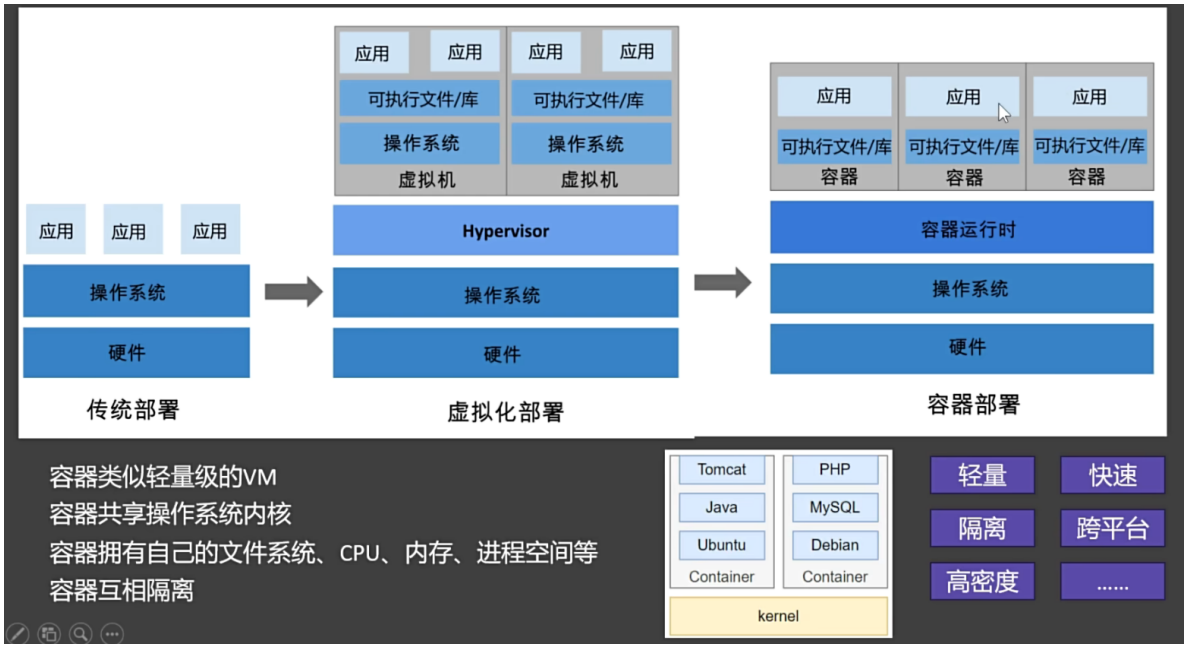


在操作系统上安装容器运行时环境，利用容器的技术启动英应用，每个应用运行在自己的容器内部，每个容器包含了本应用运行的完整环境。

容器和容器之间互相隔离（应用和应用之间相互隔离），一个容器出现问题不会影响到其他容器。

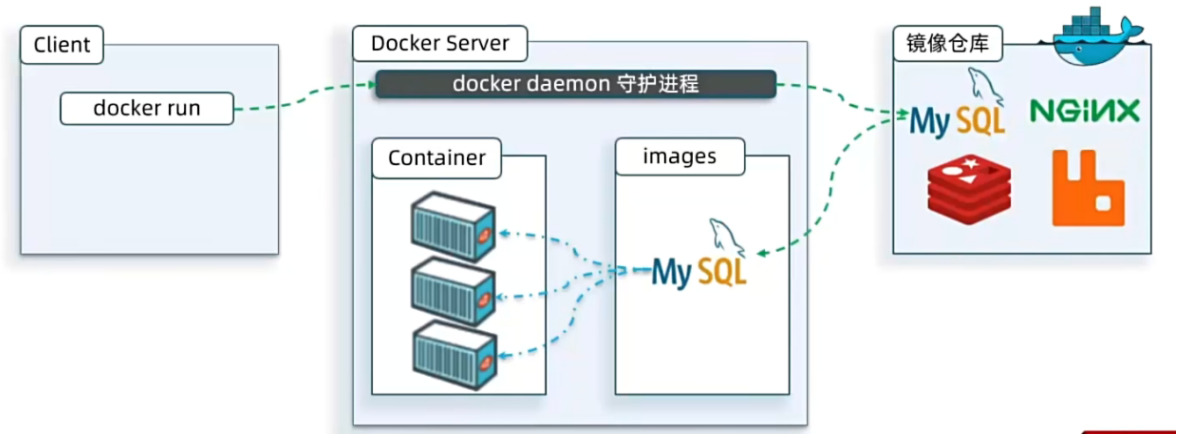
而容器共享操作系统内核，但是拥有自己的文件系统、CPU、内存、进程空间。即类似于轻量级的VM。所以容器占用的资源比较少。

容器也可以实现跨平台部署（只要安装容器运行时环境）。

镜像和容器

当我们利用Docker安装应用时，Docker会自动搜索并下载应用镜像（image）。镜像不仅包含应用本身，还包含应用运行所需要的环境、配置、系统函数库。Docker会在运行镜像时创建一个隔离环境，称为容器（container）

镜像仓库：存储和管理镜像的平台，Docker官方维护了一个公共仓库：Docker Hub



宿主机：容器所在的机器。这里的容器部署在虚拟机上，那么这台虚拟机就是宿主机。

命令解读

以

```
docker run -d \
    --name mysql \
    -p 3306:3306 \
    -e TZ=Asia/Shanghai \
    -e MYSQL_ROOT_PASSWORD=abc123 \
    mysql
```

命令为例

- docker run：创建并运行一个容器，-d是让容器在后台运行
- --name mysql：给容器起个名字，必须唯一
- -p 3306：3306：将容器的端口映射到主机的端口

  `docker run -p 8080:80 nginx`：这里的意思是将宿主机的 8080 端口映射到容器内的 80 端口。这意味着当你访问宿主机的 8080 端口时，请求会被转发到容器内的 Nginx 服务器所监听的 80 端口上。
- -e KEY=VALUE：设置环境变量
- mysql：指定运行的镜像的名字。完整格式为：[镜像名]：[tag]，其中tag是镜像的版本，不写版本就默认是最新版本。

常见命令

`systemctl start docker`：启动docker

`docker pull`：从镜像仓库拉取镜像，再run时就不用从镜像仓库再拉取了。

`docker images`：显示所有镜像

`docker rmi`：删除镜像

`docker build`：自己构建镜像

`docker save`：将镜像保存到本地

`docker load`：把保存到本地的镜像加载到镜像中

`docker push`：将镜像推送到仓库中

`docker run`：创建并运行镜像

`docker stop`：停止容器（只是停止容器中的进程，并不销毁容器）

`docker start`：启动容器

`docker ps`：查看容器运行状态

`docker ps -a`：查看包括未启动容器在内的容器状态

`docker inspect 容器名`：查看具体容器的状态

`docker rm`：删除容器

`docker logs`：查看容器日志

`docker exec`：进入容器

命令别名

打开文件：

```
vim ~/.bashrc
```

修改文件内容，给命令起别名：

```
//以指定格式输出容器状态
alias dps='docker ps --format "table
{{.ID}}\t{{.Image}}\t{{.Ports}}\t{{.Status}}\t{{.Names}}"'
```

启用配置：

```
source  ~/.bashrc
```

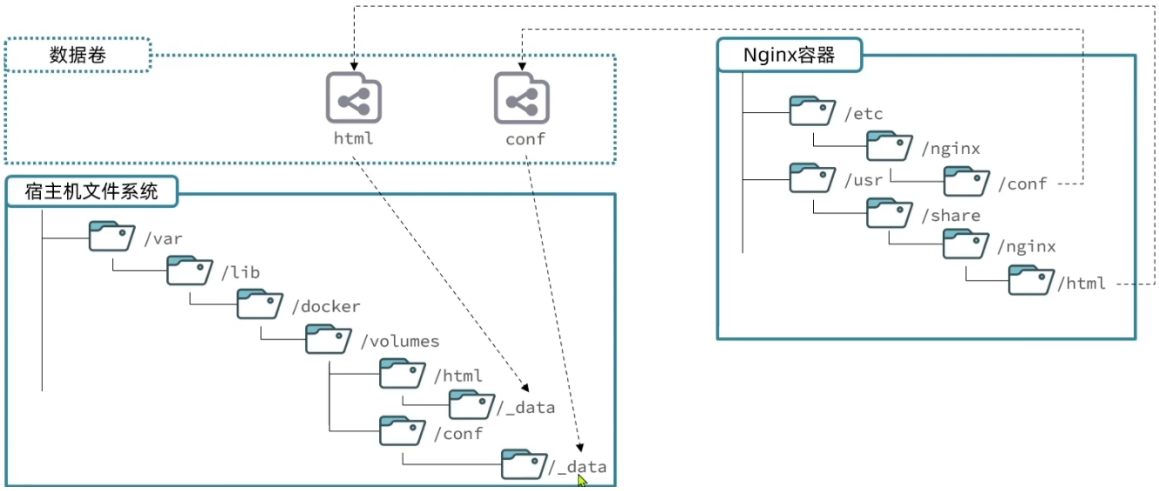
数据卷挂载

由于镜像是最小化的系统环境，只包含需要的系统函数库和依赖。

比如我们在nginx容器中，想要修改静态资源时，会发现没有vi这个命令。这就造成了在容器中想进行一些操作变得十分困难。

这时就需要用到数据卷（volume），数据卷是一个虚拟目录，是容器内目录与宿主机目录之间映射关系的桥梁。

比如我们创建一个html数据卷，那么就会自动在宿主机上创建一个对应的目录:`/var/lib/docker`下的`html/_data`目录。而容器的目录和数据卷又做了挂载，所以能实现宿主机目录与容器目录间的双向映射。



可通过`docker volumn --help`查看数据卷相关指令

`docker volume create`

`docker volume ls`

`docker volume rm`

`docker volume inspect`：查看某个数据卷的详情

`docker volume prune`：清除数据卷

举例：

需求：

- 创建Nginx容器，修改nginx容器内的html目录下的index.html文件内容
- 将静态资源部署到nginx的html目录

- 挂载的指令：在执行`docker run`时，使用`-v 数据卷：容器内目录`可以完成数据卷挂载；
- 当创建容器时，如果挂载了数据卷但数据卷不存在，会自动创建数据卷

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

然后对`/var/lib/docker/volumes/html/_data`下的index.html文件进行修改即可

本地目录挂载

启动mysql容器时，会自动创建一个数据卷将mysql的数据存储目录挂载到宿主机，这个数据卷的名字是一长串字符，我们把它称为匿名卷。

为什么MySQL要自动创建一个数据卷将mysql的数据存储目录挂载到宿主机?

mysql在运行过程中会不断产生数据，如果不挂载，这些数据都会保存在容器内的文件系统里，将来容器的体积会越来越大，相对容器作迁移会非常不方便。所以要进行挂载。

但是mysql升级时，我们需要重新创建容器，把旧的容器删掉，此时挂载的目录还在，如果要进行数据迁移将会非常麻烦，所以要将宿主机目录直接挂载到容器内的目录下。

```
使用 -v 参数挂载 MySQL 的数据存储目录后，不会自动创建匿名卷。
```

在执行`docker run`时，使用`-v 本地目录:容器内目录`可以完成本地目录挂载.

目录必须以`/`开头，即采用绝对路径，否则会被识别成数据卷而非本地目录。

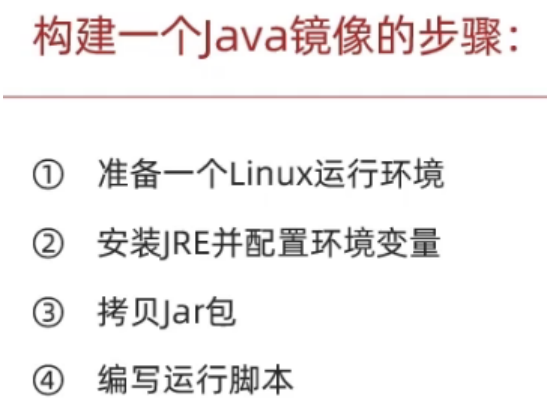
补充:mysql需要挂载的几个目录：

- 数据存储目录：`/var/lib/mysql`
- 配置文件目录：`/etc/mysql/conf.d`
- 日志文件目录（不一定有此目录）：`/var/log/mysql`
- 临时文件目录：虽然不是必须挂载，但挂载`/tmp`目录可以为 MySQL 提供更多的临时空间，特别是在处理大量数据时。
- 初始化脚本目录（如果需要）：`/docker-entrypoint-initdb.d`

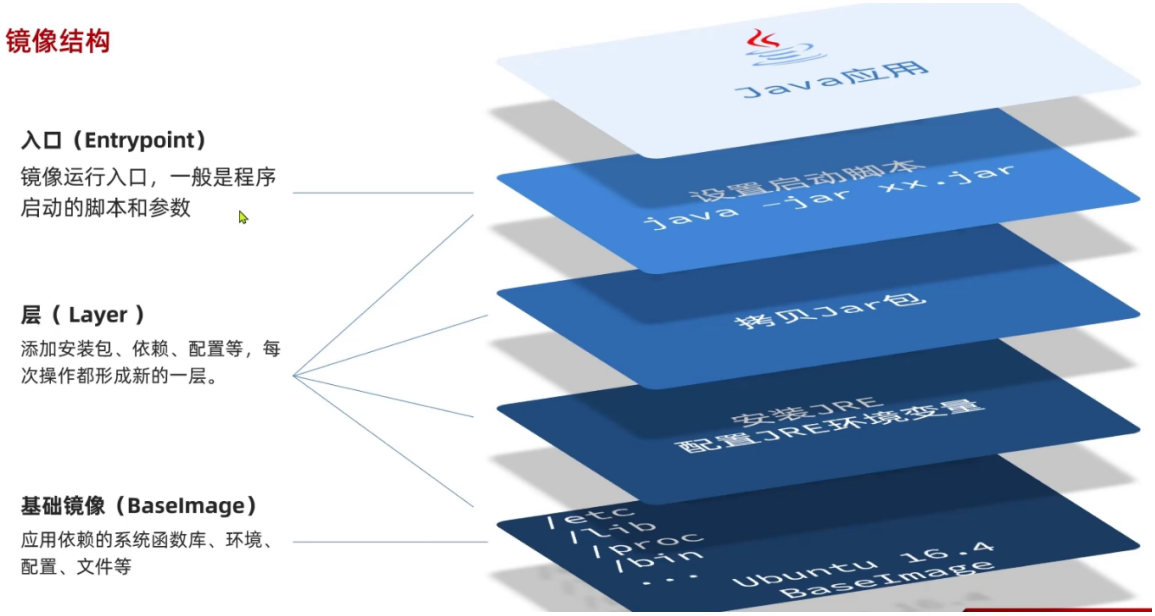
Dockerfile语法

镜像就是包含了应用程序、程序运行的系统函数库、运行配置等文件的文件包。构建镜像的过程其实就是把上述文件打包的过程。而上述文件并不是打包到一起，而是分层打包的。

构建java镜像的步骤：



镜像结构：



Dockerfile就是一个文本文件，其中包含一个个指令（instruction），用指令来说明执行什么操作来构建镜像。将来docker就可以根据Dockerfile帮我们构建镜像。常见指令：

指令	说明	示例
FROM	指定基础镜像	FROM centos:6
ENV	设置环境变量，可在后面指令使用	ENV key value
COPY	拷贝本地文件到镜像的指定目录	COPY ./jre11.tar.gz /tmp
RUN	执行Linux的shell命令，一般是安装过程的命令	RUN tar -zxvf /tmp/jre11.tar.gz && EXPORTS path=/tmp/jre11:$path
EXPOSE	指定容器运行时监听的端口，是给镜像使用者看的	EXPOSE 8080
ENTRYPOINT	镜像中应用的启动命令，容器运行时调用	ENTRYPOINT java -jar xx.jar

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

但其实我们并不需要用到所有指令，来观察如下过程：

- 首先，我们可以基于Ubuntu基础镜像，利用Dockerfile描述镜像结构：

```
# 指定基础镜像
FROM ubuntu:16.04
# 配置环境变量，JDK的安装目录、容器内时区
ENV JAVA_DIR=/usr/local
# 拷贝jdk和java项目的包
COPY ./jdk8.tar.gz $JAVA_DIR/
COPY ./docker-demo.jar /tmp/app.jar
# 安装JDK
RUN cd $JAVA_DIR \ && tar -xf ./jdk8.tar.gz \
&& mv ./jdk1.8.0_144 ./java8
# 配置环境变量
ENV JAVA_HOME=$JAVA_DIR/java8
ENV PATH=$PATH:$JAVA_HOME/bin
# 入口，java项目的启动命令
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- 可以看到，目前的步骤还是比较繁琐。于是我们来思考：不同的java应用都要用到JDK，变化的仅仅是jar包名不同，所以，把上图不变的比如JDK的配置也抽取出来，制作成一个基础镜像，就可以简化成如下的步骤：

自定义镜像

Dockerfile写完后，要怎么让docker帮我们去构建呢？这肯定需要用到一些命令

`docker build -t 镜像名：版本 Dockerfile目录`

- -t表示tag，用于给镜像起名，版本不指定默认latest
- `.`：指定dockerfile所在目录，如果就在当前目录，则指定为"."

容器网络互连

当安装docker时，docker会自动在虚拟机中创建一个虚拟的网卡（默认名为docker0），并且给这个网卡创建一个网桥。

当创建容器，会跟网桥以桥接（bridge）的方式建立连接，分配IP地址。这些容器由于处在一个网段，所以相互之间是可以ping通的

但这种方式不太好，比如容器重启，由于自动分配，IP可能发生变化，之前配置的IP可能不生效，

所以我们要用到自定义网络：

加入自定义网络的容器相互之间可以联通，并且可以通过容器名互相访问(这就解决了IP地址变化的问题)，Docker网络操作命令如下：

命令	说明	文档地址
docker network create	创建一个网络	docker network create
docker network ls	查看所有网络	docker network ls
docker network rm	删除指定网络	docker network rm
docker network prune	清除未使用的网络	docker network prune
docker network connect	使指定容器连接加入某网络	docker network connect
docker network disconnect	使指定容器连接离开某网络	docker network disconnect
docker network inspect	查看网络详细信息	docker network inspect

此外还可以通过docker run时添加：`--network 网络名`

DockerCompose

DockerCompose通过一个单独的docker-compose.yml模板文件（YAML格式）定义一组相关联的应用容器，帮助我们实现多个相互关联的Docker容器的快速部署。

一个项目对应一个文件，项目中的container称为服务（后端、前端、数据库...）

常用语法：（基本与docker run时的参数对应）

```yaml
version: "3.8"

services:
  mysql:
    image: mysql
    container_name: mysql
    ports:
      - "3306:3306"
    environment:
      TZ: Asia/Shanghai
      MYSQL_ROOT_PASSWORD: 123
    volumes:
      - "./mysql/conf:/etc/mysql/conf.d"
      - "./mysql/data:/var/lib/mysql"
      - "./mysql/init:/docker-entrypoint-initdb.d"
    networks:
      - hm-net
  hmall:
    build:
       context: .
       dockerfile: Dockerfile
    container_name: hmall
    ports:
      - "8080:8080"
    networks:
      - hm-net
    depends_on:
      - mysql
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - "18080:18080"
      - "18081:18081"
    volumes:
      - "./nginx/nginx.conf:/etc/nginx/nginx.conf"
      - "./nginx/html:/usr/share/nginx/html"
    depends_on:
      - hmall
    networks:
      - hm-net
networks:
  hm-net:
    name: hmall
```

docker compose命令格式如下：

docker compose的命令格式如下：

```
docker compose [OPTIONS] [COMMAND]
```

类型	参数或指令	说明
Options	-f	指定compose文件的路径和名称
	-p	指定project名称
Commands	up	创建并启动所有service容器
	down	停止并移除所有容器、网络
	ps	列出所有启动的容器
	logs	查看指定容器的日志
	stop	停止容器
	start	启动容器
	restart	重启容器
	top	查看运行的进程
	exec	在指定的运行中容器中执行命令

一般将所需包和Dockerfile、docker-compose.tml放在一个目录下，在这个目录下运行`docker compose up -d`。-d为后台运行

把我的register-login-API部署到linux上示例

- 在/opt 下创建了my-app目录
- 将打包好的jar从win传输进此目录
- my-app目录下创建Dockerfile,内容如下：

```
FROM openjdk:21-jdk-slim
WORKDIR /app
COPY register-login-API-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- 创建docker-compose.yml:

```
version: '3.8'
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
```

```
      - SPRING_DATASOURCE_URL=jdbc:mysql://db:3306/mydb?
useSSL=false&serverTimezone=UTC
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=abc123
      - SPRING_JPA_HIBERNATE_DDL_AUTO=update
    depends_on:
      - db

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: abc123
      MYSQL_DATABASE: mydb
    ports:
      - "3306:3306"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

volumes:
  mysql-data:
```

- 创建init.sql，初始化mysql中的数据库和表

```
CREATE DATABASE IF NOT EXISTS mydb;
USE mydb;

CREATE TABLE user (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL
);
```

- 注意：要将springboot项目中的db配置成：

```
spring.datasource.password=abc123
spring.datasource.username=root
spring.datasource.url=jdbc:mysql://db:3306/mydb?
useSSL=false&serverTimezone=UTC
```

  后面参数不用管，前面的db代表容器名，mydb为database

- 设置防火墙，使8080和3306端口能被外界访问
- `docker compose up -d` 创建并启动springboot和mysql容器
- postman进行测试