ML Server SSOT

Phase 1: OS Installation & Base Configuration:

Step 1: Download and Prepare Ubuntu Desktop ISO

1. From a jump host or internet-connected machine:
    - Go to: https://ubuntu.com/download/desktop
    - Choose Ubuntu 22.04.4 LTS (Desktop)
2. Use a tool like:
    - Rufus (Windows)
    - BalenaEtcher (cross-platform)
    - Startup Disk Creator (Linux)
      … to flash ISO to a USB stick.

Step 2 : Boot the Server and Install Ubuntu:

1. Plug in the bootable USB into the HPE ProLiant XL645d server.
2. Boot into BIOS/UEFI:
    - Usually F9, F10, or Esc.
    - Set USB as first boot device.
3. Boot into installer and follow prompts:
    - Select language, region.
    - Configure user credentials, hostname, time zone
    - Partition disks: Use entire disk (unless RAID/manual needed).
    - Install 3rd party drivers when prompted (will help with GPU compatibility).
4. Complete installation and reboot.

Step 3 : Post-Install Initial Config:

1. Log in to Ubuntu Desktop with created user.
2. Enable SSH:

```
sudo apt update
sudo apt install openssh-server
sudo systemctl enable ssh
sudo systemctl start ssh
```

3. Check IP address:

```
ip a
```

Note down for remote access.

Step 4 : Jump Host SSH Setup:

If your jump host is allowed to access the server:

- From jump host (replace IP):

```
ssh username@<ubuntu-server-ip>
```

- Optionally, setup key-based login:

```
ssh-keygen
ssh-copy-id username@<ubuntu-server-ip>
```

Step 5: System Baseline Config :

Run these:

```
sudo apt update && sudo apt upgrade -y
sudo timedatectl set-timezone Asia/Kolkata
sudo hostnamectl set-hostname deepstack-server
```

Install basic tools :

```
sudo apt install -y vim curl wget net-tools git unzip
```

Step 6 : Optional Network Tweaks (if no DHCP):

If your server doesn't get an IP via DHCP, configure static IP:

```
sudo nano /etc/netplan/01-network-manager-all.yaml
```

Example Config :

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    eno1:
      dhcp4: no
      addresses: [192.168.1.50/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

Apply:

```
sudo netplan apply
```

Phase 2: NVIDIA GPU Driver & MIG Configuration:

Goal :

- Install compatible NVIDIA driver (v575.51.03 for CUDA 12.9)
- Install CUDA toolkit (offline)
- Enable and configure MIG on A100 GPUs
- Store it all in the SSOT folder, so it’s fully executable from your jump host.

Step 1: Identify Driver & CUDA Version:

- GPU: NVIDIA A100-SXM4-80GB
- CUDA Compatible Driver: ≥ 470.x (you're using 575.51.03)
- CUDA Toolkit: 12.9
- OS: Ubuntu Desktop 22.04

Step 2 : Download Driver Installer :

A. Go to NVIDIA Driver Download
   - Product Type: Data Center / Tesla
   - Product Series: A100
   - Product: A100-SXM4-80GB
   - OS: Linux 64-bit (Ubuntu)
   - Download `.run` file — e.g., `NVIDIA-Linux-x86_64-575.51.03.run`

B. `Save to your SSOT/drivers/ folder.`

Step 3 : Download CUDA Toolkit Offline Installer:

A. Go to: CUDA Toolkit 12.3+ Archive
   Choose :
   - Version: CUDA 12.9
   - OS: Linux → x86_64 → Ubuntu → 22.04

   Download :
   - `local_installers/cuda_12.3.0_*.run` OR `.deb` files
   - Also grab the matching cudnn `.tar.xz` for 12.3+

B. Place all in `SSOT/cuda/`
   `CUDA .run file is easier for fully offline install.`

Step 4 : Prepare Driver Install Script:

Inside SSOT/scripts/driver_install.sh

```
#!/bin/bash

echo "[INFO] Stopping graphical target..."
sudo systemctl isolate multi-user.target
sudo systemctl stop gdm3

echo "[INFO] Making driver installer executable..."
chmod +x ../drivers/NVIDIA-Linux-*.run
```

```
echo "[INFO] Installing NVIDIA driver silently..."
sudo ../drivers/NVIDIA-Linux-*.run --silent --dkms --disable-nouveau

echo "[INFO] Driver install complete."
```

Step 5 : Prepare CUDA Install Script:

Inside SSOT/scripts/cuda_install.sh:

```
#!/bin/bash

echo "[INFO] Installing CUDA toolkit..."
chmod +x ../cuda/cuda_*.run
sudo ../cuda/cuda_*.run --silent --toolkit --samples --override

echo "[INFO] Exporting environment variables..."
echo 'export PATH=/usr/local/cuda/bin:$PATH' >> ~/.bashrc
echo 'export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH' >>
~/.bashrc
source ~/.bashrc
```

Step 6: MIG Enable Script :
Create SSOT/scripts/enable_mig.sh:

```
#!/bin/bash

for gpu in {0..7}; do
  echo "[INFO] Enabling MIG on GPU $gpu..."
  sudo nvidia-smi -i $gpu -mig 1
done

echo "[INFO] Reboot required for MIG changes to take effect."
```

You'll run this once and then reboot manually.

Step 7: MIG Profile Creation Script :

Create SSOT/scripts/create_mig_profiles.sh:

```
#!/bin/bash

echo "[INFO] Creating MIG instances..."

# Example: 7 x 10GB profiles on GPU 0
sudo nvidia-smi mig -cgi 19,19,19,19,19,19,19 -gi 0
```

We can later generalize this using a config `.txt` file from customer input.

SSOT Folder Structure Summary

```
SSOT/
├── drivers/
│   └── NVIDIA-Linux-x86_64-575.51.03.run
├── cuda/
│   ├── cuda_12.3.0_linux.run
│   └── cudnn-linux-x86_64-*.tar.xz
├── scripts/
│   ├── driver_install.sh
│   ├── cuda_install.sh
│   ├── enable_mig.sh
│   └── create_mig_profiles.sh
```

Phase 2 Verification Checklist:

- Ubuntu boots in GUI
- Driver installed silently (no GUI required)
- CUDA installed and tested
- MIG mode enabled & verified using `nvidia-smi -L`

- Profiles created using script

Phase 3: Anaconda Environment + Python AI Libraries:

Objective:

- Install Anaconda offline on host machine (Ubuntu Desktop 22.04).
- Set up Python environments for GPU-accelerated AI libraries.
- Include key packages: `cv2 (OpenCV with CUDA)`, `torch`, `tensorflow`, `jax`, `transformers`, `xgboost`, `nltk`, `theano`, `wandb`, DL4J, etc.
- Same setup also to be reproducible inside Docker, backed by same CUDA compatibility (12.3 / 12.9).

Step 1 : Download Anaconda Offline Installer :

- Visit: https://repo.anaconda.com/archive
- Download : Anaconda3-2024.02-1-Linux-x86_64.sh
- Save to : SSOT/anaconda/Anaconda3-2024.02-1-Linux-x86_64.sh
- This version includes Python 3.10 — compatible with CUDA 12.3.

Step 2: Install Anaconda (Offline Script) :

Script: `SSOT/scripts/anaconda_install.sh` :

```
#!/bin/bash

echo "[INFO] Installing Anaconda silently..."
chmod +x ../anaconda/Anaconda3-2024.02-1-Linux-x86_64.sh
bash ../anaconda/Anaconda3-2024.02-1-Linux-x86_64.sh -b -p
$HOME/anaconda3

echo 'export PATH="$HOME/anaconda3/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc

echo "[INFO] Anaconda installed successfully."
```

Confirm with :

```
conda --version
```

Step 3: Create Conda Environment (with CUDA packages) :

Script: SSOT/scripts/create_ai_env.sh:

```
#!/bin/bash

echo "[INFO] Creating AI conda environment..."

conda create -y -n ai_env python=3.10
conda activate ai_env

echo "[INFO] Installing core libraries..."
pip install \
    torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121 \
    tensorflow \
    jax[cuda12_pip] -f
https://storage.googleapis.com/jax-releases/jax_cuda_releases.html \
    transformers \
    opencv-python-headless \
    wandb \
    nltk \
    xgboost \
    theano \
    chromadb \
    openai \
    fastapi uvicorn
```

You'll need to:

- Place .whl files if internet isn't available and modify pip install accordingly.
- Download prebuilt .whl files ahead of time (optional but recommended).

Step 4: Reuse Same Setup Inside Docker :

Create SSOT/docker/ai-env.Dockerfile :

```
FROM nvidia/cuda:12.3.2-runtime-ubuntu22.04

ENV DEBIAN_FRONTEND=noninteractive

# Install Python and pip
RUN apt-get update && apt-get install -y python3 python3-pip
python3-venv

# Optional: Copy downloaded .whl files into image

# Install Python packages (ensure versions match host)
RUN pip install --upgrade pip && pip install \
    torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121 \
    tensorflow \
    jax[cuda12_pip] -f
https://storage.googleapis.com/jax-releases/jax_cuda_releases.html \
    transformers opencv-python-headless wandb nltk xgboost theano
chromadb openai fastapi uvicorn

CMD [ "python3" ]
```

Build Locally:

```
docker build -t ai-env -f ai-env.Dockerfile .
```

Step 5 : Conda Environment Export :

After testing, you can export the environment:

```
conda activate ai_env
conda env export > SSOT/envs/ai_env.yml
```

This allows reproducible environment creation with:

```
conda env create -f ai_env.yml
```

Summary of SSOT Additions:

```
SSOT/
├── anaconda/
│   └── Anaconda3-2024.02-1-Linux-x86_64.sh
├── scripts/
│   ├── anaconda_install.sh
│   ├── create_ai_env.sh
├── envs/
│   └── ai_env.yml
├── docker/
│   └── ai-env.Dockerfile
```

Phase 3 Verification Checklist

- Anaconda installed offline
- ai_env created with required libraries
- GPU libraries (torch, tensorflow, jax) support CUDA 12.3+
- Docker image with identical setup built

Phase 4: GPU-Aware Tools and Specialized AI/ML Frameworks:

Step 1: NVIDIA DeepStream SDK :

- What is it?
  DeepStream is NVIDIA's AI streaming analytics toolkit ,ideal for vision AI workloads using real-time inference, video decoding, and neural networks.

- Why needed?
  Your setup may run video inference (CV2, YOLO, object detection pipelines). DeepStream accelerates this with TensorRT and hardware-level optimizations.

- Version Selection
  We'll use: DeepStream 6.4 – supports CUDA 12.3 and Ubuntu 22.04

- Offline Setup

  1. Download the .deb file on internet-connected machine:
     - Go to: https://developer.nvidia.com/deepstream-sdk
     - Select: DeepStream 6.4 | Ubuntu 22.04 | deb (x86_64)

  2. Place in SSOT:
     SSOT/deepstream/deepstream-6.4_6.4.0-1_amd64.deb

  3. Script: SSOT/scripts/install_deepstream.sh :

```
#!/bin/bash
echo "[INFO] Installing DeepStream SDK..."

sudo dpkg -i ../deepstream/deepstream-6.4_6.4.0-1_amd64.deb
```

```
sudo apt --fix-broken install -y  # Resolves dependencies
```

After install :

```
deepstream-app --version
```

Step 2: DCGM & DCGM Exporter (GPU Metrics + Prometheus Export)

- What is it?

  DCGM = Data Center GPU Manager
  DCGM Exporter = Prometheus-compatible exporter for GPU health, power, and utilization.

- Purpose in your context:

  Used for resource tracking, GPU monitoring, and visual dashboards (if added later with Grafana).

- DCGM Setup (Offline)
  1. Download .deb from NVIDIA
     - Go to: https://developer.nvidia.com/dcgm
     - Pick version supporting CUDA 12.3 and Ubuntu 22.04
  2. Place it:
     SSOT/dcgm/datacenter-gpu-manager_3.2.4_amd64.deb
  3. Script: SSOT/scripts/install_dcgm.sh :

```
#!/bin/bash
echo "[INFO] Installing NVIDIA DCGM..."

sudo dpkg -i ../dcgm/datacenter-gpu-manager_3.2.4_amd64.deb
sudo systemctl enable --now dcgm
```

- DCGM Exporter (for metrics):
  1. Download the .tar.gz release from: https://github.com/NVIDIA/dcgm-exporter/releases
  2. Place in: SSOT/dcgm-exporter/dcgm-exporter.tar.gz
  3. Script: SSOT/scripts/install_dcgm_exporter.sh:

```
#!/bin/bash

echo "[INFO] Extracting DCGM Exporter..."
tar -xzf ../dcgm-exporter/dcgm-exporter.tar.gz -C /opt/
chmod +x /opt/dcgm-exporter/dcgm-exporter

echo "[INFO] Add to systemd or run manually:"
echo "Run with: /opt/dcgm-exporter/dcgm-exporter"
```

Step 3: RAPIDS AI (cuDF, cuML, cuGraph):

- What is it?
  - NVIDIA's GPU-accelerated data science suite. Includes
    cuDF → GPU-powered pandas

    cuML → GPU-based ML like XGBoost
    cuGraph → Graph analytics

- Requirements:

  - Python 3.10

  - CUDA 12.2+ (compatible with 12.3)

- Offline Strategy:

```
conda create -n rapids python=3.10

conda activate rapids
```

```
conda install -c rapidsai -c nvidia -c conda-forge \
    cudf=24.04 cuml=24.04 cugraph=24.04 \
    python=3.10 cudatoolkit=12.2
```

2. Export it :

```
conda env export > SSOT/envs/rapids.yml
```

3. On target server:

```
conda env create -f SSOT/envs/rapids.yml
```

Step 4: Deep Cognition Studio (DCS):

- What is it?
    - GUI-based deep learning IDE; not CLI friendly.
- How to Handle in Offline + Jump Host Setup
    - Not ideal for non-GUI server
    - Skip or notify customer it requires full GUI, VNC and external download

Step 5 : DL4J, OpenNN, Caffe2:

These are:

- DL4J: Java-based ML – use only if customer mandates
- OpenNN: C++ based – same logic
- Caffe2: Merged into PyTorch → Deprecated

Phase 4 Directory Structure

```
/SSOT/
├── deepstream/
│   └── deepstream-6.4_6.4.0-1_amd64.deb
├── dcgm/
│   └── datacenter-gpu-manager_3.2.4_amd64.deb
├── dcgm-exporter/
│   └── dcgm-exporter.tar.gz
├── envs/
│   └── rapids.yml
├── scripts/
│   ├── install_deepstream.sh
│   ├── install_dcgm.sh
│   └── install_dcgm_exporter.sh
```

Phase 5: Deep Learning & AI Stack Installation

- Goal:
  Install all required frameworks and libraries offline using .whl, .tar.gz, and .deb packages via SSOT.

- Key Components:

Tool/Library	GPU-	Install Method	Environment
Anaconda	✔	`.sh` installer	Host
OpenCV (cv2) w/ CUDA	✔	Prebuilt `.whl`	Conda/Docker
PyTorch + torchvision	✔	Prebuilt `.whl`	Conda/Docker
TensorFlow (CUDA)	✔	Prebuilt `.whl`	Conda/Docker
DeepStream SDK	✔	`.deb` or tarball	Host/Docker
DCGM Exporter	✔	`.deb`	Host
RAPIDS	✔	`.whl` or conda	Conda/Docker
OpenNN	✘	Source `.tar.gz`	Host
Theano, XGBoost, NLTK	✔	`pip install`	Conda/Docker
CHROMA (ChromaDB)	?	`pip`	Conda/Docker
JAX (CUDA)	✔	Prebuilt `.whl`	Conda/Docker
Transformers	✔	`pip`	Conda/Docker
W\&B (Weights & Biases)	✔	`pip`	Conda/Docker

```
| DL4J (Java)              | ✔️    | `.jar` or Maven    | Host
|
| FastAPI + Uvicorn        | ✔️    | `pip`              |
Conda/Docker |
```

Prepare the SSOT Structure for Phase 5 :

Folder Layout:

```
SSOT/
├── conda/
│   └── Anaconda3-2024.02-Linux-x86_64.sh
├── pip_wheels/
│   ├── torch-2.5.1+cu121.whl
│   ├── torchvision-0.20.1+cu121.whl
│   ├── torchaudio-2.5.1+cu121.whl
│   ├── tensorflow-*.whl
│   ├── jax_cuda_*.whl
│   ├── opencv_python_cuda-*.whl
│   ├── transformers-*.whl
│   ├── wandb-*.whl
│   ├── nltk-*.whl
│   ├── xgboost-*.whl
│   ├── theano-*.whl
│   └── chromadb-*.whl
├── deepstream/
│   └── deepstream_sdk_*.deb or tar
├── dcgm/
│   └── datacenter-gpu-manager-4-cuda12_*.deb
├── opennn/
│   └── opennn-*.tar.gz
├── dl4j/
│   └── deeplearning4j-core-*.jar
├── scripts/
│   └── install_all.sh
```

Sample `install_all.sh` Script :

```
#!/bin/bash
set -e

echo "[*] Installing Anaconda..."
bash ../conda/Anaconda3-2024.02-Linux-x86_64.sh -b -p $HOME/anaconda3
eval "$($HOME/anaconda3/bin/conda shell.bash hook)"
conda init

echo "[*] Creating ML conda environment..."
conda create -y -n ai-stack python=3.10
conda activate ai-stack

echo "[*] Installing Python packages from wheels..."
pip install --no-index --find-links=../pip_wheels torch torchvision
torchaudio tensorflow jax[cuda] \
  opencv-python-headless transformers wandb nltk xgboost theano
chromadb

echo "[*] Installing DeepStream..."
dpkg -i ../deepstream/deepstream_sdk_*.deb

echo "[*] Installing DCGM..."
dpkg -i ../dcgm/datacenter-gpu-manager-4-cuda12_*.deb
systemctl enable --now nvidia-dcgm

echo "[*] Extracting OpenNN..."
tar -xzf ../opennn/opennn-*.tar.gz -C /opt/

echo "[*] Installing DL4J Java Library (for later use in apps)..."
cp ../dl4j/deeplearning4j-core-*.jar /opt/dl4j/

echo "[✓] Phase 5 complete. AI stack is installed."
```