DEEP LEARNING SERVER

Admin Manual

SSN College of Engineering, Chennai

From

Tekcogent Solutions Private Limited Chennai

08/05/2025

Index

Prepared by
Ragul A S

Approved by
Nataraj N

About Deep Learning Server platform

We are happy to inform you that recently we had implemented a high end NVIDIA GPU platform with capability to support development and execution of Artificial Intelligence based research projects and applications. This platform has good configuration that supports large and computationally intensive operations.

Hardware Configuration:

Motherboard & CPU:

- Supermicro Dual Socket P (LGA 3647) motherboard
- Supports Intel Xeon Scalable Processors
- 2x Intel Xeon Skylake 5118:
    - Each: 12 cores / 24 threads
    - Base Clock: 2.3 GHz
    - Cache: 16.5 MB L3

Memory: 128GB DDR4-2666 ECC REG DIMM (Error-Correcting Code, Registered, suitable for server-grade workloads)

Storage: 2x 2TB SATA 7.2K RPM 2.5" HDDs — Enterprise-class drives

GPU: 2x NVIDIA Tesla V100 32GB (SMX2):

- High-end AI/HPC GPUs
- Uses CoWoS HBM2 memory
- NVLink enabled (for high-bandwidth GPU interconnect)
- Often used in deep learning, scientific simulations, and research

NVIDIA V100 Tensor Core GPU

The server includes **2× NVIDIA Tesla V100 32GB (SMX2)** GPUs, which are **data center-class GPUs** specifically designed for high-performance computing (HPC), artificial intelligence (AI), and deep learning. Here's a detailed breakdown:

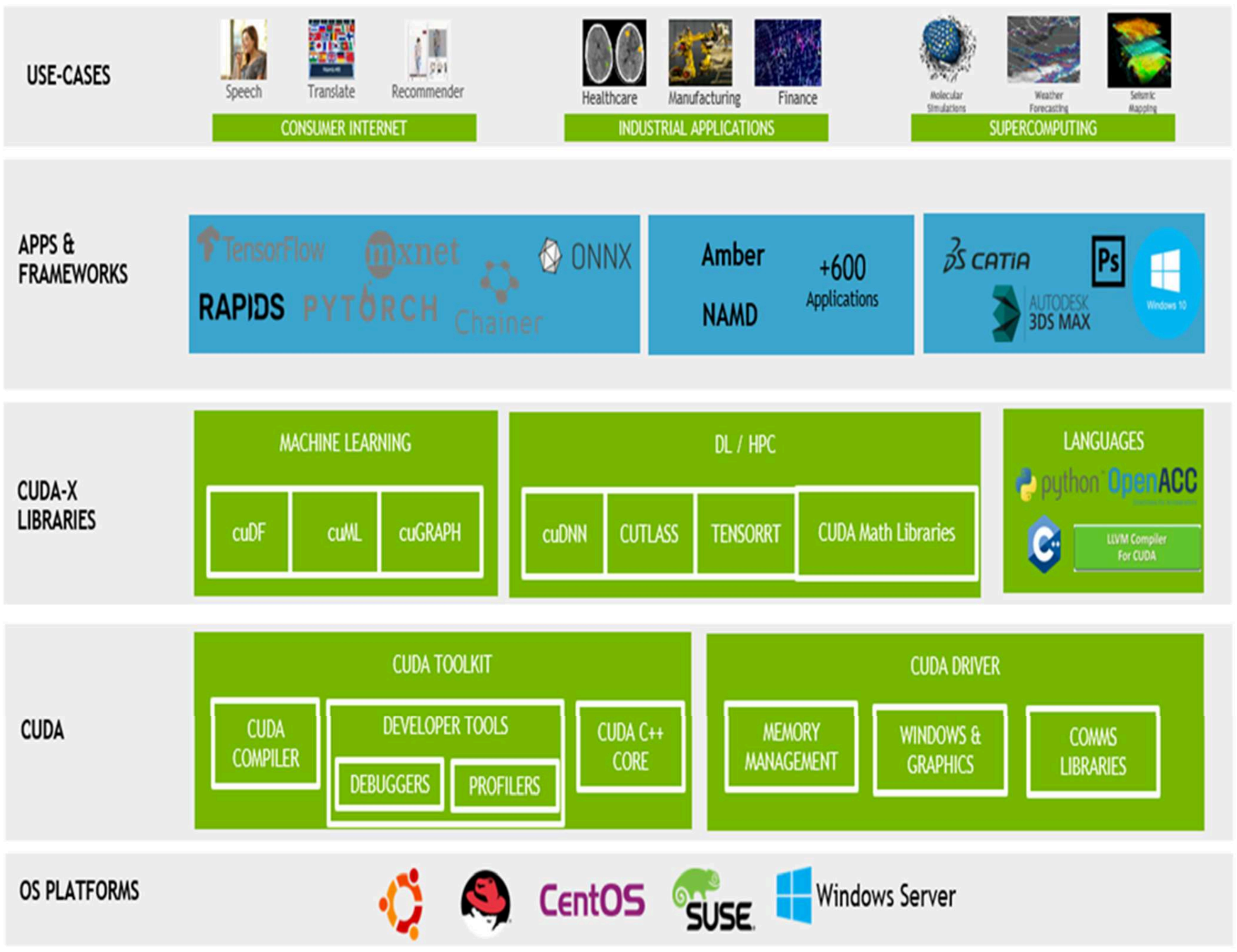
- **Model**: Tesla V100 SMX2
- **Architecture**: NVIDIA Volta
- **Interface**: SXM2 (NVLink)
- **Memory Type**: HBM2 (High Bandwidth Memory 2)
- **Memory Capacity**: 2*32 GB per GPU
- **NVLink Support**: Yes (Up to 300 GB/s GPU-to-GPU bandwidth)
- **CUDA Cores** : 5120
- **Tensor Cores :** 640

Tensor Cores: Greatly accelerate deep learning training (especially with FP16/TF32 mixed precision).

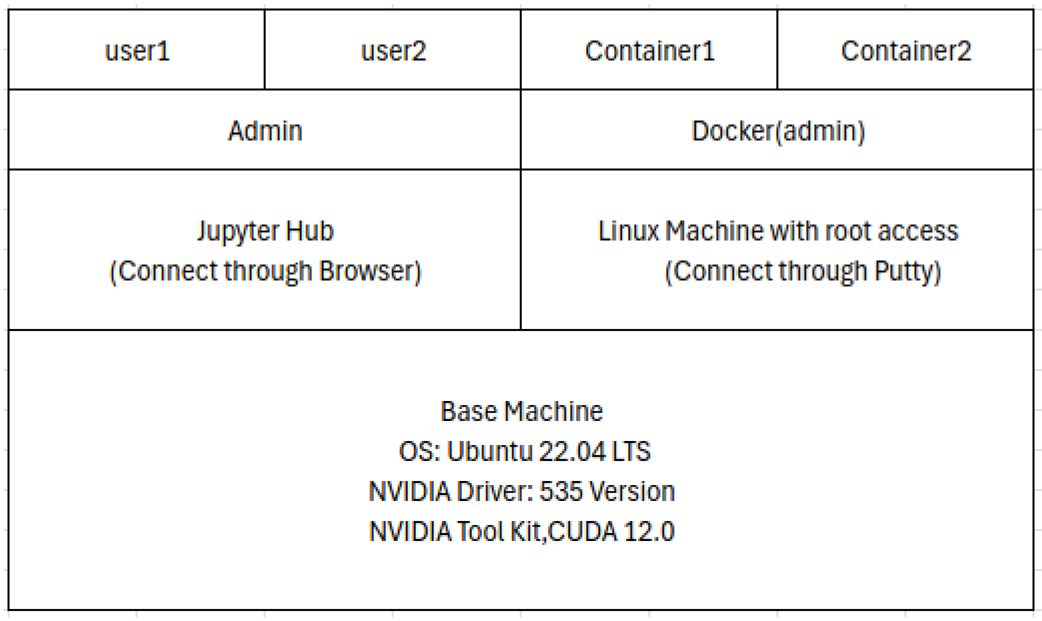
Large Memory: 32GB HBM2 is excellent for large batch sizes, 3D data, video analysis, or transformer models.

NVLink: Enables **fast GPU-GPU communication** (much faster than PCIe), which is essential for multi-GPU training.

Software stack layout – High level



Implemented Software Stack Architecture:



Jupyter Hub access:

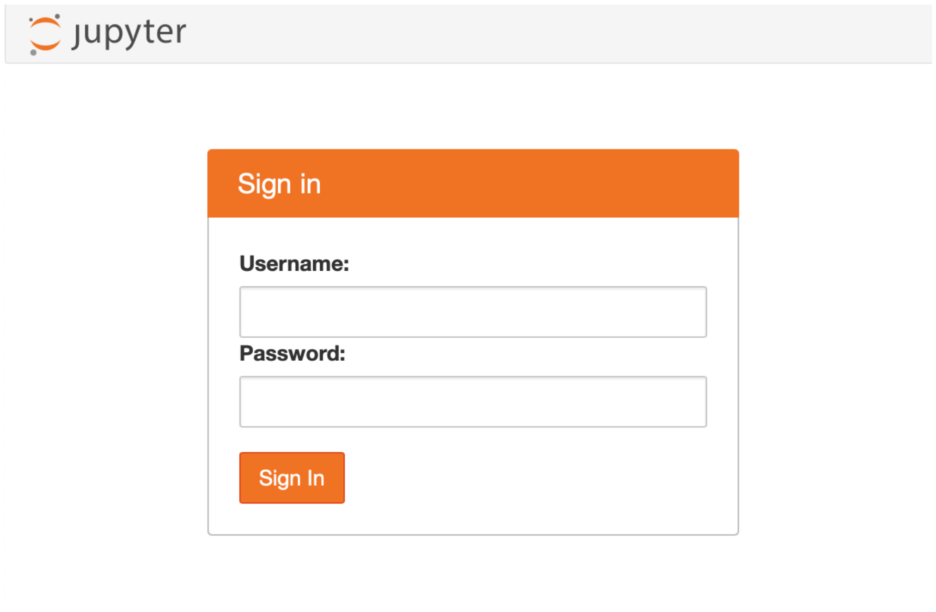
10.101.1.101 (Through Browser)

Credential:

Username:****

Password:****

1. Login to Machine with Username and Password.



Container Implementation and Access

Scenario-1

Ubuntu Container Implementation and User access

Stage-1 One Time Setup

Create a 20GB Docker volume with GPU support and appropriate access permissions for the container.
Create a 20GB Volume for Docker from the Partition
We will allocate 20GB as a file-based volume using a loop device.
1) Create a 20GB File

```
# sudo fallocate -l 20G /home/docker-volume-20g.img
```

- fallocate: Allocates space on the filesystem.
- -l 20G: Specifies the size: 20 gigabytes.
- /home/docker-volume-20g.img: The path where the virtual disk file will be stored.

2) Format the file as an ext4 filesystem

```
# sudo mkfs.ext4 /home/docker-volume-20g.img
```

- mkfs.ext4: Makes an ext4 filesystem on the file.

3) Mount the Volume for Docker
a) Create a Mount Point

```
# sudo mkdir -p /var/lib/docker-volume-20g
```

- Create a directory that will act as a mount point.
- -p: Creates the parent directory if it doesn’t exist.

b) Mount the File as a Loop Device

```
# sudo mount -o loop /home/docker-volume-20g.img /var/lib/docker-volume-20g
```

- Mounts the .img file as a loop device, simulating a physical disk.
- The contents of /var/lib/docker-volume-20g now actually reside in the .img file.

Make the Mount Persistent After Reboot

c) To ensure the volume mounts automatically:
Edit the /etc/fstab File

```
# sudo nano /etc/fstab
```

Add This Line at the End

```
# /home/docker-volume-20g.img /var/lib/docker-volume-20g ext4 loop 0 0
```

- This line is meant to be added to /etc/fstab so that the virtual disk auto-mounts on boot.
- Ensures the .img mounts at /var/lib/docker-volume-20g every time the system starts.

d) Apply the Changes

```
# sudo mount -a
```

- Now, the volume will stay mounted even after a reboot.

e) Create a Docker Volume Using the Mounted Path

```
# docker volume create \
  --driver local \
  --opt type=none \
  --opt device=/var/lib/docker-volume-20g \
  --opt o=bind \
  docker_volume_20g
```

- Creates a named Docker volume (docker_volume_20g) that maps directly to the mounted .img file.
- --opt type=none: Tells Docker not to use any special file system type.
- --opt device=...: The path on the host to bind.
- --opt o=bind: Makes Docker bind mount to that folder.

Stage-2 On going Admin Job

1) Pull the docker images

```
# docker pull ubuntu
```

2) Create container with GPU & SSH Port

```
# docker run -d --name <username> --gpus all -v my_docker_volume:/data -p
<0000:22> <docker images> sleep infinity
```

- docker run -d \ : Run container in detached mode
- --name <username>\ : Assign a name to the container
- --gpus all \ : Enable GPU access for the container
- -v my_docker_volume:/data \ : Mount the Docker volume to /data inside the container
- -p 0000:22 \ : Map port 0000 on the host to port 22 inside the container
- ubuntu \ : Use the Ubuntu image
- sleep infinity : Keep the container running indefinitely

ex,

```
# docker run -d --name ssngpu1 --gpus all -v my_docker_volume:/data -p 1669:22 ubuntu
sleep infinity
```

3) Accessing the running container:

```
# docker exec -it <username> /bin/bash
```

- docker exec: Runs a command inside an existing container.
- -it: Provides an interactive terminal.
- <username>: Refers to the container name set earlier.
- /bin/bash: Opens a shell session within the container.

ex,

```
# docker exec -it ssngpu1 /bin/bash
```

Inside the Container: Setup and Configuration

4) Updating the package list:

```
# apt update
```

- Updates the package list to ensure the latest versions are available for installation.

5) To Check the gpu usage/utilization

```
# nvidia-smi
```

6) Installing the SSH server:

```
# apt install ssh
```

- Installs the OpenSSH server to enable remote access to the container.

7) Installing the Nano text editor:

```
# apt install nano
```

- Installs the nano text editor, which allows you to modify configuration files easily.

8) Configuring SSH settings:

```
# nano /etc/ssh/sshd_config
```

- Opens the SSH server configuration file.
- Here, you can set parameters such as password authentication.
- Changes to make:
- Ensure the following line is set (if not, modify it):
- PermitRootLogin yes
- PasswordAuthentication yes
- Press Ctrl + X, then Y, and Enter to save the changes.

9)Restarting the SSH service:

```
# service ssh restart
```

- Restarts the SSH server to apply configuration changes.

10) Setting the root password:

```
# passwd root
```

- Prompts you to set a password for the root user, which will be used for SSH login.

11) Exiting the container:

```
# Exit
```

- Exits the current shell session inside the container.

Logging into the container using PuTTY(From End User Side)

Now, you can log into the running container from an external SSH client using:

- Hostname:10.101.1.101
- Port: The value you set (e.g., 0000)
- Username: root
- Password: ssn@123

If using PuTTY:

- Open PuTTY.
- Enter localhost in the "10.101.1.101" field.
- Set the port number (0000).
- Click "Open," then log in with username and password.

Scenario-2
Deepstream Implementation and container access

1) Pull Docker Images

```
# docker pull nvcr.io/nvidia/deepstream
```

2) Create container with GPU & SSH Port

```
# docker run -d --name <username> --gpus all -v my_docker_volume:/data -p
<0000:22> <docker images> sleep infinity
```

- docker run -d \ : Run container in detached mode
- --name <username>\ : Assign a name to the container
- --gpus all \ : Enable GPU access for the container
- -v my_docker_volume:/data \ : Mount the Docker volume to /data inside the container
- -p 0000:22 \ : Map port 0000 on the host to port 22 inside the container
- ubuntu \ : Use the Ubuntu image
- sleep infinity : Keep the container running indefinitely

ex,

```
# docker run -d --name deepstream1 --gpus all -v docker_volume_20g:/data -p 1671:22
nvcr.io/nvidia/deepstream:7.0-samples-multiarch sleep infinity
```

3) Accessing the running container:

```
# docker exec -it <username> /bin/bash
```

- docker exec: Runs a command inside an existing container.
- -it: Provides an interactive terminal.
- <username>: Refers to the container name set earlier.
- /bin/bash: Opens a shell session within the container.

ex,

```
# docker exec -it deepstream1 /bin/bash
```

Inside the Container: Setup and Configuration

4) Updating the package list:

```
# apt update
```

- Updates the package list to ensure the latest versions are available for installation.

5) To Check the gpu usage/utilization

```
# nvidia-smi
```

6) Installing the SSH server:

```
# apt install ssh
```

- Installs the OpenSSH server to enable remote access to the container.

7) Installing the Nano text editor:

```
# apt install nano
```

- Installs the nano text editor, which allows you to modify configuration files easily.

8) Configuring SSH settings:

```
# nano /etc/ssh/sshd_config
```

- Opens the SSH server configuration file.
- Here, you can set parameters such as password authentication.
- Changes to make:
- Ensure the following line is set (if not, modify it):
- PermitRootLogin yes
- PasswordAuthentication yes
- Press Ctrl + X, then Y, and Enter to save the changes.

9)Restarting the SSH service:

```
# service ssh restart
```

- Restarts the SSH server to apply configuration changes.

10) Setting the root password:

```
# passwd root
```

- Prompts you to set a password for the root user, which will be used for SSH login.

11) Exiting the container:

```
# Exit
```

- Exits the current shell session inside the container.

Logging into the container using PuTTY(From End User Side)

Now, you can log into the running container from an external SSH client using:

- Hostname:10.101.1.101
- Port: The value you set (e.g., 0000)
- Username: root
- Password: ssn@123

If using PuTTY:

- Open PuTTY.
- Enter localhost in the "10.101.1.101" field.
- Set the port number (0000).
- Click "Open," then log in with username and password.

Administration Commands

Advanced Monitoring Tool

Glances (overview of all resources)

```
# sudo apt install glances
# glances
```

To Check GPU usage

```
# nvidia-smi
```

Docker Administration Commands

Image Management

List images

```
# docker images
```

Pull image

```
#docker pull <image-name>
```

Remove image

```
# docker rmi <image-id or name>
```

Container Management

List all containers

```
# docker ps -a
```

Start/Stop/Restart a container

```
# docker start <container-id or name>
# docker stop <container-id or name>
# docker restart <container-id or name>
```

Remove a container

```
# docker rm <container-id or name>
```

Run a container

```
# docker exec -it <container-id or name> /bin/bash
```

troubleshooting commands

1. NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver

Check if the NVIDIA kernel module is loaded:

```
# lsmod | grep nvidia
```

If there's no output, the driver may not be loaded.

Check if the driver is installed correctly:

```
# nvidia-smi
```

If it fails,

Check GPU hardware is detected:

```
# lspci | grep -i nvidia
```

If you don't see an NVIDIA device,

Restart the NVIDIA persistence daemon

```
# sudo systemctl restart nvidia-persistenced
```

Reinstall or re-load NVIDIA kernel modules:

```
# sudo modprobe nvidia
```

If modprobe fails,

```
# sudo dkms autoinstall
# sudo systemctl reboot
```

2. docker: Error response from daemon: could not select device driver "" with capabilities: [[gpu]]

Check if nvidia-docker2 is installed

```
# dpkg -l | grep nvidia-docker2
```

If it's not installed, install it:

```
# Add NVIDIA package repositories
# distribution=$(. /etc/os-release;echo $ID$VERSION_ID) && \
# curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - && \
# curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \
# sudo tee /etc/apt/sources.list.d/nvidia-docker.list

# Install NVIDIA Docker Toolkit
#sudo apt-get update && sudo apt-get install -y nvidia-docker2
```

Restart Docker

```
# sudo systemctl restart docker
```

Test NVIDIA support inside Docker

```
# docker run --rm --gpus all nvidia/cuda:12.0-base nvidia-smi
```

If you see your GPU listed,

Run your container again