docker-课堂笔记

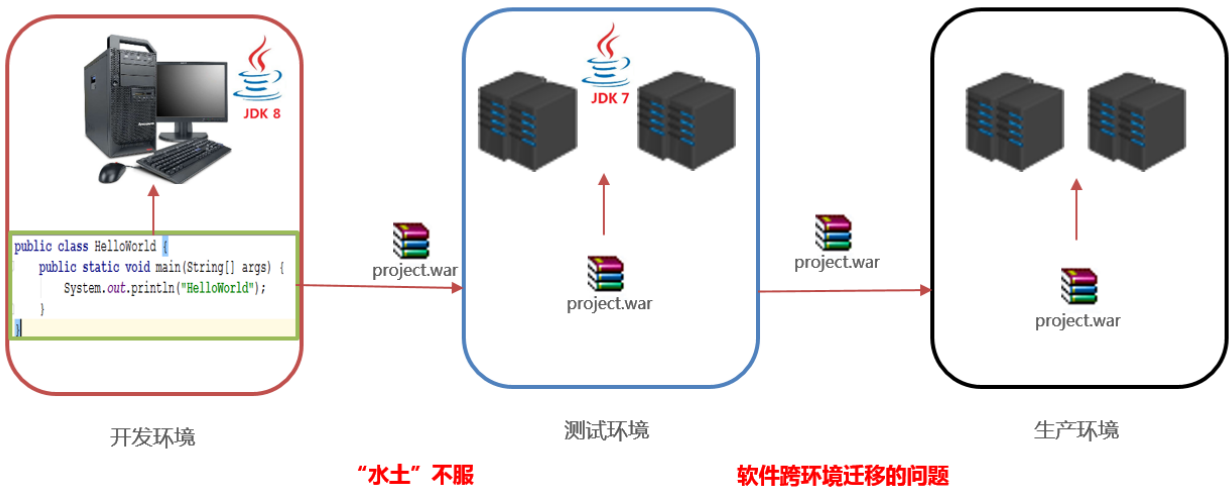
一、简介

本章节学习目标：

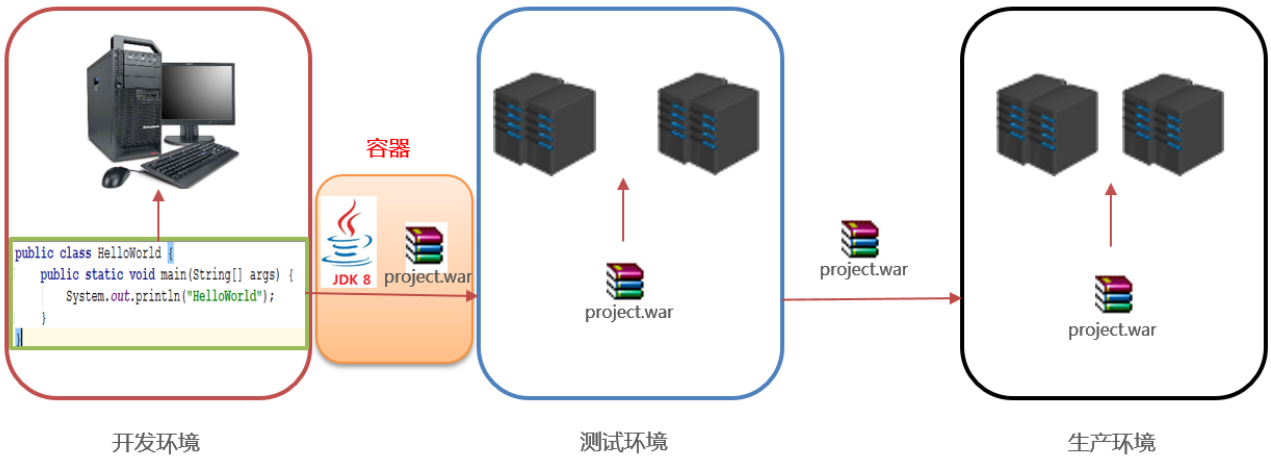
- [ ] docker解决了什么问题
- [ ] docker和虚拟机的区别
- [ ] 在CentOS7里安装docker

1. docker简介

我们写的代码会接触到好几个环境：开发环境、测试环境以及生产环境等等。多种环境去部署同一份代码，由于环境原因往往会出现**软件跨环境迁移的问题**（也就是**“水土”不服**）



针对这种问题如何解决？我们可以将工程及此工程依赖的所有软件打包到一个容器中统一部署

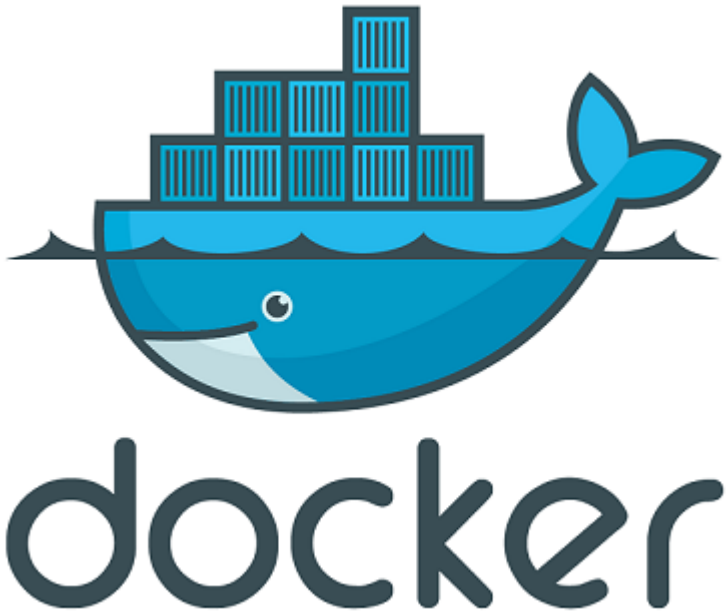


1.1 什么是docker

Docker 是一个开源的应用容器引擎，**是快速构建、运行、管理应用的工具**，它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，也可以实现虚拟化。

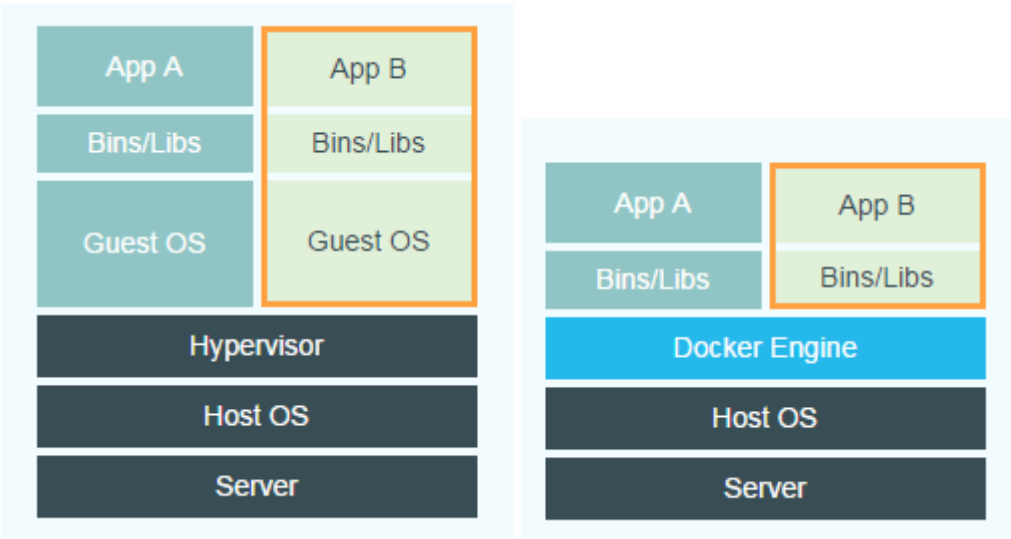
容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

Docker 从 17.03 版本之后分为 CE（Community Edition: 社区版） 和 EE（Enterprise Edition: 企业版），我们用社区版就可以了。



1.2 容器和虚拟机对比

本质的区别



容器

容器是应用层的一个抽象，它将代码和依赖关系打包在一起。

多个容器可以在同一台机器上运行，并与其他容器共享操作系统内核，每个容器作为独立的进程在用户空间运行。

容器比虚拟机占用的空间更小（容器镜像的大小一般为几十MB），可以处理更多的应用，对虚拟机和操作系统的要求也更低。

虚拟机

虚拟机(VM)是物理硬件的抽象，将一台服务器变成多台服务器。

管理程序允许多个虚拟机在一台机器上运行。每个虚拟机都包括一个操作系统、应用程序、必要的二进制文件和库的完整副本--占用几十GB的空间。

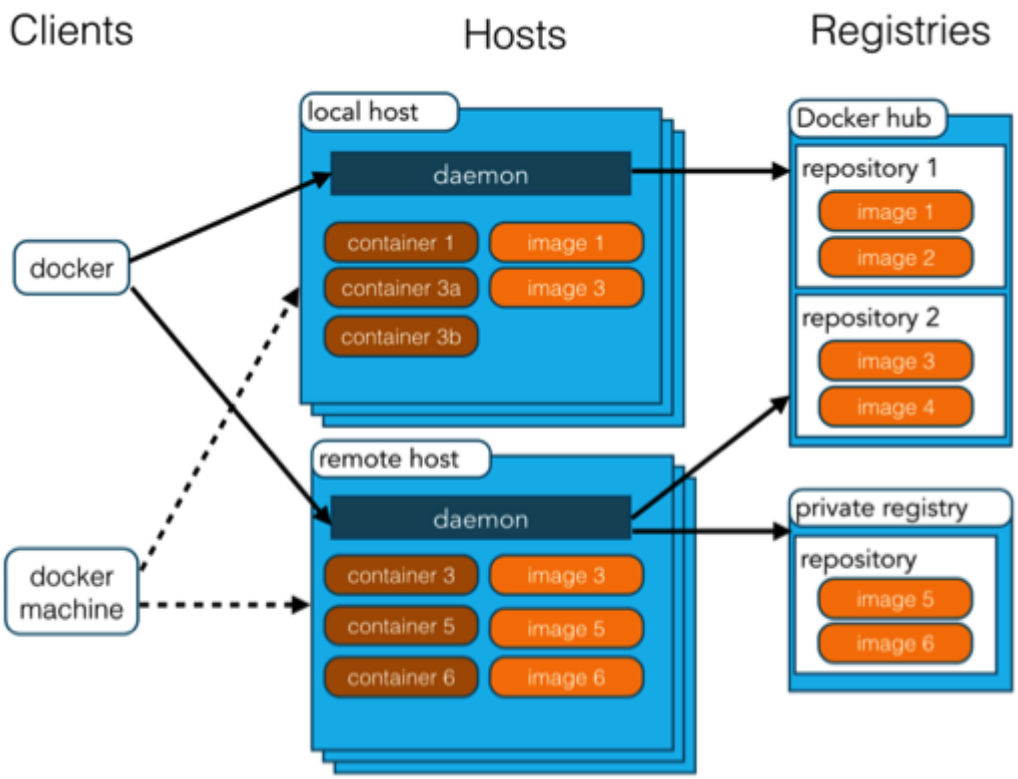
虚拟机的启动速度也会很慢。

使用的区别

特性	虚拟机	容器
隔离级别	操作系统级	进程级
隔离策略	Hypervisor	Cgroups
系统资源	5-15%	0-5%
启动时间	分钟级	秒级
镜像存储	GB-TB	KB-MB
集群规模	上百	上万

2. 安装docker

2.1 docker相关概念



镜像image

Docker 镜像（Image），就相当于是一个文件系统，是用于创建 Docker 容器的模板。也可以将镜像当作容器的“源代码”。镜像体积很小，非常“便携”，易于分享、存储和更新。

容器container

容器是独立运行的一个或一组应用，是镜像运行时的实体。（相当于是精简版的虚拟机，其中安装好了软件）

注册中心(仓库)registry

Docker 仓库用来保存镜像，有点类似于Maven的远程仓库。Registry 分为公共和私有两种。Docker 公司运营公共的 Registry 叫做 Docker Hub。用户可以在 Docker Hub 注册账号，分享并保存自己的镜像（说明：在 Docker Hub 下载镜像巨慢，可以自己构建私有的 Registry）。

2.2 安装docker

Docker 官方建议在 Ubuntu 中安装，因为 Docker 是基于 Ubuntu 发布的。

由于我们学习的环境都使用的是 CentOS，因此这里我们将 Docker 安装到 CentOS 上。

注意：这里建议安装在 CentOS7.x 以上的版本，在 CentOS6.x 的版本中，安装前需要安装其他很多的环境而且 Docker 很多补丁不支持更新。

2.2.1 安装docker

在安装docker之前，可以先做如下准备工作：

```
#1. 关闭防火墙，并禁止防火墙开机自启
systemctl stop firewalld
systemctl disable firewalld

#2. 关闭MySQL，并禁止MySQL开机自启（你的CentOS里之前安装过MySql，为防止端口冲突，需要做这一步）
systemctl stop mysqld
systemctl disable mysqld
```

启动CentOS7后，使用SecureCRT连接上CentOS7，然后依次执行命令（要联网）：

```
# 更新系统，如果需要确认，全部选 y (yes)。根据网络状况，此操作可能要花几分钟或者十几分钟时间
yum update

# 安装yum-utils工具和两个驱动依赖
yum install -y yum-utils device-mapper-persistent-data lvm2

# 设置使用阿里云的yum源，稍后会从阿里云下载docker软件
yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

# 安装docker
yum install docker-ce

# 查看docker版本
docker -v
```

2.2.2 设置注册中心(仓库)

如果不设置注册中心，将从默认的docker hub里下载镜像，速度非常慢。

USTC（中科大的镜像服务）是老牌的Linux镜像服务提供者（注册中心Registry），从Ubuntu5.04版本就在使用。USTC的docker镜像速度加载很快，它的优势之一是不需要注册，是真正的公共服务。

设置步骤：

1. 创建文件夹：`mkdir /etc/docker`
2. 编辑文件daemon.json：`vim /etc/docker/daemon.json`

   按字母`i`进入编辑模式，复制以下内容，在finalShell里右键，粘贴到文件里

   ```
   {
     "registry-mirrors":["https://docker.mirrors.ustc.edu.cn"]
   }
   ```

   保存并退出vim：按`ESC`，输`:wq`回车

   然后查看配置的是否正确：`cat /etc/docker/daemon.json`
3. 重新加载daemon文件，然后重启docker服务：

```
systemctl daemon-reload
systemctl restart docker
```

5. 查看docker信息：`docker info`，如果能看到以下信息，就说明配置成功了

```
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
 127.0.0.0/8
Registry Mirrors:
 https://docker.mirrors.ustc.edu.cn/
Live Restore Enabled: false
```

3. 小结

1. docker是什么？是一个容器引擎，可以快速构建、运行、管理容器
2. 容器和虚拟机的对比：

   容器更小，更轻量，更方便

   虚拟机更笨重，体积更大，功能更强悍

二、常用命令【重点】

本章节学习目标：

- [ ] 能够启动和关闭docker
- [ ] 能够管理docker镜像
- [ ] 能够管理docker容器

1. 服务管理

命令	说明
systemctl start docker	启动docker
systemctl status docker	查看docker运行状态
systemctl restart docker	重启docker
systemctl stop docker	关闭docker
systemctl enable docker	设置docker开机自启动

2. 镜像管理

2.1 语法说明

命令	说明
docker pull 镜像名:tag	拉取镜像包。tag通常是版本号
docker images	查看已有镜像
docker rmi 镜像id/镜像名:tag	删除镜像

- 拉取镜像时从注册中心docker hub把镜像拉取到本地。拉取下来的镜像，在`/var/lib/docker`目录下存储
- `tag`：标签名称，相当于版本号。

  如果不知道版本号，可以从https://hub.docker.com/里查询

  如果命令里不写tag，默认会拦截latest最新版本

2.2 使用练习

1. 拉取CentOS7镜像，查看镜像列表（从https://hub.docker.com/里搜索CentOS镜像，找到CentOS7）
2. 拉取nginx镜像，查看镜像列表（从https://hub.docker.com/里搜索nginx镜像，找到nginx1.14.2）

3. 容器管理

容器是独立运行的一个或一组应用，是镜像运行时的实体。（相当于是精简版的虚拟机，其中安装好了软件）

3.1 容器基本使用

3.1.1 语法说明

命令	说明

命令	说明
docker ps -a	查看所有容器（如果去掉参数`-a`，表示只查看运行中的容器）
docker run	创建并启动运行容器
docker stop 容器名	停止容器
docker start 容器名	启动容器
docker restart 容器名	重启容器
docker inspect 容器名	查看容器详细信息
docker exec -it 容器名 /bin/bash	进入容器内部（在容器内执行exit命令退回到宿主机）
docker rm 容器名	删除容器（必须先关闭容器才能删除）

- 创建容器：`docker run -di --name=容器名称 -p 宿主机端口:容器端口 -v 宿主机目录:容器目录 镜像名称:标签`

  命令介绍：根据镜像创建容器并启动运行容器

  参数介绍：

  - `d`：创建出来的容器在后台运行
  - `i`：开启STDIN交互（让容器处理活动状态）
  - `name`：用于设置容器名称
  - `p`：用于设置 宿主机端口 与 容器端口的映射
  - `v`：用于设置 宿主机数据卷(或文件夹) 与 容器文件夹的映射

3.1.2 练习:创建CentOS容器

要求：

- 创建CentOS7容器并启动
- 查看所有正在运行的容器
- 进入容器内部，查看容器里的目录结构
- 退出容器回到宿主机
- 关闭容器
- 删除容器

3.2 端口映射

3.2.1 端口映射介绍

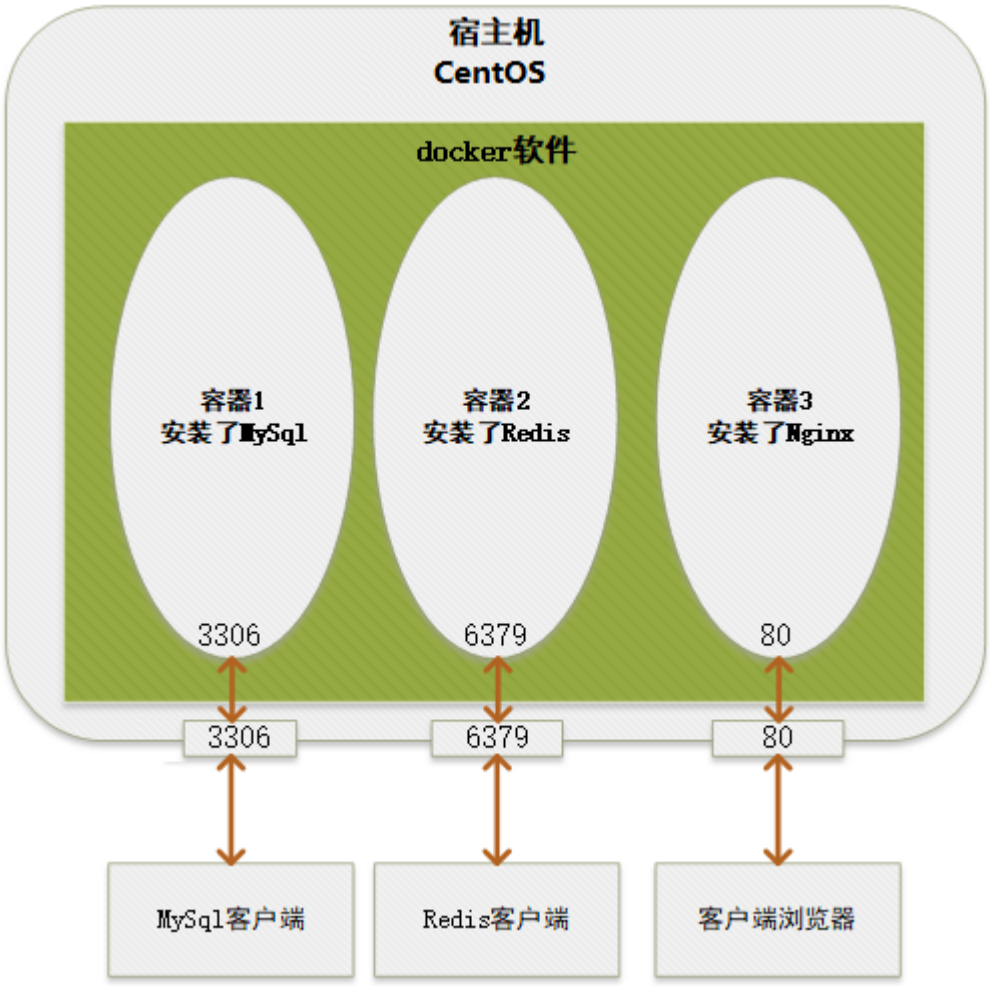
宿主机和容器的关系：

- docker所在的主机，称为宿主机；docker里还有容器

为什么要做端口映射：

- 每个容器都相当于一个独立的服务器，服务器之间是一个小型局域网，外界的客户端不能访问某一个容器

- 所以要做一个端口映射：宿主机的一个端口号---容器里的一个端口号。两个端口建议相同
- 这样：客户端就能通过“宿主机:端口”，访问到映射的容器端口



3.2.2 练习:创建Redis容器

要求:

- 创建redis容器并启动，暴露端口6379
- 查看所有正常运行的容器
- 然后用RDM工具或其它工具，访问redis

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

3.2.3 练习:创建MySQL容器

要求:

- 创建MySql容器并启动，暴露端口3306
- 查看所有正在运行的容器
- 然后使用Navicat或其它工具连接MySQL

```
# 拉取镜像
docker pull mysql:5.7

# 创建容器
docker run -id --name=mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root mysql:5.7 --character-set-server=utf8
```

3.3 挂载数据卷

之前我们创建的所有容器，并不涉及其中文件的修改。但是如果要创建一个nginx容器，或者tomcat容器，如何把项目传输到容器内部呢？docker提供了数据卷的功能，用于实现宿主机目录与容器目录的映射，这样我们在宿主机目录里的一切操作，都会作用到容器目录里了。

3.3.1 数据卷常用命令

数据卷：被docker管理的一些文件夹目录。

数据卷可以跟容器里的指定路径进行绑定，绑定后：

- 在数据卷文件夹里添加文件，docker会自动帮我们把文件同步到容器里
- 容器里的文件有所变化，docker也会同步到绑定的数据卷里

管理数据卷的命令

命令	说明
`docker volume create 数据卷名称`	创建数据卷
`docker volume ls`	列出数据卷
`docker volume inspect 数据卷名称`	查看数据卷详情
`docker volume rm 数据卷名称`	删除数据卷
`docker volume prune`	清理多余的数据卷

给容器挂载数据卷

在创建容器时，可以通过`-v`参数，将数据卷挂载给容器内的某个目录：

- `docker run -id --name=容器名称 -v 数据卷名称:容器内目录 -p 宿主机端口:容器端口 镜像名称:tag`

3.3.2 练习:创建nginx容器

要求：创建一个nginx容器，修改nginx内的html页面

已知：nginx内html目录的位置是 /usr/share/nginx/html

提示：

1. 创建一个数据卷，名称为html。查看数据卷的文件夹位置

   创建数据卷`docker volume create html`

   查看数据卷`docker volume inspect html`

```
[root@vm ~]# docker volume inspect html
[
    {
        "CreatedAt": "2023-05-15T11:57:02+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/html/_data",
        "Name": "html",
        "Options": null,
        "Scope": "local"
    }
]
```

2. 创建nginx容器，使用 -v 参数把数据卷html挂载绑定到容器内的/usr/share/nginx/html

   `docker run -id --name=mynginx -v html:/usr/share/nginx/html -p 80:80 nginx:1.14.2`

3. 使用浏览器访问nginx，先查看原始index.html页面的效果

   浏览器输入地址：`http://宿主机ip:80`

4. 直接进入数据卷html的目录，修改其中的index.html文件

   `cd /var/lib/docker/volumes/html/_data`

   使用vi修改其中的index.html文件，之后保存并退出vi

5. 再使用浏览器访问nginx，查看修改后的index.html页面效果

   浏览器输入地址：`http://宿主机ip:80`

3.4 挂载目录

3.4.1 语法说明

容器不仅仅可以挂载数据卷，也可以直接挂载到宿主机目录上。

挂载目录的语法，和挂载数据卷的语法是类似的，都是使用参数 -v ：

- 挂载数据卷：`-v 数据卷:容器内目录`
- 挂载目录：`-v 宿主机目录:容器内目录`

  注意：“宿主机目录”要以 / 或者 ./ 开头的路径

3.4.2 练习:创建tomcat容器

要求：创建一个tomcat容器，修改mysql内的html页面

已知：tomcat内webapps目录的位置是 `/usr/local/tomcat/webapps`

提示：

1. 创建文件夹 `/root/webapps` ： `mkdir /root/webapps`
2. 创建nginx容器，使用 -v 参数把 `/root/webapps` 挂载绑定到容器内的 `/usr/local/tomcat/webapps`

```
docker run -id --name=mytomcat -p 8080:8080 -v /root/webapps:/usr/local/tomcat/webapps tomcat:8.5.88
```

3. 把资料中的`appfront`文件夹上传到`/root/webapps`文件夹里
4. 再使用浏览器访问tomcat，访问 `http://宿主机ip:8080/appfront/index.html`

4. 小结

管理镜像：

- 拉取镜像：docker pull 镜像名:tag
- 查看镜像：docker images
- 删除镜像：docker rmi 镜像名:tag

管理容器：

- 基本命令：

  创建并启动容器：docker run -id --name=容器名 镜像名:tag

  查看容器列表：docker ps -a

  关闭容器：docker stop 容器名

  启动容器：docker start 容器名

  重启容器：docker restart 容器名

  删除容器：docker rm 容器名

  查看容器详情：docker inspect 容器名

  查看容器日志：docker logs 容器名

  进入容器：docker exec -it 容器名 /bin/bash。 容器里退回到宿主机：exit
- 创建容器时进行端口映射：

  docker run -id --name=容器名 -p 宿主机端口:容器端口 镜像名:tag

管理数据卷：

- 基本命令：

  查看数据卷列表：docker volume ls

  查看数据卷详情：docker volume inspect 数据卷名称

  创建数据卷：docker volume create 数据卷名称

  删除数据卷：docker volume rm 数据卷名称

  清理未使用的数据卷：docker volume prune
- 创建容器时挂载数据卷：

  docker run -id --name=容器名 -v 数据卷名称:容器里的目录 镜像名:tag

三、镜像制作【了解】

1. 备份与恢复

1.1 导出镜像

- 命令：`docker save -o 文件名称 镜像名称`

- 参数：
  - `o`：output，表示要输出到文件
- 说明：
  - 文件名称，通常是`xxx.tar`打包文件
- 示例：`docker save -o nginx.tar nginx:1.14.2`

1.2 加载镜像

- 命令：`docker load -i xxx.tar`
- 参数：
  - `i`：input，表示要从文件读取
- 示例：
  - 我们先将原有镜像删除掉：`docker rmi nginx:1.14.2`
  - 从`xxx.tar`恢复镜像：`docker load -i nginx.tar`

1.3 练习

要求：

1. 把redis镜像保存成文件 redis.tar

   `docker save -o redis.tar redis:5.0`

2. 删除所有redis的容器和镜像

   `docker stop myredis`

   `docker rm myredis`

   `docker rmi redis:5.0`

3. 重新加载redis.tar得到镜像

   `docker load -i redis.tar`

2. Dockerfile

2.1 Dockerfile介绍

前边是从一个已有容器生成一个镜像。而Dockerfile是另外一种构建镜像的方式，它是以一种基础镜像为基础，编写一系列的docker指令，每条指令构建一层镜像，通过这些指令一层层构建出一个目标镜像出来。

- 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

Dockerfile的常用指令有以下几个：

> 更新详细语法说明，请参考官网文档： https://docs.docker.com/engine/reference/builder

关键字	作用	备注

关键字	作用	备注
FROM	指定父镜像	指定dockerfile基于那个image构建
MAINTAINER	作者信息	用来标明这个dockerfile谁写的
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"]
ADD	添加文件	build的时候添加文件到image中 不仅仅局限于当前build上下文 可以来源于远程服务
ENV	环境变量	指定build时候的环境变量 可以在启动的容器的时候 通过-e覆盖 格式ENV name=value

从前面的内容可以看出，要构建一个容器，需要做很多的工作，设置很多的配置，如果我们可以把每一层修改、安装、构建、操作的命令都写入一个脚本，用这个脚本来构建、定制镜像，那么之前提及的无法重复的问题、镜像构建透明性的问题、体积的问题就都会解决。 这个脚本就是 Dockerfile。

2.2 Dockerfile示例

需求

定义dockerfile，发布springboot项目

分析

1. SpringBoot项目可以独立运行，不需要Tomcat；但是需要有jdk

   所以：我们这个容器，要提供基础的jdk环境，基于jdk环境构造一个新容器

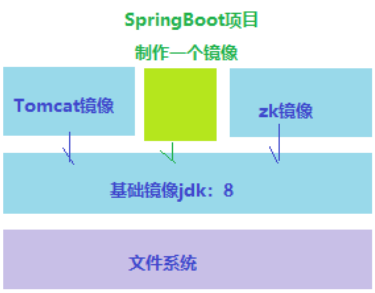
   使用：`FROM java:8`

2. 要把SpringBoot项目添加到这个容器里

   使用：`ADD demo.jar /demo.jar`

3. 当启动容器时，要同时启动运行这个SpringBoot项目

   所以：启动容器时，要执行Java命令，运行SpringBoot项目



如果需要制作一个全新镜像：可以使用Dockerfile技术
允许我们基于一个基础镜像，制作一个新镜像出来

创建dockerfile文件，内容

```
FROM jdk:8                          要创建的这个镜像，依赖的基础镜像
MAINTAINER liuyp@itheima.cn         声明一下作者
ADD demo.jar /demo.jar              要把宿主机里的demo.jar，添加到新镜像的根目录里，/demo.jar
CMD java -jar /demo.jar             在启动容器时，要执行的命令：把SpringBoot项目（demo.jar)运行起来
```

切换到dockfile文件所在的目录，然后执行命令：docker build -f dockerfile文件 -t demo ./

实现

1. 把SpringBoot项目上传到宿主机

在宿主机里创建一个文件夹`~/dockerfiles`，把SpringBoot工程打包为`demo.jar`，然后上传到CentOS的`~/dockerfiles`里

2. 创建Dockerfile

在`~/dockerfiles`文件夹里创建文件`Dockerfile`，内容如下：

```
#1.定义父镜像：
FROM openjdk:8
#2.定义作者信息：
MAINTAINER  itheima <itheima@itcast.cn>
#3.将jar包添加到容器：
ADD demo.jar /demo.jar
#4.定义容器启动执行的命令：  当通过此镜像启动容器的时候，执行的命令
CMD java -jar /demo.jar
```

3. 通过Dockerfile构建镜像

```
# 切换到dockerfile文件所在的路径
cd ~/dockerfiles

# 构建镜像
docker build -f ./Dockerfile -t demo:1 ./
```

4. 启动容器

```
#创建启动容器
docker run -id --name=demo -p 81:80 demo:1

#打开浏览器，输入地址  http://宿主机ip:81/hello  ，可以访问到SpringBoot项目
```

3. 小结

把镜像保存成文件：docker save -o xxx.tar 镜像名:tag

加载文件恢复镜像：docker load -i xxx.tar

四、服务编排【了解】

K8s：kubernetes

1. 服务编排介绍

服务编排或容器编排：按照一定的业务规则批量管理容器

Docker compose是一个用于定义和运行多个Docker容器的编排工具。可以一条命令启动多个容器。主要是解决了容器与容器之间如何管理编排的问题。

使用Docker compose 有三个步骤：

1. 利用Dockerfile定义运行环境（如果已有镜像，可省略这一步）
2. 使用 `docker-compose.yml` 定义组成应用的各服务
3. 运行 `docker-compose up -d` 启动应用

2. 安装docker compose

```
# 下载docker compose
curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

# 设置权限
chmod +x /usr/local/bin/docker-compose

# 查看版本
docker-compose -version
```

- 如果要卸载docker-compose，可执行如下命令：

```
# docker compose是二进制包方式安装的，删除二进制文件即可
rm /usr/local/bin/docker-compose
```

3. 服务编排示例

要求：通过docker-compose 批量创建三个容器（nginx，tomcat，redis）

1. 创建docker-compose目录

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

2. 编写 `docker-compose.yaml` 文件。必须叫这个名字，不能换

```
version: "3.0"
services:
  redis:
    container_name: redis
    image: redis:5.0
    ports:
      - 6379:6379
  nginx:
    container_name: nginx
    image: nginx:1.14.2
    ports:
      - 80:80
    volumes:
      - /root/volumes/nginx/html:/usr/share/nginx/html
  tomcat:
    container_name: tomcat
    image: tomcat:8.5.88
```

```
    ports:
      - 8080:8080
    volumes:
      - /root/volumes/tomcat/webapps:/usr/local/tomcat/webapps
```

3. 启动。

```
#！！！注意：必须先切换到`docker-compose.yaml`文件所在的目录后，才可以执行以下命令！！！
cd ~/docker-compose

# docker-compose up -d 以守护进程方式创建并启动容器
docker-compose up -d
```

4. 常用命令 【掌握】

注意：必须先切换到`docker-compose.yaml`文件所在的目录后，才可以执行以下命令！！！

命令	说明
`docker-compose up -d`	创建容器并后台运行
`docker-compose start`	启动容器
`docker-compose stop`	停止容器
`docker-compose restart`	重启容器
`docker-compose rm`	删除已经停止的容器
`docker-compose down`	停止并删除容器

5. 小结

五、docker私服【拓展】

我们知道docker镜像可以托管到Docker Hub中，跟代码库托管到github是一个道理。但如果我们不想把docker镜像公开放到Docker Hub中，只想在部门或团队内部共享docker镜像，能不能项gitlab一样在搭建私有的仓库呢？答案是肯定的，docker也支持将镜像存到私有仓库。

1. 搭建私服

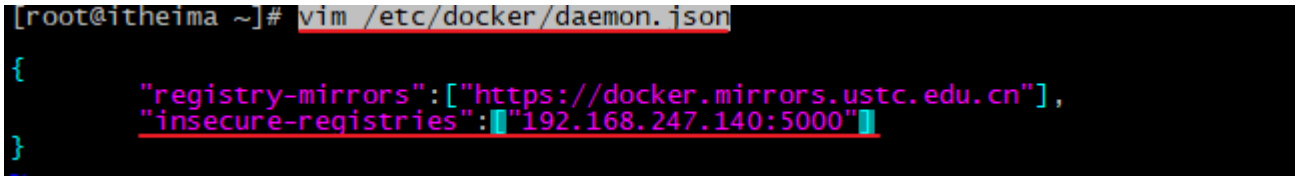
1.1 拉取私服镜像

```
# 拉取私服镜像
docker pull registry
# 启动运行私服
docker run -id --name=registry -p 5000:5000 registry
```

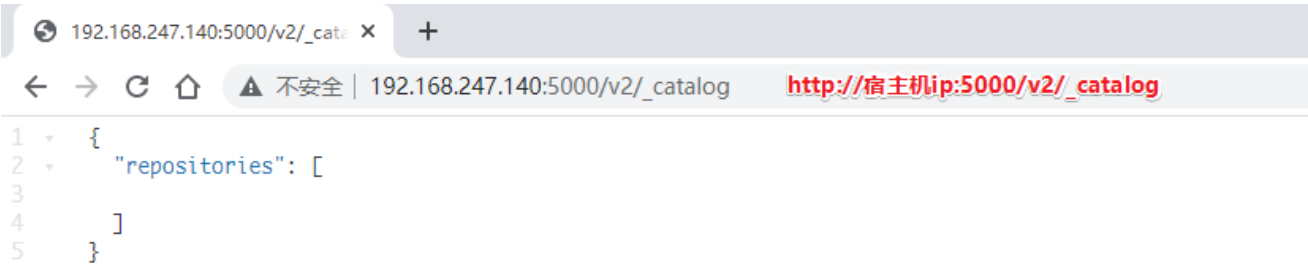
1.2 设置私服可信任

- 设置私有仓库为可信任，用vim或vi编辑文件 `/etc/docker/daemon.json`

```
"insecure-registries":["私有仓库的ip:端口"]
```



- 重启docker：`systemctl restart docker`
- 打开浏览器，通过网址：http://192.168.247.140:5000/v2/_catalog来验证是否启动成功



2. 上传镜像到私服

1. 给镜像打标记成为私有仓库镜像

- 语法：`docker tag 镜像名称 私有仓库ip:端口/镜像名称`
- 示例：`docker tag redis:5.0 192.168.200.137:5000/redis:5.0`

2. 上传到私有仓库

- 语法：`docker push 私有仓库ip地址:5000/镜像名称`
- 示例：`docker push 192.168.200.137:5000/redis:5.0`

- 上传后，可以浏览器上看到

3. 从私服拉取镜像

- 语法：`docker pull 私有仓库ip:端口/镜像名称`
- 示例：`docker pull 192.168.200.137:5000/redis:5.0`