Setting up the BRAIN project on a fresh NVIDIA Jetson AGX/Nano

Alex Baldwin

August 2025

Abstract

This document covers installing the BRAIN vision compute stack on an NVIDIA embedded compute device. The proceedures outlined below have been tested to work on an NVIDIA Jetson AGX Orin and an NVIDIA Jetson Orin Nano.

Contents

1 Prerequisites

1.1 Essential hardware

When travelling with the Jetson, it's critical to be prepared with the following items for debugging:

- A laptop with a Debian based OS (This guide has been tested with Ubuntu 24.04 and Debian 13)
- USB A/C to USB Micro B cable (for serial debugging)
- USB A/C to USB C cable (USB 3.1 capable minimum) (for firmware flashing)
- >128GB Micro SD card
- SD card reader, if your laptop cannot read them directly
- USB C Power Delivery capable power supply
- USB Keyboard
- Ethernet Cable

1.2 Flashing the base OS image

The official NVIDIA images come with a full desktop environment and a lot of other bloat. Therefore, BRAIN is currently built for a host based on the ARM64 Tegra Ubuntu Server 22.04 release. To begin, download the image and extract it to your computer. Then use `dd` to write the image to your Micro SD card:

```
alewin@NOCS000000:~$ lsblk
NAME                      MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda                         8:0    1 238.8G  0 disk
sda1                        8:1    1 238.7G  0 part
nvme0n1                     259:0    0 476.9G  0 disk
nvme0n1p1               259:1    0   512M  0 part  /boot/efi
nvme0n1p2               259:2    0   488M  0 part  /boot
nvme0n1p3               259:3    0   476G  0 part
  nvme0n1p3_crypt       253:0    0 475.9G  0 crypt
    nocs000000--vg-root   253:1    0   475G  0 lvm   /
    nocs000000--vg-swap_1 253:2    0   980M  0 lvm   [SWAP]
alewin@NOCS000000:~$ sudo dd if=ubuntu-22.04-preinstalled-server-arm64+tegra-jetson
    .img of=/dev/sda bs=16M
[sudo] password for alewin:
323+1 records in
323+1 records out
5422220800 bytes (5.4 GB, 5.0 GiB) copied, 169.935 s, 31.9 MB/s
```

The current BRAIN stack is designed to run within docker, with host NVIDIA CUDA drivers of 12.6 or later. We target a minimum firmware version of 36.4, which is sometimes installed by default. You should check the firmware version on the UEFI BIOS screen upon first boot of your Jetson. If your firmware version is older than 36.4, you will need to follow the reflashing step detailed in subsection 4.1 before continuing.

The official NVIDIA images boot the kernel driectly using ExtLinux, but the Ubuntu Server release uses GRUB. To change this setting, you must hit *ESC* at startup to enter the UEFI BIOS settings, and navigate to the submenu *Device Manager ⇒NVIDIA Configuration ⇒L4T Configuration ⇒L4T Boot Mode*.

Side note specific to the Orin AGX

The AGX has an internal 32GB eMMC chip which is usually flashed with an old NVIDIA Ubuntu Desktop image. This can be useful to keep around, or you can re-flash it to the Ubuntu Server image as well. The AGX will want to boot from the eMMC by default, so you must hit *ESC* to enter the UEFI BIOS settings, and change the default boot device to UEFI SD card. It is in the submenu *Boot Maintenance Manager* ⇒*Boot Options* ⇒*Change Boot Order*.

It's not possible to install the entire BRAIN stack to the on-board storage due to space constraints, so it's reccomended to install everything to the SD card. It is possible to install the base OS on eMMC, and leave only the docker images on the SD card, but this is a more complicated install method, and is usually not worth it unless you're unable to source a >128GB Micro SD card.

1.3 Setting up the base OS image

By default, the Ubuntu Server image will attempt to mount a cloud-init config file. After about 1m30s it will abort, and create a default user with the username `ubuntu` and password `ubuntu`, and enable this account for SSH login. You will be required to change the password immediately. The fact that this image ships with cloud-init is notable, as it is actually possible to set up the BRAIN stack automatically on Jetson without a monitor at all. However, it's much easier to set up the device manually, and a monitor is a must-have for debugging when all else fails.

The Jetson will automatically connect to any LAN with DHCP on its primary Ethernet interface. You may want to use the arp-scan tool to find it on your LAN:

```
alewin@NOCS000000:~$ sudo arp-scan 10.42.0.0/24
[sudo] password for alewin:
Interface: enp0s31f6, type: EN10MB, MAC: 28:00:af:74:97:0c, IPv4: 10.42.0.1
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
10.42.0.49      3c:6d:66:XX:XX:XX       (Unknown)

1 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.989 seconds (128.71 hosts/sec). 1
    responded

alewin@NOCS000000:~$ ssh ubuntu@10.42.0.49
The authenticity of host '10.42.0.49 (10.42.0.49)' can't be established.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.42.0.49' (ED25519) to the list of known hosts.
ubuntu@10.42.0.49's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-1020-nvidia-tegra-igx aarch64)
[...]
Last login: Tue Nov 21 22:54:35 2023
ubuntu@brain0:~$
```

Start by creating a new user for deployment, adding them to `sudo` group:

```
ubuntu@ubuntu:~$ sudo adduser deploy
[sudo] password for ubuntu:
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for deploy
Enter the new value, or press ENTER for the default
        Full Name []: Deployer McDeployface
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

```
ubuntu@ubuntu:~$ sudo usermod -aG sudo deploy
```

NOTE: We use the default password `brain!` for non-networked deployments for simplicity. You MUST change this to a secure random value from `pwgen 16` or similar before deploying to a networked environment.

You should then set the hostname for the device. In order for this to fully take affect, you should reboot the device.

```
ubuntu@ubuntu:~$ sudo hostnamectl hostname brain0
ubuntu@ubuntu:~$ sudo reboot
```

After the reboot, login with the deploy account. Now's a good time to remove `unattended-upgrades` for production systems to avoid automatic updates mid-deployment, and `cloud-init` as it's not neccesary, and can cause problems in the boot process.

```
deploy@brain0:~$ sudo apt update
Hit:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease
Get:2 http://ports.ubuntu.com/ubuntu-ports jammy-updates InRelease [128 kB]
[...]
Fetched 18.4 MB in 4s (4728 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
178 packages can be upgraded. Run 'apt list --upgradable' to see them.
deploy@brain0:~$ sudo apt remove unattended-upgrades cloud-init
[...]
deploy@brain0:~$ sudo apt upgrade
[...]
deploy@brain0:~$
```

1.4 Installing NVIDIA container runtime and other dependancies

```
sudo add-apt-repository ppa:ubuntu-tegra/updates
sudo apt update
sudo apt install -y nvidia-tegra-drivers-36
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
curl -s -L https://repo.download.nvidia.com/jetson/jetson-ota-public.asc | sudo tee /etc/apt/trusted.gpg.d/jetson-ota-public.asc
sudo chmod 644 /etc/apt/trusted.gpg.d/jetson-ota-public.asc
echo "deb https://repo.download.nvidia.com/jetson/common r36.4 main" | sudo tee -a /etc/apt/sources.list.d/nvidia-l4t-apt-source.list
echo "deb https://repo.download.nvidia.com/jetson/t234 r36.4 main" | sudo tee -a /etc/apt/sources.list.d/nvidia-l4t-apt-source.list
sudo apt update
sudo apt install docker.io docker-compose docker-buildx docker-compose-v2 nvidia-container-toolkit nvidia-container-toolkit-base
sudo usermod -aG render,docker deploy
```

Now reboot to make sure all drivers are correctly loaded. Continue to configure the newly installed NVIDIA Container Toolkit:

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo service docker restart
```

1.5 Testing host using the NOC L4T base image

The NOC L4T image is a pre-built docker image that contains PyTorch, torchvision and ultralytics (for using YOLO models). It also includes a self-test script as a placeholder to be overwritten with your intended application. This is useful for testing your configuration is correct. Your output should look similar to below:

```
deploy@brain:~/git/brain/vision$ docker run --runtime nvidia -it docker-repo.bodc.me/oceaninfo/l4t-base:j62-r36.4-2
NOC L4T (Linux 4 Tegra) base image test script

Y Standard dependencies loaded
Y PyTorch CUDA support
    PyTorch Version: 2.8.0
    CUDA Version: 12.6
    CUDA Device: Orin
Y Torchvision loaded
Y OpenCV loaded
Y Ultralytics YOLO loaded

If everything comes back without error, you have configured your Jetson correctly!
This is a base container, you should extend from it with your own application.
```

If you don't have access to the BODC repo, but have the l4t-base image, load it as follows before continuing:

```
deploy@brain:~$ docker load --input l4t-base-j62-r36.4-2.tar
deploy@brain:~$ docker image tag 67e005c29188 docker-repo.bodc.me/oceaninfo/l4t-base:j62-r36.4-2
```

NOTE: For older versions of BRAIN, it was neccesary to set up API keys for NVIDIA's container repository. You can do this by signing up for an NVIDIA developer account, and proceeding to https://org.ngc.nvidia.com/setup/api-keys. Supply the bearer token as follows:

```
deploy@brain:~/git/brain$ docker login nvcr.io
Username: $oauthtoken
Password: <Your Key>
```

2 Installing the BRAIN stack

3 Configuring the BRAIN stack

4 Debugging

4.1 Reflashing firmware

With the software stack on the NVIDIA Jetsons being a little unstable, it's entirely possible to break the firmware from a simple software upgrade. There are a few different ways to reflash firmware on Jetson devices, but the most reliable is to use the `flash.sh` tool supplied by Cannonical. NVIDIA's official instructions are to use the NVIDIA sdk-manager application, but I have found this to be extremely unreliable and time consuming.

Firmware for all Orin devices, including flash.sh can be found at https://developer.nvidia.com/-downloads/embedded/l4t/r36_release_v4.3/release/Jetson_Linux_r36.4.3_aarch64.tbz2. The instructions below are derived from instructions found on Ubuntu's NVIDIA Jetson download page.

Side note specific to the Orin Nano

One alternative method is to first boot the JetPack 5.1.3 SD card image (https://developer.nvidia.com/-downloads/embedded/l4t/r35_release_v5.0/jp513-orin-nano-sd-card-image.zip) and to use apt to update the firmware. This has had mixed results in the past and is not reccomended.