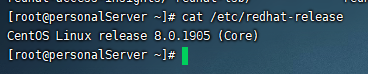
**13.2持续集成·Flask生产环境部署-Docker版**

本次操作是在阿里云上操作的，版本如下：



# 安装Docker

### 安装yum-utils工具

yum install -y yum-utils libseccomp-devel

### 添加docker源

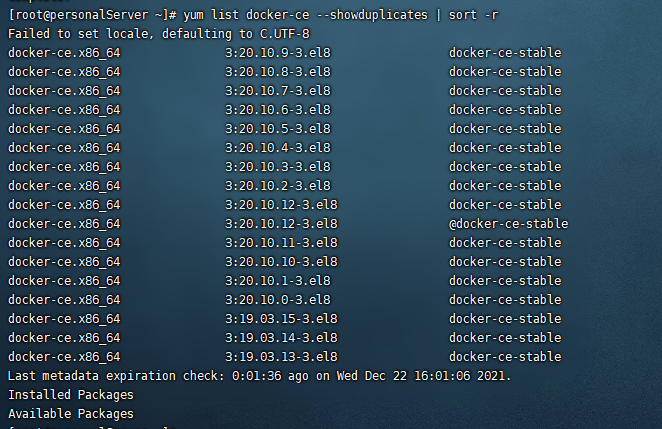
yum-config-manager \  
 --add-repo \  
 https://download.docker.com/linux/centos/docker-ce.repo

### 安装docker engine

yum install docker-ce docker-ce-cli containerd.io

### 查看docker源版本列表

yum list docker-ce --showduplicates | sort -r



### 安装指定版本

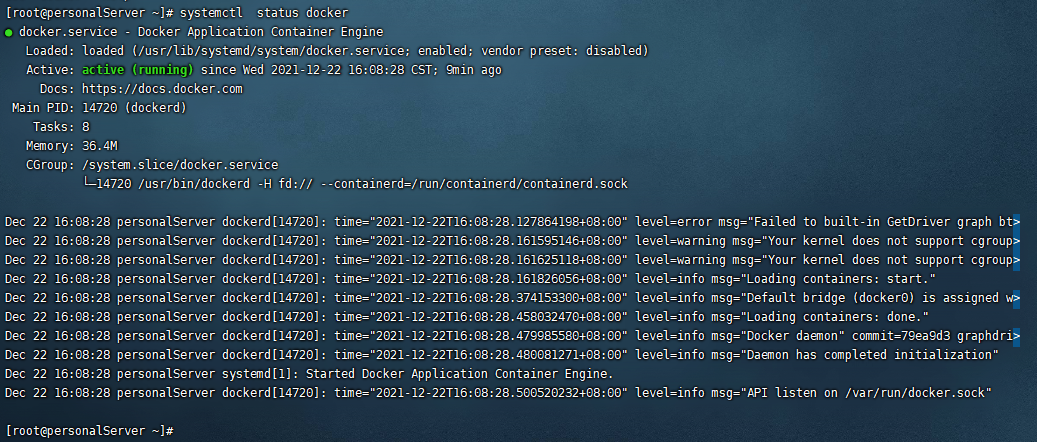
yum install docker-ce-20.10.9 docker-ce-cli-20.10.9 containerd.io

### 启动服务

systemctl start docker

### 查看启动状态

systemctl status docker



### 设置开机启动

systemctl enable docker

### 验证安装是否成功

跑一个nginx容器

* 创建目录

mkdir -p /home/docker/nginx/html

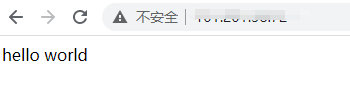
* 新增文件

echo "hello world" > /home/docker/nginx/html/index.html

* 运行nginx实例

docker run -p 80:80 -v /home/docker/nginx/html:/usr/share/nginx/html -d --name mynginx nginx

* 浏览器访问ip访问显示 hello world即为成功。



# 停止刚运行的名称为mynginx的实例  
docker ps -a | grep mynginx | awk '{print $1}' | xargs docker stop  
# 删除刚运行的名称为mynginx的实例  
docker ps -a | grep mynginx | awk '{print $1}' | xargs docker rm

# 安装docker-compose工具

使用该工具更加方便编排docker.

## 下载

curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

该工具在国内下载会比较缓慢，甚至无法下载，如果下载不了，可以多试几次，如果实在无法下载，可以通过其他途径获取。

## 设置可执行权限

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

# 使用Docker安装Mysql

## 常规梳理

### 关于镜像

这里选择mysql:5.7.28

### 关于环境变量

* MYSQL\_ROOT\_PASSWORD：数据库root密码
* MYSQL\_DATABASE：默认创建的数据库
* MYSQL\_USER：默认创建的用户名，可访问上述创建的数据库
* MYSQL\_PASSWORD：默认创建用户的密码

这是经验所得，更详细的说明可去dockerhub搜相关镜像了解。

### 关于挂载目录及文件

* /var/lib/mysql：mysql数据目录
* /etc/my.cnf：mysql配置文件
* /docker-entrypoint-initdb.d/ ：初始化后，默认执行的sql脚本目录，即存放有sql脚本，会自动执行。用于初始化数据库的。

### 关于服务端口

* 3306

### 定义文件样例

version: '2'  
services:  
 mysql:  
 environment:  
 MYSQL\_ROOT\_PASSWORD: "8Eli#gr#AUk"  
 MYSQL\_DATABASE: "app"  
 MYSQL\_USER: "u\_app"  
 MYSQL\_PASSWORD: "8Eli@gr#668"  
 image: "mysql:5.7.28"  
 restart: always  
 volumes:  
 - "/etc/localtime:/etc/localtime:ro" # 与宿主时钟同步  
 - "/home/docker/mysql/db:/var/lib/mysql"  
 - "/home/docker/mysql/conf/my.cnf:/etc/my.cnf"  
 - "/home/docker/mysql/init:/docker-entrypoint-initdb.d/"  
 ports:  
 - "3306:3306"

* services：服务定义，每个key对应一个服务
* environment：环境变量定义
* image：镜像
* restart：重启策略

1. no，默认策略，在容器退出时不重启容器
2. on-failure，在容器非正常退出时（退出状态非0），才会重启容器
3. on-failure:3，在容器非正常退出时重启容器，最多重启3次
4. always，在容器退出时总是重启容器
5. unless-stopped，在容器退出时总是重启容器，但是不考虑在Docker守护进程启动时就已经停止了的容器

* volumes：挂载目录及文件，左边为宿主，右边为容器
* ports：端口映射，左边宿主，右边容器

## 开始安装

### 初始化目录

mkdir -p /home/docker/mysql/db  
mkdir -p /home/docker/mysql/conf  
mkdir -p /home/docker/mysql/init

### 添加配置文件

cat <<EOF > /home/docker/mysql/conf/my.cnf  
[mysqld]  
user=mysql  
wait\_timeout=2880000  
interactive\_timeout = 2880000  
max\_allowed\_packet = 256M  
default-storage-engine=INNODB  
character-set-server=utf8mb4  
default-time\_zone = '+8:00'  
lower\_case\_table\_names=1  
[client]  
default-character-set=utf8mb4  
[mysql]  
default-character-set=utf8mb4  
  
EOF

### 添加定义文件

cat <<EOF > /home/docker/mysql/docker-compose.yml  
version: '2'  
services:  
 mysql:  
 environment:  
 MYSQL\_ROOT\_PASSWORD: "8Eli#gr#AUk"  
 MYSQL\_DATABASE: "app"  
 MYSQL\_USER: "u\_app"  
 MYSQL\_PASSWORD: "8Eli@gr#668"  
 image: "mysql:5.7.28"  
 restart: always  
 volumes:  
 - "/etc/localtime:/etc/localtime:ro"  
 - "/home/docker/mysql/db:/var/lib/mysql"  
 - "/home/docker/mysql/conf/my.cnf:/etc/my.cnf"  
 - "/home/docker/mysql/init:/docker-entrypoint-initdb.d/"  
 ports:  
 - "3306:3306"  
EOF

### 运行实例

* 后台运行实例   
  下面命令可暂时跳过，直接跳到**连接验证**

docker-compose -f /home/docker/mysql/docker-compose.yml up -d

### 其他命令

* 停止运行中的实例

docker-compose -f /home/docker/mysql/docker-compose.yml stop

* 开启运行中的实例

docker-compose -f /home/docker/mysql/docker-compose.yml start

* 重启运行中的实例

docker-compose -f /home/docker/mysql/docker-compose.yml restart

* 移出实例

docker-compose -f /home/docker/mysql/docker-compose.yml down

* 非后台运行实例   
  最常用的是 docker-compose -f docker-compose.yml up -d与docker-compose -f docker-compose.yml down

docker-compose -f /home/docker/mysql/docker-compose.yml up

* 其他命令查看

docker-compose -h

### 连接验证

使用Navicat等数据库连接工具验证。

## 数据库常规维护

传统的安装在宿主机上数据库，可直接登录主机维护。而使用docker后，会略微有些区别，会多加一步，进入容器。

如果仅启动了一个mysql实例，则可以使用如下命令进入容器。

docker exec -it $(docker ps -a | grep mysql | awk '{print $1}') bash

其实就是：

docker exec -it containerid bash

查看容器id命令

docker ps -a

# 使用Docker安装Redis

# 制作Docker基础镜像

制作一个可以运行Flask项目的镜像

## 创建目录

mkdir -p /home/docker/python3.8-uwsgi

## 新增构建文件

cat <<EOF > /home/docker/python3.8-uwsgi/Dockerfile  
# 指定基础镜像  
FROM python:3.8.12-slim  
# 维护者信息  
MAINTAINER mldong <524719755@qq.com>  
# 安装 uWSGI  
RUN sed -i s/deb.debian.org/mirrors.aliyun.com/g /etc/apt/sources.list \  
 && apt-get update && apt-get install -y \  
 libc6-dev \  
 musl-dev \  
 libffi-dev \  
 python3-dev \  
 gcc \  
 mime-support \  
 && pip install --no-cache-dir uWSGI>=2.0.20 \  
 && apt-get remove -y \  
 gcc \  
 libc6-dev \  
 && rm -rf /var/lib/apt/lists/\*  
EOF

## 进入目录

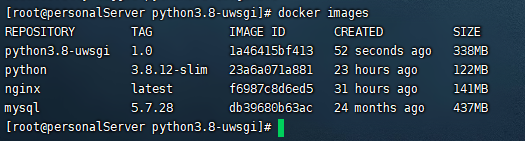
cd /home/docker/python3.8-uwsgi

## 开始构建

docker build -t python3.8-uwsgi:1.0 -f /home/docker/python3.8-uwsgi/Dockerfile .

## 查看构建的镜像

docker images



# 项目部署

记项目名为：career-reborn

## 使用docker部署Flask项目有两种方案

### 方案一

每一个版本的代码，基于基础镜像重新构建，即一版本一镜像，然后运行。

优点：

* 每个镜像版本相对独立，也方便使用镜像仓库去维护各个版本。
* 因为镜像版本都是可运行版本，遇到问题可方便回滚
* 持续集成主流的方式
* 流程更加规范

缺点：

* 每个版本的镜像会相对大，相对占用空间（不过可以使用脚本定时清理）
* 打包发布多了构建镜像的时间，频繁发包更新的话，会略微影响效率

### 方案二

基于基础镜像重新构建包含requirements.txt依赖的新镜像，运行时将代码挂载到容器中

优点：

* 更接近于传统的部署方式（更新代码、重启服务即可）
* 流程相对简化

缺点：

* 当requirements.txt依赖文件更新时，运行的镜像也要即时更新，要不然容易报错。
* 如集群环境，需要有磁盘共享机制，如nfs等

因为只有一台服务器，相对简单，所以这里使用方案二去部署Flask项目。

## 目录说明

/home/docker/career-reborn

|-- Dockerfile # 镜像构建定义文件  
|-- docker-compose.yml # docker-compose服务编排文件  
|-- rebuild.sh # 重新构建镜像并重启服务脚本  
|-- requirements.txt # 项目的依赖文件  
|-- restart.sh # 更新代码后，重启服务脚本  
`-- uwsgi.ini # uwsgi运行所需要的配置文件

## 各文件内容如下

* Dockerfile

# 基于上面的基础镜像构建  
FROM python3.8-uwsgi:1.0  
# 进入工作目录  
WORKDIR /career-reborn  
# 将当前目录文件添加到工作目录  
ADD . /career-reborn  
# 基于requirements.txt安装项目依赖  
RUN pip3 install -r requirements.txt -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com  
# 容器暴露出来的端口-即flask服务的端口  
EXPOSE 5000  
# 容器运行时启动的命令  
CMD ["uwsgi","uwsgi.ini"]

* docker-compose.yml

version: '2'  
services:  
 zyghapp:  
 env\_file: .env # 服务名称  
 container\_name: zyghapp # 容器名称  
 build: # 重新构建镜像  
 context: . # 当前目录  
 dockerfile: Dockerfile # 定义文件名  
 image: "zyghapp:1.0" # 镜像版本  
 restart: always # docker重启后，该容器也重启  
 volumes: # 目录挂载  
 - "/etc/localtime:/etc/localtime:ro" # (左为宿主，右为容器)  
 - "/home/source/career-reborn:/career-reborn" # 将源代码挂载到容器中  
 - "/home/docker/career-reborn/uwsgi.ini:/career-reborn/uwsgi.ini" # 将uwsgi.ini配置文件挂载到容器中  
 - "/c/so/:/c/so/" # 将/c/so/目录挂载到容器中  
 ports:  
 - "5000:5000" # 端口映射(左为宿主，右为容器)

注：关于目录持载，大家按需挂载

* rebuild.sh

# 进入源代码目录  
cd /home/source/career-reborn/  
# 拉取最新代码（要先安装git,yum install -y git）  
git pull  
# 将依赖文件复制到项目工作空间  
cp -rf requirements.txt /home/docker/career-reborn/  
# 进入项目工作空间  
cd /home/docker/career-reborn/  
# 停止服务  
docker-compose down  
# 重新构建镜像  
docker-compose build --no-cache  
# 删除为none的镜像->垃圾镜像  
docker rmi $(docker images | grep "none" | awk '{print $3}')  
# 后台运行容器  
docker-compose up -d

* requirements.txt

略，这个根据项目情况

* restart.sh

# 进入源代码目录  
cd /home/source/career-reborn/  
# 拉取最新代码（要先安装git,yum install -y git）  
git pull  
# 进入项目工作空间  
cd /home/docker/career-reborn/  
# 重启服务  
docker-compose restart

* uwsgi.ini

[uwsgi]  
http=:5000  
wsgi-file=app.py  
callable=app  
master=True  
processes = 2  
threads = 20  
max-requests=5000

# 一些维护命令

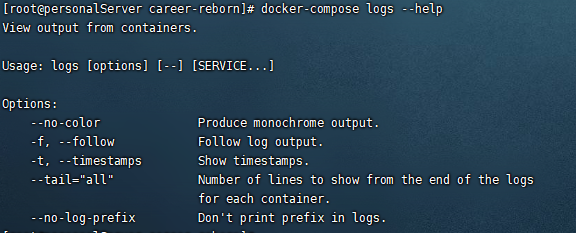
命令要先进入项目工作空间再执行

cd /home/docker/career-reborn

## 查看运行日志

### 查看帮助

docker-compose logs --help



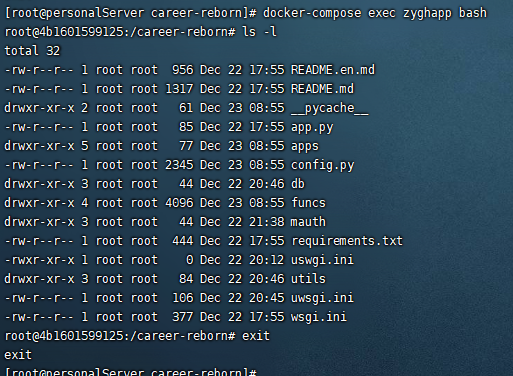
### 查看实时日志

docker-compose logs -f -t --tail=100

## 进入容器

docker-compose exec zyghapp bash

其中zyghapp为docker-compose.yml定义的services的key



退出容器使用exit

## 重新构建镜像并重启服务

如requirements.txt文件有变动，需要执行该脚本

./rebuild.sh

## 更新代码并重启服务

如requirements.txt文件无变动，可执行该脚本，简化部署流程，可以在流水线上配置钩子，代码变动后执行该脚本。

./restart.sh