How to Set Up Ollama + Open WebUI with Docker (GPU-Enabled, Windows)

1 Overview

This guide walks through installing Ollama and Open WebUI using Docker on Windows, with GPU acceleration and models stored on a fast NVMe SSD. It is intended for first-time users or users rebuilding a system.

2 Prerequisites

- Windows 11
- NVIDIA GPU with recent drivers
- Docker Desktop (WSL2 enabled)
- PowerShell
- Fast NVMe SSD

3 Verify GPU and Docker GPU Access

Verify GPU works:

```
nvidia-smi
```

Verify Docker can see the GPU:

```
docker run --rm --gpus all nvidia/cuda:12.3.0-base-ubuntu22.04 nvidia-smi
```

4 Recommended Folder Layout

```
C:\ollama\
 models\        # Ollama model files
 openwebui\     # Open WebUI data
```

Create folders:

```
mkdir C:\ollama\models
mkdir C:\ollama\openwebui
```

5 Create Docker Network

The `ai-net` network is a dedicated, user-defined Docker bridge network that allows AI-related containers (like Ollama, Open WebUI, and n8n) to communicate with each other reliably, securely, and predictably.

```
docker network create ai-net
```

6 Building a Docker Compose File to Your Needs

6.1 What Is docker-compose.yml?

`docker-compose.yml` is a configuration file that tells Docker:

- Which containers to run
- How they connect to each other
- Which folders on your computer are shared with containers
- Which ports are exposed
- Which GPUs are available

Instead of running long `docker run` commands, Docker Compose lets you start an entire AI stack with one command.

```
docker compose up -d
```

6.2 Where This File Must Live (Important)

You must create `docker-compose.yml` inside your Ollama project folder, for example:

```
C:\ollama\
 docker-compose.yml   # create it here
 models\
 openwebui\
```

Why this location matters: relative paths resolve correctly. Even if you later switch to relative volumes (e.g. `./models`), Docker resolves paths relative to the compose file location. Everything stays self-contained. Your AI stack becomes portable: copy the `C:\ollama` folder, run `docker compose up`, everything works. Avoids accidental path mistakes. Keeping the compose file next to the folders it mounts makes it obvious whats being shared.

6.3 How Docker Compose Uses This File

When you run:

```
docker compose up -d
```

Docker:

- Reads `docker-compose.yml`
- Pulls required images
- Creates containers
- Mounts folders from your PC into containers
- Connects containers to the `ai-net` network
- Starts services in the correct order

No manual wiring required.

7 Breaking Down the docker-compose.yml

Services: defines the containers in your stack. Each service becomes one running container.

```
services:
  ollama:
  openwebui:
```

7.1 Ollama Service (LLM Backend)

It pulls the official Ollama image and names the container `ollama` (used for networking).

```
ollama:
  image: ollama/ollama:latest
  container_name: ollama
```

Ports. Exposes Ollamas API to your host. Allows Open WebUI and local tools to connect.

```
ports:
  - "11434:11434"
```

Volumes (Critical). This maps your local models folder into the container. Makes models persistent across restarts. Allows you to move models without re-downloading. This is why the folder structure matters.

```
volumes:
  - "C:/ollama/models:/root/.ollama/models"
```

GPU Access. Gives Ollama access to all available NVIDIA GPUs. Required for GPU-accelerated inference.

```
gpus: all
```

Environment Variables. Enables network access. Prevent silent CPU fallback. Increase patience for large model loads. Control memory and concurrency. There is where you can really customize how you control the models.

```
environment:
  - OLLAMA_HOST=0.0.0.0
  - OLLAMA_NO_CPU_FALLBACK=1
  - OLLAMA_LOAD_TIMEOUT=1800s
  - OLLAMA_MAX_LOADED_MODELS=2
  - OLLAMA_NUM_PARALLEL=1
```

7.2 Open WebUI Service (Frontend)

Pulls the Open WebUI interface and provides a browser-based chat UI.

```
openwebui:
  image: ghcr.io/open-webui/open-webui:latest
```

Dependency. Ensures Ollama starts first which prevent race conditions at startup.

```
depends_on:
  - ollama
```

Ports. Makes WebUI accessible at localhost:3000 or whatever port you want. Other ways to access the UI is with the system IP:port.

```
ports:
  - "3000:8080"
```

`http://localhost:3000`

WebUI Storage. Stores user accounts, saves settings, chats, model metadata. Persists data across restarts.

```
volumes:
  - "C:/ollama/openwebui:/app/backend/data"
```

Ollama Connection. Uses container DNS (`ollama`), works because both services share `ai-net`.

```
environment:
  - OLLAMA_BASE_URL=http://ollama:11434
```

Networks. Places both services on the same isolated network and enables name-based service discovery. This is mainly for other tools like n8n to communicate with Ollama.

```
networks:
  ai-net:
    external: true
```

8 docker-compose.yml (Full File)

```
services:
  ollama:
    image: ollama/ollama:latest
    container_name: ollama
    restart: unless-stopped
    ports:
      - "11434:11434"
    volumes:
      - "C:/ollama/models:/root/.ollama/models"
    gpus: all
    environment:
      - OLLAMA_HOST=0.0.0.0
      - OLLAMA_NO_CPU_FALLBACK=1
      - OLLAMA_LOAD_TIMEOUT=1800s
      - OLLAMA_MAX_LOADED_MODELS=2
      - OLLAMA_NUM_PARALLEL=1
    networks:
      - ai-net

  openwebui:
    image: ghcr.io/open-webui/open-webui:latest
    container_name: openwebui
    restart: unless-stopped
    depends_on:
      - ollama
    ports:
      - "3000:8080"
    volumes:
      - "C:/ollama/openwebui:/app/backend/data"
    environment:
      - OLLAMA_BASE_URL=http://ollama:11434
      - WEBUI_NAME=Ollama Web UI
    networks:
      - ai-net

networks:
  ai-net:
```

```
    external: true
```

9 Start the Stack

```
docker compose up -d
```

10 Verify Services

```
docker ps
```

Access services:

- Ollama: `http://localhost:11434` (this will just show Ollama as running, which is normal)
- Open WebUI: `http://localhost:3000` (this is the UI to experiment with different models)

11 Adding Models (First Time)

11.1 Option 1: Using Ollama CLI (recommended)

```
docker exec -it ollama ollama pull gemma3:27b
docker exec -it ollama ollama list
```

The `list` command allows you to see the list of currently pulled models.

11.2 Option 2: Using Open WebUI with Hugging Face

Open `http://localhost:3000` → Settings → Models → Add Model → Hugging Face.

Paste a model ID such as:

```
TheBloke/Mistral-7B-Instruct-GGUF
```

Choose a GGUF quantization and download.

12 Verify GPU Usage

```
docker exec -it ollama ollama run gemma3:27b
nvidia-smi
```

13 Common Issues

- Server 500 in WebUI: Usually a bad data directory or wrong `OLLAMA_BASE_URL`. You just need to make sure the location in two places in your `docker-compose.yml` file matches where you placed the `ollama` folder.
- Models not visible: Do not include `/api` in `OLLAMA_BASE_URL`.
- Slow model load: Ensure models are on NVMe, not HDD or NAS.

14 Useful PowerShell Commands (Monitoring & Troubleshooting)

14.1 View Running Containers

Shows all active containers and their status:

```
docker ps
```

Include stopped containers:

```
docker ps -a
```

14.2 View Container Logs (Most Important)

Check Open WebUI logs:

```
docker logs openwebui --tail 200
```

Check Ollama logs:

```
docker logs ollama --tail 200
```

Follow logs live:

```
docker logs -f ollama
```

14.3 Restart Services

Restart both containers:

```
docker compose restart
```

Restart a single container:

```
docker restart ollama
```

14.4 Check Ollama Health & Models

List installed models:

```
docker exec -it ollama ollama list
```

Get Ollama runtime info:

```
docker exec -it ollama ollama info
```

Test model execution:

```
docker exec -it ollama ollama run llama3:8b
```

14.5 Verify GPU Usage (Critical)

While a model is running:

```
nvidia-smi
```

Watch GPU usage in real time:

```
nvidia-smi -l 2
```

14.6 Test Container-to-Container Connectivity

From inside Open WebUI:

```
docker exec -it openwebui sh -lc "wget -qO- http://ollama:11434"
```

Confirm Ollama API responds:

```
curl http://localhost:11434/api/tags
```

14.7 Inspect Networking (ai-net)

Confirm containers are on the same network:

```
docker inspect ollama --format "{{json .NetworkSettings.Networks}}"
docker inspect openwebui --format "{{json .NetworkSettings.Networks}}"
```

List Docker networks:

```
docker network ls
```

Inspect the AI network:

```
docker network inspect ai-net
```

14.8 Clean Restart (When Things Get Weird)

Stop and remove containers:

```
docker compose down
```

Recreate containers:

```
docker compose up -d --force-recreate
```

14.9 Reset Open WebUI Data (Fixes Many 500 Errors)

This removes WebUI settings and chats:

```
docker compose down
rmdir /s /q C:\ollama\openwebui
mkdir C:\ollama\openwebui
docker compose up -d
```

14.10 Disk & Storage Checks

Confirm where models are stored:

```
dir C:\ollama\models
```

Check disk performance (quick sanity):

```
winsat disk -drive c
```