Docker

第一章 初识 Docker

1.1 Docker概念

- Docker是一个开源的应用容器引擎
- 诞生于2013年初，基于Go语言实现，dotCloud公司出品（后改名Docker Inc）
- Docker可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的Linux机器上
- 容器是完全使用沙箱机制，相互隔离
- 容器性能开销极低

Docker可以运行在在MAC、Windows、CentOS、UBUNTU等操作系统上

官网：`https://www.docker.com`

1.2 安装Docker

1. yum包更新到最新

```
yum update
```

2. 安装需要的软件包，yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

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

3. 设置yum源

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

4. 安装docker，出现输入的界面都按 y

```
yum install -y docker-ce
```

5. 查看docker版本，验证是否成功

```
docker -v
```

1.3 Docker架构

- 镜像（Image）：Docker镜像（Image），就相当于是一个root文件系统。比如官方镜像 ubuntu:16.04就包含了完整的一套Ubuntu16.04最小系统的root文件系统
- 容器（Container）：镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和对象一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等
- 仓库（Repository）：仓库可以看成一个代码控制中心，用来保存镜像

1.4 配置Docker镜像加速器

默认情况下，将从docker hub（https://hub.docker.com/）上下载docker镜像，太慢。一般都会配置镜像加速器

- USTC：中科大镜像加速器（https://docker.mirrors.ustc.edu.cn）
- 阿里云
- 网易云
- 腾讯云

在这里我们用阿里云镜像加速器，登陆阿里云查看个人加速地址

注意：用阿里云时，每个人家加速器地址不同，然后在终端输入下面指令

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://eob5atc6.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker

###上面操作完成后输入下面代码测试是否成功
 cat /etc/docker/daemon.json
```

第二章 Docker 命令

2.1 Docker 服务相关命令

- 启动docker 服务：
  - ```
    systemctl start docker
    ```
- 停止docker 服务：
  - ```
    systemctl stop docker
    ```
- 重启docker 服务：
  - ```
    systemctl restart docker
    ```
- 查看docker 服务状态：
  - ```
    systemctl status docker
    ```
- 设置开机启动docker：
  - ```
    systemctl enable docker
    ```

2.2 Docker 镜像相关命令

- 查看镜像：查看本地所有的镜像
  - ```
    docker images
    docker images -q #查看所有镜像的id
    ```
- 搜索镜像：从网络中查找需要的镜像
  - ```
    docker search 镜像名称
    ```
- 拉取镜像：从Docker 仓库下载镜像到本地，镜像名称格式为`名称:版本号`，如果版本号不指定则是最新的版本。如果不知道镜像版本，可以去docker hub 搜索对应镜像查看
  - ```
    docker pull 镜像名称
    ```
- 删除镜像：删除本地镜像

- ```
  docker rmi 镜像id/名称号:版本号 #删除指定本地镜像
  docker rmi 'docker images -q' #删除所有本地镜像
  ```

2.3 Docker 容器相关命令

- 查看容器
  - ```
    docker ps #查看正在运行的容器
    docker ps -a #查看所有容器
    ```
- 创建并启动容器
  - ```
    docker run 参数 版本:版本号 </bin/bash>#默认为/bin/bash
    ```
  - 参数说明：
    - `-i`：保持容器运行。通常与`-t`同时使用。加入`it`这两个参数后，容器创建后自动进入容器中，退出容器后，容器自动关闭
    - `-t`：为容器重新分配一个伪输入终端，通常与`-i`同时使用
    - `-d`：以守护（后台）模式运行容器。创建一个容器在后台运行，需要使用`docker exec`进入容器`docker exec -it c2 /bin/bash`。退出后，容器不会关闭
    - `-it`创建的容器一般称为交互式容器；`-id`创建的容器一般称为守护式容器
    - `--name`：为创建的容器命名
- 进入容器
  - ```
    docker exec 参数 #退出容器，容器不会关闭
    ```
- 停止容器
  - ```
    docker stop 容器名称
    ```
- 启动容器
  - ```
    docker start 容器名称
    ```
- 删除容器：如果容器是运行状态则删除失败，需要停止容器参能删除
  - ```
    docker rm 容器名称
    ```
- 查看容器信息
    ```

- `docker inspect 容器名称`

第三章 Docker 容器的数据卷

3.1 数据卷概念及作用

思考：

- Docker 容器删除后，在容器中产生的数据还在吗？
- Docker 容器和外部继器可以交换文件吗？
- 容器之间想要进行数据交互？

数据卷

- 数据卷是宿主机中的一个目录或文件
- 当容器目录和数据卷目录绑定后，对方的修改会立即同步
- 一个数据卷可以被多个容器同时挂载
- 一个容器也可以被挂载多个数据卷

数据卷的作用

- 容器数据持久化
- 外部计价器和容器间接通信
- 容器之间数据交换

3.2 配置数据卷

- 创建启动容器时，使用`-v`参数 设置数据卷
  - `docker run ... -v 宿主机目录(文件):容器内目录(文件) ...`
- 注意事项：
  1. 目录必须是绝对路径
  2. 如果目录不存在，会自动创建
  3. 可以挂载多个数据卷

3.3 配置数据卷容器

多容器进行数据交换：

1. 多个容器挂载同一个数据卷
2. 数据卷容器

配置数据卷容器：

1. 创建启动c3数据卷容器，使用`-v`参数 设置数据卷

```
docker run -it --name=c3 -v /volume centos:7 /bin/bash
```

2. 创建启动c1 c2容器，使用`--volumes-from`参数 设置数据卷

```
docker run -it --name=c1 --volumes-from c3 centos:7 /bin/bash
docker run -it --name=c2 --volumes-from c3 centos:7 /bin/bash
```

第四章 Docker 应用部署

4.1 MySQL部署

4.1.1案例：需求

- 在Docker 容器中部署MySQL，并通过外部MySQL 客户端操作MySQL Server

4.1.2 案例：实现

1. 搜索mysql镜像
2. 拉取mysql镜像
3. 创建容器
4. 操作容器中的mysql

4.1.3 问题及解决方案

- 容器内的网络服务和外部机器不能直接通信
- 外部机器和宿主机可以直接通信
- 宿主机和容器可以直接通信

- 当容器中的网络服务需要被外部机器访问时，可以将容器中提供服务的端口映射到宿主机的端口上。外部机器访问宿主机的端口，从而间接访问容器的服务
- 这种操作称为：端口映射

4.1.4 部署MySQL

1. 搜索mysql镜像

```
docker search mysql
```

2. 拉取mysql镜像

```
docker pull mysql:5.6
```

3. 创建容器，设置端口映射、目录映射

```
# 在/root目录下创建mysql目录用于存储mysql数据信息
mkdir ~/mysql
cd ~/mysql
```

```
docker run -id \
-p 3307:3306 \
--name=c_mysql \
-v $PWD/conf:/etc/mysql/conf.d \
-v $PWD/logs:/logs \
-v $PWD/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=123456 \
mysql:5.6
```

   - 参数说明
     - `-p 3307:3306`：将容器的3306端口映射到宿主机的3307端口
     - `--v $PWD/conf:/etc/mysql/conf.d`：将主机当前目录下的`conf/my.cnf`挂载到容器`/etc/mysql/my.cnf`配置目录
     - `-v $PWD/logs:/logs`：将主机当前目录下的`logs`目录挂载到容器的`/logs`目录日志
     - `-v $PWD/data:/var/lib/mysql`：将主机当前目录下的`data`目录挂载到容器的`/var/lib/mysql`数据目录
     - `-e MYSQL_ROOT_PASSWORD=123456`：初始化root 用户密码
4. 使用外部机器访问MySQL

4.2 Tomcat部署

1. 搜索Tomcat 镜像

```
docker search tomcat
```

2. 拉取mysql镜像

```
docker pull tomcat
```

3. 创建容器，设置端口映射、目录映射

```
#在/root目录下创建tomcat目录用于存放tomcat数据信息
mkdir ~/tomcat
cd ~/tomcat
```

```
docker run -id --name=c_tomcat \
-p 8080:8080 \
-v $PWD:/usr/local/tomcat/webapps \
tomcat
```

   - 参数说明：
     - `-p 8080:8080`：将容器的8080端口映射到主机的8080端口
     - `-v $PWD:/usr/local/tomcat/webapps`：将主机中当前目录挂载到容器的`webapps`

4. 最后使用外部机器访问tomcat

4.3 Nginx部署

1. 搜索Nginx 镜像

```
docker search nginx
```

2. 拉取mysql镜像

```
docker pull nginx
```

3. 创建容器，设置端口映射、目录映射

```
#在/root目录下创建nginx目录用于存储nginx数据信息
mkdir ~/nginx
cd ~/nginx
mkdir conf
cd conf
#在~/nginx/conf/下创建nginx.conf文件，粘贴下面代码块的内容
vim nginx.conf
```

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user
[$time_local] "$request" '
                      '$status $body_bytes_sent
"$http_referer" '
                      '"$http_user_agent"
"$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

```
docker -run -id --name=c_nginx \
-p 80:800 \
-v $PWD/conf/nginx.conf:/etc/nginx/nginx.conf \
-v $PWD/logs:/var/log/nginx \
-v $PWD/html:/usr/share/nginx/html \
nginx
```

- 参数说明：
  - `-p 80:80`：将容器的80端口映射到宿主机的80端口
  - `-v $PWD/conf/nginx.conf:/etc/nginx/nginx.conf`：将主机当前目录下的`/conf/nginx.conf`挂载到容器的`:/etc/nginx/nginx.conf`配置目录
  - `-v $PWD/html:/usr/share/nginx/html`：将主机当前目录挂载到容器的`/val/log/nginx`日志目录

4. 使用外部机器访问nginx

4.4 Redis部署

1. 搜索Redis 镜像

```
docker search redis
```

2. 拉取Redis 镜像

```
docker pull redis:5.0
```

3. 创建容器，设置端口映射、目录映射

```
docker run -id --name=c_redis -p 6379:6379 redis:5.0
```

4. 使用外部机器连接redis

```
redis-cli.exe -h 192.168.187.129 -p 6379
```

第五章 Dockerfile

5.1 Docker 镜像原理

思考：

- Docker 镜像的本质是什么?
  - 是一个分层的文件系统
- Docker 中一个CentOS 镜像为什么只有200MB，而一个CentOS 操作系统的iso文件要几个G?
  - CentOS的iso镜像文件包含bootfs和rootfs，而Docker的CentOS镜像复用操作系统的bootfs，只有rootfs和其他镜像层
- Docker 中一个Tomcat 镜像为什么有500MB，而一个Tomcat 安装包只有70多MB?
  - 由于Docker中镜像是分层的，tomcat虽然只有70多MB，但他需要依赖于父镜像和基础镜像，所以整个对外暴露的tomcat镜像大小500多MB

操作系统组成部分：

- 进程调度子系统
- 进程通信子系统
- 内存管理子系统
- 设备管理子系统
- 文件管理子系统
- 网络通信子系统
- 作业控制子系统

Linux文件系统由bootfs 和rootfs 两部分组成

- bootfs：包含bootloader（引导加载程序）和kernel（内核）
- rootfs：root文件系统，包含的就是典型的Linux 系统中的/dev、/proc、/bin等标准目录和文件
- 不同的Linux 发行版，bootfs 基本一样，而rootfs 不同，如ubuntu，CentOS等

Docker 镜像原理：

- Docker 镜像是由特殊的文件系统叠加而成
- 最低端是bootfs，并使用宿主机的bootfs
- 第二层是root 文件系统rootfs ，称为base iamge
- 然后再往上可以叠加其他的镜像文件
- 统一文件系统（Union File System）技术能够将不同的层整合成一个文件系统，为这些层提供了一个统一的视角，这样就隐藏了多层的存在，在用户的角度看来，只存在一个文件系统
- 一个镜像可以放在另一个镜像的上面。位于下面的镜像称为父镜像，最底部的镜像称为基础镜像

- 当从一个镜像启动容器时，Docker会在最顶层加载一个读写文件系统作为容器

镜像制作：

- 容器转为镜像

```
docker commit 容器id 镜像名称:版本号
```

```
docker save -o 压缩文件名称 镜像名称:版本号
```

```
docker load -i 压缩文件名称
```

5.2 Dockerfile 概念及作用

Dockerfile 概念

- Dockerfile 是一个文本文件
- 包含了一条条的指令
- 每一条指令构建一层，基于基础镜像，最终构建出一个新的镜像
- 对于开发人员，可以为开发团队提供一个完全一致的开发环境
- 对于测试人员，可以直接拿开发时所构建的镜像或者通过Dockerfile 文件构建一个新的镜像开始工作了
- 对于运维人员，在部署时，可以实现应用的无缝移植

5.3 Dockerfile 关键字

关键字	作用	备注
FROM	指定父镜像	指定dockerfile基于哪个images构建
MAINTAINER	作者信息	用来标明这个dockerfile 谁写的
LABEL	标签	用来指明dockerfile 的标签，可以使用Label代替Maintainer 最终都是在docker image基本信息中嗯可以查看
RUN	执行命令	执行一段命令 默认是`/bin/sh` 格式：`RUN command` 或者 `RUN ["command","param1","param2"]`
CMD	容器启动命令	提供启动容器时候的默认命令和ENTRYPOINT配合使用。格式：`CMD command param1 param2` 或者`CMD ["command","param1","param2"]`
ENTRYPOINT	入口	一般在制作一些执行就关闭的容器中会使用
COPY	复制文件	build 的时候复制文件到image中
ADD	添加文件	build 的时候添加文件到iamge 中，不仅仅局限于当前build 上下文 可以来源于远程服务

关键字	作用	备注
ENV	环境变量	指定build 时候的环境变量 可以在启动容器的时候 通过`-e`覆盖 格式：`ENV name = value`
ARG	构建参数	构建参数 只在构建的时候使用参时 如果有ENV 那么ENV 的相同名字的值始终覆盖ARG 的值
VOLUME	定义外部可以挂载的数据卷	指定build 的image 那些目录可以启动的时候挂载到文件系统中 启动容器的时候使用`-v`绑定 格式：`VOLUME ["目录"]`
EXPOSE	暴露端口	定义容器运行的时候监听的端口 启动容器的使用`-p`来绑定暴露端口 格式：`EXPOSE 8080`或者`EXPOSE 8080/udp`
WORKDIR	工作目录	指定容器内部的工作目录 如果没有创建则自动创建 如果指定/使用是绝对地址 如果不是/开头那么实在上一条workdir 的路径的相对路径
USER	指定执行用户	指定build 或者启动的时候 用户 在RUN CMD ENTRYPOINT执行的时候的用户
HEALTHCHECK	健康检查	指定监测当前容器的健康测试的命令 基本上没有因为很多时候 应用本身由健康监测机制
ONBUILD	触发器	当存在ONBUILD 关键字的镜像作为基础镜像的时候 当执行FROM 完成之后 会执行ONBUILD的命令 但是不影响当前镜像 用处也不怎么大
STOPSIGNAL	发送信息量到宿主机	该STOPSIGNAL指令设置将发送到容器的系统调用信号以退出
SHELL	指定执行脚本的shell	指定RUN CMD ENTRYPOINT 执行命令的时候 使用的shell

5.4 案例

5.4.1 案例一

需求：

自定义CentOS7镜像。要求：

1. 默认登录路径为`/usr`
2. 可以使用vim

实现步骤：

1. 定义父镜像：`FROM centos:7`
2. 定义作者信息：`MAINTAINER crisp077 <www.crisp077.xyz>`
3. 执行安装vim命令：`RUN yum install -y vim`
4. 定义默认的工作目录：`WORKDIR /usr`
5. 定义容器启动执行的命令：`CMD /bin/bash`

创建使用dockerfile的镜像：

```
docker build -f ./centos_docker -t crisp_centos:1 .
```

5.4.2 案例二

需求：

定义dockerfile，发布springboot 项目

实现步骤：

1. 定义父镜像：`FROM java:8`
2. 定义作者信息：`MAINTAINER crisp077 <www.crisp077.xyz>`
3. 将jar包添加到容器：`ADD springboot.jar app.jar`
4. 定义容器启动执行的命令：`CMD java -jar app.jar`
5. 通过dockerfile 构建镜像：`docker build -f dockerfile文件路径 -t 镜像名称:版本`

第六章 Docker 服务编排

6.1 服务编排的概念

微服务架构的应用系统中一般包含若干个微服务，每个微服务都会部署多个实例，如果每个微服务都要手动启动，维护工作量会很大

- 要从Dockerfile build image 或者去 dockerhub 拉取image
- 要创建多个container
- 要管理这些container（启动停止删除）

服务编排：

按照一定的业务规则批量管理容器

6.2 Dockers Compose 概述

Docker Compose 是一个编排多容器分布式部署的工具，提供命令集管理器化应用的完整开发期，包括服务构建，启动和停止。使用步骤：

1. 利用 Dockerfile 定义运行环境镜像
2. 使用 docker-compose.yml 定义组成应用的各服务
3. 运行 docker-compose up 启动应用

6.2.1 安装Docker Compose

```
#Compose 目前已经完全支持Linux、MAC OS、Windows，在安装Compose之前，需要先安装Docker。下面以编译好的二进制包方式安装在Linux中
curl -L
https://github.com/docker/compose/releases/download/1.22.0/docker-compose-'uname -s'-'uname -m' -o /usr/local/bin/docker-compose
#设置文件可执行权限
chmod +x /usr/local/bin/docker-compose
#查看版本信息
docker-compose -version
```

6.2.2 卸载Docker Compose

```
#二进制包方式安装的，删除二进制文件即可
rm /usr/local/bin/docker-compose
```

6.3 案例

使用docker compose编排nginx+springboot项目

1. 创建docker-compose目录

```
mkdir ~/docker-compose
cd ~/docker-compose
```

2. 编写`docker-compose.yml`文件

```
version: '3'
services:
  nginx:
   iamge: nginx
   ports:
    - 80:80
   links:
    - app
   volumes:
    - ./nginx/conf.d:/etc/nginx/conf.d
   app:
     image: app
     expose:
       - "8080"
```

3. 创建`./nginx/conf.d`目录

```
mkdir -p ./nginx/conf.d
```

4. 在`./nginx/conf.d`目录下编写`crisp.conf`文件

```
server {
    listen 80;
    acess_log off;

    location / {
        proxy_pass https://app:8080;
    }

}
```

5. 在`~/docker-compose`目录下 使用dockers-compose 启动容器

```
docker-compose up
```

6. 测试访问

```
http://192.168.187.129/hello
```

第七章 Docker 私有仓库

7.1 搭建私有仓库

Docker 私有仓库

- Docker 官方的Docker hub（https://hub.docker.com）是一个用于管理公共镜像的仓库，我们可以从上面拉取镜像到本地，也可以把我们自己的镜像推送上去。但是，有时候我们的服务器无法访问互联网，或者你不希望将自己的镜像放到公网当中，那么我们就需要搭建自己的私有仓库来存储和管理自己的镜像。

私有仓库搭建

1. 拉取私有仓库镜像

```
docker pull registry
```

2. 启动私有仓库容器

```
docker run -id --name=registry -p 5000:5000 registry
```

3. 打开浏览器，输入地址`https://私有仓库服务器ip:5000/v2/_catalog`看到`{"repositories":[]}`表示私有仓库搭建成功

4. 修改`deamon.json`

```
vim /etc/docker/daemon.json
#在上述文件中添加一个key，保存退出。
#此步用于让docker信任私有仓库地址
#注意将私有仓库服务器ip修改为自己私有仓库服务器真实ip
{"insecure-registries": ["私有仓库服务器ip:5000"]}
```

5. 重启docker 服务

```
systemctl restart docker
docker start registry
```

7.2 上传镜像到私有仓库

1. 标记镜像为私有仓库的镜像

```
docker tag centos:7 私有仓库服务器ip:5000/centos:7
```

2. 上传标记的镜像

```
docker push 私有仓库服务器ip:5000/centos:7
```

7.3 从私有仓库拉取镜像

```
#拉取镜像
docker pull 私有仓库服务器ip:5000/centos:7
```

第八章 Docker 相关概念

Docker容器虚拟化 与 传统虚拟机比较

容器就是将软件打包成标准化单元，以用于开发、交付和部署

- 容器镜像是轻量级的、可执行的独立软件包，包含软件运行所需要的所有内容：代码、运行时环境、系统工具、系统库和设置
- 容器化软件在任何环境中都能够始终如一地运行
- 容器赋予了软件独立性，使其免受外在环境差异的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突

相同：

- 容器和虚拟机具有相似的资源隔离和分配优势

不同：

- 容器虚拟化的是操作系统，虚拟机虚拟化的时硬件
- 传统的虚拟机可以运行不同的操作系统，容器只能运行同一类型的操作系统

特性	容器	虚拟机
启动	秒级	分钟级
性能	接近原生	弱于
系统支持两	单机支持上千个容器	一般几十个

> 本讲义由B站菜鸟程序员视频教程`BV1CJ411T7BK`整理优化得到