docker

背景

- **docker是什么**

docker是用GO开发的应用容器引擎，基于容器化，沙箱机制的应用部署技术。

Docker 容器是

利用Linux Namespace做资源隔离，

Cgroup做资源上限限制，

rootfs做文件系统

运行在宿主机上面的一个特殊进程。

- **为什么使用docker**

利用docker可以将项目和依赖包(基础镜像)打成一个带有启动指令的项目镜像，然后在宿主机创建一个容器，让镜像在容器内运行，从而实现项目的部署。

项目可以是mysql、nginx、nodejs、redis等，从而可以在一台宿主机上快速搭建相互隔离的不同的环境。

基础概念

镜像（image)、容器(container)、仓库（repository)

举个例子，电脑安装操作系统，称其为镜像，镜像是一个固定的文件，这次读盘和下次读盘内容是一样的。

把这个镜像安装到电脑上，再在这个电脑上删删改改，再重新打包一个镜像刻盘，固化出一个镜像来，这就是镜像打包，

如国内以前泛滥的盗版XP系统，从微软官方镜像出发，添加小工具，系统设置修改优化，加主题，造出番茄花园，雨林木风，深度之类的盗版安装碟。

镜像装到电脑后，这个电脑就是个容器，里面包含使用者的数据和设置。

- **镜像**

镜像可以看成是由多个镜像层叠加起来的一个文件系统；

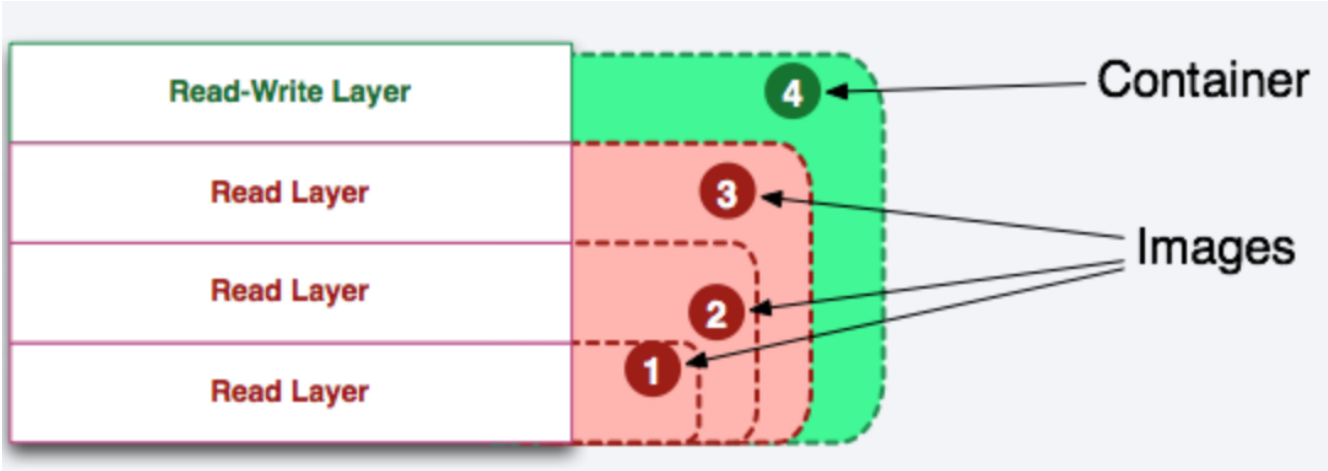
镜像层的主要组成部分包括镜像层 ID、镜像层指针 「指向父层」 、元数据；

镜像是一堆只读层的统一视角，除了最底层没有指向外，每一层都指向它的父层。统一文件系统（ AUFS）能够将不同的层整合成一个文件系统。

- **容器**

容器的定义和镜像几乎一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。

要点：容器 = 镜像 + 读写层



- **仓库**

顾名思义，docker 仓库是用来存储镜像的，理解时，可以参考 git 仓库。

Docker 环境安装，以安装 docker-ce 为例

1．Linux 安装

1）下载阿里的源

```
Markdown
```

```
1 wget -O /etc/yum.repos.d/docker-ce.repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
2 sudo yum makecache
3 sudo yum install docker-ce
```

2）docker 启动

```
Markdown
```

```
1 systemctl start docker.service
```

2. Windows 安装

https://www.runoob.com/docker/windows-docker-install.html

下载 Docker Desktop

启用 Hyper-V(win10)

安装 Docker Desttop

常用命令

1. 搜索镜像

```
Markdown
```

```
1 docker search xxx
```

2. 拉取镜像

```
1 docker pull xxx
```

3. 查看镜像

```
1 docker images
```

4. 运行镜像

```
1 docker run
2     -p 暴露端口:容器端口          #端口
3     -v 本地路径:容器内路径        #挂在本地路径
4     -e 环境变量=value             #指定环境变量
5     --add-host 域名:IP          # 绑定域名
```

-p 暴露端口：容器端口

-v 本地路径：容器内路径

-e 环境变量 =value

5. 查看运行中的容器

   docker ps

6. 容器启动、停止、重启

   docker container start | stop | restart [containername]

启动一个 nginx 服务器

C/C++

```
# 检索nginx镜像
docker search nginx --limit 5
# 拉取镜像
docker pull nginx:stable
#采用默认配置启动一个服务器
docker run -p 9080:80 --name my-nginx nginx:stable
#采用自定义配置，并挂载外部目录
docker run -d -p 9081:81 -v /home/luanhongjun/temp1/demo/nginx/conf.d:/etc/nginx/conf.d -v /home/luanhongjun/temp1/demo/nginx/html:/usr/share/nginx/html --name my-nginx nginx:stable
```

自定义打包镜像

1. Dockerfile

Markdown

```
FROM khub.wps.cn/sreopen/node:14.2.0

ADD ./ /opt/zhiliao

WORKDIR /opt/zhiliao

RUN npm install
RUN npm run build:pc
RUN npm run build:mobile

FROM khub.wps.cn/sreopen/nginx:1.18.0

COPY --from=0 /opt/zhiliao/dist/ /var/www/zhiliao/
COPY --from=0 /opt/zhiliao/m-dist/ /var/www/m-zhiliao/
COPY --from=0 /opt/zhiliao/nginx/nginx.conf /etc/nginx/

RUN nginx
CMD ["nginx","-g","daemon off;"]
```

FROM：指定基础镜像

ADD：将本地文件添加到容器中，tar 类型文件会自动解压(网络压缩资源不会被解压)，可以访问网络资源，类似 wget。

WORKDIR： 指定工作目录

RUN：构建镜像时运行的命令

COPY：功能类似 ADD，但是是不会自动解压文件，也不能访问网络资源

CMD：构建容器后调用，也就是在容器启动时才进行调用。

记住：

```Markdown
1 在 Dockerfile 中， 每一条指令都会创建一个镜像层，继而会增加整体镜像的大小。
```

2. Docker 镜像瘦身技巧：

1）善用 &&

优化前

```Markdown
1 RUN npm install
2 RUN npm run build:pc
3 RUN npm run build:mobile
```

优化后

```Markdown
1 RUN /bin/bash -c 'npm install && npm run build:pc && npm run build:mobi
  le'
```

2）使用多阶段构建

3）选择 Alpine/Distroless 基础镜像

Alpine Linux 是一个基于 musl libc 和 busybox、面向安全的轻量级 Linux 发行版。它更小、更安全。

Markdown

```
FROM khub.wps.cn/zl-open/golang-base:1.14.2 AS build-link

ADD wps-zhiliao-backend-mics-known-link /opt/kingsoft/src/wps.cn/drive/link
ADD wps-zhiliao-backend-mics-known-lib /opt/kingsoft/src/wps.cn/drive/lib
ADD wps-zhiliao-backend-lib /opt/kingsoft/src/wps.cn/lib
ADD kgo-kms /opt/kingsoft/src/ksogit.kingsoft.net/kgo/kms
ADD openksa-cksdk  /opt/kingsoft/src/wps.cn/cksdk
ADD kgo-i18n /opt/kingsoft/src/wps.cn/drive/i18n

WORKDIR /opt/kingsoft/src/wps.cn/drive/link

ENV GOPATH=/opt/kingsoft/:/opt/kingsoft/src/wps.cn/lib/Godeps/_workspace
ENV CGO_ENABLED=0

RUN go build -v -o /opt/kingsoft/src/wps.cn/drive/link/bin/main /opt/kingsoft/src/wps.cn/drive/link/main.go

FROM khub.wps.cn/sreopen/sre-base:pyjinja-supervisor-alpine AS run-link

COPY --from=0 /opt/kingsoft/src/wps.cn/drive/link/bin/main /usr/local/bin/link
COPY --from=0 /opt/kingsoft/src/wps.cn/drive/i18n /opt/apps/link/i18n/
COPY --from=0 /opt/kingsoft/src/wps.cn/drive/link/etc/link.kae.toml.j2 /etc/link.kae.toml.j2
COPY --from=0 /opt/kingsoft/src/wps.cn/drive/link/kae_start.sh /usr/local/bin/kae_start.sh
#EXPOSE 8080

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/g' /etc/apk/repositories \
    && apk add --no-cache tzdata \
    && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
    && echo "Asia/Shanghai" > /etc/timezone \
    && chmod +x /usr/local/bin/kae_start.sh
```

```
30
31 CMD ["kae_start.sh"]
32
```

打包

docker build . -t xxx:yy

docker-compose 的使用

- **简介**

Docker-Compose 项目是 Docker 官方的开源项目，负责实现对 Docker 容器集群的快速编排。

Docker-Compose 项目由 Python 编写，调用 Docker 服务提供的 API 来对容器进行管理。

Docker Compose 将所管理的容器分为三层，分别是工程（project）、服务（service）、容器（container）

Docker Compose 运行目录下的所有文件（docker-compose.yml）组成一个工程，一个工程包含多个服务，每个服务中定义了容器运行的镜像、参数、依赖，一个服务可包括多个容器实例

- **安装**

方式一：下载适应版本的 compose

C/C++

```
1 sudo curl -L https://github.com/docker/compose/releases/download/1.21.2/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
2 sudo chmod +x /usr/local/bin/docker-compose
```

方式二：

C/C++

```
1 安装python-pip
2 pip install docker-compose
```

- **docker-compose.yml 简介**

标准配置文件应该包含 `version` 、 `services` 、 `networks` 三部分：

version: 定义了版本信息

services: 定义了服务的配置信息，包含应用于该服务启动的每个容器的配置

networks: 定义了网络信息，提供给 services 中的 具体容器使用

- **实例**

C/C++

```
version: '3.2'
services:
  #Nginx
  nginx_lb:
    restart: always
    image: nginx:1.11.6-alpine
    ports:
      - 28080:80
      - 80:80
      - 443:443
    volumes:
      - /opt/kingsoft/containers/nginx/conf.d:/etc/nginx/conf.d
      - /opt/kingsoft/containers/nginx/log:/var/log/nginx
      - /opt/kingsoft/containers/nginx/www:/var/www
      - /opt/kingsoft/containers/nginx/zhiliao-management-system:/var/zhiliao-management-system
      - /opt/kingsoft/containers/nginx/ssl:/etc/nginx/ssl
    extra_hosts:
      - "msgcenter.wps.cn:120.92.124.158"
    stdin_open: true
    tty: true
  # Redis cluster
  #redis_db:
  #  image: grokzen/redis-cluster:5.0.12
  #  environment:
  #    TZ: "Asia/Shanghai"
  #  volumes:
  #    - /opt/kingsoft:/opt/kingsoft
  #  # docker network bridge mode
  #  networks:
  #      vpcbr:
  #      ipv4_address: 172.16.0.50
  #  ports:
  #    - "7000:7000"
  # MySQL DB
  mysql_db:
    image: percona:5.6
```

```
37      environment:
38        MYSQL_ROOT_PASSWORD: xxxx
39        TZ: "Asia/Shanghai"
40      volumes:
41       - /opt/kingsoft/containers/mysql/conf:/etc/mysql/conf.d
42        - /opt/kingsoft/containers/mysql/data:/var/lib/mysql
43        - /opt/kingsoft:/opt/kingsoft
44      ports:
45        - "3306:3306"
46      networks:
47        vpcbr:
48          ipv4_address: 172.16.0.51
49    zoo1:
50      image: wurstmeister/zookeeper
51      ports:
52        - "2181:2181"
53      networks:
54        vpcbr:
55          ipv4_address: 172.16.0.56
56
57    # kafka version: 1.1.0
58    # scala version: 2.12
59    kafka:
60      image: wurstmeister/kafka
61      ports:
62        - "9092:9092"
63      environment:
64        KAFKA_ADVERTISED_HOST_NAME: kafka
65        KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
66        KAFKA_BROKER_ID: 1
67        KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
68        KAFKA_CREATE_TOPICS: "drive_fsys_fop:1:1"
69      depends_on:
70        - zoo1
71      networks:
72        vpcbr:
73          ipv4_address: 172.16.0.57
74
75  networks:
76    vpcbr:
```

```
77      driver: bridge
78      ipam:
79       config:
80         - subnet: 172.16.0.0/16
81
```

结合 KAE 使用

绑定 120.92.124.158 khub.wps.cn

docker tag xximage:xxtag

docker login khub.wps.cn

docker push khub.wps.cn/zl-open/golang-base:1.14.2

问题

1. 如何从一个容器做一个镜像

C/C++

```
1 docker commit命令创建镜像
2 docker commit xxxcontaineridd imagename:tag
```

2. docker logs 能否输出 nginx 的 access.log

C/C++

```
1 docker logs xxx 只能输出容器内程序输出到终端（stdout|stderr）的日志。无法输出写
  入到文件中的日志。所以不能直接输出nginx的access.log
2 当然，如果使用一些hook程序，将access.log内的日志实时输出到终端，可以通过docker lo
  gs看到。
```