Deploying Enterprise-Ready Generative AI on VMware Private AI

VMware AI/ML

Table of contents

Deploying Enterprise-Ready Generative AI on VMware Private AI

Introduction

Executive Summary

Generative artificial intelligence (GenAI), especially in the form of Large Language Models (LLMs), is at the forefront of technological innovation, offering human-like creativity, reasoning, and language understanding. Organizations across the globe recognize its potential, but implementing LLMs, particularly in regulated industries, brings about unique challenges. On-premises deployment in the private cloud offers a strategic solution, allowing organizations to retain complete control over data and comply with industry regulations. This fosters trust and transparency, ensuring that sensitive information and intellectual property are securely protected within enterprise environments.

IT organizations now can use VMware Private AI platform for running GenAI models. This platform for AI services enables privacy and control of corporate data, choice of open source and commercial AI solutions, quick time-to-value, and integrated security and management.

Utilizing VMware Private AI, we can democratize GenAI by igniting business innovation for all enterprises and providing the following advantages:

- Get the flexibility to run a range of AI solutions for your environment: NVIDIA, open–source, and independent software vendors.
- Deploy with confidence, knowing that VMware has partnerships with NVIDIA and other partners, all of whom are respected leaders in the high-tech space
- Achieve great performance in your model with vSphere and VMware Cloud Foundation's GPU integrations.
- Augment productivity by building private chatbots, eliminating redundant tasks, and building intelligent process improvement mechanisms.

This white paper serves as an essential guide, providing insights into the architecture design, implementation, and best practices for LLM fine-tuning and inference. By embracing GenAI through the on-premises deployment of LLMs on VMware infrastructure, enterprises can unlock the full potential of this revolutionary technology in a responsible and ethical manner, significantly boosting innovation and driving sustainable growth.

Document Scope and Objectives

This white paper offers readers a detailed reference architecture that leverages VMware Private AI for GenAI workloads. A key focus of the white paper is to provide effective guidance on preparing, deploying, and automating virtual infrastructures fit for LLM fine-tuning and inference tasks. These tasks normally require the availability of hardware accelerators, such as GPU and Remote Direct Memory Access (RDMA) devices either from Virtual Machines (VMs) or containers. For that reason, we dedicate a big portion of this document to guide the reader through the integration of hardware accelerators and VMware environments.

Furthermore, the white paper presents examples of LLM fine-tuning and inference as the means for testing and validating the reference architecture deployment. The goal is to establish a scalable, high-performance, and production-ready architecture for GenAI tailored to meet the demands of enterprise-level applications.

Whether you are a seasoned AI practitioner or just beginning your journey of GenAI, this white paper serves as a technical resource, providing step-by-step instructions, best practices, and real-world insights that help you run GenAI projects on top of VMs or Tanzu Kubernetes hosted by VMware Cloud Foundation and vSphere with Tanzu. By following the guidelines presented in this paper, you can confidently embark on the path toward developing cutting-edge GenAI solutions that drive innovation and success in your organization.

Architecture Design

High-level Architecture Overview

A high-level overview of the solution is depicted below starting with the infrastructure components up to the LLM application layer.

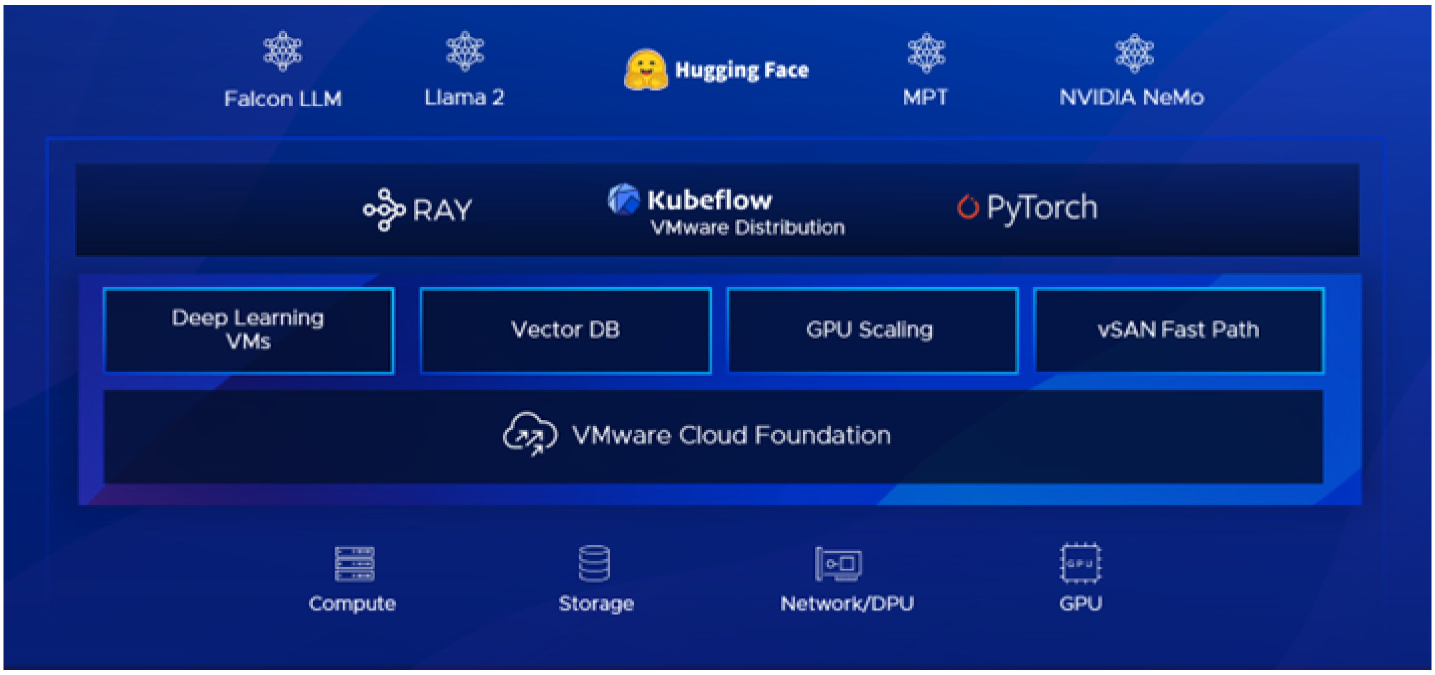


Figure 1: Solution High-Level Architecture Overview

Figure 1 illustrates the stacking of the software components used in LLM Operations starting from the AI and MLOps Platform and all the way up.

Customers can leverage the solution for two primary LLM workflows – customization (fine-tuning, prompt-tuning, and others) and inference at scale. Both workflows demand more computing capacity than traditional deep learning workloads regarding LLMs. Inference also requires GPU resources depending on application needs or number of users. By leveraging VMware Cloud Foundation and distributed computing frameworks like Ray, unused resources in an environment can be repurposed for machine learning (ML) workflows. This improves infrastructure utilization and boosts productivity for ML overall.

VMware Private AI Overview

VMware Private AI is an architectural approach that aims to balance the business gains from AI with the practical privacy and compliance needs of the organization. VMware Private AI is architected with industry partners to create a solution for AI services that ensures privacy and control of corporate data, choice of open source and commercial AI solutions, quick time-to-value, and integrated security and management. The optimized infrastructure stack comprising of VMware Cloud Foundation provides an optimized infrastructure stack for delivering Private AI within the Enterprise.

VMware Cloud Foundation Overview

Integrating LLM with VMware Cloud Foundation set the scene for a new era of capabilities and opportunities for enterprises. VMware Cloud Foundation is a turnkey platform that stands out as an ideal choice for deploying AI workloads due to its integrated infrastructure management, which simplifies complex tasks. The platform ensures consistency across various cloud environments, facilitating seamless AI workload deployment. Its scalability capabilities meet the resource demands of AI tasks, while resource efficiency features optimize infrastructure utilization. The platform's automated lifecycle management streamlines updates and upgrades, minimizing disruptions to AI operations. Furthermore, the platform's integration with AI-specific hardware accelerators further amplifies performance potential.

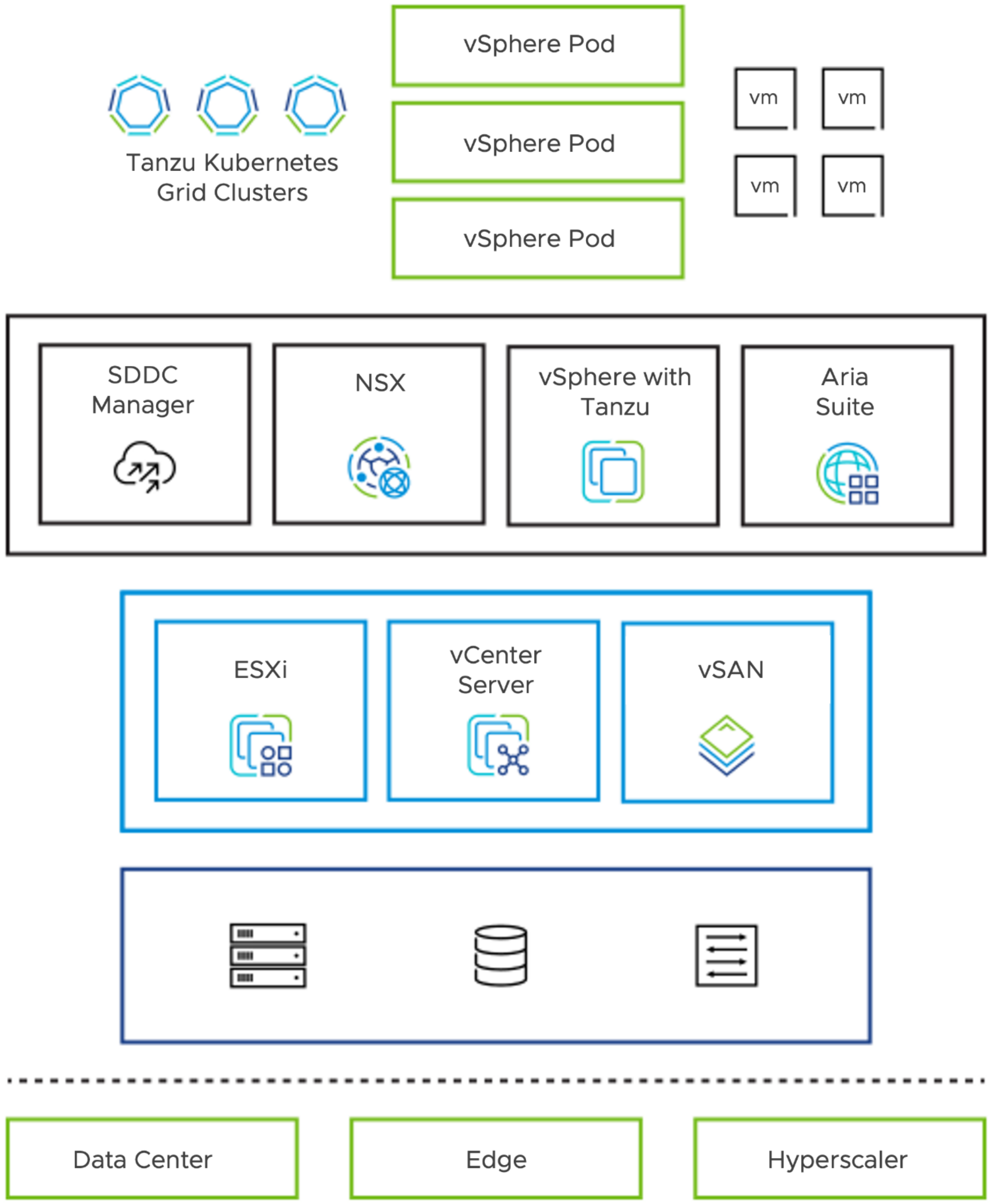


Figure 2: VMware Cloud Foundation Architecture

VMware Cloud Foundation for AI workloads yields benefits such as simplified management, enhanced scalability, improved security, and optimized resource usage, positioning organizations to harness the full potential of artificial intelligence within a resilient and efficient infrastructure.

VMware vSphere with Tanzu Overview

VMware vSphere with Tanzu, transforms traditional virtualization infrastructure into a robust platform for containerized workloads. VMware Tanzu Kubernetes Grid™ facilitates the creation and management of Tanzu Kubernetes Cluster (TKC) natively within vSphere, seamlessly integrating Kubernetes capabilities with the reliable features of vSphere. With vSphere's networking, storage, security, and VMware vSphere High Availability (vSphere HA) features, organizations achieve better visibility and operational simplicity for their hybrid application environments.

Table 1: VMware vSphere AI Capabilities

vSphere with Tanzu - AI/ML Capabilities	**Description**
NVIDIA NVSwitch support	For up to 8 GPUs per host, vSphere now supports the deployment of NVIDIA NVSwitch technology, improving large-size AI/ML workload performance by leveraging GPU to GPU direct communication. All 8 GPUs or a subset of them, can be allocated to a single VM.
Device groups	In VMware Cloud Foundation 5.0, device groups enable VM to group certain hardware devices simpler like Network Interface Card (NIC) and GPU devices to enhance the performance. Compatible vendor device drivers are required and subject to vendor release. NVIDIA will be the first partner supporting device groups with upcoming compatible drivers.
Simplified hardware consumption with device groups	Device groups leverage the existing **Add New PCI Device** to VM workflows. VMware vSphere Distributed Resource Scheduler™ (DRS) and vSphere HA are aware of device groups and will place VMs appropriately to meet the device groups' requirements.
Heterogeneous vGPU profile	vSphere reduces cost by improving GPU utilization and reducing workload fragmentation in GPUs by the addition of support for heterogenous vGPU profiles on the same GPU. This capability allows different types of workloads to be configured with different GPUs, such as VDI applications, compute applications, graphics applications.

vSphere with Tanzu also enables organizations to run application components in containers or VMs on a unified platform, streamlining workload management and deployment. By converging containers and VMs, IT teams can leverage the benefits of both paradigms while maintaining a cohesive and scalable infrastructure.

NVIDIA AI Enterprise Overview

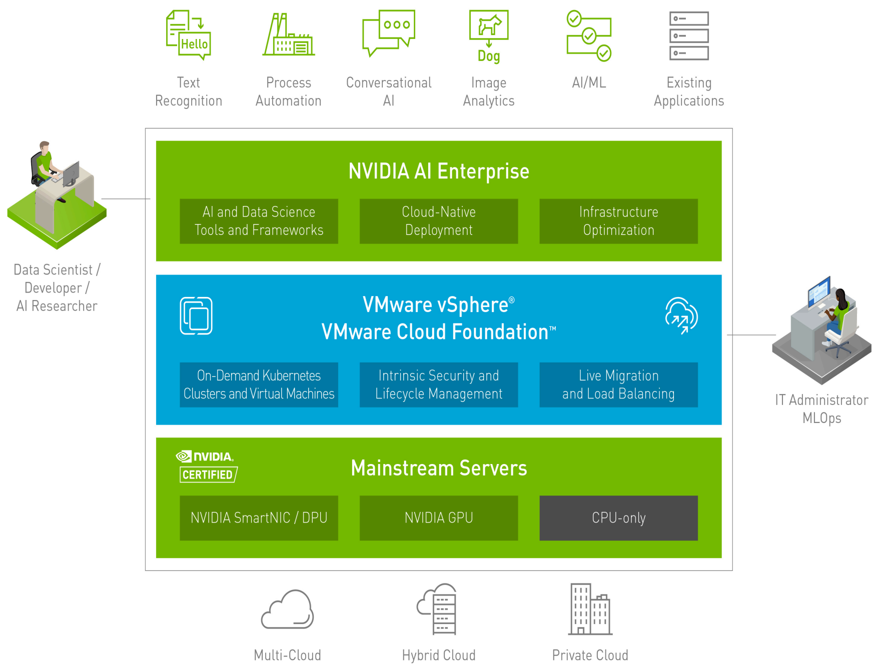


Figure 3: NVIDIA AI Enterprise on VMware Cloud Foundation with Tanzu and VMware vSphere with Tanzu

NVIDIA AI Enterprise (NVAIE) is a robust suite designed to boost AI workloads in various industries. In partnership with VMware, NVAIE delivers virtual GPUs (vGPU) and Multi-Instance GPUs (MIG) for optimal performance of AI workloads running on VMware Cloud Foundation and Tanzu Kubernetes Grid. NVAIE's vGPU technology allows multiple VMs or containers to share one physical GPU, promoting efficient resource utilization in VMware setups. Furthermore, its MIG feature partitions a single GPU into isolated fractional instances that allow the execution of simultaneous AI tasks. vGPU and MIG offer superior resource allocation and scalability for AI tasks on VMware Cloud Foundation.

GenAI Stack Overview

LLMs have revolutionized natural language processing tasks, enabling machines to understand, generate, and interact with human language in a remarkably human-like manner. These models, such as GPT-4, MPT, Vicuna and Falcon have gained popularity due to their ability to process vast amounts of text data and produce coherent and contextually relevant responses. Behind the scenes, these models rely on intricate operations components and processes that work together harmoniously to achieve their impressive capabilities. The main elements of LLM operations can be summarized as follows:

- **Deep Learning (Transformers) Neural Nets**: LLMs are built upon complex neural network architectures based on the transformer architecture. These models consist of multiple layers of self-attention mechanisms and feed-forward neural networks with billions or neurons and parameters that need to get trained over terabytes of data.
- **Hardware accelerators:** LLMs are very computationally demanding and require specialized hardware to achieve optimal performance. LLM training and inference processes often rely on high-performance GPUs, RDMA networking and high-speed storage to handle immense computational loads. Most of this reference architecture provides detailed instructions on how to set up these acceleration technologies from VMware Cloud Foundation and Tanzu Kubernetes so you can run LLM workloads on VMs and containers.
- **ML software stack**: Multiple open-source software choices provide tools to work with LLMs and GenAI. For this reference architecture, we have strived to make sound selections of open-source ML software that can help our customers in their journey to adopt AI as a central piece of their app development. The main open-source ML software components used in this document are:
    - Hugging Face Transformers. Hugging Face (HF) is a popular platform where the ML community collaborates on models, datasets, and applications. They are authors of one of the most adopted open-source PyTorch implementations of the NLP Transformers architecture. In our examples, we use many of their technologies, such as Transformers (models and the SFT Trainer), Accelerate for multi-GPU training, and parameter-efficient fine-tuning (PEFT), which dramatically simplifies the LLM fine-tuning process.
    - Ray Serve. A parallel computing platform that lets you serve ML models (in real-time or batch) using a simple Python API. In our examples, we use a combination of Ray Serve and vLLM to deploy LLM-based prompt completion services automatically scaled up and down according to user demand.
    - Kubeflow is an end-to-end ML platform for Kubernetes; it provides components for each stage in the ML lifecycle, from exploration to training and deployment. The Kubeflow on vSphere project provides codes and documents to enable Kubeflow better running on VMware vSphere and VMware Cloud.
- **Pre-training tasks:** The first stage of LLM development involves pre-training on massive amounts of text data from the internet. During this phase, the model learns to predict the next word in a sentence given the context of the preceding words. This process helps the model build a foundation of language understanding and grammar. The Hugging Face Models repository provides access to over 285k language and computer vision ML models that can be used for many types of tasks. Pre-training LLMs is a difficult and expensive task. For instance, pre-training the Falcon-40B LLM required 384 x A100 40GB GPUs running in P4d (AWS) instances and took over 2 months. Given the high cost and complexity of pre-training tasks, it is more convenient to leverage an open-source pre-trained LLM that the licensing permissions (commercial, research, and others.) and the use cases you have in mind.
- **Fine-tuning tasks:** After pre-training, the model can be fine-tuned on specialized datasets for specific tasks. This process adapts the general language model to perform more specialized tasks such as text generation, translation, sentiment analysis, or question-answering. Fine-tuning is crucial to tailoring the model's capabilities to the desired application. At the later sections of this document, we provide a complete example of how to fine-tune the Falcon 40B and 7B LLMs.
- **Inference (prompt completion) tasks:** After the LLM is pre-trained and fine-tuned, it enters the inference stage, where it processes users' prompts and generates completions in real-time. The model utilizes the previously learned information to make predictions and generate coherent and contextually relevant text. Later in this document we provide a full example on how to serve prompt completions using vLLM running on Ray Serve cluster.
- **Ethical Considerations:** As LLMs become more powerful, ethical concerns about their potential misuse and bias have gained prominence. Efforts are being made to address issues related to fairness, transparency, and responsible AI practices in language model deployment. For more information, refer to VMware's point of view on Trustworthy AI.

Physical Infrastructure Design

When it comes to AI/ML workloads, the hardware infrastructure requirements can vary depending on the specific task, dataset size, model complexity, or performance expectations. However, there are some general recommendations for hardware infrastructure for AI/ML workloads shown in the following table.

Table 2: Physical Infrastructure Design

Category	Hardware	Description	Example of Optimal Configuration (Based on NVIDIA DGX)
CPU	Intel VMware Compatibility Guide - Intel Xeon	Latest Intel Xeon 4th Gen (Sapphire Rapids) preferable, 3rd Gen (Ice Lake) acceptable, with a balance between CPU Frequency and # of cores. Latest Intel gen offers advanced features related to AI/ML such as Intel AMX (Advanced Matrix Extensions), support to DDR5 and CXL (Compute Express Link). Use Peripheral Component Interconnect Express (PCIe) Gen5 (preferable) for faster interconnects, PCIe Gen4 acceptable.	2 x Intel Xeon 8480C (Sapphire Rapids) PCIe Gen5 CPU with 56 cores
	AMD EPYC VMware Compatibility Guide - AMD EPYC	Latest AMD EPYC 4th Gen (Genoa) preferable, 3rd Gen (Milan) acceptable with a balance between CPU Frequency and # of cores. EPYC CPUs offer a high core count, exceptional memory bandwidth, and support for multi-socket configurations. They are suitable for both AI/ML and LLM workloads. Use PCIe Gen5 (preferable) for faster interconnects, PCIe Gen4 (acceptable).	2 x AMD EPYC 9554 (Genoa) PCIe Gen5 CPU with 64 cores – This is a comparable CPU to Intel 8480C specs
Memory	DDR5	Faster memory with higher bandwidth can reduce data transfer bottlenecks and enable faster access to the large datasets involved in AI/ML tasks. Additionally, the increased memory density provided by DDR5 allows for larger models and more extensive training datasets to be stored in memory, which can improve the overall performance and efficiency of AI/ML algorithms.	2TB RAM per node, depending on the configuration
GPU	NVIDIA: H100, A100, A40, L40, L40s VMware Compatibility Guide - GPUS	NVIDIA GPUs with compute capacity greater or equal to 8.0 are essential for LLM training. The support for bfloat16 in these GPUs balances precision and range, aiding in training neural networks efficiently without losing accuracy. NVLink enables efficient GPU-to-GPU communication and memory sharing, while NVSwitch enables large-scale GPU collaboration across multiple servers, facilitating the training and deployment of advanced AI models on massive datasets.	8 x NVIDIA H100 GPUs (80GB) for models above 40B parameters 4 x NVIDIA H100 GPUs (80GB) for models less than 40B parameters
Block Storage	VMWare vSAN VMware vSAN Hardware Quick Reference Guide	VMware vSAN with All Flash configuration delivers high performance, flexibility, and resiliency required for dynamic AI/ML workloads, while simplifying management through policy-driven operations.	For more information, refer to: • Design Considerations for Storage Controllers in vSAN • Design Considerations for Flash Caching Devices in vSAN • Consider NVMe or SAS SSD for vSAN Capacity devices • vSAN Storage Design Considerations
Network Adapter	Management Network VMware Compatibility Guide - NICs	10 Gb/s onboard NIC with RJ45 25 Gb/s or above Ethernet NIC Host baseboard management controller (BMC) with RJ45.	NIC: Broadcom 57504, Mellanox ConnectX-4 or Intel similar products Switch: Broadcom StrataXGS Switch Solutions BCM56080 Series or similar products
	Workload Network VMware Compatibility Guide – NICs with SRIOV and RoCE v2	LLM inference and fine-tuning within a single host is compatible with standard 25 Gb Ethernet. For fine-tuning models larger than 40B parameters, efficient multi-node communication requires low latency and necessitates 100 Gb/s or higher RDMA network (for example, Ethernet or InfiniBand) for optimal performance.	RoCE NIC: Broadcom 5750X or Mellanox ConnectX-5/6/7 or similar products RoCE Switch: Broadcom StrataXGS Switch Solutions (Trident4-X11C/BCM56890 Series) or similar products InfiniBand Host Channel Adapter (HCA): Mellanox ConnectX-5/6/7 VPI InfiniBand Switch: NVIDIA QM9700
External File or Object Storage	NAS/Object Storage VMware Compatibility Guide – NAS	For inferencing, it might be possible to store the model on a local storage. For fine tuning larger models, it might be necessary to use NAS or SAN or object storage solutions.	File storage or object storage that delivers more than 8 GB/s write and 40 GB/s read.

Keep in mind that the configuration above provides optimal configuration for training LLMs, which matches with NVIDIA DGX solutions. The requirements for your environment might be different. Consult your *OEM* to determine the proper solution.

Network Design

Figure 4 presents the network design of the GenAI deployment on VMware Cloud Foundation, which is segmented into two domains: the management domain and the workload domain.

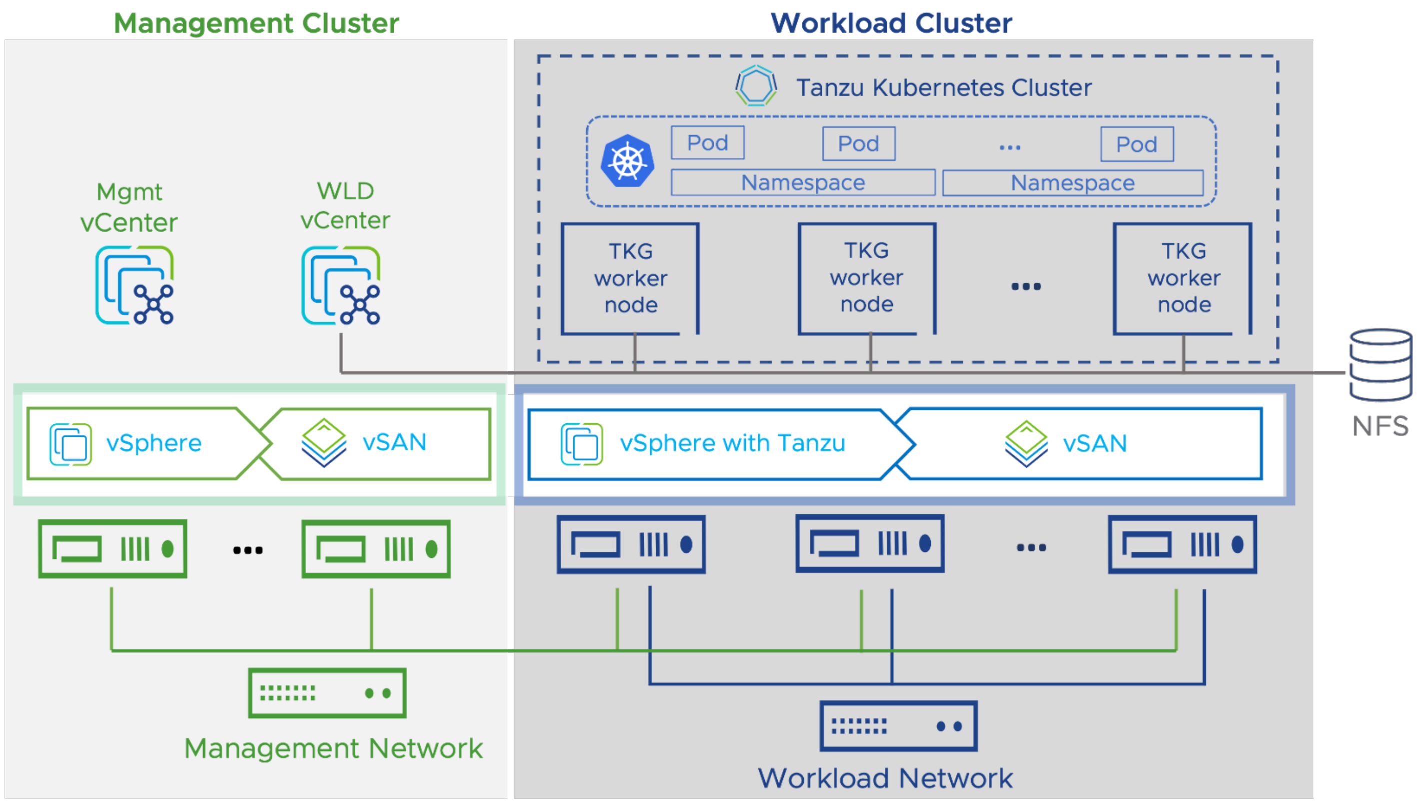


Figure 4: Network Design

The workload domain manages the ESXi hosts within the workload domain where the VMware Tanzu Supervisor Cluster resides, facilitating TKCs on top.

For the management network, each network type below in Table 3 is associated with a specific VLAN:

Table 3: Management Network

Network Type	Description	Comments
ESXi Management	ESXi management traffic	At least 1Gb/s
vSAN	vSAN traffic	vSAN OSA: at least 10Gb/s; vSAN ESA: at least 25 Gb/s
vMotion	vMotion traffic	At least 10Gb/s
VM	VM-to-VM traffic	Depends on use case or requirements
Tanzu Management	Supervisor Cluster traffic	This network can either be shared with the ESXi management port group or kept separate.

Given the architecture covering only on inference and fine-tuning, the management network's aggregate bandwidth at 25 Gbps is deemed satisfactory. We recommend using Network I/O control to reserve network bandwidth for the above different services. This approach helps mitigate potential conflicts or disruptions caused by other VMware services or VM traffics.

The workload network that resides on the workload cluster, is configured with dedicated switch and network adapters for optimal performance.

Table 4: Workload Network

Network Type	Description	Comments
Tanzu Frontend	It provides load balanced virtual IP Ranges for TKCs	This network can either be shared with the Tanzu workload network or kept separate.
Tanzu Workload	It provides connectivity to TKCs in the vSphere Namespaces.	• For LLM inferencing tasks and fine-tuning tasks within a single host, leverage an existing network infrastructure featuring 25 Gb Ethernet is sufficient to accommodate the bandwidth requirements of textual data. • For the requirement to fine-tune larger models with more than 40B parameters among GPUs across different nodes, the substantial demand for information exchange (including weights) necessitates the adoption of RDMA networking (RoCE/InfiniBand) with 100 Gb or higher bandwidth for optimal performance.

For more information about how to design the vSphere with Tanzu infrastructure, refer to: Developer Ready Infrastructure for VMware Cloud Foundation.

Note: During the installation of NICs or HCAs on servers, it is crucial to consider PCIe generation and lane compatibility on the servers' motherboard. This alignment ensures optimal data transfer speed for the devices. Neglecting this alignment can result in suboptimal performance.

Delving into storage, GenAI models can be sizable, with billions of parameters and intermediate outputs. Thus, the models require large and shared storage capacity. The need for external storage for AI model inference depends on the specific requirements and characteristics of the AI model or the deployment environment.

- For inferencing, external storage is not strictly required, as the models reside in GPU memory. However, external storage can be used as a repository for models, model versioning and management, model ensembles, and for storage and archival of inference data.
- For fine-tuning tasks, it is recommended to use a dedicated storage network to connect to a robust external storage to store the training data and intermediate outputs during training. This network helps to optimize data access and reduce latency for storage operations, which can offer the scale and speed necessary for operationalizing AI models, providing a foundational component for AI workflow.

Virtual Infrastructure Design

Designing a VMware virtual infrastructure requires consideration of several factors to ensure optimal availability, manageability, performance, recoverability, and security (AMPRS), which are the key design factors of VMware's design methodology.

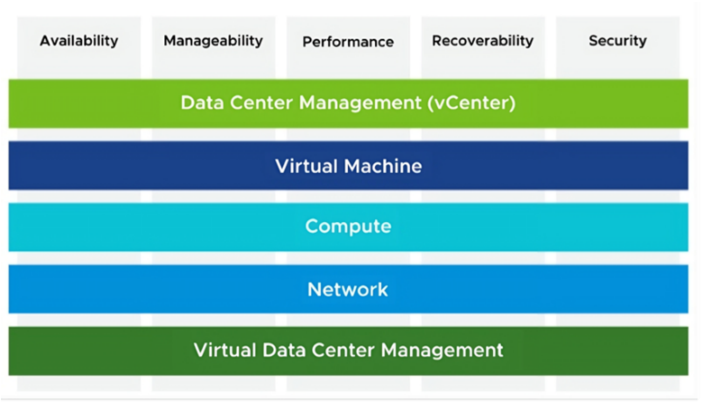


Figure 5: Virtual Infrastructure Design

Refer to our existing documentation, design guides and VMware Validated Solutions to conduct a design process that follows VMware's best practices, design decisions and operational procedures. For more information, see the following links VMware Validated Solutions and Cloud Platform Tech Zone.

Tanzu Kubernetes Grid Design

Tanzu Kubernetes Grid is a powerful and versatile solution that empowers organizations to seamlessly deploy and manage Kubernetes clusters on top of VMware vSphere. Tanzu Kubernetes Grid offers a consistent and scalable Kubernetes experience, simplifying the complexities of container orchestration, and enabling enterprises to focus on delivering value through their AI/ML workloads. Tanzu Kubernetes Grid provides a robust foundation for running AI/ML workloads at scale, leveraging Kubernetes' strengths in orchestrating distributed and resource-intensive applications.

In the AI/ML space, Tanzu Kubernetes Grid plays a critical role by providing a reliable and resilient platform for deploying and scaling AI/ML workloads and applications. With the explosive growth of data and the increasing demand for sophisticated AI-driven solutions, organizations require a stable and efficient infrastructure to process and analyze large amounts of information. Tanzu Kubernetes Grid ensures that AI/ML workloads can be seamlessly deployed and managed on Kubernetes clusters, taking advantage of Kubernetes' advanced capabilities for automating resource allocation, load balancing, and fault tolerance.

Verify the Tanzu Kubernetes Grid documentation before designing and implementing your solution. Understanding the Tanzu Kubernetes Grid design, planning, and implementation documentation is crucial as it provides a comprehensive understanding of its functionality and deployment best practices. The documentation outlines Tanzu Kubernetes Grid concepts, architecture components, security, deployment options, maintenance, and troubleshooting. Understanding the architecture helps in customizing and optimizing Tanzu Kubernetes Grid, ensuring security measures are in place, planning upgrades and migrations, and facilitating smooth integrations with other services.

The following tables provide information about VM classes used for this reference architecture as well as how Tanzu Kubernetes Grid nodes were configured.

Table 5: VM Class Configuration

VM Class Name	**Type**	**CPU**	**RAM**	**GPU**	**NIC**
guaranteed-large	Pre-defined	4	16GiB	-	-
gpu-xlarge-vgpu-vf	Customized	72	512GiB (100% Reservation)	2x NVIDIA A100 40GB	NVIDIA ConnectX-6 Virtual Function (VF)
single-vm-gpu	Customized	32	256GiB (100% Reservation)	2x NVIDIA A100 40GB	-

Table 6: Tanzu Kubernetes Grid Node Configuration

		tanzu-storage-policy		

Check out the *YAML* file to deploy TKC in this reference architecture environment.

Deployment and Configuration

Resource Planning

Hardware Resources

The following table demonstrates an example of the hardware infrastructure used in the reference architecture, our validation used four Servers and the server's specification is listed in the following table.

Table 7: Hardware Resources

Component	**Description**
CPU	2 x Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
RAM	768GB
GPU	2 x NVIDIA A100 40GB PCIe
Network	2 x Intel(R) Ethernet Controller X710 for 10GbE SFP+ (Management), 1 x NVIDIA ConnectX-6 Dx (Workload)
Storage	2 x Disk Groups, each Disk Group: • 1 x Cache Disk: 800GB WRITE-INTENSIVE SAS SSD • 6 x Capacity Disks: 960GB CLASS E SAS SSD

The hardware infrastructure requirements for AI/ML workloads are dynamic and dependent on several factors, including the nature of the task, the size of the dataset, the complexity of the model, and the desired performance levels. There is no one-size-fits-all solution when it comes to AI/ML hardware, as different tasks and projects may demand unique configurations. Organizations and individuals involved in AI/ML work should carefully analyze their specific requirements to determine the most suitable hardware setup, considering the computational power, memory capacity, storage capabilities, and networking resources necessary to achieve optimal results. Flexibility and scalability in hardware choices are crucial to meet the evolving demands of AI/ML workloads efficiently.

Software Resources

The table below demonstrates the software resources we used in our validation.

Table 8: Software Resources

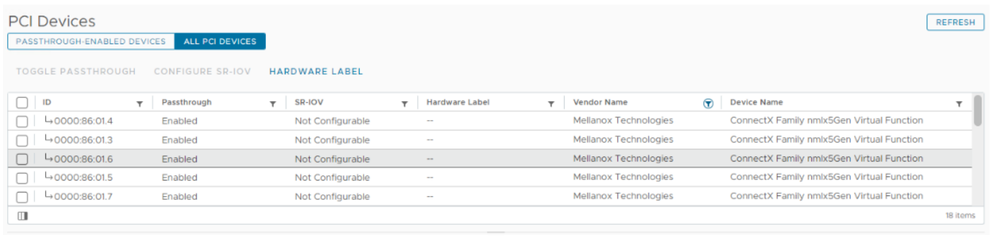
Software	**Purpose**	**Version**
VMware Cloud Foundation	Efficiently manage VM and container workloads at scale. Deliver cloud benefits to on-premises, full-stack hyperconverged infrastructure (HCI) deployments.	5.0 *We upgraded vCenter Server to 8.0 Update 1b by following this doc to avoid a known issue with vGPU and vMotion.
Tanzu Kubernetes Release (TKR)	A Tanzu Kubernetes release provides the Kubernetes software distribution signed and supported by VMware	v1.24.9---vmware.1
NVAIE for vSphere	NVAIE for vSphere includes: • The vGPU Manager vSphere Installation Bundles (VIBs) for VMware vSphere • Guest Driver for Single VM	525.105.14
NVAIE GPU Operator	The GPU Operator allows administrators of TKC to manage GPU nodes.	v23.3.1
NVAIE Network Operator	The NVIDIA Network Operator simplifies the provisioning and management of NVIDIA networking resources in a TKC.	v23.5.0
Mellanox Firmware Tools (MFT)	Manage the firmware of Mellanox NIC/HCA; for examples, enable SRIOV and Priority Flow Control	4.24.0

Hardware Configuration Considerations

Global SR-IOV

Single Root I/O Virtualization (SR-IOV) is a specification that allows a single PCIe physical device under a single root port to appear as multiple separate VFs to the hypervisor or the guest operating system.

SR-IOV is required for this reference architecture to enable VFs from a NIC/HCA and is also required for both MIG and Time-Slicing modes for vGPU. For MIG, a vGPU is associated with a VF at boot time for that VM resource. Here is an example of the available VFs for a Mellanox ConnectX-6 NIC:

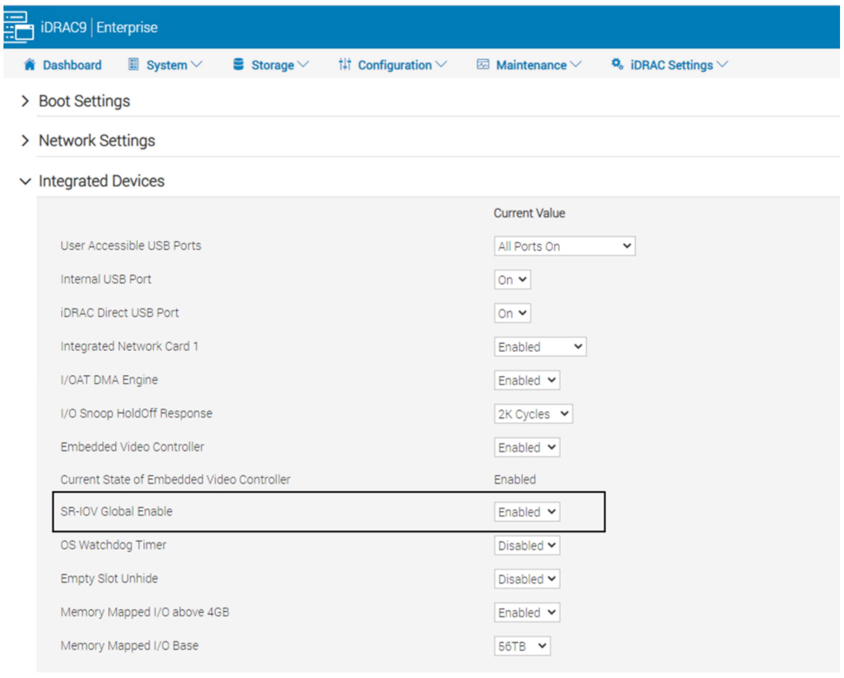


ID	Passthrough	SR-IOV	Hardware Label	Vendor Name	Device Name
↳0000:86:01.4	Enabled	Not Configurable	--	Mellanox Technologies	ConnectX Family nmlx5Gen Virtual Function
↳0000:86:01.3	Enabled	Not Configurable	--	Mellanox Technologies	ConnectX Family nmlx5Gen Virtual Function
↳0000:86:01.6	Enabled	Not Configurable	--	Mellanox Technologies	ConnectX Family nmlx5Gen Virtual Function
↳0000:86:01.5	Enabled	Not Configurable	--	Mellanox Technologies	ConnectX Family nmlx5Gen Virtual Function
↳0000:86:01.7	Enabled	Not Configurable	--	Mellanox Technologies	ConnectX Family nmlx5Gen Virtual Function

18 items

Figure 6: Virtual Functions in vSphere

Follow your server vendor's documentation on how to enable SR-IOV at the BIOS/UEFI level, here is an example on setting Global SR-IOV to enabled through Dell's iDRAC:



	Current Value
User Accessible USB Ports	All Ports On
Internal USB Port	On
iDRAC Direct USB Port	On
Integrated Network Card 1	Enabled
I/OAT DMA Engine	Enabled
I/O Snoop HoldOff Response	2K Cycles
Embedded Video Controller	Enabled
Current State of Embedded Video Controller	Enabled
SR-IOV Global Enable	Enabled
OS Watchdog Timer	Disabled
Empty Slot Unhide	Disabled
Memory Mapped I/O above 4GB	Enabled
Memory Mapped I/O Base	56TB

Figure 7: Enable Global SRIOV in BIOS

Advanced GPU Technologies: NVIDIA GPUDirect RDMA, NVLink, and NVSwitch

The combination of GPUDirect RDMA, NVLINK, and NVSwitch plays a pivotal role in enhancing GenAI. GPUDirect RDMA enables direct GPU-to-GPU memory access through network devices, efficiently reducing latency and amplifying data sharing capabilities. NVLINK serves as a high-speed, low-latency bridge between GPUs, particularly valuable in managing extensive datasets. NVSwitch orchestrates communication in multi-GPU configurations, offering a crucial foundation for scaling GenAI.

GPUDirect RDMA on VMware vSphere

GPUDirect on VMware vSphere works by leveraging NVIDIA vGPU technology and RDMA-capable network adapters like the NVIDIA ConnectX-6. For VMs Access Control Services (ACS) relax VMX settings as well as NUMA affinity or device groups must be configured, to enable peer-to-peer communication between the PCIe devices on the same Root Complex, at the time of writing, Device groups are yet supported for Tanzu Kubernetes Grid so proper NUMA and Root complex alignment cannot be assured, Device groups are expected to be available via VM classes on a future vSphere 8 release. Refer to the following links To configure ACS, NUMA affinity, or device groups, see Deploy an AI-Ready Enterprise Platform on vSphere 7 Update 2, vSphere 8 Expands Machine Learning Support: Device Groups for NVIDIA GPUs and NICs.

VMware Environment Preparation

To properly prepare VMware environments, plan the following items:

- **Assessment and Planning**: Begin by assessing your organization's requirements, including computing resources, storage, networking, and application needs. Plan the virtual infrastructure, accordingly, considering factors such as scalability, redundancy, and disaster recovery. It is crucial to understand the workload demands and design a flexible architecture to accommodate future growth.
- Hardware Selection and Compatibility*:* Choose hardware components that are compatible with VMware's virtualization platform. Ensure that the server hardware, storage devices, and networking equipment are on VMware's Hardware Compatibility List (HCL) to guarantee smooth operations and optimal performance.
- Networking Setup*:* Create virtual networks and VLANs to segregate traffic and enhance security. Consider using VMware NSX-T provided by VMware Cloud Foundation or VMware's distributed switches for more streamlined network management and monitoring. Implement proper firewall configurations, configure network security policies, and isolate sensitive workloads as required.
- **Backup and Disaster Recovery:** Establish a robust backup and disaster recovery strategy to protect critical data and ensure business continuity. Consider utilizing VMware's Data Protection tools or integrate with third-party backup solutions.
- Performance Monitoring and Optimization*:* Consider utilizing tools like vCenter Server and VMware Aria Operations to identify performance bottlenecks and make informed decisions to optimize resource utilization and maintain optimal performance.

There are multiple options of preparing the VMware environment, here we provide two options to configure the VMware environment with or without VMware Cloud Foundation. VMware Cloud Foundation Environment Preparation summarizes the high-level steps of preparing the VMware environment with VMware Cloud Foundation; in any cases that VMware Cloud Foundation environment could not be configured, you can follow the steps in vSphere Environment Preparation to prepare the VMware Environment.

Option 1. VMware Cloud Foundation Environment Preparation

1. Familiarize with the VMware Cloud Foundation design and concepts. The VMware Cloud Foundation Design Guide contains requirements and recommendations for the design of each component of the SDDC.
2. Ensure all servers and network switches are supported and meet VMware Cloud Foundation requirements by consulting the VMware Compatibility Guide.
3. Based on the chosen design options, configure the network and services (DNS, DHCP, NTP), then complete the VMware Cloud Foundation Planning and Preparation Workbook.
4. Prepare the ESXi hosts for VMware Cloud Foundation by installing the appropriate ESXi version and configuring the systems. Details on how to install and configure the systems are found in the Prepare ESXi Hosts for VMware Cloud Foundation in the VMware Cloud Foundation Deployment Guide.
5. Deploy the VMware Cloud Foundation management domain by deploying the VMware Cloud Builder and using the VMware Cloud Foundation Planning and Preparation Workbook. Cloud builder creates the management cluster, deploys the SDDC manager, then creates the vCenter and NSX-T for the management domain. More detail about the process can be found in the VMware Cloud Foundation Deployment Guide.
6. Once the VMware Cloud Foundation management domain deployment is complete, use the SDDC manager to add additional hosts and created additional VI workload domain. More detail about the process of adding and creating VI workload domain can be found in the VMware Cloud Foundation Administration Guide.

Option 2. Standalone vSphere Environment Preparation

1. VMware vCenter Server 8.0 Update 1b or later is configured to manage the ESXi servers. We recommend running it as a VM on a separate host to the above host. For the vCenter Server installation procedures and details, see About vCenter Server Installation and Setup.
2. VMware vSphere Hypervisor 8.0 Update 1 installed on each of the servers. For the Hypervisor installation procedures and details, see About VMware ESXi Installation and Setup.
3. Furthermore, About VMware vCenter Server and Host Management provides information about vCenter and ESXi server management and configuration.
4. Also, we recommend using VMware vSAN as the first tier storage for VM placement, for more information, see About vSAN Planning and Deployment.

After vSphere Cluster is configured, make sure vSphere Availability and vSphere DRS with fully automated setting are enabled on the target vSphere Cluster.

Software Configuration

Software Configuration Considerations

GPU Configuration in vSphere: vGPU or Passthrough

In a vSphere environment, using GPUs via passthrough (either DirectPath I/O or Dynamic DirectPath I/O) provides VMs with exclusive access to a physical GPU, resulting in maximum performance and predictable latency. However, this method is not efficient in scenarios with lighter GPU workloads, as any unused GPU capacity goes wasted. vGPU technologies (Time Sliced or MIG) on the other hand, allow multiple VMs to share GPU resources. This promotes optimal utilization and provides flexibility in resource allocation. MIG takes a middle ground by partitioning a GPU into isolated instances, which offers predictable performance closer to passthrough with full isolation while allowing for better hardware utilization like vGPU with Time Sliced mode. However, MIG is available for ampere or newer NVIDIA GPU Architectures, and configurations can be limited based on the specific partitioning options available. For this guide we decided to use vGPU with Time Sliced mode but any other options such as MIG or passthrough could work as well.

The following table provides benefits and tradeoffs of each GPU technology available with VMware vSphere:

Table 9: Benefits and Limitations of GPU Technology with VMware vSphere

	vGPU Time Sliced	vGPU MIG	Dynamic DirectPath /IO	DirectPath I/O
Tanzu VM Class Supported	Yes	Yes	Yes	No
Fractional GPUs	Yes	Yes	No	No
Suspend and Resume	Yes	Yes	No	No
Snapshot	Yes	Yes	No	No
Host Maintenance Mode	vMotion	vMotion	Cold Migration	VM Shutdown
Load Balance DRS	No	No	No	No
Initial Placement DRS	Yes	Yes	Yes	No
vMotion	Yes	Yes	Yes	No
Failover HA	Yes	Yes	Yes	No

Network Adapter Configuration Consideration

The adapter type to be used in VMware stacks can be selected as follows with virtualization features and performance.

Table 10: Network Adapter Configuration

Category	**Paravirtual**		**Passthrough**		
Network adapter type	**VMXNET3**	**PVRDMA**	**DirectPath IO**	**Dynamic DirectPath IO**	**SRIOV**
vMotion/ Snapshot/Suspend & Resume supported?	Yes	Yes	No	No	No
HA/DRS supported?	Yes	Yes	No	Yes* (DRS initial placement in vSphere 7)	Yes* (DRS initial placement in vSphere 8U2)
Virtual Distributed Switch required?	Yes	Yes	No	No	Yes
Multi-VMs sharing?	Yes	Yes	No. Exclusive to 1 VM	Yes. But exclusive to 1 VM once powered on.	Yes
RDMA capability	No. TCP only	RoCE only	IB or RoCE	IB or RoCE	IB or RoCE
Tanzu VM Class supported?	Yes	No	No	Yes	Yes

Considering the performance requirement for GenAI use case and resource utilization, we choose to enable SRIOV on the NIC to be used in the workload network. Through VF-level bandwidth specification, SRIOV ensures allocated bandwidths for workloads or VMs, thus meeting the SLA commitments of specific tasks and avoiding bandwidth monopolization. More related works can be found in Virtualizing HPC on VMware vSphere: Choosing a Best Method for RDMA HCA/NIC Deployment for MPI Workloads.

Install VMware Installation Bundles

vSphere Lifecycle Manager (vLCM) is introduced since vSphere 7 and can improve lifecycle management on a cluster. It can integrate the NVAIE vGPU driver and Mellanox firmware tools with the vSphere base image to enforce consistency across the ESXi hosts. Here is the flowchart for this step:

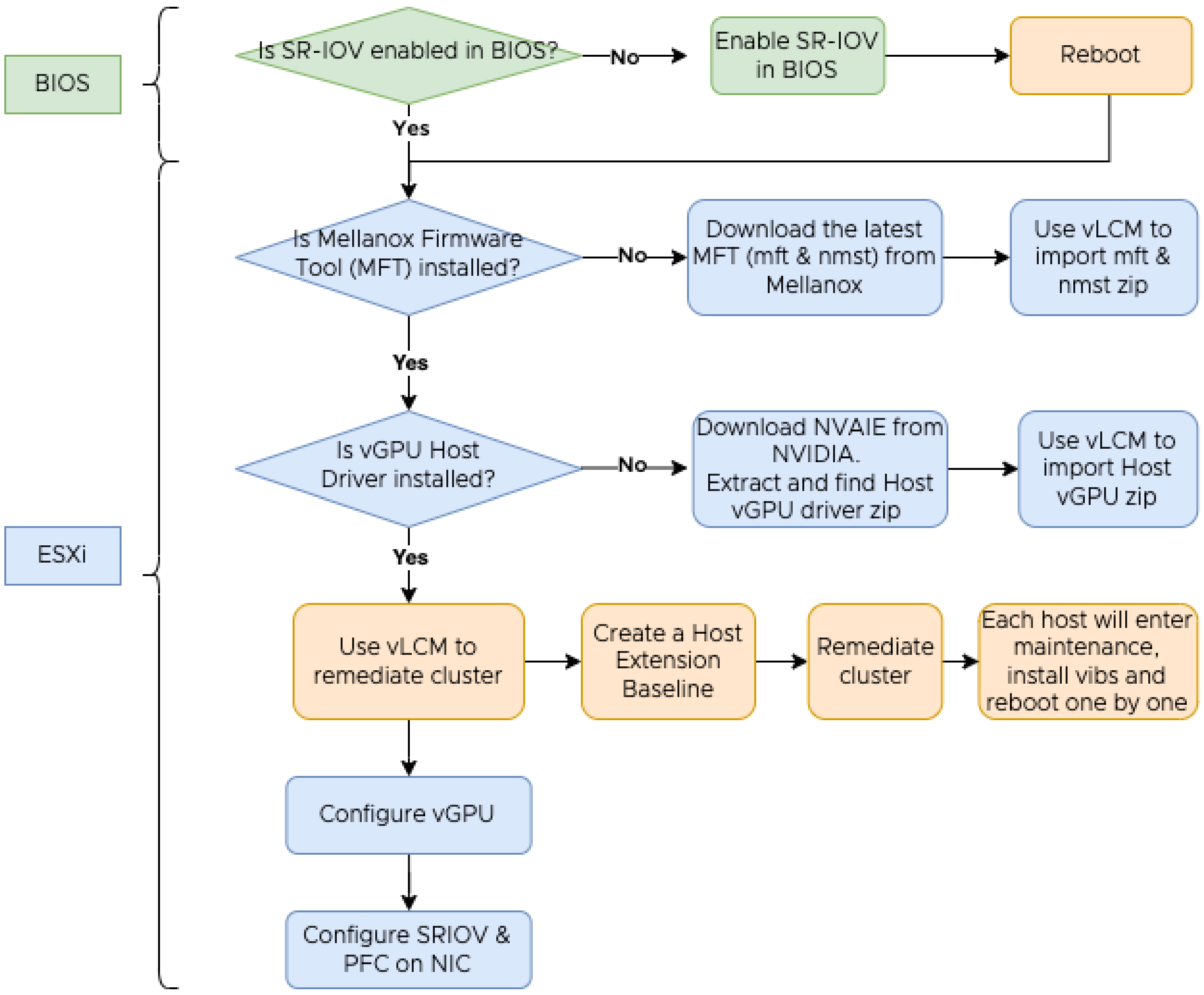
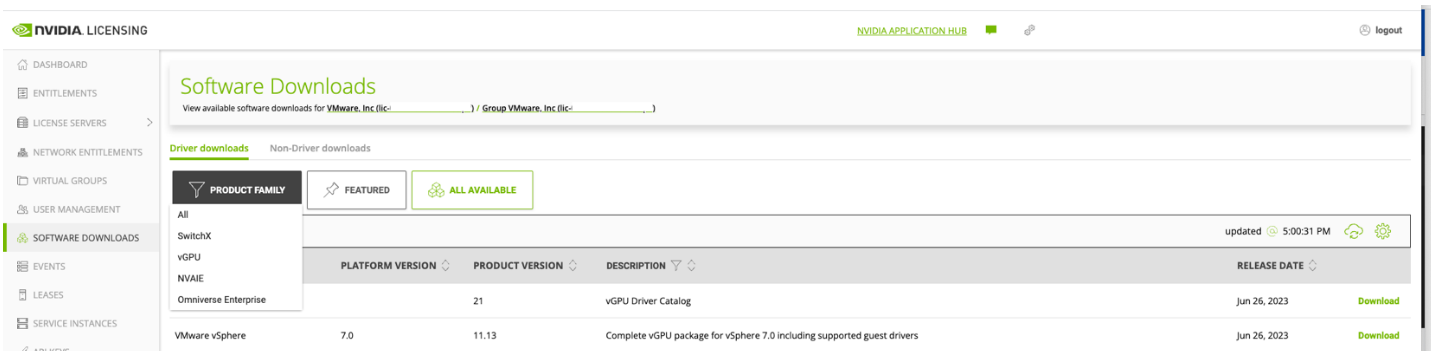

BIOS
ESXi
Is SR-IOV enabled in BIOS?
No
Enable SR-IOV in BIOS
Reboot
Yes
Is Mellanox Firmware Tool (MFT) installed?
No
Download the latest MFT (mft & nmst) from Mellanox
Use vLCM to import mft & nmst zip
Yes
Is vGPU Host Driver installed?
No
Download NVAIE from NVIDIA. Extract and find Host vGPU driver zip
Use vLCM to import Host vGPU zip
Yes
Use vLCM to remediate cluster
Create a Host Extension Baseline
Remediate cluster
Each host will enter maintenance, install vibs and reboot one by one
Configure vGPU
Configure SRIOV & PFC on NIC

Figure 8: Flowchart to Install VIB of NVAIE and MFT

The NVAIE driver is available at the NVIDIA Licensing Portal, part of the **NVIDIA APPLICATION HUB**. Go to the **SOFTWARE DOWNLOADS** option in the left menu, or go directly to https://ui.licensing.nvidia.com/software if logged in. For ML workloads, choose the **NVAIE** from the product family. The NVAIE product family provides vGPU capability for compute (ML/AI) workload.

Note: The vGPU family is for vGPU functionality for the VDI workload. Next extract the "NVAIE-AI-Enterprise-vSphere-xxx.zip" package. Then, navigate to the "Host Driver" directory and locate the "NVD-AIE-xxx.zip" file. This file contains the vGPU host driver that needs to be imported to vLCM.



Software Downloads

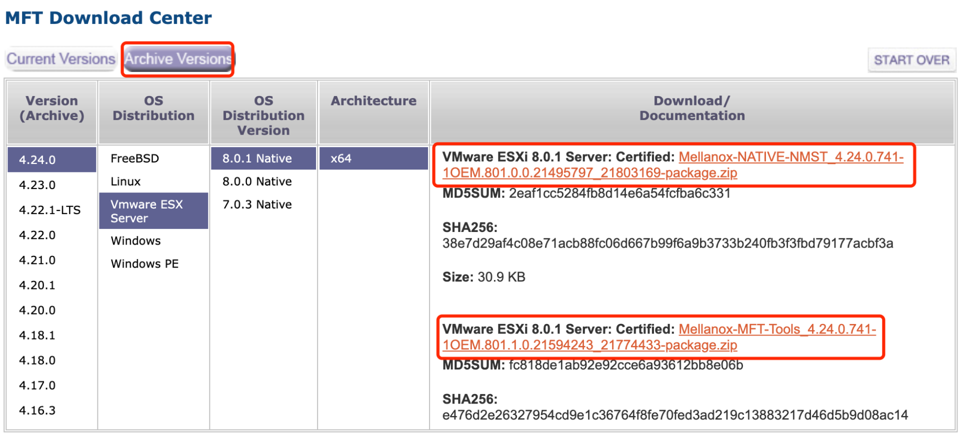
View available software downloads for VMware, Inc (lic-) / Group VMware, Inc (lic-)

updated @ 5:00:31 PM

	PLATFORM VERSION	PRODUCT VERSION	DESCRIPTION	RELEASE DATE	
		21	vGPU Driver Catalog	Jun 26, 2023	Download
VMware vSphere	7.0	11.13	Complete vGPU package for vSphere 7.0 including supported guest drivers	Jun 26, 2023	Download

Figure 9: Download the NVAIE Package from NVIDIA

Mellanox Firmware Tools (MFT) consists of MFT & NMST components. Download them for VMware vSphere 8 update 1 from this link: https://network.nvidia.com/products/adapter-software/firmware-tools/. We selected the 4.24 version in the archive versions since it is compatible with the VMware vSphere 8 update 1 used in VMware Cloud Foundation 5.0.


MFT Download Center

Current Versions | Archive Versions | START OVER

Version (Archive)	OS Distribution	OS Distribution Version	Architecture
4.24.0	FreeBSD	8.0.1 Native	x64
4.23.0	Linux	8.0.0 Native	
4.22.1-LTS	Vmware ESX Server	7.0.3 Native	
4.22.0	Windows		
4.21.0	Windows PE		
4.20.1			
4.20.0			
4.18.1			
4.18.0			
4.17.0			
4.16.3			

Download/ Documentation

VMware ESXi 8.0.1 Server: Certified: Mellanox-NATIVE-NMST_4.24.0.741-1OEM.801.0.0.21495797_21803169-package.zip
MD5SUM: 2eaf1cc5284fb8d14e6a54fcfba6c331

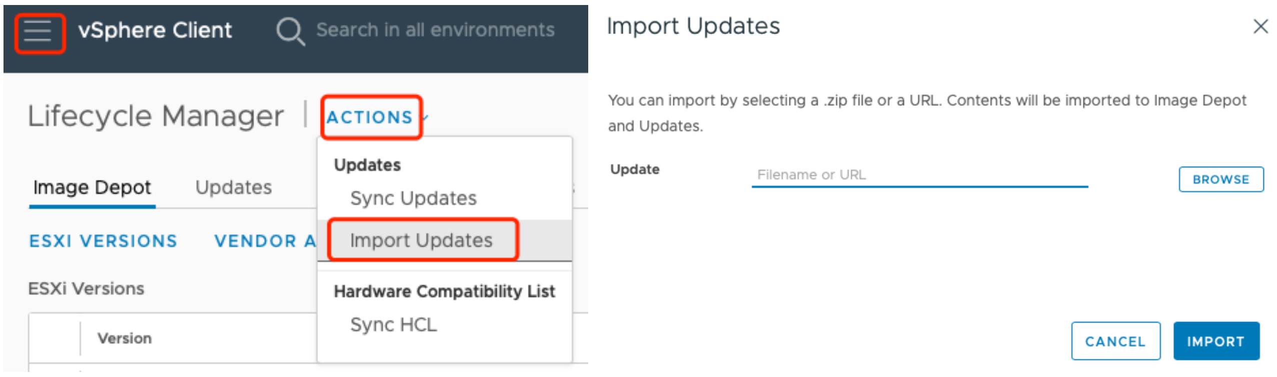
SHA256: 38e7d29af4c08e71acb88fc06d667b99f6a9b3733b240fb3f3fbd79177acbf3a

Size: 30.9 KB

VMware ESXi 8.0.1 Server: Certified: Mellanox-MFT-Tools_4.24.0.741-1OEM.801.1.0.21594243_21774433-package.zip
MD5SUM: fc818de1ab92e92cce6a93612bb8e06b

SHA256: e476d2e26327954cd9e1c36764f8fe70fed3ad219c13883217d46d5b9d08ac14

Both downloaded MFT packages must be unpacked, then upload “NVD-AIE-xxx.zip”, “Mellanox-MFT-Tools_xxx.zip”, and “Mellanox-NATIVE-NMST_xxx.zip” files to vSphere Lifecycle Manager by opening the vSphere client menu (click on the three lines in the top left corner next to vSphere Client) select **Lifecycle Manager**, select **ACTIONS**, **Import Updates**.



Import Updates

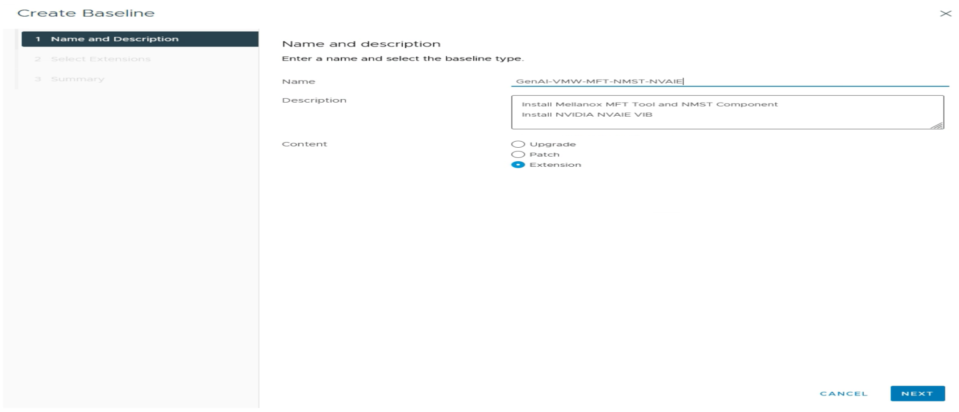
You can import by selecting a .zip file or a URL. Contents will be imported to Image Depot and Updates.

Update: Filename or URL

BROWSE | CANCEL | IMPORT

Figure 10: Import VIBs with vLCM

Next, we used vSphere Lifecycle Manager Baselines to create a host extension baseline by clicking the **targeted Cluster** > **Updates** > **Hosts** > **Baselines** > **New** Baseline > Type name, description, and select **Extension** > Find and Select “NVAIE”, “MFT” and “NMST” as extension > **Finish**.


Create Baseline

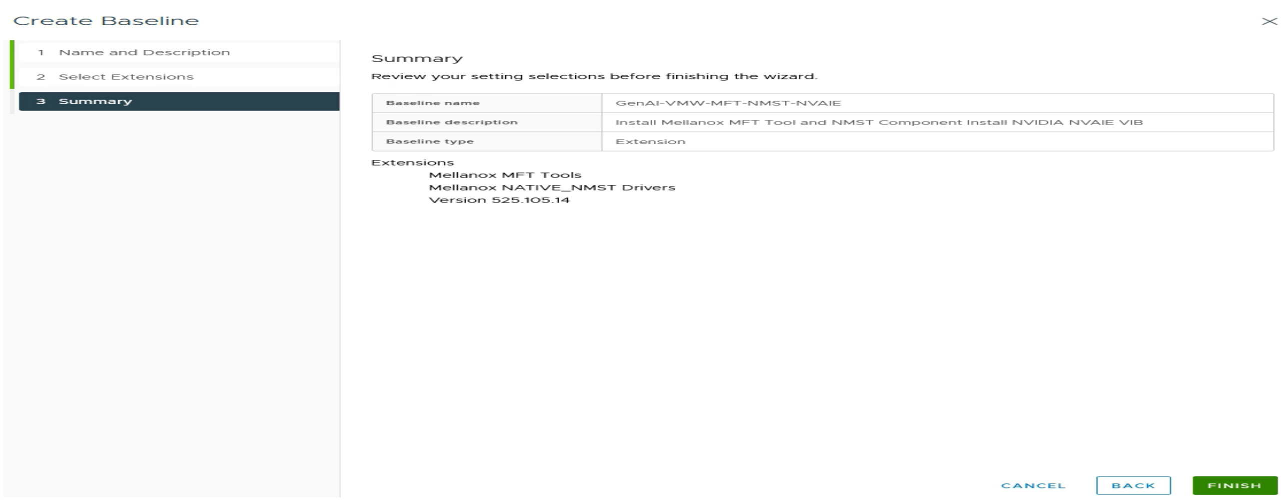
Name and description

Enter a name and select the baseline type.

Name	GenAI-VMW-MFT-NMST-NVAIE
Description	Install Mellanox MFT Tool and NMST Component Install NVIDIA NVAIE VIB
Content	Upgrade Patch [✓ Extension]

CANCEL | NEXT

Figure 11: Create a Host Extension Baseline



Summary

Review your setting selections before finishing the wizard.

Baseline name	GenAI-VMW-MFT-NMST-NVAIE
Baseline description	Install Mellanox MFT Tool and NMST Component Install NVIDIA NVAIE VIB
Baseline type	Extension

Extensions
    Mellanox MFT Tools
    Mellanox NATIVE_NMST Drivers
    Version 525.105.14

Figure 12: Summary of Baseline Extension with NVAIE and MFT

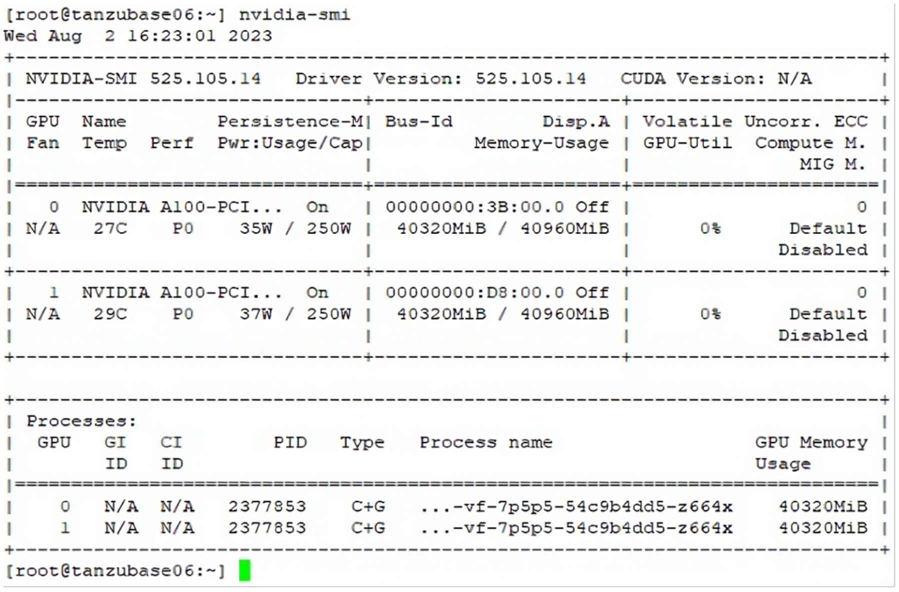
Next we remediated the cluster by clicking the **targeted Cluster** > **Updates** > **Hosts** > **Baselines** > select the baseline created > **REMEDIATE**. Then vLCM will put each host into maintenance mode, install VIBs, and reboot one by one.

Alternatively, we can manually put each host into maintenance mode and SSH into each host, then install the above three VIBs by running the following commands and reboot individually:

```
esxcli software vib install -d /PATH_TO/“NVD-AIE-xxx.zip”

esxcli software vib install -d /PATH_TO/"Mellanox-NATIVE-NMST-xxx.zip"

esxcli software vib install -d /PATH_TO/"Mellanox-MFT-Tools-xxx.zip"
```

Configure vGPU

After the NVAIE vGPU Manager VIB is installed, open an SSH session to each ESXi host to verify that the NVIDIA vGPU Manager is available and can detect the GPUs on the system:



```
[root@tanzubase06:~] nvidia-smi
Wed Aug  2 16:23:01 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.14   Driver Version: 525.105.14   CUDA Version: N/A      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A100-PCI...  On   | 00000000:3B:00.0 Off |                    0 |
| N/A   27C    P0    35W / 250W |  40320MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-PCI...  On   | 00000000:D8:00.0 Off |                    0 |
| N/A   29C    P0    37W / 250W |  40320MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   2377853    C+G   ...-vf-7p5p5-54c9b4dd5-z664x    40320MiB |
|    1   N/A  N/A   2377853    C+G   ...-vf-7p5p5-54c9b4dd5-z664x    40320MiB |
+-----------------------------------------------------------------------------+
[root@tanzubase06:~]
```

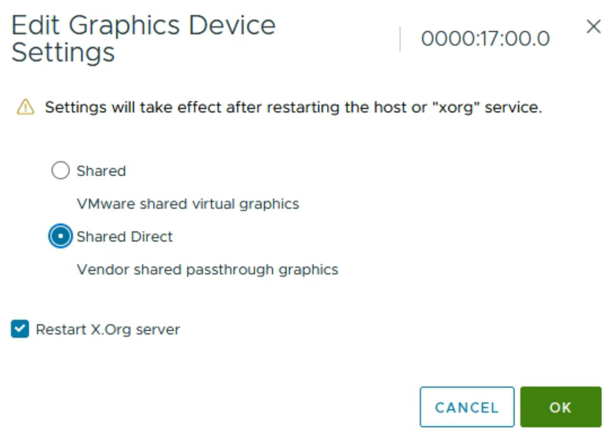
Figure 13: Verify vGPU is Detected in ESXi

The GPU(s) of each ESXi host could be configured based on the required GPU technology, either vGPU or MIG. To configure GPU as dynamic DirectPath I/O, refer to our existing documentation.

Here are the steps to configure vGPU on each ESXi host in the vSphere client.

1. Log in to vSphere Client and navigate to **vCenter > Datacenter > Cluster > Host > Configure > Graphics** and click on **HOST GRAPHICS.**
2. Under the Host Graphics click **Edit** and choose **Shared Direct** and **Spread VMs across GPUs**.

3. After these settings applied, click on tab **GRAPHIC DEVICES**, choose one GPU listed below and click **EDIT…**, ensure that **Restart X.Org server** is checked and **Shared Direct** is selected, then click **OK**.


Edit Graphics Device Settings | 0000:17:00.0

Settings will take effect after restarting the host or "xorg" service.

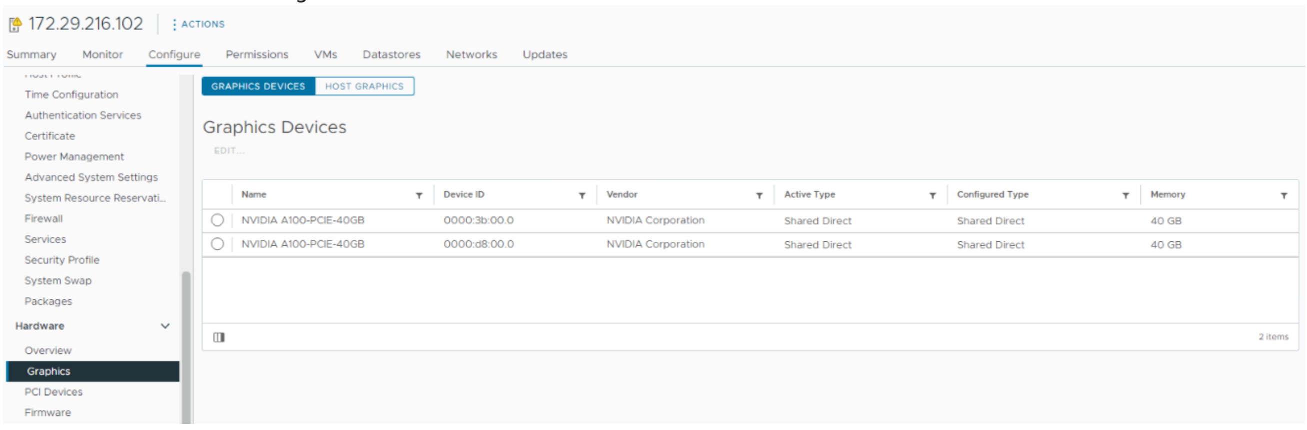
Shared [✓ Shared Direct]

- Shared: VMware shared virtual graphics
- Shared Direct: Vendor shared passthrough graphics

✓ Restart X.Org server

Figure 14: Configure vGPU in vSphere Client

4. Confirm the setting has been set:



Graphics Devices

Name	Device ID	Vendor	Active Type	Configured Type	Memory
NVIDIA A100-PCIE-40GB	0000:3b:00.0	NVIDIA Corporation	Shared Direct	Shared Direct	40 GB
NVIDIA A100-PCIE-40GB	0000:d8:00.0	NVIDIA Corporation	Shared Direct	Shared Direct	40 GB

2 items

Figure 15: Verify vGPU Status in vSphere Client

Configure SRIOV and PFC on Mellanox NICs

After MFT is installed, we will employ MFT to enable SR-IOV on NIC firmware and enable flow control and SR-IOV parameters on RoCE ports of each host.

Activate SR-IOV and Priority Flow Control (PFC) using these commands, then reboot the host:

```
/opt/mellanox/bin/mlxconfig -d mt4125_pciconf0 -y set SRIOV_EN=1 NUM_OF_VFS=16

esxcli system module parameters set -m nmlx5_core -p "pfctx=0x08 pfcrx=0x08 trust_state=2 max_vfs=8,7 ecn=1"

reboot
```

In the first command, **NUM_OF_VFS=16** creates VFs on the firmware.

In the second command, **max_vfs=8,7** configures VFs on each of the two ports, and **pfctx=0x08 pfcrx=0x08 ecn=1** explicitly enables the setting of Priority Flow Control (PFC) and Explicit Congestion Notification (ECN) on RoCE ports.

If a Mellanox VPI HCA is used in your environment, the **LINK_TYPE** parameter is required to set the transport mode as **IB** or **RoCE**. This can be determined by running:

```
/opt/mellanox/bin/mlxfwmanager -d $Your_Device_PCIE_ID
```

And then check whether the "Description" row of the device contains "VPI". For more about how to configure SRIOV RoCE or IB on NIC/HCA, refer to RoCE SR-IOV Setup and Performance Study on vSphere 7.x and InfiniBand SR-IOV Setup and Performance Study on vSphere 7.x.

Enabling vSphere with Tanzu

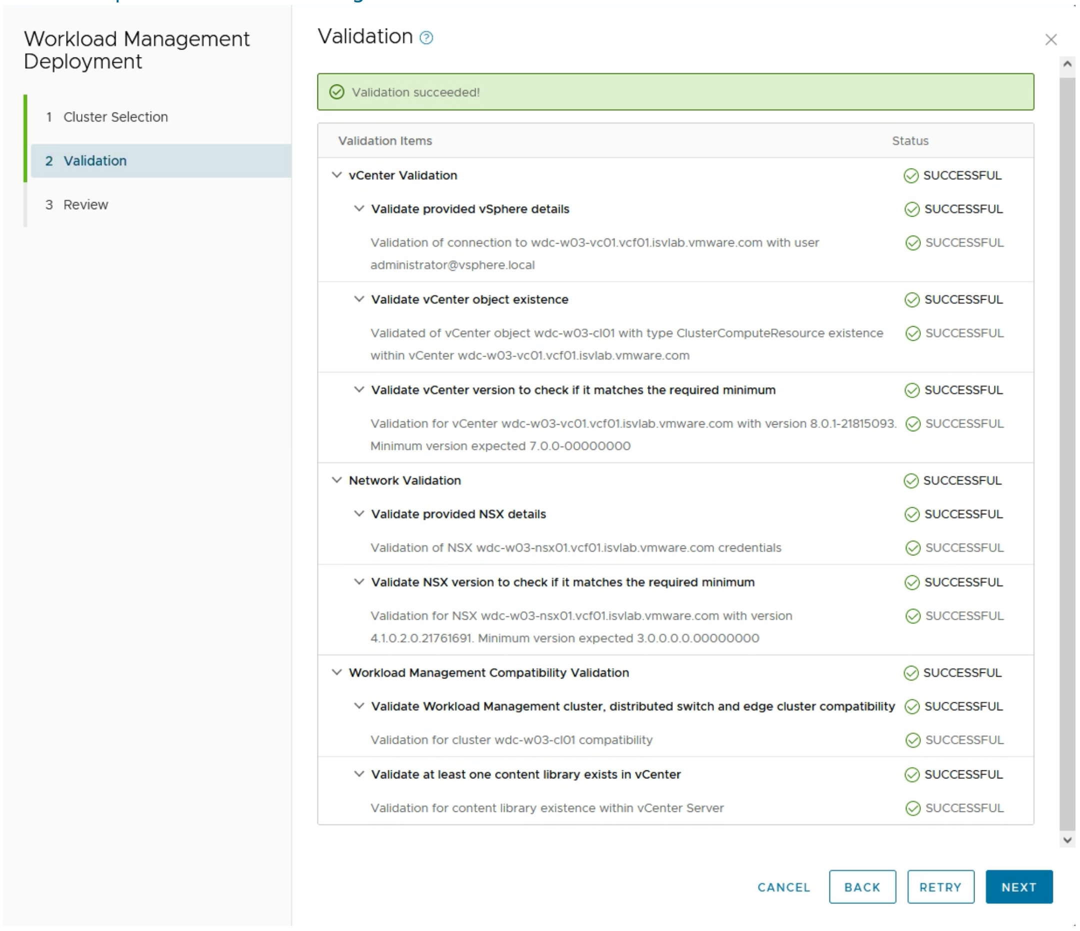
There are two options to enable vSphere with Tanzu. You can either enable vSphere with Tanzu with VMware Cloud Foundation or Service Installer. If you have VMware Cloud Foundation deployed, it is highly recommended to enable vSphere with Tanzu service by following the steps in Enabling vSphere with Tanzu with VMware Cloud Foundation. Otherwise, follow the steps in Enabling vSphere with Tanzu with Service Installer.

Option 1. Enabling vSphere with Tanzu with VMware Cloud Foundation

In VMware Cloud Foundation deployments, it is recommended to follow the Workload Management solution workflow built into SDDC Manager to streamline the enabling of Workload Control Plane (WCP) over NSX-T based networks. This solution integration reliably and efficiently manages the complete lifecycle management of all VMware Cloud Foundation components within its framework that results in consistent operations across all deployments.

The list below summarizes the high-level steps required to enable VMware Cloud Foundation with Tanzu Services using Workload Management solution. Refer to VMware Cloud Foundation for VMware Tanzu in VMware Cloud Foundation Administration Guide for more information.

1. Sizing compute and storage resources for Workload Management
2. Create a Subscribed Content library
3. Enable Workload Management
4. View Workload Management cluster details
5. Update Workload Management license


Workload Management Deployment

1 Cluster Selection
2 Validation
3 Review

Validation

Validation succeeded!

Validation Items	Status
vCenter Validation	SUCCESSFUL
Validate provided vSphere details	SUCCESSFUL
Validation of connection to wdc-w03-vc01.vcf01.isvlab.vmware.com with user administrator@vsphere.local	SUCCESSFUL
Validate vCenter object existence	SUCCESSFUL
Validated of vCenter object wdc-w03-cl01 with type ClusterComputeResource existence within vCenter wdc-w03-vc01.vcf01.isvlab.vmware.com	SUCCESSFUL
Validate vCenter version to check if it matches the required minimum	SUCCESSFUL
Validation for vCenter wdc-w03-vc01.vcf01.isvlab.vmware.com with version 8.0.1-21815093. Minimum version expected 7.0.0-00000000	SUCCESSFUL
Network Validation	SUCCESSFUL
Validate provided NSX details	SUCCESSFUL
Validation of NSX wdc-w03-nsx01.vcf01.isvlab.vmware.com credentials	SUCCESSFUL
Validate NSX version to check if it matches the required minimum	SUCCESSFUL
Validation for NSX wdc-w03-nsx01.vcf01.isvlab.vmware.com with version 4.1.0.2.0.21761691. Minimum version expected 3.0.0.0.0.00000000	SUCCESSFUL
Workload Management Compatibility Validation	SUCCESSFUL
Validate Workload Management cluster, distributed switch and edge cluster compatibility	SUCCESSFUL
Validation for cluster wdc-w03-cl01 compatibility	SUCCESSFUL
Validate at least one content library exists in vCenter	SUCCESSFUL
Validation for content library existence within vCenter Server	SUCCESSFUL

Figure 16: SDDC Manager Validating Tanzu Prerequisites

After you follow the Workload Management solution workflow to enable WCP, you may skip to the Deploy Tanzu Kubernetes Workload Cluster section below.

For non-VMware Cloud Foundation environments, you may use Service Installer for VMware Tanzu to enable WCP.

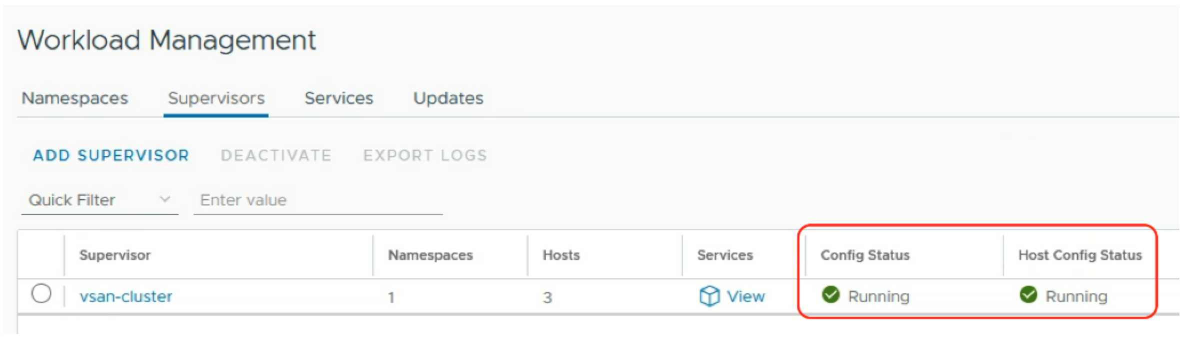
Option 2. Enabling vSphere with Tanzu with Service Installer

Service Installer for VMware Tanzu Overview

Service Installer automates the deployment of the reference designs for Tanzu for Kubernetes Operations on several cloud platforms, including vSphere with Tanzu backed by VDS. Automation with Service Installer for VMware Tanzu simplifies the deployment of a Kubernetes environment. It uses best practices for deploying and configuring the required Tanzu for Kubernetes Operations components such as Tanzu Kubernetes Grid, Tanzu Mission Control, Tanzu Service Mesh, Tanzu Observability, NSX Advanced Load Balancer, and a set of shared services including Harbor, Prometheus, Grafana, Valero, Cert Manager, Contour, and Fluentbit. For more information, refer to the Service Installer for VMware Tanzu documentation.

Installing with Service Installer

Before you deploy Tanzu for Kubernetes Operations using Service Installer for VMware Tanzu, refer to the "Prerequisites" section in Deploying VMware Tanzu for Kubernetes Operations on vSphere with Tanzu and vSphere Distributed Switch Using Service Installer for VMware Tanzu. Follow the documentation to download and deploy the Service Installer. Proceed to follow the procedure Deploy Tanzu for Kubernetes Operations and choose the deployment type "Enable Workload Control Plane", configure and generate a JSON file, and use the *arcas* CLI to initiate WCP deployment. To verify if the deployment is successful, go to "Workload Management" and select the "Supervisors" tab in vSphere Client to confirm if the "Config Status" and "Host Config Status" are both in "Running" state.



	Supervisor	Namespaces	Hosts	Services	Config Status	Host Config Status
○	vsan-cluster	1	3	View	✓ Running	✓ Running

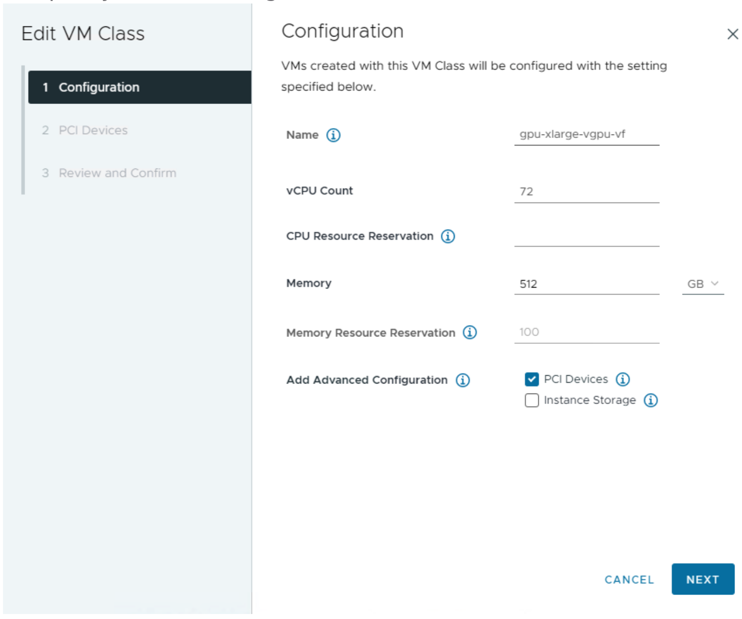
Figure 17: vSphere with Tanzu Supervisor Cluster is Up

Deploying Tanzu Kubernetes Workload Cluster

Before deploying a workload cluster, you need to follow the instructions below to create a custom VM class with the vGPU profile.

Creating VM Class

1. Log on to the vCenter Server using the vSphere Client.
2. Select **Workload Management** > **Services** > **VM Service** > **Manage**.
3. Select **VM Classes** > **Create VM Class**.
4. Specify the VM settings as shown below and select **NEXT**:

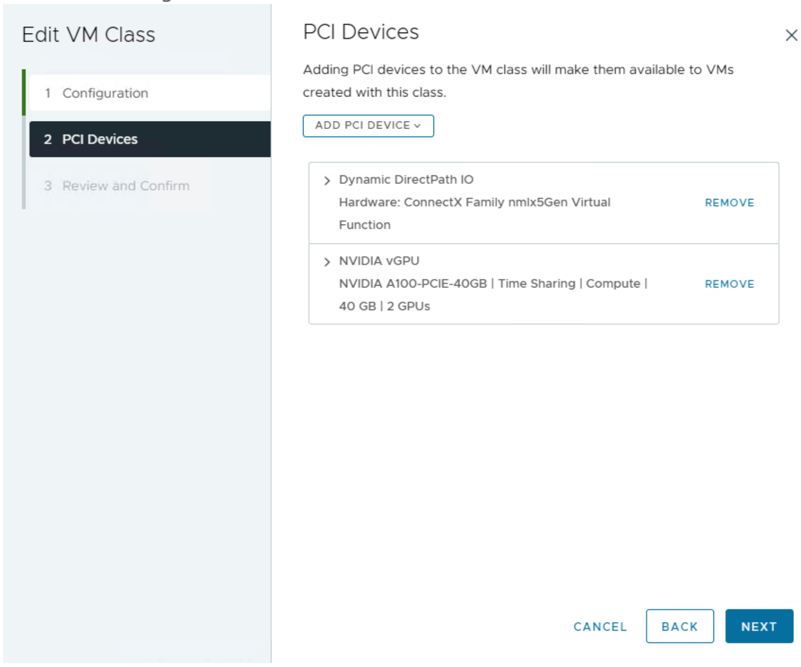


VMs created with this VM Class will be configured with the setting specified below.

Name	gpu-xlarge-vgpu-vf
vCPU Count	72
CPU Resource Reservation	
Memory	512 GB
Memory Resource Reservation	100
Add Advanced Configuration	☑ PCI Devices ☐ Instance Storage

Figure 18: Creating the VM Class

5. Select **ADD PCI DEVICE > NVIDIA vGPU**, choose the hardware model that you have, and populate the remaining settings as shown below:



Adding PCI devices to the VM class will make them available to VMs created with this class.

Dynamic DirectPath IO
Hardware: ConnectX Family nmlx5Gen Virtual Function

NVIDIA vGPU
NVIDIA A100-PCIE-40GB | Time Sharing | Compute | 40 GB | 2 GPUs

Figure 19: Adding vGPU and NIC VF to VM Class

6. Select **NEXT** and **FINISH** to finish creating the custom VM class.

Deploying Tanzu Kubernetes Workload Cluster

After the custom VM class is created and WCP is enabled, there are two options to provision a Tanzu Kubernetes workload cluster. If you have custom configurations such as volume mounts in workload cluster nodes, you can follow the vSphere product documentation Provisioning TKG Clusters on Supervisor, use the *kubectl* commands and the YAML file to provision a workload cluster.

Alternatively, if Service Installer is used and there is no custom TKC configuration requirement, you can repeat the same steps in the Service Installer but choose the deployment type “Namespace and Workload Cluster”. As you specify parameters to generate a JSON file, make sure to choose the custom VM class that was created with vGPU profile for "WORKER VM CLASS". Once a JSON file is created, use the *arcas* CLI again to initiate the creation of a Tanzu Kubernetes workload cluster.

To test the connectivity to the workload cluster, refer to the vSphere documentation Workflow for Provisioning TKG Clusters on Supervisor Using Kubectl to download and install the Kubernetes CLI tools for vSphere, then use the *kubectl* commands to monitor the cluster provisioning, and log into the workload cluster to check its resources.

Deploying NVIDIA GPU Operator

The NVIDIA GPU Operator uses the operator framework within Kubernetes to automate the management of all NVIDIA software components needed to provision GPU. These components include the NVIDIA drivers (to enable CUDA), Kubernetes device plugin for GPUs, the NVIDIA Container Toolkit, automatic node labelling using GFD, DCGM based monitoring and others.

The GPU Operator allows administrators of Kubernetes clusters to manage GPU nodes just like CPU nodes in the cluster. Instead of provisioning a special OS image for GPU nodes, administrators can rely on a standard OS image for both CPU and GPU nodes and then rely on the GPU Operator to provision the required software components for GPUs.

The GPU Operator also enables GPUDirect RDMA; a technology in NVIDIA GPUs that enables direct data exchange between GPUs and a third-party peer device using PCI Express. The third-party devices could be network interfaces such as NVIDIA ConnectX SmartNICs or BlueField DPUs amongst others.

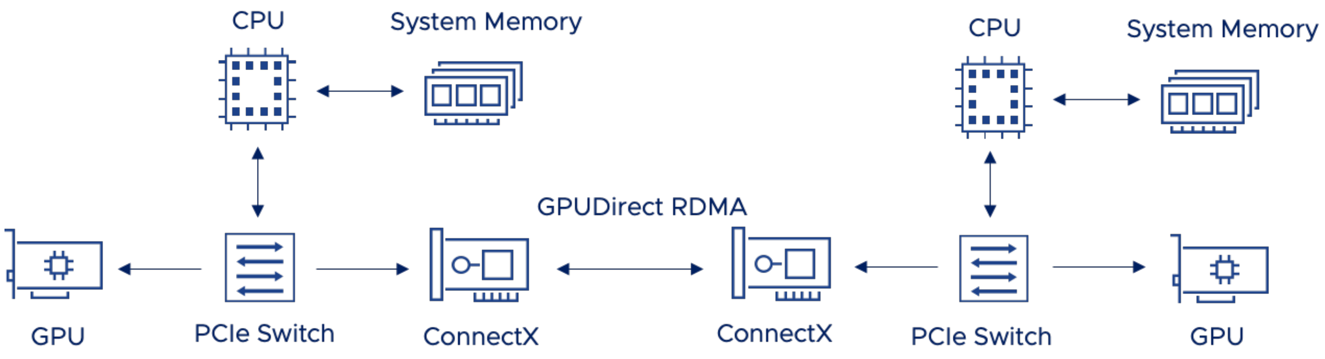


Figure 20: GPUDirect RDMA Technology

To support GPUDirect RDMA, a userspace CUDA APIs and kernel mode drivers are required. Starting with *CUDA 11.4 and R470 drivers*, a new kernel module nvidia-peermem is included in the standard NVIDIA driver installers with ".run" extensions. The kernel module provides Mellanox InfiniBand and RoCE adapters direct peer-to-peer read and write access to the GPU's memory.

In conjunction with the *Network Operator*, the GPU Operator can be used to set up the networking related components such as Mellanox drivers, nvidia-peermem and Kubernetes device plugins to enable workloads to take advantage of GPUDirect RDMA.

GPUDirect has multiple requirements:

- MOFED Drivers need to be installed: for Tanzu workloads, these drivers will be installed by NVIDIA's Network Operator. For VMs, refer to NVIDIA's documentation for details on the installation process.
- Make sure the Network interface controller and the NVIDIA GPU are in the same PCIe I/O root complex, you can leverage vSphere device groups with VMs to accomplish this, for Tanzu Worker nodes this is yet supported.
- Enable the following VMX advanced settings:
    - pciPassthru.allowP2P = true
    - pciPassthru.RelaxACSforP2P = true
    - pciPassthru.use64bitMMIO = true[1]
    - pciPassthru.64bitMMIOSizeGB = N [1],[2]

We need to install this GPU Operator on each TKC that will be used for GPU-enabled workloads. We will proceed to install it on the cluster that we previously created.

See the following high-level overview of the installation process, refer to the NVIDIA GPU Operator Step-by-Step Installation Process for a complete step-by-step installation:

1. Create a namespace.
2. Configure vGPU license via ConfigMap.
3. Create a secret to pull from the NGC repository.
4. Set up NVAIE Helm repository.
5. Install GPU Operator via Helm.
6. Verify the installation of GPUDirect with RDMA.

Deploying NVIDIA Network Operator

NVIDIA Network Operator leverages Kubernetes CRDs and Operator SDK to manage networking related components, in order to enable fast networking, RDMA and GPUDirect for workloads in a Kubernetes cluster. The Network Operator works in conjunction with the GPU Operator to enable GPUDirect RDMA on compatible systems. The goal of the Network Operator is to manage the networking related components, while enabling execution of RDMA and GPUDirect RDMA workloads in a Kubernetes cluster. This includes:

- NVIDIA Networking drivers to enable advanced features.
- Kubernetes device plugins to provide hardware resources required for a fast network.
- Kubernetes secondary network components for network intensive workloads.

The NVIDIA Network Operator can be deployed in different modes, for this guide we will deploy a network operator with Host Device Network, this deployment includes:

- SR-IOV device plugin, single SR-IOV resource pool

- Secondary network
- Mutlus CNI
- Container networking-plugins CNI plugins
- Whereabouts IPAM CNI plugin

In this mode, the Network Operator could be deployed on virtualized deployments as well. It supports both Ethernet and InfiniBand modes. From the Network Operator perspective, there is no difference between the deployment procedures. To work on a VM the PCI passthrough must be configured for SR-IOV devices. The Network Operator works both with VF and Physical Function (PF) inside the VMs or Tanzu Worker nodes.

High level overview of installation process, refer to the NVIDIA Network Operator Step-by-Step Installation Process for a complete step-by-step installation:

1. Create a namespace.
2. Create a secret to pull from NGC repository.
3. Create a *values.yaml* file.
4. Install the Network Operator via *Helm*.
5. Create Hostdev custom resource.
6. Verify the *nvidia-peermem-ctr* container.

Refer to Appendix for Virtual Function Verification and Testing Process and Verifying the Installation of GPUDirect with RDMA.

Creating VM Service for Single Node Learning

In the LLM training use case, you can provision a single node VM using VM service as follows:

1. Repeat the Creating VM Class procedure in the previous section and specify the desired settings for your VM. For example, you may want to configure two or more for the parameter “Number of vGPUs”. In this solution, we created the VM class named **single-vm-gpu** in Table 5**.**
2. Follow the instructions here to create a namespace on the Supervisor Cluster, and configure the **Permissions**, **Storage**, **Capacity**, and **Usage**.
3. Follow the instructions here to create a local content library for VM Service.
4. Download **ubuntu-20.04-vgpu.ova** from *our GitHub repo* and import it to the local content library.

```
for i in `seq 1 9`; do wget
https://raw.githubusercontent.com/vmware-ai-labs/VMware-generative-ai-reference-architecture/main/vSphere-and-
TKG-config-files/TKGs/ubuntu-template/ubuntu-20.04-vgpu.zip.00$i; done
for i in `seq 10 22`; do wget
https://raw.githubusercontent.com/vmware-ai-labs/VMware-generative-ai-reference-architecture/main/vSphere-and-
TKG-config-files/TKGs/ubuntu-template/ubuntu-20.04-vgpu.zip.0$i; done
cat ubuntu-20.04-vgpu.zip.0* > ubuntu-20.04-vgpu.zip
unzip ubuntu-20.04-vgpu.zip
```

5. Follow the instructions here to associate the local content library with the namespace created in step 2.
6. Follow the instructions here to associate the custom VM class with the namespace created in step 2.
7. Proceed to provision a single node VM for learning use case by using the custom VM class and following the instructions in Deploying a Stand-Alone VM in vSphere with Tanzu. In this solution, we created the single node VM by using this YAML file.
8. Access the single node VM via SSH after getting the IP address. And follow the instructions in NVIDIA vGPU Driver Step-by-Step Installation Process to install the NVIDIA vGPU Driver.

```
kubectl describe vmservice vm-basic-ssh | grep Ip
```

Running LLM Tasks on vSphere with Tanzu Kubernetes

Overview

After you have deployed and configured vSphere and Tanzu Kubernetes infrastructures according to the Deployment and Configuration section, your environment is ready to run different types of tasks related to LLMs and GenAI. For that purpose, we provide two working examples of the most frequent used ones: LLM fine-tuning and LLM inference. The intention of the working examples is twofold: On the one hand, it provides you with a procedure to verify the reference architecture deployment works appropriately; on the other hand, the examples show how two working examples perform those essential tasks in the LLM development lifecycle.

Task 1: LLM Fine-tuning

Introduction to Fine-tuning Task

In this section, we present a fine-tuning task example (including the instructions to set up a Python environment and the LLM fine-tuning code) to show you how to use the VMware GenAI reference architecture to perform one of the most computing-intensive tasks in the LLM development lifecycle.

The open-source community keeps releasing LLMs which you can retrieve from the popular *Hugging Face Hub*. Examples of these models are *Falcon-40B* and *Falcon-7B*, which rank close to the top of the *Open LLM Leaderboard* at the time of this writing. The Falcon pre-trained models belong to the foundational type of LLMs, which can get further trained via fine-tuning to improve their capabilities to serve specific tasks and knowledge domains.

Using LoRA and RLHF

To simplify the LLM fine-tune process, we rely on parameter-efficient fine-tuning (PEFT) methods which minimize the computational resources and time required for fine-tuning. One popular technique for parameter-efficient fine-tuning is **Low-Rank Adaptation** (LoRA). LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by three times. We utilize the LoRA implementation included the Hugging Face PEFT library.

Another common element of the LLM fine-tuning process is Reinforcement Learning from Human Feedback (RLHF) which enables LLMs to learn from human expertise and domain-specific knowledge, leading to improved performance on specific tasks. The iterative nature of RLHF allows the model to adapt and refine its behavior based on evaluators' feedback, resulting in more accurate and contextually appropriate responses. In our example we use Hugging Face supervised fine-tuning (or SFT for short) which implements RLHF based on a series of prompt completion examples provided by the Guanaco dataset which we use to teach the Falcon models to answer questions asked by humans in a conversational manner.

Accessing the Python Code Repository

As a complementary resource we have published a GitHub repo that provides the Python code required to fine-tune Falcon-7B using a single A100 (40G) GPU and Falcon-40B on two of those GPUs. As previously mentioned, the scripts leverage the Hugging Face's implementation of LoRA (part of the PEFT package) and the bits and bytes library (by Tim Dettmers) to load the models using 16-bit or 8-bit quantization.

The Python code is presented as a Jupyter notebook where we explain all the steps involved in the LLM fine-tuning process, which can be broken down into the following sections:

- How to stack the execution of Hugging Face libraries (PEFT/LoRA, SFT trainer) to load the Falcon LLMs and the instruction datasets required for the fine-tuning task.
- A description of the list of parameters required for LLMs loading and training.
- The configuration of LangChain pipelines to use the fine-tuned model for prompt completion tasks.

Requirements

A VM with a fresh installation of Ubuntu Desktop 22.04 with the following requirements, or this VM can be created by vSphere with Tanzu VM Service. For more details, refer to the Creating VM Service for Single Node Learning section of this document.

- One or two NVIDIA A100 (40GB) GPUs attached either as vGPU or DirectPath I/O devices. You can use other NVIDIA GPUs with less memory but that might limit the size of the model you will be able to load.
- 64GB of CPU RAM
- 16 vCPU
- 500GB of disk storage. Notice that the more model checkpoints you decide to keep, the more storage space you will need.

- Internet connectivity to download software packages, LLM models and datasets

Ensure the following **Advanced Configuration Parameters** are configured in the VM:

- pciPassthru.64bitMMIOSizeGB=512
- pciPassthru.use64bitMMIO=TRUE

If the VM is configured with vGPU(s), make sure the following **Advanced Configuration Parameters** are also configured to enable Unified Memory for NVIDIA GPU:

- pciPassthru[*vGPU-ID*].cfg.enable_uvm=1

For example, if there are four vGPUs configured in the VM, these parameters should be set:

- pciPassthru0.cfg.enable_uvm=1
- pciPassthru1.cfg.enable_uvm=1
- pciPassthru2.cfg.enable_uvm=1
- pciPassthru3.cfg.enable_uvm=1

Python Environment Setup

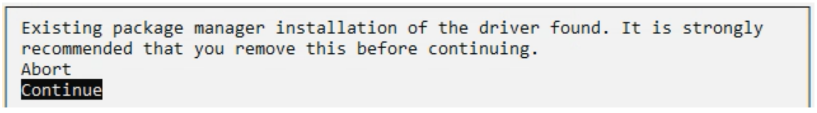
Installing NVIDIA Grid GPU Drivers and CUDA 11.8

First refer to NVIDIA vGPU Driver Step-by-Step Installation Process for Ubuntu 22.04 desktop.

Next, you need to download the CUDA 11.8 toolkit and install it. Perform the following steps:

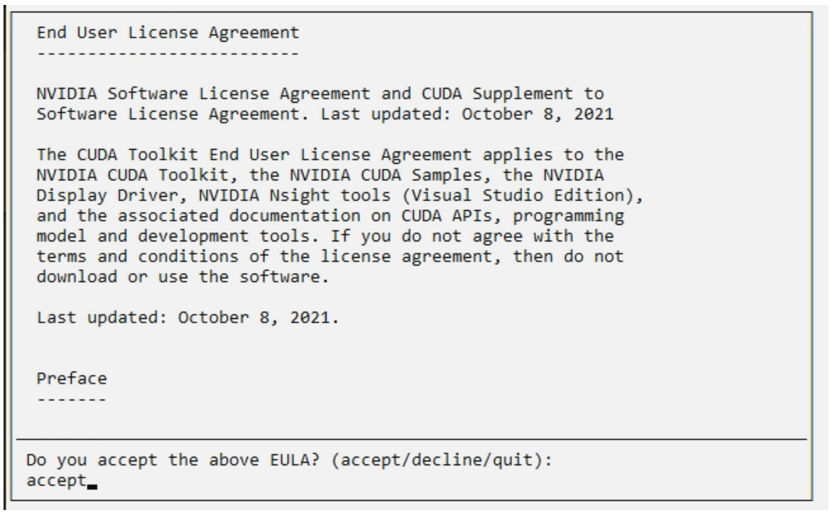
```
# Download the binaries for Ubuntu 22.04

wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda_11.8.0_520.61.05_linux.run

# Execute the run file

sudo sh cuda_11.8.0_520.61.05_linux.run
```

The CLI command will start a text-based dialog interface. You might get a warning like this:



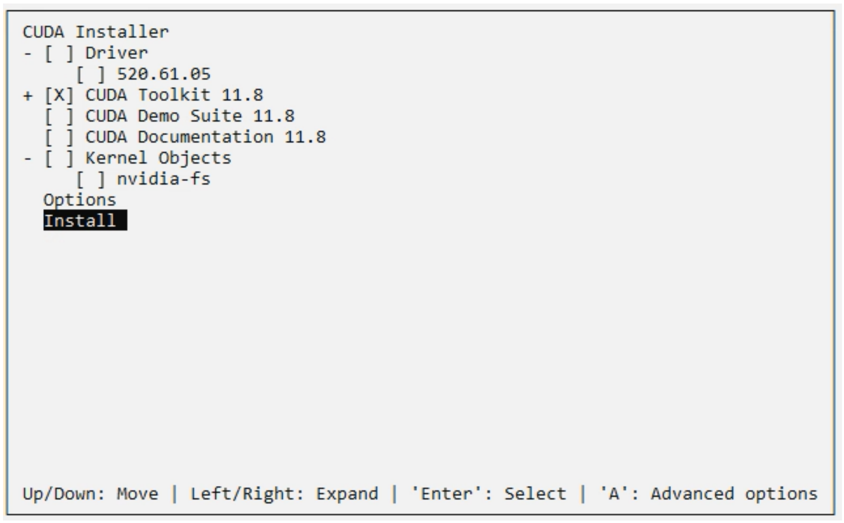
```
Existing package manager installation of the driver found. It is strongly
recommended that you remove this before continuing.
Abort
Continue
```

Using the keyboard arrows, select **Continue** and hit **Enter**. Next, you need to accept the EULA to continue.



```
End User License Agreement
--------------------------

NVIDIA Software License Agreement and CUDA Supplement to
Software License Agreement. Last updated: October 8, 2021

The CUDA Toolkit End User License Agreement applies to the
NVIDIA CUDA Toolkit, the NVIDIA CUDA Samples, the NVIDIA
Display Driver, NVIDIA Nsight tools (Visual Studio Edition),
and the associated documentation on CUDA APIs, programming
model and development tools. If you do not agree with the
terms and conditions of the license agreement, then do not
download or use the software.

Last updated: October 8, 2021.


Preface
-------

Do you accept the above EULA? (accept/decline/quit):
accept
```

Then use the keyboard to move down the screen and using the space bar, deselect the *Driver*, the *CUDA Demo Suite* and the *CUDA documentation*. Then move to the **Install** option and hit **Enter** as shown next.



```
CUDA Installer
- [ ] Driver
     [ ] 520.61.05
+ [X] CUDA Toolkit 11.8
  [ ] CUDA Demo Suite 11.8
  [ ] CUDA Documentation 11.8
- [ ] Kernel Objects
     [ ] nvidia-fs
  Options
  Install




Up/Down: Move | Left/Right: Expand | 'Enter': Select | 'A': Advanced options
```

After the installer finishes, add a new line to /etc/ld.so.conf with the */usr/local/cuda-11.8/lib64* entry. Then execute the following command:

```
# Add new line to /etc/ld.so.conf with the /usr/local/cuda-11.8/lib64, then execute the following command
echo "/usr/local/cuda-11.8/lib64" | sudo tee -a /etc/ld.so.conf
sudo ldconfig
```

CUDA will get installed under */usr/local*

```
ls /usr/local

# cuda  cuda-11.8  etc  games  include  lib  man  sbin  share  src
```

Miniconda Installation Steps

We recommend the use of Miniconda as the Python package management system over the default distributions embedded in the OS. Perform the following shell commands to set up a Python environment.

```
## Installing Miniconda

# Downloading the latest Miniconda installer for Linux.

wget -nc https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# Perform a Miniconda silent installation

bash ./Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda

# Add Conda activation, assuming you use bash as SHELL

eval "$($HOME/miniconda/bin/conda shell.bash hook)"

# With this activated shell, install conda's shell functions

conda init
```

Python Virtual Environment Setup

Clone the git repository for the Jupyter notebook that contains the LLM fine-tune code:

```
## Cloning the git repo

# Verify git is installed

git –version

# If git is not installed, install it with these two commands

sudo apt update

sudo apt install git
```

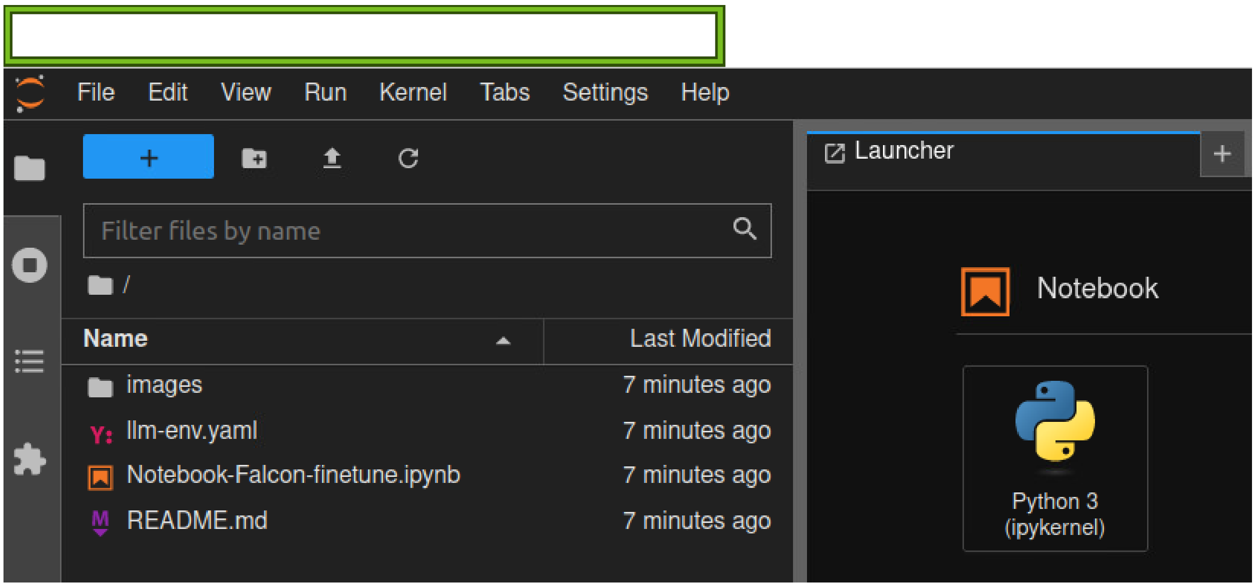
```
# Clone the git repo containing the fine-tune Jupyter notebook

git clone https://github.com/vmware-ai-labs/VMware-generative-ai-reference-architecture.git

# Enter the repo's root directory

cd VMware-generative-ai-reference-architecture/LLM-fine-tuning-example/
```

Then run the following commands to create a Conda virtual environment:

```
## Setting-up the virtual env for LLM tasks
# Create the conda virtual env u
conda env create -f llm-env.yaml
# Create the virtual env using a conda dependency specification
#  - The package versions in the YAML file have been tested by our experiments
conda activate llm-env
# OPTIONAL: login to wandb.ai using the CLI
#   - The wandb.ai dashboard allows you to follow the training process online.
#   - You'll need free account and an API key
#   - See details at https://docs.wandb.ai/quickstart
wandb login
# Start Jupyter-lab session
jupyter-lab --ip  VM_IP
```

Running the Falcon LLM Fine-tuning Job

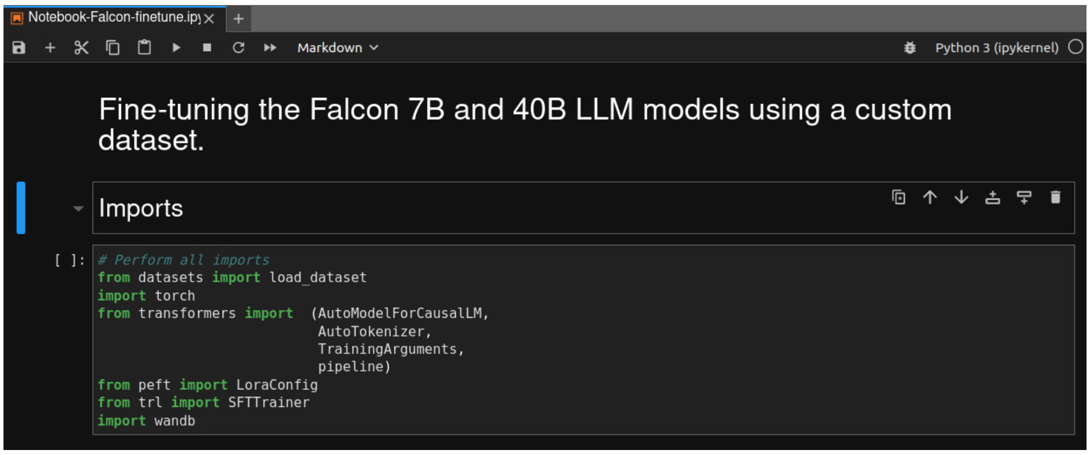
Next, your web browser will show a JupyterLab session like the one shown next. Double click on the Notebook-Falcon-finetune.ipynb file to open the notebook and get ready to go over the Falcon LLM fine-tune process. Notice that we switched the JupyterLab theme to dark mode from the **Settings** menu.



Name	Last Modified
images	7 minutes ago
llm-env.yaml	7 minutes ago
Notebook-Falcon-finetune.ipynb	7 minutes ago
README.md	7 minutes ago

Figure 21: Launch Jupyter-Lab

After the notebook is open, you can follow the notebook annotations that explain the LLM fine-tune process and the role of each component of it.



Fine-tuning the Falcon 7B and 40B LLM models using a custom dataset.

Imports

```
[ ]: # Perform all imports
from datasets import load_dataset
import torch
from transformers import  (AutoModelForCausalLM,
                           AutoTokenizer,
                           TrainingArguments,
                           pipeline)
from peft import LoraConfig
from trl import SFTTrainer
import wandb
```

Figure 22: Open Fine-tuning Notebook

Running all the cells in the Notebook-Falcon-finetune.ipynb script gives you all you need to verify the whole GenAI stack is properly configured and operational. Here is the list of things to watch for to verify the system works as expected.

Monitoring the Training Process

Cell #7 from the notebook runs the training loop. If you keep the setting that makes WandB the logging tool, you can follow the evolution of the training job online by using the random URL (hosted at wandb.ai) provided in the cell’s output. Here is the list of things to watch:

- You need to verify is that the validation loss metric decreases as the number of training steps increases. It should look like this:

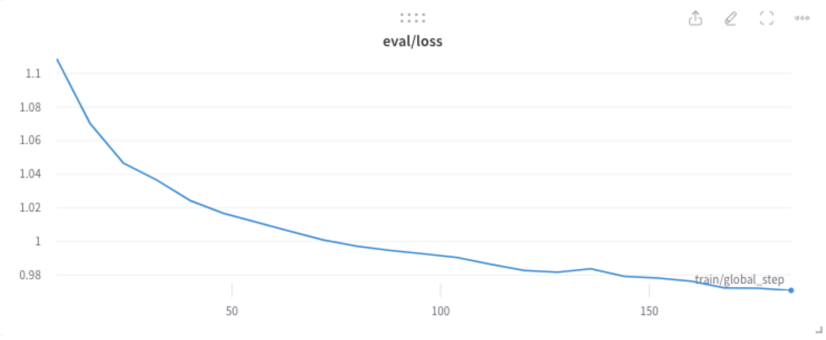
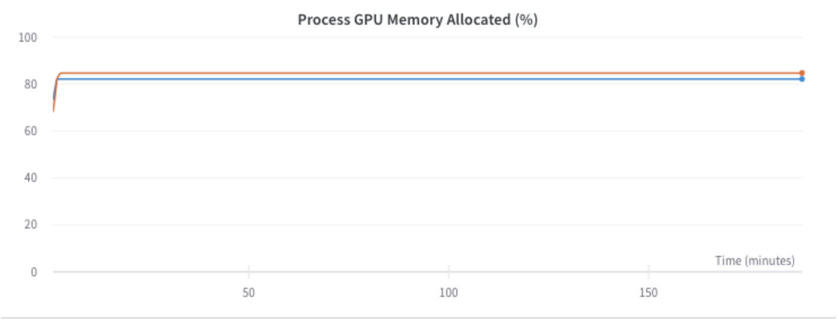


Figure 23: Evaluation Loss Metric

- When training the Falcon 40B LLM (loaded in 8-bits) on 2 x A100 (40GB) GPUs, the GPU memory and process would look like this (blue line represents GPU:0 and the orange line represents GPU:1):



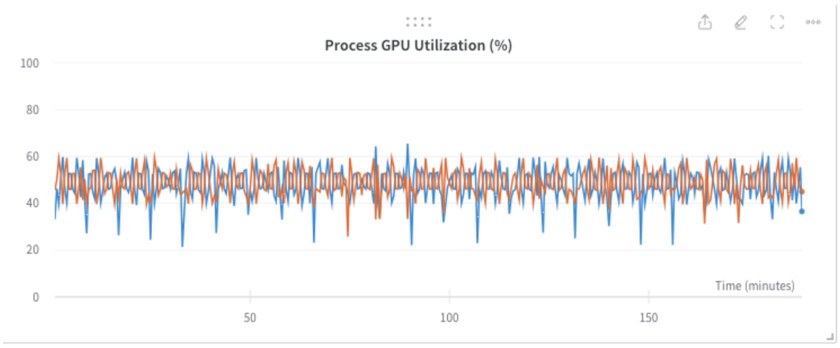
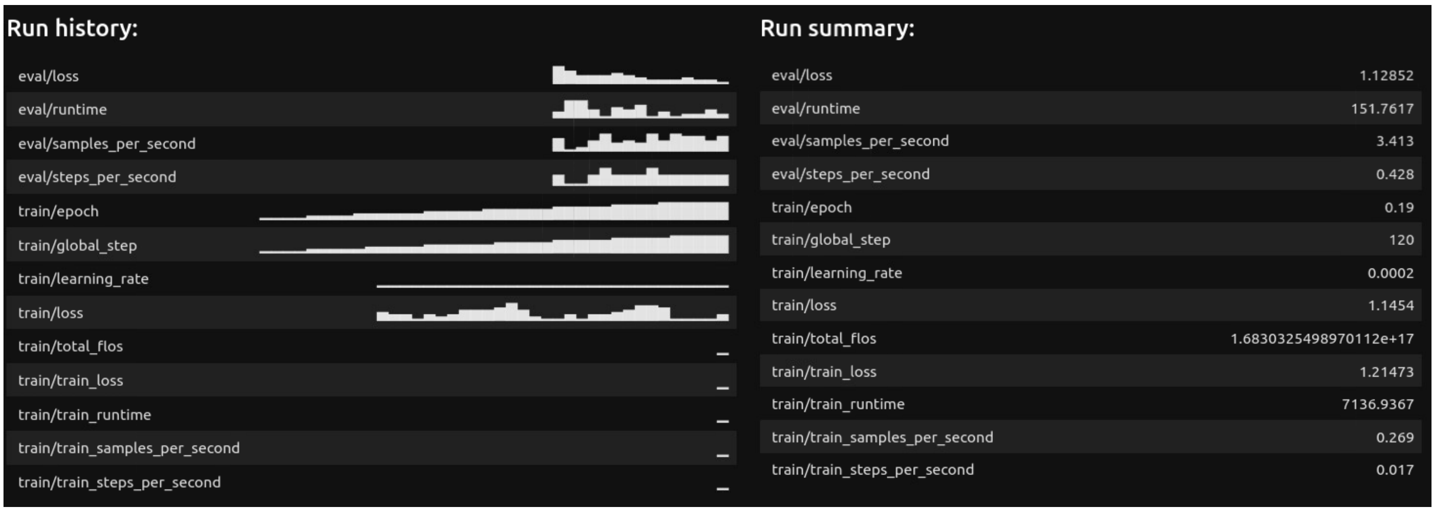


Figure 24: GPU Memory Demonstration

At the end of the training process, WandB will provide a run summary as shown below, which demonstrates a run of 120 training steps. This is also part of cell #7's output.



Run history:

eval/loss
eval/runtime
eval/samples_per_second
eval/steps_per_second
train/epoch
train/global_step
train/learning_rate
train/loss
train/total_flos
train/train_loss
train/train_runtime
train/train_samples_per_second
train/train_steps_per_second

Run summary:

eval/loss	1.12852
eval/runtime	151.7617
eval/samples_per_second	3.413
eval/steps_per_second	0.428
train/epoch	0.19
train/global_step	120
train/learning_rate	0.0002
train/loss	1.1454
train/total_flos	1.6830325498970112e+17
train/train_loss	1.21473
train/train_runtime	7136.9367
train/train_samples_per_second	0.269
train/train_steps_per_second	0.017

Figure 25: Validation Running Summary

Finding the Best Model Checkpoints

After the training (fine-tune) process is over, the trainer will save the best and the last checkpoints at the *results* directory so you should see couple of sub-directories named after the checkpoint number they contain. For instance, after 120 training steps, cell #7 could output:

```
Best check point path is ./results/checkpoint-120
```

Running Prompt Completions Locally

The last test is about using the best LLM checkpoint to run several prompt completions. Cell #9 submits a prompt to the LLM to explain a fundamental scientific concept. The LLM acts as a knowledge base.

```
prompt = f"""

The following is a friendly conversation between a Human and an Assistant. The Assistant is

concise and explains concepts in an accessible manner.

Human:  Explain the concept of magnetism.

Assistant:

""".strip()

completion = chat_llm(prompt)

print(f"Assistant:\n{completion}")

Assistant:

Magnetism is one of the four fundamental forces that governs our universe, along with gravity, electromagnetism, and
```

```
the strong nuclear force. It is responsible for many phenomena we observe on Earth, such as how magnets attract or
repel each other, why compass needles point north, and even how electricity flows through wires. In short, it is what
makes things stick together!
```

In addition, cell #10 runs a one-shot prompt asking the LLM to function as a reasoning engine and a coding assistant to solve a basic math problem and write a Python function to calculate the solution:

```
prompt = """

Human: At a store, shoes cost shoe_cost pair and socks cost sock_cost per pair.

If a customer buys shoe_p pairs of shoes and  sock_p pairs of socks, what is the total cost of the purchase?

Write a Python function that returns the answer.

Assistant:

def store_cost(shoe_cost, shoe_p, sock_cost, sock_p):

  return (shoe_cost * shoe_p) + (sock_cost * sock_p)

Human: At the cinema, tickets for adults cost adult_fee and tickets for children cost child_fee

If a family with num_adult adults and num_child children go to the movies, what is the total cost for that family?

Write a Python function that returns the answer.

Assistant:

""".strip()

completion = chat_llm(prompt)

print(f"Assistant:\n{completion}")

Assistant:

def movie_cost(adult_fee, child_fee, num_adult, num_child):

    return (num_adult * adult_fee) + (num_child * child_fee)
```

Task 2: LLM Inference

Introduction to Ray and vLLM

Ray is an open-source distributed computing framework designed for modern data-driven and AI-powered applications. It offers benefits such as streamlining large-scale computations across distributed clusters.

Ray provides the following benefits:

- Scalability: Efficient scaling across clusters for handling large-scale computations.
- Flexibility: Supports both task and data parallelism to suit specific needs.
- Fault-tolerance: Ensures computation continues despite failures.
- Python-based: Developed in Python for ease of use.
- High-performance: Optimized for speed and efficiency.
- Resource management: Handles allocation and management of resources.

Ray's working mechanism is as follows:

- Task submission: Users submit Python functions as tasks, defining computations and data dependencies.
- Distributed execution: Ray breaks down tasks into sub-tasks and schedules them across the cluster.
- Task synchronization: Manages data dependencies between tasks.
- Fault tolerance: Monitors execution and reschedules failed tasks.
- Resource management: Optimizes resource allocation based on workload.

In the Ray framework, there are two types of nodes:

- Ray head node: Acts as the central coordinator, managing the system state, scheduling tasks, and overseeing resource allocation.
- Ray worker nodes: Serves as computation units, executing tasks, and storing data.

The Ray head node maintains a global state that tracks all worker nodes, resources, and submitted tasks. Worker nodes execute the scheduled tasks, manage data, and synchronize data dependencies. One notable feature of Ray is its support for autoscaling, dynamically adjusting the number of worker nodes based on workload, which ensures efficient resource utilization and cost.

The Ray head and worker nodes' configuration we use in this reference architecture is based on the following settings:

Table 11: Ray Head and Worker Node Configuration

Role	Ray Version	Docker Image	Replicas	CPU Allocation	Memory Allocation	#of GPU
Head Node	2.6.2	rayproject/ray:2.6.2-py310-gpu	1	32	32GB	1
Worker Nodes	2.6.2	rayproject/ray:2.6.2-py310-gpu	1-4	32	32GB	1 per replica

From the multiple functions available in Ray, we leverage *Ray Serve* (or *Serve*), a scalable model serving library for building online inference APIs.

In addition to Ray Serve, we use *vLLM*, an open-source library for fast LLM inference and serving. vLLM utilizes Paged Attention, a new attention algorithm that effectively manages attention keys and values. The vLLM authors claim it delivers up to 24x higher throughput than Hugging Face Transformers without requiring any model architecture changes. You can learn more about vLLM integration with Ray from the Anyscale article *How continuous batching enables 23x throughput in LLM inference while reducing p50 latency*.

Some of the key vLLM features are:

- Continuous batching of incoming requests.
- Seamless integration with LLMs available from the Hugging Face model repository.
- High throughput serving with various decoding algorithms, including parallel sampling, beam search, and more.
- Tensor parallelism support for distributed inference.
- Streaming outputs.
- OpenAI-compatible API server.

Setting up vLLM as a Ray Serve Application

To deploy vLLM as an API service on Ray Serve you need to follow a simple setup procedure. In our example, we deploy Ray on Tanzu Kubernetes using Kuberay, an open-source Kubernetes operator that simplifies the deployment and management of Ray applications on Kubernetes. It offers several key components:

KubeRay core: This is the official, fully maintained component of KubeRay that provides three custom resource definitions, RayCluster, RayJob, and RayService. These resources are designed to help you run a wide range of workloads with ease.

Here are the steps you must complete to run the vLLM service on Ray Serve. In our example vLLM serves the Falcon-7B model after pulling it from Hugging Face repository. All the scripts and YAML configuration files required to complete the setup are accessible from the vllm_examples GitHub repository. The scripts are based on the examples provided by the Anyscale team at continuous-batching-llm-inference.

Requirements:

- First you need to have a Kubernetes (K8s) cluster up and running.
- In our case, we rely on a TKC deployed following the instructions from the Deployment and Configuration section of this document.

- Ensure you have Helm installed in your environment.

Deploying a vLLM Service on Ray Serve

1. Set `ClusterRoleBinding` to run a privileged set of workloads. This will prevent a Kuberay operator installation failure.

```
kubectl create clusterrolebinding default-tkg-admin-privileged-binding --clusterrole=psp:vmware-system-privileged --group=system:authenticated
```

2. Deploy Kuberay in your K8s cluster. More details at *KubeRay Operator install docs*.

```
# Add the Kuberay Helm repo.

helm repo add kuberay https://ray-project.github.io/kuberay-helm/

# Install both CRDs and KubeRay operator v0.6.0.

helm install kuberay-operator kuberay/kuberay-operator --version 0.6.0

# NAME: kuberay-operator

# LAST DEPLOYED: Thu Aug 10 12:41:07 2023

# NAMESPACE: kuberay

# STATUS: deployed

# REVISION: 1

# TEST SUITE: None

# Check the KubeRay operator pod in the `default` namespace.

kubectl get pods

# NAME                                READY   STATUS    RESTARTS   AGE

# kuberay-operator-6b68b5b49d-jppm7   1/1     Running   0          6m40s
```

3. Pull the Ray Serve manifest from GitHub and apply it.

```
# Pull the ray-service.vllm.yaml manifest (from this repo) from the raw URL

wget -L
https://raw.githubusercontent.com/vmware-ai-labs/VMware-generative-ai-reference-architecture/main/LLM-serving-wt-vLLM-and-RayServe-example/ray-service.vllm.yaml

# Create a Ray Serve cluster using the manifest

kubectl apply -f ray-service.vllm.yaml

# Check the Ray pods got created

kubectl get pods

# The Ray cluster starts to create the head and worker pods

# NAME                                             READY   STATUS              RESTARTS   AGE

# kuberay-operator-6b68b5b49d-jppm7                1/1     Running             0          23m

# vllm-raycluster-c9wk4-head-gw958                 0/1     ContainerCreating   0          67s

# vllm-raycluster-c9wk4-worker-gpu-group-wl7k2     0/1     Init:0/1            0          67s

# After several minutes, the Ray cluster should be up and running

kubectl get pods

# NAME                                             READY   STATUS    RESTARTS   AGE
# kuberay-operator-6b68b5b49d-jppm7              1/1     Running   0          39m

# vllm-raycluster-c9wk4-head-gw958               1/1     Running   0          17m

# vllm-raycluster-c9wk4-worker-gpu-group-wl7k2   1/1     Running   0          17m

# The vLLM service will get exposed as a LoadBalancer. In the next example

# The vLLM API service (vllm-serve-svc) gets exposed over http://172.29.214.16:8000.

# That is the URL you need to use to make prompt completion requests.

 kubectl get svc

# NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)

# kuberay-operator                  ClusterIP      10.105.14.110    <none>          8080/TCP

# vllm-head-svc                     LoadBalancer   10.100.208.111   172.29.214.17   10001...

# vllm-raycluster-c9wk4-head-svc    LoadBalancer   10.103.27.23     172.29.214.16   10001...

# vllm-serve-svc                    LoadBalancer   10.104.242.187   172.29.214.18   8000...
```

You can use the *vllm-raycluster-c9wk4-head-svc* IP on port 8265 (in our example is *http://172.29.214.16:8265. Yours could change*) to access the Ray cluster dashboard to monitor the Ray cluster status and activity. Notice *ray-service.vllm.yaml* manifest has the following section that defines the vLLM service deployment:

```
spec:
  serviceUnhealthySecondThreshold: 3600 # Health check threshold for service. Default value is 60.
  deploymentUnhealthySecondThreshold: 3600 # Deployments health check threshold. Default value 60.
  serveConfigV2: |
    applications:
      - name: vllm
        import_path: vllm_falcon_7b:deployment
        runtime_env:
          working_dir: "https://github.com/vecorro/vllm_examples/archive/refs/heads/main.zip"
          pip: ["vllm==0.1.3"]
```

Here are some remarks about the service definition:

- We increase `serviceUnhealthySecondThreshold` and `deploymentUnhealthySecondThreshold` to give Ray sufficient time to install vLLM on a virtual working environment. vLLM can take >15 minutes to install.
- `working_dir` is set to the URL of the compressed version of the Github repo. Ray will use this URL to pull the Python code that implements the vLLM service.
- We use vLLM 0.1.3 to create the Ray working environment.
- `import_path` is set to the proper `module:object` for Ray Serve to get the service definition. In this case the `module` is the `vllm_falcon_7b.py` Python script and `deployment` is a `serve.deployment.bind()` object type defined inside that script.

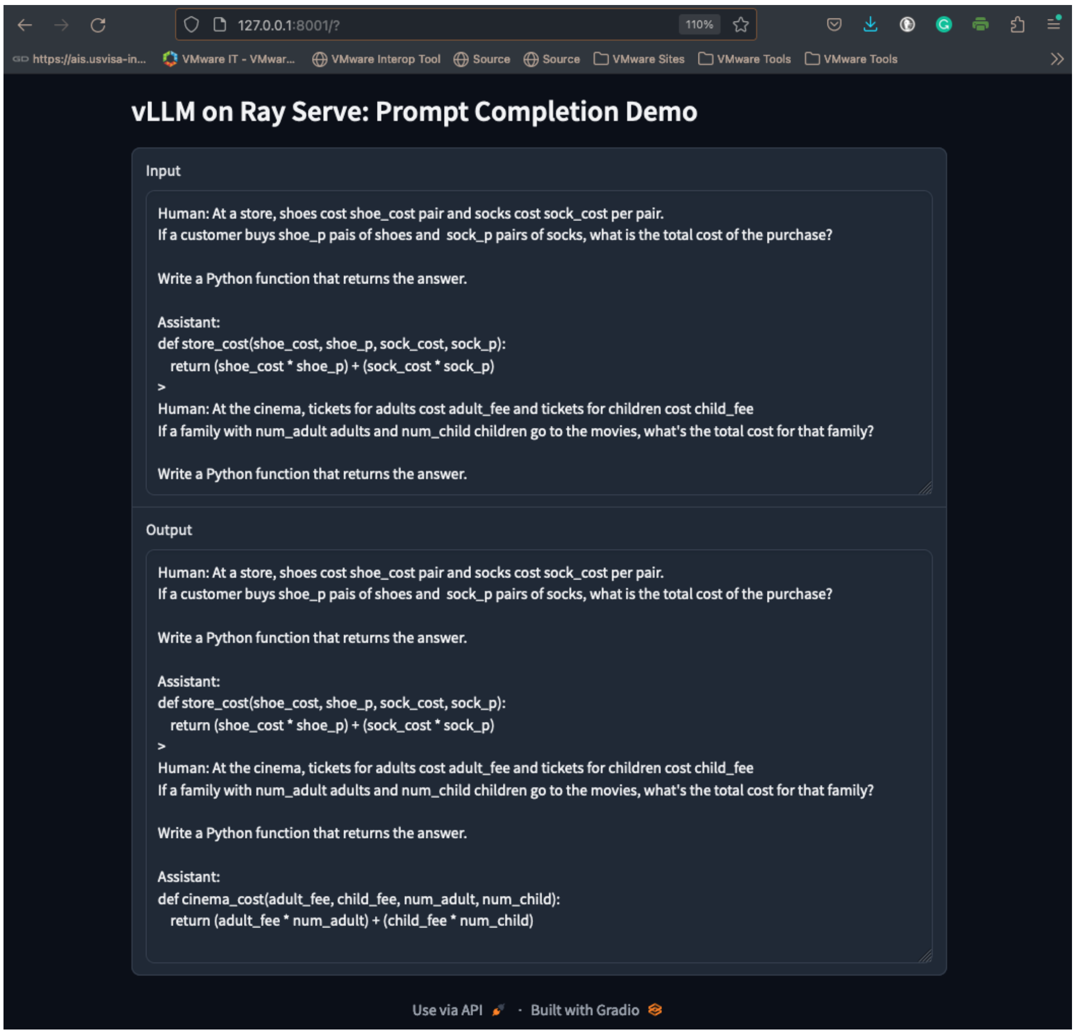
vLLM on Ray Serve Deployment Validation

Similar to the fine-tune example, we can validate our deployment by sending some prompt completion requests. The GitHub repo provides a script called *gradio_webserver.py* script to serve prompt completions from a web UI. The command line instructions to run it are as follows:

```
# Install gradio

pip install gradio

wget
https://raw.githubusercontent.com/vmware-ai-labs/VMware-generative-ai-reference-architecture/main/LLM-serving-wt-vLLM-and-RayServe-example/gradio_webserver.py

# Replace the --model-url value with the hostname or IP address of vllm-serve-svc
```

```
python gradio_webserver.py --model-url="http://172.29.214.18:8000"

# Running on local URL:  http://localhost:8001
```

Then you open *http://localhost:8001* from your web browser. The Gradio web interface will give you a chat window to interact with the LLM. Here we show a prompt completion example where the LLM acts as a reasoning engine.



vLLM on Ray Serve: Prompt Completion Demo

Input

Human: At a store, shoes cost shoe_cost pair and socks cost sock_cost per pair.
If a customer buys shoe_p pais of shoes and sock_p pairs of socks, what is the total cost of the purchase?

Write a Python function that returns the answer.

Assistant:
def store_cost(shoe_cost, shoe_p, sock_cost, sock_p):
  return (shoe_cost * shoe_p) + (sock_cost * sock_p)
>
Human: At the cinema, tickets for adults cost adult_fee and tickets for children cost child_fee
If a family with num_adult adults and num_child children go to the movies, what's the total cost for that family?

Write a Python function that returns the answer.

Output

Human: At a store, shoes cost shoe_cost pair and socks cost sock_cost per pair.
If a customer buys shoe_p pais of shoes and sock_p pairs of socks, what is the total cost of the purchase?

Write a Python function that returns the answer.

Assistant:
def store_cost(shoe_cost, shoe_p, sock_cost, sock_p):
  return (shoe_cost * shoe_p) + (sock_cost * sock_p)
>
Human: At the cinema, tickets for adults cost adult_fee and tickets for children cost child_fee
If a family with num_adult adults and num_child children go to the movies, what's the total cost for that family?

Write a Python function that returns the answer.

Assistant:
def cinema_cost(adult_fee, child_fee, num_adult, num_child):
  return (adult_fee * num_adult) + (child_fee * num_child)

Figure 26: A Prompt Completion Example

Conclusion

Properly designing a VMware virtual infrastructure for running LLM operations like fine-tuning and inference is a critical consideration in achieving optimal performance. Inference time is a vital metric in the practical application of LLMs, and appropriately sizing the infrastructure ensures that the computational resources align with the demands of the tasks. Under-provisioning can lead to sluggish performance and slow inference times, hampering user experience and efficiency. Conversely, an over-provisioned infrastructure can handle the required computations quickly, but it may lead to wasteful excess capacity, translating into higher costs without proportional benefits.

By properly designing the virtual infrastructure, organizations can strike a delicate balance, optimizing the utilization of GPU resources without unnecessary waste. This ensures that the necessary computational power is available for demanding fine-tuning processes and real-time inference tasks while avoiding the costs associated with underutilized resources. This balance not only contributes to reducing operating expenses but also promotes a more responsive and agile environment, enabling organizations to adapt to changing needs and scale their operations seamlessly.

We would like to be closer to the customers and partners interested in implementing this or reference architecture for GenAI. We encourage you to visit the companion GitHub repository for this document and provide your feedback. We will be checking for new issues related to any problems with the code and instructions we provide, as well as new issues related to feature requests. Our goal is to keep updating this document to make it useful for everyone interested in running their GenAI workloads on VMware.

References

- VMware Cloud Foundation Documentation
- VMware vSphere Documentation
- vSphere with Tanzu Documentation
- VMware vSAN Design Guide
- VMware Validated Solutions
- vSphere Storage
- vSphere Network I/O control
- VMware Compatibility Guide
- VMware Aria Operations
- vSphere Lifecycle Manager
- Developer Ready Infrastructure for VMware Cloud Foundation
- Why Your Organization Needs a Set of Ethical Principles for AI
- vSphere 8 Expands Machine Learning Support: Device Groups for NVIDIA GPUs and NICs
- Deploy an AI-Ready Enterprise Platform on vSphere 7 Update 2
- Virtualizing HPC on VMware vSphere: Choosing a Best Method for RDMA HCA/NIC Deployment for MPI Workloads
- InfiniBand SR-IOV Setup and Performance Study on vSphere 7.x
- RoCE SR-IOV Setup and Performance Study on vSphere 7.x
- InfiniBand and RoCE DirectPath IO Setup and Performance Study on vSphere 7.x
- Enabling vSphere with Tanzu with VMware Cloud Foundation
- Enabling vSphere with Tanzu with Service Installer
- Deploying VMware Tanzu for Kubernetes Operations on vSphere with Tanzu and vSphere Distributed Switch Using Service Installer for VMware Tanzu
- NVIDIA Network Operator
- NVLink & NVSwitch: Fastest HPC Data Center Platform | NVIDIA

About the Authors

Agustin Malanco Leyva, Enrique Corro Fuentes, Fabiano Teixeira, Yuankun Fu, Ramesh Radhakrishnan, Ka Kit Wong, and Chen Wei wrote the original content of this solution reference architecture. Catherine Xu supported on the editing work of this document.

- Agustin Malanco Leyva, Staff II HPC/ML Solutions Architect, Office of the CTO in VMware
- Enrique Corro Fuentes, HPC/ML Solution Architect, Office of the CTO in VMware
- Fabiano Teixeira, Emerging Workloads Solution Architect, Office of the CTO in VMware
- Yuankun Fu, Senior Member of Technical Staff, Office of the CTO in VMware
- Ramesh Radhakrishnan, Technical Director, Office of the CTO in VMware
- Ka Kit Wong, Staff Technical Marketing Architect, Workload Technical Marketing in VMware
- Chen Wei, Director of Workload Technical Marketing in VMware
- Catherine Xu, Senior Manager of Workload Technical Marketing in VMware

Feedback

Your feedback is valuable.

To comment on this paper, contact VMware Office of the CTO at *genai_tech_content_feedback@vmware.com*.

Appendix

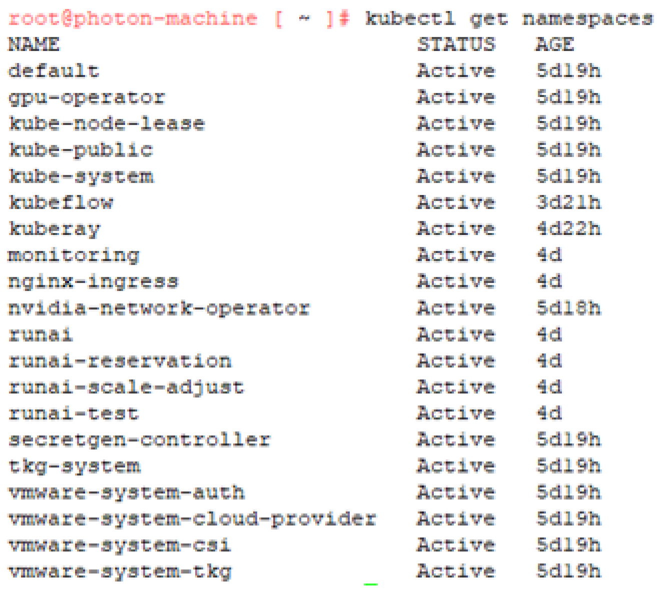
NVIDIA GPU Operator Step-by-Step Installation Process

1. Connect to your TKC with the Tanzu CLI:

```
kubectl vsphere login --server=IP_OF_YOUR_SUPERVISOR --tanzu-kubernetes-cluster-name NAME_OF_TKC_CLUSTER --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify
```

2. Create a gpu-operator namespace:

```
kubectl create namespace gpu-operator
```



```
root@photon-machine [ ~ ]# kubectl get namespaces
NAME                            STATUS   AGE
default                         Active   5d19h
gpu-operator                    Active   5d19h
kube-node-lease                 Active   5d19h
kube-public                     Active   5d19h
kube-system                     Active   5d19h
kubeflow                        Active   3d21h
kuberay                         Active   4d22h
monitoring                      Active   4d
nginx-ingress                   Active   4d
nvidia-network-operator         Active   5d18h
runai                           Active   4d
runai-reservation               Active   4d
runai-scale-adjust              Active   4d
runai-test                      Active   4d
secretgen-controller            Active   5d19h
tkg-system                      Active   5d19h
vmware-system-auth              Active   5d19h
vmware-system-cloud-provider    Active   5d19h
vmware-system-csi               Active   5d19h
vmware-system-tkg               Active   5d19h
```

3. Create an empty vGPU license configuration file:

```
sudo touch gridd.conf
```

4. After you have this file created, you will need to generate and download an NLS client license token (.tok) file, this must be done on your NVIDIA licensing server and rename the .tok file to **client_configuration_token.tok.**

5. Then create a configmap, a configmap allows you to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable. Create the licensing-configConfigMap object in the gpu-operator namespace. Both the vGPU license configuration file and the NLS client license token will be added to this ConfigMap:

```
kubectl create configmap licensing-config -n gpu-operator --from-file=gridd.conf --from-file=<path>/client_configuration_token.tok
```

You can confirm that the contents of the configmap were successfully populated by describing the configmap:

```
kubectl describe configmaps licensing-config -n gpu-operator
```

6. Create a pull secret object in the gpu-operator namespace. A secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a pod specification or in a container image. Using a secret object means that you do not need to include confidential data in your application code. We will use this secret object to pull the required image(s) from NVIDIA's private NGC registry:

```
export REGISTRY_SECRET_NAME=ngc-secret

export PRIVATE_REGISTRY=nvcr.io/nvaie

kubectl create secret docker-registry ${REGISTRY_SECRET_NAME} \

--docker-server=${PRIVATE_REGISTRY} \

--docker-username='$oauthtoken' \
--docker-password=${NGC_API_KEY} \

--docker-email='YOUREMAIL \

-n gpu-operator
```

7. Add the NVAIE Helm repository, where the password is the NGC API key for accessing the NVIDIA Enterprise Collection that you generated:

```
helm repo add nvaie https://helm.ngc.nvidia.com/nvaie \

--username='$oauthtoken' --password=${NGC_API_KEY} \

&& helm repo update
```

8. Create ClusterRoleBinding that grants access to authenticated users run a privileged set of workloads using the default PSP vmware-system-privileged:

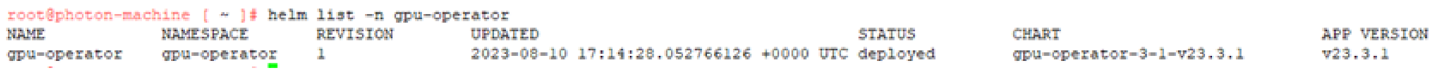
```
kubectl create clusterrolebinding psp:authenticated --clusterrole=psp:vmware-system-privileged --
group=system:authenticated
```

9. Install the NVIDIA GPU Operator with RDMA enabled with helm:

```
helm install --wait gpu-operator nvaie/gpu-operator-3-1 -n gpu-operator --set driver.repository=nvcr.io/nvaie --set
operator.repository=nvcr.io/nvaie --set driver.imagePullPolicy=Always --set migStrategy=mixed --set
driver.rdma.enabled=True
```

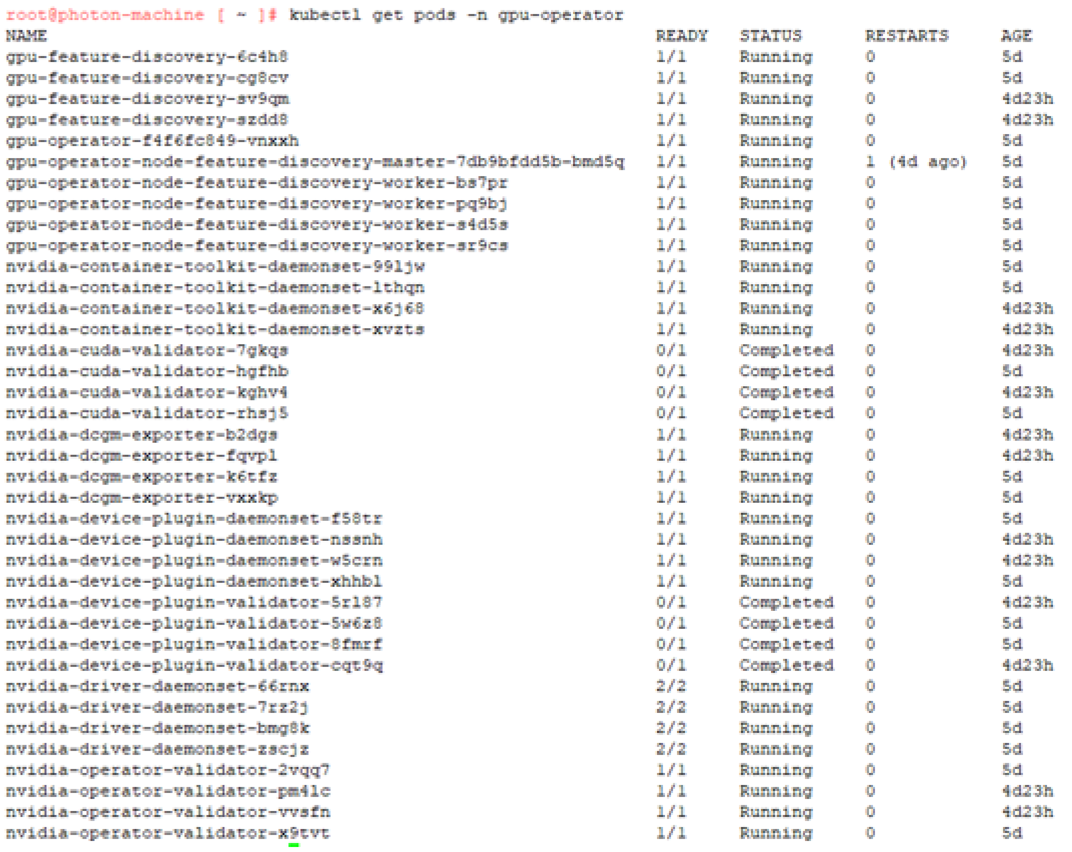
10. Confirm that both pods and the helm package are listed:

```
helm list -n gpu-operator
```



```
root@photon-machine [ ~ ]# helm list -n gpu-operator
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERSION
gpu-operator    gpu-operator    1               2023-08-10 17:14:28.052766126 +0000 UTC deployed        gpu-operator-3-1-v23.3.1        v23.3.1
```

```
kubectl get pods -n gpu-operator
```



```
root@photon-machine [ ~ ]# kubectl get pods -n gpu-operator
NAME                                                          READY   STATUS      RESTARTS      AGE
gpu-feature-discovery-6c4h8                                   1/1     Running     0             5d
gpu-feature-discovery-cg8cv                                   1/1     Running     0             5d
gpu-feature-discovery-sv9qm                                   1/1     Running     0             4d23h
gpu-feature-discovery-szdd8                                   1/1     Running     0             4d23h
gpu-operator-f4f6fc849-vnxxh                                  1/1     Running     0             5d
gpu-operator-node-feature-discovery-master-7db9bfdd5b-bmd5q   1/1     Running     1 (4d ago)    5d
gpu-operator-node-feature-discovery-worker-bs7pr              1/1     Running     0             5d
gpu-operator-node-feature-discovery-worker-pq9bj              1/1     Running     0             5d
gpu-operator-node-feature-discovery-worker-s4d5s              1/1     Running     0             5d
gpu-operator-node-feature-discovery-worker-sr9cs              1/1     Running     0             5d
nvidia-container-toolkit-daemonset-99ljw                      1/1     Running     0             5d
nvidia-container-toolkit-daemonset-lthqn                      1/1     Running     0             5d
nvidia-container-toolkit-daemonset-x6j68                      1/1     Running     0             4d23h
nvidia-container-toolkit-daemonset-xvzts                      1/1     Running     0             4d23h
nvidia-cuda-validator-7gkqs                                   0/1     Completed   0             4d23h
nvidia-cuda-validator-hgfhb                                   0/1     Completed   0             5d
nvidia-cuda-validator-kghv4                                   0/1     Completed   0             4d23h
nvidia-cuda-validator-rhsj5                                   0/1     Completed   0             5d
nvidia-dcgm-exporter-b2dgs                                    1/1     Running     0             4d23h
nvidia-dcgm-exporter-fqvpl                                    1/1     Running     0             4d23h
nvidia-dcgm-exporter-k6tfz                                    1/1     Running     0             5d
nvidia-dcgm-exporter-vxxkp                                    1/1     Running     0             5d
nvidia-device-plugin-daemonset-f58tr                          1/1     Running     0             5d
nvidia-device-plugin-daemonset-nssnh                          1/1     Running     0             4d23h
nvidia-device-plugin-daemonset-w5crn                          1/1     Running     0             4d23h
nvidia-device-plugin-daemonset-xhhbl                          1/1     Running     0             5d
nvidia-device-plugin-validator-5r187                          0/1     Completed   0             4d23h
nvidia-device-plugin-validator-5w6z8                          0/1     Completed   0             5d
nvidia-device-plugin-validator-8fmrf                          0/1     Completed   0             5d
nvidia-device-plugin-validator-cqt9q                          0/1     Completed   0             4d23h
nvidia-driver-daemonset-66rnx                                 2/2     Running     0             5d
nvidia-driver-daemonset-7rz2j                                 2/2     Running     0             5d
nvidia-driver-daemonset-bmg8k                                 2/2     Running     0             5d
nvidia-driver-daemonset-zscjz                                 2/2     Running     0             5d
nvidia-operator-validator-2vqq7                               1/1     Running     0             5d
nvidia-operator-validator-pm4lc                               1/1     Running     0             4d23h
nvidia-operator-validator-vvsfn                               1/1     Running     0             4d23h
nvidia-operator-validator-x9tvt                               1/1     Running     0             5d
```

11. Confirm the license is valid:

```
ctnname=`kubectl get pods -n gpu-operator | grep driver-daemonset | head -1 | cut -d " " -f1`
```

```
kubectl -n gpu-operator exec -it $ctnname -- /bin/bash -c "/usr/bin/nvidia-smi -q | grep -i lic"
```

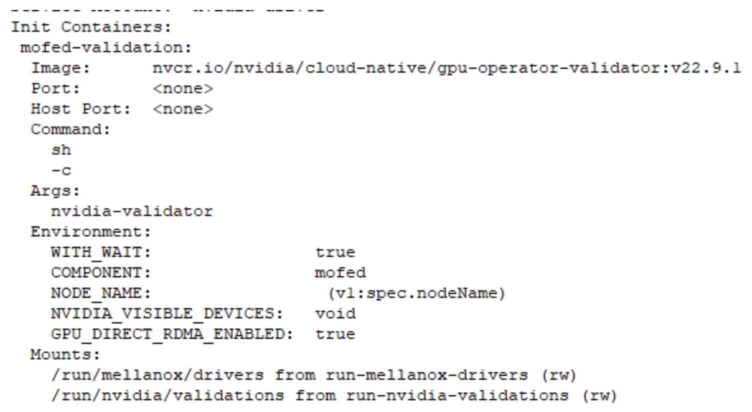
Verifying the Installation of GPUDirect with RDMA

During the installation, the NVIDIA driver daemonset runs an init container to wait on the Mellanox OFED (MOFED) drivers to be ready. This init container checks for Mellanox NICs on the worker node and ensures that the necessary kernel symbols are exported MOFED kernel drivers. After the verification is complete by the init container, the nvidia-peermem-ctr container is started inside each driver pod. At this point of the guide, we have not started installing the network operator so the init container will keep waiting for pods enabled with MOFED.

Confirm that the pod template for the driver daemonset includes the mofed-validation init container and the nvidia-driver-ctr and nvidia-peermem-ctr containers:

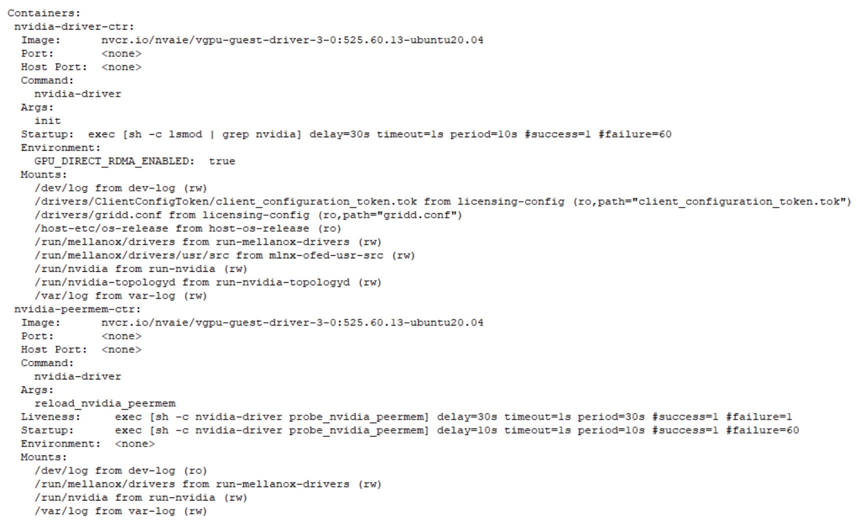
```
kubectl describe ds -n gpu-operator nvidia-driver-daemonset
```

mofed validation container:



```
 Init Containers:
  mofed-validation:
   Image:      nvcr.io/nvidia/cloud-native/gpu-operator-validator:v22.9.1
   Port:       <none>
   Host Port:  <none>
   Command:
     sh
     -c
   Args:
     nvidia-validator
   Environment:
     WITH_WAIT:                 true
     COMPONENT:                 mofed
     NODE_NAME:                  (v1:spec.nodeName)
     NVIDIA_VISIBLE_DEVICES:    void
     GPU_DIRECT_RDMA_ENABLED:   true
   Mounts:
     /run/mellanox/drivers from run-mellanox-drivers (rw)
     /run/nvidia/validations from run-nvidia-validations (rw)
```

nvidia-driver-ctr and nvidia-peermem-ctr containers



```
Containers:
 nvidia-driver-ctr:
  Image:      nvcr.io/nvaie/vgpu-guest-driver-3-0:525.60.13-ubuntu20.04
  Port:       <none>
  Host Port:  <none>
  Command:
    nvidia-driver
  Args:
    init
  Startup:  exec [sh -c lsmod | grep nvidia] delay=30s timeout=1s period=10s #success=1 #failure=60
  Environment:
    GPU_DIRECT_RDMA_ENABLED:  true
  Mounts:
    /dev/log from dev-log (rw)
    /drivers/ClientConfigToken/client_configuration_token.tok from licensing-config (ro,path="client_configuration_token.tok")
    /drivers/gridd.conf from licensing-config (ro,path="gridd.conf")
    /host-etc/os-release from host-os-release (ro)
    /run/mellanox/drivers from run-mellanox-drivers (rw)
    /run/mellanox/drivers/usr/src from mlnx-ofed-usr-src (rw)
    /run/nvidia from run-nvidia (rw)
    /run/nvidia-topologyd from run-nvidia-topologyd (rw)
    /var/log from var-log (rw)
 nvidia-peermem-ctr:
  Image:      nvcr.io/nvaie/vgpu-guest-driver-3-0:525.60.13-ubuntu20.04
  Port:       <none>
  Host Port:  <none>
  Command:
    nvidia-driver
  Args:
    reload_nvidia_peermem
  Liveness:     exec [sh -c nvidia-driver probe_nvidia_peermem] delay=30s timeout=1s period=30s #success=1 #failure=1
  Startup:      exec [sh -c nvidia-driver probe_nvidia_peermem] delay=10s timeout=1s period=10s #success=1 #failure=60
  Environment:  <none>
  Mounts:
    /dev/log from dev-log (ro)
    /run/mellanox/drivers from run-mellanox-drivers (rw)
    /run/nvidia from run-nvidia (rw)
    /var/log from var-log (rw)
```

NVIDIA Network Operator Step-by-Step Installation Process

1. Start by labeling our worker nodes with the label "worker", list your nodes on the Tanzu Cluster:

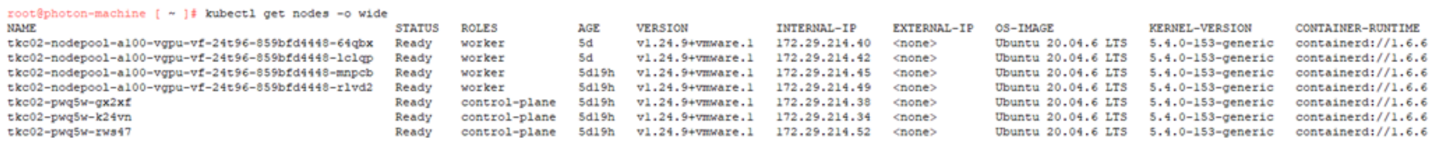
```
kubectl get nodes -o wide
```

2. Label your nodes with the following command:

```
kubectl label node tkc02-nodepool-a100-vgpu-vf-7p5p5-54c9b4dd5-7nbmk node-role.kubernetes.io/worker=worker
```

Confirm that the nodes are labeled:

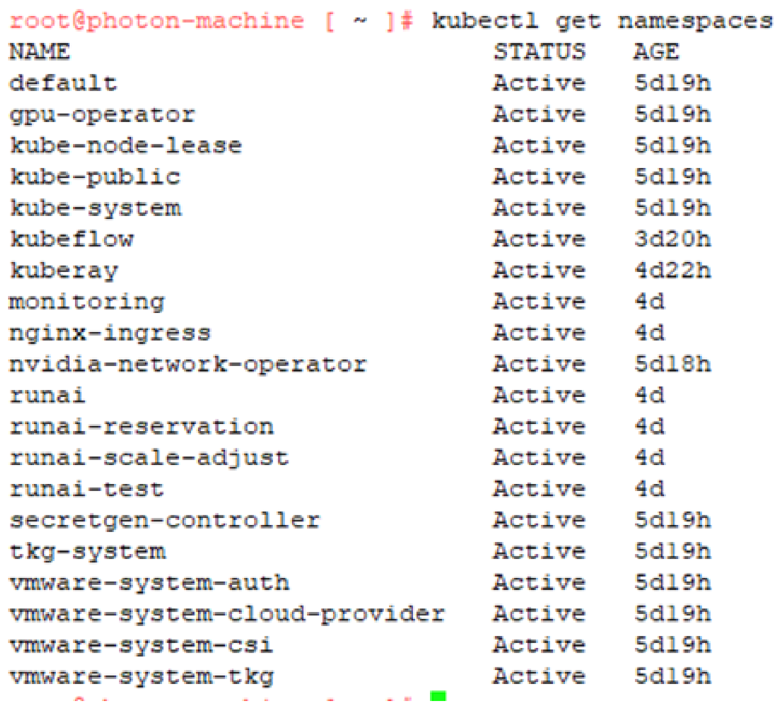
```
kubectl get nodes -o wide
```



```
root@photon-machine [ ~ ]# kubectl get nodes -o wide
NAME                                              STATUS   ROLES           AGE     VERSION              INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
tkc02-nodepool-a100-vgpu-vf-24t96-859bfd4448-64qbx   Ready    worker          5d      v1.24.9+vmware.1   172.29.214.40   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-nodepool-a100-vgpu-vf-24t96-859bfd4448-lclqp   Ready    worker          5d      v1.24.9+vmware.1   172.29.214.42   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-nodepool-a100-vgpu-vf-24t96-859bfd4448-mnpcb   Ready    worker          5d19h   v1.24.9+vmware.1   172.29.214.45   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-nodepool-a100-vgpu-vf-24t96-859bfd4448-rlvd2   Ready    worker          5d19h   v1.24.9+vmware.1   172.29.214.49   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-pwq5w-gx2xf                                    Ready    control-plane   5d19h   v1.24.9+vmware.1   172.29.214.38   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-pwq5w-k24vn                                    Ready    control-plane   5d19h   v1.24.9+vmware.1   172.29.214.34   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
tkc02-pwq5w-rws47                                    Ready    control-plane   5d19h   v1.24.9+vmware.1   172.29.214.52   <none>        Ubuntu 20.04.6 LTS   5.4.0-153-generic   containerd://1.6.6
```

3. After the nodes are labeled, we need to create a namespace for NVIDIA's network operator:

```
kubectl create namespace nvidia-network-operator
```



```
root@photon-machine [ ~ ]# kubectl get namespaces
NAME                           STATUS   AGE
default                        Active   5d19h
gpu-operator                   Active   5d19h
kube-node-lease                Active   5d19h
kube-public                    Active   5d19h
kube-system                    Active   5d19h
kubeflow                       Active   3d20h
kuberay                        Active   4d22h
monitoring                     Active   4d
nginx-ingress                  Active   4d
nvidia-network-operator        Active   5d18h
runai                          Active   4d
runai-reservation              Active   4d
runai-scale-adjust             Active   4d
runai-test                     Active   4d
secretgen-controller           Active   5d19h
tkg-system                     Active   5d19h
vmware-system-auth             Active   5d19h
vmware-system-cloud-provider   Active   5d19h
vmware-system-csi              Active   5d19h
vmware-system-tkg              Active   5d19h
```

4. Now create a secret on the namespace that was just created, this will allow to pull the network operator images while doing the installation via helm:

```
kubectl create secret docker-registry ngc-image-secret -n nvidia-network-operator --docker-server=nvcr.io --docker-username='$oauthtoken' --docker-password='YOUR NVIDIA API KEY' --docker-email='YOUR NVIDIA NGC EMAIL'
```

Check out the network operator *values.yaml* file that provides specific customizations for the network operator helm chart.

As you can see, we use the secret that we just created before in this values YAML file. For more information about the options on this file and values files for other NVIDIA network operator use cases, consult NVIDIA's documentation.

5. Create the values YAML file:

```
touch values.yaml
```

6. Now create the NVIDIA repo:

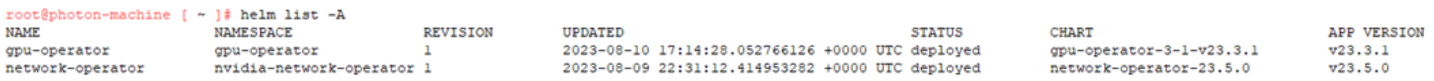
```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia  \

--username='$oauthtoken' --password=${NGC_API_KEY} \

&& helm repo update
```

7. Once created, we will install the helm chart, we will specify the latest version at the time of writing (1.23.5) and the values.yaml file as input:

```
helm install network-operator nvidia/network-operator -n nvidia-network-operator --create-namespace --version v23.5.0 -f values.yaml --debug
```

8. Verify that the chart is showing as installed:

```
helm list -A
```

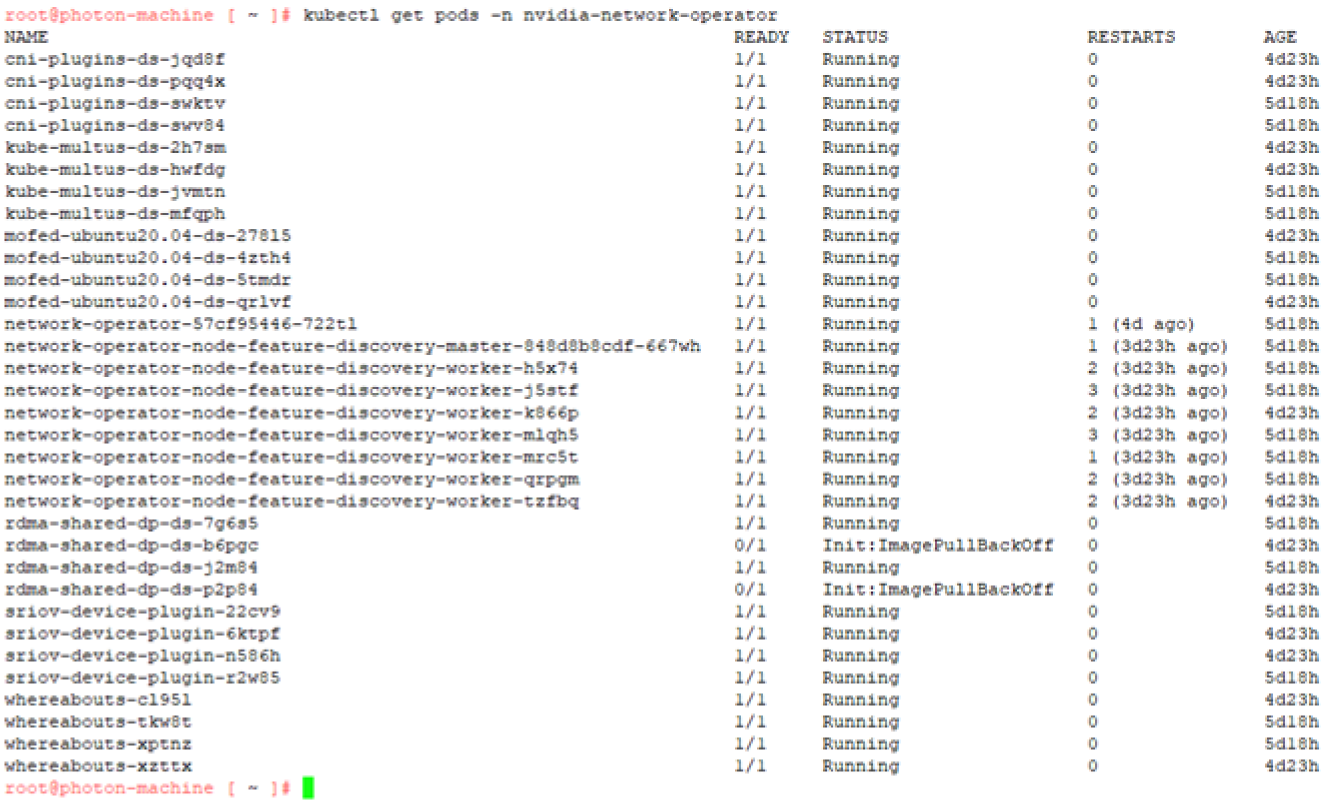


```
root@photon-machine [ ~ ]# helm list -A
NAME                NAMESPACE                 REVISION   UPDATED                                   STATUS     CHART                         APP VERSION
gpu-operator        gpu-operator              1          2023-08-10 17:14:28.052766126 +0000 UTC deployed   gpu-operator-3-1-v23.3.1      v23.3.1
network-operator    nvidia-network-operator   1          2023-08-09 22:31:12.414953282 +0000 UTC deployed   network-operator-23.5.0       v23.5.0
```

The creation of the pods that are part of the NVIDIA network operator takes 10 to 15 minutes; you can monitor the process with:

```
watch kubectl get pods -n nvidia-network-operator
```

A successful installation of the network operator should be like this:



```
root@photon-machine [ ~ ]# kubectl get pods -n nvidia-network-operator
NAME                                                              READY   STATUS                  RESTARTS        AGE
cni-plugins-ds-jqd8f                                              1/1     Running                 0               4d23h
cni-plugins-ds-pqq4x                                              1/1     Running                 0               4d23h
cni-plugins-ds-swktv                                              1/1     Running                 0               5d18h
cni-plugins-ds-swv84                                              1/1     Running                 0               5d18h
kube-multus-ds-2h7sm                                              1/1     Running                 0               4d23h
kube-multus-ds-hwfdg                                              1/1     Running                 0               4d23h
kube-multus-ds-jvmtn                                              1/1     Running                 0               5d18h
kube-multus-ds-mfqph                                              1/1     Running                 0               5d18h
mofed-ubuntu20.04-ds-27815                                        1/1     Running                 0               4d23h
mofed-ubuntu20.04-ds-4zth4                                        1/1     Running                 0               5d18h
mofed-ubuntu20.04-ds-5tmdr                                        1/1     Running                 0               5d18h
mofed-ubuntu20.04-ds-qrlvf                                        1/1     Running                 0               4d23h
network-operator-57cf95446-722tl                                  1/1     Running                 1 (4d ago)      5d18h
network-operator-node-feature-discovery-master-848d8b8cdf-667wh   1/1     Running                 1 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-h5x74              1/1     Running                 2 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-j5stf              1/1     Running                 3 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-k866p              1/1     Running                 2 (3d23h ago)   4d23h
network-operator-node-feature-discovery-worker-mlqh5              1/1     Running                 3 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-mrc5t              1/1     Running                 1 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-qrpgm              1/1     Running                 2 (3d23h ago)   5d18h
network-operator-node-feature-discovery-worker-tzfbq              1/1     Running                 2 (3d23h ago)   4d23h
rdma-shared-dp-ds-7g6s5                                           1/1     Running                 0               5d18h
rdma-shared-dp-ds-b6pgc                                           0/1     Init:ImagePullBackOff   0               4d23h
rdma-shared-dp-ds-j2m84                                           1/1     Running                 0               5d18h
rdma-shared-dp-ds-p2p84                                           0/1     Init:ImagePullBackOff   0               4d23h
sriov-device-plugin-22cv9                                         1/1     Running                 0               5d18h
sriov-device-plugin-6ktpf                                         1/1     Running                 0               4d23h
sriov-device-plugin-n586h                                         1/1     Running                 0               4d23h
sriov-device-plugin-r2w85                                         1/1     Running                 0               5d18h
whereabouts-cl951                                                 1/1     Running                 0               4d23h
whereabouts-tkw8t                                                 1/1     Running                 0               5d18h
whereabouts-xptnz                                                 1/1     Running                 0               5d18h
whereabouts-xzttx                                                 1/1     Running                 0               4d23h
root@photon-machine [ ~ ]#
```

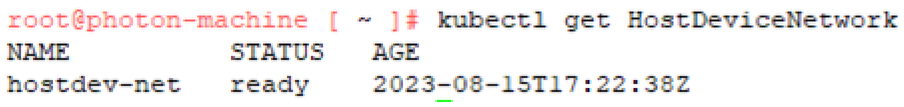
9. After deployment, the network operator should be configured, and Kubernetes networking is deployed to use it in pod configuration. *host-device-net.yaml* is the configuration file for such a deployment, this custom resource creates a **hostdev** custom resource that can be requested while creating a pod, keep in mind that the Whereabouts ipam configuration can be customized to meet your needs, see the *host-device-net.yaml* file that you need to create.

10. Apply the YAML file:

```
kubectl apply -f host-device-net.yaml
```

Once applied confirm that the custom resource was created successfully:

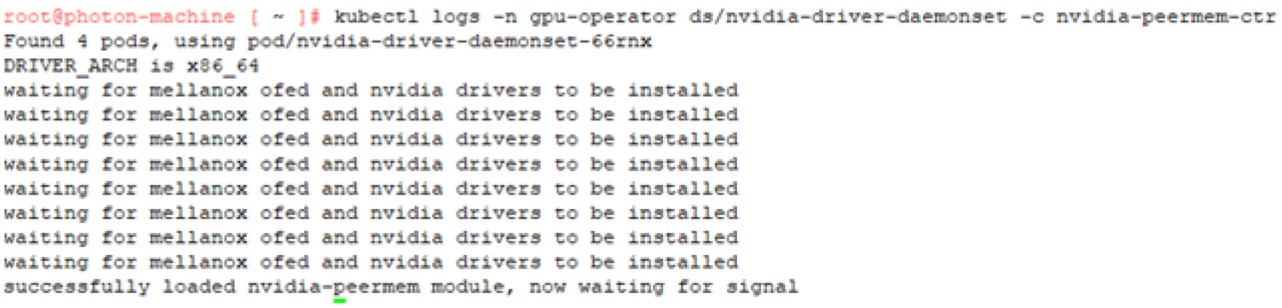
```
kubectl get HostDeviceNetwork
```



```
root@photon-machine [ ~ ]# kubectl get HostDeviceNetwork
NAME          STATUS   AGE
hostdev-net   ready    2023-08-15T17:22:38Z
```

11. We can now confirm that the nvidia-peermem-ctr container has successfully been loaded the nvidia-peermem kernel module:

```
kubectl logs -n gpu-operator ds/nvidia-driver-daemonset -c nvidia-peermem-ctr
```



```
root@photon-machine [ ~ ]# kubectl logs -n gpu-operator ds/nvidia-driver-daemonset -c nvidia-peermem-ctr
Found 4 pods, using pod/nvidia-driver-daemonset-66rnx
DRIVER_ARCH is x86_64
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
waiting for mellanox ofed and nvidia drivers to be installed
successfully loaded nvidia-peermem module, now waiting for signal
```

Virtual Function Verification and Testing Process

At this point everything that is required to test this setup is ready, we will start by creating two pods and use this YAML file as an example (change the name of the pod for each one such as hostdev-test-pod1 and hostdev-test-pod2):

Key points in this YAML:

- We are using an image from docker hub, you can access that image by creating a secret with docker hub credentials, keep in mind the free account limits.
- A hostdev device request for a VF from our SRIOV enabled Mellanox NIC is used.
- A GPU is being requested and GPU operator gets know that it should install the NVIDIA drivers on it. The vGPU, MIG, and passthrough are configured on the VM class.

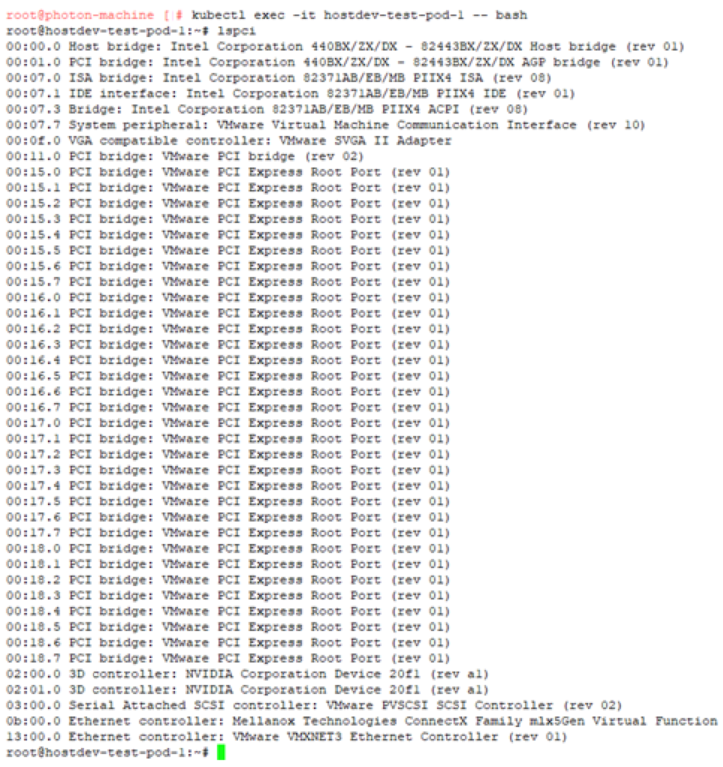
Create two pods, remember to change the name of the second pod so the result would be something like the following:

```
kubectl get pods

# NAME                  READY   STATUS     RESTARTS    AGE

# hostdev-test-pod-1    1/1     Running    0           4s

# hostdev-test-pod-2    1/1     Running    0           20s
```

After the pods are created, connect to a bash prompt with the following command:

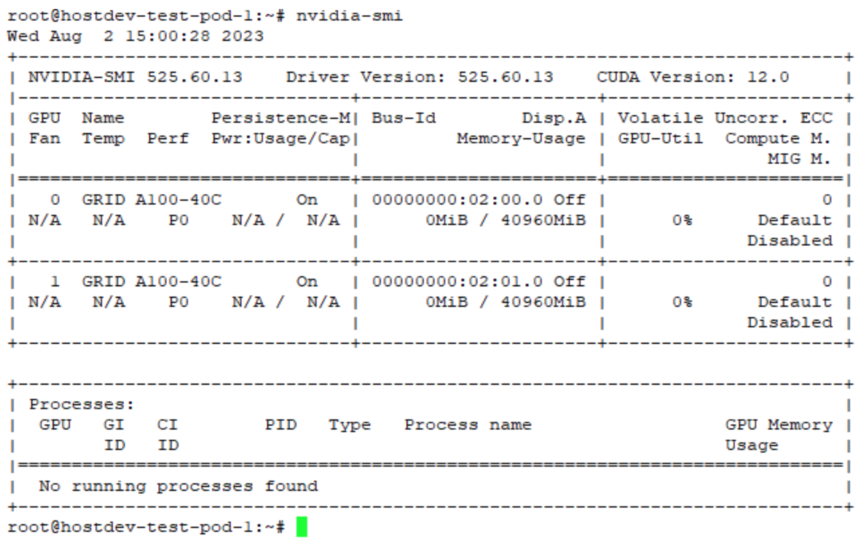
```
kubectl exec -it hostdev-test-pod-1 -- bash
```

Confirm that both pods can see the VF using lspci:



```
root@photon-machine [ ]# kubectl exec -it hostdev-test-pod-1 -- bash
root@hostdev-test-pod-1:~# lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
00:11.0 PCI bridge: VMware PCI bridge (rev 02)
00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.7 PCI bridge: VMware PCI Express Root Port (rev 01)
02:00.0 3D controller: NVIDIA Corporation Device 20f1 (rev a1)
02:01.0 3D controller: NVIDIA Corporation Device 20f1 (rev a1)
03:00.0 Serial Attached SCSI controller: VMware PVSCSI SCSI Controller (rev 02)
0b:00.0 Ethernet controller: Mellanox Technologies ConnectX Family mlx5Gen Virtual Function
13:00.0 Ethernet controller: VMware VMXNET3 Ethernet Controller (rev 01)
root@hostdev-test-pod-1:~#
```

Verify that the vGPU, MIG, or Passthrough device is detected by the *nvidia-smi* command:



```
root@hostdev-test-pod-1:~# nvidia-smi
Wed Aug  2 15:00:28 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.60.13    Driver Version: 525.60.13    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GRID A100-40C       On   | 00000000:02:00.0 Off |                    0 |
| N/A   N/A    P0    N/A /  N/A |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  GRID A100-40C       On   | 00000000:02:01.0 Off |                    0 |
| N/A   N/A    P0    N/A /  N/A |      0MiB / 40960MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
root@hostdev-test-pod-1:~#
```

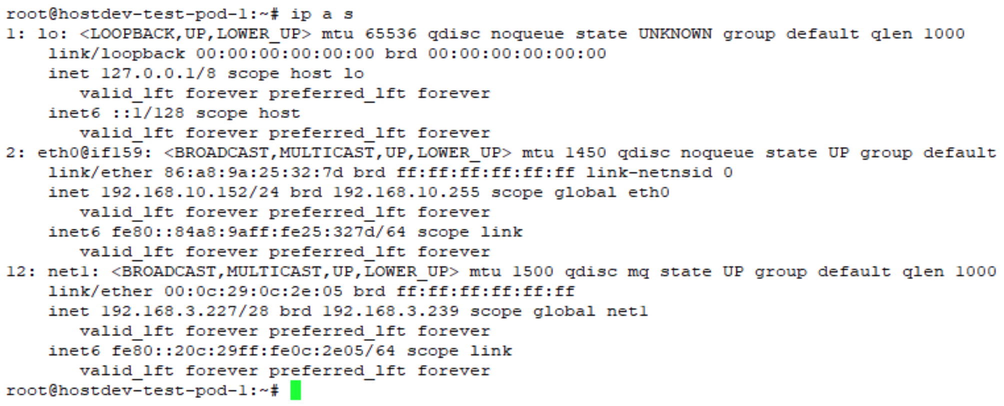
After verifying that our pods have access to the high-speed network interface and the NVIDIA GPU we can now perform and *ib_write_bw* (InfiniBand write bandwidth) test, this tool is part of *perftest* package. Perftest is a collection of tests written over uverbs intended for use as a performance micro-benchmark. The tests may be used for tuning as well as for functional testing.

Open a bash shell on both containers:

```
# Example

kubectl exec -it hostdev-test-pod-1 -- bash

kubectl exec -it hostdev-test-pod-2 -- bash
```

Get the IP address assigned to the secondary network interface backed by the Mellanox VF, the IP address for this network interface is defined based on the whereabouts config specified for the HostDeviceNetwork resource created before:

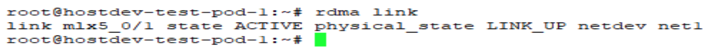
```
# Get the host IP address

ip a s
```



```
root@hostdev-test-pod-1:~# ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if159: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether 86:a8:9a:25:32:7d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.10.152/24 brd 192.168.10.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::84a8:9aff:fe25:327d/64 scope link
       valid_lft forever preferred_lft forever
12: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0c:29:0c:2e:05 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.227/28 brd 192.168.3.239 scope global net1
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe0c:2e05/64 scope link
       valid_lft forever preferred_lft forever
root@hostdev-test-pod-1:~#
```

net1 is the secondary interface, this specific pod has IP 192.168.3.227. You can confirm that your interface is up and running with the following command:

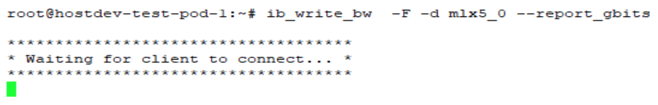
```
rdma link
```



```
root@hostdev-test-pod-1:~# rdma link
link mlx5_0/1 state ACTIVE physical_state LINK_UP netdev net1
root@hostdev-test-pod-1:~#
```

Now we will proceed to run an *ib_write_bw* test, enter the following command on pod1, this first pod will serve as the server:

```
ib_write_bw  -F -d mlx5_0 --report_gbits
```

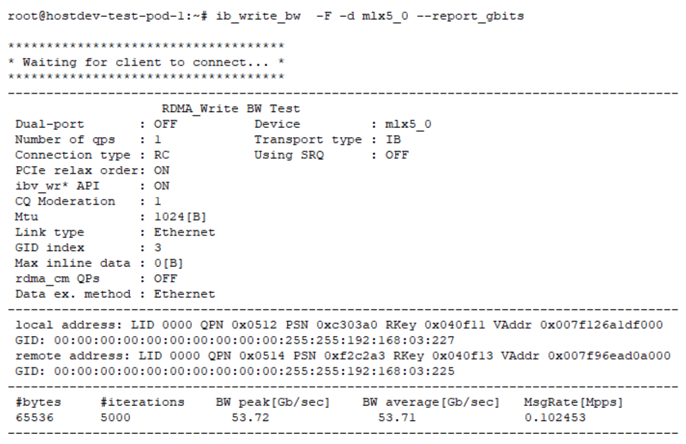


```
root@hostdev-test-pod-1:~# ib_write_bw  -F -d mlx5_0 --report_gbits

************************************
* Waiting for client to connect... *
************************************
```

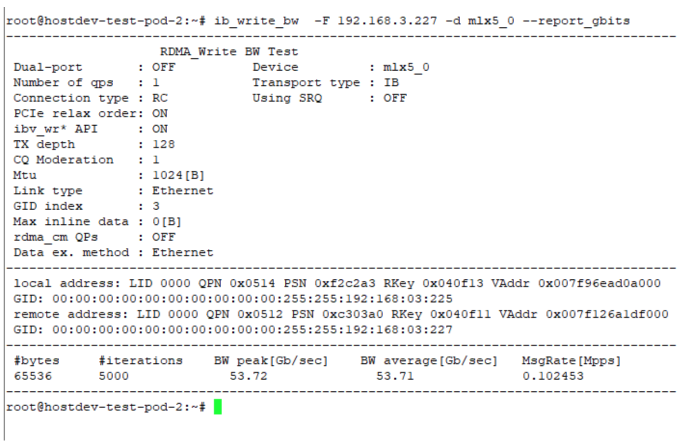
On the second pod enter the following command, this second pod will serve as the client:

```
ib_write_bw  -F 192.168.3.227 -d mlx5_0 --report_gbits
```

The result should look very similar to the following image, the output is reporting the maximum supported bandwidth:



```
root@hostdev-test-pod-1:~# ib_write_bw  -F -d mlx5_0 --report_gbits

************************************
* Waiting for client to connect... *
************************************
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx5_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API     : ON
 CQ Moderation   : 1
 Mtu             : 1024[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x0512 PSN 0xc303a0 RKey 0x040f11 VAddr 0x007f126a1df000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:227
 remote address: LID 0000 QPN 0x0514 PSN 0xf2c2a3 RKey 0x040f13 VAddr 0x007f96ead0a000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:225
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 65536      5000             53.72              53.71              0.102453
---------------------------------------------------------------------------------------
```



```
root@hostdev-test-pod-2:~# ib_write_bw  -F 192.168.3.227 -d mlx5_0 --report_gbits
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx5_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API     : ON
 TX depth        : 128
 CQ Moderation   : 1
 Mtu             : 1024[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x0514 PSN 0xf2c2a3 RKey 0x040f13 VAddr 0x007f96ead0a000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:225
 remote address: LID 0000 QPN 0x0512 PSN 0xc303a0 RKey 0x040f11 VAddr 0x007f126a1df000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:227
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 65536      5000             53.72              53.71              0.102453
---------------------------------------------------------------------------------------
root@hostdev-test-pod-2:~# 
```

GPUDirect Function Verification and Test Process

We have verified that all the required components from the GPU operator as well as our SRIOV VF are working properly, we leverage the two pods that were previously created to perform a test using Perftest's ib_write_bw, this time we enable CUDA as part of the test.

Get a bash shell to both containers:

```
kubectl exec -it hostdev-test-pod-1 -- bash
```

```
kubectl exec -it hostdev-test-pod-2 -- bash
```

Run the following command on the first container:

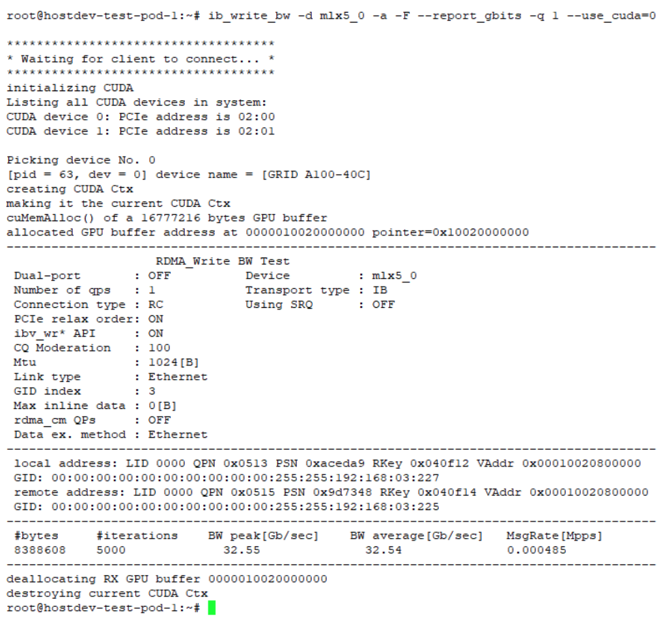
```
ib_write_bw -d mlx5_0 -a -F --report_gbits -q 1 --use_cuda=0
```

The flag "*–use_cuda=0*" will initialize a CUDA CTX (context) on the GPU 0 (you can confirm the available GPUs and their IDs with *nvidia-smi*).

On container two, run the following command:

```
ib_write_bw -d mlx5_0 -a -F --report_gbits -q 1 192.168.3.227 --use_cuda=0
```

The result looks like the following on the server side (pod one):



```
root@hostdev-test-pod-1:~# ib_write_bw -d mlx5_0 -a -F --report_gbits -q 1 --use_cuda=0

************************************
* Waiting for client to connect... *
************************************
initializing CUDA
Listing all CUDA devices in system:
CUDA device 0: PCIe address is 02:00
CUDA device 1: PCIe address is 02:01

Picking device No. 0
[pid = 63, dev = 0] device name = [GRID A100-40C]
creating CUDA Ctx
making it the current CUDA Ctx
cuMemAlloc() of a 16777216 bytes GPU buffer
allocated GPU buffer address at 0000010020000000 pointer=0x10020000000
---------------------------------------------------------------------------------------
                    RDMA_Write BW Test
 Dual-port       : OFF          Device         : mlx5_0
 Number of qps   : 1            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON
 ibv_wr* API     : ON
 CQ Moderation   : 100
 Mtu             : 1024[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x0513 PSN 0xaceda9 RKey 0x040f12 VAddr 0x00010020800000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:227
 remote address: LID 0000 QPN 0x0515 PSN 0x9d7348 RKey 0x040f14 VAddr 0x00010020800000
 GID: 00:00:00:00:00:00:00:00:00:00:255:255:192:168:03:225
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[Gb/sec]    BW average[Gb/sec]   MsgRate[Mpps]
 8388608    5000             32.55              32.54              0.000485
---------------------------------------------------------------------------------------
deallocating RX GPU buffer 0000010020000000
destroying current CUDA Ctx
root@hostdev-test-pod-1:~# 
```

NVIDIA vGPU Driver Step-by-Step Installation Process

We demonstrate the NVIDIA vGPU driver installation process as follows:

1. After SSH into the VM, update the system and install the dependencies.

```
sudo apt update && sudo apt upgrade -y
sudo apt install build-essential -y
sudo reboot
```

2. Get the driver installation file, change the permission and install the driver.

```
wget http://x.x.x.x/NVIDIA-Linux-x86_64-525.105.17-grid.run

chmod +x NVIDIA-Linux-x86_64-525.105.17-grid.run

sudo ./NVIDIA-Linux-x86_64-525.105.17-grid.run
```

3. Verify if vGPU driver is successfully installed.

```
nvidia-smi
```

4. Get the license token, place into /etc/nvidia/ClientConfigToken, change permission and restart NVIDIA Grid Service

```
cd /etc/nvidia/ClientConfigToken/

sudo wget http://x.x.x.x/client_configuration_token.tok

sudo chmod 744 /etc/nvidia/ClientConfigToken/client_configuration_token.tok

sudo systemctl restart nvidia-gridd
```

5. Verify if the vGPU driver is successfully licensed.

```
nvidia-smi -q | grep -i lic
```

---

[1] This setting is automatically configured by Tanzu VM service when a VM class with GPUs is used.

[2] 128 or multiple of the GPU memory size