Deploying NetApp HCI – AI Inferencing at the Edge

HCI

NetApp
November 04, 2020

Table of Contents

Overview . . . 1
Configure Network Switches (Automated Deployment) . . . 1
Deploy VMware Virtual Infrastructure on NetApp HCI with NDE (Automated Deployment) . . . 8
Configure NetApp H615c (Manual Deployment) . . . 10
Set Up the Deployment Jump VM and the Kubernetes Master Node VMs (Manual Deployment) . . . 13
Deploy a Kubernetes Cluster with NVIDIA DeepOps Automated Deployment . . . 15
Deploy and Configure ONTAP Select in the VMware Virtual Infrastructure (Automated Deployment) . . . 23
Deploy NetApp Trident (Automated Deployment) . . . 31
Deploy NVIDIA Triton Inference Server (Automated Deployment) . . . 34
Deploy the Client for Triton Inference Server (Automated Deployment) . . . 40
Collect Inference Metrics from Triton Inference Server . . . 41

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

Network Segment	Details	VLAN ID
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	
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	

IP Address Quantity	Details	VLAN ID	IP Address
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 conf
######
[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]
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.

```
rarvind@deployment-jump:~/deepops$ kubectl get pods -A
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-6l6x7      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-7cbrl                                    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-scraper-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 yml
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                                                 |
+-----------------------------------------------------------------------------+
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.yml` 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: truemonitor_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.

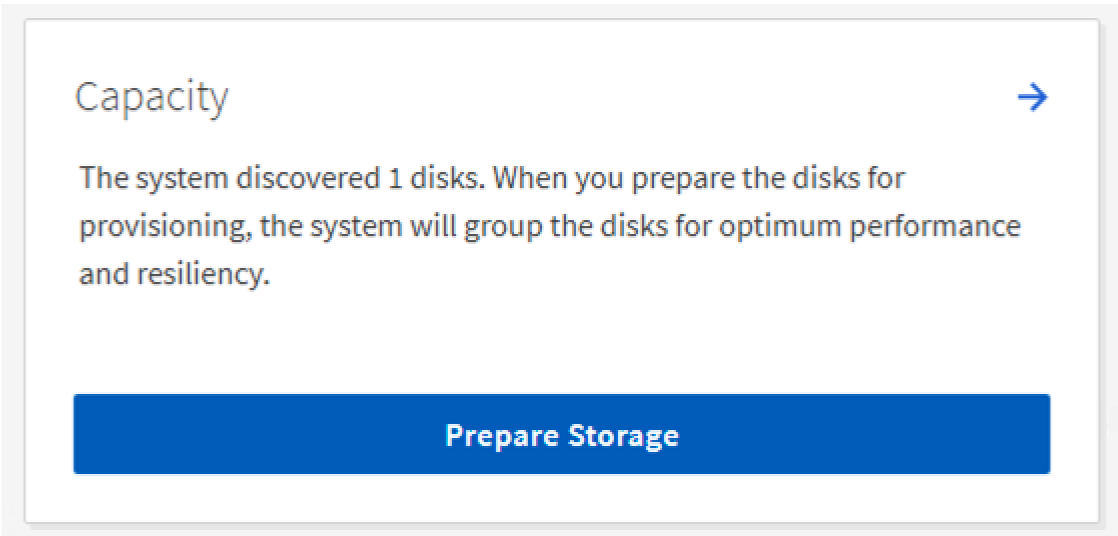
```
ansible-playbook ots_setup.yaml --extra-vars deploy_pwd=$'"P@ssw0rd"' --extra-vars
vcenter_password=$'"P@ssw0rd"' --extra-vars ontap_pwd=$'"P@ssw0rd"' --extra-vars
host_esx_password=$'"P@ssw0rd"' --extra-vars host_password=$'"P@ssw0rd"' --extra-vars
deploy_password=$'"P@ssw0rd"'
```

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).

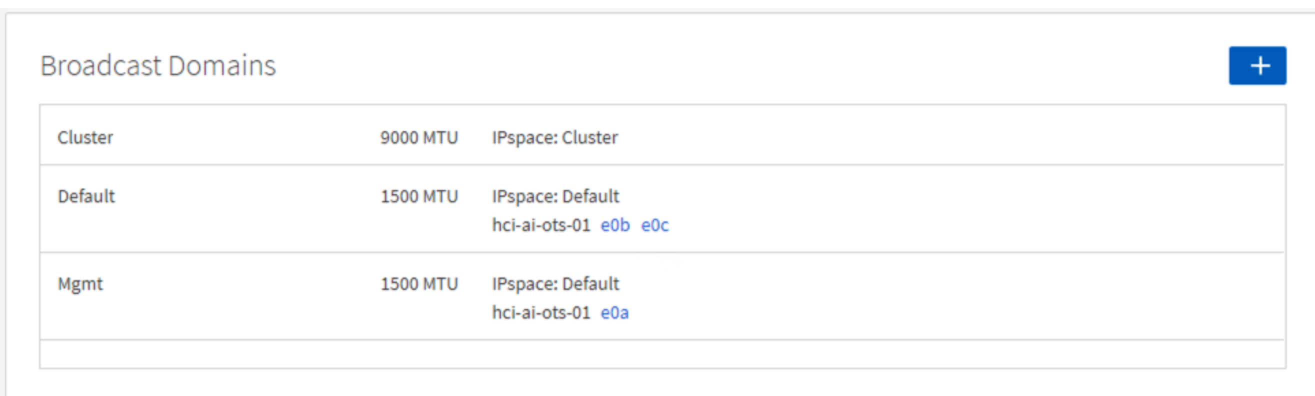
Configure the ONTAP Select Cluster – Manual Deployment

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

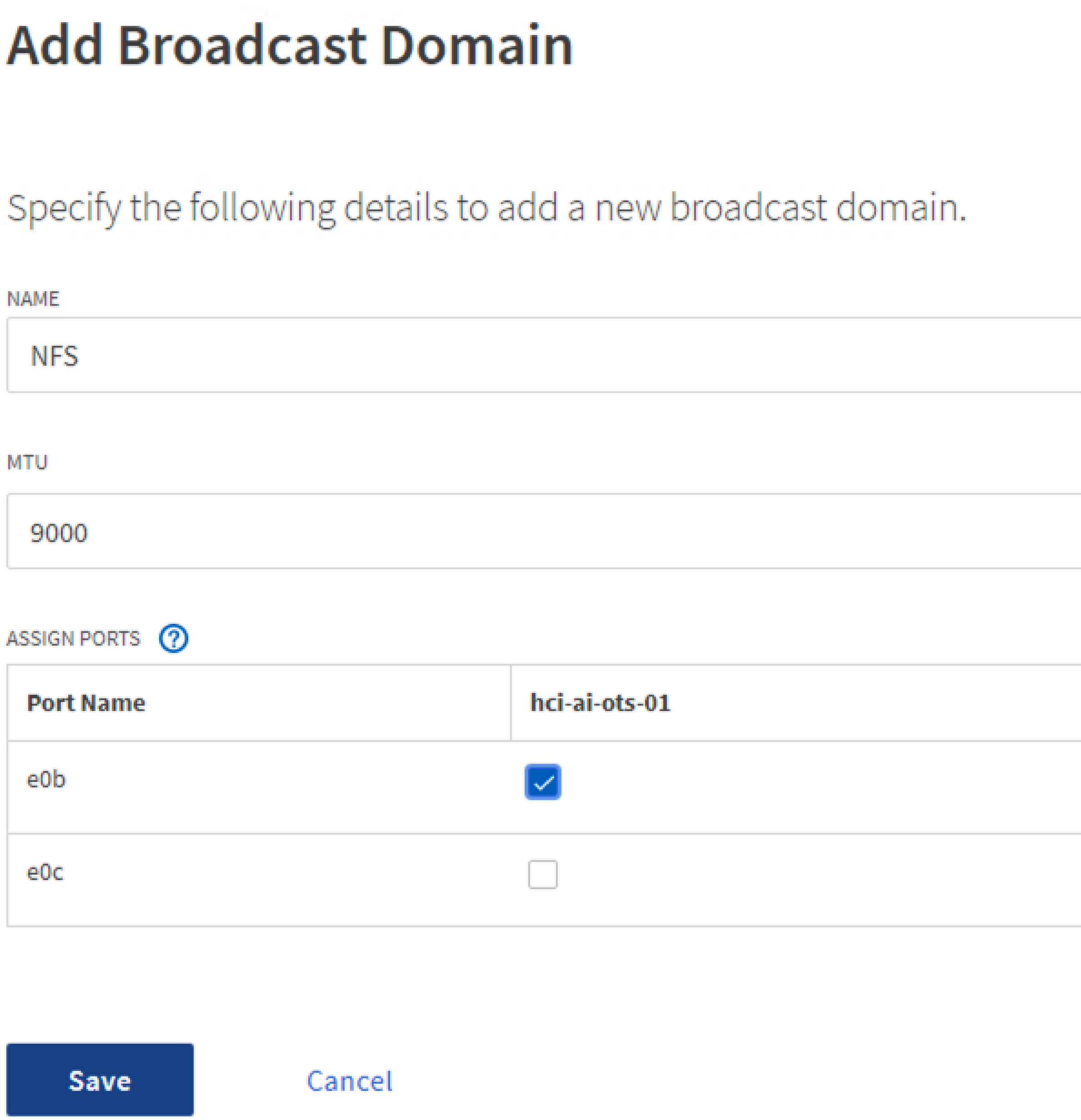
1. Open a browser and log into the ONTAP cluster's System Manager using its cluster management IP.
2. On the DASHBOARD page, click Prepare Storage under Capacity.



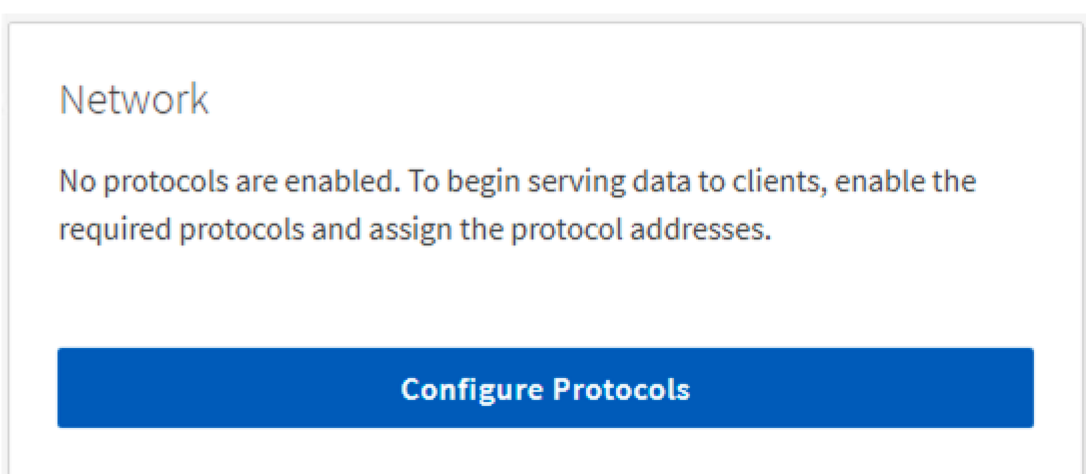
3. Select the radio button to continue without onboard key manager, and click Prepare Storage.
4. On the NETWORK page, click the `+` sign in the Broadcast Domains window.



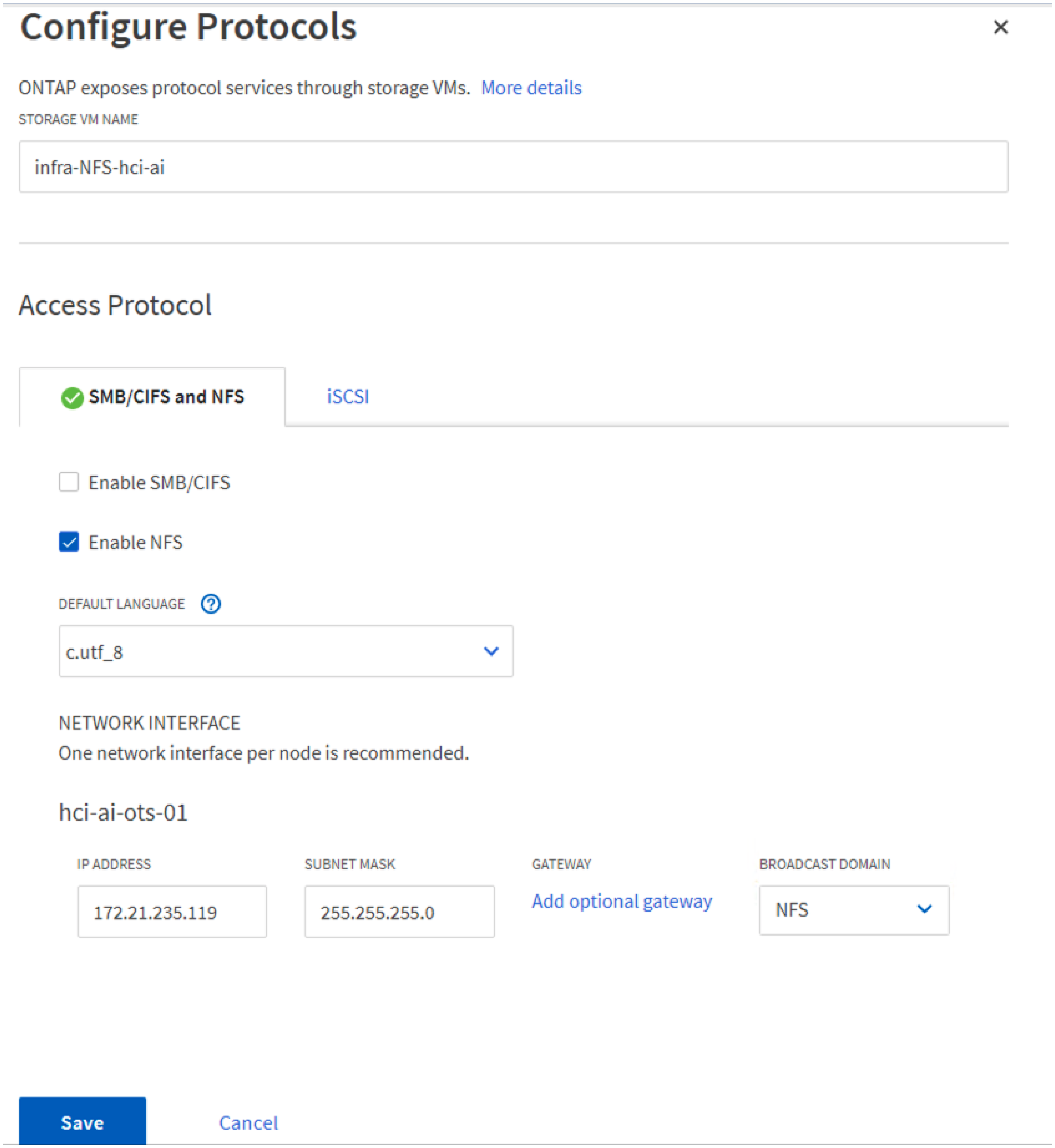
5. Enter the Name as `NFS`, set the MTU to `9000`, and select the port `e0b`. Click Save.



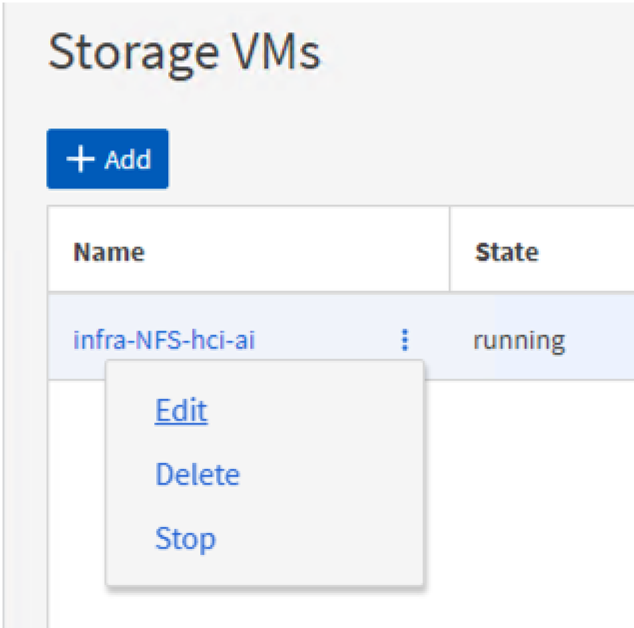
6. On the DASHBOARD page, click `Configure Protocols` under Network.



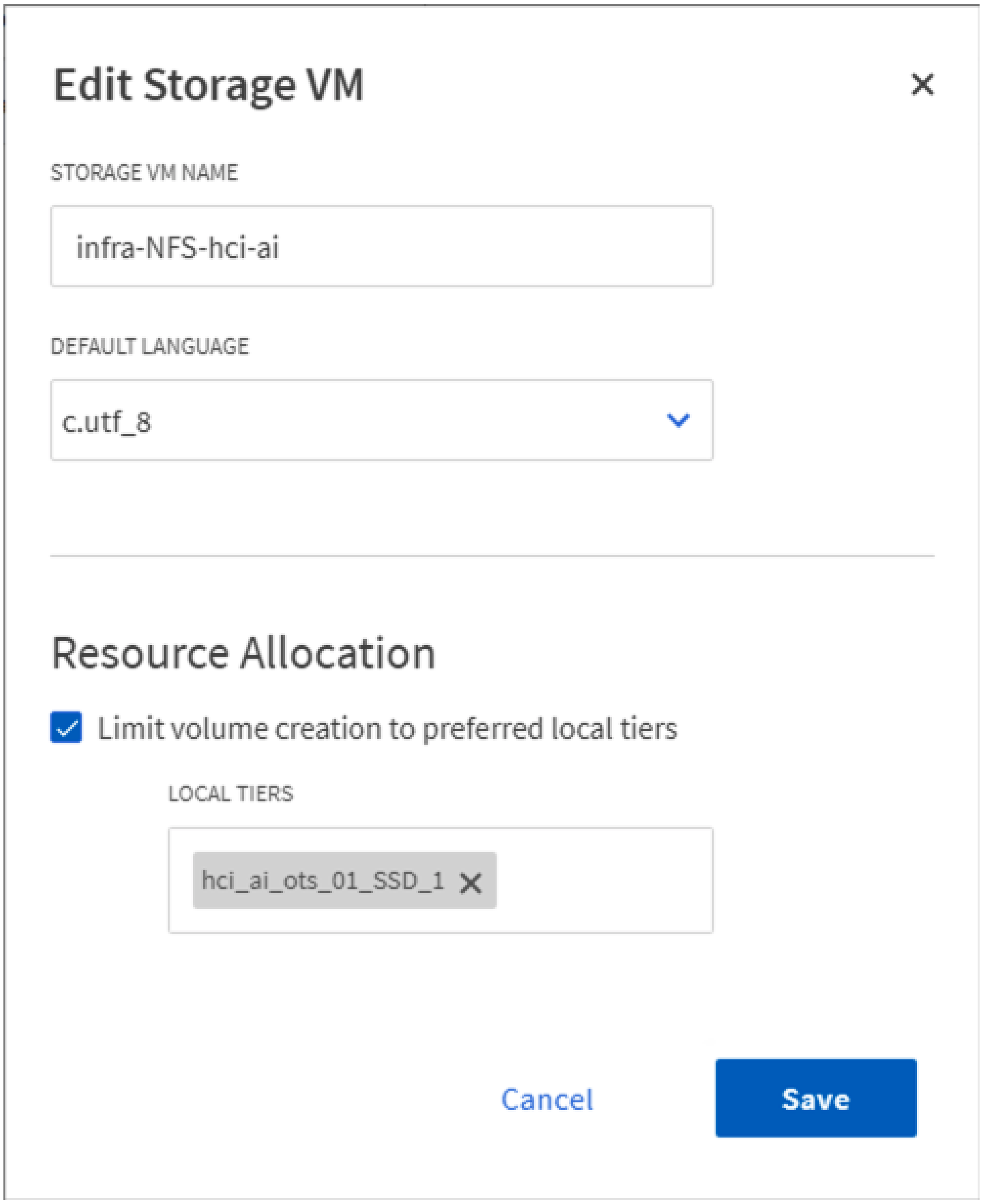
7. Enter a name for the SVM, select Enable NFS, provide an IP and subnet mask for the NFS LIF, set the Broadcast Domain to NFS, and click Save.



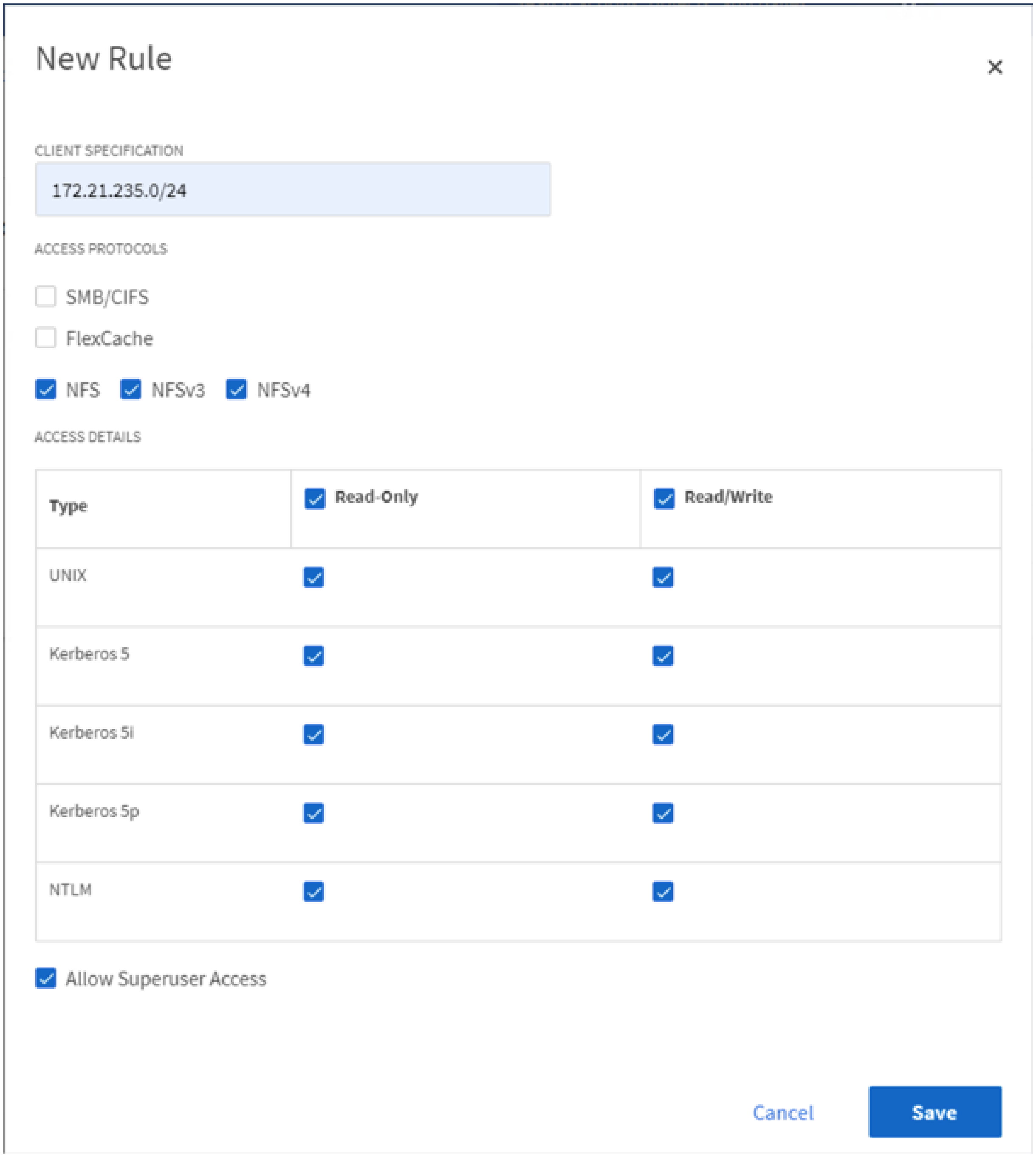
8. Click STORAGE in the left pane, and from the dropdown select Storage VMs
   a. Edit the SVM.



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



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 avaible 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 tridenctl 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
   ```

```yaml
  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

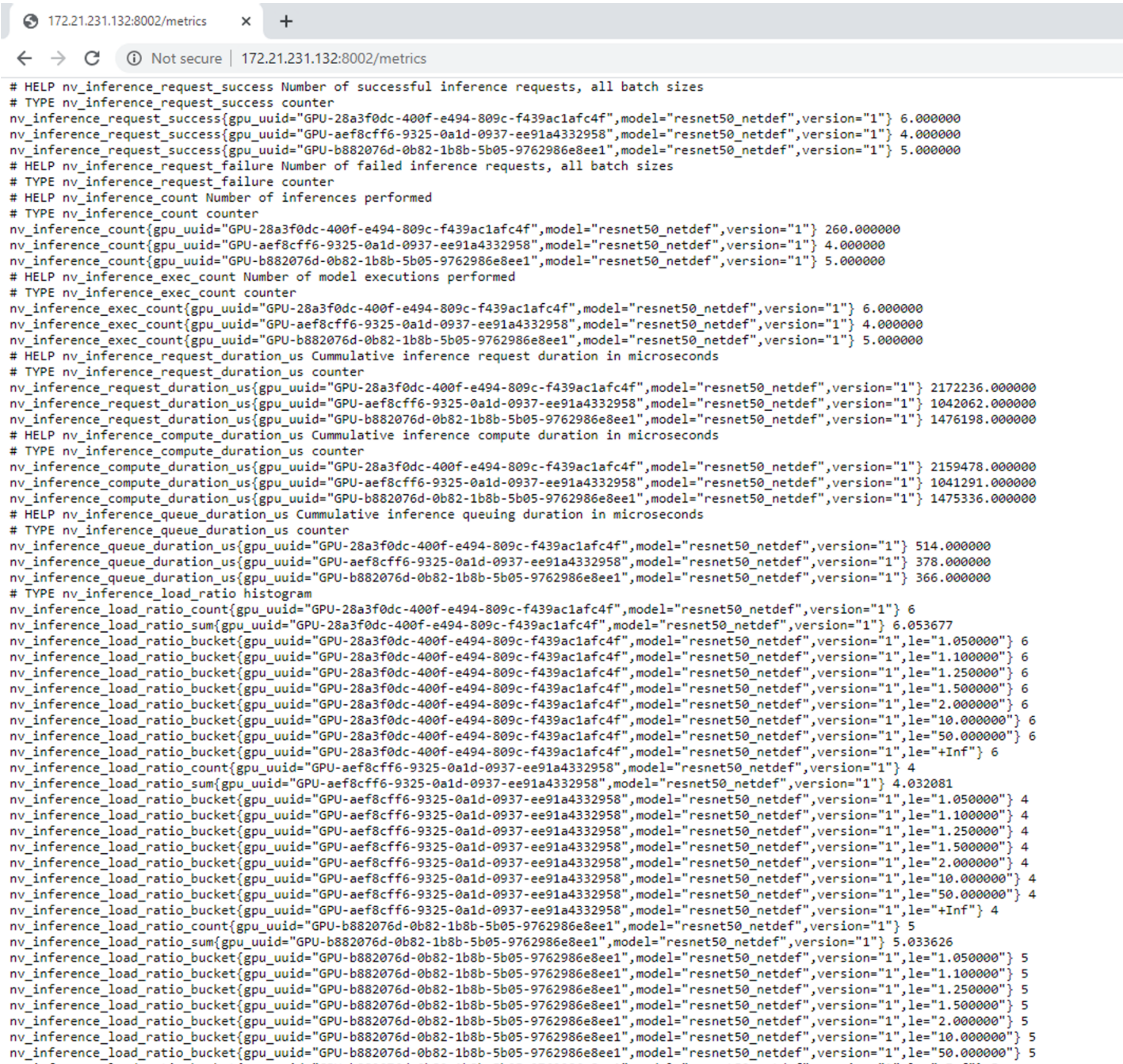
The Triton Inference Server provides Prometheus metrics indicating GPU and request statistics.

By default, these metrics are available at <a href="http://<a

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 Cummulative 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 Cummulative 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 Cummulative 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
# 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