ChatGPT

Why the `open-webui:cuda` Container Fails to Start (WSL2 GPU Issue)

Summary of the Issue: The container startup is failing during NVIDIA GPU initialization, with an error about adding device rules and a BPF (Berkeley Packet Filter) program:

> *"nvidia-container-cli: mount error: failed to add device rules: unable to generate new device filter program... load program: invalid argument: last insn is not an exit or jmp...."*

This indicates the NVIDIA Container Runtime (which injects GPU support into the container) is encountering a low-level error when setting up GPU device access. Notably, the message "Auto-detected mode as 'legacy'" appears just before the failure. The container worked a week ago, so something changed recently in the environment or image.

Likely Causes of the Failure

1. **NVIDIA Container Toolkit Update (Cgroups & eBPF Change):** A recent update to the NVIDIA container runtime or Docker Desktop likely introduced stricter use of Linux cgroups v2 for device access. Modern versions of `libnvidia-container` use cgroup v2's eBPF-based device filtering to control GPU access. If the host system has **eBPF JIT hardening** enabled at a high level, it can break the NVIDIA runtime's ability to attach its device filter. In fact, NVIDIA's 1.8+ runtime started handling cgroup v2 by injecting BPF programs, which **can trigger exactly this error** on hardened kernels [1] [2]. In your case (WSL2 backend), the Docker/WSL kernel likely has cgroup v2 enabled and *possibly* a security setting that causes the BPF program injection to fail. This is a known bug in the NVIDIA container runtime – when it tries to prepend new BPF rules to an existing cgroup device filter, the kernel rejects it if certain hardening (like "constant blinding") is active [3]. The result is the *"invalid argument: last insn is not an exit or jmp"* error, as the NVIDIA hook ends up loading a malformed BPF program of zero bytes [4]. In short, a security measure in the kernel is preventing the NVIDIA hook from installing its device access rules.

2. **Why now?** It's likely that either Docker Desktop or the WSL2 kernel was updated recently. For example, Docker Desktop 4.31+ upgraded the NVIDIA Container Toolkit to v1.15.0 [5], and newer Docker releases (you have 4.41.2) ship with up-to-date NVIDIA runtimes (v1.17+). These newer versions default to using cgroup-based device control (no-cgroups mode is off by default) [6] [7]. If previously your setup was using a legacy/compatibility mode (or if `no-cgroups=true` was set in the config), it may have avoided the BPF issue. Now, with the updated runtime, it's trying the "proper" cgroup method and hitting the bug. WSL2's Linux kernel might also have recently enabled `bpf_jit_harden=2` by default (a security default on some distros), which would trigger this issue. This aligns with reports on Linux where upgrading libnvidia-container started causing *"last insn is not an exit or jmp"* errors until cgroup use was disabled or the kernel settings adjusted [8] [9].

3. **WSL2 Specifics (Cgroups and GPU):** WSL2 is a Linux environment inside Windows, and GPU support is enabled via a paravirtualized GPU. Historically, using the NVIDIA runtime in WSL2 sometimes required special config (like disabling cgroups usage) to work around WSL's partial cgroup implementation. It's possible that previously your Docker/WSL setup implicitly didn't use cgroup device filtering (hence it "just worked"). Now, after an update, the runtime "auto-detected" an environment where it **attempted the legacy cgroup mechanism**. (Counterintuitively, **"legacy" in the log refers to the older cgroup device control scheme** which uses these BPF filters under cgroup v2). In a hardened WSL kernel, this fails. In short, WSL2 + updated NVIDIA runtime + hardened BPF = device filter injection error.

4. **Open-WebUI Image Changes:** We found no direct evidence that the `ghcr.io/open-webui/open-webui:cuda` image itself introduced this bug. The error occurs in the prestart **hook** (before the container's own code runs), suggesting it's an issue in the host runtime rather than something inside the container. However, if you pulled a new Open-WebUI image recently, it might now be based on a newer CUDA base (e.g., CUDA 12) which **requires** the host's container stack to be up-to-date. (For example, new CUDA versions or new driver capabilities sometimes require updated NVIDIA container runtimes). It's worth ensuring your Docker Desktop's NVIDIA integration is compatible with your NVIDIA driver. In your case, you have a very new driver (576.52, May 2025), and indeed NVIDIA has noted that drivers from the 555.xx series onward require Docker Desktop >=4.31 (with Container Toolkit 1.15+) to function properly [10]. Since you're on 4.41.2, this requirement is met. So, the driver/toolkit version mismatch is probably **not** the culprit here (you'd see a different error like "NVML: Unknown Error" or missing symbols if it were). The problem is more about how the runtime is configuring GPU access (the BPF/cgroup issue above), not the CUDA libraries inside the container.

5. **NVIDIA Container Runtime Bug:** In summary, what you're encountering is a **known bug** in NVIDIA's container runtime when used on systems with cgroup v2 and certain kernel security settings [3]. Multiple users on native Linux have reported identical errors after upgrading the NVIDIA toolkit, and it's the same root cause affecting your WSL2 setup. The NVIDIA team has acknowledged this and provided workarounds, though a permanent fix in the toolkit may still be pending (as of NVIDIA Container Toolkit ~1.14–1.17) [11].

How to Fix or Work Around the Issue

1. Adjust the Kernel's BPF JIT Hardening Level: The quickest fix is to relax the BPF hardening so the NVIDIA hook can load its filter. Specifically, setting the sysctl `net.core.bpf_jit_harden` to `1` (or `0`) will disable the problematic "constant blinding" feature while still keeping some security. Users who hit this issue have confirmed that **lowering** `bpf_jit_harden` **from 2 to 1 resolves the startup error** [12] [13].

- **How to do this in WSL2/Docker Desktop:** You'll need to apply the sysctl inside the Linux environment that Docker is running. Since Docker Desktop uses an internal WSL2 distro (`docker-desktop`), you can access it via PowerShell or cmd:

```
wsl -d docker-desktop sysctl -w net.core.bpf_jit_harden=1
```

This command tells the Docker Desktop's WSL kernel to use hardening level 1. If it returns no error, retry launching the container. (Alternatively, you can open a shell in `docker-desktop` and use `sudo sysctl -w ...` there.)

- **Persisting the change:** By default, this change will reset when Docker Desktop/WSL restarts. To make it persistent, you can add it to a sysctl config file. For instance:

```
wsl -d docker-desktop sh -c "echo 'net.core.bpf_jit_harden=1' >> /etc/
sysctl.d/99-nvidia.conf"
```

Then restart Docker Desktop. This uses the approach recommended by NVIDIA engineers [12]. After this, the NVIDIA runtime's BPF program should load successfully, avoiding the "last insn is not an exit or jmp" error.

- **Security note:** Setting it to 1 still keeps JIT hardening for unprivileged BPF, just disables the part that confuses the NVIDIA hook. In hardened distros, 1 is generally considered safe. Setting `0` would turn off JIT hardening entirely (the Minikube docs even suggest `0` for maximum compatibility [14], but try `1` first as it's sufficient in this case [13]).

2. Enable Legacy Device Handling (no-cgroups mode): An alternative workaround is to force the NVIDIA runtime to skip cgroup device filtering altogether and use the older "direct mount" method for GPU devices. This is done by editing the NVIDIA container runtime config to `no-cgroups=true`. In a standard Linux install, that's in `/etc/nvidia-container-runtime/config.toml`. With cgroups disabled, the runtime won't attempt any eBPF magic and will just mount the GPU device nodes into the container. This was the fix many WSL2 users needed in earlier days.

- **In practice:** If you can edit the config in the Docker Desktop's Linux environment, find the `[nvidia-container-cli]` section and set:

```
no-cgroups = true
```

Then restart Docker. (Docker Desktop might overwrite this on update, but you can try.) One user confirmed that re-enabling `no-cgroups=true` stopped the error on Ubuntu when encountering this same issue [15].

- **Important:** When `no-cgroups=true`, Docker's `--gpus all` flag may not automatically add the GPU devices. You might have to manually specify the device nodes and driver capabilities. For example:

```
docker run --rm --runtime=nvidia \
  --device /dev/nvidia0 --device /dev/nvidiactl --device /dev/nvidia-uvm --
device /dev/nvidia-modeset \
  ghcr.io/open-webui/open-webui:cuda
```

(Add all relevant `/dev/nvidia*` nodes.) This is the method suggested in NVIDIA's GitHub issue [15] as a workaround. Essentially, you're bypassing cgroup control and giving the container direct access to the GPU devices.

- In many cases, simply **setting** `no-cgroups=true` **is enough** and you can still use `--gpus all` as usual, because the runtime will detect the GPU and mount devices. But if you find the container still not "seeing" the GPU, use the explicit `--device` flags as above.

3. Verify Driver/Toolkit Compatibility: Although the error you see is not a direct driver-version issue, it's good to double-check that your setup is coherent: - Ensure Docker Desktop is fully updated (which it is, 4.41.2 is recent). - Your NVIDIA driver 576.52 is very new (Game Ready Driver from May 19, 2025). This driver should be fine with the latest Container Toolkit. (Older Docker versions would have needed an update for 576.xx, but you've already got a recent Docker Desktop, which includes an updated toolkit [5] .) - If you had *not* updated Docker, a symptom would be containers starting but failing to use the GPU (errors like "CUDA_ERROR_NOT_FOUND: named symbol not found" or NVML init failures). Since your error is happening at startup hook time, you've likely cleared this hurdle. Just keep this in mind: when either the GPU driver or Docker updates, they should be somewhat in sync. (NVIDIA's recommendation: Docker Desktop ≥4.31 for 55x+ drivers [10] .)

4. Check Open-WebUI Container Requirements: Confirm that the Open-WebUI container is meant to run with GPU on your setup: - It should be launched with `--gpus all` (as you did) and with the NVIDIA runtime active. Double-check that Docker is indeed using the `nvidia` runtime (Docker Desktop should handle this automatically when `--gpus` is used, but no harm ensuring the daemon JSON has the nvidia runtime configured). - The Open-WebUI docs mention installing the NVIDIA Container Toolkit on the host (which, in WSL2/Docker Desktop, is handled internally) and using the `:cuda` tag for GPU support [16] . You've done these steps, so that's fine. - There haven't been reports that a recent Open-WebUI update broke GPU startup; the issues reported (e.g., on Reddit) were usually due to the environment not finding `libnvidia-ml.so.1` or not having permission, which again ties back to the toolkit config rather than the container itself [17] . Your error is occurring slightly earlier (during device rule setup), so focus on the runtime fix.

5. (Temporary Test) Run Privileged: As a diagnostic (not a permanent solution), you could try running the container in privileged mode: `docker run --privileged --gpus all ... open-webui:cuda`. A privileged container will have full device access by default. If it starts up without error in privileged mode, that strongly confirms the issue was the device cgroup filter. (Do **not** use privileged mode permanently for this, as it's not ideal security-wise; it's just a useful test.)

References & Supporting Info

- The error is a known bug in NVIDIA's libnvidia-container when using cgroup v2 device filtering with certain kernel security settings. See NVIDIA's GitHub issue [2] – it shows the exact same error and setup (Ubuntu 22.04, Docker 24.x, Container Toolkit 1.14+) and was resolved by adjusting the kernel's `bpf_jit_harden` setting [12] . NVIDIA's engineer explicitly links this error to the *cgroup device filter and eBPF constant blinding* and provides the sysctl workaround.

- Markus Böhm's analysis of the root cause (on libnvidia-container issue #176) explains that *"constant blinding"* of eBPF bytecode causes the NVIDIA container hook to read back an all-zero filter, which is an invalid program (no proper exit instruction), hence the *"last insn is not an exit or jmp"* verifier error [3]. Reducing `bpf_jit_harden` disables that blinding so the program can be appended and loaded normally [13].

- Users on hardened kernels (Arch Linux, Bottlerocket, etc.) and WSL2 have hit this. A user report on GitHub noted that **disabling cgroups (legacy mode)** or tweaking the sysctl allowed containers like `nvidia/cuda:11.0-base nvidia-smi` to run without errors [15] [18]. This aligns with your observation that it was working a week ago – likely under a legacy mode – and now it's trying the new default path.

- NVIDIA's developer forums also indicate that with newer Windows drivers (e.g. 555.xx+), you must update Docker Desktop's toolkit to avoid other issues [10]. In your case, you have done so, but it's good to be aware that driver and toolkit versions are intertwined. Since your error is different, the focus remains on the cgroup/BPF fix.

In summary, the failure is caused by a change in how the NVIDIA container runtime is setting up GPU access in your WSL2 environment. The solution is to either adjust the kernel's BPF security setting or revert the runtime to a legacy device management mode. Applying the sysctl tweak (`bpf_jit_harden=1`) is the recommended approach [12], as it resolves the error while still allowing you to use `--gpus all` normally. Once that's done (and Docker is restarted), your `open-webui:cuda` container should launch successfully with GPU support.

---

[1] [3] [4] [6] [7] [8] [9] [13] [15] [18] nvidia-container-cli: mount error: failed to add device rules: unable to generate new device filter program from existing programs: unable to create new device filters program: load program: invalid argument: last insn is not an exit or jmp · Issue #176 · NVIDIA/libnvidia-container · GitHub
https://github.com/NVIDIA/libnvidia-container/issues/176

[2] [11] [12] `sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi` failing `last insn is not an exit or jmp` · Issue #117 · NVIDIA/nvidia-container-toolkit · GitHub
https://github.com/NVIDIA/nvidia-container-toolkit/issues/117

[5] [10] Windows WSL / Docker Desktop users must update to Docker Desktop v4.31.1 - NVIDIA AI Workbench - NVIDIA Developer Forums
https://forums.developer.nvidia.com/t/windows-wsl-docker-desktop-users-must-update-to-docker-desktop-v4-31-1/299421

[14] Using NVIDIA GPUs with minikube | minikube
https://minikube.sigs.k8s.io/docs/tutorials/nvidia/

[16] Open WebUI: Home
https://docs.openwebui.com/

[17] Unable to get the docker installation working through WSL2. : r/OpenWebUI
https://www.reddit.com/r/OpenWebUI/comments/1h4ugh1/unable_to_get_the_docker_installation_working/