Lenovo-Nokia AI-DC Validated Design

Lenovo-Nokia AI-DC Validated Design

3HE-21915-AAAA-TQZZA
Issue 2
September 2025

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Contents

1 Executive summary

Lenovo-Nokia AI Validated Designs is a workstream stemming from a partnership dedicated to producing validated design and implementation recommendations to the consumer based on our AI/ML datacenter portfolio across the various AI-DC segments.

This is accomplished with extensive requirement analysis from a multitude of customers along with deep research of the technology development in the industry segment to form the solutions design.

After the design has been compiled, it goes through an intense array of hardware, software, traffic, and failure tests to form the validated design. The resultant design and collateral provide the consumer with a template that can be used to deploy the solution in their own environment.

This document provides a validated design for deploying AI/ML data center solutions using a best-in-class combination of networking, compute, storage, and GPU technologies. Developed through close collaboration between leading technology vendors, the solution has been designed and tested to meet the demanding requirements of modern AI workloads across training, inference, and data processing use cases.

The design process begins with a comprehensive analysis of customer requirements and industry best practices, followed by a rigorous integration of hardware and software components. Each configuration undergoes extensive testing, including performance benchmarking, traffic validation, and fault-tolerance scenarios, to ensure reliability, scalability, and efficiency.

The resulting validated design offers customers, partners, and architects a validated design for deploying AI-ready infrastructure in enterprise and service provider environments. By leveraging this solution, organizations can accelerate adoption, reduce deployment risk, and ensure their data center investments are optimized for the next generation of AI-driven applications.

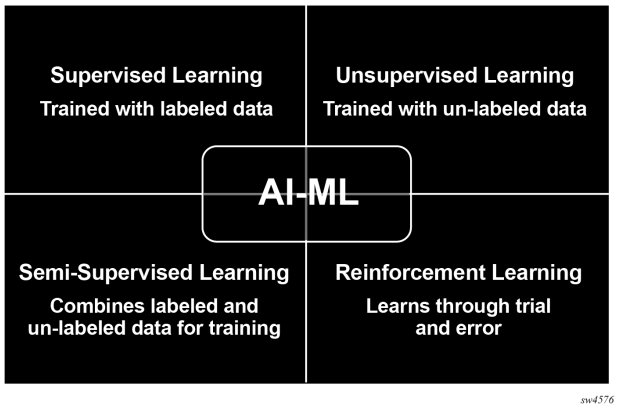
2 AI-DC training clusters

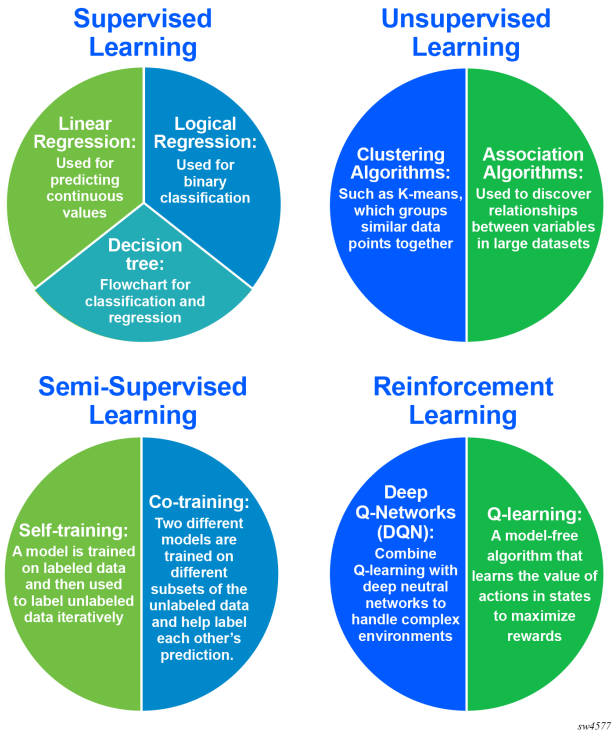
2.1 Introduction to AI/ML

2.1.1 Machine learning

Machine learning (ML) is the subset of AI that deals with creating algorithms that can be trained on data to learn and evolve. For AI backend fabric, the focus is on ML and how the network is utilized when training the model with data.

2.1.2 Types of learning and learning models





Supervised learning: a type of machine learning where a computer learns from examples that have been labeled or categorized

Training data: a dataset that contains input-output pairs; each input has a corresponding correct output (label)

Learning process: uses labeled data to learn about the relationship between inputs and outputs (labels of inputs)

Unsupervised learning: a type of machine learning where the model learns from data that has not been labeled or categorized

Training data: unlabeled and unclassified inputs; for example, pictures

Learning process: the unsupervised learning algorithm analyzes the unlabeled data to identify similarities and differences among data points

Semi-supervised learning: a type of machine learning where the model learns first from labeled and then unlabeled data

Training data: labeled and unlabeled inputs

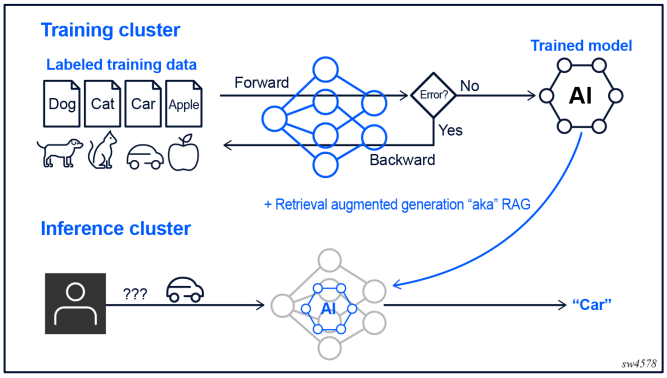
Learning process: first learn from labeled data inputs and then analyze the unlabeled data to identify similarities and differences among data points

Reinforcement Learning: a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize rewards

Training data: trial and error: the agent learns through exploration and feedback rather than through labeled data

Learning process: learns based on a reward system; correct choices are reinforced and incorrect choices are penalized

2.2 Types of clusters



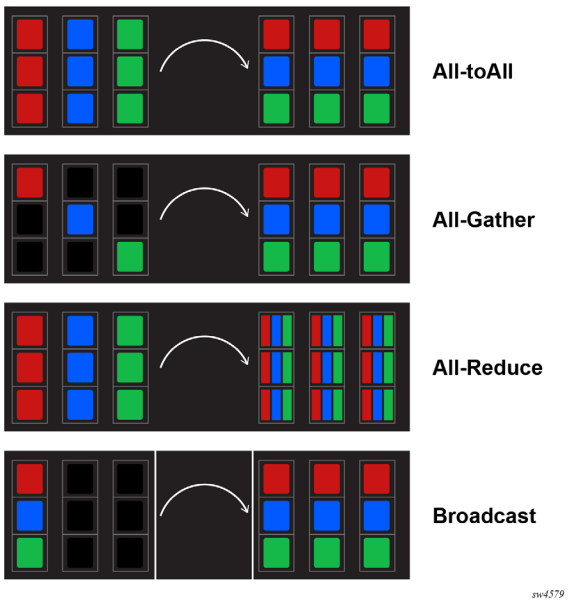
An inference cluster is used to deploy a trained model with or without reinforcements such as retrieval-augmented generation (RAG) to answer queries from the consumer based on its training data. This is less intensive than the training portions and hence has fewer infrastructure requirements.

Training clusters and inference clusters : AI-DC clusters consist of a front-end, backend compute, and backend storage networks. A detailed representation of these components is outlined in sections below.

Though they share similar components from an infrastructure design perspective, the nature of the infrastructure varies greatly because of the use cases involved.

The primary use of a training cluster is to train the learning models, which tend to have billions of parameters (internal variables of a model, which determine how comprehensive and accurate the model is) with different forms of data (labeled and unlabeled) based on the type of learning being incorporated. This is a very intensive process and requires high-end GPU servers, storage nodes, and high bandwidth networking gear.

2.3 AI/ML workload distribution



In the previous section, we discussed cluster types and their purposes.

The endpoint GPU server providers such as Nvidia and AMD have created collective libraries, called NCCL and RCCL respectively, which are essentially frameworks

A few of the AI/ML workload distribution mechanisms and their functions are:

All-to-All – This operation allows each participating process (or GPU) to send and receive data from every other process. Each process provides a specific amount of data that is distributed to all other processes.

All-Gather – This operation collects data from all processes and distributes the combined result to every process. Each process contributes its own data, and the output is an aggregation of all contributions.

All-Reduce – This operation performs a reduction (such as summation or averaging) on data across all processes and then distributes the result back to all processes. Each process starts with its own data and the result is stored in each process's memory.

Broadcast – This operation sends data from a designated root process to all other processes. The root process holds the original data, which is copied to all other ranks.

These are often intertwined with the kind of parallelism that is optimal for the model to be trained.

that guide the collective functioning of the GPUs as a cluster.

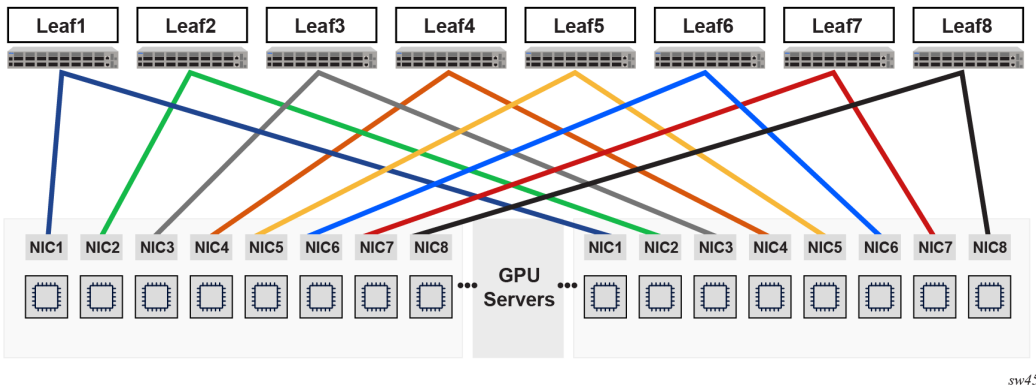
NCCL, for example, organizes the GPUs into high bandwidth zones and forms tree-ring structures that club GPU sets together for allocation of training workloads. Then it uses algorithms, as shown in the diagram above, to distribute the workload to GPUs in parcels and then synchronize among themselves.

2.4 AI-DC cluster architecture and scale

The section outlines the building blocks of an AI/ML datacenter cluster. This section describes the basic unit, called a rail-optimized stripe, and how replication of this structure results in the formation of the cluster.

Note: The design of the AI/ML cluster is defined purely by the endpoints such as the GPU, storage, and frontend servers, and the fabric is purely an isolated facilitator for each segment.

2.4.1 Rail optimization



sw4580

Rail optimization is a concept adopted first by Nvidia and then by other GPU vendors as a mechanism to minimize network interference in inter-GPU communication.

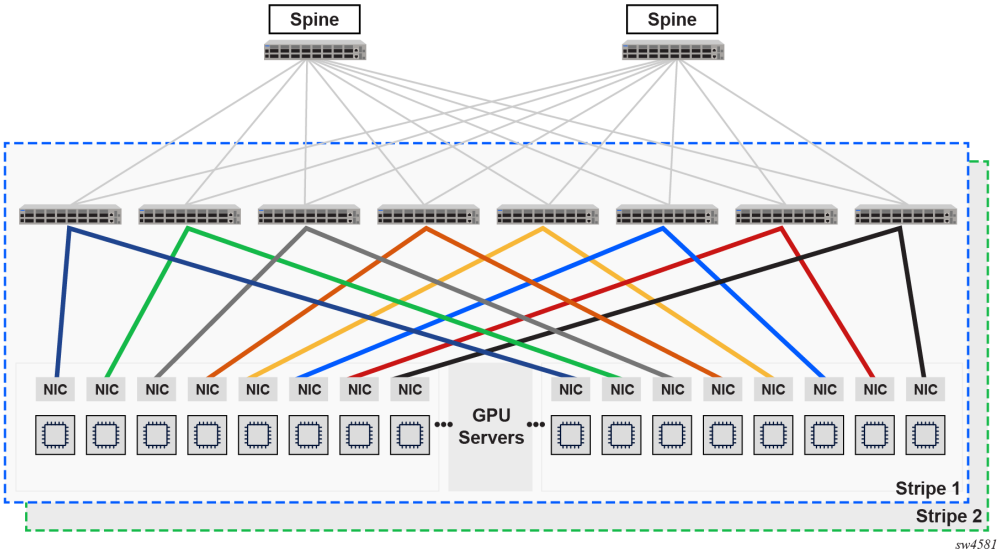
Because the earlier models of GPU servers had eight GPUs, the corresponding design of an atomic unit of a backend GPU fabric, also known as a stripe, was designed with eight leaf nodes.

Each GPU # is attached to the leaf of the same number, for example: GPU 1 of server 1 and GPU 1 of server 2 are connected to Leaf 1. This leaf, which contains all GPU #, is called a rail. Leaf 1 is “Rail 1” in this example.

With this design, intra-rail communication must only traverse the connected leaf while inter-rail communication is done via the internal switch.

2.4.2 Fully scheduled stripe

The rail-optimized stripe is the atomic unit of an AI-backend GPU cluster.



A **stripe** is an atomic unit of a GPU cluster.

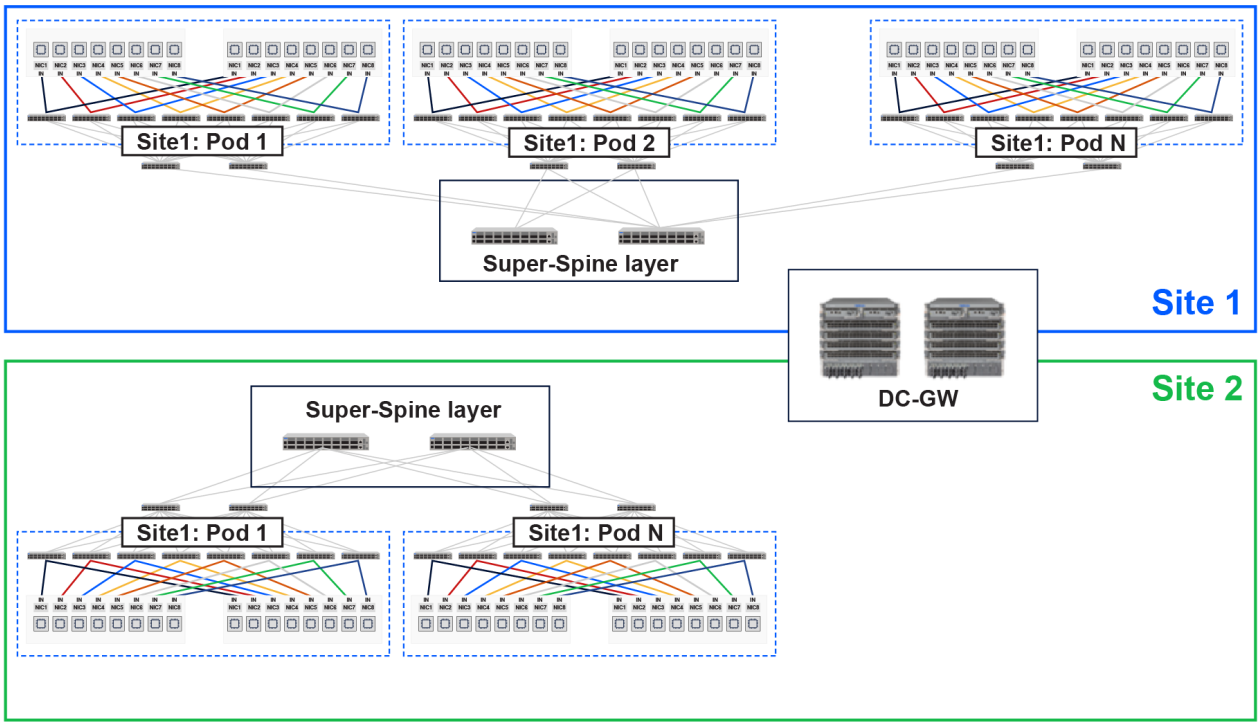
In the previous section, we discussed rail optimization which resulted in eight leafs being the design choice for rail optimization because the GPU servers could accommodate eight GPUs.

This eight leaf design, along with corresponding spines formed a stripe. By definition, a full stripe is the maximum number of GPUs that can be supported by these eight leafs.

For example: with a Nokia 7220-IXR 32 port switch in the leaf role, assuming an oversubscription ratio of 1, the number of GPUs in 1 stripe will be 16 *8 which is 128 GPUs.

The number of spines and the port density of the spines are chosen based on the scale out requirements of the cluster, that is, how many stripes are needed in the cluster.

2.4.3 Cluster scale



sw4582

The architecture shown above details the mechanism by which a backend training cluster is built.

The basic unit of a pod is a rail-optimized stripe. A stripe is fully scheduled when all the south-bound ports of all eight leafs are filled.

The pod is considered fully scheduled when all the ports of the spines are utilized. The port radix of the spines defines the size of the pod.

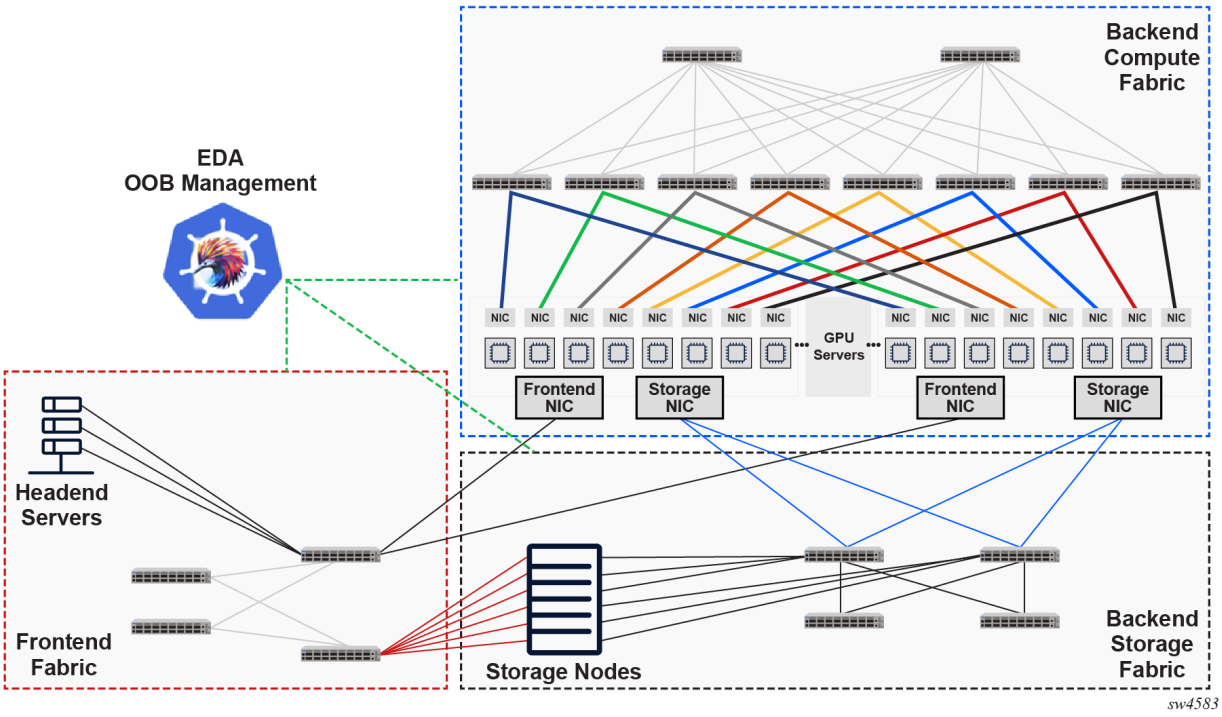
This architecture focuses on scale out; for example, using the Nokia 7220-IXR-H5-64x800G, each stripe can have 512 GPUs and each spine can support two stripes, which means each pod can have 1024 GPUs. We can extend the number of pods with the super-spine layer.

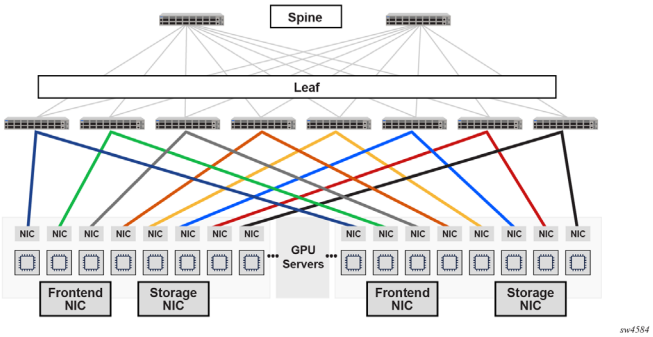
The pods can scale out within a site via a super-spine layer. The prescribed oversubscription ratio from the GPUs to the super-spine layer is 2.5:1, but this is highly subjective to the kind of workloads being deployed and the resultant traffic patterns across various layers of the fabric.

Multisite clusters replicate the same design paradigm, connect to each other via gateway routers, and employ stitching and handoff mechanisms to connect the site.

A rail or a high bandwidth zone defined by RCCL or NCCL collectives can extend from a single pod to multiple pods over multiple sites.

2.5 Components of an AI/ML cluster

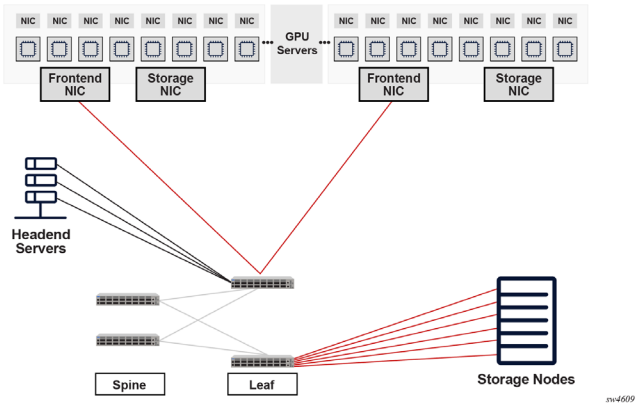




Backend compute fabric is fabric that hosts the GPU servers.

The GPU direct ports on the server are connected to this fabric. The number of GPU direct ports is usually equal to the number of GPUs in the higher-end training servers, such as the AMD Instinct MI300X and comparable Nvidia servers, such as the H200.

All AI/ML training optimization concepts, such as rail optimization, dynamic load-balancing, and congestion control, are relevant to the backend compute fabric.

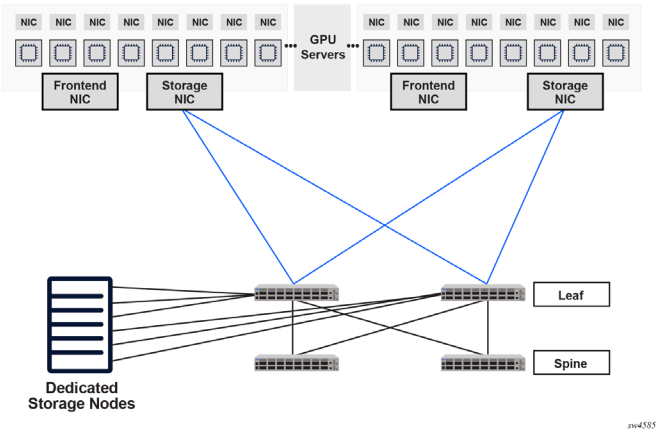


Front-end fabric is connected to the in-band front end ports of the GPU servers and the dedicated storage nodes.

All training and inference jobs are scheduled via the servers connected to the front-end fabric.

The front-end fabric is connected to the headend servers and the internet.

The front-end fabric is also connected to the dedicated storage portions to manage volumes and checkpoints.



Backend storage fabric connects the storage ports of the GPU server to the dedicated storage nodes.

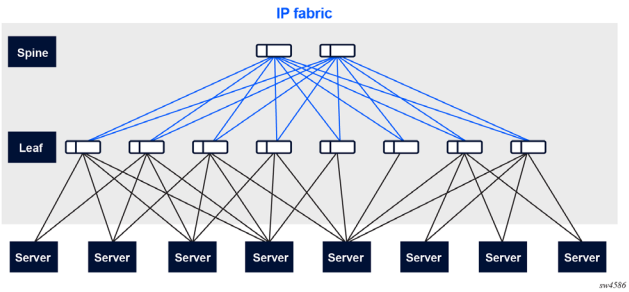
The dedicated storage nodes are used to store the data processed by the GPUs during a training or inference process. WEKA/VAST/PURE storage are examples of dedicated storage node vendors.

The number of dedicated storage nodes required depends on the number of GPUs present in the cluster.

Unlike the compute fabric, there are no GPU direct ports to the storage fabric and there are usually only one or two NICs per GPU server going to the storage fabric because compute-level traffic is not expected here.

2.6 Network design considerations

Training cluster compute - single tenant



Cost, capacity, and power are also important aspects; for instance, there are multilayer switches with a higher port radix that can accommodate the newer GPU servers and can be deployed as leafs, however, these are generally more expensive and consume more power and hence are not suitable for small to medium clusters.

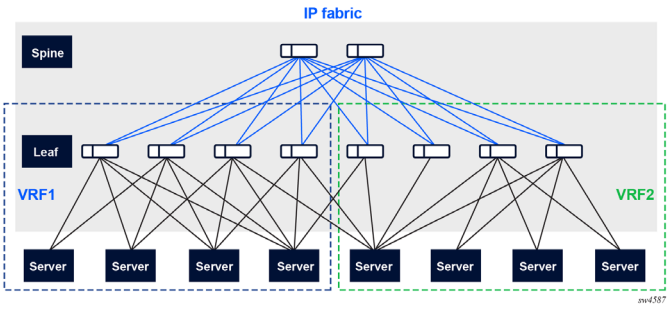
For the single tenant training compute cluster, some of the design considerations are:

There should be minimal hops between the GPUs so that the latency is low.

The packet header overhead needs to be minimum and hence a pure IP fabric or L2 fabric is chosen over any sort of tunneling.

Over subscription and availability of GPU facing ports is another design consideration. If it is a large-sized cluster with multiple stripes and a lot of traffic is expected to and from the spines, the oversubscription ratio should be close to 1. The leaf to super-spine layer oversubscription ratio is kept close to 2.5:1, however this is highly subjective to traffic patterns and individual customer use-cases.

Training cluster compute – multitenant

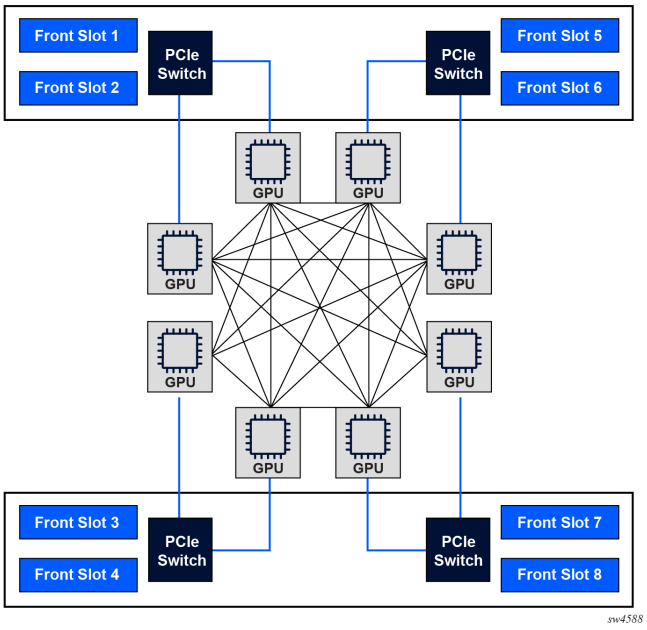


While all the considerations mentioned in the single tenant scenario still hold true, sometimes the GPU clusters are not on premise and are given out as a service, by service providers who own co-colocation spaces and the end customers can rent or lease GPU server compute as per their requirement.

In cases where the servers are leased out statically, we can maintain the simplicity of the design and bring in segmentation with VRF-lite where different customers have access to different subsets of GPUs in the cluster.

2.7 Traffic flow in AI/ML DC clusters

2.7.1 Architecture of a Lenovo GPU server

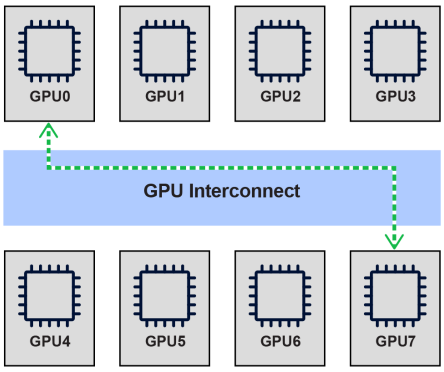


The figure shows the GPU switching reference architecture for the Lenovo SR685a GPU server capable of hosting the AMD MI300X series GPU.

The GPUs, as shown, have a full mesh connectivity with each other via the infinity fabric, which means every GPU in the server is one hop away from every other GPU.

Note that the PCIe switch is connected to their respective GPUs and provides access to the eight GPU direct ports, which connect to the compute fabric.

2.7.2 Intra-node communication

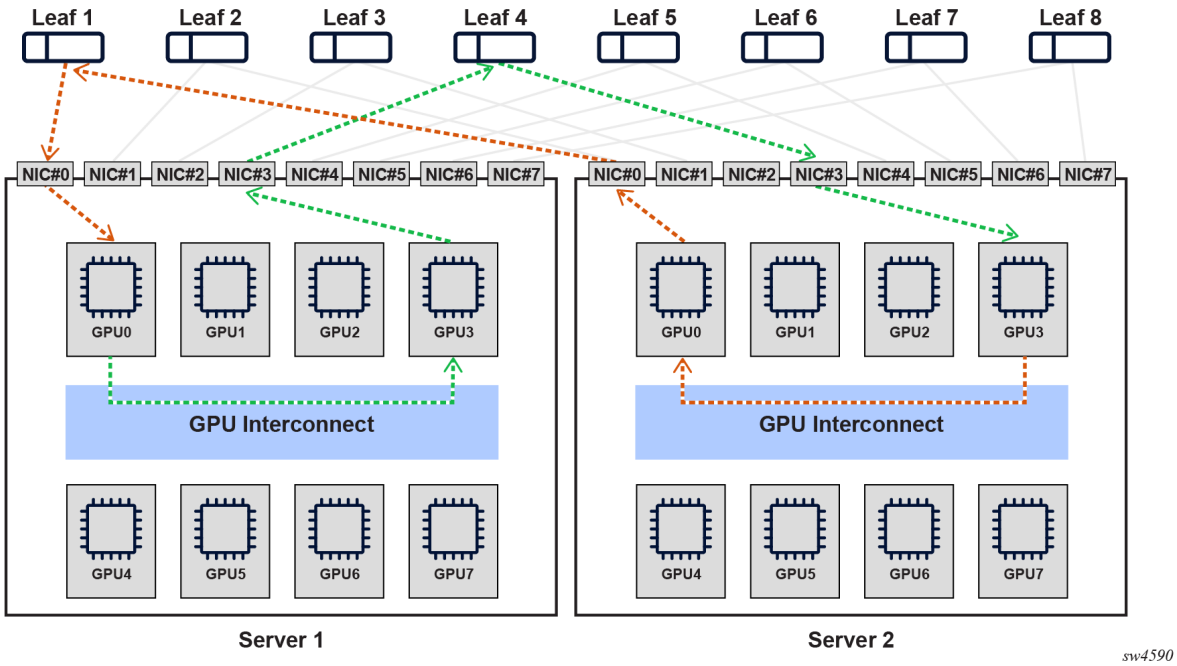


Intra-node communication refers to the communication between GPUs in a single server.

We can see in the Lenovo SR 685a schema that each AMD Mi 300X GPU has full mesh connectivity to every other GPU via the internal switch.

For example, if GPU0 wants to talk to the GPU7 rail, it can place the payload internally via the Infinity fabric or its evolution “Accelerated fabric link” and switch it to GPU7.

2.7.3 Inter-node communication



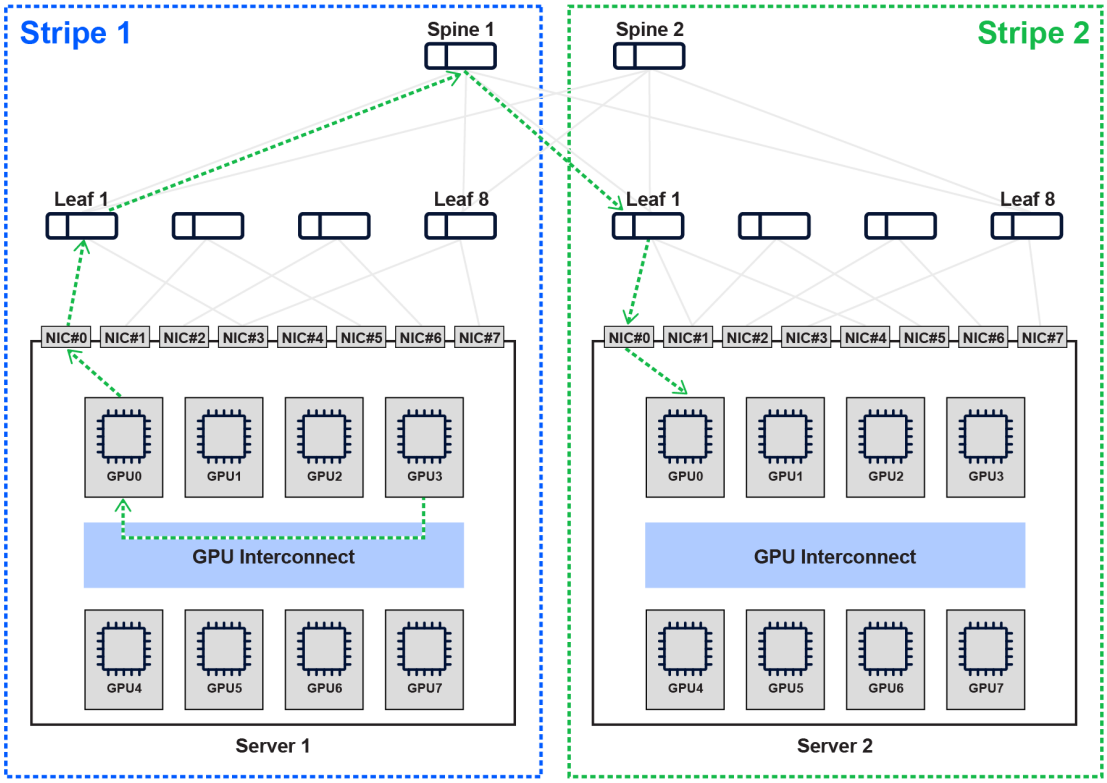
Inter-node communication refers to the communication between GPUs in a single stripe.

In a rail-optimized fabric, each GPU # is connected to the same leaf; that is, the GPU0 of all servers in the stripe are connected to Leaf1.

For example, if GPU0 on Server1 wants to send data to GPU3 on Server2, it first sends it via the GPU interconnect to its own GPU3, as shown in the previous example. Then, GPU3 on Server1 sends it to Leaf4, and then to GPU3 on Server2, as indicated by the green path.

If GPU3 on Server2 wants to send data to GPU0 on Server1, it sends it to its own GPU0, which then forwards the data to Leaf1, and then to GPU0 on Server 1, as indicated by the orange path.

2.7.4 Inter-stripe communication



A stripe is defined as all the GPUs that are connected to a set of eight leafs.

In this scenario, there are two stripes, depicted by the green and blue labels. Each stripe is connected to the same set of spines. Server 1 is part of stripe 1 and server 2 is part of stripe 2, which means they need to traverse the spine to reach each other.

In the example above, if GPU3 of server 1 wants to communicate with GPU 0 of Server 2, it places the payload onto GPU0 of server 1 via the infinity link GPU interconnect. GPU0 is connected to Leaf 1 of stripe 1 and GPU0 of server 2 is connected to Leaf1 of stripe 2.

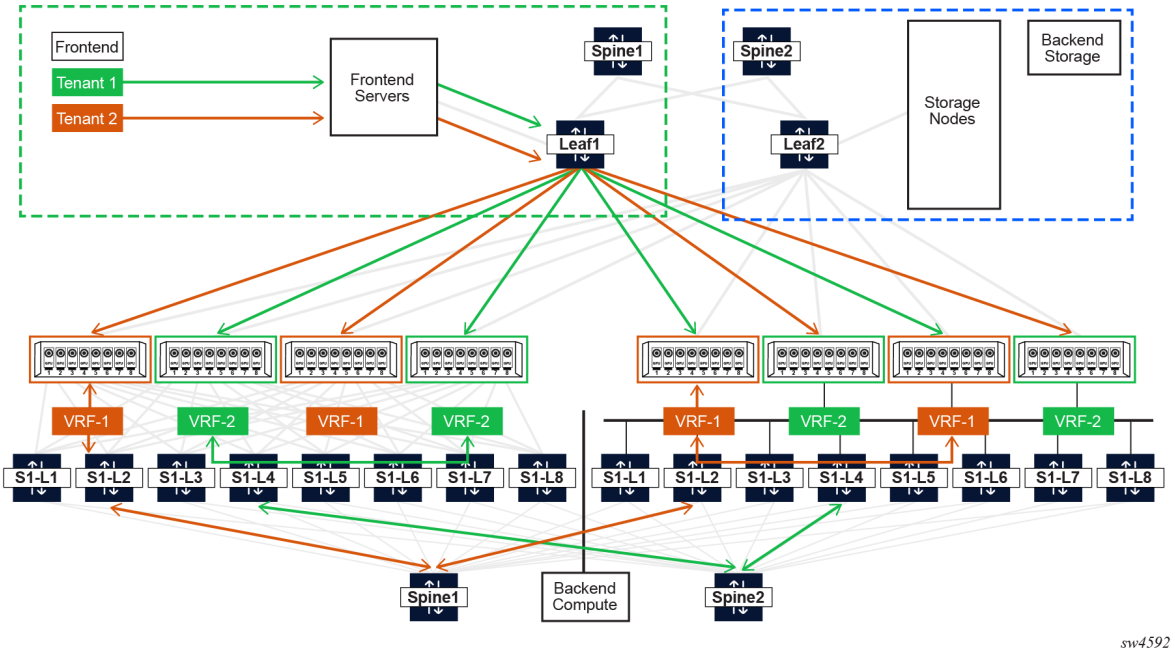
Hence, GPU0 of server 1 sends the packet to one of the spines, which is decided by the load balancing mechanism (which is spine1 in this example), and spine1 sends it down to GPU0 in stripe 2.

2.8 Multitenancy in AI/ML clusters

The main design consideration for multitenancy in an AI/ML cluster is that the GPU servers today have internal switching mechanisms that allow all the GPUs within a server to communicate with each other without traversing the external network. Due to this, there are limitations on how multitenancy can be achieved via the network fabric, and the control of this operation remains with the endpoints.

Some of the mechanisms of achieving multitenant clusters are discussed below.

2.8.1 Server isolation



One of the mechanisms of providing multitenancy in an AI/ML cluster is server isolation. The positive aspect of this design is that it can entirely be orchestrated via the fabric. The endpoints are isolated by limiting their visibility in the fabric.

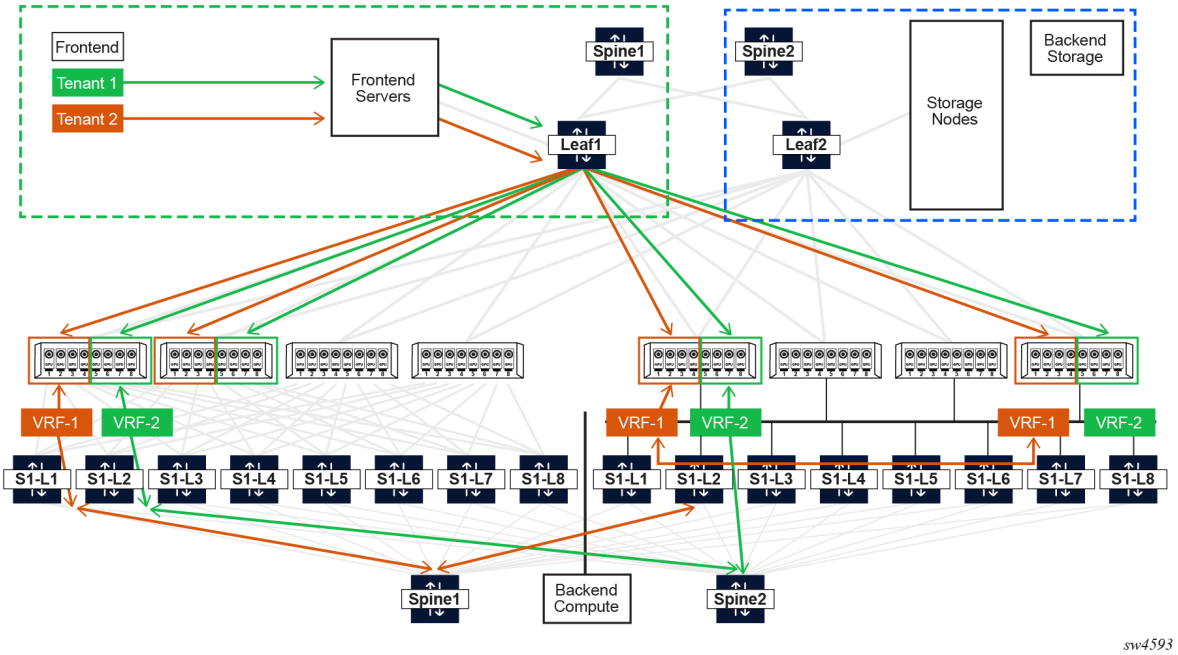
The challenge, however, is that the isolation is limited to multiples of servers, which means that we can only provide full servers to the tenants, as shown in the diagram.

With this mechanism, we cannot provide isolation at the GPU level because the recommendation by the vendor is that the internal switching mechanism remains enabled, and hence all the GPUs in a server are able to talk to each other.

The design is to make sure that all the GPU ports of servers allocated to a particular tenant are allocated to an IP-VRF, as shown in the figure above. This can be achieved either with VRF-lite on an IP fabric-based solution or an EVPN/VxLAN based solution. In an EVPN fabric, the routes from the Servers will be translated into T5 routes and extended in the fabric. In the case of VRF-lite IP fabric-based solution, the VRF will need to be extended or translated as per the tenant reach.

In the example shown above, irrespective of whether the internal switch is optimized and enabled or not, the GPUs in server in VRF1 will not be able to talk to the servers in VRF2 and vice versa. They are only able to talk to other servers in the same VRF across the cluster, hence enabling tenant-level isolation.

2.8.2 GPU level isolation



GPU-level node isolation for multitenancy, as shown above, is where a subset of a server (for example four out of eight GPUs) is allocated to a tenant. This is not limited to a single server, and the GPU allocation can be across the cluster. One mechanism to achieve this is to allocate a subset of GPUs to a tenant and provide an appropriate IP addressing schema to ensure tenant-based connectivity.

After connectivity has been established, the internal switching mechanism and/or the optimization must be disabled so that the GPUs cannot communicate internally. However, GPU vendors have prescribed that it is not the best option to disable the internal switching mechanisms because it may lead to unexpected behavior and internal optimization can still occur.

Hence, this is not the preferred option for GPU-level isolation.

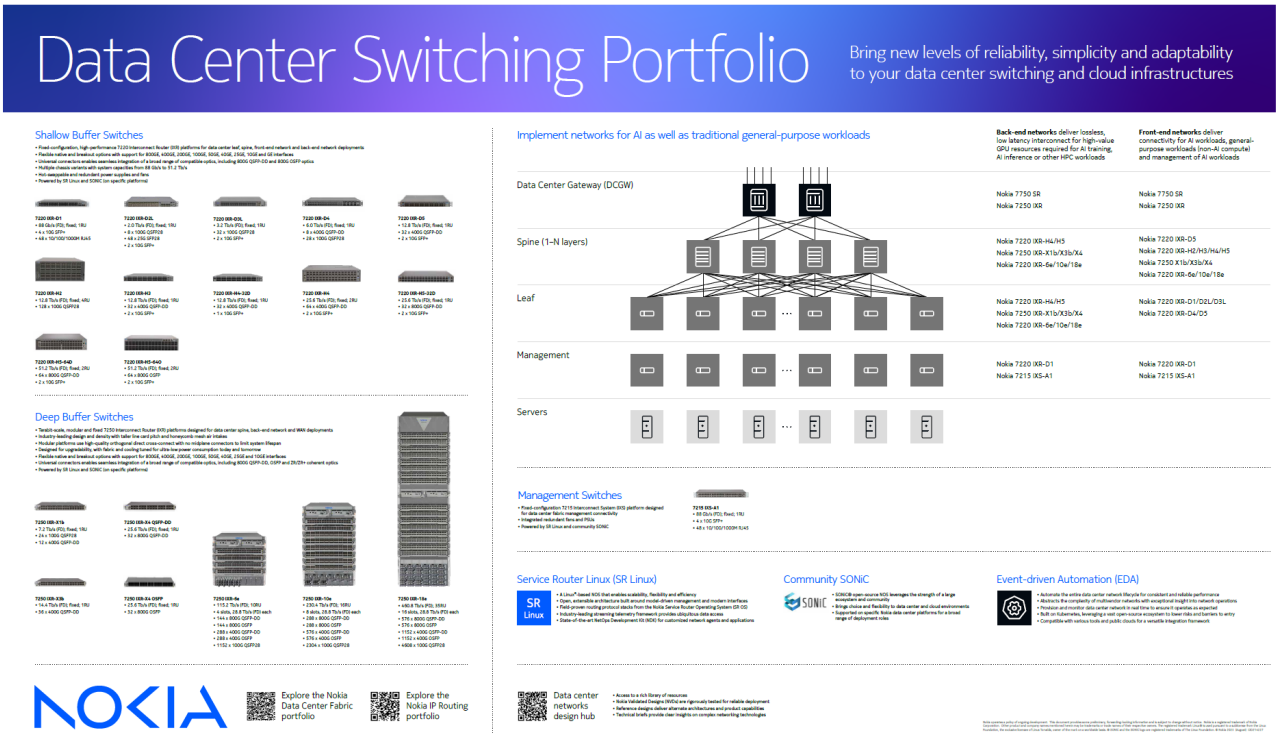
Another way to enable GPU-level isolation is to use the CCL framework over the networking layer. In this design, we can choose to provide a network-level segregation, but the actual isolation comes from the application layer.

For example: CUDA variables can define which GPU is visible to which other GPU in the framework, however, from the architecture diagram in section 2.7.1, we know that the GPU has access to more than one NIC that it can use to communicate to the fabric.

Hence, the GPU to NIC mapping also needs to be controlled manually; otherwise, there is a chance that even if the framework allocates a specific GPU to a particular tenant, it can leak data because of the incorrect NIC mapping.

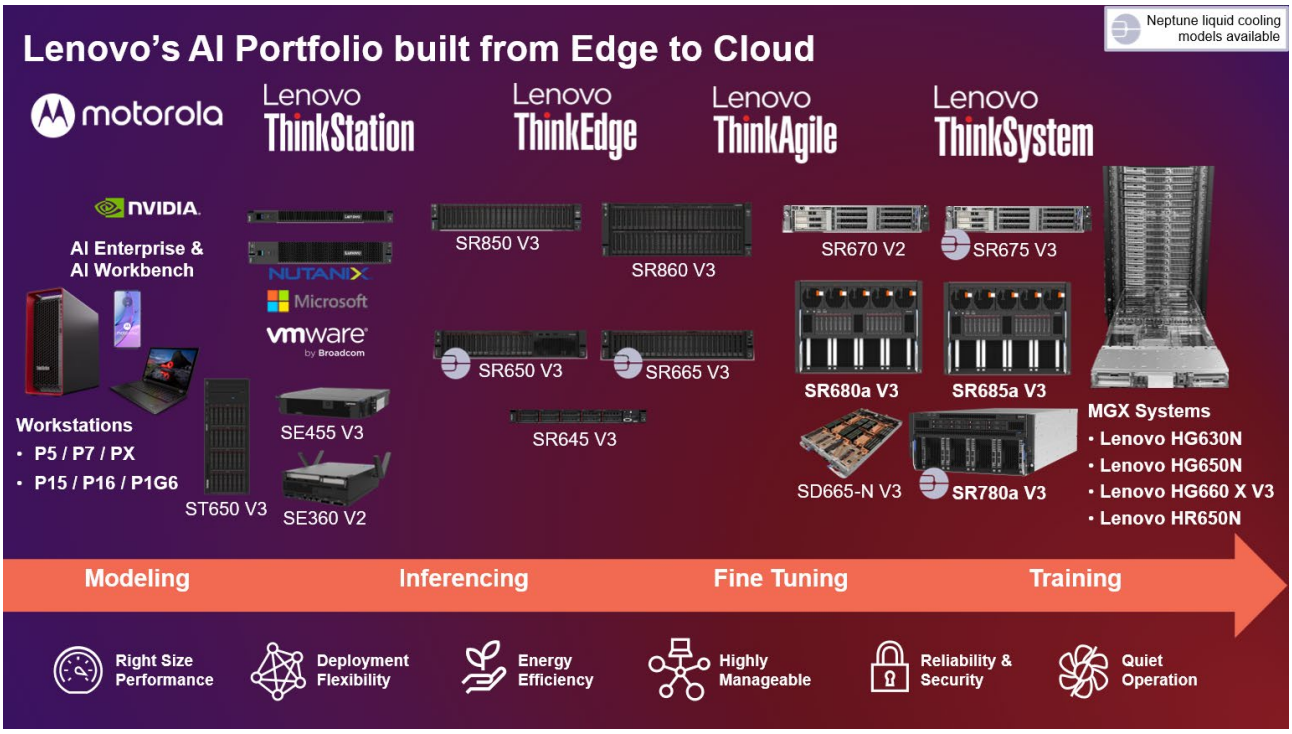
This is the preferred method for GPU isolation for multitenancy in an AI/ML cluster.

2.9 Nokia AI-DC portfolio



2.10 Lenovo AI-DC portfolio

The diagram below presents a comprehensive overview of Lenovo's AI portfolio, showcasing the full suite of solutions built to support artificial intelligence workloads from the edge to the cloud. The portfolio, including the ThinkStation, ThinkEdge, ThinkAgile, and ThinkSystem product lines are optimized for different stages of the AI lifecycle—from initial modeling and inferencing to fine-tuning and training.



3 Hardware and optics

3.1 Optics and hardware matrix

The table below shows the hardware that has been utilized in the validation process.

Connection				Connectivity Option			
Endpoints		Speed	Port Type	DAC	AOC	Transceiver	Cable
1	Broadcom 57608 P2200	400G	QSFP112		400G QSFP56-DD to 400G QSFP112 AOC 10M		
2	7220 IXR-H4	400G	QSFP56-DD				
1	Broadcom 57608 P2200	200G	QSFP56		400G QSFP-DD to 2 x 200G QSFP56 Breakout AOC 7M		
2	7220 IXR-D5	400G	QSFP56-DD				
1	7220 IXR-D5	400G	QSFP56-DD			3HE15211AA QSFP56-DD - 400G SR8 100m 0/70C (MPO16)	3HE15713AA, 16F MMF MPO-MPO Patch Cable, 3M 3HE15714AA, 16F MMF MPO-MPO Patch Cable, 10M
2	7220 IXR-D5	400G	QSFP56-DD			3HE15211AA QSFP56-DD - 400G SR8 100m 0/70C (MPO16)	
1	NVIDIA CX755106AS-HEAT	200G	QSFP112		400G QSFP-DD to 2 x 200G QSFP56 Breakout AOC 7M		
2	7220 IXR-D5	400G	QSFP56-DD				
1	Broadcom 57414	25G	SFP28	100G QSFP28 to 4x25G SFP28 Breakout DAC Cable			
2	7220 IXR-D3L	100G	QSFP28				
1	NVIDIA MCX631432A N-ADAB	25G	SFP28	100G QSFP28 to 4x25G SFP28 Breakout DAC Cable			
2	7220 IXR-D3L	100G	QSFP28				

3.2 Cooling and power

The following table lists Lenovo ThinkSystem computes and their nominal maximum power draw at full load and heat generation.

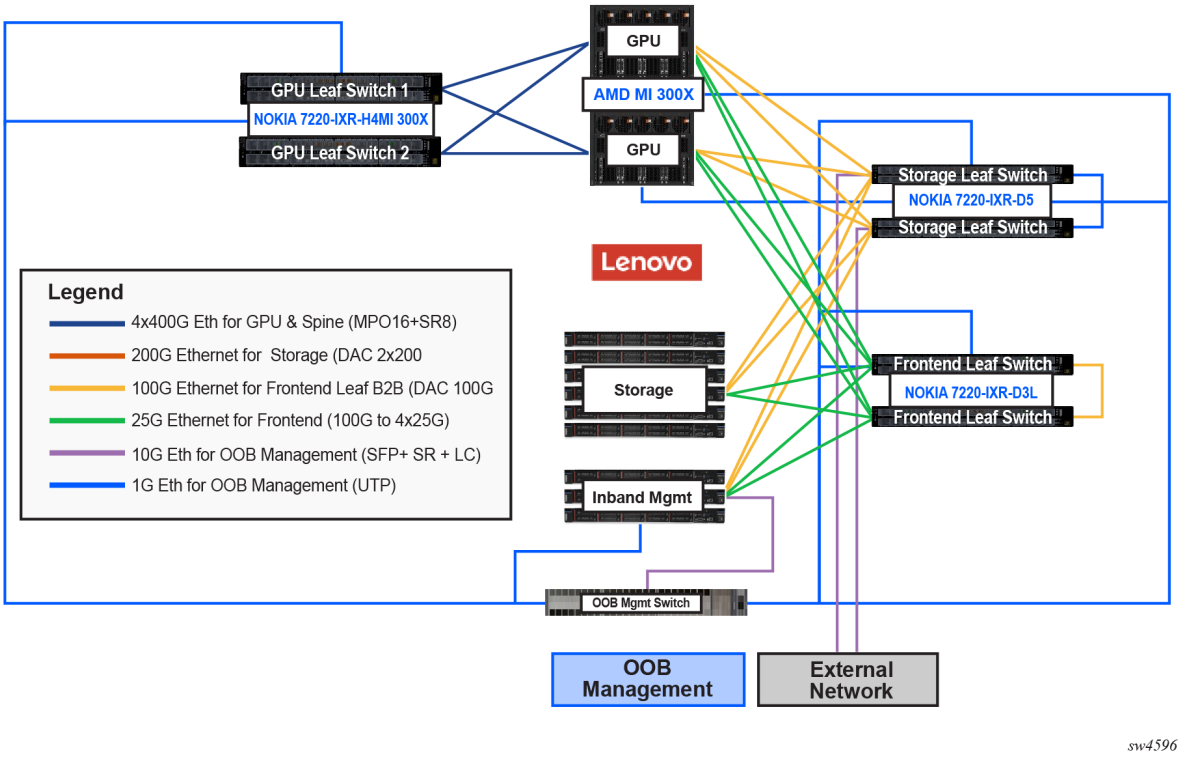
Compute	Nominal maximum power draw (W)	Heat generation (BTU/hour)
ThinkSystem SR685a V3 AI Compute with 8 x AMD MI300X GPUs	9424.6	32156.6
ThinkSystem SR630 V4 Control Node with Intel Xeon 6530P 32C 2.3GHZ CPU	795	2712
ThinkSystem SR635 V3 Control Node with AMD EPYC 9335 32C 3.0 GHZ CPU	521.8	1780

4 Network orchestration

4.1 Orchestration overview

The design being outlined in this document is a Lenovo-Nokia scalable unit that supports up to 128 GPUs (16 servers), which is suitable to deploy large language models such as BERT, DLRM, and even GPT3 and economically viable to be an inference or a hybrid cluster as well. The design aims to be frugal by utilizing the minimum hardware that is needed to orchestrate such a cluster by using network design and optimization principles to deliver a high-end network.

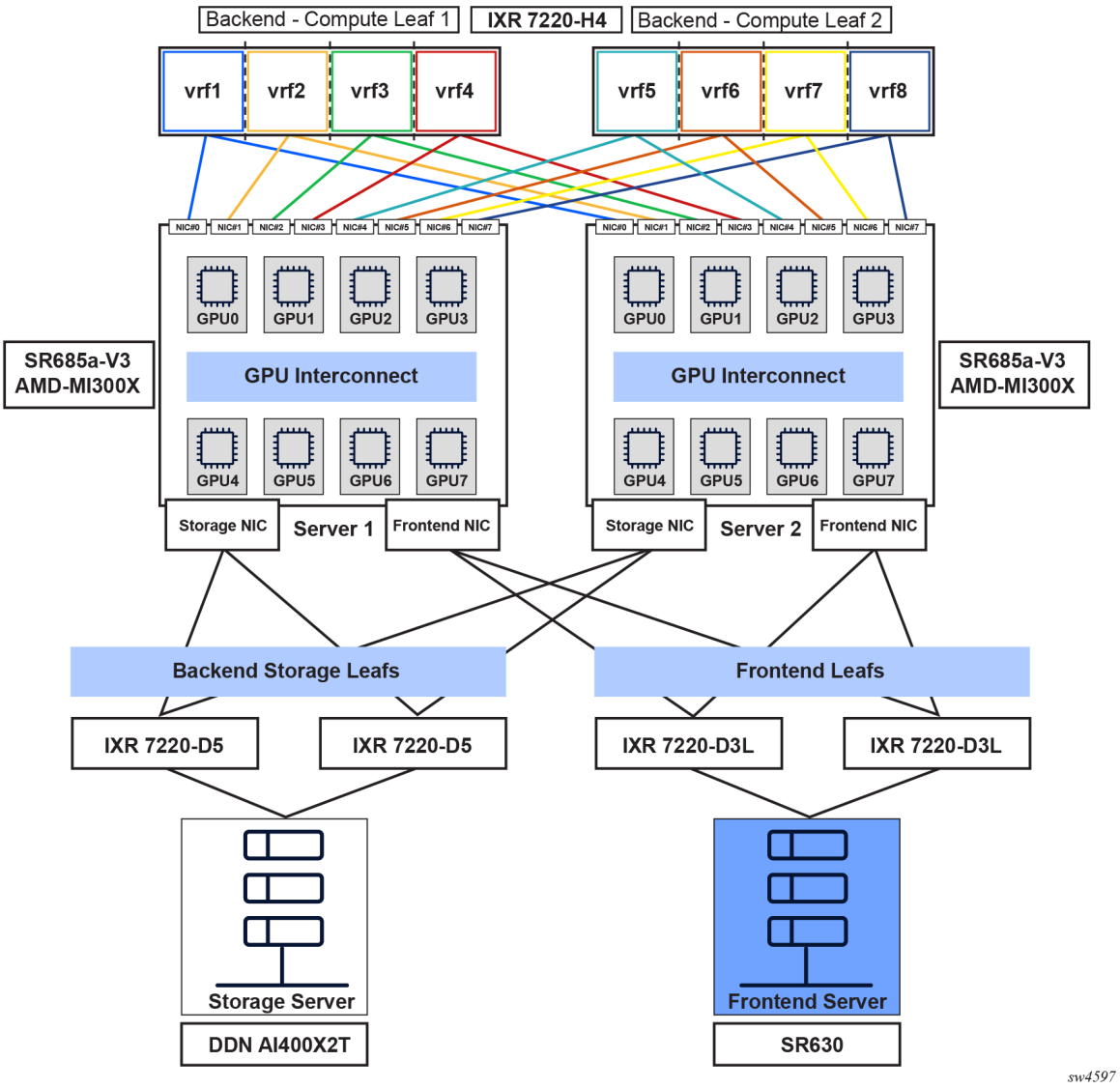
4.1.1 Lenovo-Nokia AI scalable unit



The figure above shows the Lenovo-Nokia AI validated scalable unit, which has been used as the test bed for this validated design.

Note: Though specific hardware versions and variations have been used for this validation process, the product positioning for various roles can vary based on budget and optimizations.

4.1.2 Reference architecture



The diagram above shows the reference architecture of the Lenovo-Nokia validated design pod, which demonstrates a reduced form of a typical rail-only design achieved via IP VRFs called node isolation groups.

The validated design shown above has three main segments under test:

- the backend compute segment, which comprises the 7220 IXR-H4 switches connecting the Lenovo SR685a-V3 GPU servers that host the AMD Instinct MI300X GPUs
- the backend storage fabric, which comprises the 7220 IXR-D5 switches connecting the SR630 servers with DDN storage nodes
- the frontend 7220 IXR-D3L switches, which connect the SR630 server

Design considerations

- The backend compute stripe has been designed to attain rail optimization by using logical division of the compute fabric switches via IP VRFs.
- In the figure, we can see that there are eight VRFs present on the two switches, which represent the eight rails of a rail-only or rail-optimized design.
- The GPU NICs have been mapped to the appropriate VRFs as per the principles of rail optimization; for example: NIC1 of server 1 and server 2 maps to VRF1 shown in blue. Hence, if GPU0, which is mapped to NIC1 in server 1, wants to talk to GPU0 on server 2, it must traverse via IP VRF 1 on switch 1.
- The fabric IP addressing has been orchestrated by the EDA AI fabric app in such a way that each compute fabric IP address has the stripe ID, the leaf ID and the port number encoded into the IPv6 address which will be a /96 subnet. This allows the GPU to be identified based on the IP address, which is a very useful feature in an AI cluster environment.

AI features such as congestion management via ECN/PFC can also be configured via the EDA AI fabric app.

4.2 EDA architecture

The network infrastructure is managed and orchestrated by Nokia Event-Driven Automation (EDA). Nokia's EDA platform is a cloud-native platform deployed on top of Kubernetes, leveraging the Kubernetes-provided declarative API, tooling, and the ecosystem around it. EDA can be deployed as a single or multimode cluster.

The various components of the EDA/K8s tech stack are shown below, instantiated as Kubernetes pods.

```
admin@server:~$ kubectl get pods -A
NAMESPACE        NAME                                                READY   STATUS
RESTARTS        AGE
cert-manager     cert-manager-777c6f8ff4-bscqv                       1/1     Running     0
6d11h
cert-manager     cert-manager-cainjector-6558fc6578-6clk2            1/1     Running     0
6d11h
cert-manager     cert-manager-webhook-6964489477-khzbx               1/1     Running     0
6d11h
eda-system       cert-manager-csi-driver-4b7mx                       3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-8279p                       3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-ct2wk                       3/3     Running     0
6d11h
```

```
eda-system       cert-manager-csi-driver-cttlg                  3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-mznkr                  3/3     Running     0
6d11h
eda-system       cert-manager-csi-driver-tsvnq                  3/3     Running     0
6d11h
eda-system       eda-api-bdd576c85-l2252                        1/1     Running     0
6d11h
eda-system       eda-appstore-74cdc5c964-csf8j                  1/1     Running     0
6d11h
eda-system       eda-asvr-9fd4b99fb-lxvwh                       1/1     Running     0
6d11h
eda-system       eda-bsvr-b7b84b8f5-5689z                       1/1     Running     0
6d11h
eda-system       eda-ce-58c5cbf87d-cwmmq                        1/1     Running     0
5d23h
eda-system       eda-cert-checker-bf74ccbd4-m48bn               1/1     Running     0
6d11h
eda-system       eda-fe-b8b877cf6-ntc47                         1/1     Running     0
6d11h
eda-system       eda-fluentbit-4mz4x                            1/1     Running     0
6d11h
eda-system       eda-fluentbit-5mszs                            1/1     Running     0
6d11h
eda-system       eda-fluentbit-6ksw7                            1/1     Running     0
6d11h
eda-system       eda-fluentbit-95z2p                            1/1     Running     0
6d11h
eda-system       eda-fluentbit-j6swr                            1/1     Running     0
6d11h
eda-system       eda-fluentbit-q4smx                            1/1     Running     0
6d11h
eda-system       eda-fluentd-7cd48db9c5-rxs9d                   1/1     Running     0
6d11h
eda-system       eda-git-5db9dfc7bc-pdb6m                       1/1     Running     0
6d11h
eda-system       eda-git-replica-f69b9c9f4-jznkq                1/1     Running     0
6d11h
eda-system       eda-keycloak-bcfbfd9d6-bn7mn                   1/1     Running     0
5d23h
eda-system       eda-metrics-server-8d8b8595f-r6hnd             1/1     Running     0
6d11h
eda-system       eda-npp-0                                      1/1     Running     0
5d22h
eda-system       eda-npp-1                                      1/1     Running     0
5d20h
eda-system       eda-postgres-6b59c9985-ppbkf                   1/1     Running     0
6d11h
eda-system       eda-px-55c6dd5588-cfvw8                        1/1     Running     0
43h
eda-system       eda-sa-54df7ffbc5-xjbht                        1/1     Running     0
6d11h
eda-system       eda-sc-7644c9cc4c-kj7xm                        1/1     Running     0
6d11h
```

```
eda-system       eda-se-1                                          1/1     Running     0
6d11h
eda-system       eda-toolbox-696f47749d-z457h                      1/1     Running     0
6d11h
eda-system       trust-manager-849b644bdf-88pck                    1/1     Running     0
6d11h
kube-system      coredns-578d4f8ffc-jdqq7                          1/1     Running     0
6d11h
kube-system      coredns-578d4f8ffc-n4c5n                          1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-11                             1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-12                             1/1     Running     0
6d11h
kube-system      kube-apiserver-eda-13                             1/1     Running     0
6d11h
kube-system      kube-controller-manager-eda-11                    1/1     Running     1
(6d11h ago)   6d11h
kube-system      kube-controller-manager-eda-12                    1/1     Running     0
6d11h
kube-system      kube-controller-manager-eda-13                    1/1     Running     0
6d11h
kube-system      kube-flannel-6b9vq                                1/1     Running     0
6d11h
kube-system      kube-flannel-7xd69                                1/1     Running     0
6d11h
kube-system      kube-flannel-qcd26                                1/1     Running     0
6d11h
kube-system      kube-flannel-qzkm4                                1/1     Running     0
6d11h
kube-system      kube-flannel-ss7qn                                1/1     Running     0
6d11h
kube-system      kube-flannel-xcwct                                1/1     Running     0
6d11h
kube-system      kube-proxy-4mf92                                  1/1     Running     0
6d11h
kube-system      kube-proxy-jjkm5                                  1/1     Running     0
6d11h
kube-system      kube-proxy-p9mfg                                  1/1     Running     0
6d11h
kube-system      kube-proxy-phtxl                                  1/1     Running     0
6d11h
kube-system      kube-proxy-q566g                                  1/1     Running     0
6d11h
kube-system      kube-proxy-vpxzc                                  1/1     Running     0
6d11h
kube-system      kube-scheduler-eda-11                             1/1     Running     1
(6d11h ago)   6d11h
kube-system      kube-scheduler-eda-12                             1/1     Running     0
6d11h
kube-system      kube-scheduler-eda-13                             1/1     Running     0
6d11h
metallb-system   controller-5cbffbc46b-xwzz8                       1/1     Running     0
6d11h
```

```
metallb-system   speaker-6lpjs                                        1/1     Running     0
6d11h
metallb-system   speaker-cbclx                                        1/1     Running     0
6d11h
metallb-system   speaker-mwvfw                                        1/1     Running     0
6d11h
metallb-system   speaker-pwhqn                                        1/1     Running     0
6d11h
metallb-system   speaker-pwrrt                                        1/1     Running     0
6d11h
metallb-system   speaker-rc9p2                                        1/1     Running     0
6d11h
rook-ceph        csi-cephfsplugin-d8vdb                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-f25fm                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-p5pxz                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-p92lt                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-provisioner-5b8485cdb4-4v8wp        5/5     Running     2
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-provisioner-5b8485cdb4-zzrtr        5/5     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-t4bpg                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        csi-cephfsplugin-w7mzc                               2/2     Running     1
(6d11h ago)   6d11h
rook-ceph        rook-ceph-mds-ceph-filesystem-a-649b4d448-525fp      1/1     Running     0
6d11h
rook-ceph        rook-ceph-mds-ceph-filesystem-b-6fd79c7d47-g42d2     1/1     Running     0
6d11h
rook-ceph        rook-ceph-mgr-a-6cb6b8d4fb-v2r7r                     2/2     Running     0
6d11h
rook-ceph        rook-ceph-mgr-b-7896d689d8-9wspm                     2/2     Running     0
6d11h
rook-ceph        rook-ceph-mon-a-74688f9f97-n7vqz                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-mon-b-bd754b95f-2cs5z                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-mon-c-6d9b6cbdf6-ggjjp                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-operator-6c99bbf54d-g7xpw                  1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-0-bb8968b97-r49z6                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-1-554cb45854-9z8g6                     1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-2-5cf9747c5-95rvq                      1/1     Running     0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-11-kbzsz                   0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-12-j6ftd                   0/1     Completed   0
6d11h
```

```
rook-ceph        rook-ceph-osd-prepare-eda-13-jbrsb                0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-14-77km8                0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-15-56d7r                0/1     Completed   0
6d11h
rook-ceph        rook-ceph-osd-prepare-eda-16-r6sxr                0/1     Completed   0
6d11h
rook-ceph        rook-ceph-tools-54bcc747b4-xk8rx                  1/1     Running     0
6d11h
```

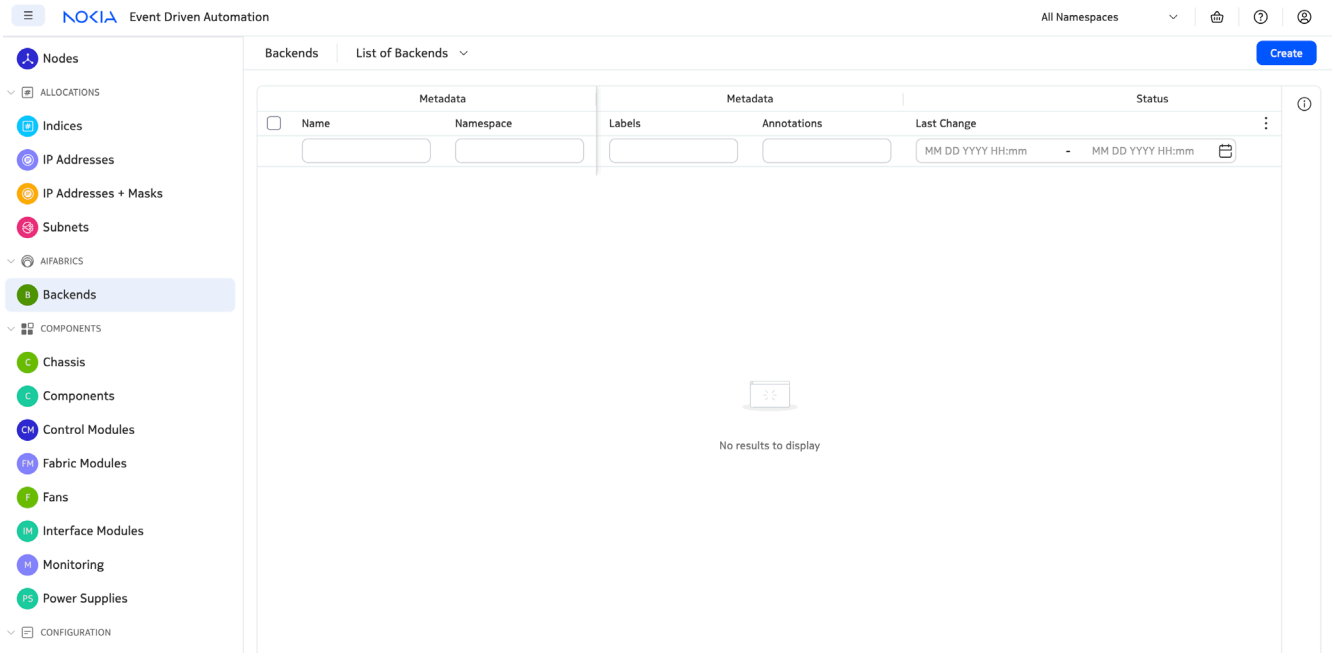
Some commonly used pods and their functionalities are:

- **eda-asvr** - the artifact server stores common artifacts used in EDA functionality. Examples include SR Linux image, SRL MD5 hash, yang path.zip, and so forth. The availability of an artifact can be verified with “kubectl get artifacts -A”.
- **eda-bsvr** – the bootstrap server is responsible for all onboarding of nodes (virtual or hardware). Onboarding involves gNMI discovery, gNMI management, and instantiation of NPP pods for node lifecycle management.
- **eda-ce** – the configuration engine keeps track of all the dependencies among the application resources and runs the application intents when needed.
- **eda-npp** – the eda-npp pod is responsible for schema validation of the generated configuration. Additionally, it is responsible for all communications to the devices for both setting configuration and retrieving state.
- **eda-api** – the eda-api pod is the REST API server, which is accessible to end users and is consumed by the GUI.
- **eda-cx** – the sandbox controller spins up simulated nodes for building digital twins of the fabric (the example above has the mode set to physical hardware only, hence the EDA CX functionality has been disabled).
- **eda-toolbox** – the toolbox provides tools such as edactl for insight into EDA transactions and an EDA topology generator that can generate a topology from a YAML file.

4.3 EDA AI backend app

EDA provides a dedicated AI backend app for deploying AI backend network infrastructure, considering the unique needs of such a fabric. The app deploys the overall fabric as stripes (a set of leafs) with all stripes connected via a common set of spines. Each stripe is given a user-defined stripe ID, which is used in building the IPv6 address assigned to GPU-facing interfaces on the leafs. This app also provides a single view to control the necessary class of service parameters such as ECN and PFC thresholds and multi-tenancy requirements for AI backend fabrics.

The app is found under Main → AIFABRICS → Backends.



An example manifest for this app is provided in Section 4.4.

4.4 EDA manifests

This section describes various manifest files that can be used to deploy an EDA-orchestrated AI backend fabric in accordance with the prescriptive validated design described in this document. Note that these manifest files must reference the correct EDA namespace.

4.4.1 EDA artifacts for SR Linux version 24.10.3

Kubernetes artifacts are created for a target SR Linux version and are used in the custom resource for the EDA node profile. This includes the creation of manifest files for the .bin image, the MD5 hash file, and the YAML zip file, samples of which are shown below.

```
# artifacts for SRL v24.10.3
apiVersion: artifacts.eda.nokia.com/v1
kind: Artifact
metadata:
  name: srlinux-24.10.3-201-bin
  namespace: eda-system
spec:
  repo: srlimages
  filePath: srlinux.bin
  remoteFileUrl:
```

```
  fileUrl: http://172.28.1.2/srlLinuxImages/srlinux-24.10.3-201.bin
---
apiVersion: artifacts.eda.nokia.com/v1
kind: Artifact
metadata:
 name: srlinux-24.10.3-201-md5
 namespace: eda-system
spec:
 repo: srlimages
 filePath: srlinux.md5
 remoteFileUrl:
  fileUrl: http://172.28.1.2/srlLinuxImages/srlinux-24.10.3-201.bin.md5
---
```

4.4.2 Subnet allocation for management of SR Linux fabric nodes

A manifest file is created to instantiate an IPv4/IPv6 subnet pool for the management of SR Linux fabric nodes. By referencing this pool in the node profile manifest, an IP address is given to the management interface of the SR Linux fabric nodes during ZTP.

```
# management pool for ZTP
apiVersion: core.eda.nokia.com/v1
kind: IPInSubnetAllocationPool
metadata:
 name: ipv4-mgmt-pool
 namespace: eda
spec:
 segments:
  - subnet: 172.32.16.0/24
    allocations:
     - name: gateway$$
       value: 172.32.16.10/24
```

4.4.3 EDA node profile for node onboarding

An EDA node profile facilitates the onboarding of fabric nodes, including the username and password for authentication into the node, a DHCP scope for assignment, and image version check (the node profile image is the expected target image).

```
# nodeprofile for v24.10.3
apiVersion: core.eda.nokia.com/v1
kind: NodeProfile
metadata:
 name: real-srlinux-24.10.3
 namespace: eda
spec:
```

```
 dhcp:
  managementPoolv4: ipv4-mgmt-pool
 images:
  - image: eda-system/srlimages/srlinux-24.10.3-201-bin/srlinux.bin
    imageMd5: eda-system/srlimages/srlinux-24.10.3-201-md5/srlinux.md5
 nodeUser: admin
 onboardingUsername: 'admin'
 onboardingPassword: 'NokiaSrl1!'
 operatingSystem: srl
 port: 57400
 version: 24.10.3
 versionMatch: v24\.10\.3.*
 versionPath: .system.information.version
 yang: https://eda-asvr.eda-system.svc/eda-system/schemaprofiles/srlinux-ghcr-24.10.3/srlinux-24.10.3.zip
```

4.4.4 Onboarding nodes in EDA with using a TopoNode Custom Resource

SR Linux nodes can be onboarded into EDA using the TopoNode custom resource. This includes the creation of labels as metadata that will be attached to the node (these labels are used as selectors when deploying the fabric), a node profile name, the platform, and serial number of the node.

```
# toponodes
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
 name: h4-spine1
 namespace: eda
 labels:
  eda.nokia.com/role: leaf
  eda.nokia.com/security-profile: managed
spec:
 platform: 7220 IXR-H4
 version: 24.10.3
 operatingSystem: srl
 nodeProfile: real-srlinux-24.10.3
 operatingSystem: srl
 platform: "7220 IXR-H4"
 version: 24.10.3
 serialNumber: <omitted>
 npp:
  mode: normal
---
```

4.4.5 Building an ASN pool for the IP fabric

The following manifest file demonstrates how ASN pools can be built to be used during fabric deployment.

```
# ASN pool for collapsed spines
apiVersion: core.eda.nokia.com/v1
kind: IndexAllocationPool
metadata:
 name: asn-pool
 namespace: eda
spec:
 segments:
  - start: 65501
    size: 30
```

4.4.6 System0 IP pool allocation

The following manifest file demonstrates how an IP allocation pool is created for assignment as a system0 IP address for the IP fabric nodes.

```
# pool for system0 address allocation
apiVersion: core.eda.nokia.com/v1
kind: IPAllocationPool
metadata:
 name: h4-system-ipv4
 namespace: eda
spec:
 segments:
  - subnet: 10.1.1.0/24
```

4.4.7 Interface creation

The following snippet of a manifest file demonstrates how interfaces are instantiated in EDA per the onboarded SR Linux node.

```
# interfaces
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
 labels:
  eda.nokia.com/tenant: gpucluster1
 name: h4-spine1-e-1-1
 namespace: eda
spec:
 enabled: true
 encapType: dot1q
```

```
 ethernet:
  speed: 400G
 lldp: true
 members:
  - enabled: true
    interface: ethernet-1-1
    node: h4-spine1
 mtu: 9500
 type: interface
---
```

4.4.8 IP fabric creation with EDA AI backend app

The following manifest demonstrates how a backend fabric is created using the EDA AI app with multitenancy requirements.

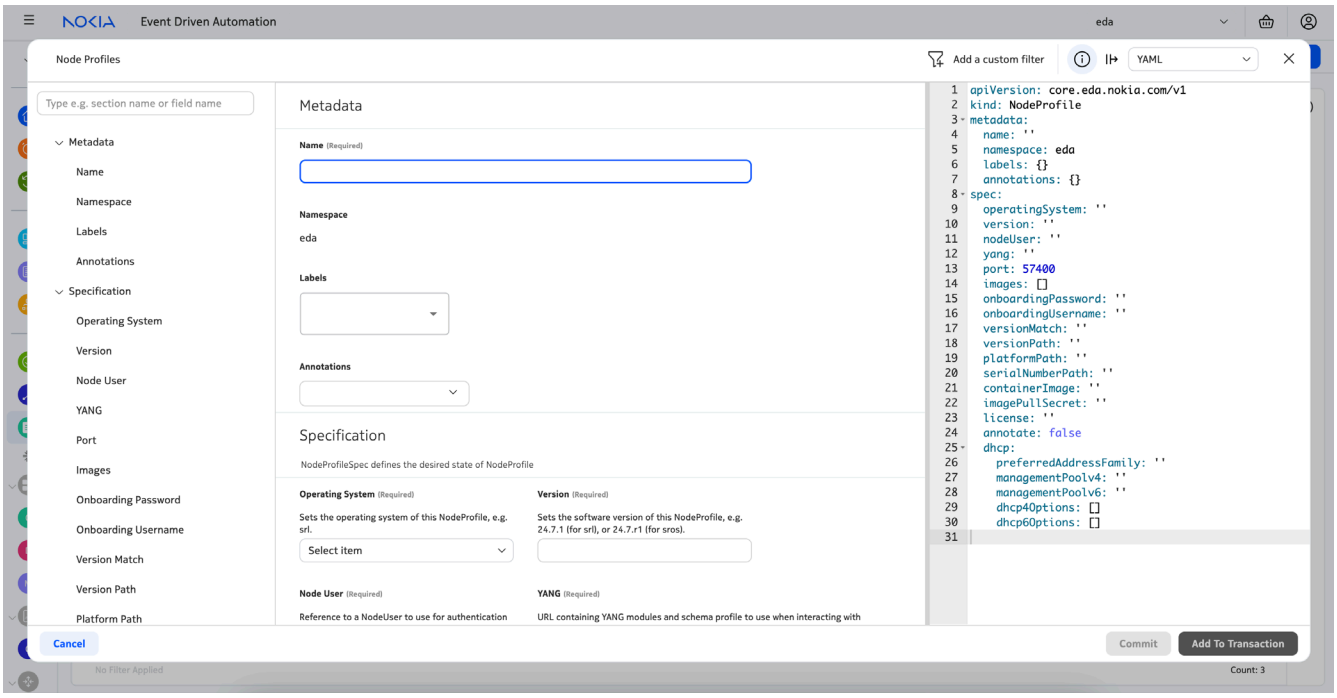
```
apiVersion: aifabrics.eda.nokia.com/v1alpha1
kind: Backend
metadata:
 name: backend1
 namespace: eda
spec:
 asnPool: asn-pool
 gpuIsolationGroups:
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster1
   name: islgrp1
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster2
   name: islgrp2
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster3
   name: islgrp3
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster4
   name: islgrp4
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster5
   name: islgrp5
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster6
   name: islgrp6
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster7
   name: islgrp7
```

```
 - interfaceSelector:
   - eda.nokia.com/tenant=gpucluster8
   name: islgrp8
 rocev2QoS:
   ecnMaxDropProbabilityPercent: 100
   ecnSlopeMaxThresholdPercent: 80
   ecnSlopeMinThresholdPercent: 5
   pfcDeadlockDetectionTimer: 750
   pfcDeadlockRecoveryTimer: 750
   queueMaximumBurstSize: 52110640
 stripes:
 - asnPool: asn-pool
   gpuVlan: 1000
   name: strp1
   nodeSelector:
   - eda.nokia.com/role=leaf
   stripeID: 100
   systemPoolIPV4: h4-system-ipv4
 systemPoolIPV4: h4-system-ipv4
---
```

4.5 EDA workflows via UI

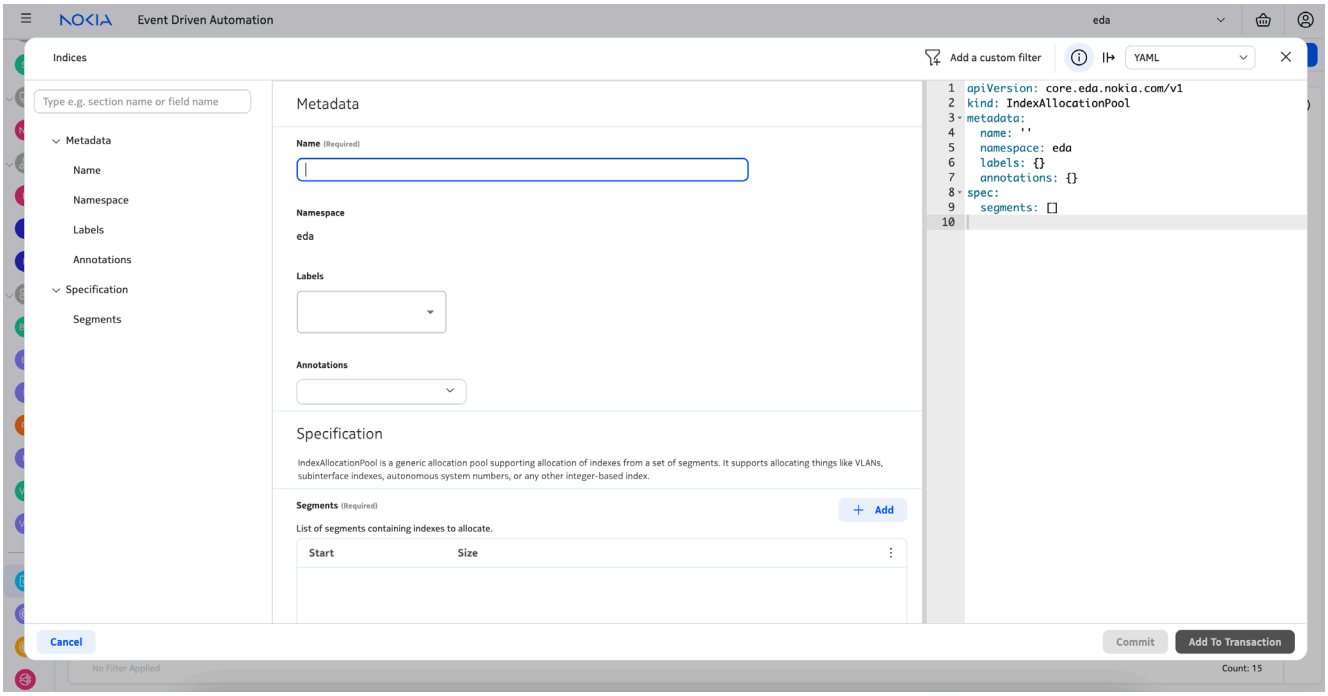
4.5.1 Node profiles for node onboarding

Node profiles are specified during node onboarding and are used to determine the IP pool from which to assign an IP address to the node, what the gNMI discovery port is, and the username and password credentials to log into the device. Node profiles can be created by navigating to **Main** → **Node Profiles**.



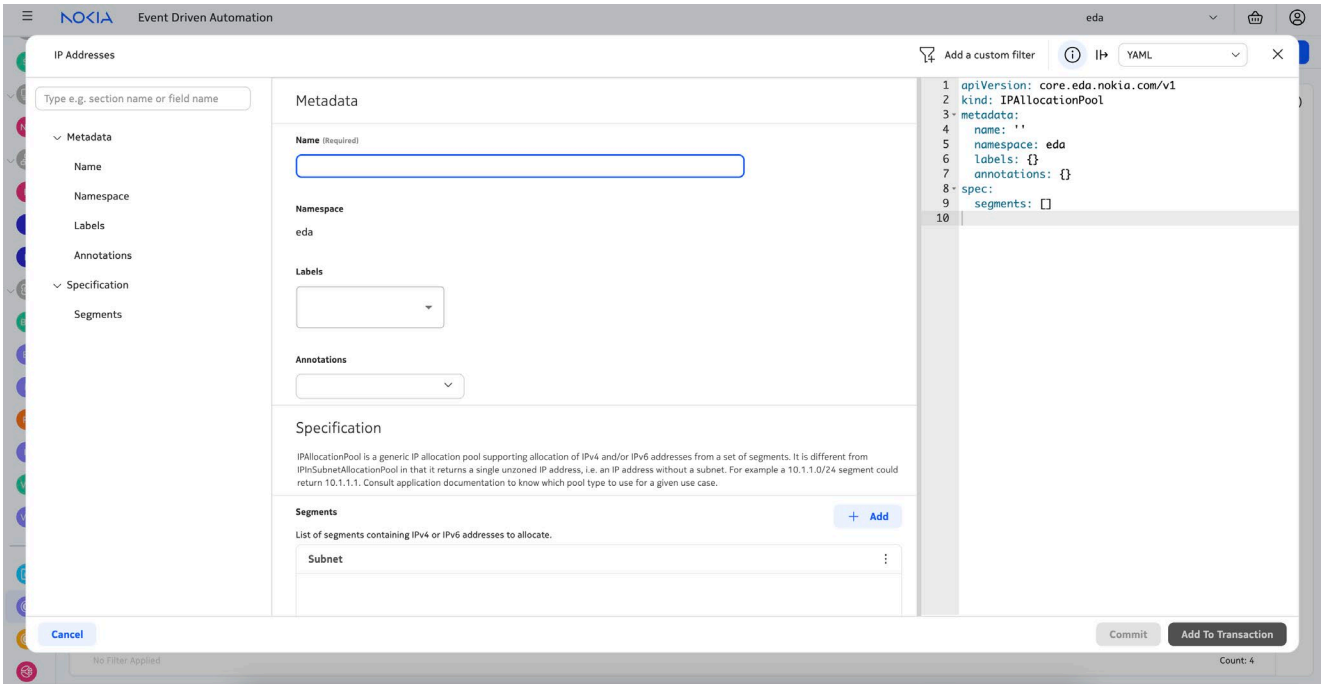
4.5.2 ASN pools

The ASN pools are created as indices pools, which can then be assigned to fabric nodes during fabric creation. These indices pools can be viewed and created by navigating to **Main** → **Indices**.



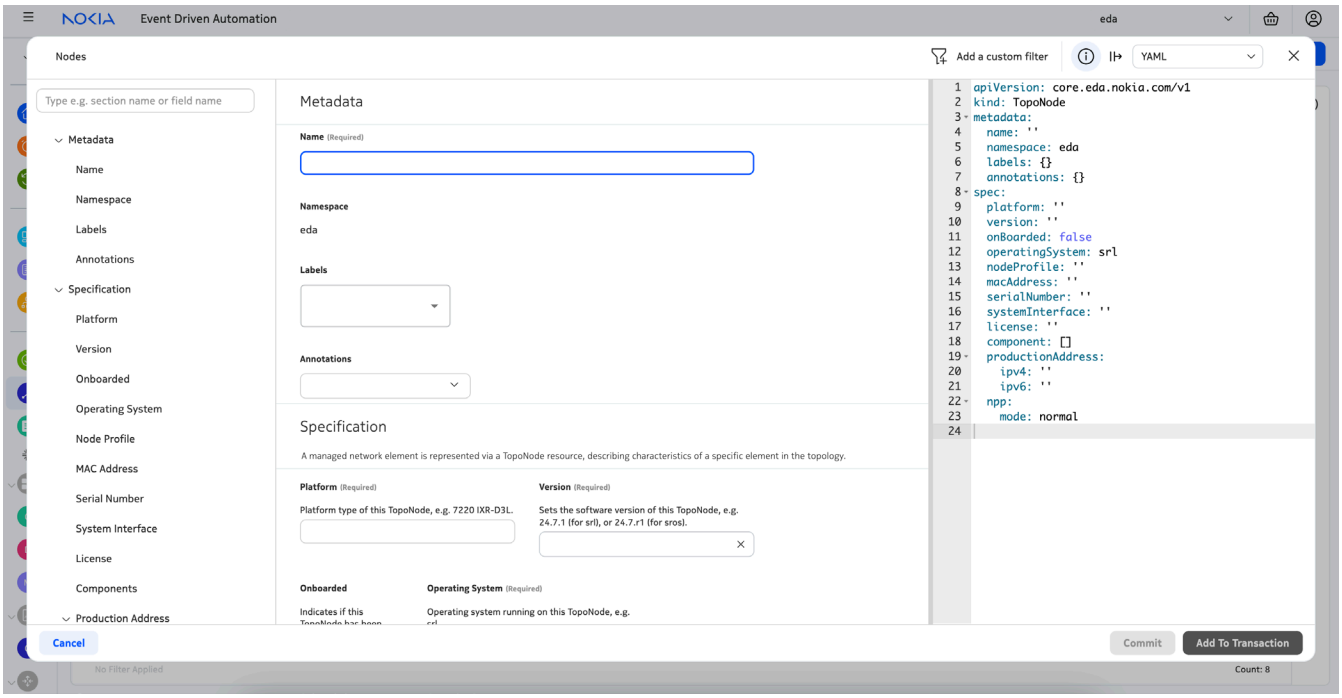
4.5.3 IP pool creation allocation

IP pools can be created for multiple reasons, for example, a subnet allocation or an exact IP address allocation. In the case of this NVD, an IP pool of type *IP Addresses* is created to assign a unique IPv4 address from an IPv4 subnet for the system0 interface of fabric nodes in the fabric. This type of IP pool can be created by navigating to **Main** → **IP Addresses**.



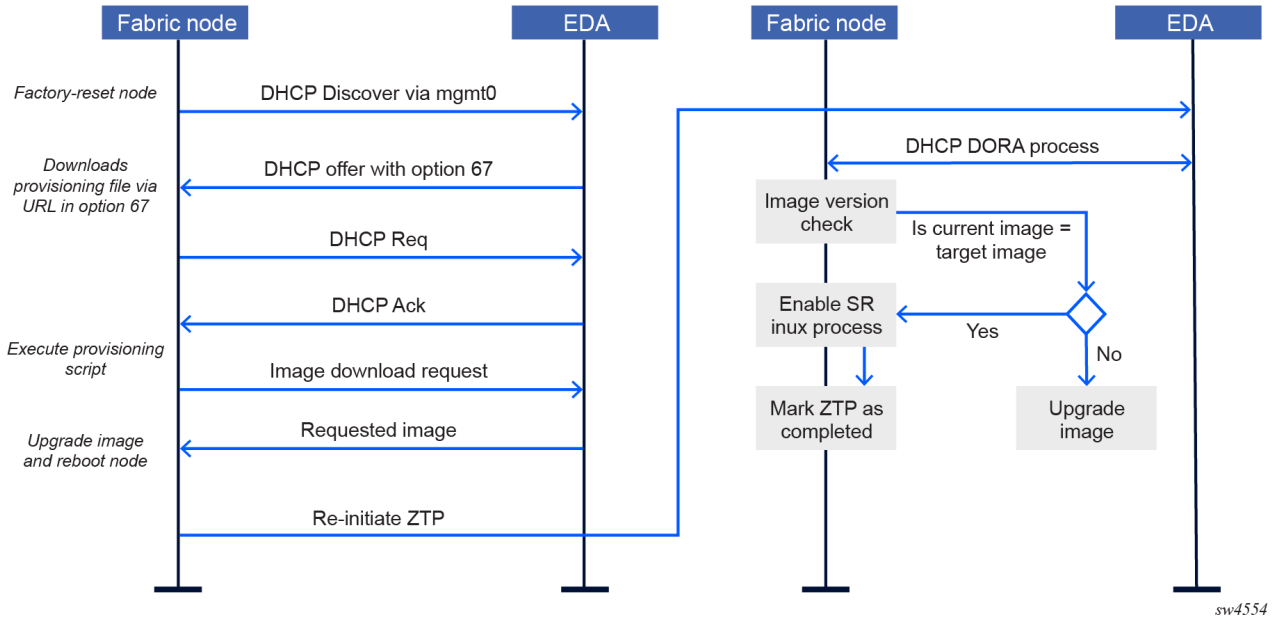
4.5.4 Onboarding nodes

Nodes can be created and viewed by navigating to **Main** → **Nodes**. These are nodes onboarded into the fabric and represented in the topology view.



4.6 Node onboarding via zero-touch provisioning (ZTP)

EDA can onboard fabric nodes via ZTP. EDA, as the ZTP server, can fully automate the end-to-end deployment of Nokia SR Linux nodes. Nodes that are in a factory default state only need to be plugged into the out-of-band (OOB) infrastructure and EDA can onboard the devices, along with pushing expected configuration (based on user intent) to them.



4.7 SRL feature configuration

Section 4.3 (EDA AI backend app for compute network infrastructure) describes the general design and usage of the EDA AI backend app, which translates the intent provided in this app into SR Linux configuration, pushed to onboarded SR Linux nodes (based on appropriate label selection). This section describes the configuration pushed to these nodes when using this app.

4.7.1 GPU-facing IPv6 interfaces on leafs

GPU-facing interfaces are assigned an IPv6 address from the unique local address (ULA) range. These addresses are encoded to include the stripe ID, the leaf number, the leaf port number for easy identification of the GPU and where it is connected in the fabric.

```
A:h4-spine1# info interface ethernet-1/2
    interface ethernet-1/2 {
        description GPUSVR1-GPU5-NIC5
        admin-state enable
        mtu 9500
        vlan-tagging true
        ethernet {
            port-speed 400G
            flow-control {
                receive false
            }
        }
        subinterface 1000 {
            type routed
            description "Backend: backend1 Stripe: strp1"
            admin-state enable
            ip-mtu 9394
            ipv6 {
                admin-state enable
                address fd00:100:1:1:0:2:0:1/96 {
                    primary
                }
                router-advertisement {
                    router-role {
                        admin-state enable
                        current-hop-limit 64
                        managed-configuration-flag false
                        other-configuration-flag false
                        max-advertisement-interval 600
                        min-advertisement-interval 200
                        reachable-time 0
                        retransmit-time 0
```

```
                        router-lifetime 1800
                    }
                }
            }
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1000
                    }
                }
            }
        }
    }
```

4.7.2 Quality of service

When building network infrastructure for AI workloads (especially for training), it is imperative to build and deploy a lossless fabric. This is typically enforced using a combination of Priority Flow Control (PFC) and Explicit Congestion Notification (ECN) quality of service features.

In SR Linux, this involves:

- Classification of ingress traffic into a forwarding class (based on DSCP value in the incoming packet, as an example).
- Creating queues and mapping appropriate forwarding classes to their respective queues.
- Creating a PFC profile that sets different PFC parameters (such as enabling PFC, defining the PFC priority value).
- Creating a scheduler policy.
- Creating a buffer management profile for ingress (PFC) and for egress (ECN) that determines the PFC and ECN thresholds (and the drop probability) and allocates buffer sizes.

```
A:h4-spine1# info qos
    qos {
        explicit-congestion-notification {
        }
        queues {
            queue unicast-0 {
                queue-index 0
            }
            queue unicast-3 {
                queue-index 3
```

```
            }
            queue unicast-6 {
                queue-index 6
            }
            pfc-queue pfc-3 {
                queue-index 3
            }
        }
        forwarding-classes {
            forwarding-class fc0 {
                output {
                    unicast-queue unicast-0
                }
            }
            forwarding-class fc3 {
                output {
                    unicast-queue unicast-3
                }
            }
            forwarding-class fc6 {
                output {
                    unicast-queue unicast-6
                }
            }
        }
        pfc-mapping-profile dcqcn1 {
            received-traffic {
                unicast-mapping {
                    pfc-queue pfc-3 {
                        forwarding-class [
                            fc3
                        ]
                        pfc-pause-frame-priority [
                            3
                        ]
                    }
                }
            }
            received-pfc-pause-frames {
                deadlock {
                    enable true
                    detection-timer 750
                    recovery-timer 750
                }
                queue unicast-3 {
                    enable-pfc true
                    pfc-pause-frame-priority [
                        3
```

```
                ]
            }
        }
    }
    classifiers {
        dscp-policy ingress-backend-backend1 {
            dscp 0 {
                forwarding-class fc0
                drop-probability low
            }
            dscp 26 {
                forwarding-class fc3
                drop-probability low
            }
            dscp 48 {
                forwarding-class fc6
                drop-probability low
            }
        }
    }
    scheduler-policies {
        scheduler-policy egress-backend-backend1 {
            scheduler 0 {
                priority strict
                input unicast-6 {
                    queue-name unicast-6
                    peak-rate-percent 10
                }
            }
            scheduler 1 {
                input unicast-3 {
                    queue-name unicast-3
                    peak-rate-percent 100
                    weight 50
                }
            }
        }
    }
    interfaces {
        interface ethernet-1/2 {
            interface-ref {
                interface ethernet-1/2
            }
            pfc {
                pfc-mapping-profile dcqcn1
                pfc-enable true
            }
            input {
```

```
                pfc-buffer-allocation-profile ingress-backend-backend1
            }
            output {
                buffer-allocation-profile egress-backend-backend1
                queues {
                    queue unicast-3 {
                        queue-management-profile egress-backend-backend1-2
                    }
                }
                scheduler {
                    scheduler-policy egress-backend-backend1
                }
            }
        }
        interface ethernet-1/2.1000 {
            interface-ref {
                interface ethernet-1/2
                subinterface 1000
            }
            input {
                classifiers {
                    dscp-policy ingress-backend-backend1
                }
            }
        }
        interface ethernet-1/3 {
            interface-ref {
                interface ethernet-1/3
            }
            pfc {
                pfc-mapping-profile dcqcn1
                pfc-enable true
            }
            input {
                pfc-buffer-allocation-profile ingress-backend-backend1
            }
            output {
                buffer-allocation-profile egress-backend-backend1
                queues {
                    queue unicast-3 {
                        queue-management-profile egress-backend-backend1-2
                    }
                }
                scheduler {
                    scheduler-policy egress-backend-backend1
                }
            }
        }
```

```
            interface ethernet-1/3.1000 {
                interface-ref {
                    interface ethernet-1/3
                    subinterface 1000
                }
                input {
                    classifiers {
                        dscp-policy ingress-backend-backend1
                    }
                }
            }
            interface ethernet-1/4 {
                interface-ref {
                    interface ethernet-1/4
                }
                pfc {
                    pfc-mapping-profile dcqcn1
                    pfc-enable true
                }
                input {
                    pfc-buffer-allocation-profile ingress-backend-backend1
                }
                output {
                    buffer-allocation-profile egress-backend-backend1
                    queues {
                        queue unicast-3 {
                            queue-management-profile egress-backend-backend1-2
                        }
                    }
                    scheduler {
                        scheduler-policy egress-backend-backend1
                    }
                }
            }
            interface ethernet-1/4.1000 {
                interface-ref {
                    interface ethernet-1/4
                    subinterface 1000
                }
                input {
                    classifiers {
                        dscp-policy ingress-backend-backend1
                    }
                }
            }
        }
        buffer-management {
            queue-management-profile egress-backend-backend1-2 {
```

```
                weight-factor 0
                wred {
                    wred-slope all drop-probability all enable-ecn true {
                        min-threshold-percent 5
                        max-threshold-percent 80
                        slope-enabled false
                        max-drop-probability-percent 100
                    }
                }
            }
            buffer-allocation-profile egress-backend-backend1 {
                queues {
                    queue unicast-0 {
                        maximum-burst-size 52110640
                    }
                    queue unicast-3 {
                        maximum-burst-size 52110640
                    }
                    queue unicast-6 {
                        maximum-burst-size 52110640
                    }
                }
            }
            buffer-allocation-profile ingress-backend-backend1 {
                queues {
                    pfc-queue pfc-3 {
                        maximum-burst-size 52110640
                    }
                }
            }
        }
        linecard 1 {
            forwarding-complex 0 {
                input {
                    pfc-buffer-reservation 2
                }
            }
        }
    }
```

4.7.3 Segmentation via IP VRFs

In SR Linux, segmentation is via IP VRFs, configured as network instances. This provides Layer 3 segmentation of GPU resources for multitenancy requirements. The example below demonstrates the configuration of a GPU-facing interface and how it is mapped to an IP VRF.

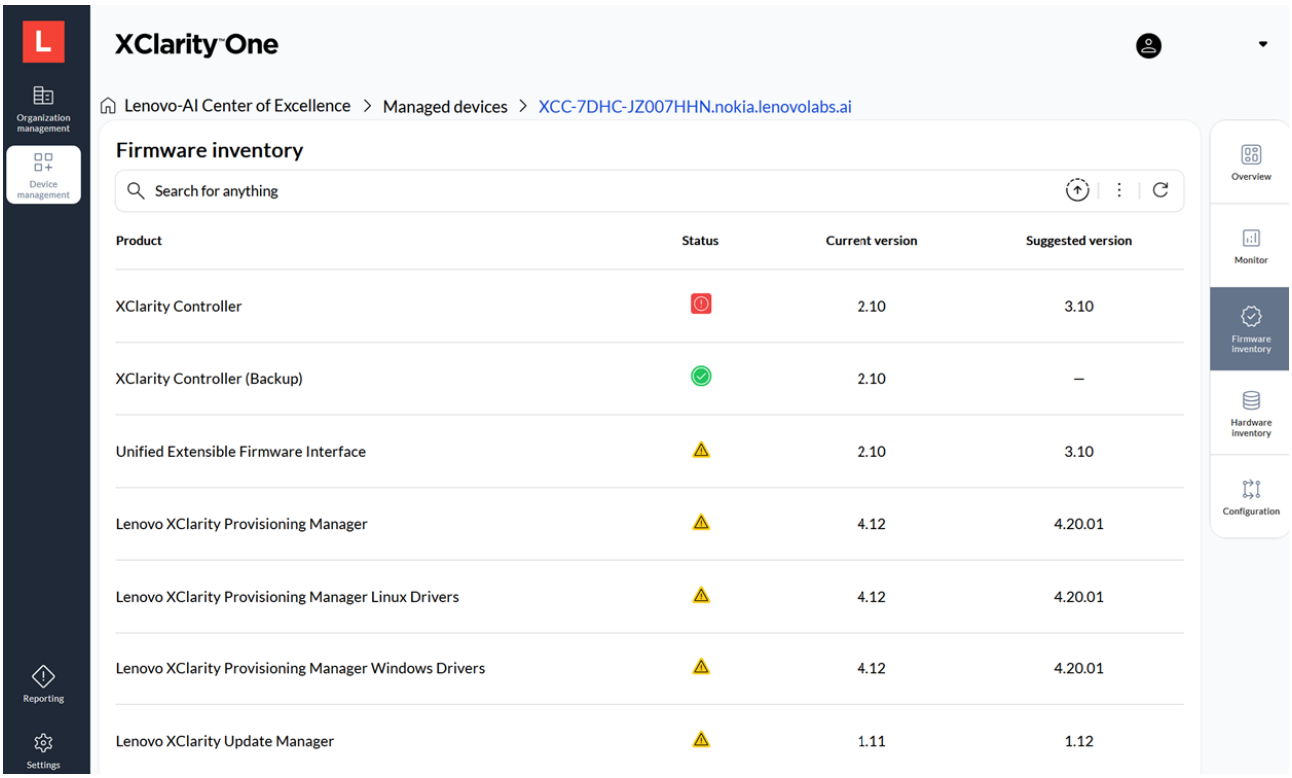
```
A:h4-spine1# info interface ethernet-1/1
    interface ethernet-1/1 {
        description GPUSVR1-GPU1-NIC1
        admin-state enable
        mtu 9500
        vlan-tagging true
        ethernet {
            port-speed 400G
            flow-control {
                receive false
            }
        }
        subinterface 1000 {
            type routed
            description "Backend: backend1 Stripe: strp1"
            admin-state enable
            ip-mtu 9394
            ipv6 {
                admin-state enable
                address fd00:100:1:1:0:1:0:1/96 {
                    primary
                }
                router-advertisement {
                    router-role {
                        admin-state enable
                        current-hop-limit 64
                        managed-configuration-flag false
                        other-configuration-flag false
                        max-advertisement-interval 600
                        min-advertisement-interval 200
                        reachable-time 0
                        retransmit-time 0
                        router-lifetime 1800
                    }
                }
            }
            vlan {
                encap {
                    single-tagged {
                        vlan-id 1000
                    }
                }
            }
        }
    }

A:h4-spine1# info network-instance islgrp1
    network-instance islgrp1 {
```

```
        type ip-vrf
        admin-state enable
        description "Backend: backend1"
        inter-instance-policies {
            apply-policy {
                import-policy import-routeleak-islgrp1-backend1
                export-policy export-routeleak-islgrp1-backend1
            }
        }
        interface ethernet-1/1.1000 {
        }
        interface ethernet-1/5.1000 {
        }
    }
```

5 Backend GPU, storage, and frontend management

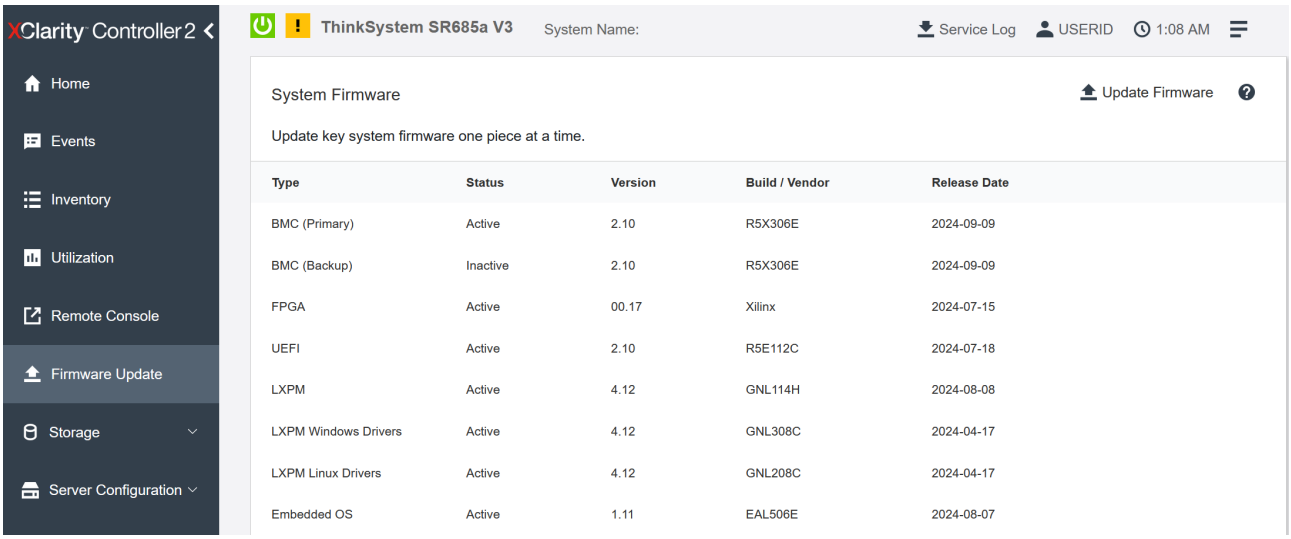
5.1 Server configuration – SR685a V3 with 8 AMD MI300x GPUs

The SR685a V3 servers should be brought up to the latest supported firmware using the XClarity One Administrator to fetch and apply the updates. Firmware updates can be found at: https://pubs.lenovo.com/sr685a-v3/



For more information, see https://pubs.lenovo.com/lxc1/devices-fw-updates

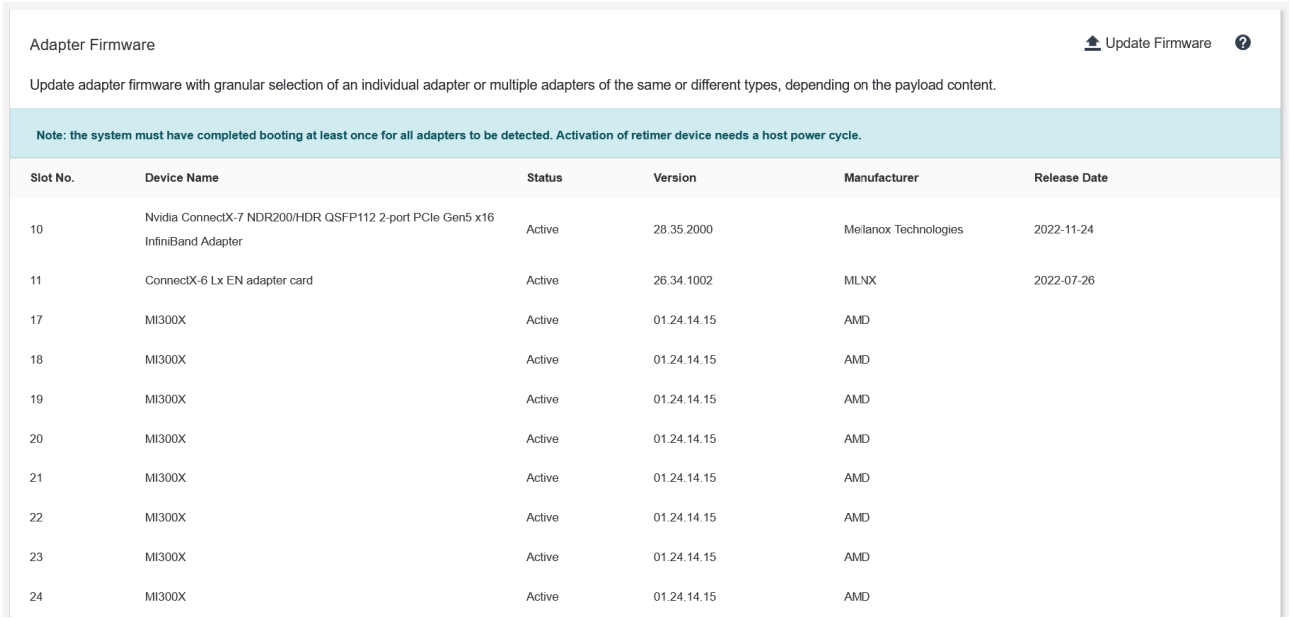
Alternatively, the firmware can be applied manually on each server using its XClarity Controller (XCC) Firmware Update Tab.



To download firmware packages and more information, see the Lenovo Support site here: https://datacentersupport.lenovo.com/us/en/products/servers/thinksystem/sr685av3/7dhc/7dhcctolww/jz007hhn\

AMD Graphics Processing Unit (GPU) Adapter Firmware

Verify the version of the AMD GPU firmware on the system and update it if necessary, using the XCC Firmware Update menu tab.



Adapter Firmware

Update Firmware

Update adapter firmware with granular selection of an individual adapter or multiple adapters of the same or different types, depending on the payload content.

Note: the system must have completed booting at least once for all adapters to be detected. Activation of retimer device needs a host power cycle.

Slot No.	Device Name	Status	Version	Manufacturer	Release Date
10	Nvidia ConnectX-7 NDR200/HDR QSFP112 2-port PCIe Gen5 x16 InfiniBand Adapter	Active	28.35.2000	Mellanox Technologies	2022-11-24
11	ConnectX-6 Lx EN adapter card	Active	26.34.1002	MLNX	2022-07-26
17	MI300X	Active	01.24.14.15	AMD	
18	MI300X	Active	01.24.14.15	AMD	
19	MI300X	Active	01.24.14.15	AMD	
20	MI300X	Active	01.24.14.15	AMD	
21	MI300X	Active	01.24.14.15	AMD	
22	MI300X	Active	01.24.14.15	AMD	
23	MI300X	Active	01.24.14.15	AMD	
24	MI300X	Active	01.24.14.15	AMD	

The AMD GPU firmware should be downloaded from the Lenovo support site here:

https://datacentersupport.lenovo.com/us/en/products/servers/thinksystem/sr685av3/7dhc/7dhcctolww/jz007hhn/downloads/ds569400-amd-graphics-processing-unit-gpu-adapter-firmware

BIOS Settings

The SR685a V3 AMD EPYC™ based system should be tuned using the instructions provided by AMD here: https://instinct.docs.amd.com/projects/amdgpu-docs/en/latest/system-optimization/mi300x.html

Use the Lenovo UEFI management menu, called LXPM, to make the changes in BIOS configuration.

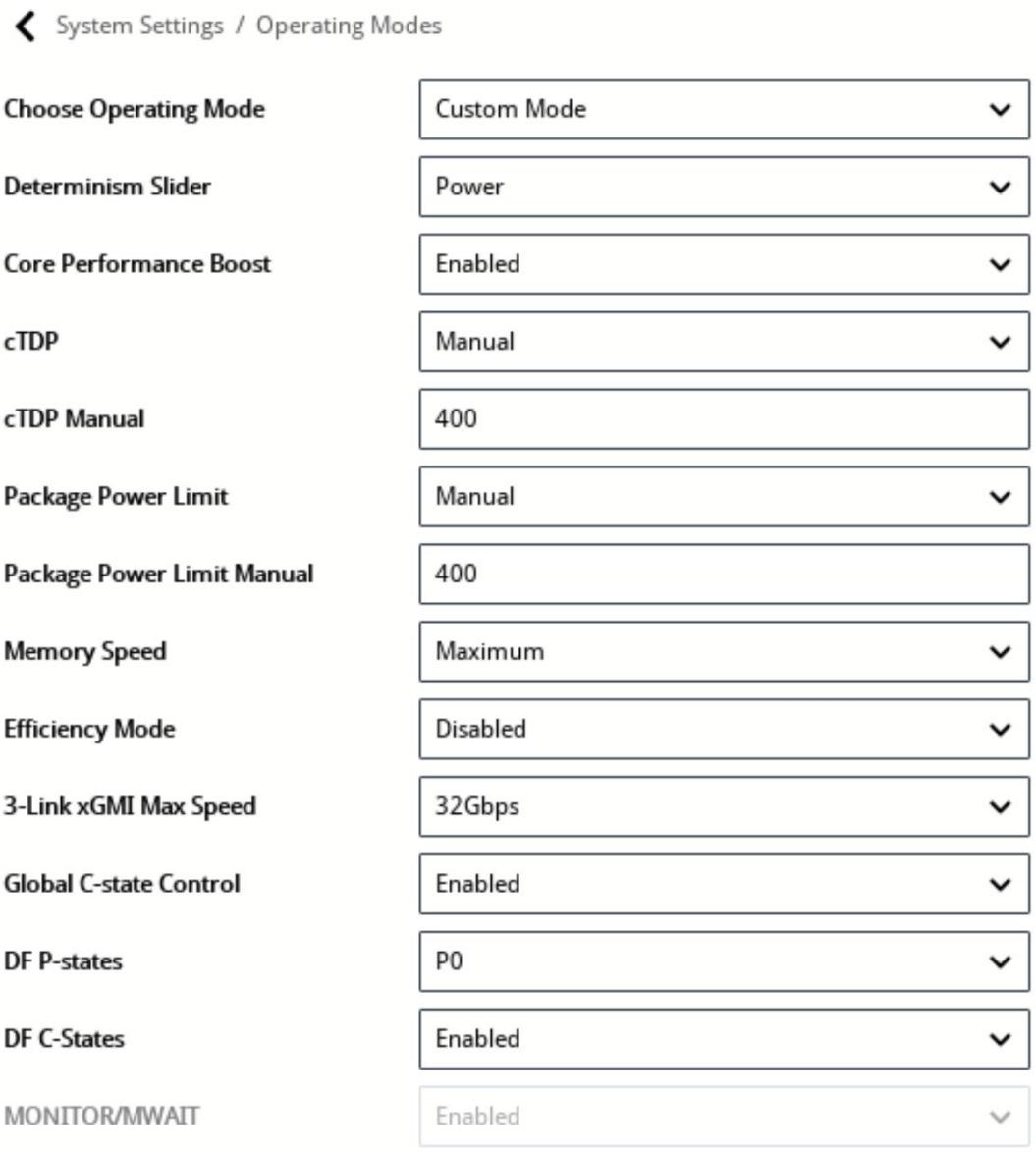
Instructions

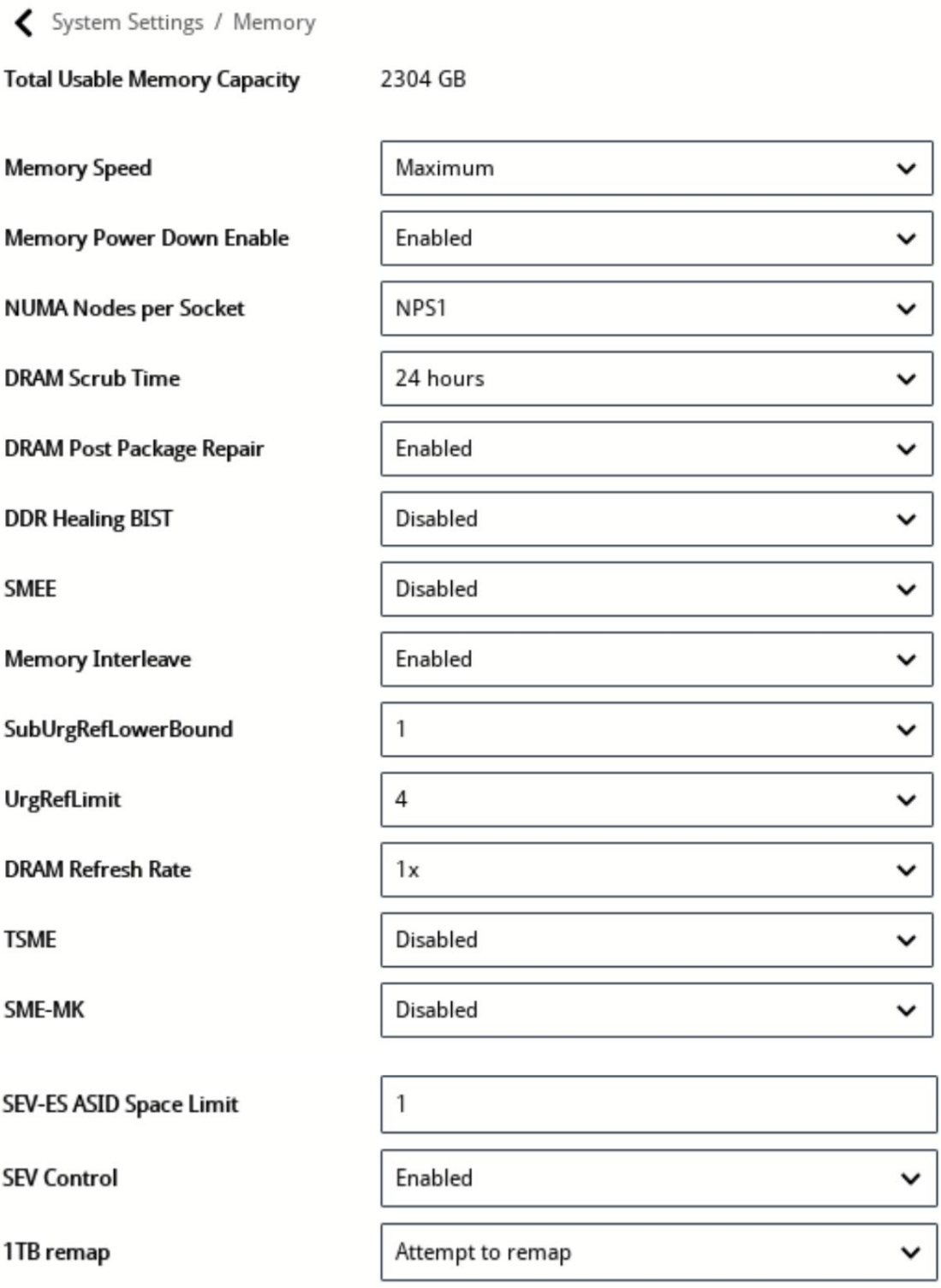
Perform the following steps to tune the system.

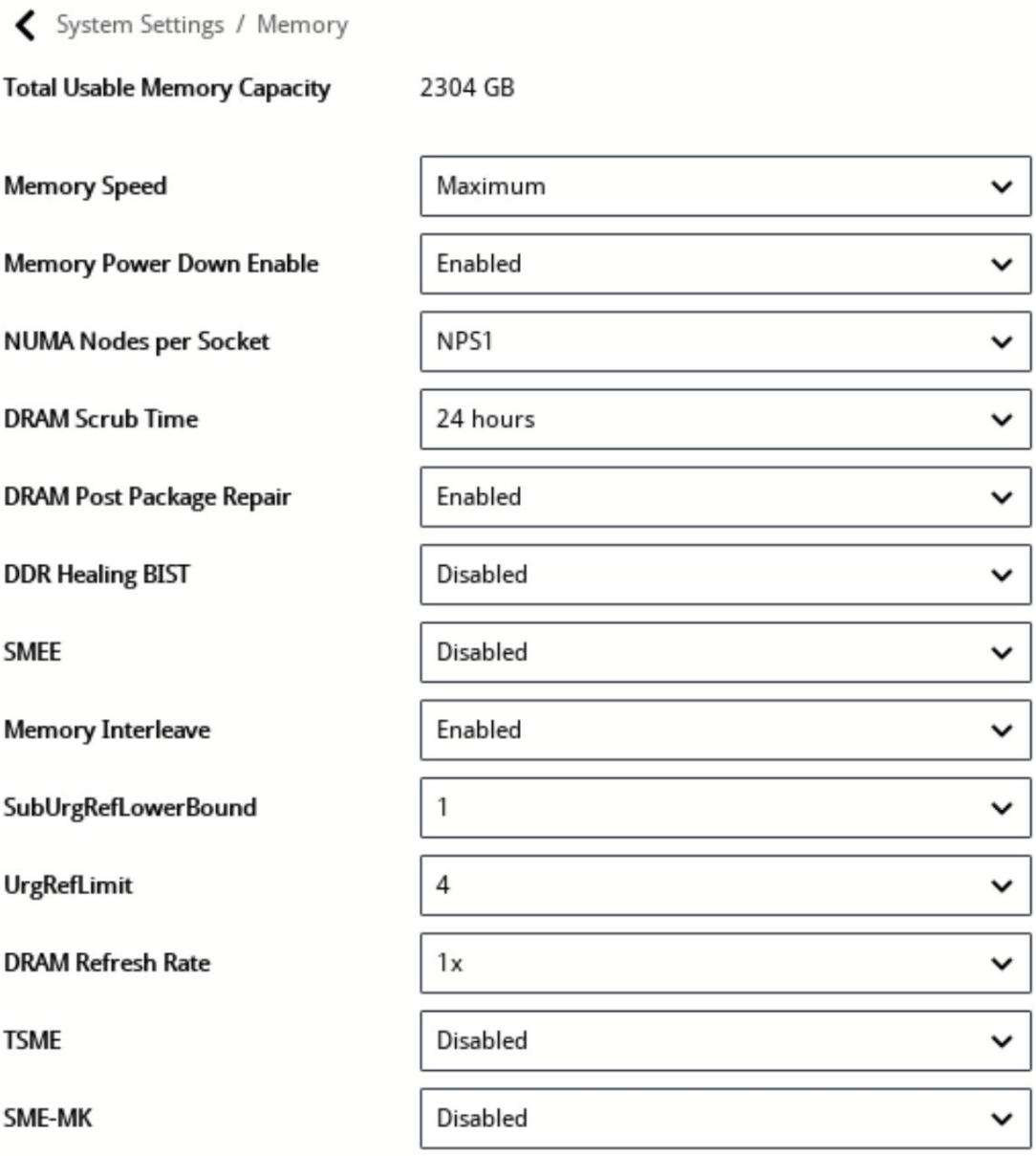
Note: Tuning the following settings can be a time-consuming and error-prone task. For large deployments, it is recommended to use a programmatic method such as the Lenovo Redfish API or Lenovo OneCli. See more information at: https://lenovopress.lenovo.com/lp2210-tuning-uefi-settings-5th-gen-amd-epyc-processor-servers

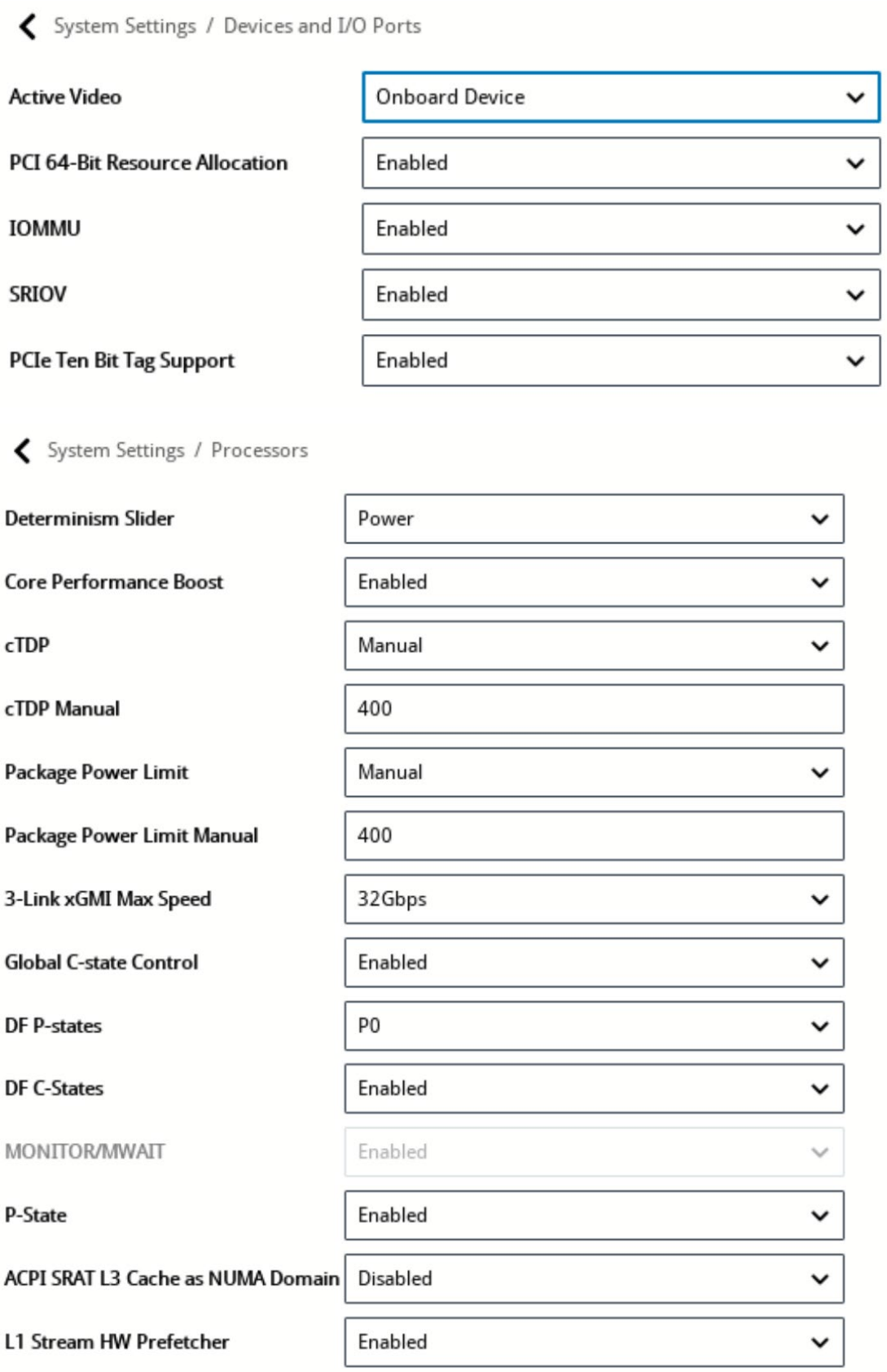
1. Load the server into System Setup using F1 at boot or setting the boot option via XCC.

2. Navigate to UEFI Settings Setup/System Settings/Operating Modes and load the MAX_PERFORMANCE profile.
3. Save the UEFI settings.
4. Navigate to **UEFI Settings Setup/System Settings/Operating Modes**, load the **CUSTOM** profile, and edit the following menu items.









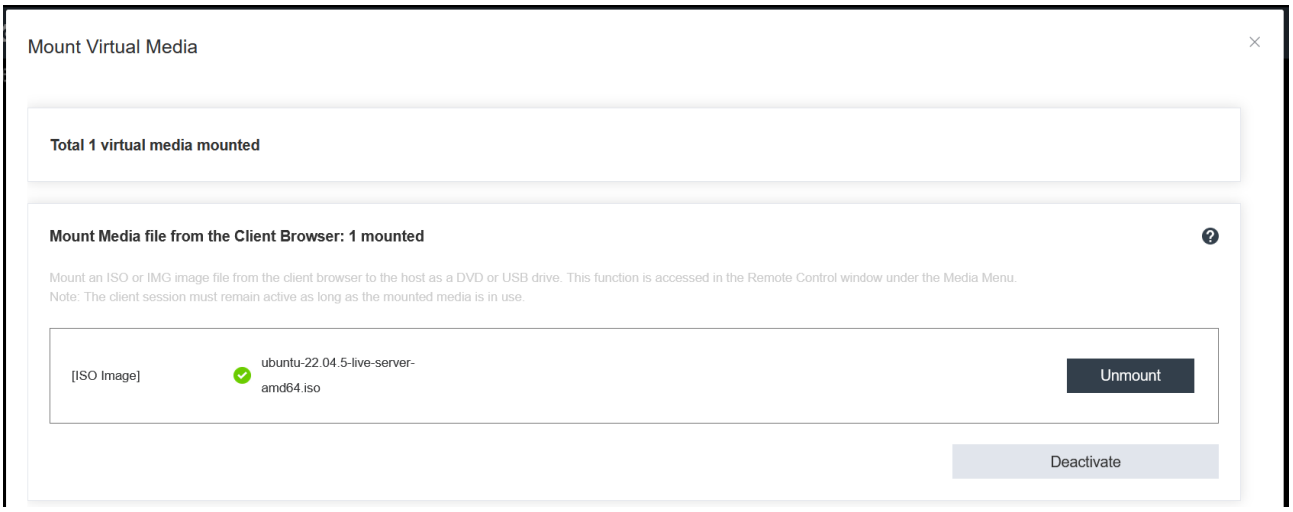
L2 Stream HW Prefetcher	Enabled
L1 Stride Prefetcher	Enabled
L1 Region Prefetcher	Enabled
L2 Up/Down Prefetcher	Enabled
SMT Mode	Disabled
CPPC	Enabled
BoostFmax	Auto
SVM Mode	Enabled
xGMI Maximum Link Width	x16
APIC Mode	Auto
SEV-SNP Support	Disabled
HSMP Support	Auto
Enhanced REP MOVSB/STOSB	Enabled
Fast Short REP MOVSB	Enabled

SNP Memory (RMP Table) Coverage	Disabled
3D V-Cache	Auto
ACPI CST C2 Latency	800
Probe Filter Organization	Dedicated
Periodic Directory Rinse (PDR) Tuning	Auto
Number of Enabled CPU Cores Per Socket	All

5. Save the UEFI settings again.
6. Exit UEFI to reboot.

Ubuntu Server LTS installation

The OS was installed using the Virtual Media mount on the SR685a BMC, which can be accessed via the XCC Remote Console, as documented at https://pubs.lenovo.com/xcc2/nn1jo_c_mountingdevices.



Reboot the server to boot to the Ubuntu installer and install Linux on the RAID-protected m.2 NVMe drives.

After installation, edit the /etc/default/grub file and change the GRUB_CMDLINE_LINUX line as below and save.

```
GRUB_CMDLINE_LINUX="panic=0 nowatchdog msr.allow_writes=on nokaslr iommu=pt
numa_balancing=disable norandmaps mitigations=off tsc=nowatchdog nmi_watchdog=0
amdgpu.noretry=1 pci=realloc=off"
```

Update grub and reboot the server

```
$ sudo update-grub
$ sudo reboot
```

Sysfs tuning

To maximize CPU performance, AMD recommends the following settings be made after each boot using cpupower to disable cstate 2 and set the performance governor to “performance”.

```
sudo apt-get install -y linux-tools-common
sudo cpupower idle-set -d 2
sudo cpupower frequency-set -g performance
```

Check the results

```
$ sudo cpupower idle-info
CPUidle driver: acpi_idle
CPUidle governor: menu
```

```
analyzing CPU 89:

Number of idle states: 3
Available idle states: POLL C1 C2
POLL:
Flags/Description: CPUIDLE CORE POLL IDLE
Latency: 0
Usage: 7131333332
Duration: 461980464310
C1:
Flags/Description: ACPI FFH MWAIT 0x0
Latency: 1
Usage: 12382
Duration: 61012993
C2 (DISABLED) :
Flags/Description: ACPI IOPORT 0x814
Latency: 800
Usage: 5380
Duration: 510016699

$ sudo cpupower frequency-info
analyzing CPU 80:
  driver: acpi-cpufreq
  CPUs which run at the same hardware frequency: 80
  CPUs which need to have their frequency coordinated by software: 80
  maximum transition latency:  Cannot determine or is not supported.
  hardware limits: 1.50 GHz - 2.45 GHz
  available frequency steps:  2.45 GHz, 1.90 GHz, 1.50 GHz
  available cpufreq governors: conservative ondemand userspace powersave performance
schedutil
  current policy: frequency should be within 1.50 GHz and 2.45 GHz.
                  The governor "performance" may decide which speed to use
                  within this range.
  current CPU frequency: 2.45 GHz (asserted by call to hardware)
  boost state support:
    Supported: yes
    Active: yes
    Boost States: 0
    Total States: 3
    Pstate-P0:  2450MHz
    Pstate-P1:  1900MHz
    Pstate-P2:  1500MHz
```

The above settings should be applied after each boot via a new systemd service.

```
$ cat << EOF | sudo tee /etc/systemd/system/cpupower.service
[Unit]
Description=CPU Performance
```

```
[Service]
Type=oneshot
ExecStartPre=/usr/bin/cpupower frequency-set -g performance
ExecStart=/usr/bin/cpupower idle-set -d 2
[Install]
WantedBy=multi-user.target
EOF
```

Restart systemd and enable the new systemd service.

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable cpupower.service
$ sudo systemctl start cpupower.service
$ sudo systemctl status cpupower.service
```

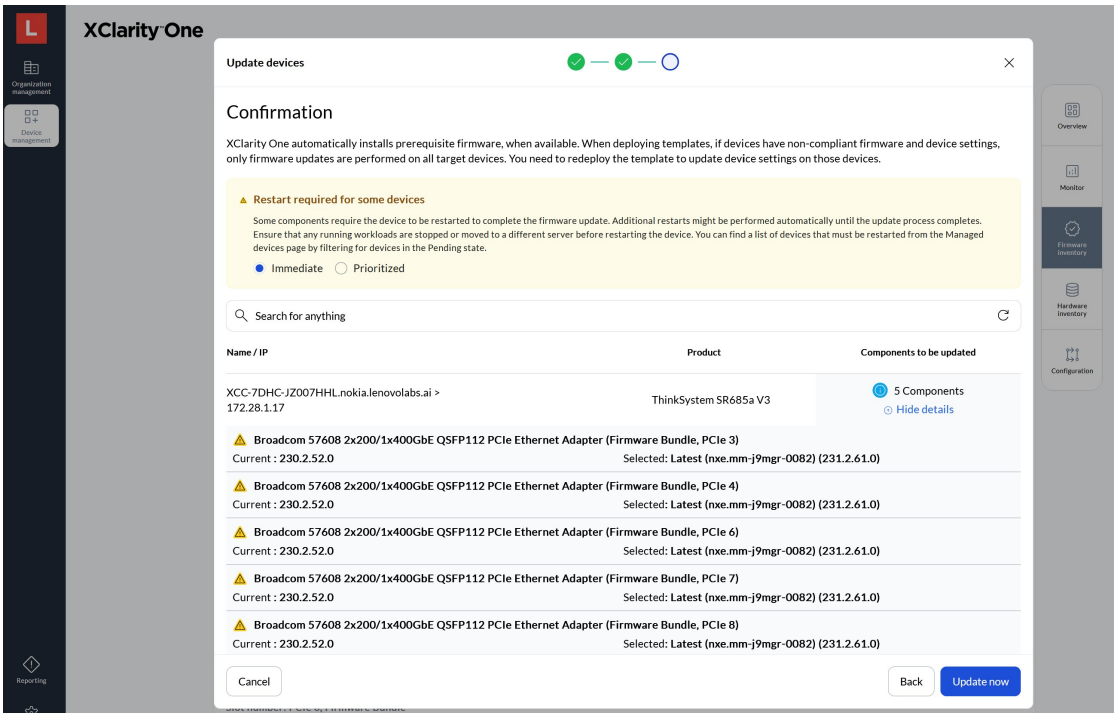
Reboot the server and verify the new service is active on boot.

```
$ sudo journalctl -u cpupower.service
```

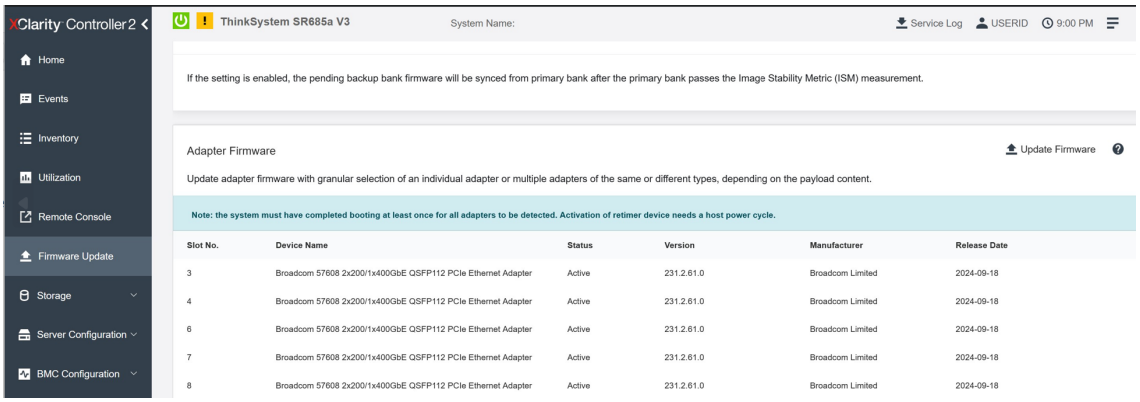
5.1.1 GPU NIC setup and configuration

This validated design uses both the Broadcom P2200L (L=Lenovo branded) and P1400G (G= Generic) Thor2 400Gb NICs for GPU NICs. Other GPU NICs are available and require different configurations, so the following instructions are only relevant to the Broadcom Thor2 NICs. Refer to the following Broadcom guide before starting: https://docs.broadcom.com/doc/957608-AN2XX

Verify the firmware running on each of the Broadcom GPU NICs. If the NICs have a Lenovo part number, the firmware update should be automated using XClarityOne.



Alternatively, update the NIC firmware using the XCC on each system.



If the NICs are generic Broadcom NICs, the firmware must be applied manually from Linux. Boot the server and download the Broadcom SW package from Broadcom. See Section 2.2.1 in the BCM957608-AN2XX document linked above, then run the software installer to install Peer Memory Direct versions of the bnxt modules.

```
$ sudo ./install.sh -v -i 05:00.0 -i 27:00.0 -i 47:00.0 -i 65:00.0 -i 85:00.0 -i a7:00.0 -i
c7:00.0 -i e5:00.0 -f -g
```

Verify the updated bnxt modules are correctly installed.

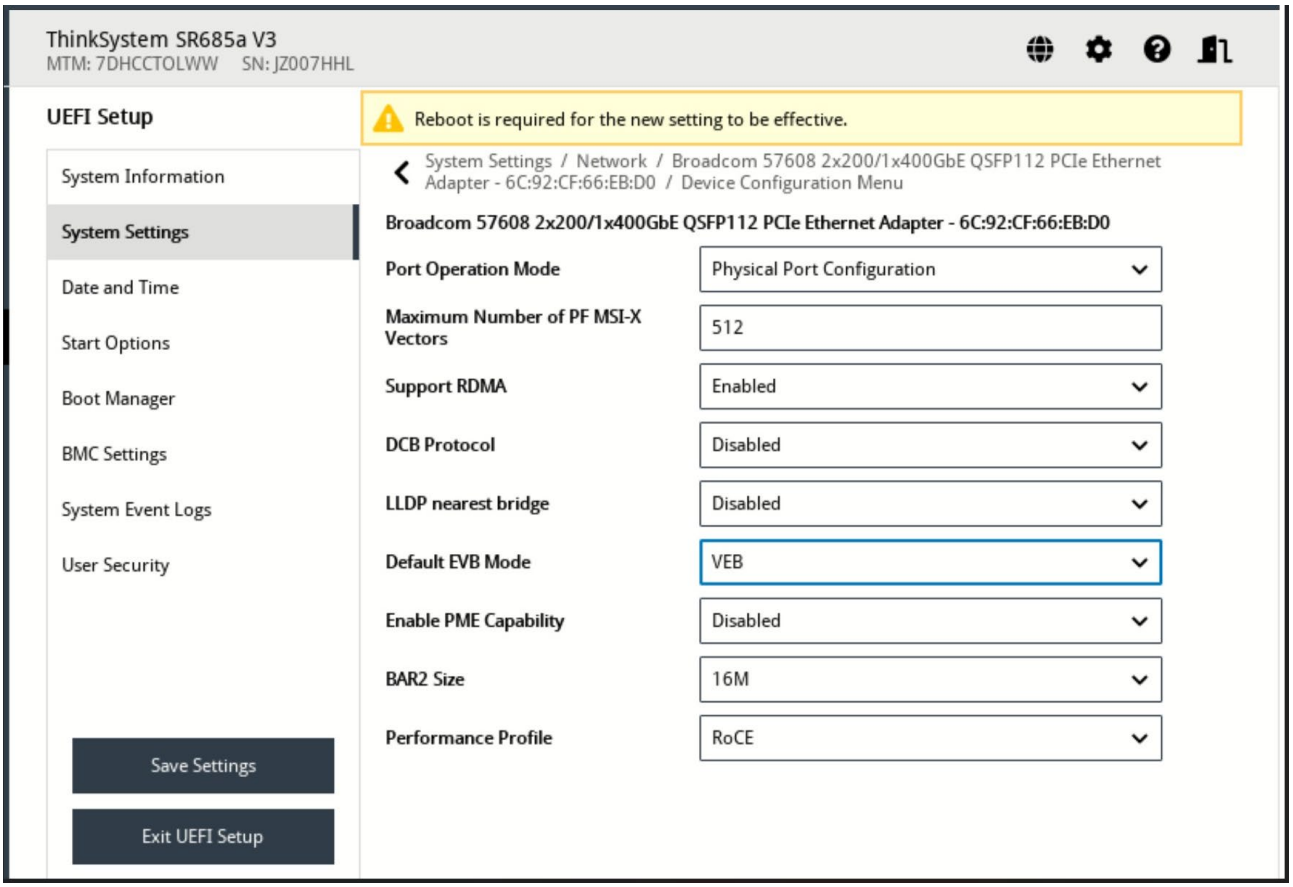
```
$ sudo lsmod | grep bnxt
bnxt_re               536576  0
ib_peer_mem            28672  3 bnxt_re
ib_uverbs             188416  4 ib_peer_mem,bnxt_re,rdma_ucm,mlx5_ib
ib_core               507904  8
rdma_cm,iw_cm,bnxt_re,ib_umad,rdma_ucm,ib_uverbs,mlx5_ib,ib_cm
bnxt_en              3612672  1 bnxt_re
tls                   155648  10 bnxt_en,mlx5_core

$ sudo dkms status
amdgpu/6.12.12-2164967.22.04, 6.8.0-65-generic, x86_64: installed
netxtreme-peer-mem/233.0.152.2, 6.8.0-65-generic, x86_64: installed
```

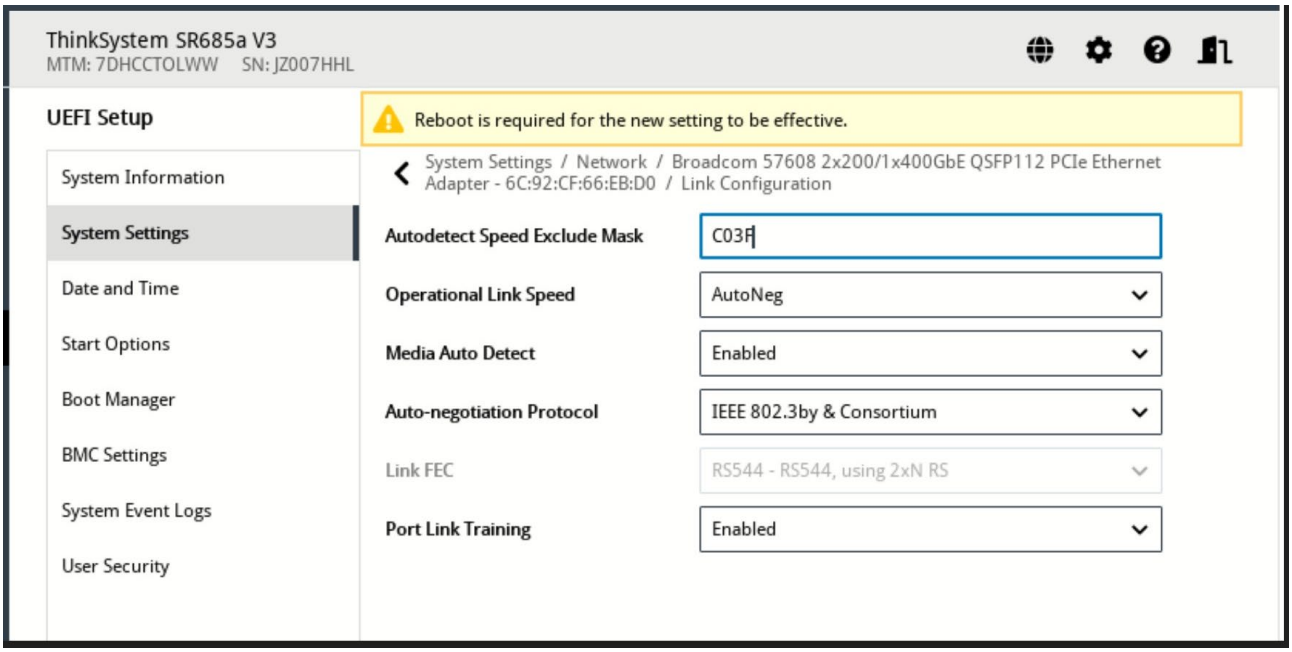
Note: The netxtreme-peer-mem build created by the installer script contains the modified, peer memory direct versions of bnxt_re and bnxt_en. It may be necessary to remove any existing installations of these modules from the running kernel using dkms commands to avoid duplicate modules of the same version.

Verify the installer created a QoS setting file.

```
$ sudo cat /etc/bnxt_re/bnxt_re.conf
ENABLE_FC=1
FC_MODE=3
ROCE_PRI=3
ROCE_DSCP=26
CNP_PRI=7
CNP_DSCP=48
ROCE_BW=50
UTILITY=3
```

For NICs with Lenovo Part numbers, use the UEFI Configuration menu (available during boot with F1) to verify that the NICs are configured to work with the Nokia 400G switch ports.





For Thor2 cards without Lenovo part numbers, it is necessary to check the settings of the NICs using the following Broadcom NIC CLI-based scripts instead of the UEFI settings.

Enable Relaxed Ordering and RoCEv2 on the NICs by creating a file and executing this bash script.

```
#!/bin/bash
#Description: Script to check and set Thor2 cards

# Check if Relaxed Ordering is enabled
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm -getoption pcie_relaxed_ordering; done
# Set Relaxed Ordering if not enabled
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm -setoption pcie_relaxed_ordering -value 1; done
```

Check if RDMA support is enabled.

```
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm -getoption support_rdma -scope 0; done
# Set RMDA support if not enabled
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm -setoption support_rdma -scope 0 -value 1; done
```

Check if RMDA support is enabled.

```
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm -getoption performance_profile; done
# Set RMDA support if not enabled
```

```
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm
-setoption performance_profile -value 1; done
```

Check that DCBx mode and FW LLDP are disabled on NICs by saving the following script to a file and running it.

```
#!/bin/bash
#Description: Script to check DCBx NVM CFG on Thor2
#Disable any of these in a separate script if they return 'Enabled'

# Check if DCBx mode is enabled
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm
-getoption dcbx_mode -scope 0; done

# Check if FW LLDP mode is enabled
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm
-getoption lldp_nearest_bridge -scope 0; done

# Check LLDP nearest non-tmpr bridge
for i in `niccli --listdev | grep 'PCI Address' | awk {'print $4'}`; do  niccli -pci $i nvm
-getoption lldp_nearest_non_tpmr_bridge -scope 0; done
```

See Section 2.2.6.7 in the BCM957608-AN2XX document https://docs.broadcom.com/doc/957608-AN2XX for information about how to disable these settings.

Reboot the system to apply settings to the NICs NVM.

ACS needs to be disabled for Peer-Memory Direct to work across the PCI bridges. ACS should be disabled by default on the Lenovo SR685a. To verify that ACS is disabled by default, create a file with the following script and check that it returns no lines.

```
#!/bin/bash
#Description: Script to return devices with ACS enabled
for BDF in `lspci -n | awk '{print $1}'`; do
if lspci -vvv -s "$BDF" | grep -i ACSCtl | grep -qi SrcValid+; then
lspci -vvv -s "$BDF" | head -1
lspci -vvv -s "$BDF" | grep -i ACSCtl
fi
done
```

5.1.2 IP addressing

This design uses an optimized rail configuration across the Nokia switch fabric and as such can utilize the main Linux route table without the need for custom source routing tables. The advantage of this approach is that the networking can be configured via a common YAML file using Netplan and managed by networkd.

Note: Device names for the single port P1400 cards are in the form ensXnp0 (where X = slot number of the NIC and np0 = Port0). Device names for the P2200 cards working in single port 400Gb mode are in the form ensXf0np0 (where X = slot number of the NIC and np0 = Port0). Changes need to be made to the following Netplan files based on the type of cards installed.

GPU server 1 Netplan file for GPU NICs

```
# Let networkd manage all devices on this system
# P1400 Device Names
network:
  version: 2
  renderer: networkd
  ethernets:
    ens1np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens2np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens3np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens4np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens5np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
```

```
      mtu: 4200
    ens6np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens7np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens8np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
  vlans:
    ens1np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens1np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:1:0:1001/96
      routes:
        - to: fd00:100:1:1:0:5:0::/96
          via: fd00:100:1:1:0:1:0:1
          on-link: true
          metric: 10
    ens2np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens2np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:1:0:1001/96
      routes:
        - to: fd00:100:2:1:0:5:0::/96
```

```
        via: fd00:100:2:1:0:1:0:1
        on-link: true
        metric: 10
    ens3np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens3np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:3:0:1001/96
      routes:
        - to: fd00:100:1:1:0:7:0::/96
          via: fd00:100:1:1:0:3:0:1
          on-link: true
          metric: 10
    ens4np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens4np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:3:0:1001/96
      routes:
        - to: fd00:100:2:1:0:7:0::/96
          via: fd00:100:2:1:0:3:0:1
          on-link: true
          metric: 10
    ens5np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens5np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:2:0:1001/96
      routes:
        - to: fd00:100:1:1:0:6:0::/96
          via: fd00:100:1:1:0:2:0:1
          on-link: true
          metric: 10
    ens6np0.1000:
      dhcp4: no
      mtu: 4200
```

```
      id: 1000
      link: ens6np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:2:0:1001/96
      routes:
        - to: fd00:100:2:1:0:6:0::/96
          via: fd00:100:2:1:0:2:0:1
          on-link: true
          metric: 10
    ens7np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens7np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:4:0:1001/96
      routes:
        - to: fd00:100:1:1:0:8:0::/96
          via: fd00:100:1:1:0:4:0:1
          on-link: true
          metric: 10
    ens8np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens8np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:4:0:1001/96
      routes:
        - to: fd00:100:2:1:0:8:0::/96
          via: fd00:100:2:1:0:4:0:1
          on-link: true
          metric: 10
```

GPU server 2 Netplan file for GPU NICs

```
# Let networkd manage all devices on this system
# P1400 Device Names
network:
  version: 2
  renderer: networkd
  ethernets:
```

```
    ens1np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens2np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens3np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens4np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens5np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens6np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
    ens7np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
```

```
      mtu: 4200
    ens8np0:
      dhcp4: no
      dhcp6: no
      addresses: []
      optional: true
      link-local: []
      mtu: 4200
  vlans:
    ens1np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens1np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:5:0:1001/96
      routes:
        - to: fd00:100:1:1:0:1:0::/96
          via: fd00:100:1:1:0:5:0:1
          on-link: true
          metric: 10
    ens2np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens2np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:5:0:1001/96
      routes:
        - to: fd00:100:2:1:0:1:0::/96
          via: fd00:100:2:1:0:5:0:1
          on-link: true
          metric: 10
    ens3np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens3np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:7:0:1001/96
      routes:
        - to: fd00:100:1:1:0:3:0::/96
```

```
        via: fd00:100:1:1:0:7:0:1
        on-link: true
        metric: 10
    ens4np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens4np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:7:0:1001/96
      routes:
        - to: fd00:100:2:1:0:3:0::/96
          via: fd00:100:2:1:0:7:0:1
          on-link: true
          metric: 10
    ens5np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens5np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:6:0:1001/96
      routes:
        - to: fd00:100:1:1:0:2:0::/96
          via: fd00:100:1:1:0:6:0:1
          on-link: true
          metric: 10
    ens6np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens6np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:6:0:1001/96
      routes:
        - to: fd00:100:2:1:0:2:0::/96
          via: fd00:100:2:1:0:6:0:1
          on-link: true
          metric: 10
    ens7np0.1000:
      dhcp4: no
      mtu: 4200
```

```
      id: 1000
      link: ens7np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:1:1:0:8:0:1001/96
      routes:
        - to: fd00:100:1:1:0:4:0::/96
          via: fd00:100:1:1:0:8:0:1
          on-link: true
          metric: 10
    ens8np0.1000:
      dhcp4: no
      mtu: 4200
      id: 1000
      link: ens8np0
      link-local: [ ipv6 ]
      accept-ra: false
      addresses:
        - fd00:100:2:1:0:8:0:1001/96
      routes:
        - to: fd00:100:2:1:0:4:0::/96
          via: fd00:100:2:1:0:8:0:1
          on-link: true
          metric: 10
```

Run ‘netplan generate’ and ‘netplan apply’ on both servers and restart networking.

```
sudo systemctl enable systemd-networkd
sudo systemctl start systemd-networkd
```

Check the RoCE device names and GID location used for the interfaces created with Netplan by creating the following script.

```
#!/bin/bash
#Description: Iterate through Broadcom NICs to determine bnxt device name and GID used for
IPv6 address
for i in `ibv_devinfo -l | grep bnxt`
  do
    echo "$i"
    ibv_devinfo -v -d $i | grep  GID | grep fd00 | grep -i v2;
done
```

```
./check-bnxt-addrs.sh
bnxt_re0            GID[  3]:               fd00:100:2:1:0:4:0:1001, RoCE v2
bnxt_re1            GID[  3]:               fd00:100:1:1:0:2:0:1001, RoCE v2
bnxt_re2            GID[  3]:               fd00:100:1:1:0:3:0:1001, RoCE v2
```

```
bnxt_re3            GID[  3]:               fd00:100:1:1:0:4:0:1001, RoCE v2
bnxt_re4            GID[  3]:               fd00:100:2:1:0:2:0:1001, RoCE v2
bnxt_re5            GID[  3]:               fd00:100:2:1:0:3:0:1001, RoCE v2
bnxt_re6            GID[  3]:               fd00:100:2:1:0:1:0:1001, RoCE v2
bnxt_re7            GID[  3]:               fd00:100:1:1:0:1:0:1001, RoCE v2
```

Note: The GID used for RoCE IPv6 should be GID[3]. If another GID is seen then reset the links on all the GPU NIC interfaces, re-apply the Netplan file, and recheck.

Repeat the same on GPU Server 2.

```
./check-bnxt-addrs.sh
bnxt_re0         GID[  3]:               fd00:100:2:1:0:8:0:1001, RoCE v2
bnxt_re1         GID[  3]:               fd00:100:1:1:0:6:0:1001, RoCE v2
bnxt_re2         GID[  3]:               fd00:100:1:1:0:7:0:1001, RoCE v2
bnxt_re3         GID[  3]:               fd00:100:1:1:0:8:0:1001, RoCE v2
bnxt_re4         GID[  3]:               fd00:100:2:1:0:6:0:1001, RoCE v2
bnxt_re5         GID[  3]:               fd00:100:2:1:0:7:0:1001, RoCE v2
bnxt_re6         GID[  3]:               fd00:100:2:1:0:5:0:1001, RoCE v2
bnxt_re7         GID[  3]:               fd00:100:1:1:0:5:0:1001, RoCE v2
```

On GPU server 1, create a script to test the reachability of the remote GPU NICs.

```
#!/bin/bash
#Descrption: Test the reachability of the remote GPUS

echo ping test remote gpu 1 from local gpu 1
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:5:0:1001 -I ens1np0.1000
echo ping test remote gpu 2 from local gpu 2
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:5:0:1001 -I ens2np0.1000
echo ping test remote gpu 3 from local gpu 3
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:7:0:1001 -I ens3np0.1000
echo ping test remote gpu 4 from local gpu 4
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:7:0:1001 -I ens4np0.1000
echo ping test remote gpu 5 from local gpu 5
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:6:0:1001 -I ens5np0.1000
echo ping test remote gpu 6 from local gpu 6
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:6:0:1001 -I ens6np0.1000
echo ping test remote gpu 7 from local gpu 7
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:8:0:1001 -I ens7np0.1000
echo ping test remote gpu 8 from local gpu 8
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:8:0:1001 -I ens8np0.1000
```

Run the script and observe all eight remote NICs are reachable from server 1.

```
./ping-remote-gpus.sh
ping test remote gpu 1 from local gpu 1
PING fd00:100:1:1:0:5:0:1001(fd00:100:1:1:0:5:0:1001) from fd00:100:1:1:0:1:0:1001
ens1np0.1000: 4096 data bytes
4104 bytes from fd00:100:1:1:0:5:0:1001: icmp_seq=1 ttl=63 time=0.140 ms
4104 bytes from fd00:100:1:1:0:5:0:1001: icmp_seq=2 ttl=63 time=0.140 ms
4104 bytes from fd00:100:1:1:0:5:0:1001: icmp_seq=3 ttl=63 time=0.129 ms
4104 bytes from fd00:100:1:1:0:5:0:1001: icmp_seq=4 ttl=63 time=0.139 ms
4104 bytes from fd00:100:1:1:0:5:0:1001: icmp_seq=5 ttl=63 time=0.142 ms
```

On GPU server 2, create a script to test the reachability of the remote GPU NICs

```
#!/bin/bash
#Descrption: Test the reachability of the remote GPUS

echo ping test remote gpu 1 from local gpu 1
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:1:0:1001 -I ens1np0.1000
echo ping test remote gpu 2 from local gpu 2
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:1:0:1001 -I ens2np0.1000
echo ping test remote gpu 3 from local gpu 3
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:3:0:1001 -I ens3np0.1000
echo ping test remote gpu 4 from local gpu 4
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:3:0:1001 -I ens4np0.1000
echo ping test remote gpu 5 from local gpu 5
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:2:0:1001 -I ens5np0.1000
echo ping test remote gpu 6 from local gpu 6
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:2:0:1001 -I ens6np0.1000
echo ping test remote gpu 7 from local gpu 7
ping -6 -c 5 -M do -s 4096 fd00:100:1:1:0:4:0:1001 -I ens7np0.1000
echo ping test remote gpu 8 from local gpu 8
ping -6 -c 5 -M do -s 4096 fd00:100:2:1:0:4:0:1001 -I ens8np0.1000
```

Run the script and observe all eight remote NICs are reachable from server 2.

```
./ping-remote-gpus.sh
ping test remote gpu 1 from local gpu 1
PING fd00:100:1:1:0:1:0:1001(fd00:100:1:1:0:1:0:1001) from fd00:100:1:1:0:5:0:1001
ens1np0.1000: 4096 data bytes
4104 bytes from fd00:100:1:1:0:1:0:1001: icmp_seq=1 ttl=63 time=0.141 ms
4104 bytes from fd00:100:1:1:0:1:0:1001: icmp_seq=2 ttl=63 time=0.154 ms
4104 bytes from fd00:100:1:1:0:1:0:1001: icmp_seq=3 ttl=63 time=0.158 ms
4104 bytes from fd00:100:1:1:0:1:0:1001: icmp_seq=4 ttl=63 time=0.154 ms
4104 bytes from fd00:100:1:1:0:1:0:1001: icmp_seq=5 ttl=63 time=0.161 ms
```

5.1.3 ROCm software installation

Consult the AMD ROCm Compatibility matrix to select the correct ROCm version for the installed operating system.

https://rocm.docs.amd.com/en/latest/compatibility/compatibility-matrix.html

Compatibility matrix

2025-08-12 39 min read time

Applies to Linux

Use this matrix to view the ROCm compatibility and system requirements across successive major and minor releases.

You can also refer to the past versions of ROCm compatibility matrix.

Accelerators and GPUs listed in the following table support compute workloads (no display information or graphics). If you're using ROCm with AMD Radeon or Radeon Pro GPUs for graphics workloads, see the Use ROCm on Radeon GPU documentation to verify compatibility and system requirements.

ROCm Version	6.4.3	6.4.2	6.3.0
Operating systems & kernels	Ubuntu 24.04.2	Ubuntu 24.04.2	Ubuntu 24.04.2
	Ubuntu 22.04.5	Ubuntu 22.04.5	Ubuntu 22.04.5
	RHEL 9.6, 9.4	RHEL 9.6, 9.4	RHEL 9.5, 9.4
	RHEL 8.10	RHEL 8.10	RHEL 8.10
	SLES 15 SP7, SP6	SLES 15 SP7, SP6	SLES 15 SP6, SP5
	Oracle Linux 9, 8 [1]	Oracle Linux 9, 8 [1]	Oracle Linux 8.10 [1]
	Debian 12 [2]	Debian 12 [2]	
	Azure Linux 3.0 [1]	Azure Linux 3.0 [1]	

Note: This reference document used Ubuntu 22.04.5 as the host OS and ROCm 6.4.3.

Install the ROCm version using the AMD Quick Start guide:

https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/quick-start.html

Quick start installation guide

2025-08-01 11 min read time

Applies to Linux

This topic provides basic installation instructions for ROCm on Linux using your distribution's native package manager. Before you begin, you should confirm your kernel version matches the ROCm system requirements.

Once you do, review your required installation instructions by selecting your operating system and version, and then run the provided commands in your terminal. The commands include the installation of the prerequisites, along with installing ROCm.

For more in-depth installation instructions, refer to ROCm on Linux detailed installation overview.

Note

If you're using ROCm with AMD Radeon or Radeon Pro GPUs for graphics workloads, see the Use ROCm on Radeon GPU documentation for installation instructions .

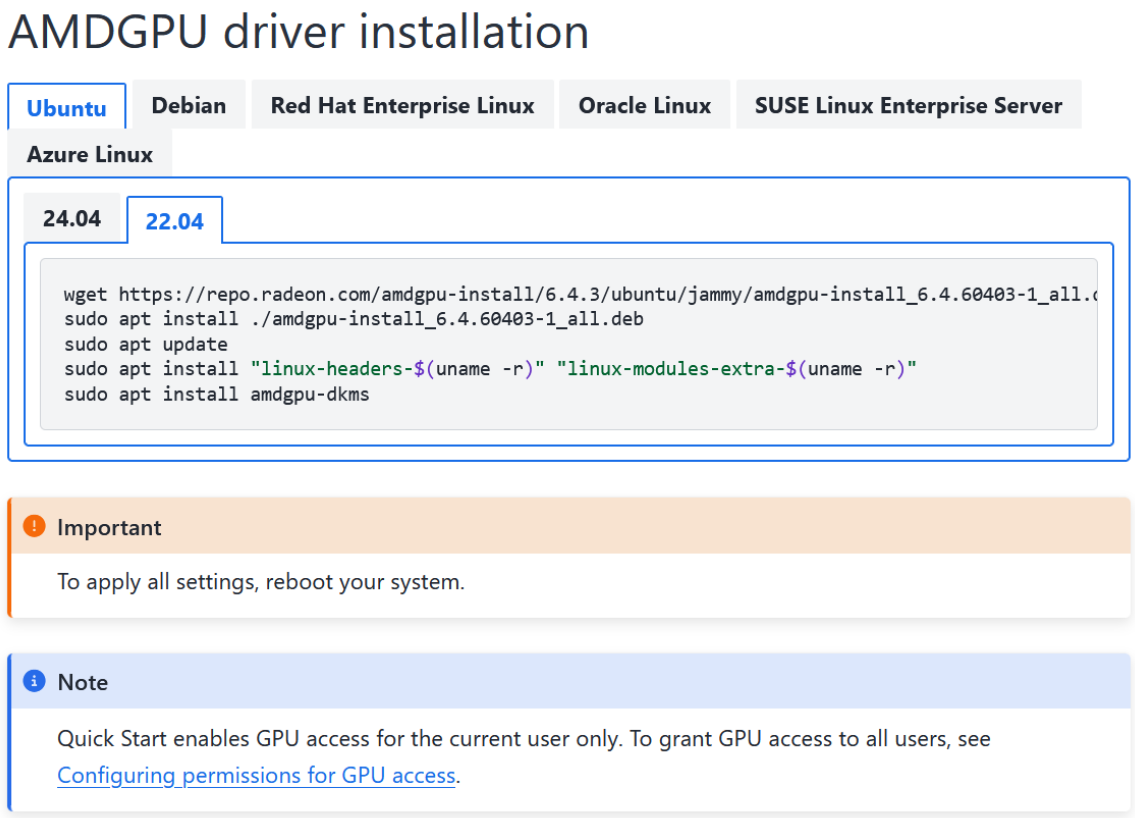
ROCm installation

Ubuntu | Debian | Red Hat Enterprise Linux | Oracle Linux | SUSE Linux Enterprise Server | Azure Linux

24.04 | 22.04

```
wget https://repo.radeon.com/amdgpu-install/6.4.3/ubuntu/jammy/amdgpu-install_6.4.60403-1_all.
sudo apt install ./amdgpu-install_6.4.60403-1_all.deb
sudo apt update
sudo apt install python3-setuptools python3-wheel
sudo usermod -a -G render,video $LOGNAME # Add the current user to the render and video groups
sudo apt install rocm
```

Then install the AMD gpu drivers and follow the post installation instructions.



5.1.4 RCCL setup

RCCL is AMD's port of NCCL. RCCL can be used to load-test one or more GPUs on a standalone server or schedule a test using multiple GPUs spread over multiple servers in a multi-node configuration. RCCL can be configured to use OpenMPI and UCX which will automatically discover the GPU topology and communicate between nodes before executing RCCL.

RCCL can be challenging to compile and run. This testing used the instructions documented in Chapter 8 of this Broadcom document: https://docs.broadcom.com/doc/957608-AN2XX.

IMPORTANT: Perform the following section using a non-root user-id as it is strongly recommended not to use OpenMPI or password-less SSH with a privileged Linux user-id such as root.

1. Set local variables to download the desired versions.

```
$ export UCX_VER=v1.18
$ export OMPI_VER=v5.0.7
```

2. Install UCX.

```
# The UCX build needs to point to the ROCm installation /opt/rocm
# as shown in the steps below.
cd $HOME
git clone --recursive -b ${UCX_VER} https://github.com/openucx/ucx.git
cd ucx
./autogen.sh
mkdir ucx_install
mkdir build
cd build
../contrib/configure-release --disable-debug --disable-assertions --disable-params-check --
with-
rocm=/opt/rocm --with-rc --with-ud --with-dc --with-dm --with-ib-hw-tm --
prefix=$HOME/ucx/ucx_install
--disable-log
make -j $(nproc)
make -j $(nproc) install
```

Verification:

```
$HOME/ucx/ucx_install/bin/ucx_info -d

# Memory domain: self
#     Component: self
#             register: unlimited, cost: 0 nsec
#           remote key: 0 bytes
#           rkey_ptr is supported
#         memory types: host (access,reg_nonblock,reg,cache)
#
#      Transport: self
#         Device: memory
#           Type: loopback
#  System device: <unknown>
#
. . .
```

3. Install OpenMPI.

```
The Open MPI build needs to point to the UCX installation $HOME/ucx/ucx_install
# as shown in the steps below.
# On Ubuntu please install flex
sudo apt install flex
cd $HOME
git clone --recursive -b $OMPI_VER https://github.com/open-mpi/ompi.git
cd ompi
./autogen.pl
mkdir build
mkdir ompi_install
cd build
```

```
../configure --prefix=$HOME/ompi/ompi_install –with ucx=$HOME/ucx/ucx_install --enable-mca-
no-
build=btl-uct
make -j $(nproc)
make -j $(nproc) install
```

Verification:

```
$HOME/ompi/ompi_install/bin/ompi_info | grep Configure
Configured architecture: x86_64-pc-linux-gnu
           Configured by: lenovo
           Configured on: Wed Jul 23 21:24:12 UTC 2025
          Configure host: sr685v3-svr2
  Configure command line: '--prefix=/home/lenovo/ompi/ompi_install' '--with-
ucx=/home/lenovo/ucx/ucx_install' '--enable-mca-no-build=btl-uct'
```

4. Compile RCCL tests.

```
# On Ubuntu install libstdc++-12-i
sudo apt install libstdc++-12-i
cd $HOME
git clone https://github.com/ROCmSoftwarePlatform/rccl-tests.git
cd rccl-tests/
MPI=1 MPI_HOME=$HOME/ompi/ompi_install/ RCCL_HOME=/opt/rocm/lib make -j $(nproc)
```

5. Configure password-less SSH between GPU servers for OpenMPI to schedule tests between nodes using a non-root login.

```
On server 1 generate a new ssh key
ssh-keygen
Copy the key to server 2 using ssh-copy-id
ssh-copy-id user@host
After the key is copied, ssh into the machine as normal:
ssh user@host
Repeat the steps above on server 2 so both servers can ssh into each other without the need
for a password.
```

6. Create a hostfile using a text editor that defines the test topology; that is, the local management IP and the associated number of slots or GPUs, and the remote management IP and the associated number of slots or GPUs. This file will be used by OpenMPI to schedule the RCCL tests.

   Example of a 1 node GPU test configuration with 8 GPU slots

```
$ cat hostfile.txt
172.28.1.112 slots=8
```

Example of a 2 node GPU test configuration, each with 8 GPU slots

```
$ cat hostfile.txt
172.28.1.112 slots=8
172.28.1.122 slots=8
```

There are many different RCCL tests and configurations. The goal is to test each node individually and then test across all 16 GPUs to exercise the switch fabric. This design uses an optimized rail architecture, so the PXN network is needed to access the remote GPU via its local GPU peer.

Start by testing all 8 GPUs on the local node. This test uses the internal links between the GPUs, not the switch fabric between the servers.

Use a single host entry in the hostfile.

```
$ cat hostfile.txt
172.28.1.112 slots=8
```

Execute the following script (single node 8 GPUs).

```
#!/bin/bash -x

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/ompi/ompi_install/lib
echo "LD Path -> $LD_LIBRARY_PATH"

export NCCL_IB_GID_INDEX=3

    /home/lenovo/ompi/build/ompi/tools/mpirun/mpirun -np 8 -N 8 --hostfile hostfile.txt \
      -x NCCL_IB_GID_INDEX=3 \
      -x NCCL_IB_HCA=bnxt_re0,bnxt_re1,bnxt_re2,bnxt_re3,bnxt_re4,bnxt_re5,bnxt_re6,bnxt_re7
\
      -x PATH=${MPI_INSTALL_DIR}/bin:${ROCM_PATH}/bin:$PATH \
      -x LD_LIBRARY_PATH=${RCCL_INSTALL_DIR}/lib:${MPI_INSTALL_DIR}/lib:$LD_LIBRARY_PATH \
      -x NCCL_IB_PCI_RELAXED_ORDERING=1 \
      -x HSA_DISABLE_CACHE=1 \
      -x HSA_FORCE_FINE_GRAIN_PCIE=1 \
      -x NCCL_NET_GDR_LEVEL=SYS \
      -x NCCL_NET_GDR_READ=1 \
      -x NCCL_P2P_LEVEL=SYS \
      -x NCCL_SHM_DISABLE=1 \
      -x NCCL_DMABUF_ENABLE=0 \
      -x NCCL_IB_TIMEOUT=22 \
      -x NCCL_IB_DISABLE=0 \
      -x NCCL_MIN_NCHANNELS=32 \
      -x NCCL_DEBUG=INFO \
      --bind-to none \
      --mca pml ucx \
      --mca osc ucx \
      --mca spml ucx \
```

```
      --mca btl ^vader,tcp,openib,ucl \
      /home/lenovo/rccl-tests/build/all_reduce_perf -b 8 -e 16G -f 2 -c 0 -g 1
```

Results:

```
#                                                              out-of-place
in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong
time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)
(us)  (GB/s)  (GB/s)
           8             2     float     sum      -1    22.11    0.00    0.00    N/A
17.14    0.00    0.00    N/A
          16             4     float     sum      -1    17.86    0.00    0.00    N/A
18.12    0.00    0.00    N/A
          32             8     float     sum      -1    18.46    0.00    0.00    N/A
17.98    0.00    0.00    N/A
          64            16     float     sum      -1    20.01    0.00    0.01    N/A
19.43    0.00    0.01    N/A
         128            32     float     sum      -1    25.46    0.01    0.01    N/A
23.89    0.01    0.01    N/A
         256            64     float     sum      -1    24.93    0.01    0.02    N/A
23.90    0.01    0.02    N/A
         512           128     float     sum      -1    24.74    0.02    0.04    N/A
24.05    0.02    0.04    N/A
        1024           256     float     sum      -1    19.25    0.05    0.09    N/A
18.90    0.05    0.09    N/A
        2048           512     float     sum      -1    16.61    0.12    0.22    N/A
16.48    0.12    0.22    N/A
        4096          1024     float     sum      -1    15.88    0.26    0.45    N/A
15.44    0.27    0.46    N/A
        8192          2048     float     sum      -1    16.93    0.48    0.85    N/A
15.11    0.54    0.95    N/A
       16384          4096     float     sum      -1    15.43    1.06    1.86    N/A
15.09    1.09    1.90    N/A
       32768          8192     float     sum      -1    15.46    2.12    3.71    N/A
15.09    2.17    3.80    N/A
       65536         16384     float     sum      -1    16.31    4.02    7.03    N/A
15.33    4.27    7.48    N/A
      131072         32768     float     sum      -1    17.31    7.57   13.25    N/A
16.47    7.96   13.93    N/A
      262144         65536     float     sum      -1    20.94   12.52   21.91    N/A
20.04   13.08   22.89    N/A
      524288        131072     float     sum      -1    25.47   20.58   36.02    N/A
24.97   20.99   36.74    N/A
     1048576        262144     float     sum      -1    28.77   36.44   63.77    N/A
28.15   37.26   65.20    N/A
     2097152        524288     float     sum      -1    36.13   58.04  101.57    N/A
35.73   58.69  102.70    N/A
     4194304       1048576     float     sum      -1    55.71   75.28  131.75    N/A
52.85   79.36  138.89    N/A
     8388608       2097152     float     sum      -1    85.37   98.26  171.95    N/A
84.88   98.83  172.95    N/A
```

```
    16777216       4194304     float     sum      -1    150.2  111.69  195.46    N/A
149.9  111.91  195.84    N/A
    33554432       8388608     float     sum      -1    221.1  151.79  265.64    N/A
221.4  151.56  265.22    N/A
    67108864      16777216     float     sum      -1    407.5  164.69  288.21    N/A
408.4  164.31  287.55    N/A
   134217728      33554432     float     sum      -1    773.7  173.47  303.56    N/A
775.1  173.16  303.04    N/A
   268435456      67108864     float     sum      -1   1511.0  177.65  310.89    N/A
1516.3  177.03  309.80    N/A
   536870912     134217728     float     sum      -1   2998.2  179.07  313.37    N/A
3002.4  178.81  312.92    N/A
  1073741824     268435456     float     sum      -1   5932.9  180.98  316.71    N/A
5940.4  180.75  316.32    N/A
  2147483648     536870912     float     sum      -1    11850  181.22  317.14    N/A
11874  180.86  316.51    N/A
  4294967296    1073741824     float     sum      -1    23718  181.09  316.90    N/A
23729  181.00  316.75    N/A
  8589934592    2147483648     float     sum      -1    47225  181.89  318.31    N/A
47209  181.96  318.42    N/A
 17179869184    4294967296     float     sum      -1    94136  182.50  319.38    N/A
94261  182.26  318.95    N/A
```

Run RCCL all-to-all on all 16 GPUs spread across both servers.

Update the hostfile to include both GPU servers.

```
$ cat hostfile.txt
172.28.1.112 slots=8
172.28.1.122 slots=8
```

Note: The PXN network is needed for multimode tests using optimized rail and needs to be activated by explicitly negating the disable option: ‘NCCL_PXN_DISABLE=0’.

Execute the following script (multinode, two nodes, each with 8 GPUs):

```
#!/bin/bash -x

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/ompi/ompi_install/lib
echo "LD Path -> $LD_LIBRARY_PATH"

export NCCL_IB_GID_INDEX=3

    /home/lenovo/ompi/build/ompi/tools/mpirun/mpirun -np 16 --hostfile hostfile.txt \
      -x PATH -x LD_LIBRARY_PATH \
      -x NCCL_IB_GID_INDEX=3 \
      -x NCCL_IB_HCA=bnxt_re0,bnxt_re1,bnxt_re2,bnxt_re3,bnxt_re4,bnxt_re5,bnxt_re6,bnxt_re7
\
      -x PATH=${MPI_INSTALL_DIR}/bin:${ROCM_PATH}/bin:$PATH \
```

```
      -x LD_LIBRARY_PATH=${RCCL_INSTALL_DIR}/lib:${MPI_INSTALL_DIR}/lib:$LD_LIBRARY_PATH \
      -x NCCL_IB_PCI_RELAXED_ORDERING=1 \
      -x HSA_DISABLE_CACHE=1 \
      -x HSA_FORCE_FINE_GRAIN_PCIE=1 \
      -x NCCL_NET_GDR_READ=1 \
      -x NCCL_DMABUF_ENABLE=0 \
      -x NCCL_IB_TIMEOUT=22 \
      -x NCCL_IB_DISABLE=0 \
      -x NCCL_MIN_NCHANNELS=32 \
      -x NCCL_MAX_NCHANNELS=32 \
      -x NCCL_DEBUG=INFO \
      -x NCCL_PXN_DISABLE=0 \
      --bind-to none \
      --mca pml ucx \
      --mca osc ucx \
      --mca spml ucx \
      --mca btl ^vader,tcp,openib,ucl \
      /home/lenovo/rccl-tests/build/alltoall_perf -b 8 -e 16G -f 2 -c 0 -g 1
```

Results:

```
           0             0     float    none      -1     0.15    0.00    0.00    N/A
0.06    0.00    0.00    N/A
           0             0     float    none      -1     0.05    0.00    0.00    N/A
0.05    0.00    0.00    N/A
           0             0     float    none      -1     0.05    0.00    0.00    N/A
0.05    0.00    0.00    N/A
           0             0     float    none      -1     0.05    0.00    0.00    N/A
0.05    0.00    0.00    N/A
           0             0     float    none      -1     0.05    0.00    0.00    N/A
0.05    0.00    0.00    N/A
         256             4     float    none      -1    43.71    0.01    0.01    N/A
43.41    0.01    0.01    N/A
         512             8     float    none      -1    41.90    0.01    0.01    N/A
41.79    0.01    0.01    N/A
        1024            16     float    none      -1    43.60    0.02    0.02    N/A
41.72    0.02    0.02    N/A
        2048            32     float    none      -1    41.67    0.05    0.05    N/A
41.50    0.05    0.05    N/A
        4096            64     float    none      -1    42.57    0.10    0.09    N/A
43.41    0.09    0.09    N/A
        8192           128     float    none      -1    42.40    0.19    0.18    N/A
41.86    0.20    0.18    N/A
       16384           256     float    none      -1    42.05    0.39    0.37    N/A
41.68    0.39    0.37    N/A
       32768           512     float    none      -1    42.14    0.78    0.73    N/A
41.45    0.79    0.74    N/A
       65536          1024     float    none      -1    42.68    1.54    1.44    N/A
42.91    1.53    1.43    N/A
      131072          2048     float    none      -1    42.95    3.05    2.86    N/A
42.90    3.06    2.86    N/A
```

```
      262144          4096     float    none      -1    42.19    6.21    5.83    N/A
43.48    6.03    5.65    N/A
      524288          8192     float    none      -1    46.69   11.23   10.53    N/A
47.42   11.06   10.37    N/A
     1048576         16384     float    none      -1    59.15   17.73   16.62    N/A
58.50   17.92   16.80    N/A
     2097152         32768     float    none      -1    70.70   29.66   27.81    N/A
71.20   29.45   27.61    N/A
     4194304         65536     float    none      -1    99.96   41.96   39.34    N/A
99.29   42.24   39.60    N/A
     8388608        131072     float    none      -1    156.3   53.66   50.31    N/A
155.8   53.84   50.47    N/A
    16777216        262144     float    none      -1    242.8   69.10   64.78    N/A
242.3   69.23   64.90    N/A
    33554432        524288     float    none      -1    431.3   77.79   72.93    N/A
431.3   77.81   72.94    N/A
    67108864       1048576     float    none      -1    798.1   84.09   78.83    N/A
798.5   84.04   78.79    N/A
   134217728       2097152     float    none      -1   1531.2   87.65   82.17    N/A
1532.2   87.60   82.13    N/A
   268435456       4194304     float    none      -1   2996.9   89.57   83.97    N/A
2997.6   89.55   83.95    N/A
   536870912       8388608     float    none      -1   5929.2   90.55   84.89    N/A
5928.2   90.56   84.90    N/A
  1073741824      16777216     float    none      -1    11792   91.05   85.36    N/A
11789   91.08   85.39    N/A
  2147483648      33554432     float    none      -1    23523   91.29   85.59    N/A
23517   91.32   85.61    N/A
  4294967296      67108864     float    none      -1    47011   91.36   85.65    N/A
46986   91.41   85.70    N/A
  8589934592     134217728     float    none      -1    94011   91.37   85.66    N/A
94017   91.37   85.66    N/A
 17179869184     268435456     float    none      -1   188078   91.34   85.64    N/A
188080   91.34   85.63    N/A
```

5.1.5 RoCE performance testing with perftest

The generic build of perftest needs to be rebuilt with AMD gpu support so that RoCE packets can use GPU memory instead of CPU memory and achieve GPU to GPU packet performance at close to line rate speeds.

The following instructions for compiling perftest with ROCm support can be found in Section 2.7 of the following document: https://docs.broadcom.com/doc/957608-AN2XX.

```
sudo apt install libibumad-dev
sudo apt install pciutils
sudo apt install libpci*
sudo apt install automake autoconf libtool libibverbs-dev ibverbs-utils infiniband-diags
ethtool
librdmacm-dev
cd $HOME
```

```
git clone https://github.com/linux-rdma/perftest.git
cd perftest
./autogen.sh
./configure --prefix=`pwd` --enable-rocm --with-rocm=/opt/rocm
Make
```

The compiled binaries ib_write_bw, ib_send_bw etc should be run from the local build directory. The presence of the “rocm help” option indicates that ROCm has been compiled into the binary and AMD GPU memory can be selected.

Example:

```
./ib_write_bw -h | grep -i rocm
--use_rocm=<rocm device id> Use selected ROCm device for GPUDirect RDMA testing
```

ldd of the ib_send_bw binary should show AMD gpu has been linked:

```
$ ldd ib_send_bw  | grep amdgpu
        libdrm.so.2 => /opt/amdgpu/lib/x86_64-linux-gnu/libdrm.so.2 (0x00007ffff60af000)
        libdrm_amdgpu.so.1 => /opt/amdgpu/lib/x86_64-linux-gnu/libdrm_amdgpu.so.1
(0x00007ffff609f000)
```

Conducting the perftest using GPU memory is the best way to test the maximum throughput of RoCEv2 UDP packets on all GPU rails simultaneously but requires a lot more initial work to set up compared to OpenMPI, which automatically discovers the GPU and NIC topology.

Map the GPU and NIC PCI addresses to determine which devices share the same PCI bridge by listing the RoCE interface names (bnxt_reX) used by each NIC.

Example: Starting with the NIC in slot 1:

```
rdma link show | grep -i ens1np0
link bnxt_re7/1 state ACTIVE physical_state LINK_UP netdev ens1np0
```

List the PCI Bus Id used by the NIC.

```
ethtool -i ens1np0 | grep -i bus
bus-info: 0000:a6:00.0
```

Find the GPU with the closest PCI address as this indicates a shared bridge and the shortest path.

```
amd-smi static | grep -e GPU -e BDF -e OAM_ID
GPU: 5
        OAM_ID: 7
        BDF: 0000:a5:00.0
```

Repeat these steps for each interface ens[1-8]np0 (or ensf0s[1-8]np0) and collate the results.

BIOS NIC Name	RocE NIC Name	NIC Slot	NIC NUMA	NIC PCI Address	Closest GPU	GPU PCI Address	GPU OAM-ID	ROCm-ID (observed)
ens1np0	bnxt_re7	1	1	00:A6:00.00	5	00:A5:00.0	7	7
ens2np0	bnxt_re6	2	1	00:86:00.00	4	00:85:00.0	6	6

BIOS NIC Name	RocE NIC Name	NIC Slot	NIC NUMA	NIC PCI Address	Closest GPU	GPU PCI Address	GPU OAM-ID	ROCm-ID (observed)
ens3np0	bnxt_re2	3	1	00:E6:00.00	7	00:E5:00.0	4	2
ens4np0	bnxt_re5	4	1	00:C6:00.00	6	00:C5:00.0	5	5
ens5np0	bnxt_re1	5	0	00:26:00.00	1	00:25:00.0	3	1
ens6np0	bnxt_re4	6	0	00:06:00.00	0	00:05:00.0	2	4
ens7np0	bnxt_re3	7	0	00:67:00.00	3	00:65:00.0	0	3
ens8np0	bnxt_re0	8	0	00:46:00.00	2	00:45:00.0	1	0

Note: The ROCm-ID passed to ib_send_bw and ib_write_bw may match the GPU OAM-ID or may have to be discovered manually. In our test system, the ROCm-Id had to be discovered manually by passing rocm-id [1-8] to ib_send_bw and making sure the argument selected the ROCm memory in the correct GPU. These discrepancies in IDs should be checked by looking at the PCIe Bus address printed in the ib_send_bw output and recorded, as in the above table.

Example:

```
---------------------------------------------------------------------------------------
 WARNING: BW peak won't be measured in this run.
Using ROCm Device with ID: 7, Name: AMD Instinct MI300X, PCI Bus ID: 0xa5, GCN Arch:
gfx942:sramecc+:xnack-
allocated 262144 bytes of GPU buffer at 0x7fbfe4200000
---------------------------------------------------------------------------------------
```

Perftest uses a client-server pair to exchange RoCEv2 packets which are coordinated between the server and client using a TCP connection as a control channel. For two-way traffic, both a server and a client ib_send_bw process needs to be started for each NIC.

The following example shows a server wrapper script for ib_send_bw that listens on TCP port 18515 for RoCE control on NIC 1 of server 2 and exchanges 256 byte MTU packets with the client connecting from NIC 1 on server 1.

```
#!/bin/bash
#Script to start ib_send_bw server, adjust vars as needed

myGpu=1             #Lenovo GPU number
myRocm=7            #AMD GPU OAM number
myPort=18515        #TCP Port used for RoCEv2 test negotiation
myMtu=256           #MTU used for RoCEv2 UDP test packets
myQueues=2          #Number of test threads
myIters=10000000
myNuma=''
myBnxt=''
myGid=''
```

```
#Determine NUMA from GPU Slot
if [ "$myGpu" -le 4 ]; then
  myNuma=1
else
  myNuma=0
fi

#Determine local IP assigned to NIC associated with GPU
mylocalIpv6=`ip -o addr  | grep ens"$myGpu"np0 | grep inet6 | grep -vE 'fe80|host' | sed -e
's/^.*inet6 \([^ ]*\)\/.
*$/\1/;t;d'`

#Iterate through Broadcom NICs to determine bnxt device name and GID used for IPv6 address
for i in `ibv_devinfo -l | grep bnxt`
  do
    ibv_devinfo -v -d $i | grep -q "$mylocalIpv6";
    if [ $? -eq 0 ];
      then
        echo "IPv6 "$mylocalIpv6" found on IB device $i"
        myBnxt=$i;
        myGid=`ibv_devinfo -v -d $myBnxt | grep "$mylocalIpv6" | awk -F "GID\[|\]" '{print
$2}'`
        echo "with GID: $myGid"
        break;
      fi;
done

while true
do
  echo "-------------------------------------------------------------------------------------
---" >> gpu$myGpu-send-server.out 2>&1
  echo "Start new test server on GPU$myGpu `date`" >> gpu$myGpu-send-server.out 2>&1
  echo "-------------------------------------------------------------------------------------
---" >> gpu$myGpu-send-server.out 2>&1
  numactl -c $myNuma ./ib_send_bw -p $myPort -n $myIters -d $myBnxt -F -x $myGid -q
$myQueues --use_rocm=$myRocm -m $my
Mtu --report_gbits --ipv6-addr  >> gpu$myGpu-send-server.out 2>&1
  sleep 1
done
```

The following example shows a client wrapper script for ib_send_bw that starts on NIC1 of server 1, connects to NIC1 on server 2 on port 18515, and sends RoCE packets using a 256-byte MTU.

```
#!/bin/bash
#Script to start ib_send_bw client, adjust vars as needed

myGpu=1                                 #Lenovo GPU Slot number
myRocm=7                                #AMD GPU OAM number
```

```
myPort=18515                              #TCP Port used for RoCEv2 test negotiation
myremoteIpv6="fd00:100:1:1:0:5:0:1001"  #Remote GPU NIC IPv6 addr
myMtu=256                                 #MTU used for RoCEv2 UDP test packets
myQueues=2                                #Number of test threads
myIters=10000000
myNuma=''
myBnxt=''
myGid=''


#Determine NUMA from GPU Slot
if [ "$myGpu" -le 4 ]; then
  myNuma=1
else
  myNuma=0
fi

#Determine local IP assigned to NIC associated with GPU
mylocalIpv6=`ip -o addr  | grep ens"$myGpu"np0 | grep inet6 | grep -vE 'fe80|host' | sed -e
's/^.*inet6 \([^ ]*\)\/.*$/\1/;t;
d'`

#Iterate through Broadcom NICs to determine bnxt device name and GID used for IPv6 address
for i in `ibv_devinfo -l | grep bnxt`
  do
    ibv_devinfo -v -d $i | grep -q "$mylocalIpv6";
    if [ $? -eq 0 ];
      then
        echo "IPv6 "$mylocalIpv6" found on IB device $i"
        myBnxt=$i;
        myGid=`ibv_devinfo -v -d $myBnxt | grep "$mylocalIpv6" | awk -F "GID\[|\]" '{print
$2}'`
        echo "with GID: $myGid"
        break;
      fi;
done

while true
do
  echo "-----------------------------------------------------------------------------------
---" >> gpu$myGpu-send-client.out 2>&1
  echo "Start new test client on GPU$myGpu `date`" >> gpu$myGpu-send-client.out 2>&1
  echo "-----------------------------------------------------------------------------------
---" >> gpu$myGpu-send-client.out 2>&1
  numactl -c $myNuma ./ib_send_bw -d $myBnxt -F -x $myGid -n $myIters -m $myMtu -q $myQueues
--use_rocm=$myRocm --report_gbit
s --bind_source_ip $mylocalIpv6 --ipv6-addr $myremoteIpv6 -p $myPort >> gpu$myGpu-send-
client.out 2>&1
  sleep 1
```

```
done
```

Output of running a perftest server using GPU memory in GPU with Bus ID: 0xa5

```
---------------------------------------------------------------------------------------
Start new test server on GPU1 Wed Jul  9 03:33:45 PM EDT 2025
---------------------------------------------------------------------------------------
 WARNING: BW peak won't be measured in this run.

************************************
* Waiting for client to connect... *
************************************
Using ROCm Device with ID: 7, Name: AMD Instinct MI300X, PCI Bus ID: 0xa5, GCN Arch:
gfx942:sramecc+:xnack-
allocated 262144 bytes of GPU buffer at 0x7fbfe4200000
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : bnxt_re7
 Number of qps   : 2            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON           Lock-free      : OFF
 ibv_wr* API     : OFF          Using DDP      : OFF
 RX depth        : 512
 CQ Moderation   : 1
 CQE Poll Batch  : 16
 Mtu             : 256[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Use ROCm memory : ON
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x2c01 PSN 0x20374a
 GID: 253:00:01:00:00:01:00:01:00:00:00:01:00:00:16:01
 local address: LID 0000 QPN 0x2c02 PSN 0xd9e3a3
 GID: 253:00:01:00:00:01:00:01:00:00:00:01:00:00:16:01
 remote address: LID 0000 QPN 0x2c01 PSN 0x77b55b
 GID: 253:00:01:00:00:01:00:01:00:00:00:05:00:00:16:01
 remote address: LID 0000 QPN 0x2c02 PSN 0x9363a9
 GID: 253:00:01:00:00:01:00:01:00:00:00:05:00:00:16:01
```

Output of running a perftest client using GPU memory in GPU Bus ID: 0xa5

```
---------------------------------------------------------------------------------------
Start new test client on GPU1 Wed Jul  9 03:35:21 PM EDT 2025
---------------------------------------------------------------------------------------
 WARNING: BW peak won't be measured in this run.
```

```
Using ROCm Device with ID: 7, Name: AMD Instinct MI300X, PCI Bus ID: 0xa5, GCN Arch:
gfx942:sramecc+:xnack-
allocated 262144 bytes of GPU buffer at 0x7fbfe4200000
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : bnxt_re7
 Number of qps   : 2            Transport type : IB
 Connection type : RC           Using SRQ      : OFF
 PCIe relax order: ON           Lock-free      : OFF
 ibv_wr* API     : OFF          Using DDP      : OFF
 TX depth        : 128
 CQ Moderation   : 1
 CQE Poll Batch  : 16
 Mtu             : 256[B]
 Link type       : Ethernet
 GID index       : 3
 Max inline data : 0[B]
 rdma_cm QPs     : OFF
 Use ROCm memory : ON
 Data ex. method : Ethernet
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x2c03 PSN 0x4336c5
 GID: 253:00:01:00:00:01:00:01:00:00:00:01:00:00:16:01
 local address: LID 0000 QPN 0x2c04 PSN 0xad2d02
 GID: 253:00:01:00:00:01:00:01:00:00:00:01:00:00:16:01
 remote address: LID 0000 QPN 0x2c03 PSN 0x9fe414
 GID: 253:00:01:00:00:01:00:01:00:00:00:05:00:00:16:01
 remote address: LID 0000 QPN 0x2c04 PSN 0xabf1a5
 GID: 253:00:01:00:00:01:00:01:00:00:00:05:00:00:16:01
```

The above sample client and server scripts should be modified to give a total of 8x server and 8x client processes on each GPU server. This loads all GPU NICs and rails simultaneously and in both directions, using a dedicated client-server TCP control port for each NIC.

The following example shows a working test directory for ib_send_bw on one GPU server using 1x wrapper script with configurable variables to start each of the 8x server and 8x client instances of ib_send_bw and a parent script to start all 8x servers and 8x clients in the background.

```
    $ ls
    4 -rwxr-xr-x 1 root root      421 Jul  2 19:01 start-all-8-rocm-servers.sh
    4 -rwxr-xr-x 1 root root      415 Jul  2 19:01 start-all-8-rocm-clients.sh
    0 lrwxrwxrwx 1 root root       13 Jul  2 19:01 ib_send_bw -> ../ib_send_bw
    4 -rwxr-xr-x 1 root root     1805 Jul  9 15:31 start_ib_send_client_rocm_gpu1.sh
. . .
4 -rwxr-xr-x 1 root root     1851 Jul  9 15:31 start_ib_send_client_rocm_gpu8.sh
    4 -rwxr-xr-x 1 root root     1674 Jul  9 15:31 start_ib_send_server_rocm_gpu1.sh
    . . .
    4 -rwxr-xr-x 1 root root     1723 Jul  9 15:31 start_ib_send_server_rocm_gpu8.sh
```

```
NUMA0
    0[    0.0%]    8[##100.0%]   16[##100.0%]   24[    0.0%]   32[    0.0%]   40[    0.0%]   48[    0.0%]   56[    0.0%]
    1[##100.0%]    9[    0.0%]   17[    0.0%]   25[    0.0%]   33[    0.0%]   41[    0.0%]   49[    0.0%]   57[    0.0%]
    2[    0.0%]   10[    0.0%]   18[    0.0%]   26[    0.0%]   34[    0.0%]   42[    0.0%]   50[    0.0%]   58[    0.0%]
    3[    0.0%]   11[    0.0%]   19[    0.0%]   27[##100.0%]   35[    0.0%]   43[    0.0%]   51[    0.0%]   59[    0.0%]
    4[    0.0%]   12[    0.0%]   20[    0.0%]   28[    0.0%]   36[    0.0%]   44[    0.0%]   52[    0.0%]   60[    0.0%]
    5[    0.0%]   13[    0.0%]   21[    0.0%]   29[    0.0%]   37[    0.0%]   45[##100.0%]   53[    0.0%]   61[    0.0%]
    6[    0.0%]   14[    0.0%]   22[    0.0%]   30[    0.0%]   38[**100.0%]   46[    0.0%]   54[##100.0%]   62[##100.0%]
    7[    0.0%]   15[    0.0%]   23[    0.0%]   31[    0.0%]   39[    0.0%]   47[    0.0%]   55[    0.0%]   63[    0.0%]

NUMA1
    64[    0.0%]   72[    0.0%]   80[    0.0%]   88[    0.0%]   96[##100.0%]  104[    0.0%]  112[    0.0%]  120[##100.0%]
    65[    0.0%]   73[    0.0%]   81[    0.0%]   89[    0.0%]   97[    0.0%]  105[    0.0%]  113[    0.0%]  121[    0.0%]
    66[    0.0%]   74[    0.0%]   82[    0.0%]   90[##100.0%]   98[    0.0%]  106[    0.0%]  114[    0.0%]  122[    0.0%]
    67[##100.0%]   75[    0.0%]   83[##100.0%]   91[    0.0%]   99[    0.0%]  107[    0.0%]  115[    0.0%]  123[    0.0%]
    68[    0.0%]   76[    0.0%]   84[    0.0%]   92[    0.0%]  100[    0.0%]  108[    0.0%]  116[    0.0%]  124[    0.0%]
    69[    0.0%]   77[    0.0%]   85[    0.0%]   93[    0.0%]  101[    0.0%]  109[    0.0%]  117[##100.0%]  125[    0.0%]
    70[    0.0%]   78[    0.0%]   86[    0.0%]   94[    0.0%]  102[    0.0%]  110[##100.0%]  118[    0.0%]  126[    0.0%]
    71[    0.0%]   79[##100.0%]   87[    0.0%]   95[    0.0%]  103[    0.0%]  111[    0.0%]  119[    0.0%]  127[    0.0%]
  Mem[||#*                                                          36.5G/2.21T]   Tasks: 101, 253 thr; 17 running
  Swp[                                                               0K/2.00G]   Load average: 15.05 8.22 3.37
                                                                                 Uptime: 4 days, 07:25:16
```

The output above shows a snapshot of htop from both servers with each GPU server running 8 ib_send_bw servers and x8 ib_send_bw clients to test bandwidth on all eight GPU rails in both directions. Note the even placement of 8x ib_send_bw pmd processes on each NUMA node with the PMD processes running at100% CPU core occupancy.

5.1.6 Local storage configuration

To run containers using local storage, a logical volume (LVM) should be created across the drives located in the SR685a front bay.

Note: These drives do not have a RAID controller installed, so any critical data should be backed up to remote online storage (see Online Storage Section).

Use this procedure to install an LVM across all 4x NVMe 3.84TB bay drives.

```
sudo apt install lvm2

sudo parted /dev/nvme0n1 mklabel gpt
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme1n1 mklabel gpt
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme2n1 mklabel gpt
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme3n1 mklabel gpt
Information: You may need to update /etc/fstab.
```

```
sudo parted /dev/nvme0n1 mkpart primary 0% 100%
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme1n1 mkpart primary 0% 100%
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme2n1 mkpart primary 0% 100%
Information: You may need to update /etc/fstab.
sudo parted /dev/nvme3n1 mkpart primary 0% 100%
Information: You may need to update /etc/fstab.


sudo pvcreate -vvv /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
...
Completed: pvcreate -vvv /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1

sudo vgcreate volgrp0 /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
  Volume group "volgrp0" successfully created

sudo lvcreate -n lvol0 -L 10T volgrp0
  Logical volume "lvol0" created.

sudo mkfs.ext4 /dev/volgrp0/lvol0
mke2fs 1.46.5 (30-Dec-2021)
Discarding device blocks: done
Creating filesystem with 2684354560 4k blocks and 335544320 inodes
Filesystem UUID: 20f71586-2612-446d-8534-0b4d5d4c8f56
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
        102400000, 214990848, 512000000, 550731776, 644972544, 1934917632,
        2560000000

Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

To ensure the LVM is mounted on boot, it is necessary to determine the UUID of the newly created volume for use in fstab.

```
$ sudo blkid /dev/volgrp0/lvol0
/dev/volgrp0/lvol0: UUID="20f71586-2612-446d-8534-0b4d5d4c8f56" BLOCK_SIZE="4096"
TYPE="ext4"
```

Using a text editor, add a line to the end of /etc/fstab with the UUID from the previous blkid output and arguments, as shown below.

```
UUID=20f71586-2612-446d-8534-0b4d5d4c8f56 /mnt/storage    ext4    defaults        0       2
```

Next, verify your entry will be mounted on boot using mount with the -a flag and check for the presence of the new logical volume.

```
$ sudo mount -a

$ df -k
Filesystem                   1K-blocks      Used   Available Use% Mounted on
tmpfs                        237770580     36568   237734012   1% /run
/dev/nvme4n1p2               921264900 756625312   117768192  87% /
tmpfs                       1188852888         0  1188852888   0% /dev/shm
tmpfs                            5120         0        5120   0% /run/lock
efivarfs                          256        94         158  38% /sys/firmware/efi/efivars
/dev/nvme4n1p1                 523248      6232      517016   2% /boot/efi
tmpfs                        237770576        60   237770516   1% /run/user/1000
/dev/mapper/volgrp0-lvol0 10651631520        28 10114744196   1% /mnt/storage
```

Use the directory /mnt/storage for local container storage location to avoid filling up the root filesystem mounted on the m.2 drives.

5.2 MLPerf configuration

The section below describes single node training methodology which adheres to the AMD MLCommons submission.

The configuration used Docker on a single GPU server to run the containerized workload. Follow the steps to install Docker on Ubuntu 22.04: https://docs.docker.com/engine/install/ubuntu/.

AMD's Llama 2 submission to MLCommons using MI325x GPUs was copied using steps outlined in the following blog and modified slightly to run on the MI300x GPUs.

https://rocm.blogs.amd.com/artificial-intelligence/mlperf-inf-4-1/README.html

Once inside the container, copy the following contents to a new file called /lab-mlperf-inference/code/harness_llm/models/llama2-70b/offline_mi300x.yaml.

```
defaults: []

# benchmark details
benchmark_name: llama2-70b
scenario: offline
test_mode: performance

env_config:
  VLLM_LOGGING_LEVEL: "ERROR"
  HARNESS_GC_LIMIT: 100000
```

```
  VLLM_LLAMA2_MLPERF_SCHED: 1
  VLLM_LLAMA2_MLPERF_MAX_TARGET_DECODE_BATCH: 1536
  VLLM_LLAMA2_MLPERF_MIN_TARGET_DECODE_BATCH: 1280
  VLLM_LLAMA2_MLPERF_STEP_DECODE_BATCH: 256

# configuration related to the LLM model.
llm_config:
  model: /model/llama2-70b-chat-hf/fp8_quantized
  tensor_parallel_size: 1
  num_scheduler_steps: 11
  quantization: fp8
  max_model_len: 2048
  swap_space: 0
  gpu_memory_utilization: 0.98
  max_seq_len_to_capture: 2048
  enforce_eager: True
  disable_custom_all_reduce: True
  max_num_batched_tokens: 40960
  max_num_seqs: 1792
  enable_chunked_prefill: False
  block_size: 32
  enable_prefix_caching: False

# configuration related to the sampling params
sampling_params:
  temperature: 0.0
  min_tokens: 1
  max_tokens: 1024
  ignore_eos: False
  detokenize: False

# configuration related to the harness tests.
harness_config:
  dataset_path: /data/processed-openorca/open_orca_gpt4_tokenized_llama.sampled_24576.pkl
  mlperf_conf_path: /app/mlperf_inference/mlperf.conf
  user_conf_path: /lab-mlperf-inference/code/user_mi300x.conf
  target_qps: -1 # 80
  total_sample_count: 24576
  output_log_dir: /app/logs
  enable_log_trace: False
  warmup_duration: 0
  enable_warm_up: True
  warm_up_sample_count_per_server: 10
  data_parallel_size: 8
```

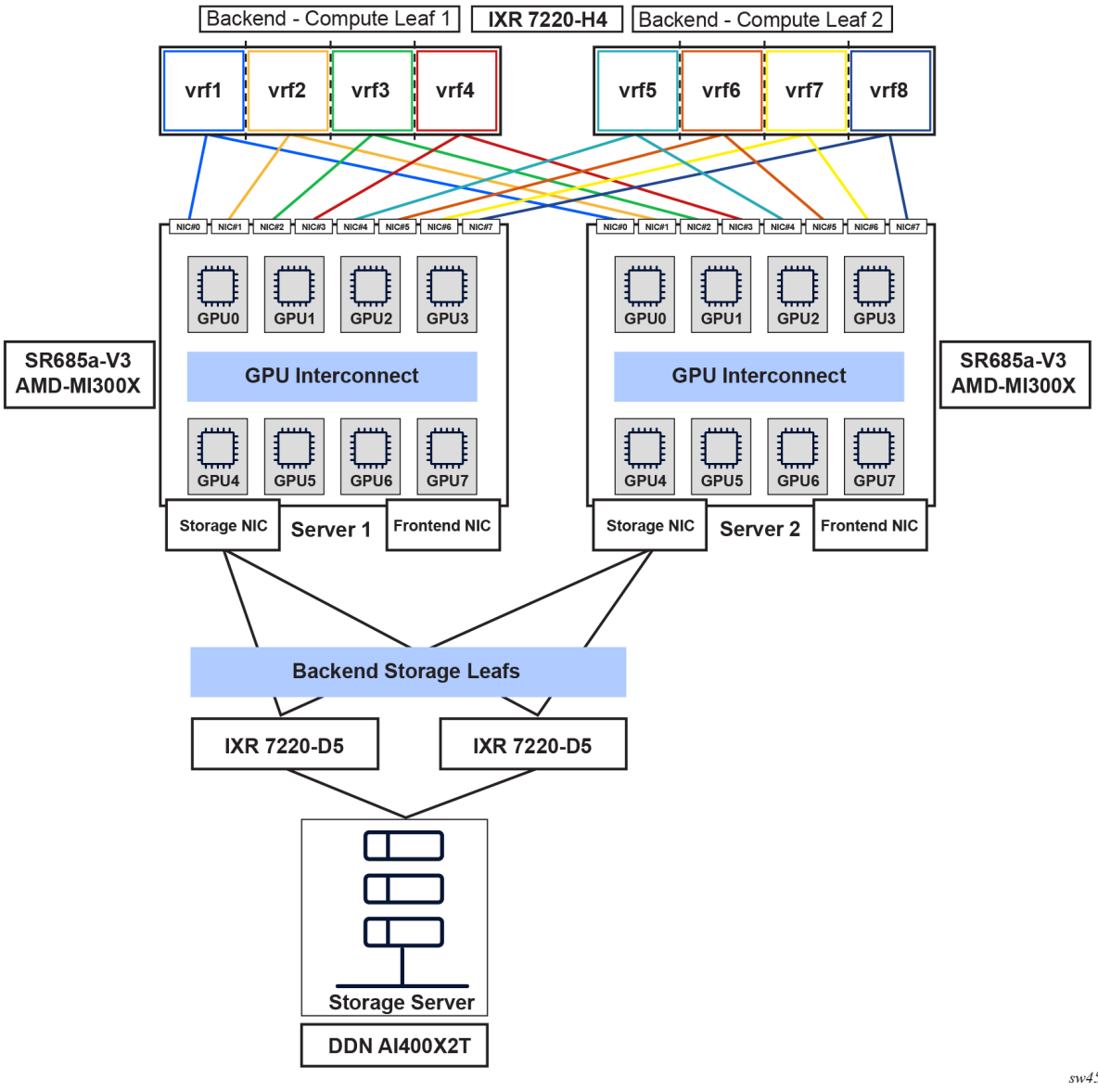
Next, copy the following contents to /lab-mlperf-inference/code/user_mi300x.conf.

```
# The format of this config file is 'key = value'.
```

```
# The key has the format 'model.scenario.key'. Value is mostly int64_t.
# Model maybe '*' as wildcard. In that case the value applies to all models.
# All times are in milliseconds
#

# LLAMA2
llama2-70b.*.performance_sample_count_override = 24576
llama2-70b.*.min_duration = 1200000
llama2-70b.Offline.target_qps = 93
llama2-70b.Server.target_qps = 85
```

5.3 Server configuration – AI400X2T

The DDN storage cluster comprises of 2x controllers each with 2x VMs running as an HA cluster. The GPU servers are connected to the storage leafs using 400 (2x200) breakout cables and the storage leafs are also connected to the DDN storage server with the same 400 (2x200) breakout cables.



For redundancy, the storage network comprises two /25 Ethernet subnets, each one served by a storage leaf.

Server 1 storage interfaces:

```
# Let networkd manage all devices on this system
network:
  version: 2
  renderer: networkd
  ethernets:
    ens10f0np0:
      dhcp4: no
```

```
      dhcp6: no
      optional: true
      addresses: [ 172.16.10.10/25 ]
      optional: true
    ens10f1np1:
      dhcp4: no
      dhcp6: no
      optional: true
      addresses: [ 172.16.10.130/25 ]
```

Server 2 storage interfaces:

```
# Let networkd manage all devices on this system
network:
  version: 2
  renderer: networkd
  ethernets:
    ens10f0np0:
      dhcp4: no
      dhcp6: no
      optional: true
      addresses: [ 172.16.10.11/25 ]
    ens10f1np1:
      dhcp4: no
      dhcp6: no
      optional: true
      addresses: [ 172.16.10.131/25 ]
    ens11f1np1:
      dhcp4: true
      dhcp6: no
      optional: true
      mtu: 1500
```

DDN storage server network settings:

```
AI400X2T Appliance
Server interface tunings none
Server interface configuration /etc/ddn/exascaler.toml
lnets = ["tcp0(mlxen0,mlxen1)"]
Network Interface || Function IP Address Netmask
c0p1 || vm00 – mlxen0 172.16.10.110/25
c0p5 || vm02 – mlxen0 172.16.10.120/25
c1p1 || vm01 – mlxen0 172.16.10.210/25
c1p5 || vm03 – mlxen0 172.16.10.220/25
```

GPU server (lustre client) settings:

```
Client Node
Client interface tunings none
Client interface configuration /etc/modprobe.d/lustre.conf
options lnet networks="tcp0(ens10f0np0,ens10f1np1)"
options libcfs cpu_npartitions=32 cpu_pattern=""
options ko2iblnd peer_credits=32 peer_credits_hiw=16 concurrent_sends=64
```

Leaf configuration:

```
   interface ethernet-1/1 {
        breakout-mode {
            num-breakout-ports 2
            breakout-port-speed 200G
        }
    }

  interface ethernet-1/1/1 {
        admin-state enable
        vlan-tagging true
        subinterface 4096 {
            type bridged
            description untagged-storage
            admin-state enable
            vlan {
                encap {
                    untagged {
                    }
                }
            }
        }
    }
interface ethernet-1/1/2 {
        admin-state enable
        vlan-tagging true
        subinterface 4096 {
            type bridged
            description untagged-storage
            admin-state enable
            vlan {
                encap {
                    untagged {
                    }
                }
            }
        }
    }
```

```
    interface ethernet-1/5 {
        breakout-mode {
            num-breakout-ports 2
            breakout-port-speed 200G
        }
    }

  interface ethernet-1/5/1 {
        admin-state enable
        vlan-tagging true
        subinterface 4096 {
            type bridged
            description untagged-storage
            admin-state enable
            vlan {
                encap {
                    untagged {
                    }
                }
            }
        }
    }
interface ethernet-1/5/2 {
        admin-state enable
        vlan-tagging true
        subinterface 4096 {
            type bridged
            description untagged-storage
            admin-state enable
            vlan {
                encap {
                    untagged {
                    }
                }
            }
        }
    }

info network-instance storage
    network-instance storage {
        type mac-vrf
        admin-state enable
        description storage
        interface ethernet-1/1/1.4096 {
        }
        interface ethernet-1/1/2.4096 {
        }
   interface ethernet-1/5/1.4096 {
```

```
        }
        interface ethernet-1/5/2.4096 {
        }

        }
        bridge-table {
            mac-learning {
                admin-state enable
                aging {
                    admin-state enable
                    age-time 300
                }
            }
            mac-duplication {
                admin-state enable
                monitoring-window 3
                num-moves 5
                hold-down-time 9
                action stop-learning
            }
        }
    }
```

5.3.1 DDN storage cluster configuration and mounting

The GPU servers access the storage node using a Lustre filesystem Linux module that is compiled and installed on the Ubuntu OS using DDN's installation tarball.

See section 7.2.1 "A3I Storage Appliances 6.3.1 Installation and Administration Guide" available on request from DDN.

```
server$ ls /scratch/EXAScaler-6.3.1
...
exa-client-6.3.1.tar.gz
client$ wget http://<EMF_IP>:7080/exa-client-6.3.1.tar.gz
client$ tar xzf exa-client-6.3.1.tar.gz
client$ cd exa-client
client$ ls
exa_client_deploy.py
exa_client_performance_validation_*.tar.gz
exa_client_performance_validation_networkavail_*.tar.gz
fscrypt
lipe
lustre-source.tar.gz
README
```

The above installation compiles and installs the Lustre kernel module into the running kernel.

```
lsmod | grep lustre
lustre               1183744  0
mdc                   290816  1 lustre
lov                   389120  2 mdc,lustre
lmv                   241664  1 lustre
ptlrpc               1585152  7 fld,osc,fid,lov,mdc,lmv,lustre
obdclass             3530752  8 fld,osc,fid,ptlrpc,lov,mdc,lmv,lustre
lnet                  671744  6 osc,obdclass,ptlrpc,ksocklnd,lmv,lustre
libcfs                565248  11
fld,lnet,osc,fid,obdclass,ptlrpc,ksocklnd,lov,mdc,lmv,lustre
```

The storage filesystem is mounted using a mountpoint that is managed by the Lustre kernel module using all 4x of its VMs on the storage server.

```
mount -v -t lustre \
172.16.10.110@tcp0,172.16.10.120@tcp0,172.16.10.210@tcp0,172.16.10.220@tcp0:ddnfs01 /ai400
```

To map a container to use the remote storage using docker:

```
$ docker run -v /ai400/myAIdataset/:/ai400/container_in_path \
--rm -ti nvcr.io/nvidia/cuda:latest \
/bin/bash
```

5.4 Server configuration – SR630-v2 – Frontend

Three-controller node K8s cluster setup for High Availability

Update the firmware using XClarity One on all the SR630 servers and install Ubuntu Server LTS as documented in Section 5.1.

Note: The BIOS profile for the servers comprising the frontend cluster should be set to MAX_EFFICIENCY and Ubuntu command-line left with its default configuration.

Next, start installing K8s on each of the three controllers.

Kubernetes can be configured in many different ways based on use cases. The following is an example of three-controller node configuration for High Availability. Canonical Kubernetes snap is used in this case for easy deployment and management of the cluster. For more information, see Canonical Kubernetes documentation https://documentation.ubuntu.com/canonical-kubernetes/latest/

Section 1: K8s installation

First control-plane node:

Install K8s with snap and bootstrap the cluster to get the first controller running:

```
sudo snap install k8s --classic --channel=1.33-classic/stable sudo k8s bootstrap
```

```
To check status of the cluster:

sudo k8s status --wait-ready
```

Second and third control-plane node:

Install K8s with snap once again. These two nodes will be joined with the already bootstrapped cluster.

```
sudo snap install k8s --classic --channel=1.33-classic/stable
```

For more information about this section, see “Getting started”:
https://documentation.ubuntu.com/canonical-kubernetes/latest/snap/tutorial/getting-started/

Section 2: Add or remove nodes

Adding second and third control-plane nodes to the cluster:

To add more controller and worker nodes, tokens are used. To add a controller, a generate control-plane token command is used.

Run the following command on the first control-plane node that hosts the bootstrapped cluster:

```
sudo k8s get-join-token control-plane
```

Use the token on the second and third control-plane node with a join command:

```
sudo k8s join-cluster <token>
```

Adding worker nodes to cluster:

To add a worker node, a generate worker token command is used.

Run the following command on the first control-plane node that hosts the bootstrapped cluster:

```
sudo k8s get-join-token --worker
```

Next, use the token on the worker node with a join command:

```
sudo k8s join-cluster <token>
```

Verification of correct node setup within the cluster can be done with the following command:

```
sudo k8s kubectl get nodes
```

Remove nodes

To remove nodes from the worker or control-plane, the following commands are used:

```
sudo k8s remove-node worker

sudo k8s remove-node control-plane
```

For more information about this section, see “Add and remove nodes”:
https://documentation.ubuntu.com/canonical-kubernetes/latest/snap/tutorial/add-remove-nodes/

Section 3: Tainting nodes

The default behavior of a canonical setup is a control plane + worker combination. To isolate the control-plane nodes, a taint needs to be set on the nodes using the following command:

```
sudo k8s kubectl taint node node1 node-role.kubernetes.io/control-plane:NoSchedule
```

Example:

```
$ sudo k8s kubectl taint node sr630v2-svr1 node-role.kubernetes.io/control-plane:NoSchedule

$ sudo k8s kubectl taint node sr630v2-svr2 node-role.kubernetes.io/control-plane:NoSchedule

$ sudo k8s kubectl taint node sr630v2-svr3 node-role.kubernetes.io/control-plane:NoSchedule
```

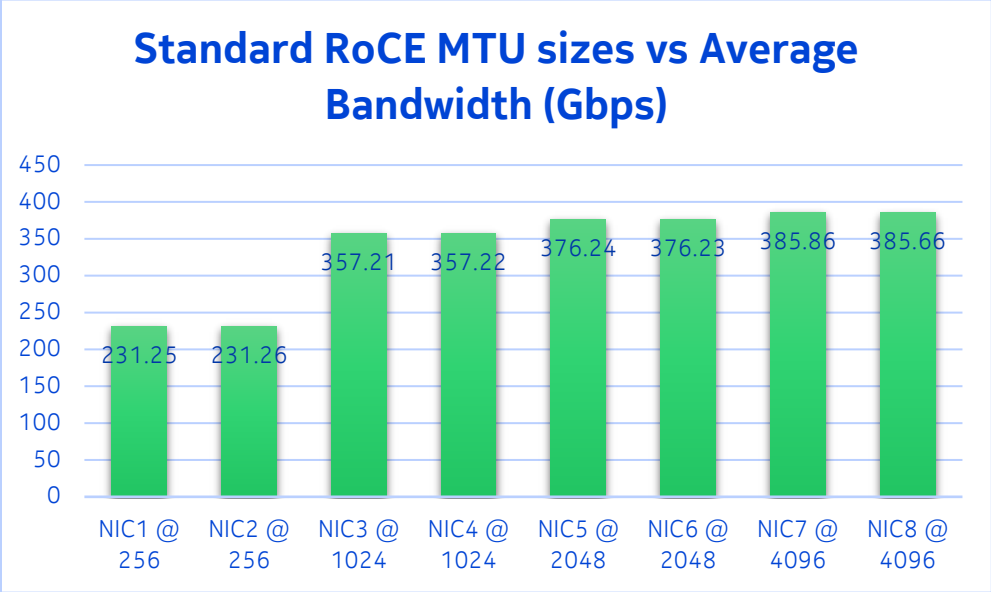
To check which nodes are tainted, use the following command:

```
$ sudo k8s kubectl get nodes -o custom-columns=NAME:.metadata.name,TAINTS:.spec.taints NAME
TAINTS

sr630v2-svr1 [map[effect:NoSchedule key:node-role.kubernetes.io/control-plane]
map[effect:NoSchedule key:dedicated value:control-plane]]

sr630v2-svr2 [map[effect:NoSchedule key:node-role.kubernetes.io/control-plane]
map[effect:NoSchedule key:dedicated value:control-plane]]

sr630v2-svr3 [map[effect:NoSchedule key:node-role.kubernetes.io/control-plane]
map[effect:NoSchedule key:dedicated value:control-plane]]
```

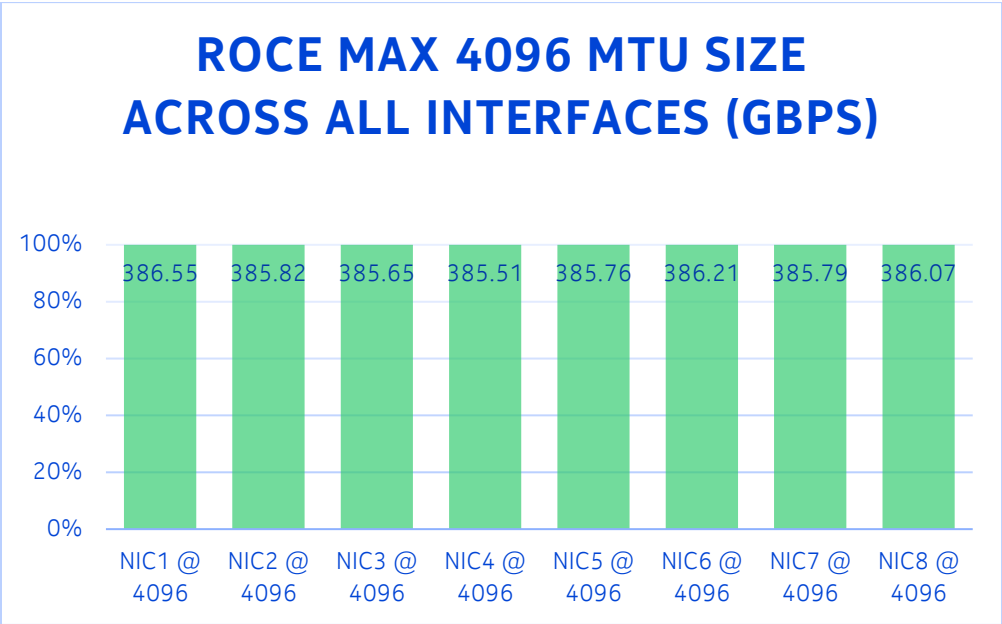
For more information about node roles and control plane isolation, see “Node roles”: https://documentation.ubuntu.com/canonical-kubernetes/latest/snap/explanation/roles

6 Validation and performance testing

6.1 Perftest and MLperf tests



Perftest results for ib_send_bw showing average bi-directional bandwidth achieved across 400Gb interfaces with the standard RoCE packet sizes from 256 to 4096 bytes



Perftest results for ib_send_bw showing average bi-directional bandwidth achieved across 400Gb interfaces, all using the maximum RoCE 4096-byte MTU size

Note: All tests were bi-directional and so the same bandwidth is reported in both directions using 2x instances of ib_send_bw. Due to reporting nature of ib_send_bw, the bandwidth reported does not factor in the RoCE ACKs in the uplink direction of the test instance running in the opposite direction and therefore slightly underreports the actual bandwidth on the link.

6.2 EDA configuration validation

To ensure the proper functioning of the AI Backend application, it is essential to conduct system-wide sanity checks and design or configure the following objects.

- **Node and Interface Labels:** Label selectors are utilized to filter and select key-value pairs assigned to TopoNodes, TopoLinks, and other fabric instances based on specific criteria, such as their functional roles within the network. The AI Backend application employs labels to match and filter objects accordingly.
- **Policy Deployment**: PolicyDeploymentSpecs specify the desired state of QoS policies to be applied to edge ports that are members of a rail.
- **Interfaces:** The interface configuration encompasses various properties, including enabling or disabling the interface, assigning descriptions, specifying interface types, configuring VLAN encapsulation, and setting parameters for Ethernet or LAG interfaces.

6.2.1 Configuration prerequisites

Given the nature of the EDA platform, validation of these system-wide objects is mandatory to confirm that the AI Backend features are correctly configured.

Each node designated as either a rail or stripe connector must possess a label that corresponds accurately to its designated role within the node profile.

```
# kubectl get toponodes -n clab-ai-fabric-1tier  -L eda.nokia.com/role -L
eda.nokia.com/stripe-id
NAME       PLATFORM     VERSION     OS  ONBOARDED MODE    NPP     NODE     AGE   ROLE
STRIPE-ID
1tier-rail-01 7220 IXR-H4   25.3.1  srl   true  normal   Connected Synced   16m  leaf
   stripe1
1tier-rail-02 7220 IXR-H4   25.3.1  srl   true  normal   Connected Synced   16m  leaf
   stripe1
```

Node specifications

```
apiVersion: core.eda.nokia.com/v1
kind: TopoNode
metadata:
  labels:
    containerlab: managedSrl
    eda-connector.nokia.com/topology: ai-fabric-1tier
    eda.nokia.com/role: rail
    eda.nokia.com/security-profile: managed
    eda.nokia.com/stripe-id: stripe1
  name: 1tier-rail-01
  namespace: clab-ai-fabric-1tier
  annotations: {}
```

AI Backends App specifications

```
[…]
  stripes:
    - gpuVlan: 1
      name: stripe01
      nodeSelector:
        - eda.nokia.com/stripe-id=stripe1
      stripeID: 1
  systemPoolIPV4: system0
[…]
```

Interfaces generated during the topology creation process are required to have labels that match those configured within the AI Backend application.

```
# kubectl get interfaces -n clab-ai-fabric-1tier -L eda.nokia.com/tenant-id
NAME                       ENABLED   OPERATIONAL STATE   SPEED   LAST CHANGE   AGE
TENANT-ID
1tier-rail-01-ethernet-1-1   true    up                  400G      30m         31m
fintecdpt
1tier-rail-01-ethernet-1-2   true    up                  400G      30m         31m
fintecdpt
1tier-rail-01-ethernet-1-3   true    up                  400G      30m         31m
fintecdpt
1tier-rail-01-ethernet-1-4   true    up                  400G      30m         31m
fintecdpt
```

Interface specifications

```
apiVersion: interfaces.eda.nokia.com/v1alpha1
kind: Interface
metadata:
  labels:
```

```
    eda.nokia.com/role: edge
    eda.nokia.com/tenant-id: fintecdpt
  name: 1tier-rail-01-ethernet-1-1
  namespace: clab-ai-fabric-1tier
  annotations: {}
[…]
```

AI Backends App specifications

```
[…]
spec:
  asnPool: asn-pool
  gpuIsolationGroups:
    - interfaceSelector:
        - eda.nokia.com/tenant-id=fintecdpt
      name: group-fintecdpt
```

Interfaces that are members of an isolation group must be configured to utilize dot1q tagging.

```
# kubectl get interfaces -n clab-ai-fabric-1tier -o yaml
[…]
  spec:
    description: edge link to 1tier-gpu-01
    enabled: true
    encapType: dot1q
```

Edge port quality of service (QoS) deployment settings must be explicitly specified within the EDA manifests.

```
apiVersion: qos.eda.nokia.com/v1alpha1
kind: PolicyDeployment
metadata:
  name: ai-fabric-edge-qos
  namespace: clab-ai-fabric-1tier
  labels: {}
  annotations: {}
spec:
  egressPolicy: egress-backend-ai-backend-fabric
  ingressPolicy: ingress-backend-ai-backend-fabric
  interfaceSelector: eda.nokia.com/tenant-id=fintecdpt
  interfaceType: ACCESS
  nodeSelector: eda.nokia.com/role=rail
  node: ''
  interfaces: []
```

The IP MTU settings for each interface must be configured to accommodate jumbo frames, thereby ensuring optimal performance of the fabric nodes.

```
A:root@1tier-rail-01# info interface ethernet-1/1 subinterface 1
    !!! EDA Source CRs: aifabrics.eda.nokia.com/v1alpha1/Backend/ai-backend-fabric
    type routed
    description "Backend: ai-backend-fabric Stripe: stripe01"
    admin-state enable
    ip-mtu 9394
    ipv6 {
        admin-state enable
        address fd00:1:2:1:0:1:0:1/96 {
            primary
        }
        router-advertisement {
            router-role {
                admin-state enable
                current-hop-limit 64
                managed-configuration-flag false
                other-configuration-flag false
                max-advertisement-interval 600
                min-advertisement-interval 200
                reachable-time 0
                retransmit-time 0
                router-lifetime 1800
            }
        }
    }
    vlan {
        encap {
            single-tagged {
                vlan-id 1
            }
        }
    }
```

6.2.2 Configuration validation

These configuration verifications encompass not only objects related to the AI Backends application but also overall system-wide integrity checks.

Use the following command to verify if TopoNodes are onboarded/connected/synced.

```
# kubectl get toponodes -n clab-ai-fabric-1tier
NAME           PLATFORM      VERSION   OS    ONBOARDED   MODE     NPP          NODE     AGE
1tier-rail-01   7220 IXR-H4   25.3.1    srl   true        normal   Connected   Synced   9h
1tier-rail-02   7220 IXR-H4   25.3.1    srl   true        normal   Connected   Synced   9h
```

Use the following command to verify if backends are up and running.

```
# kubectl get backends -n clab-ai-fabric-1tier -o yaml
apiVersion: v1
items:
- apiVersion: aifabrics.eda.nokia.com/v1alpha1
  kind: Backend
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
    generation: 1
    name: ai-backend-fabric
    namespace: clab-ai-fabric-1tier
    resourceVersion: "12757250"
    uid: b4371070-ca68-4ff2-bbb9-c3478c461076
  spec:
    asnPool: asn-pool
    gpuIsolationGroups:
    - interfaceSelector:
      - eda.nokia.com/tenant-id=fintecdpt
      name: group-fintecdpt
    - interfaceSelector:
      - eda.nokia.com/tenant-id=pmodpt
      name: group-pmodpt
    rocev2QoS:
      ecnMaxDropProbabilityPercent: 100
      ecnSlopeMaxThresholdPercent: 80
      ecnSlopeMinThresholdPercent: 5
      pfcDeadlockDetectionTimer: 750
      pfcDeadlockRecoveryTimer: 750
      queueMaximumBurstSize: 52110640
    stripes:
    - gpuVlan: 1
      name: stripe01
      nodeSelector:
      - eda.nokia.com/stripe-id=stripe1
      stripeID: 1
    systemPoolIPV4: system0
  status:
    lastChange: "2025-08-23T18:58:38.000Z"
    operationalState: up
    stripes:
    - leafNodes:
      - node: 1tier-rail-02
        operatingSystem: srl
        operatingSystemVersion: 25.3.1
      - node: 1tier-rail-01
        operatingSystem: srl
```

```
        operatingSystemVersion: 25.3.1
      name: stripe01
```

Check the DCQCN parameters applied to edge ports.

```
info flat qos interfaces
set / qos interfaces interface ethernet-1/1 interface-ref interface ethernet-1/1
set / qos interfaces interface ethernet-1/1 pfc pfc-mapping-profile ai-fabric-edge-qos
set / qos interfaces interface ethernet-1/1 pfc pfc-enable true
set / qos interfaces interface ethernet-1/1 input pfc-buffer-allocation-profile ingress-
backend-ai-backend-fabric
set / qos interfaces interface ethernet-1/1 output buffer-allocation-profile egress-backend-
ai-backend-fabric
set / qos interfaces interface ethernet-1/1 output queues queue unicast-3 queue-management-
profile egress-backend-ai-backend-fabric-2
set / qos interfaces interface ethernet-1/1 output scheduler scheduler-policy egress-
backend-ai-backend-fabric    }
```

Check interface statistics for ECN-marked packets.

```
info from state qos interfaces interface ethernet-1/1 output queues queue unicast-3 queue-
statistics aggregate-statistics
    transmitted-packets 0
    transmitted-octets 0
    dropped-packets 0
    dropped-octets 0

info from state qos interfaces interface ethernet-1/1 output  queues queue unicast-3 active-
queue-management wred-slope all drop-probability * enable-ecn true
    wred-slope all drop-probability low enable-ecn true {
    }
    wred-slope all drop-probability medium enable-ecn true {
    }
    wred-slope all drop-probability high enable-ecn true {
    }

--{ running }--[  ]-
```

PFC buffer reservation.

```
info from state /platform linecard 1 forwarding-complex 0  buffer-memory
    platform {
        linecard 1 {
            forwarding-complex 0 {
                buffer-memory {
                    used 0
                    free 113613184
                    reserved 5566664
                }
```

```
            }
        }
    }
}
```

Monitor buffer-memory.

```
#monitor on-change platform linecard 1 forwarding-complex 0 buffer-memory
[2025-08-24 05:41:36.193852]: update /platform/linecard[slot=1]/forwarding-
complex[name=0]/buffer-memory/used:0
[2025-08-24 05:41:36.193852]: update /platform/linecard[slot=1]/forwarding-
complex[name=0]/buffer-memory/free:113613184
[2025-08-24 05:41:36.193852]: update /platform/linecard[slot=1]/forwarding-
complex[name=0]/buffer-memory/reserved:5566664lidation
```

Monitor PFC frames.

```
#monitor on-change qos interfaces interface * pfc statistics pfc-priority 6 pfc-pause-
frames-generated
[2025-08-24 05:44:06.079278]: update /qos/interfaces/interface[interface-id=ethernet-
1/1]/pfc/statistics/pfc-priority[index=6]/pfc-pause-frames-generated:0
```

6.3 RoCEv2 performance validation

The RoCE statistics from the Broadcom NICs can be obtained from sysfs.

Example. To view the RoCE stats for the device named bnxt_re0

```
cat /sys/kernel/debug/bnxt_re/bnxt_re0/info
bnxt_re debug info:
=====[ IBDEV bnxt_re0 ]============================
        link state: UP
        Max QP:         131073
        Max SRQ:        65536
        Max CQ:         262144
        Max MR:         262144
        Max MW:         262144
        Max AH:         262144
        Max PD:         131072
        Active QP:      1
        Active RC QP:   0
        Active UD QP:   0
        Active SRQ:     0
        Active CQ:      1
```

```
        Active MR:      0
        Active DMABUF MR: 0
        Active MW:      0
        Active AH:      0
        Active HW AH:   0
        Active PD:      1
        QP Watermark:   3
        RC QP Watermark: 2
        UD QP Watermark: 0
        SRQ Watermark:  0
        CQ Watermark:   3
        MR Watermark:   1
        DMABUF MR Watermark: 0
        MW Watermark:   0
        AH Watermark:   0
        AH HW Watermark:        0
        PD Watermark:   2
        Resize CQ count: 0
        Recoverable Errors: 0
        Rx Pkts: 149760000000
        Rx Bytes: 625697280000000
        Tx Pkts: 43316538316
        Tx Bytes: 3725222295176
        CNP Tx Pkts: 0
        CNP Rx Pkts: 0
        RoCE Only Rx Pkts: 149760000000
        RoCE Only Rx Bytes: 625697280000000
        RoCE Only Tx Pkts: 43316538316
        RoCE Only Tx Bytes: 3725222295176
        rx_roce_error_pkts: 0
        rx_roce_discard_pkts: 0
        tx_roce_error_pkts: 0
        tx_roce_discards_pkts: 0
        res_oob_drop_count: 0
        tx_atomic_req: 0
        rx_atomic_req: 0
        tx_read_req: 0
        tx_read_resp: 0
        rx_read_req: 0
        rx_read_resp: 0
        tx_write_req: 0
        rx_write_req: 0
        tx_send_req: 0
        rx_send_req: 149760000000
        rx_good_pkts: 149760000000
        rx_good_bytes: 625697280000000
        rx_dcn_payload_cut: 0
        te_bypassed: 0
```

```
        rx_ecn_marked_pkts: 0
        max_retry_exceeded: 0
        to_retransmits: 0
        seq_err_naks_rcvd: 0
        rnr_naks_rcvd: 0
        missing_resp: 0
        dup_req: 0
        unrecoverable_err: 0
        bad_resp_err: 0
        local_qp_op_err: 0
        local_protection_err: 0
        mem_mgmt_op_err: 0
        remote_invalid_req_err: 0
        remote_access_err: 0
        remote_op_err: 0
        res_exceed_max: 0
        res_length_mismatch: 0
        res_exceeds_wqe: 0
        res_opcode_err: 0
        res_rx_invalid_rkey: 0
        res_rx_domain_err: 0
        res_rx_no_perm: 0
        res_rx_range_err: 0
        res_tx_invalid_rkey: 0
        res_tx_domain_err: 0
        res_tx_no_perm: 0
        res_tx_range_err: 0
        res_irrq_oflow: 0
        res_unsup_opcode: 0
        res_unaligned_atomic: 0
        res_rem_inv_err: 0
        res_mem_error64: 0
        res_srq_err: 0
        res_cmp_err: 0
        res_invalid_dup_rkey: 0
        res_wqe_format_err: 0
        res_cq_load_err: 0
        res_srq_load_err: 0
        res_tx_pci_err: 0
        res_rx_pci_err: 0
        res_oos_drop_count: 0
        num_irq_started : 1
        num_irq_stopped : 0
        poll_in_intr_en : 0
        poll_in_intr_dis : 0
        cmdq_full_dbg_cnt : 0
        fw_service_prof_type_sup : 1
        dbq_int_recv: 0
```

```
        dbq_pacing_resched: 0
        dbq_pacing_complete: 0
        dbq_pacing_alerts: 0
        dbq_dbr_fifo_reg: 0x7fff8001
        latency_slab [0 - 1] sec = 6578
```

Verify any error or discard counts after any performance runs and take note of ‘cnp’ counts that are an indication of congestion.

Please refer to the online Broadcom tech docs for more information on their RoCEv2 statistics https://techdocs.broadcom.com/us/en/storage-and-ethernet-connectivity/ethernet-nic-controllers/bcm957xxx/adapters/statistics.html

7 Mlcommons model benchmarking

The table below shows the Mlperf benchmarking values for this validated design

Workload	Number of accelerators	Performance	Hardware
Inference Llama2	8	25020.5 tokens/sec (throughput)	AMD MI 300X accelerators, Lenovo SR685a server, Nokia IXR 7220-H4 switch
Inference Llama2	16	50594.5 tokens/sec (throughput)	
Training Llama2 70B parameters	16	16.257 minutes (latency)	

Snippets from the training and inference runs are given below of the training and inference runs.

Training

```
STARTING TIMING RUN AT 2025-08-18 09:34:20 PM
W0818 21:34:24.989000 3722 site-packages/torch/distributed/run.py:792]
W0818 21:34:24.989000 3722 site-packages/torch/distributed/run.py:792]
*****************************************
W0818 21:34:24.989000 3722 site-packages/torch/distributed/run.py:792] Setting
OMP_NUM_THREADS environment variable for each process to be 1 in default, to avoid your
system being overloaded, please further tune the variable for optimal performance in your
application as needed.
W0818 21:34:24.989000 3722 site-packages/torch/distributed/run.py:792]
*****************************************
FlashAttention Installed
FlashAttention Installed
Initializing distributed: GLOBAL_RANK: 1, MEMBER: 2/16
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
HPU available: False, using: 0 HPUs
```

```
`Trainer(limit_train_batches=1.0)` was configured so 100% of the batches per epoch will be
used..
`Trainer(limit_val_batches=1.0)` was configured so 100% of the batches will be used..
25-08-18 21:34:54 - PID:3788 - rank:(0, 0, 0, 0) - microbatches.py:39 - INFO - setting
number of micro-batches to constant 1
Initializing distributed: GLOBAL_RANK: 7, MEMBER: 8/16
Initializing distributed: GLOBAL_RANK: 2, MEMBER: 3/16
Initializing distributed: GLOBAL_RANK: 6, MEMBER: 7/16
Initializing distributed: GLOBAL_RANK: 4, MEMBER: 5/16
You are using a CUDA device ('AMD Instinct MI300X') that has Tensor Cores. To properly
utilize them, you should set `torch.set_float32_matmul_precision('medium' | 'high')` which
will trade-off precision for performance. For more details, read
https://pytorch.org/docs/stable/generated/torch.set_float32_matmul_precision.html#torch.set_
float32_matmul_precision
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
----------------------------------------------------------------------------------------------
--------
distributed_backend=nccl
All distributed processes registered. Starting with 16 processes
----------------------------------------------------------------------------------------------
--------


Initializing distributed: GLOBAL_RANK: 3, MEMBER: 4/16
Initializing distributed: GLOBAL_RANK: 5, MEMBER: 6/16
RCCL version : 2.22.3-HEAD:271856d
HIP version  : 6.4.43482-0f2d60242
ROCm version : 6.4.0.0-47-73dae9c
Hostname     : sr685v3-svr1
Librccl path : /opt/rocm/lib/librccl.so.1


Loading distributed checkpoint with TensorStoreLoadShardedStrategy
make: Entering directory
'/workspace/deps/nemo/nemo/collections/nlp/data/language_modeling/megatron'
g++ -O3 -Wall -shared -std=c++11 -fPIC -fdiagnostics-color -
I/opt/conda/envs/py_3.10/include/python3.10 -I/opt/conda/envs/py_3.10/lib/python3.10/site-
packages/pybind11/include helpers.cpp -o helpers.cpython-310-x86_64-linux-gnu.so
make: Leaving directory
'/workspace/deps/nemo/nemo/collections/nlp/data/language_modeling/megatron'
> building indices for blendable datasets ...
 > sample ratios:
   dataset 0, input: 1, achieved: 1
LOCAL_RANK: 1 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 4 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 3 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 6 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 7 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 5 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 2 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]


  | Name  | Type           | Params | Mode
```

```
-----------------------------------------------
0 | model | Float16Module | 69.0 B | train
-----------------------------------------------
44.6 M    Trainable params
69.0 B    Non-trainable params
69.0 B    Total params
276,084.851Total estimated model params size (MB)
2490      Modules in train mode
0         Modules in eval mode
:::MLLOG {"namespace": "", "time_ms": 1755553043758, "event_type": "POINT_IN_TIME", "key":
"cache_clear", "value": true, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 233}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "INTERVAL_START", "key":
"init_start", "value": null, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 234}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"submission_benchmark", "value": "llama2_70b_lora", "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 235}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"submission_org", "value": "MangoBoost", "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 235}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"submission_division", "value": "closed", "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 235}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"submission_status", "value": "onprem", "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 235}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"submission_platform", "value": "2xMI300X", "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 235}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"seed", "value": 1, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 236}}
:::MLLOG {"namespace": "", "time_ms": 1755553043759, "event_type": "POINT_IN_TIME", "key":
"global_batch_size", "value": 8, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 242}}
:::MLLOG {"namespace": "", "time_ms": 1755553044187, "event_type": "POINT_IN_TIME", "key":
"train_samples", "value": 3901, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 247}}
:::MLLOG {"namespace": "", "time_ms": 1755553044207, "event_type": "POINT_IN_TIME", "key":
"eval_samples", "value": 173, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 251}}
:::MLLOG {"namespace": "", "time_ms": 1755553044207, "event_type": "POINT_IN_TIME", "key":
"lora_alpha", "value": 32, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 272}}
RCCL version : 2.22.3-HEAD:271856d
HIP version  : 6.4.43482-0f2d60242
ROCm version : 6.4.0.0-47-73dae9c
Hostname     : sr685v3-svr1
Librccl path : /opt/rocm/lib/librccl.so.1
RCCL version : 2.22.3-HEAD:271856d
HIP version  : 6.4.43482-0f2d60242
ROCm version : 6.4.0.0-47-73dae9c
Hostname     : sr685v3-svr1
```

```
Librccl path : /opt/rocm/lib/librccl.so.1
RCCL version : 2.22.3-HEAD:271856d
HIP version  : 6.4.43482-0f2d60242
ROCm version : 6.4.0.0-47-73dae9c
Hostname     : sr685v3-svr1
Librccl path : /opt/rocm/lib/librccl.so.1

:::MLLOG {"namespace": "", "time_ms": 1755554098394, "event_type": "INTERVAL_END", "key":
"eval_stop", "value": null, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 222,
"samples_count": 3072}}
:::MLLOG {"namespace": "", "time_ms": 1755554098401, "event_type": "INTERVAL_END", "key":
"run_stop", "value": null, "metadata": {"file":
"/workspace/mlperf_training/src/callbacks/custom_callbacks.py", "lineno": 185,
"samples_count": 3072, "status": "success", "duration": "975.4288368225098 sec ->
16.25714728037516 minutes"}}
ENDING TIMING RUN AT 2025-08-18 09:56:04 PM
RESULT,LLM_FINETUNING,,1304,AMD,2025-08-18 09:34:20 PM
Config shell script: config_MI300X_2x8x1.sh
Starting multi-node MLPerf training benchmark...
Master address: 172.28.1.112
Running in multi-node mode...
MLPerf training benchmark completed.
```

Inference

```
================================================
MLPerf Results Summary
================================================
SUT name : Multi-Node SUT: Network SUT, Network SUT
Scenario : Offline
Mode     : PerformanceOnly
Samples per second: 167.795
Tokens per second: 50594.5
Result is : VALID
  Min duration satisfied : Yes
  Min queries satisfied : Yes
  Early stopping satisfied: Yes

================================================
Additional Stats
================================================
Min latency (ns)                : 60733958537
Max latency (ns)                : 4393940629827
Mean latency (ns)               : 2199546666031
50.00 percentile latency (ns)   : 2200430454915
90.00 percentile latency (ns)   : 3901171782846
95.00 percentile latency (ns)   : 4111010403044
97.00 percentile latency (ns)   : 4188823042306
99.00 percentile latency (ns)   : 4267032919704
99.90 percentile latency (ns)   : 4347671690590
```

```
================================================
Test Parameters Used
================================================
samples_per_query : 728640
target_qps : 184
ttft_latency (ns): 2000000000
tpot_latency (ns): 200000000
max_async_queries : 1
min_duration (ms): 3600000
max_duration (ms): 0
min_query_count : 1
max_query_count : 0
qsl_rng_seed : 1780908523862526354
sample_index_rng_seed : 14771362308971278857
schedule_rng_seed : 18209322760996052031
accuracy_log_rng_seed : 0
accuracy_log_probability : 0
accuracy_log_sampling_target : 0
print_timestamps : 0
performance_issue_unique : 0
performance_issue_same : 0
performance_issue_same_index : 0
performance_sample_count : 24576
WARNING: sample_concatenate_permutation was set to true.
Generated samples per query might be different as the one in the setting.
Check the generated_samples_per_query line in the detailed log for the real
samples_per_query value
```

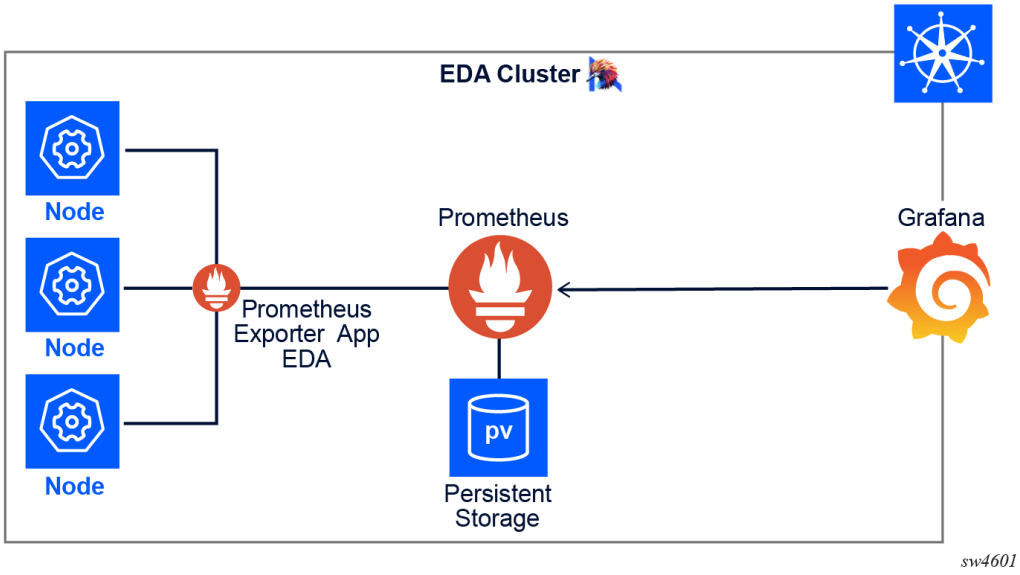
8 Telemetry

8.1 Introduction

In AI datacenter networks, real-time visibility and proactive monitoring are essential to ensure performance, reliability, and scalability. Telemetry provides live streaming of data from the networking devices, which enables operators to gain deeper insights into network health and optimize resources. EDA leverages gNMI to subscribe to datacenter switches, EDA's Prometheus-Exporter App exports the metrics to the EDA service endpoint, from where any tool such as Prometheus can collect the data by HTTP scrape and store in its own time-series database. Grafana is integrated as the visualization layer for dashboarding.

The telemetry stack is deployed using Helm charts, which bundle all required components such as ConfigMaps, templates, and configuration files. Both Prometheus and Grafana come with preconfigured settings, ensuring seamless integration and minimal manual intervention. If the stack is installed within the EDA Kubernetes cluster, the monitoring and visualization services are automatically provisioned and ready to use.

8.2 Architecture



8.3 Prometheus Exporter App

The app lets us configure the metrics that we wish to export to an external system like Prometheus by defining EDB path (which is also called jsPath). The same path can be used or extracted from the EDA UI > Queries.

The Prometheus Exporter App also lets us customize the following:

- Renaming the metric names.
- Adding static and dynamic labels.
- Mapping non-numeric values to Prometheus-compatible numeric values.

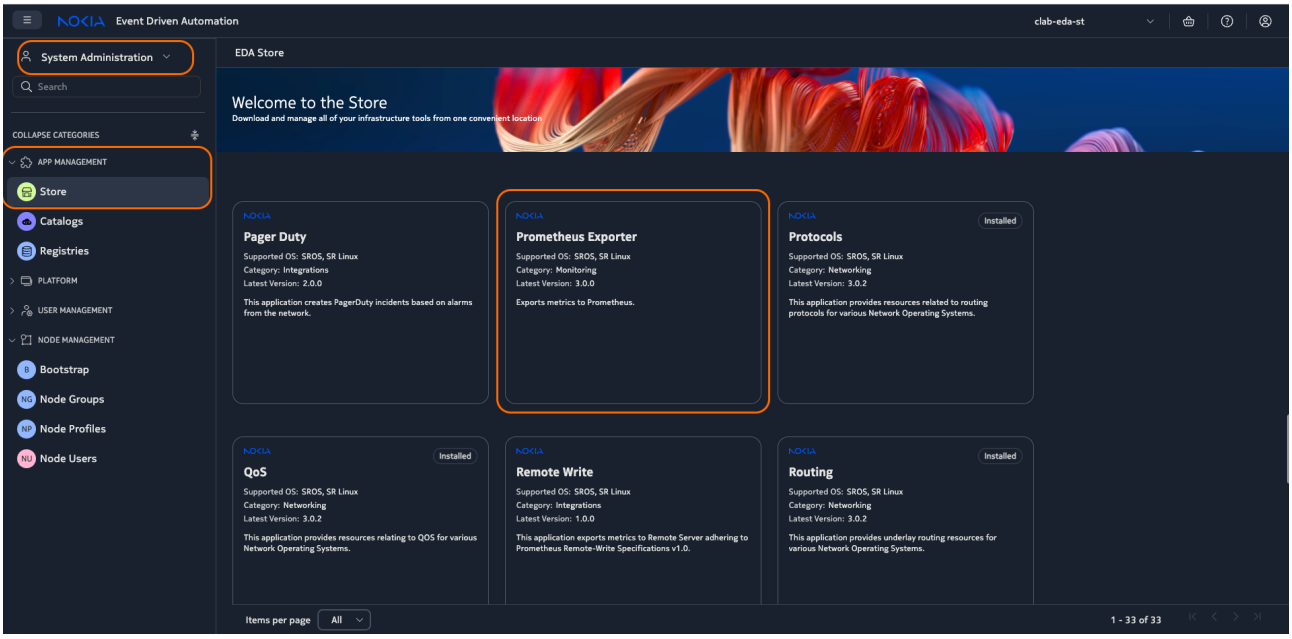
The Prometheus-exporter app can be installed either using kubectl cli or EDA-UI

Option 1: Using kubectl CLI

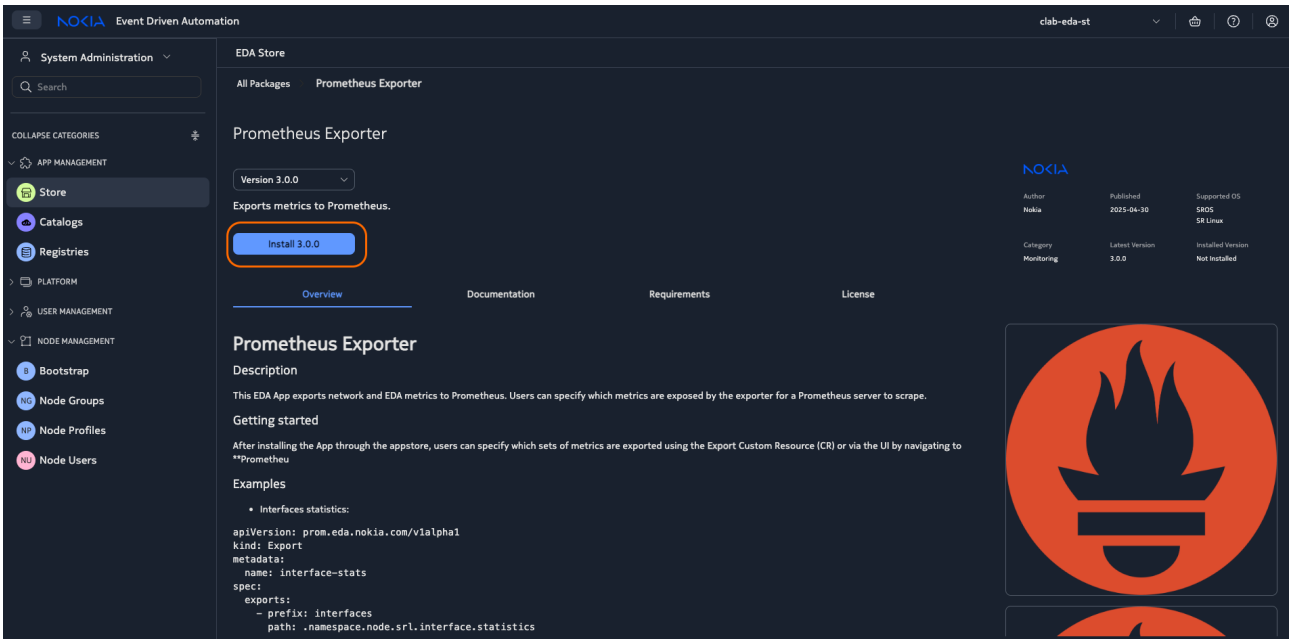
```
cat << 'EOF' | kubectl apply -f -
apiVersion: core.eda.nokia.com/v1
kind: Workflow
metadata:
  name: prom-exporter-app
  namespace: eda-system
spec:
  type: app-installer
  input:
    operation: install
    apps:
      - app: prom-exporter
        catalog: eda-catalog-builtin-apps
        vendor: nokia
        version:
          type: semver
          value: v3.0.0
EOF
```

Option 2: Using EDA UI

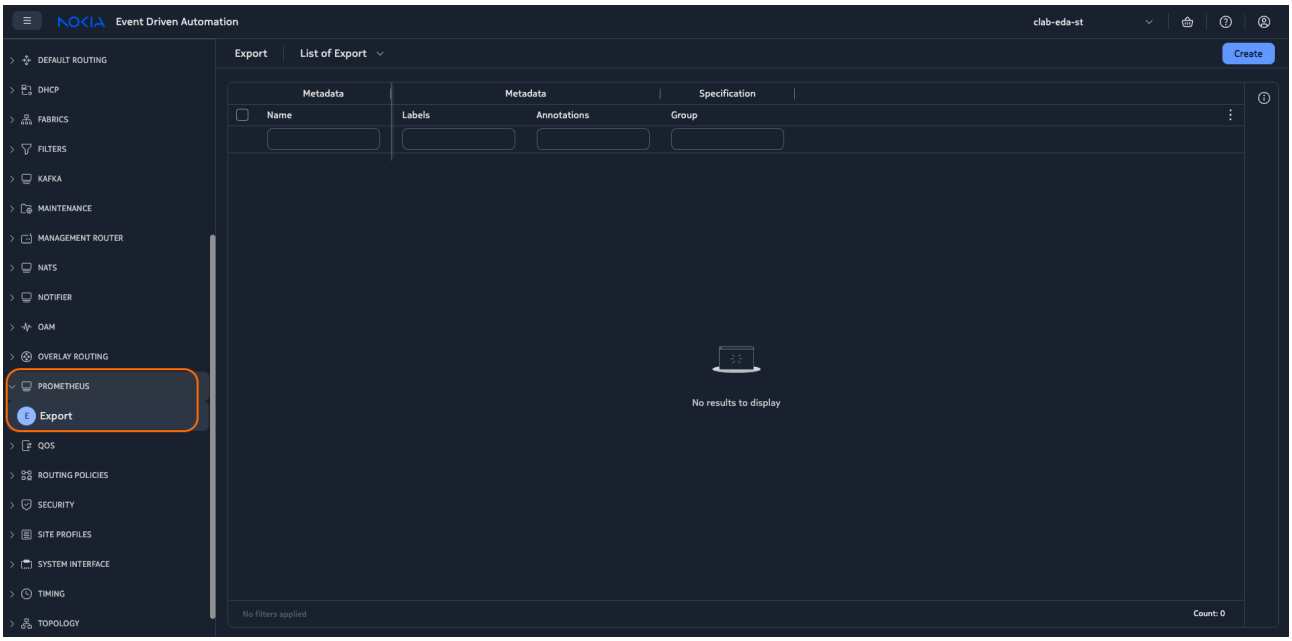
Navigate to App Store under System Administration > Store > Prometheus Exporter.



Click Install to download and install the Prometheus Exporter App.



When you have successfully installed the app, you can navigate to the Prometheus tab under Main options.



This can be verified using kubectl cli that promether-exporter pod is deployed and running.

```
[root@ai-eda ~]# kubectl get pods -A
NAMESPACE            NAME                                        READY   STATUS    RESTARTS       AGE
cert-manager         cert-manager-777c6f8ff4-86t7z               1/1     Running   1 (7d9h ago)   36d
cert-manager         cert-manager-cainjector-6558fc6578-k7wwc    1/1     Running   1 (7d9h ago)   36d
eda-system           eda-notifier-845f8d688b-vz9fs               1/1     Running   1 (7d9h ago)   35d
eda-system           eda-npp-0                                   1/1     Running   1 (7d9h ago)   36d
eda-system           eda-postgres-7ff6b48dc9-zflg5               1/1     Running   0              2d
eda-system           eda-px-55c6dd5588-445lz                     1/1     Running   0              2m1s
eda-system           eda-sa-59fb5c5769-6s6mk                     1/1     Running   1 (7d9h ago)   36d
eda-system           eda-sc-7dcbdcb9c-5clcp                      1/1     Running   1 (7d9h ago)   36d
```

8.4 Metrics export

After the Prometheus Exporter App is installed, you can define the metrics to be exported by specifying an EDB path (also known as jsPath). The output of the given path can be verified using the EDA Query UI also under Main> Tools > Queries.

During each scrape, the app generates Prometheus-compatible metrics with support for regex-based renaming, label enrichment, and value mapping.

The metrics are shown at:

`https://<EDA_API_ADDRESS>/core/httpproxy/v1/prometheus-exporter/metrics`

The metrics can be exposed at the URL using either kubectl CLI or UI. The user needs the path, which is the YANG path of the metrics that need to be exported.

Option 1: Using kubectl CLI

```
cat << 'EOF' | kubectl apply -f -
apiVersion: prom.eda.nokia.com/v1alpha1
kind: Export
metadata:
  name: interface
  namespace: eda-system
spec:
  exports:
    - path: .namespace.node.srl.interface
      mappings:
        - destination: "1"
          source: up
        - destination: "0"
          source: down
      metricName:
        regex: namespace_(.+)
        replacement: $1
EOF
```

Verification:

```
[root@ai-dev]# kubectl get exports -n eda-system
NAME                     AGE
cpu                      13d
default-router-status    7d21h
fabric-status            7d21h
in-packets               10d
interface                2m43s
interface-traffic-rate   10d
ipv4-statistics          10d
ipv6-packets             10d
memory                   7d21h
```

EDA lets users customize metrics in number of ways, such as renaming metric names, adding mapping to non-numeric values, and adding labels and filters.

Path	The EDA Yang path to export, example: .namespace.node.srl.interface.statistics
Fields	List of fields to expose as part of the metric. The EDB path has numerous fields; if not defined, all fields under the configured path are exposed.
Where	A where clause to use for the query, such as “oper-state = down”.
Prefix	An optional prefix to add to all metrics exposed by this export, for example “customerA” and the resulting metric name would be “**customerA**_<metricname>”
Metric name	Metric names using regex and replacements for customization.
Labels	Static or dynamic labels to add to metrics.
Mappings	Rules to map non-numeric values to numeric equivalents.

The following example shows the mapping of non-numeric values where, for example, the operational status of an interface when “up” is mapped as 1 and when “down” is mapped as 0.

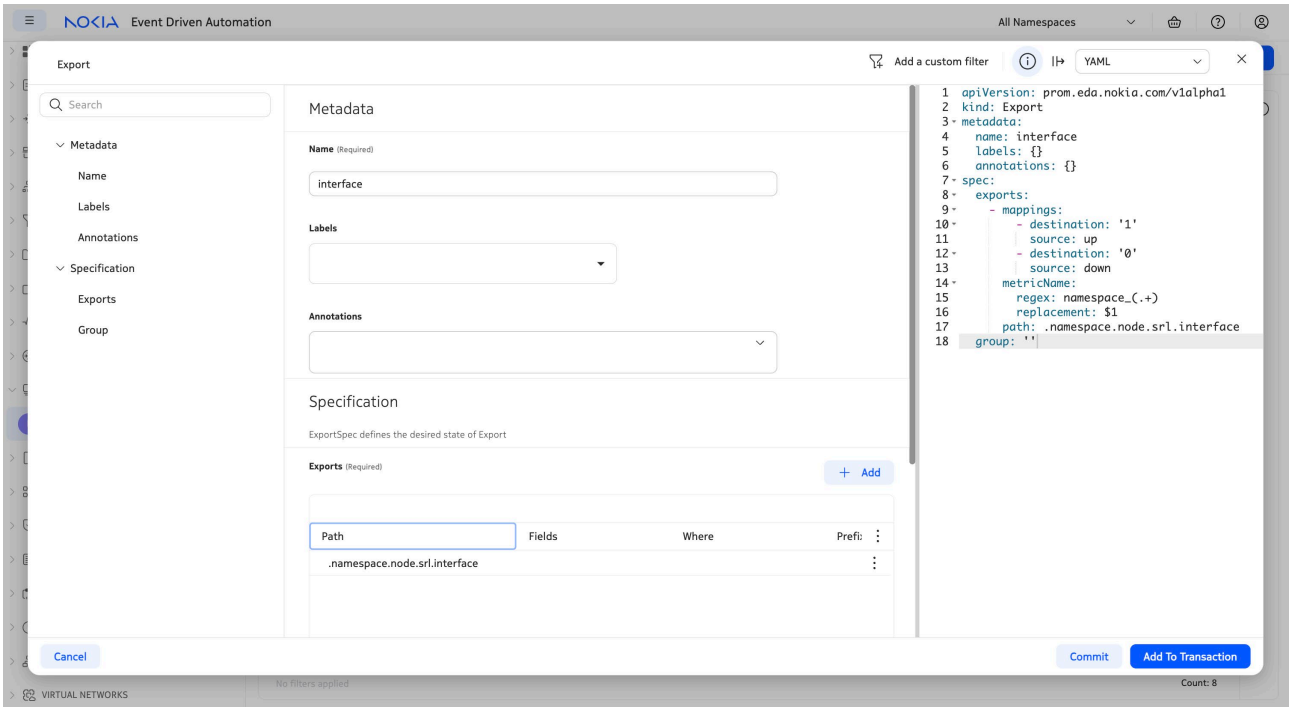
```
mappings:
        - destination: "1"
          source: up
        - destination: "0"
          source: down
```

The prefix “ai” is prepended to the metrics name.

```
# HELP ai_node_srl_interface_oper_state Dynamically created metric for oper-state
# TYPE ai_node_srl_interface_oper_state untyped
ai_node_srl_interface_oper_state{interface_name="ethernet-1/1",namespace_name="clab-ai-fabric-1tier",node_name="1tier-rail-01"} 1
ai_node_srl_interface_oper_state{interface_name="ethernet-1/1",namespace_name="clab-ai-fabric-1tier",node_name="1tier-rail-02"} 1
ai_node_srl_interface_oper_state{interface_name="ethernet-1/10",namespace_name="clab-ai-fabric-1tier",node_name="1tier-rail-01"} 0
ai_node_srl_interface_oper_state{interface_name="ethernet-1/10",namespace_name="clab-ai-fabric-1tier",node_name="1tier-rail-02"} 0
```

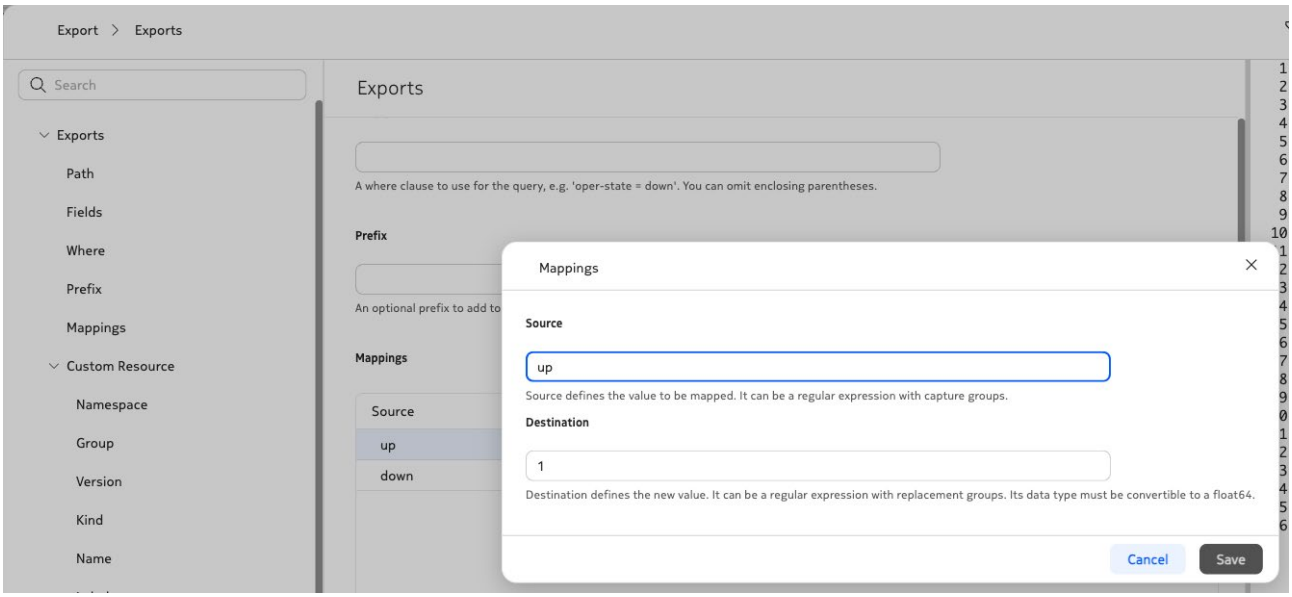
Option 2: Using EDA UI

Navigate to Main > Prometheus > Export > Create.



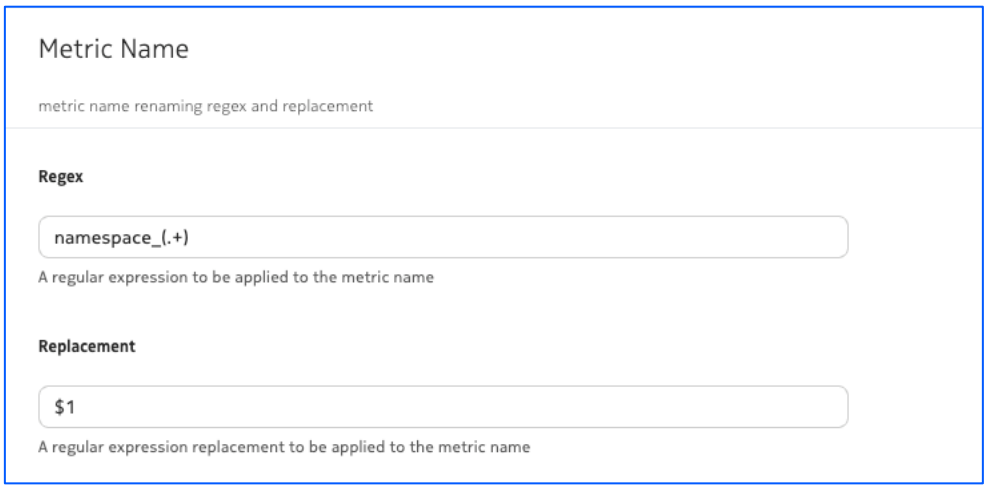
Mapping: Here you can map non-numeric values metrics easily using EDA UI.

This option is under the PATH label.



Regex: Metric Names can also be modified using regex using EDA UI.

This option is under the PATH:



8.5 Installing the telemetry stack - Prometheus and Grafana

The telemetry stack, which includes Prometheus and Grafana, can be installed using Helm charts with the provided `install_telemetry_stack.sh` script.

```
[root@dev-node]# ./install_telemetry_stack.sh
Installing telemetry-stack helm chart...
NAME: telemetry-stack
LAST DEPLOYED: Wed Aug 20 10:14:29 2025
```

```
NAMESPACE: eda-telemetry
STATUS: deployed
REVISION: 1
TEST SUITE: None

Step-1: Run the following command to access Grafana:
kubectl port-forward -n eda-telemetry service/grafana 3000:3000 --address=0.0.0.0 >/dev/null
2>&1 & disown
kubectl port-forward -n eda-telemetry service/prometheus 9090:9090 --address=0.0.0.0
>/dev/null 2>&1 & disown

Step-2: Open your browser and access the following URLs:
You can open Grafana at http://<eda-host-ip>:3000
You can open Prometheus at http://<eda-host-ip>:9090
```

Verification:

After the charts are deployed, you will see two new pods in the EDA kubernetes cluster.

```
[root@eda2 eda-telemetry-lab]# kubectl get pods -n eda-telemetry
NAME                          READY   STATUS    RESTARTS   AGE
grafana-7cf578d5db-cbpz8      1/1     Running   0          9h
prometheus-765878c867-rvndp   1/1     Running   0          9h
```

8.5.1 Prometheus configurations

Users do not need to modify the following settings; these are preconfigured by Helm charts. These charts are provided for reference only, where `targets` specify the IP/FQDN/service endpoint of the eda-api exposing metrics, and `metrics_path` defines the path used to access them.

```
kubectl port-forward -n eda-telemetry service/grafana 3000:3000 --address=0.0.0.0 >/dev/null
2>&1 & disown
kubectl port-forward -n eda-telemetry service/prometheus 9090:9090 --address=0.0.0.0
>/dev/null 2>&1 & disown

global:
  scrape_interval: 5s

scrape_configs:

  - job_name: 'eda-exporter'
    scrape_interval: 5s
    static_configs:
      - targets: ['eda-api.eda-system.svc']
    metrics_path: /core/httpproxy/v1/prometheus-exporter/metrics
    scheme: https
    tls_config:
```

```
      insecure_skip_verify: true
```

Prometheus UI can be accessed using the link in the output shown below

Step 1: Port forward the Prometheus and Grafana service.

Step 2: You can access the Grafana UI at your browser:

```
Grafana:        http://<eda-host-ip>:3000
Prometheus:     http://<eda-host-ip>:9090
```

8.6 Grafana

Grafana acts as the visualization platform enabling users to build dashboards that provide a unified view of network link utilization and health of the nodes. It uses Prometheus as a data source, retrieving time-series metrics through PromQL queries.

9 Digital twin

Digital twins are an integral part of Day-0 through Day-2 operations, providing the operations and deployment teams with the opportunity to continuously validate the look and feel of any deployment. These virtual fabrics also grant the ability to learn and play with technologies and designs—in this case, a prescriptive rail-only AI fabric that has been validated and tuned to provide maximum

efficiency and redundancy. A digital twin of this NVD can be deployed using Containerlab and containerized SR Linux. The repository can be found here:

https://github.com/nokia/nokia-validated-designs/tree/main/validated-designs/ai-dc/lenovo-nokia-ai-fabric