MiPhi Semiconductors Pvt Ltd
GUI Instruction manual

MiPhi Semiconductors Pvt Ltd
Tel: +91 89044 05190 // +91 80421 66207
E-mail: inquire@miphi.in

Table of Contents

1. Environment Preparation

1.1 Supported OS

- Ubuntu 22.04.4 LTS - using this version helps avoid unexpected compatibility issues.

1.2 Install and start Docker

- Docker bundles your front end, back end, and database into one package.
- You start the entire setup with a single command i.e. **docker run**, so you don’t need to install each part separately or worry about version mismatches.

1.3 NVIDIA Container Toolkit

- NVIDIA Container Toolkit installed for GPU support inside Docker
- This makes heavy tasks run much faster than if they only used the regular processor.

2. Retrieve Docker .tar Files

Download the Docker image .tar files directly into the server’s working directory via wget, ensuring they’re ready for immediate loading

```
wget http://<server>/stream.tar
wget http://<server>/gpu.tar
```

3. Load Docker Images

3.1 Prepare & Import Docker Image Archives

Make sure you're in the **same directory** where you ran your wget commands (where stream.tar and gpu.tar were downloaded).

- Import the GUI image:
  - The stream.tar file is a complete snapshot of your GUI application—including the web interface, backend services, and database schema—all pre-configured and packaged together.
  - Loading it into Docker unpacks that package, calls it **integration-stream**, and lets you start everything with a single command.

```
docker load -i stream.tar
```

- Import the GPU image:
  - The gpu.tar file provides the tools your container needs to read GPU metrics—like DRAM and memory usage—so you can keep an eye on performance.
  - Loading it makes the **testing-gpu** image available in Docker, ready for use by your GPU utility container.

```
docker load -i gpu.tar
```

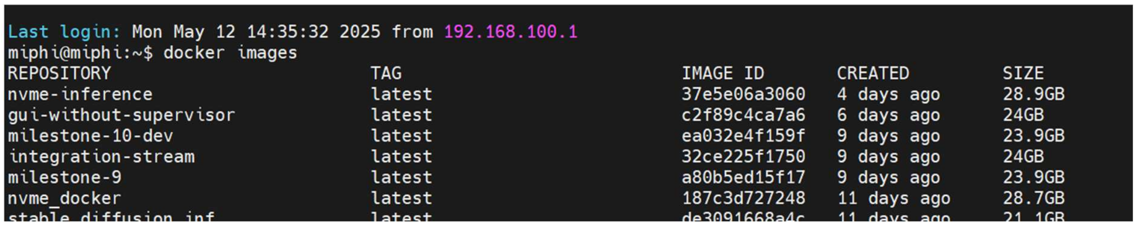
3.2 Verify the Docker Images

- Run the following command to list Docker images:
  - Make sure both your GUI and GPU images are ready to use before starting any containers.

```
docker images
```

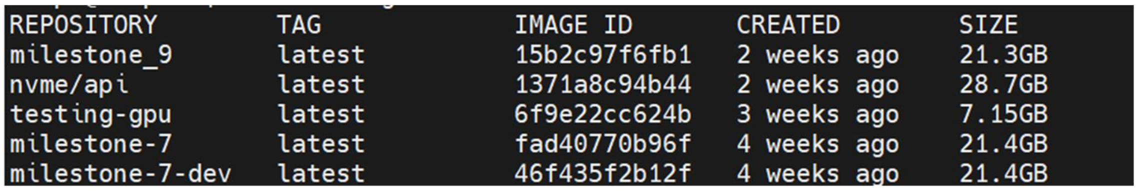
You should see:

- integration-stream (GUI image)



```
Last login: Mon May 12 14:35:32 2025 from 192.168.100.1
miphi@miphi:~$ docker images
REPOSITORY                   TAG                             IMAGE ID       CREATED        SIZE
nvme-inference               latest                          37e5e06a3060   4 days ago     28.9GB
gui-without-supervisor       latest                          c2f89c4ca7a6   6 days ago     24GB
milestone-10-dev             latest                          ea032e4f159f   9 days ago     23.9GB
integration-stream           latest                          32ce225f1750   9 days ago     24GB
milestone-9                  latest                          a80b5ed15f17   9 days ago     23.9GB
nvme_docker                  latest                          187c3d727248   11 days ago    28.7GB
stable_diffusion_inf         latest                          de3091668a4c   11 days ago    21.1GB
```

- testing-gpu (GPU image)



```
REPOSITORY          TAG             IMAGE ID       CREATED        SIZE
milestone_9         latest          15b2c97f6fb1   2 weeks ago    21.3GB
nvme/api            latest          1371a8c94b44   2 weeks ago    28.7GB
testing-gpu         latest          6f9e22cc624b   3 weeks ago    7.15GB
milestone-7         latest          fad40770b96f   4 weeks ago    21.4GB
milestone-7-dev     latest          46f435f2b12f   4 weeks ago    21.4GB
```

3.3 Check Docker Network

- Containers need to share a common network to communicate (your GUI talks to the GPU helper over this network).
- We're using gui_network as that shared channel.

  Ensure the gui_network exists by running:

  ```
  docker network ls
  ```

  This lists all Docker networks so you can see if gui_network is already there.
- If you don't see gui_network in the list, create it using:
  - You must create it so your containers can connect to each other.
  - Once it's created, any container you attach to gui_network will automatically be able to talk to the others.

```
docker network create gui_network
```

4. Run GUI Container

- It starts the full GUI application (front end, back end, database, and inference) all at once inside a single container.
- It gives the container access to your GPUs and fast NVMe storage so that heavy processing runs smoothly.
- It opens the right network ports so you can reach the interface in your browser—and if it ever crashes, Docker will restart it automatically.

Use the following command to start your GUI container:

```
sudo docker run -d \
  --name <container_name> \             # Unique name for the container
  --restart=unless-stopped \
  --gpus "<specified_gpus>" \           # GPUs to expose (e.g. "device=0,1")
  -it \
  --pid=host \
  --ipc=host \
  --privileged=true \
  --ulimit memlock=-1 \
  --ulimit stack=67108864 \
  --network gui_network \
  -p <port_1>:5432 \                     # database port
  -p <port_2>:3005 \                     # backend API port
  -p <port_3>:80 \                       # frontend port
  -p <port_4>:5004 \                     # inference port
  -p <port_5>:8000 \                     # VLLM host port
  -v /dev:/dev \
  -v /models:/models \
  -v <mount_point>:<mount_point> \      # NVMe mount path inside container
  -v /media:/media \
  -v /proc:/proc \
  -v /sys:/sys \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /etc/os-release:/etc/os-release:ro \
  -v /usr/local/datasets:/usr/local/datasets \
  -v /usr/local/models:/usr/local/models \
  -v /dev/mapper:/dev/mapper \
  -e PORT=<port_2> \
  -e INFERENCE=<port_4> \
  -e VLLM_HOST=<port_5> \
  -e NVME_PATH=<mount_point> \
  -e SPECIFIED_GPUS="<specified_gpus>" \
  <image_name>                          # Docker image to launch
```

4.1 Parameters to Update:

- **<container_name>**

  A unique name for your GUI container (e.g. miphi-gui, demo-1).
- **--gpus'"device=0,1"'**

  Which GPU devices on your host you want to expose to the container ("device=0", "device=1", or "device=0,1", etc.).
- **-p <port_1>:5432**

  Database port – maps your host DB port (e.g. 5432) into the contain
- **-p <port_2>:3005**

  Backend API port – maps your host backend port (e.g. 3005) into the container.
- **-p <port_3>:80**

  Frontend port – maps your host HTTP port (e.g. 80) into the container.
- **-p <port_4>:5004**

  Inference port – maps your host inference-service port (e.g. 5004) into the container.
- **-p <port_5>:8000**

  VLLM host port – maps your host VLLM service port (e.g. 8000) into the container.
- **-v <mount_point>:<mount_point>**

  NVMe mount – e.g. if your ultra-fast storage is mounted at /mnt/nvme0 on the host, use -v /mnt/nvme0:/mnt/nvme0.

  This gives your container direct access to that NVMe path.
- **-e NVME_PATH=<mount_point>**

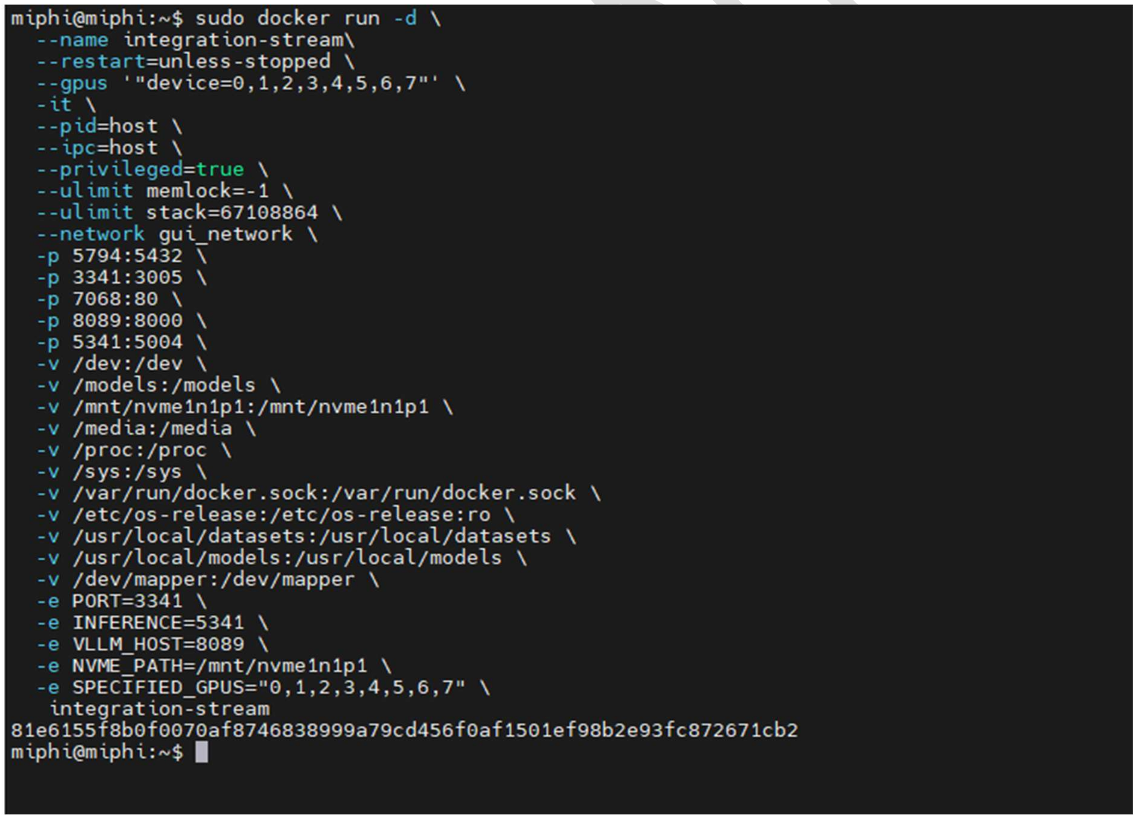
  Set this to your same NVMe path (e.g. NVME_PATH=/mnt/nvme0) so your app knows where to find it inside the container.
- **<image_name>**

  The Docker image you loaded (e.g. integration-stream, testing-gpu, or your custom).

4.2 Example Parameter Values:

Replace the placeholders in your template with the values shown :

- <container_name> → integration-stream
- <image_name> → integration-stream
- "<specified_gpus>" → "device=0,1,2,3,4,5,6,7"
- <port_1> → 5794
- <port_2> → 3341
- <port_3> → 7068
- <port_4> → 5341
- <port_5> → 8089
- <mount_point> → /mnt/nvme1n1p1
- NVME_PATH=<mount_point> → NVME_PATH=/mnt/nvme1n1p1



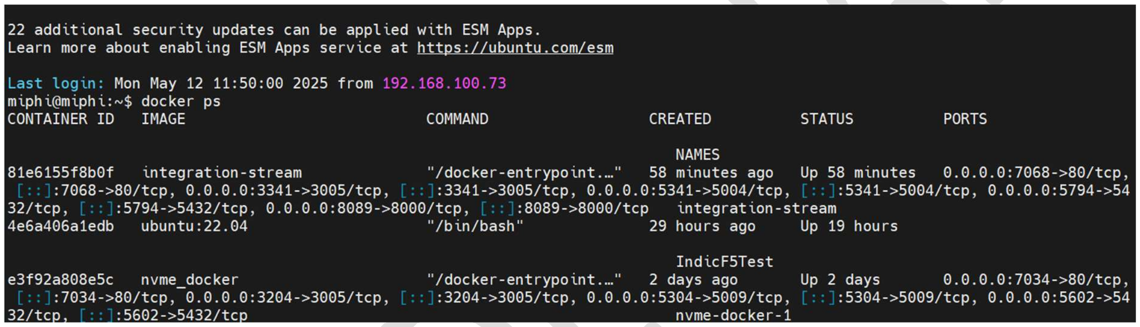
```
miphi@miphi:~$ sudo docker run -d \
  --name integration-stream\
  --restart=unless-stopped \
  --gpus '"device=0,1,2,3,4,5,6,7"' \
  -it \
  --pid=host \
  --ipc=host \
  --privileged=true \
  --ulimit memlock=-1 \
  --ulimit stack=67108864 \
  --network gui_network \
  -p 5794:5432 \
  -p 3341:3005 \
  -p 7068:80 \
  -p 8089:8000 \
  -p 5341:5004 \
  -v /dev:/dev \
  -v /models:/models \
  -v /mnt/nvme1n1p1:/mnt/nvme1n1p1 \
  -v /media:/media \
  -v /proc:/proc \
  -v /sys:/sys \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /etc/os-release:/etc/os-release:ro \
  -v /usr/local/datasets:/usr/local/datasets \
  -v /usr/local/models:/usr/local/models \
  -v /dev/mapper:/dev/mapper \
  -e PORT=3341 \
  -e INFERENCE=5341 \
  -e VLLM_HOST=8089 \
  -e NVME_PATH=/mnt/nvme1n1p1 \
  -e SPECIFIED_GPUS="0,1,2,3,4,5,6,7" \
   integration-stream
81e6155f8b0f0070af8746838999a79cd456f0af1501ef98b2e93fc872671cb2
miphi@miphi:~$
```

4.3 Verify Container Status

- Open your terminal and run

```
docker ps
```

- This command will display a table of all currently running containers, showing their names, IDs, and status.
- In the output, find the row with integration-stream under the NAMES column and check that its STATUS is "Up." That confirms your GUI container is running correctly.



```
22 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon May 12 11:50:00 2025 from 192.168.100.73
miphi@miphi:~$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS

                                                                            NAMES
81e6155f8b0f   integration-stream             "/docker-entrypoint.…"   58 minutes ago   Up 58 minutes   0.0.0.0:7068->80/tcp,
 [::]:7068->80/tcp, 0.0.0.0:3341->3005/tcp, [::]:3341->3005/tcp, 0.0.0.0:5341->5004/tcp, [::]:5341->5004/tcp, 0.0.0.0:5794->54
32/tcp, [::]:5794->5432/tcp, 0.0.0.0:8089->8000/tcp, [::]:8089->8000/tcp   integration-stream
4e6a406a1edb   ubuntu:22.04                   "/bin/bash"              29 hours ago     Up 19 hours

                                                                            IndicF5Test
e3f92a808e5c   nvme_docker                    "/docker-entrypoint.…"   2 days ago       Up 2 days       0.0.0.0:7034->80/tcp,
 [::]:7034->80/tcp, 0.0.0.0:3204->3005/tcp, [::]:3204->3005/tcp, 0.0.0.0:5304->5009/tcp, [::]:5304->5009/tcp, 0.0.0.0:5602->54
32/tcp, [::]:5602->5432/tcp                                                 nvme-docker-1
```

5. Run GPU Utility Container

- It launches a small helper container that has direct access to **all** your GPUs for easy monitoring and testing.
- It opens a host port (e.g. 9999) so you can connect to the GPU utility service from your machine or other tools.

```
sudo docker run -d \
  --gpus all \
  --network gui_network \
  -p <port_no>:9999 \           # host port for GPU utility service
  --name testing-gpu-container \  # container name for easy reference
  <image_gpu_name>
```

5.1 Parameters to Update:

- **-p<port_no>:9999**

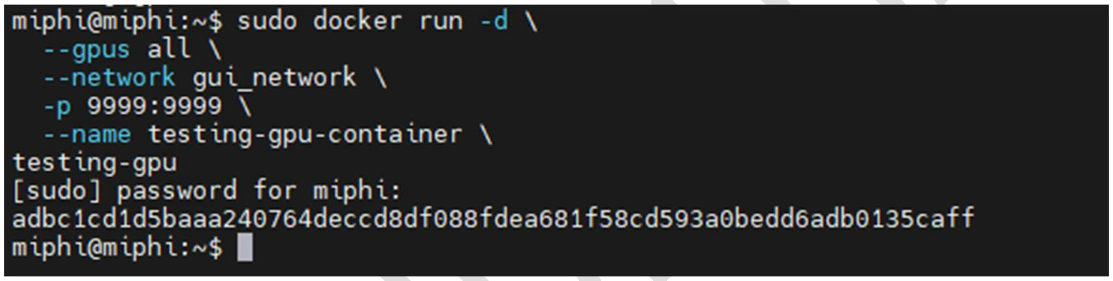
  Host port for the GPU utility service (e.g. 9999).

- **<image_gpu_name>**

  The name of your GPU utility image (e.g. testing-gpu).

5.2 Example Parameter Values:

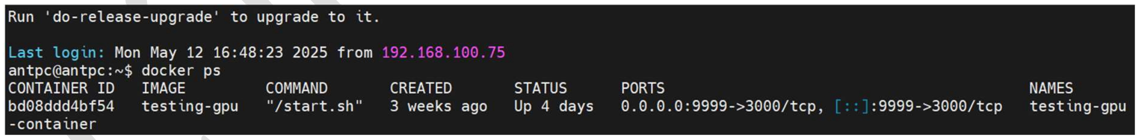
- <port_no> → 9999
- <image_gpu_name> → testing-gpu



```
miphi@miphi:~$ sudo docker run -d \
  --gpus all \
  --network gui_network \
  -p 9999:9999 \
  --name testing-gpu-container \
testing-gpu
[sudo] password for miphi:
adbc1cd1d5baaa240764deccd8df088fdea681f58cd593a0bedd6adb0135caff
miphi@miphi:~$ 
```

5.3 Verify GPU Container:

- Run **docker ps** and look for testing-gpu-container in the list.
- Check that its STATUS is “Up” to confirm the GPU helper is running.

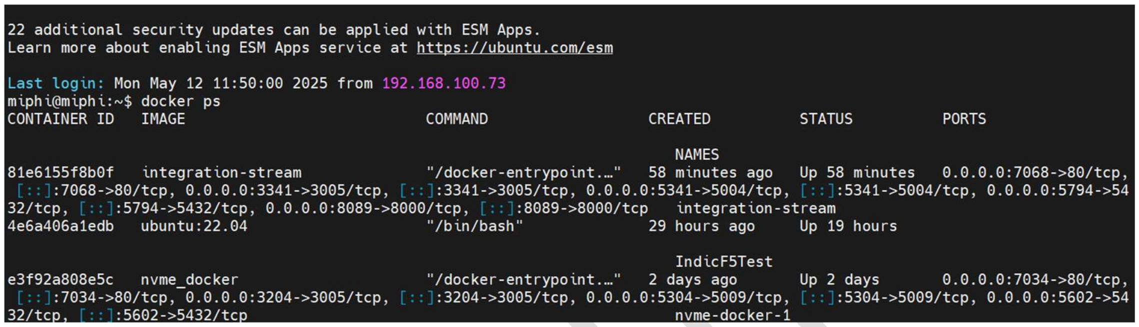


```
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon May 12 16:48:23 2025 from 192.168.100.75
antpc@antpc:~$ docker ps
CONTAINER ID   IMAGE         COMMAND       CREATED       STATUS     PORTS                                         NAMES
bd08ddd4bf54   testing-gpu   "/start.sh"   3 weeks ago   Up 4 days  0.0.0.0:9999->3000/tcp, [::]:9999->3000/tcp   testing-gpu
-container
```

6. Post-Deployment Checklist

6.1 Verify running containers

- Run **docker ps** and ensure both your GUI and GPU containers are listed and in the "Up" state.



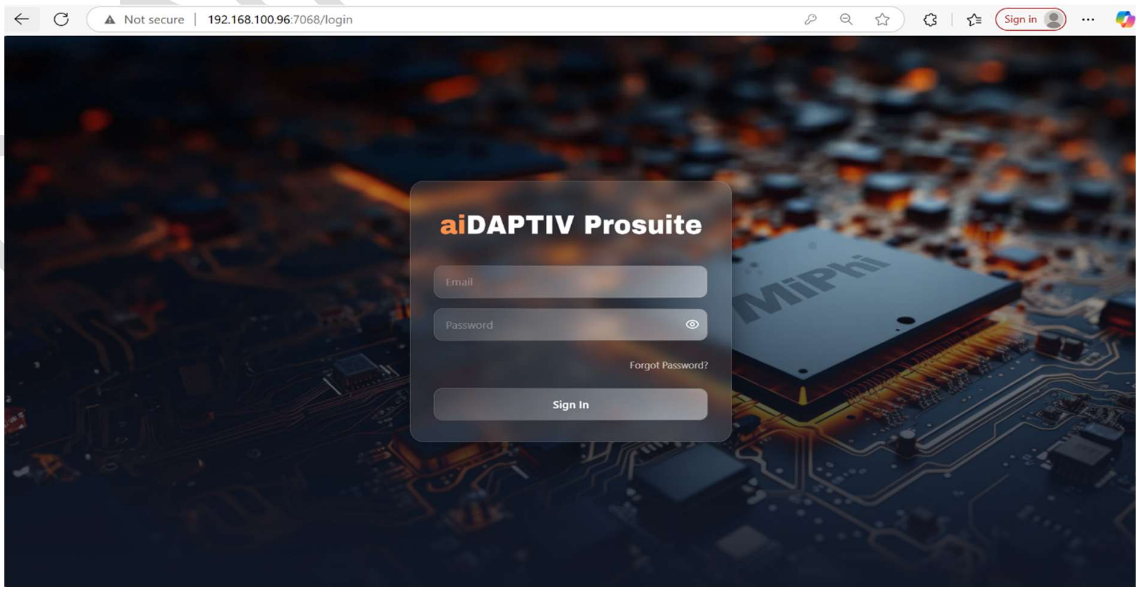
```
22 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon May 12 11:50:00 2025 from 192.168.100.73
miphi@miphi:~$ docker ps
CONTAINER ID   IMAGE                           COMMAND                  CREATED          STATUS          PORTS

                                                                          NAMES
81e6155f8b0f   integration-stream              "/docker-entrypoint.…"   58 minutes ago   Up 58 minutes   0.0.0.0:7068->80/tcp,
 [::]:7068->80/tcp, 0.0.0.0:3341->3005/tcp, [::]:3341->3005/tcp, 0.0.0.0:5341->5004/tcp, [::]:5341->5004/tcp, 0.0.0.0:5794->54
32/tcp, [::]:5794->5432/tcp, 0.0.0.0:8089->8000/tcp, [::]:8089->8000/tcp   integration-stream
4e6a406a1edb   ubuntu:22.04                    "/bin/bash"              29 hours ago     Up 19 hours

                                                                          IndicF5Test
e3f92a808e5c   nvme_docker                     "/docker-entrypoint.…"   2 days ago       Up 2 days       0.0.0.0:7034->80/tcp,
 [::]:7034->80/tcp, 0.0.0.0:3204->3005/tcp, [::]:3204->3005/tcp, 0.0.0.0:5304->5009/tcp, [::]:5304->5009/tcp, 0.0.0.0:5602->54
32/tcp, [::]:5602->5432/tcp                                               nvme-docker-1
```

6.2 Access the GUI

- In your docker ps output's PORTS column, find the entry that maps to 80/tcp (e.g. 0.0.0.0:7068->80/tcp); the number before :80 (here, 7068) is your front-end port

```
http://<server-ip>:<gui_port>
```

(e.g. http://192.168.100.96:7068)

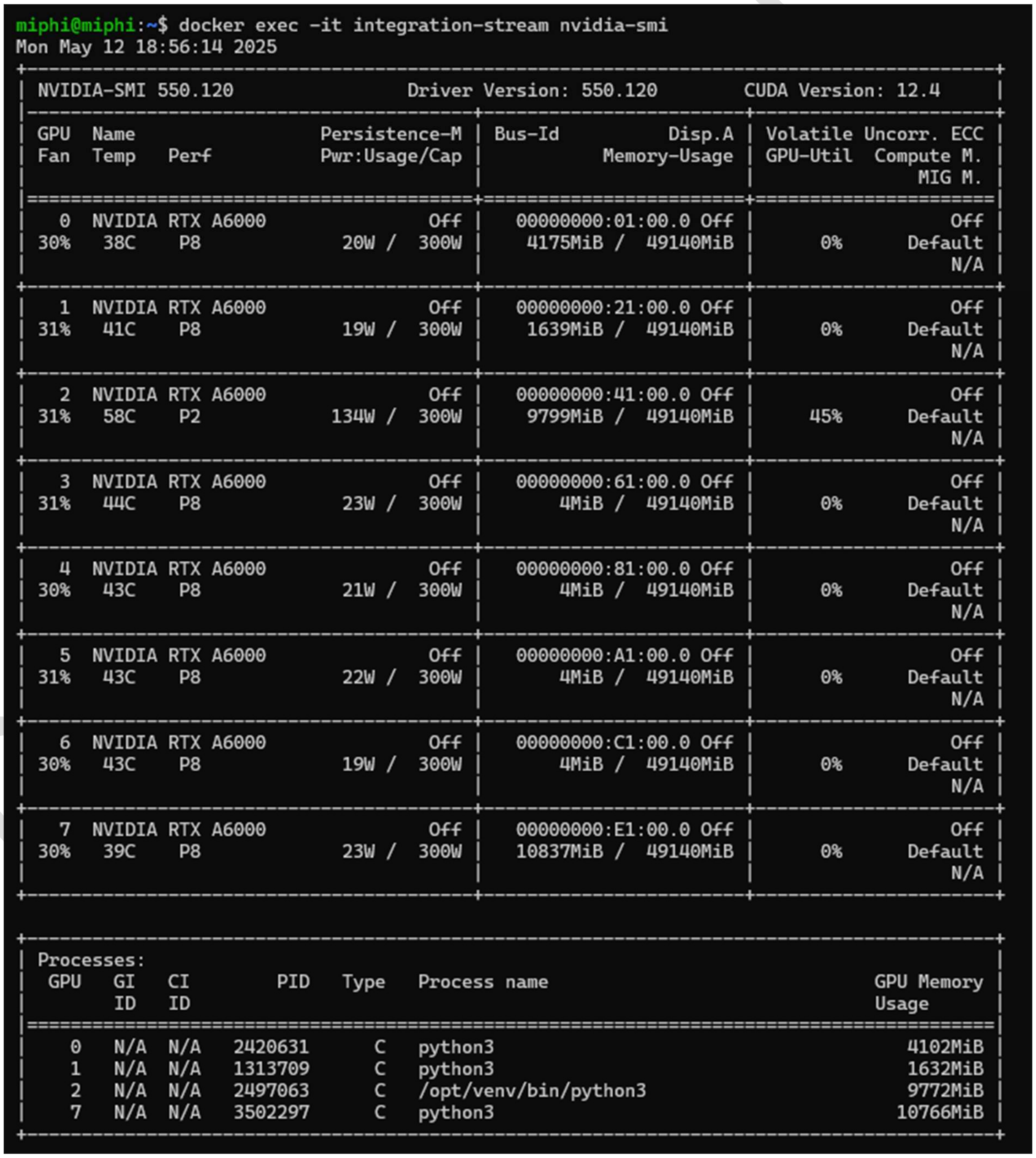


6.3 Confirm GPU access

- Execute inside your GUI container:

```
docker exec -it <container_name> nvidia-smi
```

- You should see your GPU(s) listed with their status and memory usage.



```
miphi@miphi:~$ docker exec -it integration-stream nvidia-smi
Mon May 12 18:56:14 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.120                Driver Version: 550.120        CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A6000               Off |   00000000:01:00.0 Off |                  Off |
| 30%   38C    P8             20W /  300W |    4175MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX A6000               Off |   00000000:21:00.0 Off |                  Off |
| 31%   41C    P8             19W /  300W |    1639MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA RTX A6000               Off |   00000000:41:00.0 Off |                  Off |
| 31%   58C    P2            134W /  300W |    9799MiB /  49140MiB |     45%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA RTX A6000               Off |   00000000:61:00.0 Off |                  Off |
| 31%   44C    P8             23W /  300W |       4MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   4  NVIDIA RTX A6000               Off |   00000000:81:00.0 Off |                  Off |
| 30%   43C    P8             21W /  300W |       4MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   5  NVIDIA RTX A6000               Off |   00000000:A1:00.0 Off |                  Off |
| 31%   43C    P8             22W /  300W |       4MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   6  NVIDIA RTX A6000               Off |   00000000:C1:00.0 Off |                  Off |
| 30%   43C    P8             19W /  300W |       4MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   7  NVIDIA RTX A6000               Off |   00000000:E1:00.0 Off |                  Off |
| 30%   39C    P8             23W /  300W |   10837MiB /  49140MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A   2420631      C   python3                                     4102MiB |
|    1   N/A  N/A   1313709      C   python3                                     1632MiB |
|    2   N/A  N/A   2497063      C   /opt/venv/bin/python3                       9772MiB |
|    7   N/A  N/A   3502297      C   python3                                    10766MiB |
+-----------------------------------------------------------------------------------------+
```