



《软件架构与中间件》

作业二：Docker 的安装与应用

学号：2022211939

姓名：杨涛

一、Docker 环境搭建

1、Docker 环境搭建的流程与要点

(一) 安装操作步骤

1. 前置准备

首先，服务器换源，设置同步时间，更新并升级 apt 等。

2. 安装依赖包

Docker 需要一些额外的依赖包，使用以下命令：

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

```
root@MiddleWareDocker:~# clear
root@MiddleWareDocker:~# sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
ca-certificates is already the newest version (20240205).
ca-certificates set to manually installed.
curl is already the newest version (8.9.1-2ubuntu2.2).
curl set to manually installed.
software-properties-common is already the newest version (0.102).
software-properties-common set to manually installed.
Installing:
  apt-transport-https
Summary:
  Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 0
  Download size: 3,968 B
  Space needed: 38.9 kB / 4,895 MB available
Get:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular/universe amd64 apt-transport-https all 2.9.8 [3,968 B]
Fetched 3,968 B in 0s (16.0 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 126703 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.9.8_all.deb ...
```

3. 添加 Docker 官方的 GPG 密钥

执行以下命令来下载并添加 Docker 的官方 GPG 密钥：

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
root@MiddleWareDocker:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
curl: (35) Recv failure: Connection reset by peer
gpg: no valid OpenPGP data found.
root@MiddleWareDocker:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
file /usr/share/keyrings/docker-archive-keyring.gpg exists. Overwriter (y/N) y
root@MiddleWareDocker:~#
```

4. 设置 Docker 官方的 APT 源

添加 Docker 仓库到 Ubuntu 的软件源列表：

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

```

root@MiddleWareDocker:~# clear
root@MiddleWareDocker:~# echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
root@MiddleWareDocker:~#

```

5. 更新 apt 包索引

运行以下命令更新包索引，以便 Ubuntu 可以从新的 Docker 仓库安装 Docker：
sudo apt update

```

root@MiddleWareDocker:~# sudo apt update
Ign:2 https://download.docker.com/linux/ubuntu oracular InRelease
Hit:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular InRelease
Get:3 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-updates InRelease [126 kB]
Get:4 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-backports InRelease [126 kB]
Get:5 http://security.ubuntu.com/ubuntu oracular-security InRelease [126 kB]
Get:6 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-updates/main amd64 Components [24.3 kB]
Get:7 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-updates/restricted amd64 Components [216 B]
Get:8 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-updates/universe amd64 Components [28.7 kB]
Get:9 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-updates/multiverse amd64 Components [212 B]
Get:10 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-backports/main amd64 Components [212 B]
Get:11 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-backports/restricted amd64 Components [216 B]
Get:12 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-backports/universe amd64 Components [9,696 B]
Get:13 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular-backports/multiverse amd64 Components [216 B]
Ign:2 https://download.docker.com/linux/ubuntu oracular InRelease
Get:14 http://security.ubuntu.com/ubuntu oracular-security/main amd64 Components [2,548 B]
Get:15 http://security.ubuntu.com/ubuntu oracular-security/restricted amd64 Components [212 B]
Get:16 http://security.ubuntu.com/ubuntu oracular-security/universe amd64 Components [2,464 B]
Get:17 http://security.ubuntu.com/ubuntu oracular-security/multiverse amd64 Components [212 B]
Get:2 https://download.docker.com/linux/ubuntu oracular InRelease [32.9 kB]
Get:18 https://download.docker.com/linux/ubuntu oracular/stable amd64 Packages [5,794 B]
Fetched 486 kB in 4s (124 kB/s)
All packages are up to date.
root@MiddleWareDocker:~#

```

6. 安装 Docker

使用以下命令安装 Docker：**sudo apt install docker-ce docker-ce-cli containerd.io -y**

```

root@MiddleWareDocker:~# sudo apt install docker-ce docker-ce-cli containerd.io -y
Installing:
  containerd.io  docker-ce  docker-ce-cli

Installing dependencies:
  docker-buildx-plugin  docker-ce-rootless-extras  docker-compose-plugin  libltdl7  libslirp0  pigz  slirp4netns

Suggested packages:
  aufs-tools  cgroupfs-mount  |  cgroup-lite

Summary:
  Upgrading: 0, Installing: 10, Removing: 0, Not Upgrading: 0
  Download size: 124 MB
  Space needed: 446 MB / 4,895 MB available

Get:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular/main amd64 libltdl7 amd64 2.4.7-7build1 [40.3 kB]
Get:3 https://download.docker.com/linux/ubuntu oracular/stable amd64 containerd.io amd64 1.7.24-1 [29.5 MB]
Get:4 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular/main amd64 libslirp0 amd64 4.8.0-1ubuntu1 [65.7 kB]
Get:5 http://mirrors.tuna.tsinghua.edu.cn/ubuntu oracular/universe amd64 slirp4netns amd64 1.2.1-1build2 [34.9 kB]
Get:6 https://download.docker.com/linux/ubuntu oracular/stable amd64 docker-buildx-plugin amd64 0.19.3-1-ubuntu.24.10-oracular [30.7 MB]
Get:7 https://download.docker.com/linux/ubuntu oracular/stable amd64 docker-ce-cli amd64 5:27.4.1-1-ubuntu.24.10-oracular [15.1 MB]
Get:8 https://download.docker.com/linux/ubuntu oracular/stable amd64 docker-ce amd64 5:27.4.1-1-ubuntu.24.10-oracular [25.8 MB]
70% [8 docker-ce 11.0 MB/25.8 MB 43%] 3,455 kB/s 10s

```

7. 启动 Docker 并设置开机自启

安装完成后，启动 Docker 服务并设置其在系统启动时自动启动：

```
sudo systemctl start docker
sudo systemctl enable docker
```

```
root@MiddleWareDocker:~# sudo systemctl start docker
sudo systemctl enable docker
root@MiddleWareDocker:~#
```

8. 检查 Docker 安装情况

使用以下命令检查 Docker 是否安装成功：

```
sudo docker --version
```

```
root@MiddleWareDocker:~# sudo docker --version
Docker version 27.4.1, build b9d17ea
root@MiddleWareDocker:~#
```

运行 hello-world 镜像来确认 Docker 是否正常工作：

```
root@MiddleWareDocker:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
docker: Error response from daemon: Get "https://registry-1.docker.io/v2/": net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers).
See 'docker run --help'.
root@MiddleWareDocker:~#
```

发现不能正常拉取镜像源，于是开始换源。

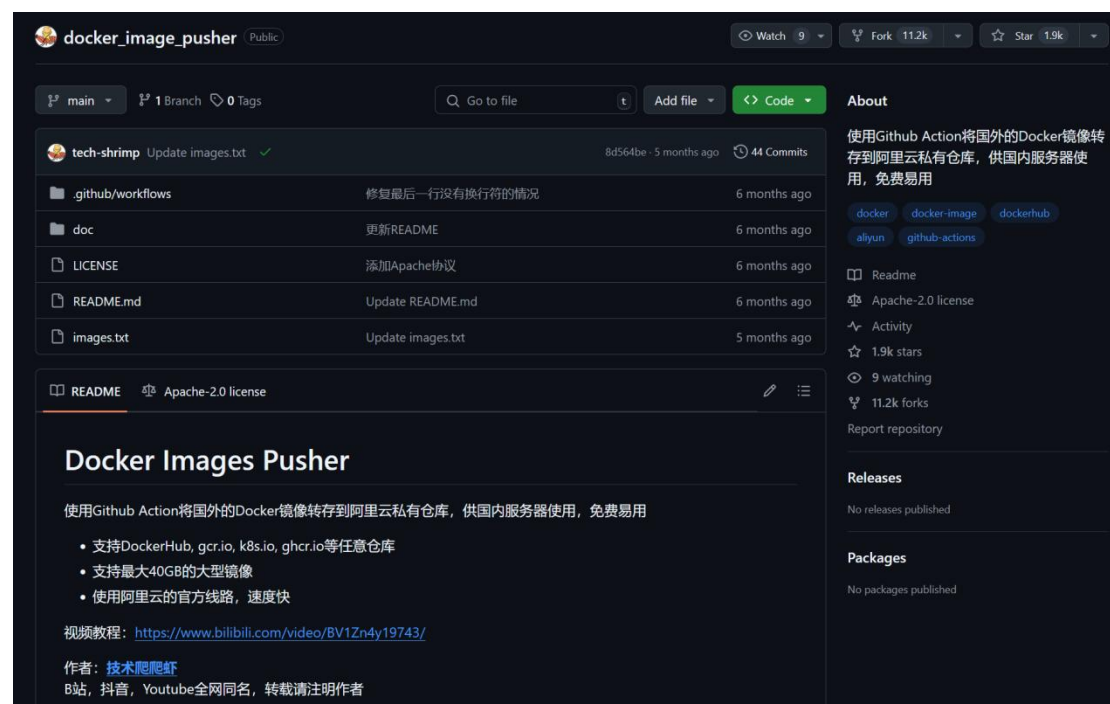
(二) 流程图



2、Docker 环境搭建的难点与问题

在安装完成后发现由于不可抗力不可以直接访问国外的 dockerhub，同时了解到国内的镜像源大部分都被下架，并且难以保持镜像源版本同步更新。于是最终选择了使用 GithubAction 和阿里云个人镜像仓库的办法。

项目地址：https://github.com/yangtaoytt/docker_image_pusher



具体操作过程如下：

1. 登录阿里云的容器镜像服务，选择创建个人实例



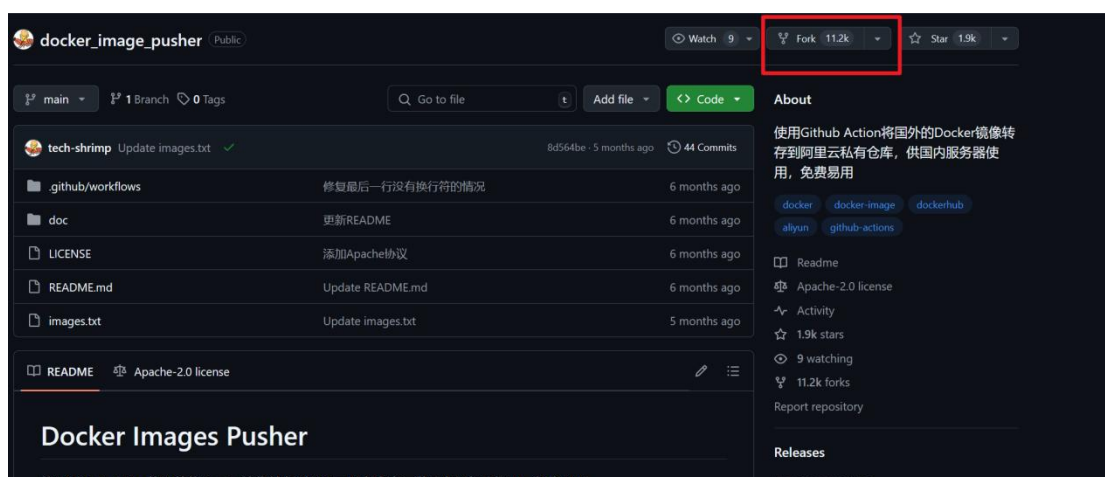
2. 新建一个命名空间，设置登陆用户名和密码。



3. 点击凭据管理，记录下凭证信息。

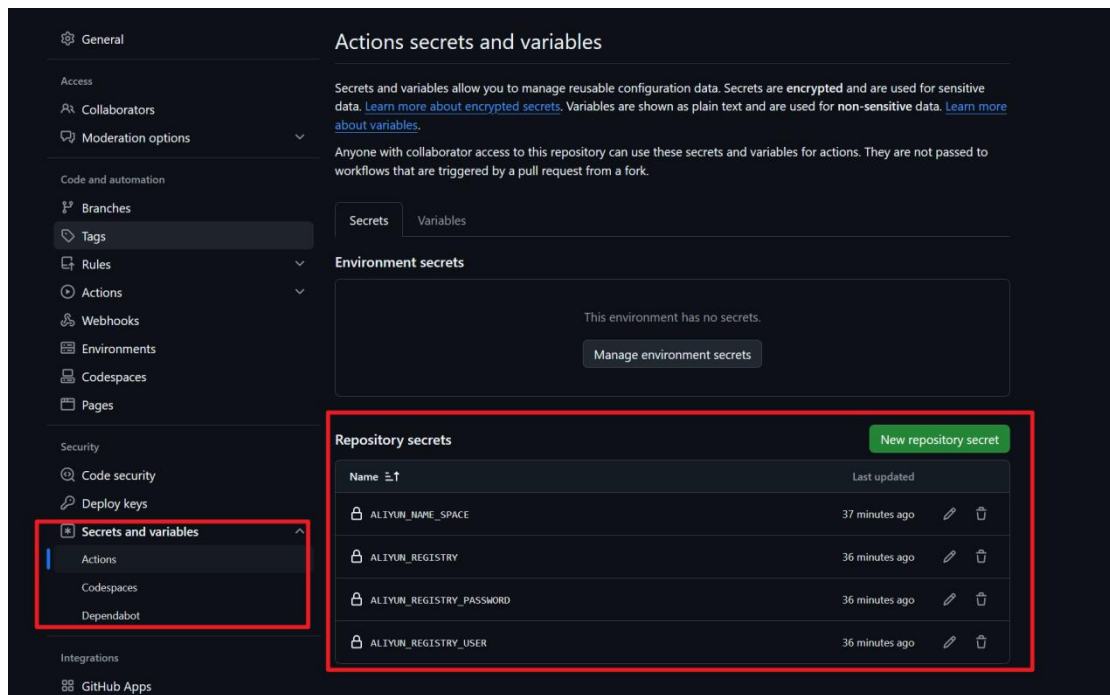


4. 进入项目主页，选择 fork 项目。

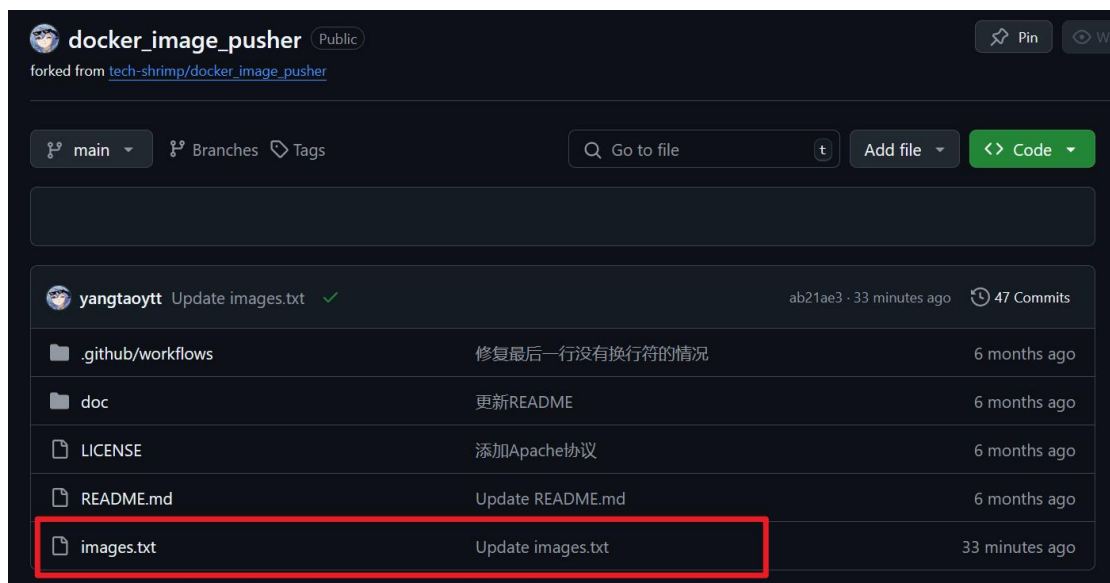


5. 在项目设置里的 Action 处依次填入环境变量：

用户名（ALIYUN_REGISTRY_USER）
密码（ALIYUN_REGISTRY_PASSWORD）
仓库地址（ALIYUN_REGISTRY）
命名空间（ALIYUN_NAME_SPACE）

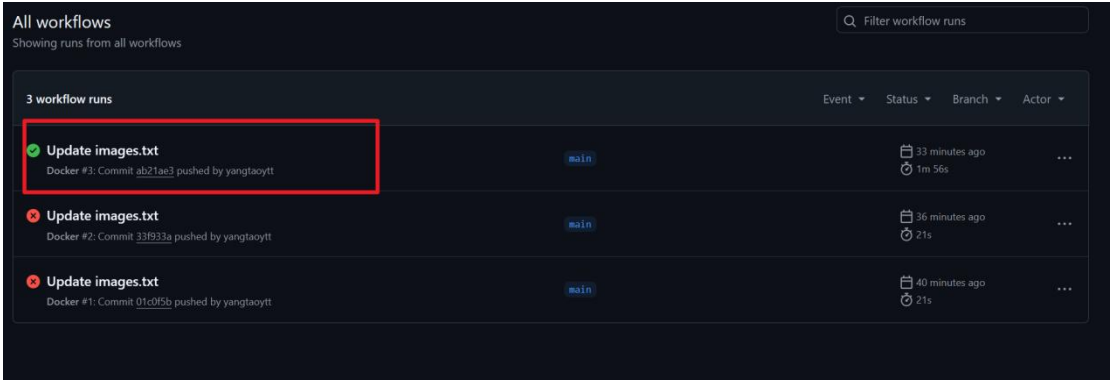


6. 修改 fork 后项目的 images.txt，加入想要的镜像。

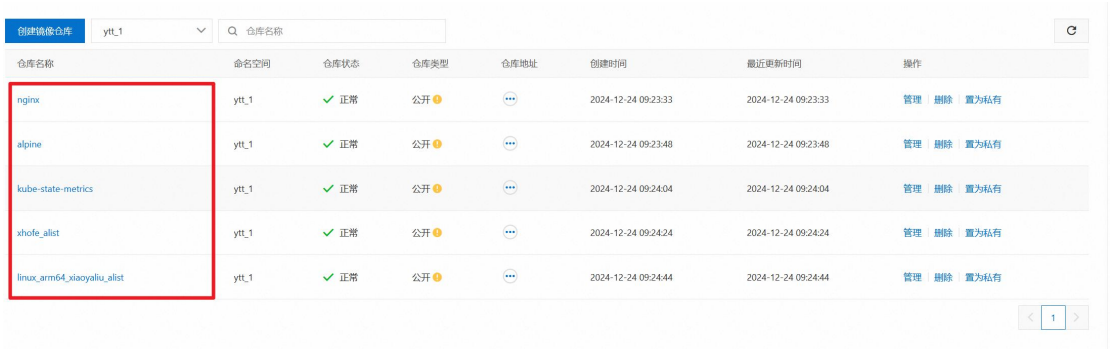


```
nginx
alpine
#支持私库
k8s.gcr.io/kube-state-metrics/kube-state-metrics:v2.0.0
xhofs/alist:latest
#支持指定架构
--platform=linux/arm64 xiaoyaliu/alist
#dgg
```


7. 等待 Action 运行成功



8. 回到阿里云镜像仓库，即可看到保存的镜像。



二、Docker 容器的启动与卸载

1、Docker 容器的启动与卸载的流程与要点

(一) 启动卸载操作步骤

1. 查看当前运行的容器：

sudo docker ps

```
root@MiddleWareDocker:~# sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
root@MiddleWareDocker:~#
```

2. 查看所有容器（包括已停止的容器）：

sudo docker ps -a

```
root@MiddleWareDocker:~# sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

3. 从 Registry 中拉取 ubuntu 镜像：

docker pull crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu (阿里云个人仓库)

```
root@MiddleWareDocker:~# docker pull crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu
Using default tag: latest
latest: Pulling from ytt_1/ubuntu
54609b48ebc1: Pull complete
Digest: sha256:b0c08a4b639b5fca9aa4943ecec614fe241a0cebd1a7b460093ccaeae70df698
Status: Downloaded newer image for crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu:latest
crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu:latest
root@MiddleWareDocker:~#
```

4. 启动一个简单的 Ubuntu 容器：

sudo docker run -it crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu /bin/bash

```
root@MiddleWareDocker:~# clear
root@MiddleWareDocker:~# sudo docker run -it crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu /bin/bash
root@8694e1de20e0:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@8694e1de20e0:/# clear
root@8694e1de20e0:/# init
bash: init: command not found
root@8694e1de20e0:/# init 0
bash: init: command not found
root@8694e1de20e0:/# exit
exit
root@MiddleWareDocker:~#
```

5. 后台启动容器:

```
sudo docker run -d --name mycontainer
```

```
crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu sleep 1000
```

```
root@MiddlewareDocker:~#  
root@MiddlewareDocker:~# sudo docker run -d --name mycontainer crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/ubuntu sleep 1000  
e0d07f22/cb51a74d2e57b24ab/1/T0bca8/839b67ec40b9e3/ceb/Te2a4Tb9/  
root@MiddlewareDocker:~#
```

6. 拉取 nginx 镜像:

```
docker pull crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/nginx
```

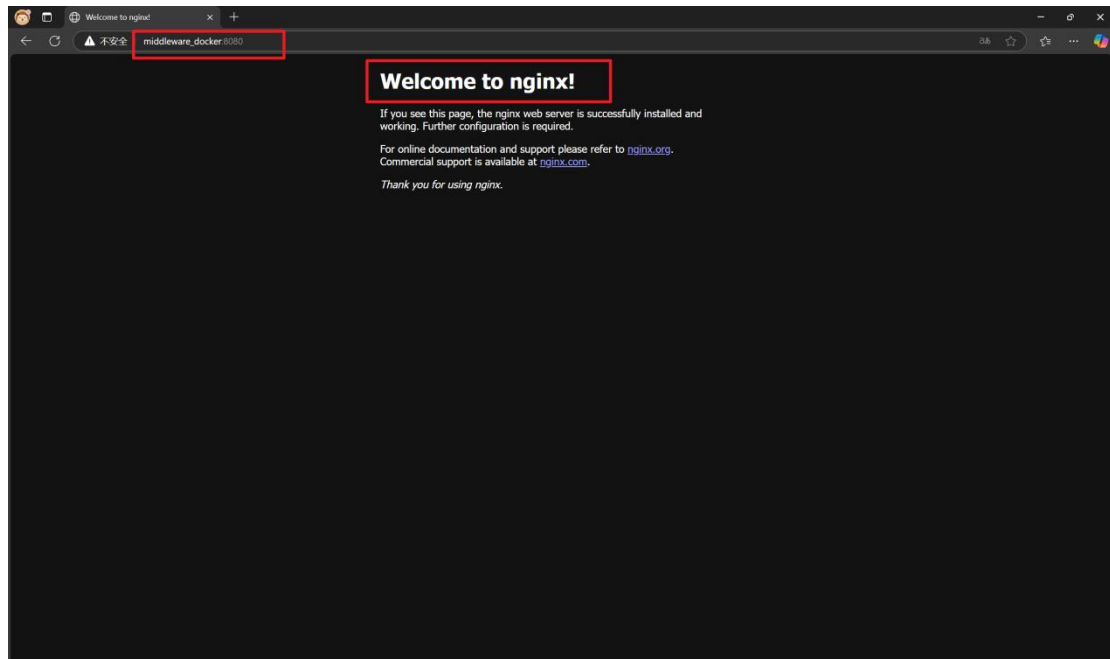
```
root@MiddlewareDocker:~# clear  
root@MiddlewareDocker:~# docker pull crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/nginx  
Using default tag: latest  
latest: Pulling from ytt_1/nginx  
8cf9fb7a0b56: Downloading [=====>] 11.48MB/29.15MB  
650ee30bbe5e: Downloading [=====>] 12.15MB/43.84MB  
8cc1569e58f5: Download complete  
362f35df001b: Download complete  
13e320bf29cd: Download complete  
7b50399908e1: Download complete  
57b64962dd94: Download complete
```

7. 运行一个 Web 服务器 (如 Nginx) , 并将容器的端口 80 映射到主机的端口 8080:

```
sudo docker run -d -p 8080:80 --name mynginx
```

```
crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/nginx
```

```
root@MiddlewareDocker:~#  
root@MiddlewareDocker:~# sudo docker run -d -p 8080:80 --name mynginx crpi-g8iob0oj8g4u8sia.cn-qingdao.personal.cr.aliyuncs.com/ytt_1/nginx  
1a5c1e9c6157b7dfcbcd516edd48ffa4b79b2d05e55242df4e33c6fcb8a8ea3  
root@MiddlewareDocker:~#
```



8. 进入正在运行的容器:

`sudo docker exec -it mycontainer /bin/bash`

```
root@MiddleWareDocker:~#  
root@MiddleWareDocker:~# sudo docker exec -it mycontainer /bin/bash  
root@e0d7df2275b5:~# ll  
total 56  
drwxr-xr-x  1 root root 4096 Dec 24 02:08 ./  
drwxr-xr-x  1 root root 4096 Dec 24 02:08 ../  
-rwxr-xr-x  1 root root   0 Dec 24 02:08 .dockerenv*  
lrwxrwxrwx  1 root root   7 Apr 22 2024 bin -> usr/bin/  
drwxr-xr-x  2 root root 4096 Apr 22 2024 boot/  
drwxr-xr-x  5 root root 340 Dec 24 02:08 dev/  
drwxr-xr-x  1 root root 4096 Dec 24 02:08 etc/  
drwxr-xr-x  3 root root 4096 Nov 19 09:52 home/  
lrwxrwxrwx  1 root root   7 Apr 22 2024 lib -> usr/lib/  
lrwxrwxrwx  1 root root   9 Apr 22 2024 lib64 -> usr/lib64/  
drwxr-xr-x  2 root root 4096 Nov 19 09:46 media/  
drwxr-xr-x  2 root root 4096 Nov 19 09:46 mnt/  
drwxr-xr-x  2 root root 4096 Nov 19 09:46 opt/  
dr-xr-xr-x 314 root root   0 Dec 24 02:08 proc/  
drwx----- 2 root root 4096 Nov 19 09:52 root/
```

9. 停止运行的容器:

`sudo docker stop mycontainer`

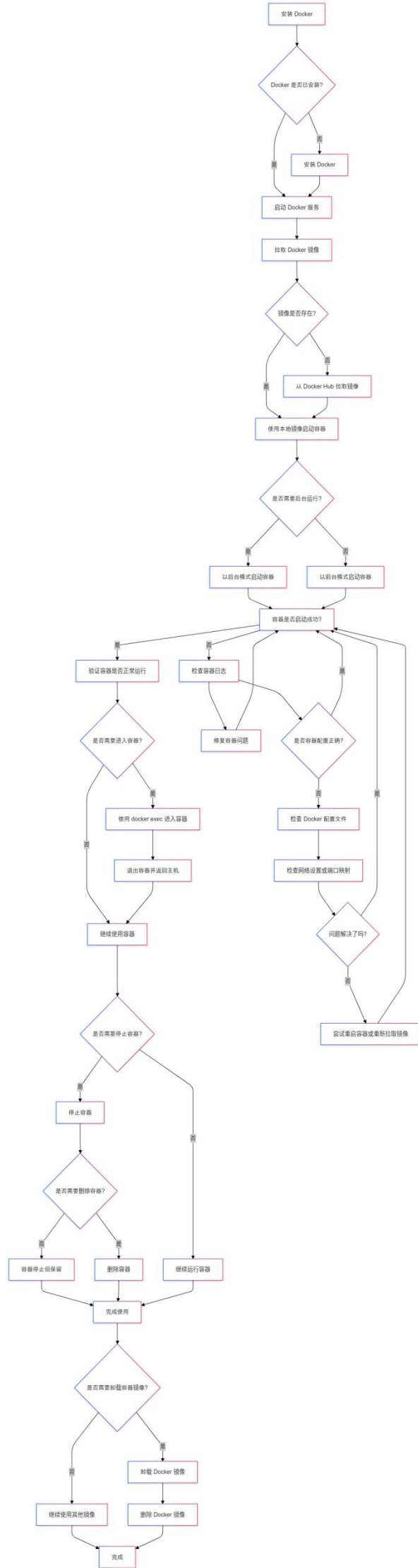
```
root@MiddleWareDocker:~# sudo docker stop mycontainer  
mycontainer  
root@MiddleWareDocker:~#
```

10. 删除容器:

sudo docker rm mycontainer

```
root@MiddleWareDocker:~#  
root@MiddleWareDocker:~# sudo docker rm mycontainer  
mycontainer  
root@MiddleWareDocker:~#
```

(二) 流程图



2、Docker 容器的启动与卸载的难点与问题

（按照自身实践的经验，总结 Docker 容器的启动与卸载过程中容易犯错的点，描述实践过程中遇到的难题及解决的方法。）

1. 端口冲突

启动容器时，常常会遇到端口冲突的问题。例如，容器内的应用需要使用一些常见的端口（比如 80 或 3306），但这些端口可能已经被宿主机上的其他程序占用了。这样，容器就无法正常启动。

解决方法： 只要稍微调整一下端口映射就能解决这个问题。如果不想使用默认端口，可以把容器内部的端口映射到宿主机的其他端口。

2. 镜像拉取失败

有时候，拉取 Docker 镜像的时候会遇到失败，通常是网络不稳定或 Docker Hub 访问问题。错误信息可能是 `network timeout` 或 `repository not found`。

解决方法： 在国内使用 Docker 时，拉取镜像会比较慢，甚至失败。这个时候，配置一个镜像加速器（比如阿里云的加速器）会大大提升速度，避免拉取失败。

3. 挂载卷的权限问题

启动容器时，如果你挂载了宿主机的目录到容器中，有时会遇到权限问题。比如容器中的应用可能没有权限写入宿主机目录，这会导致容器启动失败或功能异常。

解决方法： 最好先检查一下宿主机的目录权限，确保容器能正常访问这些目录。我通常会设置较宽松的权限（比如 777），保证容器可以读写。

三、基于 Docker 的应用部署

1. 制作 dockerfile

(1) 项目介绍

是一个我参与制作的单机关系型数据库，可以分为前端和后端两部分。

前端负责提供一个 web 接口，可以在 web 浏览器直接向后端数据库发请求。

(2) 依赖

①前端依赖

前端依赖主要是与 Node.js 和 npm 相关。以下是关键部分：

1. **Node.js 18.x:** 安装 Node.js 18.x 版本来支持前端的 JavaScript 构建工具和相关服务。Node.js 的安装包括了 npm 的安装，npm 是一个流行的前端包管理工具。
2. **npm 镜像配置:** 为了提高速度，配置 npm 使用淘宝镜像源。

②后端依赖

后端依赖主要是与环境配置和运行有关的组件，具体包括：

1. **Python 3:** 安装了 Python3 及其虚拟环境工具 (python3-venv) 和 setuptools，支持后端 Python 脚本和应用程序的运行。
2. **Git:** 用于克隆远程仓库，获取需要的脚本或源代码。
3. **make 和 build-essential:** 安装编译工具和基础库，支持源代码的构建和安装。
4. **Wetty:** 一个 Web 终端，提供了对后端系统的访问接口。

(3) Dockerfile 介绍

①基础镜像选择

```
FROM ubuntu:24.10
```

选择 Ubuntu 24.10 作为基础镜像，它是一个稳定且广泛使用的操作系统版本，适合构建开发环境。

②更新与安装基础工具

```
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y curl
RUN apt-get install -y gnupg2
RUN apt-get install -y lsb-release
RUN apt-get install -y build-essential
RUN apt-get install -y make
RUN apt-get install -y python3
```

```
RUN apt-get install -y python3-venv python3-setuptools
```

1. **更新包列表:** 使用 `apt-get update` 确保镜像中的包信息是最新的。
2. **升级已安装包:** 通过 `apt-get upgrade` 升级所有现有的软件包。
3. **安装常用工具:** 安装 `curl`、`gnupg2`、`lsb-release`、`build-essential`、`make` 等常见开发工具以及 `Python3` 相关依赖。

③安装 Node.js 和配置 npm

```
RUN curl -fsSL https://deb.nodesource.com/setup_18.x | bash -  
RUN apt-get install -y nodejs  
RUN npm config set registry https://registry.npmmirror.com
```

1. **安装 Node.js:** 设置 Node.js 18.x 的安装仓库，并安装 Node.js。
2. **配置 npm 镜像:** 更换 npm 镜像为淘宝源，提升依赖下载速度，尤其是在中国地区。

④安装 Wetty

```
RUN npm install -g wetty@2.5
```

通过 npm 安装 Wetty，一个可以通过浏览器访问的终端程序。它允许用户通过 Web 界面连接到容器的命令行环境，增强交互性。

⑤克隆仓库并配置 Bustub 环境

```
RUN apt-get install -y git dos2unix  
RUN git clone https://gitee.com/yangtaoytt/bustub-shell-preparation.git /opt/bustub-shell-preparation  
WORKDIR /opt/bustub-shell-preparation  
RUN dos2unix packages.sh  
RUN chmod +x packages.sh && ./packages.sh
```

1. **Git 克隆:** 克隆指定的 Git 仓库，获取 Bustub 环境的相关脚本和代码。
2. **文件格式转换:** 使用 `dos2unix` 转换 Windows 格式的脚本文件为 Unix 格式，确保脚本在 Linux 环境下正常执行。
3. **运行环境配置脚本:** 运行 `packages.sh` 配置必要的环境依赖。

⑥配置执行文件

```
RUN chmod +x bustub-shell && mv bustub-shell /usr/local/bin/bustub-shell
```

设置可执行文件权限，并将其移动到系统的 `/usr/local/bin` 路径，使其可以全局访问。

⑦创建特定用户及执行脚本

```
RUN echo '#!/bin/bash\n/usr/local/bin/bustub-shell\nexit' > /usr/local/bin/limited_shell.sh  
RUN chmod +x /usr/local/bin/limited_shell.sh  
RUN useradd -m specialuser && echo "specialuser:114514" | chpasswd
```

```
RUN usermod -s /usr/local/bin/limited_shell.sh specialuser
```

1. **创建用户**: 创建一个新用户 `specialuser`, 并设置密码。
2. **自定义 Shell**: 将用户的默认 `shell` 设置为 `limited_shell.sh`, 此脚本会在用户登录时启动指定的环境。

⑧配置容器启动命令

```
ENTRYPOINT ["wetty", "--host", "0.0.0.0", "--port", "3000"]
```

配置容器启动时自动运行 `Wetty`, 并监听在端口 `3000` 上, 允许通过浏览器访问终端。

(4) 遇到的问题 (部分)

①问题一: wetty 版本问题

`Wetty` 最新版为 `2.7.0`, 通过 `npm` 安装, 但是最新版 `wetty` 在部分系统上会出现黑屏卡住崩溃的现象。

解决办法为使用 `2.5.0` 版本的 `wetty` 代替。

②问题二: ubuntu python-distutils 问题

`python-distutils` 是 `npm` 安装所需 `py` 环境之一, 但是在 `ubuntu22` 以上版本被默认移除, 尤其是 `docker` 的镜像版本中, 我测试时使用的虚拟机是自带 `py3` 环境的, 因此没有注意。在 `dockerbuild` 的时候就出了问题, 并且这个包已经不被 `apt` 支持了, 最后只能使用 `python-evn`, 其中会携带这个包来代替。

2. 在一个配置好 docker 服务的主机上上传 dockerfile, 并在相同

目录下执行 docker 构建命令

```
docker build -t bustub_shell_web .
```

```
root@MiddlewareDocker:/dockerfile/bustub_shell_web# docker build -t bustub_shell_web .
[+] Building 72.0s (4/22)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfiles: 3.76kB
=> [internal] load metadata for docker.io/library/ubuntu:22.10
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/26] FROM docker.io/library/ubuntu:22.10@sha256:102bc1874fdb36fc2d218c7e7f09cf8a35d67486fcd8a026c0f558cfe1b6d
=> => resolve docker.io/library/ubuntu:22.10@sha256:102bc1874fdb36fc2d218c7e7f09cf8a35d67486fcd8a026c0f558cfe1b6d
=> => sha256:102bc1874fdb36fc2d218c7e7f09cf8a35d67486fcd8a026c0f558cfe1b6d 6.69kB / 6.69kB
=> => sha256:1ed5bc722518d87179cf29291d63815d77e07ef74e31a65256ef18d6523c0d 424B / 424B
=> => sha256:c40b6c1b3d3bc64c467e7e2163b1f688383b364276117d3c4b6c2d4f83d 2.84kB / 3.90kB
=> => sha256:11724b16291417d5f3676d1e364912c691c76d51e16522c548af31f98618 80.50kB / 80.50kB
=> => extracting sha256:83974d1936b14b7d6f55f0d11d69c942cc61ac79d391d5525c5d8f7b1f8819 2.75
=> [2/26] RUN apt-get update
=> => # Get:7 http://security.ubuntu.com/ubuntu oracular-security/restricted amd64 Packages [86.6 kB]
=> => # Get:8 http://archive.ubuntu.com/ubuntu oracular-backports InRelease [126 kB]
=> => # Get:9 http://archive.ubuntu.com/ubuntu oracular/multiverse amd64 Packages [308 kB]
=> => # Get:10 http://archive.ubuntu.com/ubuntu oracular/main amd64 Packages [1835 kB]
=> => # Get:11 http://archive.ubuntu.com/ubuntu oracular/restricted amd64 Packages [67.0 kB]
=> => # Get:12 http://archive.ubuntu.com/ubuntu oracular/universe amd64 Packages [19.6 MB]
```



```

=> [ 3/26] RUN apt-get upgrade -y 5.6s
=> [ 4/26] RUN apt-get install -y curl 15.9s
=> [ 5/26] RUN apt-get install -y gnupg2 9.8s
=> [ 6/26] RUN apt-get install -y lsb-release 4.0s
=> [ 7/26] RUN apt-get install -y build-essential 120.8s
=> [ 8/26] RUN apt-get install -y make 2.8s
=> [ 9/26] RUN apt-get install -y python3 13.4s
=> [10/26] RUN apt-get install -y python3-venv python3-setuptools 12.7s
=> [11/26] RUN curl -fsSL https://deb.nodesource.com/setup_16.x | bash - 12.1s
=> [12/26] RUN apt-get install -y nodejs 21.3s
=> [13/26] RUN npm config set registry https://registry.npmjs.com 1.6s
=> [14/26] RUN npm install -g wetty@2.5 52.4s
=> [15/26] WORKDIR /root 0.0s
=> [16/26] RUN apt-get install -y git dos2unix 10.0s
=> [17/26] RUN git clone https://github.com/yanptaoytt/bustub-shell-preparation.git /opt/bustub-shell-preparation 10.3s
=> [18/26] WORKDIR /opt/bustub-shell-preparation 0.0s
=> [19/26] RUN dos2unix packages.sh 0.4s
=> [20/26] RUN chmod +x packages.sh && ./packages.sh 103.9s
=> [21/26] RUN chmod +x bustub-shell && mv bustub-shell /usr/local/bin/bustub-shell 1.0s
=> [22/26] RUN echo '#!/bin/bash\n/usr/local/bin/bustub-shell\nexit' > /usr/local/bin/limited_shell.sh 0.9s
=> [23/26] RUN chmod +x /usr/local/bin/limited_shell.sh 1.3s
=> [24/26] RUN useradd -m specialuser && echo "specialuser:il45i4" | chpasswd 1.5s
=> [25/26] RUN usermod -s /usr/local/bin/limited_shell.sh specialuser 1.1s
=> [26/26] WORKDIR /home/specialuser 0.1s
=> exporting to image 16.0s
=> exporting layers 15.9s
=> writing image sha256:159de312e5a2cf0e1a7fe580f60aa96932107a503c6de5b417d31bce7cd8e36d 0.0s
=> naming to docker.io/library/bustub_shell_web 0.0s
root@MiddlewareDocker:/dockerfile/bustub_shell_web#

```

3. 构建完成后查看 docker 镜像是否存在

docker images

```

root@MiddlewareDocker:/dockerfile/bustub_shell_web# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
bustub_shell_web latest 159de312e5a2 About a minute ago 1.85GB
root@MiddlewareDocker:/dockerfile/bustub_shell_web#

```

4. 启动 docker 镜像，并将主机 80 端口映射到 3000

docker run -d -p 80:3000 bustub_shell_web

```

root@MiddlewareDocker:/dockerfile/bustub_shell_web# docker run -d -p 80:3000 bustub_shell_web
b4bf0d561d15/ac05a1e1002b9018e3770ce209743c192460380a9ec86a9
root@MiddlewareDocker:/dockerfile/bustub_shell_web#

```

5. 查看容器是否正常运行

docker ps

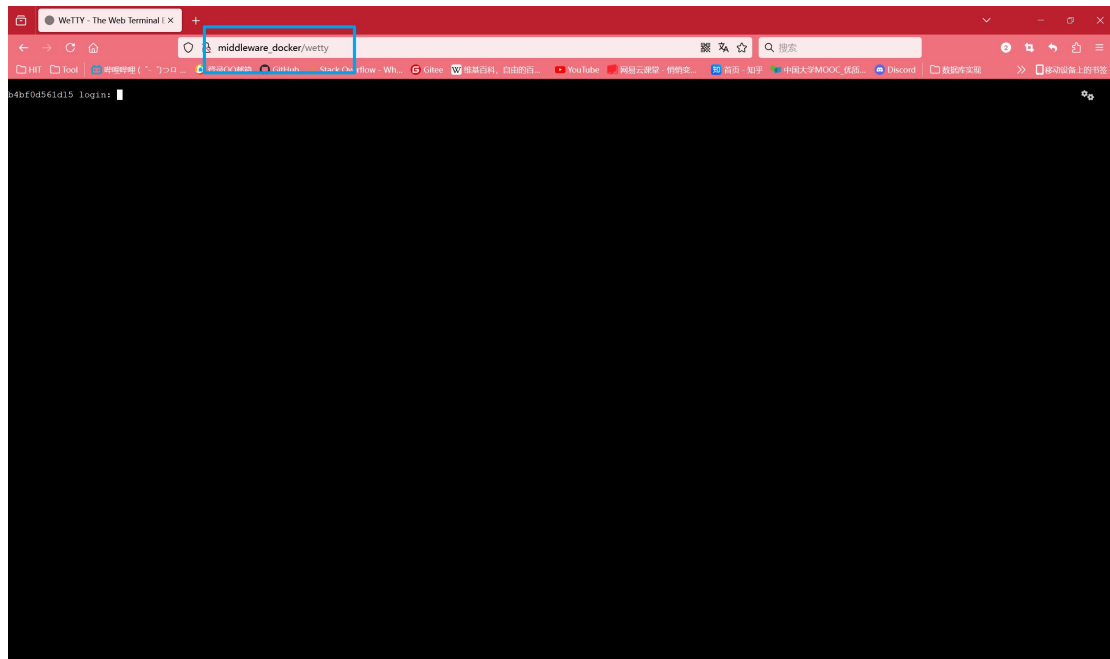
```

root@MiddlewareDocker:/dockerfile/bustub_shell_web# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b4bf0d561d15 bustub_shell_web "wetty --host 0.0.0.0..." 29 seconds ago Up 28 seconds 0.0.0.0:80->3000/tcp, [::]:80->3000/tcp pensive_hofstadter
root@MiddlewareDocker:/dockerfile/bustub_shell_web#

```

6. 在其他主机上访问运行 docker 的主机

浏览器访问 http://middleware_docker/wetty



7. 输入预先设定的账号密码登录系统

账号 specialuser

密码 114514

```
b4bf0d561d15 login: specialuser
Password:
Welcome to Ubuntu 24.10 (GNU/Linux 6.11.0-13-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Welcome to the BusTub shell! Type \help to learn more.
bustub>
```

登陆成功

8. 测试数据库是否正常工作

(1) 内置命令

\help

```

bustub> \help
Welcome to the Bustub shell!

\dt: show all tables
\di: show all indices
\dbgmvc <table>: show version chain of a table
\help: show this message again
\txn: show current txn information
\txn <txn_id>: switch to txn
\txn gc: run garbage collection
\txn -1: exit txn mode

Bustub shell currently only supports a small set of Postgres queries. We'll set
up a doc describing the current status later. It will silently ignore some parts
of the query, so it's normal that you'll get a wrong result when executing
unsupported SQL queries. This shell will be able to run 'create table' only
after you have completed the buffer pool manager. It will be able to execute SQL
queries after you have implemented necessary query executors. Use 'explain' to
see the execution plan of your query.

bustub>

```

\dt

```

bustub> \dt
+-----+-----+-----+
| oid | name | cols |
+-----+-----+-----+
| 23 | test_2 | (colA:INTEGER, colB:INTEGER, colC:INTEGER) |
| 21 | test_simple_seq_2 | (col1:INTEGER, col2:INTEGER) |
| 19 | empty table | (colA:INTEGER) |
| 17 | mock_t8 | (v4:INTEGER) |
| 16 | mock_t7 | (v:INTEGER, v1:INTEGER, v2:INTEGER) |
| 15 | mock_t6_lm | (x:INTEGER, y:INTEGER) |
| 13 | mock_t4_lm | (x:INTEGER, y:INTEGER) |
| 8 | mock_table_schedule_2022 | (day_of_week:VARCHAR, has_lecture:INTEGER) |
| 22 | test_1 | (colA:INTEGER, colB:INTEGER, colC:INTEGER, colD:INTEGER) |
| 18 | mock_t9 | (x:INTEGER, y:INTEGER) |
| 11 | mock_graph | (src:INTEGER, dst:INTEGER, src_label:VARCHAR, dst_label:VARCHAR, distance:INTEGER) |
| 1 | mock_table_2 | (colC:VARCHAR, colD:VARCHAR) |
| 20 | test_simple_seq_1 | (col1:INTEGER) |
| 6 | mock_agg_input_small | (v1:INTEGER, v2:INTEGER, v3:INTEGER, v4:INTEGER, v5:INTEGER, v6:VARCHAR) |
| 2 | mock_table_3 | (colE:INTEGER, colF:VARCHAR) |
| 14 | mock_t5_lm | (x:INTEGER, y:INTEGER) |
| 3 | mock_table_tas_2022 | (github_id:VARCHAR, office_hour:VARCHAR) |
| 4 | mock_table_tas_2023 | (github_id:VARCHAR, office_hour:VARCHAR) |
| 0 | mock_table_1 | (colA:INTEGER, colB:INTEGER) |
| 5 | mock_table_tas_2023_fall | (github_id:VARCHAR, office_hour:VARCHAR) |
| 7 | mock_agg_input_big | (v1:INTEGER, v2:INTEGER, v3:INTEGER, v4:INTEGER, v5:INTEGER, v6:VARCHAR) |
| 9 | mock_table_schedule_2023 | (day_of_week:VARCHAR, has_lecture:INTEGER) |
| 10 | mock_table_123 | (number:INTEGER) |
| 12 | mock_t1 | (x:INTEGER, y:INTEGER, z:INTEGER) |
+-----+-----+-----+
bustub>

```

(2) 创建和插入表

①创建表

```

bustub> CREATE TABLE t1 (
...     v1 INT,
...     v2 INT,
...     v3 VARCHAR(128)
... );
Table created with id = 24

```

②插入数据

```

bustub> INSERT INTO t1 VALUES (1, 2, '😄'), (3, 4, '🌟'), (5, 6, '👉');
+-----+-----+-----+
| __bustub_internal.insert_rows |
+-----+-----+-----+
| 3 |
+-----+-----+-----+

```

(3) 查询操作

①查询表

```
bustub> SELECT * FROM t1;
+-----+-----+-----+
| t1.v1 | t1.v2 | t1.v3 |
+-----+-----+-----+
| 1     | 2     | 😊     |
| 3     | 4     | 🌞     |
| 5     | 6     | 🍕     |
+-----+-----+-----+
```

②查询特定列和条件

```
bustub> SELECT v1, v2 FROM t1 WHERE v3 = '😊';
+-----+-----+
| t1.v1 | t1.v2 |
+-----+-----+
| 1     | 2     |
+-----+-----+
```

③聚合查询：按 v3 分组并计算 v1 的个数

```
bustub> SELECT v3, COUNT(v1) FROM t1 GROUP BY v3;
+-----+-----+
| t1.v3 | <unnamed> |
+-----+-----+
| 🍕     | 1         |
| 🌞     | 1         |
| 😊     | 1         |
+-----+-----+
```

④使用 HAVING 筛选聚合结果：求 v1 的和大于 4

```
bustub> SELECT v3, SUM(v1) FROM t1 GROUP BY v3 HAVING SUM(v1) > 4;
+-----+-----+
| t1.v3 | <unnamed> |
+-----+-----+
| 🍕     | 5         |
+-----+-----+
```

(4) 更新和删除操作

①更新操作：将 v1 为 3 的记录的 v3 更新

```
bustub> UPDATE t1 SET v3 = '🌈' WHERE v1 = 3;
+-----+-----+
| __bustub_internal.update_rows |
+-----+-----+
| 1                               |
+-----+-----+
```

②删除操作：删除 v2 大于 3 的记录

(5) 创建索引和查询优化

①创建索引

```
bustub> DELETE FROM t1 WHERE v2 > 3;
+-----+
| __bustub_internal.delete_rows |
+-----+
| 2                               |
+-----+
```

②使用索引查询

```
bustub> CREATE INDEX idx_v1 ON t1(v1);
Index created with id = 0
```

(6) JOIN 操作

```
bustub> SELECT * FROM t1 WHERE v1 = 3;
+-----+-----+-----+
| t1.v1 | t1.v2 | t1.v3 |
+-----+-----+-----+
```

①创建第二个表 t2

```
bustub> CREATE TABLE t2 (
...     v1 INT,
...     v4 VARCHAR(128)
... );
Table created with id = 25
```

②插入数据到 t2

```
bustub> INSERT INTO t2 VALUES (1, '🍏'), (3, '🌻'), (5, '🍓');
+-----+
| __bustub_internal.insert_rows |
+-----+
| 3                               |
+-----+
```

③INNER JOIN 操作

```
bustub> SELECT t1.v1, t1.v3, t2.v4
... FROM t1 INNER JOIN t2 ON t1.v1 = t2.v1;
+-----+-----+-----+
| t1.v1 | t1.v3 | t2.v4 |
+-----+-----+-----+
| 1     | 🍏    | 🍏    |
| 3     | 🌻    | 🌻    |
| 5     | 🍓    | 🍓    |
+-----+-----+-----+
```

④LEFT OUTER JOIN 操作

```
bustub> SELECT t1.v1, t1.v3, t2.v4
... FROM t1 LEFT OUTER JOIN t2 ON t1.v1 = t2.v1;
+-----+-----+-----+
| t1.v1 | t1.v3 | t2.v4 |
+-----+-----+-----+
| 1     | 🍏    | 🍏    |
| 3     | 🌻    | 🌻    |
| 5     | 🍓    | 🍓    |
+-----+-----+-----+
```


(7) 排序和限制结果

①排序操作：按 v1 降序排列

```
bustub> SELECT * FROM t1 ORDER BY v1 DESC;
+-----+-----+-----+
| t1.v1 | t1.v2 | t1.v3 |
+-----+-----+-----+
| 5     | 6     | 🍕    |
| 3     | 4     | ☀️    |
| 1     | 2     | 😊    |
+-----+-----+-----+
```

②限制查询结果：只返回前 2 条记录

```
bustub> SELECT * FROM t1 LIMIT 2;
+-----+-----+-----+
| t1.v1 | t1.v2 | t1.v3 |
+-----+-----+-----+
| 1     | 2     | 😊    |
| 3     | 4     | ☀️    |
+-----+-----+-----+
```

(8) 聚合函数与窗口函数

①聚合函数：求最大值、最小值

```
bustub> SELECT MAX(v1), MIN(v2) FROM t1;
+-----+-----+
| <unnamed> | <unnamed> |
+-----+-----+
| 5         | 2         |
+-----+-----+
```

②窗口函数：计算每行的累计和

```
bustub> SELECT v1, SUM(v1) OVER (ORDER BY v1) AS cumulative_sum FROM t1;
+-----+-----+
| t1.v1 | cumulative_sum |
+-----+-----+
| 1     | 1              |
| 3     | 4              |
| 5     | 9              |
+-----+-----+
```

③分区窗口函数：按 v3 分组计算 v1 的最小值

```
bustub> SELECT v1, MIN(v1) OVER (PARTITION BY v3) AS avg_v1 FROM t1;
+-----+-----+
| t1.v1 | avg_v1 |
+-----+-----+
| 1     | 1      |
| 3     | 3      |
| 5     | 5      |
+-----+-----+
```