1. Docker镜像原理
2. Linux文件系统

Linux文件系统由bootfs和rootfs两部分组成，fs表示file system文件系统。bootfs包含bootloader（引导加载程序）和kernel（内核）；rootfs是root文件系统，包含典型Linux系统中的/dev, /proc, /bin, /etc等标准目录和文件。而不同的Linux发行版本，如ubuntu，centos，bootfs基本是一样的，只是rootfs不同。

1. Docker镜像原理

Docker镜像是由特殊的文件系统叠加而成的，最底层是bootfs，没错，docker使用的都是宿主机的bootfs，这也解释了为什么docker创建容器如此地快速。在bootfs之上叠加rootfs文件系统，称为基础镜像，如ubuntu, centos。再往上叠加其他的镜像文件，如JDK镜像，继续叠加Tomcat镜像。

当我们下载Tomcat镜像时，它就会首先查看是否有rootfs基础镜像，再查看是否有JDK镜像，如果都没有，就下载rootfs，JDK，Tomcat三个镜像文件，而对于用户来说，它只是知道下载了Tomcat这一个镜像文件。这种技术称为统一文件系统，能够将不同的层整合为一个文件系统，为这些层提供一个统一的视角，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。

那么，为什么要用层级的结构呢？这是因为层级的结构利于镜像的复用，当其他的镜像依赖于我们之前已经下载并且加载完毕的镜像时，就无须再次下载了。位于下面的镜像称为父镜像，最底层的镜像称为基础镜像。

1. 修改镜像

当从一个镜像中启动容器时，Docker会在最顶层加载一个读写文件系统作为容器，那么我们就可以在这个容器中对镜像进行修改，修改完成后，再将该容器作为一个新的镜像。

1. Docker问题解答
2. Docker 镜像的本质是什么？

分层文件系统。

1. Docker中的一个centos镜像为什么只有200MB，而一个centos操作系统的iso文件有几个G？

因为iso文件包括bootfs和rootfs，而docker的centos镜像复用了操作系统的bootfs，只有rootfs和其他镜像层，因此大小少了很多。

1. Docker中一个tomcat镜像为什么有500MB，而一个tomcat的安装包只有70MB？

由于docker中镜像是分层的，tomcat虽然只有70MB，但是它需要依赖父镜像和基础镜像，所以整个对外暴露的tomcat镜像大小为500MB。

1. Dockerfile
2. 镜像制作

如何制作一个镜像呢？可以将一个容器进行想要的修改，修改完成后，将该容器通过以下命令转为镜：

docker commit 容器id 镜像名称：版本号

docker commit 123452e2 new\_Tomcat:1.0

如果想将自定义的镜像传递给其他人，需要将镜像进行压缩，然后传递压缩包，收到压缩包的人加压后就可以直接将镜像部署为容器，直接使用。

镜像压缩命令：

docker save -o 压缩文件名称 镜像名称：版本号

镜像解压命令：

docker load -I 压缩文件名称

注意，用这种直接压缩镜像的方式，数据卷的挂载是不能一起复制过去的。

1. Dockerfile
2. 概述

Dockerfile是一个包含了很多指令的文件，每一条指令都会构建一层镜像，基于基础镜像，最终构建出一个新的镜像。这个文件对于真正的开发生产是非常高效的，能够为开发团队提供一个完全一致的开发环境，同时对于测试运维人员，能够直接使用开发时构建的镜像或者Dockerfile文件构建一个新的镜像，无缝衔接。

2． 实现自定义centos镜像

编写Dockerfile基于centos进行自定义镜像。文件内容如下：

1. 定义父镜像：FROM centos:7
2. 定义作者信息：MAINTAINER wxx
3. 执行安装vim命令： RUN apt-get install 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

编写完Dockfile后，使用如下build命令生成镜像，-f表示当前的Dockerfile文件的路径，-t表示生成的镜像名称与版本，最后还要加上镜像的寻址路径。

docker build -f ./centos\_dockerfile -t wxx:1 .

命令执行后，会按照dockerfile文件中的步骤一层层地构建镜像，下载依赖。

1. Docker compose
2. 概述

之前我们讨论的都是容器数量比较少的情况，当在开发中有大量的镜像，容器需要部署时，这个管理维护工作量会非常大，特别是微服务架构的系统中一般会有许多微服务，每个微服务都会部署多个实例，这时就需要Docker compose来管理服务的编排了。

1. 服务编排

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

首先通过Dockerfile定义运行环境镜像；

然后使用Docker Compose提供的docker-compose.yml设置需要的各个服务以及它们之间的启动顺序，依赖关系；

最后通过运行docker-compose up命令启动应用，可以一次性地启动所有配置的服务。

1. Docker compose的安装
2. 下载Docker Compose

curl -L https://get.daocloud.io/docker/compose/releases/download/1.24.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

1. 设置文件可执行权限

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

1. 检查版本

docker-compose -v

1. 使用Docker Compose编排nginx+springboot项目
2. 创建docker-compose目录

mkdir ./docker-compose

cd ./docker-compose

1. 编写docker-compose.yml文件

version: ‘3’

services:

nginx:  
 image: nginx

ports:

* 80:80

links:

* app

volumes:

* ./nginx/conf.d:/etc/nginx/conf.d

app:

image: app

expose:

* “8080”

yml格式的文件采用缩进表示层级关系，image表示镜像，ports表示端口映射，links表示链接到app服务，volumes表示数据卷的映射。expose表示暴露的端口号，nginx链接到app服务中。

1. 创建./nginx/conf.d目录

mkdir -p ./nginx/conf.d

1. 在./nginx/conf.d目录下编写wxx.conf文件

wxx.conf文件是用于配置nginx的反向代理访问app：

server{

listen 80;

access\_log off;

location /{

proxy\_pass http://app:8080

}

}

1. 在./docker-compose 目录下使用docker-compose启动容器

docker-compose up

1. Docker私有仓库
2. 概述

通常我们是从Docker hub上面拉取镜像，这是一个用于管理公共镜像的仓库。有时我们不希望让人访问到自己的镜像，那么就需要搭建自己的私有仓库来存储和管理自己的镜像。

1. 创建私有仓库
2. 拉取私有仓库镜像

私有仓库本身就是一个镜像，需要从docker-hub上面拉取:

docker pull registry

1. 启动私有仓库容器

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

1. 验证私有仓库创建成功

访问http://私有仓库服务器ip:5000/v2/\_catalog，如果显示 {"repositories”:[]}表示私有仓库创建成功。

1. 修改daemon.json

我们需要让docker信任该私有仓库，这样才能进行上传，拉取镜像。在docker的配置文件daemon.json中添加该私有仓库的地址：

{“insecure-registries”:[“私有仓库服务器ip:5000”]}

1. 重启docker

systemctl restart docker

docker start registry

1. 将镜像上传到私有仓库
2. 标记镜像为私有仓库的镜像

可以为镜像作标记，标记为私有仓库中的镜像。

docker tag centos:7 私有仓库ip:5000/centos:7

做完标记后，可以发现多了一个镜像，名称为我们打的标记，其实它和centos:7是同一个镜像，注意如果要删除其中某一个镜像的话，用名称来区分删除。

1. 上传标记的镜像

docker push 私有仓库服务器ip:5000/centos:7

1. 从私有仓库拉取镜像

docker pull 私有仓库服务器ip:5000/centos:7

1. Docker容器与传统虚拟机
2. 相同点

容器和虚拟机都是用于资源隔离。

1. 不同点

容器虚拟化的是操作系统，它与宿主机共享的是操作系统；

虚拟机虚拟化的是硬件，它与宿主机共享的是硬盘，网卡等资源；

虚拟机可以运行不同的操作系统，而容器只能运行同一类型的操作系统。

容器在性能上接近于原生，因为它共享了宿主机的操作系统。