Docker 教程

Ubuntu 版

零基础入门系列教程

作者: 魏明择

2025 年版

https://weimingze.com

目录

第一章、Docker 容器简介

前言

本课程是针对网络运维人员、网站前端，后端开发工程师、App后端开发工程师、架构设计师、测试工程师的基础教程。

本课程将从容器原理开始讲解，直至能够部署自己需要的应用服务器。

本课程的内容由作者（魏明择）本人编写，请在线观看，禁止任何形式的复制、转发和修改。

本教程的案例均在 Ubuntu 24.04 LTS 上验证，学习本教程前请先自行安装 Ubuntu 24.04 LTS 操作系统或下载由本站制作的 vmware 虚拟机镜像。

Ubuntu24.04 VMware镜像:

- 百度网盘下载地址:
- https://pan.baidu.com/s/1jYAdvByctakhRVrDIteoLA 提取码: m8ee

Ubuntu下载和安装方法详见: Ubuntu Linux 简介

前置课程

在学习此内容之前你需要先学习 《Linux教程》。关于Ubuntu 操作系统的安装和虚拟机的下载和安装都在 《Linux教程》 中讲过，这里不在赘述。

为什么要学习 Docker?

Docker（容器技术）已经成为现代软件开发和运维的必备技能，无论是开发、测试、运维还是架构设计，Docker 都能显著提升效率。

为了更深入了解容器技术，我们需要先从**虚拟机技术**说起。

虚拟机技术

虚拟机技术（Virtual Machine, VM）是一种通过软件模拟完整计算机系统的技术，他允许在单台物理机上运行多个隔离的 **虚拟计算机**。

主流虚拟机产品

技术/产品	公司	特点
VMware ESXi	VMware	企业级虚拟化，高性能，付费授权。
KVM	Linux 社区	开源，集成到 Linux 内核，云厂商广泛使用。
VMware Workstation 或 VMware Fusion	VMware	个人电脑虚拟化，付费。
Hyper-V	Microsoft	Windows 内置，支持 Linux 虚拟机。
VirtualBox	Oracle	免费，适合开发和测试。
Xen	Linux 基金会	早期开源虚拟化，AWS EC2 初代采用。

国内云厂商大多使用的是 `VMware ESXi` 和 `KVM` 平台。

虚拟机的特点

特性	说明
完全隔离	虚拟机之间互不干扰，崩溃或病毒不影响宿主机和其他 VM。
硬件虚拟化	模拟 CPU、内存、磁盘等硬件资源，支持跨平台运行（如 Mac 上跑 Windows）。
独立系统	每个 VM 需独立安装操作系统和驱动，占用资源较多。
快照与克隆	可保存虚拟机状态（快照），快速复制环境（克隆）。

虚拟机的应用场景

1. 企业服务器整合：单台物理机运行多个业务 VM，节省硬件成本。
2. 跨平台开发：Mac 用户通过 VM 运行 Windows 专属软件（如 IE 浏览器）。
3. 安全测试：在隔离的 VM 中分析恶意软件或漏洞。
4. 传统应用迁移：老旧系统（如 Windows Server 2003）在 VM 中延续使用。
5. 云服务构建：使用虚拟机技术创建云平台（公有云和私有云）。

虚拟机的局限性

1. 资源开销：运行多个 VM 需大量 CPU 和内存。
2. 启动慢：不适合需要快速扩缩容的场景。
3. 管理复杂：需维护多个 OS 的补丁和许可证。

虚拟机虽然好，但由于存在多个操作系统，因此操作系统占用的系统开销比较大。综上所述，容器的技术变得更加流行。

1. 容器概述

什么是容器

容器（Container）是一种轻量级的虚拟化技术，他是一种轻量级、可移植、自包含的软件打包技术，用于将应用程序及其依赖环境打包成一个独立的、可移植的单元。容器通过共享宿主机的操作系统内核，实现了资源隔离和进程隔离，但比传统虚拟机更高效（无需模拟完整操作系统）。

一个容器（Container）通常是一个轻量级、可执行的独立软件包，它包含了运行某个应用程序所需的所有内容，包括：根文件系、网络、名字空间、运行时环境、系统工具、运行时库和配置等。

使用容器技术，开发人员在自己笔记本上创建并测试好的容器，无需任何修改就能够在生产系统的虚拟机、物理服务器或公有云主机上运行。

容器的概念和轮船的集装箱的设计和理念高度相似。

容器的特点如下：

1. 标准化
   - 无论内部封装的是什么，外部体现都是一致的，有统一的接口。
2. 隔离性：
   - 每个容器内部封闭且独立，容器之间互不干扰。

3. 便携性：
   - 封装后的容器在不同的运行环境中无需重新配置。一次创建，处处运行。
4. 效率
   - 容器共享主机操作系统内核，无需为每个应用启动完整的虚拟机，资源占用极低。
5. 分层与复用
   - 镜像通过分层存储实现复用（如基础 Ubuntu 层 + 应用层），减少重复占用空间。

主流容器技术如下：

容器技术	所属公司	是否收费	效率特点
Docker	Docker Inc.	社区版免费，企业版收费	轻量级，启动快（秒级），资源占用低，镜像分层复用
Podman	Red Hat	开源免费	兼容 Docker CLI，无需守护进程，Rootless 模式更安全
LXC/LXD	Canonical (Ubuntu)	开源免费	接近虚拟机的隔离性，启动速度较慢（分钟级）
Kata Containers	OpenStack 基金会	开源免费	通过轻量级 VM 实现强隔离，性能损耗较高（约 10-20%）
gVisor	Google	开源免费	用户态内核拦截，安全性高，性能损耗显著（约 40-50%）
Firecracker	AWS	开源免费	微虚机（MicroVM）技术，启动快（毫秒级），内存开销极低
CRI-O	Red Hat	开源免费	专为 Kubernetes 优化，极简设计，性能与 containerd 相当
Windows Containers	Microsoft	免费（需 Windows 授权）	基于 Hyper-V 隔离，Windows 生态专用，资源占用较高

容器的原理

一个容器就是一个进程（可以由此进程创建多个子进程）的运行时环境，这些进程共享宿主机的操作系统内核，由操作系统统一调度管理。以后我们把容器内最先启动的进程叫 **容器主进程**。

容器主进程 通常是容器管理进程的子进程，**容器主进程** 继承了父进程（容器管理进程）的权限，并且子进程的权限一定会小于父进程的权限。

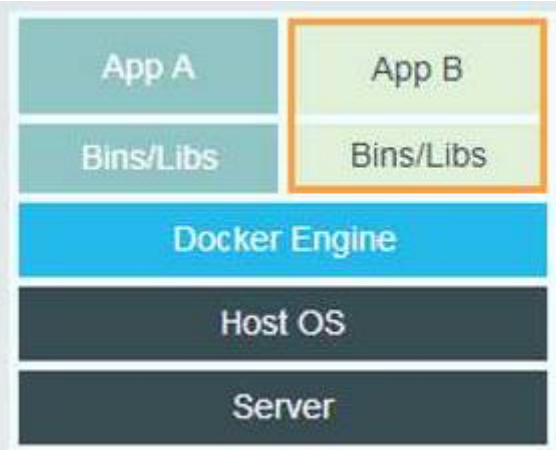
容器管理进程通过操作系统内核对命名空间（Namespaces）、 控制组（Cgroups）、 联合文件系统（UnionFS）、网络等进行隔离，让**容器主进程**在自己独立的环境中运行，从而达到虚拟化的效果。

容器与虚拟机的架构设计

容器和虚拟机之间的主要区别在于虚拟化层的位置和操作系统资源的使用方式。 如下图所示:

虚拟机结构

容器结构



可见，虚拟机通过 Hypervisor 层有了自己的客户端操作系统 (Guest OS), 而容器是没有操作系统的，他是通过容器引擎(Docker Engine) 来共用主机操作系统(Host OS)。

容器与虚拟机参数对比

参数	容器（Docker）	对比	虚拟机
创建	启动应用	>	启动Guest OS+启动应用
部署	容器镜像	=	虚拟机镜像
密度	单节点 100~1000个	>	单节点 10~100个
更新管理	对增量内容进行分发，存储，节点启动	>	向虚拟机推送安装，升级应用软件补丁包
启动时间	秒级启动	>	分钟级启动
轻量级	镜像大小通常以M为单位	>	虚拟机以G为单位
性能	共享宿主机内核，系统级虚拟化，占用资源少，性能基本接近物理机	>	虚拟机需要 Hypervisor 层支持，具有完整的GuestOS，虚拟化开销大。
安全性	Docker具有宿主机 `root` 权限，有一定安全隐患	<	硬件隔离，相对安全
高可用性	通过业务本身的高可用性来保证	<	工具丰富：快照，克隆，HA，动态迁移，异地容灾，异地双活

使用要求	共享宿主机内核，不用考虑CPU是否支 持虚拟化技术	>	基于硬件的完全虚拟化，需要硬件 CPU 虚拟化技术支持

容器的优点

1. 轻量高效：共享宿主机内核，无需虚拟化完整操作系统，资源占用少，启动速度快（秒级）
2. 环境一致性：镜像打包应用及其依赖，确保开发、测试、生产环境一致，避免环境影响。
3. 快速部署与扩展：支持自动化部署（CI/CD），结合编排工具（如：Kubernetes）可快速横向扩展。
4. 资源隔离与限制：通过 Cgroups 限制 CPU、内存等资源，避免单个容器耗尽系统资源。
5. 微服务友好：将业务拆分为独立容器，实现低耦合的微服务架构，便于开发和扩展。

容器的缺点

1. 安全性较弱：共享内核，若内核漏洞被利用，可能导致容器逃逸（攻击宿主机）。
2. 环境依赖：基于操作系统内核，Windows 下的镜像无法在 Linux 下运行。
3. 存储与数据持久化复杂：默认容器内数据易丢失，需额外配置卷（Volume）或外部存储。
4. 网络配置复杂：多容器通信需管理网络插件、端口映射等，跨主机网络更复杂。
5. 不适合所有场景：对图形界面、高性能计算（HPC）或强隔离需求的应用支持较差。

本教程将以 Docker 为例对容器进行讲解。

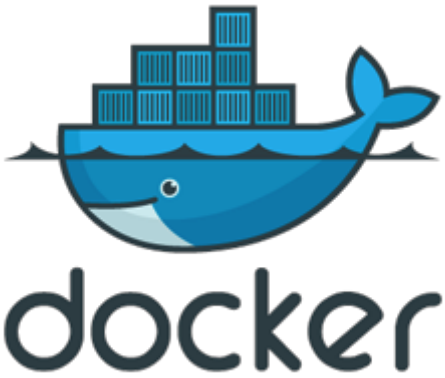
2. Docker 简介和安装

什么是Docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux/Windows等机器上，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。

> Docker 是最受大众关注的容器技术，并且现在“几乎”成为事实上的容器标准。

Docker 的 Logo



Docker的官方网站

- 网址: https://www.docker.com/

在官网我们可以找到 `Download Docker Desktop` 来下载各个平台的 Docker 桌面版安装包，如下图所示：

Docker的应用场景

- Web 应用的自动化打包和发布，如 Nginx。
- 各种服务的搭建，如: MySQL、Redis、httpd、uWSGI 等。
- 自动化测试和持续集成、发布。

3. 安装和启动Docker

Docker 是以服务的形式存在的，要使用 Docker 需要再操作系统上安装 Docker 服务。

Docker 服务的安装

- Ubuntu 24.04 安装 Docker 服务

```
sudo apt install docker.io
```

如果没有找到 docker.io 这个包请使用 `apt update` 更新软件包列表，如果再没有成功，请更换 apt 源，详见：《Ubuntu 换源》。

- CentOS7 安装 Docker 服务

```
sudo yum install docker
```

验证 Docker 服务安装是否成功

```
weimingze@mzstudio:~$ docker --version
Docker version 27.5.1, build 27.5.1-0ubuntu3~24.04.2
```

如果运行 `docker --version` 命令能够正确的显示 Docker 的版本，说明 Docker 已经安装成功。

启动 Docker 服务

只有当 Docker 服务启动时才能对 Docker 进行操作。

- Ubuntu/CentOS 下启动服务

```
sudo systemctl start docker
```

停止 Docker 服务

- Ubuntu/CentOS 下停止服务

```
sudo systemctl stop docker
```

Docker 设置开机自启动

让系统开机后自动启动 Docker，否则重启系统后容器不会自动运行。

```
sudo systemctl enable docker
```

检查是否设置了 Docker 开机自启动。

```
systemctl is-enabled docker
```

Docker 禁用开机自启动

```
sudo systemctl disable docker
```

第二章、搭建 MySQL 服务器

本节课我们先学习使用 Docker 来搭建一个 MySQL 数据库服务器，然后再来深入了解 Docker 的操作。

要使用 Docker 我们需要理解 **仓库**、**镜像** 和 **容器** 三个概念.

Docker 的三大核心概念

- 镜像：镜像是包含容器运行所需的文件系统和配置信息的集合，Docker的镜像是只读的。镜像你可以理解成虚拟机的镜像或一台笔记本电脑的硬盘，开机即可启动的硬盘。
- 仓库：使用来存储、分发和管理镜像的地方。方便分布式系统分发镜像。
- 容器：容器是运行在隔离环境中的容器主进程及其子进程。容器是镜像运行的实例，一个镜像可以运行成为多个容器。你可以理解成已经开机并运行了一个进程的电脑。

要搭建 MySQL 服务器，我们需要先有一个别人做好的 MySQL 镜像，然后再启动这个镜像，让其成为容器即可。

实验步骤如下：

1. 获取 MySQL 镜像。
2. 使用 镜像搭建 MySQL 服务（容器）。
3. 在宿主机使用 MySQL 客户端连接服务器。

1. 获取 MySQL 镜像

获取镜像有两种方法：

1. 从远程仓库中下载镜像。
2. 从本地文件中导入镜像。

如果同学们第一种方法失败可以尝试使用第二种方法。

1. 从远程仓库中下载镜像

使用 `docker pull` 命令可以从远程仓库中下载名为 `mysql`，标签为 `latest` 的镜像。命令如下：

```
sudo docker pull docker.io/library/mysql:latest
```

查看镜像是否下载成功。命令和结果如下：

```
weimingze@mzstudio:~$ sudo docker images
[sudo] password for weimingze:
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
```

上面如果出现了 mysql 这一行，说明 mysql 镜像下载成功。不成功则看不到镜像，如下所示：

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
```

目前为了安全，目前 `docker pull` 命令无法访问国外的 Docker 的镜像仓库（`"https://registry-1.docker.io/`），如果非要访问需要非常规手段。

因国家强制要求国内的 Docker 镜像仓库必须保证所有的镜像都是安全的。因此国内的 Docker 镜像源服务器也相继停止服务。

截止到今天 2025年6月17日，国内的 Docker 镜像仓库都无法访问，因此我们需要使用 从本地文件中导入镜像 的方法来安装 mysql 镜像。

2. 从本地文件中导入镜像

先从百度网盘下载 镜像压缩包。

百度网盘链接地址：https://pan.baidu.com/s/1U2CUtna3oq9NFkh8ewMzdg 提取码: 9gy6

找到 `docker教程资料/mysql_docker镜像` 文件夹下的 `mysql9.3_images.tar.xz` 并下载。然后使用 scp 命令复制到 Ubuntu24.04 操作系统内。

> 声明： `mysql9.3_images.tar.xz` 是官方下载的镜像，未经验证是否存在安全漏洞，只建议学习使用，商用后果自行承担。

执行 `docker load` 命令导入 `mysql9.3_images.tar.xz` 如下:

```
weimingze@mzstudio:~$ sudo docker load -i mysql9.3_images.tar.xz
[sudo] password for weimingze:
Loaded image: mysql:latest
```

导入前使用 `docker images` 命令查看全部镜像，如下:

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
```

导入后查看全部镜像.

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
```

可见镜像已经导入成功。

2. 搭建 MySQL 服务

使用 `docker run` 命令启动容器，镜像使用 `mysql`，容器名自定义为 `mysql_server`,root 用户的密码定义为 `weimingze.com`，服务器端口号依旧为 3306 并将其映射到容器外部。

命令如下:

```
sudo docker run --name mysql_server -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=weimingze.com -d mysql
```

运行结果:

```
weimingze@mzstudio:~$ sudo docker run --name mysql_server -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=weimingze.com -d mysql
43f8c0d15ef2a35360df6188687c3774b50de6a122c3c9c05b95f4d17ef3a749
```

> 其中 `43f8c0d15ef2a35360df6188687c3774b50de6a122c3c9c05b95f4d17ef3a749` 这一串数字是启动容器的容器 ID，

使用 `docker ps` 命令查看是否存在正在运行的容器 mysql_server,命令如下：

```
weimingze@mzstudio:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
          PORTS                                                 NAMES
43f8c0d15ef2   mysql     "docker-entrypoint.s…"   About a minute ago   Up About
a minute   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql_server
```

可见 容器已经正常运行，如果容器没有运行成功，则结果如下：

```
weimingze@mzstudio:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

3. 连接 MySQL 服务

下面我们使用 mysql 的客户端软件连接 MySQL 服务器。

首先下载并安装 MySQL 的客户端软件 `mysql-client`，命令如下：

```
sudo apt install mysql-client
```

使用如下命令连接 MySQL 服务器：

```
mysql -h 127.0.0.1 -u root -p
```

执行结果

```
weimingze@mzstudio:~$ mysql -h 127.0.0.1 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 9.3.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)

mysql> quit;
Bye
```

以上 MySQL 的 `root` 密码是 `weimingze.com`

在 MySQL 客户端内使用 `show databases;` 可以查看内部的数据库，说明 数据库安装成功。

至此，MySQL数据库安装结束。

第三章、Docker 镜像操作

Docker 的镜像是 Docker 容器的核心，Docker 的镜像必须在本地才能运行 Docker 容器。

Docker 镜像的获取方法:

1. 从 Docker 镜像仓库下载
2. 从镜像的压缩tar 包中导入镜像。
3. 自己制作镜像

常用的 Docker 镜像相关的命令有如下几种：

功能	命令
列出本地镜像	`docker images` 或 `docker image ls`
下载镜像	`docker pull <镜像名:标签>`
从 Docker Hub 搜索镜像	`docker search <关键词>`
从文件导入镜像	`docker load -i <tar包文件路径名>`
导出镜像为文件	`docker save -o <文件名.tar> <镜像名:标签>`
删除镜像	`docker rmi <镜像ID或名称>` 或 `docker image rm <镜像ID或名称>`
清理无用镜像	`docker image prune`
查看镜像详情	`docker inspect <镜像ID或名称>`
查看镜像历史	`docker history <镜像名>`
通过 Dockerfile 创建镜像	`docker build -t <镜像名:标签> <上下文路径>`
从容器创建新镜像	`docker commit <容器ID> <新镜像名:标签>`
镜像标签管理(打标签)	`docker tag <原镜像名> <新镜像名:标签>`
查看镜像磁盘占用	`docker system df`

因为 Docker 是运行在 root 下的服务。因此在执行上述命令式经常要使用 sudo 进行操作。

1. 列出本地镜像

查看本地 Docker 镜像可以使用 如下命令进行:

```
sudo docker images
```

也可以使用 `docker images` 命令的别名 `docker image ls` 或 `docker image list` 代替 `docker images`

示例

```
weimingze@mzstudio:~$ sudo docker images;
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
mysql         latest    edbdd97bf78b   2 months ago   859MB
ubuntu        24.04     602eb6fb314b   2 months ago   78.1MB
```

上述列出了所有的本地 Docker 镜像，每一列内容如下：

- 第一列 `REPOSITORY` 列是镜像的名称, 如 `mysql`。
- 第二列 `TAG` 列是镜像的标签，通常用于区分同名镜像的版本，如: `24.04` 表示是 `ubuntu` 镜像的 `24.04` 版本； `latest` 表示最新的版本。
- 第三列 `IMAGE ID` 是镜像的ID，用于标识一个唯一的镜像，是在制作镜像时自动生成的 `sha256` 算法的签名信息。
- 第四列 `CREATED` 是 镜像创建的时间，如： `2 months ago` 表示两个月前。
- 第五列 `SIZE` 是 镜像占用磁盘空间的大小。

docker image 命令的常用选项

选项	说明
`-a` 或 `--all`	显示所有镜像，默认不显示中间层镜像。
`-q`	仅显示镜像的 ID 列表（常用于批量删除镜像）。
`--no-trunc`	显示全部信息，不截断 `IMAGE ID`

> 使用 `docker images --help` 可以查看 `docker images` 子命令的所有选项。后续所有子命令选项查询方法相同。

示例

```
weimingze@mzstudio:~$ sudo docker images -a
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker images -q
edbdd97bf78b
602eb6fb314b
weimingze@mzstudio:~$ sudo docker images --no-trunc
REPOSITORY   TAG       IMAGE ID
                CREATED        SIZE
mysql        latest    sha256:edbdd97bf78b4338bbb96fa1348c8743a328b97ea3290b20ad
cab25bc17637de   2 months ago   859MB
ubuntu       24.04     sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b
70d7ac0e554b1c   2 months ago   78.1MB
```

2. Docker Hub 上搜索镜像

Docker Hub 是一个由 Docker 公司运营的公共容器镜像仓库，它是全球最大的容器镜像库之一，由Docker公司维护。Docker Hub 上面有 Docker 公司提供的镜像及大量用户上传的镜像。也是 Docker 生态系统的重要组成部分。

Docker Hub 的官方网址：

- 网址：https://hub.docker.com

我们可以在这里查找、下载、存储和分享 Docker 镜像。

> 注意：目前 Docker Hub 网站可能无法打开，要打开可能需要非常规方法。

这里我们来使用终端命令来查看 Docker Hub 镜像仓库上的镜像名称和版本。

命令格式

```
sudo docker search <关键词>
```

示例

查找 mysql 相关的镜像

```
weimingze@mzstudio:~$ sudo docker search mysql;
NAME                            DESCRIPTION                                     S
TARS     OFFICIAL   AUTOMATED
mysql                           MySQL is a widely used, open-source relation…   1
5689     [OK]
```

```
mariadb                         MariaDB Server is a high performing open sou…   6
240      [OK]
mysql/mysql-server              Optimized MySQL Server Docker images. Create…   1
114                  [OK]
percona                         Percona Server is a fork of the MySQL relati…   1
261      [OK]
phpmyadmin                      phpMyAdmin - A web interface for MySQL and M…   1
034      [OK]
bitnami/mysql                   Bitnami MySQL Docker Image                      8
9                    [OK]
circleci/mysql                  MySQL is a widely used, open-source relation…   3
1
ubuntu/mysql                    MySQL open source fast, stable, multi-thread…   4
7
mysql/mysql-cluster             MySQL Cluster provides linear scalability an…   5
7
databack/mysql-backup           Simple MySQL backup solution                    3
9                    [OK]
```

每一列的字段说明如下：

- `NAME` 镜像名称（带命名空间的如 mysql/mysql-server 表示非官方镜像）。
- `DESCRIPTION` 镜像的简要描述。
- `STARS` 镜像的星标数（受欢迎程度）。
- `OFFICIAL` 是否官方镜像（标记为 [OK] 的表示 Docker 官方维护）。
- `AUTOMATED` 是否由 Docker Hub 自动创建（通过 Dockerfile 自动创建的镜像）。

我们下载镜像优先选择 Docker 官网维护的镜像。

常用选项

选项	说明
`--filter=is-official=true`	只显示官方镜像。
`--filter=stars=N`	只显示星标数 ≥ N 的镜像
`--no-trunc`	显示完整的描述信息（不截断）

3. 远程镜像仓库下载镜像

使用 `docker pull` 命令可以直接从远程镜像仓库下载 Docker 镜像到本地。

命令格式

```
docker pull <镜像名:标签>
```

示例

```
docker pull ubuntu:24.04
docker pull mysql
```

> 如果不给出标签，则默认会下载最新的版本 `latest`

4. 本地镜像的导出和导入

如果远程镜像仓库无法用 `docker pull` 命令下载 Docker 镜像，我们可以直接从镜像仓库下载镜像包，然后导入到本地系统中。也可以将本地 Docker 内部的镜像导出成为 归档文件。

1. 导出镜像的命令 `docker save`

> 命令别名: `docker image save`

命令格式

导出成为 tar 包文件的命令:

```
docker save <镜像名:标签>
# 或
docker save -o <文件名.tar> <镜像名:标签>
```

默认导出镜像为标准输出 `STDOUT`，然后可以通过管道交给 `gzip`或 `xz`等命令进行数据压缩。可以使用`-o` 现象直接导出为 `.tar` 归档文件。

> 如果不给出标签，默认导出标签为`latest` 的镜像。

docker save 命令的常用选项

选项	说明
`-o <归档文件路径>`	将文件导出到指定位置，代替标准输出 `STDOUT`。

示例

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker save -o ubuntu24.04.tar ubuntu:24.04  # 导出为 tar 包
weimingze@mzstudio:~$ sudo docker save ubuntu:24.04 | gzip > ubuntu24.04.tar.gz  # 导出为 tar.gz 压缩包
weimingze@mzstudio:~$ sudo docker save ubuntu:24.04 | xz > ubuntu24.04.tar.xz  # 导出为 tar.xz 压缩包
weimingze@mzstudio:~$ sudo docker save mysql | xz > mysql.tar.xz  # 导出 mysql:latest 为 .tar.xz 压缩包
weimingze@mzstudio:~$ ls
mysql.tar.xz  ubuntu24.04.tar  ubuntu24.04.tar.gz  ubuntu24.04.tar.xz
```

> 导出为 `.xz` 归档文件时间很长，压缩算法复杂导致。

2. 导入镜像的命令 `docker load`

> 命令别名:`docker image load`

命令格式

```
docker load < 归档文件路径
# 或
docker load -i 归档文件路径
```

归档文件可以是 `.tar`、`.tar.gz`、`.tar.xz` 等格式的镜像包。

docker load 命令的常用选项

选项	说明
`-i <归档文件路径>`	从归档文件中读取数据，代替标准输入 `STDIN`。

示例

```
weimingze@mzstudio:~$ ls
mysql.tar.xz  ubuntu24.04.tar  ubuntu24.04.tar.gz  ubuntu24.04.tar.xz
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
weimingze@mzstudio:~$ sudo docker load -i ubuntu24.04.tar.xz  # 使用.tar.gz镜像包导入镜像
3abdd8a5e7a8: Loading layer
[==================================================>]  80.61MB/80.61MB
Loaded image: ubuntu:24.04
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker rmi ubuntu:24.04  # 删除镜像(后面会讲)
Untagged: ubuntu:24.04
Deleted: sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e554b1c
Deleted: sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1059d44b3
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
weimingze@mzstudio:~$ sudo docker load < ubuntu24.04.tar  # 使用标准输入导入镜像
Loaded image: ubuntu:24.04
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker rmi ubuntu:24.04
Untagged: ubuntu:24.04
Deleted: sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e554b1c
Deleted: sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1059d44b3
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
weimingze@mzstudio:~$ sudo docker load < ubuntu24.04.tar.xz
3abdd8a5e7a8: Loading layer
[==================================================>]  80.61MB/80.61MB
Loaded image: ubuntu:24.04
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
```

5. 删除本机镜像

如果有不用的镜像，可以使用 `docker rmi` 命令在本地删除镜像，以减少磁盘占用空间。

`docker rmi` 命令

命令格式

```
docker rmi <镜像ID1> <镜像ID2> ...
# 或
docker rmi <镜像名1[:标签名1]> <镜像名2[:标签名2]> ...
```

> 命令别名: `docker image rm`, `docker image remove`。

如果不给出 标签名 默认为 `latest` 标签。

docker rmi 常用选项

选项	说明
`-f` 或 `--force`	强制删除(删除已经创建容器的镜像)。

示例

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker rmi ubuntu:24.04
Untagged: ubuntu:24.04
Deleted: sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e554b1c
Deleted: sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1059d44b3
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
weimingze@mzstudio:~$ sudo docker rmi edbdd97bf78b
Untagged: mysql:latest
Deleted: sha256:edbdd97bf78b4338bbb96fa1348c8743a328b97ea3290b20adcab25bc17637de
Deleted: sha256:5ee46883cd23ccea0c130113edfca5ef2b202eb1866c6a0e868fc1f556755c75
Deleted: sha256:cc7bc92145face1c727f2014d0341f21d18247e0c7df0543e2d6ac020a6dbdbc
Deleted: sha256:a0229cd926f7a0e0afeea987e0b9d4419964e02d6d84330aee7615863687deb0
Deleted: sha256:54366fcda8e9d2e5cd0f3452adb3b2ab34c6216aa73e24889cbdcdb15788f399
Deleted: sha256:c3e04f405b930b3c39ffeeaa9583e72d1f007fc3d6a208a894ac8c793f02fa84
Deleted: sha256:8ec93f6326350a63a5858a425afcd3028ed2dcc6303784c51359b48c032916ad
Deleted: sha256:24882858314385467bf296310805d4b7229bff39c29ed9a434689adc37543291
Deleted: sha256:35caa70a7a17d9478f62ffd10c7fe27493877644fdbe82d181bc5bb93e8e0f62
Deleted: sha256:c222c78b1ef751569013c1ec5a3ad2b5a8254b3600920aaaf358e0b32872bf23
Deleted: sha256:825f4932222fe8e4ee14d896e0f324da04fb8420e5335f3ee039f32fa47e5ee3
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
weimingze@mzstudio:~$
```

删除所有镜像

组合使用 `docker images -q` 和 `docker rmi` 可以删除所有镜像。

```
sudo docker rmi $(sudo docker images -q)
```

示例

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker images -q
edbdd97bf78b
602eb6fb314b
weimingze@mzstudio:~$ sudo docker rmi $(sudo docker images -q)
Untagged: mysql:latest
Deleted: sha256:edbdd97bf78b4338bbb96fa1348c8743a328b97ea3290b20adcab25bc17637de
Deleted: sha256:5ee46883cd23ccea0c130113edfca5ef2b202eb1866c6a0e868fc1f556755c75
Deleted: sha256:cc7bc92145face1c727f2014d0341f21d18247e0c7df0543e2d6ac020a6dbdbc
Deleted: sha256:a0229cd926f7a0e0afeea987e0b9d4419964e02d6d84330aee7615863687deb0
Deleted: sha256:54366fcda8e9d2e5cd0f3452adb3b2ab34c6216aa73e24889cbdcdb15788f399
Deleted: sha256:c3e04f405b930b3c39ffeeaa9583e72d1f007fc3d6a208a894ac8c793f02fa84
Deleted: sha256:8ec93f6326350a63a5858a425afcd3028ed2dcc6303784c51359b48c032916ad
Deleted: sha256:24882858314385467bf296310805d4b7229bff39c29ed9a434689adc37543291
Deleted: sha256:35caa70a7a17d9478f62ffd10c7fe27493877644fdbe82d181bc5bb93e8e0f62
Deleted: sha256:c222c78b1ef751569013c1ec5a3ad2b5a8254b3600920aaaf358e0b32872bf23
Deleted: sha256:825f4932222fe8e4ee14d896e0f324da04fb8420e5335f3ee039f32fa47e5ee3
Untagged: ubuntu:24.04
Deleted: sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e554b1c
Deleted: sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1059d44b3
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

清理无用镜像

`docker image prune` 命令用于 清理未被使用的 Docker 镜像（悬空镜像或未被容器引用的镜像）。以减少多磁盘空间的占用。

术语

未被容器引用的镜像 是指有标签等信息，但未创建容器。 **悬空镜像** (`dangling images`) 通常是创建过程中产生的中间层镜，他没有标签，通常标签为 `<none>:<none>`，他无法创建容器。

`docker image prune` 命令

命令格式

```
docker image prune [选项]
```

如果不给出选项，默认删除 **悬空镜像** (`dangling images`)。

docker image prune 命令的常用选项

选项	说明
`-a` 或 `--all`	删除所有未被容器引用的镜像（包括可能有用的缓存镜像）。
`-f` 或 `--force`	强制删除。

示例

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker image prune -a
WARNING! This will remove all images without at least one container associated t
o them.
Are you sure you want to continue? [y/N] y
Deleted Images:
untagged: ubuntu:24.04
deleted: sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e554b1c
deleted: sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1059d44b3

Total reclaimed space: 78.1MB
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
weimingze@mzstudio:~$
```

从上述示例可以看出，这里没有悬空镜像，因此 `sudo docker image prune` 没有删除任何镜像。

在上述三个镜像中，`ubuntu` 没有创建容器。因此 `sudo docker image prune -a` 删除了 `ubuntu` 镜像，保留了 `mysql` 镜像。

查看镜像磁盘占用

使用 `docker system df` 命令可以查看当前镜像和容器占用系统磁盘的空间。

`docker system df` 命令

命令格式

```
docker system df
```

示例

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
mysql        latest    edbdd97bf78b   2 months ago   859MB
weimingze@mzstudio:~$ sudo docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          1         1         859.4MB   0B (0%)
Containers      1         1         6B        0B (0%)
Local Volumes   4         0         627.3MB   627.3MB (100%)
Build Cache     0         0         0B        0B
```

可见当前有个镜像(Image) 占用 `859.4MB`；容器(Containers) 1 个，正在运行(ACTIVE:1)，占用磁盘空间 6B；本地卷占用 `627.3MB`；缓存没有占用磁盘空间。

6. 镜像标签管理

使用 镜像标签管理命令 `docker tag` 可以为镜像添加名称和标签。

使用 docker tag 命令看似是创建一个新的镜像，实际只是多添加了一个标签，因为镜像都是只读的，因此镜像不需要复制。

`docker tag` 命令

命令格式

```
docker tag 原镜像名[:原标签] 新镜像名[:新标签]
```

> 命令别名: `docker image tag`

示例

将镜像 `ubuntu:24.04` 添加标签为 `myubuntu:latest` 和 `myubuntu:release`。

```
weimingze@mzstudio:~$ sudo docker images;
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker tag ubuntu:24.04 myubuntu
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
myubuntu     latest    602eb6fb314b   2 months ago   78.1MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker tag ubuntu:24.04 myubuntu:release
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
myubuntu     latest    602eb6fb314b   2 months ago   78.1MB
myubuntu     release   602eb6fb314b   2 months ago   78.1MB
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
```

添加标签的镜像看似是复制了镜像 `ubuntu:24.04` 但此时实际并没有复制，使用 `docker system df` 命令查看你会发现重定义标签的镜像并没有占用空间。

7. 查看镜像详情

使用 docker inspect 命令可以查看一个镜像的信息，包括镜像的 CPU 架构 `Architecture` 和镜像的宿主操作系统 `Os` 等信息。

信息默认是以 `JSON` 格式显示，也可以通过 `-f` 选项重新定义格式。

`docker inspect` 命令

命令格式

```
docker inspect 镜像ID
# 或
docker inspect 镜像名称[:标签名]
```

示例:

显示 `ubuntu:24.04` 镜像的信息

```
weimingze@mzstudio:~$ sudo docker inspect ubuntu:24.04
[
    {
        "Id": "sha256:602eb6fb314b5fafad376a32ab55194e535e533dec6552f82b70d7ac0e
```

```
554b1c",
        "RepoTags": [
            "myubuntu:latest",
            "myubuntu:release",
            "ubuntu:24.04"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "",
        "Created": "2025-04-08T10:43:15.147460451Z",
        "DockerVersion": "24.0.7",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin"
            ],
            "Cmd": [
                "/bin/bash"
            ],
            "Image": "sha256:9e8e73ef7ecf967799835237964467 4e54a6b3104ae7da6d120
4d83a4118826f",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {
                "org.opencontainers.image.ref.name": "ubuntu",
                "org.opencontainers.image.version": "24.04"
            }
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 78102235,
        "GraphDriver": {
            "Data": {
                "MergedDir": "/var/lib/docker/overlay2/
cd76bb5aae922326889b8ee6807cb7d32f5fe329f88c6d0f217dbf69cdc8bdee/merged",
                "UpperDir": "/var/lib/docker/overlay2/
cd76bb5aae922326889b8ee6807cb7d32f5fe329f88c6d0f217dbf69cdc8bdee/diff",
                "WorkDir": "/var/lib/docker/overlay2/
cd76bb5aae922326889b8ee6807cb7d32f5fe329f88c6d0f217dbf69cdc8bdee/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:3abdd8a5e7a8909e1509f1d36dcc8b85a0f95c68a69e6d86c6e9e3c1
```



```
059d44b3"
            ]
        },
        "Metadata": {
            "LastTagTime": "2025-06-19T18:21:27.240834391+08:00"
        }
    }
]
weimingze@mzstudio:~$
```

上述信息中的内容解析：

- `Id` 本镜像的ID
- `RepoTags` 本镜像的名字和标签。我们上节课为该镜像添加了三个标签。
- `Created` 镜像创建的时间。
- `Config` 镜像的配置
  - `Hostname` 镜像内部的主机名。
  - `User` 镜像内部的用户名。
  - `Env` 容器运行时的环境变量
  - `Cmd` 容器主进程的运行时命令。
  - `Volumes` 容器内卷的位置。
  - `WorkingDir` 容器内的当前工作路径。
- `Architecture` CPU 架构
- `Os` 宿主操作系统
- `Size` 容器大小
- `RootFS` 根文件系统的类型和签名信息。

8. 查看镜像历史

`docker history` 命令用于查看 Docker 镜像的创建历史，显示该镜像的每一层（Layer）的详细信息。

这个命令能够提供如下信息：

- 每层的 创建命令（Dockerfile 中的指令）。
- 每层的 大小。
- 创建时间、作者等元数据。

通过该命令分析镜像的组成，可以排查层体积过大问题或验证创建过程是否符合预期要求。

命令格式


```
docker history 镜像ID
# 或
docker history 镜像名称[:标签名称]
```

示例

```
weimingze@mzstudio:~$ sudo docker history ubuntu:24.04
IMAGE          CREATED        CREATED BY                                      SI
ZE      COMMENT
602eb6fb314b   2 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      2 months ago   /bin/sh -c #(nop) ADD file:1d7c45546e94b90e9…
78.1MB
<missing>      2 months ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago   /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      2 months ago   /bin/sh -c #(nop)  ARG RELEASE                  0B
```

信息显示 `ubuntu:24.04` 这个镜像一共有 6 层(每次修改会多出一层)，最后一层是 `602eb6fb314b`，并将启动命令改为 `/bin/bash`。

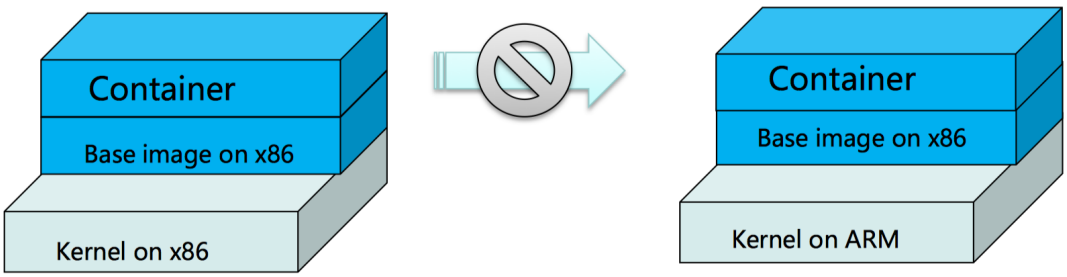
第四章、Docker 容器的操作

前面我们讲过，在 Docker 中，容器（Container）是一个轻量级、可执行的独立软件包，它包含了运行某个应用所需的所有内容：根文件系、网络、名字空间、运行时环境、系统工具、运行时库和配置等信息。

Docker 的容器有如下特性：

1. 基于镜像运行
   - 容器是从Docker镜像创建的实例。镜像类似于模板，而容器是模板的运行实例（类似于面相对象中的 "类" 和 "对象" 的关系）。
   - 例如：你可以从使用 mysql 镜像运行多个独立的 mysql 容器，每个mysql 后台服务用于不同的业务。
2. 隔离性
   - 每个容器拥有独立的文件系统、网络、进程空间（通过Linux内核的cgroups和namespaces实现）。
   - 容器之间互不干扰，但可以通过配置进行通信。
3. 轻量级
   - 容器共享宿主机的操作系统内核，无需像虚拟机（VM）那样模拟完整操作系统，因此启动更快、资源占用更少。
4. 临时性（可销毁）
   - 容器默认是无状态的，停止后数据会丢失（除非使用卷（Volumes）持久化存储）。
5. 可移植性
   - 容器可以在相同宿主操作系统的 Docker 环境中制作，并能在相同类型的宿主操作系统上运行（有限的可移植性）。
6. 不可跨硬件平台运行
   - 在基于ARM的平台中创建的镜像均为基于 ARM 平台的，无法在 x86 平台上运行。

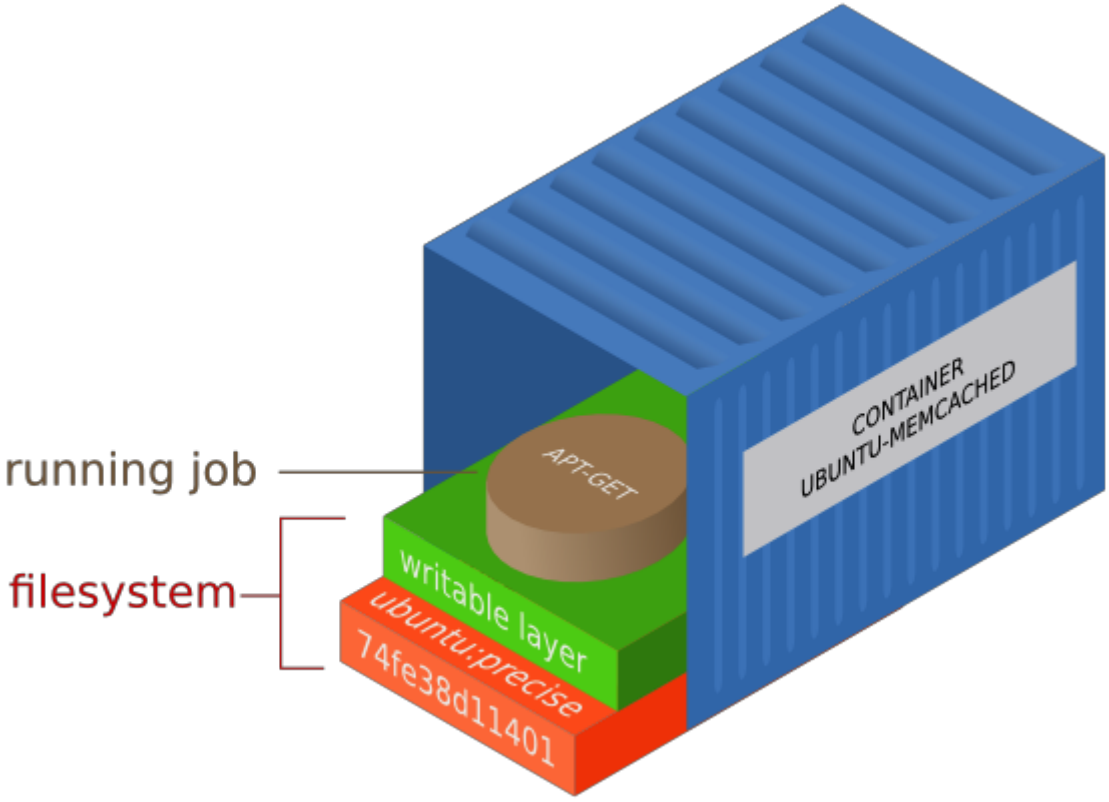
   - 跨平台的容器无法运行，会出现格式错误，如下图所示：



Docker 容器的结构模型

一个容器通常有一个或多个只读的镜像层（如下图的 `74fe38d11401`）和一个可写层（Writeable layer）组成，可写层用于容器运行时存放临时数据。

如下图所示:



上图最上层的 `running job` 就是 **容器主程序**。

一个容器一定要有一个 **容器主程序**，并且这个 **容器主程序** 一旦推向后台或者退出，则整个容器运行结束（重要）。

1. 创建并运行 docker 容器

使用 `docker run` 命令可以创建并运行一个容器。

docker run 命令

命令格式

```
docker run [选项] 镜像名[:标签]或镜像ID [命令] [参数...]
```

> 命令别名: `docker container run`

命令格式说明

- `docker run` 一个容器运行必须依赖一个镜像，我们可以用 `镜像名[:标签]` 或 `镜像ID`的形式给出镜像.
- `[命令] [参数...]` 是容器内的命令和参数，如果不给定这个参数，则用镜像内的 `CMD` 设置给定。

常用选项

选项	说明
`-d`	在后台运行容器（detached 模式），打印容器ID
`--name`	指定容器名称
`-p <宿主机端口>:<容器端口>`	端口映射（如 -p 80:8080）
`-v <宿主机路径>:<容器路径>`	挂载卷，将宿主机文件或文件夹挂载在容器内某个路径下）。
`-e 环境变量名=值`	设置容器内环境变量（如 -e MYSQL_ROOT_PASSWORD=123456）
`--restart=重启模式`	退出后重启策略（如 --restart=always），默认是 `no` 不重启，`always` 退出就重启，`unless-stopped` 不会恢复手动停止的容器，`on-failure[:max-retries]` 容器非正常退出时重启（可指定最大重试次数）。
`--rm`	当容器退出后自动删除容器。
`--network`	指定网络连接方式（如 `--network=host` 直接暴露主机端口，`bridge` 容器间通信(默认），`container:<容器名或ID>`共享其他容器的网络。`none` 容器不配置任何网络
`-i`	交互式终端，保持标准输入 `STDIN` 打开。
`-t`	分配一个伪终端（通常和 `-i` 选项一起使用）。
`-w`	指定容器内部的工作路径
`-ip <IP地址>`	设置容器运行时的 IP 地址,如: `-ip 127.17.0.100`
`-cpus <数字>`	设置容器运行的占用 CPU 的数量。
`-m <字节数>`	设置容器的内存限制
`-h <主机名>`	设置容器的主机名

> 使用 `sudo docker run --help` 可以查看全部选项。

示例

```
weimingze@mzstudio:~$ sudo docker run \
    --name mysql_server \
    -p 3306:3306 \
    -e MYSQL_ROOT_PASSWORD=weimingze.com \
    -v /home/weimingze/mysql_vol/data:/var/lib/mysql \
    -v /home/weimingze/mysql_vol/logs:/var/log/mysql \
    --restart=unless-stopped \
    -d \
    mysql
4d68e66019f347a444aed488acfc8aec7ebf3c9210f6ccb0467a03186eceb483
weimingze@mzstudio:~$
```

> 反斜杠 `\` 是 Linux Shell 的折行符，当命令行太长时，可以在一行的末尾写入 反斜杠 `\`，然后在下一行可以继续输入此命令的选项和参数而不会执行当前命令。

下面我们来详细解读上述 `docker run` 的选项:

1. `--name mysql_server` 是为此运行的容器取一个名字 `mysql_server`，便于查找、停止此容器等操作。
2. `-p 3306:3306`，将容器内部的 3306 端口号映射到 宿主机的 3306 端口号。也就是 宿主机通过 3306 端口号和容器内的 3306 对应。不映射，我们不能从外部访问服务器。
3. `-e MYSQL_ROOT_PASSWORD=weimingze.com` 通过环境变量 `MYSQL_ROOT_PASSWORD` 来设定 容器内 MySQL 服务器的 `root` 密码是 `weimingze.com`, 这个环境变量的名称是制作镜像的作者来定义的。
4. `-v /home/weimingze/mysql_vol/data:/var/lib/mysql` 将 `MySQL` 容器内部的 `/var/lib/mysql` 映射成宿主机的 `/home/weimingze/mysql_vol/data` 文件夹。`/var/lib/mysql` 是 MySQL 数据保存数据的默认路径
5. `-v /home/weimingze/mysql_vol/logs:/var/log/mysql` 将 MySQL 保存日志的路径 `/var/log/mysql` 映射成宿主机的路径，便于查看和保存。
6. `--restart=unless-stopped` 如果容器非正常退出则重新启动容器。

7. `-d` 将 正在运行的容器推向后台运行，脱离终端。去掉此选项，终端会连接 **容器主进程** 并能看到容器主进程的输出结果。使用 `Control + c` 可以终止容器运行。

8. 终端输出 `4d68e66019f347a444aed488acfc8aec7ebf3c9210f6ccb0467a03186eceb483` 是容器的 ID。

宿主机 `/home/weimingze/mysql_vol` 的内容如下：

```
weimingze@mzstudio:~$ tree mysql_vol/
mysql_vol/
├── data
│   ├── #ib_16384_0.dblwr
│   ├── #ib_16384_1.dblwr
│   ├── #innodb_redo  [error opening dir]
│   ├── #innodb_temp  [error opening dir]
│   ├── auto.cnf
│   ├── binlog.000001
│   ├── binlog.000002
│   ├── binlog.000003
│   ├── binlog.index
│   ├── ca-key.pem
│   ├── ca.pem
│   ├── client-cert.pem
│   ├── client-key.pem
│   ├── ib_buffer_pool
│   ├── ibdata1
│   ├── ibtmp1
│   ├── mysql  [error opening dir]
│   ├── mysql.ibd
│   ├── mysql.sock -> /var/run/mysqld/mysqld.sock
│   ├── mysql_upgrade_history
│   ├── performance_schema  [error opening dir]
│   ├── private_key.pem
│   ├── public_key.pem
│   ├── server-cert.pem
│   ├── server-key.pem
│   ├── sys  [error opening dir]
│   ├── undo_001
│   └── undo_002
└── logs

8 directories, 23 files
```

可见容器内的 `mysql` 启动后将运行的数据库内容映射到了宿主机的 `/home/weimingze/mysql_vol` 文件夹下。这样可以保证数据的安全，一但容器损坏，数据还在，还可以恢复。

2. 容器信息查询

使用 `docker ps` 命令可以查看容器的列表，包括正在运行的容器和处于停止状态的容器

docker ps 命令

命令格式

```
docker ps [选项]
```

> 命令别名: `docker container ls`, `docker container list`, `docker container ps`

常用选项

选项	说明
`-a`	列出所有的容器（默认只显示正在运行的容器）。
`-n <整数n>`	列出最后创建的 n 个容器。
`-l`	列出最后创建的容器（包括所有的状态）。
`--no-trunc`	显示完整信息（不截断信息）。
`-q`	只列出容器的 ID。
`-s`	列出容器总文件的大小。
`--format "格式字符串"`	显示默认的列，如 `--format "table {{.ID}}\t{{.Status}}\t{{.Names}}"` 则只显示 `ID`、 `STATUS` 和 `NAMES` 三列。

示例

```
weimingze@mzstudio:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
   PORTS                                                  NAMES
a9cffc0d7700   mysql     "docker-entrypoint.s…"   23 minutes ago   Up 23
minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql_server
weimingze@mzstudio:~$ sudo docker ps  -a
CONTAINER ID   IMAGE          COMMAND                  CREATED             STAT
US                         PORTS
   NAMES
1c5d86585cf7   ubuntu:24.04   "/bin/bash"              About a minute ago   Exit
ed (0) About a minute ago
   myubuntu_server
a9cffc0d7700   mysql          "docker-entrypoint.s…"   23 minutes ago       Up 2
3 minutes                  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/
tcp   mysql_server
weimingze@mzstudio:~$ sudo docker ps  -a -q
1c5d86585cf7
a9cffc0d7700
weimingze@mzstudio:~$ sudo docker ps  -a -s
CONTAINER ID   IMAGE          COMMAND                  CREATED             STAT
US                         PORTS
   NAMES             SIZE
1c5d86585cf7   ubuntu:24.04   "/bin/bash"              About a minute ago   Exit
ed (0) About a minute ago
   myubuntu_server   0B (virtual 78.1MB)
a9cffc0d7700   mysql          "docker-entrypoint.s…"   23 minutes ago       Up 2
3 minutes                  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/
```

```
tcp   mysql_server      6B (virtual 859MB)
weimingze@mzstudio:~$
```

容器 `myubuntu_server` 是停止运行的容器，因此只有在 `docker ps -a` 时能够列出。

字段说明

使用 `docker ps` 命令时，每一列的信息列表如下：

- `CONTAINER ID` 容器ID，ID字符串的前 12 个字符。
- `IMAGE` 创建容器的镜像。
- `COMMAND` 容器主进程的启动命令。
- `CREATED` 创建时间。
- `STATUS` 容器的当前状态:
  - `Up` 运行状态
  - `Exited` 停止状态
  - `Pause` 暂停状态
  - `Up` 运行状态
  - `Created` 容器已经创建，但是没有启动（`docker create` 后）。
  - `Dead` 容器因错误无法恢复（需手动清理）。
- `PORTS` 容器端口映射的列表
- `NAMES` 容器的名称，在启动时用 `--name` 设置。

3. 容器的运行管理

容器的管理分为：启动（`start`）、停止（`stop`）、重启动（`restart`）、暂停（`pause`）、恢复（`unpause`）、删除（`rm`）操作。

容器功能	命令格式	说明
启动	`docker start [选项] 容器名或容器ID`	启动处于停止状态的容器。
停止	`docker stop [选项] 容器名或容器ID`	停止正在运行的容器。
重启动	`docker restart [选项] 容器名或容器ID`	停止容器的运行重新启动。
暂停	`docker pause [选项] 容器名或容器ID`	暂停容器的运行。
恢复	`docker unpause [选项] 容器名或容器ID`	恢复已暂停容器的运行。
删除	`docker rm [选项] 容器名或容器ID`	删除容器。

停止和暂停状态的区别

- 停止状态：是正常终止容器内运行的进程，释放资源，并将容器状态标记为 Exited。
  - 特点：不占用CPU和内存等资源。容器内所有进程终止运行。
- 暂停状态：是冻结容器内运行的进程，挂起所有进程，不终止进程。
  - 特点：不释放CPU和内存等资源。可以快速恢复运行。

示例

```
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS          NAMES
a9cffc0d7700   mysql     Up 17 minutes   mysql_server
weimingze@mzstudio:~$ sudo docker stop mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS    NAMES
weimingze@mzstudio:~$ sudo docker ps -a --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE          STATUS                      NAMES
a9cffc0d7700   mysql          Exited (0) 9 seconds ago    mysql_server

weimingze@mzstudio:~$ sudo docker start mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS         NAMES
a9cffc0d7700   mysql     Up 3 seconds   mysql_server
weimingze@mzstudio:~$ sudo docker pause mysql_server
```

```
mysql_server
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS                   NAMES
a9cffc0d7700   mysql     Up 30 seconds (Paused)   mysql_server
weimingze@mzstudio:~$ sudo docker unpause mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS          NAMES
a9cffc0d7700   mysql     Up 46 seconds   mysql_server
weimingze@mzstudio:~$ sudo docker stop mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS    NAMES
weimingze@mzstudio:~$ sudo docker ps -a --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE          STATUS                      NAMES
a9cffc0d7700   mysql          Exited (0) 16 seconds ago   mysql_server
weimingze@mzstudio:~$ sudo docker rm mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps -a --format "table {{.ID}}\t{{.Image}}
\t{{.Status}}\t{{.Names}}"
CONTAINER ID   IMAGE     STATUS    NAMES
```

> 上述 `docker ps` 命令使用 `--format "table {{.ID}}\t{{.Image}}\t{{.Status}}\t{{.Names}}"` 选项是只显示 `CONTAINER ID`, `IMAGE`, `STATUS`, `NAMES` 这四列。

4. 创建 Docker 容器

使用 `docker create` 命令可以创建容器，但是不启动此容器。手续可以通过 `docker start` 启动该容器。

docker create 命令

命令格式

```
docker create [选项] 镜像名[:标签]或镜像ID [命令] [参数...]
```

`docker create` 命令的选项和 `docker run` 命令的选项基本一致，但 `docker create` 没有 `-d` 选项，因为此容器并不运行。

示例

使用 `docker create` 创建 mysql 程序，然后使用 `docker start` 命令运行此容器。

```
weimingze@mzstudio:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
weimingze@mzstudio:~$ sudo docker create \
    --name mysql_server \
    -p 3306:3306 \
    -e MYSQL_ROOT_PASSWORD=weimingze.com \
    -v /home/weimingze/mysql_vol/data:/var/lib/mysql \
    -v /home/weimingze/mysql_vol/logs:/var/log/mysql \
    --restart=on-failure:2 \
    mysql
04bc8f62cabb22ac69d1976079ed3358acf589ab97e5b05654723249063dde8f
weimingze@mzstudio:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS    PORT
S     NAMES
04bc8f62cabb   mysql     "docker-entrypoint.s…"   4 seconds ago   Created
      mysql_server
weimingze@mzstudio:~$ sudo docker start mysql_server
mysql_server
weimingze@mzstudio:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
  PORTS                                                 NAMES
04bc8f62cabb   mysql     "docker-entrypoint.s…"   31 seconds ago   Up 4 seconds
  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp   mysql_server
weimingze@mzstudio:~$
```

当然使用 `docker create` 创建 mysql_server 容器时，容器的状态是 `Created`，然后使用 `docker start` 运行容器后，容器的状态变为 `Up`（运行状态）。

5. Docker 容器相关的子命令

使用 `docker container --help` 可以查看容器相关的所有子命令。再进一步使用 `--help` 选项可以查看 各个子命令人用法和详细描述。如查看 `docker container ls --help` 就可以查看 `docker ps` 命令(`docker container ls`的别名)的用法了。

Docker 容器相关的全部子命令如下：

```
weimingze@mzstudio:~$ sudo docker container --help

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a runnin
g container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
```

```
  diff        Inspect changes to files or directories on a container's filesyste
m
  exec        Execute a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Create and run a new container from an image
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit
codes

Run 'docker container COMMAND --help' for more information on a command.
```

第五章、容器交互

在使用容器时，尤其是在创建镜像时，经常需要查看容器内部运行时环境和运行过程。这时我们可以使用 `docker attach` 和 `docker exec` 等命令产来查看容器的运行环境。

1. docker attach 命令

`docker attach` 命令用于将本地终端（标准输入、标准输出和标准错误输出）附加到一个正在运行的容器上，允许用户与容器主进程进行交互。它通常用于调试或查看容器的实时输出。

命令格式

```
docker attach [选项] 容器名或容器ID
```

> 命令别名: `docker container attach`

常用选项

选项	说明
`--no-stdin`	不连接标准输入，只连接标准输出和标准错误输出。
`--sig-proxy=<true/false>`	代理收到的信号到容器进程（默认 true，如 `Control + c` 会终止容器）。

示例

连接 mysql_server 的终端，在使用 `Control + c` 是断开连接且不终止容器主进程。

```
weimingze@mzstudio:~$ sudo docker attach  --sig-proxy=false mysql_server
^C^C^C
got 3 SIGTERM/SIGINTs, forcefully exiting
weimingze@mzstudio:~$
```

连续按三下 `Control + c` 断开连接。

2. docker exec 命令

`docker exec` 命令用于在正在运行的 Docker 容器中执行命令（容器内的命令）。

命令格式

```
docker exec [选项] 容器名或容器ID 命令 [命令选线和参数...]
```

> 命令别名: `docker container exec`

常用选项

选项	说明
`-d`	在后台运行命令。
`-e <环境变量名=环境变量值>`	设置环境变量。
`-i`	保持 STDIN 打开，即使没有附加。
`-t`	分配一个伪终端。
`-u <容器用户名>`	指定执行命令的用户名或UID。
`-w <容器内路径>`	指定命令的工作路径。

示例

进入到正在运行的容器 `mysql_server` 中并执行内部 的 `bash` 来查看 MySQL 的配置文件 `/etc/my.cnf`

```
weimingze@mzstudio:~$ sudo docker exec -it mysql_server /bin/bash
bash-5.1# ls
afs  bin  boot  dev  docker-entrypoint-initdb.d  etc  home  lib  lib64  media  m
nt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
bash-5.1# cat /etc/my.cnf
# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/9.3/en/server-configuration-defaults.html

[mysqld]
... # 以下略
bash-5.1#
```

`/bin/bash` 是容器内的 `Shell`，我们还可以直接运行程序内的其他命令。

`bash-5.1#` 是容器内 bash 的提示符，这个提示符可以通过修改 `PS1` 环境变量来重新设置。如:

```
weimingze@mzstudio:~$ sudo docker exec -it mysql_server /bin/bash
bash-5.1# PS1="\u@\h# "
root@b0f4852e19af# PS1="\s-\v # "
bash-5.1 # exit
exit
weimingze@mzstudio:~$
```

PS1 中的含义

- `\u` 用户名。
- `\h` 主机名（这里是容器ID）。
- `\w` 当前工作路径。
- `-s` Shell 的名称。
- `-v` Shell 的版本。

3. 查看容器运行日志

使用 `docker logs` 命令可以查看容器的日志输出（容器主进程的标准输出和标准错误输出）。

命令格式

```
docker logs [选项] 容器名或容器ID
```

> 命令别名: `docker container logs`

常用选项

选项	说明
`-t`	显示时间戳信息。
`-n <整数n>`	显示末尾的 n 行信息。
`-f`	实时跟踪日志信息，不断开连接。

示例

查看 `mysql_server` 的日志信息的最后 5 行。

```
weimingze@mzstudio:~$ sudo docker logs  -n 5  mysql_server
2025-06-20T12:06:44.806865Z 0 [Warning] [MY-010068] [Server] CA certificate ca.p
em is self signed.
2025-06-20T12:06:44.807161Z 0 [System] [MY-013602] [Server] Channel mysql_main c
onfigured to support TLS. Encrypted connections are now supported for this chann
el.
2025-06-20T12:06:44.809979Z 0 [Warning] [MY-011810] [Server] Insecure configurat
ion for --pid-file: Location '/var/run/mysqld' in the path is accessible to all
OS users. Consider choosing a different directory.
2025-06-20T12:06:44.856067Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: re
ady for connections. Version: '9.3.0'  socket: '/var/run/mysqld/mysqld.sock'  po
rt: 3306  MySQL Community Server - GPL.
2025-06-20T12:06:44.856406Z 0 [System] [MY-011323] [Server] X Plugin ready for c
onnections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock
```

4. 查看容器信息

使用 `docker inspect` 命令可以查看镜像和容器的信息，这个命令会显示镜像的信息，还额外显示容器的运行信息，如：配置、网络等信息。

命令格式

```
docker inspect 容器名或容器ID
```

常用选项

选项	说明
`-f <格式字符串>`	设置输出格式，默认是json。
`-n <整数n>`	显示末尾的 n 行信息。
`-f`	实时跟踪日志信息，不断开连接。

示例

```
weimingze@mzstudio:~$ sudo docker inspect mysql_server;
[
    {
        "Id":
"b0f4852e19aff9662825209c42a3b1788f392d3f342a23441464fc8474baf6b8",
        "Created": "2025-06-20T12:06:40.548207295Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mysqld"
```

```
        ],
        ... # 此处省略部分内容。
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID": "13e62a4c00d0043804ff03ca83fd078235fa486297749e8c7f04ab
38494da465",
            "SandboxKey": "/var/run/docker/netns/13e62a4c00d0",
            "Ports": {
                "3306/tcp": [
                    {
                        "HostIp": "0.0.0.0",
                        "HostPort": "3306"
                    },
                    {
                        "HostIp": "::",
                        "HostPort": "3306"
                    }
                ],
                "33060/tcp": null
            },
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "6ed48799ba3ff35df917df0b2bd43805bf8d339932bdcb93eb849
ca0e82792a9",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null,
                    "NetworkID": "5ac38be5fbc6ac119251ef8ddf8abe494bcf9966915e51
dfe77082a930c60d13",
                    "EndpointID": "6ed48799ba3ff35df917df0b2bd43805bf8d339932bdc
b93eb849ca0e82792a9",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "DNSNames": null
                }
            }
        }
    }
```

```
]
weimingze@mzstudio:~$
```

`NetworkSettings` 部分内容是我们调试网络时常用到的信息。

5. 查看容器内运行的进程

使用 `docker top` 可以查看容器内运行的进程的个数和信息。

命令格式

```
docker top 容器名或容器ID [ps命令的选项]
```

> 命令别名：`docker container top`

示例

```
weimingze@mzstudio:~$ sudo docker top mysql_server
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
dnsmasq             6768                6744                0
20:06               ?                   00:00:31            mysqld
weimingze@mzstudio:~$ sudo docker top mysql_server aux
USER                PID                 %CPU                %MEM
VSZ                 RSS                 TTY                 STAT
START               TIME                COMMAND
dnsmasq             6768                0.9                 11.9
1852368             471788              ?                   Ssl
20:06               0:31                mysqld
weimingze@mzstudio:~$
```

6. 容器资源监控命令

`docker stats` 命令可以实时监控容器的资源使用情况。包括：

- CPU使用率 `CPU %`。
- 内存使用量和限制 `MEM USAGE / LIMIT`。
- 内存占比 `MEM %`。
- 网络 I/O 流量 `NET I/O`。
- 磁盘 I/O 流量 `BLOCK I/O`。
- 进程数 `PIDS`。

docker stats 命令

命令格式

```
docker stats [选项] 容器名或容器ID
```

> 命令别名：`docker container stats`

`Control + c` 退出监控界面

示例

查看 `mysql_server` 容器的情况。

```
weimingze@mzstudio:~$ sudo docker stats mysql_server
CONTAINER ID   NAME           CPU %     MEM USAGE / LIMIT     MEM %     NET I/O
      BLOCK I/O         PIDS
b0f4852e19af   mysql_server   0.86%     446.1MiB / 3.776GiB   11.54%    4.
81kB / 0B   49.4MB / 17.8MB   35
```

7. 容器内文件复制命令

`docker cp` 命令用于在 Docker 容器 和 本地宿主机 之间复制文件或目录，双向传输支持。

docker cp 命令

命令格式

```
# 容器 到 宿主机复制
docker cp [选项] 容器名或容器ID:容器内源路径 宿主机目标路径或-

# 宿主机 到 容器复制
docker cp [选项] 宿主机源路径或- 容器名或容器ID:容器内目标路径
```

> 命令别名：`docker container cp`

使用 减号 `-` 代替宿主机源路径，则从标准输入中读取归档的 `tar` 包并解压缩到目标容器。

使用 减号 `-` 代替宿主机目标路径，则从标准输出中返回 `tar` 包的归档数据流，可用管道等进行保存和压缩。

常用选项

选项	说明
`-a`	归档模式复制（复制所有 UID/GID 信息）。
`-L`	复制符号链接指向的实际文件。

示例

复制 `mysql_server` 容器内的配置文件 `/etc/my.cnf` 到当前路径。

```
weimingze@mzstudio:~$ sudo docker cp mysql_server:/etc/my.cnf .
Successfully copied 2.56kB to /home/weimingze/.
weimingze@mzstudio:~$ ls
hello.py   my.cnf  myproject.tar.xz
weimingze@mzstudio:~$
```

复制当前路径下的 `hello.py` 文件到 容器 `mysql_server`的 `/root/`文件夹下

```
weimingze@mzstudio:~$ sudo docker cp hello.py mysql_server:/root/
Successfully copied 2.05kB to mysql_server:/root/
weimingze@mzstudio:~$
```

复制归档文件 `myproject.tar.xz` 并解压缩到容器 `mysql_server` 的 `/root/` 文件下。

```
weimingze@mzstudio:~$ cat myproject.tar.xz | sudo docker cp - mysql_server:/
root/
Successfully copied 0B to mysql_server:/root/
weimingze@mzstudio:~$ sudo docker exec -it mysql_server /bin/bash
bash-5.1# ls /root/
hello.py  myproject
bash-5.1#
```

8. 容器文件系统比较命令

`docker diff` 命令用于查看容器文件系统的变化(与原始镜像相比)。

命令格式

```
docker diff 容器名或容器ID
```

> 命令别名：`docker container diff`

示例

查看 `mysql_server` 容器内的文件系统都有哪些变化。

```
weimingze@mzstudio:~$ sudo docker diff mysql_server
C /root
A /root/.bash_history
A /root/hello.py
A /root/myproject
A /root/myproject/hello.py
C /var
C /var/log
A /var/log/mysql
```

`/root/hello.py` 等是上一节我们复制的文件。 `/var/log/mysql` 是 mysql 容器运行时产生的文件。

第六章、创建自定义镜像

这一章我们来学习如何自己制作 Docker 的镜像。

创建自定义镜像通常有两种方法:

1. 用 docker commit 命令制作镜像
2. 用 Dockerfile 文件制作镜像

如下图所示

也就是说我们可以使用 `Dockerfile` 创建一个镜像，也可以使用一个原始镜像来创建一个容器，在容器中安装运行时软件，然后再使用 `docker commit` 命令将容器制作成镜像。

无论从哪种方式来制作镜像，我们需要在容器建立一个同真实的物理机器一样的运行环境来验证应用是否可用。只用实际应用在真实环境中运行起来，才能保证制作的容器能够正常运行。

1. Http 服务器搭建

下面我们使用 Python 中的 `http.server` 模块在搭建一个静态网页的 Web 服务器。然后我们在用同样的方法和步骤来创建 Docker 容器。

准备资料

网站内容的压缩包 `mywebsite.tar.gz` （需要从百度网盘下载或手动生成）

上述资料可以从百度网盘下载：

- 百度网盘链接地址：https://pan.baidu.com/s/1U2CUtna3oq9NFkh8ewMzdg 提取码: 9gy6

`mywebsite.tar.gz` 网页的压缩包位于 `docker教程资料/` 文件夹下的 `mywebsite.tar.gz`。

解压缩此压缩包到当前文件夹下，如:

```
weimingze@mzstudio:~$ tar -xzvf mywebsite.tar.gz
mywebsite/
mywebsite/index.html
mywebsite/second_page.html
weimingze@mzstudio:~$ tree mywebsite
mywebsite
├── index.html
└── second_page.html

1 directory, 2 files
```

手动制作网站内容

如果你无法下载此资料，你也可以手动制作这个压缩包。方法如下:

1. 在 当前路径创建文件夹 `mywebsite`, 并在 mywebsite 下创建两个文件 `index.html` 和 `second_page.html`。

```
weimingze@mzstudio:~$ mkdir mywebsite
weimingze@mzstudio:~$ touch mywebsite/index.html
weimingze@mzstudio:~$ touch mywebsite/second_page.html
weimingze@mzstudio:~$ tree mywebsite
mywebsite
├── index.html
└── second_page.html

1 directory, 2 files
weimingze@mzstudio:~$
```

1. 编辑 `mywebsite/index.html` 文件，写入如下内容:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Docker 教程 - 明择工作室</title>
</head>
<body>
    <h1>Docker 测试主页</h1>
    <a href="./second_page.html">进入下一页</a>
</body>
</html>
```

1. 编辑 `mywebsite/second_page.html` 文件，写入如下内容:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Docker 教程 - 明择工作室</title>
</head>
<body>
    <h1>Docker 测试第二页</h1>
    <a href="./">返回主页</a>
</body>
</html>
```

1. 将 `mywebsite` 文件夹压缩成 `mywebsite.tar.gz` 的归档包。供下节课使用

```
weimingze@mzstudio:~$ tar -czvf mywebsite.tar.gz mywebsite/
mywebsite/
mywebsite/index.html
mywebsite/second_page.html
weimingze@mzstudio:~$
```

使用 Python 启动 Web 服务器

在 Ubuntu 24.04 的终端中运行如下命令，启动 Web 服务器。

```
python3 -m http.server --directory ./mywebsite 8000
```

- `-m http.server` 是让 Python 执行 `http.server` 模块。
- `./mywebsite` 是网站网页的根文件夹。
- `8000` 是访问网站开放的端口号（默认就是 `8000`），你也可以改成浏览器默认的 `80` 端口号（需要 `root` 权限）。

> 在 MacOS 和 Linux 下使用 `python3` 命令，在 Windows 下需要使用 `python` 命令。

在 MacOS 和 Windows 下需要安装 Python 3.6 及以上版本的 Python 解释执行器。

安装方法详见:https://weimingze.com/pythonbase/mac_install_python.html

运行结果

```
weimingze@mzstudio:~$ python3 -m http.server --directory ./mywebsite 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

在本机打开浏览器，输入网址 http://0.0.0.0:8000/ 就可以访问这个网站了。

如果在局域网的其他电脑或手机访问此网站，需要使用网站 `http://<启动Web服务器的IP地址>:8000/` 来访问此机器，效果如下:

这样，我们完成了在 Ubuntu 24.04 上搭建一个静态网页的 Web 服务器。

下节课我们这个 Web 服务器封装到 Docker 容器中。

2. docker commit 制作镜像

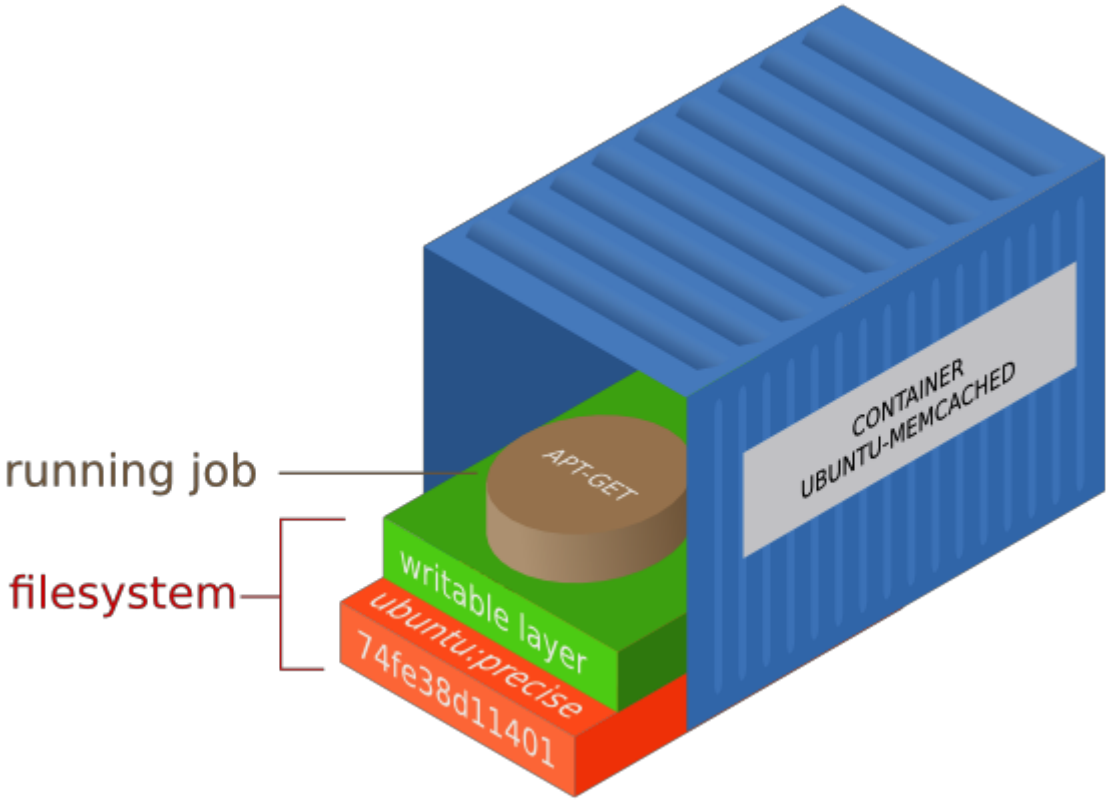
docker commit 命令

`docker commit` 命令是 Docker 中一个用于将运行中的容器保存为新镜像的命令。它的主要作用是将容器的当前状态（包括文件系统变更和配置）捕获为一个新的镜像。

本小节我们讲述如何使用 `docker commit` 命令来制作 Docker 镜像。

一个容器通常有一个或多个只读的镜像层（如下图的 `74fe38d11401`）和一个可写层（Writeable layer）组成，可写层用于容器运行时存放临时数据。

如下图所示:



使用 `docker commit` 制作镜像就是将当前容器的可写层（Writable Layer）转换为一个新的只读层，并基于原镜像的其他只读层生成一个新的镜像。这个转换为只读层的可写层，会作为新镜像的最上层。

制作步骤:

1. 创建并运行一个容器。
2. 向容器内添加应用程序和运行环境。
3. 用 docker commit 命令将容器打包为镜像。
4. 将镜像发布到其他 Linux 平台进行测试。

本节目标

我们将上节课用 Python 中的 `http.server` 模块搭建的 Web 服务器封装成 Docker 容器。

2.1 创建并运行一个容器

准备资料

1. `ubuntu:24.04` 官方镜像（可以使用 `sudoc docker pull ubuntu:24.04` 从官方下载，也可以从百度网盘下载）。
2. `mywebsite.tar.gz` 网页的压缩包（需要从百度网盘下载或手动生成-参照上节课制作）

上述资料百度网盘下载地址：

- 链接地址：https://pan.baidu.com/s/1U2CUtna3oq9NFkh8ewMzdg 提取码: 9gy6
- `ubuntu:24.04` 官方镜像位于 `docker教程资料/ubuntu24.04_docker镜像/` 文件夹下的 `ubuntu24.04_docker_images.tar.xz`。
- 声明： `ubuntu24.04_docker_images.tar.xz` 是官方下载的镜像，未经验证是否存在安全漏洞，只建议学习使用，商用后果自行承担。
- `mywebsite.tar.gz` 网页的压缩包位于 `docker教程资料/` 文件夹下的 `mywebsite.tar.gz`。

执行 `docker load` 命令导入 `ubuntu24.04_docker_images.tar.xz` 如下:

```
weimingze@mzstudio:~$ sudo docker load -i ubuntu24.04_docker_images.tar.xz
3abdd8a5e7a8: Loading layer
[==================================================>]  80.61MB/80.61MB
Loaded image: ubuntu:24.04
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
```

我们通过 `docker history` 命令可以查看这个 `ubuntu:24.04` 由几层组成。

```
weimingze@mzstudio:~$ sudo docker history ubuntu:24.04
IMAGE          CREATED        CREATED BY                                      SI
ZE      COMMENT
602eb6fb314b   2 months ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
<missing>      2 months ago   /bin/sh -c #(nop) ADD file:1d7c45546e94b90e9…
78.1MB
<missing>      2 months ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago   /bin/sh -c #(nop)  LABEL org.opencontainers.…   0B
<missing>      2 months ago   /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH     0B
<missing>      2 months ago   /bin/sh -c #(nop)  ARG RELEASE                  0B
```

从上述信息可知 `ubuntu:24.04` 镜像一共有 6 层只读层，默认的容器主进程的执行命令是 `/bin/bash`

下面我们以 `ubuntu:24.04` 作为基础镜像创建 Python Web 服务器。

使用 `docker run` 命令以 `ubuntu:24.04` 作为基础镜像，创建并运行器一个名为 `my_ubuntu` 的容器。并使用 `-it` 选项让容器主进程 `/bin/bash` 和终端绑定。

```
weimingze@mzstudio:~$ sudo docker run -it --name my_ubuntu ubuntu:24.04
root@0d0695780c8e:/#
```

> 如果运行失败则需要使用 `sudo docker rm my_ubuntu` 来删除容器后从创建并运行。

2.2 容器内安装程序运行环境

基础镜像 `ubuntu:24.04` 并没有安装 Python，但已经安装了 apt 命令。我们可以使用 apt 命令来安装 python3。

安装步骤如下：

- `apt update` 命令更新 apt 源列表。
- `apt install python3` 命令安装 Python3
- 解压缩 `mywebsite.tar.gz` 的网页内容到容器的 `/root/` 文件夹下。

1. 更新 apt 源列表

```
root@26371130cbef:/# apt update
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages
[1108 kB]
... 此处内容省略
9 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@26371130cbef:/#
```

2. 安装 Python3

```
root@26371130cbef:/# apt install python3
Reading package lists... Done
Building dependency tree... Done
... 此处内容省略

After this operation, 30.2 MB of additional disk space will be used.
Do you want to continue? [Y/n] y  # 此处输入 y 同意。

Unpacking libreadline8t64:amd64 (8.2-4build1) ...
Selecting previously unselected package libsqlite3-0:amd64.
... 此处内容省略

Please select the geographic area in which you live. Subsequent configuration qu
estions will narrow this down by presenting a list of cities, representing the t
ime zones in which
they are located.

  1. Africa  2. America  3. Antarctica  4. Arctic  5. Asia  6. Atlantic  7. Aust
ralia  8. Europe  9. Indian  10. Pacific  11. Etc  12. Legacy
Geographic area: 5  # 此处输入 5 选择位于亚洲
```

```
Please select the city or region corresponding to your time zone.

  1. Aden       10. Bahrain     19. Chongqing  28. Harbin       37. Jerusalem
46. Kuala_Lumpur  55. Novokuznetsk  64. Qyzylorda      73. Tashkent     82. Ust-
Nera
  2. Almaty     11. Baku        20. Colombo    29. Hebron       38. Kabul
47. Kuching       56. Novosibirsk   65. Riyadh         74. Tbilisi      83. Vien
tiane
  3. Amman      12. Bangkok     21. Damascus   30. Ho_Chi_Minh  39. Kamchatka
48. Kuwait        57. Omsk          66. Sakhalin       75. Tehran       84. Vlad
ivostok
  4. Anadyr     13. Barnaul     22. Dhaka      31. Hong_Kong    40. Karachi
49. Macau         58. Oral          67. Samarkand      76. Tel_Aviv     85. Yaku
tsk
  5. Aqtau      14. Beirut      23. Dili       32. Hovd         41. Kashgar
50. Magadan       59. Phnom_Penh    68. Seoul          77. Thimphu      86. Yang
on
  6. Aqtobe     15. Bishkek     24. Dubai      33. Irkutsk      42. Kathmandu
51. Makassar      60. Pontianak     69. Shanghai       78. Tokyo        87. Yeka
terinburg
  7. Ashgabat   16. Brunei      25. Dushanbe   34. Istanbul     43. Khandyga
52. Manila        61. Pyongyang     70. Singapore      79. Tomsk        88. Yere
van
  8. Atyrau     17. Chita       26. Famagusta  35. Jakarta      44. Kolkata
53. Muscat        62. Qatar         71. Srednekolymsk  80. Ulaanbaatar
  9. Baghdad    18. Choibalsan  27. Gaza       36. Jayapura     45. Krasnoyarsk
54. Nicosia       63. Qostanay      72. Taipei         81. Urumqi
Time zone: 69  # 此处是选择时区。选择 69. Shanghai


Current default time zone: 'Asia/Shanghai'
Local time is now:      Sun Jun 22 01:16:47 CST 2025.
Universal Time is now:  Sat Jun 21 17:16:47 UTC 2025.
Run 'dpkg-reconfigure tzdata' if you wish to change it.

Setting up netbase (6.4) ...
... 此处内容省略
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@26371130cbef:/#
```

然后退出容器

```
root@26371130cbef:/# exit
exit
weimingze@mzstudio:~$
```

3. 解压缩网页内容到容器的 `/root/` 文件夹下

在终端使用 `docker cp` 命令将 `mywebsite.tar.gz` 的内容解压缩到 容器的 `/root/` 文件夹下。

```
weimingze@mzstudio:~$ cat mywebsite.tar.gz | sudo docker cp - my_ubuntu:/root/
[sudo] password for weimingze:
Successfully copied 0B to my_ubuntu:/root/
```

在容器内查看结果，然后退出容器。

```
root@26371130cbef:/# cd root/
root@26371130cbef:~# ls
mywebsite
root@26371130cbef:~# ls -l
total 4
drwxrwxrwx 2 501 dialout 4096 Jun 21 21:26 mywebsite
root@26371130cbef:~# cd mywebsite/
root@26371130cbef:~/mywebsite# ls -l
total 8
-rwxrwxrwx 1 501 dialout 309 Jun 21 21:25 index.html
-rwxrwxrwx 1 501 dialout 293 Jun 21 21:26 second_page.html
root@26371130cbef:~/mywebsite# exit
exit
weimingze@mzstudio:~$
```

从上面可见 mywebsite 文件夹已经存在了。

至此，容器内部的文件系统准备完毕。

2.3 docker commit 创建镜像

使用 `docker commit` 命令制作镜像。

docker commit 命令作用

- 保存容器状态：将容器的文件系统变更（如安装的软件、修改的配置等）保存为新的镜像层。
- 创建自定义镜像：基于现有容器快速创建自定义镜像，无需从头编写 Dockerfile。
- 调试和实验：在容器中进行实验性修改后保存结果。

命令格式

```
docker commit [选项] 容器名或容器ID [仓库[:标签]]
```

> 命令别名:`docker container commit`

常用选项

选项	说明
`-a` 或 `--author`	指定镜像作者（如 -a "weimz author@weimingze.com"）。
`-c` 或 `--change`	应用 Dockerfile 指令到创建的镜像（可多次使用）。
`-m` 或 `--message`	提交消息，描述变更内容。
`-p` 或 `--pause`	在提交过程中暂停容器（默认 true）。

在终端内使用 `docker commit` 命令将容器打包为镜像。

目标

- 新的镜像名称定位 `python_web`
- 使用 `-c 'EXPOSE 8000'` 选项暴露 8000 端口供容器外部使用。
- 使用 `-c 'CMD ["python3", "-m", "http.server", "--directory", "/root/mywebsite", "8000"]'` 修改容器主进程的启动命令。
- 使用 `-c 'WORKDIR /root'` 修改容器主进程当前工作路径是 `/root/`。
- 使用 `-a weimingze` 指定新镜像作者是 `weimingze`。
- 使用 `-m 'docker commit 制作的第一个镜像'` 来添加描述信息。

示例

执行如下命令来创建镜像。

```
sudo docker commit \
   -a weimingze \
   -m 'docker commit 制作的第一个镜像' \
   -c 'EXPOSE 8000' \
   -c 'CMD ["python3", "-m", "http.server", "--directory", "/root/mywebsite",
"8000"]' \
   -c 'WORKDIR /root' \
   my_ubuntu python_web
```

运行结果如下：

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker commit -a weimingze -m 'docker commit 制作的第一
个镜像' -c 'EXPOSE 8000' -c 'CMD ["python3", "-m", "http.server", "--directory",
"/root/mywebsite", "8000"]' my_ubuntu python_web
sha256:d0cf826ce9ffc231eebeb96c45e58b458b825bec8dd71c45992e5e6e875bf49d
```

可见多了一个 docker 镜像 `python_web`。镜像制作完成。

查看制作好的镜像。

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
python_web   latest    d0cf826ce9ff   49 seconds ago   168MB
ubuntu       24.04     602eb6fb314b   2 months ago     78.1MB
```

使用 `docker history` 命令查看镜像制作历史记录。

```
weimingze@mzstudio:~$ sudo docker history python_web
IMAGE          CREATED          CREATED BY
SIZE      COMMENT
d0cf826ce9ff   48 minutes ago   /bin/bash
89.8MB    docker commit 制作的第一个镜像
602eb6fb314b   2 months ago     /bin/sh -c #(nop)  CMD ["/bin/bash"]
0B
<missing>      2 months ago     /bin/sh -c #(nop) ADD file:1d7c45546e94b90e9…
78.1MB
<missing>      2 months ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…
0B
<missing>      2 months ago     /bin/sh -c #(nop)  LABEL org.opencontainers.…
0B
<missing>      2 months ago     /bin/sh -c #(nop)  ARG LAUNCHPAD_BUILD_ARCH
0B
<missing>      2 months ago     /bin/sh -c #(nop)  ARG RELEASE
0B
```

用 `docker history` 查看镜像时最上层多了一层 ID 为 `d0cf826ce9ff` 的只读层，最后一层占用 `89.8MB` 的磁盘空间。

2.4 测试镜像

使用 `docker run` 命令创建并运行新的容器 `myweb`：

命令如下

```
sudo docker run --rm --name myweb -p 8000:8000 -d python_web:latest
```

运行结果

```
weimingze@mzstudio:~$ sudo docker run --rm --name myweb -p 8000:8000 -d
python_web:latest
1e084acb807eb70b884f9a2d96179bc12a9f1691cd55d16b3e260bc81b70074a
```

使用浏览器查看运行结果。

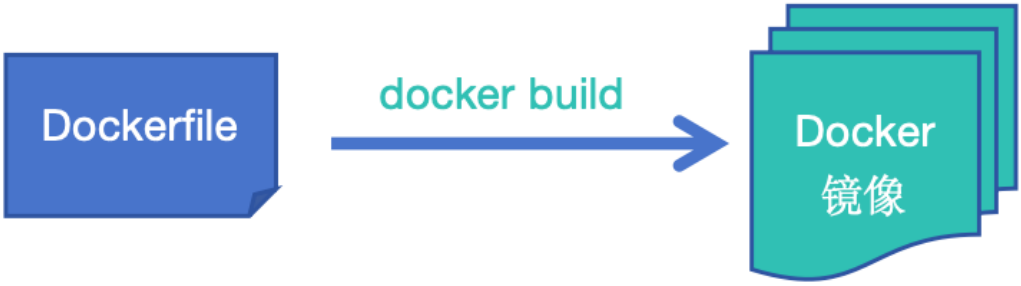
在本机打开浏览器，输入网址 http://0.0.0.0:8000/ 就可以访问这个网站了。

3. Dockerfile 制作镜像

什么是 Dockerfile

Dockerfile 是一个文本格式的配置文件，包含创建镜像所需要的全部指令。基于在 Dockerfile 中的指令，用户可以快速创建自定义的镜像。

如下图所示



通常使用 `docker commit` 命令只是临时修改和验证镜像。最通用的制作镜像的方法通常是使用 Dockerfile 来制作镜像。

Dockerfile 的作用:

- 用于描述 Docker 镜像的制作过程。
- 用简单语法用来创建镜像.
- 用自动化的脚本创建镜像

使用 Dockerfile 制作镜像的步骤:

1. 编写 Dockerfile 文件
2. 用 `docker build` 命令制作镜像
3. 测试和运行镜像

3.1 Dockerfile 的语法

Dockerfile 是用来描述镜像制作过程的文本文件，文件内可以由 Docker 规定的一些制作指令。

注释

Dockerfile中允许添加注释，注释的语法同 Python 和 Shell 一致，是 `#` 号开头，直至物理行末尾。

Dockerfile 的全部指令

指令	说明	示例
`FROM`	从一个基础镜像创建一个新的镜像。	`FROM ubuntu:24.04`
`COPY`	复制一个文件或文件夹到新的镜像	`COPY ./ubuntu.sources /etc/apt/sources.list.d/` apt 换源
`ADD`	类似于 `COPY`，`ADD` 添加 本地文件和远程文件或文件夹，并能够自动解压缩归档的 tar 包。	`COPY ./mywebsite.tar.gz /root/`
`ARG`	定义创建时的变量（创建后不存在于镜像中）。	`ARG VERSION=24.04` 后使用 `FROM ubuntu:${VERSION}`
`CMD`	指定容器主进程执行的缺省命令和参数(可以被 `docker run` 覆盖)	`CMD ["python3", "run.py"]`
`ENTRYPOINT`	指定容器主进程执行的缺省命令。	`ENTRYPOINT ["python3"]` 然后使用 `CMD ["run.py"]` 提供参数和选项
`ENV`	设置环境变量。	`ENV PATH=$PATH:/root/bin`

`EXPOSE`	描述你的应用监听的端口，并暴露出来供宿主机使用 `-p` 选项映射连接。	`EXPOSE 8000 5000/tcp`、`EXPOSE 7001-7005`
`HEALTHCHECK`	在启动时定义容器健康检查。	`HEALTHCHECK --interval=5m --timeout=3s CMD curl -f http://localhost:8000/ || exit 1`
`LABEL`	在一个镜像上添加元数据。如：版本，作者，描述等。	`LABEL maintainer="docker@weimingze.com"`、`LABEL version="1.0"`
`MAINTAINER`	设置镜像作者的信息（通常是姓名和联系方式）。	`MAINTAINER laowei <docker@weimingze.com>`
`ONBUILD`	设置当本镜像被用作基础镜像时执行的指令。	`ONBUILD COPY . /app/src` 或 `ONBUILD RUN make /app/src`
`RUN`	执行创建命令并创建新的镜像层。	`RUN apt update && apt install python3 -y`
`SHELL`	设置镜像默认的 Shell（覆盖）。	`SHELL ["/bin/bash", "-c"]`

`STOPSIGNAL`	设置一个退出容器的系统信号。	`STOPSIGNAL SIGINT` 设置容器停止时发送 SIGINT 信号 (等同于 Ctrl+C)。
`USER`	设置运行时的用户(镜像内要添加此用户)。	`USER weimingze` 或 `USER 1001`
`VOLUME`	创建挂载点，外部可以使用 `-v` 挂载宿主机路径。	`VOLUME /root/mywebsite`
`WORKDIR`	设置工作路径	`WORKDIR /root`

上述指令详见官方文档: https://docs.docker.com/reference/dockerfile/

接下来我们使用 Dockerfile 来创建 Docker 镜像。

1. 编写文件 `Dockerfile`

```
weimingze@mzstudio:~$ vim Dockerfile
```

2. 写入内容

```
# 设置基础镜像
FROM ubuntu:24.04

# 设置作者信息
MAINTAINER laowei <docker@weimingze.com>

LABEL version="0.1" description="这是 基于ubuntu 24.04 制作的 python web 服务器镜像" by="魏明择"

# 设置当前时区环境变量为 "Asia/Shanghai"
ENV TZ="Asia/Shanghai"

# 运行命令安装 python3
RUN apt update;
RUN apt install python3 -y;
```

```
# 打印一条信息到 `docker build` 命令的标准输出
RUN echo "Python3 Install OK!"

# 设置当前工作路径(等同于 cd 命令)
WORKDIR /root

# 将网页放入镜像的 /root/mywebsite 文件夹
ADD ./mywebsite.tar.gz /root/

# ENV：设置环境变量
ENV PATH=$PATH:/root/bin

# 暴露 8000 端口
EXPOSE 8000

# 设置 挂载点，可以在 docker run 中通过 -v 选项挂载宿主机路径。
VOLUME /root/mywebsite

CMD ["python3", "-m", "http.server", "--directory", "/root/mywebsite", "8000"]
```

> 确保上节课用到的mywebsite.tar.gz 文件也在 Dockerfile 所在的文件夹下。

至此 Dockerfile 编写完毕。

3.2 docker build 命令

使用 `docker build` 可以用 Dockerfile 创建一个 Docker 镜像。它会执行 Dockerfile 中的指令，创建可执行的容器镜像。

命令格式

```
docker build [选项] 本地路径、URL连接或标准输入(-)
```

> 命令别名: `docker image build`, `docker builder build`。

命令说明:

- `-` 表示通过 标准输入 `STDIN` 给出 Dockerfile。

常用选项

选项	说明
`-f <Dockerfile文件路径>`	指定Dockerfile 的路径（默认是`./Dockerfile`）
`-t <镜像的名称[:标签]>`	设置新镜像的名称和标签(可以定义多个)
`--build-arg`	设置创建时的变量（对应 Dockerfile 中的 ARG）。
`--pull`	总是尝试拉取基础镜像的新版本
`--no-cache`	创建时不使用缓存
`-q`	安静模式，只输出镜像 ID
`--rm`	创建成功后删除中间容器（默认 true）
`--target`	创建多阶段镜像中的特定阶段

示例

我们使用 上节课编写的 Dockerfile 来创建名为 `myubuntu_img` 的镜像。

创建执行命令

```
docker build -t myubuntu_img -f Dockerfile .
```

- 新镜像的名称和标签为 `myubuntu_img:latest`
- 使用的 Dockerfile 文件为 `Dockerfile`
- `.` 是创建时使用本地的路径（当前路径）

执行过程如下：

```
weimingze@mzstudio:~$ sudo docker images;
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       24.04     602eb6fb314b   2 months ago   78.1MB
weimingze@mzstudio:~$ sudo docker build -t myubuntu_img -f Dockerfile .
DEPRECATED: The legacy builder is deprecated and will be removed in a future rel
ease.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/
... # 此处内容省略
Step 12/13 : VOLUME /root/mywebsite
 ---> Running in ad665e896f88
 ---> Removed intermediate container ad665e896f88
 ---> 761f35ffe848
Step 13/13 : CMD ["python3", "-m", "http.server", "--directory", "/root/
mywebsite", "8000"]
 ---> Running in a57c2dec6218
 ---> Removed intermediate container a57c2dec6218
 ---> aadcd3546f14
```

```
Successfully built aadcd3546f14
Successfully tagged myubuntu_img:latest
weimingze@mzstudio:~$ sudo docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
myubuntu_img   latest    aadcd3546f14   26 seconds ago   168MB
ubuntu         24.04     602eb6fb314b   2 months ago     78.1MB
```

可见 `myubuntu_img` 创建成功。

3.3 测试和运行镜像

使用 `docker run` 命令创建并运行新的容器，如下：

```
weimingze@mzstudio:~$ sudo docker run --rm --name myweb -p 8000:8000 -d
myubuntu_img
c76feeb87940ab3bc8b0378a872ca287790a10117bff4db55651247fa856962e
weimingze@mzstudio:~$
```

使用浏览器查看运行结果。

在本机打开浏览器，输入网址 http://0.0.0.0:8000/ 或 http://localhost:8000/ 就可以访问这个网站了。

重新定义端口号和网页路径

- 将本机 的 80 端口号映射为容器端口号的 8000，这样在浏览器中就不用输入端口号了（http 协议默认使用 80 端口）。
- 将本地的 `html` 内部存放的页面映射到容器内部的 `/root/mywebsite` 文件夹以达到即时更换网站页面的目的。网站升级是不需要重新创建镜像，只需要修改宿主机器的 `html` 文件夹。

执行命令

```
sudo docker run --rm --name myweb -p 80:8000 -v ./html:/root/mywebsite -d
myubuntu_img
```

> 注意：需要再当前工作路径下的 `html` 文件夹存放 html 页面。

此时在浏览器，输入网址 http://0.0.0.0/ 或 http://localhost/ 就可以访问这个网站了。无需输入端口号 8000。并且网页的内容是本地宿主机的 `./html` 文件夹的内容。

第七章、搭建私有仓库

在某些私密或 Docker 镜像源稳定性要求比较高的场合可以在本地搭建私有仓库。以此降低 `docker pull` 下载镜像的风险和事故。

搭建私有 Docker 仓库可以达到 **安全**、**高效**、**可控** 的目的。

应用场景

1. 隐私保护
   - 项目保密级别比较高。不想让自己的镜像发布到公网。
2. 提升镜像分发效率
   - 在某些分布式系统中，快速分发镜像到多个节点（可能是上万个计算节点或存储节点）。
3. 资源稳定性控制
   - 防止外网因为网络故障等原因导致项目部署出现问题。

搭建 Docker 私有仓库最简单的方法是使用 Docker 的 `registry` 镜像。

下面我们来说一下使用 Docker 镜像搭建私有仓库的方法。

搭建和导入镜像大致分为如下几个步骤：

1. 安装 docker（略）。
2. 下载和安装 registry 镜像。
3. 使用 registry 镜像创建容器并运行。
4. 验证私有仓库是否可用。
5. 导入本地镜像。
6. 从私有仓库下载镜像。

在安装了 Docker 的系统上，可以使用官方镜像 `registry:latest` 来安装私有仓库。

1. 安装 registry 镜像

方法1：使用 `docker pull` 下载和安装镜像

命令如下:


```
sudo docker pull registry:latest
```

执行结果如下：

```
weimingze@mzstudio:~$ sudo docker pull registry:latest
latest: Pulling from library/registry
f18232174bc9: Pull complete
e5a9c19e7b9d: Pull complete
e8a894506e86: Pull complete
e1822bac1992: Pull complete
b5da7f963a9e: Pull complete
Digest: sha256:1fc7de654f2ac1247f0b67e8a459e273b0993be7d2beda1f3f56fbf1001ed3e7
Status: Downloaded newer image for registry:latest
docker.io/library/registry:latest
weimingze@mzstudio:~$
```

方法2：使用 docker load 导入镜像

先从百度网盘下载 镜像压缩包。

- 链接地址：https://pan.baidu.com/s/1U2CUtna3oq9NFkh8ewMzdg 提取码: 9gy6

找到 `docker教程资料/registry_docker仓库镜像/` 文件夹下的 `registry_docker_image.tar.xz` 并下载。然后使用 scp 命令复制到 Ubuntu24.04 操作系统内。

> 声明： `registry_docker_image.tar.xz` 是官方下载的镜像，未经验证是否存在安全漏洞，只建议学习使用，商用后果自行承担。

执行 `docker load` 命令导入 `registry_docker_image.tar.xz` 如下:

```
sudo docker load -i registry_docker_image.tar.xz
```

2. 创建容器并运行

创建一个名称为 my_local_registry 的 `registry` 容器.

创建命令：

```
sudo docker run -d \
    -p 5000:5000 \
    --restart=always \
```

```
    --name my_local_registry \
    registry:latest
```

默认私有仓库的存放位置是容器内的 `/var/lib/registry`。 在创建仓库时，可以使用 -v 选项将其映射到宿主机的某个路径下。这样保证了数据的安全性，同时避免容器可写层过大的问题。

将 本地 当前路径下的 `data` 文件夹映射到容器内的 `/var/lib/registry` 文件夹。

命令如下:

```
sudo docker run -d \
    -p 5000:5000 \
    --restart=always \
    --name my_local_registry \
    -v "$(pwd)"/data:/var/lib/registry \
    registry:latest
```

执行结果

```
weimingze@mzstudio:~$ sudo docker run -d \
    -p 5000:5000 \
    --restart=always \
    --name my_local_registry \
    -v "$(pwd)"/data:/var/lib/registry \
    registry:latest
a78221f81b7af8dc0e541654db0e4a38f50f90245cbfe736674194b584be0f15
```

3. 验证私有仓库是否可用

使用浏览器或 `curl` 命令 访问私有仓库的 http://localhost:5000/v2/ ，如果返回一个 JSON 字符串 `{}` 说明服务器正常。

使用 curl 执行结果如下

```
weimingze@mzstudio:~$ curl http://localhost:5000/v2/
{}weimingze@mzstudio:~$
```

> 注意 `{}` 在命令提示符前面。

使用浏览器或 `curl` 命令 访问私有仓库的 http://localhost:5000/v2/_catalog，将返回一个仓库内的镜像的列表。此时没有放入镜像，列表也为空。

运行结果如下：

```
weimingze@mzstudio:~$ curl http://localhost:5000/v2/_catalog
{"repositories":[]}
```

以上结果说明私有仓库的服务器正常运行了。

4. 导入本地镜像

下面我们来讲解一下如何将本地镜像导入到搭建的本地私有仓库。

我们要确认如下信息:

- 私有仓库地址是本机，因此地址是 `localhost:5000`
- 私有仓库地址是本网络内的其他主机，则地址是 `<主机IP地址>:5000` 如 `192.168.33.148:5000`。

步骤如下:

1. 给本地的镜像打标签，指向仓库。
2. 使用 `docker push` 命令将打完标签的本地镜像推送到私有仓库。

注意事项

如果私有仓库需要认证，需要在推送前先登录，如:

```
sudo docker login localhost:5000
```

1. 打标签

在推送到仓库前，需要将镜像打标签，并指向私有仓库的位置。

命令格式

```
docker tag 镜像[:标签] 仓库地址/新的镜像名称:新的标签名
```

示例

将 官方镜像 `ubuntu:24.04` 推送到 `localhost:5000` 的仓库，重新命令为 `ubuntu2404:latest`。

```
weimingze@mzstudio:~$ sudo docker tag ubuntu:24.04 localhost:5000/
ubuntu2404:latest
```

查看打完标签的镜像

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY                  TAG       IMAGE ID       CREATED        SIZE
ubuntu                      24.04     602eb6fb314b   2 months ago   78.1MB
localhost:5000/ubuntu2404   latest    602eb6fb314b   2 months ago   78.1MB
registry                    latest    3dec7d02aaea   2 months ago   57.7MB
```

2. docker push 推送到仓库

使用 `docker push` 命令将 标签为 `localhost:5000/ubuntu2404:latest` 的镜像推送到 私有仓库。

docker push 命令

命令格式

```
docker push [选项] 镜像名[:标签]
```

> 命令别名:`docker image push`。

常用选项

选项	说明
`-q`	静默模式，不打印输出。

示例

推送 `localhost:5000/ubuntu2404:latest` 的到私有仓库。

```
weimingze@mzstudio:~$ sudo docker push localhost:5000/ubuntu2404:latest
The push refers to repository [localhost:5000/ubuntu2404]
3abdd8a5e7a8: Pushed
latest: digest: sha256:0b9e751164f0f576086bb03062186c5643fe6dc56223b6bfef0be2a9d
4828c67 size: 529
```

验证推送结果

通过浏览器或 `curl` 命令访问 http://localhost:5000/v2/_catalog 查看仓库信息。

```
weimingze@mzstudio:~$ curl http://localhost:5000/v2/_catalog
{"repositories":["ubuntu2404"]}
```

可见本地仓库内已经有 名为 `ubuntu2404` 的镜像

查看 仓库内 镜像的标签信息的链接地址格式:

```
http://localhost:5000/v2/<镜像名>/tags/list
```

如，可以通过如下方式查看 `ubuntu2404` 的镜像的标签信息

```
weimingze@mzstudio:~$ curl http://localhost:5000/v2/ubuntu2404/tags/list
{"name":"ubuntu2404","tags":["latest"]}
```

5. 私有仓库下载镜像

从私有仓库内下载镜像分为两种情况：

1. 从本机 `localhost` 下载镜像
2. 从其他的私有仓库下载镜像（需要用到域名或 IP 地址)

1. 从本机 `localhost` 下载镜像

使用 `docker pull` 命令就可以下载本地镜像，

命令格式:

```
docker pull localhost[:端口号]/镜像名[:标签]
```

示例

下载 `localhost:5000/ubuntu24.04:latest` 镜像。

```
weimingze@mzstudio:~$ sudo docker pull localhost:5000/ubuntu2404
Using default tag: latest
latest: Pulling from ubuntu2404
Digest: sha256:0b9e751164f0f576086bb03062186c5643fe6dc56223b6bfef0be2a9d4828c67
Status: Downloaded newer image for localhost:5000/ubuntu2404:latest
localhost:5000/ubuntu2404:latest
```

查看镜像

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY                  TAG       IMAGE ID       CREATED        SIZE
ubuntu                      24.04     602eb6fb314b   2 months ago   78.1MB
localhost:5000/ubuntu2404   latest    602eb6fb314b   2 months ago   78.1MB
registry                    latest    3dec7d02aaea   2 months ago   57.7MB
```

2. 从 其他的私有仓库下载镜像

如果下载镜像的私有仓库，不是本机 `localhost` 则需要修改 Docker 配置文件 `/etc/docker/daemon.json`，将允许不安全的仓库信息加入配置文件中。

方法如下:

- 编辑 `/etc/docker/daemon.json`
- 添加 `"insecure-registries"` 键值对，指向 私有仓库的主机地址和端口号，格式如下：

```
{
  "insecure-registries" : ["私有仓库主机域名或IP地址:5000"]
}
```

示例

我私有仓库的 IP 是 `192.168.33.148`，我的配置文件修改如下：

```
{
  "insecure-registries" : ["192.168.33.148:5000"]
}
```

然后重启 当前主机的 docker 服务。

```
sudo systemctl restart docker
```

接下来就可以使用 docker pull 下载私有仓库的镜像了。

示例

下载 `192.168.33.148:5000/ubuntu24.04:latest` 镜像。

```
weimingze@mzstudio:~$ sudo docker pull  192.168.33.148:5000/ubuntu2404
Using default tag: latest
latest: Pulling from ubuntu2404
Digest: sha256:0b9e751164f0f576086bb03062186c5643fe6dc56223b6bfef0be2a9d4828c67
Status: Downloaded newer image for 192.168.33.148:5000/ubuntu2404:latest
192.168.33.148:5000/ubuntu2404:latest
```

查看下载的镜像

```
weimingze@mzstudio:~$ sudo docker images
REPOSITORY                          TAG       IMAGE ID       CREATED        SIZE
ubuntu                              24.04     602eb6fb314b   2 months ago   78.1MB
192.168.33.148:5000/ubuntu2404      latest    602eb6fb314b   2 months ago   78.1MB
registry                            latest    3dec7d02aaea   2 months ago   57.7MB
```

至此，私有仓库搭建并测试完成。