



# Win10 安装 Ubuntu



win10安装ubuntu



在 Windows 10 上安装 Ubuntu 的方法有多种，主要取决于你的需求。以下是三种常见的方法：

## 方法 1：使用 Windows Subsystem for Linux (WSL)

这是微软官方提供的一种方式，适合需要在 Windows 上运行 Ubuntu 命令行的用户。

### 步骤：

#### 1. 启用 WSL 功能：

- 打开“控制面板”→“程序”→“启用或关闭 Windows 功能”。
- 勾选“适用于 Linux 的 Windows 子系统”。
- 点击“确定”并重启电脑。

#### 2. 安装 Ubuntu：

- 打开 Microsoft Store。
- 搜索并选择想要的 Ubuntu 版本（例如 Ubuntu 22.04 LTS）。
- 点击“获取”并安装。

#### 3. 配置 Ubuntu：

- 安装完成后，打开 Ubuntu 应用。
- 设置用户名和密码。

#### 4. 开始使用：

- 现在你可以直接在 Windows 终端或 Ubuntu 应用中运行 Linux 命令。

## 方法 2：使用虚拟机 (Virtual Machine)

适合需要完整桌面环境的用户。

### 所需工具：

- [VMware Workstation Player](#) 或 [VirtualBox](#)。
- Ubuntu ISO 文件（可从 [Ubuntu官网](#) 下载）。





## 步骤：

1. **安装虚拟机软件：**
  - 下载并安装 VMware 或 VirtualBox。
2. **创建虚拟机：**
  - 打开虚拟机软件并创建一个新虚拟机。
  - 选择下载的 Ubuntu ISO 文件作为安装源。
  - 设置虚拟机配置（推荐至少 2 核 CPU、4GB 内存、20GB 硬盘空间）。
3. **安装 Ubuntu：**
  - 启动虚拟机并按照屏幕提示完成 Ubuntu 的安装。
4. **完成后使用：**
  - 启动虚拟机即可进入 Ubuntu 系统。

---

## 方法 3：双系统安装

适合希望在电脑上运行完整 Ubuntu 系统的用户。

## 注意事项：

- 安装双系统有风险，请提前备份重要数据。
- 需要 USB 启动盘。

## 步骤：

1. **准备工作：**
  - 下载 Ubuntu ISO 文件。
  - 使用工具（如 Rufus）制作启动盘。
2. **分区：**
  - 打开 Windows 磁盘管理工具，为 Ubuntu 留出一个空闲分区（建议 20GB 以上）。
3. **禁用快速启动：**
  - 打开“控制面板” → “电源选项” → “选择电源按钮的功能”。
  - 取消勾选“启用快速启动”。
4. **安装 Ubuntu：**
  - 插入 USB 启动盘并重启电脑。
  - 在 BIOS 中设置从 USB 启动。
  - 启动后选择“安装 Ubuntu”。
  - 在分区设置中选择“与 Windows 并存”或手动分区。



## 5. 完成安装：

- 安装完成后重启电脑，你将看到一个启动菜单，可以选择进入 Ubuntu 或 Windows。

## 哪种方法适合你？

- 需要轻量级命令行环境：WSL。
- 需要完整桌面体验但不想改动硬盘：虚拟机。
- 需要完整独立的双系统：双系统安装。

需要更多具体细节或步骤可以随时告诉我！



在公司工作是用2台vm、一台win10連進另一台ubuntu開發linux base的docker image，公司不開放遠端工作，我在家想從事開發，但只有1台windows筆電怎麼辦



如果你只有一台 Windows 笔电，但想在家进行基于 Linux 的 Docker 开发，可以考虑以下方案：

## 方法 1：使用 Windows 的 WSL2

Windows Subsystem for Linux (WSL2) 提供了接近原生 Linux 的环境，可以运行 Docker 和其他 Linux 工具。

### 配置步骤：

#### 1. 启用 WSL 和安装 Ubuntu：

- 按照 [Win10安装Ubuntu](#) 的步骤安装 WSL 和 Ubuntu。

#### 2. 安装 Docker：

- 在 Windows 上安装 [Docker Desktop](#)。
- 在 Docker Desktop 的设置中启用 WSL 集成：
  - 打开 Docker Desktop → 设置 → Resources → WSL Integration → 勾选安装的 Ubuntu。

#### 3. 在 WSL 中开发 Docker：

- 启动 WSL 的 Ubuntu 环境。
- 使用 `docker` 命令开发和管理 Docker 镜像。

### 优势：





- 不需要虚拟机，性能较高。
- 完全在 Windows 上完成开发。

---

## 方法 2：安装虚拟机并运行 Ubuntu

如果需要完整的 Linux 桌面体验或模拟你的公司环境，可以使用虚拟机。

### 配置步骤：

1. **安装虚拟机软件：**
  - 下载并安装 [VirtualBox](#) 或 [VMware Workstation Player](#)。
2. **安装 Ubuntu 虚拟机：**
  - 使用 Ubuntu ISO 文件创建虚拟机。
  - 为虚拟机分配足够资源（建议 4GB 内存以上，20GB 硬盘空间）。
3. **安装 Docker 和开发工具：**
  - 在虚拟机中安装 Docker 和你的开发工具。
4. **配置网络共享：**
  - 配置虚拟机的网络为桥接模式，方便 Windows 和虚拟机之间的互联。

### 优势：

- 模拟与公司相似的开发环境。
- 可以同时运行 Windows 和 Linux。

---

## 方法 3：双系统安装

如果需要更高性能且愿意重启切换系统，可以考虑安装 Windows + Ubuntu 双系统。

### 配置步骤：

按照 [双系统安装方法](#) 安装 Ubuntu。

- 在 Ubuntu 中直接进行 Docker 开发。

### 优势：

- 性能接近原生。
- 开发环境完全与 Linux 系统一致。



## 方法 4：使用外部设备

如果你不想改动当前笔电的配置，可以使用以下方法：

### 1. USB 启动盘运行 Linux：

- 制作一个 Ubuntu Live USB 启动盘。
- 在 BIOS 中选择从 USB 启动。
- 在 USB 环境中安装开发工具并进行开发。

### 2. 云服务器：

- 租用一个云服务器（如 AWS、Google Cloud、阿里云）。
- 在云服务器中安装 Docker 和开发工具。
- 使用 Windows 的 SSH 客户端（如 PuTTY 或 WSL 中的 `ssh` 命令）连接到服务器进行开发。

## 推荐方案

1. **初学者或轻量开发**：WSL2。
2. **模拟公司环境**：虚拟机。
3. **需要高性能或完整 Linux**：双系统或 USB 启动盘。
4. **预算充足或需要远程支持**：云服务器。

如需详细指导任何一种方法，请告诉我！



詳細比較方法一、二



以下是 **方法一（WSL2）** 和 **方法二（虚拟机）** 的详细比较，从多方面分析，帮助你选择适合自己的开发环境：

## 1. 性能

### WSL2：

- 性能接近原生 Linux，直接利用 Windows 的硬件资源。
- 启动速度快，几乎是秒级。
- 资源分配动态调整，不需要手动设置内存和硬盘大小。

### 虚拟机：

- 性能略逊于 WSL2，因为虚拟机需要通过 Hypervisor 模拟硬件。





- 启动较慢，尤其是当虚拟机分配的资源较多时。
- 需要手动设置资源（如内存、CPU、硬盘），可能导致资源浪费或不足。

**结论：**性能方面 WSL2 更优。

---

## 2. 使用难度

### WSL2：

- 安装简单，通过 Microsoft Store 安装 Ubuntu 即可。
- 与 Windows 的文件和工具集成度高，Windows 和 Linux 文件可以直接互通。
- 对新手友好，无需考虑复杂的网络和硬件配置。

### 虚拟机：

- 安装和配置需要更多时间，包括下载虚拟机软件、安装 Ubuntu、配置网络等。
- 文件共享需要额外配置（如安装 VirtualBox Guest Additions 或 VMware Tools）。
- 虚拟机可能需要额外的硬件支持（如启用 BIOS 中的虚拟化）。

**结论：**WSL2 使用难度更低。

---

## 3. 功能和兼容性

### WSL2：

- 支持绝大多数 Linux 命令行工具和 Docker。
- 不能直接运行完整的 Linux 桌面环境，但可以通过 X Server 实现图形界面（需要额外配置）。
- 与 Windows 系统深度集成，可轻松调用 Windows 应用程序和文件。

### 虚拟机：

- 提供完整的 Linux 桌面和图形环境。
- 可以完全模拟公司的开发环境，包括 GUI 工具和网络配置。
- 不依赖 Windows，与底层硬件的兼容性更高。

**结论：**如果只需要命令行，WSL2 足够；需要完整 Linux 环境时选择虚拟机。

---

## 4. 硬件资源利用

### WSL2：

- 动态使用硬件资源，不需要提前分配固定内存和存储。
- 对笔电性能要求较低，适合只有单台 Windows 笔电的情况。



### 虚拟机：

- 需要手动分配资源（如 CPU、内存、硬盘空间），可能导致资源浪费或不足。
- 对硬件性能要求较高，低配置的笔电可能会卡顿。

**结论：**WSL2 更高效地利用硬件资源。

---

## 5. 文件和网络共享

### WSL2：

- Windows 和 WSL2 文件系统无缝共享，无需额外配置。
- 网络共享默认启用，Windows 和 WSL2 可以直接互相访问。

### 虚拟机：

- 文件共享需要安装额外工具（如 VirtualBox Guest Additions）。
- 虚拟机的网络需要配置桥接模式或 NAT 模式，设置稍复杂。

**结论：**WSL2 在文件和网络共享方面更方便。

---

## 6. Docker 支持

### WSL2：

- 直接支持 Docker，通过 Docker Desktop 集成。
- 性能高，支持 Docker 容器的本地运行和开发。

### 虚拟机：

- Docker 支持良好，但需要额外安装 Docker。
- 如果虚拟机性能不足，可能会影响 Docker 容器的运行速度。

**结论：**两者都支持 Docker，但 WSL2 配置更简单。

---

## 7. 灵活性

### WSL2：

- 仅限于命令行环境（可以通过额外工具实现部分 GUI）。
- 无法模拟完整的 Linux 桌面或公司网络环境。

### 虚拟机：

- 提供完整 Linux 桌面体验。
- 可以完全模拟公司开发环境，包括网络设置和硬件行为。



结论：虚拟机更灵活，但 WSL2 更轻量。

## 8. 使用场景

场景	推荐方法
命令行开发和 Docker 镜像构建	WSL2
需要完整 Linux 桌面和 GUI 工具	虚拟机
笔电性能有限	WSL2
模拟公司开发环境	虚拟机
快速切换 Windows 和 Linux 环境	WSL2

## 总结

- **选择 WSL2**：如果你只需要命令行环境来开发 Docker 镜像，并且希望简单快速，WSL2 是更好的选择。
- **选择虚拟机**：如果需要完整的 Linux 桌面环境，并且硬件性能允许，虚拟机更适合。

如果你对任何一种方法的配置有疑问，可以告诉我，我会为你提供详细步骤！



教我使用WSL2



以下是详细的 **Windows 10 配置 WSL2** 的步骤，让你可以轻松在 Windows 上运行 Ubuntu，并使用 Docker 开发环境：

## 步骤 1：启用 WSL2

1. 打开 **Windows PowerShell**（管理员权限）：
  - 右键点击开始菜单，选择“Windows PowerShell（管理员）”。
2. 启用 **WSL 功能**：在 PowerShell 中运行以下命令：

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

- 这会启用 WSL 和虚拟机平台功能。
3. 重启电脑：







- 完成上述命令后，按照提示重启电脑。

#### 4. 设置默认 WSL 版本为 WSL2：

- 在重启后再次打开 PowerShell（管理员权限），运行：

```
wsl --set-default-version 2
```

---

## 步骤 2：安装 Ubuntu

### 1. 打开 Microsoft Store：

- 搜索“Ubuntu”，选择一个版本（例如 **Ubuntu 22.04 LTS**）。
- 点击“获取”按钮并安装。

### 2. 启动 Ubuntu：

- 安装完成后，从开始菜单中打开 Ubuntu。
- 首次启动会初始化 Linux 文件系统，可能需要几分钟。

### 3. 设置用户名和密码：

- 按提示设置一个 Linux 用户名和密码。
- 这将是你在 WSL2 中的默认用户。

---

## 步骤 3：安装 Docker（可选）

### 1. 安装 Docker Desktop：

- 从 [Docker 官方网站](#) 下载并安装 Docker Desktop。
- 在安装过程中，确保启用 **WSL2 集成**。

### 2. 启用 WSL2 与 Docker 集成：

- 打开 Docker Desktop → 设置 → Resources → WSL Integration。
- 勾选安装的 Ubuntu 版本。

### 3. 在 Ubuntu 中测试 Docker：

- 打开 Ubuntu，运行以下命令测试：

```
docker --version
```

- 如果安装成功，你将看到 Docker 的版本信息。



## 步骤 4：安装开发工具

1. **更新系统**：在 Ubuntu 中运行：

```
sudo apt update && sudo apt upgrade -y
```

2. **安装常用开发工具**：

- 安装 Git：

```
sudo apt install git -y
```

- 安装 Docker Compose：

```
sudo apt install docker-compose -y
```

3. **设置文件共享**：

- WSL 的文件位于 `\\wsl\$` 路径下，可以通过文件资源管理器直接访问。

---

## 步骤 5：运行你的开发环境

1. **启动 Docker 容器**：

- 使用 `docker run` 启动你的开发环境。例如：

```
docker run -it ubuntu:latest /bin/bash
```

- 这会运行一个基础的 Ubuntu 容器。

2. **使用 VSCode 集成开发**：

- 安装 VSCode 和 **Remote - WSL** 扩展。
- 在 VSCode 中点击左下角绿色按钮，选择“Remote - WSL”。
- 选择你的 Ubuntu 环境，即可直接在 WSL 中进行开发。

---

## 步骤 6：验证和优化

1. **验证 WSL2 环境**：在 PowerShell 中运行以下命令，查看当前 WSL 的状态：

```
wsl --list --verbose
```

- 确保 Ubuntu 显示为版本 2。

2. **优化性能**：





- 在用户主目录下创建 `.wslconfig` 文件，配置资源限制（例如内存和 CPU）：

```
[wsl2]
memory=4GB # 最大内存
processors=2 # CPU 核心数
```

- 保存后重启 WSL：

```
wsl --shutdown
```

## 优势与提醒

- 优点：**性能高，与 Windows 集成度高，轻量级，适合 Docker 开发。
- 提醒：**WSL2 不支持完整的 Linux 桌面环境，仅支持命令行。如果需要图形界面，可以使用额外的 X Server 软件。

如有任何问题或需要额外说明，随时告诉我！



詳細說明步骤 5、6



## 步骤 5：运行你的开发环境（详细步骤）

在这一步中，你将学会如何在 WSL2 中运行 Docker 容器并设置适合开发的工作流。

### 5.1 启动 Docker 容器

在 WSL2 的 Ubuntu 中，使用 Docker 来运行你的开发环境。

1. **拉取基础镜像：**首先，确保你的 Ubuntu 环境可以访问 Docker。拉取一个常见的基础镜像（例如 Ubuntu）：

```
docker pull ubuntu:latest
```

- 这会下载最新的 Ubuntu 镜像到你的系统。

2. **运行容器：**使用 `docker run` 命令启动一个容器：

```
docker run -it ubuntu:latest /bin/bash
```

- `-it`：交互模式，允许你与容器进行交互。
- `ubuntu:latest`：指定使用的镜像。
- `/bin/bash`：运行 Bash Shell。

3. **验证容器是否运行：**在容器中运行以下命令查看版本或测试开发环境：



```
lsb_release -a
```

- 如果看到 Ubuntu 的版本信息，则说明容器运行正常。

4. **列出正在运行的容器**：在容器运行时，你可以在 Ubuntu 中运行以下命令查看容器状态：

```
docker ps
```

## 5.2 挂载项目目录

为了方便开发，可以将 Windows 的文件夹挂载到 Docker 容器中。

1. **创建一个 Windows 项目目录**：在 Windows 文件系统中创建一个文件夹，例如：

```
C:\Projects\MyDockerProject
```

2. **运行容器并挂载目录**：使用 `-v`` 参数挂载 Windows 文件夹到容器中。例如：

```
docker run -it -v /mnt/c/Projects/MyDockerProject:/workspace ubuntu:latest /bin/bash
```

- ``/mnt/c/Projects/MyDockerProject``：Windows 路径的 WSL 表示法。
- ``/workspace``：容器中的挂载点。

3. **在容器中查看挂载文件夹**：进入容器后运行：

```
ls /workspace
```

- 你应该可以看到挂载的 Windows 文件夹中的内容。

## 5.3 使用 VSCode 进行开发

Visual Studio Code 可以无缝集成到 WSL 和 Docker 容器中。

1. **安装必要扩展**：在 VSCode 的扩展商店中安装以下扩展：

- **Remote - WSL**：用于直接在 WSL 中开发。
- **Docker**：用于管理 Docker 容器。

2. **打开 WSL 环境**：在 VSCode 中点击左下角的绿色按钮，选择“Remote-WSL”。

- 选择你的 Ubuntu 环境。

3. **打开挂载的文件夹**：在 VSCode 中，打开 ``/mnt/c/Projects/MyDockerProject`` 路径进行开发。

4. **运行代码**：你可以直接在 VSCode 中修改挂载目录中的文件，并使用 Docker 容器运行或调试代码。



## 步骤 6：验证和优化（详细步骤）

这一步中，你将验证 WSL2 的配置，并通过优化配置文件来提升性能和开发体验。

### 6.1 验证 WSL2 环境

1. **检查 WSL 状态**：在 PowerShell 中运行以下命令：

```
wsl --list --verbose
```

- 输出示例：

NAME	STATE	VERSION
* Ubuntu	Running	2

确保：

- 你的 Ubuntu 版本显示为 **2**（即 WSL2）。
- 状态为 **Running**。

2. **检查 Docker 状态**：在 Ubuntu 中运行以下命令：

```
docker info
```

- 检查输出中的 **Cgroup Driver** 是否为 ``cgroup2``。
- 如果 Docker 正常运行，则说明环境配置成功。

### 6.2 优化 WSL2 的资源配置

默认情况下，WSL2 动态分配资源，但你可以手动限制资源使用。

1. **创建 `.wslconfig` 文件**：在 Windows 用户主目录下（通常是 ``C:\Users\<你的用户名>``）创建一个 `.wslconfig` 文件。
2. **配置资源限制**：在文件中添加以下内容：

```
[wsl2]
memory=4GB          # 最大内存为 4GB
processors=2         # 使用 2 个 CPU 核心
swap=2GB            # 交换分区大小为 2GB
localhostForwarding=true # 启用本地主机转发
```

- 根据你的笔电硬件性能调整 ``memory`` 和 ``processors``。

3. **重启 WSL**：在 PowerShell 中运行以下命令以应用更改：



```
wsl --shutdown
```

- 然后重新打开 Ubuntu 环境。

### 6.3 优化文件访问性能

WSL2 默认情况下，访问 Windows 文件系统（例如 `/mnt/c`）的速度较慢。为提高性能，尽量将项目代码存放在 WSL 的 Linux 文件系统中。

1. **创建 WSL 内部目录：**在 Ubuntu 中创建一个开发目录，例如：

```
mkdir ~/workspace
```

2. **将项目文件复制到 WSL 内部：**使用 `cp` 命令将 Windows 文件夹的内容复制到 WSL：

```
cp -r /mnt/c/Projects/MyDockerProject ~/workspace/
```

3. **在 WSL 内部开发：**使用 VSCode 打开 WSL 内部目录进行开发，性能会显著提升。

### 6.4 验证 Docker 和 WSL 的集成

1. **测试 Docker 容器与网络连接：**启动一个容器并验证网络访问是否正常：

```
docker run -it ubuntu:latest ping google.com
```

- 如果可以 ping 通，则网络配置正常。

2. **测试容器间通信：**启动两个容器，验证它们是否可以互相访问。例如：

```
docker network create mynetwork
docker run -it --name container1 --network mynetwork ubuntu:latest /bin/bash
docker run -it --name container2 --network mynetwork ubuntu:latest /bin/bash
```

- 在 `container1` 中运行：

```
ping container2
```

通过这些详细步骤，你可以在 WSL2 中创建高效的 Docker 开发环境，并优化资源使用和性能。如果需要进一步帮助，可以告诉我！





Installing, this may take a few minutes...

WslRegisterDistribution failed with error: 0x800701bc

Error: 0x800701bc WSL 2 ?????????????????? visit <https://aka.ms/wsl2kernel>



这个错误（`0x800701bc`）表明你的 Windows 系统需要安装或更新 WSL2 的内核组件。这是一个常见问题，可以通过以下步骤解决：

## 解决步骤

### 1. 下载并更新 WSL2 内核组件

1. 打开浏览器访问以下链接：[WSL2 Linux 内核更新包](#)
2. 下载并运行安装包：
  - 下载完成后，双击运行安装包。
  - 按照提示完成安装。

### 2. 确保 Windows 已更新

1. 按下 `Win + I` 打开 **设置**。
2. 进入 **更新和安全** → **Windows 更新**。
3. 点击 **检查更新**，安装所有可用更新。
  - 如果你的 Windows 版本低于 1903 或 2004，则需要更新到更高版本。
  - 你可以通过运行以下命令检查版本：

```
winver
```

- 如果版本低于 1903，请通过 Windows 更新将系统更新至最新。

### 3. 确保启用了 WSL 和虚拟机平台功能

1. 打开 **PowerShell (管理员权限)**。
2. 运行以下命令：

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart  
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

3. 重启电脑以应用更改。





## 4. 设置 WSL 的默认版本为 WSL2

在重启后，打开 **PowerShell (管理员权限)** 并运行以下命令：

```
wsl --set-default-version 2
```

## 5. 再次运行 Ubuntu

1. 重新打开安装的 Ubuntu 应用。
2. 如果问题解决，Ubuntu 应该会正常初始化。

## 检查和验证

1. 检查 WSL 的状态：

```
wsl --list --verbose
```

- 确保 Ubuntu 的版本为 **2**。
- 如果仍然是 **1**，可以手动切换版本：

```
wsl --set-version Ubuntu 2
```

2. 确保 WSL2 内核已成功安装：

- 如果问题未解决，请重新运行内核更新包并重启电脑。

## 提示

- 如果你正在使用公司电脑，确保管理员权限可以修改系统配置。
- 如果问题依然存在，请告诉我你的具体 Windows 版本和配置（例如：`winver` 命令输出）。我可以进一步提供支持。



### Docker Desktop WSL 2 backend on Windows

Windows Subsystem for Linux (WSL) 2 is a full Linux kernel built by Microsoft, which lets Linux distributions run without managing virtual machines. With Docker Desktop running on WSL 2, users can leverage Linux workspaces and avoid maintaining both Linux and Windows build scripts. In addition, WSL 2 provides improvements to file system sharing and boot time.

Docker Desktop uses the dynamic memory allocation feature in WSL 2 to improve the resource consumption. This means Docker Desktop only uses the required amount of CPU and memory resources it needs, while allowing CPU and memory-intensive tasks such as building a container, to run much faster.





Additionally, with WSL 2, the time required to start a Docker daemon after a cold start is significantly faster.

#### Prerequisites

Before you turn on the Docker Desktop WSL 2 feature, ensure you have:

At a minimum WSL version 1.1.3.0., but ideally the latest version of WSL to avoid Docker Desktop not working as expected.

Met the Docker Desktop for Windows' system requirements.

Installed the WSL 2 feature on Windows. For detailed instructions, refer to the Microsoft documentation.

#### Tip

For a better experience on WSL, consider enabling the WSL autoMemoryReclaim setting available since WSL 1.3.10 (experimental).

This feature enhances the Windows host's ability to reclaim unused memory within the WSL virtual machine, ensuring improved memory availability for other host applications. This capability is especially beneficial for Docker Desktop, as it prevents the WSL VM from retaining large amounts of memory (in GBs) within the Linux kernel's page cache during Docker container image builds, without releasing it back to the host when no longer needed within the VM.

#### Turn on Docker Desktop WSL 2

##### Important

To avoid any potential conflicts with using WSL 2 on Docker Desktop, you must uninstall any previous versions of Docker Engine and CLI installed directly through Linux distributions before installing Docker Desktop.

Download and install the latest version of Docker Desktop for Windows.

Follow the usual installation instructions to install Docker Desktop. Depending on which version of Windows you are using, Docker Desktop may prompt you to turn on WSL 2 during installation. Read the information displayed on the screen and turn on the WSL 2 feature to continue.

Start Docker Desktop from the Windows Start menu.

Navigate to Settings.

From the General tab, select Use WSL 2 based engine..

If you have installed Docker Desktop on a system that supports WSL 2, this option is turned on by default.

Select Apply & Restart.

Now docker commands work from Windows using the new WSL 2 engine.

#### Tip

By default, Docker Desktop stores the data for the WSL 2 engine at C:\Users\[USERNAME]\AppData\Local\Docker\wsl. If you want to change the location, for example, to another drive you can do so via the Settings -> Resources -> Advanced page from the Docker Dashboard. Read more about this and other Windows settings at [Changing settings](#)

#### Enabling Docker support in WSL 2 distributions

WSL 2 adds support for "Linux distributions" to Windows, where each distribution



behaves like a VM except they all run on top of a single shared Linux kernel.

Docker Desktop does not require any particular Linux distributions to be installed. The docker CLI and UI all work fine from Windows without any additional Linux distributions. However for the best developer experience, we recommend installing at least one additional distribution and enable Docker support:

Ensure the distribution runs in WSL 2 mode. WSL can run distributions in both v1 or v2 mode.

To check the WSL mode, run:

```
wsl.exe -l -v
```

To upgrade the Linux distribution to v2, run:

```
wsl.exe --set-version (distribution name) 2
```

To set v2 as the default version for future installations, run:

```
wsl.exe --set-default-version 2
```

When Docker Desktop starts, go to Settings > Resources > WSL Integration.

The Docker-WSL integration is enabled on the default WSL distribution, which is Ubuntu. To change your default WSL distribution, run:

```
wsl --set-default <distribution name>
```

If WSL integrations isn't available under Resources, Docker may be in Windows container mode. In your taskbar, select the Docker menu and then Switch to Linux containers.

Select Apply & Restart.

#### Note

With Docker Desktop version 4.30 and earlier, Docker Desktop installed two special-purpose internal Linux distributions docker-desktop and docker-desktop-data. docker-desktop is used to run the Docker engine dockerd, while docker-desktop-data stores containers and images. Neither can be used for general development.

With fresh installations of Docker Desktop 4.30 and later, docker-desktop-data is no longer created. Instead, Docker Desktop creates and manages its own virtual hard disk for storage. The docker-desktop distribution is still created and used to run the Docker engine.

Note that Docker Desktop version 4.30 and later keeps using the docker-desktop-data distribution if it was already created by an earlier version of Docker Desktop and has not been freshly installed or factory reset.

#### Additional resources

Explore best practices

Understand how to develop with Docker and WSL 2

Learn about GPU support with WSL 2



以下是针对 **Docker Desktop WSL 2 后端** 在 **Windows** 上使用 的详细说明和步骤总

结：



---

## 安装和配置 Docker Desktop WSL 2 后端

### 前提条件

在开始前，请确保：

#### 1. WSL 和 Docker 兼容：

- WSL 至少版本为 **1.1.3.0**（建议最新版本）。
- 已满足 [Docker Desktop 系统要求](#)。
- 在 Windows 上启用了 WSL 2。

#### 2. 推荐优化（可选）：

- 启用 `autoMemoryReclaim`（适用于 WSL 1.3.10 及更高版本），帮助回收未使用的内存，提升性能。

---

### 步骤 1：安装 Docker Desktop

#### 1. 下载 Docker Desktop：

- 访问 [Docker Desktop 官方下载页面](#) 并下载最新版本。

#### 2. 安装 Docker Desktop：

- 按照安装向导进行操作。
- 如果系统支持 WSL 2，安装过程中可能会提示启用 WSL 2 后端，请选择启用。

#### 3. 完成安装：

- 安装完成后，重启系统。

---

### 步骤 2：启用 WSL 2 后端

#### 1. 启动 Docker Desktop：

- 从 Windows 开始菜单启动 Docker Desktop。

#### 2. 配置 WSL 2 后端：

- 在 Docker Desktop 中，导航至 **Settings > General**。
- 勾选 **Use WSL 2 based engine**。

#### 3. 应用更改：

- 点击 **Apply & Restart**，Docker 将切换至 WSL 2 后端。



## 步骤 3：验证和测试 WSL 2 配置

1. **检查 WSL 版本：** 打开 PowerShell，运行：

```
wsl --list --verbose
```

- 确保分发版（如 Ubuntu）运行在 WSL 2 模式下。

输出示例：

NAME	STATE	VERSION
* Ubuntu	Running	2

如果版本为 **1**，可以运行以下命令升级到 WSL 2：

```
wsl --set-version Ubuntu 2
```

2. **启用 WSL 与 Docker 集成：**

- 在 Docker Desktop 中导航至 **Settings > Resources > WSL Integration**。
- 勾选要与 Docker 集成的分发版（如 Ubuntu）。
- 点击 **Apply & Restart**。

---

## 步骤 4：运行和测试 Docker

1. **运行基础镜像容器：** 打开 WSL 的 Linux 环境（如 Ubuntu），运行以下命令：

```
docker run hello-world
```

- 如果成功，你会看到 "Hello from Docker!" 的输出，表明 Docker 配置正确。

2. **检查 Docker 状态：**

- 在 WSL 中运行：

```
docker info
```

- 确认输出中 Docker 引擎正常运行。

---

## 可选配置

### 1. 更改 Docker 数据存储位置

默认情况下，Docker 的数据存储在：

```
C:\Users\[USERNAME]\AppData\Local\Docker\wsl
```



如果需要更改存储位置：

1. 打开 **Settings > Resources > Advanced**。
2. 配置新的存储路径。

---

## 2. 使用 VSCode 集成 Docker 和 WSL

1. 安装 **Visual Studio Code** 和以下扩展：
  - **Remote - WSL**：用于连接 WSL 环境。
  - **Docker**：用于管理 Docker 容器。
2. 打开 WSL 环境：
  - 点击 VSCode 左下角绿色按钮，选择 **Remote-WSL**。
3. 直接在 VSCode 中管理 Docker 和开发容器化应用。

---

## 性能优化

### 启用 autoMemoryReclaim

1. 确保你的 WSL 版本为 **1.3.10** 或更高版本。
2. 在 PowerShell 中运行：

```
wsl --set-memory-reclaim enabled
```

这将优化内存回收，提升主机系统性能。

---

## 注意事项

1. **避免冲突：**
  - 如果之前通过 Linux 直接安装了 Docker Engine 和 CLI，建议卸载这些组件，以避免与 Docker Desktop 冲突。
2. **模式切换：**
  - 如果 Docker Desktop 显示为 Windows 容器模式，可在任务栏的 Docker 图标上右键，选择 **Switch to Linux containers**。
3. **docker-desktop 和 docker-desktop-data：**
  - 这些是 Docker 内部使用的分发版，不适合一般开发，请勿删除或更改。



通过这些步骤，你可以在 Windows 上使用 WSL 2 和 Docker Desktop 创建高效的开发环境。如果有任何问题，随时告诉我！



Complete the installation of Docker Desktop.

Use recommended settings (requires administrator password)  
Docker Desktop automatically sets the necessary configurations that work for most developers.

Use advanced settings  
You manually set your preferred configurations.



## 选择安装模式：推荐设置 vs 高级设置

安装 Docker Desktop 时，你会被提示选择以下两种设置模式：

### 1. 推荐设置 (Use recommended settings)：

- 自动设置所有必要的配置。
- 适合大多数开发者，不需要手动调整。
- 简化安装过程，快速上手。
- 需要管理员权限来自动启用和配置 WSL 2。

### 2. 高级设置 (Use advanced settings)：

- 手动设置所需的配置，例如：
  - 数据存储位置。
  - 网络代理。
  - WSL 集成的具体设置。
- 适合对 Docker 和系统配置有特殊需求的高级用户。
- 更灵活，但需要更多手动操作和对配置的了解。

---

## 推荐选择

如果你是首次使用 Docker Desktop，建议选择 **推荐设置 (Use recommended settings)**，因为它会自动完成以下任务：

- 启用 WSL 2 后端。
- 自动设置 Docker 与默认 WSL 分发版（如 Ubuntu）的集成。
- 配置默认的存储位置和网络设置。

---

## 使用推荐设置安装



### 1. 确认并开始安装：

- 选择 **Use recommended settings**。
- 系统可能会提示输入管理员密码，输入后点击确认。

### 2. 等待安装完成：

- Docker Desktop 会自动完成所有配置。
- 完成后，会自动启动 Docker。

### 3. 测试安装：

- 打开 WSL（如 Ubuntu），运行以下命令：

```
docker run hello-world
```

- 如果看到成功的输出，说明安装完成并配置正确。

## 使用高级设置安装

如果你选择 **高级设置 (Use advanced settings)**，请按照以下步骤配置：

### 1. 手动配置存储路径：

- 你可以选择将 Docker 数据存储在默认位置（`C:\Users\[USERNAME]\AppData\Local\Docker\wsl`）或其他磁盘路径。
- 例如，将数据存储在 `D:\DockerData`：
  - 在 **Settings > Resources > Advanced** 中更改路径。

### 2. 手动选择 WSL 集成分发版：

- 在 **Settings > Resources > WSL Integration** 中，选择需要与 Docker 集成的 Linux 分发版（如 Ubuntu）。
- 勾选对应分发版，点击 **Apply & Restart**。

### 3. 配置网络代理（如果需要）：

- 如果你在受限的网络环境下工作（如公司网络），可能需要配置代理。
- 在 **Settings > Resources > Proxies** 中设置代理服务器地址和端口。

## 总结选择

- 如果你没有特殊需求：选择 **推荐设置**。
- 如果你需要自定义配置：选择 **高级设置**，但需具备一定的 Docker 和系统配置知识。

安装完成后，如果需要进一步指导或调整设置，可以随时告诉我！





Docker Engine stopped



如果 Docker Engine 停止运行（`Docker Engine stopped`），可能是以下几种原因引起的。以下是逐步排查和解决问题的方法：

## 1. 检查 Docker Desktop 状态

### 1. 重启 Docker Desktop：

- 点击任务栏中的 Docker 图标，右键选择 **Quit Docker Desktop**。
- 重新启动 Docker Desktop。

### 2. 检查 Docker Desktop 配置：

- 打开 Docker Desktop。
- 导航到 **Settings > General**，确保勾选 **Use WSL 2 based engine**。

### 3. 验证 Docker Engine 状态：

- 在 Docker Desktop 的 **Dashboard** 查看 Docker 是否正常启动。
- 如果仍然显示 `Docker Engine stopped`，继续以下步骤。

## 2. 重启 WSL 和 Docker 服务

### 1. 关闭 WSL：在 PowerShell（管理员权限）中运行：

```
wsl --shutdown
```

### 2. 重启 Docker 服务：打开 PowerShell（管理员权限）并运行：

```
Restart-Service com.docker.service
```

### 3. 重新启动 WSL 和 Docker：

- 启动 Docker Desktop，检查问题是否解决。

## 3. 检查系统资源和配置

### 1. 检查 WSL2 的资源分配：如果系统资源不足，Docker Engine 可能无法启动。

- 检查是否在主目录下创建了 `.wslconfig` 文件。
- 配置示例：





```
[ws12]
memory=4GB          # 限制内存为 4GB
processors=2         # 使用 2 个 CPU 核心
swap=2GB            # 交换分区大小为 2GB
localhostForwarding=true
```

- 修改后，重启 WSL：

```
wsl --shutdown
```

## 2. 检查磁盘空间：

- 确保 Docker 的存储目录（默认在 `C:\Users\[USERNAME]\AppData\Local\Docker\ws1`）有足够的磁盘空间。

---

## 4. 检查是否存在冲突的 Docker 安装

1. **卸载本地安装的 Docker（如果有）：** 如果你之前在 WSL 中直接安装了 Docker Engine 和 CLI，可能与 Docker Desktop 冲突。

- 进入 WSL 环境（如 Ubuntu）并运行以下命令：

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

2. **清理残留配置文件：**

```
sudo rm -rf /var/lib/docker
sudo rm -rf /etc/docker
```

3. **重启 Docker Desktop。**

---

## 5. 查看日志进行深入排查

1. **查看 Docker Desktop 日志：**

- 在 Docker Desktop 中，导航到 **Troubleshoot > Support**。
- 点击 **Get Support** 导出日志文件，查看是否有错误提示。

2. **查看 WSL2 日志：**

- 打开 PowerShell，运行以下命令：

```
wsl --list --verbose
```

确保分发版（如 Ubuntu）运行在 **VERSION=2**。

- 如果 WSL 分发版未启动或异常，可尝试重新安装：



```
wsl --unregister Ubuntu
wsl --install -d Ubuntu
```

## 6. 重装 Docker Desktop（最后手段）

如果以上步骤都无法解决问题，可以尝试完全重装 Docker Desktop。

### 1. 卸载 Docker Desktop：

- 打开 设置 > 应用 > 已安装的应用。
- 找到 Docker Desktop 并卸载。

### 2. 清理 Docker 配置文件：

- 删除以下目录：

```
C:\Users\[USERNAME]\AppData\Local\Docker
C:\Users\[USERNAME]\.docker
```

### 3. 重新安装 Docker Desktop：

- 下载最新版本并安装。

如果经过以上步骤问题仍未解决，可以将具体错误信息或日志发给我，我将进一步帮助你分析！



```
root@LAPTOP-M3KHGG4S:~/dev_code# docker build -t fastapi-python-mariadb .
[+] Building 380.7s (7/12)
docker:default
=> [internal] load build definition from Dockerfile
0.0s
=> => transferring dockerfile: 1.35kB
0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim
2.6s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 2B
0.0s
=> [1/8] FROM docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cf 50.3s
=> => resolve docker.io/library/python:3.9-slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a7166951cfe 0.0s
=> =>
sha256:d381bf0bfd6e4697fb23da0e7d50d080411236ffcd2853432e12fd0f6b372778
248B / 248B 0.6s
```





```
=> =>
sha256:75a2bc32319e3d6a5bc12888b074f86338a9e3a4304d99d1dddc07155b8ba7
6e 14.93MB / 14.93MB          34.3s
=> =>
sha256:96df0e5e81799ba220e250fc3d2c1da017b41302df5954c705bece1407dcab03
3.32MB / 3.32MB              14.9s
=> =>
sha256:fd674058ff8f8cfa7fb8a20c006fc0128541cbbad7f7f7f28df570d08f9e4d92
28.23MB / 28.23MB           48.3s
=> => extracting
sha256:fd674058ff8f8cfa7fb8a20c006fc0128541cbbad7f7f7f28df570d08f9e4d92
1.1s
=> => extracting
sha256:96df0e5e81799ba220e250fc3d2c1da017b41302df5954c705bece1407dcab03
0.2s
=> => extracting
sha256:75a2bc32319e3d6a5bc12888b074f86338a9e3a4304d99d1dddc07155b8ba7
6e 0.6s
=> => extracting
sha256:d381bf0bfd6e4697fb23da0e7d50d080411236ffcd2853432e12fd0f6b372778
0.0s
=> [internal] load build context
0.0s
=> => transferring context: 92B
0.0s
=> [2/8] RUN apt-get update && apt-get install -y mariadb-server mariadb-
client libmariadb-dev gcc 322.3s
=> ERROR [3/8] RUN mysqld --initialize-insecure --user=mysql && mysqld --skip-
networking & sleep 5 && mysq 5.4s
-----
> [3/8] RUN mysqld --initialize-insecure --user=mysql && mysqld --skip-networking
& sleep 5 && mysql -u root -e "CREATE DATABASE fastapi_crud" && mysql -u
root -e "CREATE USER 'fastapi_user'@'%' IDENTIFIED BY 'fastapi_password'" &&
mysql -u root -e "GRANT ALL PRIVILEGES ON fastapi_crud.* TO 'fastapi_user'@'%' &&
mysql -u root -e "FLUSH PRIVILEGES":
0.427 2025-01-05 10:32:15 0 [Note] Starting MariaDB 10.11.6-MariaDB-0+deb12u1
source revision as process 9
0.451 2025-01-05 10:32:15 0 [Note] InnoDB: Compressed tables use zlib 1.2.13
0.453 2025-01-05 10:32:15 0 [Note] InnoDB: Number of transaction pools: 1
0.453 2025-01-05 10:32:15 0 [Note] InnoDB: Using crc32 + pclmulqdq instructions
0.453 2025-01-05 10:32:15 0 [Note] mysqld: O_TMPFILE is not supported on /tmp
(disabling future attempts)
0.454 2025-01-05 10:32:15 0 [Warning] mysqld: io_uring_queue_init() failed with
errno 95
0.454 2025-01-05 10:32:15 0 [Warning] InnoDB: liburing disabled: falling back to
innodb_use_native_aio=OFF
0.456 2025-01-05 10:32:15 0 [Note] InnoDB: Initializing buffer pool, total size =
128.000MiB, chunk size = 2.000MiB
0.484 2025-01-05 10:32:15 0 [Note] InnoDB: Completed initialization of buffer pool
0.681 2025-01-05 10:32:15 0 [Note] InnoDB: Buffered log writes (block size=512
bytes)
0.696 2025-01-05 10:32:15 0 [Note] InnoDB: End of log at LSN=46996
0.725 2025-01-05 10:32:15 0 [Note] InnoDB: 128 rollback segments are active.
0.726 2025-01-05 10:32:15 0 [Note] InnoDB: Setting file './ibtmp1' size to 12.000MiB.
Physically writing the file full; Please wait ...
0.727 2025-01-05 10:32:15 0 [Note] InnoDB: File './ibtmp1' size is now 12.000MiB.
0.729 2025-01-05 10:32:15 0 [Note] InnoDB: log sequence number 46996;
transaction id 14
0.729 2025-01-05 10:32:15 0 [Note] Plugin 'FEEDBACK' is disabled.
0.729 2025-01-05 10:32:15 0 [Note] InnoDB: Loading buffer pool(s) from
/var/lib/mysql/ib_buffer_pool
```



```
0.735 2025-01-05 10:32:15 0 [ERROR] mysqld: unknown option '--initialize-insecure'
0.735 2025-01-05 10:32:15 0 [ERROR] Aborting
5.392 ERROR 2002 (HY000): Can't connect to local server through socket
'/run/mysqld/mysqld.sock' (2)
-----

2 warnings found (use docker --debug to expand):
- SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV
"MARIADB_ROOT_PASSWORD") (line 15)
- SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV
"MARIADB_PASSWORD") (line 18)
Dockerfile:21
-----
20 | # 初始化 MariaDB 数据库
21 | >>> RUN mysqld --initialize-insecure --user=mysql && \
22 | >>> mysqld --skip-networking & \
23 | >>> sleep 5 && \
24 | >>> mysql -u root -e "CREATE DATABASE ${MARIADB_DATABASE}" && \
25 | >>> mysql -u root -e "CREATE USER '${MARIADB_USER}'@ '%' IDENTIFIED BY
'${MARIADB_PASSWORD}'" && \
26 | >>> mysql -u root -e "GRANT ALL PRIVILEGES ON ${MARIADB_DATABASE}.*
TO '${MARIADB_USER}'@ '%' && \
27 | >>> mysql -u root -e "FLUSH PRIVILEGES"
28 |
-----
ERROR: failed to solve: process "/bin/sh -c mysqld --initialize-insecure --user=mysql
&& mysqld --skip-networking & sleep 5 && mysql -u root -e \"CREATE
DATABASE ${MARIADB_DATABASE}\" && mysql -u root -e \"CREATE USER
'${MARIADB_USER}'@ '%' IDENTIFIED BY '${MARIADB_PASSWORD}'\" && mysql -u
root -e \"GRANT ALL PRIVILEGES ON ${MARIADB_DATABASE}.* TO
'${MARIADB_USER}'@ '%'\" && mysql -u root -e \"FLUSH PRIVILEGES\"" did not
complete successfully: exit code: 1
root@LAPTOP-M3KHGG4S:~/dev_code#
```



## 问题分析

根据错误信息：

1. ``mysqld: unknown option '--initialize-insecure'``
  - ``--initialize-insecure`` 是用于初始化 MariaDB 数据库的选项，但当前 MariaDB 版本可能不支持此选项。
2. ``Can't connect to local server through socket '/run/mysqld/mysqld.sock'``
  - MariaDB 服务未能成功启动，导致无法通过 ``mysql`` 命令连接。
3. **Dockerfile 中的环境变量问题：**

警告显示敏感数据（如密码）被直接定义在 Dockerfile 中，这可能存在安全隐患。

## 解决方案

### 1. 修改 ``mysqld`` 初始化方式

MariaDB 的初始化可能与版本有关，请调整初始化逻辑：





更新 **Dockerfile** 中的第 3 阶段指令：

```
RUN mariadb-install-db --user=mysql --datadir=/var/lib/mysql && \
    mysqld --skip-networking & \
    sleep 5 && \
    mysql -u root -e "CREATE DATABASE ${MARIADB_DATABASE}" && \
    mysql -u root -e "CREATE USER '${MARIADB_USER}'@ '%' IDENTIFIED BY '${MARIADB_PASSWORD}'" && \
    mysql -u root -e "GRANT ALL PRIVILEGES ON ${MARIADB_DATABASE}.* TO '${MARIADB_USER}'@ '%' && \
    mysql -u root -e "FLUSH PRIVILEGES"
```

解释：

- 使用 `mariadb-install-db` 初始化数据目录。
- `mysqld --skip-networking` 启动 MariaDB 服务以避免外部连接。

## 2. 优化敏感数据的传递方式

为了避免在 Dockerfile 中暴露敏感信息（如密码），将敏感数据通过 Docker 构建参数传递：

更新 **Dockerfile**：

```
# 定义环境变量，接收外部参数
ARG MARIADB_DATABASE
ARG MARIADB_USER
ARG MARIADB_PASSWORD

# 将这些变量设置为环境变量
ENV MARIADB_DATABASE=${MARIADB_DATABASE}
ENV MARIADB_USER=${MARIADB_USER}
ENV MARIADB_PASSWORD=${MARIADB_PASSWORD}
```

构建命令：

```
docker build --build-arg MARIADB_DATABASE=fastapi_crud \
    --build-arg MARIADB_USER=fastapi_user \
    --build-arg MARIADB_PASSWORD=fastapi_password \
    -t fastapi-python-mariadb .
```

## 3. 确保依赖和服务顺序正确

Docker 构建过程中，MariaDB 的依赖和启动顺序可能需要微调。可考虑在构建镜像时，仅准备数据库文件夹，运行容器时再执行数据库初始化。

调整 **Dockerfile**：

```
# 初始化数据库目录，仅在运行容器时进行初始化
COPY init.sql /docker-entrypoint-initdb.d/
```

创建 `init.sql` 文件：



```
CREATE DATABASE IF NOT EXISTS fastapi_crud;  
CREATE USER IF NOT EXISTS 'fastapi_user'@'%' IDENTIFIED BY 'fastapi_password';  
GRANT ALL PRIVILEGES ON fastapi_crud.* TO 'fastapi_user'@'%';  
FLUSH PRIVILEGES;
```

MariaDB 会在容器启动时自动运行 `/docker-entrypoint-initdb.d/` 中的 SQL 脚本。

#### 4. 验证修复后的构建过程

##### 1. 清理旧的构建缓存：

```
docker builder prune -f
```

##### 2. 重新构建镜像：

```
docker build --build-arg MARIADB_DATABASE=fastapi_crud \  
--build-arg MARIADB_USER=fastapi_user \  
--build-arg MARIADB_PASSWORD=fastapi_password \  
-t fastapi-python-mariadb .
```

##### 3. 运行容器并验证：

```
docker run -d --name mariadb-test \  
-e MYSQL_ROOT_PASSWORD=root_password \  
-p 3306:3306 fastapi-python-mariadb
```

##### 4. 检查日志和连接：查看 MariaDB 是否正常启动：

```
docker logs mariadb-test
```

测试连接：

```
docker exec -it mariadb-test mysql -u root -p
```

通过上述调整，你的 MariaDB 应该可以正确初始化并运行。如果问题依然存在，请提供更多日志或信息。



```
root@LAPTOP-M3KHGG4S:~/dev_code# docker build -t fastapi-python-mariadb .  
[+] Building 45.9s (12/12) FINISHED  
docker:default  
=> [internal] load build definition from Dockerfile  
0.0s  
=> => transferring dockerfile: 1.03kB  
0.0s  
=> [internal] load metadata for docker.io/library/python:3.9-slim
```



```
1.0s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 2B
0.0s
=> [1/7] FROM docker.io/library/python:3.9-
slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a716695
1cfea      0.0s
=> => resolve docker.io/library/python:3.9-
slim@sha256:caaf1af9e23adc6149e5d20662b267ead9505868ff07c7673dc4a716695
1cfea      0.0s
=> [internal] load build context
0.0s
=> => transferring context: 92B
0.0s
=> CACHED [2/7] RUN apt-get update && apt-get install -y mariadb-server
mariadb-client libmariadb-dev gcc libffi- 0.0s
=> [3/7] COPY init.sql /docker-entrypoint-initdb.d/
0.0s
=> [4/7] COPY requirements.txt .
0.0s
=> [5/7] RUN pip install --no-cache-dir -r requirements.txt
19.1s
=> [6/7] WORKDIR /app
0.1s
=> [7/7] COPY main.py /app/
0.0s
=> exporting to image
25.4s
=> => exporting layers
20.8s
=> => exporting manifest
sha256:fe4dd0209a1c4c6c9b46b2ccaaf75900d2e2380d9dcf7b31d6c2aa54373233c8
0.0s
=> => exporting config
sha256:e19d60f96da34b5f1af28c8ed186623f1e99b14f343edb31fbbf6e6f6a8a1e66
0.0s
=> => exporting attestation manifest
sha256:f1fdbb5e454e728755a37bef306243f3be159e6d3f22a520de47ca08479c08c2
0.0s
=> => exporting manifest list
sha256:c8e28038b870fece90ad49e55ab235dd81cffedb86bf2ba98e08598871135845
0.0s
=> => naming to docker.io/library/fastapi-python-mariadb:latest
0.0s
=> => unpacking to docker.io/library/fastapi-python-mariadb:latest
4.5s

1 warning found (use docker --debug to expand):
- SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV
"MARIADB_ROOT_PASSWORD") (line 15)
root@LAPTOP-M3KHGG4S:~/dev_code# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
PORTS         NAMES
bf757d5ec41e   ubuntu:latest  "/bin/bash"             2 hours ago   Exited (0) About an hour
ago            agitated_villani
root@LAPTOP-M3KHGG4S:~/dev_code# docker logs agitated_villani
root@bf757d5ec41e:/# exit
exit
root@LAPTOP-M3KHGG4S:~/dev_code# ls
Dockerfile 'Dockerfile copy' init.sql main.py requirements.txt
```





```
root@LAPTOP-M3KHGG4S:~/dev_code# docker logs agitated_villani
root@bf757d5ec41e:/# exit
exit
```



从你的日志来看，镜像构建成功，但在运行容器时可能未按照预期的步骤完成配置或初始化。以下是你可以检查和处理的问题，以及如何验证容器是否正常运行：

## 1. 检查运行容器的命令

在 `docker ps -a` 中，你的容器 `agitated_villani` 是基于 `ubuntu:latest`，而不是你刚刚构建的 `fastapi-python-mariadb`。请确保运行的是正确的镜像。

尝试以下命令运行你的镜像：

```
docker run -d --name fastapi-container \
-e MARIADB_ROOT_PASSWORD=root_password \
-p 3306:3306 fastapi-python-mariadb
```

## 2. 查看 MariaDB 是否正确初始化

MariaDB 的初始化脚本是否被执行可以通过日志进行检查：

```
docker logs fastapi-container
```

期望日志中出现类似以下内容：

- `CREATE DATABASE` 被成功执行。
- `CREATE USER` 和 `GRANT ALL PRIVILEGES` 完成。

如果未看到这些日志内容，可能是 `init.sql` 没有被正确加载或执行。

## 3. 验证 MariaDB 服务是否正常运行

进入容器并手动验证：

```
docker exec -it fastapi-container bash
```

在容器内，使用以下命令检查 MariaDB 服务状态：

```
service mysql status
```

如果服务未启动，尝试启动 MariaDB：





```
service mysql start
```

测试 MariaDB 是否可以正常连接：

```
mysql -u root -p
```

输入 `root\_password` 进行登录，并查看是否有 `fastapi\_crud` 数据库。

## 4. 检查 `init.sql` 文件是否正确

确保你的 `init.sql` 文件内容如下：

```
CREATE DATABASE IF NOT EXISTS fastapi_crud;  
CREATE USER IF NOT EXISTS 'fastapi_user'@'%' IDENTIFIED BY 'fastapi_password';  
GRANT ALL PRIVILEGES ON fastapi_crud.* TO 'fastapi_user'@'%';  
FLUSH PRIVILEGES;
```

验证 `init.sql` 被正确复制：检查 Docker 镜像中的 `/docker-entrypoint-initdb.d/init.sql` 文件是否存在：

```
docker exec -it fastapi-container ls /docker-entrypoint-initdb.d/
```

## 5. 查看 MariaDB 数据目录权限

MariaDB 可能因为数据目录的权限问题无法正常运行。检查 `/var/lib/mysql` 是否被正确初始化：

```
docker exec -it fastapi-container ls -l /var/lib/mysql
```

如果权限不正确，可以尝试重新初始化数据目录：

```
rm -rf /var/lib/mysql/*  
mariadb-install-db --user=mysql --datadir=/var/lib/mysql  
service mysql start
```

## 6. 重新运行并验证

如果以上步骤未解决问题，尝试以下流程重新运行镜像：

1. 清理旧容器：

```
docker rm -f fastapi-container
```



## 2. 重新运行镜像：

```
docker run -d --name fastapi-container \
-e MARIADB_ROOT_PASSWORD=root_password \
-p 3306:3306 fastapi-python-mariadb
```

## 3. 检查日志：

```
docker logs fastapi-container
```

## 4. 验证 MariaDB 服务和初始化：使用上文中提到的方法检查 MariaDB 是否成功启动。

通过这些步骤，你应该能够排查和解决 MariaDB 容器初始化失败或服务未启动的问题。如果仍有问题，请分享相关的日志和具体错误信息，我会进一步帮你分析！



```
root@LAPTOP-M3KHGG4S:~/dev_code# docker run -p 3306:3306 -p 8000:8000 --
name fastapi-container -d fastapi-python-mariadb
76a4ba05be26fded8fe72a97ccbf54662cd88c8f614ef537b8a7d38cc7f8fba8
docker: Error response from daemon: Ports are not available: exposing port TCP
0.0.0.0:3306 -> 0.0.0.0:0: listen tcp 0.0.0.0:3306: bind: Only one usage of each
socket address (protocol/network address/port) is normally permitted.
root@LAPTOP-M3KHGG4S:~/dev_code# docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS
PORTS         NAMES
76a4ba05be26   fastapi-python-mariadb "sh -c 'mysqld_safe ...'" 6 seconds ago   Created
bf757d5ec41e   ubuntu:latest        "/bin/bash"             2 hours ago    Exited (0)
About an hour ago   agitated_villani
root@LAPTOP-M3KHGG4S:~/dev_code# mysql -h 127.0.0.1 -u root -p
Command 'mysql' not found, but can be installed with:
apt install mysql-client-core-8.0 # version 8.0.39-0ubuntu0.22.04.1, or
apt install mariadb-client-core-10.6 # version 1:10.6.18-0ubuntu0.22.04.1
root@LAPTOP-M3KHGG4S:~/dev_code#
```



## 问题分析

根据你的日志，出现了以下两个问题：

### 1. 端口冲突：

错误信息 `bind: Only one usage of each socket address` 表明主机的 `3306` 端口已被占用，导致 MariaDB 无法绑定到该端口。

### 2. 客户端工具缺失：

本地未安装 `mysql` 或 `mariadb-client`，因此无法通过命令行工具连接到 MariaDB。





## 解决方法

### 1. 解决端口冲突

使用以下步骤检查并释放端口：

1. **检查端口占用情况**：在 WSL 中运行以下命令，确认 `3306` 是否被占用：

```
sudo netstat -tuln | grep 3306
```

如果显示某个进程在使用 `3306`，记录其 PID（最后一列）。

2. **停止占用进程**：假设 PID 是 `12345`，运行以下命令停止该进程：

```
sudo kill -9 12345
```

3. **运行容器时绑定到其他端口**：如果无法释放 `3306`，可以让 MariaDB 绑定到另一个端口，例如 `3307`：

```
docker run -p 3307:3306 -p 8000:8000 --name fastapi-container -d fastapi-python-mariadb
```

4. **连接 MariaDB 时指定新端口**：如果使用非默认端口（如 `3307`），连接时需要指定端口：

```
mysql -h 127.0.0.1 -P 3307 -u root -p
```

---

### 2. 安装客户端工具

为了使用 `mysql` 或 `mariadb-client`，你需要安装相关工具：

1. **安装 `mariadb-client`**：在 WSL 中运行以下命令：

```
sudo apt update
sudo apt install mariadb-client
```

2. **验证安装**：运行以下命令，确认 `mariadb-client` 是否正确安装：

```
mysql --version
```

3. **连接到 MariaDB**：成功安装后，可以通过以下命令连接 MariaDB：

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

输入 `root\_password` 以登录。

---

### 3. 检查容器运行状态



1. **查看容器日志**：确保 MariaDB 服务已成功启动：

```
docker logs fastapi-container
```

2. **检查容器端口映射**：验证容器是否正确绑定端口：

```
docker ps
```

输出示例：

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
76a4ba05be26	fastapi-python-mariadb	"sh -c 'mysqld_safe ...'"	2 minutes ago	Up 2 minutes	0.0.0.0:3307->3306/tcp

3. **进入容器手动检查 MariaDB**：如果容器启动后仍无法连接，可以进入容器检查 MariaDB：

```
docker exec -it fastapi-container bash
```

在容器内，运行以下命令查看 MariaDB 状态：

```
service mysql status
```

如果未启动，可以尝试手动启动：

```
service mysql start
```

---

#### 4. 测试 FastAPI 和数据库连接

1. **确认 FastAPI 服务**：你的 Dockerfile 中似乎包含了 FastAPI 服务，确保它也正确绑定到端口 `8000`：

```
docker ps
```

2. **通过浏览器或 curl 访问 FastAPI 服务**：打开浏览器，访问：

```
http://127.0.0.1:8000
```

或使用 `curl` 测试：

```
curl http://127.0.0.1:8000
```

3. **测试 FastAPI 与 MariaDB 的连接**：确保 FastAPI 配置的数据库连接字符串使用了正确的主机（如 `127.0.0.1`）和端口（如 `3307`）。





通过上述方法，你可以解决端口冲突、客户端工具缺失以及容器运行中遇到的问题。如果问题仍然存在，请提供更多日志信息，我将进一步帮助你分析！



```
root@LAPTOP-M3KHGG4S:~/dev_code# sudo apt install netstat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package netstat
root@LAPTOP-M3KHGG4S:~/dev_code#
```



在现代的 Linux 系统中，`netstat` 已被废弃，推荐使用 `ss` 或 `ip` 等工具进行网络分析。以下是解决 `netstat` 未找到的问题的步骤：

## 1. 安装 `net-tools` 包（包含 `netstat`）

尽管 `netstat` 已被废弃，但仍可以通过安装 `net-tools` 包使用：

```
sudo apt update
sudo apt install net-tools
```

安装完成后，运行以下命令确认 `netstat` 是否可用：

```
netstat --version
```

## 2. 使用现代替代工具

如果你不想安装 `net-tools`，可以直接使用 `ss` 命令，它是 `netstat` 的现代替代工具：

### 检查端口占用

查看所有监听端口：

```
sudo ss -tuln
```

输出示例：

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
tcp	LISTEN	0	128	0.0.0.0:3306	0.0.0.0:*	

### 筛选特定端口

检查特定端口（如 `3306`）的占用情况：

```
sudo ss -tuln | grep 3306
```





### 3. 使用 `lsof` 检查端口

`lsof` 是另一个强大的工具，可以列出所有占用某个端口的进程。

安装 `lsof`：

```
sudo apt update
sudo apt install lsof
```

使用 `lsof` 检查端口 `3306` 的占用情况：

```
sudo lsof -i :3306
```

输出示例：

```
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
mysqld   1234 mysql  10u   IPv4  12345      0t0  TCP *:3306 (LISTEN)
```

### 4. 停止占用进程

一旦找到占用端口的进程（例如上面的 PID 是 `1234`），可以停止该进程：

```
sudo kill -9 1234
```

通过上述步骤，你可以选择安装 `net-tools` 或使用现代工具如 `ss` 和 `lsof` 来分析网络端口占用情况。如果还有其他问题，可以继续告诉我！



```
fastapi-python-mariadb改名root@LAPTOP-M3KHGG4S:~/dev_code# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
fastapi-python-mariadb latest    077452890c33   28 minutes ago 1.06GB
ubuntu               latest    80dd3c3b9c6c   6 weeks ago   117MB
```



如果你想将 `fastapi-python-mariadb` 镜像改名为其他名称，可以使用以下步骤：

### 1. 使用 `docker tag` 命令重命名镜像

Docker 并不直接支持重命名镜像，但你可以通过打一个新标签来实现相同效果。

命令格式：



```
docker tag <旧镜像名>:<旧标签> <新镜像名>:<新标签>
```

示例：

假设你想将 `fastapi-python-mariadb:latest` 改名为 `myapp-db:latest`，运行以下命令：

```
docker tag fastapi-python-mariadb:latest myapp-db:latest
```

运行后，使用以下命令查看结果：

```
docker images
```

输出示例：

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
fastapi-python-mariadb	latest	077452890c33	28 minutes ago	1.06GB
myapp-db	latest	077452890c33	28 minutes ago	1.06GB

## 2. 删除旧镜像（可选）

如果你不需要旧镜像名称，可以删除旧的标签：

```
docker rmi fastapi-python-mariadb:latest
```

确认删除后，使用 `docker images` 查看是否只剩下新的镜像。

## 3. 使用新名称运行容器

运行新镜像：

```
docker run -p 3306:3306 -p 8000:8000 --name myapp-container -d myapp-db:latest
```

通过以上步骤，你可以成功将镜像改名并使用新名称运行容器。如果有进一步问题，请告诉我！



```
刪掉運行的mysqld_saferoot@fc4f811c31bc:/app# mysqld_safe
250105 12:34:16 mysqld_safe Logging to syslog.
250105 12:34:16 mysqld_safe Starting mariadb daemon with databases from
/var/lib/mysql
q
^Z
[1]+  Stopped                  mysqld_safe
```





```
root@fc4f811c31bc:/app# mysqld_safe &
[2] 144
root@fc4f811c31bc:/app# 250105 12:35:36 mysqld_safe Logging to syslog.
250105 12:35:36 mysqld_safe A mysqld process already exists
^C
[2]- Exit 1          mysqld_safe
root@fc4f811c31bc:/app#
```



根据日志，你的 `mysqld_safe` 已经在运行，因此当你尝试再次启动时会提示 `A mysqld process already exists`。以下是解决该问题的步骤：

## 步骤 1：查看现有的 MariaDB 进程

在容器中运行以下命令，列出正在运行的 MariaDB 进程：

```
ps aux | grep mysqld
```

输出示例：

```
mysql      123    0.0  0.1 123456 1234 ?    Ssl  12:34   0:00 /usr/sbin/mysqld
root       456    0.0  0.0 123456  123 ?    Ss+  12:35   0:00 mysqld_safe
```

找到与 `mysqld_safe` 或 `mysqld` 相关的进程，记录其 **PID**。

## 步骤 2：停止现有 MariaDB 进程

使用以下命令终止运行的 `mysqld_safe` 或 `mysqld` 进程：

### 终止特定进程

假设 `mysqld_safe` 的 PID 是 `456`，运行：

```
kill -9 456
```

### 终止所有相关进程

如果有多个相关进程，也可以批量终止：

```
killall mysqld_safe mysqld
```

## 步骤 3：确认进程已停止

再次运行以下命令，确认没有 `mysqld` 或 `mysqld_safe` 的进程：







```
ps aux | grep mysqld
```

如果没有输出与 `mysqld` 或 `mysqld\_safe` 相关的进程，则表示已成功停止。

## 步骤 4：重新启动 MariaDB

确保进程被清理后，可以重新启动 MariaDB。

使用 `mysqld\_safe` 启动：

```
mysqld_safe &
```

使用服务启动：

如果你在使用标准的 MariaDB 服务：

```
service mysql start
```

## 步骤 5：查看日志

检查是否成功启动：

```
tail -f /var/log/mysql/error.log
```

如果出现任何错误，可以从日志中进一步排查。

## 提示

- 如果 MariaDB 数据目录损坏，可能需要重新初始化或修复数据目录：

```
mysql_install_db --user=mysql --datadir=/var/lib/mysql
```

通过以上步骤，你应该可以清理冲突的 MariaDB 进程并正常重新启动。如果问题仍然存在，请提供更多错误信息，我会进一步帮助你！



執行mysqld\_safe & 就不能執行其他指令



在后台执行 `mysqld\_safe &` 时，终端可能会被挂起或阻塞，影响你执行其他命令。这通常是因为 `mysqld\_safe` 的输出占用了当前终端。

以下是解决这个问题的方法：



## 1. 使用 `nohup` 执行

`nohup` 命令可以让进程在后台运行，并将输出重定向到文件。

```
nohup mysqld_safe > mysqld_safe.log 2>&1 &
```

- **解释：**
  - `nohup`：防止进程因终端关闭而被终止。
  - `> mysqld\_safe.log`：将标准输出重定向到日志文件。
  - `2>&1`：将错误输出重定向到同一个日志文件。

运行后，你可以继续在当前终端执行其他命令。日志文件可以通过以下命令查看：

```
tail -f mysqld_safe.log
```

## 2. 使用 `& disown`

在 `&` 后加 `disown`，将进程与当前终端分离。

```
mysqld_safe & disown
```

- **解释：**
  - `&`：将进程放到后台。
  - `disown`：从当前会话中移除进程，使其不再与当前终端关联。

此时，你可以自由执行其他命令。

## 3. 使用 `screen` 或 `tmux`

`screen` 和 `tmux` 是多任务终端工具，允许你在一个终端中运行多个会话。

**安装：**

如果未安装，可以运行：

```
sudo apt update
sudo apt install screen
```

**使用 `screen`：**

1. 创建一个新的会话：

```
screen -S mysqld_safe
```

2. 在会话中运行：



```
mysqld_safe
```

3. 分离会话（退出但不停止进程）：按下 `Ctrl+A`，然后按 `D`。
4. 恢复会话：

```
screen -r mysqld_safe
```

---

## 4. 使用 `systemd` 服务管理

如果容器支持 `systemd`，可以通过创建一个服务文件来管理 MariaDB。

创建服务文件：

1. 编辑服务文件：

```
nano /etc/systemd/system/mysqld_safe.service
```

2. 添加以下内容：

```
[Unit]
Description=MariaDB using mysqld_safe
After=network.target

[Service]
ExecStart=/usr/bin/mysqld_safe
Restart=always
User=mysql

[Install]
WantedBy=multi-user.target
```

3. 启动服务：

```
systemctl start mysqld_safe
```

4. 查看服务状态：

```
systemctl status mysqld_safe
```

---

## 5. 使用后台进程管理工具

如需要更精细的控制，可以使用 `jobs` 和 `bg` 命令：

1. 启动进程：

```
mysqld_safe &
```



2. 暂停进程：按下 `Ctrl+Z`。
3. 让进程在后台继续运行：

```
bg
```

4. 查看后台任务：

```
jobs
```

通过这些方法，你可以在运行 `mysqld_safe` 的同时继续执行其他命令。如果还有其他需求，请告诉我！



```
root@LAPTOP-M3KHGG4S:~/dev_code# docker run -it --entrypoint bash -p 3307:3306
-p 8000:8000 --name fastapi-container fastapi-python-mariadb bash
/usr/bin/bash: /usr/bin/bash: cannot execute binary file
root@LAPTOP-M3KHGG4S:~/dev_code# docker ps -a
CONTAINER ID   IMAGE                COMMAND             CREATED        STATUS
PORTS         NAMES
64b35269c23d   fastapi-python-mariadb "bash bash"        17 seconds ago Exited (126)
16 seconds ago fastapi-container
bf757d5ec41e   ubuntu:latest       "/bin/bash"        4 hours ago    Exited (0) 3 hours
ago            agitated_villani
root@LAPTOP-M3KHGG4S:~/dev_code# docker logs fastapi-container
/usr/bin/bash: /usr/bin/bash: cannot execute binary file
root@LAPTOP-M3KHGG4S:~/dev_code#
```



错误信息 `/usr/bin/bash: cannot execute binary file` 表明容器内的 `/usr/bin/bash` 并非有效的可执行文件，或者你的 Docker 镜像中没有正确安装 Bash。

以下是解决问题的步骤：

## 1. 检查镜像中是否存在 Bash

首先，验证 `fastapi-python-mariadb` 镜像是否包含有效的 Bash：

```
docker run -it --entrypoint /bin/sh fastapi-python-mariadb
```

- 如果可以进入容器，说明镜像中可能没有 Bash，而是使用了 `/bin/sh`。
- 如果也报错，则说明镜像中的 Shell 有问题。

解决方法：

如果镜像中没有安装 Bash，可以通过以下方式修复：





1. 进入容器并安装 Bash：

```
apt update && apt install -y bash
```

2. 确保 Bash 可执行：

```
which bash
```

---

## 2. 修复 Docker 镜像的 EntryPoint

当前的 Docker 镜像可能定义了不正确的 `ENTRYPOINT`，导致无法正常运行。你可以使用以下方式解决：

### 检查 Dockerfile 的 ENTRYPOINT

打开 `Dockerfile`，确认是否有类似以下内容：

```
ENTRYPOINT ["bash"]
```

如果没有，添加以下内容：

```
ENTRYPOINT ["/bin/bash"]
```

重新构建镜像：

```
docker build -t fastapi-python-mariadb .
```

---

## 3. 使用 `--entrypoint` 强制覆盖

当镜像中的 `ENTRYPOINT` 定义有误时，可以通过 `--entrypoint` 参数临时覆盖：

```
docker run -it --entrypoint /bin/sh -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb
```

如果确认镜像中 Bash 存在并可用，也可以指定 `/bin/bash`：

```
docker run -it --entrypoint /bin/bash -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb
```

---

## 4. 检查镜像的基础镜像

确保 `fastapi-python-mariadb` 使用的基础镜像是支持 Bash 的。例如，如果使用 `python:3.9-slim`，该镜像默认没有 Bash。

解决方法：



在 `Dockerfile` 中修改基础镜像，或者安装 Bash：

1. 修改基础镜像：

```
FROM python:3.9
```

这样会包含更完整的系统工具，包括 Bash。

2. 或者安装 Bash：在 `Dockerfile` 中添加以下命令：

```
RUN apt update && apt install -y bash
```

重新构建镜像：

```
docker build -t fastapi-python-mariadb .
```

## 5. 删除失败的容器并重试

删除之前的失败容器：

```
docker rm fastapi-container
```

然后重新运行修复后的镜像：

```
docker run -it -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb
```

## 总结

- 错误原因通常是镜像中缺少 Bash 或 `ENTRYPOINT` 配置错误。
- 通过修复 `Dockerfile` 或强制指定 `--entrypoint`，可以解决问题。
- 如果仍有问题，请分享你的 `Dockerfile` 或完整的错误日志，我将进一步帮你分析！



`docker run -d -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb -c "tail -f /dev/null"`，移除 `-c "tail -f /dev/null"` 有什麼差別



在 Docker 命令中添加 `-c "tail -f /dev/null"` 的行为和不添加的行为存在明显差异。

以下是具体区别和作用的详细分析：

### 1. 使用 `-c "tail -f /dev/null"` 的作用



在 Docker 命令中加上 `-c "tail -f /dev/null"`，通常是会让容器保持运行状态。

- **具体含义：**
  - `tail -f /dev/null` 是一个阻塞命令，会持续运行且不会退出。
  - 通过这个命令，容器会一直保持运行状态，而不会因为缺少前台进程而自动退出。
- **适用场景：**
  - **调试和开发：** 例如，你希望容器启动后可以进入容器调试或运行额外命令。

```
docker exec -it fastapi-container bash
```

- **测试环境：** 如果主要的应用进程（如 FastAPI 或 MariaDB）未启动时，仍需要容器保持运行以便进一步操作。

## 2. 不使用 `-c "tail -f /dev/null"` 的行为

如果不加 `-c "tail -f /dev/null"`，容器的运行状态取决于镜像的 `CMD` 或 `ENTRYPOINT` 指令。

- **情况 1：镜像中定义了 `CMD` 或 `ENTRYPOINT`**
  - 容器启动后会运行镜像中定义的默认命令。例如，在 `fastapi-python-mariadb` 镜像中，可能会运行：

```
mysqld_safe
```

- 如果这些命令是长期运行的（如数据库服务、Web 服务），容器会保持运行。
- **情况 2：镜像中没有定义 `CMD` 或 `ENTRYPOINT`**
  - 如果镜像没有指定默认命令，容器会立即退出，状态变为 `Exited`。

## 3. 哪种方式更适合？

- **需要容器保持运行，但没有特定服务：**
  - 使用 `-c "tail -f /dev/null"` 保持容器运行，方便后续操作（调试或补充运行服务）。

```
docker run -d -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb -c "tail -f /dev/null"
```

- **容器主要用于运行服务：**
  - 不需要 `-c "tail -f /dev/null"`，让容器按预期启动服务（如 MariaDB 或 FastAPI）。

示例：

```
docker run -d -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb
```



## 4. 如何验证行为？

### 1. 查看容器状态：

```
docker ps -a
```

### 2. 查看容器日志：

```
docker logs fastapi-container
```

### 3. 进入容器调试：如果添加了 ``-c "tail -f /dev/null"``，可以进入容器并手动启动服务：

```
docker exec -it fastapi-container bash
```

## 总结

- 添加 ``-c "tail -f /dev/null"``：容器会保持运行，无需依赖镜像内的服务或命令，适用于调试和开发环境。
- 不添加 ``-c "tail -f /dev/null"``：容器按镜像的默认行为运行，适用于生产环境，启动服务后直接使用。

选择方式取决于你对容器运行行为的需求。如果容器只是服务的宿主，不需要添加 ``-c "tail -f /dev/null"``。



```
# 使用 Python 3.9 slim 作為基礎映像
FROM python:3.9-slim

# 更新系統並安裝必要依賴和 MariaDB
RUN apt-get update && apt-get install -y \
    mariadb-server \
    mariadb-client \
    libmariadb-dev \
    gcc \
    libffi-dev \
    libssl-dev \
    && apt-get clean

# 設置 MariaDB root 密碼和數據庫名稱
ENV MARIADB_ROOT_PASSWORD=yourpassword
ENV MARIADB_DATABASE=fastapi_crud

# 初始化 MariaDB 數據目錄並設置 root 密碼
RUN mariadb-install-db --user=mysql --datadir=/var/lib/mysql && \
    mysqld_safe & \
    sleep 30 && \
```





```
mysqladmin --user=root password "${MARIADB_ROOT_PASSWORD}" && \
mysql -u root -p"${MARIADB_ROOT_PASSWORD}" -e "FLUSH PRIVILEGES" && \
pkill -f mysqld_safe

# 複製初始化 SQL 文件到容器
COPY init.sql /docker-entrypoint-initdb.d/

# 安裝 FastAPI 和 Python 依賴
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 複製應用程式碼
WORKDIR /app
COPY . /app/

# 暴露服務端口
EXPOSE 3306 8000

# # 啟動 MariaDB 和 FastAPI
# CMD ["sh", "-c", "mysqld_safe & sleep 30 && uvicorn main:app --host 0.0.0.0 --port 8000"]

# # 啟動 Bash，允許手動執行 MariaDB 和 FastAPI
# CMD ["/bin/bash"]

# 啟動 Bash，允許手動執行 MariaDB 和 FastAPI
CMD ["/bin/bash", "tail -f /dev/null"]
, root@LAPTOP-M3KHGG4S:~/dev_code# docker run -d -p 3307:3306 -p 8000:8000 --
name fastapi-container fastapi-python-mariadb
333af840249953be9852d306b1aa3ca346a8768fc4535df6e8ada54ac60319d6
root@LAPTOP-M3KHGG4S:~/dev_code# docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS
PORTS         NAMES
333af8402499   fastapi-python-mariadb "/bin/bash 'tail -f ..." 5 seconds ago   Exited
(127) 4 seconds ago          fastapi-container
bf757d5ec41e   ubuntu:latest       "/bin/bash"             6 hours ago    Exited (0) 5
hours ago          agitated_villani
root@LAPTOP-M3KHGG4S:~/dev_code# docker logs fastapi-container
/bin/bash: tail -f /dev/null: No such file or directory
root@LAPTOP-M3KHGG4S:~/dev_code#
```



从日志可以看出，容器退出的原因是命令 `/bin/bash 'tail -f /dev/null'` 格式错误。单引号导致了命令被解析为一个整体，而不是两个独立的命令。以下是问题的详细分析和解决方案。

## 问题分析

在 Dockerfile 的 `CMD` 指令中使用了：

```
CMD ["/bin/bash", "tail -f /dev/null"]
```

这会导致 Docker 尝试运行：





```
/bin/bash 'tail -f /dev/null'
```

问题在于：

1. 单引号 `'tail -f /dev/null'` 导致整个命令被视为一个参数。
2. `/bin/bash` 无法找到路径为 `'tail -f /dev/null'` 的文件，因此报错 `No such file or directory`。

## 解决方法

### 1. 修复 CMD 指令

将 `CMD` 修改为正确的格式，确保命令和参数被正确分开：

```
CMD ["/bin/bash", "-c", "tail -f /dev/null"]
```

- **解释：**
  - `/bin/bash`：运行 Bash。
  - `-c`：告诉 Bash 执行后续的命令。
  - `tail -f /dev/null`：阻止容器退出。

### 完整修订后的 Dockerfile

```
# 使用 Python 3.9 slim 作為基礎映像
FROM python:3.9-slim

# 更新系統並安裝必要依賴和 MariaDB
RUN apt-get update && apt-get install -y \
    mariadb-server \
    mariadb-client \
    libmariadb-dev \
    gcc \
    libffi-dev \
    libssl-dev \
    && apt-get clean

# 設置 MariaDB root 密碼和數據庫名稱
ENV MARIADB_ROOT_PASSWORD=yourpassword
ENV MARIADB_DATABASE=fastapi_crud

# 初始化 MariaDB 數據目錄並設置 root 密碼
RUN mariadb-install-db --user=mysql --datadir=/var/lib/mysql && \
    mysqld_safe & \
    sleep 30 && \
    mysqladmin --user=root password "${MARIADB_ROOT_PASSWORD}" && \
    mysql -u root -p"${MARIADB_ROOT_PASSWORD}" -e "FLUSH PRIVILEGES" && \
    pkill -f mysqld_safe

# 複製初始化 SQL 文件到容器
COPY init.sql /docker-entrypoint-initdb.d/

# 安裝 FastAPI 和 Python 依賴
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
```



```
# 複製應用程式碼
WORKDIR /app
COPY . /app/

# 暴露服務端口
EXPOSE 3306 8000

# 啟動 Bash，允許手動執行 MariaDB 和 FastAPI
CMD ["/bin/bash", "-c", "tail -f /dev/null"]
```

## 2. 重建镜像

修改完 Dockerfile 后，重新构建镜像：

```
docker build -t fastapi-python-mariadb .
```

## 3. 运行容器

运行容器时不需要额外参数：

```
docker run -d -p 3307:3306 -p 8000:8000 --name fastapi-container fastapi-python-mariadb
```

检查容器状态：

```
docker ps
```

进入容器查看是否正常运行：

```
docker exec -it fastapi-container bash
```

## 验证

- 如果容器正常运行，则 `docker ps` 应显示容器的状态为 `Up`。
- 日志中不应再出现 `No such file or directory` 的错误。

通过这些修复，容器应该可以正常保持运行，并支持 MariaDB 和 FastAPI 的后续配置。如果还有问题，请提供新的错误信息，我会进一步帮助你！