Lambda GPU Cloud: SSH, Instance Launch, and Isaac Sim Deployment Guide

December 1, 2025

Contents

1 Prerequisites: Generating and Uploading Your SSH Key

An SSH key pair (public and private key) is required to securely connect to your cloud instance.

1.1 Step 1: Generate and Name Your SSH Key

The easiest method is to let the Lambda Cloud platform generate a key for you, which results in a private key (‘.pem‘ file) that you must secure.

1. Navigate to the SSH key management section of the Lambda dashboard.
2. Click on the option to **Generate a new SSH key**.
3. Provide a **Name** (e.g., `test-key`) and click **Create**. The private key file (‘test-key.pem‘) will download to your computer.

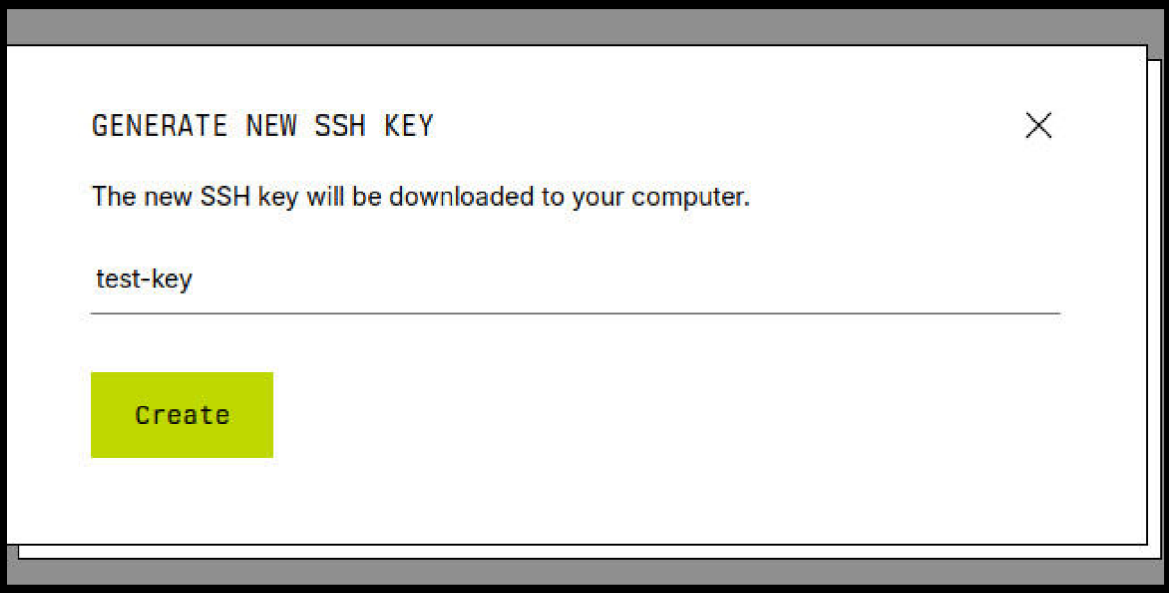

Figure 1: **Generate New SSH Key Dialog**: Prompt to enter a name for the new key before creation and download.

Purpose: *This step creates the secure cryptographic key pair necessary for passwordless login and downloads the private key file.*

1.2 Step 2: Securing Your Private Key File

The private key file (‘.pem‘ or ‘id_rsa‘) must be secured and have the correct permissions to prevent unauthorized access.

Linux/macOS Commands

1. **Move the Key:** Move the downloaded private key (e.g., `test-key.pem`) from your Downloads folder to your secure `.ssh` directory:

```
mv ~/Downloads/test-key.pem ~/.ssh
```

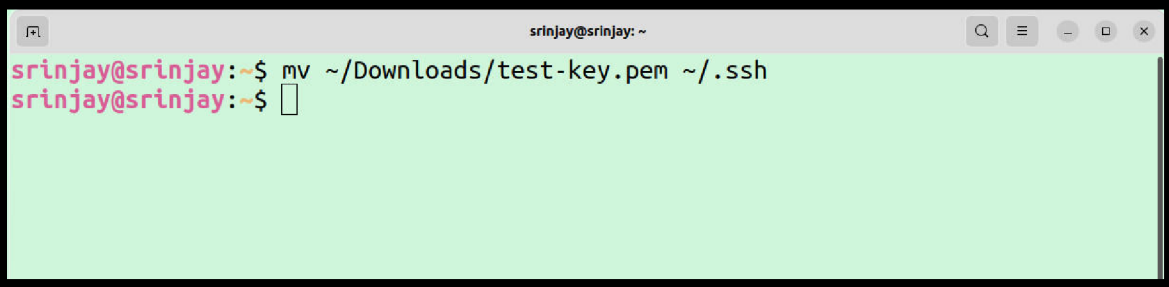

Figure 2: **Moving Private Key (Linux)**: Terminal command to move the private key file into the /.ssh directory.

Purpose: *Moves the key to the standard hidden directory where SSH clients expect to find authentication files.*

2. **Set Permissions:** Set the restrictive permissions (‘600‘) so only the owner can read and write the file.

```
chmod 600 ~/.ssh/test-key.pem
```

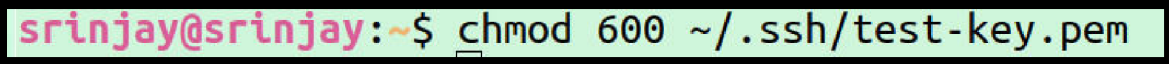

Figure 3: **Setting Private Key Permissions (Linux)**: Executing the ’chmod 600’ command to restrict key access.

Purpose: *Restricts file access to only the current user, a security requirement for SSH clients to use the key.*

Windows Setup (File Explorer/GUI)

On Windows, you must restrict permissions so only your user account can access the file using the graphical interface:

1. **Locate and Right-Click:** Find your private key file (e.g., `id_rsa` or `test-key.pem`). Right-click the file and select **Properties**.
2. **Open Security Settings:** Go to the **Security** tab and click **Advanced**.
3. **Disable Inheritance:** Click the **Disable inheritance** button. When prompted, select **Remove all inherited permissions from this object**.
4. **Add User Permissions:** Click **Add**, then select a principal (your Windows user account). Grant it **Full control** or **Read & execute** permissions only. Ensure no other users or groups (except SYSTEM) have access.

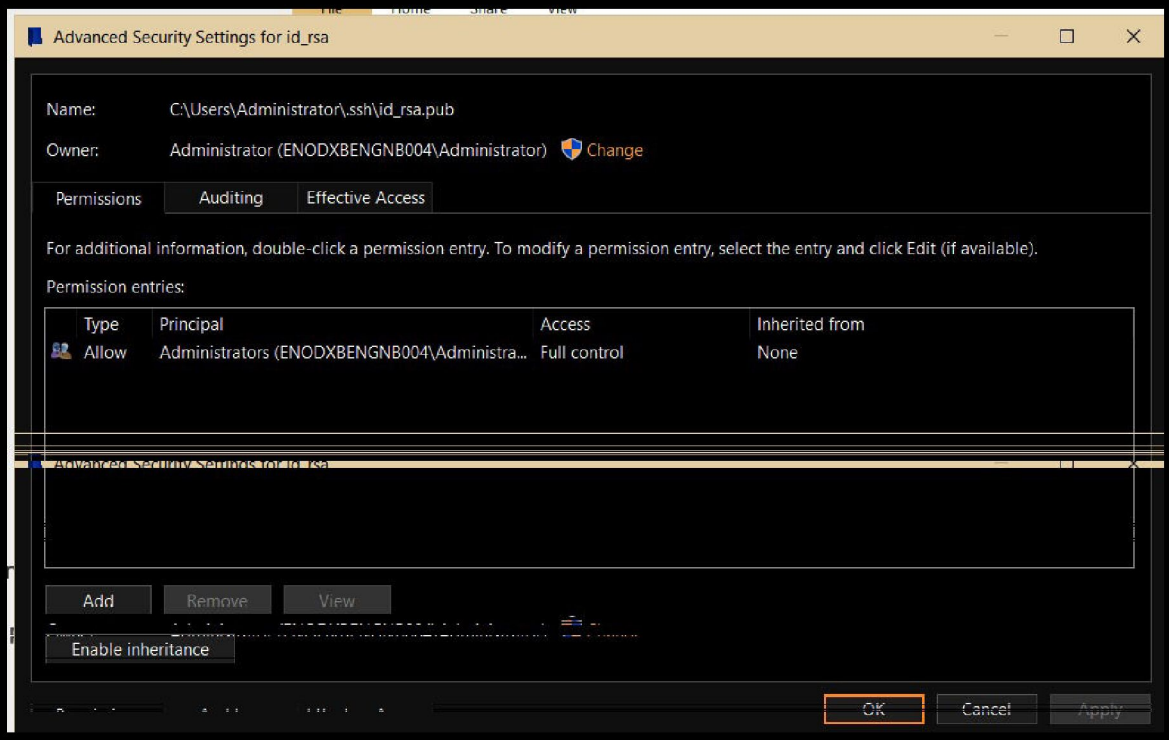

Figure 4: **Windows Advanced Security Settings**: Advanced Security Settings dialogue box showing that inheritance is disabled and permissions are restricted to the owner (Administrator).

Purpose: *Using the graphical interface ensures the operating system's security model is correctly configured to restrict access to the private key, which is mandatory for SSH.*

2 Launching and Connecting to the GPU Instance

2.1 Step 3: Launching the GPU Instance

1. From the GPU Instances dashboard, click the **Launch instance** button.
2. **Configure Instance:** Select the type (`1x A100`), Base Image (`Lambda Stack 22.04`), Filesystem, and ensure the correct **SSH key** (e.g., `test-key`) is selected.
3. Click **Launch instance**. Wait for the instance status to change to **Running**.

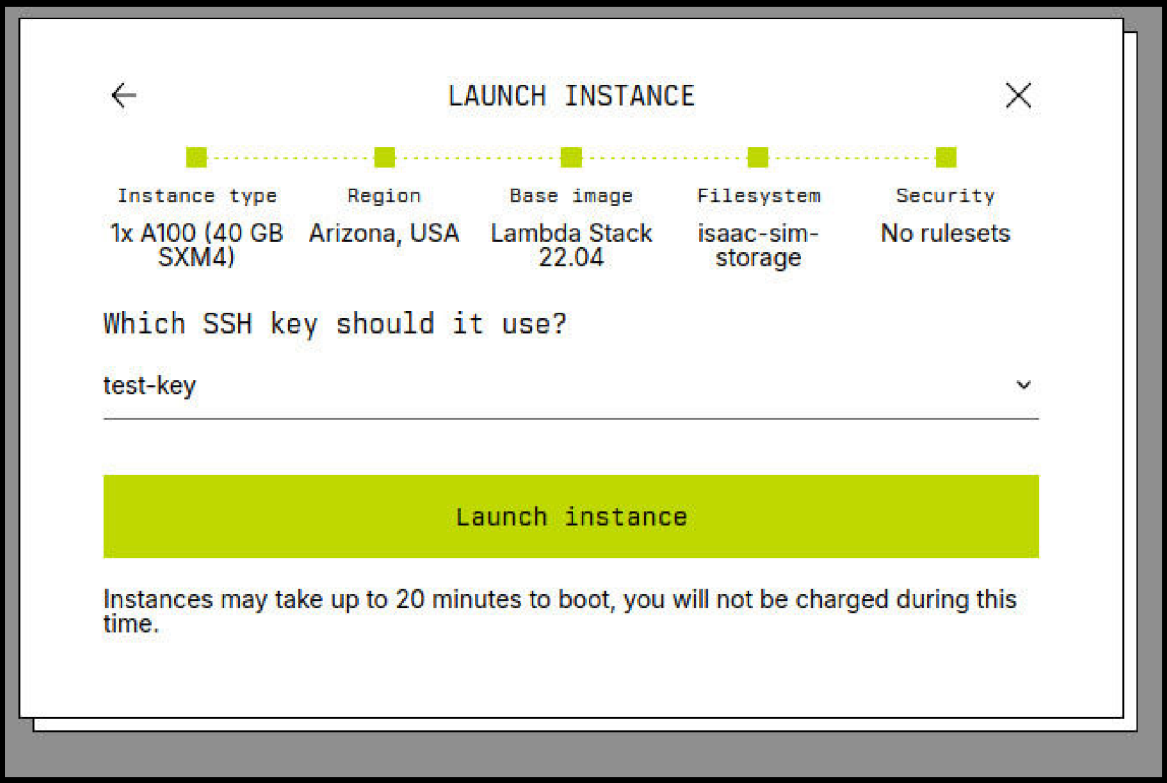

Figure 5: **Review and Launch Instance**: Final configuration step showing instance type, image, filesystem, and selected key before launch.

Purpose: *Selects the GPU hardware, operating system image, and authentication method to provision the cloud server.*

2.2 Step 4: Connecting via SSH

Once the instance is running, the dashboard displays the IP address (e.g., `137.131.3.39`) and login username (`ubuntu`).

Linux/macOS Commands

1. Open your terminal.
2. Use the SSH login command, pointing to your private key file (`test-key.pem`) with the `-i` flag:

   ```
   ssh -i ~/.ssh/test-key.pem ubuntu@<IP_ADDRESS>
   ```

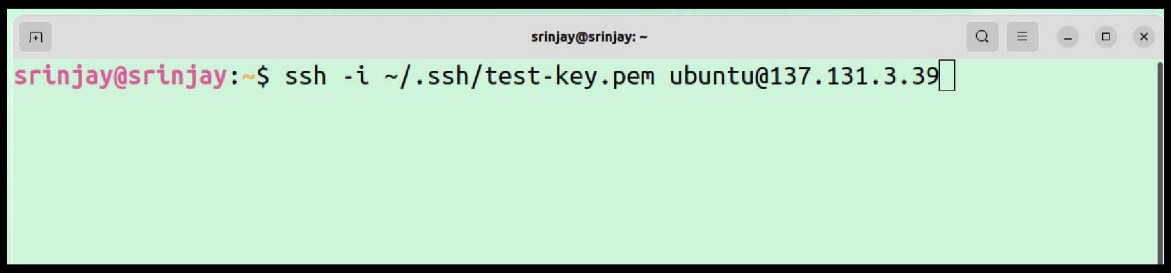

Figure 6: **SSH Connection Command (Linux)**: Executing the SSH command using the private key path and IP address.

Purpose: *Initiates a secure connection to the remote instance using the specified key file and the default user (‘ubuntu‘).*

3. **Accept Host Key:** On the first connection, type `yes` when prompted about the host's authenticity.

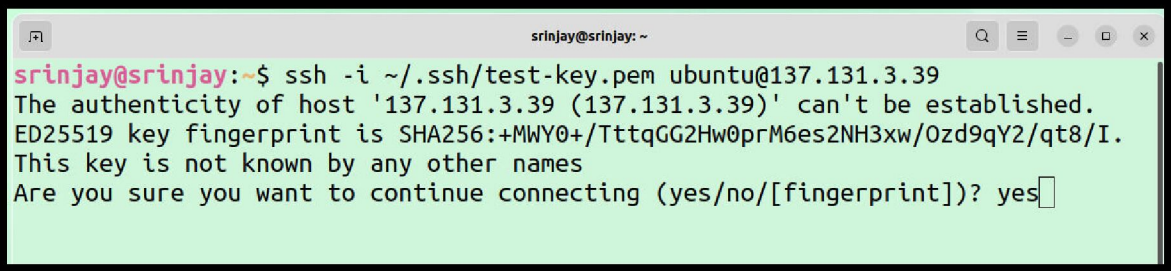

Figure 7: **SSH Host Authenticity Prompt**: Confirming the host's authenticity on first connection by typing 'yes'.

Purpose: *Adds the remote server's unique identifying key to your local list of known hosts for future verification.*

4. You should see the welcome banner and the command line prompt.

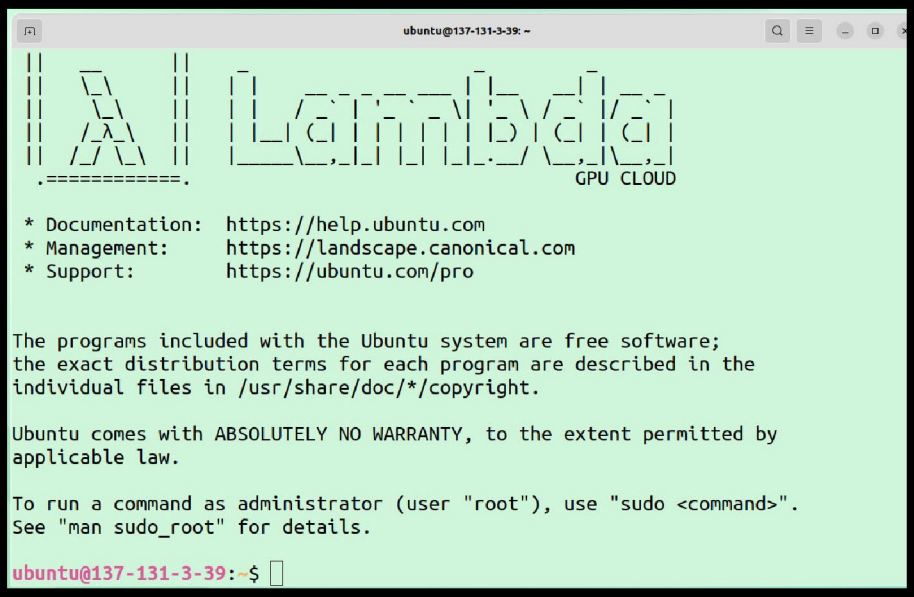

Figure 8: **Successful SSH Login Banner**: Lambda GPU Cloud welcome banner seen after a successful SSH connection.

Purpose: *Indicates a successful, secure login session to your cloud GPU instance.*

Windows Commands (Command Prompt/PowerShell)

Use your Windows terminal (CMD or PowerShell) to connect, using the full path to your key:

```
ssh -i "C:\Users\<USERNAME>\.ssh\<KEY_NAME>" ubuntu@<IP_ADDRESS>
```

Purpose: *Initiates the secure connection from Windows, specifying the local path to the private key.*

3 GPU and Isaac Sim Deployment

3.1 Step 5: Configure Docker and Verify GPU Access

These steps ensure your user has the necessary permissions to run Docker containers and that the NVIDIA GPU is ready.

1. **Verify Docker Installation:** Check the Docker version to ensure it is operational on the instance.

   ```
   docker -v
   ```

   Purpose: *Confirms that the Docker client and engine are installed and available on the system.*

2. **Add User to Docker Group:** Add your current user to the ‘docker‘ group to run Docker commands without ‘sudo‘.

   ```
   sudo usermod -aG docker $USER
   ```

   Purpose: *Grants the current user the necessary permissions to execute Docker commands, improving workflow.*

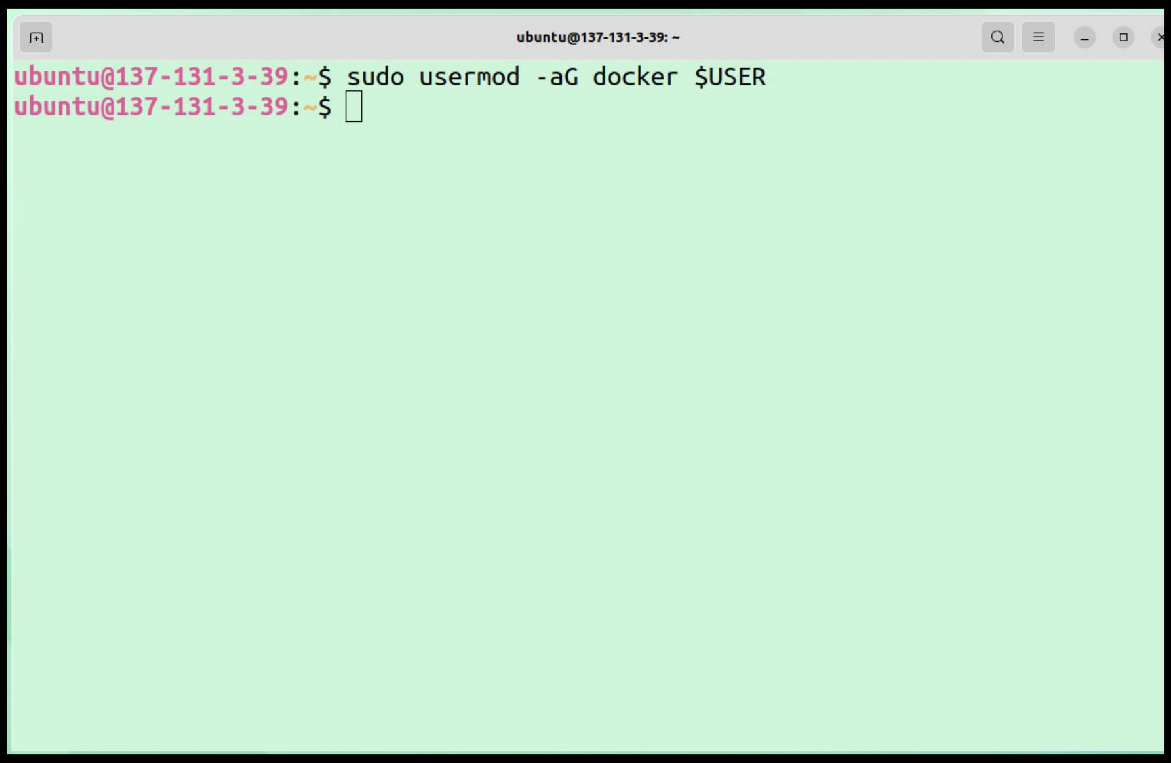

Figure 9: **Adding User to Docker Group**: Terminal output confirming the execution of the command to add the current user to the docker group.

3. **Apply Group Changes:** Activate the new group membership for the current session.

   ```
   newgrp docker
   ```

   Purpose: *Immediately applies the group changes without requiring you to log out and log back in.*

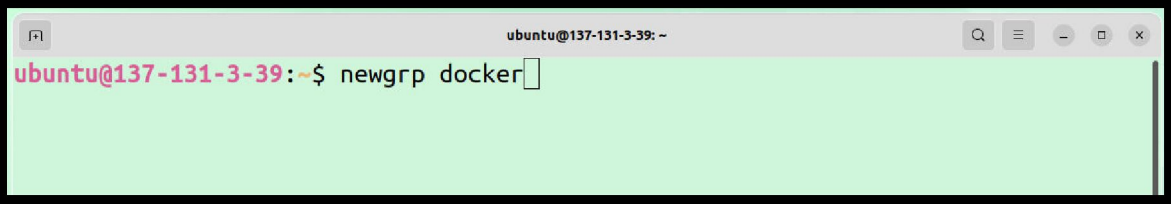

Figure 10: **Applying New Group Membership**: Terminal command to switch to the new group context, enabling Docker commands without sudo.

4. **Verify GPU Access:** Run the NVIDIA System Management Interface tool to ensure the GPU is detected.

   ```
   nvidia-smi
   ```

   Purpose: *Verifies that the NVIDIA drivers and the GPU hardware are correctly installed and recognized by the operating system.*

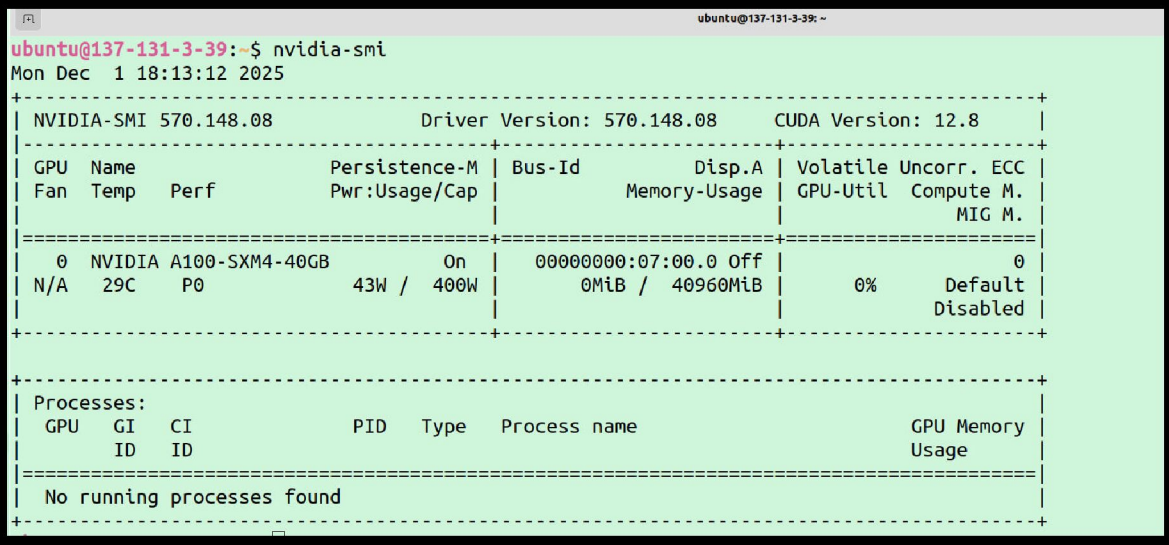

Figure 11: **NVIDIA-SMI Output**: Terminal output showing the detected NVIDIA A100 GPU with driver and CUDA version information.

3.2 Step 6: Prepare Filesystem and Pull Isaac Sim Container

5. **Create Local Cache Directory:** Create the directory structure needed for persistent Isaac Sim cache data.

   ```
   mkdir -p ~/docker/isaac-sim/cache/ov
   ```

   Purpose: *Sets up the necessary host directory to store Omniverse cache files, ensuring data persists between container runs.*

Figure 12: **Creating Docker Cache Directories**: Terminal commands for creating the host directories used by the Isaac Sim container for caching and persistent data.

6. **Pull Isaac Sim Container:** Use the `sudo docker pull` command to download the specified Isaac Sim version.

   ```
   sudo docker pull nvcr.io/nvidia/isaac-sim:4.2.0
   ```

```
ubuntu@129-146-178-160:~$ ubuntu@129-146-178-160:~$ sudo docker pull nvcr.io/nvidia/isaac-sim:5.1.0
5.1.0: Pulling from nvidia/isaac-sim
b08e2ff4391e: Pull complete
88a179c20f7b: Pull complete
1da7123d9f20: Pull complete
4f4fb700ef54: Pull complete
02c044224ab8: Pull complete
```

Figure 13: **Pulling Isaac Sim Docker Image**: Successful pull of the 'isaac-sim:5.1.0' container from the NGC registry.

Purpose: *Downloads the official Isaac Sim Docker image (version 4.2.0) containing the simulation environment, drivers, and necessary libraries.*

3.3 Step 7: Run Isaac Sim Container

This command runs the Docker container with all the necessary permissions and volume mounts required for Isaac Sim to operate correctly, including graphics, networking, and persistent storage.

```
docker run --name isaac-sim --entrypoint bash -it --runtime=nvidia --gpus all --rm --n
      -e "ACCEPT_EULA=Y" \
      -e "PRIVACY_CONSENT=Y" \
      -v ~/docker/isaac-sim/cache/kit:/isaac-sim/kit/cache:rw \
      -v ~/docker/isaac-sim/cache/ov:/root/.cache/ov:rw \
      -v ~/docker/isaac-sim/cache/pip:/root/.cache/pip:rw \
      -v ~/docker/isaac-sim/cache/glcache:/root/.cache/nvidia/GLCache:rw \
      -v ~/docker/isaac-sim/cache/computecache:/root/.nv/ComputeCache:rw \
      -v ~/docker/isaac-sim/logs:/root/.nvidia-omniverse/logs:rw \
      -v ~/docker/isaac-sim/data:/root/.local/share/ov/data:rw \
      -v ~/docker/isaac-sim/documents:/root/Documents:rw \
      nvcr.io/nvidia/isaac-sim:4.2.0
```

```
isaac-sim@137-131-3-39: ~
ubuntu@137-131-3-39:~$ docker run --name isaac-sim --entrypoint bash -it --gpus all -e "ACCEPT_EULA=Y" --rm --network=host \
    -e "PRIVACY_CONSENT=Y" \
    -v ~/docker/isaac-sim/cache/main:/isaac-sim/.cache:rw \
    -v ~/docker/isaac-sim/cache/computecache:/isaac-sim/.nv/ComputeCache:rw \
    -v ~/docker/isaac-sim/logs:/isaac-sim/.nvidia-omniverse/logs:rw \
    -v ~/docker/isaac-sim/config:/isaac-sim/.nvidia-omniverse/config:rw \
    -v ~/docker/isaac-sim/data:/isaac-sim/.local/share/ov/data:rw \
    -v ~/docker/isaac-sim/pkg:/isaac-sim/.local/share/ov/pkg:rw \
    -u 1234:1234 \
    nvcr.io/nvidia/isaac-sim:5.1.0
isaac-sim@137-131-3-39:~$
```

Figure 14: **Executing Docker Run Command**: The final, long command to launch the Isaac Sim container (version 5.1.0) with all required volume mounts and environment variables.

Purpose: *Launches the Isaac Sim container. Key flags ensure: **'–runtime=nvidia –gpus all'** grants full GPU access; **'–network=host'** enables seamless network integration; **'-it –entrypoint*

*bash‘** opens an interactive terminal inside the container; and the **‘-v‘** flags mount local directories for caching, logs, and persistent data.*