

# ASPIRE 2A+ INTRODUCTORY WORKSHOP

**National Supercomputing Centre (NSCC) Singapore**

A national research infrastructure providing supercomputing solutions for all

By NSCC Singapore Managed Services Team

## A. Introduction to NSCC Singapore

## B. HPC Introduction

## C. ASPIRE 2A+

1. Architecture	5. Service Unit Allocation
2. Login Nodes	6. Software Environment
3. Storage	7. PBS Professional (Job Scheduler)
4. File Transfer	8. Usage Best Practices

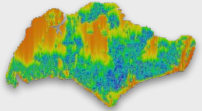
# A. Introduction to NSCC Singapore

All terms of use are subject to the prevailing

**NSCC Acceptable Use Policy**

<https://help.nscg.sg/aspire2aplus/aup/>

## Democratising Access to Supercomputing



Enhance Singapore's Research Capabilities



Support Singapore's R&D Initiatives



Attract Industrial Research Collaborations

NATIONAL  
RESEARCH  
FOUNDATION

PRIME MINISTER'S OFFICE  
SINGAPORE

A **National Research Infrastructure** funded by Singapore's **National Research Foundation (NRF)** providing supercomputing resources as a horizontal enabling platform of Singapore's Research, Innovation and Enterprise (RIE) ecosystem



**National-level resource** open to **all national research initiatives** at Institutes of Higher Learning, Research Institutes and the industry.



Singapore's **first national petascale facility and national platform** that manages the nation's high-performance computing (HPC) research resources.



**High speed networks** for HPC  
That connect researchers locally and globally.

The NSCC Singapore's overarching goal is to accelerate research and innovation in Singapore by providing the computing power and expertise needed to solve complex and data-intensive research problems, ultimately leading to the development of new technologies, products, and services that benefit society.



## Our Local Partners

### Founding Organisations



### Research Institutes



### Healthcare



### Institutes of Higher Learning



### Industry



## Our International Partners

### Alliance of Supercomputing Centres (ASC)



جامعة الملك عبد الله  
للعلوم والتقنية  
King Abdullah University of  
Science and Technology



RIKEN  
Center for  
Computational Science

# B. HPC Introduction

# What is HPC?

- HPC stands for **High Performance Computing**.
- Tightly coupled servers that embody millions of processor cores with high-speed interconnect



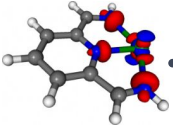
- The use of distributed computing facilities for solving problems that need large computing power.
- Usually Measured in **FLOPS (Floating point Operations Per Second)**

## Terminologies

<b>Login Node</b>	The access point for users to interact with the HPC cluster
<b>Compute Node</b>	Dedicated servers within the HPC cluster that perform the actual computations

# Why Supercomputing?

SCIENTIFIC  
ANALYSIS &  
RESEARCH



BIG DATA  
ANALYTICS



LIFE  
SCIENCES



COMPUTATIONAL  
FINANCE

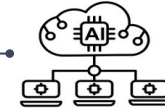


**More power for more complex research**

**Faster and more efficient solutions**

**Greater competitive edge for users**

**Long-term savings**



DEEP  
LEARNING  
/ AI



CLIMATE  
MODELLING



ADVANCED  
MANUFACTURING



DATA CENTRE &  
NETWORKING



# Major Domains Where HPC Is In Use

## LIFE SCIENCES



- **Genome** Processing
- Molecular modeling
- Pharmaceutical design

## AEROSPACE



- CFD
- Component Light weighting



## CLIMATE MODELLING

- Weather Prediction
- Climate Modeling
- Environmental Simulations

## AUTOMOTIVE; DISCRETE MANUFACTURING



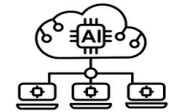
- Computational Fluid dynamics (CFD)
- Crash Test Simulations

## ENERGY



- Seismic Data Processing
- Wind Energy Enablement
- Optimization In Energy Sector

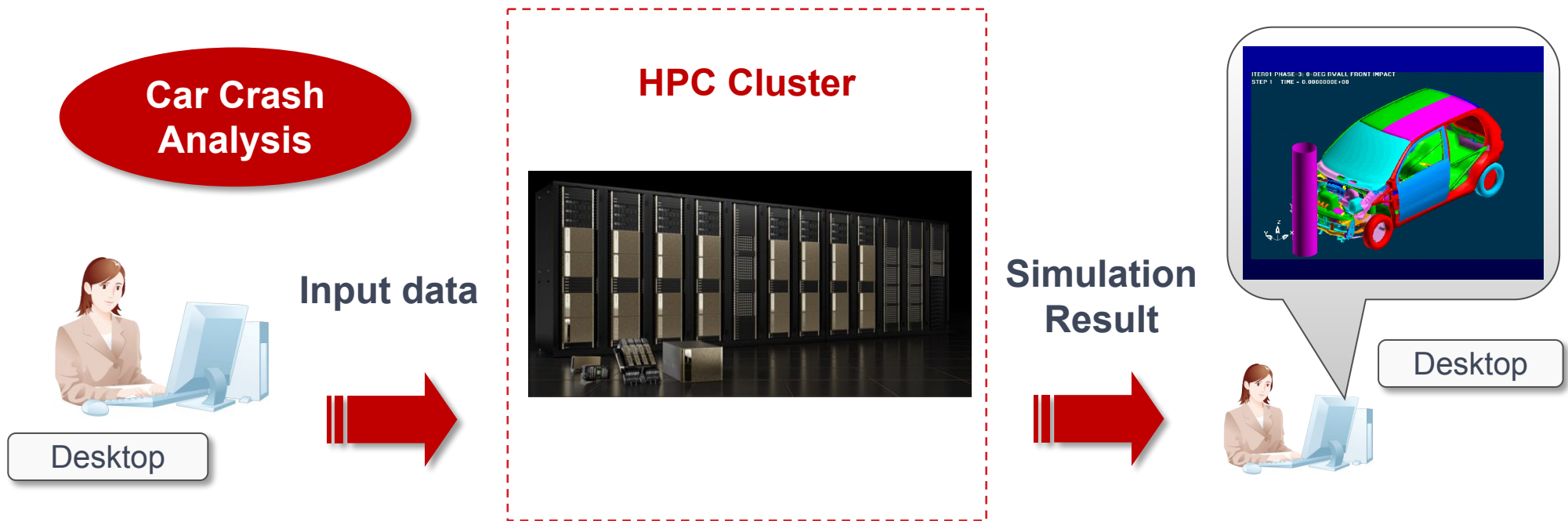
## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING



- Training Complex Models
- Running Large scale data Analysis
- Natural language Processing

# HPC Use Case – Car Crash Analysis

- HPC (High-Performance Computing) indicates that **high-speed** calculation is used with **large amounts of data**
- HPC is used for areas such as global meteorological analysis, automobile crash analysis, wind flow analysis, oil exploration etc.



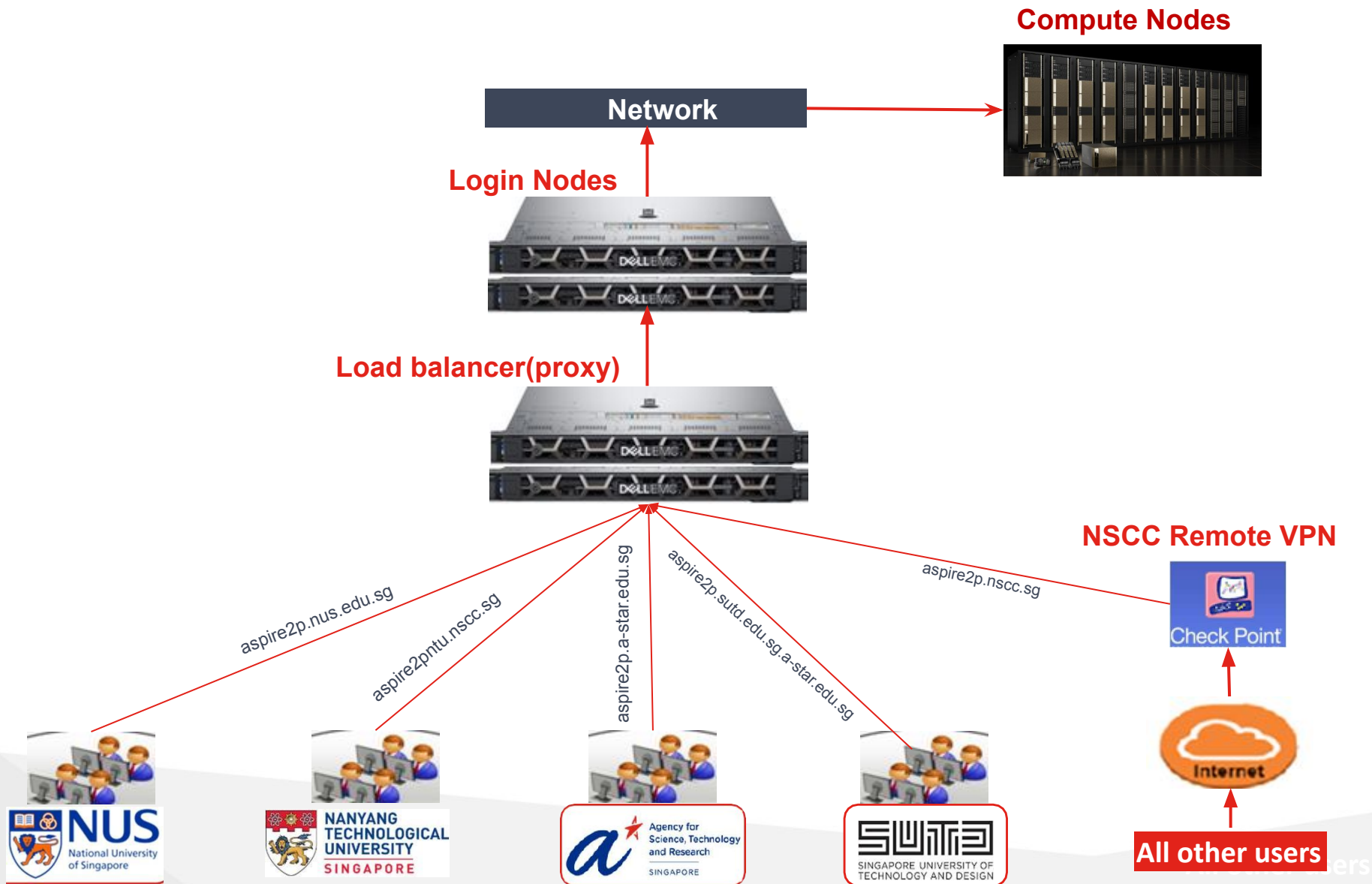
① Generate car mesh and materials as input data for a simulation

② Simulation software runs on the compute cluster

③ Display Simulation Result

# C. ASPIRE 2A+ Architecture

# ASPIRE 2A+ Architecture Overview



ASPIRE 2A+ Components	Specification
Compute Nodes	NVIDIA DGX SuperPOD™ 40 Nodes of DGX H100
Interconnect	NVIDIA Quantum 2 base NDR InfiniBand
Storage	Scratch ~ 2.5 PB, Home ~ 27.5 PB
DGX H100	Specification
CPU	Dual Intel Xeon Platinum 8480C Processors
	Total Cores = 2 X 56 Cores = 112 Cores, 2.00 GHz (Base), 3.80 GHz (Max Boost)
System Memory GPU	2 TB
GPU	8 X NVIDIA H100 Tensor Core GPUs
GPU Memory	640 GB (80 GB on each GPU Card)
Storage	8 X 3.84 TB NVMe drives
Network	4 x OSFP Ports for 8 x NVIDIA ConnectX - 7 Single Port InfiniBand Cards 8 x 400 Gb/s InfiniBand
NVSwitch	4 x 4th generation NVLink that provides 900GB/s GPU-to-GPU bandwidth
Operating System	DGX OS Ubuntu 22.04
Performance	FP64-272 teraFLOPS, TF32 (Tensor Core) - 7.9 petaFLOPS, FP8-32 petaFLOPS

# 1 PFLOP (2016) to 10-20 PFLOPS (2025)

## PREVIOUS ASPIRE 1



### 1 PFLOPS System

- 1,288 nodes (dual socket, 12 cores/CPU ES-2690v3)
- 128GB DDR4 RAM/node
- 10 large memory nodes (1x6TB, 4x2TB, 5x1TB)

### Accelerator Nodes

- 128 nodes with Tesla K40 GPUs

### 13PB Storage

- GPFS & Lustre File Systems
- I/O bandwidth up to 500GB/s

### Infiniband Interconnection

- EDR (100Gbps) Fat Tree with full bisectional bandwidth within cluster

## Add-on Systems (ASPIRE 1+)

- AI Platform (6 x DGX-1)
- 1,000 cores HTC System
- Koppen - 160 TFLOPS Cray XC-50, Climate System

## PREVIOUS ASPIRE 2A



## Awarded - 27 April 2021 (ASPIRE 2A)

- 3.33PF GPU, 2.58PF CPU compute power
- 7x more powerful than previous ASPIRE1

## CURRENT ASPIRE 2A+



## ASPIRE 2A+

- 40 x DGX H100
- 320 H100 GPUs total
- 13 PF (HPL Benchmark)
- 27.5 PB (projects) + 2.5 PB (scratch) Storage



Innovation i4.0 Building

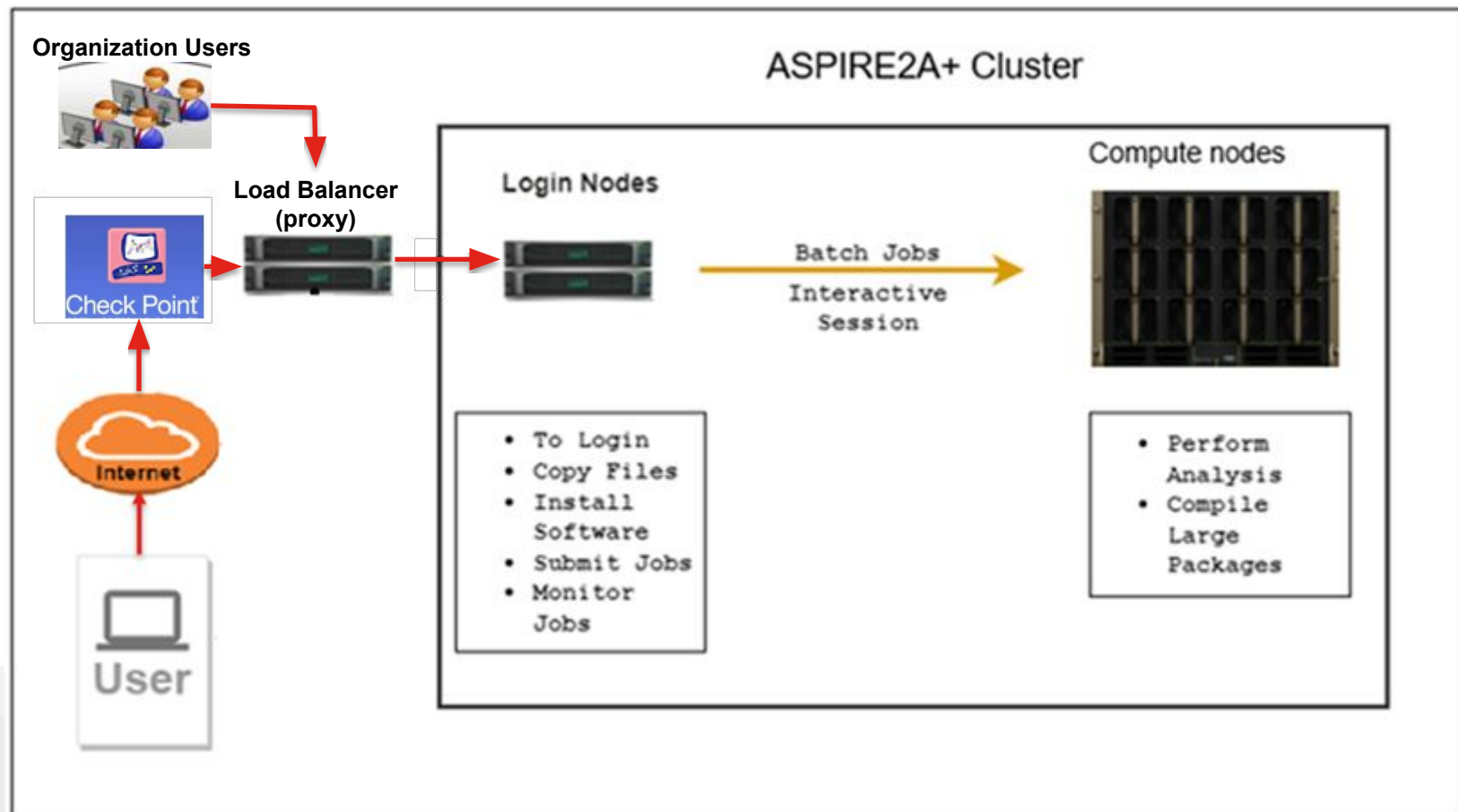


## 2. Login Nodes



# Purpose of Login Nodes

- ASPIRE 2A + features two login nodes configured in high availability (HA), allowing users to connect to DGX compute nodes for job execution.
- NSCC login nodes are the entry point for users to access the **ASPIRE 2A+ HPC system** via the load balancer.
- NSCC login nodes should not be used to verify or build your applications.



- All users from the following organisations are recommended to connect via high-speed access link to access the system as shown below.
- Organisation Users: **Direct Access**


Organisation	Login Hostname FQDN
NUS	aspire2p.nus.edu.sg
NTU	aspire2pntu.nscs.sg
<p><b>Note:- NTU users will need to request NTU jumphost to access ASPIRE2A+ via <a href="#">aspire2pntu.nscs.sg</a></b> Please email your jumphost access request to <a href="mailto:hpcsupport@ntu.edu.sg">hpcsupport@ntu.edu.sg</a> <b>How to access jump host:- <a href="#">Using-NTU-JumpHost-to-NSCC-ASPIRE-2A+</a></b></p>	
ASTAR	aspire2p.a-star.edu.sg
SUTD	aspire2p.sutd.edu.sg

- All other users are required to connect via the NSCC VPN to access the system.
- VPN Users: **VPN Access**

Hostname FQDN
aspire2p.nscs.sg
login.asp2p.nscs.sg

# Accessing Login Nodes

- To access a login node, users typically use an SSH client to connect to the login node's hostname or IP address
  - Windows ssh client : MobaXterm, putty
  - Mac ssh client : Mac terminal
  - Linux ssh client : Linux terminal



A screenshot of a MobaXterm terminal window. The top bar shows a calendar icon, the date '29/07/2024', a clock icon, the time '15:29.51', a folder icon, and the path '/home/mobaxterm'. The main terminal area shows the command 'ssh YourUserName@<HOSTNAME>' entered at the prompt.

**Example : ssh to remote host using MobaXterm**

**Note : Users who are not connected to NUS, NTU, A\*STAR, or SUTD network must ensure they are connected to their organization's VPN (default option) or NSCC VPN (for selected users only).**

# 3. Storage

File System	Mount Point	Quota Per user	Data Retention Policy
Lustre	\$HOME/scratch	100TB <b>[Fixed]</b>	30 Days Purge Policy
Lustre	/home/users/<your_org>/<userid> \$HOME	50GB <b>[Fixed]</b>	1 Year From Account Expiry
Lustre	/data/projects/<project-id>	Based on project	Based On Project Expiry**
Local NVME	/raid	NA	After Job completion

**\*\*Project data will be archived 30 days after the project's expiry date.**

## **Note : Data Management and Retention Policy**

User and project storage allocated will be purged according to AUP

<https://help.nscc.sg/aspire2p/aup/>

# Check Storage Quota

To Check Home quota, please run command as shown below.

```
]$ myquota
```

```
ganesan@azap-log1n01:~$ myquota
```

Filesystem Type		Block Limit		Inode Limit	
		Usage	Quota	Usage	Quota
/home	LUSTRE	18.52G	50G	87	0
/scratch	LUSTRE	4k	100T	1	200000000

**Note:** By default, each user has inode quota set to 200,000,000 inodes on the `/scratch` filesystem.

As shown in the screenshot, only 1 inode has been used out of the total 200,000,000 inodes available. Even if there is sufficient storage space remaining on `/scratch`, you will not be able to create any new files or directories once your inode quota is reached.

# Check Storage Quota

Check Project quota, please run command as shown below.

```
]$ myquota -p <Project-ID>
```

```
ganesan@a2ap-login01:~$ myquota -p [redacted]
```

Project : [redacted]  
[redacted]  
Organis : [redacted]  
P.Inves : NA  
Appl.Id : NA

---

P.Title : [redacted]

		Block Limit		Inode Limit	
Project ID		Usage	Quota	Usage	Quota
[redacted]	LUSTRE	710.4G	900G	7878	0



# Check Project Usage Breakdown

The `myprojects` command provides detailed information about project usage, user/project-specific details, and historical data reporting with customizable date ranges.

To check Project usage breakdown by project users, please run a command as shown below.

```
]$ myprojects -p <Project-ID> -l -s <yyyy-mm-dd> -e <yyyy-mm-dd>
```

## Where:

- p : Fetch project details for a specific project ID.
- l : Show detailed project usage.
- s : Specify the start date (in YYYY-MM-DD format) for detailed usage reporting. Defaults to yesterday if not specified.
- e : Specify the end date (in YYYY-MM-DD format) for detailed usage reporting. Defaults to yesterday if not specified.
- h : Display help for the command.

# Check Project Usage Breakdown

```
@a2ap-login01:~$ myprojects -p [redacted] -l -s 2024-09-01 -e 2024-09-31
Valid project member.

Project : [redacted]
ExpDate : [redacted]
Organis : NSCC
P.Inves : [redacted]
Appl.Id : [redacted]
Members : [redacted]
P.Title : [redacted]

Project [redacted] balance as of 12/02/2025-11:40:02
+-----+-----+-----+-----+-----+
| Unit | Grant | Used | Balance | In Doubt |
+-----+-----+-----+-----+-----+
| SU | 1000000000.000 | 1367023.653 | 98607547.012 | 25429.335 |
+-----+-----+-----+-----+-----+
In doubt - SU deducted for current running jobs (Prepaid)

Project [redacted] SU Usage breakdown
+-----+-----+-----+-----+
| Unit | Usage | SU Rate | SU Used |
+-----+-----+-----+-----+
| CPU Hour | 38.913 | 1 | 38.913 |
| GPU Hour | 10679.568 | 128 | 1366984.740 |
+-----+-----+-----+-----+

Project Usage Summary for Project: [redacted]
Date Range: 2024-09-01 to 2024-09-31
Note: Detailed usage data is only available starting from 1 Sep 2024 and onwards.

+-----+-----+
| Metric | Total Usage |
+-----+-----+
| CPU Hours | 38.91 |
| GPU Hours | 3744.54 |
| SU | 479339.92 |
+-----+-----+

Usage Breakdown by User
+-----+-----+-----+-----+
| User | CPU Hours | GPU Hours | SU |
+-----+-----+-----+-----+
| [redacted] | 0.00 | 11.32 | 1449.53 |
| [redacted] | 0.00 | 2863.70 | 366553.14 |
| [redacted] | 0.00 | 8.90 | 1139.48 |
| [redacted] | 0.00 | 145.57 | 18632.78 |
| [redacted] | 0.00 | 16.55 | 2118.08 |
| [redacted] | 0.00 | 10.34 | 1324.12 |
| [redacted] | 38.91 | 193.49 | 24805.52 |
| [redacted] | 0.00 | 5.23 | 669.26 |
| [redacted] | 0.00 | 14.00 | 1792.00 |
| [redacted] | 0.00 | 196.68 | 25174.86 |
| [redacted] | 0.00 | 278.76 | 35681.14 |
+-----+-----+-----+-----+
| Total | 38.91 | 3744.54 | 479339.92 |
+-----+-----+-----+-----+

[a2ap-login01:~$ ]
```

# 4. File Transfer

## Filezilla

- **User-friendly:** Easy graphical user interface.

## SCP (Secure Copy Protocol)

- **Simple & Secure:** Use SSH for secure transfers.

```
]$ scp /path/to/<sourcefile> <username>@<destination>:/path/to/<destination>
```

## Rsync

```
]$ rsync -avz /path/to/<source> <username>@<destination>:/path/to/<destination>
```

# Transfer Files within ASPIRE 2A+ File System

How to navigate and move/copy data between Lustre File System

## STEP 1 : Start a Screen Session

```
]$ screen -S <screen_name>
```

## STEP 2 : Submit an Interactive Job

```
]$ qsub -I -q normal -l select=1:ncpus=14:ngupu=1:mem=50GB -l  
walltime=00:30:00 -P <yourProject>
```

## STEP 3 : Initiate the Transfer

```
]$ cp -rv /path/to/source_file /path/to/destination_directory/
```

- How to share my folder with others?
- Refer to FAQ Point 5 <https://help.nscg.sg/aspire2aplus/faqs>

# Introduction to Screen Command

## What is Screen?

Screen is a terminal multiplexer that allows you to:

- Start a terminal session and keep it running even if you disconnect.
- Reconnect to the session at a later time.
- Run multiple terminal sessions within a single window.

## Basic Screen Commands

- Starting a new Screen Session with Screen Name

```
]$ screen -S <screen_name>
```

- Listing All Screen Sessions

```
]$ screen -ls
```

- Reattaching to a Session

```
]$ screen -r session_name
```

- Detaching from a Session

```
]$ Ctrl + A, then D
```

# 5. Service Unit Allocation

The **Service Unit (SU)** is the billing metric used by NSCC to measure compute and storage resource usage on an hourly basis.

## Resource-Based Costing

- Different resources (e.g., CPU, GPU) have varying costs per hour.
- SUs serve as a currency to "purchase" these resources.

## ASPIRE 2A+ Cost

- 1 GPU hour = 128 SUs (The GPUs are H100)

**Example:** If a job request 16 GPUs (spread across two nodes) for 3 hours.

Total SU usage will be:  $2 \text{ nodes} * 8 \text{ GPUs} * 3 \text{ hours} * 128 \text{ SUs} = 6144 \text{ SUs}$

## Purpose

- Enables fair and accurate accounting for resource consumption.
- Helps users manage and budget their resource usage efficiently.



## SU allocations are based on approved projects

- Projects are allocated based on approval.
- Project ID will be allocated once the project is approved.
- Project period is created based on the project allocation cycles.
- Project related approval, please help to contact **[projects-admin@nscg.sg](mailto:projects-admin@nscg.sg)**
- Project member updates can be requested at any time by the project PI by sending a request to **[projects-admin@nscg.sg](mailto:projects-admin@nscg.sg)**

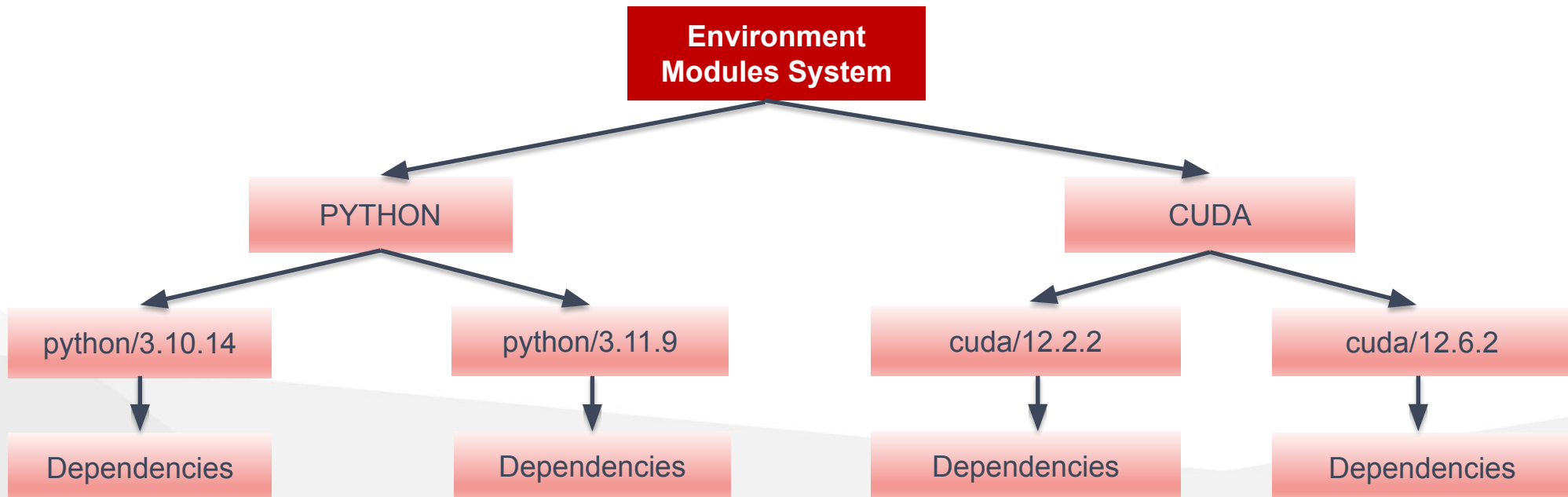
**More information in FAQs under <https://help.nscg.sg>**

# 6. Software Environment

- **Environment modules** are a tool for dynamically modifying the user environment via modulefiles.
- They are commonly used in HPC systems to manage different software versions and their dependencies.

## Benefits of Using Modules

- **Simplifies Environment Management:** Easily switch between different software versions.
- **Version Control :** Different versions of the same software can coexist, allowing users to select the appropriate version of their task.
- **Simplifies Dependency Management**



Command	Description
<code>module list</code>	List the loaded modules
<code>module avail</code>	List all module
<code>module load &lt;package name&gt;</code>	Load module
<code>module rm/unload &lt;package name&gt;</code>	Unload module **Please read the note below
<code>module swap modulefile/1 modulefile/2</code>	Swap loaded modulefile/1 with modulefile/2
<code>module -l avail 2&gt;&amp;1   egrep -i "name"</code>	Searching for module
<code>module show &lt;package name&gt;</code>	Show the configure information of module

**\*\* Unloading module or module purge can cause software incompatibility.**

- **ASPIRE 2A+** is designed for enroot containers. It is designed to integrate well with high-performance computing (HPC) environments, leveraging the capabilities of enroot to optimize execution times.
- **Key Concepts for enroot** (<https://github.com/NVIDIA/enroot>)
  - Adheres to the KISS principle and Unix philosophy
  - Standalone (no daemon)
  - Fully unprivileged and multi-user capable (no setuid binary, cgroup inheritance, per-user configuration/container store...)
  - Easy to use (simple image format, scriptable, root remapping...)
  - Little to no isolation (no performance overhead, simplifies HPC deployments)
  - Fast Docker image import (3x to 5x speedup on large images)
  - Built-in GPU support with libnvidia-container
- Users are highly recommended to **build their applications in an enroot container** prior to running them in ASPIRE 2A+.

## **LIGHTWEIGHT AND MINIMALISTIC**

Enroot is designed to be minimalistic, focusing on the essentials required to run containers without the overhead that comes with more complex runtimes.

## **NON-DAEMONISED**

Unlike Docker, which relies on a daemon to manage containers, Enroot runs without a background service. This makes it easier to integrate into existing HPC systems and avoid potential security and performance issues related to daemon processes.

## **ROOTLESS CONTAINERS**

Enroot emphasizes security by enabling unprivileged (rootless) container execution. This is crucial in HPC environments where granting root privileges is often not an option.

## **FILESYSTEM ISOLATION**

It uses Linux namespaces for isolating the filesystem, but it doesn't rely on cgroups or other advanced namespace features. This approach is simple yet effective for many HPC workloads.

## **COMPATIBILITY**

Enroot is compatible with Docker images, allowing users to run Docker containers without needing Docker itself. This is particularly useful in environments where Docker might not be available or suitable due to its complexity or security concerns.

## 7. PBS Professional (Job Scheduler)

# PBS Pro (Portable Batch System)

**PBS Pro is a powerful job scheduler** widely used in high-performance computing (HPC) environments.

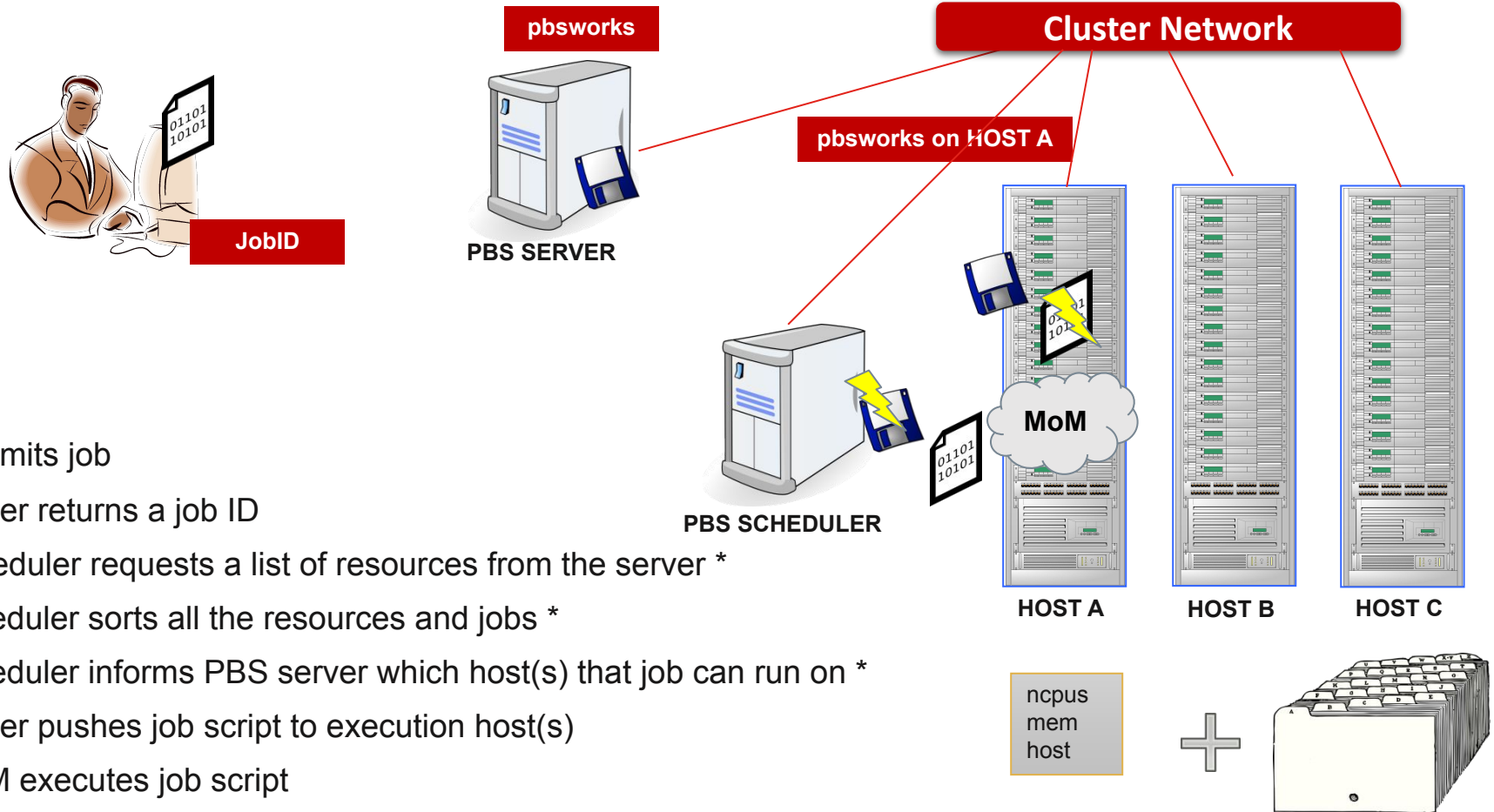
It efficiently manages and schedules computing resources, enabling users to submit both batch and interactive jobs.

## Key Features:

- **Advanced Job Scheduling** : Optimizes the order and timing of job execution to maximize resource utilization.
- **Resource Allocation & Budgeting** : Ensures fair and efficient distribution of resources across multiple users and projects.
- **Comprehensive Job Monitoring** : Provides detailed tracking and management of job statuses, ensuring smooth operation and timely completion.



# Process Flow of a PBS Job



1. User submits job
2. PBS server returns a job ID
3. PBS scheduler requests a list of resources from the server \*
4. PBS scheduler sorts all the resources and jobs \*
5. PBS scheduler informs PBS server which host(s) that job can run on \*
6. PBS server pushes job script to execution host(s)
7. PBS MoM executes job script
8. PBS MoM periodically reports resource usage back to PBS server \*
9. When job is completed PBS MoM copies output and error files
10. Job execution completed/user notification sent

Note: \* This information is for debugging purposes only. It may change in future releases.

# Job Queues & Scheduling Policies

**qsub**

**normal**

default queue

route destinations

**aidev**

Max 8 ngpus  
Max walltime=12hrs  
Interactive Jobs

**aiq1**

Max ngpus=7  
Max walltime=24hrs  
Small batch jobs

6 Nodes

**aiq2**

Min 8 ngpus  
Max walltime=120hrs  
Large batch jobs

32 Nodes\*

(Number of nodes will reduce if reservations are made)

## Current default limits when not defined

- Default “ngpus” = 1
- Default “ncpus” = 14
- Default “mem” = 235GB
- Default Queue = normal
- Default Walltime = 00:05:00
- Resource Ratio = 1 ngpus:14 ncpus:235GB mem
- PBS will modify the request according to ratio, based on the “ngpus” requested, regardless of the “ncpus” and “mem” values in original request

**PBS commands** are used to submit, manage, and modify jobs in a queue.

- **Submit batch Jobs**

- `qsub <job-script>`
- `<job-script>` file containing the instructions for the job, including the commands to execute, resource requests (e.g., CPU, memory), walltime, and other configuration settings.

- **Check Job Status**

- `qstat -answ`
- command output provides information about the currently running job on a PBS

```
a2ap-bcm01:~# qstat -answ
pbs111:
Job ID          Username      Queue        Jobname      SessID      NDS   TSK      Req'd  Req'd   Elap
-----
6634.pbs111     [redacted]    aiq1         STDIN        91914       1    112     1880g  96:00  R 95:02:55
a2ap-dgx009/0*112
Job run at Sat Oct 19 at 17:19 on (a2ap-dgx009:ncpus=112:ncpus=8:mem=1971322880kb)
```

## Delete Jobs

- Please find a JOB ID running by user before running `qdel <job-id>`
- `qstat -au <username>`

```
[redacted]@a2ap-dgx016:~$ qstat -au [redacted]
```

pbs111:

Job ID	Username	Queue	Jobname	SessID	NDS	TSK	Req'd Memory	Req'd Time	Elap S	Time
6792.pbs111	[redacted]	aiq1	STDIN	27126*	1	112	1880g	24:00	R	00:03

```
ganesan@a2ap-dgx016:~$
```

- `qdel <job-id>`

```
[redacted]@a2ap-dgx016:~$ qdel 6792.pbs111
```

Terminated

```
[redacted]@a2ap-login01:~$ qstat -au [redacted]
```

```
[redacted]@a2ap-login01:~$
```

# Types of Job Submission

## 1. Interactive Jobs

- Allows users to use compute node interactively.
- Useful for debugging, running short tests, transfer files and compile/build applications.
- **Avoid using interactive jobs for production workloads. Use batch jobs instead.**

```
$] qsub -I -l select=1:ncpus=14:mem=235gb:ngpus=1 -l walltime=01:00:00 -q normal
-P <Project-id>
```

- -I : Interactive job
- -l select=1:ncpus=14:mem=235gb:ngpus=1 : Request for 1 node with 14 CPU cores and 235gb memory and 1 GPU
- -l walltime=01:00:00 : Interactive job will be terminated after one hour

## 2. Batch Jobs

- Batch jobs are submitted using job scripts, which specify tasks, resource requirements, and execution commands.
- The scheduler manages the job script execution, allocating resources as requested.
- Users don't need to stay logged in to monitor the batch job.

# Software Environment (Enroot example)

- # Download and import the CUDA 10.0 base image from NVIDIA GPU Cloud
- \$ enroot import 'docker://\$oauth\_token@nvcr.io#nvidia/cuda:10.0-base'
- # Create a container from the imported image
- \$ enroot create --name cuda nvidia+cuda+10.0-base.sqsh
- # List all available containers
- \$ enroot list
- cuda
- # Compile the nbody sample inside the container
- \$ enroot start --root --rw cuda sh -c 'apt update && apt install -y cuda-samples-10.0'
- \$ enroot start --rw cuda sh -c 'cd /usr/local/cuda/samples/5\_Simulations/nbody && make -j'
- # Run nbody leveraging the X server from the host
- \$ export ENROOT\_MOUNT\_HOME=y NVIDIA\_DRIVER\_CAPABILITIES=all
- \$ enroot start --env DISPLAY --env NVIDIA\_DRIVER\_CAPABILITIES --mount /tmp/.X11-unix:/tmp/.X11-unix cuda \
- /usr/local/cuda/samples/5\_Simulations/nbody/nbody
- # Remove the container
- \$ enroot remove cuda

The **ENROOT ADAPTER** provides a **seamless experience** for running enroot containerized workloads in PBS Pro.

Here's a breakdown of the key features:



## AUTOMATED CONTAINER SETUP

- Validates the Enroot image path and mount points, ensuring they are correctly specified
- Automatically set the default `orte_launch_agent` supports orted and prted.



## EFFICIENT JOB PREPARATION

- Runs ``enroot create`` to initialize the container environment.
- Setup the necessary environment variables and mount points as defined in the job script.



## STREAMLINED CLEANUP

- Remove enroot container after job completion.
- Clean up any Enroot directories created during the job.

## PBS Sample Batch Job – Single node enroot container job

```
#!/bin/bash
```

```
#PBS -N my_enroot_job
```

```
### Here select=1 is to select 1 node with 7 gpus
```

```
#PBS -l select=1:ncpus=112:mem=1887gb:ngpus=7:mpiprocs=7:container_engine=enroot
```

```
#PBS -l walltime=24:00:00
```

```
#PBS -P <project-ID>
```

```
#PBS -q <queue-name>
```

```
#PBS -j oe
```

```
#PBS -l container_image=/absolute/path/to/tensorflow_image.sqsh
```

```
#PBS -l container_name=tensorflow
```

```
#PBS -l enroot_mounts="/source/path:/dest/path;/another/source:/another/dest"
```

```
#PBS -l enroot_env_file=/path/to/env_file
```

```
#Start enroot container and run the command
```

```
enroot start tensorflow mpirun -hostfile $PBS_NODEFILE python
```

```
/workspace/nvidia-examples/cnn/resnet.py --layers 50 -b 512 -i 100
```



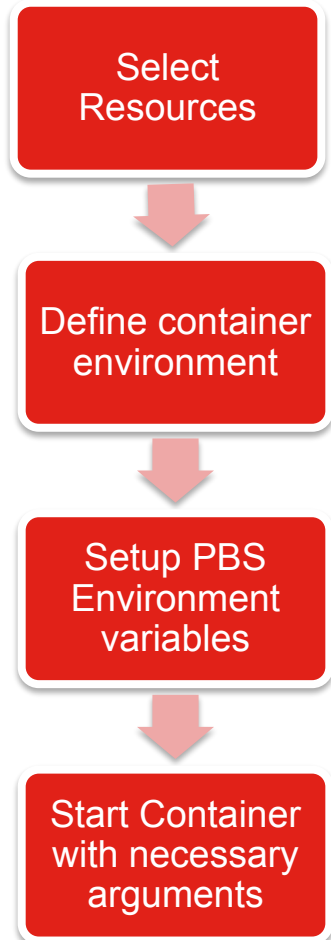
# Pytorch Data Parallel Ex – Single Node – Multi-GPU

```
#!/bin/bash
#PBS -N pytorch_dp_job
#PBS -l select=1:ngpus=2:container_engine=enroot
#PBS -l walltime=00:10:00
#PBS -P <project-id>
#PBS -q normal
```

```
#PBS -l container_image=~/.enroot_images/nvidia+pytorch+23.10-py3.sqsh
#PBS -l container_name=nvidia+pytorch+23.10-py3
#PBS -l enroot_env_file=~/.sample_jobs/container_env.conf
```

```
cd $PBS_O_WORKDIR
```

```
enroot start nvidia+pytorch+23.10-py3 python
~/pytorch_examples/torch_data_parallel_model.py --model simplenet --epochs 10
--batch_size 128 --lr 0.0001 >>
${PBS_O_WORKDIR}/pytorch_dp_job_${PBS_JOBID}.out
```



# Explanation of Pytorch Data Parallel Example

**#PBS -N pytorch\_dp\_job**

**# Job Name**

**#PBS -l select=1:ngpus=2:container\_engine=enroot**

**# Resource selection**  
**# - Select 1 node with 2 GPUs**  
**# - Use PBS-Enroot container adapter**

**#PBS -l walltime=00:10:00**

**# Walltime: Job will run for a maximum of 10 minutes**

**#PBS -P <project-id>**

**# Project ID: Assign job to a project**

**#PBS -q normal**

**# Queue: Submitting to the 'normal' queue. Change if necessary.**

**#PBS -l container\_image=~/.enroot\_images/nvidia+pytorch+23.10-py3.sqsh**

**#PBS -l container\_name=nvidia+pytorch+23.10-py3**

**#PBS -l enroot\_env\_file=~/.sample\_jobs/container\_env.txt**

**# Container details:**  
**- Specify the container image path**  
**- Set a container name for easier management**  
**- Provide an environment file for container settings**

**cd \$PBS\_O\_WORKDIR**

**# Navigate to the job's working directory**

# NCCL MPI JOB – Multi Node – Multi-GPU

```
#!/bin/bash
#PBS -l
select=2:ncpus=112:mem=1880gb:ngpus=8:mpiprocs=8:container_engine=enroot
#PBS -l walltime=1:00:00
#PBS -P <project-id>
#PBS -q normal
```

```
#PBS -l container_name=nvidianccl
#PBS -l container_image=$HOME/NCCL/nvidia+tensorrt+23.12-py3.sqsh
#PBS -l enroot_env_file=~/.NCCL/myenv.conf
```

```
cd $PBS_O_WORKDIR
```

```
enroot start nvidianccl mpirun -hostfile $PBS_NODEFILE
/usr/local/bin/all_reduce_perf_mpi -b 8 -f 2 -g 1 -e 16G -n 8000
```

Select  
Resources

Define  
container  
environment

Setup PBS  
Environment  
variables

Start Container  
with necessary  
arguments

# NCCL MPI JOB - env config in container namespace

- To set environment variables across all containers in multi-node jobs, create a file with the variables and specify its path using the **enroot\_env\_file** directive:

```
#PBS -l enroot_env_file=~/.NCCL/myenv.conf
```

- This is optional and needed only when setting up the environment across all container namespaces.

```
altairsup@a2ap-dgx001:~/NCCL$ cat myenv.conf
UCX_TLS=rc
OMPI_MCA_pml=ucx
OMPI_MCA_coll_hcoll_enable=0
OMPI_MCA_btl=openib,smcud
NVSHMEM_ENABLE_NIC_PE_MAPPING=1
NVSHMEM_HCA_LIST=mlx5_0:1,mlx5_3:1,mlx5_4:1,mlx5_5:1,mlx5_6:1,mlx5_9:1,mlx5_10:1,mlx5_11:1
RDMA_IF="mlx5_0,mlx5_3,mlx5_4,mlx5_5,mlx5_6,mlx5_9,mlx5_10,mlx5_11"
NCCL_IB_QPS_PER_CONNECTION=2
NCCL_IB_SPLIT_DATA_ON_QPS=0
NCCL_IB_HCA="mlx5_0,mlx5_3,mlx5_4,mlx5_5,mlx5_6,mlx5_9,mlx5_10,mlx5_11"
NCCL_DEBUG=INFO
```

```
#!/bin/bash
```

```
#PBS -N my_enroot_job
```

```
##Here select=3 is to select 3 node with 8 gpus each
```

```
#PBS -l select=3:ncpus=112: mem=1887gb:ngpus=8:mpiprocs=8:container_engine=enroot
```

```
#PBS -l walltime=48:00:00
```

```
#PBS -P <project-id>
```

```
#PBS -q <queue-id>
```

```
#PBS -j oe
```

```
#PBS -l container_image=/absolute/path/to/tensorflow_image.sqsh
```

```
#PBS -l container_name=tensorflow
```

```
#PBS -l enroot_mounts="/source/path:/dest/path;/another/source:/another/dest"
```

```
#PBS -l enroot_env_file=/path/to/env_file
```

```
#Start enroot container and run the command
```

```
enroot start tensorflow mpirun -hostfile $PBS_NODEFILE python  
/workspace/nvidia-examples/cnn/resnet.py --layers 50 -b 512 -i 100
```

**\$cat /path/to/env\_file**

```
# Sample environment file
```

```
VAR1=value1
```

```
VAR2=value2
```

```
PATH=/path/to/binary:$PATH
```

```
LD_LIBRARY_PATH=/path/to/lib:$LD_LIBRARY_PATH
```

# 8. Usage Best Practices

## OPTIMIZE RESOURCE ALLOCATION

- **Distribute Workloads:** ensure that workloads are evenly distributed across the memory and GPU cards to avoid bottlenecks.

## NVIDIA DGX H100 GPU SYSTEM SPECIFICATIONS:

- 8 X NVIDIA H100 Tensor Core GPUs
- 640 GB (80 GB on each GPU Card)

### Example : Memory distribution [1:4]

- 1 Core Configuration: 4 GB \* 1 core
- 128 Cores Configuration: 440GB \* 128 cores



- **ALWAYS Start with Single GPU first (8 GPU card)**
  - ▢ **Efficient Utilization** : Ensure your application can utilize close to 100% of the compute capacity on GPU card before scaling your job.
  - ▢ **Usage Monitoring**: Always monitor your workload using **top** and **nvidia-smi** for CPU and GPU workload respectively.
- **Scale to Multiple GPU Cards**
  - ▢ **Incremental Scaling** : Gradually increase the number of nodes or GPU cards example : 8, 16, 24, 32 etc. measuring performance for each configuration
  - ▢ **Parallel Efficiency** : Study the workload performance for each configuration and choose the one the offers the best parallel efficiency for future workloads.
- **GPU Workload** (Make sure to install or load GPU version of your application)
  - ▢ **Read Documentation** : Running workloads on GPUs requires using the correct application options. Ensure you are using the optimized options.

## Do's

- Install applications in Home or Project directory.
- Share files through Project directory.
- Regular housekeeping in Home, Project and Scratch directories.
- Use Scratch directory for temporary files.
  - Move essential files to project directory after job completion. Note : Files in scratch are subjected to purging.
- Use striping for large files on scratch directory by using 'lfs setstripe' command.
- Execute data transfer on loginnodes for optimal performance.
- Utilize the 'find' and 'rm' commands for efficient file management.

## Don'ts

- Avoid running any computational workload on login nodes.
- Avoid installing applications on Scratch directory.
- Avoid copying & pasting job scripts from web or Microsoft Windows OS.
- Avoid static settings in .bashrc, instead make use of environment modules.
- Avoid setting world readable/writable files & directories on Home, Scratch and Project directories.
- Avoid creating/accessing large number of small files in scratch directory.
- Avoid deleting large number of files using "rm -rf \*".

# Contact Us



**Website** : <https://nscg.sg>



**Self Service  
Portal** : <https://help.nscg.sg/>



**Helpdesk** : <https://keris.service-now.com/csm>



**Email** : [help@nscg.sg](mailto:help@nscg.sg)



**Contact** : +65 6645 3412



Email : [help@nscg.sg](mailto:help@nscg.sg)

# Thank You