.apps/kube-flannel-ds created

# kubectl get pods -A

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system coredns-7ff77c879f-7vj79 1/1 Running 0 14m

kube-system coredns-7ff77c879f-nvclj 1/1 Running 0 14m

kube-system etcd-master 1/1 Running 0 14m

kube-system kube-apiserver-master 1/1 Running 0 14m

kube-system kube-controller-manager-master 1/1 Running 0 14m

kube-system kube-flannel-ds-d5p4g 1/1 Running 0 11m

kube-system kube-proxy-2gstw 1/1 Running 0 14m

kube-system kube-scheduler-master 1/1 Running 0 14m

# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master Ready master 17m v1.18.1

### （3）安装Dashboard

创建证书：

# mkdir dashboard-certs

# cd dashboard-certs/

# kubectl create namespace kubernetes-dashboard

# openssl genrsa -out dashboard.key 2048

Generating RSA private key, 2048 bit long modulus

......................................+++

...........................................................+++

e is 65537 (0x10001)

# openssl req -days 36000 -new -out dashboard.csr -key dashboard.key -subj '/CN=dashboard-cert'

# openssl x509 -req -in dashboard.csr -signkey dashboard.key -out dashboard.crt

Signature ok

subject=/CN=dashboard-cert

Getting Private key

# kubectl create secret generic kubernetes-dashboard-certs --from-file=dashboard.key --from-file=dashboard.crt -n kubernetes-dashboard

安装Dashboard：

# kubectl apply -f recommended.yaml

namespace/kubernetes-dashboard created

serviceaccount/kubernetes-dashboard created

service/kubernetes-dashboard created

secret/kubernetes-dashboard-certs created

secret/kubernetes-dashboard-csrf created

secret/kubernetes-dashboard-key-holder created

configmap/kubernetes-dashboard-settings created

role.rbac.authorization.k8s.io/kubernetes-dashboard created

clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created

rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created

clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created

deployment.apps/kubernetes-dashboard created

service/dashboard-metrics-scraper created

deployment.apps/dashboard-metrics-scraper created

查看Dashboard 关联Pod和Service的状态：

# kubectl get pod,svc -n kubernetes-dashboard

NAME READY STATUS RESTARTS AGE

pod/dashboard-metrics-scraper-6b4884c9d5-f7qxd 1/1 Running 0 62s

pod/kubernetes-dashboard-5585794759-2c6xt 1/1 Running 0 62s

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

service/dashboard-metrics-scraper ClusterIP 10.105.228.249 <none> 8000/TCP 62s

service/kubernetes-dashboard NodePort 10.98.134.7 <none> 443:30000/TCP 62s

创建serviceaccount和clusterrolebinding

# kubectl apply -f dashboard-adminuser.yaml

serviceaccount/dashboard-admin created

clusterrolebinding.rbac.authorization.k8s.io/dashboard-admin-bind-cluster-role created

在火狐浏览器访问dashboard（https://IP:30000），如图2.5-1所示：



图2.5-1 Dashboard认证界面

获取Token：

# kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep dashboard-admin | awk '{print $1}')

Name: dashboard-admin-token-x9fnq

Namespace: kubernetes-dashboard

Labels: <none>

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

kubernetes.io/service-account.uid: f780f22d-f620-4cdd-ad94-84bf593ca882

Type: kubernetes.io/service-account-token

Data

====

ca.crt: 1025 bytes

namespace: 20 bytes

token: eyJhbGciOiJSUzI1NiIsImtpZCI6IkRWNXNrYWV6dFo4bUJrRHVZcmwtcTVpNzdFMDZYZjFYNzRzQlRyYmlVOGsifQ..h5BGk2yunmRcA8U60wIJh0kWpRLI1tZqS58BaDy137k1SYkvwG4rfG8MGnoDMAREWd9JIX43N4qpfbivIefeKIO\_CZhYjv4blRefjAHo9c5ABChMc1lrZq9m\_3Br\_fr7GonsYulkaW6qYkCcQ0RK1TLlxntvLTi7gWMSes8w-y1ZumubL4YIrUh-y2OPoi2jJNevn4vygkgxtX5Y9LlxegVYJfeE\_Sb9jV9ZL7e9kDqmBIYxm5PBJoPutjsTBmJf3IFrf6vUk6bBWtE6-nZgdf6FAGDd2W2-1YcidjITwgUvj68OfQ5tbB94EYlJhuoAGVajKxO14XaE9NH0\_NZjqw

输入Token后进入Dashboard仪表盘界面，如图2.5-2所示：

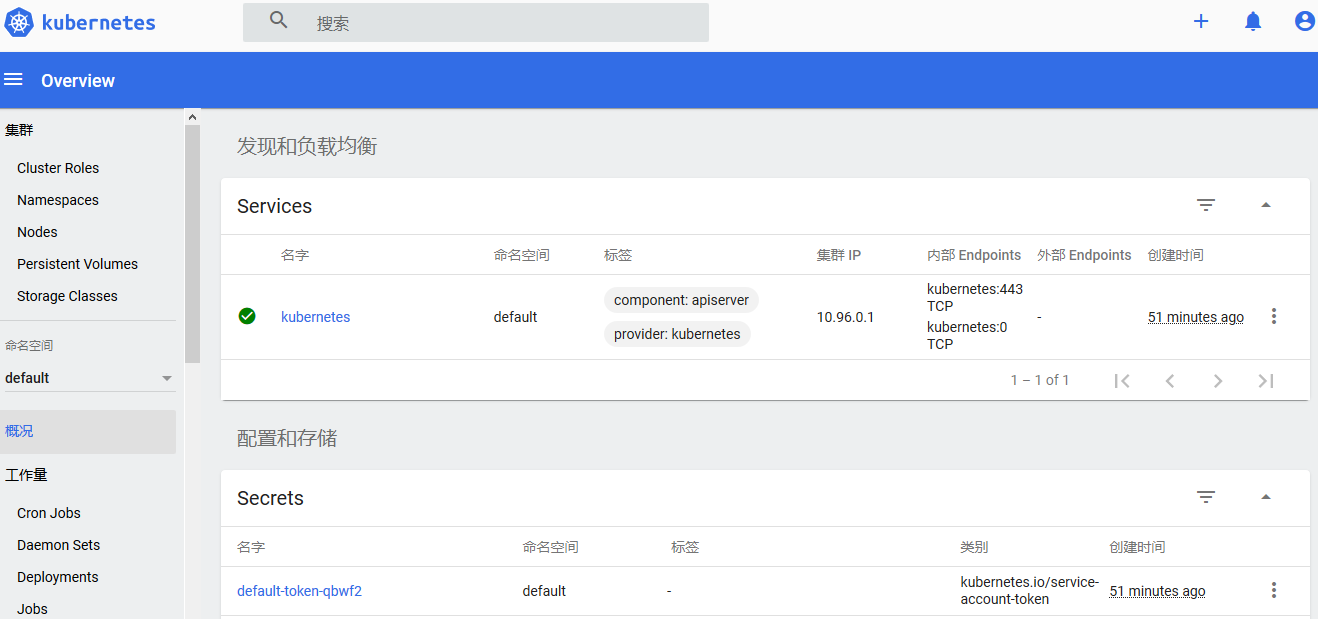


图2.5-2 Kubernetes仪表盘

### （4）删除污点

出于安全考虑，默认配置下Kubernetes不会将Pod调度到Master节点。如果希望将master也当作Node节点使用，可以执行如下命令：

# kubectl taint node master node-role.kubernetes.io/master-

node/master untainted

## 2.6、node节点加入集群

在node节点执行脚本k8s\_node\_install.sh即可将node节点加入集群，具体步骤参考如下（1）-（2）步骤。

### （1）node节点加入集群

在node节点执行以下命令加入集群：

# kubeadm join 10.18.4.10:6443 \

--token cxtb79.mqg7drycn5s82hhc --discovery-token-ca-cert-hash \

sha256:d7465b10f81ecb32ca30459efc1e0efe4f22bfbddc0c17d9b691f611082f415c

### （2）查看节点信息

在master节点查看节点状态：

# kubectl get nodes

NAME STATUS ROLES AGE VERSION

master Ready master 56m v1.18.1

node Ready <none> 60s v1.18.1

# 3、Kubernetes的基本使用

## 3.1、基本操作

### （1）kubectl语法

kubectl命令语法如下：

# kubectl [command] [TYPE] [NAME] [flags]

命令语法解析：

⚫ command：子命令。

⚫ TYPE：资源类型。

⚫ NAME：资源名称。

⚫ flags：命令参数。

kubectl命令选项如下：

⚫ --alsologtostderr[=false]：同时输出日志到标准错误控制台和文件。

⚫ --api-version=""：和服务端交互使用的API版本。

⚫ --certificate-authority=""：用以进行认证授权的.cert文件路径。

⚫ --client-certificate=""：TLS使用的客户端证书路径。

⚫ --client-key=""：TLS使用的客户端密钥路径。

⚫ --cluster=""：指定使用的kubeconfig配置文件中的集群名。

⚫ --context=""：指定使用的kubeconfig配置文件中的环境名。

⚫ --insecure-skip-tls-verify[=false]：如果为true，将不会检查服务器凭证的有效性，这会导致HTTPS链接变得不安全。

⚫ --kubeconfig=""：命令行请求使用的配置文件路径。

⚫ --log-backtrace-at=：0：当日志长度超过定义的行数时，忽略堆栈信息。

⚫ --log-dir=""：如果不为空，将日志文件写入此目录。

⚫ --log-flush-frequency=5s：刷新日志的最大时间间隔。

⚫ --logtostderr[=true]：输出日志到标准错误控制台，不输出到文件。

⚫ --match-server-version[=false]：要求服务端和客户端版本匹配。

⚫ --namespace=""：如果不为空，命令将使用此namespace。

⚫ --password=""：APIServer进行简单认证使用的密码。

⚫ -s,--server=""：Kubernetes API Server的地址和端口号。

⚫ --stderrthreshold=2：高于此级别的日志将被输出到错误控制台。

⚫ --token=""：认证到APIServer使用的令牌。

⚫ --user=""：指定使用的kubeconfig配置文件中的用户名。

⚫ --username=""：APIServer进行简单认证使用的用户名。

⚫ --v=0：指定输出日志的级别。

⚫ --vmodule=：指定输出日志的模块。

常用命令如下：

⚫ kubectl annotate：更新资源的注解。

⚫ kubectl api-versions：以“组/版本”的格式输出服务端支持的API版本。

⚫ kubectl apply：通过文件名或控制台输入，对资源进行配置。

⚫ kubectl attach：连接到一个正在运行的容器。

⚫ kubectl autoscale：对replication controller进行自动伸缩。

⚫ kubectl cluster-info：输出集群信息。

⚫ kubectl config：修改kubeconfig配置文件。

⚫ kubectl create：通过文件名或控制台输入，创建资源。

⚫ kubectl delete：通过文件名、控制台输入、资源名或者label selector删除资源。

⚫ kubectl describe：输出指定的一个或多个资源的详细信息。

⚫ kubectl edit：编辑服务端的资源。

⚫ kubectl exec：在容器内部执行命令。

⚫ kubectl expose：输入rc、svc或Pod，并将其暴露为新的kubernetes service。

⚫ kubectl get：输出一个或多个资源。

⚫ kubectl label：更新资源的label。

⚫ kubectl logs：输出Pod中一个容器的日志。

⚫ kubectl namespace：（已停用）设置或查看当前使用的namespace。

⚫ kubectl patch：通过控制台输入更新资源中的字段。

⚫ kubectl port-forward：将本地端口转发到Pod。

⚫ kubectl proxy：为Kubernetes API Server启动代理服务器。

⚫ kubectl replace：通过文件名或控制台输入替换资源。

⚫ kubectl rolling-update：对指定的replication controller执行滚动升级。

⚫ kubectl run：在集群中使用指定镜像启动容器。

⚫ kubectl scale：为replication controller设置新的副本数。

⚫ kubectl version：输出服务端和客户端的版本信息。

### （2）创建资源对象

使用kubectl create命令可进行资源的创建。

根据Yaml配置文件一次性创建service和rc：

# kubectl create -f my-service.yaml -f my-rc.yaml

根据<directory>目录下所有.yaml、.yml、.json文件的定义进行创建操作：

# kubectl create -f <directory>

### （3）查看资源对象

查看所有Pod列表：

# kubectl get pods

查看rc和service列表：

# kubectl get rc,service

### （4）描述资源对象

显示Node的详细信息：

# kubectl describe nodes <node-name>

显示Pod的详细信息：

# kubectl describe pods/<pod-name>

显示由RC管理的Pod的信息：

# kubectl describe pods <rc-name>

### （5）删除资源对象

基于Pod.yaml定义的名称删除Pod：

# kubectl delete -f pod.yaml

删除所有包含某个label的Pod和service：

# kubectl delete pods,services -l name=<label-name>

删除所有Pod：

# kubectl delete pods --all

### （6）容器操作

执行Pod的data命令，默认是用Pod中的第一个容器执行：

# kubectl exec <pod-name> data

指定Pod中某个容器执行data命令：

# kubectl exec <pod-name> -c <container-name> data

通过bash获得Pod中某个容器的TTY，相当于登录容器：

# kubectl exec -it <pod-name> -c <container-name> bash

### （7）Pod的扩容与缩容

执行扩容缩容Pod的操作：

# kubectl scale rc redis --replicas=3

需要确认的是，在rc配置文件中定义的replicas数量，当执行上述命令的结果大于replicas的数量时，则执行的命令相当于扩容操作，反之则为缩容。可以理解为填写的数量即需要的Pod数量。

**注意：当需要进行永久性扩容时，不要忘记修改rc配置文件中的replicas数量。**

### （8）Pod的滚动升级

执行滚动升级操作：

# kubectl rolling-update redis -f redis-rc.update.yaml

**注意：当执行rolling-update命令前需要准备好新的rc配置文件以及ConfigMap配置文件。rc配置文件中需要指定升级后需要使用的镜像名称，或者可以使用kubeclt rolling-update redis --image=redis-2.0命令直接指定镜像名称来直接升级。**

## 3.2、运行容器

### （1）运行Nginx应用

运行Nginx应用。

[root@master ~]# kubectl run nginx --image=10.18.4.10/library/nginx:latest

kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.

deployment.apps/nginx created

### （2）查看Pods

验证Pods是否正常运行。

[root@master ~]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-ccb467dc5-92tbn 1/1 Running 0 93s

……

### （3）开放端口

使用expose将service的80端口开放出去。

[root@master ~]# kubectl expose deploy/nginx --port 80

service/nginx exposed

### （4）测试Nginx应用

[root@master ~]# kubectl get svc

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 3h18m

nginx ClusterIP 10.100.220.6 <none> 80/TCP 9s

[root@master ~]# curl 10.100.220.6:80

……

<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>

使用curl工具可以成功获取到Nginx网页信息，说明80端口已开放成功。

## 3.3、Kubernetes运维

### （1）Node的隔离与恢复

在硬件升级、硬件维护等情况下，需要将某些Node隔离。使用kubectl cordon <node\_name>命令可禁止Pod调度到该节点上，在其上运行的Pod并不会自动停止，管理员需要手动停止在该Node上运行的Pod。

[root@master ~]# kubectl cordon node

查看Node的状态，可以观察到在node的状态中增加了一项SchedulingDisabled，对于后续创建的Pod，系统将不会再向该Node进行调度。

[root@master ~]# kubectl get nodes

通过kubectl uncordon命令可完成对Node的恢复。

[root@master ~]# kubectl uncordon node

[root@master ~]# kubectl get nodes

可以看到Node节点已恢复调度，允许Pod调度到该节点上。

通过kubectl drain <node>命令可实现对node节点的驱逐，该命令会删除该节点上的所有Pod（DaemonSet除外），在其他Node上重新启动它们。

### （2）Pod动态扩容和缩放

在实际生产系统中，经常会遇到某个服务需要扩容的场景，也可能会遇到由于资源紧张或者工作负载降低而需要减少服务实例数量的场景。此时可以利用kubectl scale deployment命令来完成这些任务。

以Nginx Deployment为例，已定义的最初副本数量为1。

[root@master ~]# kubectl run nginx --image=10.18.4.10/library/nginx:latest

kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.

deployment.apps/nginx created

[root@master ~]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-ccb467dc5-jlr4c 1/1 Running 0 40s

通过执行下面的命令将Nginx Deployment控制的Pod副本数量从初始的1更新为5。

[root@master ~]# kubectl scale deployment nginx --replicas=5

deployment.extensions/nginx scaled

执行kubectl get pods命令来验证Pod的副本数量增加到5。

[root@master ~]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-ccb467dc5-2f6n2 1/1 Running 0 34s

**......**

将--replicas设置为比当前Pod副本数量更小的数字，系统将会“杀掉”一些运行中的Pod，即可实现应用集群缩容。

[root@master ~]# kubectl scale deployment nginx --replicas=2

deployment.extensions/nginx scaled

[root@master ~]# kubectl get pods

NAME READY STATUS RESTARTS AGE

nginx-ccb467dc5-bl4jp 1/1 Running 0 2m50s

nginx-ccb467dc5-jlr4c 1/1 Running 0 5m21s

### （3）将Pod调度到指定的Node

Kubernetes的Scheduler服务（kube-scheduler进程）负责实现Pod的调度，整个调度过程通过执行一系列复杂的算法最终为每个Pod计算出一个最佳的目标节点，这一过程是自动完成的，用户无法知道Pod最终会被调度到哪个节点上。有时可能需要将Pod调度到一个指定的Node上。此时，可以通过Node的标签（Label）和Pod的nodeSelector属性相匹配，来达到上述目的。

Label（标签）作为用户可灵活定义的对象属性，在已创建的对象上，仍然可以随时通过kubectl label命令对其进行增加、修改、删除等操作。使用kubectl label给node打标签的用法如下：

# kubectl label nodes <node-name> <label-key>=<label-value>

下面的示例，为node打上一个project=gcxt的标签。

[root@master ~]# kubectl label nodes node project=gcxt

node/node labeled

如果想删除Label，只需要在命令行最后指定Label的key名，并加一个减号即可。

[root@master ~]# kubectl label node node project-

node/node labeled

在Pod中加入nodeSelector定义，示例如下。

[root@master ~]# cat nginx.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: memcached-gcxt

labels:

name: memcached-gcxt

spec:

replicas: 1

selector:

name: memcached-gcxt

template:

metadata:

labels:

name: memcached-gcxt

spec:

containers:

- name: memcached-gcxt

image: memcached

command:

- memcached

- -m 64

ports:

- containerPort: 11211

nodeSelector:

project: gcxt

运行kubectl create -f命令创建Pod，scheduler就会将该Pod调度到拥有project=gcxt标签的Node上去。

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

replicationcontroller/memcached-gcxt created

查看Pod。

[root@master ~]# kubectl get pods -owide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

memcached-gcxt-hdt5x 1/1 Running 0 14s 10.24.9.2 node <none> <none>

可以看到，Pod已成功调度到指定的Node节点。

这种基于Label标签的调度方式灵活性很高，比如，可以把一组Node分别贴上“开发环境”、“测试环境”、“生产环境”这3组标签中的一种，此时一个Kubernetes集群就承载了3个环境，这将大大提高开发效率。

**注意：如果指定了Pod的nodeSelector条件，且集群中不存在包含相应标签的Node时，即使还有其他可供调度的Node，这个Pod也最终会调度失败。**

### （4）应用滚动升级

当集群中的某个服务需要升级时，需要停止目前与该服务相关的所有Pod，然后重新拉取镜像并启动。如果集群规模比较大，这个工作就变成了一个挑战。如果采取先全部停止，然后逐步升级的方式，会导致较长时间的服务不可用。Kubernetes提供了rolling-update（滚动升级）功能来解决上述问题。

滚动升级通过执行kubectl rolling-update命令一键完成，该命令创建了一个新的Deployment，然后自动控制旧的Deployment中的Pod副本数量逐渐减少到0，同时新的Deployment中的Pod副本数量从0逐步增加到目标值，最终实现了Pod的升级。

**注意：系统要求新的Deployment需要与旧的Deployment在相同的命名空间（Namespace）内，即不能把别人的资产偷偷转移到自家名下。**

下面的示例在第一次部署时使用httpd:2.2.31，然后使用滚动升级更新到httpd:2.2.32。

定义httpd.yaml文件。

[root@master ~]# cat httpd.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: httpd

spec:

replicas: 3

template:

metadata:

labels:

run: httpd

spec:

containers:

- name: httpd

image: 10.18.4.10/library/httpd:2.2.31

ports:

- containerPort: 80

启动Deployment。

[root@master ~]# kubectl create -f httpd.yaml

deployment.apps/httpd created

[root@master ~]# kubectl get pods

NAME READY STATUS RESTARTS AGE

httpd-5ddb558f47-46tf5 1/1 Running 0 49s

httpd-5ddb558f47-5dg96 1/1 Running 0 49s

httpd-5ddb558f47-672sk 1/1 Running 0 49s

查看Deployment。

[root@master ~]# kubectl get deployments httpd -o wide

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

httpd 3/3 3 3 93s httpd httpd:2.2.31 run=httpd

可以看到，IMAGES为httpd:2.2.31。

把配置文件中的httpd:2.2.31改为httpd:2.2.32，再次启动。

[root@master ~]# cat httpd.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: httpd

spec:

replicas: 3

template:

metadata:

labels:

run: httpd

spec:

containers:

- name: httpd

image: 10.18.4.10/library/httpd:2.2.32 //将2.2.31更改为2.2.32

ports:

- containerPort: 80

[root@master ~]# kubectl apply -f httpd.yaml

deployment.apps/httpd configured

再次查看Deployment。

[root@master ~]# kubectl get deployments httpd -o wide

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

httpd 3/3 3 3 7m53s httpd httpd:2.2.32 run=httpd

查看Deployment的详细信息。

[root@master ~]# kubectl describe deployment httpd

…...

Events:

Type Reason Age From Message

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

Normal ScalingReplicaSet 22m deployment-controller Scaled up replica set httpd-5ddb558f47 to 3

Normal ScalingReplicaSet 15m deployment-controller Scaled up replica set httpd-8bdffc6d8 to 1

Normal ScalingReplicaSet 14m deployment-controller Scaled down replica set httpd-5ddb558f47 to 2

Normal ScalingReplicaSet 14m deployment-controller Scaled up replica set httpd-8bdffc6d8 to 2

Normal ScalingReplicaSet 14m deployment-controller Scaled down replica set httpd-5ddb558f47 to 1

Normal ScalingReplicaSet 14m deployment-controller Scaled up replica set httpd-8bdffc6d8 to 3

Normal ScalingReplicaSet 13m deployment-controller Scaled down replica set httpd-5ddb558f47 to 0

上面的日志信息就描述了滚动升级的过程：

① 启动一个新版Pod。

② 把旧版Pod数量降为2。

③ 再启动一个新版，数量变为2。

④ 把旧版Pod数量降为1。

⑤ 再启动一个新版，数量变为3。

⑥ 把旧版Pod数量降为0。

这就是滚动的意思，始终保持副本数量为3，控制新旧Pod的交替，实现了无缝升级。

kubectl apply每次更新应用时，kubernetes都会记录下当前的配置，保存为一个revision，这样就可以回滚到某个特定的版本。

创建3个配置文件，内容中唯一不同的就是镜像的版本号。

httpd.v1.yaml：

[root@master ~]# cat httpd.v1.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: httpd

spec:

revisionHistoryLimit: 10 # 指定保留最近的几个revision

replicas: 3

template:

metadata:

labels:

run: httpd

spec:

containers:

- name: httpd

image: 10.18.4.10/library/httpd:2.2.16

ports:

- containerPort: 80

httpd.v2.yaml：

[root@master ~]# cat httpd.v2.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: httpd

spec:

revisionHistoryLimit: 10 # 指定保留最近的几个revision

replicas: 3

template:

metadata:

labels:

run: httpd

spec:

containers:

- name: httpd

image: 10.18.4.10/library/httpd:2.2.17

ports:

- containerPort: 80

httpd.v3.yaml：

[root@master ~]# cat httpd.v3.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: httpd

spec:

revisionHistoryLimit: 10 # 指定保留最近的几个revision

replicas: 3

template:

metadata:

labels:

run: httpd

spec:

containers:

- name: httpd

image: 10.18.4.10/library/httpd:2.2.18

ports:

- containerPort: 80

部署Deployment。

[root@master ~]# kubectl apply -f httpd.v1.yaml --record

deployment.apps/httpd configured

[root@master ~]# kubectl apply -f httpd.v2.yaml --record

deployment.apps/httpd configured

[root@master ~]# kubectl apply -f httpd.v3.yaml --record

deployment.apps/httpd configured

--record的作用是将当前命令记录到revision中，可以知道每个revision对应的是哪个配置文件。

查看Deployment。

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

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

httpd 3/3 1 3 31m httpd 10.18.4.10/library/httpd:2.2.18 run=httpd

查看revision历史记录。

[root@master ~]# kubectl rollout history deployment httpd

deployment.extensions/httpd

REVISION CHANGE-CAUSE

1 <none>

2 <none>

3 kubectl apply --filename=httpd.v1.yaml --record=true

4 kubectl apply --filename=httpd.v2.yaml --record=true

5 kubectl apply --filename=httpd.v3.yaml --record=true

CHANGE-CAUSE即-record的结果。

执行如下操作，回滚到指定版本revision 1。

[root@master ~]# kubectl rollout undo deployment httpd --to-revision=1

deployment.extensions/httpd rolled back

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

NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES SELECTOR

httpd 3/3 3 3 35m httpd 10.18.4.10/library/httpd:2.2.31 run=httpd

再次查看Deployment可以看到httpd版本已经回退了。

查看revision历史记录，可以看到revision记录也发生了变化。

[root@master ~]# kubectl rollout history deployment httpd

deployment.extensions/httpd

......

## 3.4、部署博客系统

### （1）配置NFS服务

master节点安装NFS与RPC服务：

# yum install -y nfs-utils rpcbind

node节点安装NFS客户端：

# yum install -y nfs-utils

启动RPC服务：

# systemctl start rpcbind

# systemctl enable rpcbind

查看NFS服务项RPC服务器注册的端口列表：

# rpcinfo -p localhost

启动NFS服务：

# systemctl start nfs

# systemctl enable nfs

启动NFS服务后RPC服务已经启用了对NFS的端口映射列表

# rpcinfo -p localhost

在服务器端创建2个共享目录并设置权限:

# mkdir -p /home/pvdata/mysql

# mkdir -p /home/pvdata/wordpress

# chmod 777 /home/pvdata/mysql

# chmod 777 /home/pvdata/wordpress

修改NFS配置文件/etc/exports

# vim /etc/exports

/home/pvdata/mysql \*(rw,sync,insecure,no\_subtree\_check,no\_root\_squash)

/home/pvdata/wordpress \*(rw,sync,insecure,no\_subtree\_check,no\_root\_squash)

NFS配置文件生效

# exportfs -r

# exportfs

### （2）配置NFS PV持久化存储

创建mysql pv卷

# cat mysql-persistent-storage.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-persistent-storage

spec:

capacity:

storage: 20Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Recycle

storageClassName: nfs

nfs:

path: /home/pvdata/mysql/

server: 10.18.4.10

# kubectl apply -f mysql-persistent-storage.yaml

创建wordpress pv卷：

# cat wordpress-persistent-storage.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: wordpress-persistent-storage

spec:

capacity:

storage: 20Gi

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Recycle

storageClassName: nfs

nfs:

path: /home/pvdata/wordpress/

server: 10.18.4.10

# kubectl apply -f wordpress-persistent-storage.yaml

### （3）部署应用

通过以下命令在kustomization.yaml中添加一个Secret生成器：

# cat <<EOF >./kustomization.yaml

secretGenerator:

- name: mysql-pass

literals:

- password=123456ok

EOF

MySQL容器将PersistentVolume挂载在/var/lib/mysql。MYSQL\_ROOT\_PASSWORD环境变量设置来自Secret的数据库密码：

# cat mysql-deployment.yaml

apiVersion: v1

kind: Service

metadata:

name: wordpress-mysql

labels:

app: wordpress

spec:

ports:

- port: 3306

selector:

app: wordpress

tier: mysql

clusterIP: None

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mysql-pv-claim

labels:

app: wordpress

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 20Gi

---

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

name: wordpress-mysql

labels:

app: wordpress

spec:

selector:

matchLabels:

app: wordpress

tier: mysql

strategy:

type: Recreate

template:

metadata:

labels:

app: wordpress

tier: mysql

spec:

containers:

- image: 10.18.4.10/library/mysql:5.6

name: mysql

env:

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-pass

key: password

ports:

- containerPort: 3306

name: mysql

volumeMounts:

- name: mysql-persistent-storage

mountPath: /var/lib/mysql

volumes:

- name: mysql-persistent-storage

persistentVolumeClaim:

claimName: mysql-pv-claim

WordPress容器将网站数据文件位于/var/www/html的PersistentVolume。WORDPRESS\_DB\_HOST环境变量集上面定义的MySQLService的名称，WordPress将通过Service访问数据库。WORDPRESS\_DB\_PASSWORD环境变量设置从Secretkustomize生成的数据库密码。

# cat wordpress-deployment.yaml

apiVersion: v1

kind: Service

metadata:

name: wordpress

labels:

app: wordpress

spec:

ports:

- port: 80

selector:

app: wordpress

tier: frontend

type: LoadBalancer

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: wp-pv-claim

labels:

app: wordpress

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 20Gi

---

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2

kind: Deployment

metadata:

name: wordpress

labels:

app: wordpress

spec:

selector:

matchLabels:

app: wordpress

tier: frontend

strategy:

type: Recreate

template:

metadata:

labels:

app: wordpress

tier: frontend

spec:

containers:

- image: 10.18.4.10/library/wordpress:4.8-apache

name: wordpress

env:

- name: WORDPRESS\_DB\_HOST

value: wordpress-mysql

- name: WORDPRESS\_DB\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-pass

key: password

ports:

- containerPort: 80

name: wordpress

volumeMounts:

- name: wordpress-persistent-storage

mountPath: /var/www/html

volumes:

- name: wordpress-persistent-storage

persistentVolumeClaim:

claimName: wp-pv-claim

将以上两个文件补充到kustomization.yaml文件中：

# cat <<EOF >>./kustomization.yaml

resources:

- mysql-deployment.yaml

- wordpress-deployment.yaml

EOF

kustomization.yaml包含用于部署 WordPress 网站的所有资源以及 MySQL 数据库。通过以下方式部署应用：

# kubectl apply -k ./

secret/mysql-pass-ccht6ddg2m created

service/wordpress-mysql created

service/wordpress created

deployment.apps/wordpress-mysql created

deployment.apps/wordpress created

persistentvolumeclaim/mysql-pv-claim created

persistentvolumeclaim/wp-pv-claim created

### （4）验证应用

通过运行以下命令验证Secret是否存在：

# kubectl get secrets

NAME TYPE DATA AGE

mysql-pass-ccht6ddg2m Opaque 1 39s

验证是否已动态配置PersistentVolume，设置和绑定PV可能要花费几分钟：

# kubectl get pvc

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

mysql-pv-claim Bound pvc-1f300520-7d33-4a2e-acdc-d09ef27f794a 20Gi RWO local-path 10m

wp-pv-claim Bound pvc-b13dddbd-0f1c-4b04-b6c0-677a1f50587b 20Gi RWO local-path 10m

通过运行以下命令来验证Service是否正在运行：

# kubectl get services wordpress

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

wordpress LoadBalancer 10.109.26.194 <pending> 80:31355/TCP 10m

### （5）访问应用

通过http://10.18.4.10:31355访问wordpress，如图3.4-1所示：

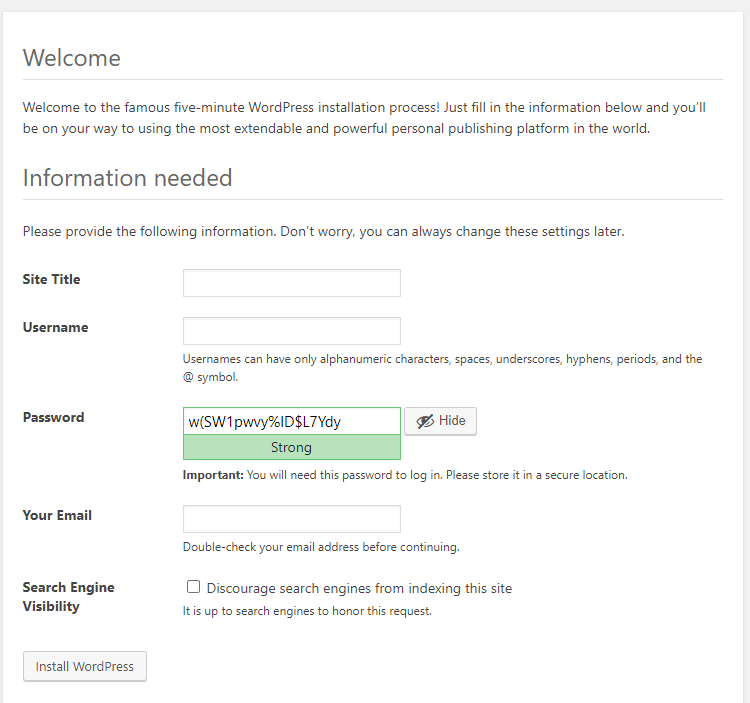


图3.4-1 wordpress初始化界面

创建用户并登录wordpress，如图3.4-2所示：

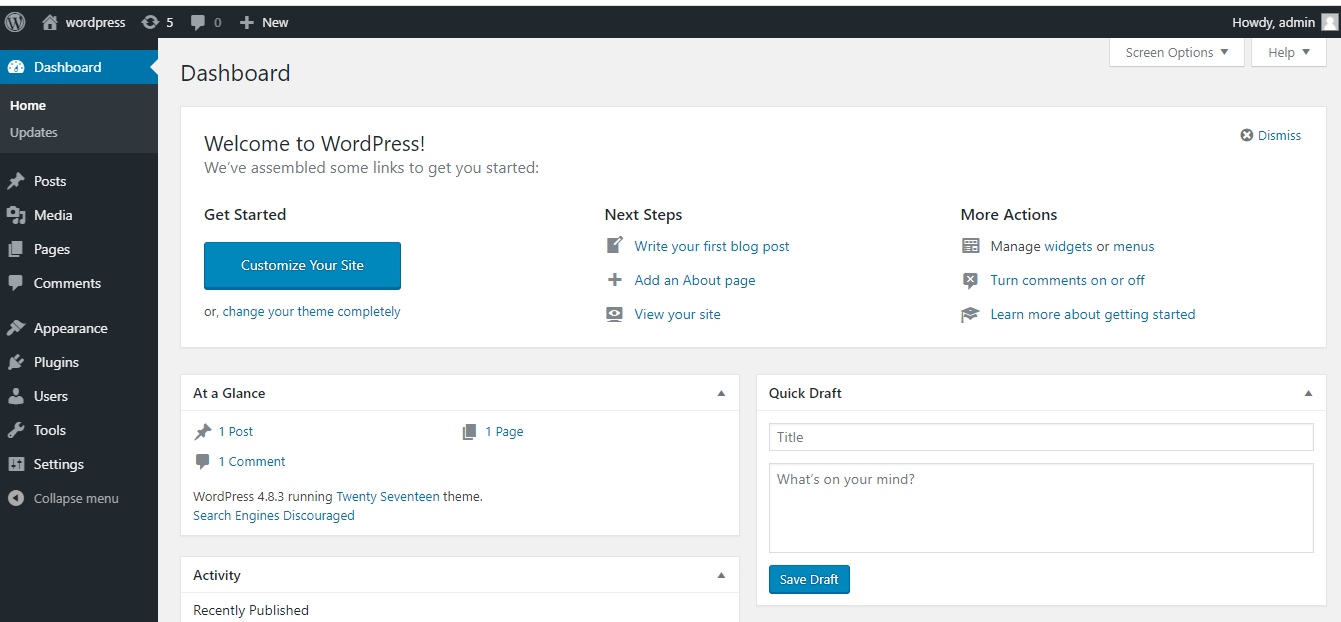


图3.4-2 wordpress首页