



NVIDIA DGX SuperPOD

User Guide

Featuring NVIDIA DGX H100 and DGX A100 Systems



Note: With the release of NVIDIA Base Command Manager 10.23.09, the NVIDIA DGX SuperPOD User Guide is no longer being maintained. Refer instead to the NVIDIA Base Command Manager User Manual on the [Base Command Manager documentation site](#).

Contents

1.	NVIDIA DGX SuperPOD Overview	1
1.1	Logical System Diagram	1
1.2	Navigating the DGX SuperPOD	3
2.	Workload Management.....	4
2.1	Introduction	4
2.2	Viewing System State	4
2.3	Running Jobs	5
2.3.1	Running Jobs with sbatch.....	5
2.3.2	Running Jobs with srun	5
2.3.3	Running Interactive Jobs with srun.....	6
2.4	Specifying Resources when Submitting Jobs	6
2.5	Monitoring Jobs	7
2.6	Canceling Jobs.....	7
2.7	Additional Resources	7
3.	Using Containers	8
3.1	Examples	8

1. NVIDIA DGX SuperPOD Overview

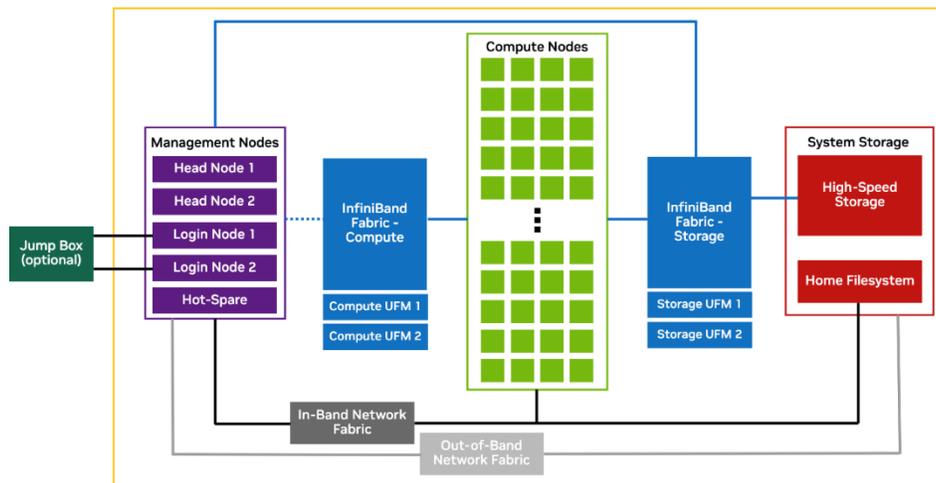
The NVIDIA DGX SuperPOD™ is a multi-user system designed to run large artificial intelligence (AI) and high-performance computing (HPC) applications efficiently. While the system is composed of many different components, it should be thought of as a single system that can manage simultaneous use by many users and provide advanced access controls for queuing and scheduling resources. This ensures maximum performance, provides the tools for collaboration between users, and security controls to protect data and limit user interaction where necessary.

This document does not cover information about the DGX SuperPOD that is specific to local policies or general Unix/Linux topics such as access, queuing, quotas, compiling, and editing and manipulating files and data.

1.1 Logical System Diagram

Figure 1 provides a logical depiction of the DGX SuperPOD and all the components that enable it to work as a single multi-user system.

Figure 1. Logical depiction of the DGX SuperPOD



The boxes and connections in Figure 1 indicate that these components are not a part of the user-experience. Any lines that are dotted indicate that there is some connectivity between the two resources, but not necessarily every sub-component is connected. The optional jump box is an optional component outside of the DGX SuperPOD that enables remote access into it.

Components from Figure 1 are further described in Table 1.

Table 1. DGX SuperPOD components

Component	Description
Jump Box/Entry Point	The Jump Box/Entry Point is the gateway into the DGX SuperPOD intended to provide a single-entry point into the cluster and additional security when required. It is not actually apart of the DGX SuperPOD, but of the corporate IT environment. This function is defined by local IT requirements.
Management Nodes	The management nodes are the entry point for the user into the DGX SuperPOD. A login node is a CPU-only node for light-weight tasks where the user can develop code, submit, and monitor jobs, and manage your data.
Compute Nodes	The compute nodes are where the user work gets done on the system. Each compute node is an individual server, but with the high-speed fabric, applications can be efficiently spread out across multi-nodes.
High-Speed Storage	High-speed storage is optimized for efficient reading and writing of large data files. The high-speed storage is often treated as scratch, as it is difficult or impossible to back-up all data stored on the system. This is where datasets, checkpoints, and other large files should be stored.
Home File System	The home file system is a traditional highly reliable network file system that trades performance for stability and enterprise management features. The assigned space is generally smaller than what is available on high-speed storage. Users should store scripts, code, Dockerfiles, and other small and important files.
Compute Fabric	The compute fabric is the high-speed network fabric connecting all the compute nodes together to enable high-bandwidth and low-latency communication between the compute nodes.
Storage Fabric	The storage fabric is the high-speed network fabric dedicated for storage traffic. Storage traffic is dedicated to its own fabric to remove interference with the node-to-node application traffic that can degrade overall performance.
In-band Management Network	The In-band Network Fabric provides fast Ethernet connectivity between all the nodes in the DGX SuperPOD. While its use should be transparent to the users, it hosts important traffic for node management and home file system access.

1.2 Navigating the DGX SuperPOD

When a user first logs into the DGX SuperPOD, it will look like any other Linux system. They will be placed into their home directory and standard Linux commands will work.

For example:

```
# pwd
/home/dgxuser
# ls -al
./bashrc
```

In addition, the high-speed file system will be available on the login nodes and all the compute nodes:

```
# ls /lustre/fs1/
projects
```

The DGX SuperPOD is a collection of nodes and access is managed through the workload management system. The default workload management system is Slurm. Slurm enables submitting and managing jobs. See [Workload Management](#) for more details.



Note: The description regarding accessing the DGX SuperPOD is the default way of using the command line to interact with the system. Local deployments may provide other user interfaces for interacting with the system. In addition, the examples in this document use a standard naming convention for system names and directories but may be changed for a given environment.

2. Workload Management

2.1 Introduction

Workload management is the submission and control of work on the system. [Slurm](#) is the workload management system used. Slurm is an open-source job scheduling system for Linux clusters, most frequently used for HPC applications. This guide covers some of the basics to get started using Slurm as a user on the DGX SuperPOD, including how to use Slurm commands such as [sinfo](#), [srun](#), [sbatch](#), [squeue](#), and [scancel](#).

The basic flow of a workload management system is that the user submits a job to the queue. A job is a collection of work to be executed. Shell scripts are the most common because a job often consists of many different commands.

The system will take all the jobs submitted that are not yet running, look at the state of the system, and then map those jobs to the available resources. This workflow enables users to manage their work within large groups with the system determining the optimal way to order jobs for maximum system utilization (or other metrics that system administrators can configure).

2.2 Viewing System State

To see all nodes in the cluster and their current state, ssh to the Slurm login node for your cluster and run the `sinfo` command:

```
$ sinfo
PARTITION    AVAIL    TIMELIMIT    NODES    STATE    NODELIST
batch*       up       infinite     9        idle    dgx[1-9]
```

There are nine nodes available in this example, all in an idle state. If a node is busy, its state will change from `idle` to `alloc` when the node is in use:

```
$ sinfo
PARTITION    AVAIL    TIMELIMIT    NODES    STATE    NODELIST
batch*       up       infinite     1        alloc   dgx1
batch*       up       infinite     8        idle    dgx[2-9]
```

2.3 Running Jobs

There are three ways to run jobs under Slurm. Jobs can be run with `sbatch`, where the work is queued in the system and control is returned to the prompt. The second is with `srun`, which will run the job on the system and the command will block while it waits to run and then runs to completion. The third way is to submit interactive jobs where `srun` is used to create the job, but shell access is given.

2.3.1 Running Jobs with `sbatch`

While the `srun` command blocks any other execution in the terminal, `sbatch` can be run to queue a job for execution when resources are available in the cluster. Also, a batch job will enable several jobs to queue up and run as nodes become available. It is therefore good practice to encapsulate everything that must be run into a script and then execute with `sbatch`.

```
$ cat script.sh
#!/bin/bash
/bin/hostname sleep 30

$ sbatch script.sh
2322
$ squeue
JOBID PARTITION   NAME     USER ST       TIME  NODES NODELIST(REASON)
 2322   batch     script.sh user  R         0:00     1    dgx1
$ ls
slurm-2322.out
$ cat slurm-2322.out
dgx1
```

2.3.2 Running Jobs with `srun`

To run a job, use the `srun` command:

```
$ srun hostname
dgx1
```

This instructed Slurm to find the first available node and run `hostname` on it. It returned the result in our command prompt. It is just as easy to run a different command that runs a python script or a container using `srun`.

Sometimes it is necessary to run on multiple systems:

```
$ srun --ntasks 2 -l hostname
dgx1
dgx2
```

2.3.3 Running Interactive Jobs with srun

When developing and experimenting, it is helpful to run an interactive job, which requests a resource and provides a command prompt as an interface to it:

```
slurm-login:~$ srun --pty /bin/bash
dgx1:~$ hostname
dgx1
dgx1:~$ exit
```

During interactive mode, the resource is being reserved for use until the prompt is exited. Commands can be run in succession.

Before starting an interactive session with `srun`, it may be helpful to create a session on the login node with a tool like `tmux` or `screen`. This will prevent a user from losing interactive jobs if there is a network outage or the terminal is closed.

**Note:** Local administrative policies may restrict or prevent interactive jobs. Ask a local system administrator for specific information about running interactive jobs.

2.4 Specifying Resources when Submitting Jobs

When submitting a job with `srun` or `sbatch`, request the specific resources needed for the job. Allocations are all based on tasks. A task is a unit of execution. Multiple GPUs, CPUs, or other resources can be associated to a task. A task cannot span a node. A single task or multiple tasks can be assigned to a node.

As shown in Table 2 Resources can be requested several different ways.

Table 2. Methods to specify `sbatch` and `srun` options

<code>sbatch/srun</code> Option	Description
<code>-N, --nodes=</code>	Specify the total number of nodes to request
<code>-n, --ntasks=</code>	Specify the total number of tasks to request
<code>--ntasks-per-node=</code>	Specify the number of tasks per node
<code>-G, --gpus=</code>	Total number of GPUs to allocate for the job
<code>--gpus-per-task=</code>	Number of GPUs per task
<code>--gpus-per-node=</code>	Number of GPUs to be allocated per node
<code>--exclusive</code>	Guarantee that nodes are not shared among jobs

While there are many combinations of options, here are a few common ways to submit jobs:

- > Request two tasks:

```
srun -n 2 <cmd>
```

- > Request two nodes, eight tasks per node, and one GPU per task:

```
sbatch -N 2 --ntasks-per-node=8 --gpus-per-task=1 <cmd>
```

- > Request 16 nodes, eight GPUs per node:

```
sbatch -N 16 --gpus-per-node=8 --exclusive <cmd>
```

2.5 Monitoring Jobs

To see which jobs are running in the cluster, use the `squeue` command:

```
$ squeue -a -l
Tue Nov 17 19:08:18 2020
JOBID PARTITION NAME USER STATE TIME TIME_LIMIT NODES NODELIST(REASON)
9 batch bash user01 RUNNING 5:43 UNLIMITED 1 dgx1
10 batch Bash user02 RUNNING 6:33 UNLIMITED 2 dgx[2-3]
To see just the running jobs for a particular user USERNAME:
$ squeue -l -u USERNAME
```

The `squeue` command has many different options available. See the man page for more details.

2.6 Canceling Jobs

To cancel a job, use the `scancel` command:

```
$ scancel JOBID
```

2.7 Additional Resources

Additional resources include:

- > [SchedMD Slurm Quickstart Guide](#)
- > [LLNL Slurm Quickstart Guide](#)
- > <https://github.com/NVIDIA/deepops/blob/master/docs/slurm-cluster/slurm-usage.md>

3. Using Containers

Containers provide a way to encapsulate all the software dependencies of an application and enable it to be deployed on different systems. Containers are the preferred way to run applications on the DGX SuperPOD.

The DGX SuperPOD is deployed with two tools, [Pyxis](#) and [Enroot](#), to help simplify the secure use of containers on the DGX SuperPOD. Pyxis extends the functionality of Slurm so that jobs can be launched directly into a container with `srun`. Enroot is a light-weight container-runtime that enables traditional container images to be run in unprivileged mode.

3.1 Examples

Here are some example commands for working with user containers:

- > Submit a job to Slurm on a worker node.

```
$ srun grep PRETTY /etc/os-release
PRETTY_NAME="Ubuntu 20.04.4 LTS"
```

- > Submit a job to Slurm and launching it in a container.

The `--container-image` option is used to specify which container to use.

```
$ srun --container-image=centos grep PRETTY /etc/os-release
PRETTY_NAME="CentOS Linux 7 (Core)"
```

- > Mount a file from the host and run the command on it from inside the container.

```
$ srun --container-image=nvcr.io/nvidia/pytorch:22.12-py3 --container-mounts=/etc/os-
release:/host/os-release grep PRETTY /host/os-release
pyxis: importing docker image: nvcr.io/nvidia/pytorch:22.12-py3
pyxis: imported docker image: nvcr.io/nvidia/pytorch:22.12-py3
PRETTY_NAME="Ubuntu 20.04.4 LTS"
```

- > The `--container-mounts` option can be used to mount both files and directories into the container environment. Multiple options should be separated by commas.

```
$ srun -N 2 --ntasks-per-node=1 --container-image=nvcr.io/nvidia/pytorch:22.12-py3 --
container-mounts=/etc/os-release:/host/os-release grep PRETTY /host/os-release
pyxis: imported docker image: nvcr.io/nvidia/pytorch:22.12-py3
pyxis: imported docker image: nvcr.io/nvidia/pytorch:22.12-py3
```

- > Submit the same command across two nodes, mounting the current directory as /work in the container.

The full network name of the container is different. Enroot requires the separator between the network repository name (nvcr.io in this case) to be separated by a #, not a slash (/).

```
srun -N 2 --ntasks-per-node=1 \  
--container-image=nvcr.io/nvidia/pytorch:22.12-py3 --container-mounts=$(pwd):/work \  
/bin/bash -c 'uname -n && cat /etc/os-release | grep PRETTY_NAME'  
dgx1  
PRETTY_NAME="Ubuntu 20.04.5 LTS"  
dgx2  
PRETTY_NAME="Ubuntu 20.04.5 LTS"
```

Further resources are available at these links:

- > For a tutorial on running a multi-node Pyxis/Enroot BERT container, see this [guide](#).
- > For a hello world tutorial on using MPI to run multi-gpu and multi-node jobs, see this [guide](#).
- > For a tutorial on running a multi-node machine learning job using Dask on Slurm, see this [guide](#).

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of or der acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, NVIDIA DGX and NVIDIA DGX SuperPOD are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2023 NVIDIA Corporation. All rights reserved.