www.locuz.com

DGX-A100 User Training

Presented By: Mandeep Kumar

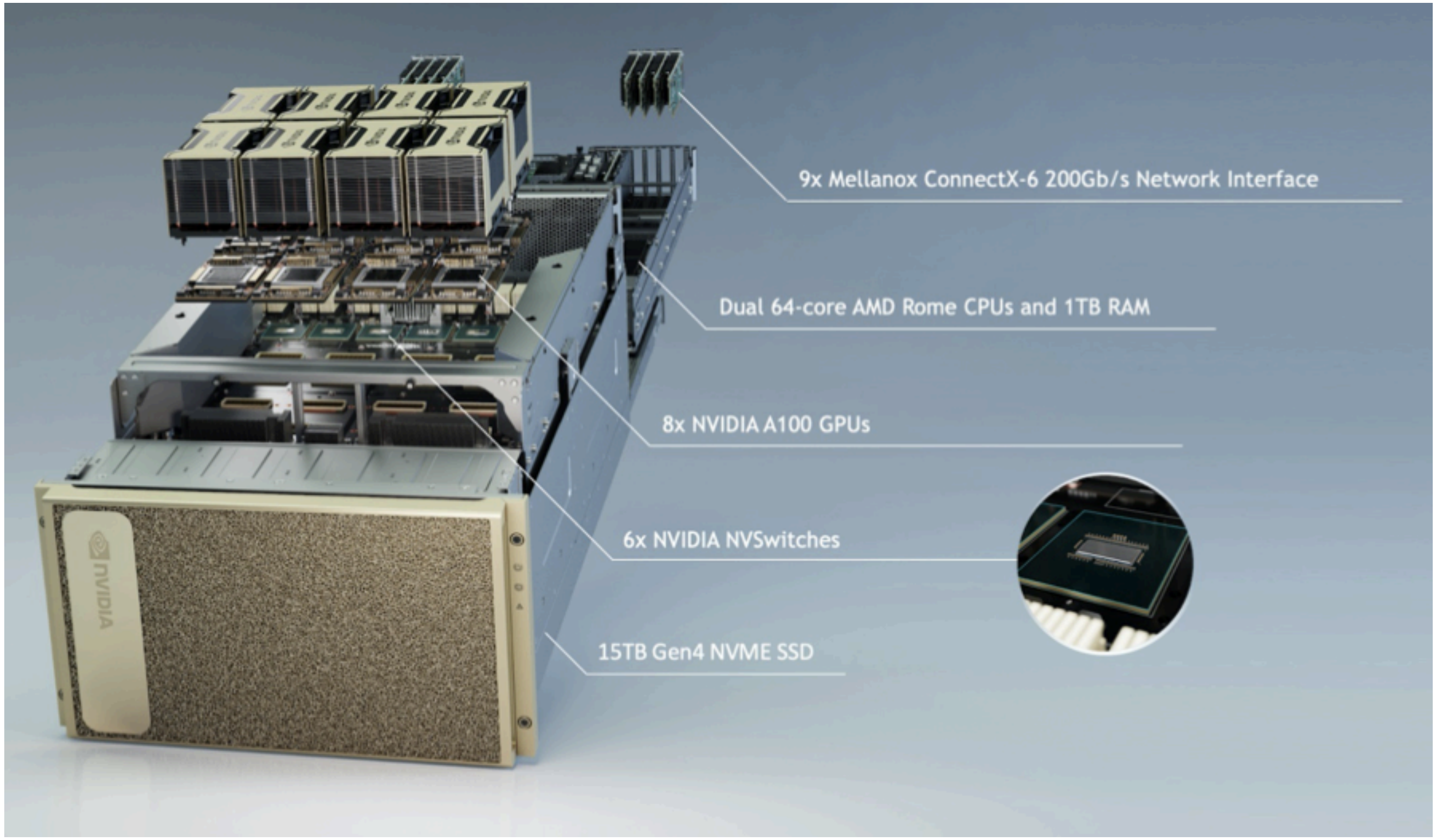
NVIDIA

LOCUZ Converge to the Cloud

Agenda

- DGX-A100 Overview
- NVIDIA GPU Cloud (NGC)
- Virtual Machine vs. Container
- What’s Docker?
- Why NVIDIA Docker?
- NVIDIA Docker Sub-Commands
- Running Docker Containers
- Docker on HPC Systems
- Singularity: A Container Engine for HPC
- Running with Singularity
- Containers (Singularity) with PBS Sample Script

NVIDIA DGX A100 - Game-Changing Performance for Innovators

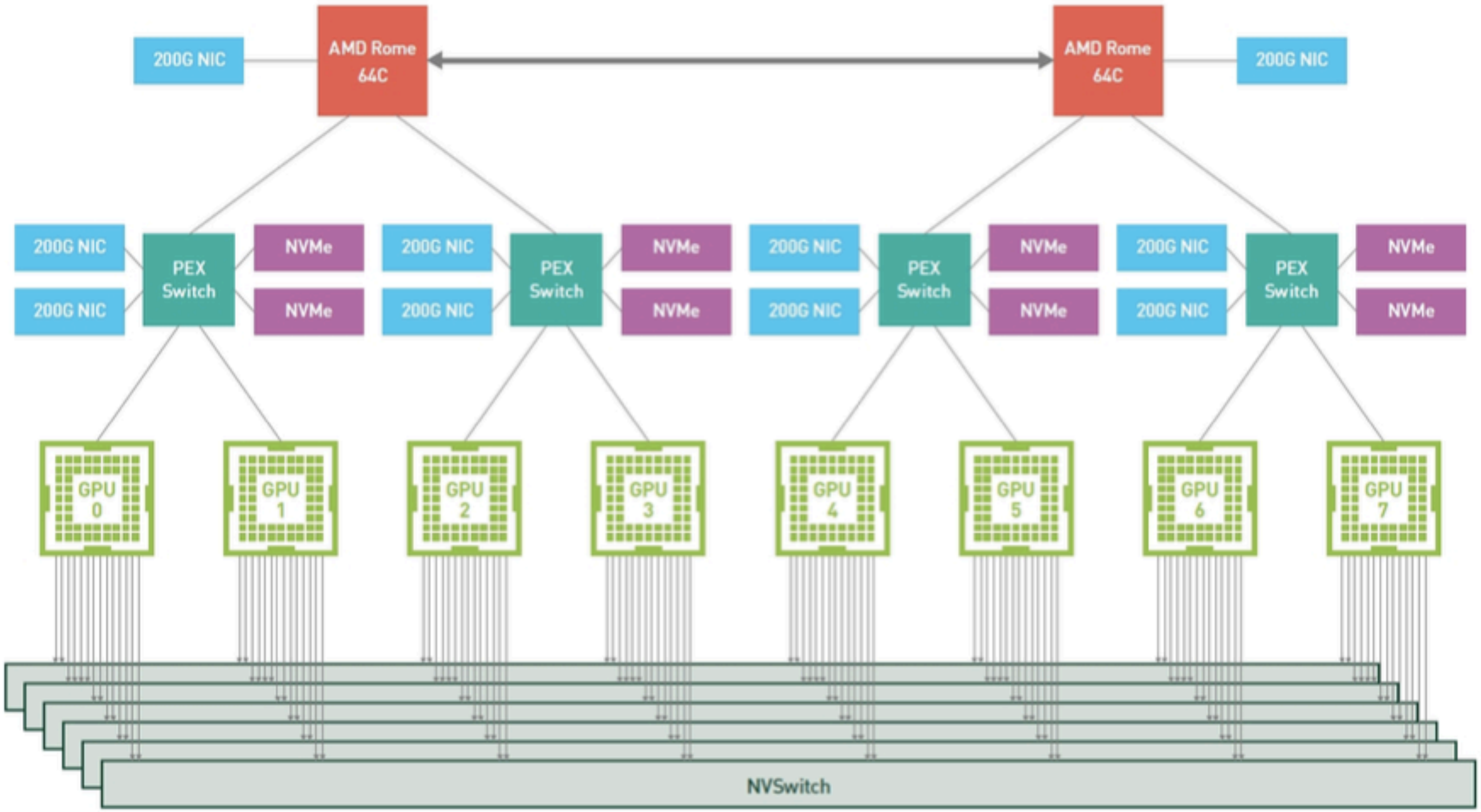


NVIDIA DGX A100 System Specs

Apps Focus Components	
GPUs	8x NVIDIA A100 Tensor Core GPUs
GPU Memory	320 GB Total
NVIDIA NVSwitch	6
Performance	5 petaFLOPS AI 10 petaFLOPS, INT8
CPU	Dual AMD Rome, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost)
System Memory	1TB
Networking	9x Mellanox ConnectX-6 VPI HDR InfiniBand/200GigE 10th Dual-port ConnectX-6 optional
Storage	OS: 2x 1.92TB M.2 NVME drives Internal Storage: 15TB (4x 3.84TB) U.2 NVME drives

Power and Physical Dimensions	
System Power Usages	6.5 kW Max
System Weight	271 lbs (123 kg)
System Dimensions	6 Rack Units (RU) Height: 10.4 in (264.0 mm) Width: 19.0 in (482.3 mm) Max Length: 35.3 in (897.1 mm) Max
Operating Temperature	5°C to 30°C (41°F to 86°F)
Cooling	Air

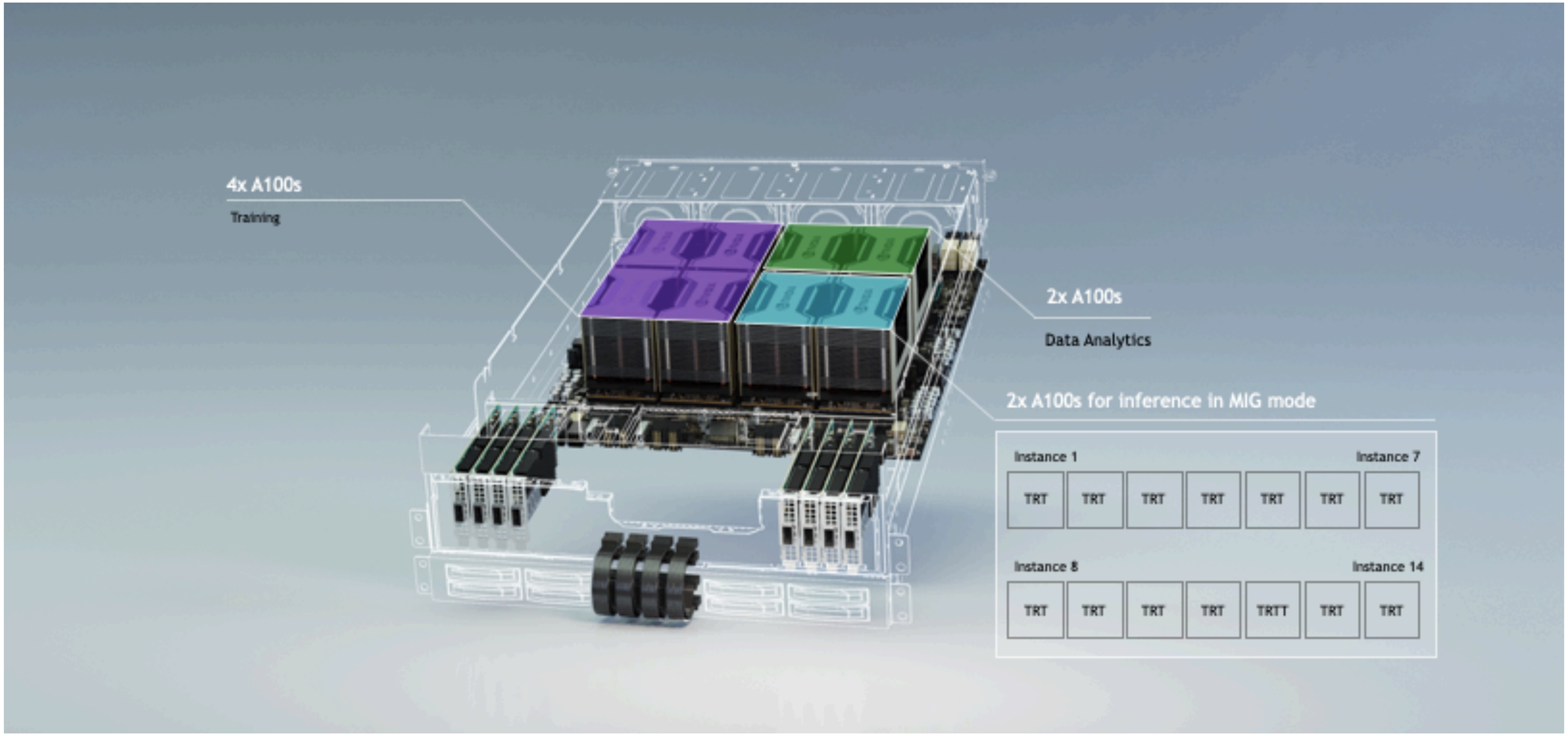
NVIDIA DGX A100 with Eight A100 GPUs



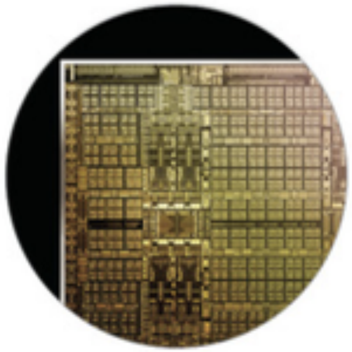
NOTE: Third-Generation NVLink connectivity through NVSwitches

NVIDIA DGX A100 - The Universal System for AI Infrastructure

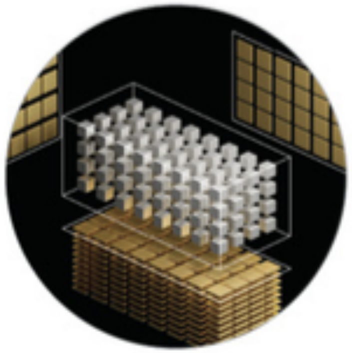
One Platform for Training, Inference and Data Analytics



New Technologies in NVIDIA A100

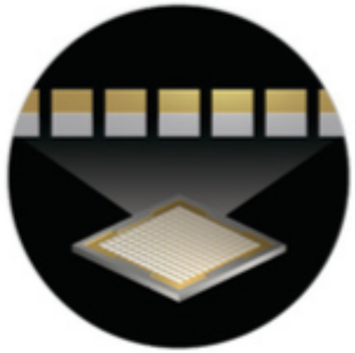


54 BILLION XTORS



3^{rd} GEN
TENSOR CORES

SPARSITY
ACCELERATION



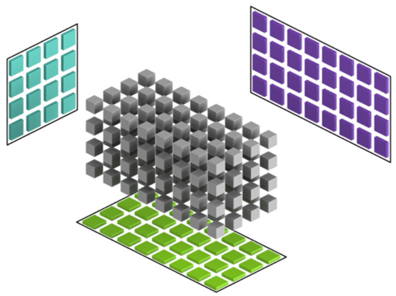
MIG

3^{rd} GEN
NVLINK & NVSWITCH

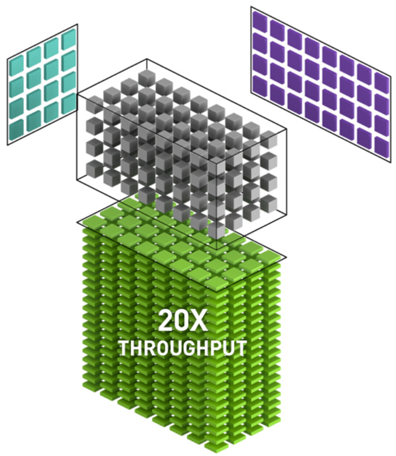
New TF32 Tensor Cores on A100

20X Higher FLOPS for AI, Zero Code Change

NVIDIA V100 FP32

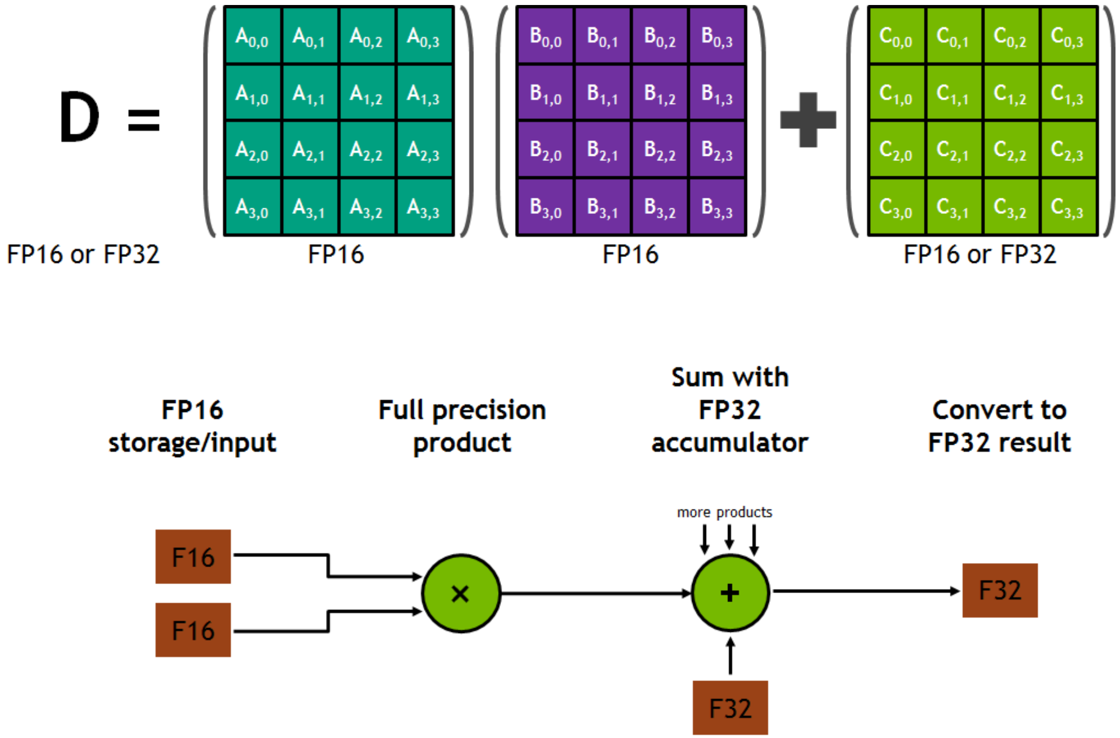


NVIDIA A100 Tensor Core TF32 with Sparsity



20X Faster than Volta FP32 | Works like FP32 for AI with Range of FP32 and Precision of FP16
No Code Change Required for End Users | Supported on PyTorch, TensorFlow and MXNet Frameworks Containers

Tensor Cores



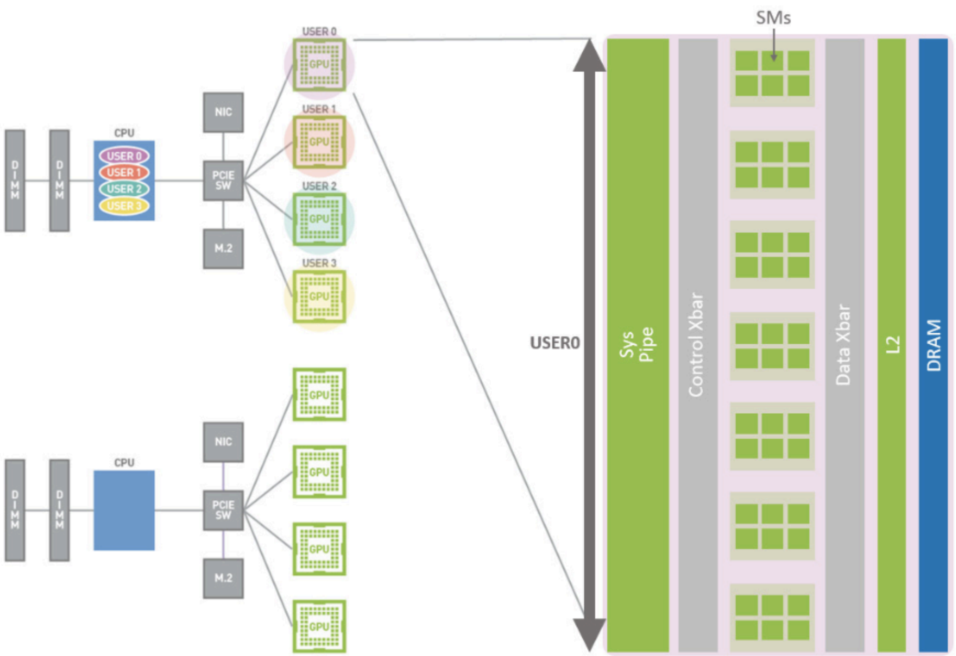
Most Flexible AI Platform with MULTI-INSTANCE GPU (MIG)

Optimize GPU Utilization, Expand Access to More Users with Guaranteed Quality of Service

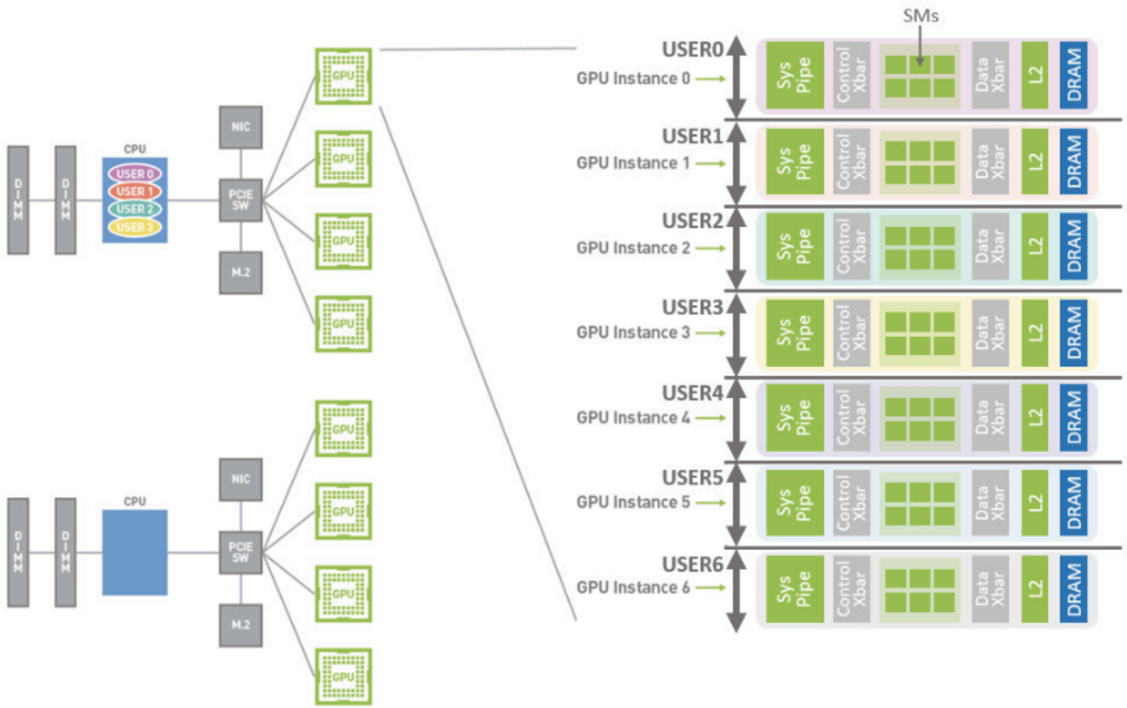
- **Up To 7 GPU Instances In a Single A100**
- **Simultaneous Workload Execution With Guaranteed Quality Of Service**
  All MIG instances run in parallel with predictable throughput & latency
- **Flexibility** to run any type of workload on a MIG instance
- **Right Sized GPU Allocation**
  Different sized MIG instances based on target workloads

Important Use Case for MIG

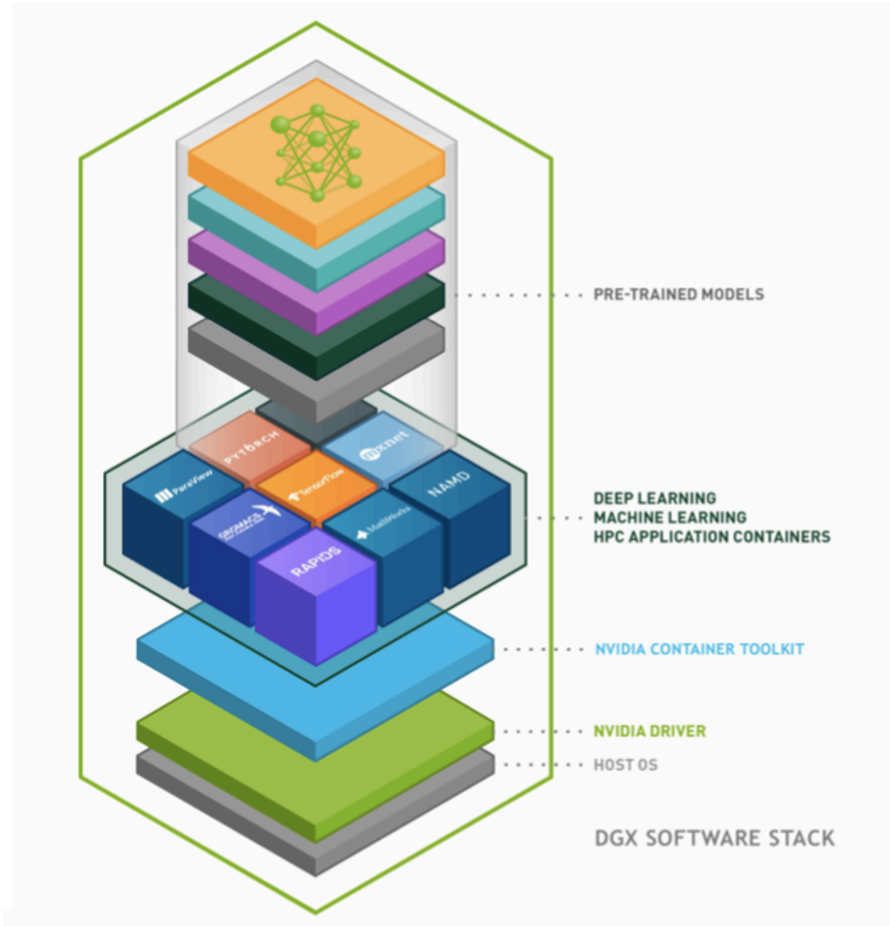
Multi-User Node Today (pre-A100)



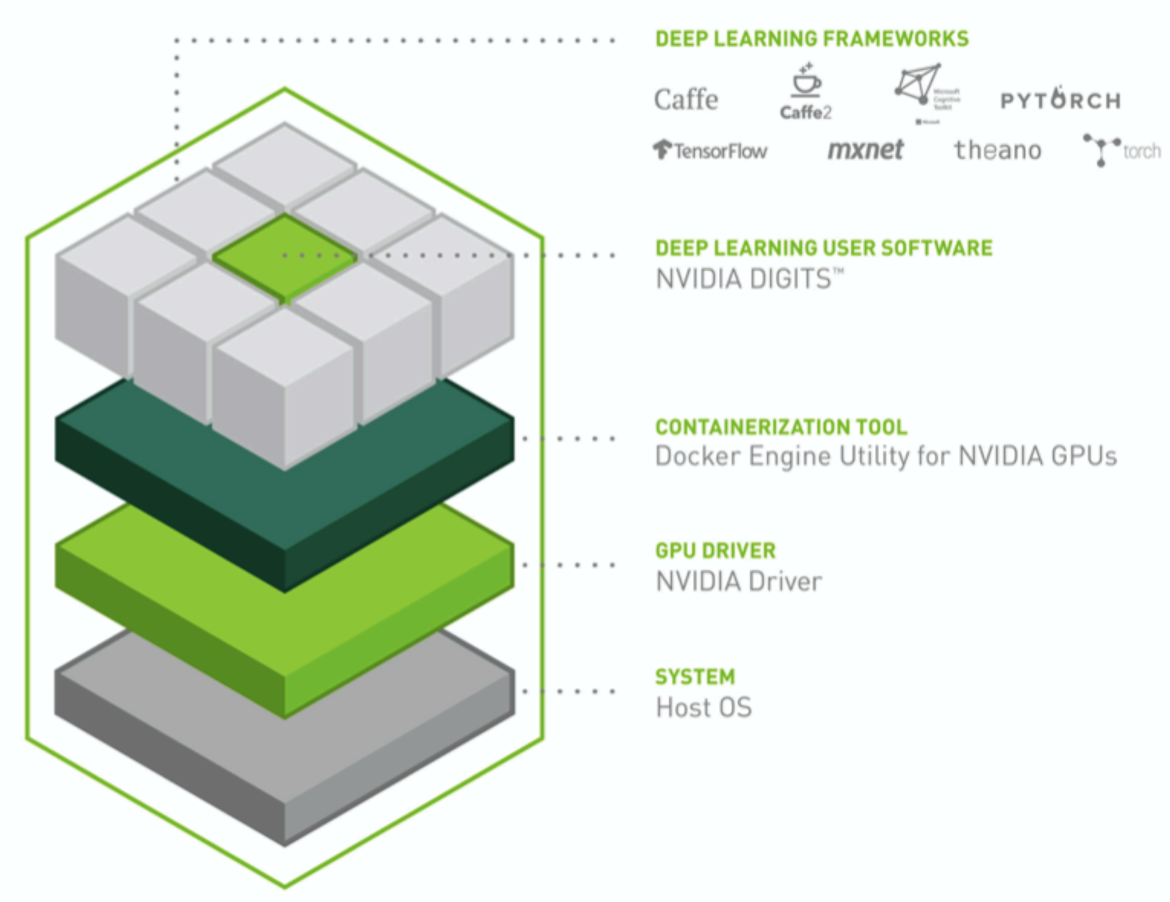
Multi-Instance GPU (MIG)



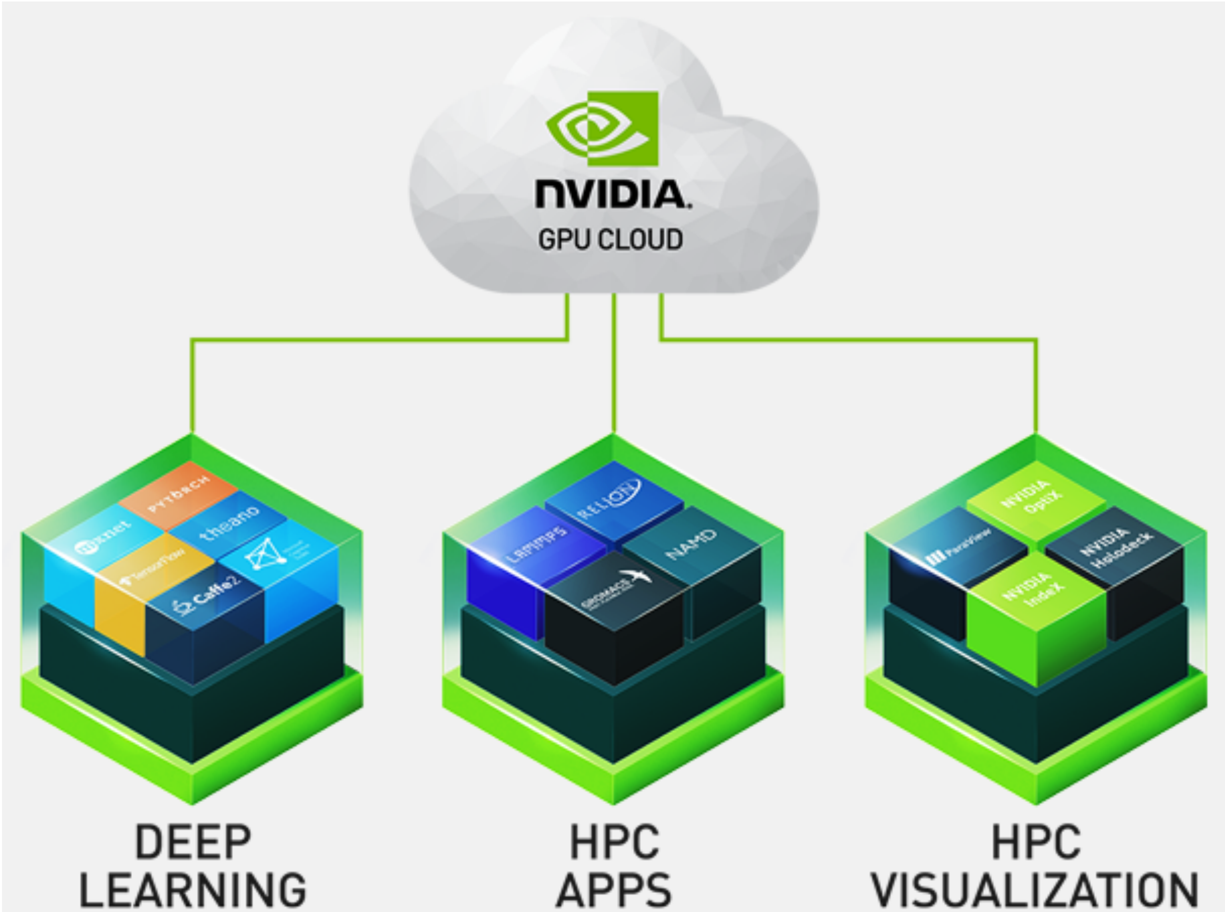
NVIDIA DGX Software Stack



The DGX-A100 Deep Learning Software Stack

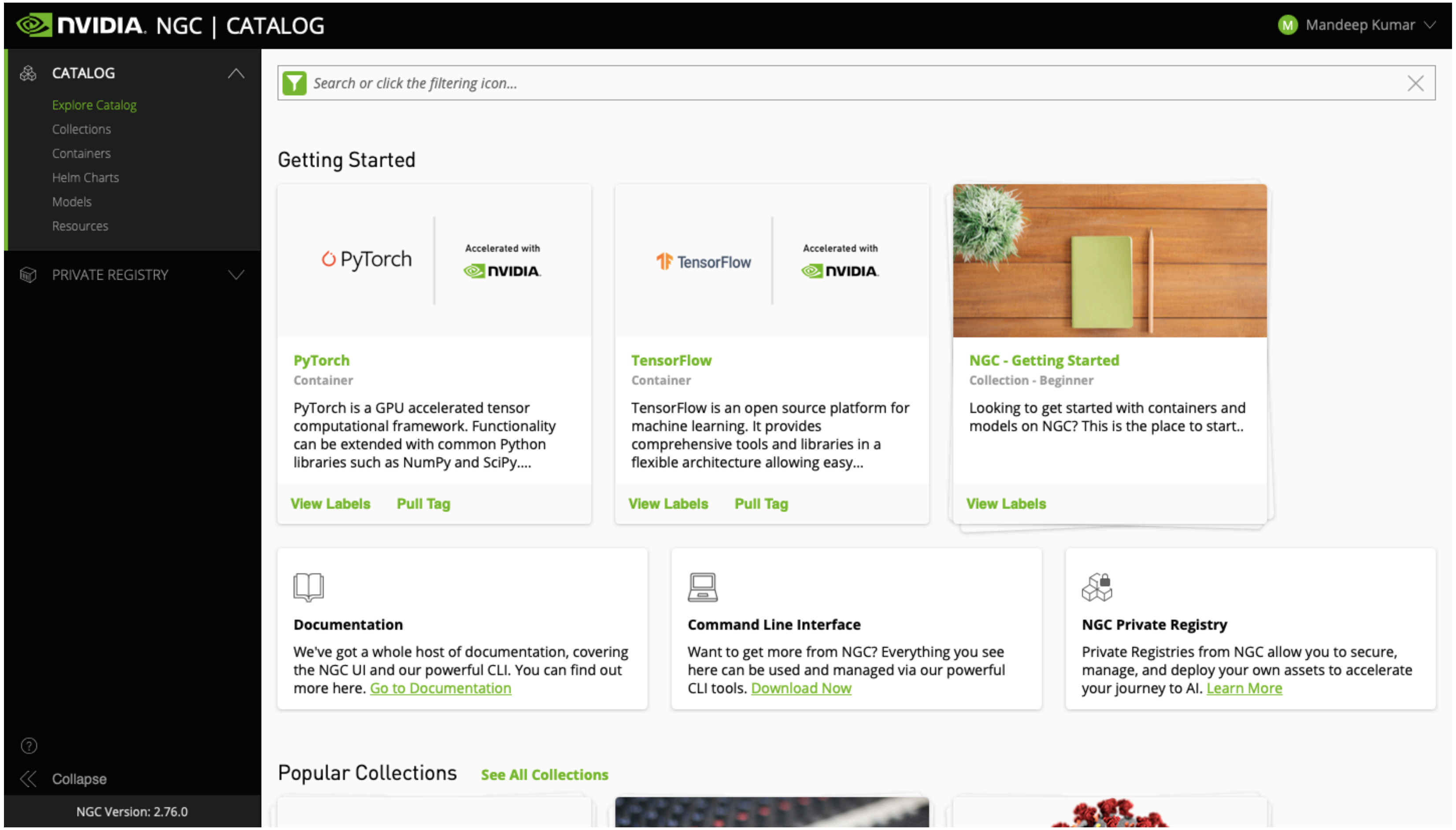


NVIDIA GPU Cloud (NGC)



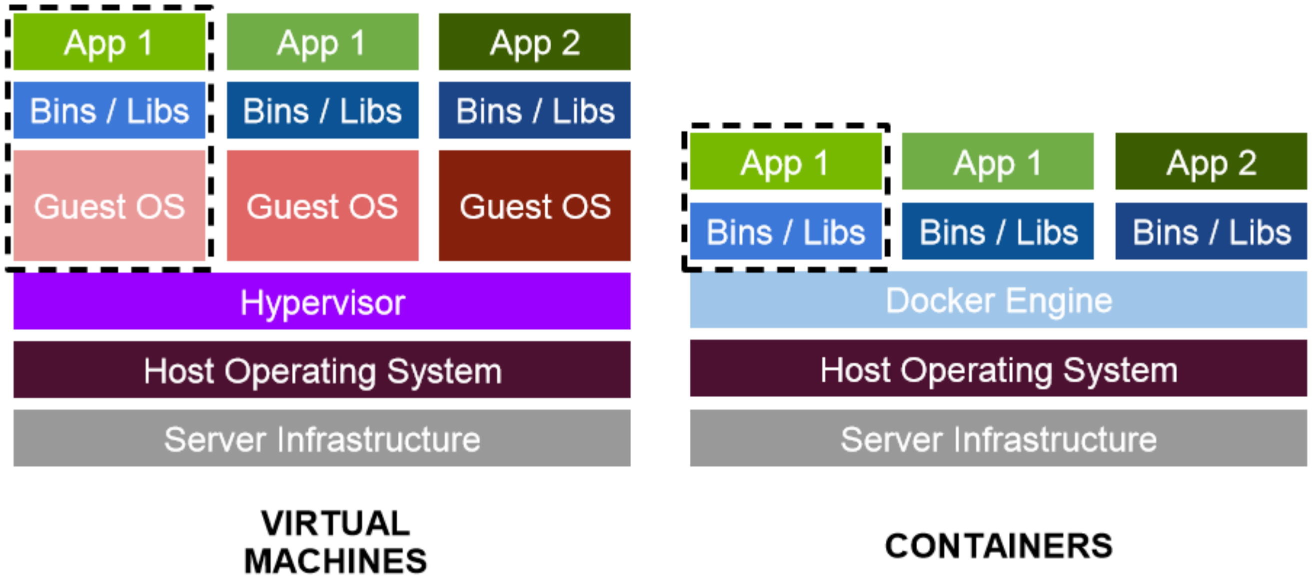
NVIDIA GPU Cloud (NGC)

https://ngc.nvidia.com/catalog



VIRTUAL MACHINE VS. CONTAINER

Not so similar



What’s Docker?

“an open-source project that automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux”

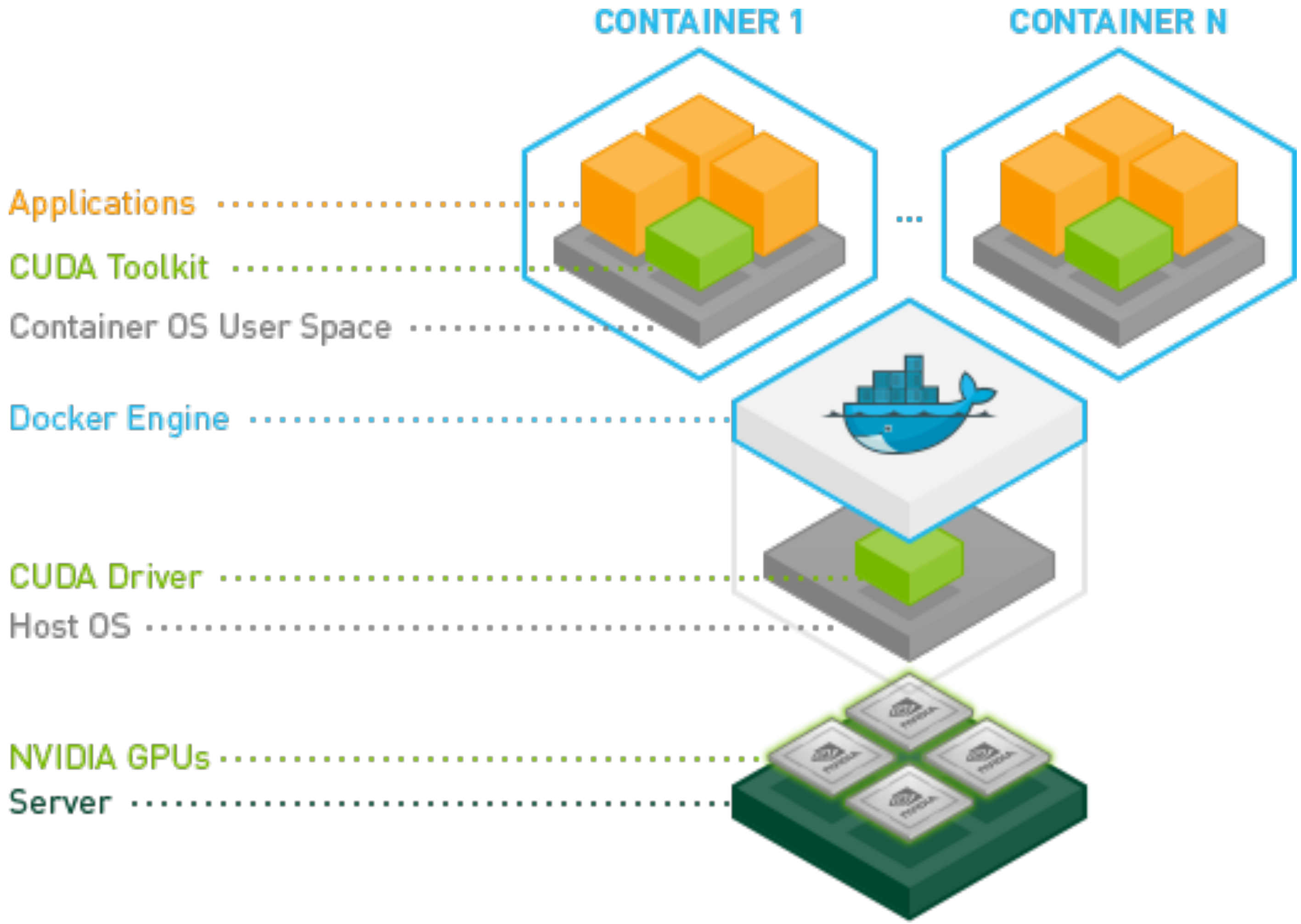
The key benefit of Docker:

- It allows users to **package an application with all of its dependencies into a standardized unit** for software development

Why NVIDIA Docker?

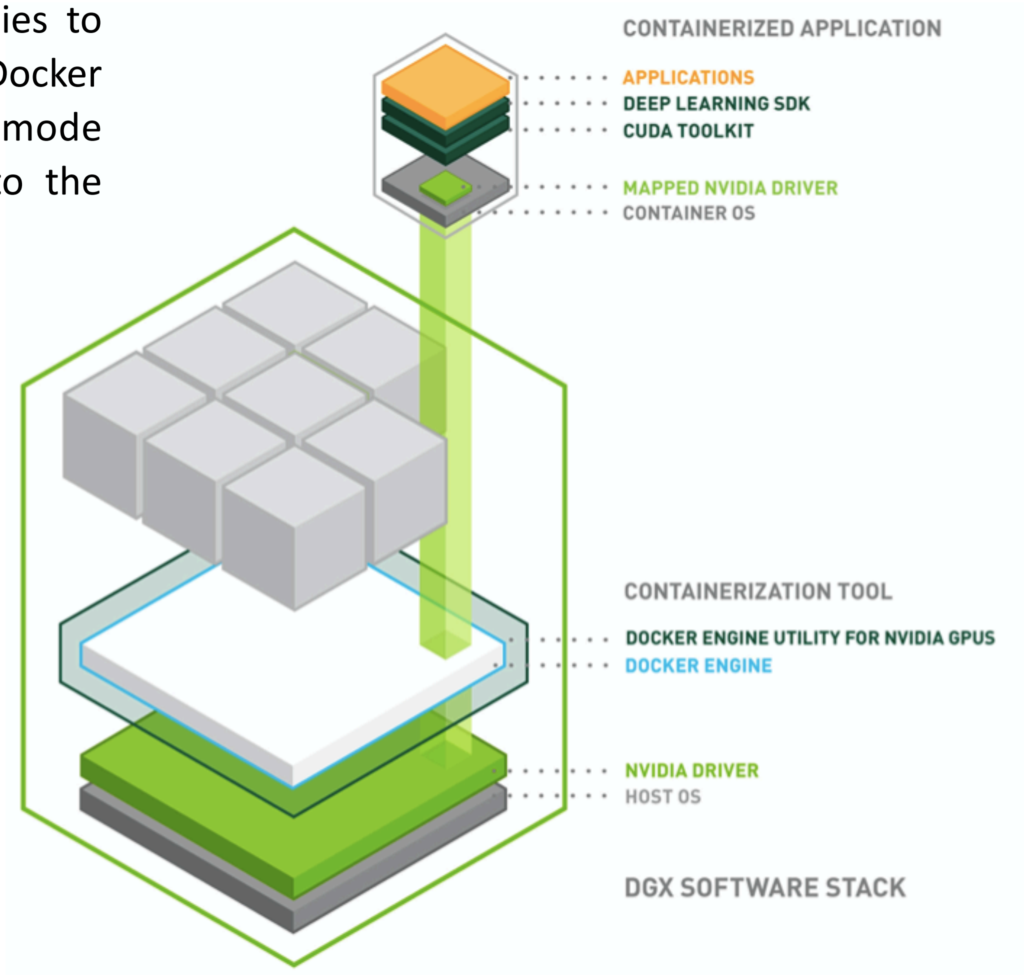
- Docker containers are hardware-agnostic and platform-agnostic
- NVIDIA GPUs are specialized hardware that require the NVIDIA driver
- Docker does not natively supported NVIDIA GPUs with containers
- nvidia-docker makes the images agnostic of the NVIDIA driver

NVIDIA Docker



NVIDIA Docker

Docker containers encapsulate application dependencies to provide reproducible and reliable execution. The Docker Engine Utility for NVIDIA GPUs maps the user-mode components of the NVIDIA driver and the GPUs into the Docker container at launch



NVIDIA Docker Sub-Commands

nvidia-docker pull

nvidia-docker images

nvidia-docker run

nvidia-docker ps

nvidia-docker exec

nvidia-docker commit

nvidia-docker logs

Running Containers

nvidia-docker run -it --rm --name <container_name> -u $(id -u):$(id -g) -p 8080:8888 --net=host -v local_dir:container_dir nvcr.io/nvidia/<framwork_name>:<xx.xx>

Docker run Options:

- **--rm** remove the container after it exits
- **-i -t** or **-it** interactive, and connect a “tty”
- **--name** give the container a name
- **-u $(id -u):$(id -g)** set the ID of the user in the container
- **-p 8080:8888** port map from host to container
- **--net=host** networking stack in the container
- **-v ~/data:/data** map storage volume from host to container (bind mount) i.e. bind the ~data directory in your home directory to /data in the container

Docker on HPC Systems

- HPC systems are shared resources
- Docker’s security model is designed to support trusted users running trusted containers; e.g., users can escalate to root
- Docker not designed to support batch-based workflows
- Docker not designed to support tightly-coupled, highly distributed parallel applications (MPI)
- No native support with open source workload managers

Overcome these Issues with Singularity

Singularity: A Container Engine for HPC

- Reproducible, portable, sharable, and distributable containers
- No trust security model: untrusted users running untrusted containers
- No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
- It automatically derived user's home directory; user can also bind other directories at runtime

Running with Singularity

Save the NGC container as a local Singularity image file:

singularity build <framwork_name>.sif docker://nvcr.io/nvidia/<framwork_name>:<xx.xx>

e.g,

singularity build tensorflow_21.12-tf2-py3.sif docker://nvcr.io/nvidia/tensorflow:21.12-tf2-py3

Run Singularity image file on NVIDIA GPU:

singularity run --nv --bind local_dir:container_dir <framwork_name>.sif <Container-Name>

e.g,

singularity run --nv /opt/sif/tensorflow_21.12-tf2-py3.sif yourcode.py

Containers (Singularity) with PBS Sample Script

TensorFlow Container with PBS Sample Script:

```
#!/bin/bash
#PBS -N testjob
#PBS -l select=1:host=dgx:ncpus=16:ngpus=1
#PBS -q dgx
#PBS -joe
cd $PBS_O_WORKDIR
/usr/local/bin/singularity run --nv /opt/apps/sif/tensorflow_21.12-tf2-py3.sif python -c 'import tensorflow as tf;
print(tf.__version__)'
```

PyTorch Container with PBS Sample Script:

```
#!/bin/bash
#PBS -N testjob
#PBS -l select=1:host=dgx:ncpus=16:ngpus=1
#PBS -q dgx
#PBS -joe
cd $PBS_O_WORKDIR
/usr/local/bin/singularity run --nv /opt/apps/sif/pytorch_21.12-py3.sif python -c 'import torch;
print(torch.__version__)'
```

Thanks!