



Artificial Intelligence (AI) HCI

NetApp
November 04, 2020

This PDF was generated from https://docs.netapp.com/us-en/hci-solutions/hciaiedge_use_cases.html on November 04, 2020. Always check docs.netapp.com for the latest.



Table of Contents

Artificial Intelligence (AI)	1
NVA-1144: NetApp HCI AI Inferencing at the Edge Data Center with H615c and NVIDIA T4	1

Artificial Intelligence (AI)

NVA-1144: NetApp HCI AI Inferencing at the Edge Data Center with H615c and NVIDIA T4

Arvind Ramakrishnan, NetApp

This document describes how NetApp HCI can be designed to host artificial intelligence (AI) inferencing workloads at edge data center locations. The design is based on NVIDIA T4 GPU-powered NetApp HCI compute nodes, an NVIDIA Triton Inference Server, and a Kubernetes infrastructure built using NVIDIA DeepOps. The design also establishes the data pipeline between the core and edge data centers and illustrates implementation to complete the data lifecycle path.

Modern applications that are driven by AI and machine learning (ML) have pushed the limits of the internet. End users and devices demand access to applications, data, and services at any place and any time, with minimal latency. To meet these demands, data centers are moving closer to their users to boost performance, reduce back-and-forth data transfer, and provide cost-effective ways to meet user requirements.

In the context of AI, the core data center is a platform that provides centralized services, such as machine learning and analytics, and the edge data centers are where the real-time production data is subject to inferencing. These edge data centers are usually connected to a core data center. They provide end-user services and serve as a staging layer for data generated by IoT devices that need additional processing and that is too time sensitive to be transmitted back to a centralized core.

This document describes a reference architecture for AI inferencing that uses NetApp HCI as the base platform.

Customer Value

NetApp HCI offers differentiation in the hyperconverged market for this inferencing solution, including the following advantages:

- A disaggregated architecture allows independent scaling of compute and storage and lowers the virtualization licensing costs and performance tax on independent NetApp HCI storage nodes.
- NetApp Element storage provides quality of service (QoS) for each storage volume, which provides guaranteed storage performance for workloads on NetApp HCI. Therefore, adjacent workloads do not negatively affect inferencing performance.
- A data fabric powered by NetApp allows data to be replicated from core to edge to cloud data centers, which moves data closer to where application needs it.
- With a data fabric powered by NetApp and NetApp FlexCache software, AI deep learning models trained on NetApp ONTAP AI can be accessed from NetApp HCI without having to export the model.
- NetApp HCI can host inference servers on the same infrastructure concurrently with multiple

workloads, either virtual-machine (VM) or container-based, without performance degradation.

- NetApp HCI is certified as NVIDIA GPU Cloud (NGC) ready for NVIDIA AI containerized applications.
- NGC-ready means that the stack is validated by NVIDIA, is purpose built for AI, and enterprise support is available through NGC Support Services.
- With its extensive AI portfolio, NetApp can support the entire spectrum of AI use cases from edge to core to cloud, including ONTAP AI for training and inferencing, Cloud Volumes Service and Azure NetApp Files for training in the cloud, and inferencing on the edge with NetApp HCI.

[Next: Use Cases](#)

Use Cases

Although all applications today are not AI driven, they are evolving capabilities that allow them to access the immense benefits of AI. To support the adoption of AI, applications need an infrastructure that provides them with the resources needed to function at an optimum level and support their continuing evolution.

For AI-driven applications, edge locations act as a major source of data. Available data can be used for training when collected from multiple edge locations over a period of time to form a training dataset. The trained model can then be deployed back to the edge locations where the data was collected, enabling faster inferencing without the need to repeatedly transfer production data to a dedicated inferencing platform.

The NetApp HCI AI inferencing solution, powered by NetApp H615c compute nodes with NVIDIA T4 GPUs and NetApp cloud-connected storage systems, was developed and verified by NetApp and NVIDIA. NetApp HCI simplifies the deployment of AI inferencing solutions at edge data centers by addressing areas of ambiguity, eliminating complexities in the design and ending guesswork.

This solution gives IT organizations a prescriptive architecture that:

- Enables AI inferencing at edge data centers
- Optimizes consumption of GPU resources
- Provides a Kubernetes-based inferencing platform for flexibility and scalability
- Eliminates design complexities

Edge data centers manage and process data at locations that are very near to the generation point. This proximity increases the efficiency and reduces the latency involved in handling data. Many vertical markets have realized the benefits of an edge data center and are heavily adopting this distributed approach to data processing.

The following table lists the edge verticals and applications.

Vertical	Applications
Medical	Computer-aided diagnostics assist medical staff in early disease detection
Oil and gas	Autonomous inspection of remote production facilities, video, and image analytics
Aviation	Air traffic control assistance and real-time video feed analytics
Media and entertainment	Audio/video content filtering to deliver family-friendly content
Business analytics	Brand recognition to analyze brand appearance in live-streamed televised events
E-Commerce	Smart bundling of supplier offers to find ideal merchant and warehouse combinations
Retail	Automated checkout to recognize items a customer placed in cart and facilitate digital payment
Smart city	Improve traffic flow, optimize parking, and enhance pedestrian and cyclist safety
Manufacturing	Quality control, assembly-line monitoring, and defect identification
Customer service	Customer service automation to analyze and triage inquiries (phone, email, and social media)
Agriculture	Intelligent farm operation and activity planning, to optimize fertilizer and herbicide application

Target Audience

The target audience for the solution includes the following groups:

- Data scientists
- IT architects
- Field consultants
- Professional services
- IT managers
- Anyone else who needs an infrastructure that delivers IT innovation and robust data and application services at edge locations

[Next: Architecture](#)

Architecture

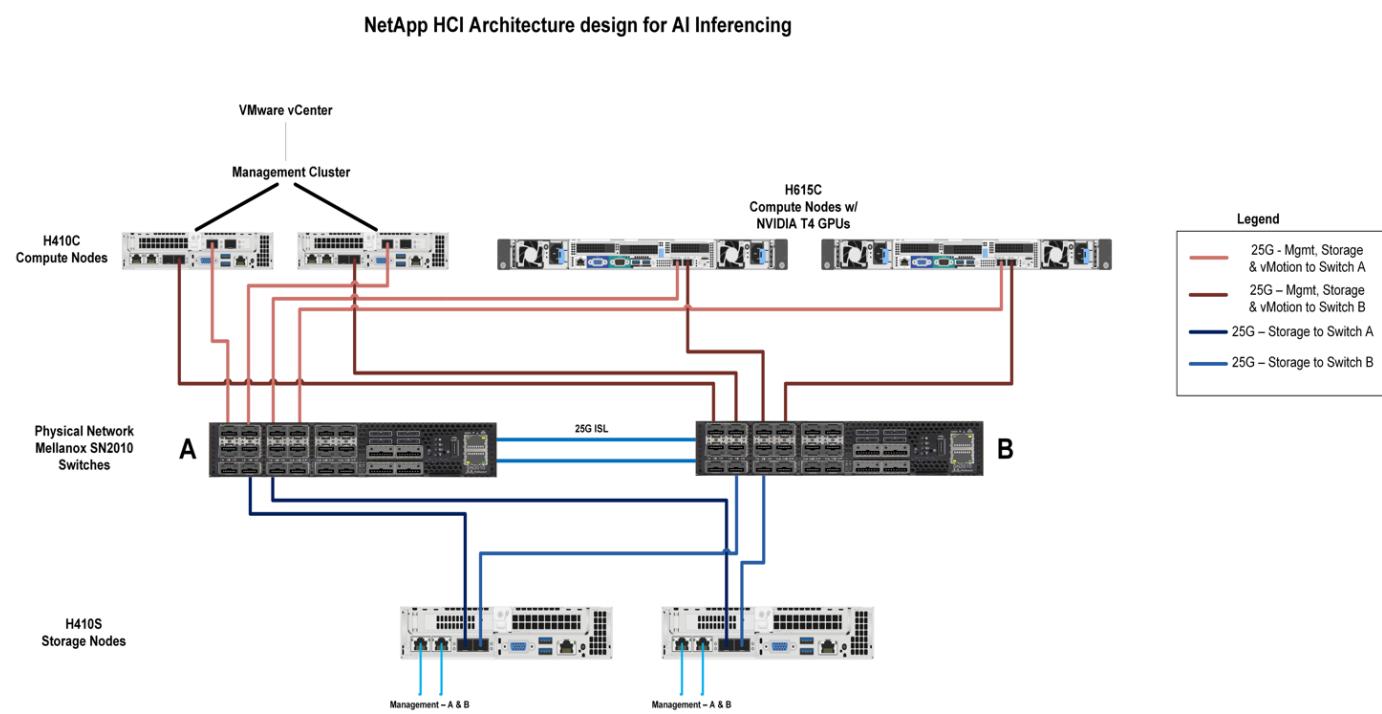
Solution Technology

This solution is designed with a NetApp HCI system that contains the following components:

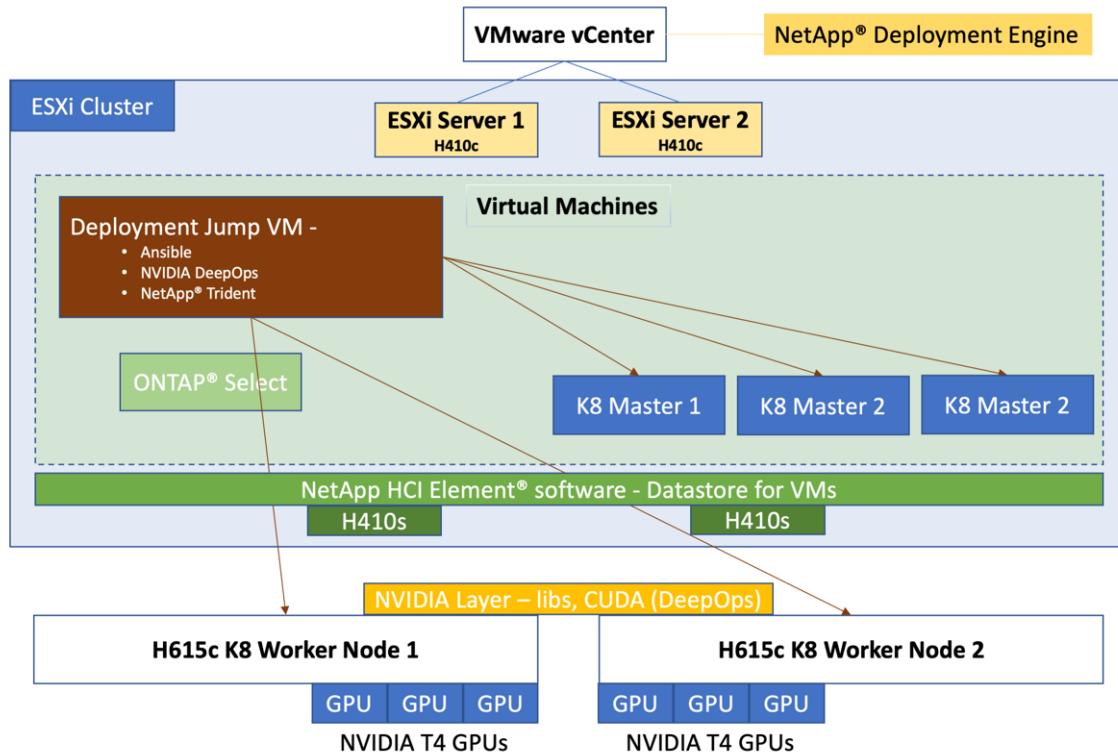
- Two H615c compute nodes with NVIDIA T4 GPUs
- Two H410c compute nodes
- Two H410s storage nodes
- Two Mellanox SN2010 10GbE/25GbE switches

Architectural Diagram

The following diagram illustrates the solution architecture for the NetApp HCI AI inferencing solution.



The following diagram illustrates the virtual and physical elements of this solution.



A VMware infrastructure is used to host the management services required by this inferencing solution. These services do not need to be deployed on a dedicated infrastructure; they can coexist with any existing workloads. The NetApp Deployment Engine (NDE) uses the H410c and H410s nodes to deploy the VMware infrastructure.

After NDE has completed the configuration, the following components are deployed as VMs in the virtual infrastructure:

- **Deployment Jump VM.** Used to automate the deployment of NVIDIA DeepOps. See [NVIDIA DeepOps](#) and storage management using NetApp Trident.
- **ONTAP Select.** An instance of ONTAP Select is deployed to provide NFS file services and persistent storage to the AI workload running on Kubernetes.
- **Kubernetes Masters.** During deployment, three VMs are installed and configured with a supported Linux distribution and configured as Kubernetes master nodes. After the management services have been set up, two H615c compute nodes with NVIDIA T4 GPUs are installed with a supported Linux distribution. These two nodes function as the Kubernetes worker nodes and provide the infrastructure for the inferencing platform.

Hardware Requirements

The following table lists the hardware components that are required to implement the solution. The hardware components that are used in any particular implementation of the solution might vary based on customer requirements.

Layer	Product Family	Quantity	Details
Compute	H615c	2	3 NVIDIA Tesla T4 GPUs per node
	H410c	2	Compute nodes for management infrastructure
Storage	H410s	2	Storage for OS and workload
Network	Mellanox SN2010	2	10G/25G switches

Software Requirements

The following table lists the software components that are required to implement the solution. The software components that are used in any particular implementation of the solution might vary based on customer requirements.

Layer	Software	Version
Storage	NetApp Element software	12.0.0.333
	ONTAP Select	9.7
	NetApp Trident	20.07
NetApp HCI engine	NDE	1.8
Hypervisor	Hypervisor	VMware vSphere ESXi 6.7U1
	Hypervisor Management System	VMware vCenter Server 6.7U1
Inferencing Platform	NVIDIA DeepOps	20.08
	NVIDIA GPU Operator	1.1.7
	Ansible	2.9.5
	Kubernetes	1.17.9
	Docker	Docker CE 18.09.7
	CUDA Version	10.2
	GPU Device Plugin	0.6.0
	Helm	3.1.2
	NVIDIA Tesla Driver	440.64.00
	NVIDIA Triton Inference Server	2.1.0 – NGC Container v20.07

Layer	Software	Version
K8 Master VMs	Linux	<p>Any supported distribution across NetApp IMT, NVIDIA DeepOps, and GPUOperator</p> <p>Ubuntu 18.04.4 LTS was used in this solution</p> <p>Kernel version: 4.15</p>
Host OS/ K8 Worker Nodes	Linux	<p>Any supported distribution across NetApp IMT, NVIDIA DeepOps, and GPUOperator</p> <p>Ubuntu 18.04.4 LTS was used in this solution</p> <p>Kernel version: 4.15</p>

[Next: Design Considerations](#)

Design Considerations

Network Design

The switches used to handle the NetApp HCI traffic require a specific configuration for successful deployment.

Consult the NetApp HCI Network Setup Guide for the physical cabling and switch details. This solution uses a two-cable design for compute nodes. Optionally, compute nodes can be configured in a six-node cable design affording options for deployment of compute nodes.

The diagram under [Architecture](#) depicts the network topology of this NetApp HCI solution with a two-cable design for the compute nodes.

Compute Design

The NetApp HCI compute nodes are available in two form factors, half-width and full-width, and in two rack unit sizes, 1 RU and 2 RU. The 410c nodes used in this solution are half-width and 1 RU and are housed in a chassis that can hold a maximum of four such nodes. The other compute node that is used in this solution is the H615c, which is a full-width node, 1 RU in size. The H410c nodes are based on Intel Skylake processors, and the H615c nodes are based on the second-generation Intel Cascade Lake processors. NVIDIA GPUs can be added to the H615c nodes, and each node can host a maximum of three NVIDIA Tesla T4 16GB GPUs.

The H615c nodes are the latest series of compute nodes for NetApp HCI and the second series that can support GPUs. The first model to support GPUs is the H610c node (full width, 2RU), which can support two NVIDIA Tesla M10 GPUs.

In this solution, H615c nodes are preferred over H610c nodes because of the following advantages:

- Reduced data center footprint, critical for edge deployments
- Support for a newer generation of GPUs designed for faster inferencing
- Reduced power consumption
- Reduced heat dissipation

NVIDIA T4 GPUs

The resource requirements of inferencing are nowhere close to those of training workloads. In fact, most modern hand-held devices are capable of handling small amounts of inferencing without powerful resources like GPUs. However, for mission-critical applications and data centers that are dealing with a wide variety of applications that demand very low inferencing latencies while subject to extreme parallelization and massive input batch sizes, the GPUs play a key role in reducing inference time and help to boost application performance.

The NVIDIA Tesla T4 is an x16 PCIe Gen3 single-slot low-profile GPU based on the Turing architecture. The T4 GPUs deliver universal inference acceleration that spans applications such as image classification and tagging, video analytics, natural language processing, automatic speech recognition, and intelligent search. The breadth of the Tesla T4's inferencing capabilities enables it to be used in enterprise solutions and edge devices.

These GPUs are ideal for deployment in edge infrastructures due to their low power consumption and small PCIe form factor. The size of the T4 GPUs enables the installation of two T4 GPUs in the same space as a double-slot full-sized GPU. Although they are small, with 16GB memory, the T4s can support large ML models or run inference on multiple smaller models simultaneously.

The Turing- based T4 GPUs include an enhanced version of Tensor Cores and support a full range of precisions for inferencing FP32, FP16, INT8, and INT4. The GPU includes 2,560 CUDA cores and 320 Tensor Cores, delivering up to 130 tera operations per second (TOPS) of INT8 and up to 260 TOPS of INT4 inferencing performance. When compared to CPU-based inferencing, the Tesla T4, powered by the new Turing Tensor Cores, delivers up to 40 times higher inference performance.

The Turing Tensor Cores accelerate the matrix-matrix multiplication at the heart of neural network training and inferencing functions. They particularly excel at inference computations in which useful and relevant information can be inferred and delivered by a trained deep neural network based on a given input.

The Turing GPU architecture inherits the enhanced Multi-Process Service (MPS) feature that was introduced in the Volta architecture. Compared to Pascal-based Tesla GPUs, MPS on Tesla T4 improves inference performance for small batch sizes, reduces launch latency, improves QoS, and enables the servicing of higher numbers of concurrent client requests.

The NVIDIA T4 GPU is a part of the NVIDIA AI Inference Platform that supports all AI frameworks and provides comprehensive tooling and integrations to drastically simplify the development and

deployment of advanced AI.

Storage Design: Element Software

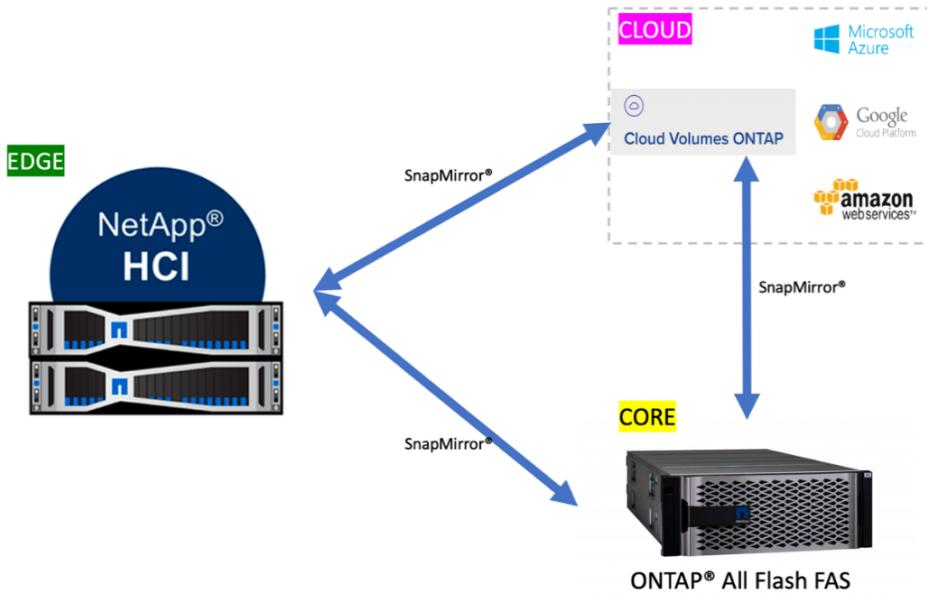
NetApp Element software powers the storage of the NetApp HCI systems. It delivers agile automation through scale-out flexibility and guaranteed application performance to accelerate new services.

Storage nodes can be added to the system non-disruptively in increments of one, and the storage resources are made available to the applications instantly. Every new node added to the system delivers a precise amount of additional performance and capacity to a usable pool. The data is automatically load balanced in the background across all nodes in the cluster, maintaining even utilization as the system grows.

Element software supports the NetApp HCI system to comfortably host multiple workloads by guaranteeing QoS to each workload. By providing fine-grained performance control with minimum, maximum, and burst settings for each workload, the software allows well-planned consolidations while protecting application performance. It decouples performance from capacity and allows each volume to be allocated with a specific amount of capacity and performance. These specifications can be modified dynamically without any interruption to data access.

As illustrated in the following figure, Element software integrates with NetApp ONTAP to enable data mobility between NetApp storage systems that are running different storage operating systems. Data can be moved from the Element software to ONTAP or vice versa by using NetApp SnapMirror technology. Element uses the same technology to provide cloud connectivity by integrating with NetApp Cloud Volumes ONTAP, which enables data mobility from the edge to the core and to multiple public cloud service providers.

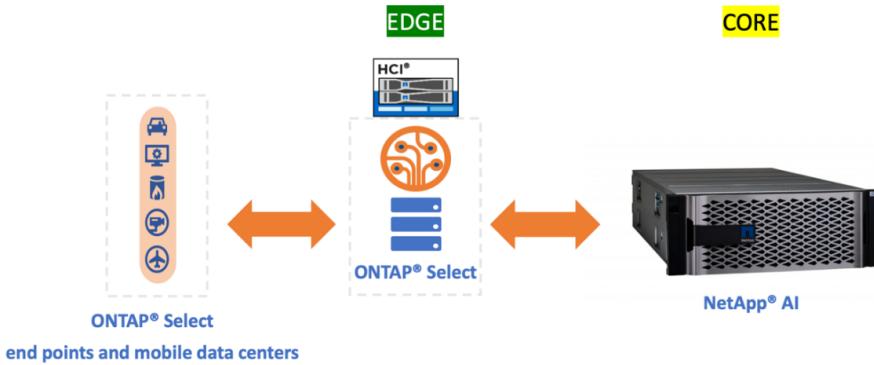
In this solution, the Element-backed storage provides the storage services that are required to run the workloads and applications on the NetApp HCI system.



Storage Design: ONTAP Select

NetApp ONTAP Select introduces a software-defined data storage service model on top of NetApp HCI. It builds on NetApp HCI capabilities, adding a rich set of file and data services to the HCI platform while extending the data fabric.

Although ONTAP Select is an optional component for implementing this solution, it does provide a host of benefits, including data gathering, protection, mobility, and so on, that are extremely useful in the context of the overall AI data lifecycle. It helps to simplify several day-to-day challenges for data handling, including ingestion, collection, training, deployment, and tiering.



12

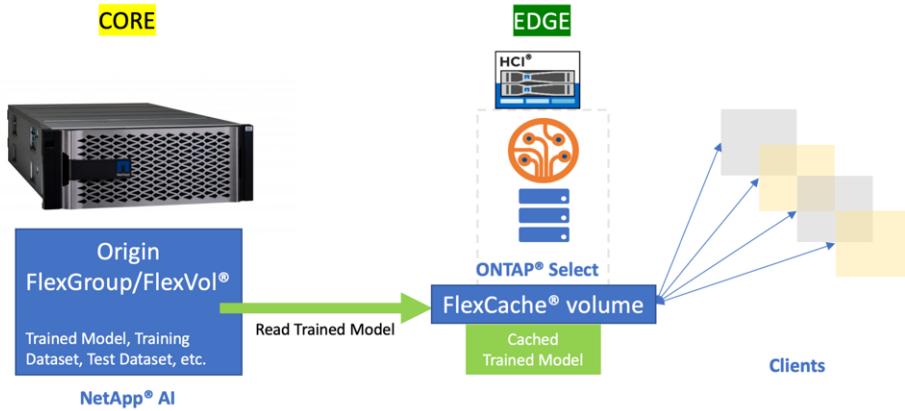
© 2019 NetApp, Inc. All rights reserved. — NETAPP CONFIDENTIAL —

NetApp

ONTAP Select can run as a VM on VMware and still bring in most of the ONTAP capabilities that are available when it is running on a dedicated FAS platform, such as the following:

- Support for NFS and CIFS
- NetApp FlexClone technology
- NetApp FlexCache technology
- NetApp ONTAP FlexGroup volumes
- NetApp SnapMirror software

ONTAP Select can be used to leverage the FlexCache feature, which helps to reduce data-read latencies by caching frequently read data from a back-end origin volume, as is shown in the following figure. In the case of high-end inferencing applications with a lot of parallelization, multiple instances of the same model are deployed across the inferencing platform, leading to multiple reads of the same model. Newer versions of the trained model can be seamlessly introduced to the inferencing platform by verifying that the desired model is available in the origin or source volume.



NetApp Trident

NetApp Trident is an open-source dynamic storage orchestrator that allows you to manage storage resources across all major NetApp storage platforms. It integrates with Kubernetes natively so that persistent volumes (PVs) can be provisioned on demand with native Kubernetes interfaces and constructs. Trident enables microservices and containerized applications to use enterprise-class storage services such as QoS, storage efficiencies, and cloning to meet the persistent storage demands of applications.

Containers are among the most popular methods of packaging and deploying applications, and Kubernetes is one of the most popular platforms for hosting containerized applications. In this solution, the inferencing platform is built on top of a Kubernetes infrastructure.

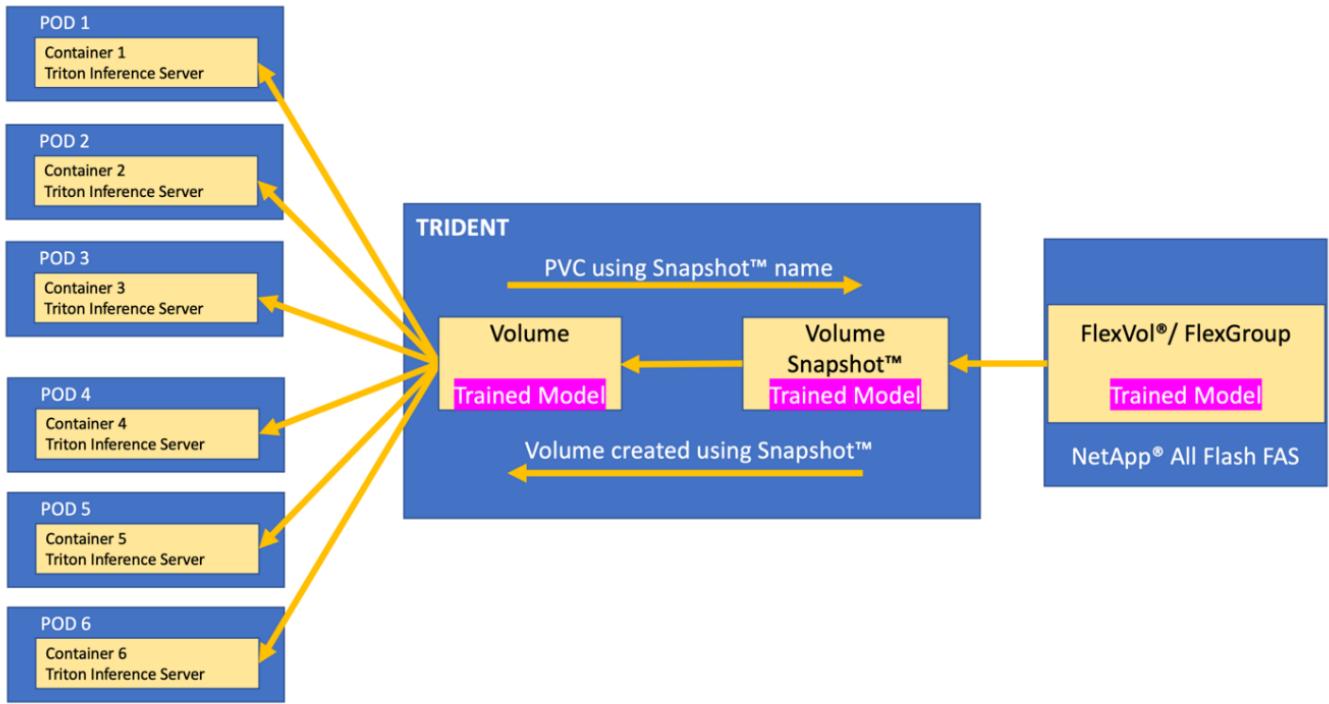
Trident currently supports storage orchestration across the following platforms:

- ONTAP: NetApp AFF, FAS, and Select
- Element software: NetApp HCI and NetApp SolidFire all-flash storage
- NetApp SANtricity software: E-Series and EF-series
- Cloud Volumes ONTAP
- Azure NetApp Files
- NetApp Cloud Volumes Service: AWS and Google Cloud

Trident is a simple but powerful tool to enable storage orchestration not just across multiple storage platforms, but also across the entire spectrum of the AI data lifecycle, ranging from the edge to the core

to the cloud.

Trident can be used to provision a PV from a NetApp Snapshot copy that makes up the trained model. The following figure illustrates the Trident workflow in which a persistent volume claim (PVC) is created by referring to an existing Snapshot copy. Following this, Trident creates a volume by using the Snapshot copy.



This method of introducing trained models from a Snapshot copy supports robust model versioning. It simplifies the process of introducing newer versions of models to applications and switching inferencing between different versions of the model.

NVIDIA DeepOps

NVIDIA DeepOps is a modular collection of Ansible scripts that can be used to automate the deployment of a Kubernetes infrastructure. There are multiple deployment tools available that can automate the deployment of a Kubernetes cluster. In this solution, DeepOps is the preferred choice because it does not just deploy a Kubernetes infrastructure, it also installs the necessary GPU drivers, NVIDIA Container Runtime for Docker (nvidia-docker2), and various other dependencies for GPU-accelerated work. It encapsulates the best practices for NVIDIA GPUs and can be customized or run as individual components as needed.

DeepOps internally uses Kubespray to deploy Kubernetes, and it is included as a submodule in DeepOps. Therefore, common Kubernetes cluster management operations such as adding nodes, removing nodes, and cluster upgrades should be performed using Kubespray.

A software based L2 LoadBalancer using MetallB and an Ingress Controller based on NGINX are also

deployed as part of this solution by using the scripts that are available with DeepOps.

In this solution, three Kubernetes master nodes are deployed as VMs, and the two H615c compute nodes with NVIDIA Tesla T4 GPUs are set up as Kubernetes worker nodes.

NVIDIA GPU Operator

The GPU operator deploys the NVIDIA k8s-device-plugin for GPU support and runs the NVIDIA drivers as containers. It is based on the Kubernetes operator framework, which helps to automate the management of all NVIDIA software components that are needed to provision GPUs. The components include NVIDIA drivers, Kubernetes device plug-in for GPUs, the NVIDIA container runtime, and automatic node labeling, which is used in tandem with Kubernetes Node Feature Discovery.

The GPU operator is an important component of the [NVIDIA EGX](#) software-defined platform that is designed to make large-scale hybrid-cloud and edge operations possible and efficient. It is specifically useful when the Kubernetes cluster needs to scale quickly—for example, when provisioning additional GPU-based worker nodes and managing the lifecycle of the underlying software components. Because the GPU operator runs everything as containers, including NVIDIA drivers, administrators can easily swap various components by simply starting or stopping containers.

NVIDIA Triton Inference Server

NVIDIA Triton Inference Server (Triton Server) simplifies the deployment of AI inferencing solutions in production data centers. This microservice is specifically designed for inferencing in production data centers. It maximizes GPU utilization and integrates seamlessly into DevOps deployments with Docker and Kubernetes.

Triton Server provides a common solution for AI inferencing. Therefore, researchers can focus on creating high-quality trained models, DevOps engineers can focus on deployment, and developers can focus on applications without the need to redesign the platform for each AI-powered application.

Here are some of the key features of Triton Server:

- **Support for multiple frameworks.** Triton Server can handle a mix of models, and the number of models is limited only by system disk and memory resources. It can support the TensorRT, TensorFlow GraphDef, TensorFlow SavedModel, ONNX, PyTorch, and Caffe2 NetDef model formats.
- *Concurrent model execution. *Multiple models or multiple instances of the same model can be run simultaneously on a GPU.
- **Multi-GPU support.** Triton Server can maximize GPU utilization by enabling inference for multiple models on one or more GPUs.
- **Support for batching.** Triton Server can accept requests for a batch of inputs and respond with the corresponding batch of outputs. The inference server supports multiple scheduling and batching algorithms that combine individual inference requests together to improve inference throughput. Batching algorithms are available for both stateless and stateful applications and need to be used appropriately. These scheduling and batching decisions are transparent to the client that is

requesting inference.

- **Ensemble support.** An ensemble is a pipeline with multiple models with connections of input and output tensors between those models. An inference request can be made to an ensemble, which results in the execution of the complete pipeline.
- **Metrics.** Metrics are details about GPU utilization, server throughput, server latency, and health for auto scaling and load balancing.

NetApp HCI is a hybrid multi-cloud infrastructure that can host multiple workloads and applications, and the Triton Inference Server is well equipped to support the inferencing requirements of multiple applications.

In this solution, Triton Server is deployed on the Kubernetes cluster using a deployment file. With this method, the default configuration of Triton Server can be overridden and customized as required. Triton Server also provides an inference service using an HTTP or GRPC endpoint, allowing remote clients to request inferencing for any model that is being managed by the server.

A Persistent Volume is presented via NetApp Trident to the container that runs the Triton Inference Server and this persistent volume is configured as the model repository for the Inference server.

The Triton Inference Server is deployed with varying sets of resources using Kubernetes deployment files, and each server instance is presented with a LoadBalancer front end for seamless scalability. This approach also illustrates the flexibility and simplicity with which resources can be allocated to the inferencing workloads.

[Next: Deploying NetApp HCI – AI Inferencing at the Edge](#)

Overview

This section describes the steps required to deploy the AI inferencing platform using NetApp HCI. The following list provides the high-level tasks involved in the setup:

1. [Configure network switches](#)
2. [Deploy the VMware virtual infrastructure on NetApp HCI using NDE](#)
3. [Configure the H615c compute nodes to be used as K8 worker nodes](#)
4. [Set up the deployment jump VM and K8 master VMs](#)
5. [Deploy a Kubernetes cluster with NVIDIA DeepOps](#)
6. [Deploy ONTAP Select within the virtual infrastructure](#)
7. [Deploy NetApp Trident](#)
8. [Deploy NVIDIA Triton inference Server](#)
9. [Deploy the client for the Triton inference server](#)

10. Collect inference metrics from the Triton inference server

Next: Configure Network Switches

Configure Network Switches (Automated Deployment)

Prepare Required VLAN IDs

The following table lists the necessary VLANs for deployment, as outlined in this solution validation. You should configure these VLANs on the network switches prior to executing NDE.

Network Segment	Details	VLAN ID
Out-of-band management network	Network for HCI terminal user interface (TUI)	16
In-band management network	Network for accessing management interfaces of nodes, hosts, and guests	3488
VMware vMotion	Network for live migration of VMs	3489
iSCSI SAN storage	Network for iSCSI storage traffic	3490
Application	Network for Application traffic	3487
NFS	Network for NFS storage traffic	3491
IPL*	Interpeer link between Mellanox switches	4000
Native	Native VLAN	2

*Only for Mellanox switches

Switch Configuration

This solution uses Mellanox SN2010 switches running Onyx. The Mellanox switches are configured using an Ansible playbook. Prior to running the Ansible playbook, you should perform the initial configuration of the switches manually:

1. Install and cable the switches to the uplink switch, compute, and storage nodes.
2. Power on the switches and configure them with the following details:
 - a. Host name
 - b. Management IP and gateway
 - c. NTP
3. Log into the Mellanox switches and run the following commands:

```
configuration write to pre-ansible  
configuration write to post-ansible
```

The **pre-ansible** configuration file created can be used to restore the switch's configuration to the state before the Ansible playbook execution.

The switch configuration for this solution is stored in the **post-ansible** configuration file.



The configuration playbook for Mellanox switches that follows best practices and requirements for NetApp HCI can be downloaded [here](#).

4. Fill out the credentials to access the switches and variables needed for the environment. The following text is a sample of the variable file for this solution.

```
# vars file for nar_hci_mellanox_deploy  
#These set of variables will setup the Mellanox switches for NetApp HCI that uses a 2-cable compute connectivity option.  
#Ansible connection variables for mellanox  
ansible_connection: network_cli  
ansible_network_os: onyx  
#-----  
# Primary Variables  
#-----  
#Necessary VLANs for Standard NetApp HCI Deployment [native, Management, iSCSI_Storage, vMotion, VM_Network, IPL]  
#Any additional VLANs can be added to this in the prescribed format below  
netapp_hci_vlans:  
- {vlan_id: 2 , vlan_name: "Native" }  
- {vlan_id: 3488 , vlan_name: "IB-Management" }  
- {vlan_id: 3490 , vlan_name: "iSCSI_Storage" }  
- {vlan_id: 3489 , vlan_name: "vMotion" }  
- {vlan_id: 3491 , vlan_name: "NFS " }  
- {vlan_id: 3487 , vlan_name: "App_Network" }  
- {vlan_id: 4000 , vlan_name: "IPL" }#Modify the VLAN IDs to suit your environment  
#Spanning-tree protocol type for uplink connections.  
#The valid options are 'network' and 'normal'; selection depends on the uplink switch model.  
uplink_stp_type: network  
#-----  
# IPL variables  
#-----  
#Inter-Peer Link Portchannel  
#ipl_portchannel to be defined in the format - Po100  
ipl_portchannel: Po100  
#Inter-Peer Link Addresses
```

```

#The IPL IP address should not be part of the management network. This is typically a
private network
ipl_ipaddr_a: 10.0.0.1
ipl_ipaddr_b: 10.0.0.2
#Define the subnet mask in CIDR number format. Eg: For subnet /22, use ipl_ip_subnet:
22
ipl_ip_subnet: 24
#Inter-Peer Link Interfaces
#members to be defined with Eth in the format. Eg: Eth1/1
peer_link_interfaces:
  members: ['Eth1/20', 'Eth1/22']
  description: "peer link interfaces"
#MLAG VIP IP address should be in the same subnet as that of the switches' mgmt0
interface subnet
#mlag_vip_ip to be defined in the format - <vip_ip>/<subnet_mask>. Eg: x.x.x.x/y
mlag_vip_ip: <><mlag_vip_ip>>
#MLAG VIP Domain Name
#The mlag domain must be unique name for each mlag domain.
#In case you have more than one pair of MLAG switches on the same network, each domain
(consist of two switches) should be configured with different name.
mlag_domain_name: MLAG-VIP-DOM
-----
# Interface Details
-----
#Storage Bond10G Interface details
#members to be defined with Eth in the format. Eg: Eth1/1
#Only numerical digits between 100 to 1000 allowed for mlag_id
#Operational link speed [variable 'speed' below] to be defined in terms of bytes.
#For 10 Gigabyte operational speed, define 10G. [Possible values - 10G and 25G]
#Interface descriptions append storage node data port numbers assuming all Storage
Nodes' Port C -> Mellanox Switch A and all Storage Nodes' Port D -> Mellanox Switch B
#List the storage Bond10G interfaces, their description, speed and MLAG IDs in list of
dictionaries format
storage_interfaces:
- {members: "Eth1/1", description: "HCI_Storage_Node_01", mlag_id: 101, speed: 25G}
- {members: "Eth1/2", description: "HCI_Storage_Node_02", mlag_id: 102, speed: 25G}
#In case of additional storage nodes, add them here
#Storage Bond1G Interface
#Mention whether or not these Mellanox switches will also be used for Storage Node
Mgmt connections
#Possible inputs for storage_mgmt are 'yes' and 'no'
storage_mgmt: <><yes or no>>
#Storage Bond1G (Mgmt) interface details. Only if 'storage_mgmt' is set to 'yes'
#Members to be defined with Eth in the format. Eg: Eth1/1
#Interface descriptions append storage node management port numbers assuming all
Storage Nodes' Port A -> Mellanox Switch A and all Storage Nodes' Port B -> Mellanox
Switch B
#List the storage Bond1G interfaces and their description in list of dictionaries

```

```

format
storage_mgmt_interfaces:
- {members: "Ethx/y", description: "HCI_Storage_Node_01"}
- {members: "Ethx/y", description: "HCI_Storage_Node_02"}
#In case of additional storage nodes, add them here
#LACP load balancing algorithm for IP hash method
#Possible options are: 'destination-mac', 'destination-ip', 'destination-port',
'source-mac', 'source-ip', 'source-port', 'source-destination-mac', 'source-
destination-ip', 'source-destination-port'
#This variable takes multiple options in a single go
#For eg: if you want to configure load to be distributed in the port-channel based on
the traffic source and destination IP address and port number, use 'source-
destination-ip source-destination-port'
#By default, Mellanox sets it to source-destination-mac. Enter the values below only
if you intend to configure any other load balancing algorithm
#Make sure the load balancing algorithm that is set here is also replicated on the
host side
#Recommended algorithm is source-destination-ip source-destination-port
#Fill the lacp_load_balance variable only if you are using configuring interfaces on
compute nodes in bond or LAG with LACP
lacp_load_balance: "source-destination-ip source-destination-port"
#Compute Interface details
#Members to be defined with Eth in the format. Eg: Eth1/1
#Fill the mlag_id field only if you intend to configure interfaces of compute nodes
into bond or LAG with LACP
#In case you do not intend to configure LACP on interfaces of compute nodes, either
leave the mlag_id field unfilled or comment it or enter NA in the mlag_id field
#In case you have a mixed architecture where some compute nodes require LACP and some
don't,
#1. Fill the mlag_id field with appropriate MLAG ID for interfaces that connect to
compute nodes requiring LACP
#2. Either fill NA or leave the mlag_id field blank or comment it for interfaces
connecting to compute nodes that do not require LACP
#Only numerical digits between 100 to 1000 allowed for mlag_id.
#Operational link speed [variable 'speed' below] to be defined in terms of bytes.
#For 10 Gigabyte operational speed, define 10G. [Possible values - 10G and 25G]
#Interface descriptions append compute node port numbers assuming all Compute Nodes'
Port D -> Mellanox Switch A and all Compute Nodes' Port E -> Mellanox Switch B
#List the compute interfaces, their speed, MLAG IDs and their description in list of
dictionaries format
compute_interfaces:
- members: "Eth1/7"#Compute Node for ESXi, setup by NDE
  description: "HCI_Compute_Node_01"
  mlag_id: #Fill the mlag_id only if you wish to use LACP on interfaces towards
compute nodes
  speed: 25G
- members: "Eth1/8"#Compute Node for ESXi, setup by NDE
  description: "HCI_Compute_Node_02"

```

```

    mlag_id: #Fill the mlag_id only if you wish to use LACP on interfaces towards
    compute nodes
    speed: 25G
#In case of additional compute nodes, add them here in the same format as above-
members: "Eth1/9">#Compute Node for Kubernetes Worker node
    description: "HCI_Compute_Node_01"
    mlag_id: 109 #Fill the mlag_id only if you wish to use LACP on interfaces towards
    compute nodes
    speed: 10G
- members: "Eth1/10"#Compute Node for Kubernetes Worker node
    description: "HCI_Compute_Node_02"
    mlag_id: 110 #Fill the mlag_id only if you wish to use LACP on interfaces towards
    compute nodes
    speed: 10G
#Uplink Switch LACP support
#Possible options are 'yes' and 'no' - Set to 'yes' only if your uplink switch
supports LACP
uplink_switch_lacp: <>yes or no>>
#Uplink Interface details
#Members to be defined with Eth in the format. Eg: Eth1/1
#Only numerical digits between 100 to 1000 allowed for mlag_id.
#Operational link speed [variable 'speed' below] to be defined in terms of bytes.
#For 10 Gigabyte operational speed, define 10G. [Possible values in Mellanox are 1G,
10G and 25G]
#List the uplink interfaces, their description, MLAG IDs and their speed in list of
dictionaries format
uplink_interfaces:
- members: "Eth1/18"
    description_switch_a: "SwitchA:Ethx/y -> Uplink_Switch:Ethx/y"
    description_switch_b: "SwitchB:Ethx/y -> Uplink_Switch:Ethx/y"
    mlag_id: 118 #Fill the mlag_id only if 'uplink_switch_lacp' is set to 'yes'
    speed: 10G
    mtu: 1500

```



The fingerprint for the switch's key must match with that present in the host machine from where the playbook is being executed. To ensure this, add the key to `/root/.ssh/known_host` or any other appropriate location.

Rollback the Switch Configuration

1. In case of any timeout failures or partial configuration, run the following command to roll back the switch to the initial state.

```
configuration switch-to pre-ansible
```



This operation requires a reboot of the switch.

2. Switch the configuration to the state before running the Ansible playbook.

```
configuration delete post-ansible
```

3. Delete the post-ansible file that had the configuration from the Ansible playbook.

```
configuration write to post-ansible
```

4. Create a new file with the same name post-ansible, write the pre-ansible configuration to it, and switch to the new configuration to restart configuration.

IP Address Requirements

The deployment of the NetApp HCI inferencing platform with VMware and Kubernetes requires multiple IP addresses to be allocated. The following table lists the number of IP addresses required. Unless otherwise indicated, addresses are assigned automatically by NDE.

IP Address Quantity	Details	VLAN ID	IP Address
One per storage and compute node*	HCI terminal user interface (TUI) addresses	16	
One per vCenter Server (VM)	vCenter Server management address	3488	
One per management node (VM)	Management node IP address		
One per ESXi host	ESXi compute management addresses		
One per storage/witness node	NetApp HCI storage node management addresses		
One per storage cluster	Storage cluster management address		
One per ESXi host	VMware vMotion address	3489	
Two per ESXi host	ESXi host initiator address for iSCSI storage traffic	3490	

IP Address Quantity	Details	VLAN ID	IP Address
Two per storage node	Storage node target address for iSCSI storage traffic		
Two per storage cluster	Storage cluster target address for iSCSI storage traffic		
Two for mNode	mNode iSCSI storage access		

The following IPs are assigned manually when the respective components are configured.

IP Address Quantity	Details	VLAN ID	IP Address
One for Deployment Jump Management network	Deployment Jump VM to execute Ansible playbooks and configure other parts of the system – management connectivity	3488	
One per Kubernetes master node – management network	Kubernetes master node VMs (three nodes)	3488	
One per Kubernetes worker node – management network	Kubernetes worker nodes (two nodes)	3488	
One per Kubernetes worker node – NFS network	Kubernetes worker nodes (two nodes)	3491	
One per Kubernetes worker node – application network	Kubernetes worker nodes (two nodes)	3487	
Three for ONTAP Select – management network	ONTAP Select VM	3488	
One for ONTAP Select – NFS network	ONTAP Select VM – NFS data traffic	3491	
At least two for Triton Inference Server Load Balancer – application network	Load balancer IP range for Kubernetes load balancer service	3487	

*This validation requires the initial setup of the first storage node TUI address. NDE automatically assigns the TUI address for subsequent nodes.

DNS and Timekeeping Requirement

Depending on your deployment, you might need to prepare DNS records for your NetApp HCI system. NetApp HCI requires a valid NTP server for timekeeping; you can use a publicly available time server if you do not have one in your environment.

This validation involves deploying NetApp HCI with a new VMware vCenter Server instance using a fully qualified domain name (FQDN). Before deployment, you must have one Pointer (PTR) record and one Address (A) record created on the DNS server.

[Next: Virtual Infrastructure with Automated Deployment](#)

Deploy VMware Virtual Infrastructure on NetApp HCI with NDE (Automated Deployment)

NDE Deployment Prerequisites

Consult the [NetApp HCI Prerequisites Checklist](#) to see the requirements and recommendations for NetApp HCI before you begin deployment.

1. Network and switch requirements and configuration
2. Prepare required VLAN IDs
3. Switch configuration
4. IP Address Requirements for NetApp HCI and VMware
5. DNS and time-keeping requirements
6. Final preparations

NDE Execution

Before you execute the NDE, you must complete the rack and stack of all components, configuration of the network switches, and verification of all prerequisites. You can execute NDE by connecting to the management address of a single storage node if you plan to allow NDE to automatically configure all addresses.

NDE performs the following tasks to bring an HCI system online:

1. Installs the storage node (NetApp Element software) on a minimum of two storage nodes.
2. Installs the VMware hypervisor on a minimum of two compute nodes.
3. Installs VMware vCenter to manage the entire NetApp HCI stack.
4. Installs and configures the NetApp storage management node (mNode) and NetApp Monitoring Agent.



This validation uses NDE to automatically configure all addresses. You can also set up DHCP in your environment or manually assign IP addresses for each storage node and compute node. These steps are not covered in this guide.

As mentioned previously, this validation uses a two-cable configuration for compute nodes.

Detailed steps for the NDE are not covered in this document.

For step-by-step guidance on completing the deployment of the base NetApp HCI platform, see the [Deployment guide](#).

5. After NDE has finished, login to the vCenter and create a Distributed Port Group [NetApp HCI VDS 01-NFS_Network](#) for the NFS network to be used by ONTAP Select and the application.

[Next: Configure NetApp H615c \(Manual Deployment\)](#)

Configure NetApp H615c (Manual Deployment)

In this solution, the NetApp H615c compute nodes are configured as Kubernetes worker nodes. The Inferencing workload is hosted on these nodes.

Deploying the compute nodes involves the following tasks:

- Install Ubuntu 18.04.4 LTS.
- Configure networking for data and management access.
- Prepare the Ubuntu instances for Kubernetes deployment.

Install Ubuntu 18.04.4 LTS

The following high-level steps are required to install the operating system on the H615c compute nodes:

1. Download Ubuntu 18.04.4 LTS from [Ubuntu releases](#).
2. Using a browser, connect to the IPMI of the H615c node and launch Remote Control.
3. Map the Ubuntu ISO using the Virtual Media Wizard and start the installation.
4. Select one of the two physical interfaces as the [Primary network interface](#) when prompted.

An IP from a DHCP source is allocated when available, or you can switch to a manual IP configuration later. The network configuration is modified to a bond-based setup after the OS has been installed.

5. Provide a hostname followed by a domain name.
6. Create a user and provide a password.
7. Partition the disks according to your requirements.

8. Under Software Selection, select **OpenSSH server** and click Continue.

9. Reboot the node.

Configure Networking for Data and Management Access

The two physical network interfaces of the Kubernetes worker nodes are set up as a bond and VLAN interfaces for management and application, and NFS data traffic is created on top of it.



The inferencing applications and associated containers use the application network for connectivity.

1. Connect to the console of the Ubuntu instance as a user with root privileges and launch a terminal session.
2. Navigate to `/etc/netplan` and open the `01-netcfg.yaml` file.
3. Update the netplan file based on the network details for the management, application, and NFS traffic in your environment.

The following template of the netplan file was used in this solution:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp59s0f0: #Physical Interface 1
      match:
        macaddress: <<mac_address Physical Interface 1>>
      set-name: enp59s0f0
      mtu: 9000
    enp59s0f1: # Physical Interface 2
      match:
        macaddress: <<mac_address Physical Interface 2>>
      set-name: enp59s0f1
      mtu: 9000
  bonds:
    bond0:
      mtu: 9000
      dhcp4: false
      dhcp6: false
      interfaces: [ enp59s0f0, enp59s0f1 ]
      parameters:
        mode: 802.3ad
        mii-monitor-interval: 100
  vlans:
```

```
vlan.3488: #Management VLAN
  id: 3488
  xref: bond0
  dhcp4: false
  addresses: [ipv4_address/subnet]
  routes:
    - to: 0.0.0.0/0
      via: 172.21.232.111
      metric: 100
      table: 3488
    - to: x.x.x.x/x # Additional routes if any
      via: y.y.y.y
      metric: <>metric>>
      table: <>table #>>
  routing-policy:
    - from: 0.0.0.0/0
      priority: 32768#Higher Priority than table 3487
      table: 3488
  nameservers:
    addresses: [nameserver_ip]
    search: [ search_domain ]
  mtu: 1500
vlan.3487:
  id: 3487
  xref: bond0
  dhcp4: false
  addresses: [ipv4_address/subnet]
  routes:
    - to: 0.0.0.0/0
      via: 172.21.231.111
      metric: 101
      table: 3487
    - to: x.x.x.x/x
      via: y.y.y.y
      metric: <>metric>>
      table: <>table #>>
  routing-policy:
    - from: 0.0.0.0/0
      priority: 32769#Lower Priority
      table: 3487
  nameservers:
    addresses: [nameserver_ip]
    search: [ search_domain ]
  mtu: 1500    vlan.3491:
  id: 3491
  xref: bond0
  dhcp4: false
  addresses: [ipv4_address/subnet]
```

```
mtu: 9000
```

4. Confirm that the priorities for the routing policies are lower than the priorities for the main and default tables.
5. Apply the netplan.

```
sudo netplan --debug apply
```

6. Make sure that there are no errors.
7. If Network Manager is running, stop and disable it.

```
systemctl stop NetworkManager  
systemctl disable NetworkManager
```

8. Add a host record for the server in DNS.
9. Open a VI editor to [/etc/iproute2/rt_tables](#) and add the two entries.

```
#  
# reserved values  
#  
255    local  
254    main  
253    default  
0      unspec  
#  
# local  
#  
#1     inr.ruhep  
101   3488  
102   3487
```

10. Match the table number to what you used in the netplan.
11. Open a VI editor to [/etc/sysctl.conf](#) and set the value of the following parameters.

```
net.ipv4.conf.default.rp_filter=0  
net.ipv4.conf.all.rp_filter=0net.ipv4.ip_forward=1
```

12. Update the system.

```
sudo apt-get update && sudo apt-get upgrade
```

13. Reboot the system
14. Repeat steps 1 through 13 for the other Ubuntu instance.

[Next: Set Up the Deployment Jump and the Kubernetes Master Node VMs \(Manual Deployment\)](#)

Set Up the Deployment Jump VM and the Kubernetes Master Node VMs (Manual Deployment)

A Deployment Jump VM running a Linux distribution is used for the following purposes:

- Deploying ONTAP Select using an Ansible playbook
- Deploying the Kubernetes infrastructure with NVIDIA DeepOps and GPU Operator
- Installing and configuring NetApp Trident

Three more VMs running Linux are set up; these VMs are configured as Kubernetes Master Nodes in this solution.

Ubuntu 18.04.4 LTS was used in this solution deployment.

1. Deploy the Ubuntu 18.04.4 LTS VM with VMware tools

You can refer to the high-level steps described in section [Install Ubuntu 18.04.4 LTS](#).

2. Configure the in-band management network for the VM. See the following sample netplan template:

```
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    ens160:
      dhcp4: false
      addresses: [ipv4_address/subnet]
      routes:
        - to: 0.0.0.0/0
          via: 172.21.232.111
          metric: 100
          table: 3488
      routing-policy:
        - from: 0.0.0.0/0
          priority: 32768
          table: 3488
      nameservers:
        addresses: [nameserver_ip]
        search: [ search_domain ]
    mtu: 1500
```

This template is not the only way to setup the network. You can use any other approach that you prefer.

3. Apply the netplan.

```
sudo netplan --debug apply
```

4. Stop and disable Network Manager if it is running.

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

5. Open a VI editor to [/etc/iproute2/rt_tables](#) and add a table entry.

```
#  
# reserved values  
#  
255    local  
254    main  
253    default  
0      unspec  
#  
# local  
#  
#1      inr.ruhep  
101    3488
```

6. Add a host record for the VM in DNS.
7. Verify outbound internet access.
8. Update the system.

```
sudo apt-get update && sudo apt-get upgrade
```

9. Reboot the system.
10. Repeat steps 1 through 9 to set up the other three VMs.

[Next: Deploy a Kubernetes Cluster with NVIDIA DeepOps \(Automated Deployment\)](#)

Deploy a Kubernetes Cluster with NVIDIA DeepOps Automated Deployment

To deploy and configure the Kubernetes Cluster with NVIDIA DeepOps, complete the following steps:

1. Make sure that the same user account is present on all the Kubernetes master and worker nodes.
2. Clone the DeepOps repository.

```
git clone https://github.com/NVIDIA/deepops.git
```

3. Check out a recent release tag.

```
cd deepops  
git checkout tags/20.08
```

If this step is skipped, the latest development code is used, not an official release.

4. Prepare the Deployment Jump by installing the necessary prerequisites.

```
./scripts/setup.sh
```

5. Create and edit the Ansible inventory by opening a VI editor to [deepops/config/inventory](#).

- a. List all the master and worker nodes under [all].
- b. List all the master nodes under [kube-master]
- c. List all the master nodes under [etcd]
- d. List all the worker nodes under [kube-node]

```
#####
# ALL NODES
# NOTE: Use existing hostnames here, DeepOps will config them
#####
[all]
hci-ai-k8-master-01      ansible_host=172.21.232.114
hci-ai-k8-master-02      ansible_host=172.21.232.115
hci-ai-k8-master-03      ansible_host=172.21.232.116
hci-ai-k8-worker-01      ansible_host=172.21.232.109
hci-ai-k8-worker-02      ansible_host=172.21.232.110

#####
# KUBERNETES
#####
[kube-master]
hci-ai-k8-master-01
hci-ai-k8-master-02
hci-ai-k8-master-03

# Odd number of nodes required
[etcd]
hci-ai-k8-master-01
hci-ai-k8-master-02
hci-ai-k8-master-03

# Also add mgmt/master nodes here if they will run non kube-node
[kube-node]
hci-ai-k8-worker-01
hci-ai-k8-worker-02

[k8s-cluster:children]
kube-master
kube-node
```

6. Enable GPUOperator by opening a VI editor to `deepops/config/group_vars/k8s-cluster.yml`.

```
# Provide option to use GPU Operator instead of setting up NVIDIA driver and
# Docker configuration.
deepops_gpu_operator_enabled: true
```

7. Set the value of `deepops_gpu_operator_enabled` to true.

8. Verify the permissions and network configuration.

```
ansible all -m raw -a "hostname" -k -K
```

- If SSH to the remote hosts requires a password, use -k.
- If sudo on the remote hosts requires a password, use -K.

9. If the previous step passed without any issues, proceed with the setup of Kubernetes.

```
ansible-playbook --limit k8s-cluster playbooks/k8s-cluster.yml -k -K
```

10. To verify the status of the Kubernetes nodes and the pods, run the following commands:

```
kubectl get nodes
```

```
rarvind@deployment-jump:~/deepops$ kubectl get nodes
NAME           STATUS  ROLES   AGE    VERSION
hci-ai-k8-master-01  Ready  master  2d19h  v1.17.6
hci-ai-k8-master-02  Ready  master  2d19h  v1.17.6
hci-ai-k8-master-03  Ready  master  2d19h  v1.17.6
hci-ai-k8-worker-01  Ready  <none>  2d19h  v1.17.6
hci-ai-k8-worker-02  Ready  <none>  2d19h  v1.17.6
```

```
kubectl get pods -A
```

It can take a few minutes for all the pods to run.

NAMESPACE	NAME	READY	STATUS
default	gpu-operator-74c97448d9-ppdlc	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-master-ffccb57dx9wtl	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-worker-2lr9t	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-worker-616x7	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-worker-jf696	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-worker-tmtwv	1/1	Running
default	nvidia-gpu-operator-node-feature-discovery-worker-z4nlh	1/1	Running
gpu-operator-resources	nvidia-container-toolkit-daemonset-7jbl4	1/1	Running
gpu-operator-resources	nvidia-container-toolkit-daemonset-x5ktb	1/1	Running
gpu-operator-resources	nvidia-dcgm-exporter-5x94p	1/1	Running
gpu-operator-resources	nvidia-dcgm-exporter-7cb1	1/1	Running
gpu-operator-resources	nvidia-device-plugin-daemonset-n8vrk	1/1	Running
gpu-operator-resources	nvidia-device-plugin-daemonset-z7j6s	1/1	Running
gpu-operator-resources	nvidia-device-plugin-validation	0/1	Completed
gpu-operator-resources	nvidia-driver-daemonset-7h752	1/1	Running
gpu-operator-resources	nvidia-driver-daemonset-v4rbj	1/1	Running
gpu-operator-resources	nvidia-driver-validation	0/1	Completed
kube-system	calico-kube-controllers-777478f4ff-jknxg	1/1	Running
kube-system	calico-node-2j9mr	1/1	Running
kube-system	calico-node-czk76	1/1	Running
kube-system	calico-node-jpdxn	1/1	Running
kube-system	calico-node-nwnvn	1/1	Running
kube-system	calico-node-ssjrx	1/1	Running
kube-system	coredns-76798d84dd-5pvgf	1/1	Running
kube-system	coredns-76798d84dd-w7l2j	1/1	Running
kube-system	dns-autoscaler-85f898cd5c-qqrbp	1/1	Running
kube-system	kube-apiserver-hci-ai-k8-master-01	1/1	Running
kube-system	kube-apiserver-hci-ai-k8-master-02	1/1	Running
kube-system	kube-apiserver-hci-ai-k8-master-03	1/1	Running
kube-system	kube-controller-manager-hci-ai-k8-master-01	1/1	Running
kube-system	kube-controller-manager-hci-ai-k8-master-02	1/1	Running
kube-system	kube-controller-manager-hci-ai-k8-master-03	1/1	Running
kube-system	kube-proxy-5znxk	1/1	Running
kube-system	kube-proxy-fk6h6	1/1	Running
kube-system	kube-proxy-hphfb	1/1	Running
kube-system	kube-proxy-qzxhr	1/1	Running
kube-system	kube-proxy-rkjds	1/1	Running
kube-system	kube-scheduler-hci-ai-k8-master-01	1/1	Running
kube-system	kube-scheduler-hci-ai-k8-master-02	1/1	Running
kube-system	kube-scheduler-hci-ai-k8-master-03	1/1	Running
kube-system	kubernetes-dashboard-5fcff756f-dmswt	1/1	Running
kube-system	kubernetes-metrics-scrapers-747b4fd5cd-4q4p2	1/1	Running
kube-system	nginx-proxy-hci-ai-k8-worker-01	1/1	Running
kube-system	nginx-proxy-hci-ai-k8-worker-02	1/1	Running
kube-system	nodelocaldns-2dmjr	1/1	Running
kube-system	nodelocaldns-b7xrw	1/1	Running
kube-system	nodelocaldns-jrhs2	1/1	Running
kube-system	nodelocaldns-jztzs	1/1	Running
kube-system	nodelocaldns-wgx84	1/1	Running

11. Verify that the Kubernetes setup can access and use the GPUs.

```
./scripts/k8s_verify_gpu.sh
```

Expected sample output:

```
rarvind@deployment-jump:~/deepops$ ./scripts/k8s_verify_gpu.sh
job_name=cluster-gpu-tests
Node found with 3 GPUs
Node found with 3 GPUs
total_gpus=6
Creating/Deleting sandbox Namespace
updating test yaml
downloading containers ...
```

job.batch/cluster-gpu-tests condition met

executing ...

Mon Aug 17 16:02:45 2020

```
+-----+  
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00     CUDA Version: 10.2 |  
+-----+  
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |  
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |  
|-----+-----+-----+-----+-----+-----+  
| 0  Tesla T4          On   | 00000000:18:00.0 Off |                0 |  
| N/A  38C    P8    10W / 70W |      0MiB / 15109MiB |      0%     Default |  
+-----+-----+-----+-----+-----+-----+
```

```
+-----+  
| Processes:                               GPU Memory |  
| GPU      PID  Type  Process name           Usage  |  
|-----+-----+-----+-----+-----+-----+  
| No running processes found               |  
+-----+-----+-----+-----+-----+-----+
```

Mon Aug 17 16:02:45 2020

```
+-----+  
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00     CUDA Version: 10.2 |  
+-----+  
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |  
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |  
|-----+-----+-----+-----+-----+-----+  
| 0  Tesla T4          On   | 00000000:18:00.0 Off |                0 |  
| N/A  38C    P8    10W / 70W |      0MiB / 15109MiB |      0%     Default |  
+-----+-----+-----+-----+-----+-----+
```

```
+-----+  
| Processes:                               GPU Memory |  
| GPU      PID  Type  Process name           Usage  |  
|-----+-----+-----+-----+-----+-----+  
| No running processes found               |  
+-----+-----+-----+-----+-----+-----+
```

Mon Aug 17 16:02:45 2020

```
+-----+  
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00     CUDA Version: 10.2 |  
+-----+  
| GPU  Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |  
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |  
|-----+-----+-----+-----+-----+-----+  
| 0  Tesla T4          On   | 00000000:18:00.0 Off |                0 |  
| N/A  38C    P8    10W / 70W |      0MiB / 15109MiB |      0%     Default |  
+-----+-----+-----+-----+-----+-----+
```

```
+-----+  
| Processes:                               GPU Memory |  
| GPU      PID  Type  Process name           Usage  |  
|-----+-----+-----+-----+-----+-----+  
| No running processes found               |  
+-----+-----+-----+-----+-----+-----+
```

```
| No running processes found
+-----+
Mon Aug 17 16:02:45 2020
+-----+
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00    CUDA Version: 10.2 |
|-----+-----+-----+-----+-----+-----+-----+-----+
| GPU Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0 Tesla T4          On | 00000000:18:00.0 Off |                0 |
| N/A 38C   P8    10W / 70W |      0MiB / 15109MiB |      0%    Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU       PID  Type  Process name        Usage   |
|-----+-----+-----+-----+-----+-----+
| No running processes found
+-----+
Mon Aug 17 16:02:45 2020
+-----+
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00    CUDA Version: 10.2 |
|-----+-----+-----+-----+-----+-----+-----+-----+
| GPU Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0 Tesla T4          On | 00000000:18:00.0 Off |                0 |
| N/A 38C   P8    10W / 70W |      0MiB / 15109MiB |      0%    Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU       PID  Type  Process name        Usage   |
|-----+-----+-----+-----+-----+-----+
| No running processes found
+-----+
Mon Aug 17 16:02:45 2020
+-----+
| NVIDIA-SMI 440.64.00     Driver Version: 440.64.00    CUDA Version: 10.2 |
|-----+-----+-----+-----+-----+-----+-----+-----+
| GPU Name      Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0 Tesla T4          On | 00000000:18:00.0 Off |                0 |
| N/A 38C   P8    10W / 70W |      0MiB / 15109MiB |      0%    Default |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                               GPU Memory |
| GPU       PID  Type  Process name        Usage   |
|-----+-----+-----+-----+-----+-----|

```

```
| No running processes found  
+-----+  
Number of Nodes: 2  
Number of GPUs: 6  
6 / 6 GPU Jobs COMPLETED  
job.batch "cluster-gpu-tests" deleted  
namespace "cluster-gpu-verify" deleted
```

12. Install Helm on the Deployment Jump.

```
./scripts/install_helm.sh
```

13. Remove the taints on the master nodes.

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

This step is required to run the LoadBalancer pods.

14. Deploy LoadBalancer.

15. Edit the `config/helm/metallb.yaml` file and provide a range of IP ddresses in the `Application Network` to be used as LoadBalancer.

```
---  
# Default address range matches private network for the virtual cluster  
# defined in virtual/.  
# You should set this address range based on your site's infrastructure.  
configInline:  
  address-pools:  
    - name: default  
      protocol: layer2  
      addresses:  
        - 172.21.231.130-172.21.231.140#Application Network  
controller:  
  nodeSelector:  
    node-role.kubernetes.io/master: ""
```

16. Run a script to deploy LoadBalancer.

```
./scripts/k8s_deploy_loadbalancer.sh
```

17. Deploy an Ingress Controller.

```
./scripts/k8s_deploy_ingress.sh
```

Next: Deploy and Configure ONTAP Select in the VMware Virtual Infrastructure (Automated Deployment)

Deploy and Configure ONTAP Select in the VMware Virtual Infrastructure (Automated Deployment)

To deploy and configure an ONTAP Select instance within the VMware Virtual Infrastructure, complete the following steps:

1. From the Deployment Jump VM, login to the [NetApp Support Site](#) and download the ONTAP Select OVA for ESXi.
2. Create a directory OTS and obtain the Ansible roles for deploying ONTAP Select.

```
mkdir OTS
cd OTS
git clone https://github.com/NetApp/ansible.git
cd ansible
```

3. Install the prerequisite libraries.

```

pip install requests
pip install pyvmomi
Open a VI Editor and create a playbook "ots_setup.yaml" with the below content to
deploy the ONTAP Select OVA and initialize the ONTAP cluster.

---
- name: Create ONTAP Select Deploy VM from OVA (ESXi)
  hosts: localhost
  gather_facts: false
  connection: 'local'
  vars_files:
    - ots_deploy_vars.yaml
  roles:
    - na_ots_deploy
- name: Wait for 1 minute before starting cluster setup
  hosts: localhost
  gather_facts: false
  tasks:
    - pause:
        minutes: 1
- name: Create ONTAP Select cluster (ESXi)
  hosts: localhost
  gather_facts: false
  vars_files:
    - ots_cluster_vars.yaml
  roles:
    - na_ots_cluster

```

4. Open a VI editor, create a variable file [ots_deploy_vars.yaml](#), and fill in hte following parameters:

```

target_vcenter_or_esxi_host: "10.xxx.xx.xx"# vCenter IP
host_login: "yourlogin@yourlab.local" # vCenter Username
ovf_path: "/run/deploy/ovapath/ONTAPdeploy.ova"# Path to OVA on Deployment Jump VM
datacenter_name: "your-Lab"# Datacenter name in vCenter
esx_cluster_name: "your Cluster"# Cluster name in vCenter
datastore_name: "your-select-dt"# Datastore name in vCenter
mgt_network: "your-mgmt-network"# Management Network to be used by OVA
deploy_name: "test-deploy-vm"# Name of the ONTAP Select VM
deploy_ipAddress: "10.xxx.xx.xx"# Management IP Address of ONTAP Select VM
deploy_gateway: "10.xxx.xx.1"# Default Gateway
deploy_proxy_url: ""# Proxy URL (Optional and if used)
deploy_netMask: "255.255.255.0"# Netmask
deploy_product_company: "NetApp"# Name of Organization
deploy_primaryDNS: "10.xxx.xx.xx"# Primary DNS IP
deploy_secondaryDNS: ""# Secondary DNS (Optional)
deploy_searchDomains: "your.search.domain.com"# Search Domain Name

```

Update the variables to match your environment.

5. Open a VI editor, create a variable file `ots_cluster_vars.yaml`, and fill it out with the following parameters:

```
node_count: 1#Number of nodes in the ONTAP Cluster
monitor_job: true
monitor_deploy_job: true
deploy_api_url: #Use the IP of the ONTAP Select VM
deploy_login: "admin"
vcenter_login: "administrator@vsphere.local"
vcenter_name: "172.21.232.100"
esxi_hosts:
  - host_name: 172.21.232.102
  - host_name: 172.21.232.103
cluster_name: "hci-ai-ots"# Name of ONTAP Cluster
cluster_ip: "172.21.232.118"# Cluster Management IP
cluster_netmask: "255.255.255.0"
cluster_gateway: "172.21.232.1"
cluster_ontap_image: "9.7"
cluster_ntp:
  - "10.61.186.231"
cluster_dns_ips:
  - "10.61.186.231"
cluster_dns_domains:
  - "sddc.netapp.com"
mgt_network: "NetApp HCI VDS 01-Management_Network"# Name of VM Port Group for Mgmt Network
data_network: "NetApp HCI VDS 01-NFS_Network"# Name of VM Port Group for NFS Network
internal_network: ""# Not needed for Single Node Cluster
instance_type: "small"
cluster_nodes:
  - node_name: "{{ cluster_name }}-01"
    ipAddress: 172.21.232.119# Node Management IP
    storage_pool: NetApp-HCI-Datastore-02 # Name of Datastore in vCenter to use
    capacityTB: 1# Usable capacity will be ~700GB
    host_name: 172.21.232.102# IP Address of an ESXi host to deploy node
```

Update the variables to match your environment.

6. Start ONTAP Select setup.

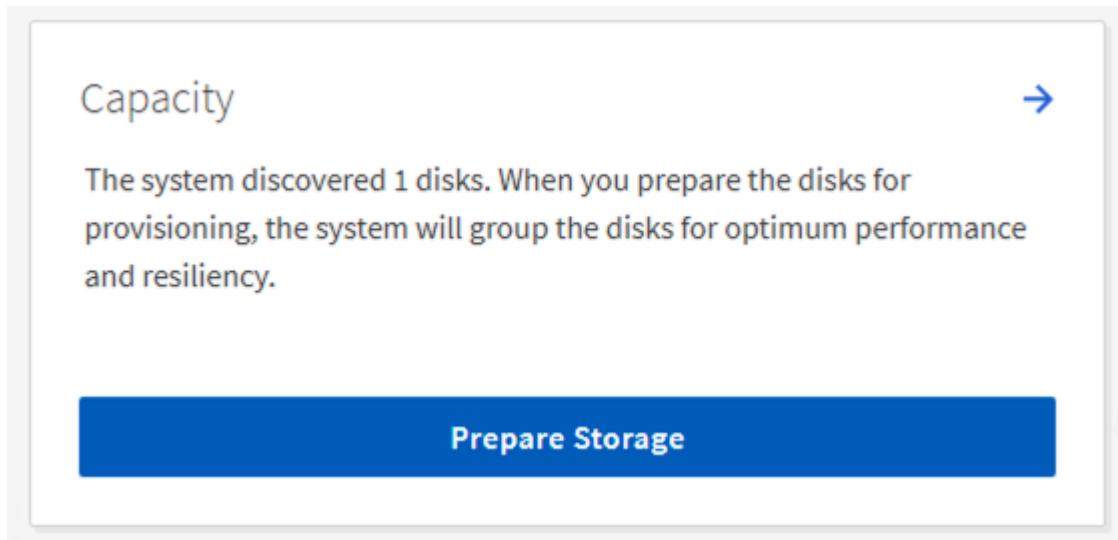
```
ansible-playbook ots_setup.yaml --extra-vars deploy_pwd=$'"P@ssw0rd"' --extra-vars
vcenter_password=$'"P@ssw0rd"' --extra-vars ontap_pwd=$'"P@ssw0rd"' --extra-vars
host_esx_password=$'"P@ssw0rd"' --extra-vars host_password=$'"P@ssw0rd"' --extra-vars
deploy_password=$'"P@ssw0rd"'
```

7. Update the command with `deploy_pwd` '(ONTAP Select VM instance), `\vcenter_password`(vCenter), `ontap_pwd` (ONTAP login password), `host_esx_password` (VMware ESXi), `host_password` (vCenter), and `deploy_password` (ONTAP Select VM instance).

Configure the ONTAP Select Cluster – Manual Deployment

To configure the ONTAP Select cluster, complete the following steps:

1. Open a browser and log into the ONTAP cluster's System Manager using its cluster management IP.
2. On the DASHBOARD page, click Prepare Storage under Capacity.



3. Select the radio button to continue without onboard key manager, and click Prepare Storage.
4. On the NETWORK page, click the + sign in the Broadcast Domains window.

Broadcast Domains		
Cluster	9000 MTU	IPspace: Cluster
Default	1500 MTU	IPspace: Default hci-ai-ots-01 e0b e0c
Mgmt	1500 MTU	IPspace: Default hci-ai-ots-01 e0a

5. Enter the Name as **NFS**, set the MTU to **9000**, and select the port **e0b**. Click Save.

Add Broadcast Domain

Specify the following details to add a new broadcast domain.

NAME

NFS

MTU

9000

ASSIGN PORTS [?](#)

Port Name	hci-ai-ots-01
e0b	<input checked="" type="checkbox"/>
e0c	<input type="checkbox"/>

Save

[Cancel](#)

6. On the DASHBOARD page, click **Configure Protocols** under Network.

Network

No protocols are enabled. To begin serving data to clients, enable the required protocols and assign the protocol addresses.

Configure Protocols

7. Enter a name for the SVM, select Enable NFS, provide an IP and subnet mask for the NFS LIF, set the Broadcast Domain to NFS, and click Save.

Configure Protocols

X

ONTAP exposes protocol services through storage VMs. [More details](#)

STORAGE VM NAME

infra-NFS-hci-ai

Access Protocol

SMB/CIFS and NFS

iSCSI

Enable SMB/CIFS

Enable NFS

DEFAULT LANGUAGE [?](#)

c.utf_8



NETWORK INTERFACE

One network interface per node is recommended.

hci-ai-ots-01

IP ADDRESS

172.21.235.119

SUBNET MASK

255.255.255.0

GATEWAY

[Add optional gateway](#)

BROADCAST DOMAIN

NFS



Save

[Cancel](#)

8. Click STORAGE in the left pane, and from the dropdown select Storage VMs

- a. Edit the SVM.

Storage VMs

+ Add

Name	State
infra-NFS-hci-ai	running

⋮

[Edit](#)

[Delete](#)

[Stop](#)

- b. Select the checkbox under Resource Allocation, make sure that the local tier is listed, and click Save.

Edit Storage VM

X

STORAGE VM NAME

infra-NFS-hci-ai

DEFAULT LANGUAGE

c.utf_8



Resource Allocation

Limit volume creation to preferred local tiers

LOCAL TIERS

hci_ai_ots_01_SSD_1 X

Cancel

Save

9. Click the SVM name, and on the right panel scroll down to Policies.
10. Click the arrow within the Export Policies tile, and click the default policy.
11. If there is a rule already defined, you can edit it; if no rule exists, then create a new one.
 - a. Select NFS Network Clients as the Client Specification.
 - b. Select the Read-Only and Read/Write checkboxes.
 - c. Select the checkbox to Allow Superuser Access.

New Rule

CLIENT SPECIFICATION

172.21.235.0/24

ACCESS PROTOCOLS

SMB/CIFS
 FlexCache
 NFS NFSv3 NFSv4

ACCESS DETAILS

Type	<input checked="" type="checkbox"/> Read-Only	<input checked="" type="checkbox"/> Read/Write
UNIX	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kerberos 5	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kerberos 5i	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kerberos 5p	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
NTLM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Allow Superuser Access

[Cancel](#)
[Save](#)

[Next: Deploy NetApp Trident \(Automated Deployment\)](#)

Deploy NetApp Trident (Automated Deployment)

NetApp Trident is deployed by using an Ansible playbook that is available with NVIDIA DeepOps. Follow these steps to set up NetApp Trident:

1. From the Deployment Jump VM, navigate to the DeepOps directory and open a VI editor to `config/group_vars/netapp-trident.yml`. The file from DeepOps lists two backends and two storage classes. In this solution only one backend and storage class are used.

Use the following template to update the file and its parameters (highlighted in yellow) to match

your environment.

```
---
```

```
# vars file for netapp-trident playbook
# URL of the Trident installer package that you wish to download and use
trident_version: "20.07.0"># Version of Trident desired
trident_installer_url: "https://github.com/NetApp/trident/releases/download/v{{trident_version }}/trident-installer-{{ trident_version }}.tar.gz"
# Kubernetes version
# Note: Do not include patch version, e.g. provide value of 1.16, not 1.16.7.
# Note: Versions 1.14 and above are supported when deploying Trident with DeepOps.
# If you are using an earlier version, you must deploy Trident manually.
k8s_version: 1.17.9# Version of Kubernetes running
# Denotes whether or not to create new backends after deploying trident
# For more info, refer to: https://netapp-trident.readthedocs.io/en/stable-v20.04/kubernetes/operator-install.html#creating-a-trident-backend
create_backends: true
# List of backends to create
# For more info on parameter values, refer to: https://netapp-trident.readthedocs.io/en/stable-v20.04/kubernetes/operations/tasks/backends/ontap.html
# Note: Parameters other than those listed below are not available when creating a backend via DeepOps
# If you wish to use other parameter values, you must create your backend manually.
backends_to_create:
  - backendName: ontap-flexvol
    storageDriverName: ontap-nas # only 'ontap-nas' and 'ontap-nas-flexgroup' are supported when creating a backend via DeepOps
    managementLIF: 172.21.232.118# Cluster Management IP or SVM Mgmt LIF IP
    dataLIF: 172.21.235.119# NFS LIF IP
    svm: infra-NFS-hci-ai# Name of SVM
    username: admin# Username to connect to the ONTAP cluster
    password: P@ssw0rd# Password to login
    storagePrefix: trident
    limitAggregateUsage: ""
    limitVolumeSize: ""
    nfsMountOptions: ""
    defaults:
      spaceReserve: none
      snapshotPolicy: none
      snapshotReserve: 0
      splitOnClone: false
      encryption: false
      unixPermissions: 777
      snapshotDir: false
      exportPolicy: default
      securityStyle: unix
```

```

tieringPolicy: none
# Add additional backends as needed
# Denotes whether or not to create new StorageClasses for your NetApp storage
# For more info, refer to: https://netapp-trident.readthedocs.io/en/stable-
v20.04/kubernetes/operator-install.html#creating-a-storage-class
create_StorageClasses: true
# List of StorageClasses to create
# Note: Each item in the list should be an actual K8s StorageClass definition in yaml
format
# For more info on StorageClass definitions, refer to https://netapp-
trident.readthedocs.io/en/stable-v20.04/kubernetes/concepts/objects.html#kubernetes-
storageclass-objects.
storageClasses_to_create:
  - apiVersion: storage.k8s.io/v1
    kind: StorageClass
    metadata:
      name: ontap-flexvol
    annotations:
      storageclass.kubernetes.io/is-default-class: "true"
    provisioner: csi.trident.netapp.io
    parameters:
      backendType: "ontap-nas"
# Add additional StorageClasses as needed
# Denotes whether or not to copy tridentctl binary to localhost
copy_tridentctl_to_localhost: true
# Directory that tridentctl will be copied to on localhost
tridentctl_copy_to_directory: ... # will be copied to 'deepops/' directory

```

2. Setup NetApp Trident by using the Ansible playbook.

```
ansible-playbook -l k8s-cluster playbooks/netapp-trident.yml
```

3. Verify that Trident is running.

```
./tridentctl -n trident version
```

The expected output is as follows:

```
rarvind@deployment-jump:~/deepops$ ./tridentctl -n trident version
+-----+-----+
| SERVER VERSION | CLIENT VERSION |
+-----+-----+
| 20.07.0        | 20.07.0        |
+-----+-----+
```

[Next: Deploy NVIDIA Triton Inference Server \(Automated Deployment\)](#)

Deploy NVIDIA Triton Inference Server (Automated Deployment)

To set up automated deployment for the Triton Inference Server, complete the following steps:

1. Open a VI editor and create a PVC yaml file `vi pvc-triton-model-repo.yaml`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: triton-pvc  namespace: triton
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
  storageClassName: ontap-flexvol
```

2. Create the PVC.

```
kubectl create -f pvc-triton-model-repo.yaml
```

3. Open a VI editor, create a deployment for the Triton Inference Server, and call the file `triton_deployment.yaml`.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-3gpu
    name: triton-3gpu
    namespace: triton
  spec:
```

```
ports:
- name: grpc-trtis-serving
  port: 8001
  targetPort: 8001
- name: http-trtis-serving
  port: 8000
  targetPort: 8000
- name: prometheus-metrics
  port: 8002
  targetPort: 8002
selector:
  app: triton-3gpu
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: triton-1gpu
    name: triton-1gpu
    namespace: triton
spec:
  ports:
- name: grpc-trtis-serving
  port: 8001
  targetPort: 8001
- name: http-trtis-serving
  port: 8000
  targetPort: 8000
- name: prometheus-metrics
  port: 8002
  targetPort: 8002
selector:
  app: triton-1gpu
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-3gpu
    name: triton-3gpu
    namespace: triton
spec:
  replicas: 1
  selector:
    matchLabels:
      app: triton-3gpu      version: v1
```

```
template:
  metadata:
    labels:
      app: triton-3gpu
      version: v1
  spec:
    containers:
      - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
        command: ["/bin/sh", "-c"]
        args: ["trtserver --model-store=/mnt/model-repo"]
        imagePullPolicy: IfNotPresent
        name: triton-3gpu
        ports:
          - containerPort: 8000
          - containerPort: 8001
          - containerPort: 8002
        resources:
          limits:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
          requests:
            cpu: "2"
            memory: 4Gi
            nvidia.com/gpu: 3
        volumeMounts:
          - name: triton-model-repo
            mountPath: /mnt/model-repo      nodeSelector:
              gpu-count: "3"
        volumes:
          - name: triton-model-repo
            persistentVolumeClaim:
              claimName: triton-pvc---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-1gpu
    name: triton-1gpu
    namespace: triton
spec:
  replicas: 3
  selector:
    matchLabels:
      app: triton-1gpu
      version: v1
  template:
    metadata:
```

```

labels:
  app: triton-1gpu
  version: v1
spec:
  containers:
    - image: nvcr.io/nvidia/tritonserver:20.07-v1-py3
      command: ["/bin/sh", "-c", "sleep 1000"]
      args: ["trtserver --model-store=/mnt/model-repo"]
      imagePullPolicy: IfNotPresent
      name: triton-1gpu
      ports:
        - containerPort: 8000
        - containerPort: 8001
        - containerPort: 8002
      resources:
        limits:
          cpu: "2"
          memory: 4Gi
          nvidia.com/gpu: 1
        requests:
          cpu: "2"
          memory: 4Gi
          nvidia.com/gpu: 1
      volumeMounts:
        - name: triton-model-repo
          mountPath: /mnt/model-repo      nodeSelector:
            gpu-count: "1"
      volumes:
        - name: triton-model-repo
          persistentVolumeClaim:
            claimName: triton-pvc

```

Two deployments are created here as an example. The first deployment spins up a pod that uses three GPUs and has replicas set to 1. The other deployment spins up three pods each using one GPU while the replica is set to 3. Depending on your requirements, you can change the GPU allocation and replica counts.

Both of the deployments use the PVC created earlier and this persistent storage is provided to the Triton inference servers as the model repository.

For each deployment, a service of type LoadBalancer is created. The Triton Inference Server can be accessed by using the LoadBalancer IP which is in the application network.

A nodeSelector is used to ensure that both deployments get the required number of GPUs without any issues.

4. Label the K8 worker nodes.

```
kubectl label nodes hci-ai-k8-worker-01 gpu-count=3  
kubectl label nodes hci-ai-k8-worker-02 gpu-count=1
```

5. Create the deployment.

```
kubectl apply -f triton_deployment.yaml
```

6. Make a note of the LoadBalancer service external LPS.

```
kubectl get services -n triton
```

The expected sample output is as follows:

```
rarvind@deployment-jump:~/triton-inference-server$ kubectl get services -n triton  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP     PORT(S)          AGE  
triton-1gpu-v20-07-v1   LoadBalancer   10.233.21.185  172.21.231.133  8001:31238/TCP,8000:30171/TCP,8002:32348/TCP  10h  
triton-3gpu-v20-07-v1   LoadBalancer   10.233.13.17   172.21.231.132  8001:31549/TCP,8000:30220/TCP,8002:31517/TCP  10h
```

7. Connect to any one of the pods that were created from the deployment.

```
kubectl exec -n triton --stdin --tty triton-1gpu-86c4c8dd64-545lx -- /bin/bash
```

8. Set up the model repository by using the example model repository.

```
git clone  
cd triton-inference-server  
git checkout r20.07
```

9. Fetch any missing model definition files.

```
cd docs/examples  
.fetch_models.sh
```

10. Copy all the models to the model repository location or just a specific model that you wish to use.

```
cp -r model_repository/resnet50_netdef/ /mnt/model-repo/
```

In this solution, only the resnet50_netdef model is copied over to the model repository as an example.

11. Check the status of the Triton Inference Server.

```
curl -v <<LoadBalancer_IP_recorded_earlier>>:8000/api/status
```

The expected sample output is as follows:

```
curl -v 172.21.231.132:8000/api/status
* Trying 172.21.231.132...
* TCP_NODELAY set
* Connected to 172.21.231.132 (172.21.231.132) port 8000 (#0)
> GET /api/status HTTP/1.1
> Host: 172.21.231.132:8000
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< NV-Status: code: SUCCESS server_id: "inference:0" request_id: 9
< Content-Length: 1124
< Content-Type: text/plain
<
id: "inference:0"
version: "1.15.0"
uptime_ns: 377890294368
model_status {
    key: "resnet50_netdef"
    value {
        config {
            name: "resnet50_netdef"
            platform: "caffe2_netdef"
            version_policy {
                latest {
                    num_versions: 1
                }
            }
        }
        max_batch_size: 128
        input {
            name: "gpu_0/data"
            data_type: TYPE_FP32
            format: FORMAT_NCHW
            dims: 3
            dims: 224
            dims: 224
        }
        output {
            name: "gpu_0/softmax"
            data_type: TYPE_FP32
            dims: 1000
            label_filename: "resnet50_labels.txt"
        }
    }
}
```

```
    }
    instance_group {
        name: "resnet50_netdef"
        count: 1
        gpus: 0
        gpus: 1
        gpus: 2
        kind: KIND_GPU
    }
    default_model_filename: "model.netdef"
    optimization {
        input_pinned_memory {
            enable: true
        }
        output_pinned_memory {
            enable: true
        }
    }
}
version_status {
    key: 1
    value {
        ready_state: MODEL_READY
        ready_state_reason {
        }
    }
}
}
ready_state: SERVER_READY
* Connection #0 to host 172.21.231.132 left intact
```

Next: [Deploy the Client for Triton Inference Server \(Automated Deployment\)](#)

Deploy the Client for Triton Inference Server (Automated Deployment)

To deploy the client for the Triton Inference Server, complete the following steps:

1. Open a VI editor, create a deployment for the Triton client, and call the file `triton_client.yaml`.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: triton-client
    name: triton-client
    namespace: triton
spec:
  replicas: 1
  selector:
    matchLabels:
      app: triton-client
      version: v1
  template:
    metadata:
      labels:
        app: triton-client
        version: v1
    spec:
      containers:
        - image: nvcr.io/nvidia/tritonserver:20.07- v1- py3-clientsdk
          imagePullPolicy: IfNotPresent
          name: triton-client
          resources:
            limits:
              cpu: "2"
              memory: 4Gi
            requests:
              cpu: "2"
              memory: 4Gi
```

2. Deploy the client.

```
kubectl apply -f triton_client.yaml
```

Next: [Collect Inference Metrics from Triton Inference Server](#)

Collect Inference Metrics from Triton Inference Server

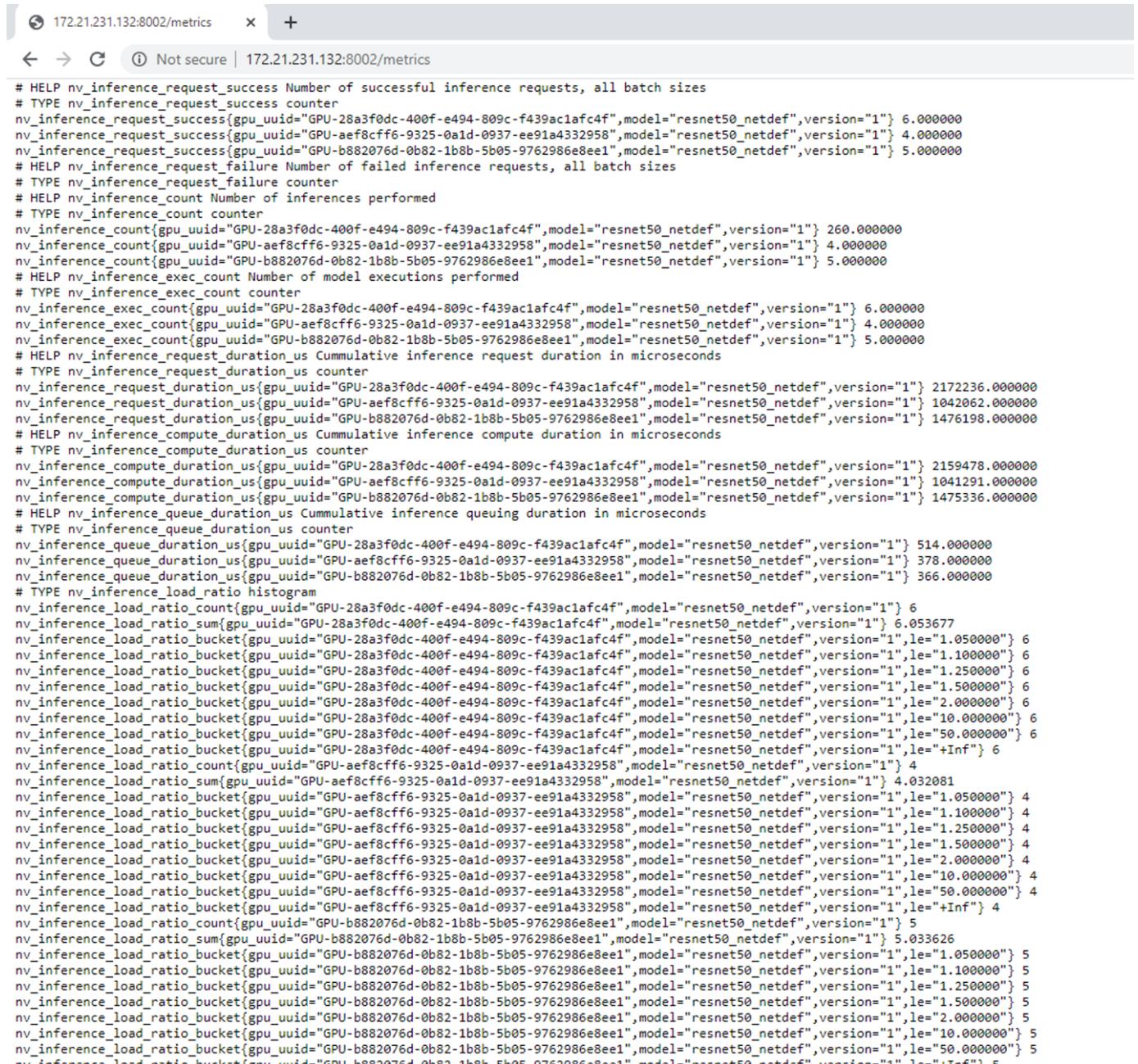
The Triton Inference Server provides Prometheus metrics indicating GPU and request statistics.

By default, these metrics are available at [http://localhost:8000/metrics](http://<a href=)

anchor="triton_inference_server_IP">[triton_inference_server_IP]:>8002/metrics" class="bare">>http://[triton_inference_server_IP]:>8002/metrics.

The Triton Inference Server IP is the LoadBalancer IP that was recorded earlier.

The metrics are only available by accessing the endpoint and are not pushed or published to any remote server.



```
# HELP nv_inference_request_success Number of successful inference requests, all batch sizes
# TYPE nv_inference_request_success counter
nv_inference_request_success{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.000000
nv_inference_request_success{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_request_success{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_request_failure Number of failed inference requests, all batch sizes
# TYPE nv_inference_request_failure counter
# HELP nv_inference_count Number of inferences performed
# TYPE nv_inference_count counter
nv_inference_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 260.000000
nv_inference_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_exec_count Number of model executions performed
# TYPE nv_inference_exec_count counter
nv_inference_exec_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.000000
nv_inference_exec_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.000000
nv_inference_exec_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.000000
# HELP nv_inference_request_duration_us Cumulative inference request duration in microseconds
# TYPE nv_inference_request_duration_us counter
nv_inference_request_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 2172236.000000
nv_inference_request_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 1042062.000000
nv_inference_request_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 1476198.000000
# HELP nv_inference_compute_duration_us Cumulative inference compute duration in microseconds
# TYPE nv_inference_compute_duration_us counter
nv_inference_compute_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 2159478.000000
nv_inference_compute_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 1041291.000000
nv_inference_compute_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 1475336.000000
# HELP nv_inference_queue_duration_us Cumulative inference queuing duration in microseconds
# TYPE nv_inference_queue_duration_us counter
nv_inference_queue_duration_us{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 514.000000
nv_inference_queue_duration_us{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 378.000000
nv_inference_queue_duration_us{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 366.000000
# TYPE nv_inference_load_ratio histogram
nv_inference_load_ratio_count{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6
nv_inference_load_ratio_sum{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1"} 6.053677
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.050000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.100000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.250000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="1.500000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="2.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="10.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="50.000000"} 6
nv_inference_load_ratio_bucket{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f",model="resnet50_netdef",version="1",le="+Inf"} 6
nv_inference_load_ratio_count{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4
nv_inference_load_ratio_sum{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1"} 4.032081
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.050000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.100000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.250000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="1.500000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="2.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="10.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="50.000000"} 4
nv_inference_load_ratio_bucket{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958",model="resnet50_netdef",version="1",le="+Inf"} 4
nv_inference_load_ratio_count{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5
nv_inference_load_ratio_sum{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1"} 5.033626
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.050000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.100000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.250000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="1.500000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="2.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="10.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="50.000000"} 5
nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="+Inf"} 5
```

```

nv_inference_load_ratio_bucket{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1",model="resnet50_netdef",version="1",le="+Inf"} 5
# HELP nv_gpu_utilization GPU utilization rate [0.0 - 1.0]
# TYPE nv_gpu_utilization gauge
nv_gpu_utilization{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 0.000000
nv_gpu_utilization{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 0.000000
nv_gpu_utilization{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 0.000000
# HELP nv_gpu_memory_total_bytes GPU total memory, in bytes
# TYPE nv_gpu_memory_total_bytes gauge
nv_gpu_memory_total_bytes{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 15843721216.000000
nv_gpu_memory_total_bytes{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 15843721216.000000
nv_gpu_memory_total_bytes{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 15843721216.000000
# HELP nv_gpu_memory_used_bytes GPU used memory, in bytes
# TYPE nv_gpu_memory_used_bytes gauge
nv_gpu_memory_used_bytes{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 1466236928.000000
nv_gpu_memory_used_bytes{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 13004767232.000000
nv_gpu_memory_used_bytes{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 1466236928.000000
# HELP nv_gpu_power_usage GPU power usage in watts
# TYPE nv_gpu_power_usage gauge
nv_gpu_power_usage{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 27.999000
nv_gpu_power_usage{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 28.428000
nv_gpu_power_usage{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 27.632000
# HELP nv_gpu_power_limit GPU power management limit in watts
# TYPE nv_gpu_power_limit gauge
nv_gpu_power_limit{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 70.000000
nv_gpu_power_limit{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 70.000000
nv_gpu_power_limit{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 70.000000
# HELP nv_energy_consumption GPU energy consumption in joules since the Triton Server started
# TYPE nv_energy_consumption counter
nv_energy_consumption{gpu_uuid="GPU-b882076d-0b82-1b8b-5b05-9762986e8ee1"} 9796.449000
nv_energy_consumption{gpu_uuid="GPU-28a3f0dc-400f-e494-809c-f439ac1afc4f"} 9997.538000
nv_energy_consumption{gpu_uuid="GPU-aef8cff6-9325-0a1d-0937-ee91a4332958"} 9669.536000

```

Next: Validation Results

Validation Results

To run a sample inference request, complete the following steps:

1. Get a shell to the client container/pod.

```
kubectl exec --stdin --tty <<client_pod_name>> -- /bin/bash
```

2. Run a sample inference request.

```
image_client -m resnet50_netdef -s INCEPTION -u <<LoadBalancer_IP_recorded
earlier>>:8000 -c 3 images/mug.jpg
```

```
root@triton-client-v20-07-v1-5566895bc-zqz6w:/workspace# image_client -m resnet50_netdef -s INCEPTION -u 172.21.231.133:8000 -c 3 images/mug.jpg
Request 0, batch size 1
Image 'images/mug.jpg':
  504 (COFFEE MUG) = 0.723991
  968 (CUP) = 0.270953
  967 (ESPRESSO) = 0.00115996
```

This inferencing request calls the `resnet50_netdef` model that is used for image recognition. Other clients can also send inferencing requests concurrently by following a similar approach and calling out the appropriate model.

Next: Where to Find Additional Information

=
:hardbreaks:
:nofooter:

```
:icons: font
:linkattrs:
:imagesdir: ./media/
```

Additional Information

To learn more about the information that is described in this document, review the following documents and/or websites:

- NetApp HCI Theory of Operations

<https://www.netapp.com/us/media/wp-7261.pdf>

- NetApp Product Documentation

docs.netapp.com

- NetApp HCI Solution Catalog Documentation

<https://docs.netapp.com/us-en/hci/solutions/index.html>

- HCI Resources page

<https://mysupport.netapp.com/info/web/ECMLP2831412.html>

- ONTAP Select

<https://www.netapp.com/us/products/data-management-software/ontap-select-sds.aspx>

- NetApp Trident

<https://netapp-trident.readthedocs.io/en/stable-v20.01/>

- NVIDIA DeepOps

<https://github.com/NVIDIA/deepops>

- NVIDIA Triton Inference Server

<https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-master-branch-guide/docs/index.html>

Copyright Information

Copyright © 2020 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademark Information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.