《软件架构与中间件》

作业二：Docker 的安装与应用

学号：2022211917

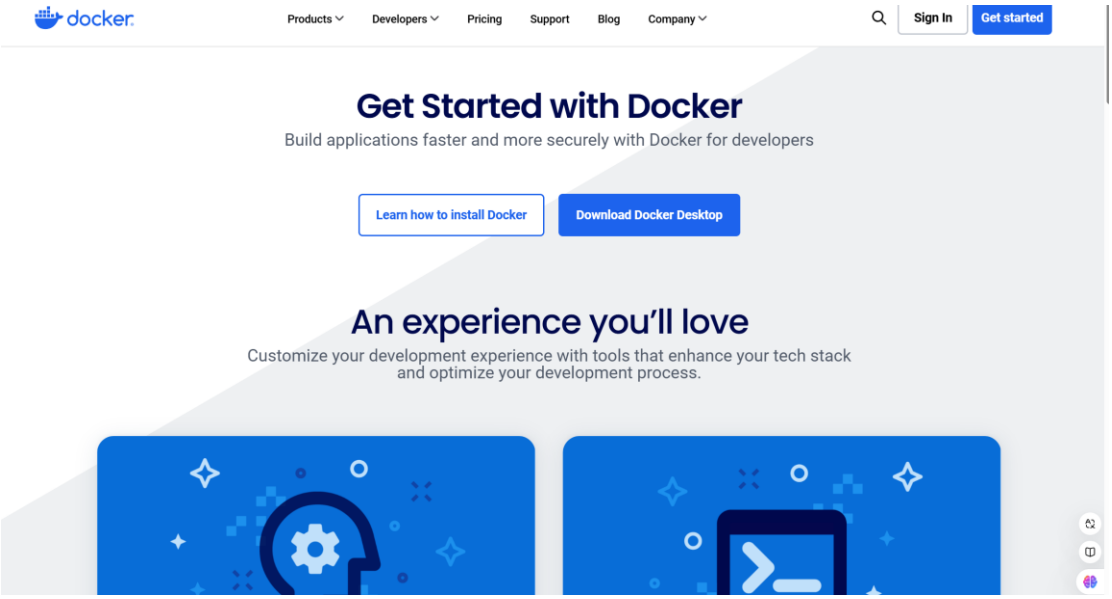
姓名：周雨凡

一、Docker 环境搭建

1、Docker 环境搭建的流程与要点

a. 下载和安装 Docker Desktop

- 访问官网并下载：
  - 打开 Docker 官方网站，点击 "Download Docker Desktop for Windows" 链接下载适用于 Windows 的安装包。
  - 注意：确保下载版本符合您的操作系统要求。



- 安装 Docker Desktop：
  - 双击安装包后，根据提示完成安装。安装过程中，确保勾选了“Enable WSL 2”选项。
  - 如果未启用 Hyper-V 和 虚拟化，安装程序会提示并要求您启用这些功能。可以选择自动启用，或者在 Windows 中手动设置。

安装过程中可能遇到的常见问题：

- WSL 2 安装失败：如果安装过程中 WSL 2 未能正确配置，系统可能会要求您手动安装或启用相关功能。
- 系统重启：安装过程中，Docker 可能需要重启系统来启用 Hyper-V 和虚拟化支持，务必确保重启完成。

b. 启用 Windows 的虚拟化和 Hyper-V

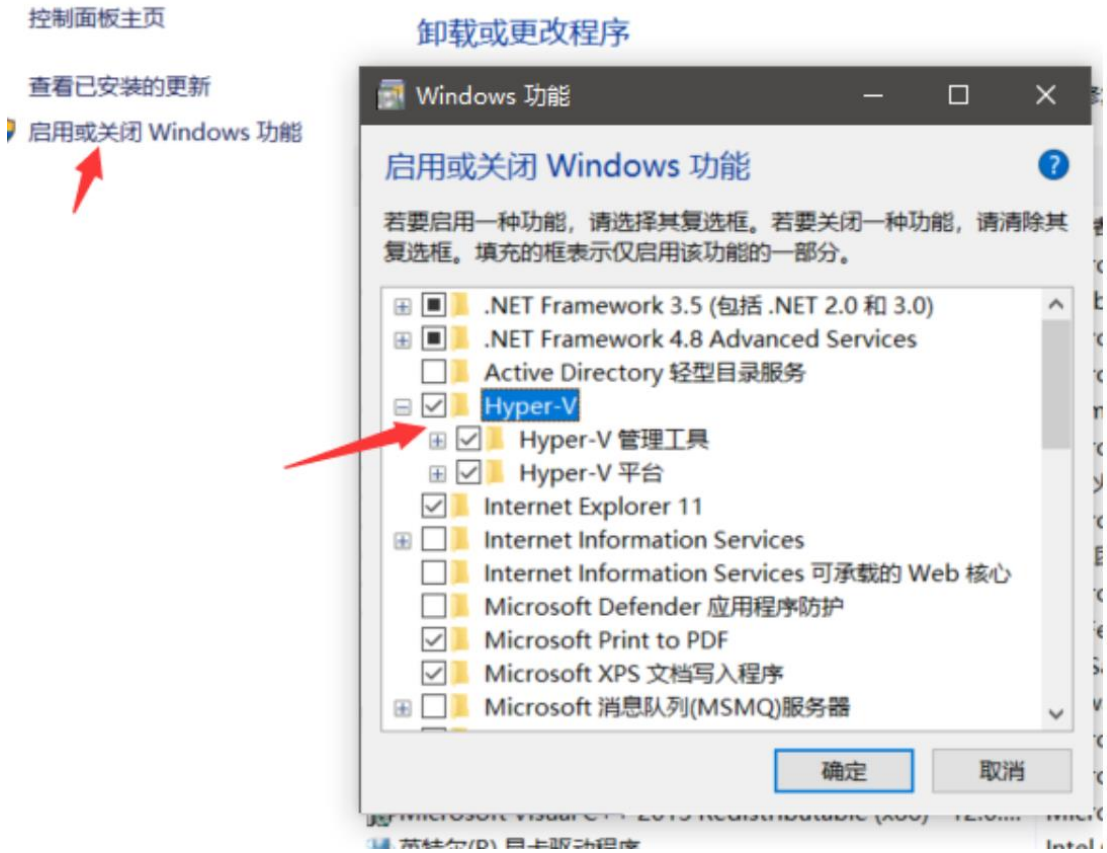
- 检查虚拟化支持：
  - 按 Win + R，输入 msinfo32，并按 Enter，查看“系统信息”窗口。检查“虚拟化已启用”项，确保显示为“是”。如果没有启用，您可能

需要手动开启。r

- 启用 Hyper-V 和虚拟化：
  - 在 Windows 中，启用虚拟化的方式：
    - 打开 “控制面板” -> “程序” -> “启用或关闭 Windows 功能”，勾选 Hyper-V 和 Windows Subsystem for Linux（WSL）。
    - 如果出现问题，检查 BIOS 设置中是否启用了虚拟化技术（Intel VT-x 或 AMD-V）。
  - 重启计算机使更改生效。

注意事项：

- Windows 版本要求：Hyper-V 仅适用于 Windows 10 专业版、企业版或教育版。如果是家庭版，您可能需要升级到专业版，或者使用 Docker Toolbox 替代 Docker Desktop。



c. **启用 Windows Subsystem for Linux 2 (WSL 2)**

- 安装 WSL 2：
  - 在 PowerShell（管理员）中运行以下命令来安装 WSL 2：

    wsl --set-default-version 2

这将确保默认使用 WSL 2。WSL 2 支持运行 Docker，提供更好的性能和兼容性。

- 安装 Linux 发行版：

- 访问 Microsoft Store，搜索并安装 Ubuntu 或其他 Linux 发行版。安装完成后，首次运行时，系统会提示设置用户名和密码。

注意事项：

- 如果之前安装过 WSL 1 版本，确保使用 wsl --set-version <distribution_name> 2 命令将现有的 Linux 发行版切换到 WSL 2。

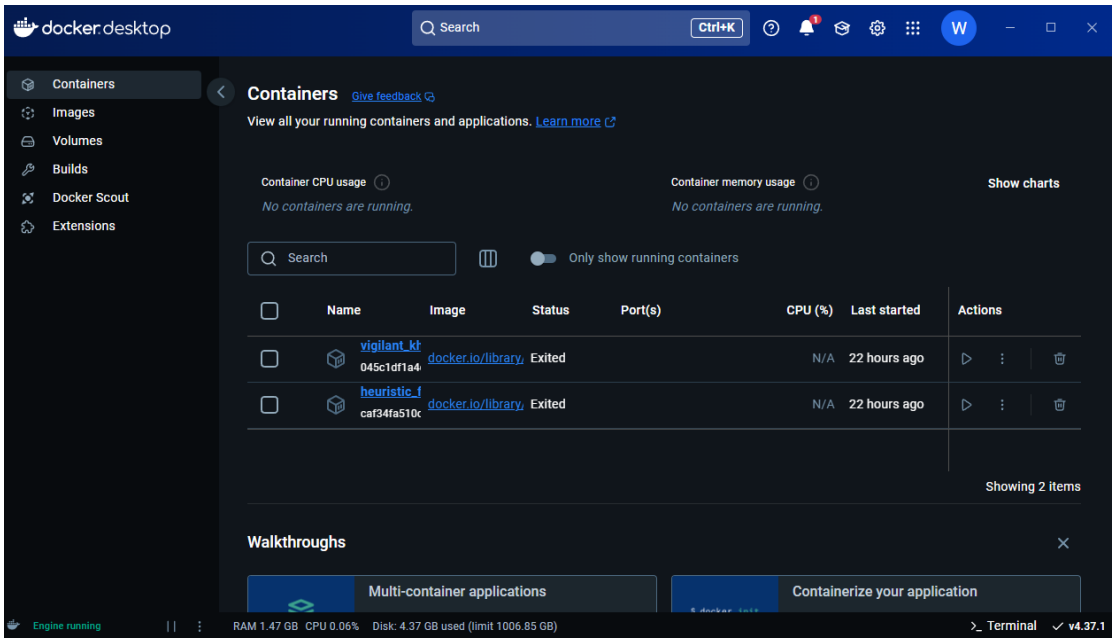
在这里，我使用的是阿里云的服务器：

d. **安装 Docker Desktop**

- 启动 Docker Desktop：
  - 完成安装后，启动 Docker Desktop。首次启动时，Docker Desktop 会自动检测并配置 WSL 2 作为 Docker 的后端引擎，完成初步配置。
- 配置 Docker Desktop：
  - 在 Docker Desktop 的设置界面中，您可以选择：
    - 资源分配：为 Docker 分配 CPU、内存、硬盘空间等资源。
    - 镜像加速器：配置国内镜像源来加速 Docker 镜像的拉取速度（例如使用阿里云镜像源）。
    - 文件共享：设置哪些文件夹可以从容器中访问。

注意事项：

- Docker Desktop 会自动检测是否安装了 WSL 2。如果检测不到 WSL 2，Docker 会提示进行手动配置。



e. **验证 Docker 安装**

- 打开命令行（CMD 或 PowerShell），执行以下命令，确认 Docker 是否安装成功：

  docker --version

  docker info

  - docker --version：返回 Docker 版本号，验证是否安装成功。
  - docker info：显示 Docker 的详细配置信息，包括运行时环境、配置的驱动程序等。

```
C:\Users\21029>docker --version
Docker version 27.4.0, build bde2b89

C:\Users\21029>docker info
Client:
 Version:    27.4.0
 Context:    desktop-linux
 Debug Mode: false
 Plugins:
  ai: Ask Gordon - Docker Agent (Docker Inc.)
    Version:  v0.5.1
    Path:     C:\Program Files\Docker\cli-plugins\docker-ai.exe
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.19.2-desktop.1
    Path:     C:\Program Files\Docker\cli-plugins\docker-buildx.exe
  compose: Docker Compose (Docker Inc.)
    Version:  v2.31.0-desktop.2
    Path:     C:\Program Files\Docker\cli-plugins\docker-compose.exe
  debug: Get a shell into any image or container (Docker Inc.)
    Version:  0.0.37
    Path:     C:\Program Files\Docker\cli-plugins\docker-debug.exe
  desktop: Docker Desktop commands (Beta) (Docker Inc.)
    Version:  v0.1.0
    Path:     C:\Program Files\Docker\cli-plugins\docker-desktop.exe
  dev: Docker Dev Environments (Docker Inc.)
    Version:  v0.1.2
    Path:     C:\Program Files\Docker\cli-plugins\docker-dev.exe
```

常见问题及解决：

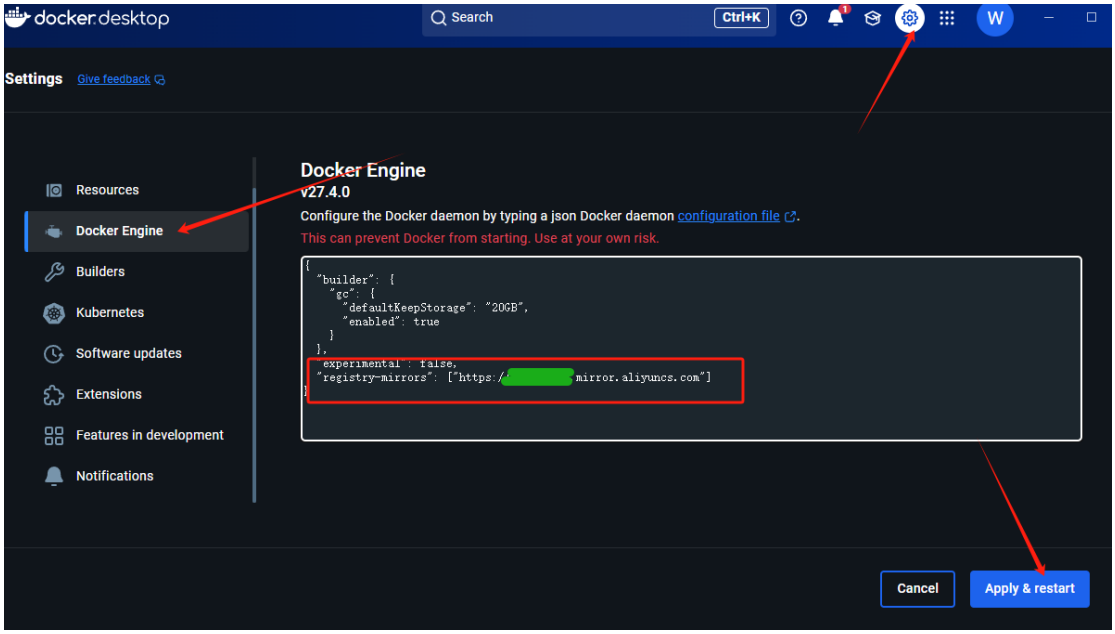
- 如果返回 command not found，可能是 Docker Desktop 未成功启动，尝试重新启动应用并检查 Docker Desktop 的图标是否显示在系统托盘。
- 如果 Docker 显示为无法连接，检查 Docker 服务是否启动，可以在服务管理器中手动启动 Docker 服务。

f. **配置 Docker 镜像加速器（可选）**

- 镜像加速器：
  - 国内访问 Docker Hub 时，下载速度可能较慢。为了提高拉取镜像的速度，可以配置镜像加速器。
  - 常见的镜像加速器有：
    - 阿里云加速器：https://cr.console.aliyun.com
    - 网易云加速器：https://hub-mirror.c.163.com
- 配置方法：
  - 打开 Docker Desktop 设置界面，进入 Docker Engine，在 registry-mirrors 配置项中添加镜像加速器，例如：

```
{
  "registry-mirrors": ["https://<your_id>.mirror.aliyuncs.com"]
}
```

- 保存并重启 Docker，确保配置生效。



g. **拉取 Docker 镜像**

- 使用 docker pull 命令从 Docker Hub 或自定义的镜像仓库拉取所需的 Docker 镜像，例如：

docker pull ubuntu

```
C:\Users\21029>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu
```

docker pull nginx

```
C:\Users\21029>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:fb197595ebe76b9c0c14ab68159fd3c08bd067ec62300583543f0ebda353b5be
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview nginx
```

- 通过 Docker Hub 拉取镜像时，请确保网络连接正常，尤其是网络较慢时可能会导致拉取失败。

h. **运行 Docker 容器**

- 使用 docker run 命令创建并启动 Docker 容器：

docker run -it ubuntu /bin/bash

- 该命令会启动一个 Ubuntu 容器并进入其终端，您可以在容器内执

行 Linux 命令。

- 通过 -d 参数可以让容器在后台运行：

  docker run -d -p 80:80 nginx

```
C:\Users\21029>docker run -it ubuntu /bin/bash
root@f726ef8bb3db:/#
```

i. **管理 Docker 容器与镜像**

- 查看正在运行的容器：

  docker ps

```
C:\Users\21029>docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
f726ef8bb3db   ubuntu    "/bin/bash"   46 seconds ago   Up 45 seconds             wonderful_archimedes
```

- 查看所有容器（包括已停止的容器）：

  docker ps -a

```
C:\Users\21029>docker ps -a
CONTAINER ID   IMAGE                                  COMMAND                  CREATED             STATUS
        PORTS     NAMES
f726ef8bb3db   ubuntu                                 "/bin/bash"              About a minute ago  Up About a minute
                  wonderful_archimedes
045c1df1a4c1   m.daocloud.io/docker.io/library/nginx   "/docker-entrypoint.…"   22 hours ago        Exited (0) 18 hours
 ago              vigilant_khayyam
caf34fa510c2   m.daocloud.io/docker.io/library/nginx   "/docker-entrypoint.…"   22 hours ago        Exited (0) 22 hours
 ago              heuristic_feistel
```

- 停止和删除容器：

  docker stop <container_id>

  docker rm <container_id>

```
C:\Users\21029>docker stop f726ef8bb3db
f726ef8bb3db

C:\Users\21029>docker rm f726ef8bb3db
f726ef8bb3db
```

- 查看本地镜像：

  docker images

```
C:\Users\21029>docker images
REPOSITORY                                TAG          IMAGE ID       CREATED        SIZE
wiserx/python-tkinter-app                 latest       cf1f7fce3273   22 hours ago   1.02GB
python-tkinter-app                        latest       cf1f7fce3273   22 hours ago   1.02GB
<none>                                    <none>       8987dcfa6635   22 hours ago   1.02GB
python                                    3.11-alpine  a748f03d63e3   2 weeks ago    53.7MB
python                                    3.12         efef85b1e82b   2 weeks ago    1.02GB
nginx                                     latest       66f8bdd3810c   3 weeks ago    192MB
m.daocloud.io/docker.io/library/nginx     latest       66f8bdd3810c   3 weeks ago    192MB
ubuntu                                    latest       b1d9df8ab815   4 weeks ago    78.1MB
jittor/jittor                             latest       01fb628ef5ec   3 years ago    992MB
```

- 删除镜像：

  docker rmi <image_id>

```
C:\Users\21029>docker rmi jittor/jittor
Untagged: jittor/jittor:latest
Untagged: jittor/jittor@sha256:d61e0359f335b6aaca651f21214a61b90f9445faa9fd20db9b358dd9994f2a8d
Deleted: sha256:01fb628ef5ecfe00c36bf489f8bcddf5461c2b8bb2499047b9ea5c2198d3c529
Deleted: sha256:da08098df5e94f8791bf43060fa30a2fb75223a2bd85b35fb710216a83e8d3d2
Deleted: sha256:ddb21cfc21c0dd6579c6729d0bbdc024d8724f6e2117f97422077fa2cc8716e1
Deleted: sha256:d47649642ef0e55e2f4914a7a156d79d220beed4ef4d87de54c3dceb4de094ec
Deleted: sha256:fbe96e2988683c4c43e973dd9a0e38619dbe10384078128c6254b691e316931f
Deleted: sha256:80ee0c5bb12b512c056727bbf6fbb82608d87e63f4a9af366bfee7f874313522
Deleted: sha256:79a23ae5600c7332d7ac01e76335d4cb85ec1ebc24aa43612dba70faaeadfcba
Deleted: sha256:dda399e3943000fd507ee1e86e94854b25530e1c51cd2852073470cd1c5e0ff0
Deleted: sha256:57f1d9bb8738bba26ff5d1794b733d091f7cdc45ec22c9200fef583398f792f6
Deleted: sha256:0b83249f20103d42b4df812af8784968a98a56b6272f262e2200b978d830a05c
Deleted: sha256:2b9acb02117f9fcaa339d48723a1795879a8475af5c55ec845f830e902b9dcb4
Deleted: sha256:399d16984f863d726fe2a9b798d59ac2d42aac3f00031865e40987b7dbd94cde
Deleted: sha256:b0a08a9641126a5ad3a21f362a6429bbfb9c6251a0100b8a7280d25529889cb6
Deleted: sha256:423dac4f329bef3a1d6b304f03098b3548b15c3c467ba3bc927b998d27ad4e8d
Deleted: sha256:bf1b016ccde8bf358a145878d4c5b2845b1b62baf369fb418b561400797d3d38
Deleted: sha256:9459b6a89846db0723e467610b841e6833dbb2aae6133319a91f2f70c388afac
Deleted: sha256:9a9311f7fcddf94f7476ce89f9703e8360e8cf347ef486a280735f5cf98888cd
Deleted: sha256:b43408d5f11b7b2faf048ae4eb25c296536c571fb2f937b4f1c3883386e93d64
```

j. **使用 Docker Compose（可选）**

- 安装 Docker Compose： Docker Compose 已包含在 Docker Desktop 中，因此您无需额外安装。
- 创建 docker-compose.yml 文件：
  - Docker Compose 用于定义和运行多个容器应用。您可以通过配置 docker-compose.yml 文件来自动化服务管理。
  - 示例文件：

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
```

- 启动服务：

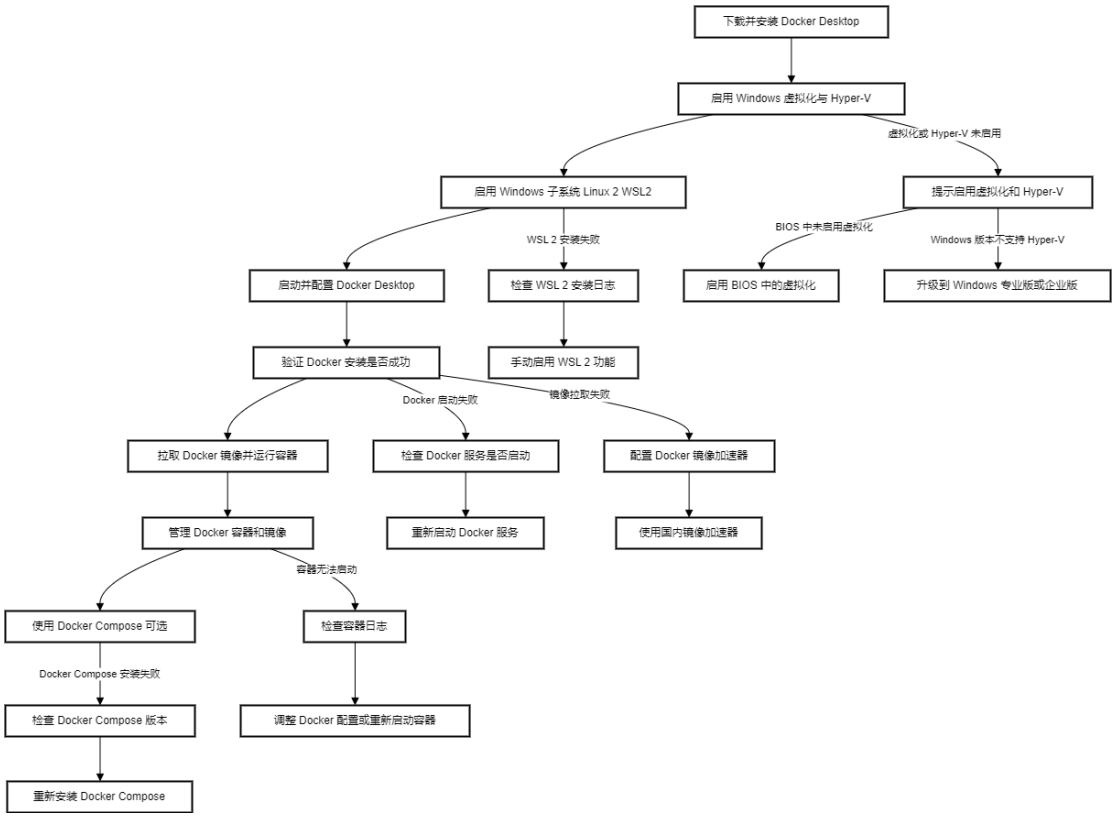
```
docker-compose up
```

总结要点

1. 安装 Docker Desktop：确保正确安装并启用 WSL 2 和 Hyper-V，适用于 Windows 10 专业版及以上版本。
2. 启用虚拟化功能：检查 BIOS 中是否启用了虚拟化技术，并在 Windows 中启用 Hyper-V 和 WSL 2。

3. 配置 Docker 环境：通过 Docker Desktop 配置资源限制、镜像加速器和文件共享等设置。
4. 验证安装：确保 Docker 正常运行并能够访问 Docker Hub。
5. 使用 Docker Compose：学习如何利用 Docker Compose 管理多个容器服务。

Windows 环境下 Docker 环境搭建流程图



2、Docker 环境搭建的难点与问题

a. **Docker 镜像下载慢 / 镜像拉取失败**

难点：

- 在国内，由于网络问题，直接从 Docker Hub 拉取镜像的速度非常慢，甚至可能由于超时导致拉取失败。这是 Docker 安装过程中的一个常见问题。

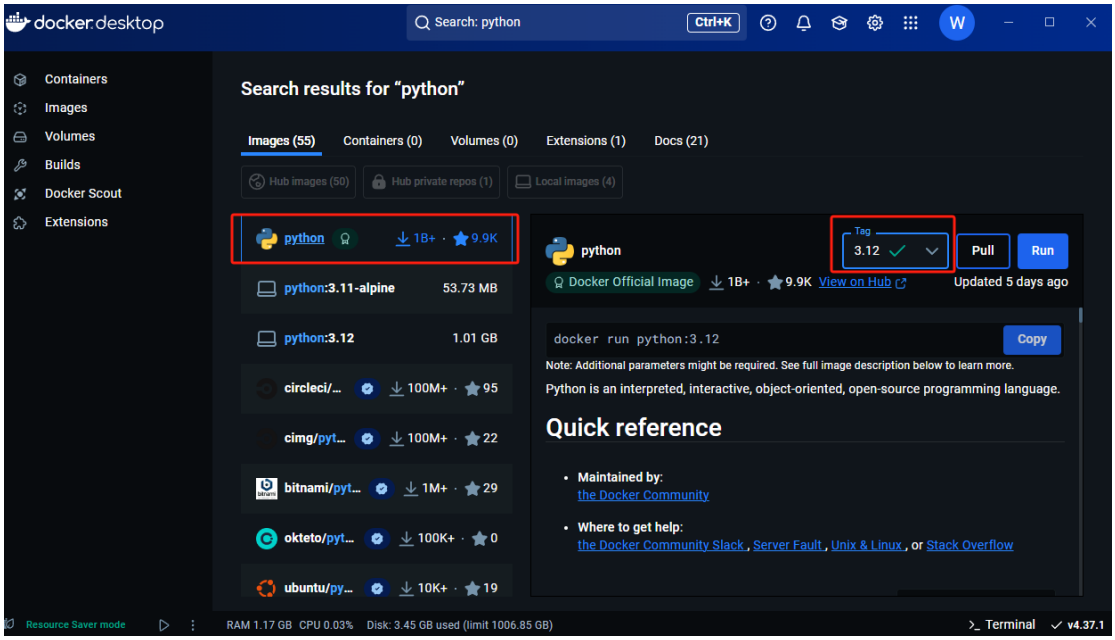
解决方法：

- 使用镜像加速器：通过配置镜像加速器，可以大大提高镜像下载的速度。常用的加速器有：
  - 阿里云镜像加速器：https://cr.console.aliyun.com
  - 网易云镜像加速器：https://hub-mirror.c.163.com

  在 Docker Desktop 中配置加速器：
  - 打开 Docker Desktop，进入 Settings（设置）。
  - 选择 Docker Engine，然后在配置文件中添加镜像加速器：

  ```
  {
    "registry-mirrors": ["https://<your_id>.mirror.aliyuncs.com"]
  }
  ```

  - 保存设置并重启 Docker。
- 切换到国内源：可以使用 docker pull 命令时指定特定的镜像源，例如：*docker pull registry.cn-hangzhou.aliyuncs.com/ubuntu:latest*
- 使用 Docker Desktop 软件进行选择下载，如图：



b. **Docker 服务无法启动 / 无法与 Docker 引擎通信**

难点：

- 在 Docker Desktop 启动后，有时可能会遇到 Docker 服务无法启动 或无法与 Docker 引擎通信 的问题。此时命令行会提示 Docker 引擎未启动或无法连接。

解决方法：

- 重新启动 Docker：打开 Task Manager（任务管理器），结束 Docker 相关进程，然后重新启动 Docker Desktop。
- 检查 Windows 服务：在 Windows 服务管理器中检查 Docker 服务是否正在运行。如果 Docker 服务未启动，可以手动启动 Docker 服务。
- 调整 Docker 配置：如果 Docker 配置错误，可以在 Docker Desktop 的 Settings 中恢复默认配置。

总结：

在 Docker 环境搭建过程中，最常见的难点和问题通常涉及 虚拟化与 Hyper-V 的启用、WSL 2 配置问题、镜像拉取问题、端口冲突 等。这些问题可能会阻碍 Docker 的顺利运行，但通过合理配置、及时调整设置和使用网络加速器等方法，大多数问题都可以顺利解决。

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

在 Docker 中，容器的 启动 和 卸载 是最常见的操作，通常涉及以下步骤：

启动 Docker 容器的要点：

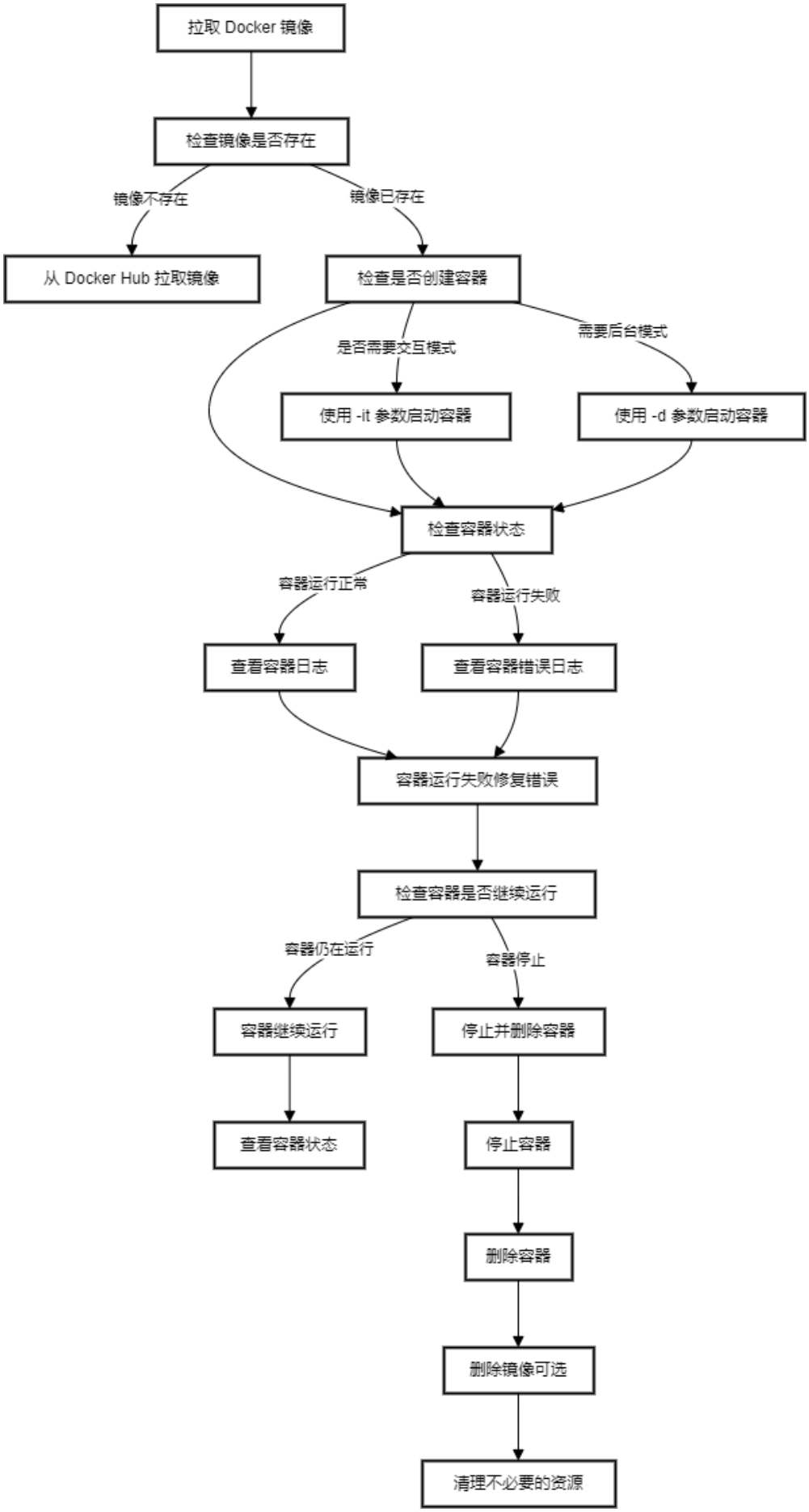
a) 拉取镜像：启动容器之前，首先需要从 Docker Hub 或其他镜像仓库拉取所需的镜像。
b) 创建容器：容器可以通过 docker run 命令启动，指定镜像及所需的运行参数（如端口映射、环境变量、挂载卷等）。
c) 容器运行模式：
    - 交互模式 (-it): 启动容器并进入容器的交互式终端。
    - 后台模式 (-d): 容器在后台运行。
d) 端口映射：容器内的端口需要映射到主机上的端口，以便外部访问。
e) 容器日志：查看容器启动后的日志，确认是否正常运行。
f) 检查容器状态：通过 docker ps 检查容器是否在运行。

卸载 Docker 容器的要点：

a) 停止容器：在删除容器之前，必须先停止容器。使用 docker stop 命令停止运行中的容器。
b) 删除容器：停止容器后，通过 docker rm 命令删除容器。
c) 删除镜像（可选）：如果不再需要容器的镜像，可以通过 docker rmi 删除镜像。
d) 清理无用资源：使用 docker system prune 清理不再需要的容器、镜像和网络资源。

Docker 容器的启动与卸载流程图

以下是绘制的 Docker 容器启动与卸载的流程图。该流程图涵盖了常见的条件判断和步骤。


拉取 Docker 镜像
检查镜像是否存在
镜像不存在
镜像已存在
从 Docker Hub 拉取镜像
检查是否创建容器
是否需要交互模式
需要后台模式
使用 -it 参数启动容器
使用 -d 参数启动容器
检查容器状态
容器运行正常
容器运行失败
查看容器日志
查看容器错误日志
容器运行失败修复错误
检查容器是否继续运行
容器仍在运行
容器停止
容器继续运行
停止并删除容器
查看容器状态
停止容器
删除容器
删除镜像可选
清理不必要的资源

流程图解释：

a) 拉取 Docker 镜像：
   - 先检查本地是否已经有该镜像。如果镜像存在，跳过拉取步骤。
   - 如果镜像不存在，执行拉取命令，从 Docker Hub 或其他镜像仓库拉取镜像。

b) 检查容器创建和启动模式：
   - 在容器创建过程中，根据需求选择启动模式：
     - 交互模式（使用 -it 参数）：启动容器并进入容器的交互式终端。
     - 后台模式（使用 -d 参数）：容器在后台运行。

c) 检查容器状态：
   - 使用 docker ps 命令检查容器是否已成功启动并运行。
   - 如果容器运行失败，需要查看容器日志，找出原因并修复。

d) 停止并删除容器：
   - 如果容器停止运行，则使用 docker stop 命令停止容器。
   - 使用 docker rm 删除容器，删除后可以选择是否删除相关镜像。

e) 清理不必要的资源：
   - 使用 docker system prune 清理不再需要的容器、镜像和网络资源，以释放磁盘空间。

关键命令总结：

- 拉取镜像：

  docker pull <image_name>

```
C:\Users\21029>docker pull python
Using default tag: latest
latest: Pulling from library/python
fdf894e782a2: Already exists
5bd71677db44: Already exists
551df7f94f9c: Already exists
ce82e98d553d: Already exists
5f0e19c475d6: Pull complete
abab87fa45d0: Pull complete
2ac2596c631f: Pull complete
Digest: sha256:9255d1993f6d28b8a1cd611b108adbdfa38cb7ccc46ddde8ea7d734b6c845e32
Status: Downloaded newer image for python:latest
docker.io/library/python:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview python
```

- 运行容器：
  - 交互模式：

    docker run -it <image_name> <command>

```
C:\Users\21029>docker run -it python python --version
Python 3.13.1
```

- 后台模式：

  docker run -d -p <host_port>:<container_port> <image_name>

- 查看容器状态：

  docker ps　　# 查看运行中的容器

  docker ps -a　　# 查看所有容器（包括停止的容器）

- 查看容器日志：

  docker logs <container_id>

```
C:\Users\21029>docker ps -a
CONTAINER ID   IMAGE
    PORTS     NAMES
e7acc5681d5d   python
o             eloquent_black
a7c114b8bfc8   python
             gallant_panini
045c1df1a4c1   m.daocloud.io/docker.io/li
            vigilant_khayyam
caf34fa510c2   m.daocloud.io/docker.io/li
            heuristic_feistel

C:\Users\21029>docker logs e7acc5681d5d
Python 3.13.1
```

- 停止容器：

  docker stop <container_id>

- 删除容器：

  docker rm <container_id>

- 删除镜像：

  docker rmi <image_id>

- 清理无用资源：

  docker system prune

总结要点

- 启动容器：
  - 确保首先拉取镜像。
  - 选择适当的容器启动模式（交互模式或后台模式）。
  - 检查容器状态，并查看日志排查可能的启动错误。
- 卸载容器：

- 确保停止并删除容器，防止未清理的容器占用资源。
- 可以选择删除不再需要的镜像以及使用 docker system prune 清理不必要的资源。

2、Docker 容器的启动与卸载的难点与问题

a. 容器启动失败或挂起

难点描述： 容器启动后可能会卡在“挂起”状态，或者在某些情况下无法正常启动。常见的原因包括配置错误、资源不足或镜像问题。

常见原因与解决方法：

- 容器无法找到所需的文件或目录：
  - 例如，如果使用 docker run 启动一个容器时，设置了一个错误的工作目录或没有挂载所需的卷，容器会启动失败或挂起。
  - 解决方法：
    - 确保镜像内部需要的文件或目录已正确挂载。
    - 使用 docker run 时，确保卷（-v）和工作目录（-w）的配置是正确的：

      docker run -v /host/path:/container/path -w /container/path my_image
- 端口冲突：
  - 启动容器时，如果容器内的端口已与主机上的其他服务冲突，容器将无法启动。
  - 解决方法：
    - 在运行容器时指定不同的端口映射：

      docker run -d -p 8081:80 my_image

```
C:\Users\21029>docker run -d -p 8081:80 python
Unable to find image 'python:latest' locally
latest: Pulling from library/python
fdf894e782a2: Pull complete
5bd71677db44: Pull complete
551df7f94f9c: Pull complete
ce82e98d553d: Pull complete
5f0e19c475d6: Pull complete
abab87fa45d0: Pull complete
2ac2596c631f: Pull complete
Digest: sha256:9255d1993f6d28b8a1cd611b108adbdfa38cb7ccc46ddde8ea7d734b6c845e32
Status: Downloaded newer image for python:latest
11e98f903873b704eff9a043903bf7e17a618c714bc1ca36d6f975aae5ce8e95
```

    - 使用 docker ps 查看当前占用端口的服务并调整端口。
- 镜像问题或损坏：
  - 如果镜像损坏或不完整，容器可能无法启动。

- 解决方法：
  - 删除并重新拉取镜像：
    docker rmi <image_id>
    docker pull <image_name>
  - 使用 docker images 查看本地镜像。

调试容器启动问题：

- 使用 docker logs <container_id> 查看容器日志，检查容器启动失败的详细信息。
- 使用 docker inspect <container_id> 查看容器的详细信息，尤其是启动命令和环境变量配置。

---

b. **容器运行时资源不足（内存/CPU）**

难点描述： Docker 容器运行过程中，可能会遇到内存或 CPU 资源不足的情况，导致容器崩溃或被自动停止。

常见原因与解决方法：

- 资源限制：
  - 默认情况下，Docker 为每个容器分配的内存和 CPU 资源是有限的。对于一些高负载应用，如果没有足够的资源，容器可能会退出或崩溃。
  - 解决方法：
    - 使用 --memory 和 --cpus 参数限制容器的资源，或者增加容器的资源配额：
      docker run -d --memory="2g" --cpus="2" my_image

```
C:\Users\21029>docker run -d --memory="2g" --cpus="2" python
1536f08913393da141036921155cd850ab55c1713571173c8f8c6810f90881e1
```

- 容器内存溢出：
  - 某些应用程序（如数据库或大数据处理程序）可能需要大量内存，如果没有适当的内存分配，容器会因内存溢出而停止。
  - 解决方法：
    - 使用 docker stats 命令查看容器的资源使用情况。
    - 确保容器有足够的内存，如果资源不够，可以使用 --memory 增加内存。

---

c. **容器与主机之间的网络连接问题**

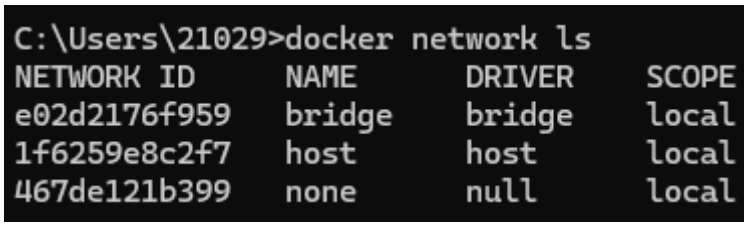
难点描述： 在某些情况下，容器与宿主机或其他容器之间的网络连接可能

存在问题，导致无法访问容器的服务或 API。

常见原因与解决方法：

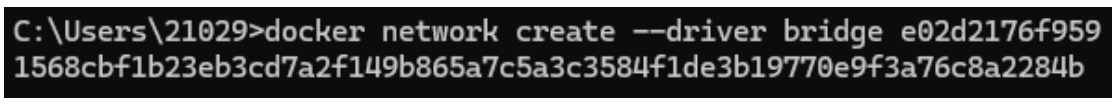
- 端口映射错误：
  - 如果启动容器时没有正确映射端口，容器内的服务将无法被主机或外部访问。
  - 解决方法：
    - 确保使用正确的 -p 参数进行端口映射，例如：

      docker run -d -p 8080:80 my_image
- Docker 网络配置问题：
  - 默认情况下，Docker 会为每个容器分配一个虚拟网络。如果网络配置不正确，可能会导致容器间的通信失败。
  - 解决方法：
    - 使用 docker network ls 查看现有的网络，并检查容器是否正确连接到指定网络。

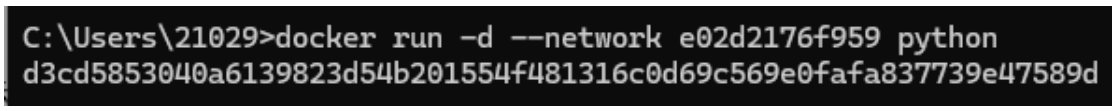


    - 如果需要自定义网络，可以使用：

      docker network create --driver bridge my_network



docker run -d --network my_network my_image



- 防火墙和安全组限制：
  - 在某些企业或云环境中，防火墙或安全组规则可能会限制容器与主机之间的网络通信。
  - 解决方法：
    - 检查防火墙或云服务的安全组设置，确保相应端口和协议没有被阻止。

---

d. **容器卸载时的错误**

难点描述： 在卸载容器时，可能遇到一些常见问题，如容器无法停止、容

器无法删除、容器仍占用资源等。

常见原因与解决方法：

- 容器无法停止：
  - 如果容器长时间处于运行状态或没有响应，使用 docker stop <container_id> 可能无法停止容器。
  - 解决方法：
    - 使用 docker kill <container_id> 强制停止容器：
      docker kill <container_id>
- 容器无法删除：
  - 如果容器仍在运行或存在挂载的卷，删除容器时可能会失败。
  - 解决方法：
    - 确保容器已停止并删除所有挂载的卷：
      docker rm -v <container_id>
- 镜像被容器占用无法删除：
  - 如果镜像正在被某个容器使用，则不能直接删除该镜像。
  - 解决方法：
    - 停止并删除使用该镜像的容器，然后再删除镜像：
      docker ps -a # 查看容器
      docker rm <container_id>
      docker rmi <image_id>

---

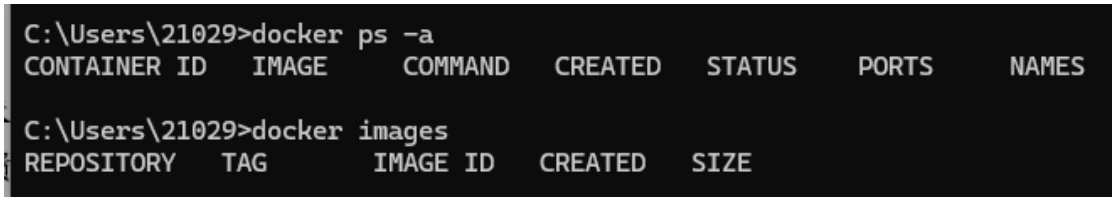
e. **清理无用资源**

难点描述：随着容器的创建和删除，系统中可能会堆积大量未使用的容器、镜像、卷和网络。清理这些资源时容易出错。

常见原因与解决方法：

- 未清理的资源：
  - Docker 容器、镜像和网络占用磁盘空间，定期清理是必要的。
  - 解决方法：
    - 使用以下命令清理不再使用的资源：
      docker system prune -a

- 镜像和容器依赖关系：
    - 删除容器时可能会遇到镜像被其他容器依赖的情况，导致镜像无法删除。
    - 解决方法：
        - 使用 docker ps -a 查看所有容器，确保没有正在使用镜像的容器。
        - 使用 docker images 查看所有镜像，确认镜像的依赖情况。



发现都不存在，因此没有问题。

总结：

在 Docker 容器的启动与卸载过程中，常见的难点包括容器启动失败、资源不足、网络问题、容器卸载问题以及清理无用资源。通过合理配置资源、检查容器和镜像的依赖关系、及时清理无用资源等方法，大多数问题都能得到解决。

解决这些问题的关键是：

- 细致检查容器状态和日志，帮助定位启动失败的根本原因。
- 合理配置资源，避免资源不足导致容器崩溃。
- 清理无用资源，保持 Docker 环境的整洁，避免积累不必要的镜像、容器和卷。

三、基于 Docker 的应用部署

在本节中，我将详细描述如何将服务使能课程中编写完成的微服务应用打包成 Docker 镜像，并使用 Docker 部署该微服务应用。具体的步骤包括编写 Dockerfile、构建镜像、推送镜像到 Docker 仓库、运行容器以及验证应用的效果。

---

a. 前期准备

在 Ubuntu 上，首先需要确保以下环境和工具已准备好：

1. 安装 Docker:

使用命令在 Ubuntu 上安装 Docker：

sudo apt update

sudo apt install -y docker.io

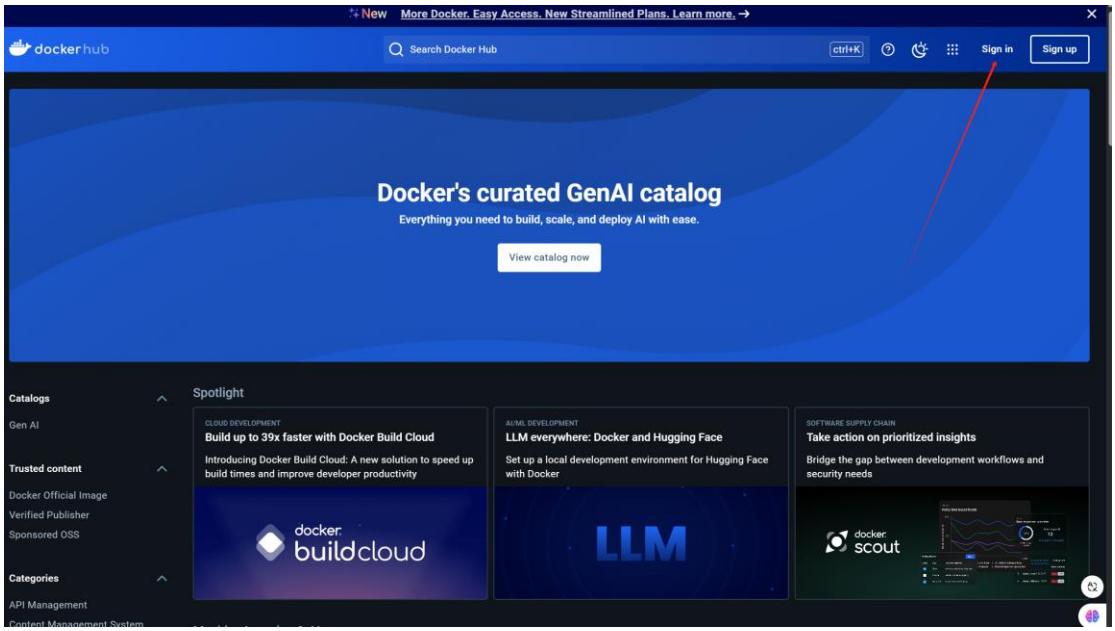
sudo systemctl enable --now docker

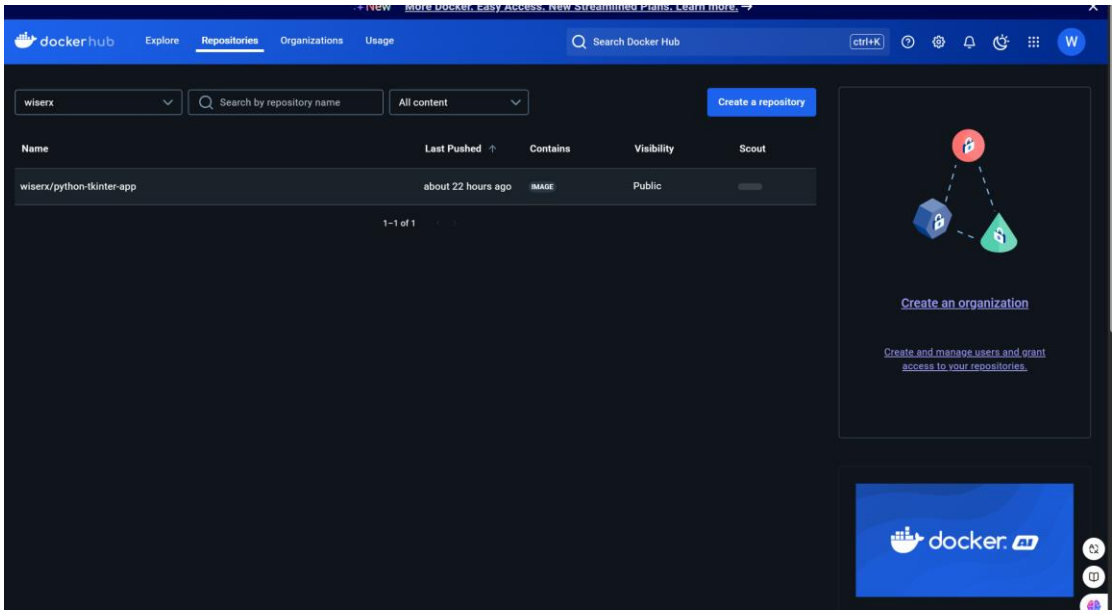
然后，验证 Docker 是否安装成功：

docker --version

2. Docker Hub 账户：

如果要将镜像推送到 Docker Hub，首先需要注册 Docker Hub 账户。我通过访问 Docker Hub 官网 注册账户。





3. 微服务应用代码：

我编写了 Python 应用代码，并保存在当前目录中。代码如下（文件名为 app.py）：

```
def show_welcome():
    name = input("请输入您的名字: ")  # 获取用户输入
    welcome_message = f"欢迎, {name}!"
    print(welcome_message)  # 打印欢迎信息

def show_sum():
    try:
        n = int(input("请输入一个整数: "))  # 获取用户输入并转换为整数
        total_sum = sum(range(1, n + 1))  # 计算从 1 加到 n 的总和
        print(f"总和: {total_sum}")  # 打印总和
    except ValueError:
        print("请输入一个有效的整数")  # 输入无效时显示错误信息

# 调用函数
show_welcome()
show_sum()
```

b. 编写 Dockerfile

确保在项目的根目录下创建 Dockerfile，内容如下：

```
# 选择官方的 Python 3.11.7 镜像作为基础镜像
```

```
FROM python:3.11.7-slim

# 安装 Tkinter 和其他依赖
RUN apt-get update && apt-get install -y \
    python3-tk \
    && rm -rf /var/lib/apt/lists/*

# 设置工作目录
WORKDIR /app

# 将当前目录的代码复制到容器中
COPY . /app

# 安装 Python 的依赖
RUN pip install --no-cache-dir -r requirements.txt

# 设置容器启动时的命令
CMD ["python", "app.py"]
```

c. 构建 Docker 镜像

在 Dockerfile 文件和 Python 应用代码（例如 app.py）所在的根目录中，执行以下命令来构建 Docker 镜像：

```
docker build -t python-tkinter-app .
```

这条命令的解释：

- docker build：构建镜像的命令。
- -t python-tkinter-app：给构建的镜像命名为 python-tkinter-app。
- .：表示 Dockerfile 和应用代码所在的当前目录。

```
C:\Users\21029\Desktop\软件架构与中间件\实验\实验二\docker_test>docker build -t python-tkinter-app .
[+] Building 9.0s (9/9) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                               0.0s
 => => transferring dockerfile: 496B                                                                               0.0s
 => [internal] load metadata for docker.io/library/python:3.12                                                     0.0s
 => [internal] load .dockerignore                                                                                  0.0s
 => => transferring context: 2B                                                                                    0.0s
 => [1/4] FROM docker.io/library/python:3.12                                                                       0.1s
 => [internal] load build context                                                                                  0.0s
 => => transferring context: 523B                                                                                  0.0s
 => [2/4] RUN apt-get update && apt-get install -y     python3-tk     && rm -rf /var/lib/apt/lists/*               8.8s
 => [3/4] WORKDIR /test                                                                                            0.0s
 => [4/4] COPY . /test                                                                                             0.0s
 => exporting to image                                                                                             0.1s
 => => exporting layers                                                                                            0.0s
 => => writing image sha256:8987dcfa6635aa17664b4e6fad6afb8f60b9cbaa8ff9d8f588fb9292775353d3                       0.0s
 => => naming to docker.io/library/python-tkinter-app                                                              0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/1q7l9f8lgkpzrk6lybodfb0hz

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

d. 运行 Docker 容器

在 Ubuntu 上运行容器时，使用以下命令来运行 Docker 容器：

*docker run -it *

*--rm *

*-e DISPLAY=$DISPLAY *

python-tkinter-app

命令解释：

- -it：以交互模式运行容器，并连接容器的终端。
- --rm：容器停止时自动删除容器。
- -e DISPLAY=$DISPLAY：将宿主机的显示环境变量传递给容器，确保 Tkinter 能够显示在宿主机的 X11 显示服务器上。
- python-tkinter-app：指定要运行的镜像名称。

```
C:\Users\21029\Desktop\软件架构与中间件\实验\实验二\docker_test>docker run -it --rm -e DISPLAY=$DISPLAY python-tkinter-a
pp
请输入您的名字：zhouyufan
欢迎，zhouyufan!
请输入一个整数：100
总和：5050
```

e. 验证容器是否运行

运行容器后，你可以使用以下命令来检查容器的状态：

docker ps

```
C:\Users\21029\Desktop\软件架构与中间件\实验\实验二\docker_test>docker ps
CONTAINER ID   IMAGE                                    COMMAND                  CREATED         STATUS         PORTS
                NAMES
045c1df1a4c1   m.daocloud.io/docker.io/library/nginx   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:3
2769->80/tcp   vigilant_khayyam
```

此时，应用运行的结果已经显示在命令行界面里。

f. 测试应用

在容器中运行应用后，能够看到打印出来的实验结果。用户可以输入名字并输入数字，程序会在窗口中显示欢迎信息并打印计算结果。

g. 推送镜像到 Docker Hub

将镜像推送到 Docker Hub，首先需要在 Docker Hub 上创建一个仓库。

1. 登录 Docker Hub：

使用以下命令登录到 Docker Hub：

docker login

2. 推送镜像：

给本地镜像打标签并将镜像推送到 Docker Hub：

docker tag python-tkinter-app wiserx/python-tkinter-app:latest

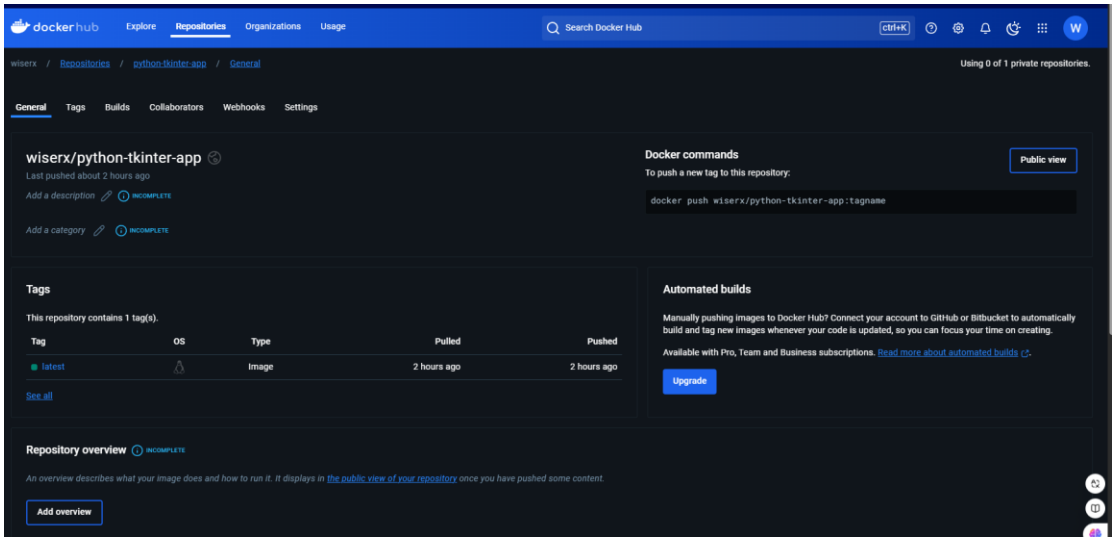
docker push wiserx/python-tkinter-app:latest

解释：

- docker tag：为本地镜像打标签，使其符合 Docker Hub 仓库的命名规则。
- docker push：将镜像推送到 Docker Hub。

```
C:\Users\21029\Desktop\软件架构与中间件\实验\实验二\docker_test>docker push wiserx/python-tkinter-app:latest
The push refers to repository [docker.io/wiserx/python-tkinter-app]
b5c257452a71: Pushed
0992508966c9: Pushed
d11a9ae9e2e0: Pushed
26b4b5ee3530: Mounted from library/python
0c968b3c14da: Mounted from library/python
4c05d5237664: Mounted from library/python
0aeeeb7c293d: Mounted from library/python
c81d4fdb67fc: Mounted from library/python
0e82d78b3ea1: Mounted from library/python
301c1bb42cc0: Mounted from library/python
latest: digest: sha256:2db71d7eab108b237a05a496543386114b3aaf82aaa1c7be08143ed321f24a5d size: 2420
```

可以看到，已经将镜像推送到了 Docker Hub。



3. 拉取镜像：

如果在其他机器上使用该镜像，可以使用以下命令拉取镜像：

docker pull wiserx/python-tkinter-app:latest

h. 容器卸载与清理

如果不再需要容器和镜像，可以执行以下命令来清理资源：

1. 停止并删除容器：

docker stop my-tkinter-container

docker rm my-tkinter-container

2. 删除镜像：

docker rmi python-tkinter-app

总结

通过 Docker 部署 Python Tkinter 应用，我能够将应用打包成容器镜像，便于跨平台的部署和管理。具体的步骤包括：

1. 编写 Dockerfile
2. 构建 Docker 镜像
3. 运行容器并映射宿主机的显示环境
4. 测试应用
5. 将镜像推送到 Docker Hub（如果需要）

6. 清理不必要的容器和镜像

这使得我的 Python 应用具有更高的可移植性和管理效率。