

Docker 入门与实战

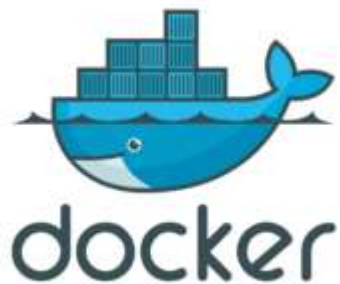
于新雨@Rosayxy

2024.7.27

- Why We Need Docker
- 基本概念和原理
- 基本命令
- Dockerfile
- Docker Compose
- Registry

What Is Docker

- Docker 是一套平台即服务 (PaaS) 产品, 它使用操作系统级别的虚拟化技术来提供名为容器的软件包 (ref from wikipedia)
- 2013 年面向公众亮相, 2013年3月开源 (Apache License 2.0)
- 吉祥物 Moby Dock



Why We Need Docker



清华大学
Tsinghua University

网络研究院
INSC

- 配环境是程序员的基本素养
 - Linux 环境配置, 小学期配置本地环境
 - 软件工程专业课, 配置 CI/CD
 - 科研中复现前人工作 ...



Source: <https://imgflip.com/379x/5679elcy1832330643323>

Why We Need Docker

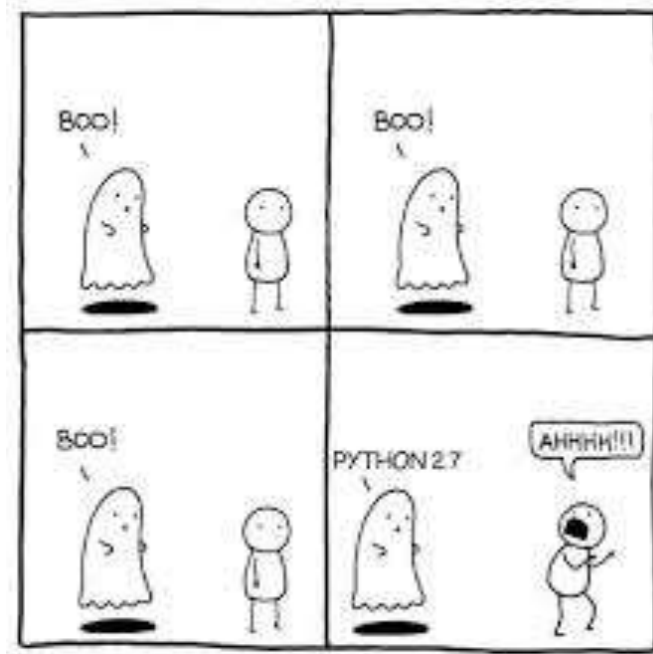


清华大学
Tsinghua University

网络研究院
INSC

- 可能遇到的问题

- 依赖项冲突
- 用 WSL2 的朋友可能遇到需要原生 linux 发行版的情况
- 异构 (比如说需要 arm64, arm32 的环境)
- 需要基于已有框架线上部署环境
- 可能会运行一些对本机有风险的程序
- [您配吗](#) 中所提到的种种情况..



- 因此，我们需要一个可以承载特定环境的东西（容器）来运行所需要的程序
- 这个容器应当有以下需求
 - 和程序可以一起被打包派发（善良的开发一定会考虑到不想配环境的大家！）
 - 和原有操作系统环境有一定程度的隔离
 - 不同容器之间运行环境互不影响
 - 一定程度上的安全性
 - 运行较为方便

- 环境一致性

- 可以确保应用在不同环境（开发、测试、生产）中运行时的一致性，从而减少了 “It works on my machine (摊手)” 的问题

- 便捷依赖管理

- 在容器中封装应用及其所有依赖，使依赖管理更简单，不再需要在不同环境中重复配置环境

- 资源高效利用

- 相比传统的虚拟机，Docker 使用更少的资源（如CPU和内存），因为容器共享主机的操作系统，而不需要单独的操作系统

- 可扩展性
 - 非常适合微服务架构，因为可以将应用分解为多个独立的服务，每个服务运行在自己的容器中。
- 方便线上部署
 - 容易集成到现有的 CI/CD 管道中
- 一定的隔离与安全性
 - Docker 容器提供了相对隔离的环境，减少了应用之间的冲突。此外，容器的隔离性也提高了应用的安全性。
 - 或许大家听说过 Docker 逃逸，但是在正确的配置下 Docker 很难被攻击利用

- 虚拟机也可以保存封装程序和运行环境，并且虚拟机镜像也容易打包给他人
- 在这种情况下，为什么我们还是经常选用 Docker 呢 ~
- 可以注意到，相比于虚拟机，Docker 有如下特点

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个



- 如图可以看到，和虚拟机相比 Docker 更加轻量级，运行时的性能更优，也确实，它直接运行在主机内核上，使用主机的资源
 - 虚拟机同时会有硬件虚拟化，比如 VMWare，直接在计算机硬件或主机操作系统上面插入一个软件层
 - 而本质上与其说 Docker 运用了虚拟化技术，不如说是用了隔离技术，和 Linux 的 chroot 命令所做的操作有些类似

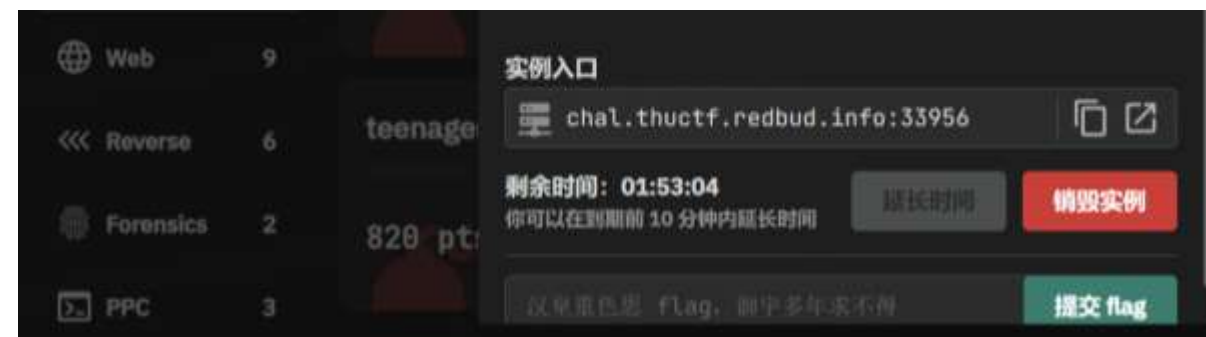
特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

- 镜像 (Image)

- 是一个轻量级、独立且可执行的软件包，包括运行软件所需的一切，包括代码、运行时环境、库、环境变量和配置文件。
- Docker 镜像本质上是一个特殊的 root 文件系统，具体来说，在 Docker 内部直接 `pwd` 之类的得到的是 `"/"` 但是在宿主机上有目录的对应，如 `"var/lib/docker/overlay2/1234567890abcdef/merged"`
- **镜像是静态的，build 之后不会改变内容**
- 此外，Docker 镜像的文件系统组成和构建方法都是分层的，这种 layering 机制也是计算机很多设计的核心思想

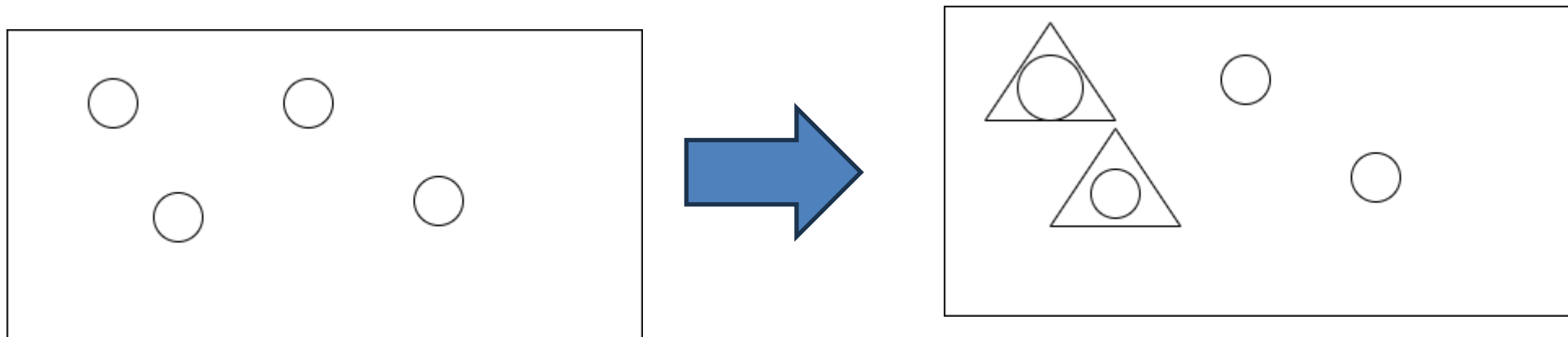
- 容器 (container)

- 容器和镜像的关系相当于面向对象中类与对象的对应关系。镜像规定了运行程序时的系统类型，文件系统组成，环境变量之类的，容器则是实例，可以被创建，启动，销毁
- 回忆面向对象程序设计中类和对象的讲解，容器内对环境的修改（e.g 环境变量，文件系统）并不会导致镜像中内容的变化



- 容器 (container)

- 容器其实依赖于隔离而非虚拟化技术，更深一层有如下两个隔离方法：控制隔离，资源隔离
- 控制隔离依赖于 namespace 技术，具体来说，Docker 内进程都可以在宿主主机上查看到，但是内外 pid 不一样



- 容器 (container)

- 容器其实依赖于隔离而非虚拟化技术，更深一层有如下两个隔离方法：控制隔离，资源隔离
- 资源隔离依赖于 linux 上的 [cgroup](#) 它是一个特别的文件系统类型，可以限制、记录任务组所使用的物理资源，从而防止容器里的程序占用太多的宿主机 cpu

- 服务 (service)

- 如果在 Docker 开启之前就尝试运行的话，可能会遇到以下报错，这时候你可能会产生这样的疑问，这个 daemon 是啥，我为什么又要用 service 这个 keyword 去开启 Docker 呢
- Service (服务) 是在后台运行的进程或应用程序，它执行一些预定义的任务或等待某些事件发生。
- 具体来说，service 对应 /usr/sbin/service，可以调用任意的可执行文件

```
yuxinyu@~$ docker build .  
ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?  
yuxinyu@~$ sudo service docker start  
[sudo] password for yuxinyu:  
* Starting Docker: docker
```

- Daemon

- Daemon 其实是小精灵的意思hhh，在计算机术语中，叫做守护进程
- 它是后台长时间执行的进程，很多系统服务都由 Daemon 提供，比如 sshd (管理 ssh 连接)，httpd (Apache http server)



Emma Best 🏳️‍🌈 | **Demon Hacker** @NatSecGeek · 1m

Gab CEO: "What did you just say?"

Gab tech: "I said the hacker accessed a daemon to--."

Gab CEO, running away: "DEMON HACKERS! DEMON HACKERS!"



5



13



- Docker run <参数> <image> <command>
 - 基于给定镜像启动一个新容器
 - Image 参数可以由 ImageId 或者 name 指定
 - Command 就是正常的 linux 命令行写法，如 echo "hello world",pwd,ls 之类的命令
 - 一些常用参数
 - -rm: 在运行完容器后删除实例
 - -it: 使用交互模式并且分配伪终端（退出伪终端的话在伪终端执行 exit 就行）
 - --name=name: 为容器分配名字，不然自动分配的名字不太好记

- `docker build <Options> <Path| Url>`

➤ 常用参数

- `-t <name>`
- `-q (quiet)` : 如果看到一堆输出觉得烦的话可以用一下

- docker images

- 列出已有镜像

```
PS C:\Users\yuxinyu> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rosayxy              latest             842487532ced       26 minutes ago    72.8MB
<none>              <none>            c036b9c7cc3c       29 minutes ago    72.8MB
<none>              <none>            035b6e3c4dc5       24 hours ago      208MB
bblar                latest             3431ee2d5f85       25 hours ago      208MB
<none>              <none>            f64db7b0e31b       2 days ago        322MB
alpine               latest             324bc02ae123       3 days ago        7.8MB
hello-world          latest             d2c94e258dcb       15 months ago     13.3kB
```

- docker ps (--all)

- 列出正在运行的/所有 container

```
PS C:\Users\yuxinyu> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
PS C:\Users\yuxinyu> docker ps --all
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
763c7384b853   rosayxy   "pwd"     4 minutes ago   Exited (0) 4 minutes ago   PORTS   clever_lew
in
4d2bea8f2f3a   rosayxy   "echo 'hello world!'" 5 minutes ago   Exited (0) 5 minutes ago   PORTS   amazing_wi
les
alf26ff3c3f2   rosayxy   "/bin/bash" 5 minutes ago   Exited (0) 5 minutes ago   PORTS   laughing_w
iles
06976d5f8524   bblar     "/home/apache/bin/ht..." 9 minutes ago   Exited (0) 9 minutes ago   PORTS   dreamy_boyd
```

- `docker rmi <Options> <Image>`
 - 删除镜像
- `docker rm <Options> <Container>`
 - 删除容器
- `docker cp <Options> <Container:Src_Path> <Dst_Path>`
- `docker cp <Options> <Src_Path> <Container:Dst_Path>`
 - 解锁 docker 的新用法：获得正常渠道（比如说 google 或者官网）较难获取的库文件等资源

- Demo

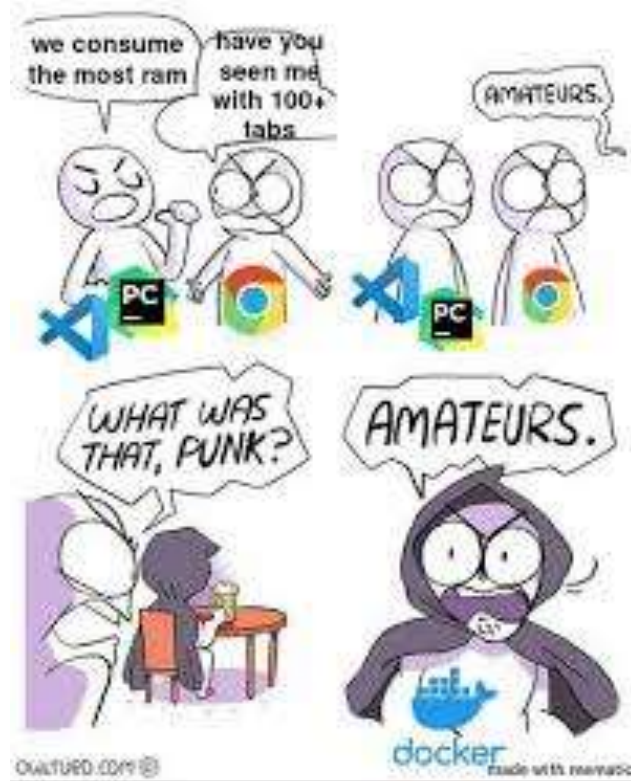
- `docker run alpine echo "hello world!"`

- From Dockerfile to working Docker !

- `cd <Workdir>`
 - `sudo service docker start`
 - `docker build -t "rosa" .`
 - `docker images`
 - `docker run --name=rosadocker -it rosa`
 - (internal commands...)
 - `docker cp`



- 看到别人的 Dockerfile, 我也想自己写一个!
- 基本思路:
 - 在之前我们已经剧透过, Docker 是多层的, 所以我们需要做的只是在基础镜像上再添加一层或者几层!



- FROM

- 指定基础镜像

- FROM ubuntu:22.04 | FROM python:3.10-slim

- RUN

- 指定创建 containers 所运行的命令，和 linux 命令行同格式，可以用来布置好运行程序的环境

- RUN apt update && apt install -y --no-install-recommends socat

- 加上 -y 参数代表对安装时所有 prompt 都 say yes

- COPY

- COPY <src> <dst>
- src 以 Dockerfile 所在目录为相对路径，dst 对应 docker 内文件系统的路径
- COPY server.py /server.py

- CMD

- 容器启动的时候运行的命令
- CMD ["/bin/bash"] | CMD echo "hello world"

- WORKDIR

- 用于为 Dockerfile 中随后的任何 RUN、CMD、ENTRYPOINT、COPY 和 ADD 指令设置工作目录。类似于对 Dockerfile 来说的 cd
- WORKDIR /rosa

- ENV

- Dockerfile 中设置环境变量
- ENV ROSA= "HAPPY"

- 我们 build 所需要的文件可能需要很多依赖项，需要把它们都塞到容器里面
 - 派发一个很大的镜像给用户显然不太好！
- 但是真正运行 build 出来的文件可能只需要一些简单的库依赖
- 我们可以在构建中使用多个阶段，每个阶段用一个可以不同的镜像，把最后一阶段的镜像给用户



- 实例：thuctf 2023 某题 Dockerfile

```
1  # Set sha256 of ubuntu:22.04
2  FROM ubuntu:22.04@sha256:f154feaf13b51d16e2b4b5575d69abc808da40c4b80e3a5055aaa4bcc5099d5b AS builder
3
4  COPY src/teenagerstack.c /teenagerstack.c
5  RUN apt update && \
6      apt --no-install-recommends install -y gcc gcc-multilib
7  RUN gcc -fno-stack-protector -no-pie -D_FORTIFY_SOURCE=0 -Og -o /teenagerstack /teenagerstack.c
8
9  FROM ghcr.io/thuctf/ctf5:5099d5b AS d:alpine
10
11 COPY init.sh /init.sh
12 COPY xinetd.conf /etc/xinetd.conf
13
14 RUN chmod +x /init.sh && \
15     chown -R ctf:ctf /home/ctf && \
16     chmod -R 750 /home/ctf && \
17     cp -R /lib* /home/ctf && \
18     mkdir /home/ctf/lib64 && \
19     mkdir /home/ctf/dev && \
20     mknod /home/ctf/dev/null c 1 3 && \
21     mknod /home/ctf/dev/zero c 1 5 && \
22     mknod /home/ctf/dev/random c 1 8 && \
23     mknod /home/ctf/dev/urandom c 1 9 && \
24     chmod 666 /home/ctf/dev/* && \
25     mkdir /home/ctf/bin && \
```


- Dockerfile 总体来说不难，适合使用 copilot 等工具





- 在构建 Docker 的时候，会遇到更复杂的情况...
 - 挂载数据卷
 - 创建数据库
 - 不同 Docker 容器之间网络连接

- Solution: Docker Compose

➤ 或许你在日常生活中已经见到过 docker-compose.yml ?

```
docker-compose.yml - The Compose specification establishes a standard for
1  service:
2  build: ./
3  volumes:
4    - ./share:/home/username:ro
5    - ./xinetd:/etc/xinetd.d/:ro
6    - ./tmp:/tmp:ro
7  ports:
8    - "6666:6666"
9  expose:
10    - "6666"
11
```



- Yaml format
- tl;dr: yaml 和 json 格式有类似于如下的等价转换 (例子 ref: [What is YAML? The YAML File Format \(freecodecamp.org\)](https://www.freecodecamp.org/yaml/))

```
Employees:  
- name: John Doe  
  department: Engineering  
  country: USA  
- name: Kate Kateson  
  department: IT support  
  country: United Kingdom
```



```
{  
  "Employees": [  
    {  
      "name": "John Doe",  
      "department": "Engineering",  
      "country": "USA"  
    },  
    {  
      "name": "Kate Kateson",  
      "department": "IT support",  
      "country": "United Kingdom"  
    }  
  ]  
}
```

- Yaml format
- tl;dr: yaml 和 json 格式有类似于如下的等价转换 (例子 ref: [What is YAML? The YAML File Format \(freecodecamp.org\)](https://www.freecodecamp.org/yaml/))

```
Employees:  
- name: John Doe  
  department: Engineering  
  country: USA  
- name: Kate Kateson  
  department: IT support  
  country: United Kingdom
```



```
{  
  "Employees": [  
    {  
      "name": "John Doe",  
      "department": "Engineering",  
      "country": "USA"  
    },  
    {  
      "name": "Kate Kateson",  
      "department": "IT support",  
      "country": "United Kingdom"  
    }  
  ]  
}
```


- 以右图中 Docker-Compose.yml 为例
 - Volumes 用来创建数据卷，我们提供了一个 db-vol 的空词典，代表用默认配置创建一个名叫 db-vol 的数据卷，数据卷的作用是把容器里面的数据保存在宿主机上，它也可以在容器之间共享
 - 每个 service 对应一个 Docker，对于需要构建镜像的后端，我们用 build 指定 docker build 的目录，否则我们也需要指定选用的镜像

```
volumes:
  db-vol:

services:
  backend:
    build: .
    ports:
      - "9000:80"
    volumes:
      - "${PWD}/config.json:/config/config.json"
    restart: unless-stopped

  mysql:
    image: mysql:latest
    volumes:
      - "db-vol:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: "my-secret-pw"
      MYSQL_DATABASE: leaderboard
```

- 以右图中 Docker-Compose.yml 为例
 - 每一项中 volumes 指定了需要挂载的数据卷或者绑定宿主机的目录（此时宿主机目录需要用绝对路径）
 - ports 代表了 Docker 内外端口的映射关系
 - environment 一项体现容器的环境变量不仅可以用 Dockerfile 设置，也可以设置在这里

```
volumes:
  db-vol:

services:
  backend:
    build: .
    ports:
      - "9000:80"
    volumes:
      - "${PWD}/config.json:/config/config.json"
    restart: unless-stopped

  mysql:
    image: mysql:latest
    volumes:
      - "db-vol:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: "my-secret-pw"
      MYSQL_DATABASE: leaderboard
```

- 以右图中 Docker-Compose.yml 为例
 - **restart: unless-stopped** 一项指不断重启直到成功为止
- 在这个例子中，我们不确定 mysql 和 backend 的建立先后顺序，然而我们的后端需要等待 MySQL 初始化完成之后才能够开始运行
- 所以可以让后端如果因为数据库的问题运行失败，就重新启动，直到成功

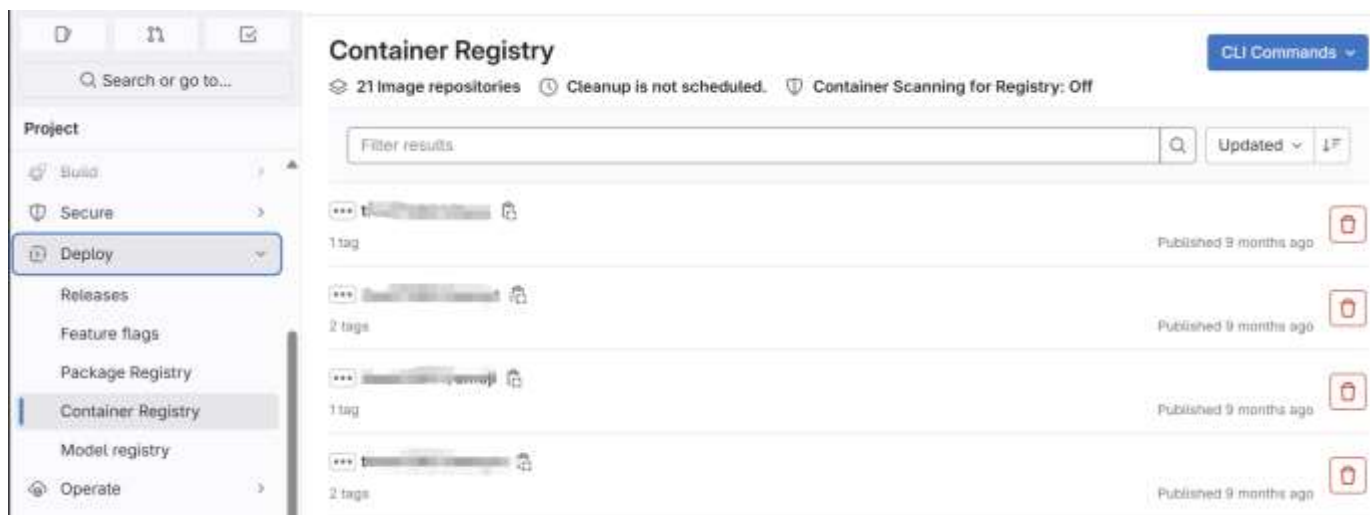
```
volumes:
  db-vol:

services:
  backend:
    build: .
    ports:
      - "9000:80"
    volumes:
      - "${PWD}/config.json:/config/config.json"
    restart: unless-stopped

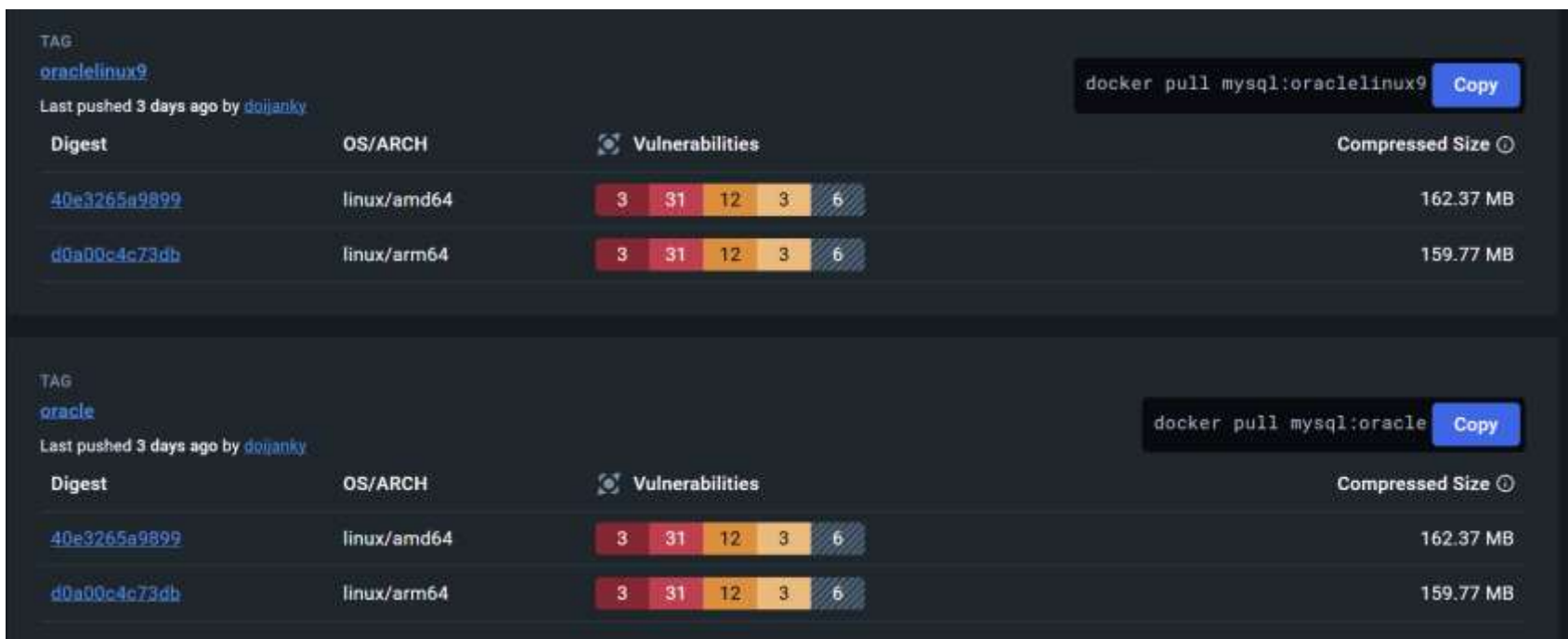
  mysql:
    image: mysql:latest
    volumes:
      - "db-vol:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: "my-secret-pw"
      MYSQL_DATABASE: leaderboard
```

- Docker compose up
 - 根据 docker-compose.yml 一键运行所有操作
- 其他命令如 `docker compose down`: 停止所有容器, `docker compose logs <服务名>`: 查看某个容器的日志, 大家可以课后探索一下

- 镜像构建完成后，需要一个集中的存储，分发镜像的服务
- 这类服务被称为注册服务（Registry），[Docker Hub](#) 是目前最大的 Docker 镜像注册服务
- 同时，GitLab, GitHub 这种常用的代码托管平台也会为每个仓库在部署时提供 Registry 的管理功能



- 每一个仓库存放同一个应用的不同镜像，由 tag 标识版本
- 完整的仓库名由用户名和仓库名组成，例如 library/alpine:latest



The screenshot displays two Docker Hub image pages for MySQL. The top page is for 'mysql:oraclelinux9' and the bottom page is for 'mysql:oracle'. Both pages show a table of image digests, OS/ARCH, vulnerabilities, and compressed size.

Digest	OS/ARCH	Vulnerabilities	Compressed Size
40e3265a9899	linux/amd64	3 31 12 3 6	162.37 MB
d0a00c4c73db	linux/arm64	3 31 12 3 6	159.77 MB



谢谢大家!



清华大学
Tsinghua University

网络研究院
INSC





- [bwgg 讲义](#)
- [2022 暑培讲义](#)
- [Docker 隔离技术](#)
- [Docker 原理](#)
- [Services and Daemon](#)
- [Docker 官网](#)
- [Docker Hub 官网](#)