Setup Guide: Docker Desktop with WSL2, Ubuntu, Python 3.11, and VS Code

This document provides a step-by-step guide to setting up Docker Desktop on Windows with the WSL2 backend, installing Ubuntu as the Linux environment, configuring Python 3.11 as the default interpreter, and integrating the workflow with Visual Studio Code. It also covers error handling, edge cases, and best practices.

# 1. Prerequisites

- Windows 10 (2004+) or Windows 11  
- Administrator access  
- Stable internet connection  
- Optional: NVIDIA GPU (if you want GPU-accelerated Docker containers)

# 2. Enable WSL2

1. Open PowerShell (Admin) and run:  
 dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart  
 dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart  
  
2. Restart your system.  
  
3. Update WSL and set it to version 2 by default:  
 wsl --update  
 wsl --set-default-version 2  
  
Verification:  
 wsl --version  
 → should show WSL version 2.x.x and a kernel version 5.x.x or later.

Common Errors:

- If you see 'The requested operation requires elevation', ensure you ran PowerShell as Administrator.  
- If WSL auto-launches a CMD window after reboot, press any key to finish the update or run `wsl --update` manually.

# 3. Install a Linux Distribution (Ubuntu)

Preferred Method: Microsoft Store → search 'Ubuntu 22.04 LTS' → Install.  
  
Alternatives if Store is broken:  
- PowerShell (Admin): wsl --install -d Ubuntu-22.04  
- Manual appx download: https://learn.microsoft.com/en-us/windows/wsl/install-manual → download Ubuntu 22.04 .appx package → install with Add-AppxPackage.  
  
First Launch:  
- Open Ubuntu from Start Menu.  
- Create a username and password (Linux credentials, separate from Windows).  
- Update packages:  
 sudo apt update && sudo apt upgrade -y  
  
Tip: First update may take 5–15 minutes depending on internet speed.

Edge Cases:

- If updates seem stuck, press Ctrl+C and rerun `sudo apt update` then `sudo apt upgrade -y` separately.  
- Keep your projects inside Linux home (~/projects) for better performance, not under /mnt/c.

# 4. Install Docker Desktop with WSL2 Backend

1. Download from: https://www.docker.com/products/docker-desktop/  
2. Run installer → select 'Use WSL 2 instead of Hyper-V'.  
3. Allow it to restart if prompted.  
4. Open Docker Desktop → Settings → General → ensure 'Use WSL 2 based engine' is enabled.  
5. Go to Settings → Resources → WSL Integration → enable integration with Ubuntu-22.04.

Verification:

- Inside Ubuntu, run:  
 docker --version  
 docker run hello-world  
  
- If you see 'permission denied while connecting to /var/run/docker.sock':  
 sudo usermod -aG docker $USER  
 newgrp docker  
 docker run hello-world  
  
Temporary workaround: prepend `sudo` before docker commands.

# 5. Optional: GPU Integration

If you have an NVIDIA GPU (e.g., RTX 3060):  
1. Install the latest NVIDIA drivers (with CUDA WSL support).  
2. Restart WSL: wsl --shutdown  
3. Test with:  
 docker run --rm --gpus all nvidia/cuda:12.3.2-base-ubuntu22.04 nvidia-smi  
  
→ Should display your GPU details.

# 6. VS Code Setup

1. Install VS Code: https://code.visualstudio.com/  
2. Open Extensions sidebar (Ctrl+Shift+X).  
3. Install extensions:  
 - Remote – WSL (Microsoft)  
 - Docker (Microsoft)  
 - Python (Microsoft)  
  
4. Connect to WSL:  
 - Click bottom-left green >< button → 'Connect to WSL: Ubuntu-22.04'.  
 - New window opens attached to WSL.  
 - Open your project folder inside Linux (/home/<username>/projects/ProjectName).  
 - Do NOT open /mnt/c/... for active projects (slower performance).  
  
5. Open Terminal in VS Code (Ctrl+`) → should show Linux prompt.

Common Edge Case:

- If you accidentally open a Windows folder after attaching to WSL, press Ctrl+Shift+P → 'Remote-WSL: Reopen Folder in WSL'.  
- Ensure settings.json has Python interpreter pointing to WSL’s Python 3.11.

# 7. Verify Full Workflow

Inside Ubuntu WSL (or VS Code WSL Terminal):  
1. Test Docker:  
 docker run hello-world  
  
2. Spin up Postgres:  
 docker run --name pgtest -e POSTGRES\_PASSWORD=mysecret -d -p 5432:5432 postgres:15  
  
3. Verify container:  
 docker ps  
  
4. Connect to DB:  
 docker exec -it pgtest psql -U postgres  
  
5. (Optional) docker-compose.yml for Postgres with persistence:  
  
version: '3.9'  
services:  
 db:  
 image: postgres:15  
 container\_name: pgdb  
 environment:  
 POSTGRES\_USER: postgres  
 POSTGRES\_PASSWORD: mysecret  
 POSTGRES\_DB: myapp  
 ports:  
 - "5432:5432"  
 volumes:  
 - pgdata:/var/lib/postgresql/data  
volumes:  
 pgdata: