Docker分享之镜像制作与K8S简单使用

**作者：张伟**

**日期：2019年9月24日**

目 录

[Docker分享之镜像制作与K8S简单使用 1](#_Toc20399726)

[第1章 Docker镜像制作 4](#_Toc20399727)

[1.1 服务器版本介绍 4](#_Toc20399728)

[1.2 内核信息 4](#_Toc20399729)

[1.3 内存信息 5](#_Toc20399730)

[1.4 查看物理CPU个数、核数、逻辑CPU个数 5](#_Toc20399731)

[1.4.1 查看物理CPU个数 5](#_Toc20399732)

[1.4.2 查看每个物理CPU中core的个数（核数） 5](#_Toc20399733)

[1.4.3 查看逻辑CPU的个数 5](#_Toc20399734)

[1.5 部署Docker 6](#_Toc20399735)

[1.5.1 卸载Docker 6](#_Toc20399736)

[1.5.2 安装Docker 6](#_Toc20399737)

[1.5.2.1 设置镜像仓库 6](#_Toc20399738)

[1.5.2.2 安装软件包以允许apt通过HTTPS使用存储库 6](#_Toc20399739)

[1.5.2.3 添加Docker官方GPG key 6](#_Toc20399740)

[1.5.2.4 建立稳定版仓库 7](#_Toc20399741)

[1.5.2.5 验证Docker 7](#_Toc20399742)

[1.5.2.6 建立docker用户组 8](#_Toc20399743)

[1.5.3 配置镜像加速器 8](#_Toc20399744)

[1.6 使用Dockerfile定制镜像 9](#_Toc20399745)

[1.6.1 书写条件 9](#_Toc20399746)

[1.6.2 相关指令 9](#_Toc20399747)

[1.6.2.1 FROM 9](#_Toc20399748)

[1.6.2.2 MAINTAINER 10](#_Toc20399749)

[1.6.2.3 COPY 10](#_Toc20399750)

[1.6.2.4 ADD 10](#_Toc20399751)

[1.6.2.5 WORKDIR 11](#_Toc20399752)

[1.6.2.6 VOLUME 11](#_Toc20399753)

[1.6.2.7 EXPOSE 12](#_Toc20399754)

[1.6.2.8 ENV 12](#_Toc20399755)

[1.6.2.9 RUN 13](#_Toc20399756)

[1.6.2.10 CMD 13](#_Toc20399757)

[1.6.2.11 ENTRYPOINT 13](#_Toc20399758)

[1.6.3 Docker定制镜像操作 14](#_Toc20399759)

[1.7 推送镜像至远端仓库 16](#_Toc20399760)

[第2章 K8S的基础应用 17](#_Toc20399761)

[2.1 K8S部署 17](#_Toc20399762)

[2.2 本地K8S集群说明 17](#_Toc20399763)

[2.3 创建控制器并管理相关Pod 17](#_Toc20399764)

[2.4 创建Service 18](#_Toc20399765)

[2.5 创建控制器及两个Pod 19](#_Toc20399766)

[2.6 Pod的扩缩容 20](#_Toc20399767)

[2.7 Pod的升级与回滚 20](#_Toc20399768)

[2.7.1 Pod的升级 20](#_Toc20399769)

[2.7.2 Pod的回滚 21](#_Toc20399770)

[第3章 自定义YAML文件 21](#_Toc20399771)

[3.1 自定义Pod 21](#_Toc20399772)

[3.1.1 标签 22](#_Toc20399773)

[3.1.1.1 查看标签 22](#_Toc20399774)

[3.1.1.2 打标签 22](#_Toc20399775)

[3.1.1.3 标签过滤 22](#_Toc20399776)

[3.1.1.4 更改标签 23](#_Toc20399777)

[3.1.1.5 节点标签选择器 23](#_Toc20399778)

[3.2 Deployment控制器 24](#_Toc20399779)

[3.3 创建Service 26](#_Toc20399780)

# Docker镜像制作

## 服务器版本介绍

study@docker:~$ cat /etc/os-release

NAME="Ubuntu"

VERSION="16.04.5 LTS (Xenial Xerus)"

ID=ubuntu

ID\_LIKE=debian

PRETTY\_NAME="Ubuntu 16.04.5 LTS"

VERSION\_ID="16.04"

HOME\_URL="http://www.ubuntu.com/"

SUPPORT\_URL="http://help.ubuntu.com/"

BUG\_REPORT\_URL="http://bugs.launchpad.net/ubuntu/"

VERSION\_CODENAME=xenial

UBUNTU\_CODENAME=xenial

study@docker:~$ lsb\_release -a

No LSB modules are available.

Distributor ID: Ubuntu

Description: Ubuntu 16.04.5 LTS

Release: 16.04

Codename: xenial

## 内核信息

study@docker:~$ uname -r

4.15.0-29-generic

## 内存信息

study@docker:~$ free -m

total used free shared buff/cache available

Mem: 3921 2170 210 61 1540 1371

Swap: 974 0 974

## 查看物理CPU个数、核数、逻辑CPU个数

CPU总核数=物理CPU的个数\*每颗物理CPU的核数

总逻辑CPU数=CPU总核数\*超线程数

### 查看物理CPU个数

study@docker:~$ cat /proc/cpuinfo|grep "physical id"|sort|uniq|wc -l

1

### 查看每个物理CPU中core的个数（核数）

study@docker:~$ cat /proc/cpuinfo |grep "cpu cores"|uniq

cpu cores : 2

### 查看逻辑CPU的个数

study@docker:~$ cat /proc/cpuinfo |grep "processor"|wc -l

2

## 部署Docker

参考文档：<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

### 卸载Docker

老版本的docker称之为docker，docker.io、docker-engine，如果先前下载了需要删除：

$ sudo apt-get remove docker docker-engine docker.io containerd runc

### 安装Docker

安装docker的方式有三种：

1. 使用Docker镜像仓库的方式：便于安装和升级，推荐；
2. 通过DEB安装包的方式安装：适用于无网络的环境；
3. 通过自动化脚本安装：适用于测试和预发布环境。

**本次安装使用第一种方式**，其他两种方式可参考：<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

#### 设置镜像仓库

sudo apt-get update

#### 安装软件包以允许apt通过HTTPS使用存储库

$ sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

#### 添加Docker官方GPG key

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

通过搜索指纹的后8个字符，验证您现在是否拥有带有指纹的密钥

study@docker:~$ sudo apt-key fingerprint 0EBFCD88

pub 4096R/0EBFCD88 2017-02-22

Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid Docker Release (CE deb) <docker@docker.com>

sub 4096R/F273FCD8 2017-02-22

#### 建立稳定版仓库

$ sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io  **#<==安装最新版本的**

**# 选择指定版本的**

apt-cache madison docker-ce

sudo apt-get install docker-ce=<VERSION\_STRING> docker-ce-cli=<VERSION\_STRING> containerd.io

sudo apt-get install docker-ce=5:18.09.6~3-0~ubuntu-xenial docker-ce-cli=5:18.09.6~3-0~ubuntu-xenial containerd.io

#### 验证Docker

sudo docker run hello-world

#### 建立docker用户组

默认情况下，docker命令会使用Unix socket与Docker引擎通讯。而只有root用户和docker组用户才能访问Docker引擎的Unix socket。出于安全考虑，一般Linux系统上不会直接使用root用户，因此，更好地做法是将需要使用docker的用户加入docker用户组。

**建立docker组：**

study@docker:~$ sudo groupadd docker

groupadd: group 'docker' already exists

**将当前用户加入docker用户组：**

study@docker:~$ sudo usermod -aG docker $USER

**退出当前终端并重新登录验证测试。**

### 配置镜像加速器

国内从Docker Hub拉取镜像有时会遇到困难，此时可以配置镜像加速器。Docker官方和国内很多云服务商都提供了国内加速器服务，例如：

* Docker官方提供的中国register mirror（<https://registry.docker-cn.com>）；
* 阿里云镜像加速器（需要账号登录）；
* 七牛云加速器（https://reg-mirror.qiniu.com/）

对于使用systemd系统，请在/etc/docker/daemon.json中写入如下的内容：（如果文件不存在请以新建改文件）

study@docker:~$ cat /etc/docker/daemon.json

{

"registry-mirrors": [

"https://registry.docker-cn.com"

],

"insecure-registries": [

"192.168.1.14"

]

}

study@docker:~$ sudo systemctl daemon-reload

study@docker:~$ sudo systemctl restart docker.service  **#<==重启docker服务**

## 使用Dockerfile定制镜像

### 书写条件

* 必须在某个特定的目录下；
* 文件内的首字母需大写；
* 基于Dockerfile制作镜像时，它所引用的文件的路径应该是该工作目录下的（可以是子目录下）
* 子目录下此次制作镜像不包含在内的文件可以写入到.dockerignore文件内；

**注意：**

基于dockerfile制作镜像时，我们可以在dockerfile中执行很多shell命令，但是该shell命令不是宿主机的，而是底层镜像中所包含的命令，如果底层的镜像中没有包含该命令则在dockerfile文件中执行一些shell命令是做不到的。

### 相关指令

#### FROM

* FROM指令是最重要的一个且**必须为Dockerfile文件开篇的第一个非注释行**。用于为镜像文件构建过程指定基准镜像，后续的指令运行于此基准镜像所提供的的运行环境；
* 基准镜像可以是任何可用的镜像文件，默认情况下，docker build会在docker主机上查找指定的镜像文件，如果不存在就会从Docker Hub Register上拉取所需的镜像文件（如果还找不到指定的镜像文件，docker build会返回一个错误的信息）

Syntax

FROM <repository>[:<tag>]或

FROM <repository>@<digest> digest:镜像的哈希码

<repository>：指定作为base image的名称

<tag>:base image的标签，省略时默认为latest

#### MAINTAINER

* 用于让Dockerfile制作者提供本人的详细信息
* Dockerfile并不限制MAINTAINER指令可在出现的位置，但是推荐将其放置在FROM指令之后

Syntax

MAINTAINER <author’s detail> #<== <author’s detail>可以是任何文本信息，但是约定俗成的使用作者名称及邮件地址

#### COPY

用于从Docker宿主机复制文件至创建的新的镜像文件

Syntax

COPY <src>….<dest>或

COPY [“src”],……”<dest>” #<==在路径中有空白字符时，建议使用这种方式

<src>：要复制的源文件或者目录，必须是build上下文中的路径，不能是其父目录中的文件，支持使用通配符；如果src是目录，则其内部文件或子目录会被递归复制，但src目录自身不会被复制；指定了多个src，或者在src中使用了通配符，则dest必须是一个目录，且必须以”/”结尾

<dest>：目标路径，即正在创建的image的文件的系统路径，建议为<dest>使用绝对路径，否则COPY指定则以WORKDIR为其起始路径；dest不存在会被自动创建，包括其父目录路径

#### ADD

ADD指令类似于COPY指令，ADD支持使用TAR文件和URL路径

Syntax

ADD –chown=<user>:<group> <src>……<dest>或

ADD –chown=<user>:<group> [“<src>”,……”<dest>”]

* 如果<src>为URL且<dest>不以”/”结尾，则<src>指定的文件将被下载并直接被创建为<dest>；如<dest>以”/”结尾，则文件名URL指定的文件将被直接下载并保存为<dest>/<filename>;
* 如果<src>是一个本地文件系统上的压缩格式的tar文件，它将被展开为一个目录，其行为类似于“tar -x”命令；然而通过URL获取到的tar文件将不会自动展开；
* 如果<src>有多个，或其间接或直接使用通配符，则<dest>必须是一个以”/”结尾的目录路径，如果<dest>不以”/”结尾，则其被视为一个普通文件，<src>的内容将被直接写入到<dest>

**注意：**

根据Docker官方的Dockerfile最佳实践文档中要求，**尽可能使用COPY**，因为COPY的语义明确，就是复制文件而已，而ADD则包含了更复杂的功能，其行为也不一定清晰。**ADD最适合的场合就是需要自动解压缩文件**。

#### WORKDIR

用于为Dockerfile中所有的RUN、CMD、COPY和ADD指定设定工作目录

Syntax

WORKDIR <dirpath>

在Dockerfile文件中，WORKDIR指令可出现多次，其路径也可以为相对路径，不过，其是相对此前一个WORKDIR指令指定的路径

另外，WORKDIR也可以调用由ENV指定定义的变量

#### VOLUME

用于在image中创建一个挂载点目录，以挂载Docker host上的卷或其他容器上的卷。

Syntax：

VOLUMN [“<mountpoint1>”,”<mountpoint2>”……]

VOLUMN [“<mountpoint>”]

如果挂载点目录路径下此前有文件存在，docker run命令会在卷挂载完成后将此前的所有文件复制到新挂载的卷中。

VOLUMN /data/mysql/ **#<==镜像文件中挂载的目录**

这里的/data/mysql/目录就会在运行时自动挂载为匿名卷，任何向/data/mysql/中写入的信息都不会纪录进容器存储层， 从而保证了容器存储层的无状态化。**当然，运行时可以覆盖这个挂在设置**，如：

docker run –d –v mydata:/data/mysql/ xxxxxxx

在这行命令中，就使用了mydata这个命名卷挂载到了/data/mysql这个位置，**替代了Dockerfile中定义的匿名卷的挂载配置。**

#### EXPOSE

用于为容器打开指定要监听的端口以实现与外部通信。

Syntax：

EXPOSE <port>[/<protocol>] [<port>[/<protocol>]……] #<==protocol用于指定传输层协议，可为tcp或udp二者之一，默认为TCP协议

EXPOSE指令是声明运行时容器提供服务端口，**这只是一个声明，在运行时并不会因为这个声明应用就会开启这个端口的服务。**在Dockerfile中写入这样的声明有两个好处：

一个是帮助镜像使用者理解这个镜像服务的守护端口，以便配置映射；

二是在运行时使用随机端口映射时，也就是docker run –P时，会自动随机映射EXPOSE的端口。

要将EXPOSE和在运行时使用**-p <宿主端口>：<容器端口>**区分开来。-p是映射宿主机端口和容器端口，就是将容器的对应端口服务公开给外界访问，而EXPOSE仅仅是声明容器打算是用什么端口而已，并不会主动在宿主进行端口映射。

#### ENV

用于为镜像定义所需要的环境变量，可被Dockerfile文件中位于其后的其他指令（如ADD、COPY）所调用；

Syntax:

ENV <key>=<value> #<==<key>之后的所有内容均会被视为其<value>的组成部分，一次只能设置一个变量；

ENV <key1>=<value1> <key2>=<value2>…… #<==可一次设置多个变量，如果<value>中包含空格，可以以反斜线（\）进行转义，也可以通过对<value>加引号进行标识

ENV VERSION=1.0 NAME=”Happy Time” **#<==对含有空格的值使用双引号括起来的办法**

#### RUN

用于指定docker build过程中运行的程序，可以是任何命令

Syntax：

RUN <Command> #<==就像直接在命令行输入命令一样

Exec格式：RUN [“可执行文件”, “参数1”，“参数2”]

#### CMD

类似于RUN命令，CMD指令也可以运行任何命令或应用程序 ，不过，二者的运行时间点不同：

* **RUN指令运行于映像文件构建过程中**，而**CMD指令运行于基于Dockerfile构建出的新映像文件启动一个容器时**；
* CMD指令的首要目的在于为启动的容器指定默认要运行的程序，且其运行结束后，容器也将终止，不过CMD指定的命令其可以被docker run命令行选项所覆盖；
* **在Dockerfile中可以存在多个CMD指令，但仅最后一个会生效**。

CMD <Command> 🡸🡺CMD [“sh”,”-c”,”Command”]

CMD [”<executable>”,”<patam1>”,”<patam2>”]

CMD [“<param1>”,”<param2>”]

#### ENTRYPOINT

CMD的第三种参数：

* 类似CMD指令的功能，用于为容器指定默认运行程序，从而使得容器像是一个单独的可执行程序；
* 与CMD不同，**由ENTRYPOINT启动的程序不会被docker run命令指定的参数所覆盖，而且这些命令行参数会被当做参数传递给ENTRYPOINT指定的程序**
* 不过**docker run命令的—entrypoint选项的参数可覆盖ENTRYPOIINT指定指定的程序**
* Docker run 命令传入的命令参数会覆盖CMD指令的内容并且附加到ENTRYPOINT命令最后作为其参数使用
* Dockerfile文件中也可以存在多个ENTRYPOINT指令，**但仅最后一个生效**

Syntax：

ENTRYPOINT <Command>

ENTRYPOINT [“<executable>”,”<param1>”,”param2”]

当指定了ENTRYPOINT后，CMD的含义就发生了改变，不再是直接运行其命令，而是将CMD的内容作为参数传给ENTRYPOINT指令，换句话说实际执行将变为：

<ENTRYPOINT>”<CMD>”

### Docker定制镜像操作

study@docker:~$ sudo mkdir /docker/images -p && sudo chown -R study.study /docker/images

study@docker:~$ cd /docker/images/

study@docker:/docker/images$ ls

Dockerfile index.html nginx-1.15.2.tar.gz

study@docker:/docker/images$ cat Dockerfile

FROM centos:7

MAINTAINER "zhangwei <zhangiwei@163.com>"

ENV WEB\_VERSION="1.15.2"

# 将nginx包放入/usr/local/src并自动解压

ADD nginx-${WEB\_VERSION}.tar.gz /usr/local/src

# 改变当前目录到nginx安装目录

WORKDIR /usr/local/src/nginx-${WEB\_VERSION}

# 安装依赖包 && 创建用户

RUN yum install -y gcc gcc-c++ glibc make autoconf openssl openssl-devel libxslt-devel -y gd gd-devel GeoIP GeoIP-devel pcre pcre-devel && \

useradd -M -s /sbin/nologin nginx && \

./configure --user=nginx --group=nginx --prefix=/usr/local/nginx --with-file-aio --with-http\_ssl\_module --with-http\_realip\_module --with-http\_addition\_module --with-http\_xslt\_module --with-http\_image\_filter\_module --with-http\_geoip\_module --with-http\_sub\_module --with-http\_dav\_module --with-http\_flv\_module --with-http\_mp4\_module --with-http\_gunzip\_module --with-http\_gzip\_static\_module --with-http\_auth\_request\_module --with-http\_random\_index\_module --with-http\_secure\_link\_module --with-http\_degradation\_module --with-http\_stub\_status\_module && \

make && \

make install

COPY index.html /usr/local/nginx/html/

EXPOSE 80

#添加环境变量

ENV PATH /usr/local/nginx/sbin:$PATH

#entrypoint的作用是，把整个container变成了一个可执行的文件，这样不能够通过替换CMD的方法来改变创建container的方式。

#但是可以通过参数传递的方法影响到container内部,每个Dockerfile只能够包含一个entrypoint

#当定义了entrypoint以后，CMD只能够作为参数进行传递

ENTRYPOINT ["nginx"]

#相当于在容器执行：nginx -g "daemon off;"

CMD ["-g","daemon off;"]

study@docker:/docker/images$ cat index.html

<h1>Hello Myweb1.</h1>

docker build -t myweb:v1 .

docker images|grep myweb

myweb v2 151faac03a39 6 minutes ago 443MB

myweb v1 328c209af573 10 hours ago 488MB

docker run -d -P --name myweb myweb:v1 启动镜像

docker exec -it 877e5506e96b /bin/sh 进入容器文件 877e5506e96b为容器ID（docker ps -a获取）

## 推送镜像至远端仓库

docker login --username=惜你疼伊 registry.cn-hangzhou.aliyuncs.com

Password:

WARNING! Your password will be stored unencrypted in /home/study/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

docker tag myweb:v1 registry.cn-hangzhou.aliyuncs.com/zhangiwei/myweb:v1

docker push registry.cn-hangzhou.aliyuncs.com/zhangiwei/myweb:v1

docker save myweb:v1 -o myweb.tar

docker load -i myweb.tar

# K8S的基础应用

## K8S部署

详见：<http://192.168.1.4:8888/MS/OPS/mservice-k8s-foundation/blob/master/installation/%E7%B1%B3%E8%B5%A2K8S%E6%80%BB%E7%BB%93%E6%96%87%E6%A1%A3.docx>

## 本地K8S集群说明

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **IP地址** | **系统** | **登录账户** |
| k8s-master1 | 192.168.0.50 | CentOS 7.5.1804 | root/abc123\_ |
| k8s-master2 | 192.168.0.90 | CentOS 7.5.1804 | root/abc123\_ |
| k8s-master3 | 192.168.0.98 | CentOS 7.5.1804 | root/abc123\_ |
| k8s-node1 | 192.168.0.94 | CentOS 7.5.1804 | root/abc123\_ |
| K8s-node2 | 192.168.0.209 | CentOS 7.5.1804 | root/abc123\_ |

## 创建控制器并管理相关Pod

[root@k8s-master1 ~]# kubectl run -h

[root@k8s-master1 ~]# kubectl run nginx-deployment --image=nginx **#<==创建控制器名称为nginx-deployment，基于nginx镜像**

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-deployment created

[root@k8s-master1 ~]# kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 1/1 1 1 69m

[root@k8s-master1 ~]# kubectl get pods|grep nginx

nginx-deployment-69f748fb86-7hcnc 1/1 Running 0 69m

[root@k8s-master1 ~]# kubectl get pods nginx-deployment-69f748fb86-7hcnc -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

nginx-deployment-69f748fb86-7hcnc 1/1 Running 0 71m 10.244.5.4 k8s-node2 <none> <none>

[root@k8s-node1 ~]# curl -s 10.244.5.4 –I **#<==从集群其他节点访问**

HTTP/1.1 200 OK

Server: nginx/1.17.4

Date: Thu, 26 Sep 2019 00:28:48 GMT

Content-Type: text/html

Content-Length: 612

Last-Modified: Tue, 24 Sep 2019 14:49:10 GMT

Connection: keep-alive

ETag: "5d8a2ce6-264"

Accept-Ranges: bytes

[root@k8s-master1 ~]# kubectl delete pods nginx-deployment-69f748fb86-7hcnc **#<==当删除一个pod时deployment会自动拉起一个pod**

pod "nginx-deployment-69f748fb86-7hcnc" deleted

[root@k8s-master1 ~]# kubectl get pods|grep nginx

nginx-deployment-69f748fb86-xrjt9 0/1 ContainerCreating 0 15s

## 创建Service

[root@k8s-master1 ~]# kubectl expose –h

# Create a service for an nginx deployment, which serves on port 80 and connects to the containers on port 8000.

kubectl expose deployment nginx --port=80 --target-port=8000

[root@k8s-master1 ~]# kubectl expose deployment nginx-deployment --name=nginx-service --port=80 --target-port=80 --protocol=TCP

service/nginx-service exposed

**注：**

deployment nginx-deployment为deployment的名称

--name=nginx-service 为service的名称

--port=80 为service的port

--target-port=80 为pod的port

[root@k8s-master1 ~]# kubectl get svc

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 61d

nginx-service ClusterIP 10.104.25.251 <none> 80/TCP 77s

[root@k8s-master1 ~]# curl -I 10.104.25.251

HTTP/1.1 200 OK

Server: nginx/1.17.4

Date: Mon, 29 Jul 2019 15:12:14 GMT

Content-Type: text/html

Content-Length: 612

Last-Modified: Tue, 24 Sep 2019 14:49:10 GMT

Connection: keep-alive

ETag: "5d8a2ce6-264"

Accept-Ranges: bytes

## 创建控制器及两个Pod

[root@k8s-master1 ~]# kubectl run myapp --image=ikubernetes/myapp:v1 --replicas=2

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/myapp created

[root@k8s-master1 ~]# kubectl get pods|grep myapp

myapp-5bc569c47d-v8th5 1/1 Running 0 73s

myapp-5bc569c47d-zzt2r 1/1 Running 0 73s

[root@k8s-master1 ~]# kubectl expose deployment myapp --name=myapp-service --port=80

service/myapp-service exposed

[root@k8s-master1 ~]# kubectl get svc|grep myapp

myapp-service ClusterIP 10.107.104.127 <none> 80/TCP 29s

## Pod的扩缩容

[root@k8s-master1 ~]# kubectl scale --replicas=5 deployment myapp

deployment.extensions/myapp scaled

[root@k8s-master1 ~]# kubectl get pods |grep myapp -w

myapp-5bc569c47d-6xm4t 1/1 Running 0 11s

myapp-5bc569c47d-bsgkm 1/1 Running 0 11s

myapp-5bc569c47d-sx486 1/1 Running 0 11s

myapp-5bc569c47d-v8th5 1/1 Running 0 5m25s

myapp-5bc569c47d-zzt2r 1/1 Running 0 5m25s

## Pod的升级与回滚

### Pod的升级

[root@k8s-master1 ~]# kubectl describe pods myapp-5bc569c47d-v8th5

[root@k8s-master1 ~]# kubectl set image deployment myapp myapp=ikubernetes/myapp:v2

deployment.extensions/myapp image updated

[root@k8s-master1 ~]# kubectl rollout status deployment myapp **#<==查看容器升级状况**

deployment "myapp" successfully rolled out

[root@k8s-master1 ~]# kubectl get pods -w -o wide

### Pod的回滚

[root@k8s-master1 ~]# kubectl rollout undo deployment myapp

deployment.extensions/myapp rolled back

[root@k8s-master1 ~]# curl 10.244.3.11

Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>

# 自定义YAML文件

## 自定义Pod

[root@k8s-master1 ~]# mkdir /srv/k8s/ -p

[root@k8s-master1 ~]# cd /srv/k8s/

[root@k8s-master1 k8s]# cat pod-demo.yml

apiVersion: v1

kind: Pod

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80 **#<==容器的端口信息，不定义并不影响容器的端口暴露**

[root@k8s-master1 k8s]# kubectl apply -f pod-demo.yml

pod/pod-demo created

### 标签

#### 查看标签

[root@k8s-master1 k8s]# kubectl get pods --show-labels

NAME READY STATUS RESTARTS AGE LABELS

busybox 1/1 Running 3 58d <none>

pod-demo 1/1 Running 0 7m58s app=myapp

#### 打标签

[root@k8s-master1 k8s]# kubectl label pods pod-demo tier=fronted

pod/pod-demo labeled

[root@k8s-master1 k8s]# kubectl get pods --show-labels

NAME READY STATUS RESTARTS AGE LABELS

busybox 1/1 Running 3 58d <none>

pod-demo 1/1 Running 0 11m app=myapp,tier=fronted

#### 标签过滤

[root@k8s-master1 k8s]# kubectl get pods -l tier **#<==-l:对应的pod对象上拥有标签tier**

NAME READY STATUS RESTARTS AGE

pod-demo 1/1 Running 0 11m

#### 更改标签

[root@k8s-master1 k8s]# kubectl label pods pod-demo tier=backend --overwrite

pod/pod-demo labeled

[root@k8s-master1 k8s]# kubectl get pods -o wide --show-labels

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES LABELS

busybox 1/1 Running 3 58d 10.244.3.5 k8s-node1 <none> <none> <none>

pod-demo 1/1 Running 0 15m 10.244.5.13 k8s-node2 <none> <none> app=myapp,tier=backend

#### 节点标签选择器

[root@k8s-master1 k8s]# kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

busybox 1/1 Running 4 58d 10.244.3.5 k8s-node1 <none> <none>

pod-demo 1/1 Running 0 21m 10.244.5.13 **k8s-node2** <none> <none>

[root@k8s-master1 k8s]# kubectl label nodes k8s-node1 disktype=ssd

node/k8s-node1 labeled

[root@k8s-master1 k8s]# tail -3 pod-demo.yml

containerPort: 80

nodeSelector:

disktype: ssd

[root@k8s-master1 k8s]# kubectl delete -f pod-demo.yml

pod "pod-demo" deleted

[root@k8s-master1 k8s]# kubectl apply -f .

pod/pod-demo created

[root@k8s-master1 k8s]# kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

busybox 1/1 Running 4 58d 10.244.3.5 k8s-node1 <none> <none>

pod-demo 1/1 Running 0 9s 10.244.3.13 k8s-node1 <none> <none>

## Deployment控制器

[root@k8s-master1 k8s]# kubectl delete -f pod-demo.yml

pod "pod-demo" deleted

[root@k8s-master1 k8s]# cat pod-deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

namespace: default

spec:

replicas: 2

selector:

matchLabels:

app: myapp

template:  **#<==kubectl explain deployment.spec.template**

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80

nodeSelector:

disktype: ssd

[root@k8s-master1 k8s]# kubectl apply -f .

deployment.apps/myapp-deployment created

[root@k8s-master1 k8s]# kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

busybox 1/1 Running 4 58d 10.244.3.5 k8s-node1 <none> <none>

myapp-deployment-65cbc5f4d7-m8wrm 1/1 Running 0 95s 10.244.3.15 k8s-node1 <none> <none>

myapp-deployment-65cbc5f4d7-s69c2 1/1 Running 0 95s 10.244.3.14 k8s-node1 <none> <none>

如果将nodeSelector注释掉的后果：

[root@k8s-master1 k8s]# kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

busybox 1/1 Running 4 58d 10.244.3.5 k8s-node1 <none> <none>

myapp-deployment-544dff57d5-fxtbc 1/1 Running 0 14s 10.244.5.14 **k8s-node2** <none> <none>

myapp-deployment-544dff57d5-whchm 1/1 Running 0 11s 10.244.3.16 **k8s-node1** <none> <none>

## 创建Service

[root@k8s-master1 k8s]# cat pod-deployment.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp-deployment

namespace: default

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

name: pod-demo

namespace: default

labels:

app: myapp

spec:

containers:

- name: myapp

image: ikubernetes/myapp:v1

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 80

#nodeSelector:

# disktype: ssd

---

apiVersion: v1

kind: Service

metadata:

name: myapp-service

namespace: default

spec:

selector:

app: myapp

clusterIP: 10.99.99.99

type: Nodeport

ports:

- port: 80 **#<==表示为Service对外提供服务的端口**

targetPort: 80 **#<==表示为后端容器上的端口**

nodePort: 30000 **#<==映射到node节点上的端口**

[root@k8s-master1 k8s]# kubectl apply -f .

deployment.apps/myapp-deployment unchanged

service/myapp-service created

[root@k8s-master1 k8s]# kubectl get svc

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

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 61d

myapp-service NodePort 10.99.99.99 <none> 80:30000/TCP 14s

nginx-service ClusterIP 10.104.25.251 <none> 80/TCP 166m

[root@k8s-master1 k8s]# curl 192.168.56.101:30000/hostname.html

myapp-deployment-544dff57d5-fxtbc

[root@k8s-master1 k8s]# curl 192.168.56.101:30000/hostname.html

myapp-deployment-544dff57d5-whchm

[root@k8s-master1 k8s]# curl 192.168.56.101:30000/hostname.html

myapp-deployment-544dff57d5-48x4p

[root@k8s-master1 k8s]# curl 192.168.56.101:30000/hostname.html

myapp-deployment-544dff57d5-fxtbc