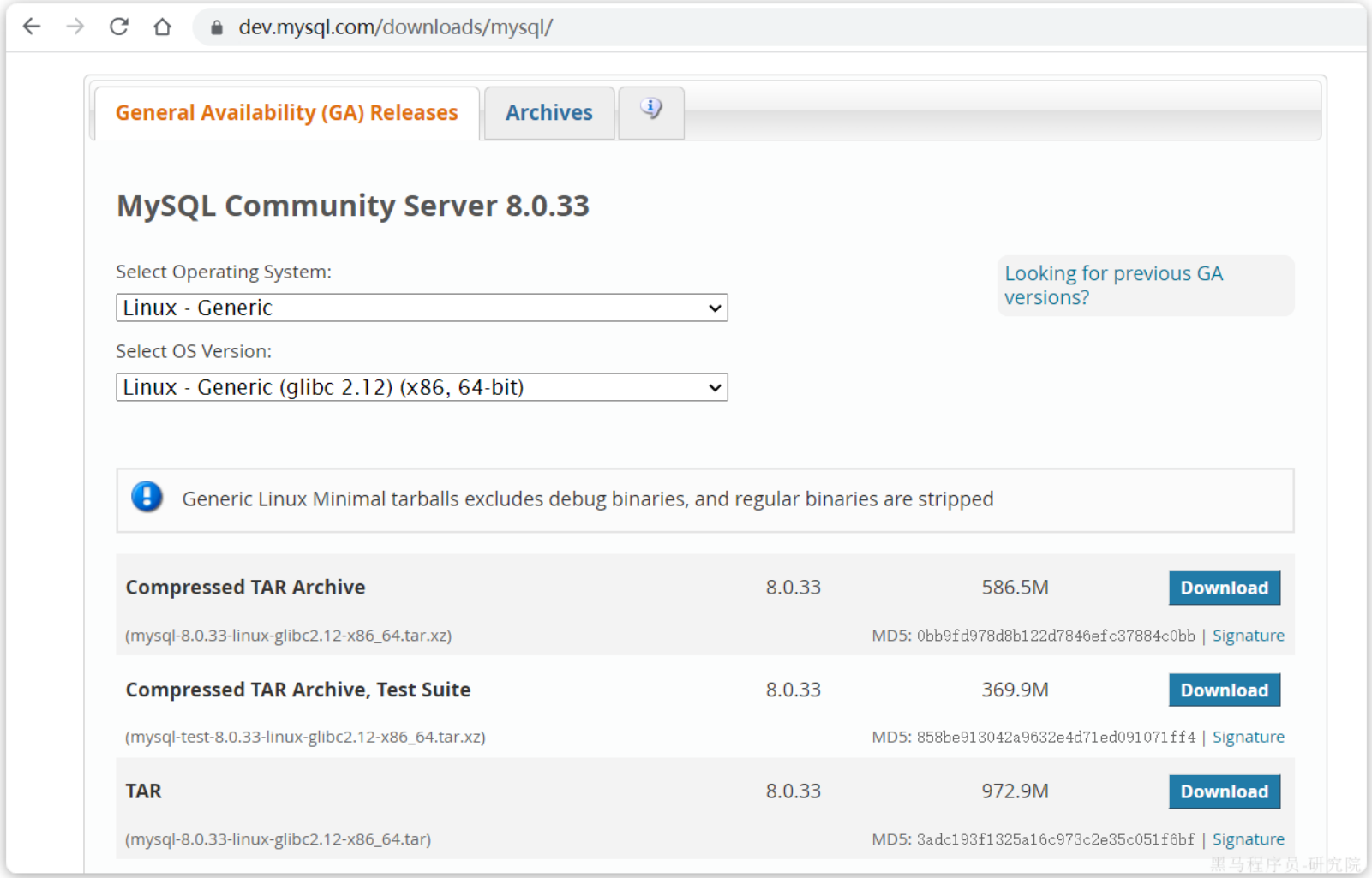
# Docker

快速构建、运行、管理应用的工具

```
[root@heima ~]# # 查看已安装的MySQL
[root@heima ~]# rpm -qa | grep mysql
[root@heima ~]# rpm -qa | grep mariadb
mariadb-libs-5.5.68-1.el7.x86_64

[root@heima ~]# # 卸载MySQL
[root@heima ~]# rpm -e --nodeps mariadb-libs-5.5.68-1.el7.x86_64
```


dev.mysql.com/downloads/mysql/
General Availability (GA) Releases
Archives
MySQL Community Server 8.0.33
Select Operating System:
Linux - Generic
Select OS Version:
Linux - Generic (glibc 2.12) (x86, 64-bit)
Looking for previous GA versions?
Generic Linux Minimal tarballs excludes debug binaries, and regular binaries are stripped
Compressed TAR Archive 8.0.33 586.5M Download
(mysql-8.0.33-linux-glibc2.12-x86_64.tar.xz) MD5: 0bb9fd978d8b122d7846efc37884c0bb | Signature
Compressed TAR Archive, Test Suite 8.0.33 369.9M Download
(mysql-test-8.0.33-linux-glibc2.12-x86_64.tar.xz) MD5: 858be913042a9632e4d71ed091071ff4 | Signature
TAR 8.0.33 972.9M Download
(mysql-8.0.33-linux-glibc2.12-x86_64.tar) MD5: 3adc193f1325a16c973c2e35c051f6bf | Signature

```
[root@heima ~]# # 上传MySQL到 /usr/local/mysql
[root@heima ~]# mkdir /usr/local/mysql

[root@heima ~]# # 解压缩
[root@heima ~]# tar -zxvf mysql-5.7.25-1.el7.x86_64.rpm-bundle.tar.gz -C /usr/local/mysql
```


命令太多！
记不住！
安装步骤太复杂！
容易出错！
安装包太多！
不知道去哪里下！
[root@heima ~]# rpm -ivh mysql-community-common-5.7.25-1.el7.x86_64.rpm
[root@heima ~]# rpm -ivh mysql-community-libs-5.7.25-1.el7.x86_64.rpm
[root@heima ~]# rpm -ivh mysql-community-devel-5.7.25-1.el7.x86_64.rpm
[root@heima ~]# rpm -ivh mysql-community-client-5.7.25-1.el7.x86_64.rpm
[root@heima ~]# rpm -ivh mysql-community-server-5.7.25-1.el7.x86_64.rpm
[root@heima ~]# yum install net-tool
[root@heima ~]# yum install openssl
[root@heima ~]# rpm -ivh mysql-community-server-5.7.25-1.el7.x86_64.rpm

```
[root@heima ~]# docker run -d --name ms -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123 mysql
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
72a69066d2fe: Pull complete
93619dbc5b36: Pull complete
99da31dd6142: Pull complete
626033c43d70: Pull complete
Status: Downloaded newer image for mysql:latest
a6cec8ff4765ca0876d0453f3ccab205fca29b5dce74f8cfbad8d76571bf79be
[root@heima ~]#
```

# 目录

Contents

# 目录

Contents

# 目录

Contents

# 目录

Contents


# 快速入门

# 目录

Contents

## 部署MySQL

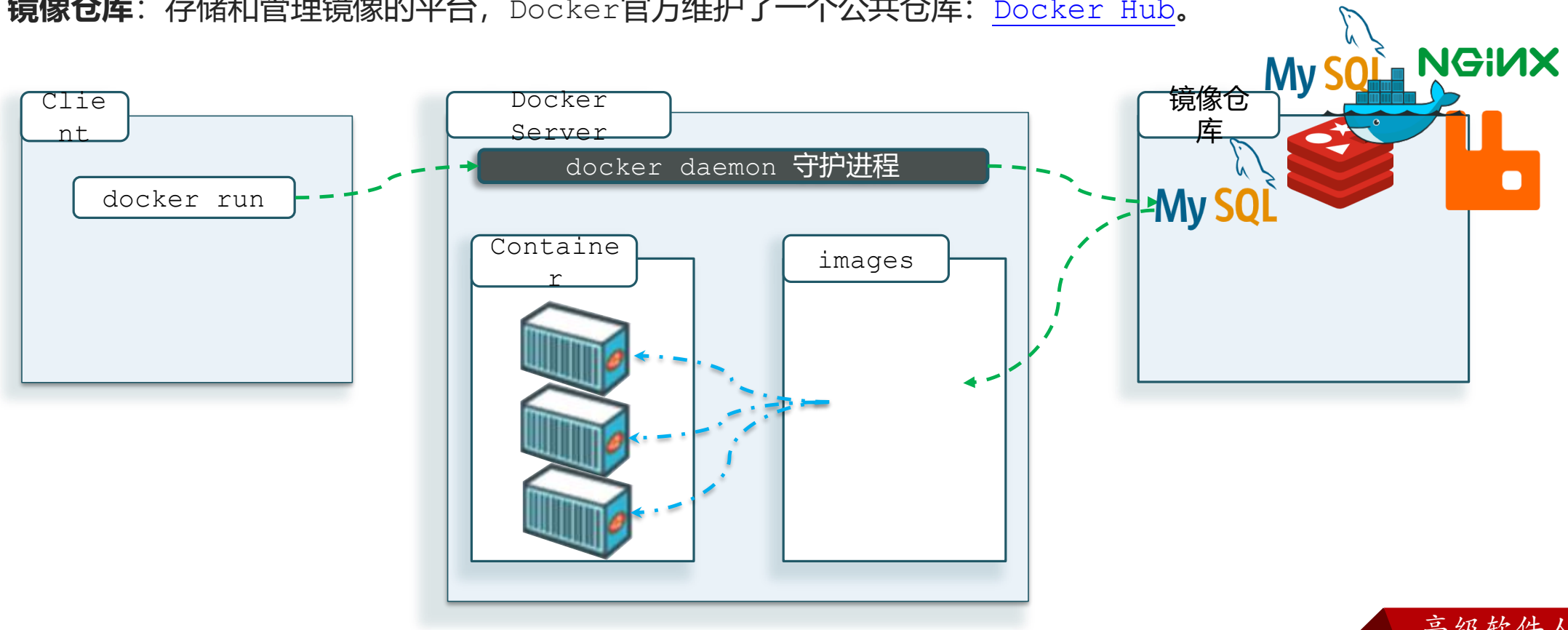
先停掉虚拟机中的MySQL，确保你的虚拟机已经安装Docker，且网络开通的情况下，执行下面命令即可安装MySQL：

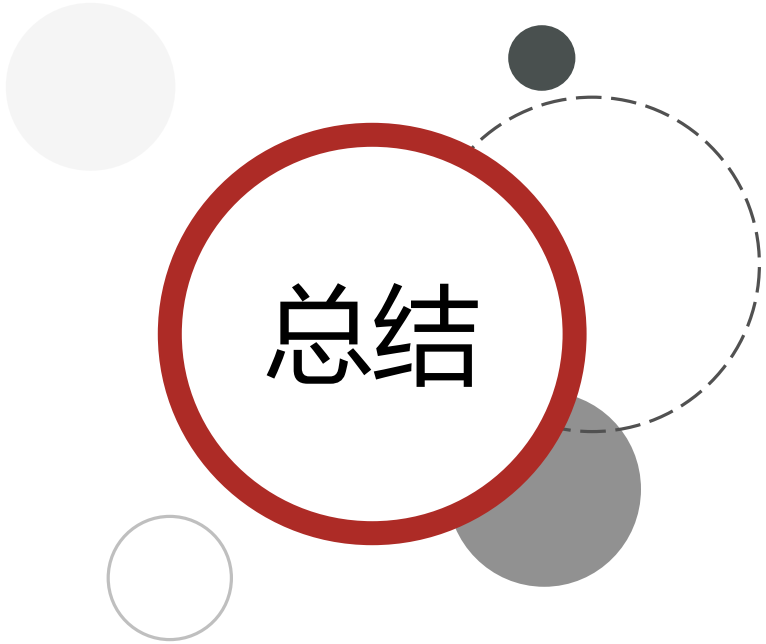
```
docker run -d \
  --name mysql \
  -p 3306:3306 \
  -e TZ=Asia/Shanghai \
  -e MYSQL_ROOT_PASSWORD=123 \
  mysql
```

## 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。






## Docker是做什么的?

- Docker可以帮助我们下载应用镜像，创建并运行镜像的容器，从而快速部署应用

## 什么是镜像?

- 将应用所需的函数库、依赖、配置等与应用一起打包得到的就是镜像

## 什么是容器?

- 为每个镜像的应用进程创建的隔离运行环境就是容器

## 什么是镜像仓库?

- 存储和管理镜像的服务就是镜像仓库，
- DockerHub是目前最大的镜像仓库，其中包含各种常见的应用镜像

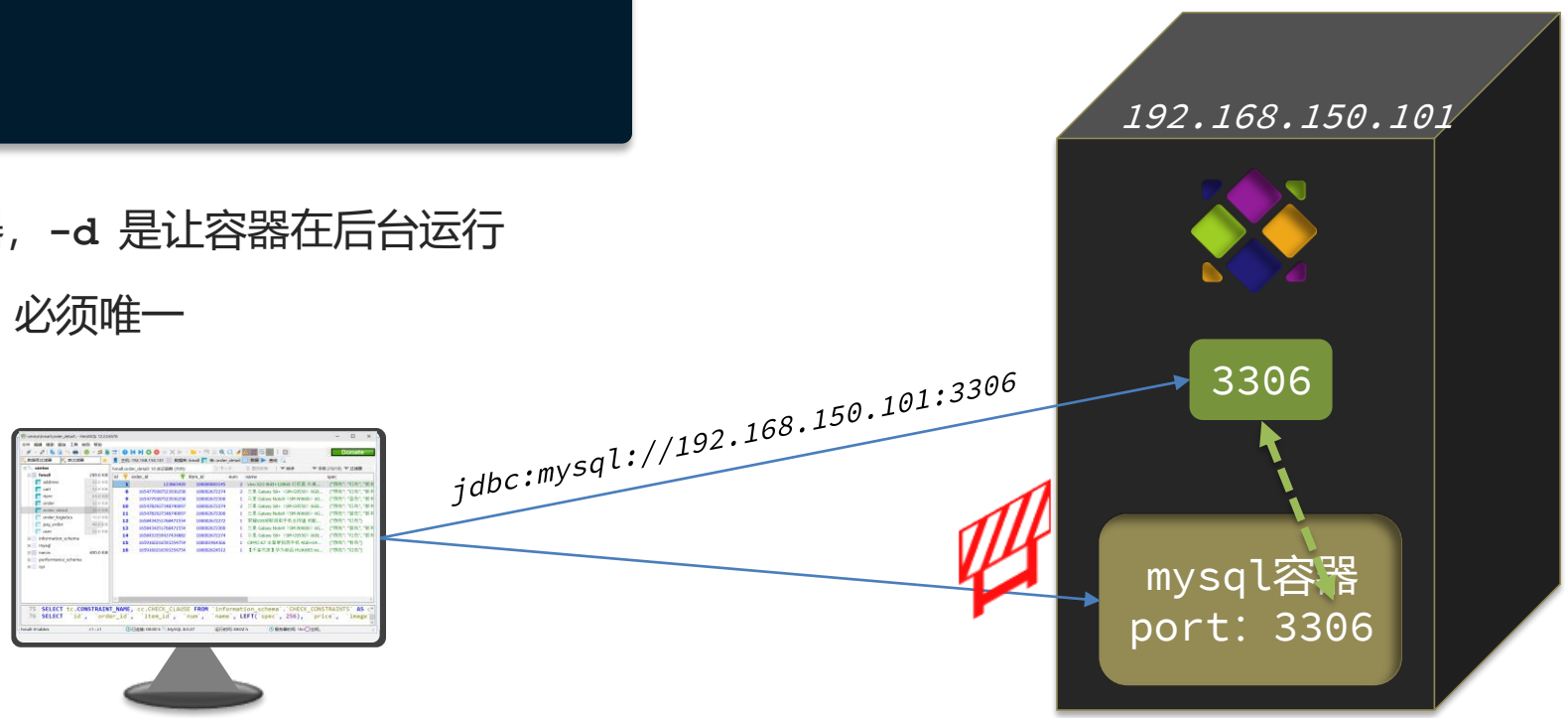
# 目录

Contents

## 命令解读

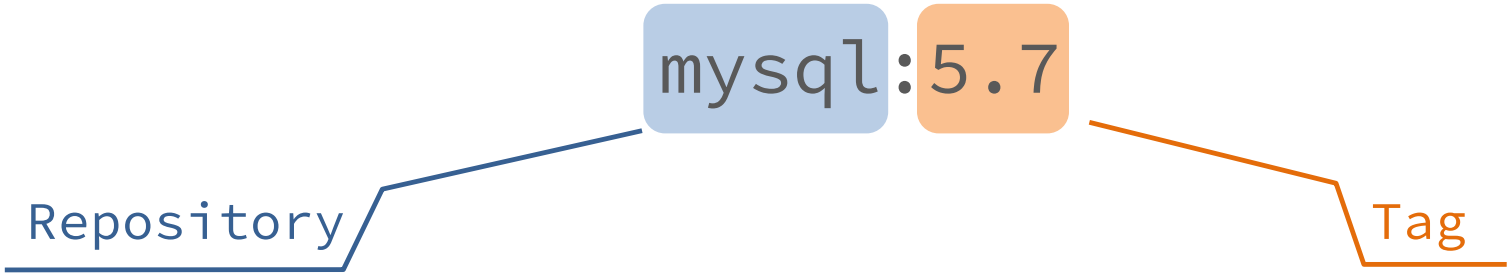
```
docker run -d \
  --name mysql \
  -p 3306:3306 \
  -e TZ=Asia/Shanghai \
  -e MYSQL_ROOT_PASSWORD=123 \
  mysql
```

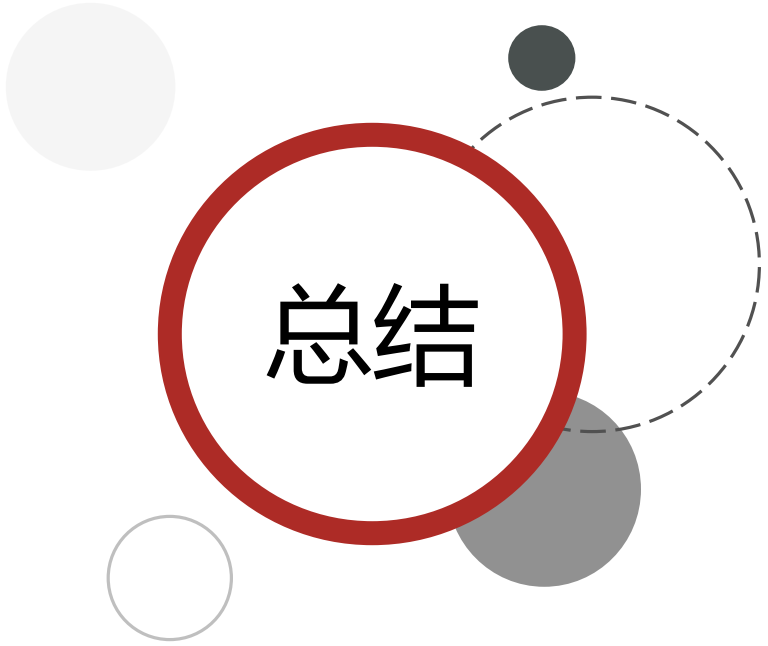
- **docker run** ：创建并运行一个容器，**-d** 是让容器在后台运行
- **--name mysql** ：给容器起个名字，必须唯一
- **-p 3306:3306** ：设置端口映射
- **-e KEY=VALUE** ：是设置环境变量
- **mysql** ：指定运行的镜像的名字



## 镜像命名规范

- 镜像名称一般分两部分组成：[repository]:[tag]。
  - 其中repository就是镜像名
  - tag是镜像的版本
- 在没有指定tag时，默认是latest，代表最新版本的镜像






docker run命令中的常见参数：

- **-d** ：让容器后台运行
- **--name**： 给容器命名
- **-e** ：环境变量
- **-p** ：宿主机端口映射到容器内端口

```
docker run -d \
  --name nginx \
  -p 80:80 \
  nginx
```

镜像名称结构：

- Repository:TAG

镜像名　　版本号

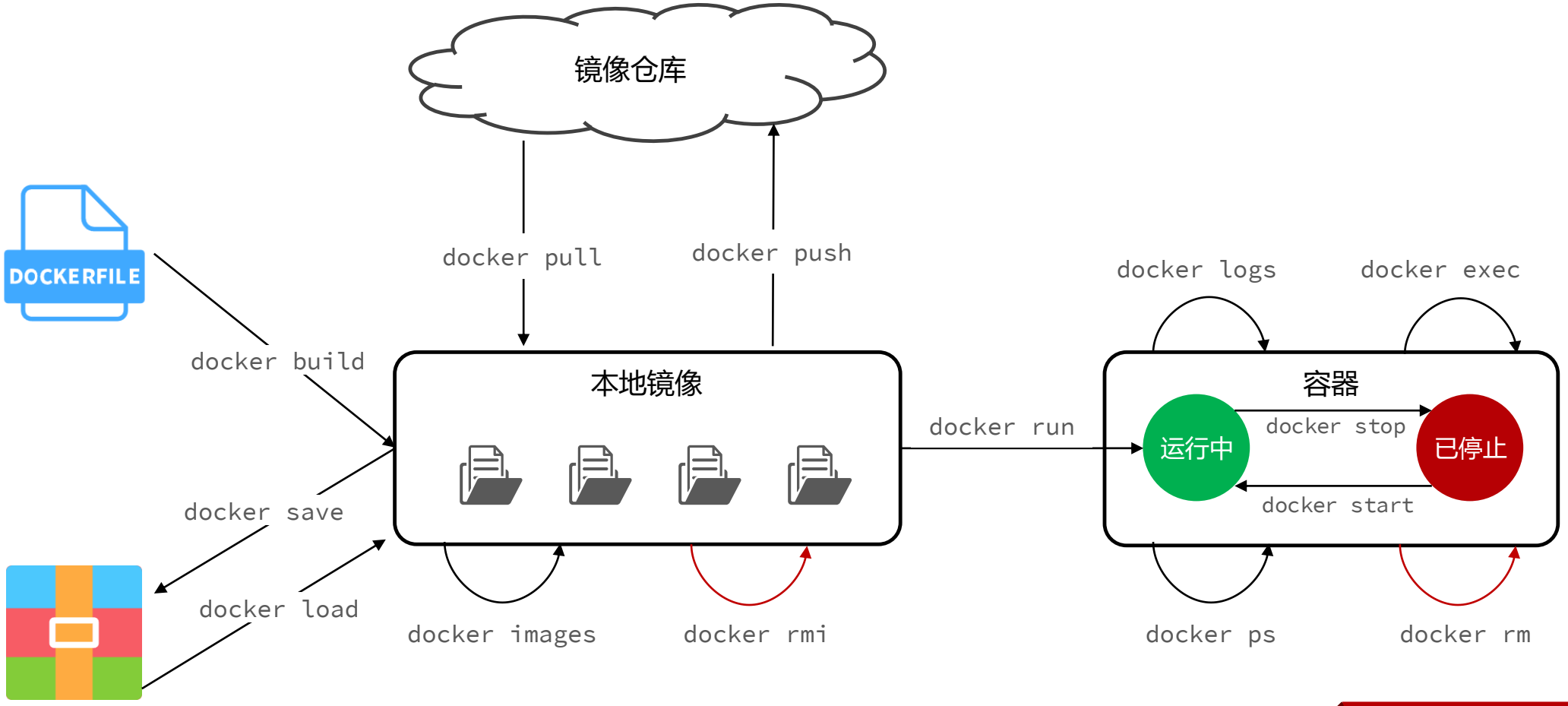

# Docker基础

# 目录

Contents

## 常见命令

Docker最常见的命令就是操作镜像、容器的命令，详见官方文档：https://docs.docker.com/



## 案例 查看DockerHub，拉取Nginx镜像，创建并运行Nginx容器

需求：

- 在DockerHub中搜索Nginx镜像，查看镜像的名称
- 拉取Nginx镜像
- 查看本地镜像列表
- 创建并运行Nginx容器
- 查看容器
- 停止容器
- 再次启动容器
- 进入Nginx容器
- 删除容器

# 目录

Contents

案例

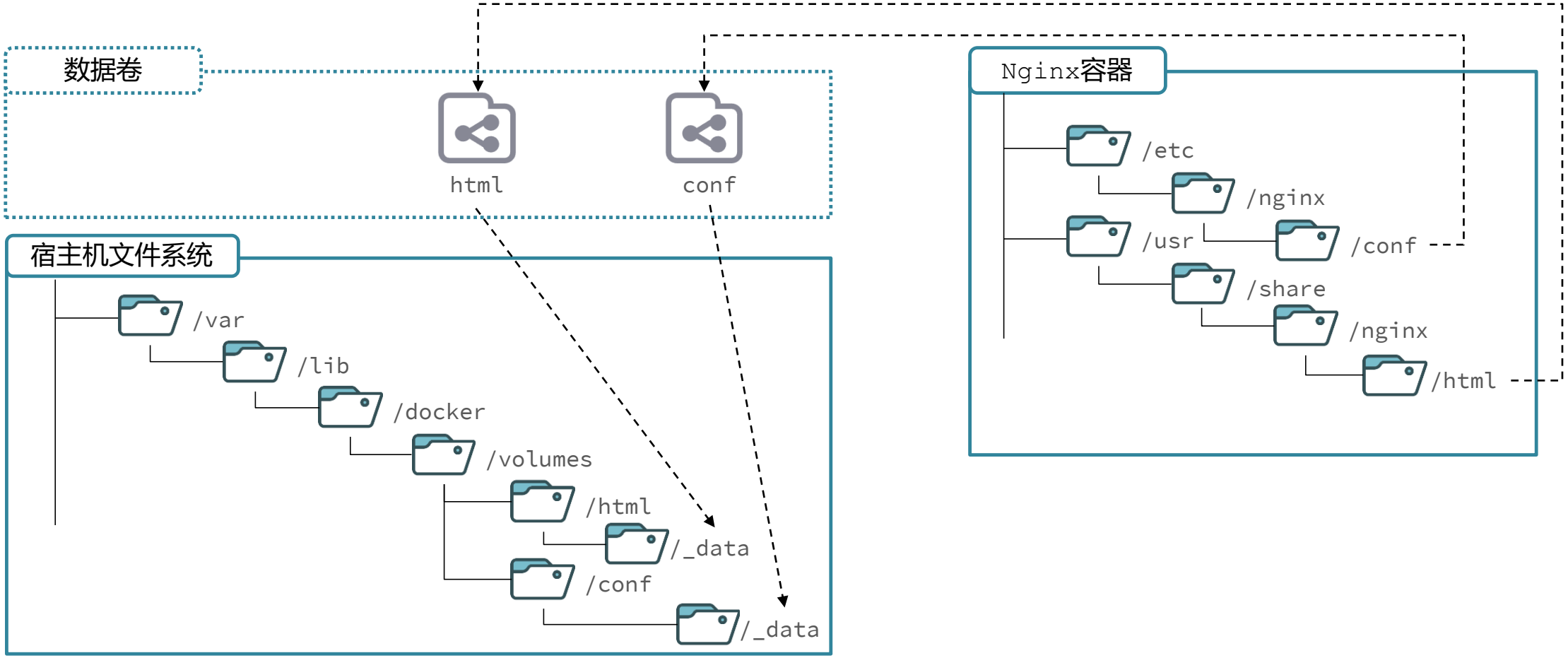
## 案例1-利用Nginx容器部署静态资源

需求：

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

# 数据卷

**数据卷（volume）**是一个虚拟目录，是**容器内目录**与**宿主机目录**之间映射的桥梁。



## 数据卷

| 命令 | 说明 | 文档地址 |
| --- | --- | --- |
| docker volume create | 创建数据卷 | docker volume create |
| docker volume ls | 查看所有数据卷 | docker volume ls |
| docker volume rm | 删除指定数据卷 | docker volume rm |
| docker volume inspect | 查看某个数据卷的详情 | docker volume inspect |
| docker volume prune | 清除数据卷 | docker volume prune |

案例

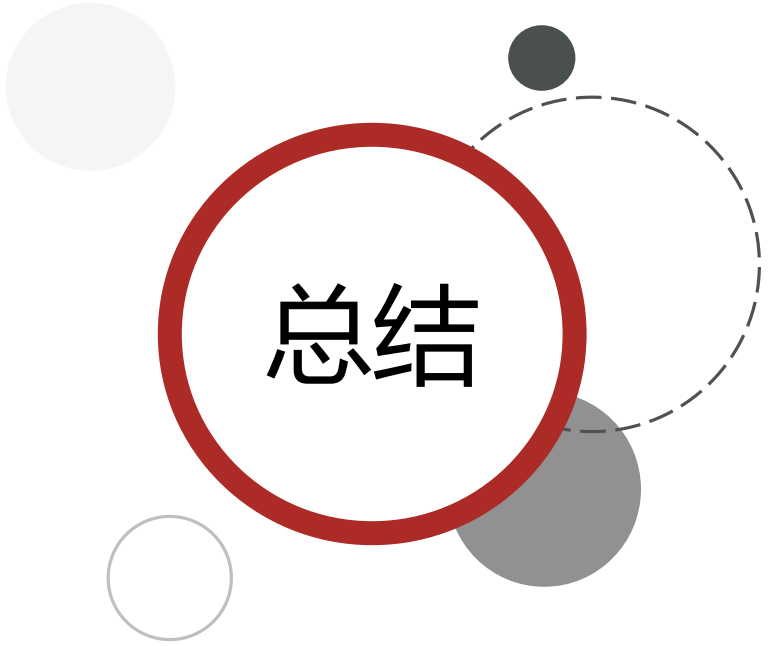
## 案例1-利用Nginx容器部署静态资源

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

提示

- 在执行docker run命令时，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，如果挂载了数据卷且数据卷不存在，会自动创建数据卷




## 什么是数据卷？

- 数据卷是一个虚拟目录，它将宿主机目录映射到容器内目录，方便我们操作容器内文件，或者方便迁移容器产生的数据

## 如何挂载数据卷？

- 在创建容器时，利用 `-v 数据卷名：容器内目录`完成挂载
- 容器创建时，如果发现挂载的数据卷不存在时，会自动创建

## 数据卷的常见命令有哪些？

- `docker volume ls`：查看数据卷
- `docker volume rm`：删除数据卷
- `docker volume inspect`：查看数据卷详情
- `docker volume prune`：删除未使用的数据卷

案例

# 案例2-mysql容器的数据挂载

需求：

- 查看mysql容器，判断是否有数据卷挂载
- 基于宿主机目录实现MySQL数据目录、配置文件、初始化脚本的挂载（查阅官方镜像文档）
  - ① 挂载/root/mysql/data到容器内的/var/lib/mysql目录
  - ② 挂载/root/mysql/init到容器内的/docker-entrypoint-initdb.d目录，携带课前资料准备的SQL脚本
  - ③ 挂载/root/mysql/conf到容器内的/etc/mysql/conf.d目录，携带课前资料准备的配置文件

提示

- 在执行docker run命令时，使用 **-v 本地目录：容器内目录** 可以完成本地目录挂载
- 本地目录必须以“/”或 "./" 开头，如果直接以名称开头，会被识别为数据卷而非本地目录
  - -v mysql：/var/lib/mysql 会被识别为一个数据卷叫mysql
  - -v ./mysql：/var/lib/mysql 会被识别为当前目录下的mysql目录

# 目录

Contents

# 自定义镜像

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。

**部署一个Java应用的步骤**

：

① 准备一个Linux服务器
② 安装JRE并配置环境变量
③ 拷贝Jar包
④ 运行Jar包

**构建一个Java镜像的步骤**

：

① 准备一个Linux运行环境
② 安装JRE并配置环境变量
③ 拷贝Jar包
④ 编写运行脚本

# 自定义镜像

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。

## 构建一个Java镜像的步骤

：

① 准备一个Linux运行环境

② 安装JRE并配置环境变量

③ 拷贝Jar包

④ 编写运行脚本

# 镜像结构

**入口（Entrypoint）**

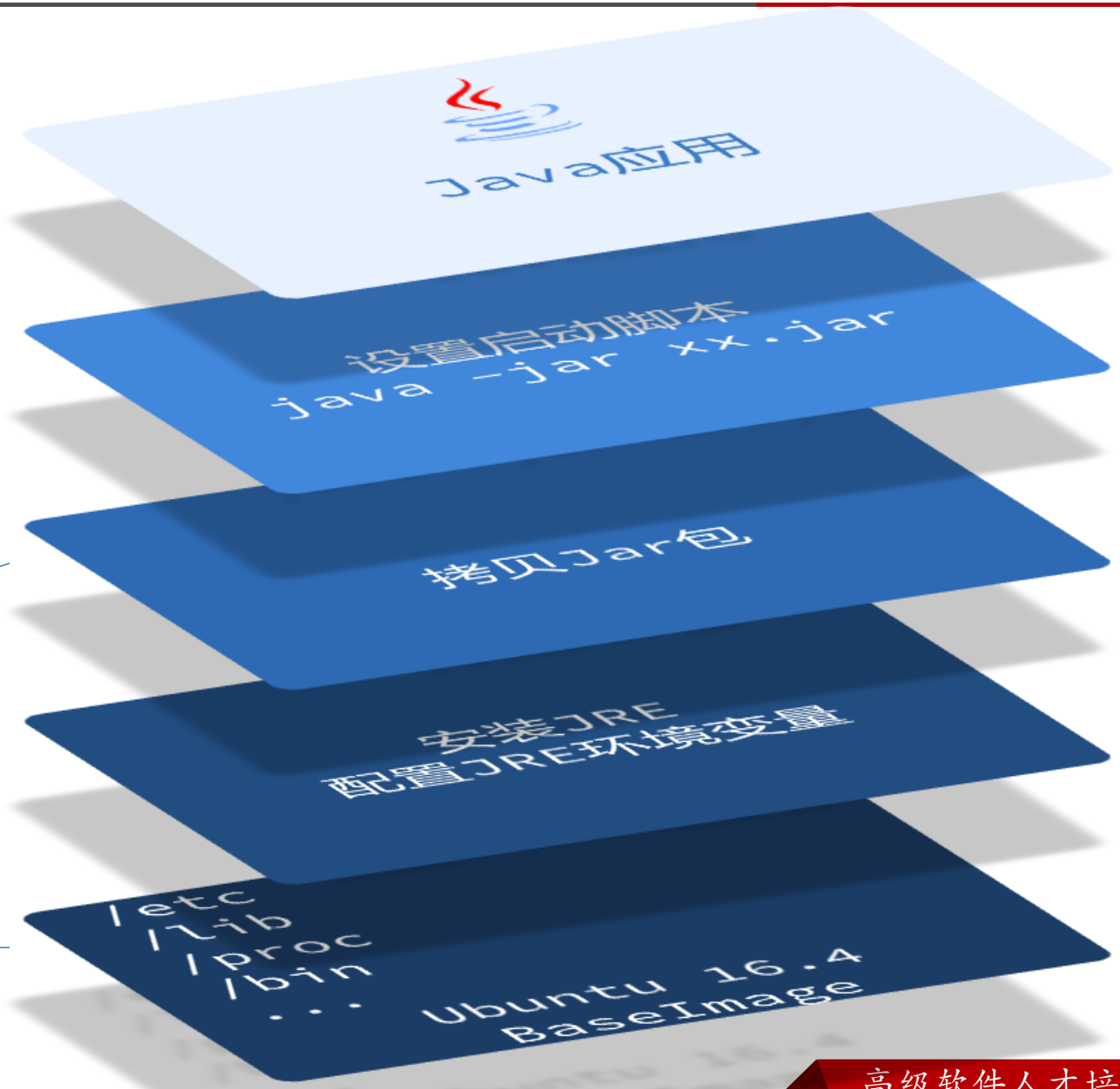
镜像运行入口，一般是程序启动的脚本和参数

**层（ Layer ）**

添加安装包、依赖、配置等，每次操作都形成新的一层。

**基础镜像（BaseImage）**

应用依赖的系统函数库、环境、配置、文件等



## Dockerfile

**Dockerfile**就是一个文本文件，其中包含一个个的**指令**(Instruction)，用指令来说明要执行什么操作来构建镜像。将来Docker可以根据Dockerfile帮我们构建镜像。常见指令如下：

| 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

# Dockerfile

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"]
```

# Dockerfile

我们可以基于Ubuntu基础镜像，利用Dockerfile描述镜像结，也可以直接基于JDK为基础镜像，省略前面的步骤：

```
# 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"]
```

```
# 基础镜像
FROM openjdk:11.0-jre-buster
# 拷贝jar包
COPY docker-demo.jar /app.jar
# 入口
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

## Dockerfile

我们可以基于Ubuntu基础镜像，利用Dockerfile描述镜像 ，也可以直接基于JDK为基础镜像，省略前面的步骤：

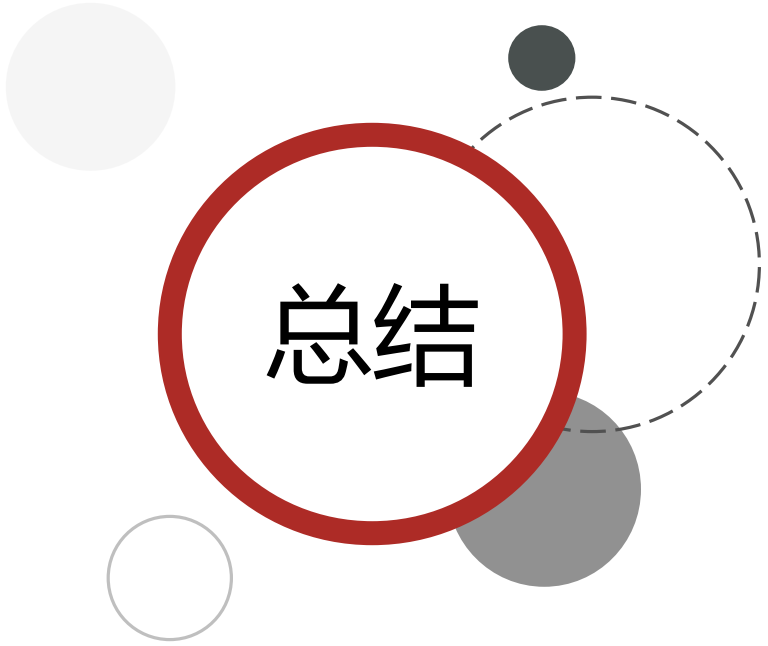
```
# 基础镜像
FROM openjdk:11.0-jre-buster
# 拷贝jar包
COPY docker-demo.jar /app.jar
# 入口
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

结构

当编写好了Dockerfile，可以利用下面命令来构建镜像

```
docker build -t myImage:1.0_.
```

- `-t`：是给镜像起名，格式依然是repository:tag的格式，不指定tag时，默认为latest
- `.` ：是指定Dockerfile所在目录，如果就在当前目录，则指定为"."




## 镜像的结构是怎样的?

- 镜像中包含了应用程序所需要的运行环境、函数库、配置、以及应用本身等各种文件，这些文件分层打包而成。

## Dockerfile是做什么的?

- Dockerfile就是利用固定的指令来描述镜像的结构和构建过程，这样Docker才可以依次来构建镜像

## 构建镜像的命令是什么?

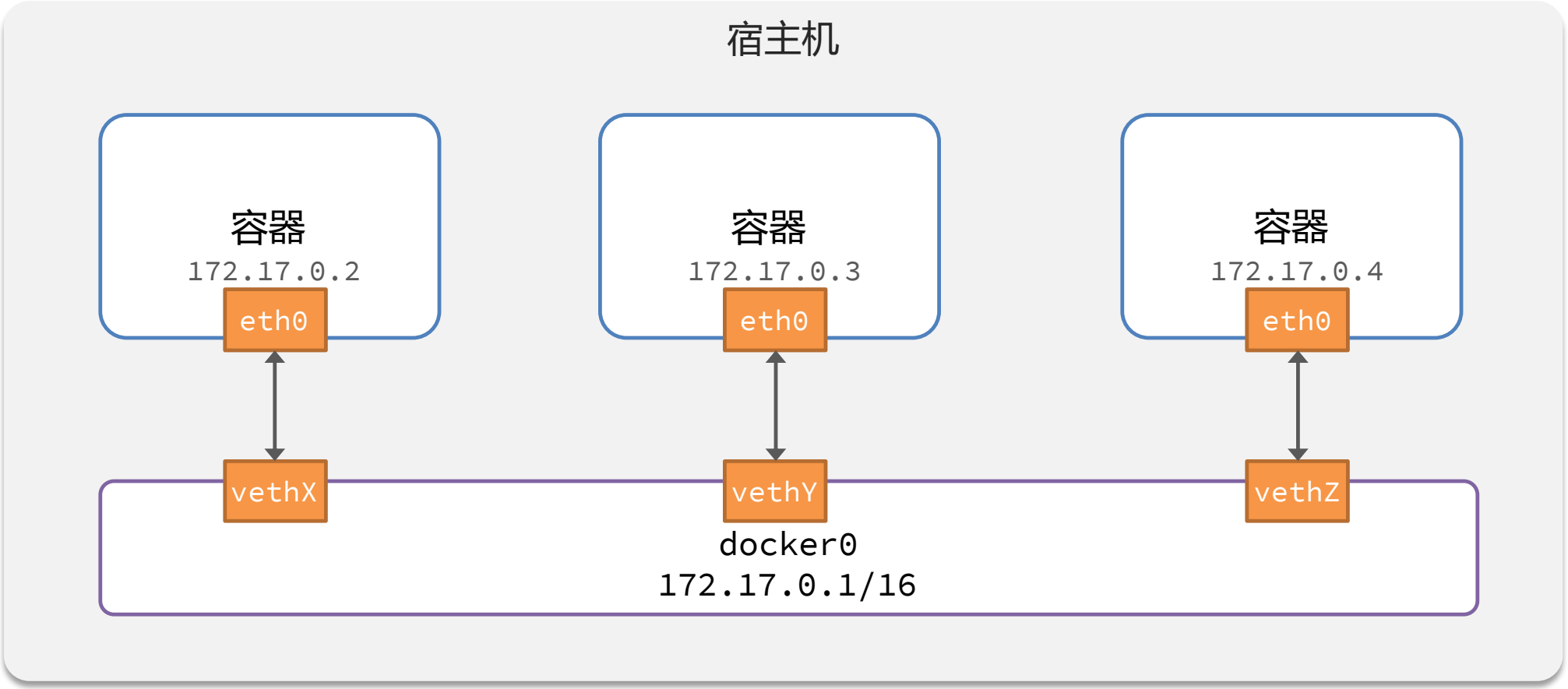
- docker build -t 镜像名 Dockerfile目录

# 目录

Contents

# 网络

默认情况下，所有容器都是以bridge方式连接到Docker的一个虚拟网桥上：



## 网络

加入自定义网络的容器才可以通过容器名互相访问，Docker的网络操作命令如下：

| 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 |

# 03 项目部署

# 目录

Contents

## 案例 部署Java应用

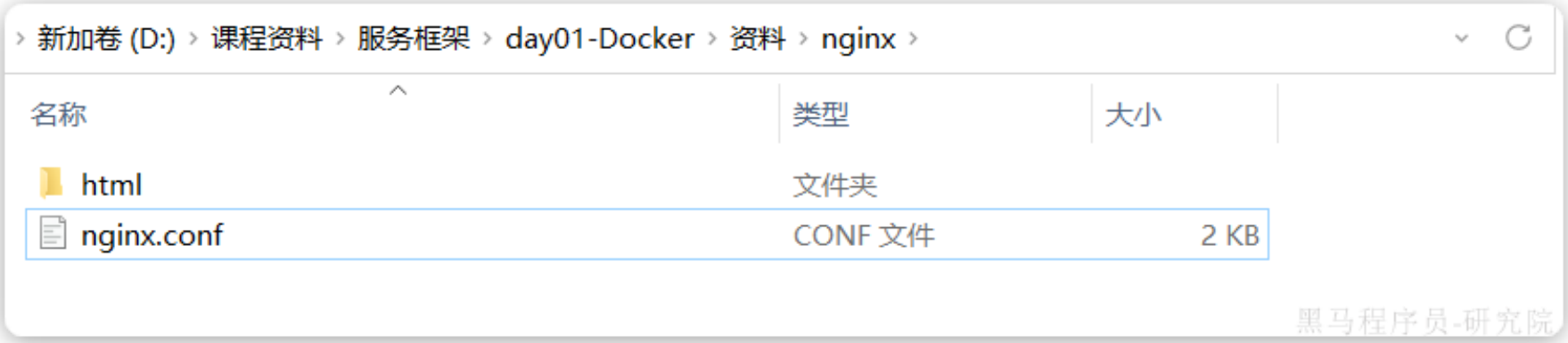
需求：将课前资料提供的hmall项目打包为镜像并部署，镜像名hmall

# 目录

Contents

## 案例 部署前端

需求：创建一个新的nginx容器，将课前资料提供的nginx.conf、html目录与容器挂载



# 目录

Contents

# DockerCompose

Docker Compose通过一个单独的**docker-compose.yml** 模板文件（YAML 格式）来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。**

**项目（Project）**

**服务（Service）**

```
version: "3.8"

services:
  containerA:
    image: A
    container_name: A
    ports:
      - "11:11"
  containerB:
    image: B
    container_name: B
    ports:
      - "22:22"
  containerC:
    image: C
    container_name: C
    ports:
      - "33:33"
```

# DockerCompose

Docker Compose通过一个单独的**docker-compose.yml** 模板文件（YAML 格式）来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。**

```
docker run -d \
  --name mysql \
  -p 3306:3306 \
  -e TZ=Asia/Shanghai \
  -e MYSQL_ROOT_PASSWORD=123 \
  -v ./mysql/data:/var/lib/mysql \
  -v ./mysql/conf:/etc/mysql/conf.d \
  -v ./mysql/init:/docker-entrypoint-initdb.d \
  --network hmall
  mysql
```

```
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:
      - hmall
```

# DockerCompose

Docker Compose通过一个单独的docker-comp

，帮助我们实现**多个相互关联的Docker容器的快**

```
docker run -d \
  --name mysql \
  -p 3306:3306 \
  -e TZ=Asia/Shanghai \
  -e MYSQL_ROOT_PASSWORD=123 \
  -v ./mysql/data:/var/lib/mysql \
  -v ./mysql/conf:/etc/mysql/conf.d \
  -v ./mysql/init:/docker-entrypoint-initdb.d
  --network hmall
  mysql
```

```
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:
      - new
  hmall:
    build:
       context: .
       dockerfile: Dockerfile
    container_name: hmall
    ports:
      - "8080:8080"
    networks:
      - new
    depends_on:
      - mysql
  nginx:
    image: nginx
```

# DockerCompose

Docker Compose通过一个单独的docker-comp

，帮助我们实现**多个相互关联的Docker容器的快**

```
docker run -d \
  --name mysql \
  -p 3306:3306 \
  -e TZ=Asia/Shanghai \
  -e MYSQL_ROOT_PASSWORD=123 \
  -v ./mysql/data:/var/lib/mysql \
  -v ./mysql/conf:/etc/mysql/conf.d \
  -v ./mysql/init:/docker-entrypoint-initdb.d \
  --network hmall
  mysql
```

```
    networks:
      - new
  hmall:
    build:
       context: .
       dockerfile: Dockerfile
    container_name: hmall
    ports:
      - "8080:8080"
    networks:
      - new
    depends_on:
      - mysql
  nginx:
    image: nginx
    container_name: nginx
    ports:
      - "18080:18080"
      - "18081:18081"
    volumes:
      - "./nginx/nginx.conf:/etc/nginx/nginx.conf"
      - "./nginx/html:/etc/nginx/html"
    depends_on:
      - hmall
    networks:
      - new
networks:
  new:
    name: hmall
```

# DockerCompose

**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 |