Troubleshooting Journey: A Guide to Deploying a Dockerized AI Project on Windows with WSL 2

Date: June 20, 2025
Goal: To successfully run a complex, GPU-accelerated AI application (SAM2 + Grounding DINO) distributed as a Docker project on a Windows machine.

Introduction

This document outlines the series of challenges encountered and the solutions applied to successfully deploy a Linux-based Docker application on Windows 10/11. This journey highlights common pitfalls related to system setup, permissions, resource allocation, and networking in a modern development environment using the Windows Subsystem for Linux (WSL 2).

Issue #1: Running Linux Scripts on Windows

- **Problem:** Attempting to run the ./deploy command directly in the Windows Command Prompt (cmd.exe) resulted in the error: '.' is not recognized as an internal or external command.
- **Diagnosis:** The command and the script (./deploy) were designed for a Linux/Unix environment (like Bash shell), not for the Windows Command Prompt.
- **Solution:** The entire project must be run from within a Linux environment installed on Windows. This led to the requirement for the Windows Subsystem for Linux (WSL).

Issue #2: Setting Up the Linux Environment (WSL)

- **Problem:** A suitable Linux environment was not installed. The initial installation attempt was stuck or corrupted.
- **Diagnosis:** The Ubuntu WSL instance was not starting correctly, presenting a black, unresponsive terminal screen. A subsequent installation attempt was stuck at 0% download, indicating a network or system state issue.
- **Solution:** A full, clean re-installation was performed:
  1. We forced a shutdown of any stuck WSL processes using PowerShell: wsl --shutdown.
  2. We completely removed the corrupted installation: wsl --unregister Ubuntu.
  3. We performed a full system reboot to clear any transient system or network errors.
  4. We reinstalled Ubuntu cleanly using PowerShell: wsl --install -d Ubuntu.

Issue #3: User Permissions in the New Linux Environment

- **Problem:** After a fresh install, the user could not remember the password for their new Ubuntu user account, preventing the use of sudo for administrative tasks.
- **Diagnosis:** Forgotten user credentials.
- **Solution:** We used a WSL-specific feature to reset the password:
    1. From Windows PowerShell, logged into Ubuntu as the super-user: wsl --user root.
    2. From the root shell, changed the password for the nikola user: passwd nikola.

Issue #4: Docker Integration with WSL

- **Problem:** Commands inside the deploy script were failing with the error: The command 'docker' could not be found in this WSL 2 distro.
- **Diagnosis:** Docker Desktop was running on Windows, but the necessary connection to the new Ubuntu environment (WSL 2) was not enabled.
- **Solution:** We enabled the WSL Integration within the Docker Desktop application:
    1. Opened Docker Desktop Settings > Resources > WSL Integration.
    2. Toggled the switch for "Ubuntu" **OFF**, clicked "Apply & Restart".
    3. Toggled the switch for "Ubuntu" back **ON**, clicked "Apply & Restart". This "off-and-on" cycle forced a clean refresh of the connection.

Issue #5: Docker Permissions

- **Problem:** After fixing the integration, Docker commands failed with: permission denied while trying to connect to the Docker daemon socket.
- **Diagnosis:** This is a standard Linux security feature. By default, a regular user does not have permission to control the Docker service.
- **Solution:** We added the user to the docker group:
    1. Ran the command: sudo usermod -aG docker $USER.
    2. **Crucially, we closed the Ubuntu terminal and opened a new one** for the group membership change to take effect.

Issue #6: System Resource Exhaustion during Docker Build

- **Problem:** The docker build process ran almost to completion but crashed at the final "exporting to image" stage with a SIGBUS: bus error and a subsequent Input/output error.
- **Diagnosis:** This low-level system error indicated the computer was running out of resources—either Memory (RAM) or Disk Space. A check with df -h confirmed the **Windows C: drive was 94% full**.
- **Solution:**
    1. **Disk Space:** The primary solution was to **free up a significant amount of space** on the Windows C: drive by deleting old files, uninstalling programs, and using the Disk Cleanup utility.

2. **Memory:** As a best practice, we also created a .wslconfig file in the Windows user profile folder (C:\Users\Nikola) to explicitly grant WSL more memory, preventing future RAM issues. The file contained:
   ```
   [wsl2]
   memory=12GB
   swap=16GB
   ```

3. A full restart (wsl --shutdown and restarting Docker Desktop) was performed to apply all changes.

Issue #7: Docker Container Configuration and Networking

- **Problem #1:** After the image was built, the container would start, but the application inside would crash with RuntimeError: Found no NVIDIA driver on your system.
  - **Diagnosis:** The docker run command in the deploy script was missing the flag to grant the container access to the host's GPU.
  - **Solution:** We edited the deploy script to add the --gpus all flag to the docker run command.
- **Problem #2:** The service was running inside the container, confirmed by the logs (Uvicorn running on http://0.0.0.0:8090), but was inaccessible from the web browser (ERR_CONNECTION_REFUSED).
  - **Diagnosis:** The container's network port was not exposed to the Windows host machine.
  - **Solution:** We edited the deploy script again to add the port mapping flag -p 8090:8090 to the docker run command. This connected port 8090 on the host to port 8090 in the container.

Issue #8: Sharing the Service with Others

- **Problem:** The user wanted to share the http://localhost:8090 URL with a friend.
- **Diagnosis:** localhost only works on the machine where the server is running. It is not accessible over the internet.
- **Solution:** Instead of a complex and insecure port forwarding setup, the recommended solution was to use a tunneling service.
  1. We used **ngrok** to create a secure, temporary public URL.
  2. The command .\ngrok.exe http 8090 was run on Windows to create a public link that forwards to localhost:8090, which could then be safely shared.

Conclusion

Through a methodical, step-by-step process, every system-level, permission-based, resource-related, and networking issue was resolved. The successful deployment required configuring Windows features (WSL 2), installing and managing a Linux distribution, managing user permissions, integrating with Docker Desktop, allocating system resources, and correctly configuring the container runtime options. This journey serves as a comprehensive guide to the real-world challenges of modern cross-platform development.