Juniper Networks | Engineering Simplicity

AI Data Center Network with Juniper Apstra, AMD GPUs, Broadcom NIC, AMD Pollara NIC, and Vast Storage—Juniper Validated Design (JVD)

Published
2025-12-23

Table of Contents

AI Data Center Network with Juniper Apstra, AMD GPUs, Broadcom NIC, AMD Pollara NIC, and Vast Storage—Juniper Validated Design (JVD)

Juniper Networks Validated Designs provide a comprehensive, end-to-end blueprint for deploying Juniper solutions in your network. These designs are created by Juniper's expert engineers and tested to ensure they meet your requirements. Using a validated design, you can reduce the risk of costly mistakes, save time and money, and ensure that your network is optimized for maximum performance.

About this Document

This document describes the design requirements and implementation of an AI cluster infrastructure connecting AMD MI300X GPU servers and Vast Storage systems, based on AI-optimized Juniper Data Center Juniper QFX5240 series switches, which are configured and managed by Juniper Apstra and Terraform automation. As part of this solution, both Broadcom Thor2 and AMD Pollara network interface cards (NICs) have been validated for compatibility and performance.

All validation tests were conducted in Juniper's AI Innovation Lab in Sunnyvale, CA, USA. In this open lab, Juniper collaborates closely with customers and technology partners to develop AI solutions and test deployments for a range of AI applications and models.

The AI Innovation Lab allows customers to see AI training and inference in action. Juniper performs these tests running both customer-specific models as well as those from MLCommons for MLPerf performance benchmarking and comparisons.

Nomenclature: For brevity, AMD Pensando Pollara 400 NIC will be referred to as AMD Pollara NIC throughout thisdocument.

Solution Benefits

IN THIS SECTION

Juniper Networks has excelled in building and supporting AI networks following a scalable, robust, and automated approach suitable for a range of cluster sizes. Unlike proprietary solutions that lock in enterprises and can stifle AI innovation, Juniper's standards-based solution assures the fastest innovation, maximizes design flexibility, and prevents vendor lock-in on the Frontend, GPU Backend, and Storage Backend AI fabric networks.

The Juniper Validated Design (JVD) for AI describes a structured approach for deploying high-performance AI training and inference networks that minimize job completion time and maximize GPU performance. Additionally, it incorporates industry best practices, and leverages Juniper's extensive expertise in building high-performance data center networks.

The design in this JVD employs a 3-stage Clos IP fabric architecture, utilizing Juniper QFX-series switches as leaf and spine nodes. It integrates multi-vendor GPU servers and storage devices and is deployed and managed using Juniper Apstra and Terraform Automation.

The integration with Juniper's Apstra software and Terraform enables customers to orchestrate the network infrastructure systematically, without requiring in-depth knowledge of the products and technologies involved. This allows customers to easily build high-capacity, easy-to-operate network fabrics that deliver high performance and increased reliability, which results in optimal JCT (Job Completion Time) and maximized GPU utilization in the AI cluster.

The solution has been extensively tested and thoroughly documented by Juniper subject matter experts, resulting in a validated design that is easy to follow, guarantees successful implementation, and simplified management and troubleshooting tasks. This document provides comprehensive guidance on how to deploy this solution, with clear descriptions of its components and step by step instructions to connect and configure them.

Juniper Validated Design Benefits

JVDs are prescriptive blueprints for building data center fabrics using repeatable, validated, predictable, and well documented network architecture solutions with guidelines for a successful deployment. Each solution has been designed, fully tested, and documented by Juniper Networks experts with all the necessary implementation details, including hardware components, software versions, connectivity, and configuration steps.

To become a validated solution (JVD) and be approved for release, a solution must pass rigorous testing with real-world workloads and applications. All features must satisfy operational and performance criteria in real-world scenarios. Testing not only includes validating the design topology and configuration steps, but also that all products in the JVD work together as expected, thereby mitigating potential risks while deploying the solution.

The core benefits of JVDs solutions can be summarized as:

- Qualified Deployments—Qualified network design blueprints for data center fabrics, that follow best practices and meet the requirements of each specific use case, and make the solution deployment quicker, simpler, and more reliable.
- Scalable—Solutions that can scale beyond the initial design and support the adoption of different hardware platforms based on customer requirements, and customers' feedback can meet the needs of most Juniper's data center customers.
- Risk Mitigation— Prescriptive implementation guidelines guarantee that you have the right products, the right software versions, an optimal architecture, and comprehensive deployment steps.
- Systematically Verified—Tested solutions using a suite of automated testing tools validate the performance and reliability of all the components.
- Predictability— Detailed testing and careful documentation of the solution, including the capabilities and limitations of its components, guarantees that the solution will operate as expected when implemented according to the JVD guidelines.
- Repeatability— Unlocked value with repeatable network designs due to the prescriptive nature of JVD designs as well as their applicability to common use cases in the data center environment. All JVD customers benefit from lessons learned through lab testing and real-world deployments.
- Reliability— Tested with real traffic, JVD solutions are qualified to operate as designed after deployment and with real-world traffic.
- Accelerated Deployment— Ease installation with step-by-step guidance automation, and prebuilt integrations simplifies, and accelerates deployment, while reducing risks.

- Accelerated Decision-Making— Predefined combination of products, software, and architecture removes the need to spend time comparing products, and deciding how the network should be built, allowing to bridge business and technology requirements faster and reducing risks.
- Best Practice Networks— Better outcomes for a better experience. Juniper Validated Designs have known characteristics and performance profiles to help you make informed decisions about your network.

Juniper Apstra Benefits

Juniper Validated Designs in the data center start with the Apstra software, a multi-vendor, intent-based networking system (IBNS) that provides closed-loop automation and assurance. Apstra translates vendor-agnostic business intent and technical objectives to essential policy and device-specific configurations. The system also validates user intent, as part of the initial deployment and continuously thereafter, to ensure that the network state does not deviate from the intended state. Any anomaly or deviation can be flagged, and remediation actions can be taken directly from Apstra.

The core benefits of Apstra are:

- Intent-based networking—Apstra automates configuration creation to realize the intent, deploys the configuration to appropriate devices, and continuously validates the operating state against intended state.
- Network Automation—Apstra is a multi-vendor network automation platform that is continuously updated to work with the latest hardware and is extensively tested using modern DevOps practices.
- Recoverability—The Built-in rollback capability of Apstra allows to quickly restore the system to a known-working configuration if needed.
- Day 2+ Management—Apstra's rich data analysis capabilities, including Flow Data, reduce Mean Time to Resolution (MTTR).
- Simplicity—Apstra simplifies network deployment and management. As an example, using Apstra to implement a Data Center Interconnection (DCI), reduces complexity and makes it easy to unify multiple data centers, while isolating failure domains for high availability and resilience.

AI Use Case and Reference Design

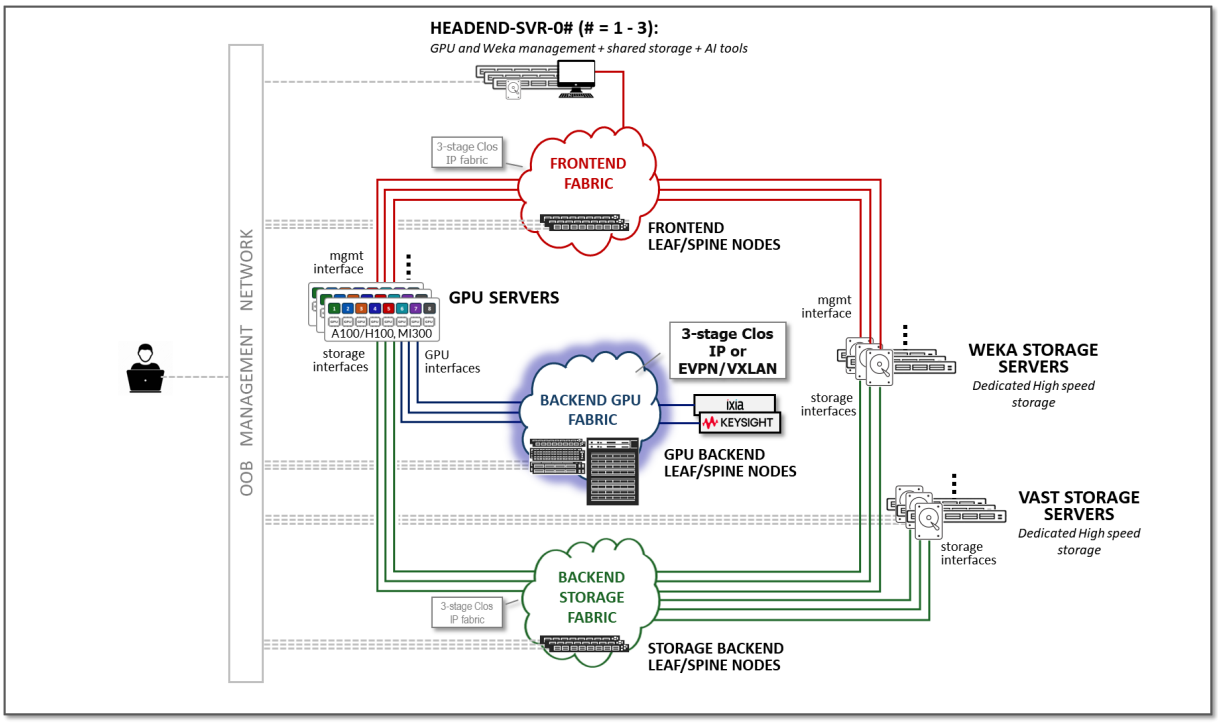
IN THIS SECTION

The **AI JVD Reference Design** covers a complete end-to-end ethernet-based AI infrastructure, which includes the Frontend fabric, GPU Backend (Graphics Processing Unit) fabric and Storage Backend fabric. These three fabrics have a symbiotic relationship, while each provides unique functions to support AI training and inference tasks. The use of Ethernet Networking in AI Fabrics enables our customers to build high-capacity, easy-to-operate network fabrics that deliver the fastest job completion times, maximize GPU utilization, and use limited IT resources.

The AI JVD reference design shown in Figure 1 on page 6 includes:

- **Frontend Fabric**: This fabric is the gateway network to the GPU nodes and storage nodes from the AI tools residing in the headend servers. The Frontend GPU fabric allows users to interact with the GPU and storage nodes to initiate training or inference workloads and to visualize their progress and results, and provides an out-of-band path for RCCL (ROCm Communication Collectives Library).
- **GPU Backend Fabric**: This fabric connects the GPU nodes (which perform the computations tasks for AI workflows). The GPU Backend fabric transfers high-speed information between GPUs during training jobs, in a lossless matter. Traffic generated by the GPUs is transferred using RoCEv2 (RDMA over Ethernet v2).
- **Storage Backend Fabric**: This fabric connects the high-availability storage systems (which hold the large model training data) and the GPUs (which consume this data during training or inference jobs). The Storage Backend fabric transfers high volumes of data in a seamless and reliable matter.

Figure 1: AI JVD Reference Design



Frontend Overview

The AI Frontend for AI encompasses the interface, tools, and methods that enable users to interact with the AI systems, and the infrastructure that allows these interactions. The Frontend gives users the ability to initiate training or inference tasks, and to visualize the results, while hiding the underlying technical complexities.

The key components of the Frontend systems include:

- **Model Scheduling**: Tools and methods for managing scripted AI model jobs and commonly based on SLURM (Simple Linux Utility for Resource Management) Workload Manager. These tools enable users to send instructions, commands, and queries, either through a shell CLI or through a graphical web-based interface to orchestrate learning and inference jobs running on the GPUs. Users can configure model parameters, input data, and interpret results as well as initiate or terminate jobs interactively. In the AI JVD, these tools are hosted on the *Headend Servers* connected to the AI Frontend fabric.
- **Management of AI Systems**: Tools for managing (configuring, monitoring and performing maintenance tasks) the AI storage and processing components. These tools facilitate building,

running, training, and utilizing AI models efficiently. Examples include SLURM, TensorFlow, PyTorch, and Scikit-learn.

- **Management of Fabric Components**: Mechanisms and workflows designed to help users effortlessly deploy and manage fabric devices according to their requirements and goals. It includes tasks such as device onboarding, configuration management, and fabric deployment orchestration. This functionality is provided by ***Juniper Apstra*** .
- **Performance Monitoring and Error Analysis**: Telemetry systems tracking key performance metrics related to AI models, such as accuracy, precision, recall, and computational resource utilization (e.g. CPU, GPU usage) which are essential for evaluating model effectiveness during training and inference jobs. These systems also provide insights into error rates and failure patterns during training and inference operations, and help identify issues such as model drift, data quality problems, or algorithmic errors that may affect AI performance. Examples of these systems include Juniper Apstra dashboards, TIG Stack, and Elasticsearch.
- **Data Visualization**: Applications and tools that allow users to visually comprehend insights generated by AI models and workloads. They provide effective visualization that enhances understanding and decision-making based on AI outputs. The same telemetry systems used to monitor and measure System and Network level performance usually provide this visualization as well. Examples of these tools include Juniper Apstra dashboards, TensorFlow, and TIG stack.
- **User Interface**: Routing and switching infrastructure that allows communication between the user interface applications and tools and the AI systems executing the jobs, including GPUs and storage devices. This infrastructure ensures seamless interaction between users and the computational resources needed to leverage AI capabilities effectively.
- **GPU-to-GPU control**: Communication establishment, information exchange including, QP GIDs (Global IDs), Local and remote buffer addresses, and RDMA keys (RKEYs for memory access permissions)

GPU Backend Overview

The GPU Backend for AI encompasses the devices that execute learning and inference jobs or computational tasks, that is the GPU servers where the data processing occurs, and the infrastructure that allows the GPUs to communicate with each other to complete the jobs.

The key components of the GPU Backend systems include:

- **AI Systems**: Specialized hardware such as GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) that can execute numerous calculations concurrently. GPUs are particularly adept at handling AI workloads, including complex matrix multiplications and convolutions required to

complete learning and inference tasks. The selection and number of GPU systems significantly impacts the speed and efficiency of these tasks.

- **AI Software**: Operating systems, libraries, and frameworks essential for developing and executing AI models. These tools provide the environment necessary for coding, training, and deploying AI algorithms effectively. The functions of these tools include:
  - **Data Management**: Preprocessing, and transformation of data utilized in training and executing AI models. This encompasses tasks such as cleaning, normalization, and feature extraction. Given the volume and complexity of AI datasets, efficient data management strategies like parallel processing and distributed computing are crucial.
  - **Model Management**: Tasks related to the AI models themselves, including evaluation (e.g., cross-validation), selection (choosing the optimal model based on performance metrics), and deployment (making the model accessible for real-world applications).
- **GPU Backend Fabric**: Routing and switching infrastructure that allows GPU-to-GPU communication for workload distribution, memory sharing, synchronization of model parameters, exchange of results, etc. The design of this fabric can significantly impact the speed and efficiency of AI/ML model training and inference jobs and in most cases shall provide lossless connectivity for GPU-to-GPU traffic.

Storage Backend Overview

The AI storage backend for AI encompasses the hardware and software components for storing, retrieving, and managing the vast amounts of data involved in AI workloads, and the infrastructure that allows the GPUs to communicate with these storage components.

The key aspects of the storage backend include:

- **High-Performance Storage Devices**: Optimized for high I/O throughput, which is essential for handling the intensive data processing requirements of the AI tasks such as deep learning. This includes high-performance storage devices designed to facilitate fast access to data during model training and to accommodate the storage needs of large datasets. These storage devices must provide:
  - **Data Management Capabilities**: Supports efficient data querying, indexing, and retrieval which are crucial for minimizing preprocessing and feature extraction times in AI workflows, as well as for facilitating quick data access during inference.
  - **Scalability**: Accommodates growing data volumes and efficiently manages and stores massive amounts of data over time, to support AI workloads often involving large-scale datasets.

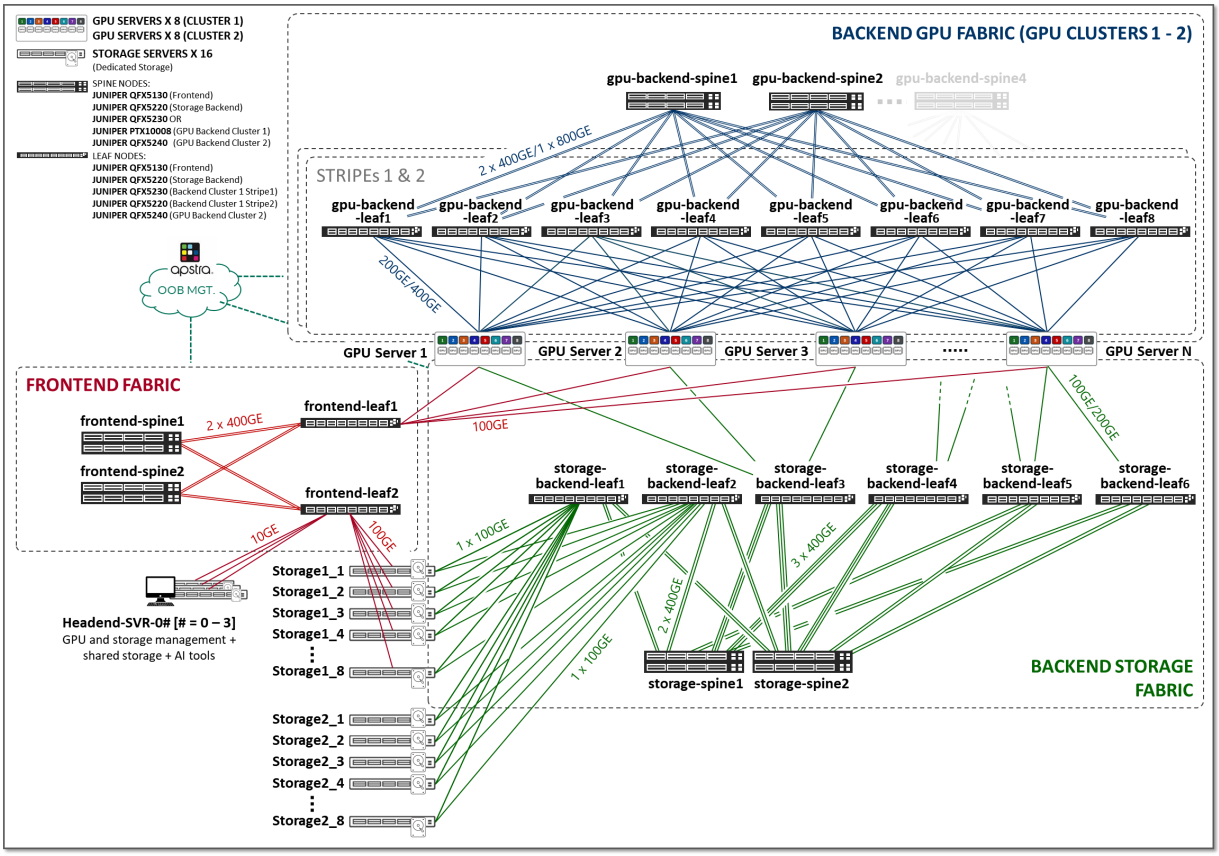
- **Storage Backend Fabric**: Routing and switching infrastructure that provides the connectivity between the GPU and the storage devices. This integration ensures that data can be efficiently transferred between storage and computational resources, optimizing overall AI workflow performance. The performance of the storage backend significantly impacts the efficiency and JCT of AI/ML workflows. A storage backend that provides quick access to data can significantly reduce the amount of time for training AI/ML models.

Solution Architecture

IN THIS SECTION

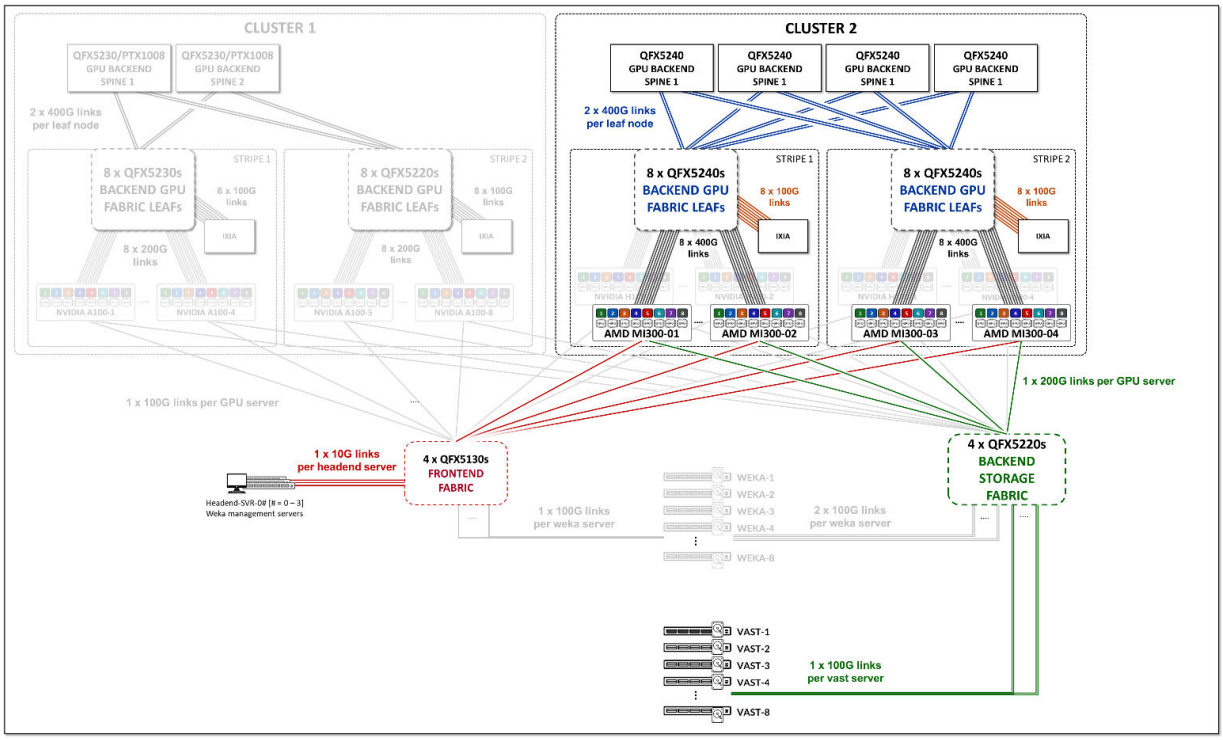
The three fabrics described in the previous section (Frontend, GPU Backend, and Storage Backend), are interconnected together in the overall AI JVD solution architecture as shown in Figure 2.

Figure 2: AI JVD Solution Architecture'



We have built two different Clusters, as shown in Figure 3, which share the " Frontend fabric " on page 12 and "Storage Backend fabric" on page 26 but have separate " GPU Backend fabrics " on page 13. Each cluster is made of two stripes following the " Rail Optimized Stripe Architecture " on page 21, but include different switch models as Leaf and Spine nodes, as well as GPU server models.

Figure 3: AI JVD Lab Clusters



The GPU Backend in Cluster 1 consists of Juniper QFX5220, and QFX5230 switches as leaf nodes and either QFX5230s switches or PTX10008 routers acting as spine nodes. The QFX5230s and PTX10008 have been validated acting as spine nodes separately, while maintaining the leaf nodes the same. Apstra blueprints are used to switch between the setups with QFX5230s acting as spine nodes and the one with PTX10008 acting as spine.

The GPU Backend in Cluster 2 consists of Juniper QFX5240 switches acting as both leaf nodes and spine nodes and includes AMD MI300X GPU servers and Nvidia H100 GPU servers.

Details about Cluster 1, and the Nvidia GPU servers in Cluster 2 are included in the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD).

The rest of this document focuses on the AMD MI300X GPU servers and VAST storage and includes server and storage configurations, specific for these systems.

It is important to notice that the type of switch and the number of switches acting as leaf and spine nodes, as well as the number and speed of the links between them, is determined by the type of fabric (Frontend, GPU Backend or Storage Backend) as they present different requirements. More details will be included in the respective fabric description sections.

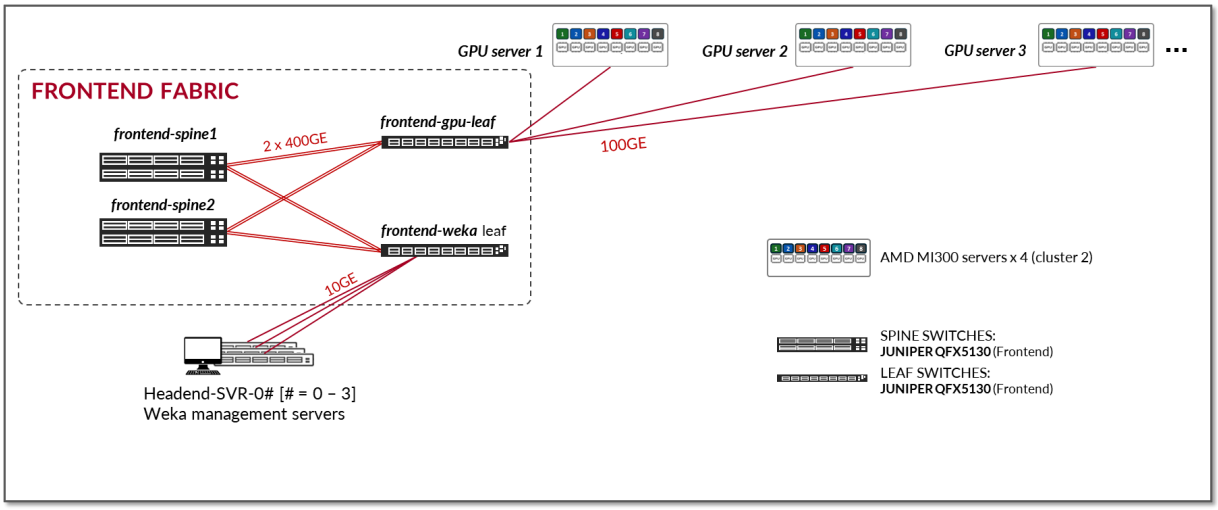
In the case of the GPU Backend fabric, the number of GPU servers, as well as the number of GPUs per server, are also factors determining the number and switch type of the leaf and spine nodes.

Frontend Fabric

The **Frontend Fabric** provides the infrastructure for users to interact with the AI systems to orchestrate training and inference tasks workflows using tools such as SLURM. These interactions do not generate heavy data flows nor have rigorous requirements regarding latency or packet drops; thus, they do not impose rigorous demands on the fabric.

The **Frontend Fabric** design described in this JVD follows a traditional 3-stage IP Fabric architecture without HA, as shown in Figure 4. This architecture provides a simple and effective solution for the connectivity required in the Frontend. However, any fabric architecture including EVPN/VXLAN, could be used. If an HA-capable Frontend Fabric is required we recommend following the 3-Stage with Juniper Apstra JVD. An EVPN/VXLAN JVD specifically for AI will be developed in the future.

Figure 4: Frontend Fabric Architecture



NOTE: The Vast Storage cluster is not connected to the Frontend fabric.

The devices included in the Frontend fabric, and the connections between them, are summarized in the following tables:

Table 1: Frontend devices

AMD GPU Servers	Headend Servers	Frontend Leaf Nodes switch model	Frontend Spine Nodes switch model
MI300X x 4 (MI300X-01 to MI300X-04)	Headend-SVR x 3 (Headend-SVR-01 to Headend-SVR-03)	QFX5130-32CD x 2 (frontend-leaf#; #= 1-2	QFX5130-32CD x 2 (frontend-spine#; #= 1-2)

Table 2: Connections between servers, leaf and spine nodes per cluster and stripe in the Frontend

GPU Servers <=> Frontend Leaf Nodes	Headend Servers <=> Frontend Leaf Nodes	Frontend Leaf Nodes <=> Frontend Spine Nodes
1 x 100GE links per GPU server to leaf connection Total number of 100GE links between GPU servers and frontend leaf nodes = 4 (4 servers x 1 link per server)	1 x 10GE links per headend server to lead connection Total number of 10GE links between headend servers and frontend leaf nodes = 3 (3 servers x 1 link/server)	2 x 400GE links per leaf node to spine node connection Total number of 400GE links between frontend leaf nodes and spine nodes = 8 (2 leaf nodes x 2 spines nodes x 2 links per leaf to spine connection)

This fabric is an L3 IP fabric using EBGP for route advertisement. The IP addressing and EBGP configuration details are described in the networking section on this document.

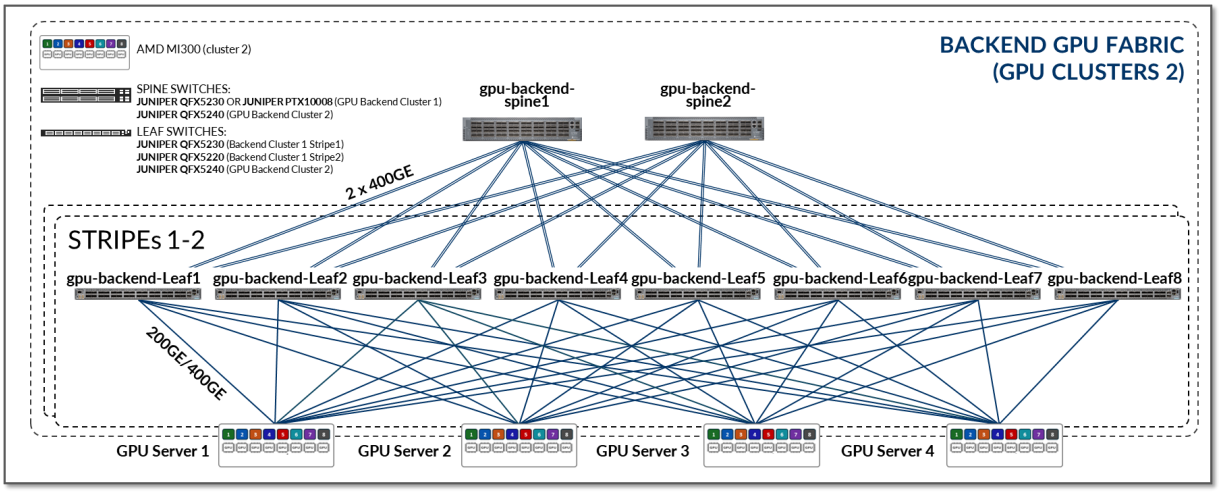
GPU Backend Fabric

The GPU Backend fabric provides the infrastructure for GPUs to communicate with each other within a cluster, using RDMA over Converged Ethernet (RoCEv2). RoCEv2 enhances data center efficiency, reduces complexity, and optimizes data delivery across high-speed Ethernet networks.

Packet loss can significantly impact job completion times and therefore should be avoided. Therefore, when designing the compute network infrastructure to support RoCEv2 for an AI cluster, one of the key objectives is to provide a near lossless fabric, while also achieving maximum throughput, minimal latency, and minimal network interference for the AI traffic flows. ROCEv2 is more efficient over lossless networks, resulting in optimum job completion times.

The GPU Backend fabric in this JVD was designed with these goals in mind. The design follows a 3-stage IP clos and "Rail Optimized Stripe architecture" on page 21 as shown in Figure 5.

Figure 5: GPU Backend Fabric Architecture



The devices that are part of the GPU Backend fabric, and the connections between them, are summarized in the following tables:

Table 3: GPU Backend devices per cluster and stripe

Stripe	GPU Servers	GPU Backend Leaf nodes switch model	GPU Backend Spine nodes switch model
1	MI300X x 2 (MI300X-01 & MI300X-02)	QFX5240-64OD x 8 (gpu-backend-001_leaf#; #=1-8)	QFX5240-64OD x 4 (gpu-backend-spine#; #=1-4)
2	MI300X x 2 (MI300X-03 & MI300X-04)	QFX5240-64OD x 8 (gpu-backend-002_leaf#; #=1-8)	

Table 4: GPU Backend connections between servers, leaf nodes and spine nodes.

Stripe	GPU Servers <=> GPU Backend Leaf Nodes	GPU Backend Leaf Nodes <=> GPU Backend Spine Nodes
1	**8** (number of GPUs per server) **x 400GE links** **per MI300X server to leaf connections** **Total number of 400GE links between** **servers and leaf nodes = 16** **(2 server x 8 links/server)**	**2 x 400GE links** **per leaf node to spine node connection** **Total number of 400GE links between** **frontend leaf nodes and spine nodes = 64** (8 leaf nodes x 4 spines nodes x 2 links per leaf to spine connection) each leaf node and each spine node)
2	**8** (number of GPUs per server) **x 400GE links** **per MI300X server to leaf connections** **Total number of 400GE links between** **servers and leaf nodes = 16** **(2 server x 8 links/server)**	**2 x 400GE links** **per leaf node to spine node connection** **Total number of 400GE links between** **frontend leaf nodes and spine nodes = 64** (8 leaf nodes x 4 spines nodes x 2 links per leaf to spine connection) each leaf node and each spine node)

- All the AMD MI300X GPU servers are connected to the GPU backend fabric using 400GE interfaces.
- This fabric is an L3 IP fabric that uses EBGP for route advertisement (This is described in the networking section).
- Connectivity between the servers and the leaf nodes is L2 untagged vlan-based with IRB interfaces on each leaf node acting as default gateway for the servers (described in the networking section).

The speed and number of links between the GPU servers and leaf nodes and between the leaf and spine nodes determines the oversubscription factor. As an example, consider the number of GPU servers available in the lab, and how they are connected to the GPU backend fabric as described above.

The bandwidth between the servers and the leaf nodes is 12.8 Tbps (Table 5), while the bandwidth available between the leaf and spine nodes is also 25.6 Tbps (Table 6). This means that the fabric has enough capacity to process all traffic between the GPUs even when this traffic is 100% inter-stripe and has extra capacity to accommodate 4 more servers. With 4 additional servers the subscription factor would be 1:1 (no oversubscription).

Table 5: Per stripe Server to Leaf Bandwidth

Server to Leaf Bandwidth per Stripe				
Stripe	Number of servers per Stripe	Number of 400 Gbps server ó leaf links per server (Same as number of leaf nodes & number of GPUs per server)	Server <=> Leaf Link Bandwidth [Gbps]	Total Servers <=> Leaf Links Bandwidth per stripe [Tbps]
1	2	8	400 Gbps	2 x 8 x 400 Gbps = 6.4 Tbps
2	2	8	400 Gbps	2 x 8 x 400 Gbps = 6.4 Tbps
			Total Server <=> Leaf Bandwidth	12.8 Tbps

Table 6: Per stripe Leaf to Spine Bandwidth

Leaf nodes to spine nodes bandwidth per Stripe					
Stripe	Number of leaf nodes	Number of spine nodes	Number of 400 Gbps leaf ó spine links per leaf node	Server <=> Leaf Link Bandwidth [Gbps]	Bandwidth Leaf <=> Spine Per Stripe [Tbps]
1	8	4	2	400	8 x 2 x 2 x 400 Gbps = 12.8Tbps
2	8	4	2	400	8 x 2 x 2 x 400 Gbps = 12.8Tbps
				Total Leaf <=> Spine Bandwidth	25.6 Tbps

Optimization in rail-optimized topologies refers to how GPU communication is managed to minimize congestion and latency while maximizing throughput. A key part of this optimization strategy is keeping traffic local whenever possible. By ensuring that GPU communication remains within the same rail or stripe, or even within the server, the need to traverse spines or external links is reduced, which lowers latency, minimizes congestion, and enhances overall efficiency.

While localizing traffic is prioritized, inter-stripe communication will be necessary in larger GPU clusters. Inter-stripe communication is optimized by means of proper routing and balancing techniques over the available links to avoid bottlenecks and packet loss.

The essence of optimization lies in leveraging the topology to direct traffic along the shortest and least-congested paths, ensuring consistent performance even as the network scales. Traffic between GPUs in the same servers can be forwarded locally across the internal Server fabric (vendor dependent), while traffic between GPUs in different servers happens across the external GPU backend infrastructure. Communication between GPUs in different servers can be intra-rail, or inter-rail/inter-stripe.

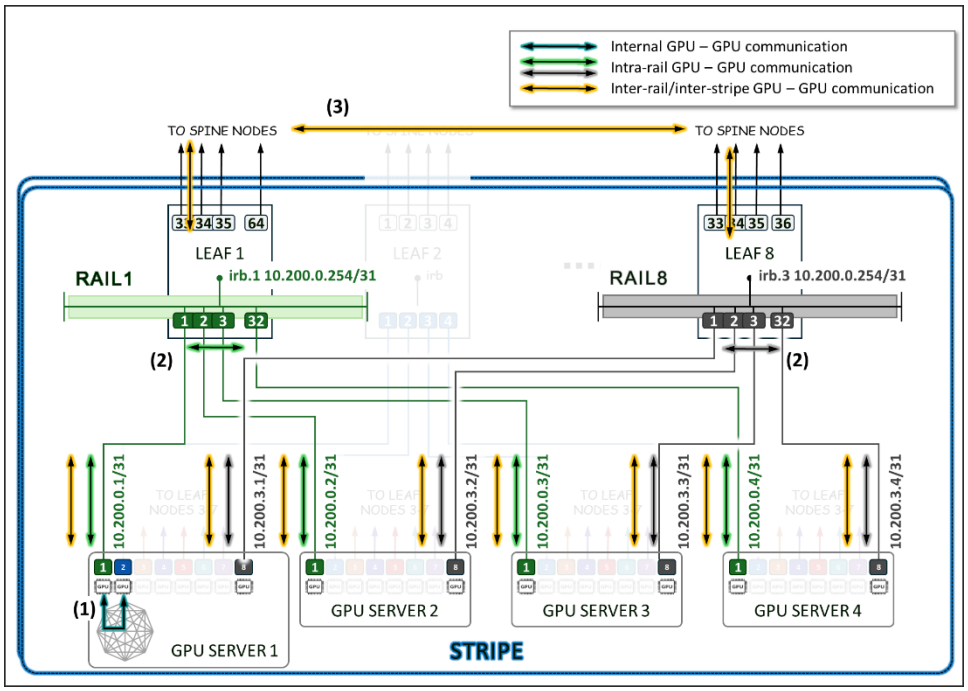
Intra-rail traffic is switched (processed at Layer 2) on the local leaf node. Following this design, data between GPUs on different servers (but in the same stripe) is always moved on the same rail and across one single switch. This guarantees GPUs are 1 hop away from each other and will create separate independent high-bandwidth channels, which minimize contention and maximize performance. On the other hand, inter-rail/inter-stripe traffic is routed across the IRB interfaces on the leaf nodes and the spine nodes connecting the leaf nodes (processed at Layer 3).

Using the example for calculating the number of servers per stripe provided in the previous section, we can see how:

- Communication between GPU 1 and GPU 2 in server 1 happens across the server's internal fabric (1),
- Communication between GPUs 1 in servers 1- 4, and between GPUs 8 in servers 1- 4 happens across Leaf 1 and Leaf 8 respectively (2), and
- Communication between GPU 1 and GPU 8 (in servers 1- 4) happens across leaf1, the spine nodes, and leaf8 (3)

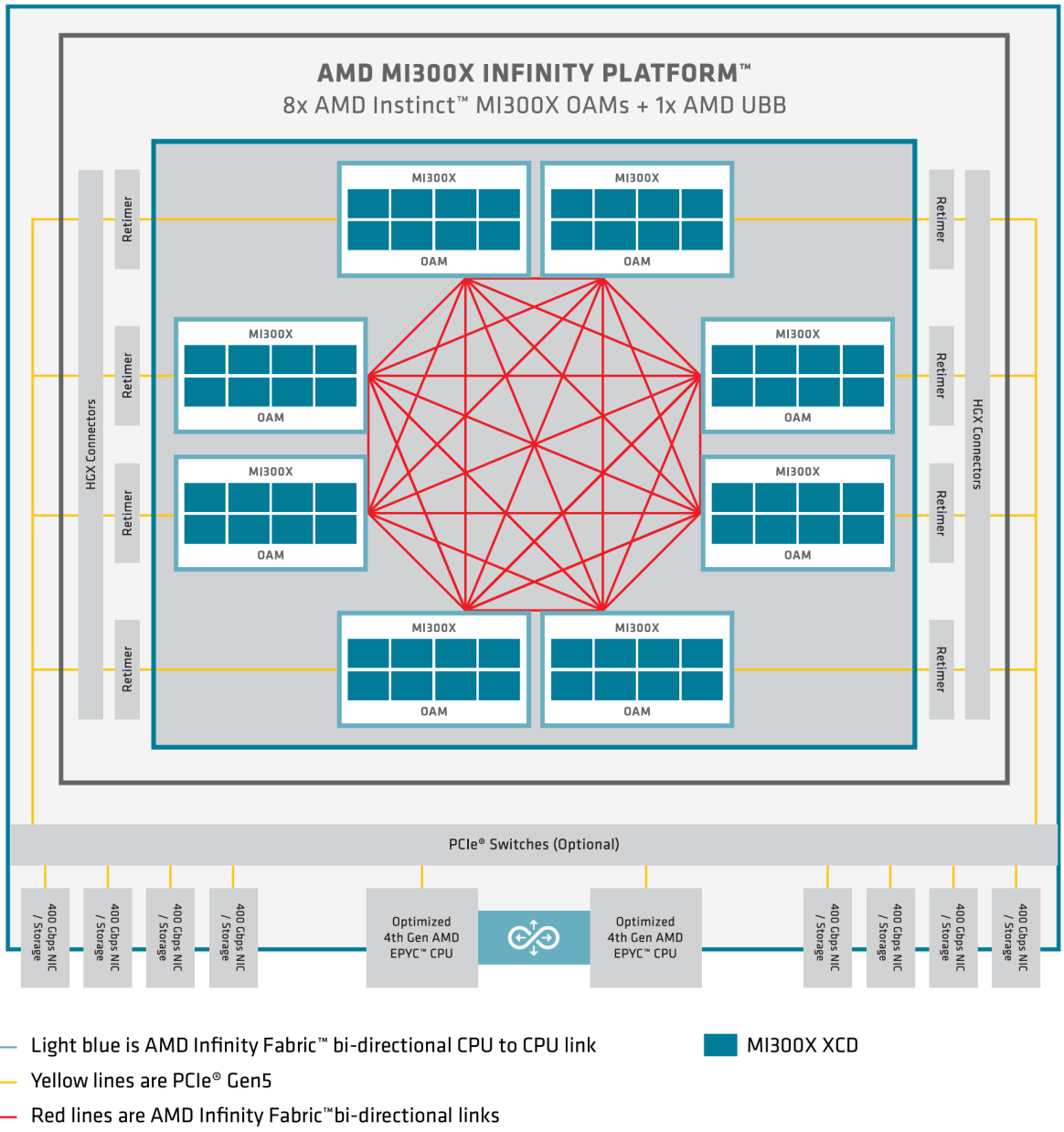
This is illustrated in Figure 12.

Figure 12: Inter-rail vs. Intra-rail GPU-GPU communication



On the AMD GPU servers specifically, the GPUs are connected via the *AMD Infinity fabric* (which provides bidirectional 7x128GB/s per GPU). This fabric consists of seven high-bandwidth low-latency links that create an interconnected 8-GPU mesh as shown in Figure 13.

Figure 13. AMD MI300XX architecture.

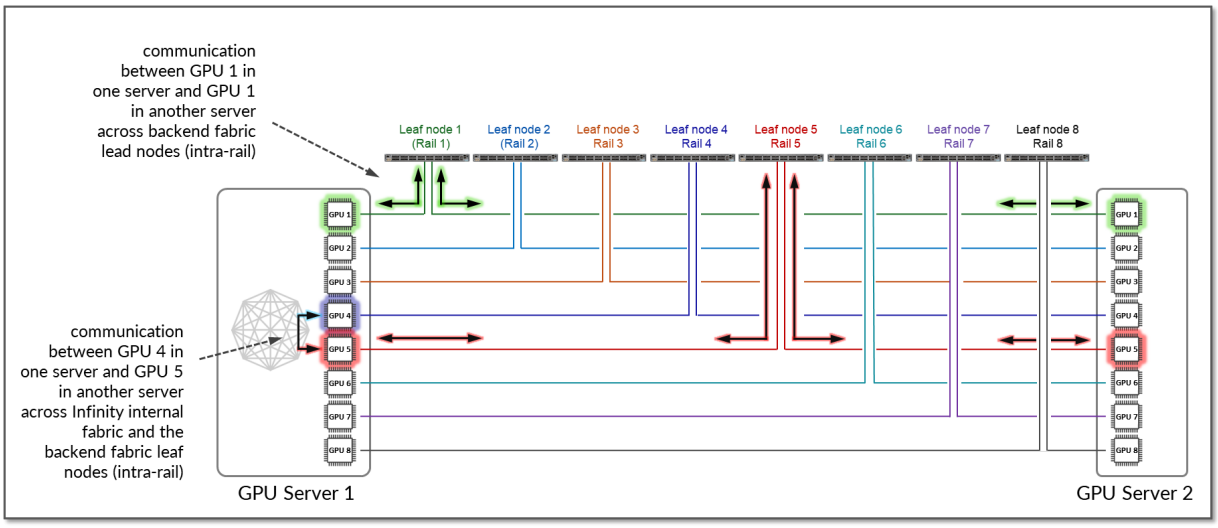


AMD MI300X GPUs leverage Infinity Fabric, to provide high-bandwidth, low-latency communication between GPUs, CPUs, and other components. This interconnect can dynamically manage traffic prioritization across links, providing an optimized path for communication within the node.

By default, AMD MI300X devices implement local optimization to minimize latency for GPU-to-GPU traffic. Traffic between GPUs of the same rank remains intra-stripe. Figure 14 shows an example where GPU1 in Server 1 communicates with GPU1 in Server 2. The traffic is forwarded by Leaf Node 1 and remains within Rail 1.

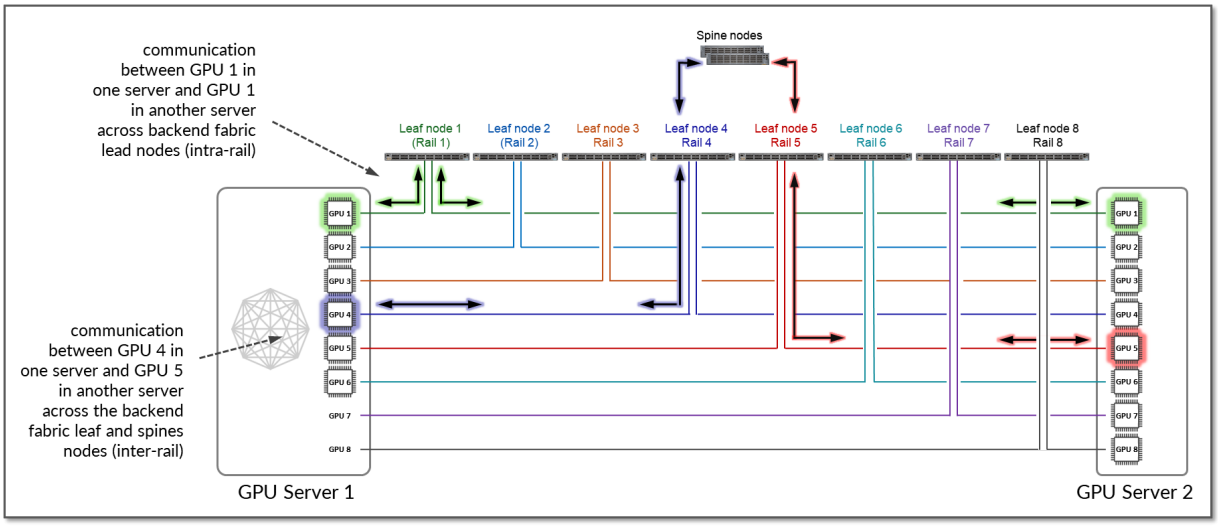
Additionally, if GPU4 in Server 1 wants to communicate with GPU5 in Server 2, and GPU5 in Server 1 is available as a local hop in AMD's Infinity Fabric, the traffic naturally prefers this path to optimize performance and keep GPU-to-GPU communication intra-rail.

Figure 14: GPU to GPU inter-rail communication between two servers **with local optimization**.



If local optimization is not feasible because of workload constraints, for example, the traffic must bypass local hops (internal fabric) and use RDMA (off-node NIC-based communication). In such case, GPU4 in Server 1 communicates with GPU5 in Server 2 by sending data directly over the NIC using RDMA, which is then forwarded across the fabric, as shown in Figure 15.

Figure 15: GPU to GPU inter-rail communication between two servers **without local optimization**.



Backend GPU Rail Optimized Architecture

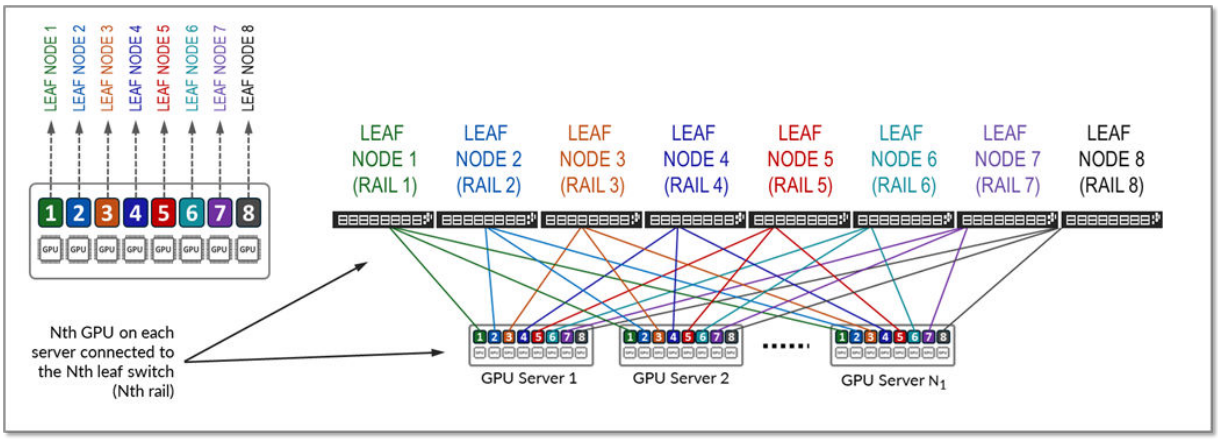
As previously described a Rail Optimized Stripe Architecture provides efficient data transfer between GPUs, especially during computationally intensive tasks such as AI Large Language Models (LLM) training workloads, where seamless data transfer is necessary to complete the tasks within a reasonable timeframe. A Rail Optimized topology aims to maximize performance by providing minimal bandwidth contention, minimal latency, and minimal network interference, ensuring that data can be transmitted efficiently and reliably across the network.

In a Rail Optimized Architecture there are two important concepts: **rail** and **stripe**.

The GPUs on a server are numbered 1-8, where the number represents the GPU's position in the server, as shown in Figure 6. This number is sometimes called **rank** or more specifically "**local rank**" in relationship to the GPUs in the server where the GPU sits, or "**global rank**" in relationship to all the GPUs (in multiple servers) assigned to a single job.

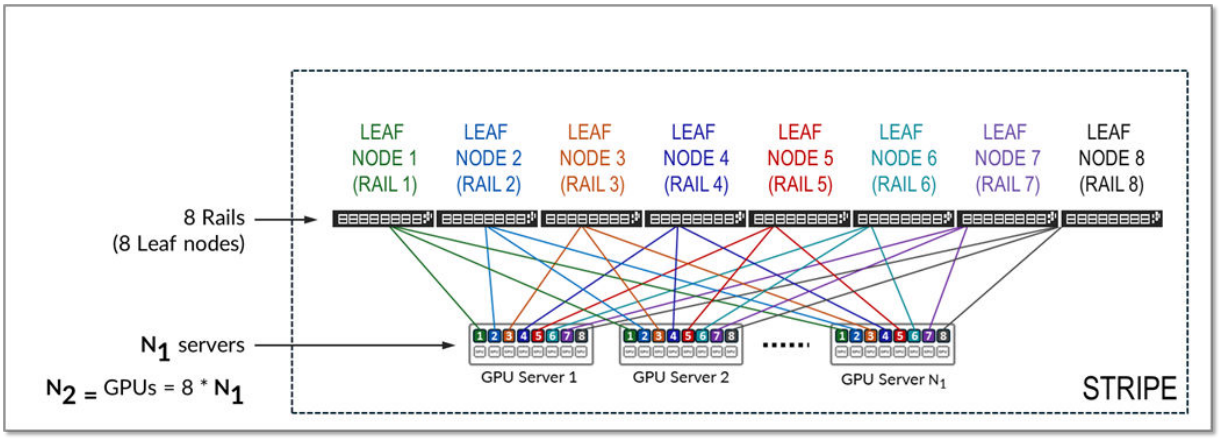
A **rail** connects GPUs of the same order across one of the leaf nodes in the fabric; that is, rail Nth connects all GPU in position Nth on all the servers, to leaf node Nth, as shown in Figure 6.

Figure 6: Rails in a Rail Optimized Architecture



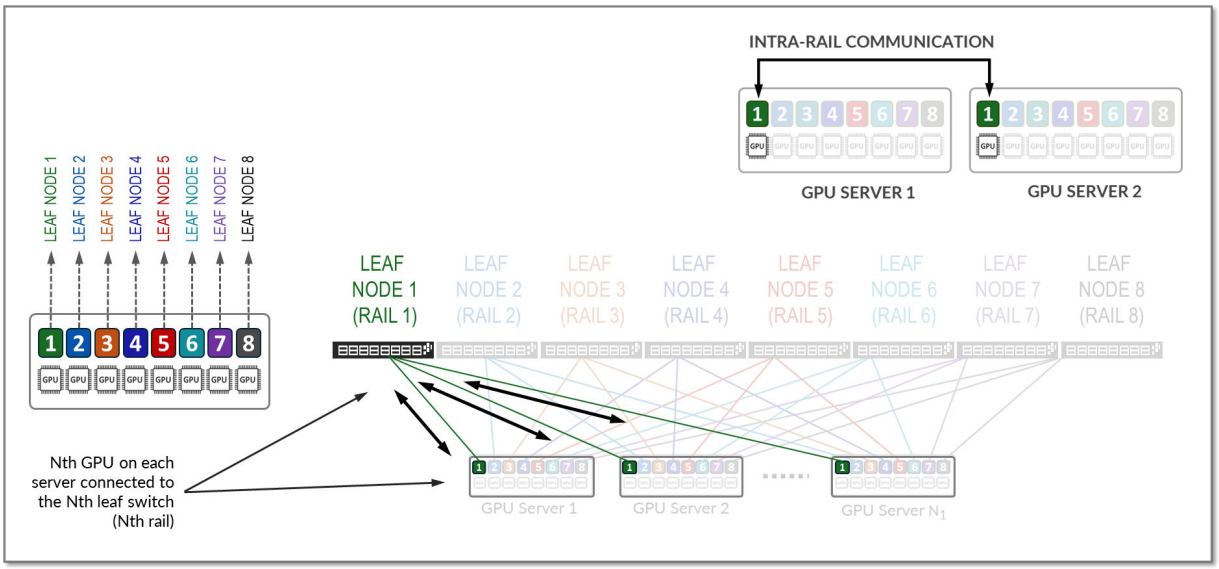
A **stripe** refers to a design module or building block, comprised of multiple **rails**, and that includes a number of Leaf nodes and GPU servers, as shown in Figure 7, that can be replicated to scale up the AI cluster.

Figure 7: Stripes in a Rail Optimized Architecture



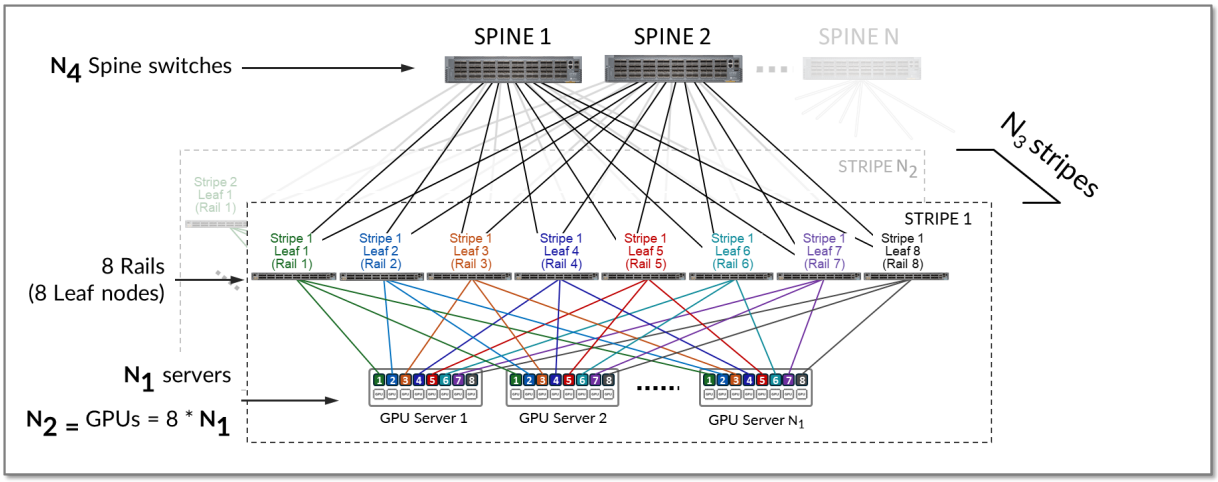
All traffic between GPUs of the same rank (intra-rail traffic) is forwarded at the leaf node level as shown in Figure 11.

Figure 11: Intra-rail traffic example.



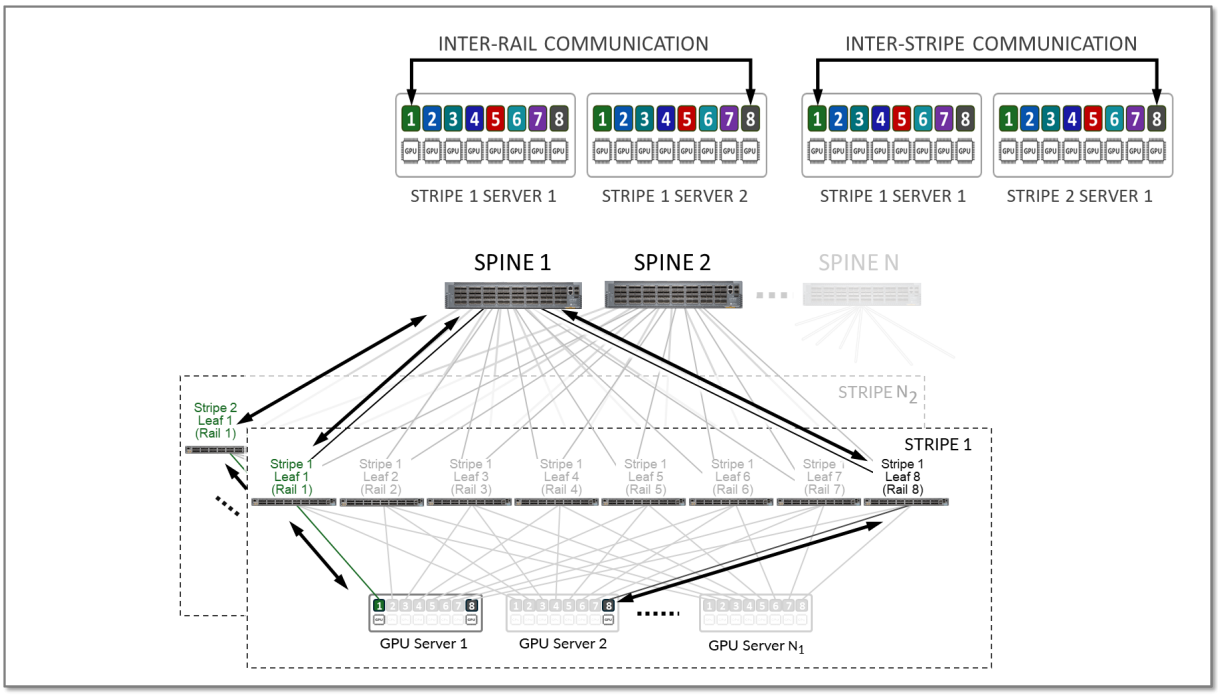
A stripe can be replicated to scale up the number of servers (N_1) and GPUs (N_2) in an AI cluster. Multiple stripes (N_3) are then connected across Spine switches as shown in Figure 8.

Figure 8: Multiple stripes connected via Spine nodes



Both Inter-rail and inter-stripe traffic will be forwarded across the spines nodes as shown in figure 9.

Figure 9. Inter-rail, and Inter-stripe GPU to GPU traffic example.



Calculating the number of leaf, spines, servers and GPUs.

The **number of leaf nodes** in a single stripe in a rail optimized architecture is defined by the number of GPUs per server (number of rails). Each AMD MI300X GPU server includes 8 AMD Instinct MI300Xx GPUs. Therefore, a single stripe includes 8 leaf nodes (8 rails).

Number of leaf nodes = number of GPUs x server = 8

The **maximum number of servers** supported in a single stripe (N1) is defined by the **number of available ports** on the Leaf node which depends on the switches model.

The total bandwidth between the GPU servers and leaf nodes must match the total bandwidth between leaf and spine nodes to maintain a 1:1 subscription ratio.

Assuming all the interfaces on the leaf node operate at the same speed, half of the interfaces will be used to connect to the GPU servers, and the other half to connect to the spines. Thus, the **maximum number of servers** in a stripe is calculated as half the **number of available ports** on each leaf node. Some examples are included in Table 7.

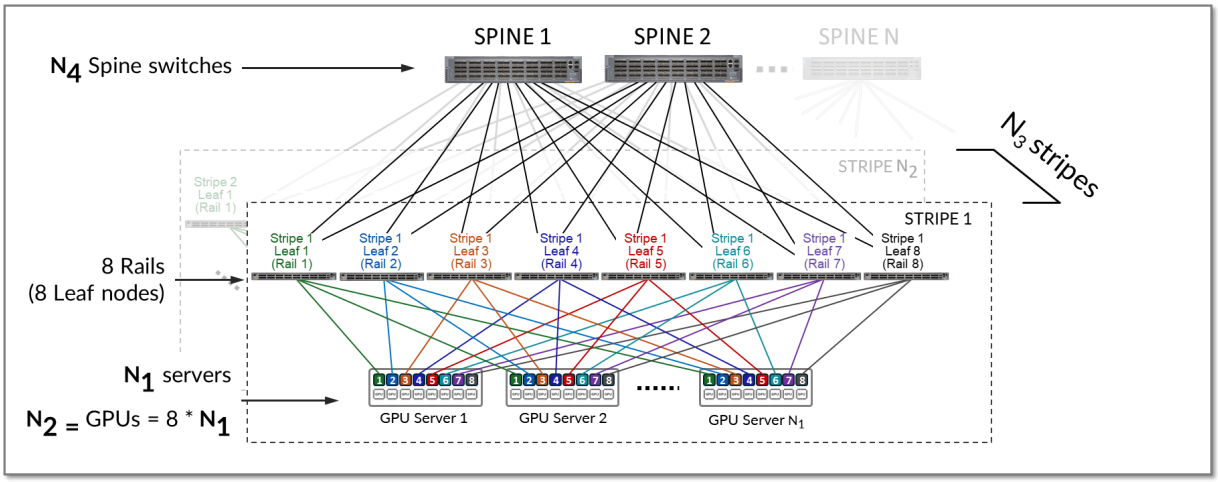
Table 7: Maximum number of GPUs supported per stripe

Leaf Node QFX switch Model	**number of available 400 GE ports per switch**	**Maximum number of servers supported per stripe for 1:1 Subscription (N_1)**	**GPUs per server**	**Maximum number of GPUs supported per stripe (N_2)**
QFX5220-32CD	32	32 ÷ 2 = 16	8	16 servers x 8 GPUs/ server = 128 GPUs
QFX5230-64CD	64	64 ÷ 2 = 32	8	32 servers x 8 GPUs/ server = 256 GPUs
QFX5240-64OD	128	128 ÷ 2 = 64	8	64 servers x 8 GPUs/ server = 512 GPUs

- QFX5220-32CD switches provide 32 x 400 GE ports => 16 will be used to connect to the servers and 16 will be used to connect to the spine nodes.
- QFX5230-64CD switches provide up to 64 x 400 GE ports => 32 will be used to connect to the servers and 32 will be used to connect to the spine nodes.
- QFX5240-64OD switches provide up to 128 x 400 GE ports => 64 will be used to connect to the servers and 64 will be used to connect to the spine nodes.

NOTE: QFX5240-64OD switches come with 64 x 800GE ports which can break out into 2x400GE ports, for a maximum of 128 400GE interfaces was shown in table 7.

To achieve larger scales, multiple stripes (N_3) can be connected using a set of Spine nodes (N_4), as shown in Figure 9.



The **number of stripes required** (N_3) is calculated based on the **required number of GPUs**, and the **maximum number of GPUs per stripe (N_2)**.

For example, assume that the **required number of GPUs (GPUs)** is 16,000 and the fabric is using QFX5240-64OD as leaf nodes.

The **number of available** 400G ports is 128 which means that:

- **maximum number of servers per stripe (N_1) = 64**
- **maximum number of GPUs per stripe (N_2) = 512**

To **number of stripes** (N_3) required is calculated by diving the number of GPUs required, and the number of GPUs per stripe as shown:

N_3 (number of stripes required) = GPUs ÷ N_2 (maximum number of GPUs per stripe) = 16000 ÷ 256 ≈ 64 stripes

- With 64 stripes & 256 servers per stripe the cluster can provide 16,384 GPUs.

Knowing the **number of stripes required** (N_3) and the **number of uplinks ports per leaf node** (Y) you can calculate how many spine nodes are required.

Remember X = Y = N1

First the **total number of leaf nodes** can be calculated by multiplying the **number of stripes required** by 8 (number of leaf nodes per stripe).

Total number of leaf nodes = N_3 x 8 = 64 x 8 = 512

Then the **total number of uplinks** can be obtained multiplying the **number of uplinks per leaf node** (N_1), and the **total number of leaf nodes**.

Total number of uplinks = N_1 x N_3 = 64 x 512 = 32768

The **number of spines required** (N_4) can then be determined by dividing the **total number of uplinks** by the **number of available ports on each spine node**, which as for the leaf nodes, depends on the switch model used for the spine role.

Number of spines required (N_4) = **32768 / number of available ports on each spine node**

For example, if the spine nodes are QFX5240, the **number of available ports on each spine node** is 128.

Table 8: Number of spines nodes for two stripes.

Spine Node QFX switch Model	Maximum number of 400 GE interfaces per switch	Number of spines required (N_4) with 64 stripes
QFX5240-64OD	128	**32768** ÷ 128 = 256
PTX10008	288	**32768** ÷ 288 ~ 128

Storage Backend Fabric

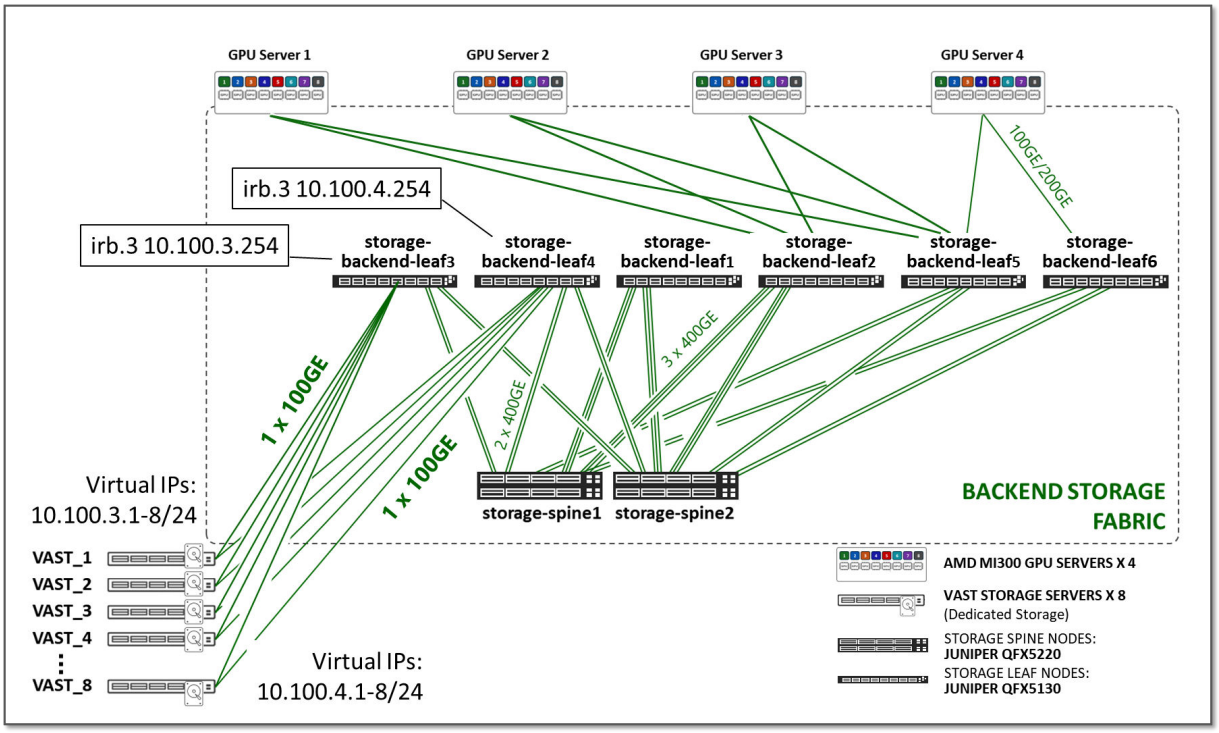
In small clusters, it may be sufficient to use the local storage on each GPU server, or to aggregate this storage together using open-source or commercial software. In larger clusters with heavier workloads, an external dedicated storage system is required to provide dataset staging for ingest, and for cluster checkpointing during training.

Two leading platforms, WEKA and Vast Storage, provide cutting-edge solutions for shared storage in GPU environments. While we have tested both solutions in our lab, this JVD focuses on the Vast Storage Solution. Thus, the rest of this, as well as other sections in this document will cover details about Vast Storage devices and connectivity to the Storage Backend Fabric. Details about the WEKA storage are included in the *AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD).*

The Storage Backend Fabric provides the connectivity infrastructure for storage devices to be accessible from the GPU servers. The performance of this fabric significantly impacts the efficiency of AI workflows. A storage system that provides quick access to data can significantly reduce the amount of time for training AI models. Similarly, a storage system that supports efficient data querying and indexing can minimize the completion time of preprocessing and feature extraction in an AI workflow.

The Storage Backend Fabric design in the JVD also follows a 3-stage IP clos architecture as shown in Figure 16. There is no concept of rail-optimization in a storage cluster. Each GPU server has single connections to the leaf nodes, instead of one connection per GPU.

Figure 16: Storage Backend Fabric Architecture



The Storage Backend devices included in this fabric, and the connections between them, are summarized in the following tables:

Table 9: Storage Backend devices

Number of GPU Servers	**Number of storage server**	**Storage Backend Leaf nodes switch model**	**Storage Backend Spine nodes (*storage-spine#)* switch model**
AMD MI300X x 4 (MI300X-#; # = 1-4)	Vast storage server x 8 (Vast#; #=1-8)	QFX5130-32CD x 6 (storage-leaf#; #=1-6) 4 *connecting the GPU servers*, and 2 *connecting the storage devices*	QFX5130-32CD x 2 (storage-spine#; #=1-2)

Number of GPU Servers	**Number of storage server**	**Storage Backend Leaf nodes (*storage-leaf#*) switch model**	**Storage Backend Spine nodes (*storage-spine#)* switch model**
AMD MI300X x 4 (MI300X-1 to MI300X-4)	Vast storage server x 8 (vast-1 to vast-8)	QFX5130-32CD x 6 (4 *connecting the GPU servers*, and 2 *connecting the storage devices*)	QFX5130-32CD x 2

QFX5230 and QFX5240 were also validated for the Storage Backend Leaf and Spine roles.

Table 10: Connections between servers, leaf and spine nodes in the Storage Backend

GPU Servers <=> Storage Backend Leaf Nodes	**Storage Servers <=> Storage Backend Leaf Nodes**	**Storage Backend Leaf Nodes <=> Storage Backend Spines nodes**
1 x 200GE links per GPU server to leaf node connection Total number of 200GE links between GPU servers and storage leaf nodes = 8 (4 servers x 2 leaf nodes x 1 link per server to leaf connections) " *(1)* " on page 28	1 x 100GE links per Vast C-node to leaf node connection Total number of 100GE links between Vast C-node and storage leaf nodes = 16 (8 c-nodes x 2 leaf nodes x 1 link per node to lead connection) " *(2)* " on page 28	2 x 400GE links OR Total number of 400GE links between storage Backend leaf nodes and spine nodes = 8 (4 leaf nodes x 2 spines nodes x 2 links per leaf to spine connection + (2 leaf nodes x 2 spines nodes x 3 links per leaf to spine connection

(1) AMD MI300X servers are dual homed

(2) Vast storage C-nodes are dual homed

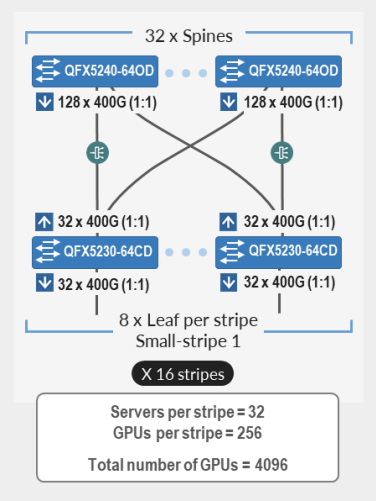
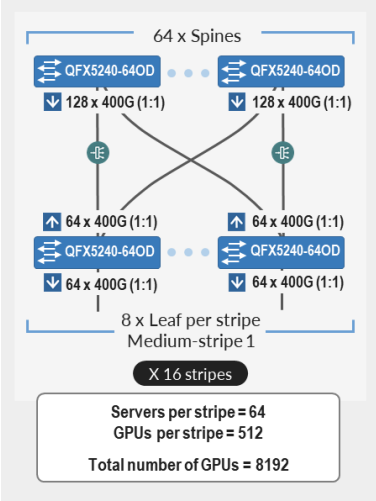
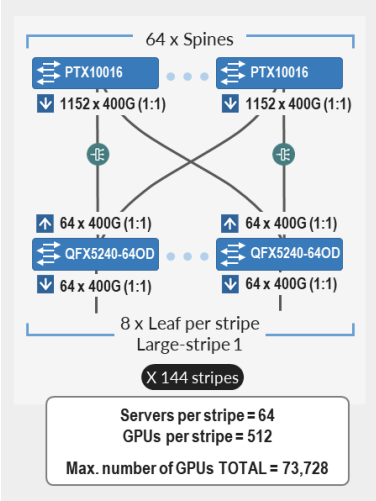
Scaling

The size of an AI cluster varies significantly depending on the specific requirements of the workload. The number of nodes in an AI cluster is influenced by factors such as the complexity of the machine learning models, the size of the datasets, the desired training speed, and the available budget. The number varies from a small cluster with less than 100 nodes to a data center-wide cluster comprised of 10000s of compute, storage, and networking nodes. A minimum of 4 spines must always be deployed for path diversity and reduction of PFC failure paths.

Table 11: Fabric Scaling- Devices and Positioning

Fabric Scaling		
Small	**Medium**	**Large**
upto 4096 GPU	upto 8192 GPU	8192 and upto 73,728 GPU
Support for up to 4096 GPUs with Juniper QFX5240-64CDs as spine nodes and QFX5230-64CD as leaf nodes (single or multi-stripe implementations). This 3-stage rail-based fabric consists of up to 32 Spines and 128 leaf nodes, maintaining a 1:1 subscription. The fabric provides physical connectivity for up to 16 stripes, with 32 servers (256 GPUs) per stripe, for a total of 4096 GPUs.	Support for more than 4096 GPU and upto 8192 GPUs with Juniper QFX5240-64CDs as both Spine and Leaf nodes. This 3-stage rail-based fabric consists of up to 64 spines, and up to 128 leaf nodes, maintaining a 1:1 subscription. The fabric provides physical connectivity for up to 16 Stripes, with 64 servers (512 GPUs) per stripe, for a total of 8192 GPUs	Support of more than 8192 GPU and upto . 73,728 GPUs with Juniper PTX10000 Chassis as Spine nodes and Juniper QFX5240-64CDs as leaf nodes. This 3-stage rail-based fabric consists of up to 64 spines, and up to 1152 leaf nodes, maintaining a 1:1 subscription. The fabric provides physical connectivity for up to 144 Stripes, with 64 servers (512 GPUs) per stripe, for a total of 73,728 GPUs.

(Continued)

Fabric Scaling		
Small	**Medium**	**Large**
		

NOTE: To follow best practices, a minimum of 4 Spines should be deployed, even in a single-stripe fabric.

Juniper Hardware and Software Solution Components

The Juniper products and software versions listed below pertain to the latest validated configuration for the AI DC use case. As part of an ongoing validation process, we routinely test different hardware models and software versions and update the design recommendations accordingly.

The version of Juniper Apstra in the setup is 5.0.0-a-12.

The following table summarizes the validated Juniper devices for this JVD and includes devices tested for the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD).

Table 12: Validated Devices and Positioning

Validated Devices and Positioning		
Fabric	**Leaf Switches**	**Spine Switches**
Frontend	QFX5130-32CD	QFX5130-32CD
GPU Backend	QFX5230-64CD (CLUSTER 1-STRIPE 1) QFX5220-32CD (CLUSTER 1-STRIPE 2) QFX5240-64CD/QFX5241-64CD (CLUSTER 2)	QFX5230-64CD (CLUSTER 1) PTX10008 JNP10K-LC1201 (CLUSTER 1) QFX5240-64CD/ QFX5241-64CD (CLUSTER 2)
Storage Backend	QFX5220-32CD QFX5230-64CD QFX5240-64CD/QFX5241-64CD	QFX5220-32CD QFX5230-64CD QFX5240-64CD/ QFX5241-64CD

The following table summarizes the software versions tested and validated by role.

Table 13: Platform Recommended Release

Platform	Role	Junos OS Release
QFX5240-64CD	GPU Backend Leaf	23.4X100-D20
QFX5241-64CD	GPU Backend Spine	23.4X100-D42
QFX5220-32CD	GPU Backend Leaf	23.4X100-D20
QFX5230-64CD	GPU Backend Leaf	23.4X100-D20
QFX5240-64CD	GPU Backend Spine	23.4X100-D20
QFX5241-64CD	GPU Backend Spine	23.4X100-D42
QFX5230-64CD	GPU Backend Spine	23.4X100-D20

(Continued)

Platform	Role	Junos OS Release
PTX10008 with LC1201	GPU Backend Spine	23.4R2-S3
QFX5130-32CD	Frontend Leaf	23.43R2-S3
QFX5130-32CD	Frontend Spine	23.43R2-S3
QFX5220-32CD	Storage Backend Leaf	23.4X100-D20
QFX5230-64CD	Storage Backend Leaf	23.4X100-D20
QFX5240-64CD	Storage Backend Leaf	23.4X100-D20
QFX5241-64CD	Storage Backend Leaf	23.4X100-D42
QFX5220-32CD	Storage Backend Spine	23.4X100-D20
QFX5230-64CD	Storage Backend Spine	23.4X100-D20
QFX5240-64CD	Storage Backend Spine	23.4X100-D20
QFX5241-64CD	Storage Backend Spine	23.4X100-D42

NOTE: For minimum software released for QFX5220-64CD, QFX5230-64CD, PTX10008 in the GPU backend fabric, check the *Recommendations Section i*n the *AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD)* .

IP Services for AI Networks

In the next few sections, we describe the various strategies that can be employed to handle traffic congestion and traffic load distribution in the Backend GPU fabric.

Congestion Management

AI clusters pose unique demands on network infrastructure due to their high-density, and low-entropy traffic patterns, characterized by frequent elephant flows with minimal flow variation. Additionally, most AI modes require uninterrupted packet flow with no packet loss for training jobs to be completed.

For these reasons, when designing a network infrastructure for AI traffic flows, the key objectives include maximum throughput, minimal latency, and minimal network interference over a lossless fabric, resulting in the need to configure effective congestion control methods.

Data Center Quantized Congestion Notification (DCQCN), has become the industry-standard for end-to-end congestion control for RDMA over Converged Ethernet (RoCEv2) traffic. DCQCN congestion control methods offer techniques to strike a balance between reducing traffic rates and stopping traffic all together to alleviate congestion, without resorting to packet drops.

It is important to note that DCQCN is primarily required in the GPU backend fabric, where the majority of AI workload traffic resides, while it is generally unnecessary in the frontend or storage backend."

DCQCN combines two different mechanisms for flow and congestion control:

- Priority-Based Flow Control (PFC), and
- Explicit Congestion Notification (ECN).

Priority-Based Flow Control (PFC) helps relieve congestion by halting traffic flow for individual traffic priorities (IEEE 802.1p or DSCP markings) mapped to specific queues or ports. The goal of PFC is to stop a neighbor from sending traffic for an amount of time (PAUSE time), or until the congestion clears. This process consists of sending PAUSE control frames upstream requesting the sender to halt transmission of all traffic for a specific class or priority while congestion is ongoing. The sender completely stops sending traffic to the requesting device for the specific priority.

While PFC mitigates data loss and allows the receiver to catch up processing packets already in the queue, it impacts performance of applications using the assigned queues during the congestion period. Additionally, resuming traffic transmission post-congestion often triggers a surge, potentially exacerbating or reinstating the congestion scenario.

We recommend configuring PFC only on the QFX devices acting as leaf nodes.

Explicit Congestion Notification (ECN), on the other hand, curtails transmit rates during congestion while enabling traffic to persist, albeit at reduced rates, until congestion subsides. The goal of ECN is to reduce packet loss and delay by making the traffic source decrease the transmission rate until the congestion clears. This process entails marking packets with ECN bits at congestion points by setting the ECN bits to 11 in the IP header. The presence of this ECN marking prompts receivers to generate Congestion Notification Packets (CNPs) sent back to source, which signal the source to throttle traffic rates.

Combining PFC and ECN offers the most effective congestion relief in a lossless IP fabric supporting RoCEv2, while safeguarding against packet loss. To achieve this, when implementing PFC and ECN together, their parameters should be carefully selected so that ECN is triggered before PFC.

For more information refer to *Introduction to Congestion Control in Juniper AI Networks* which explores how to build a lossless fabric for AI workloads using DCQCN (ECN and PFC) congestion control methods and DLB. The document was based on DLRM training model as a reference and demonstrates how different congestion parameters such as ECN and PFC counters, input drops and tail drops can be monitored to adjust configuration and build a lossless fabric infrastructure for RoCEv2 traffic.

NOTE: We provide general recommendations and describe the parameters validated in the lab. However, each language model has a unique traffic profile and characteristics. Class of Service and load balancing attributes must be tuned to meet your specific model requirements.

Load Balancing

The fabric architecture used in this JVD for both the Frontend and backend follows the 2-stage clos design, with every leaf node connected to all the available spine nodes, and via multiple interfaces. As a result, multiple paths are available between the leaf and spine nodes to reach other devices.

AI traffic characteristics may impede optimal link utilization when implementing traditional Equal Cost Multiple Path (ECMP) Static Load Balancing (SLB) over these paths. This is because the hashing algorithm which looks at specific fields in the packet headers will result in multiple flows mapped to the same link due to their similarities. Consequently, certain links will be favored, and their high utilization may impede the transmission of smaller low-bandwidth flows, leading to potential collisions, congestion and packet drops. To improve the distribution of traffic across all the available paths either Dynamic Load Balancing (DLB) or Global Load Balancing (GLB) can be implemented instead.

For this JVD Dynamic Load Balancing flowlet-mode was implemented on all the QFX leaf and spines nodes. Global Load Balancing is also included as an alternative solution.

Additional testing was conducted on the QFX5240-64OD in the "GPU Backend Fabric" on page 13, to evaluate the benefits of Selective Dynamic Load Balancing, and Reactive path rebalancing. Notice that these load balancing mechanisms are only available on QFX devices.

Dynamic Load Balancing (DLB)

Dynamic Load Balancing (DLB) ensures that all paths are utilized more fairly, by not only looking at the packet headers, but also considering real-time link quality based on port load (link utilization) and port

queue depth when selecting a path. This method provides better results when multiple long-lived flows moving large amounts of data need to be load balanced.

DLB can be configured in two different modes:

- Per packet mode: packets from the same flow are sprayed across link members of an IP ECMP group, which can cause packets to arrive out of order.

- Flowlet Mode: packets from the same flow are sent across a link member of an IP ECMP group. A flowlet is defined as bursts of the same flow separated by periods of inactivity. If a flow pauses for longer than the configured inactivity timer, it is possible to reevaluate the link members' quality, and for the flow to be reassigned to a different link.

Selective Dynamic Load Balancing (SDLB): allows implementing DLB only to certain traffic. This feature is only supported on QFX5240-64OD, and QFX5240-64QD, starting in Junos OS Evolved Release 23.4R2, at the time of this document's publication.

Reactive Path Rebalancing: allows a flow to be reassigned to a different (better) link, when the current link quality deteriorates, even if no pause in the traffic flow has exceeded the configured inactivity timer. This feature is only supported on QFX5240-64OD, and QFX5240-64QD, starting in Junos OS Evolved Release 23.4R2, at the time of this document's publication.

Global Load Balancing (GLB):

GLB is an improvement on DLB which only considers the local link bandwidth utilization. GLB on the other hand, has visibility into the bandwidth utilization of links at the next-to-next-hop (NNH) level. As a result, GLB can reroute traffic flows to avoid traffic congestion farther out in the network than DLB can detect.

AI-ML data centers have less entropy and larger data flows than other networks. Because hash-based load balancing does not always effectively load-balance large data flows of traffic with less entropy, dynamic load balancing (DLB) is often used instead. However, DLB considers only the local link bandwidth utilization. For this reason, DLB can effectively mitigate traffic congestion only on the immediate next hop. GLB more effectively load-balances large data flows by taking traffic congestion on remote links into account.

GLB is only supported for QFX-5240 (TH5) starting on 23.4R2 and 24.4R1, requires a full 3-tier CLOS architecture, and is limited to only one link between each spine and leaf. When there is more than one interface or a bundle between a pair of leaf and spine, GLB won't work. Also, GLB supports 64 profiles in the table. This means there can be 64 leaves in the 3-stage Clos topology where GLB is running.

For additional details on the operation and configuration of GLB refer to Avoiding AI/ML traffic congestion with global load balancing | HPE Juniper Networking Blogs

ADDITIONAL REFERENCES: ***Introduction to Congestion Control in Juniper AI Networks*** **explores how to build a lossless fabric for AI workloads using DCQCN (ECN and PFC) congestion control methods and DLB. The document was based on DLRM training model as a reference and demonstrates how different**

congestion parameters such as ECN and PFC counters, input drops and tail drops can be monitored to adjust configuration and build a lossless fabric infrastructure for RoCEv2 traffic.

Load Balancing in the Data Center provides a comprehensive deep dive into the various load-balancing mechanisms and their evolution to suit the needs of the data center.

Ethernet Network Adapter (NICs) for AI Data centers

AI/ML workloads have increased in complexity and scale; networking becomes crucial for efficient job completion times. The Network Adapters (NIC) are the connection points that connect the GPUs to the data center fabrics and hence these NICs should be able to handle large amounts of data and should be able to support high-speed, low-latency communications between GPU servers. Due to this at a minimum, the NICs should be able to support some of the key AI/ML functionality such as below:

- RDMA over Converged Ethernet (RoCE) and congestion control.
- Ability to handle 400G data bidirectional with low latency.
- Advanced congestion control mechanisms that are sensitive and able to react to network congestion and optimize traffic flow.
- Support GPU scalability ensuring robust performance even with increasing GPUs.

For Server NICs, we have two options:

- Broadcom Thor2—The Broadcom Thor2 network adapters were validated for AI/ML workloads and job completion times.
- AMD Pollara—AMD Pollara 400 ethernet network adapters

For more information on AMD Pensando Pollara 400 (ethernet adapter) refer to this link.

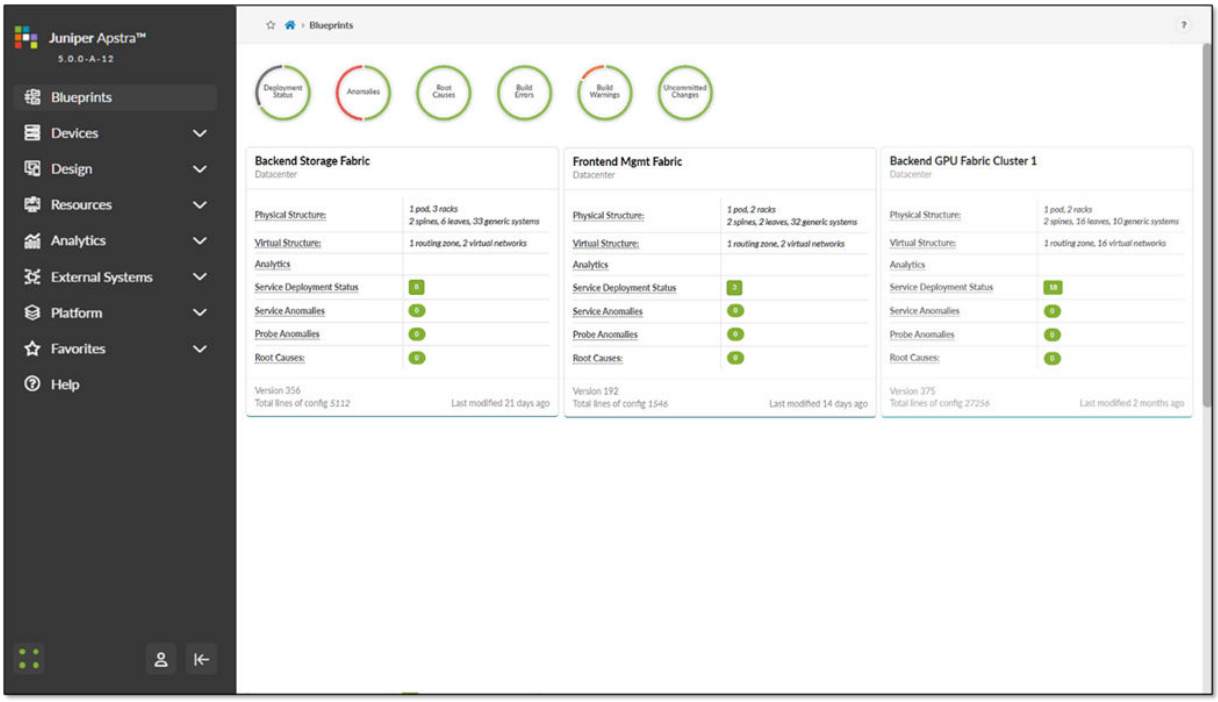
Fabric configuration Walkthrough using Juniper Apstra

IN THIS SECTION

This section describes the steps to deploy the AI GPU Backend IP fabrics in the AI JVD lab, as an example of how to deploy a fabric using Juniper Apstra. These steps will cover the AI GPU Backend IP fabric using QFX5240-64CD switches in the spine and leaf role along with AMD GPU servers and Vast storage devices.

The Apstra Blueprints for the Frontend, Storage Backend, and GPU backend for cluster 1 and cluster 2 fabrics have been created and can be seen in the Blueprint tab, as shown in Figure 17. We will show the steps to create GPU backend for cluster 2 blueprint as an example. Similar steps should be followed to set up the Frontend and Storage Backend fabrics. The configurations for these are included in the Terraform repository described in the next section.

Figure 17: AI Fabric Blueprints in Apstra



Setting up the Apstra Server and Apstra ZTP Server

A configuration wizard launches upon connecting to the Apstra server VM for the first time. At this point, passwords for the Apstra server, Apstra UI, and network configuration can be configured.

For more detailed information about installation and step-by-step configuration with Apstra, refer to the Juniper Apstra User Guide. Additional guidance in this walkthrough is provided in the form of notes.

Onboarding devices in the Apstra WEB UI

There are two methods for adding Juniper devices into Apstra: manually (recommended method) or in bulk using ZTP.

To add devices manually:

- In the Apstra UI navigate to **Devices >> Agents >> Create Offbox Agents**:
- This requires that the devices are preconfigured with a root password, a management IP and proper static routing if needed, as well as ssh Netconf, so that they can be accessed and configured by Apstra.

To add devices via ZTP:

- From the Apstra ZTP server, follow the ZTP steps described in the Juniper Apstra User Guide.

NOTE: Apstra imports the configuration from the devices into a baseline configuration called **pristine configuration**, which is a clean, minimal, and free of any pre-existing settings that could interfere with the intended network design managed by Apstra.

Apstra ignores the Junos configuration 'groups' stanza and does not validate any group configuration listed in the inheritance model, refer to the configuration groups usage guide.

It is best practice to avoid setting loopbacks, interfaces (except management interface), routing-instances (except management-instance) or any other settings as part of this baseline configuration.

Apstra sets the protocols LLDP and RSTP when the device is successfully Acknowledged.

The QFX5240 devices were onboarded using the manual method following these steps in the **Apstra Web UI**:

Step 1: Create **Agent Profile**.

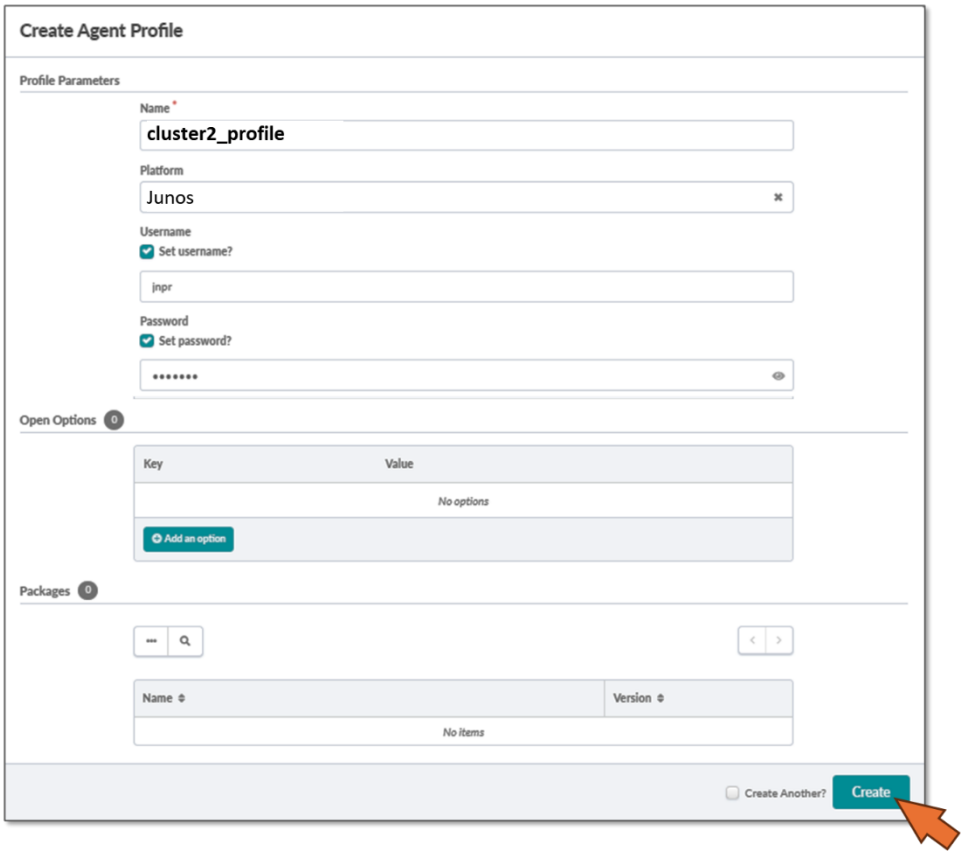
To create an Agent Profile, navigate to **Devices** >> **Agent Profiles** and then click on **Create Agent Profile**.

Figure 18. Create Agent Profile in Apstra

Enter the Agent profile name, select platform (Junos), and enter the username and password that will be used by Apstra to communicate with the devices. Make sure the devices are configured with these same username and password and have ssh Netconf enabled under system services.

For the purposes of this JVD, the same username and password are used across all devices. Thus, only one Apstra Agent Profile is needed to onboard all the devices, making the process more efficient.

Figure 19: Apstra Agent Profile Parameters



After entering the required information click **Create** and confirm Agent Profile creation.

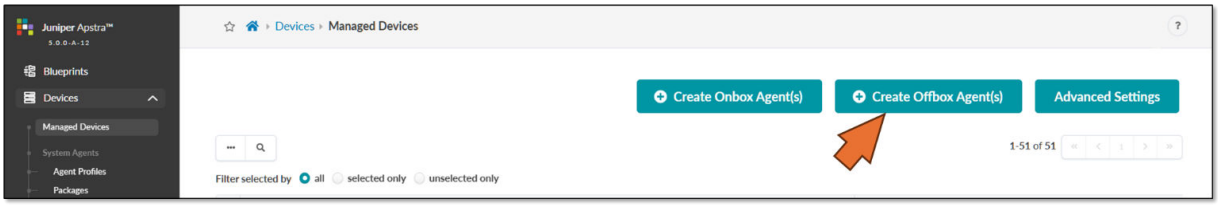
Figure 20: New Apstra Agent Profile Verification

Step 2: Create **Offbox Agents** for **QFX devices.**

To create Offbox Agents for the QFX devices navigate to **Devices** >> **Managed Devices** and then click on **Create Offbox Agents**.

Make sure you select **Offbox Agent(s).**

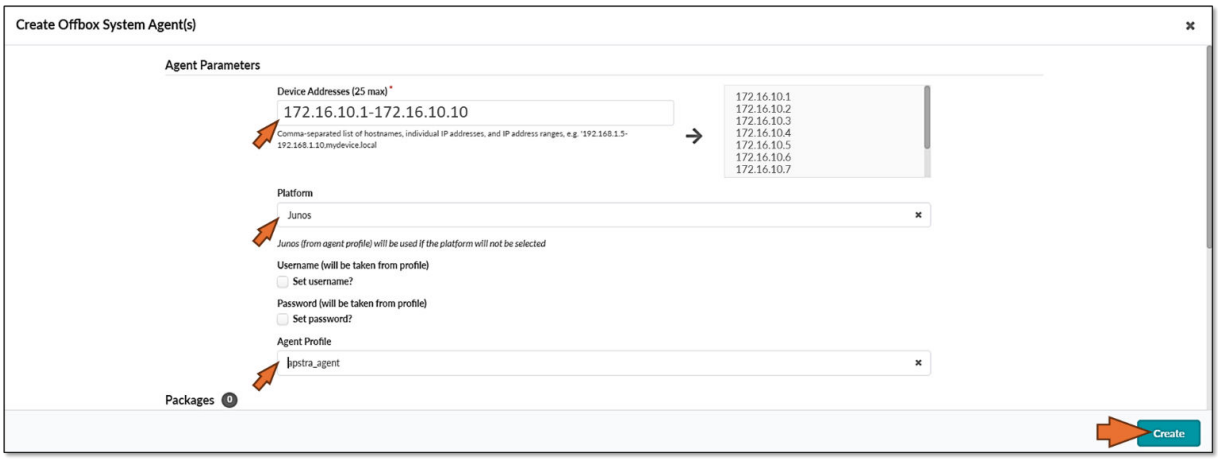
Figure 21: Create Offbox Agent in Apstra



Identify the device(s) to onboard. You can enter a comma-separated list of hostnames, individual IP addresses, or IP address ranges. An IP address range allows you to onboard multiple devices at once.

Select the platform, and **Agent Profile** (select the profile created in the previous step).

Figure 22: Adding a Range of IP Addresses in Apstra

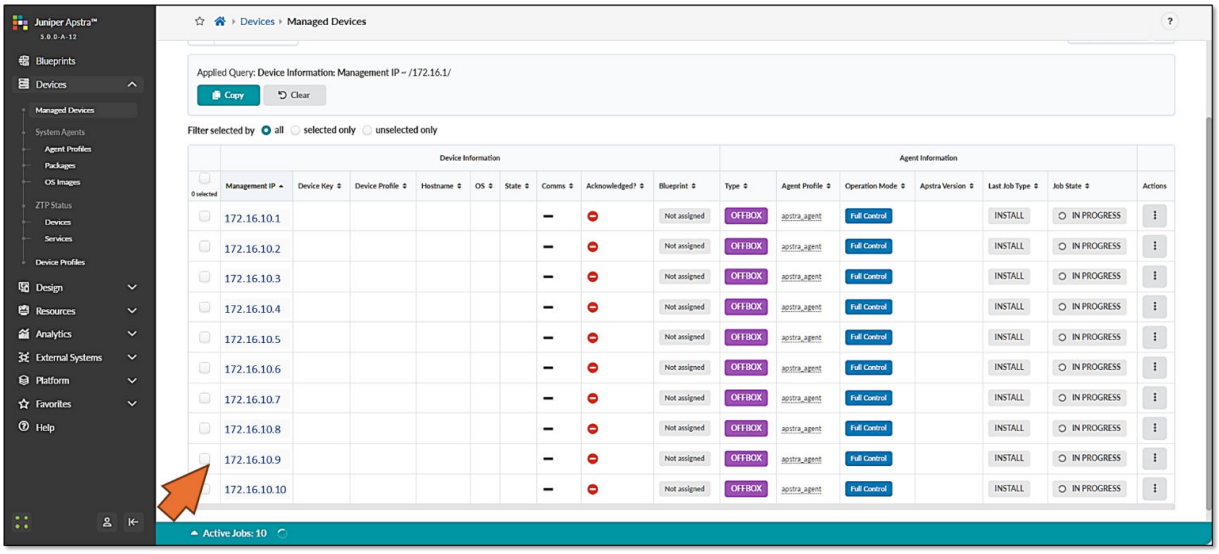


Apstra uses the information from the profile that was created in the previous step, if " ***Set username?"*** and " ***Set password?"*** are unchecked,

After entering the required information click **Create**.

Confirm that the devices have been added for onboarding. Once the offbox agent has been created, devices will be added to the list of Managed Devices. Apstra will attempt to connect and if successful it will populate the relevant information.

Figure 23: New devices being onboarded

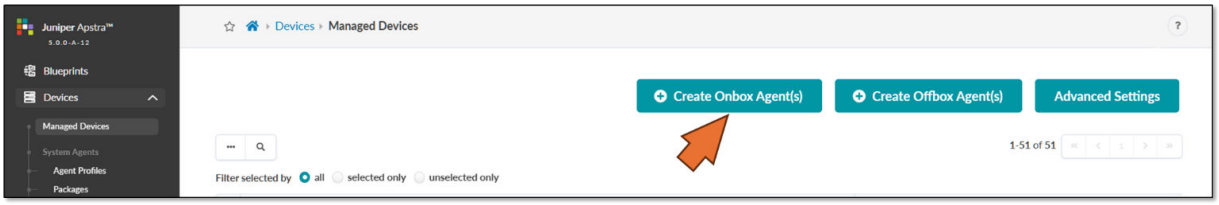


Step 3: Create **Onbox Agents** and onboard AMD servers.

To create Offbox Agents for the AMD servers navigate to **Devices** >> **Managed Devices** and then click on **Create Offbox Agents**.

Make sure you select **Onbox Agent(s).**

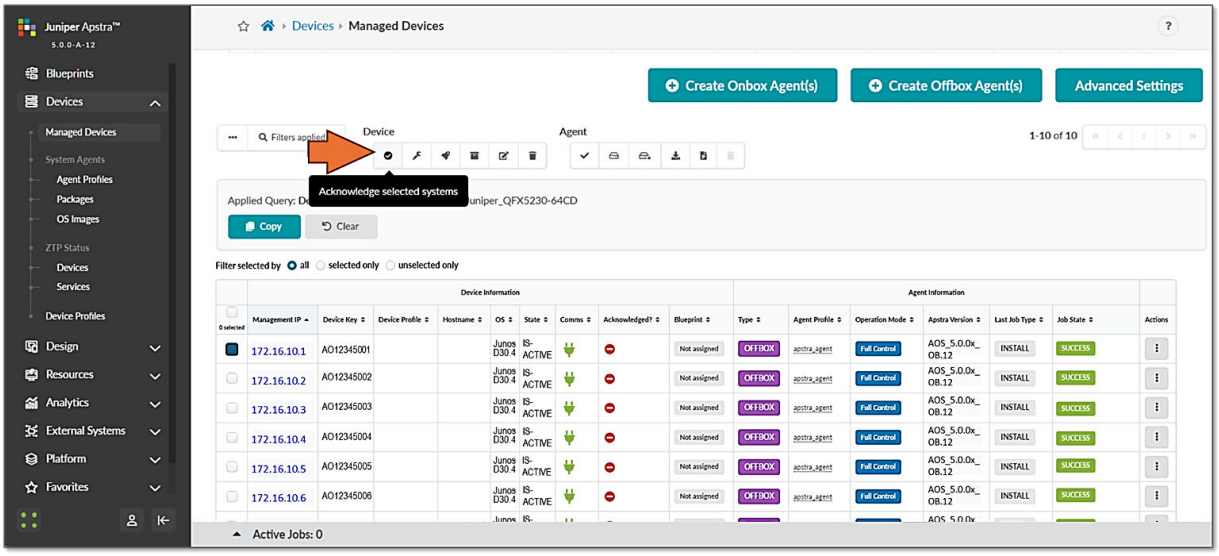
Figure 24: Create Offbox Agent in Apstra



Step 4: Acknowledge Managed Devices for Use in Apstra Blueprints.

The devices must be acknowledged by the user to complete the onboarding and allow them to be part of Apstra Blueprints.

Figure 25: Acknowledging Managed Devices in Apstra Blueprints



Fabric Provisioning in the Apstra Web UI

The following steps outline the provisioning of the GPU Backend Fabric with Apstra.

Step 1: Create Interface Maps and Logical Devices for the QFX5240 devices.

NOTE: The QFX5240s port numbering in Junos OS Release 23.4R2 was modified. Table 14 shows the differences between the old and the new port mappings.

Table 14. QFX5240-64DC port mappings

OLD PORT MAPPING (22.2X100															
0 (8x100G)	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
1 (unused)	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
32 (8x100G)	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
33 (unused)	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63
NEW PORT MAPPING (23.4R2)															
0 (8x100G)	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
1 (2x400G or 1x800G)	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
2 (8x100G)	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
3 (2x400G or 1x800G)	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

The Interface Map and Logical Devices for the QFX5240-64DC leaf and spine nodes were created following the new port mapping as shown in Figures 26-29

Figure 26: Apstra Interface Map for the QFX5240 Spine Nodes

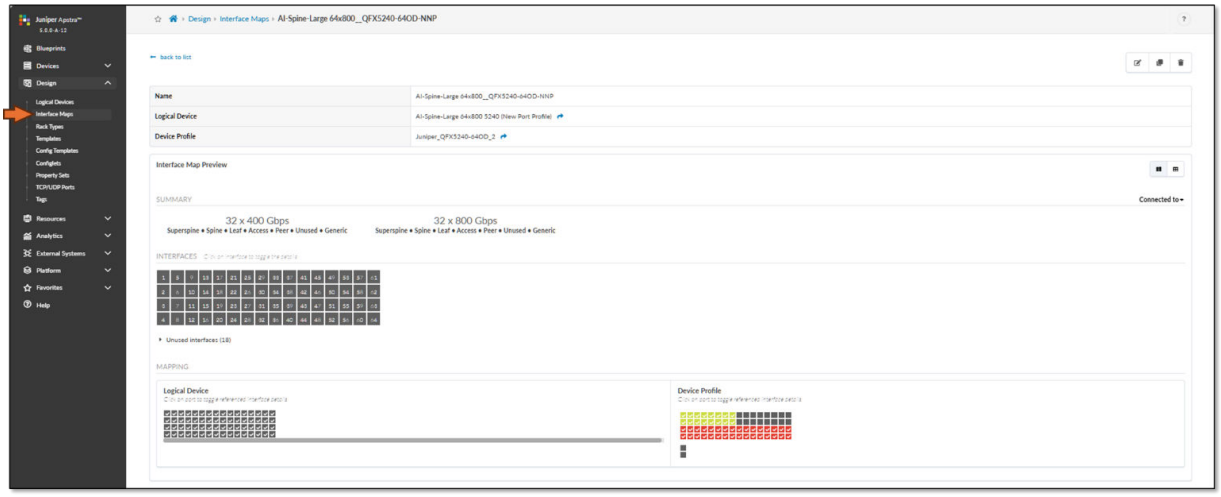


Figure 27: Apstra Logical Device for the QFX5240 Spine Nodes

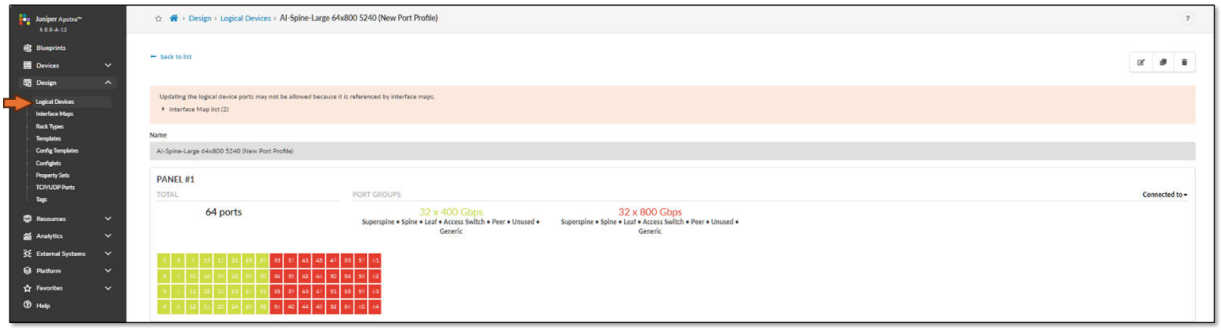


Figure 28: Apstra Interface Map for the QFX5240 Leaf Nodes

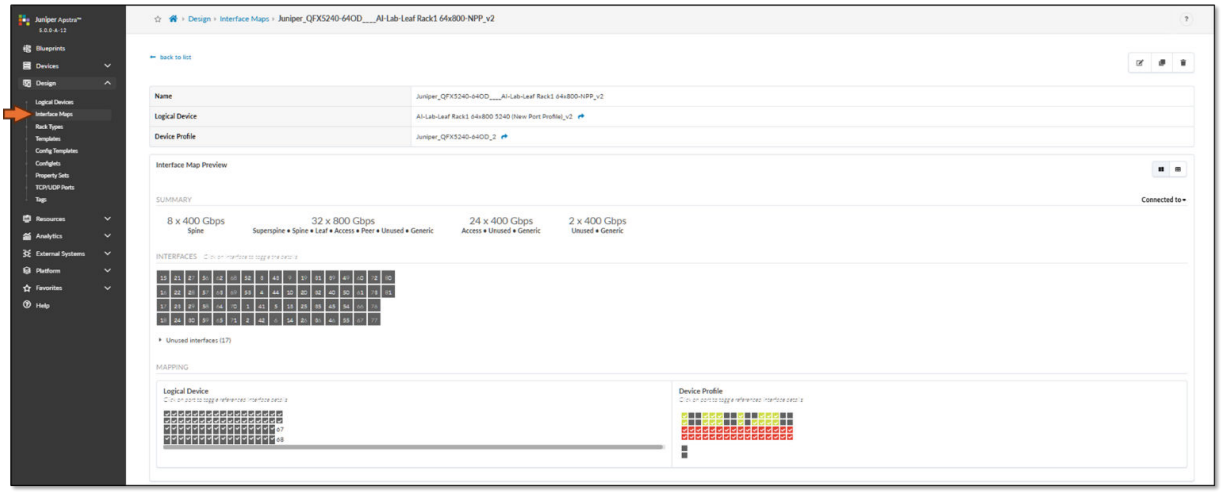


Figure 29: Apstra Logical Device for the QFX5240 Leaf Nodes

Step 2: Create Interface Map and Logical Device for the AMD GPU servers.

The logical Device and Interface Map for the AMD MI300X GPU servers are shown in Figures 30-31 respectively.

Figure 30: Apstra Interface Map for the AMD Servers

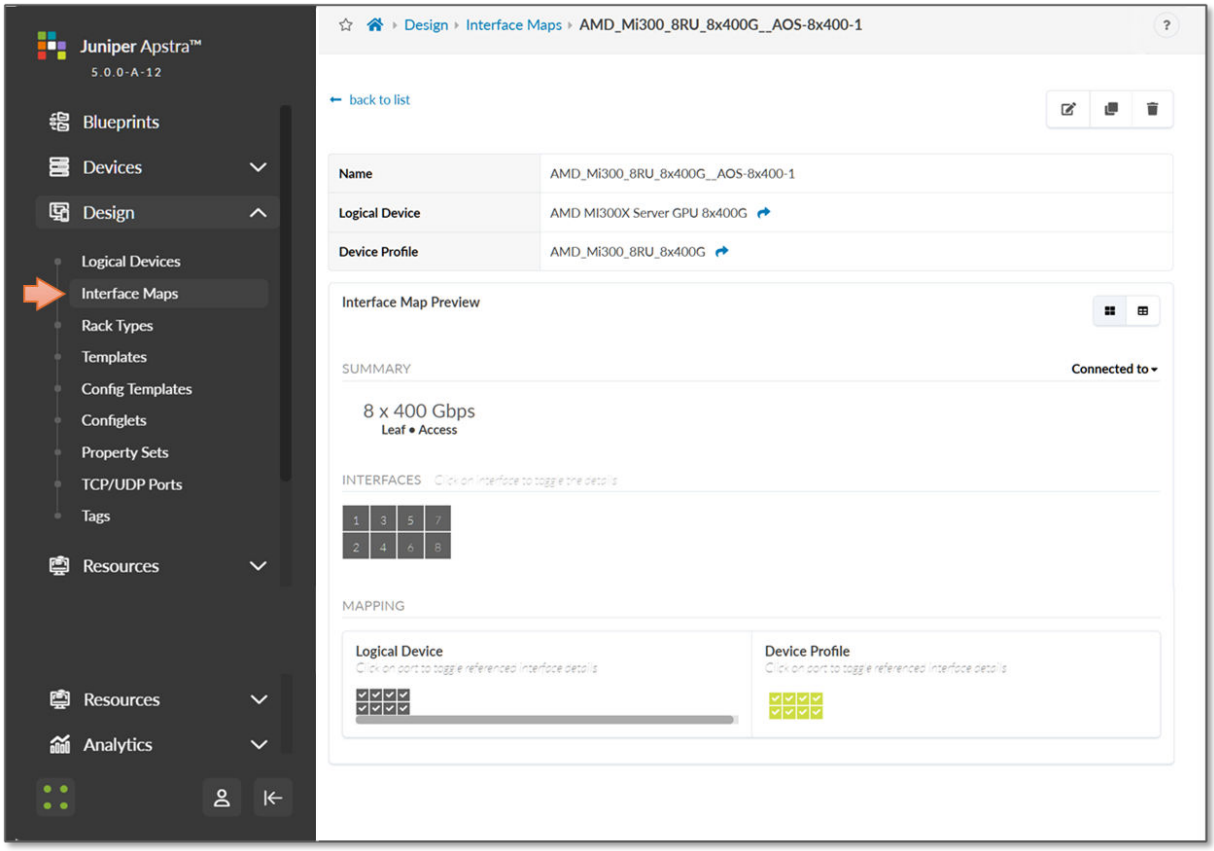


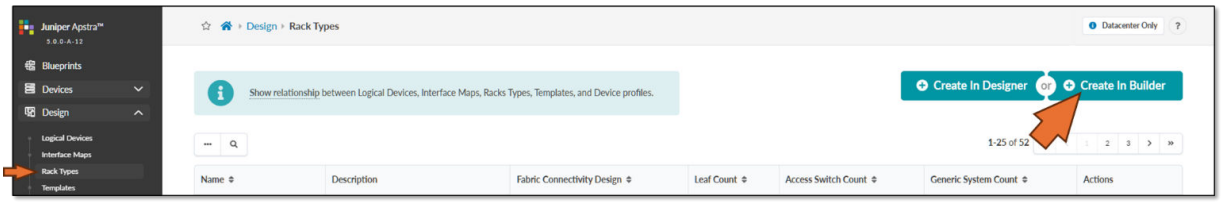
Figure 31: Apstra Logical Device for the AMD Servers

Step 3: Create Rack type.

In Apstra, a Rack is technically equivalent to a stripe in the context of an AI Fabric

To create rack types and templates for the GPU Backend fabric, referencing the Logical Devices created in the previous step, navigate to **Design → Rack Types → Create in Builder**, as shown in Figure 32.

Figure 32: Create Rack Type in Apstra



You can choose between **Create In Builder** or **Create In Designer** (graphical version). We demonstrate the **Create In Builder** option here.

Enter a Rack type, and description, and select L3 Clos.

Figure 33: Creating a Rack in Apstra using the **Create In Builder** option.

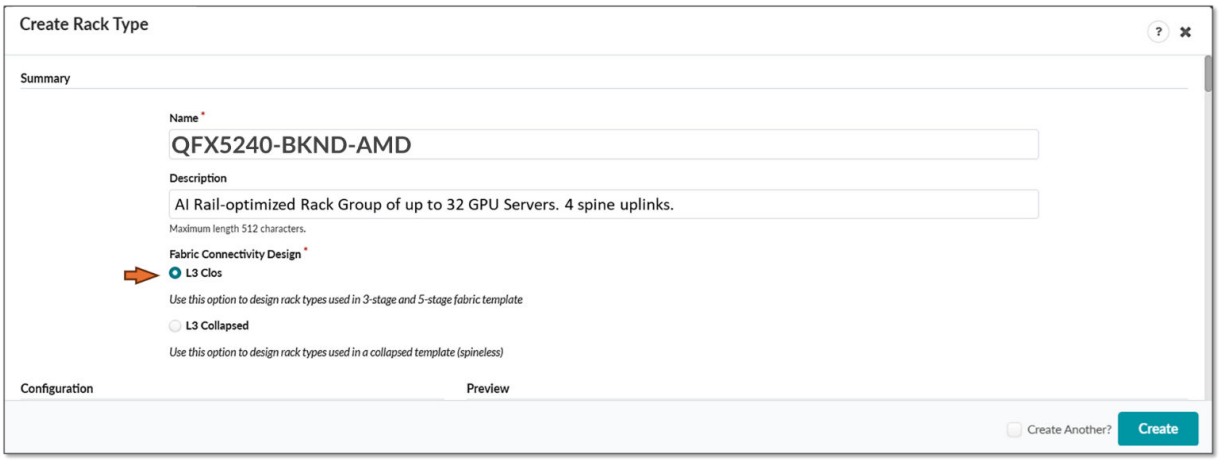
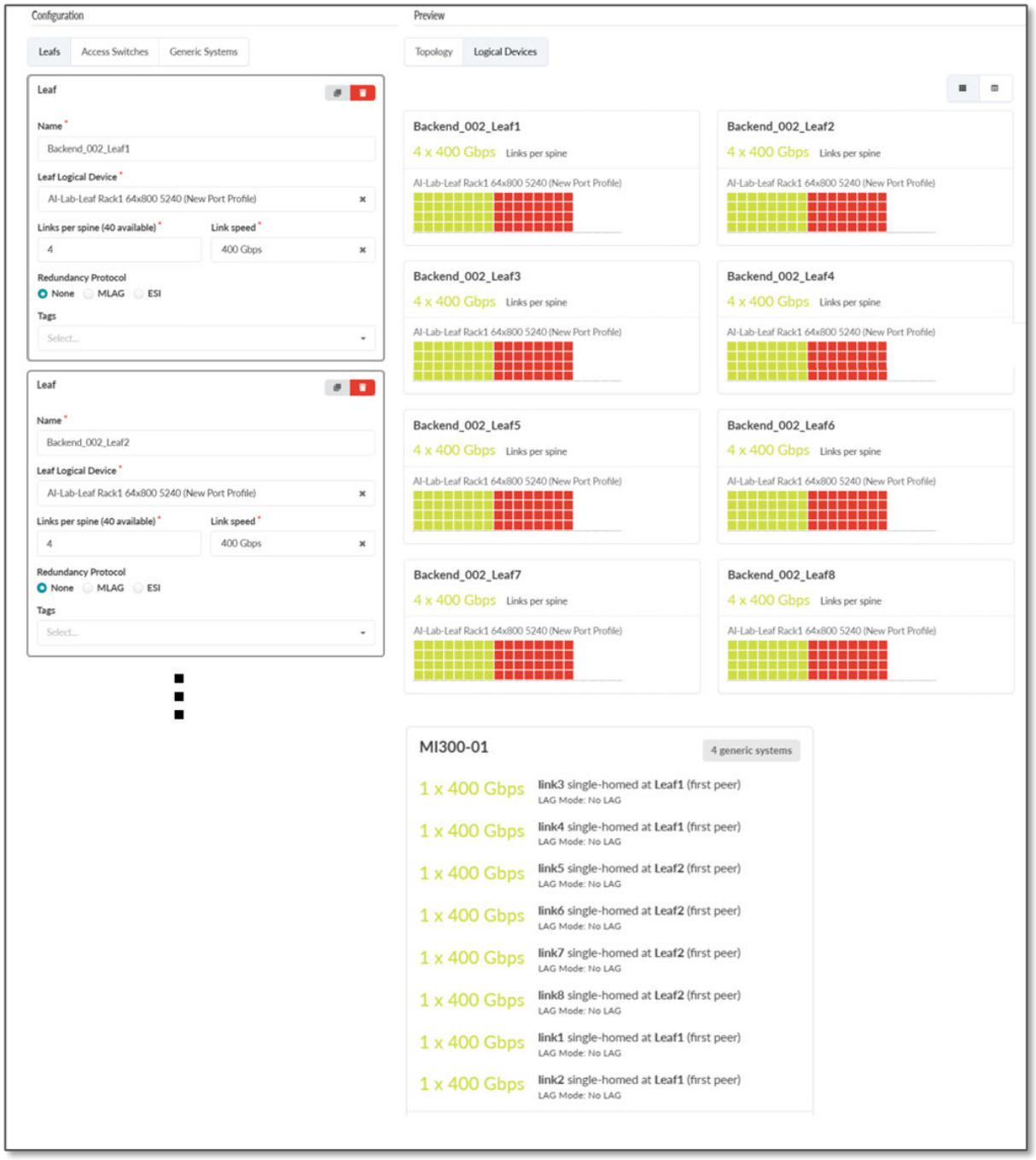
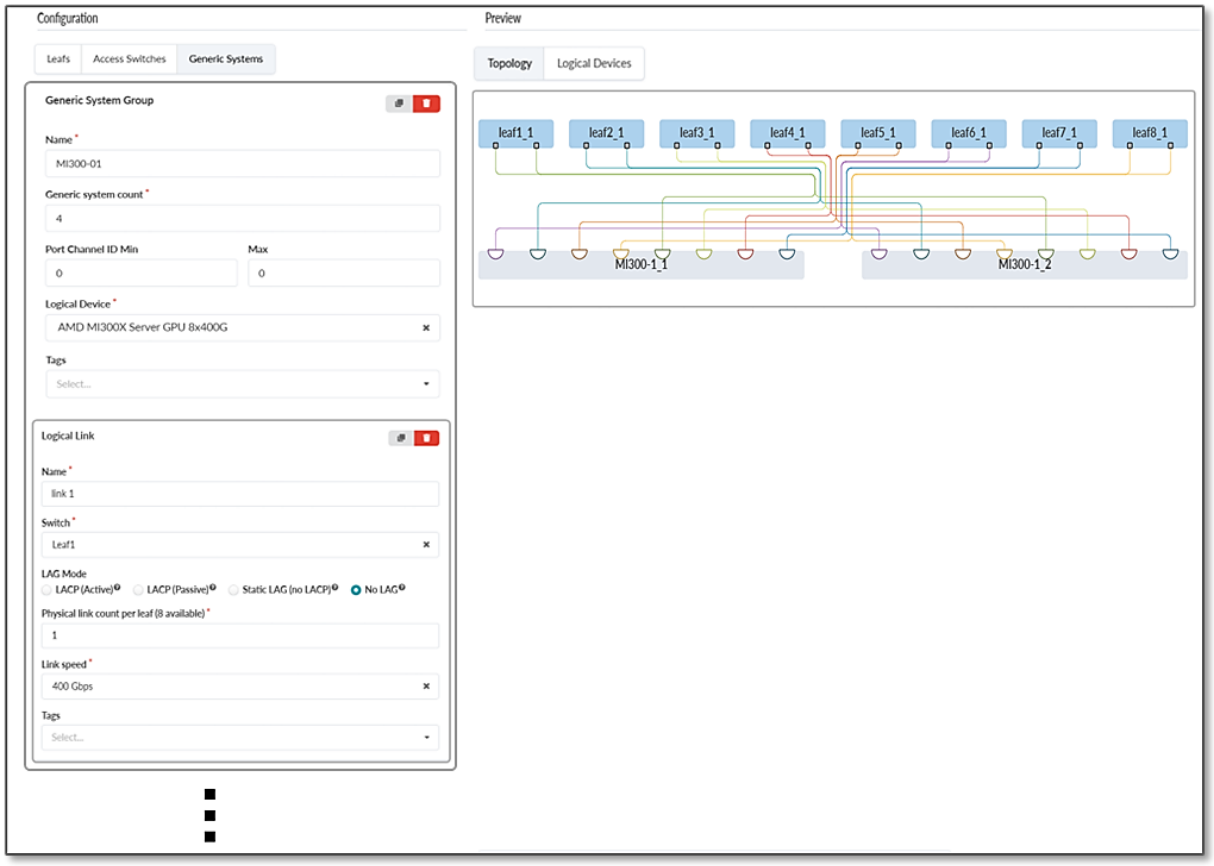


Figure 34: Creating a Rack in Apstra using the **Create In Builder** option - Leaf nodes details



Under the **Generic Systems** tab add the MI300X devices and reference the logical device created in the previous step, as shown in Figure 35.

Figure 35: Creating a Rack in Apstra using the **Create In Builder** option – Generic Systems (GPU servers) details



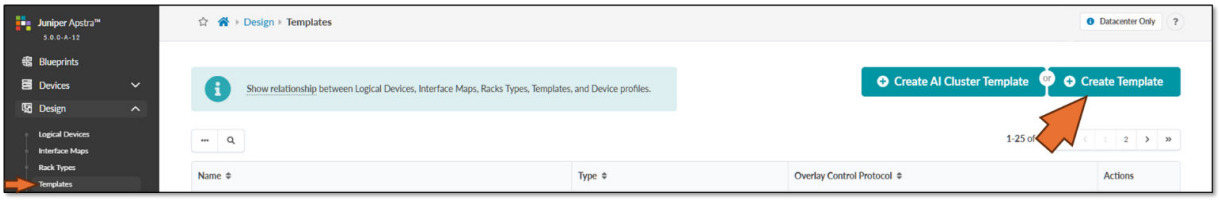
Notice that as you add the devices, the preview section on the right side will be updated. You can choose between viewing Logical Devices (Figures 34) or Topology (Figures 35).

Also, the topology created by Apstra follows the rail optimized architecture, which is a new feature introduced in Apstra.

Step 4: Create Template.

To create a template that references the QFX5240 rack type created in the previous step, navigate to **Design** -> **Templates** -> **Create Template** as shown in Figure 36.

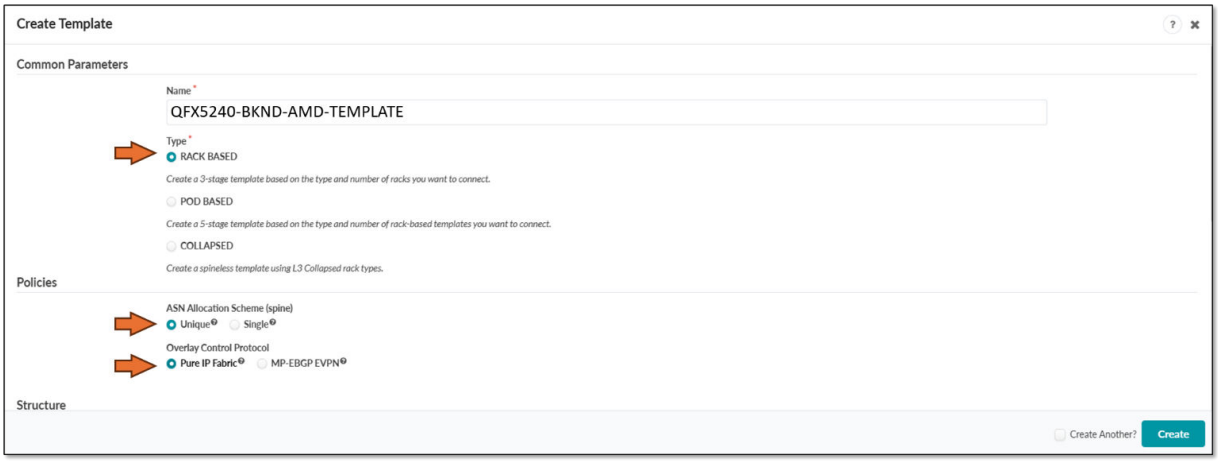
Figure 36: Create Apstra Template



NOTE: You can choose between **Create Template** or **Create AI Cluster Template** (Select from pre-existing designs). We demonstrate the **Create Template** option here.

Enter the name of the template, and select Type RACK BASED, policies ASN allocation Unique, and Overlay Pure IP Fabric.

Figure 37: Creating a Template in Apstra - Parameters



Scroll down and select the Rack type and Spine logical device created in previous steps, set the number of Racks (which is equivalent to saying two stripes) as 2, and the number of spines as 4. Click on **create** when ready, as shown in Figure 38.

Figure 38: Creating a Template in Apstra - Structure

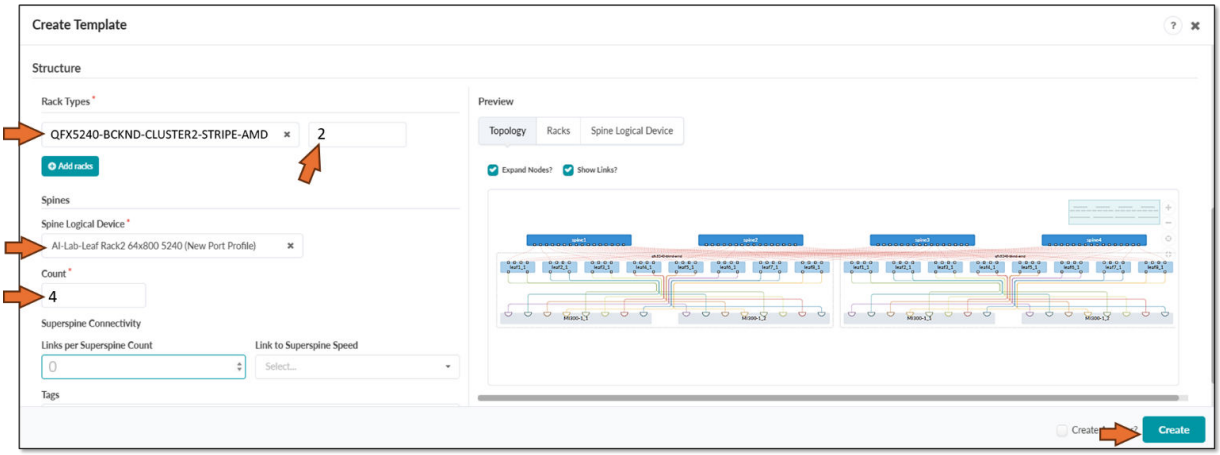
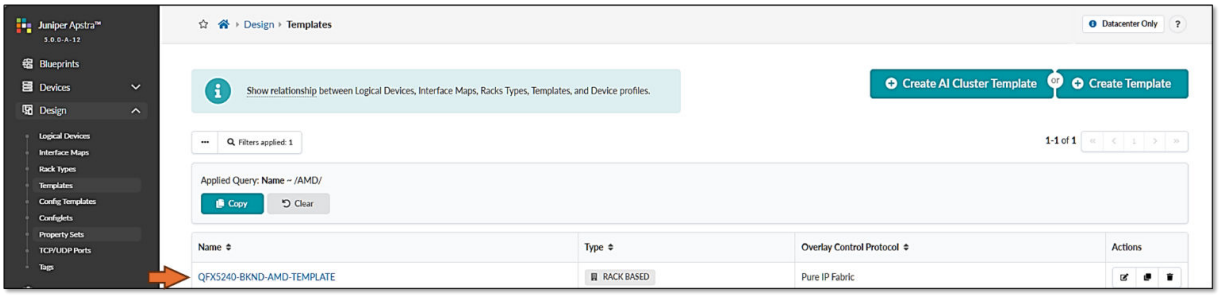


Figure 39: Verifying new template creation



Step 5: Create Configlets for DCQCN and DLB.

In the Apstra version used for this JVD, features such as ECN, PFC (DCQCN), and DLB are not natively available. Therefore, Apstra configlets should be used to add these features to the configurations before they are deployed to the fabric devices.

The configlet used for the DCQCN and DLB features on the QFX leaf nodes is as follows:

```
/* DLB configuration for Thor NIC2 Adapter */
hash-key {
    family inet {
        layer-3;
        layer-4;
    }
}
enhanced-hash-key {
    ecmp-dlb {
        flowlet {
            inactivity-interval 128;
            flowset-table-size 2048;
        }
        ether-type {
            ipv4;
            ipv6;
        }
        sampling-rate 1000000;
    }
}
protocols {
    bgp {
        global-load-balancing {
            load-balancer-only;
        }
    }
```

```
}
/* DCQCN configuration */
classifiers {
    dscp mydscp {
        forwarding-class CNP {
            loss-priority low code-points 110000;
        }
        forwarding-class NO-LOSS {
            loss-priority low code-points 011010;
        }
    }
}
drop-profiles {
    dp1 {
        interpolate {
            fill-level [ 55 90 ];
            drop-probability [ 0 100 ];
        }
    }
}
    shared-buffer {
        ingress {
       buffer-partition lossless {
          percent 66;
          dynamic-threshold 10;
       }
       buffer-partition lossless-headroom {
          percent 24;
       }
       buffer-partition lossy {
          percent 10;
       }
      }
      egress {
        buffer-partition lossless {
           percent 66;
        }
        buffer-partition lossy {
           percent 10;
        }
      }
    }
forwarding-classes {
```

```
    class CNP queue-num 3;
    class NO-LOSS queue-num 4 no-loss pfc-priority 3;
}
congestion-notification-profile {
    cnp {
        input {
            dscp {
                code-point 011010 {
                    pfc;
                }
           }
        }
        output {
            ieee-802.1 {
                code-point 011 {
                    flow-control-queue 4;
                }
            }
        }
    }
}
interfaces {
    et-* {
        congestion-notification-profile cnp;
        scheduler-map sm1;
        unit * {
            classifiers {
                dscp mydscp;
            }
        }
    }
}
scheduler-maps {
    sm1 {
        forwarding-class CNP scheduler s2-cnp;
        forwarding-class NO-LOSS scheduler s1;
    }
}
schedulers {
    s1 {
        drop-profile-map loss-priority any protocol any drop-profile dp1;
        explicit-congestion-notification;
    }
```

```
    s2-cnp {
        transmit-rate percent 5;
        priority strict-high;
    }
}
```

The configlet used for the DCQCN and DLB features on the QFX spine nodes is as follows:

```
/* DLB configuration */
hash-key {
    family inet
        layer-3;
        layer-4;
    }
}
enhanced-hash-key {
    ecmp-dlb {
        flowlet {

            inactivity-interval 128;

            flowset-table-size 2048;

        }

        ether-type {

            ipv4;

            ipv6;

        }

        sampling-rate 1000000;

    }

}

protocols {

    bgp {
```

```
        global-load-balancing {
            helper-only;
        }
    }
}
/* DCQCN configuration */
class-of-service {
    classifiers {
        dscp mydscp {
            forwarding-class CNP {
                loss-priority low code-points 110000;
            }
            forwarding-class NO-LOSS {
                loss-priority low code-points 011010;
            }
        }
    }
    drop-profiles {
        dp1 {
            interpolate {
                fill-level [ 55 90 ];
```

```
                drop-probability [ 0 100 ];
            }
        }
    }
    shared-buffer {
        ingress {
       buffer-partition lossless {
          percent 66;
          dynamic-threshold 10;
       }
       buffer-partition lossless-headroom {
          percent 24;
       }
       buffer-partition lossy {
          percent 10;
       }
      }
      egress {
        buffer-partition lossless {
            percent 66;
        }
        buffer-partition lossy {
```

```
        percent 10;
      }
    }
  }
  forwarding-classes {
      class CNP queue-num 3;
      class NO-LOSS queue-num 4 no-loss pfc-priority 3;
  }
  congestion-notification-profile {
      cnp {
          input {
              dscp {
                  code-point 011010 {
                      pfc;
                  }
              }
          }
          output {
              ieee-802.1 {
                  code-point 011 {
                      flow-control-queue 4;
```

```
                }
            }
        }
    }
}
interfaces {
    et-* {
        congestion-notification-profile cnp;
        scheduler-map sm1;
        unit * {
            classifiers {
                dscp mydscp;
            }
        }
    }
}
scheduler-maps {
    sm1 {
        forwarding-class CNP scheduler s2-cnp;
        forwarding-class NO-LOSS scheduler s1;
    }
}
```

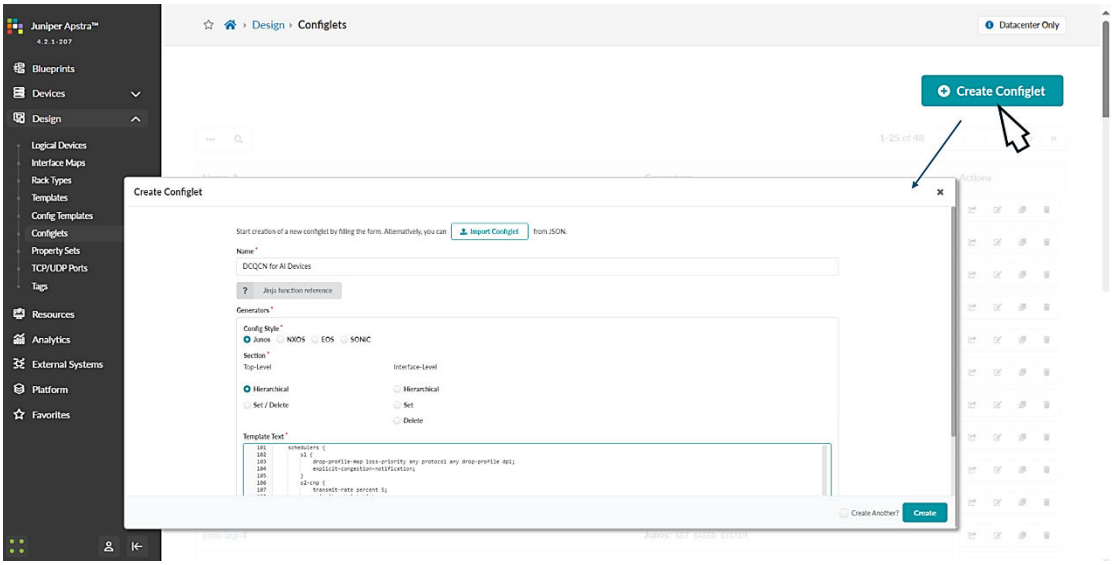
```
    schedulers {

        s1 {

            drop-profile-map loss-priority any protocol any drop-profile dp1;

            explicit-congestion-notification;
        }
        s2-cnp {
            transmit-rate percent 5;
           priority strict-high;
        }
    }
}
```

To create these configlets navigate to:

Design -> **Configlets** -> **Create Configlet** and click on **Create configlet**.

Provide a name for the configlet, select the operating system, vendor and configuration mode and paste the above configuration snippet on the template text box as shown below:

Figure 40: DCQCN Configlet Creation in Apstra



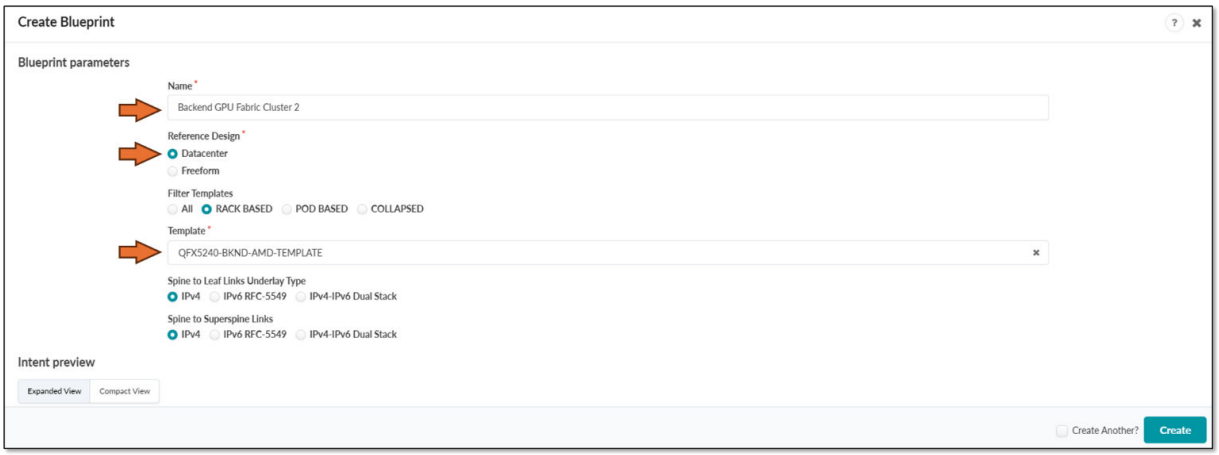
Step 6: Create the GPU Backend Fabric Blueprint.

Navigate to the **Blueprints** section and click on **Create Blueprint,** as shown in Figure 41.

Figure 41: Creating a Blueprint in Apstra

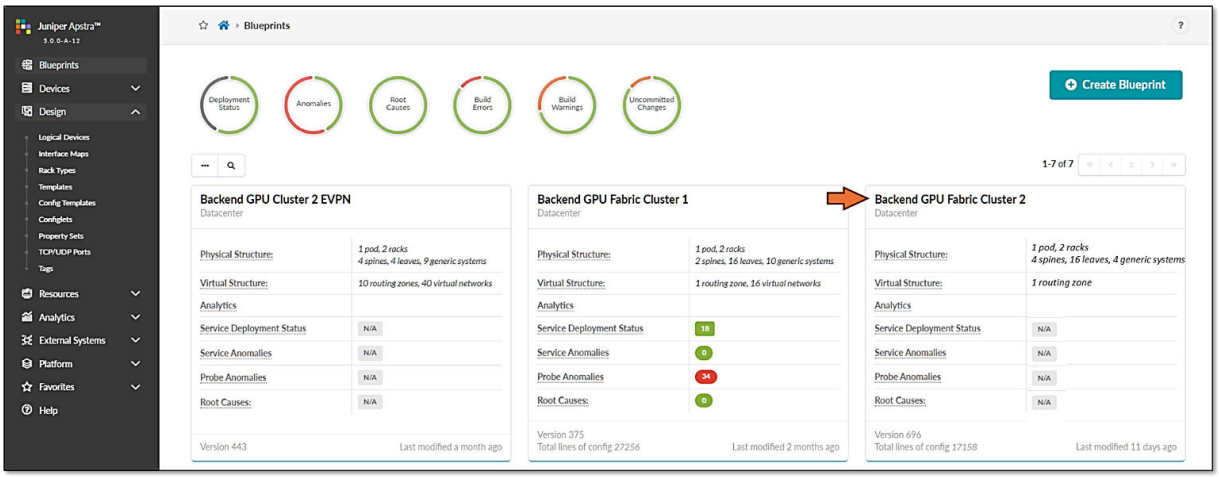
Provide a name for the new blueprint, select data center as the reference design, and select Rack-based. Then select the template that was created in the previous step. You can review the Intent preview before clicking Create.

Figure 42: New Blueprint Attributes in Apstra



Once the blueprint is successfully initiated by Apstra, it will be included in the Blueprint dashboard as shown below.

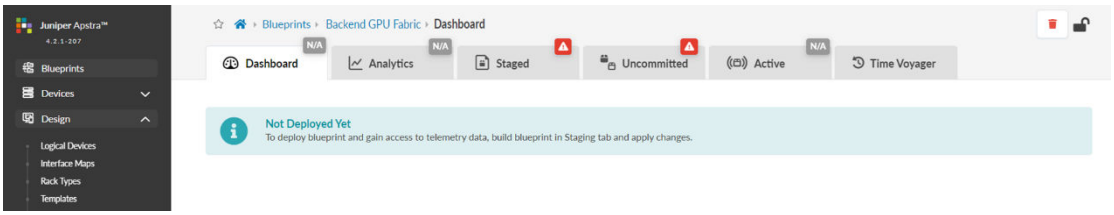
Figure 43: New Blueprint Added to Blueprint Dashboard



Notice that the Deployment Status, Service Anomalies, Probe Anomalies and Root Causes are all shown as N/A. This is because you will need to complete additional steps that include mapping the different roles in the blueprint to the physical devices, defining which interfaces will be used, etc.

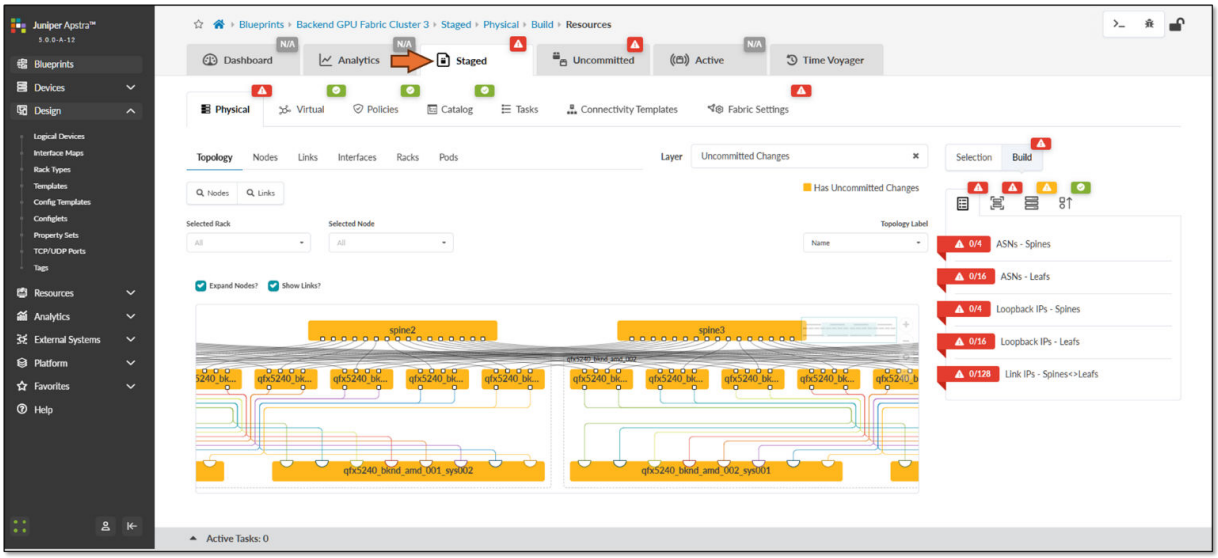
When you click on the blueprint name and enter the blueprint dashboard it will indicate that the blueprint has not been deployed yet.

Figure 44: New Blueprint's dashboard



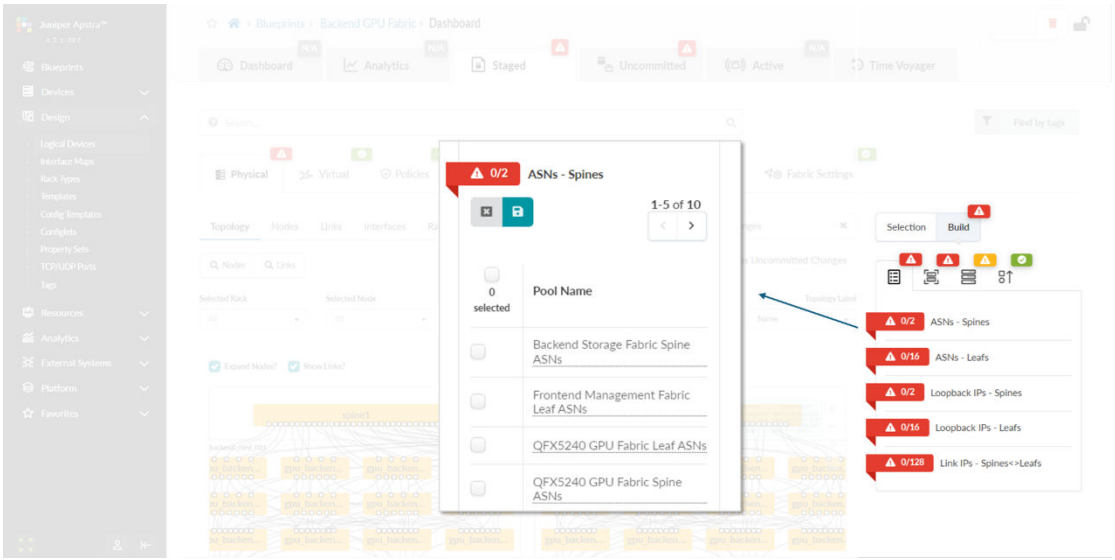
The Staged view, as depicted in Figure 40, shows that the topology is correct, but attributes such as mandatory ASNs and loopback addresses for the spines and the leaf nodes, and the spine to leaf links addressing must still be provided by the user.

Figure 45: Undeployed Blueprint Dashboard



You will need to edit each one of these attributes and select from predefined pools of addresses and ASNs, as shown in the example on Figure 46, to fix this issue.

Figure 46: Selecting ASN Pool for Spine Nodes



You will also need to select Interface Maps for each device's role and along with assignment of system IDs as shown in Figures 47-48.

Figure 47: Mapping Interface Maps to Spine Nodes

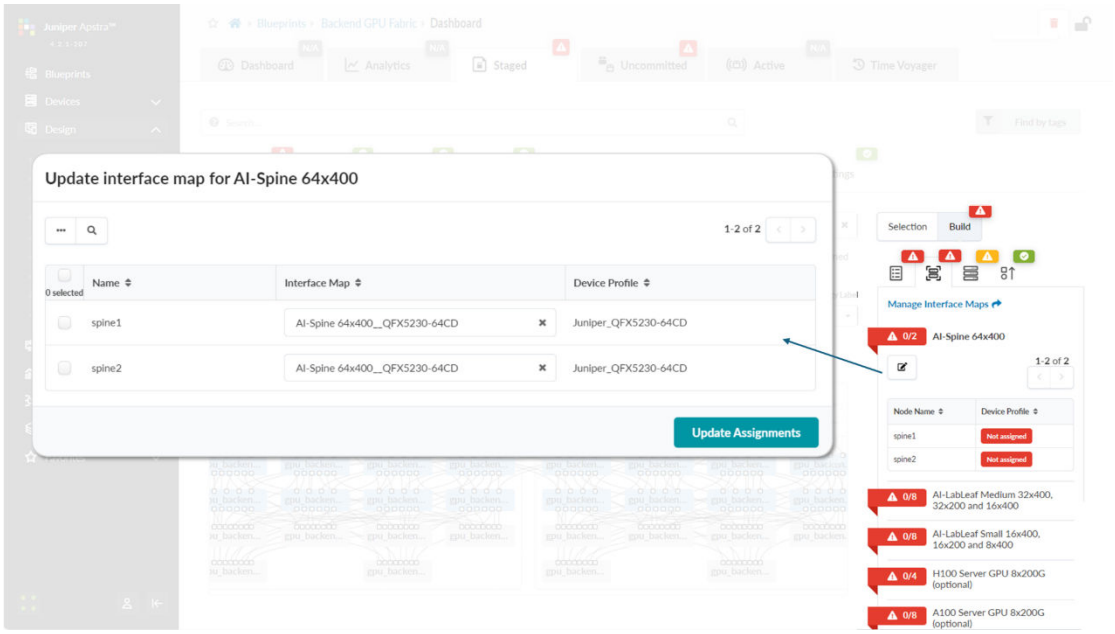
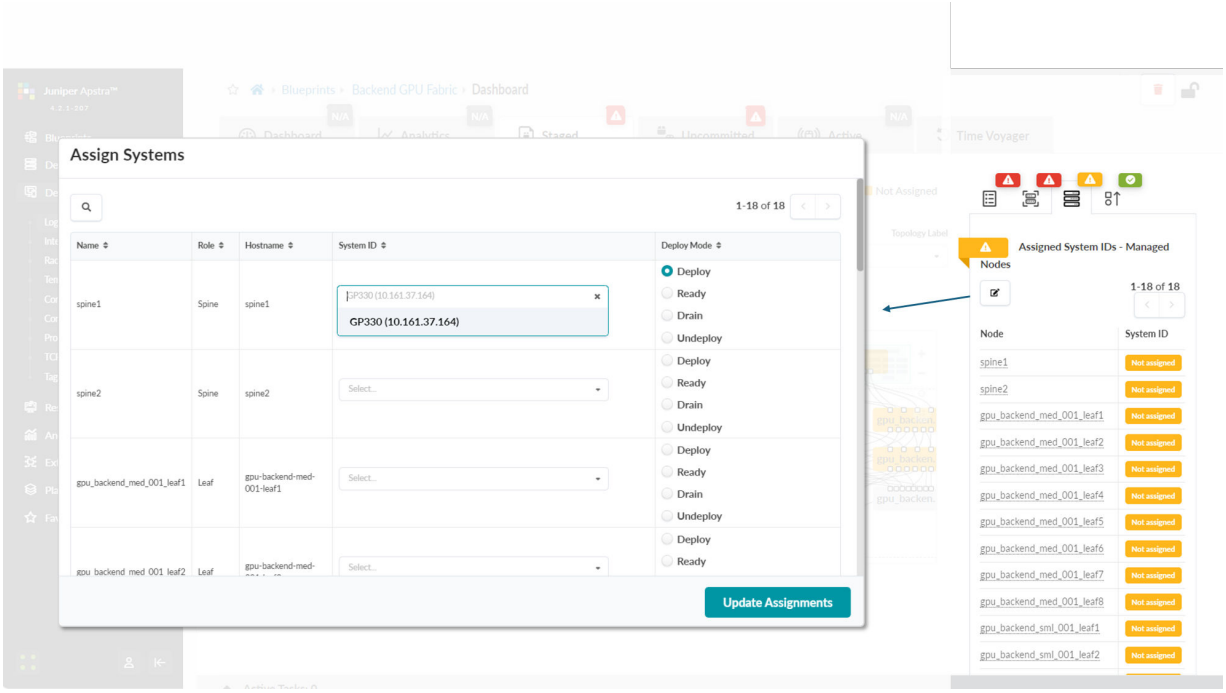
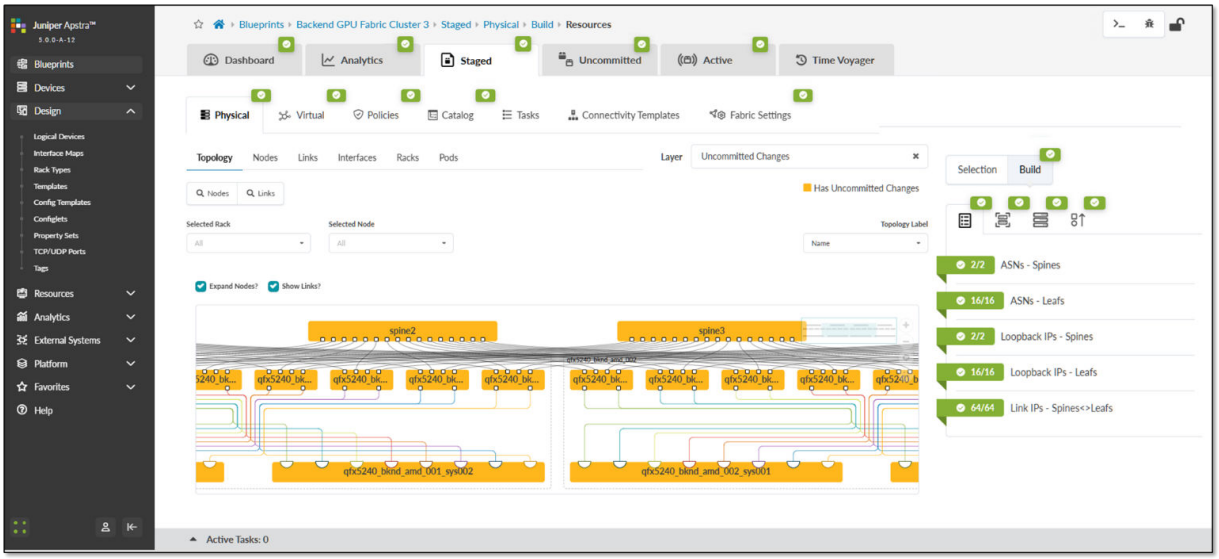


Figure 48: Mapping Spine Nodes to Physical Devices (System IDs)



Once all these steps are completed, you can commit the changes, and Apstra will generate and push the vendor and device-type specific configurations to all devices in the blueprint. After this process is complete, the fabric should be successfully deployed, as indicated by the green checkmarks shown in Figure 49.

Figure 49: Active Blueprint.



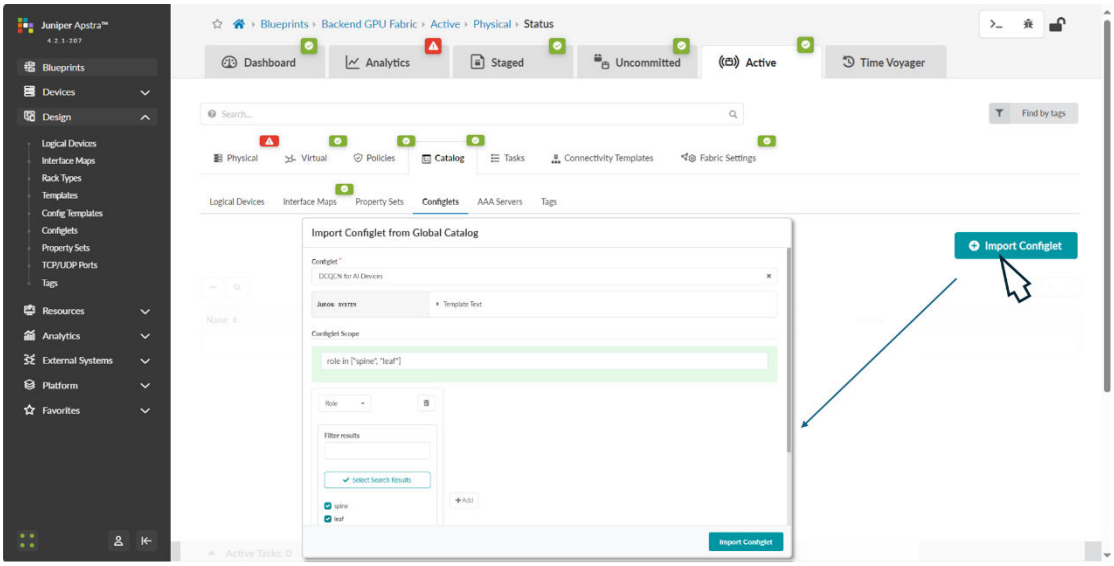
Step 7: Apply the configlets previously created to the Blueprint.

The configlet should be applied to the devices, both leaf and spine roles within the blueprint.

Navigate back to the blueprint dashboard and the move to **Staged** -> **Catalog** -> **Import.**

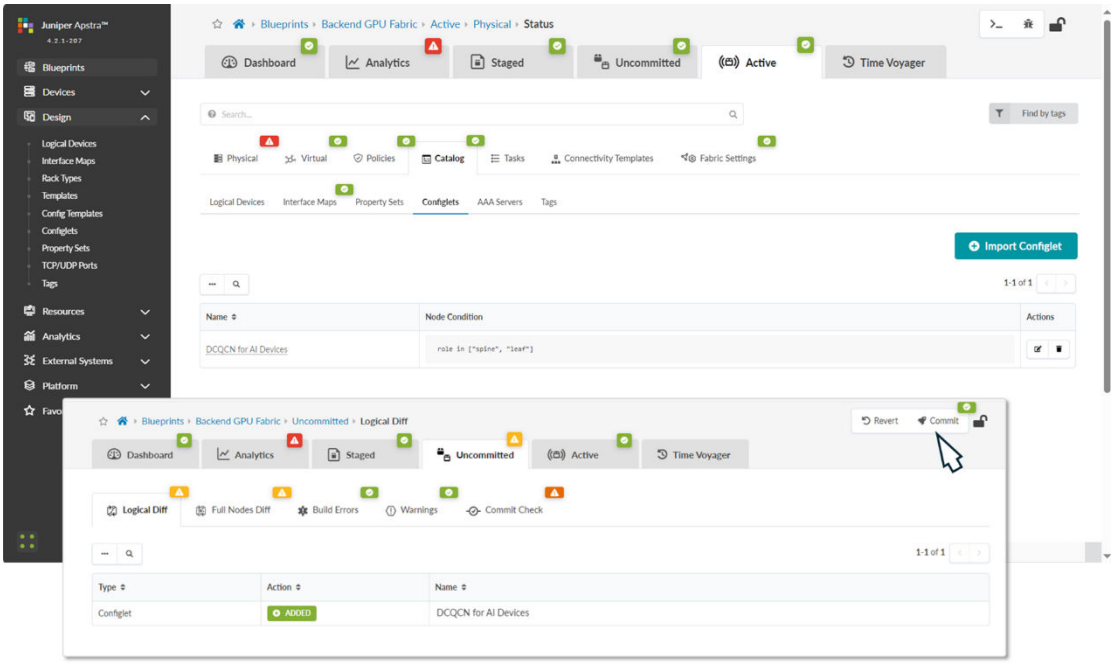
Select the configlet you want to apply, and the device role where you want to apply it.

Figure 50: Applying DCQCN Configlets to Devices in Apstra



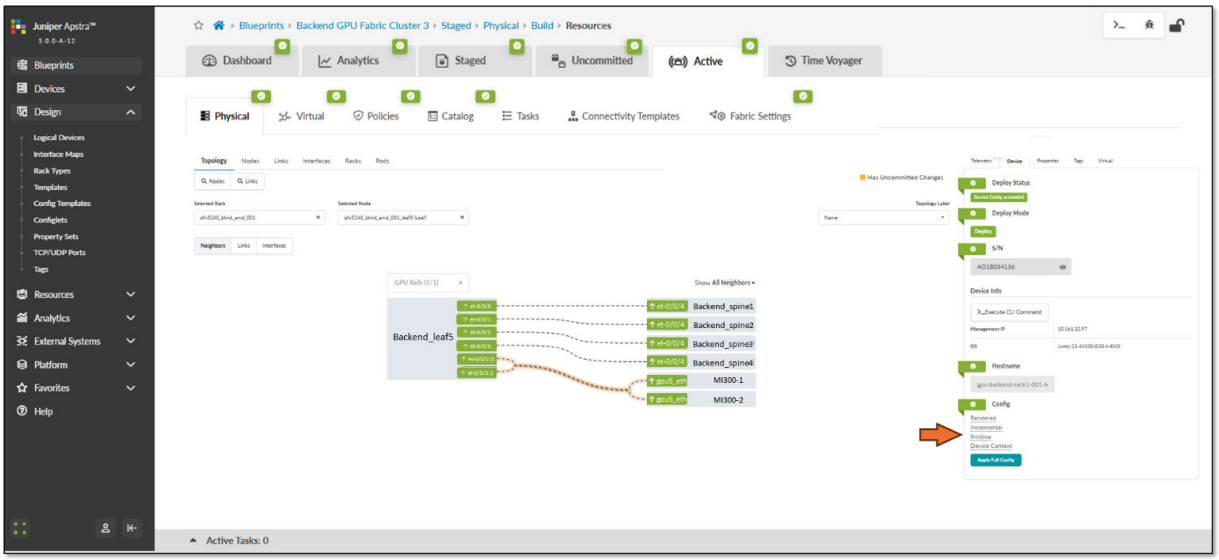
After successfully importing the configlet into the blueprint it should be listed in the catalog. You need to commit the changes for the configuration to be deployed to the devices.

Figure 51: Applying DCQCN Configlets to Devices in Apstra



You can quickly check the status and the deployed configuration by clicking on each device in the Active tab and selecting the rendered config under the Device tab on the right side.

Figure 52: Device configuration verification in Apstra



Terraform Automation of Apstra for the AI Fabric

IN THIS SECTION

AI Terraform Configs

Juniper has compiled a set of Terraform configs to help set up data center fabrics for an AI cluster.

The github repository for AI designs using Apstra can be found:

https://github.com/Juniper/terraform-apstra-examples/tree/master/ai-cluster-designs/

AI JVD Specific Terraform Configs

The AI JVD Specific Terraform Configs for Apstra will build blueprints for the reference AI cluster's including the rail-optimized GPU Backend fabric, the Storage Backend fabric, and the Frontend fabric.

The github repository for this specific AI JVD can be found:

https://github.com/Juniper/terraform-apstra-examples/tree/master/ai-cluster-jvd/

AMD Configuration

IN THIS SECTION

The AI servers covered as part of the JVD include 2 Supermicro AS-8125GS-TNMR2 Dual AMD EPYC 8U GPU and 2 Dell PowerEdge XE9680.

This section provides some guidelines to install and configure the interfaces and other relevant parameters based on the AI JVD lab testing. Always refer to the official manufacturer documentation when making changes and for more details.

AMD MI300Xx Setting BIOS Parameters

Each vendor has different BIOS settings based on differences in its UI and GPU mappings and the servers' internal architectures.

SuperMicro AS-8125GS-TNMR2

Boot the server into Setup mode (the boot to supermicro splash will take several minutes to appear):

UEFI/BIOS Area	Value
Advanced -> NB Configuration	ACS Enable = Disable
Advanced -> NB Configuration -> xGMI	xGMI Link Width Control = Manual
	xGMI Force Link Width Control = Force
	xGMI Force Link Width = 2
	xGMI Max Link Width control = Manual
	xGMI Link Max Speed = Auto
Advanced -> PCIe/PCI/PnP Configuration	Above 4G Encoding: Enabled
	Re-Size BAR Support: Enabled
	SR-IOV Support: Enabled
	Workload = Not Configured

DELL XE9680

The following BIOS settings are recommended by Dell for their XE9680 AI/ML server. The BIOS settings disable IOMMU and ACS on the host as well.

UEFI/BIOS Area	Value
BIOS -> Processor Settings	Logical Processor = Disable
	Virtualization Technology = Disable
	SubNumaCluster = Disable
	MADt Core cluster = Linear
[1] BIOS -> Integrated Devices	Global SRIOV = Disable [1]
BIOS -> System Profile Setting	Server System Profile = Performance
	Workload = Not Configured
BIOS -> System Security	AC Recovery Delay = Random (highly recommended)

[1] *Dell recommends "enabling" Global SR-IOV, but on the Dell DUTs in this lab setup, this setting was incompatible with the THOR2 NIC port mode 0 for the storage and frontend fabrics (2x200Gb vs. 1x400Gb), causing the DUT to fault on boot. Consult with your Dell account team for recommendations about this setting in your setup."*

Follow the configuration steps described in the Single-node network configuration for AMD Instinct accelerators — GPU cluster networking documentation. Notice that the disable ACS script used in step 6, must also be run before any workloads, after a server has been rebooted.

Identifying NICs and GPUs mappings

Along with the fabric and GPU server setup, this JVD which covers the configuration and setup of ethernet Network Adapters (or NIC) as below. The Broadcom BCM57608 (Thor2) ethernet network adapter was validated in Phase 1. And in Phase2, the AMD Pollara 400 NIC cards are validated.

All 4 servers are equipped with:

- 8 x AMD Instinct MI300XX OAM GPUs

and either of the below NICs

- 8 x Single port 400/200/100/50/25/10GbE Broadcom BCM57608 (Thor2) adapter with 400Gbps QDD-400G-DR4 transceivers used to connect to the GPU backend Fabric.

Or

- 8 x Single port 400G and 2 ports 200G and 4 ports 100/50/25G AMD Pensando Pollara 400 ethernet network adapter with Q112-400G-DR4 transceivers used to connect to the GPU backend Fabric.

Note: Most of the setup commands will apply to both Thor2 and AMD Pollara 400 NIC. However in case of any differences in steps same will be called out at appropriate places within the document.

Dell devices:

- 1 x Mellanox MT2910 Family NVIDIA® ConnectX®-7 SmartNIC with 100Gbps QSFP28 transceivers to connect to the Frontend Fabric
- 2 x Mellanox MT2910 Family NVIDIA® ConnectX®-7 SmartNIC with 200Gbps QDD-2X200G-AOC-5M transceivers to connect to the Frontend Fabric

AMD MI300x GPU Server and NIC Firmware and RCCL libraries

For the purposes of Broadcom Thor 2 NIC validation following are the main OS and firmware versions configured on the MI300x GPU servers:

Broadcom Thor 2 Ethernet Adapter

Below are the details of the Operating System (OS), firmware and AMD libraries installed:

OS/Firmware	Version
Ubuntu	Ubuntu 24.04.2 LTS
Broadcom Thor2 NIC Firmware version	231.2.63.0

Following are the libraries installed for RCCL test for Thor2 Network adapter:

RCCL Test libraries	Version	command
rocm/noble	6.4.0.60400-47~24.04 amd64	apt list rocm

(Continued)

RCCL Test libraries	Version	command
Rccl [1]	2.22.3.60400-47~24.04 amd64	apt list rccl
mpi (Open MPI)	5.0.8a1	mpirun –version
UCX https://github.com/openucx/ucx.git	1.15.0	/opt/ucx/bin/ucx_info -v

Note: For AMD drivers and host utilities, please reach out to your regional AMD representative.

AMD Pensando Pollara 400 Ethernet Adapter

For the purposes of the AMD Pollara 400 NIC validation, the following are the main OS and firmware versions configured on the MI300x GPU servers:

OS/Firmware	Version
Ubuntu	Ubuntu 22.04.5 LTS
AMD Pollara NIC Firmware version	1.110.0-a-79

Output of the ubuntu version 22.04 installed on the MI300 servers.

```
jnpr@mi300-01:~$ cat /etc/os-release

PRETTY_NAME="Ubuntu 22.04.5 LTS"

NAME="Ubuntu"

VERSION_ID="22.04"

VERSION="22.04.5 LTS (Jammy Jellyfish)"

VERSION_CODENAME=jammy
```

```
ID=ubuntu

ID_LIKE=debian

HOME_URL="https://www.ubuntu.com/"

SUPPORT_URL="https://help.ubuntu.com/"

BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"

PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
```

Output of the AMD Pollara 400 NIC card Firmware version

```
jnpr@mi300-01:~$ sudo nicctl show card --detail | grep Firmware

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79
```

Following are the libraries installed for RCCL test for AMD Pollara 400 NIC adaper:

RCCL Test libraries	**Version**	**command**
rocm/jammy	6.3.3.60303-74~22.04 amd64	apt list rocm
Rccl [1]	7961624	

(Continued)

RCCL Test libraries	Version	command
mpi (Open MPI)	5.1.0a1	/opt/ompi/bin/mpirun –version
UCX https://github.com/openucx/ucx.git	1.20.0	/opt/ucx/bin/ucx_info -v
rccl-tests	revision 6704fc6	Git branch https://github.com/ROCm/rccl-tests.git
ANP Plugin[2]		

Note: For AMD drivers and host utilities, please reach out to your regional AMD representative.

1. The RCCL library is private build version that AMD provided.

2. ANP Plugin version is private build provided by AMD.

For more information on installing these software and dependent libraries, high level steps are provided later in section "AMD Pollara firmware and dependent libraries" on page 116, as these steps can only be performed once the NICs and GPUs are mapped as described in sections below.

In this section we will explore some of the options to find information about and configure the NICs and GPUs.

ROCm Communication Collectives Library (RCCL)

In AMD servers, the ROCm provides multi-GPU and multi-node collective communication primitives optimized for AMD GPUs. These collectives implement send and receive operations such as all-reduce, all-gather, reduce, broadcast, all-to-all, and so on across multiple GPUs in one or more GPU servers.

Communication between GPUs in a single server is implemented using xGMI (inter-chip global memory interconnect), part of AMD's Infinity Fabric technology. The Infinity Fabric is a high-bandwidth, low-latency interconnect for the various components within a system including CPUs, GPUs, memory, NICs and other devices. xGMI provides socket-to-socket communication, allowing direct CPU-to-CPU or GPU-to-GPU communication.

Communication between different servers is processed by RDMA-capable NICs (e.g., RoCEv2 over Ethernet) and routed across the GPU backend fabric. These NICs can be used by any GPU at any time as

there is no hard coded 1-to-1 GPU to NIC mapping. However, the use of preferred communication paths between GPUs and NICs creates the appearance of a 1:1 correspondence.

RCCL will always choose the path that has the best connection between GPUs and between GPUs and NICs, aiming to optimize bandwidth, and latency. Optimized intra-node path will be taken before forwarding inter-node.

The `rocm-smi` (Radeon Open Compute Platform System Management Interface) cli provides tools for configuring and monitoring AMD GPUs. It can be used to identify GPUs hardware details as well as topology information using the options such as:

`--showproductname`: show product details

`--showtopo` : show hardware topology information

`--showtopoaccess` : shows the link accessibility between GPUs

`--showtopohops`: shows the number of hops between GPUs

`--showtopotype` : shows the link type between GPUs

`--showtoponuma` : shows the numa nodes

`--shownodesbw`: shows the numa nodes bandwidth

`--showhw`: shows the hardware details

Examples from AMD Instinct MI300XX OAM:

The `--showproductname` shows the GPU series, model, and vendor along with additional details. The example output shows AMD Instinct™ MI300XX Platform GPUs are installed in the server.

```
jnpr@MI300X-01:/proc$ rocm-smi --showproductname

============================ ROCm System Management Interface ============================

==================================== Product Info ====================================

GPU[0]          : Card Series:          AMD Instinct MI300XX OAM

GPU[0]          : Card Model:           0x74a1

GPU[0]          : Card Vendor:          Advanced Micro Devices, Inc. [AMD/ATI]

GPU[0]          : Card SKU:             M3000100
```

```
GPU[0]          : Subsystem ID:         0x74a1

GPU[0]          : Device Rev:           0x00

GPU[0]          : Node ID:              2

GPU[0]          : GUID:                 28851

GPU[0]          : GFX Version:          gfx942

GPU[1]          : Card Series:          AMD Instinct MI300XX OAM

GPU[1]          : Card Model:           0x74a1

GPU[1]          : Card Vendor:          Advanced Micro Devices, Inc. [AMD/ATI]

GPU[1]          : Card SKU:             M3000100

GPU[1]          : Subsystem ID:         0x74a1

GPU[1]          : Device Rev:           0x00

GPU[1]          : Node ID:              3

GPU[1]          : GUID:                 51499

GPU[1]          : GFX Version:          gfx942

---more--
```

The `--showhw` options shows information about the GPUs in the system, including ID

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# rocm-smi --showhw -v

======================================= ROCm System Management Interface
================================

========================================= Concise Hardware Info
=======================================

GPU  NODE  DID     GUID   GFX VER  GFX RAS  SDMA RAS  UMC RAS  VBIOS           BUS
PARTITION ID
```

```
0    2     0x74a1  28851  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:05:00.0
0

1    3     0x74a1  51499  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:27:00.0
0

2    4     0x74a1  57603  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:47:00.0
0

3    5     0x74a1  22683  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:65:00.0
0

4    6     0x74a1  53458  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:85:00.0
0

5    7     0x74a1  26954  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:A7:00.0
0

6    8     0x74a1  16738  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:C7:00.0
0

7    9     0x74a1  63738  gfx942   ENABLED  ENABLED   ENABLED  113-M3000100-102 0000:E5:00.0
0

==========================================================================================
=========

========================================== End of ROCm SMI Log
========================================

======================================== VBIOS ========================================

GPU[0]          : VBIOS version: 113-M3000100-102

GPU[1]          : VBIOS version: 113-M3000100-102

GPU[2]          : VBIOS version: 113-M3000100-102

GPU[3]          : VBIOS version: 113-M3000100-102

GPU[4]          : VBIOS version: 113-M3000100-102
```

```
GPU[5]          : VBIOS version: 113-M3000100-102

GPU[6]          : VBIOS version: 113-M3000100-102

GPU[7]          : VBIOS version: 113-M3000100-102

==============================================================================================
```

The fields are defined as follows:

GPU	Index of the GPU on the system, starting from 0.
NODE	NUMA (Non-Uniform Memory Access) node ID associated with the GPU. Helps identify memory locality. Optimal GPU/NIC mapping often relies on NUMA proximity
DID	Device ID of the GPU. This is a unique identifier for the specific GPU model. Useful for verifying the exact GPU model. For example, 0x74a1 corresponds to an MI300X-series GPU.
GUID	GPU Unique Identifier. This value is specific to each GPU and may relate to its PCIe device. Useful for distinguishing GPUs in a multi-GPU environment.
GFX VER	The version of the GPU architecture (e.g., gfx942 is part of AMD's RDNA2 family). In AMD GPUs, the GFX prefix is part of AMD's internal naming convention for their GPU microarchitecture families. GPU architecture hardware specifications — ROCm Documentation
GFX RAS	Status of GPU RAS (Reliability, Availability, Serviceability) features. Indicates error handling.
SDMA RAS	Status of SDMA (System Direct Memory Access) RAS features.
UMC RAS	Status of Unified Memory Controller (UMC) RAS features.

VBIOS	VBIOS (Video BIOS) version. Indicates the firmware version running on the GPU. Identical firmware version (113-M3000100-102) for all GPUs indicates a uniform configuration.
BUS	PCIe bus address of the GPU. Helps map the GPU to its physical slot. For example, 0000:05:00.0 is the PCIe address. It allows you to correlate GPUs to physical slots or NUMA nodes.
PARTITION ID	GPU partition or instance ID. For multi-instance GPUs (e.g., MI300X), this would identify instances.All values are 0 indicate no multi-instance partitioning is enabled for these GPUs.

The `--showbus` options shows PCI bus related information, including correspondence between GPU IDs and PCI Bus IDs.

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# rocm-smi --showbus -i

============================ ROCm System Management Interface ============================

========================================= ID =========================================

GPU[0]          : Device Name:          AMD Instinct MI300XX OAM

GPU[0]          : Device ID:            0x74a1

GPU[0]          : Device Rev:           0x00

GPU[0]          : Subsystem ID:         0x74a1

GPU[0]          : GUID:                 28851

GPU[1]          : Device Name:          AMD Instinct MI300XX OAM

GPU[1]          : Device ID:            0x74a1

GPU[1]          : Device Rev:           0x00

GPU[1]          : Subsystem ID:         0x74a1
```

```
GPU[1]          : GUID:                 51499

GPU[2]          : Device Name:          AMD Instinct MI300XX OAM

GPU[2]          : Device ID:            0x74a1

GPU[2]          : Device Rev:           0x00

GPU[2]          : Subsystem ID:         0x74a1

GPU[2]          : GUID:                 57603

---more---

==========================================================================================

======================================= PCI Bus ID =======================================

GPU[0]          : PCI Bus: 0000:05:00.0

GPU[1]          : PCI Bus: 0000:27:00.0

GPU[2]          : PCI Bus: 0000:47:00.0

GPU[3]          : PCI Bus: 0000:65:00.0

GPU[4]          : PCI Bus: 0000:85:00.0

GPU[5]          : PCI Bus: 0000:A7:00.0

GPU[6]          : PCI Bus: 0000:C7:00.0

GPU[7]          : PCI Bus: 0000:E5:00.0

==========================================================================================

================================== End of ROCm SMI Log ===================================
```

The `--showmetrics` option provides comprehensive information about the GPU status and performance, including metrics such as temperature, clock frequency, power, and pcie bandwidth.

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# rocm-smi --showmetrics | grep
GPU.0

GPU[0]          : Metric Version and Size (Bytes): 1.6 1664

GPU[0]          : temperature_edge (C): N/A

GPU[0]          : temperature_hotspot (C): 42

GPU[0]          : temperature_mem (C): 35

GPU[0]          : temperature_vrgfx (C): N/A

GPU[0]          : temperature_vrsoc (C): 41

GPU[0]          : temperature_vrmem (C): N/A

GPU[0]          : average_gfx_activity (%): 0

GPU[0]          : average_umc_activity (%): 0

GPU[0]          : average_mm_activity (%): N/A

GPU[0]          : average_socket_power (W): N/A

GPU[0]          : energy_accumulator (15.259uJ (2^-16)): 4291409153508

GPU[0]          : system_clock_counter (ns): 508330314785091

GPU[0]          : average_gfxclk_frequency (MHz): N/A

GPU[0]          : average_socclk_frequency (MHz): N/A

GPU[0]          : average_uclk_frequency (MHz): N/A

GPU[0]          : average_vclk0_frequency (MHz): N/A

GPU[0]          : average_dclk0_frequency (MHz): N/A
```

```
GPU[0]          : average_vclk1_frequency (MHz): N/A
GPU[0]          : average_dclk1_frequency (MHz): N/A
GPU[0]          : current_gfxclk (MHz): 134
GPU[0]          : current_socclk (MHz): 28
GPU[0]          : current_uclk (MHz): 900
GPU[0]          : current_vclk0 (MHz): 29
GPU[0]          : current_dclk0 (MHz): 22
GPU[0]          : current_vclk1 (MHz): 29
GPU[0]          : current_dclk1 (MHz): 22
GPU[0]          : throttle_status: N/A
GPU[0]          : current_fan_speed (rpm): N/A
GPU[0]          : pcie_link_width (Lanes): 16
GPU[0]          : pcie_link_speed (0.1 GT/s): 320
GPU[0]          : gfx_activity_acc (%): 682809151
GPU[0]          : mem_activity_acc (%): 60727622
GPU[0]          : temperature_hbm (C): ['N/A', 'N/A', 'N/A', 'N/A']
GPU[0]          : firmware_timestamp (10ns resolution): 507863813273800
GPU[0]          : voltage_soc (mV): N/A
GPU[0]          : voltage_gfx (mV): N/A
GPU[0]          : voltage_mem (mV): N/A
GPU[0]          : indep_throttle_status: N/A
GPU[0]          : current_socket_power (W): 123
```

```
GPU[0]          : vcn_activity (%): [0, 0, 0, 0]

GPU[0]          : gfxclk_lock_status: 0

GPU[0]          : xgmi_link_width: 0

GPU[0]          : xgmi_link_speed (Gbps): 0

GPU[0]          : pcie_bandwidth_acc (GB/s): 626812796806

GPU[0]          : pcie_bandwidth_inst (GB/s): 18

---more---
```

The `--showtopo` options show how the GPUs in the systems can communicate with each other via XGMI (Link Type) representing one hop between any two GPUs. The weight of 15 indicates this direct communication is the preferred path.

```
jnpr@MI300X-01:~$ rocm-smi --showtopo

============================ ROCm System Management Interface ============================

================================ Weight between two GPUs =================================

        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7

GPU0    0       15      15      15      15      15      15      15

GPU1    15      0       15      15      15      15      15      15

GPU2    15      15      0       15      15      15      15      15

GPU3    15      15      15      0       15      15      15      15

GPU4    15      15      15      15      0       15      15      15

GPU5    15      15      15      15      15      0       15      15

GPU6    15      15      15      15      15      15      0       15

GPU7    15      15      15      15      15      15      15      0
```

```
================================= Hops between two GPUs =================================

        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7

GPU0    0       1       1       1       1       1       1       1

GPU1    1       0       1       1       1       1       1       1

GPU2    1       1       0       1       1       1       1       1

GPU3    1       1       1       0       1       1       1       1

GPU4    1       1       1       1       0       1       1       1

GPU5    1       1       1       1       1       0       1       1

GPU6    1       1       1       1       1       1       0       1

GPU7    1       1       1       1       1       1       1       0

=============================== Link Type between two GPUs ===============================

        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7

GPU0    0       XGMI    XGMI    XGMI    XGMI    XGMI    XGMI    XGMI

GPU1    XGMI    0       XGMI    XGMI    XGMI    XGMI    XGMI    XGMI

GPU2    XGMI    XGMI    0       XGMI    XGMI    XGMI    XGMI    XGMI

GPU3    XGMI    XGMI    XGMI    0       XGMI    XGMI    XGMI    XGMI

GPU4    XGMI    XGMI    XGMI    XGMI    0       XGMI    XGMI    XGMI

GPU5    XGMI    XGMI    XGMI    XGMI    XGMI    0       XGMI    XGMI

GPU6    XGMI    XGMI    XGMI    XGMI    XGMI    XGMI    0       XGMI

GPU7    XGMI    XGMI    XGMI    XGMI    XGMI    XGMI    XGMI    0

================================== Numa Nodes ==================================
```

```
GPU[0]          : (Topology) Numa Node: 0

GPU[0]          : (Topology) Numa Affinity: 0

GPU[1]          : (Topology) Numa Node: 0

GPU[1]          : (Topology) Numa Affinity: 0

GPU[2]          : (Topology) Numa Node: 0

GPU[2]          : (Topology) Numa Affinity: 0

GPU[3]          : (Topology) Numa Node: 0

GPU[3]          : (Topology) Numa Affinity: 0

GPU[4]          : (Topology) Numa Node: 1

GPU[4]          : (Topology) Numa Affinity: 1

GPU[5]          : (Topology) Numa Node: 1

GPU[5]          : (Topology) Numa Affinity: 1

GPU[6]          : (Topology) Numa Node: 1

GPU[6]          : (Topology) Numa Affinity: 1

GPU[7]          : (Topology) Numa Node: 1

GPU[7]          : (Topology) Numa Affinity: 1

================================== End of ROCm SMI Log ==================================

Usage:

        cma_roce_tos OPTIONS

Options:

        -h              show this help

        -d <dev>        use IB device <dev> (default mlx5_0)
```

```
        -p <port>       use port <port> of IB device (default 1)

        -t <TOS>        set TOS of RoCE RDMA_CM applications (0)
```

The link type, number of hops, and weight can be also obtained using the specific options `--showtopoweight` , `--showtopotype`, and `-showtopoweight`:

```
jnpr@MI300X-01:~/SCRIPTS$ rocm-smi --showtopoweight

============================ ROCm System Management Interface ============================

================================ Weight between two GPUs =================================

       GPU0         GPU1         GPU2         GPU3         GPU4         GPU5
GPU6         GPU7

GPU0   0            15           15           15           15           15
15           15

GPU1   15           0            15           15           15           15
15           15

GPU2   15           15           0            15           15           15
15           15

GPU3   15           15           15           0            15           15
15           15

GPU4   15           15           15           15           0            15
15           15

GPU5   15           15           15           15           15           0
15           15

GPU6   15           15           15           15           15           15
0            15

GPU7   15           15           15           15           15           15
15           0

================================== End of ROCm SMI Log ===================================
```

```
jnpr@MI300X-01:~/SCRIPTS$ rocm-smi --showtopohops

============================ ROCm System Management Interface ============================

================================= Hops between two GPUs ==================================

       GPU0         GPU1         GPU2         GPU3         GPU4         GPU5
GPU6         GPU7

GPU0   0            1            1            1            1            1
1            1

GPU1   1            0            1            1            1            1
1            1

GPU2   1            1            0            1            1            1
1            1

GPU3   1            1            1            0            1            1
1            1

GPU4   1            1            1            1            0            1
1            1

GPU5   1            1            1            1            1            0
1            1

GPU6   1            1            1            1            1            1
0            1

GPU7   1            1            1            1            1            1
1            0

================================== End of ROCm SMI Log ===================================

jnpr@MI300X-01:~/SCRIPTS$ rocm-smi --showtopotype

============================ ROCm System Management Interface ============================

================================ Link Type between two GPUs ==============================

       GPU0         GPU1         GPU2         GPU3         GPU4         GPU5
```

```
GPU6         GPU7

GPU0   0           XGMI         XGMI         XGMI         XGMI         XGMI
XGMI         XGMI

GPU1   XGMI         0           XGMI         XGMI         XGMI         XGMI
XGMI         XGMI

GPU2   XGMI         XGMI         0           XGMI         XGMI         XGMI
XGMI         XGMI

GPU3   XGMI         XGMI         XGMI         0           XGMI         XGMI
XGMI         XGMI

GPU4   XGMI         XGMI         XGMI         XGMI         0           XGMI
XGMI         XGMI

GPU5   XGMI         XGMI         XGMI         XGMI         XGMI         0
XGMI         XGMI

GPU6   XGMI         XGMI         XGMI         XGMI         XGMI         XGMI
0            XGMI

GPU7   XGMI         XGMI         XGMI         XGMI         XGMI         XGMI
XGMI         0

================================== End of ROCm SMI Log ==================================
```

The `--shownodesbw` shows the bandwidth available internally for GPU to GPU internal communication:

```
jnpr@MI300X-01:/home/ben$ rocm-smi --shownodesbw

============================ ROCm System Management Interface ============================

======================================== Bandwidth =========================================

        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7

GPU0    N/A     50000-50000     50000-50000     50000-50000     50000-50000     50000-50000
50000-50000     50000-50000

GPU1    50000-50000     N/A     50000-50000     50000-50000     50000-50000     50000-50000
```

```
50000-50000     50000-50000

GPU2    50000-50000     50000-50000     N/A     50000-50000     50000-50000     50000-50000
50000-50000     50000-50000

GPU3    50000-50000     50000-50000     50000-50000     N/A     50000-50000     50000-50000
50000-50000     50000-50000

GPU4    50000-50000     50000-50000     50000-50000     50000-50000     N/A     50000-50000
50000-50000     50000-50000

GPU5    50000-50000     50000-50000     50000-50000     50000-50000     50000-50000     N/A
50000-50000     50000-50000

GPU6    50000-50000     50000-50000     50000-50000     50000-50000     50000-50000
50000-50000     N/A     50000-50000

GPU7    50000-50000     50000-50000     50000-50000     50000-50000     50000-50000
50000-50000     50000-50000     N/A

Format:         min-max;        Units:  mps

"0-0"   min-max         bandwidth       indicates       devices         are     not
connected       directly

================================== End of ROCm SMI Log ==================================
```

For additional options and details check:

```
rocm-smi-h
```

For more information about ROCm-SMI as well as for the newer AMD-SMI cli please check: ROCm Documentation, AMD SMI documentation, ROCm and AMD SMI

NICs and GPUs mappings

Next is to perform mapping of the NIC to GPUs as shown in the below steps. These will be same for both Thor2 and AMD Pollara 400 NIC.

The information from other commands can be combined with some of the options above to find correlation between GPU and NICs following these steps:

1. **Identify NUMA Nodes and GPUs**

   Use the output from `rocm-smi --showtoponuma` or just `rocm-smi --showtopo` to find mappings between GPUs and NUMA nodes.

   Look for **NUMA Affinity** for each GPU in the output. A description of what this attribute means is included later in this section.

   Note down which GPUs are associated with which NUMA nodes.

   Example:

```
jnpr@MI300X-01:/proc$ rocm-smi --showtoponuma

============================ ROCm System Management Interface ============================

======================================= Numa Nodes =======================================

GPU[0]          : (Topology) Numa Node: 0

GPU[0]          : (Topology) Numa Affinity: 0

GPU[1]          : (Topology) Numa Node: 0

GPU[1]          : (Topology) Numa Affinity: 0

GPU[2]          : (Topology) Numa Node: 0

GPU[2]          : (Topology) Numa Affinity: 0

GPU[3]          : (Topology) Numa Node: 0

GPU[3]          : (Topology) Numa Affinity: 0

GPU[4]          : (Topology) Numa Node: 1

GPU[4]          : (Topology) Numa Affinity: 1

GPU[5]          : (Topology) Numa Node: 1

GPU[5]          : (Topology) Numa Affinity: 1
```

```
GPU[6]          : (Topology) Numa Node: 1

GPU[6]          : (Topology) Numa Affinity: 1

GPU[7]          : (Topology) Numa Node: 1

GPU[7]          : (Topology) Numa Affinity: 1

================================== End of ROCm SMI Log ==================================
```

GPU 0–3 → NUMA Node 0

GPU 4–7 → NUMA Node 1

2. Identify NUMA Nodes for NICs

   Navigate to the `/sys/class/net/` directory and check the **NUMA node affinity** for each network interface (excluding lo or docker interfaces):

```
for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker)'); do

    numa_node=$(cat /sys/class/net/$iface/device/numa_node 2>/dev/null)

    echo "Interface: $iface, NUMA Node: $numa_node"

done
```

   Note the NUMA node affinity for each NIC interface.

   EXAMPLE:

```
jnpr@MI300X-01:~/SCRIPTS$ for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker)'); do

    numa_node=$(cat /sys/class/net/$iface/device/numa_node 2>/dev/null)

    echo "Interface: $iface, NUMA Node: $numa_node"

done

Interface: ens61f1np1, NUMA Node: 1

Interface: enxbe3af2b6059f, NUMA Node:
```

```
Interface: gpu0_eth, NUMA Node: 0

Interface: gpu1_eth, NUMA Node: 0

Interface: gpu2_eth, NUMA Node: 0

Interface: gpu3_eth, NUMA Node: 0

Interface: gpu4_eth, NUMA Node: 1

Interface: gpu5_eth, NUMA Node: 1

Interface: gpu6_eth, NUMA Node: 1

Interface: gpu7_eth, NUMA Node: 1

Interface: mgmt_eth, NUMA Node: 1

Interface: stor0_eth, NUMA Node: 0

Interface: stor1_eth, NUMA Node: 0
```

3. Correlate GPUs to NICs Based on NUMA Affinity

Using the NUMA node affinity from Step 1 (GPUs) and Step 2 (NICs), to map each GPU to NICs within the same NUMA node:

EXAMPLE:

```
GPU0 (NUMA 0):

  - NIC: gpu0_eth (NUMA 0)

  - NIC: gpu1_eth (NUMA 0)

  - NIC: gpu2_eth (NUMA 0)

  - NIC: gpu3_eth (NUMA 0)

  - NIC: stor0_eth (NUMA 0)
```

```
  - NIC: stor1_eth (NUMA 0)

GPU4 (NUMA 1):

  - NIC: gpu4_eth (NUMA 1)

  - NIC: gpu5_eth (NUMA 1)

  - NIC: gpu6_eth (NUMA 1)

  - NIC: gpu7_eth (NUMA 1)

  - NIC: mgmt_eth (NUMA 1)
```

NOTE: You can also use the following script to automate the steps above:

```
jnpr@MI300X-01:~/SCRIPTS$ cat GPU-to-NIC_YL.sh

#!/bin/bash

# Temporary data files

gpu_to_numa_file="GPU-to-NUMA.tmp"

nic_to_numa_file="NIC-to-NUMA.tmp"

output_file="NIC-to-GPU.txt"

# Clear or create the output file

> "$output_file"

# Step 1: Parse GPUs and NUMA nodes

echo "Step 1: Parsing GPUs and NUMA Nodes..."

rocm-smi --showtoponuma > /tmp/rocm_smi_output.tmp 2>/dev/null

if [[ $? -ne 0 ]]; then

    echo "Error: rocm-smi is not installed or failed to run."
```

```
    exit 1

fi

# Extract GPU and NUMA information

grep "GPU" /tmp/rocm_smi_output.tmp | grep "Numa Node" | awk -F'[ :]' '{print $2, $NF}' | sed
's/^/GPU /' > "$gpu_to_numa_file"

# Step 2: Parse NICs and NUMA nodes

echo "Step 2: Parsing NICs and NUMA Nodes..."

> "$nic_to_numa_file"

for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker)'); do

    numa_node=$(cat /sys/class/net/$iface/device/numa_node 2>/dev/null)

    if [[ $numa_node -ge 0 ]]; then

        echo "NIC $iface, NUMA Node: $numa_node" >> "$nic_to_numa_file"

    fi

done

# Step 3: Match GPUs to NICs based on NUMA affinity

echo "Step 3: Mapping GPUs to NICs..."

while read -r gpu_entry; do

    gpu=$(echo "$gpu_entry" | awk '{print $2}')

    gpu_numa=$(echo "$gpu_entry" | awk '{print $NF}')

    echo "GPU$gpu (NUMA $gpu_numa):" >> "$output_file"

    while read -r nic_entry; do

        nic=$(echo "$nic_entry" | awk '{print $2}' | sed 's/,//')
```

```
        nic_numa=$(echo "$nic_entry" | awk '{print $NF}')
        if [[ "$gpu_numa" == "$nic_numa" ]]; then
            echo "  - NIC: $nic" >> "$output_file"
        fi
    done < "$nic_to_numa_file"
done < "$gpu_to_numa_file"
# Output the result
echo "Mapping complete! Results saved in $output_file."
cat "$output_file"
```

EXAMPLE:

```
jnpr@MI300X-01:~/SCRIPTS$ ./GPU-to-NIC_YL.sh
Step 1: Parsing GPUs and NUMA Nodes...
Step 2: Parsing NICs and NUMA Nodes...
Step 3: Mapping GPUs to NICs...
Mapping complete! Results saved in NIC-to-GPU.txt.
GPU0 (NUMA 0):
  - NIC: gpu0_eth
  - NIC: gpu1_eth
  - NIC: gpu2_eth
  - NIC: gpu3_eth
  - NIC: stor0_eth
```

```
  - NIC: stor1_eth

GPU0 (NUMA 0):

  - NIC: gpu0_eth

  - NIC: gpu1_eth

  - NIC: gpu2_eth

  - NIC: gpu3_eth

  - NIC: stor0_eth

  - NIC: stor1_eth

GPU0 (NUMA 0):

  - NIC: gpu0_eth

  - NIC: gpu1_eth

  - NIC: gpu2_eth

  - NIC: gpu3_eth

  - NIC: stor0_eth

  - NIC: stor1_eth

GPU0 (NUMA 0):

  - NIC: gpu0_eth

  - NIC: gpu1_eth

  - NIC: gpu2_eth

  - NIC: gpu3_eth

  - NIC: stor0_eth

  - NIC: stor1_eth
```

```
GPU1 (NUMA 1):
  - NIC: ens61f1np1
  - NIC: gpu4_eth
  - NIC: gpu5_eth
  - NIC: gpu6_eth
  - NIC: gpu7_eth
  - NIC: mgmt_eth
GPU1 (NUMA 1):
  - NIC: ens61f1np1
  - NIC: gpu4_eth
  - NIC: gpu5_eth
  - NIC: gpu6_eth
  - NIC: gpu7_eth
  - NIC: mgmt_eth
GPU1 (NUMA 1):
  - NIC: ens61f1np1
  - NIC: gpu4_eth
  - NIC: gpu5_eth
  - NIC: gpu6_eth
  - NIC: gpu7_eth
  - NIC: mgmt_eth
```

```
GPU1 (NUMA 1):

  - NIC: ens61f1np1

  - NIC: gpu4_eth

  - NIC: gpu5_eth

  - NIC: gpu6_eth

  - NIC: gpu7_eth

  - NIC: mgmt_eth
```

You will notice that there is not a 1:1 GPU to NIC association. Instead, multiple NIC interfaces are associated with the GPU. This is because they belong to the same Non-Uniform Memory Access (NUMA) node affinity.

Systems employing a NUMA architecture contain collections of hardware resources including CPUs, GPUs memory, and PCIe devices (including NICs), grouped together in what is known as a “NUMA node”. These resources are considered "local" to each other. From the point of view of a GPU, devices in the same NUMA node are the most closely associated with that GPU. The NUMA node is identified by the NUMA Affinity.

Multiple NICs and GPUs may be connected to the same PCIe complex or switch within a NUMA node. This makes the NICs accessible to all GPUs sharing that complex. However, while all NICs in a NUMA node are accessible to any GPU in the same node, the NICs are allocated dynamically for usage by a given GPU, based on availability, traffic type, latency, and so on.

Communication Between GPUs on the Same NUMA Node (e.g., GPU1 ↔ GPU2):

GPUs on the same NUMA node (e.g., GPU1 and GPU2) communicate directly over the high-bandwidth, low-latency interconnect, such as **Infinity Fabric** (in AMD systems).

These interconnects avoid the CPU and main memory entirely, offering much faster communication compared to NUMA-crossing communication. Since both GPUs are "local" to the same memory controller and CPU, the communication path is highly optimized.

Communication Between GPUs on Different NUMA Nodes (e.g., GPU1 ↔ GPU4):

Communication between GPUs on different NUMA nodes (e.g., GPU1 on NUMA 0 and GPU4 on NUMA 1) must traverse additional layers of the system architecture, which introduces higher latency. The path typically follows:

- GPU1 → CPU (NUMA 0): Data is sent from GPU1 to the CPU on NUMA 0.
- Inter-NUMA Link: The CPUs in NUMA 0 and NUMA 1 are connected via an interconnect such as Infinity Fabric or UPI (Ultra Path Interconnect).
- CPU (NUMA 1) → GPU4: The data is forwarded from the CPU on NUMA 1 to GPU4.

Changing NIC attributes

This section shows you how to add or change a NIC's *Interface Name*, *MTU*, *DNS*, *IP Addresses* and *Routing table entries*.

Editing and reapplying the network configuration (netplan) file

The network configuration is described in the netplan *.yaml file found under: `/etc/netplan/`.

Notice that the actual file name might vary. Examples:

/etc/netplan/01-netcfg.yaml

/etc/netplan/00-installer-config.yaml

Changing any interface attribute involves editing this file and reapplying the network plan as shown below:

1. Find the default names of the logical interfaces.
   You can use the following steps to achieve this:

   Thor2 NIC output:

```
jnpr@MI300X-01:~$

> devnames1;
```

```
for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker|virbr)'); do

   device=$(ethtool -i $iface 2>/dev/null | grep 'bus-info' | awk '{print $2}');

   if [[ $device != 0000:* ]];

      then device="0000:$device"; fi;

   model=$(lspci -s $device 2>/dev/null | awk -F ': ' '{print $2}'); echo "$iface:$model" >>
devnames1;

done

jnpr@MI300X-01:~$ cat devnames1

ens61f1np1:Mellanox Technologies MT2910 Family [ConnectX-7]

enxbe3af2b6059f:

ens41np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens42np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens32np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens31np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens21np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens22np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens12np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens11np0:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

ens61f0np0:Mellanox Technologies MT2910 Family [ConnectX-7]

ens50f0np0:Mellanox Technologies MT2910 Family [ConnectX-7]

ens50f1np1:Mellanox Technologies MT2910 Family [ConnectX-7]
```

Interface `ens31np0`:

Where

- en: ethernet network interface.
- s31: indicates the physical location of the network interface on the system bus. slot number 31 on the bus.
- np0:
- n: Network (indicates it's a network port).
- p0: Port 0 (indicates it's the first port of this network interface).

AMD Pollara 400 NIC output

```
jnpr@mi300-01:~# > devnames1;

for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker|virbr)'); do

   device=$(ethtool -i $iface 2>/dev/null | grep 'bus-info' | awk '{print $2}');

   if [[ $device != 0000:* ]];

      then device="0000:$device"; fi;

   model=$(lspci -s $device 2>/dev/null | awk -F ': ' '{print $2}'); echo "$iface:$model" >>
devnames1;

done

jnpr@mi300-01:~# cat devnames1

ens61f1np1:Mellanox Technologies MT2910 Family [ConnectX-7]

eth3:

gpu0_eth:Pensando Systems DSC Ethernet Controller

gpu1_eth:Pensando Systems DSC Ethernet Controller

gpu2_eth:Pensando Systems DSC Ethernet Controller

gpu3_eth:Pensando Systems DSC Ethernet Controller
```

```
gpu4_eth:Pensando Systems DSC Ethernet Controller

gpu5_eth:Pensando Systems DSC Ethernet Controller

gpu6_eth:Pensando Systems DSC Ethernet Controller

gpu7_eth:Pensando Systems DSC Ethernet Controller

mgmt_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor0_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor1_eth:Mellanox Technologies MT2910 Family [ConnectX-7]
```

You can use the script `gpunic.py` to find mappings between GPUs and NIC per pcie bus, to identify how the NICS need to be renamed for consistency.

EXAMPLE:

```
jnpr@MI300X-01:~/SCRIPTS$ sudo python3 amd_map_nic_gpu.py

bus 0000:00:01.1:

  0000:05:00.0 (gpu) - GPU0

  0000:08:00.0 (gpu) - -

  0000:08:00.1 (gpu) - -

  0000:08:00.2 (gpu) - -

  0000:09:00.0 (nic) - gpu0_eth

bus 0000:20:01.1:

  0000:25:00.0 (gpu) - -

  0000:25:00.1 (gpu) - -

  0000:25:00.2 (gpu) - -

  0000:26:00.0 (nic) - gpu1_eth
```

```
  0000:29:00.0 (gpu) - GPU1
bus 0000:20:03.1:
  0000:31:00.0 (nic) - stor0_eth
  0000:31:00.1 (nic) - stor1_eth
bus 0000:40:01.1:
  0000:45:00.0 (gpu) - -
  0000:45:00.1 (gpu) - -
  0000:45:00.2 (gpu) - -
  0000:46:00.0 (nic) - gpu2_eth
  0000:49:00.0 (gpu) - GPU2
bus 0000:60:01.1:
  0000:65:00.0 (gpu) - GPU3
  0000:68:00.0 (gpu) - -
  0000:68:00.1 (gpu) - -
  0000:68:00.2 (gpu) - -
  0000:69:00.0 (nic) - gpu3_eth
bus 0000:60:05.4:
  0000:6e:00.0 (gpu) - -
bus 0000:80:01.1:
  0000:85:00.0 (gpu) - GPU4
  0000:88:00.0 (gpu) - -
```

```
  0000:88:00.1 (gpu) - -
  0000:88:00.2 (gpu) - -
  0000:89:00.0 (nic) - gpu4_eth
bus 0000:a0:01.1:
  0000:a5:00.0 (gpu) - -
  0000:a5:00.1 (gpu) - -
  0000:a5:00.2 (gpu) - -
  0000:a6:00.0 (nic) - gpu5_eth
  0000:a9:00.0 (gpu) - GPU5
bus 0000:c0:01.1:
  0000:c5:00.0 (gpu) - -
  0000:c5:00.1 (gpu) - -
  0000:c5:00.2 (gpu) - -
  0000:c6:00.0 (nic) - gpu6_eth
  0000:c9:00.0 (gpu) - GPU6
bus 0000:c0:03.1:
  0000:d2:00.0 (nic) - mgmt_eth
  0000:d2:00.1 (nic) - ens61f1np1
bus 0000:e0:01.1:
  0000:e5:00.0 (gpu) - GPU7
  0000:e8:00.0 (gpu) - -
  0000:e8:00.1 (gpu) - -
```

```
  0000:e8:00.2 (gpu) - -

  0000:e9:00.0 (nic) - gpu7_eth
```

To further identify the interfaces, you can use the `sudo ethtool <device> | grep Speed` command.

```
jnpr@MI300X-01:~/SCRIPTS$ sudo ethtool ens61f0np0| grep Speed

        Speed: 400000Mb/s

jnpr@MI300X-01:~/SCRIPTS$ sudo ethtool enp47s0f0np0| grep Speed

        Speed: 200000Mb/s

jnpr@MI300X-01:~/SCRIPTS$ sudo ethtool enp208s0f0np0| grep Speed

        Speed: 100000Mb/s
```

You want to make sure that the NICs connected to the GPU Backend fabric, the Storage Backend fabric, and the Frontend fabric are 400GE interfaces, 200GE interfaces, and 100GE interfaces respectively.

DEFAULT INTERFACE NAME	NEW NAME	Speed
enp6s0np0	gpu0_eth	400GE
enp35s0np0	gpu1_eth	400GE
enp67s0np0	gpu2_eth	400GE
enp102s0np0	gpu3_eth	400GE
enp134s0np0	gpu4_eth	400GE
enp163s0np0	gpu5_eth	400GE
enp195s0np0	gpu6_eth	400GE
enp230s0np0	gpu7_eth	400GE

(Continued)

DEFAULT INTERFACE NAME	NEW NAME	Speed
enp47s0f0np0	stor0_eth	200GE
enp47s0f0np1	stor1_eth	200GE
enp208s0f0np0	mgmt_eth	100GE

2. Find the interface's MAC address:

   You can use the `ip link show <device>` command.

   EXAMPLE:

```
jnpr@MI300X-01:~/SCRIPTS$ ip link show ens61f0np0 | grep "link/ether"

    link/ether 5c:25:73:66:c3:ee brd ff:ff:ff:ff:ff:ff

jnpr@MI300X-01:~/SCRIPTS$ ip link show enp35s0np0 | grep "link/ether"

    link/ether 5c:25:73:66:bc:5e brd ff:ff:ff:ff:ff:ff
```

DEFAULT INTERFACE NAME	NEW NAME	MAC address
enp6s0np0	gpu0_eth	7c:c2:55:bd:75:d0
enp35s0np0	gpu1_eth	7c:c2:55:bd:79:20
enp67s0np0	gpu2_eth	7c:c2:55:bd:7d:f0
enp102s0np0	gpu3_eth	7c:c2:55:bd:7e:20
enp134s0np0	gpu4_eth	7c:c2:55:bd:75:10
enp163s0np0	gpu5_eth	7c:c2:55:bd:7d:c0
enp195s0np0	gpu6_eth	7c:c2:55:bd:84:90

(Continued)

DEFAULT INTERFACE NAME	NEW NAME	MAC address
enp230s0np0	gpu7_eth	7c:c2:55:bd:83:10
enp47s0f0np0	stor0_eth	5c:25:73:66:bc:5e
enp47s0f0np1	stor1_eth	5c:25:73:66:bc:5f
enp208s0f0np0	mgmt_eth	5c:25:73:66:c3:ee

3. Modify the netplan configuration file using the new name and MAC addresses determined in the previous steps.
   EXAMPLE:

```
network:

  version: 2

  ethernets:

    gpu0_eth:

      match:

        macaddress: 7c:c2:55:bd:75:d0   <= MAC address associated to the original ens61f0np0.
Will become gpu0_eth.

      dhcp4: false

      mtu: 9000 <= Interface's MTU (default = 1500)

      addresses:

        - 10.200.16.18/24       <= New IP address(s)

      routes:

        - to: 10.200.0.0/16     <= New route(s). Example shows route for 10.200.0.0/16 via
10.200.16.254
```

```
          via: 10.200.16.254
          from: 10.200.16.18
      set-name: gpu0_eth        <= New interface name
---more---
```

Make sure to keep proper indentation, and hyphens were appropriate (e.g. before IP addresses, routes, etc.) when editing the file. For the IP addresses make sure to include the subnet mask.

The following is an example of the netplan configuration file for one of the MI300X servers in the lab:

```
jnpr@MI300X-01:/etc/netplan$ cat 00-installer-config.yaml
network:
  version: 2
  ethernets:
    mgmt_eth:
      match:
        macaddress: 5c:25:73:66:c3:ee
      dhcp4: false
      addresses:
        - 10.10.1.25/31
      nameservers:
        addresses:
          - 8.8.8.8
```

```
    routes:
      - to: default
        via: 10.10.1.24
    set-name: mgmt_eth
  stor0_eth:
    match:
      macaddress: 5c:25:73:66:bc:5e
    dhcp4: false
    mtu: 9000
    addresses:
      - 10.100.5.3/31
    routes:
      - to: 10.100.0.0/21
        via: 10.100.5.2
    set-name: stor0_eth
  stor1_eth:
    match:
      macaddress: 5c:25:73:66:bc:5f
    dhcp4: false
    mtu: 9000
    addresses:
```

```
      - 10.100.5.5/31
    routes:
      - to: 10.100.0.0/21
        via: 10.100.5.4
    set-name: stor1_eth
  gpu0_eth:
    match:
      macaddress: 7c:c2:55:bd:75:d0
    dhcp4: false
    mtu: 9000
    addresses:
      - 10.200.16.18/24
    routes:
      - to: 10.200.0.0/16
        via: 10.200.16.254
        from: 10.200.16.18
    set-name: gpu0_eth
  gpu1_eth:
    match:
      macaddress: 7c:c2:55:bd:79:20
    dhcp4: false
    mtu: 9000
```

```
    addresses:
      - 10.200.17.18/24
    routes:
      - to: 10.200.0.0/16
        via: 10.200.17.254
        from: 10.200.17.18
    set-name: gpu1_eth
  gpu2_eth:
    match:
      macaddress: 7c:c2:55:bd:7d:f0
    dhcp4: false
    mtu: 9000
    addresses:
      - 10.200.18.18/24
    routes:
      - to: 10.200.0.0/16
        via: 10.200.18.254
        from: 10.200.18.18
    set-name: gpu2_eth
  gpu3_eth:
    match:
```

```
        macaddress: 7c:c2:55:bd:7e:20
    dhcp4: false
    mtu: 9000
    addresses:
      - 10.200.19.18/24
    routes:
      - to: 10.200.0.0/16
        via: 10.200.19.254
        from: 10.200.19.18
    set-name: gpu3_eth
  gpu4_eth:
    match:
      macaddress: 7c:c2:55:bd:75:10
    dhcp4: false
    mtu: 9000
    addresses:
      - 10.200.20.18/24
    routes:
      - to: 10.200.0.0/16
        via: 10.200.20.254
        from: 10.200.20.18
    set-name: gpu4_eth
```

```
gpu5_eth:
  match:
    macaddress: 7c:c2:55:bd:7d:c0
  dhcp4: false
  mtu: 9000
  addresses:
    - 10.200.21.18/24
  routes:
    - to: 10.200.0.0/16
      via: 10.200.21.254
      from: 10.200.21.18
  set-name: gpu5_eth
gpu6_eth:
  match:
    macaddress: 7c:c2:55:bd:84:90
  dhcp4: false
  mtu: 9000
  addresses:
    - 10.200.22.18/24
  routes:
    - to: 10.200.0.0/16
```

```
          via: 10.200.22.254
          from: 10.200.22.18
      set-name: gpu6_eth
    gpu7_eth:
      match:
        macaddress: 7c:c2:55:bd:83:10
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.23.18/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.23.254
          from: 10.200.23.18
      set-name: gpu7_eth
```

4. Save the file and apply the changes using the netplan apply command.

   jnpr@MI300X-01:/etc/netplan$ sudo netplan apply

   jnpr@MI300X-01:/etc/netplan$

5. Verify the changes were correctly applied.

Check that the new interface names are correct:

Thor2 NIC output:

```
root@MI300X-01:/home/jnpr/SCRIPTS#
> devnames;
```

```
for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker|virbr)'); do

   device=$(ethtool -i $iface 2>/dev/null | grep 'bus-info' | awk '{print $2}');

   if [[ $device != 0000:* ]];

      then device="0000:$device"; fi;

   model=$(lspci -s $device 2>/dev/null | awk -F ': ' '{print $2}'); echo "$iface:$model" >>
devnames;

done

root@MI300X-01:/home/jnpr/SCRIPTS# cat devnames

ens61f1np1:Mellanox Technologies MT2910 Family [ConnectX-7]

enxbe3af2b6059f:

gpu0_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu1_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu2_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu3_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu4_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu5_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu6_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu7_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

mgmt_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor0_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor1_eth:Mellanox Technologies MT2910 Family [ConnectX-7]
```

Notice that the gpu#_eth (#=0-7) interfaces are Broadcom BCM97608 interfaces while the mgmt_eth and stor#_eth interfaces are Mellanox MT2910 (ConnectX-7) interfaces. This will become important in the next section where we will cover the interfaces CoS configuration.

AMD Pollara NIC output for same command:

```
jnpr@mi300-01:~$ > devnames;

for iface in $(ls /sys/class/net/ | grep -Ev '^(lo|docker|virbr)'); do

   device=$(ethtool -i $iface 2>/dev/null | grep 'bus-info' | awk '{print $2}');

   if [[ $device != 0000:* ]];

      then device="0000:$device"; fi;

   model=$(lspci -s $device 2>/dev/null | awk -F ': ' '{print $2}'); echo "$iface:$model" >>
devnames;

done

jnpr@mi300-01:~$ cat devnames

ens61f1np1:Mellanox Technologies MT2910 Family [ConnectX-7]

eth3:

gpu0_eth:Pensando Systems DSC Ethernet Controller

gpu1_eth:Pensando Systems DSC Ethernet Controller

gpu2_eth:Pensando Systems DSC Ethernet Controller

gpu3_eth:Pensando Systems DSC Ethernet Controller

gpu4_eth:Pensando Systems DSC Ethernet Controller

gpu5_eth:Pensando Systems DSC Ethernet Controller

gpu6_eth:Pensando Systems DSC Ethernet Controller

gpu7_eth:Pensando Systems DSC Ethernet Controller
```

```
mgmt_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor0_eth:Mellanox Technologies MT2910 Family [ConnectX-7]

stor1_eth:Mellanox Technologies MT2910 Family [ConnectX-7]
```

Notice that the gpu#_eth (#=0-7) interfaces are AMD Pollara 400 NIC interfaces while the mgmt_eth and stor#_eth interfaces are Mellanox MT2910 (ConnectX-7) interfaces. This will become important in the next section where we will cover the interfaces CoS configuration. eth3 interface is the supermicro IPMI interface.

Verify that the IP addresses were configured correctly:

```
user@MI300X-03:~/scripts$ ip address show gpu0_eth

4: gpu0_eth: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group default qlen 1000

    link/ether 6c:92:cf:87:cc:00 brd ff:ff:ff:ff:ff:ff

    inet 10.200.24.22/24 brd 10.200.24.255 scope global gpu0_eth

       valid_lft forever preferred_lft forever

    inet6 fe80::6e92:cfff:fe87:cc00/64 scope link

       valid_lft forever preferred_lft forever
```

OR

```
jnpr@MI300X-01:/etc/netplan$ ifconfig  gpu0_eth

gpu0_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000

        inet 10.200.16.18  netmask 255.255.255.0  broadcast 10.200.16.255

        inet6 fe80::7ec2:55ff:febd:75d0  prefixlen 64  scopeid 0x20<link>

        ether 7c:c2:55:bd:75:d0  txqueuelen 1000  (Ethernet)

        RX packets 253482  bytes 28518251 (28.5 MB)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 38519  bytes 10662707 (10.6 MB)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Check that the routes were added correctly to the routing table:

```
jnpr@MI300X-01:/etc/netplan$ route | grep mgmt_eth

default         _gateway        0.0.0.0         UG    0      0        0 mgmt_eth

10.10.1.24      0.0.0.0         255.255.255.254 U     0      0        0 mgmt_eth

jnpr@MI300X-01:/etc/netplan$ route | grep gpu0_eth

10.200.0.0      10.200.16.254   255.255.0.0     UG    0      0        0 gpu0_eth

10.200.16.0     0.0.0.0         255.255.255.0   U     0      0        0 gpu0_eth
```

OR

```
user@MI300X-03:~/scripts$ ip route show | grep gpu0_eth

10.200.24.0/24 dev gpu0_eth proto kernel scope link src 10.200.24.22
```

Check address resolution:

```
jnpr@MI300X-01:/etc/netplan$ ping google.com -c 5 -n

PING google.com (142.250.188.14) 56(84) bytes of data.

64 bytes from 142.250.188.14: icmp_seq=1 ttl=113 time=2.16 ms

64 bytes from 142.250.188.14: icmp_seq=2 ttl=113 time=2.43 ms

64 bytes from 142.250.188.14: icmp_seq=3 ttl=113 time=191 ms

64 bytes from 142.250.188.14: icmp_seq=4 ttl=113 time=50.6 ms

64 bytes from 142.250.188.14: icmp_seq=5 ttl=113 time=12.0 ms
```

```
--- google.com ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 4005ms

rtt min/avg/max/mdev = 2.158/51.596/190.818/71.851 ms
```

AMD Pollara firmware and dependent libraries

Note: For AMD drivers and host utilities, please reach out to your regional AMD representative.

For brevity the steps described here only pertain to enabling RCCL test for AMD Pollara 400 NIC and hence all the necessary dependent software and libraries are required to be installed for RCCL test to run. The steps involved pertain to the libraries listed in AMD Server and NIC Firmware and RCCL supporting libraries table.

1. Ensure that ubuntu OS version is 22.04 as suggested in section AMD Server and NIC Firmware and RCCL supporting libraries

```
Install ROCm library as suggested in below steps.

wget https://repo.radeon.com/amdgpu-install/6.3.3/ubuntu/jammy/amdgpu-
install_6.3.60303-1_all.deb

sudo apt install ./amdgpu-install_6.3.60303-1_all.deb

sudo apt update

sudo apt install amdgpu-dkms rocm

sudo apt install cmake libstdc++-12-dev
```

2. Install RCCL library as suggested in below steps. Note that the RCCL and ANP are private libraries provided by AMD.

```
tar xf rccl-7961624_may21.tgz

cd rccl-7961624
```

```
./install.sh -l --prefix=build --disable-mscclpp --disable-msccl-kernel
```

3. Install Unified Communication Framework (UCX). The Unified Communication Framework (UCX), is an open source, cross-platform framework designed to provide a common set of communication interfaces for various network programming models and interfaces, refer AMD documentation for more information.

```
sudo apt install libtool

git clone https://github.com/openucx/ucx.git

cd ucx

./autogen.sh

mkdir build

cd build

../configure --prefix=/opt/ucx --with-rocm=/opt/rocm

../configure --prefix=/opt/ucx

make -j $(nproc)

sudo make -j $(nproc) install
```

4. Next Install OpenMPI. Note that the OpenMPI is a GitHub link and hence GitHub credentials may be required. The Open MPI Project is an open source Message Passing Interface implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available, refer OpenMPI for more information.

```
sudo apt install flex

git clone --recursive https://github.com/open-mpi/ompi.git

cd ompi
```

```
./autogen.pl

mkdir build

cd build

../configure --prefix=/opt/ompi --with-ucx=/opt/ucx --with-rocm=/opt/rocm

make -j $(nproc)

sudo make install
```

5. Install Pollara drivers and firmware. This is AMD provided firmware bundle. This firmware will also install the 'nicctl' command line utility to interact with the Pollara NICs and run commands to reset cards or configure QOS etc.

```
# Prerequisites

sudo apt install device-tree-compiler policycoreutils ninja-build jq pkg-config libnl-3-dev
libnl-route-3-dev libpci-dev

# Untar the bundle itself

tar xf ainic_bundle_1.113.0-a-2.tar.gz

# cd into the bundle directory

cd ainic_bundle_1.113.0-a-2

# untar the host software

tar xf host_sw_pkg.tar.gz

# cd into the host software directory

cd host_sw_pkg

# install the drivers and software

sudo ./install.sh
```

Output of the Firmware version:

```
jnpr@MI300-01:~/ainic_bundle_1.110.0-a-79$ sudo nicctl update firmware --image ./
ainic_fw_salina.tar --log-file /tmp/amd_ainic_upgrade.log

--------------------------------------------------------------------------------

Card Id                                 Stage                   Progress

--------------------------------------------------------------------------------

42424650-4c32-3530-3130-313346000000    Done                    100% [02:50.941]

42424650-4c32-3530-3130-313844000000    Done                    100% [02:41.200]

42424650-4c32-3530-3130-313242000000    Done                    100% [02:51.584]

42424650-4c32-3530-3130-304341000000    Done                    100% [02:51.281]

42424650-4c32-3530-3130-313434000000    Done                    100% [02:31.062]

42424650-4c32-3530-3130-314537000000    Done                    100% [02:51.480]

42424650-4c32-3530-3130-314436000000    Done                    100% [02:51.077]

42424650-4c32-3530-3130-304435000000    Done                    100% [02:51.367]

NIC 42424650-4c32-3530-3130-313346000000 (0000:06:00.0) : Successful

NIC 42424650-4c32-3530-3130-313844000000 (0000:23:00.0) : Successful

NIC 42424650-4c32-3530-3130-313242000000 (0000:43:00.0) : Successful

NIC 42424650-4c32-3530-3130-304341000000 (0000:66:00.0) : Successful

NIC 42424650-4c32-3530-3130-313434000000 (0000:86:00.0) : Successful

NIC 42424650-4c32-3530-3130-314537000000 (0000:a3:00.0) : Successful

NIC 42424650-4c32-3530-3130-314436000000 (0000:c3:00.0) : Successful
```

```
NIC 42424650-4c32-3530-3130-304435000000 (0000:e6:00.0) : Successful
```

6. Once the firmware install is complete, then run the reset card so as to reflect the firmware version.
   Pollara NIC output of Firmware update

```
jnpr@mi300-01:~$ sudo nicctl reset card --all

NIC 42424650-4c32-3530-3130-313346000000 (0000:06:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-313844000000 (0000:23:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-313242000000 (0000:43:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-304341000000 (0000:66:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-313434000000 (0000:86:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-314537000000 (0000:a3:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-314436000000 (0000:c3:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-304435000000 (0000:e6:00.0) : Card reset triggered, wait for
completion (75 secs)

NIC 42424650-4c32-3530-3130-313346000000 (0000:06:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-313844000000 (0000:23:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-313242000000 (0000:43:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-304341000000 (0000:66:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-313434000000 (0000:86:00.0) : Card reset successful
```

```
NIC 42424650-4c32-3530-3130-314537000000 (0000:a3:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-314436000000 (0000:c3:00.0) : Card reset successful

NIC 42424650-4c32-3530-3130-304435000000 (0000:e6:00.0) : Card reset successful

jnpr@mi300-01:~$ sudo nicctl show card --detail | grep Firm

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79

Firmware version               : 1.110.0-a-79
```

7. Install ANP plugin. ANP is a plugin library designed to enhance the RCCL collective communication library with extended network transport support. The ANP Plugin library is a private AMD library.

```
sudo apt install libboost-dev

export RCCL_BUILD=/home/${User}/pollara/rccl-7961624/build/release

export MPI_INCLUDE=/opt/ompi/include

export MPI_LIB_PATH=/opt/ompi/lib

make RCCL_BUILD=$RCCL_BUILD MPI_INCLUDE=$MPI_INCLUDE MPI_LIB_PATH=$MPI_LIB_PATH
```

8. Lastly Build the RCCL tests.

```
Build rccl-tests
```

```
git clone https://github.com/ROCm/rccl-tests.git

cd rccl-tests

make MPI=1 MPI_HOME=/opt/ompi NCCL_HOME=/home/${User}/pollara/rccl-7961624/build/release
CUSTOM_RCCL_LIB=/home/${User}/pollara/rccl-7961624/build/release/librccl.so -j $(nproc)

make MPI=1 MPI_HOME=/opt/ompi NCCL_HOME=/home/dbarmann/pollara/rccl-7961624/build/release
HIP_HOME=/home/${User}/pollara/rccl-7961624/build/release CUSTOM_RCCL_LIB=/home/${User}/
pollara/rccl-7961624/build/release/librccl.so -j $(nproc)
```

Configuring AMD DCQCN (ECN/PFC) and TOS/DSCP for RDMA Traffic

In the "IP Services for AI Networks section" on page 32 we discussed the need for congestion control and traffic prioritization in the Backend GPU fabric to transport RoCE traffic between GPU servers. For these mechanisms to work properly, the servers need to be configured to properly react to congestions notifications from both ECN and PFC, and to mark the RDMA and non-RDMA traffic properly (matching the classification configuration of the fabric). We will cover how to configure the AMD servers to meet this requirement.

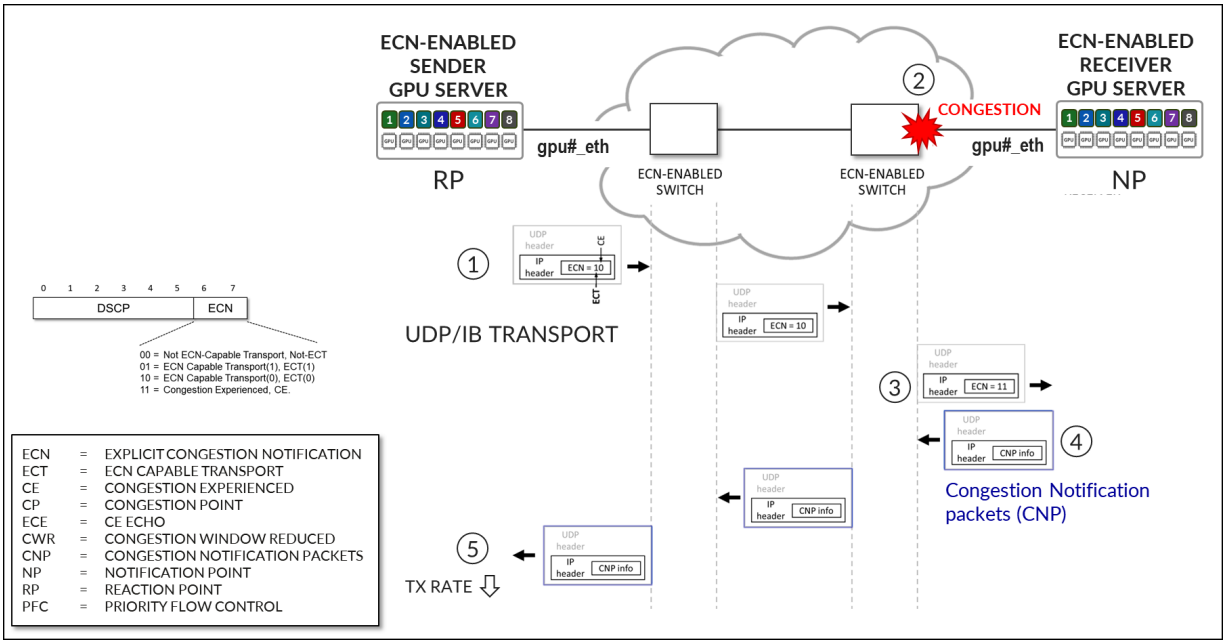
Congestion Control (CC) or ECN (Explicit congestion Notification)

Congestion Control (CC) or ECN (Explicit congestion Notification) is a standard (RFC 3168) backpressure mechanism for ethernet network devices that signals congestion and causes the traffic to temporarily slow down to avoid packet drops

ECN for RoCE traffic relies on fabric switches that can detect congestion and implement ECN marking for traffic downstream, and devices that can respond to these markings, as shown in Figure 63.

- the receiving NIC or Notification point (NP) which transmits CNP when receiving ECN marked packets
- the sending NIC or Reaction point (RP) that receives the CNP packets and reacts accordingly.

Figure 53: DCQCN – ECN Operation



Details about DCQCN – ECN (Congestion Control in Broadcom terminology) implementation in the **BCM5741X** Ethernet network adapter acting as NP and RP, can be found in the following documents Traffic Control Synopsis and RoCE Congestion Control.

Priority Flow Control (PFC)

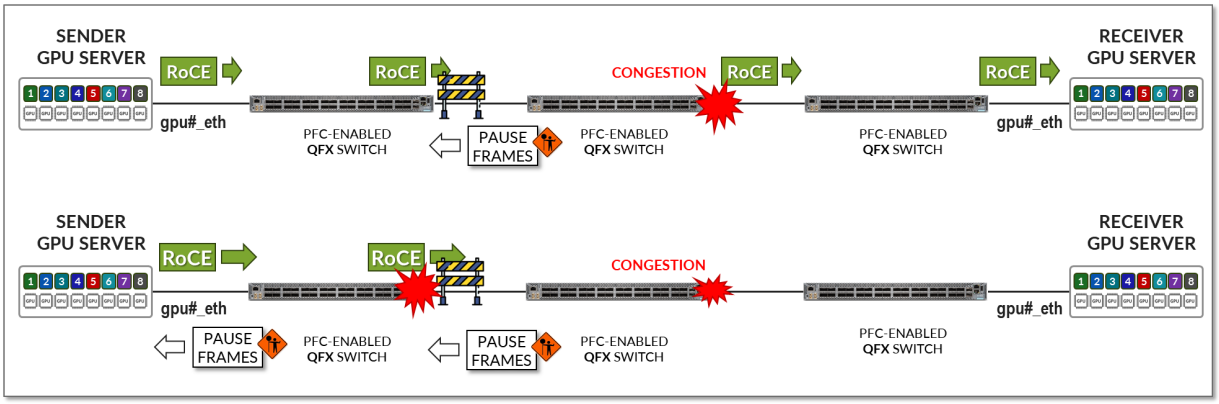
Priority Flow Control (PFC) is a standard (IEEE 802.1Qbb) backpressure mechanism for ethernet network devices that signals congestion and causes traffic on a particular priority to temporarily stop to avoid packet drops.

PFC for RoCE traffic relies on fabric switches that can detect congestion and generate PFC Pause frames upstream and devices that can respond to these markings:

- the sending NIC that receives the PFC Pause frames and reacts accordingly.

Details about DCQCN – PFC implementation in **BCM5741X** Ethernet network adapters acting as RP can be found in the following documents Traffic Control Synopsis, Priority Flow Control Feature in Ethernet Network Adapters, and Quality of Service

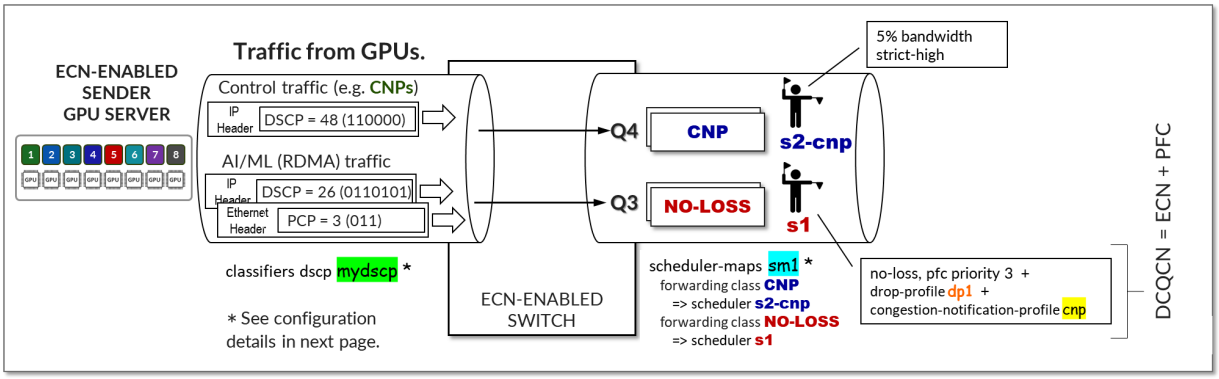
Figure 54: DCQCN – PFC Operation



TOS/DSCP for RDMA Traffic

RDMA traffic must be properly marked to allow the switch to correctly classify it, and to place it in the lossless queue for proper treatment. Marking can be either DSCP within the IP header, or PCP in the ethernet frame vlan-tag field. Whether DSCP or PCP is used depends on whether the interface between the GPU server and the switch is doing vlan tagging (802.1q) or not. Figure 64 shows how RDMA and CNP are marked differently and as a result, the fabric switch classified and schedules the two types of packets differently.

Figure 55: TOS/DSCP operation



DCQCN configuration for RDMA Traffic on NICs

IN THIS SECTION

Broadcom BCM57608 Thor2 DCQCN configuration for RDMA Traffic

Default DCQN-ECN/PFC attributes in AMD servers.

The network interface adapters are configured with the following Class of Service (including DCQCN-ECN) parameters for RoCE traffic:

For Thor2 NIC adapter:

- RoCEv2 (RDMA over IPv4) enabled
- Congestion Control (ECN) and PFC enabled

- RoCE traffic tagged with **DSCP 26** on PRIORITY 3
- RoCE CNP traffic tagged with **DSCP 48** and PRIORITY 7

Mapping Broadcom and logical interface names to configure DCQN-ECN/PFC and TOS/DSCP for RDMA Traffic attributes in AMD servers

DCQCN ECN, PFC and traffic marking need to be configured on the interfaces connected to the GPU backend; that is on the gpu#_eth (#=0-7) interfaces only.

On the section "Changing NIC attributes" on page 96 section of these document, we determined that the gpu#_eth interfaces in our servers, are Broadcom BCM97608 (shown below) NICs.

```
root@MI300X-01:/home/jnpr/SCRIPTS# cat devnames | grep gpu

gpu0_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu1_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu2_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu3_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu4_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu5_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu6_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

gpu7_eth:Broadcom Inc. and subsidiaries BCM57608 25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
```

All the steps for configuring Class of Service in this section will be focused on these Broadcom interfaces.

We will be using a combination of Linux system commands and Broadcom tools to enable, tune and monitor DCQCN ECN/PFC operation and RoCE traffic marking. For some of these commands we will need to find the Broadcom interface name associated with each gpu interface. Follow these steps to find these mappings:

1. Find the PCI address of each gpu#_eth interface using the following logic:

```
for iface in $(ls /sys/class/net | grep -E 'gpu[0-9]+_eth'); do

    pci_addr=$(readlink -f /sys/class/net/$iface/device | awk -F '/' '{print $NF}')

    echo "$iface => $pci_addr"

done1
```

EXAMPLE:

```
root@MI300X-01:/home/jnpr/SCRIPTS# for iface in $(ls /sys/class/net | grep -E
'gpu[0-9]+_eth'); do

    pci_addr=$(readlink -f /sys/class/net/$iface/device | awk -F '/' '{print $NF}')

    echo "$iface => $pci_addr"

done

gpu0_eth => 0000:06:00.0

gpu1_eth => 0000:23:00.0

gpu2_eth => 0000:43:00.0

gpu3_eth => 0000:66:00.0

gpu4_eth => 0000:86:00.0

gpu5_eth => 0000:a3:00.0

gpu6_eth => 0000:c3:00.0

gpu7_eth => 0000:e6:00.0
```

2. Find the bnxt_re# (#=0-7) devices that corresponds to each PCI address using the following logic:

```
for pci in $(find /sys/class/infiniband -type l -exec basename {} \;); do

    pci_addr=$(readlink -f /sys/class/infiniband/$pci/device | awk -F '/' '{print $NF}')

    echo "$pci => $pci_addr" |grep bnxt

done
```

EXAMPLE:

```
root@MI300X-01:/home/jnpr/SCRIPTS# for pci in $(find /sys/class/infiniband -type l -exec
basename {} \;); do

    pci_addr=$(readlink -f /sys/class/infiniband/$pci/device | awk -F '/' '{print $NF}')

    echo "$pci => $pci_addr" |grep bnxt

done

bnxt_re5 => 0000:a3:00.0

bnxt_re3 => 0000:66:00.0

bnxt_re1 => 0000:23:00.0

bnxt_re6 => 0000:c3:00.0

bnxt_re4 => 0000:86:00.0

bnxt_re2 => 0000:43:00.0

bnxt_re0 => 0000:06:00.0

bnxt_re7 => 0000:e6:00.0
```

3. MAP the GPU interface bnxt_re# or mlx5_# interface names.

Combine the outputs from steps 1 and 2 to create a full mapping from gpu#_eth to bnxt_re# or mlx5_#. You can see from the outputs that for example gpu0_eth corresponds to bnxt_re3 (0000:66:00.0)

You can use the following logic to simplify the process:

```
echo "GPU-to-NIC Mapping:"

for iface in $(ls /sys/class/net | grep -E 'gpu[0-9]+_eth'); do

    pci_addr=$(readlink -f /sys/class/net/$iface/device | awk -F '/' '{print $NF}')

    rdma_dev=$(find /sys/class/infiniband -type l -exec basename {} \; | while read rdma; do

        rdma_pci=$(readlink -f /sys/class/infiniband/$rdma/device | awk -F '/' '{print $NF}')

        if [[ "$pci_addr" == "$rdma_pci" ]]; then echo "$rdma"; fi

    done)

    echo "$iface => $pci_addr => $rdma_dev"

done
```

EXAMPLE:

```
root@MI300X-01:/home/jnpr/SCRIPTS# echo "GPU-to-NIC Mapping:"

for iface in $(ls /sys/class/net | grep -E 'gpu[0-9]+_eth'); do

    pci_addr=$(readlink -f /sys/class/net/$iface/device | awk -F '/' '{print $NF}')

    rdma_dev=$(find /sys/class/infiniband -type l -exec basename {} \; | while read rdma; do

        rdma_pci=$(readlink -f /sys/class/infiniband/$rdma/device | awk -F '/' '{print $NF}')

        if [[ "$pci_addr" == "$rdma_pci" ]]; then echo "$rdma"; fi

    done)

    echo "$iface => $pci_addr => $rdma_dev"

done

GPU-to-NIC Mapping:
```

```
gpu0_eth => 0000:06:00.0 => bnxt_re0

gpu1_eth => 0000:23:00.0 => bnxt_re1

gpu2_eth => 0000:43:00.0 => bnxt_re2

gpu3_eth => 0000:66:00.0 => bnxt_re3

gpu4_eth => 0000:86:00.0 => bnxt_re4

gpu5_eth => 0000:a3:00.0 => bnxt_re5

gpu6_eth => 0000:c3:00.0 => bnxt_re6

gpu7_eth => 0000:e6:00.0 => bnxt_re7
```

Configuring DCQN-ECN/PFC and TOS/DSCP for RDMA Traffic attributes in AMD servers (Broadcom interfaces)

Some of the parameters related to DCQN-ECN/PFC and TOS/DSCP are listed in the following table:

Table 15. Server DCQCN configuration parameters

PARAMETER	DESCRIPTION	DEFAULT
cc_mode	0 for Deterministic Marking (DCQCN-D) 1 for Probabilistic Marking (DCQCN-P)	1
cnp_ecn	Enables/disables ECN	0x1 (enabled)
cnp_dscp	DSCP value for RoCE congestion notification packets	48
cnp_prio	Priority for RoCE congestion notification packets	7

(Continued)

PARAMETER	DESCRIPTION	DEFAULT
cnp_ratio_th	Defines the threshold ratio for generating CNPs. It determines the rate at which CNPs are sent in response to congestion, helping to control the feedback mechanism's aggressiveness.	0x0
ecn_enable	Enable congestion control.	0x1 (enabled)
ecn_marking	Enables tagging of packets as ECN-enabled. ECN = 01	0x1 (enabled)
default_roce_mode	Sets the default RoCE mode for RDMA	RoCE v2
default_roce_tos	Sets the default ToS value for RDMA traffic	104
roce_dscp	DSCP value for RoCE packets.	26
roce_prio	Priority for RoCE packets.	3
rtt	Time period (µs) over which cnp and transmitted packets counts accumulate. At the end of rtt, the ratio between CNPs and TxPkts is computed, and the CP is updated.	40 µs.

BCM95741X Ethernet network adapters support three transmit and receive queues for each Ethernet port: 0, 4, and 5.

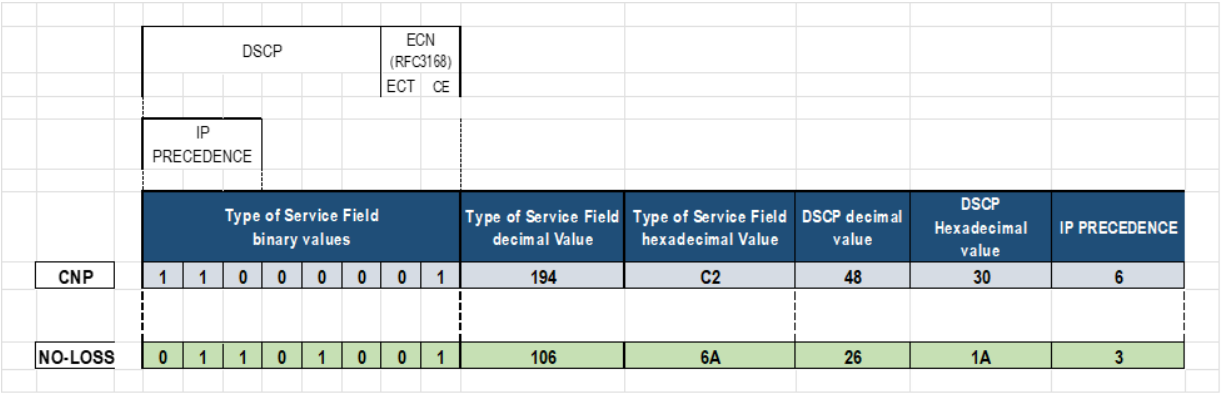
BCM95750X Ethernet network adapters support eight transmit and receive queues for each Ethernet port: 0 through 7.

By default, all queues are configured for weighted-fair-queueing (WFQ), with priority 0 traffic mapped to queue 4.

When the RoCE bnxt_re driver is loaded, CoSQ 0 is configured for lossless traffic, and CoSQ 5 is changed from WFQ to strict priority (SP) for CNP processing.

RoCE and CNP traffic can be tagged with different DSCP values or use VLAN tags instead.

By default, the ToS field is set to 104, which means DSCP is set to 48 and the ECN bits are set to 10 (ECN-enabled).



	Type of Service Field binary values								Type of Service Field decimal Value	Type of Service Field hexadecimal Value	DSCP decimal value	DSCP Hexadecimal value	IP PRECEDENCE
CNP	1	1	0	0	0	0	0	1	194	C2	48	30	6
NO-LOSS	0	1	1	0	1	0	0	1	106	6A	26	1A	3

These parameters can be adjusted using three different methods:

- Configuring DCQCN/RDMA marking values directly
- Configuring DCQCN/RDMA marking values using Broadcom tools such as `niccli`, or `lldptool` directly
- Configuring DCQCN/RDMA marking values using the `bnxt_setupcc.sh` utility, which uses either `niccli` or `lldptool` (default) behind the scenes.

The following sections will describe the steps to make changes using these different options.

NOTE: Please ensure all changes are consistent with the configuration of switches within the fabric. Example:

```
set class-of-service classifiers dscp mydscp forwarding-class CNP loss-priority low code-points
110000

set class-of-service classifiers dscp mydscp forwarding-class NO-LOSS loss-priority low code-
points 011010

set class-of-service forwarding-classes class NO-LOSS pfc-priority 3
```

Configuring DCQN-ECN/PFC and TOS/DSCP for RDMA Traffic attributes directly

You can make changes to the DCQCN and traffic marking by directly editing the files that contain the values of each parameter. This method is the easiest, and does not require installation of any additional

tools, however, it is not an option for PFC related parameters, nor is it supported on all types of network adapters.

To complete these changes for a specific interface, you must be under in the proper interface directory, following these steps:

1. Create interface directories for qos related values

   We determined the mappings between the gpu#_eth interfaces and the corresponding Broadcom interface names

   GPU-to-NIC Mapping:

   gpu0_eth => 0000:06:00.0 => bnxt_re0

   gpu1_eth => 0000:23:00.0 => bnxt_re1

   gpu2_eth => 0000:43:00.0 => bnxt_re2

   gpu3_eth => 0000:66:00.0 => bnxt_re3

   gpu4_eth => 0000:86:00.0 => bnxt_re4

   gpu5_eth => 0000:a3:00.0 => bnxt_re5

   gpu6_eth => 0000:c3:00.0 => bnxt_re6

   gpu7_eth => 0000:e6:00.0 => bnxt_re7

   We will use the Broadcom interface names to create the directories (rdma_cm and bnxt_re) where the DCQCN attributes as well as other parameters and statistics will be located for each interface.

   The interface specific directories do not exist until created using the following commands:

```
cd /sys/kernel/config

mkdir -p /rdma_cm/<Broadcom-interface-name>

mkdir -p /bnxt_re/<Broadcom-interface-name>
```

   Notice that these two directories must be present.

```
root@MI300X-01:/# cd /sys/kernel/config/ls

bnxt_re  rdma_cm
```

If the rdma_cm directory for example is missing, try the following:

```
root@MI300X-01:/sys/kernel/config# sudo modprobe rdma_cm



root@MI300X-01:/sys/kernel/config# lsmod | grep rdma_cm

rdma_cm               147456  0

iw_cm                  61440  1 rdma_cm

ib_cm                 151552  1 rdma_cm

ib_core               507904  6 rdma_cm,iw_cm,bnxt_re,ib_uverbs,mlx5_ib,ib_cm
```

EXAMPLE:

```
root@MI300X-01:/# cd /sys/kernel/config/bnxt_re

root@MI300X-01:/sys/kernel/config/bnxt_re#

(NO FILES LISTED)

root@MI300X-01:/# cd /sys/kernel/config/rdma_cm

root@MI300X-01:/sys/kernel/config/rdma_cm# ls

(NO FILES LISTED)



root@MI300X-01:/sys/kernel/config# mkdir -p rdma_cm/bnxt_re0

root@MI300X-01:/sys/kernel/config# mkdir -p bnxt_re/bnxt_re0

root@MI300X-01:/sys/kernel/config# ls rdma_cm

bnxt_re0

root@MI300X-01:/sys/kernel/config# ls bnxt_re
```

```
bnxt_re0

root@MI300X-01:/sys/kernel/config# mkdir -p rdma_cm/bnxt_re1

root@MI300X-01:/sys/kernel/config# mkdir -p bnxt_re/bnxt_re1

root@MI300X-01:/sys/kernel/config# ls rdma_cm

bnxt_re0 bnxt_re1

root@MI300X-01:/sys/kernel/config# ls bnxt_re

bnxt_re0 bnxt_re1
```

Repeat these steps for all the gpu interfaces.

NOTE: You must be a root user to make these changes.

```
jnpr@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc$ sudo echo -n 0x1 > ecn_enable

-bash: ecn_enable: Permission denied.

jnpr@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc$ sudo bash

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# sudo echo -n 0x1 > ecn_enable

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc#
```

The new directories will contain values pertaining to ECN, ROCE traffic and other functions:

```
root@MI300X-01:/sys/kernel/config# cd rdma_cm/bnxt_re0/ports/1

root@MI300X-01:/sys/kernel/config/rdma_cm/bnxt_re0/ports/1# ls

default_roce_mode  default_roce_tos

root@MI300X-01:/sys/kernel/config/rdma_cm/bnxt_re0/ports/1# cd /sys/kernel/config/bnxt_re/
bnxt_re0/ports/1

root@MI300X-02:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1$ ls
```

```
cc  tunables

root@MI300X-02:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1$ ls tunables

acc_tx_path     cq_coal_en_ring_idle_mode       dbr_pacing_algo_threshold

en_qp_dbg       snapdump_dbg_lvl        user_dbr_drop_recov_timeout

cq_coal_buf_maxtime     cq_coal_normal_maxbuf           dbr_pacing_enable

gsi_qp_mode     stats_query_sec cq_coal_during_maxbuf

dbr_def_do_pacing       dbr_pacing_time min_tx_depth

user_dbr_drop_recov

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/# ls cc

abs_max_quota   act_cr_factor   act_rel_cr_th   actual_cr_shift_correction_en

advanced        ai_rate_incr    ai_rtt_th1      ai_rtt_th2

apply   bw_avg_weight   cc_ack_bytes    cc_mode

cf_rtt_th       cnp_dscp        cnp_ecn cnp_prio

cnp_ratio_th    cp_bias cp_bias_en      cp_exp_update_th

cr_min_th       cr_prob_fac     cr_width        disable_prio_vlan_tx

ecn_enable      ecn_marking     exp_ai_rtts     exp_crcp_ratio

fair_cr_th      fr_num_rtts     g       inact_th

init_cp init_cr init_tr l64B_per_rtt

lbytes_per_usec max_cp_cr_th    max_quota       min_quota

min_time_bet_cnp        random_no_red_en        red_div red_rel_rtts_th

reduce_cf_rtt_th        reset_cc_cr_th  roce_dscp       roce_prio

rt_en   rtt     rtt_jitter_en   sc_cr_th1
```

```
sc_cr_th2       tr_lb   tr_prob_fac     tr_update_cyls

tr_update_mode
```

You can find a description of some of these parameters, as well as their current value using `cat apply` within the `/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc#` directory.

EXAMPLE:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat apply

ecn status (ecn_enable)                                 : Enabled

ecn marking (ecn_marking)                               : ECT(1)

congestion control mode (cc_mode)                       : DCQCN-P

send priority vlan (VLAN 0)                             : Disabled

running avg. weight(g)                                  : 8

inactivity threshold (inact_th)                         : 10000 usec

initial current rate (init_cr)                          : 0xc8

initial target rate (init_tr)                           : 0x320

cnp header ecn status (cnp_ecn)                         : ECT(1)

rtt jitter (rtt_jitter_en)                              : Enabled

link bytes per usec (lbytes_per_usec)                   : 0x7fff byte/usec

current rate width (cr_width)                           : 0xe bits

minimum quota period (min_quota)                        : 0x4

maximum quota period (max_quota)                        : 0x7

absolute maximum quota period(abs_max_quota)            : 0xff
```

```
64B transmitted in one rtt (l64B_per_rtt)              : 0xf460

roce prio (roce_prio)                                  : 3

roce dscp (roce_dscp)                                  : 26

cnp prio (cnp_prio)                                    : 7

cnp dscp (cnp_dscp)                                    : 48
```

2. Enable RoCEv2 operation.
   Even though RoCEv2 should be the default mode, the command to enable RoCEv2 is shown here.

   NOTE: This change is made under the rdma_cm directory

```
root@MI300X-01:/# cd /sys/kernel/config/rdma_cm/bnxt_re0/ports/1

root@MI300X-01:/sys/kernel/config/rdma_cm/bnxt_re0/ports/1# ls

default_roce_mode  default_roce_tos

root@MI300X-01:/sys/kernel/config/rdma_cm/bnxt_re0/ports/1# echo RoCE v2 > default_roce_mode
```

   NOTE: Enter the value exactly as shown including the space: “RoCE v2” (case sensitive).

   After setting the parameter, apply the new values as follows:

```
echo -n 0x1 > apply
```

   Verify the changes:

```
root@MI300X-01:/sys/kernel/config/rdma_cm/bnxt_re1/ports/1# cat default_roce_mode

RoCE v2
```

3. Enable ECN response and notification functions.

Even though ECN should be enabled by default, the command to enable ECN is shown here.

```
root@MI300X-01:/# cd /sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc
```

NOTE: This change is made under the bnxt_re0 directory.

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# echo -n 0x1 > ecn_enable
```

If needed, you can disable ECN by entering `echo -n 0x0 > ecn_enable` instead.

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# echo -n 0x1 > ecn_enable
```

When ECN is enabled on the Broadcom interfaces, they will respond to CNP packets (RP) and will generate CNP packets when ECN-marked are received (NP).

To disable it, enter `echo -n 0x0 > cnp_ecn` instead.

After setting the parameter, apply the new values:

```
echo -n 0x1 > apply
```

Verify the changes:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat ecn_enable

0x1
```

You can also enable the marking of both CNP and ROCE packets as ECN-eligible (meaning, these packets can be marked across the network when congestion occurs).

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_ecn 0x1
```

To summarize these attributes:

ecn_enable	Enables/Disables the RP (response point) side of ECN. It enables the device to respond to CNP packets. Default = 1 (enable)

cnp_ecn	Configures marking CNP packets as ECN-eligible. Either a value of 01 or 10 for ECT field.
ecn_marking	Configures marking ROCE packets as ECN-eligible. Either a value of 01 or 10 for ECT field.

1. Configure the DSCP and PRIO values for CNP and RoCEv2 packets.

   NOTE: Configuring these values manually, as shown below, is not an option for all types of Broadcom interface cards. For example, for BCM95741X devices you can use this method to configure the ECN, and RoCE priority values but on the BCM95750X/BCM97608 devices you can configure `roce_dscp`, `ecn_dscp` .

   See Broadcom Ethernet Network Adapter Congestion Control Parameters

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc#

echo -n 0x30 > cnp_dscp

# DSCP value as 48 (30 in HEX)
```

   NOTE: These changes are made under the bnxt_re0 directory.

```
echo -n 0x1a > roce_dscp

# DSCP value as 26 (1a in HEX)

echo -n 0x7 > cnp_prio

echo -n 0x3 > roce_prio
```

   NOTE: The following error indicates that changing the value of this parameter directly is not supported. In the case of BCM97608 roce_prio, and cnp_prio need to be configured using `bnxt_setupcc.sh` (described later).

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# echo -n 0x3 > roce_prio

bash: echo: write error: Invalid argument
```

After setting the parameter, apply the new values:

```
echo -n 0x1 > apply
```

Verify the changes:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_dscp

0x30

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_dscp

0x1a



root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_prio

0x7

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_prio

0x3
```

2. Configure the DCQCN algorithm (under the bnxt_re directory).

   The default DCQCN Congestion Control (cc-mode) algorithm in Broadcom Ethernet network adapter is **DCQCN-P**. The mode can be changed using these commands:

   NOTE: This change is made under the bnxt_re0 directory.

   To use DCQCN-P configure:

```
cd /sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc/

echo -n 1 > cc_mode

echo -n 1 > apply

cat apply
```

To use DCQCN-D configure:

```
root@MI300X-01:/

cd /sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc/

echo -n 0 > cc_mode

echo -n 1 > apply
```

3. Check all the attributes that were configured.

The following command shows all the interface parameters:

```
root@MI300X-01:/

cd /sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc/

echo -n 1 > advanced

echo -n 1 > apply

cat apply
```

For more information on the DCQCN algorithm in Broadcom Ethernet network adapter check the following documents: Changing Congestion Control Mode Settings and RoCE Congestion Control

EXAMPLE:

We have highlighted some ECN/CNP related parameters:

```
root@MI300X-01:/sys/kernel/config#

cd /sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc/

echo -n 1 > advanced

echo -n 1 > apply

cat apply
```

```
ecn status (cnp_ecn)                                    : Enabled
              ecn marking (ecn_marking)                                 : ECT(1)
              congestion control mode (cc_mode)                         : DCQCN-P

send priority vlan (VLAN 0)                             : Disabled

running avg. weight(g)                                  : 8

inactivity threshold (inact_th)                         : 10000 usec

initial current rate (init_cr)                          : 0xc8

initial target rate (init_tr)                           : 0x320

round trip time (rtt)                                   : 45 usec

cnp header ecn status (cnp_ecn)                         : ECT(1)

rtt jitter (rtt_jitter_en)                              : Enabled

link bytes per usec (lbytes_per_usec)                   : 0x7fff byte/usec

current rate width (cr_width)                           : 0xe bits

minimum quota period (min_quota)                        : 0x4

maximum quota period (max_quota)                        : 0x7

absolute maximum quota period(abs_max_quota)            : 0xff

64B transmitted in one rtt (l64B_per_rtt)               : 0xf460

minimum time between cnps (min_time_bet_cnp)            : 0x0 usec

initial congestion probability (init_cp)                : 0x3ff

target rate update mode (tr_update_mode)                : 1

target rate update cycle (tr_update_cyls)               : 0x0

fast recovery rtt (fr_num_rtts)                         : 0x5 rtts

active increase time quanta (ai_rate_incr)              : 0x1
```

```
reduc. relax rtt threshold (red_rel_rtts_th)           : 0x2 rtts

additional relax cr rtt (act_rel_cr_th)                : 0x50 rtts

minimum current rate threshold (cr_min_th)             : 0x0

bandwidth weight (bw_avg_weight)                       : 0x5

actual current rate factor (act_cr_factor)             : 0x0

current rate level to max cp (max_cp_cr_th)            : 0x3ff

cp bias state (cp_bias_en)                             : Disabled

log of cr fraction added to cp (cp_bias)               : 0x3

cr threshold to reset cc (reset_cc_cr_th)              : 0x32a

target rate lower bound (tr_lb)                        : 0x1

current rate probability factor (cr_prob_fac)          : 0x3

target rate probability factor (tr_prob_fac)           : 0x5

current rate fairness threshold (fair_cr_th)           : 0x64

reduction divider (red_div)                            : 0x1

rate reduction threshold (cnp_ratio_th)                : 0x0 cnps

extended no congestion rtts (exp_ai_rtts)              : 0x8 rtt

log of cp to cr ratio (exp_crcp_ratio)                 : 0x7

use lower rate table entries (rt_en)                   : Disabled

rtts to start cp track cr (cp_exp_update_th)           : 0x1a4 rtt

first threshold to rise ai (ai_rtt_th1)                : 0x40 rtt

second threshold to rise ai (ai_rtt_th2)               : 0x80 rtt
```

```
actual rate base reduction threshold (cf_rtt_th)        : 0x15e rtt

first severe cong. cr threshold (sc_cr_th1)             : 0x0

second severe cong. cr threshold (sc_cr_th2)            : 0x0

cc ack bytes (cc_ack_bytes)                             : 0x44

reduce to init rtts threshold(reduce_cf_rtt_th)         : 0x3eb rtt

random no reduction of cr (random_no_red_en)            : Enabled

actual cr shift correction (actual_cr_shift_correction_en)              : Enabled

roce prio (roce_prio)                                   : 3

roce dscp (roce_dscp)                                   : 26

cnp prio (cnp_prio)                                     : 7
              cnp dscp (cnp_dscp)                                    : 0
```

Configuring DCQN-ECN/PFC and TOS/DSCP for RDMA Traffic attributes using niccli

You can make changes to the DCQCN and traffic marking using the NICCLI Configuration Utility.

`niccli` is a management tool for Broadcom Ethernet network adapters that provides detailed information, including type, status, serial number, and firmware version. It also enables the configuration of interface attributes such as DCQCN-ECN, PFC, and TOS/DSCP for optimizing RDMA traffic.

NOTE: The niccli tools needs to be installed in your system.

Installing the NICCLI Configuration Utility

```
root@MI300X-01:/$ which niccli

/usr/bin/niccli

root@MI300X-01:/usr/bin$ ls niccli -l
```

```
lrwxrwxrwx 1 18896 1381 18 Sep 25 18:52 niccli -> /opt/niccli/niccli
```

You can obtain a summary of the interface adapters and ethernet ports that can be managed with niccli present on the server using `niccli listdev, or list-eth` as show in the example below.

```
root@MI300X-01:/home/jnpr# niccli --listdev

1 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#1 Port#1)

     Device Interface Name     : gpu0_eth

     MAC Address               : 7C:C2:55:BD:75:D0

     PCI Address               : 0000:06:00.0

2 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#2 Port#1)

     Device Interface Name     : gpu1_eth

     MAC Address               : 7C:C2:55:BD:79:20

     PCI Address               : 0000:23:00.0

3 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#3 Port#1)

     Device Interface Name     : gpu2_eth

     MAC Address               : 7C:C2:55:BD:7D:F0

     PCI Address               : 0000:43:00.0

4 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#4 Port#1)

     Device Interface Name     : gpu3_eth

     MAC Address               : 7C:C2:55:BD:7E:20

     PCI Address               : 0000:66:00.0
```

```
5 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#5 Port#1)

     Device Interface Name     : gpu4_eth

     MAC Address               : 7C:C2:55:BD:75:10

     PCI Address               : 0000:86:00.0

6 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#6 Port#1)

     Device Interface Name     : gpu5_eth

     MAC Address               : 7C:C2:55:BD:7D:C0

     PCI Address               : 0000:A3:00.0

7 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#7 Port#1)

     Device Interface Name     : gpu6_eth

     MAC Address               : 7C:C2:55:BD:84:90

     PCI Address               : 0000:C3:00.0

8 ) Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet Controller (Adp#8 Port#1)

     Device Interface Name     : gpu7_eth

     MAC Address               : 7C:C2:55:BD:83:10

     PCI Address               : 0000:E6:00.0

root@MI300X-01:/home/jnpr# niccli --list-eth

     BoardId      Interface       PCIAddr

  1) BCM57608     gpu0_eth        0000:06:00.0

  2) BCM57608     gpu1_eth        0000:23:00.0

  3) BCM57608     gpu2_eth        0000:43:00.0

  4) BCM57608     gpu3_eth        0000:66:00.0
```

```
  5) BCM57608     gpu4_eth        0000:86:00.0

  6) BCM57608     gpu5_eth        0000:A3:00.0

  7) BCM57608     gpu6_eth        0000:C3:00.0

  8) BCM57608     gpu7_eth        0000:E6:00.0
```

You can use `niccli` in either **oneline mode**, **interactive mode**, or **batch mode**. The `niccli -h help` provides a high level description of these modes. In this section, we will show some examples of how to use the **oneline** and **interactive modes** for DCQCN-ECN, PFC, and TOS/DSCP configuration.

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# niccli --help

----------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

----------------------------------------------------------------------------

NIC CLI - Help Option

--help / -h Displays the following help page.

Utility provides three modes of execution,

     1. Interactive Mode

         To launch in interactive mode :

             <NIC CLI executable> [-i <index of the target>] | -pci <NIC pci address>

         After launching in interactive mode, execute 'help' command to

         display the list of available commands.

     2. Oneline Mode

         To launch in Oneline mode :
```

```
            <NIC CLI executable> [-i <index of the target>] | -pci <NIC pci address> <command>

         To list available commands in Oneline mode :

            <NIC CLI executable> [-i <index of the target>] | -pci <NIC pci address> help

         Legacy Nic command syntax :

         To launch in Oneline mode :

            <NIC CLI executable> [-dev [<index of the target> | <mac addr> | <NIC pci
address>]] <command>

         To list available commands in Oneline mode :

            <NIC CLI executable> [-dev [<index of the target> | <mac addr> | <NIC pci
address>]] help

     3. Batch Mode

         To launch in batch mode :

            <NIC CLI executable> [-i <index of the target>] | -pci <NIC pci address> --batch
<batch file>

         NOTE: Batch mode requires flat text file with utility supported commands.

         Commands have to be provided in ascii format with the valid parameters.

         Supported commands can be listed using One-Line mode or Interactive mode

         Upon failure of any commands, utility will exit without continuing with other commands

List available targets for Oneline or Batch mode

     <NIC CLI executable> --list

     <NIC CLI executable> --listdev
```

Entering `niccli` with no options allows you to work in the interactive mode, where you select an adapter/interface (by index) and then the proper `<command>` (e.g. show, get_qos, set_map) to obtain information or make changes to the selected interface.

You can identify the interface index corresponding to each interface using the method described in the Mapping Broadcom interface name with logical interface name section. This will give you the mappings between interfaces and pcie address which you can then correlate with the output of `niccli` below.

Once identified, enter the interface index (first column in the output) as shown in the example below.

EXAMPLE:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# niccli

--------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

     BoardId     MAC Address        FwVersion    PCIAddr        Type   Mode

  1) BCM57608    7C:C2:55:BD:75:D0  230.2.49.0   0000:06:00.0   NIC    PCI

  2) BCM57608    7C:C2:55:BD:79:20  230.2.49.0   0000:23:00.0   NIC    PCI

  3) BCM57608    7C:C2:55:BD:7D:F0  230.2.49.0   0000:43:00.0   NIC    PCI

  4) BCM57608    7C:C2:55:BD:7E:20  230.2.49.0   0000:66:00.0   NIC    PCI

  5) BCM57608    7C:C2:55:BD:75:10  230.2.49.0   0000:86:00.0   NIC    PCI

  6) BCM57608    7C:C2:55:BD:7D:C0  230.2.49.0   0000:A3:00.0   NIC    PCI

  7) BCM57608    7C:C2:55:BD:84:90  230.2.49.0   0000:C3:00.0   NIC    PCI

  8) BCM57608    7C:C2:55:BD:83:10  230.2.49.0   0000:E6:00.0   NIC    PCI

Enter the target index to connect with : 1



BCM57608>


Once you are at the prompt for the selected NIC, you can enter commands such as show,
device_health_check, listdev, and listeth)
```

```
BCM57608> show

NIC State                          : Up

Device Type                        : THOR2

PCI Vendor ID                      : 0x14E4

PCI Device ID                      : 0x1760

PCI Revision ID                    : 0x11

PCI Subsys Vendor ID               : 0x15D9

PCI Subsys Device ID               : 0x1D42

Device Interface Name              : gpu0_eth

MAC Address                        : 7C:C2:55:BD:75:D0

Base MAC Address                   : 7C:C2:55:BD:75:D0

Serial Number                      : OA248S074777

Part Number                        : AOC-S400G-B1C

PCI Address                        : 0000:06:00.0

Chip Number                        : BCM57608

Chip Name                          : THOR2

Description                        : Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet
Controller

---more---



BCM57608> devid

Device Interface Name              : gpu0_eth
```

```
PCI Vendor ID                       : 0x14E4

PCI Device ID                       : 0x1760

PCI Revision ID                     : 0x11

PCI Subsys Vendor ID                : 0x15D9

PCI Subsys Device ID                : 0x1D42

PCI Address                         : 0000:06:00.0

BCM57608> device_health_check

Device Health Information :

   SBI Mismatch Check               : OK

   SBI Booted Check                 : OK

   SRT Mismatch Check               : OK

   SRT Booted Check                 : OK

   CRT Mismatch Check               : OK

   CRT Booted Check                 : OK

   Second RT Image                  : CRT Image

   Second RT Image Redundancy       : Good

   Image Fastbooted Check           : OK

   Directory Header Booted Check    : OK

   Directory Header Mismatch Check  : OK

   MBR Corrupt Check                : OK

   NVM Configuration                : OK
```

```
   FRU Configuration                : OK

   ------------------------------------------

   Overall Device Health            : Healthy

BCM57608> devid

Device Interface Name               : gpu0_eth

PCI Vendor ID                       : 0x14E4

PCI Device ID                       : 0x1760

PCI Revision ID                     : 0x11

PCI Subsys Vendor ID                : 0x15D9

PCI Subsys Device ID                : 0x1D42

PCI Address                         : 0000:06:00.0
```

Entering

```
niccli -i <interface-index>
<command>
```

allows you to issue the same commands but including the target interface and then the command, all in one line. The `niccli -list` command can be used to determine the interface index.

EXAMPLE

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# niccli --list

----------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

----------------------------------------------------------------------------

     BoardId     MAC Address        FwVersion    PCIAddr        Type   Mode
```

```
  1) BCM57608    7C:C2:55:BD:75:D0  230.2.49.0   0000:06:00.0   NIC    PCI

  2) BCM57608    7C:C2:55:BD:79:20  230.2.49.0   0000:23:00.0   NIC    PCI

  3) BCM57608    7C:C2:55:BD:7D:F0  230.2.49.0   0000:43:00.0   NIC    PCI

  4) BCM57608    7C:C2:55:BD:7E:20  230.2.49.0   0000:66:00.0   NIC    PCI

  5) BCM57608    7C:C2:55:BD:75:10  230.2.49.0   0000:86:00.0   NIC    PCI

  6) BCM57608    7C:C2:55:BD:7D:C0  230.2.49.0   0000:A3:00.0   NIC    PCI

  7) BCM57608    7C:C2:55:BD:84:90  230.2.49.0   0000:C3:00.0   NIC    PCI

  8) BCM57608    7C:C2:55:BD:83:10  230.2.49.0   0000:E6:00.0   NIC    PCI
```

The `sudo niccli help` provides an extensive list of commands and options available for both interactive and one-line mode.

```
root@MI300X-01:/home/jnpr# sudo niccli help

-------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

-------------------------------------------------------------------------------

Commands sets - Generic/Offline

-------------------------------------------------------------------------------

list                      - Lists all the compatible devices

listdev                   - Lists all the compatible devices (NIC legacy syntax)

devid                     - Query Broadcom device id's.

pkgver                    - Display FW PKG version installed on the device.

verify                    - Verify FW packages & NVM

nvm-list                  - Display NVM components and its associated versions.
```

```
nvmview                    - View NVM directories data

list-eth                   - Lists all NIC devices with ethernet interface names

help                       - Lists the available commands

quit                       - Quits from the application

Commands for platform 'BCM57xxx Performance NIC' and interface 'Direct PCIe'

------------------------------------------------------------------------------

show                       - Shows NIC specific device information

coredump                   - Retrieves coredump data from device.

snapdump                   - Retrieves snapdump data from device.

version                    - Display the current version of the application

txfir                      - Network Interface Card Transmission Finite

                           - Impulse Response

msixmv                     - Display and configure the number of MSIX max

                           - vectors values for VF's per each PF

scan                       - Scan PCI devices in the topology

pcie                       - Show/Execute pcie operation

nvm                        - NVRAM Option Management

pfalloc                    - Configure and Query for the number of PFs per PCIe

                           - endpoint

rfd                        - Restores NVM configuration to factory defaults

backuppowercfg             - Backup Power Configuration
```

```
tsio                    - TSIO function capability on the pin

ingressqos              - Query and configure the ingressqos parameters

egressqos               - Query and configure the egressqos parameters

dutycycle               - Set duty cycle on TSIO outgoing signal

dllsource               - Set the DLL source for PHC

vf                      - Configure and Query for a trusted VF

rxportrlmt              - Configure the receive side port rate limit

rxrlmt                  - Query the configured receive side rate control parameters

rxeprlmt                - Configure the receive side rate control parameters for a given
endpoint

txpartitionrlmt         - Query and Configure the transmit side partition rate limit applies
to traffic

                        - sent from a partition, which is one PF and all of its child VFs

txportrlmt              - Query and Configure the transmit side of port rate limit

txeprlmt                - Query and Configure the PCIe endpoint transmit rate control

vf                      - Configure and Query for a trusted VF

pfc                     - Configure the priority-based flow control for a given priority

apptlv                  - Configure the priority for the AppTLV

tcrlmt                  - Configure the rate limit for each traffic class

ets                     - Configure the enhanced transmission selection, priority to traffic
class and bandwidths

up2tc                   - Configure the user priorities to traffic classes

getqos                  - Query the configured enhanced transmission selection, priority to
traffic class and bandwidths
```

```
listmap                   - List the priority to traffic class and queueid mapping

dscp2prio                 - Query the dscp to priority mapping

reset                     - Reset the device

synce                     - Configure the synchronous ethernet profile

dscdump                   - Retrieves dscdump for device

ptp                       - PTP extended parameters operation

prbs_test                 - Run PRBS loopback test

serdes                    - Plots the serdes pci and ethernet eye and prints the horizontal and
vertical margin values



Legacy NVM commands :     - Query commands

---------------------     - ---------------

device_info               - Query Broadcom device information and default hardware

                          - resources profile version.

device_temperature        - Query the device temperature in Celsius.

get_backup_power_config   - Query backup power configuration of the device.

moduleinfo                - Query the PHY module information.

nvm_measurement           - Query the active NVM configuration.

get_ptp_extended          - Query the PTP extended parameters.

getoption                 - Query current NVM configuration option settings

                          - of a device.

pcie_counters             - Display the pcie counters.
```

```
saveoptions               - Save NVM configuration options on the device
                          - to a file.
get_sync_ethernet         - Get the synchronous ethernet frequency profile
get_txfir                 - Query the TX FIR settings.
cert_provision_state      - Query the imported certificate chain on the device.
read                      - Read the NVM item data and write its contents to a file.
mh_pf_alloc               - Query the number of PFs per PCIe endpoint.
                          - This command is supported only on Thor devices.
get_tsio_function_pin     - Query TSIO function capability on the pin.


Legacy NVM commands :     - Debug commands
---------------------     - ---------------
device_health_check       - Checks the device health.
backup                    - Backup NVM contents to a file


Legacy NVM commands :     - Configuration commands
---------------------     - ---------------
reset_ap                  - Reset management processor.
setoption                 - Configure NVM configuration option settings
                          -  of a device.
msix_max_vectors          - Configure the number of MSI-X max vectors per
```

```
                          - VF for each PF.

loopback                  - Query/perform loopback config.

add_ntuple_filter         - Add ntuple flow filter.

free_ntuple_filter        - Free ntuple flow filter.

cfgtunnel                 - query/config custom tunnel port/rss.

write                     - Create or overwrite NVM data item with a file.

set_txfir                 - Configures the TX FIR settings

set_ptp_extended          - Set PTP extended parameters

mh_pf_alloc               - Query/Configure the number of PFs per PCIe endpoint.

                          - This command is supported only on Thor devices.

restore_factory_defaults  - Restores NVM configuration to factory defaults

resmgmt                   - Query and Configure resources of the device.



Legacy NVM commands :     - FW update commands

---------------------     - ---------------

fw_sync                   - Synchronize primary & secondary FW images

livepatch                 - Query, Activate and Deactivate the patch in live

install                   - Install/Update FW



Legacy QoS Rx commands :  - Rx Qos commands

---------------------     - ---------------

rx_port_ratelimit         - The user can configure rx rate control that applies to all traffic
```

```
in a rx CoS queue group.

rx_endpoint_ratelimit     - The user can configure endpoint rx rate control that applies to all
traffic in a rx CoS queue group.

get_rx_ratelimits         - The user can query the rx rate limits.



Legacy QoS Tx commands :  - Tx Qos commands

----------------------    - ---------------

partition_tx_ratelimit    - This command is used to configure partition tx rate limit.

get_partition_tx_ratelimit - This command is used to query the partition rate limit
configuration for a given partition.

get_tx_port_ratelimit     - This command is used to query the tx side of port rate limit.

tx_port_ratelimit         - This command is used to configure the tx side of port rate limit

tx_endpoint_ratelimit     - This command is used to configure PCIe endpoint tx rate limit.

get_tx_endpoint_ratelimits - This command is used to query the tx endpoint rate limits.



Legacy DCB commands :     - Data Center Bridging commands

----------------------    - ---------------

set_pfc                   - This command is used to enable PFC on a given priority

set_apptlv                - This command is used to configure the priority of the AppTLV.

ratelimit                 - This command is used to configure the rate limit for each traffic
class.

set_ets                   - This command is used to configure the DCB parameters.

set_map                   - This command is used to configure the priority to traffic class.
```

```
get_qos                  - This command is used to query the DCB parameters.

dump                     - This command is used to dump the priority to cos mapping.

get_dscp2prio            - This command is used to query the dscp to priority mapping.
```

NOTE: We will use the one-line mode for all the examples below to obtain information and make configuration changes.

The following examples show you how to use `niccli` to obtain information about a specific interface.

1. Check interface status.

   The `niccli -i <interface> show` provides details about the interface such as type, MAC address, firmware, serial number, device health, temperature and so on.

   EXAMPLE:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# sudo niccli -i 1 show

-------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

-------------------------------------------------------------------------------

NIC State                          : Up

Device Type                        : THOR2

PCI Vendor ID                      : 0x14E4

PCI Device ID                      : 0x1760

PCI Revision ID                    : 0x11

PCI Subsys Vendor ID               : 0x15D9

PCI Subsys Device ID               : 0x1D42

Device Interface Name              : gpu0_eth

MAC Address                        : 7C:C2:55:BD:75:D0
```

```
Base MAC Address                    : 7C:C2:55:BD:75:D0

Serial Number                       : OA248S074777

Part Number                         : AOC-S400G-B1C

PCI Address                         : 0000:06:00.0

Chip Number                         : BCM57608

Chip Name                           : THOR2

Description                         : Supermicro PCIe 400Gb Single port QSFP56-DD Ethernet
Controller

Firmware Name                       : PRIMATE_FW

Firmware Version                    : 230.2.49.0

RoCE Firmware Version               : 230.2.49.0

HWRM Interface Spec                 : 1.10.3

Kong mailbox channel                : Not Applicable

Active Package Version              : 230.2.52.0

Package Version on NVM              : 230.2.52.0

Active NVM config version           : 0.0.5

NVM config version                  : 0.0.5

Reboot Required                     : No

Firmware Reset Counter              : 0

Error Recovery Counter              : 0

Crash Dump Timestamp                : Not Available

Secure Boot                         : Enabled
```

```
Secure Firmware Update              : Enabled

FW Image Status                     : Operational

Crash Dump Available in DDR         : No

Device Temperature                  : 57 Celsius

PHY Temperature                     : Not Available

Optical Module Temperature          : 65 Celsius

Device Health                       : Good
```

2. Check QoS settings

The `sudo niccli -i <interface-index> dscp2prio` and `sudo niccli -i 1 listmap -pri2cos` commands show mappings between DSCP and Priority vales, and between priority vales, traffic classes (TC) and the output queues.

```
root@MI300X-01:/home/jnpr# sudo niccli -i 1 dscp2prio

----------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

----------------------------------------------------------------------------

dscp2prio mapping:

          priority:7  dscp: 48

          priority:3  dscp: 26

root@MI300X-01:/home/jnpr# sudo niccli -i 2 listmap -pri2cos

----------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

----------------------------------------------------------------------------
```

```
Base Queue is 0 for port 0

----------------------------

Priority   TC   Queue ID

------------------------

 0         0    4

 1         0    4

 2         0    4

 3         1    0

 4         0    4

 5         0    4

 6         0    4

 7         2    5
```

The outputs in the example show the defaults for:

- Queues status. Only queues 0, 1, and 2 are enabled.
- Priority to DSCP mappings: priority 7 => DSCP 48 & priority 3 => DSCP 26.
- Priority to TC (traffic class) and queue mappings: priority 7 => TC2 (queue 0) => DSCP 48 & priority 3 => TC1 (queue 5) => DSCP 26.

NOTE: The output might be confusing, the Queue ID displayed is an internal CoS queue number. This really means queuing of traffic class 0, 1, and 2 are enabled, all other traffic classes are disabled.

The `sudo niccli -i <interface-index> get_qos` command provides a summary of the QoS configuration on the interface.

EXAMPLE:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# sudo niccli -i 1 get_qos

-------------------------------------------------------------------------------
```

```
NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

------------------------------------------------------------------------------

IEEE 8021QAZ ETS Configuration TLV:

         PRIO_MAP: 0:0 1:0 2:0 3:1 4:0 5:0 6:0 7:2

         TC Bandwidth: 50% 50% 0%

         TSA_MAP: 0:ets 1:ets 2:strict

IEEE 8021QAZ PFC TLV:

         PFC enabled: 3

IEEE 8021QAZ APP TLV:

         APP#0:

          Priority: 7

          Sel: 5

          DSCP: 48

         APP#1:

          Priority: 3

          Sel: 5

          DSCP: 26

         APP#2:

          Priority: 3

          Sel: 3

          UDP or DCCP: 4791
```

```
TC Rate Limit: 100% 100% 100% 0% 0% 0% 0% 0%
```

IEEE 802.1Qaz ETS Configuration TLV: shows the Enhanced Transmission Selection (ETS) configuration	
PRIO_MAP: 0:0 1:0 2:0 3:1 4:0 5:0 6:0 7:2	Maps priorities to Traffic Classes (TC) Priority 0, 1, 2, 4, 5, 6 → TC 0 Priority 3 → TC 1 Priority 7 → TC 2
TC Bandwidth: 50% 50% 0%	Allocates bandwidth percentages to traffic classes. TC 0: 50% of the total bandwidth. TC 1: 50%. TC 2: 0%.
TSA_MAP: 0:ets 1:ets 2:strict	Together with TC Bandwidth, TSA_MAP allocates resources and defines service priority for each TC. Equivalent to schedulers & scheduler-map in Junos. Specifies the Transmission Selection Algorithm (TSA) used for each TC: TC 0 and TC 1 use ETS (Enhanced Transmission Selection) and share the available bandwidth 50/50 TC 2 uses strict priority, meaning TC 2 traffic will always be sent first
IEEE 802.1Qaz PFC TLV: defines traffic classification using the APP TLV (Type-Length-Value) format	
PFC enabled: 3	Indicates that PFC is enabled on priority 3. Other priorities do not have PFC enabled. PFC ensures that traffic with this priority can pause instead of being dropped during congestion.
IEEE 802.1Qaz APP TLV	

(Continued)

IEEE 802.1Qaz ETS Configuration TLV: shows the Enhanced Transmission Selection (ETS) configuration	
APP#0: **Priority: 7** Sel: 5 **DSCP: 48** APP#1: **Priority: 3** Sel: 5 **DSCP: 26** APP#2: **Priority: 3** Sel: 3 **UDP or DCCP: 4791**	Maps traffic to Traffic Classes. Equivalent to multifield classifiers in Junos. APP#0: Traffic marked with DSCP = 48 is mapped to priority 7 APP#1: Traffic marked with DSCP = 48 is mapped to priority 3 APP#2: UDP or DCCP traffic with port = 4791 (RoCEv2) is mapped to priority 3
TC Rate Limit: 100% 100% 100% 0% 0% 0% 0% 0%	TC 0, TC 1, and TC 2 can use up to 100% of the bandwidth allocated to them. TC 3 through TC 7 are set to 0%, meaning they are not currently configured to transmit traffic.

If needed, change the priority to traffic class mappings or the applications to traffic class mappings.

We recommend keeping the default settings and making sure they are consistent with the class-of-service configuration on the leaf nodes in the GPU backend fabric.

```
[edit class-of-service classifiers]

jnpr@gpu-backend-rack1-001-leaf1# show

dscp mydscp {

    forwarding-class CNP {

        loss-priority low code-points 110000;  <= DSCP = 48
```

```
    }
    forwarding-class NO-LOSS {
        loss-priority low code-points 011010;  <= DSCP = 26
    }
 }
}
[edit class-of-service forwarding-classes]
jnpr@gpu-backend-rack1-001-leaf1# show
class CNP queue-num 3;
class NO-LOSS queue-num 4 no-loss pfc-priority 3;
```

If there are any requirements to change the priority to traffic class mappings or the applications to traffic class mappings the following commands can be used:

Priority to traffic class mappings

```
BCM57608> help up2tc
   DESCRIPTION :
     This command is used to set the user priorities to traffic classes.
   SYNTAX :
     up2tc      -p <priority[0-7]:tc>, ...>


-p: Comma separated list mapping user priorities to traffic classes.
```

EXAMPLE:

```
BCM57608> sudo niccli -i 1 get_qos

--------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

IEEE 8021QAZ ETS Configuration TLV:

PRIO_MAP: 0:1 1:1 2:0 3:0 4:1 5:1 6:0 7:0   <= default

---more---

     BCM57608> up2tc -p 0:0,1:0,2:1,3:1,4:1,5:1,6:1,7:0

User priority to traffic classes are configured successfully.

BCM57608> sudo niccli -i 1 get_qos

--------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

IEEE 8021QAZ ETS Configuration TLV:

PRIO_MAP: 0:0 1:0 2:1 3:1 4:1 5:1 6:1 7:0

---more---
```

Applications to traffic class mappings

```
BCM57608> help apptlv

   DESCRIPTION :
```

```
    This command is used to configure the priority of the AppTLV

  SYNTAX :

    apptlv -add -app <priority,selector,protocol>

    apptlv -del -app <priority,selector,protocol>
```

EXAMPLE:

```
BCM57608> sudo niccli -i 1 get_qos

---more---

IEEE 8021QAZ APP TLV:

        APP#1:

         Priority: 7

         Sel: 5

         DSCP: 48



        APP#2:

         Priority: 3

         Sel: 5

         DSCP: 26



        APP#3:

         Priority: 3

         Sel: 3
```

```
          UDP or DCCP: 4791

BCM57608> apptlv -add -app 5,1,35093

AppTLV configured successfully.

BCM57608> sudo niccli -i 1 get_qos

---more---

IEEE 8021QAZ APP TLV:

         APP#0:

          Priority: 5

          Sel: 1

          Ethertype: 0x8915



         APP#1:

          Priority: 7

          Sel: 5

          DSCP: 48



         APP#2:

          Priority: 3

          Sel: 5

          DSCP: 26
```

```
          APP#3:
           Priority: 3
           Sel: 3
           UDP or DCCP: 4791
BCM57608> BCM57608> apptlv -del -app 5,1,35093
AppTLV deleted successfully.
BCM57608> sudo niccli -i 1 get_qos
---more---
IEEE 8021QAZ APP TLV:
          APP#0:
           Priority: 7
           Sel: 5
           DSCP: 48

          APP#1:
           Priority: 3
           Sel: 5
           DSCP: 26

          APP#2:
           Priority: 3
           Sel: 3
```

```
          UDP or DCCP: 4791

---more---
```

If needed, change ETS configuration attributes

We recommend keeping the default settings and making sure they are consistent with the class-of-service configuration on the leaf nodes in the GPU backend fabric.

```
[edit class-of-service forwarding-classes]

jnpr@gpu-backend-rack1-001-leaf1# show

class CNP queue-num 3;

class NO-LOSS queue-num 4 no-loss pfc-priority 3;

BCM57608> help ets

   DESCRIPTION :

     This command is used to configure the enhanced transmission selection,

     priority to traffic class and traffic class bandwidths.

   SYNTAX :

     ets -tsa <tc[0-7]:[ets|strict], ...> -up2tc <priority[0-7]:tc>, ...> -tcbw <list>



        -tsa:   Transmission selection algorithm, sets a comma separated list of traffic classes
to

        the corresponding selection algorithm. Valid algorithms include "ets" and "strict".

         -up2tc:        Comma separated list mapping user priorities to traffic classes.

         -tcbw: Comma separated list of bandwidths for each traffic class the first value

        being assigned to traffic class 0 and the second to traffic class 1 and so on.
```

EXAMPLE:

```
BCM57608> sudo niccli -i 1 get_qos

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

IEEE 8021QAZ ETS Configuration TLV:

         PRIO_MAP: 0:1 1:1 2:0 3:0 4:1 5:1 6:0 7:0

         TC Bandwidth: 50% 50% 0%

         TSA_MAP: 0:ets 1:ets 2:strict

IEEE 8021QAZ PFC TLV:

         PFC enabled: 3

---more---

BCM57608> ets -tsa 0:ets,1:ets,2:ets -up2tc 0:0,1:0,2:0,3:0,4:0,5:1,6:0,7:0 -tcbw 50,25,25

Enhanced transmission selection (ets) configured successfully.

BCM57608> sudo niccli -i 1 get_qos

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

IEEE 8021QAZ ETS Configuration TLV:

         PRIO_MAP: 0:0 1:0 2:0 3:0 4:0 5:1 6:0 7:0

         TC Bandwidth: 50% 25% 25%

         TSA_MAP: 0:ets 1:ets 2:ets
```

If needed, configure PFC

```
BCM57608> help pfc

   DESCRIPTION :

     This command is used to enable priority-based flow control on a given priority.

   SYNTAX :

     pfc -enable <pfc list>

     The valid range is from 0 to 7. Where list is a comma-separated value for each pfc.

     To disable the pfc, user needs to provide a value of 0xFF.
```

EXAMPLE:

```
BCM57608> sudo niccli -i 1 get_qos

---more---

IEEE 8021QAZ PFC TLV:

         PFC enabled: 3         <= default; PFC enabled for priority 3

---more---

BCM57608> pfc -enable 0xFF   <= disables pfc on all priorities.

pfc configured successfully.

BCM57608> sudo niccli -i 1 get_qos

---more---

IEEE 8021QAZ PFC TLV:

         PFC enabled: none      <= pfc disabled on all priorities.

---more---
```

```
BCM57608> pfc -enable 5

pfc configured successfully.

BCM57608> sudo niccli -i 1 get_qos

---more---

IEEE 8021QAZ PFC TLV:

         PFC enabled: 5         <= PFC enabled for priority 5

---more---
```

The following command attempts to enable the pfc on priority 5 and 6 and demonstrates that only one queue (one priority) can be configured as a lossless queue (PFC-enabled).

```
BCM57608> pfc -enable 5,6

ERROR: Hardware doesn't support more than 1 lossless queues to configure pfc.

ERROR: Failed to enable pfc.
```

Non-volatile memory (NVM) options related to class of service.

The `sudo niccli -i <target> nvm -getoption` retrieves and displays the configurable Non-Volatile Memory (NVM) options for the specified NIC interface, including details like their names, current values, and configuration scopes.

```
root@MI300X-01:/# sudo niccli -i 1 nvm -getoption

-------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

-------------------------------------------------------------------------------
```

```
                NAME                              SCOPE

  ------------------------------------      ----------

  mac_address                               FUNCTION

  port_hide_capable                         DEVICE

  port_operation_capability                 DEVICE

  port_operation_mode                       DEVICE

  crypto_offload_selection                  DEVICE

  pf_pci_bar2_size                          PORT

  max_num_pf_msix_vectors                   DEVICE

  min_num_pf_msix_vectors                   DEVICE

  pcie_relaxed_ordering                     DEVICE

  multiroot_mode                            DEVICE

  pf_pps_rate                               DEVICE

  vf_pps_rate                               DEVICE

---more---
```

where the SCOPE column in the output of the niccli command indicates the level at which a particular option is applied or configured within the device hierarchy.

SCOPE	DESCRIPTION
FUNCTION	Applies to a specific physical function (PF) of the NIC, where a PF represents a full PCIe function with independent configuration (e.g. a single NIC is divided into multiple logical functions)
DEVICE	Applies to the entire physical device (the setting affects the whole NIC, regardless of how many ports, functions, or virtual resources are configured on it).

(Continued)

SCOPE	DESCRIPTION
PORT	Applies to an individual port on the NIC. Multi-port NICs may allow port-specific configurations that do not impact other ports.

NOTE: The scope in the command is only required for FUNCTION and PORT scope options.

`niccli nvm` also allows checking and adjusting CoS related hardware-level settings stored in the NIC's firmware.

```
root@MI300X-01:/# sudo niccli -i 1 nvm -getoption |egrep "roce|ecn|cnp|rdma"

  rdma_capable                                DEVICE

  roce_hw_responder_retx                      DEVICE

  roce_hw_requester_retx                      DEVICE

  disable_roce_hwrm_host_log                  DEVICE

  support_rdma                                FUNCTION

  disable_rdma_sriov                          FUNCTION

  enable_roce_dcn                             DEVICE

root@MI300X-01:/# sudo niccli -dev 1 getoption -name
rdma_capable

---------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

---------------------------------------------------------------------------

rdma_capable = Enabled

root@MI300X-01:/# sudo niccli -dev 1 getoption -name support_rdma -scope 0

---------------------------------------------------------------------------
```

```
NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

------------------------------------------------------------------------------

support_rdma = True
```

Although it is unlikely that you would need to change the values on these options, you can use

```
sudo niccli -dev <target>
setoption -name <option>
```

for that purpose.

EXAMPLE:

```
root@MI300X-01:/# sudo niccli -i 1 getoption -name cosq_mode -h

------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

------------------------------------------------------------------------------

Name                  : cosq_mode

Description           : Bitmask indicating which CoS queues (RX and TX) are lossy or lossless.

                        Each bit represents a specific queue where bit 0 represents

                        queue 0 and bit 7 represents queue 7.

                        A value of 0 indicates that the queue is lossy.

                        A value of 1 indicates that the queue is lossless.

                        If option cosq_mode_valid is set to 0, then this option is not honored.

Option Type           : Single Instance Type

Valid values          : 0 to 255
```

```
root@MI300X-01:/# sudo niccli -i 1 getoption -name cosq_mode

---------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

---------------------------------------------------------------------------

cosq_mode = 0x0 (0)

root@MI300X-01:/# sudo niccli -i 1 setoption -name cosq_mode -value 128

---------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

---------------------------------------------------------------------------

cosq_mode is set successfully

Please reboot the system to apply the configuration
```

Notice that a reboot is required for the change to be applied

```
root@MI300X-01:/# sudo niccli -i 1 getoption -name cosq_mode

---------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

---------------------------------------------------------------------------

cosq_mode = 0x80 (128)
```

Configuring DCQCN and RoCE traffic marking values using bnxt_setupcc.sh

Using the `bnxt_setupcc.sh` utility, which can simplify the process.

The bnxt_setupcc.sh utility simplifies enabling or disabling both ECN and PFC, and changing the values of DSCP and PRIO for both ROCE and CNP packets for a given interface.

Under the hood it uses niccli (default) or lldptool which can be selected as part of the command.

You need to enter bnxt_setupcc.sh followed by your selected options as described in the help menu:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# bnxt_setupcc.sh

Usage: bnxt_setupcc.sh [OPTION]...

  -d          RoCE Device Name (e.g. bnxt_re0, bnxt_re_bond0)

  -i          Ethernet Interface Name (e.g. p1p1 or for bond, specify slave interfaces like -i
p6p1 -i p6p2)

  -m [1-3]    1 - PFC only

              2 - CC only

              3 - PFC + CC mode

  -v          1 - Enable priority vlan

  -r [0-7]    RoCE Packet Priority

  -s VALUE    RoCE Packet DSCP Value

  -c [0-7]    RoCE CNP Packet Priority

  -p VALUE    RoCE CNP Packet DSCP Value

  -b VALUE    RoCE Bandwidth percentage for ETS configuration - Default is 50%

  -t [2]      Default mode (Only RoCE v2 is supported - Input Ignored)

  -C VALUE    Set CNP Service Type

  -u [1-3]    Utility to configure QoS settings

              1 - Use bnxtqos utility. Will disable lldptool if enabled. (default)

              2 - Use lldptool
```

```
              3 - Use Broadcom niccli utility. Will disable lldptool if enabled.

  -h          display help
```

EXAMPLE:

The default DSCP marking for CNP packets for interface gpu0 (bnxt_re0) is 0 as shown in the output below:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat apply | grep cnp

ecn status (cnp_ecn)                                : Enabled

cnp header ecn status (cnp_ecn)                     : ECT(1)

minimum time between cnps (min_time_bet_cnp)        : 0x0 usec

rate reduction threshold (cnp_ratio_th)             : 0x0 cnps

cnp prio (cnp_prio)                                 : 7

cnp dscp (cnp_dscp)                                 : 0

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat apply | grep cc

congestion control mode (cc_mode)                   : DCQCN-P

cr threshold to reset cc (reset_cc_cr_th)           : 0x32a

cc ack bytes (cc_ack_bytes)                         : 0x44

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_prio

0x7

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_dscp

0x0
```

`bnxt_setupcc.sh` can be used to change it to the value expected by the fabric (48) as follows:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc#

bnxt_setupcc.sh -d bnxt_re0 -i gpu0_eth -u 3 -p 48 -c 6 -s 26 -r 5 -m 3

ENABLE_PFC = 1 ENABLE_CC = 1

ENABLE_DSCP = 1 ENABLE_DSCP_BASED_PFC = 1

L2 50 RoCE 50

Using Ethernet interface gpu0_eth   and RoCE interface bnxt_re0

Setting pfc/ets 0000:06:00.0

---more---

AppTLV configured successfully.
```

Where:

- -u 3: Uses Broadcom niccli utility
- -p 48: Sets the DSCP value for CNP packets to 48 (0x30)
- -c: Configures the priority for CNP packets to 6
- -s: Defines the DSCP value for regular RoCE packets to 26 (0x1a)
- -r: Sets the priority for regular RoCE packets to 5
- -m 3: Configures both PFC and congestion control (ECN).

NOTE: Device (-i) is required for the script to complete. Also, you cannot configure only one of the DSCP/PRIO values. You need to configure CNP-DSCP value (-p) , CNP-PRI value (-c), RoCE-DSCP (-s), and RoCE-PRIO (-r) for the command to work.

Verify the results with:

```
root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat apply | grep cnp

ecn status (cnp_ecn)                                : Enabled
```

```
cnp header ecn status (cnp_ecn)                          : ECT(1)

minimum time between cnps (min_time_bet_cnp)             : 0x0 usec

rate reduction threshold (cnp_ratio_th)                  : 0x0 cnps

cnp prio (cnp_prio)                                      : 6

cnp dscp (cnp_dscp)                                      : 48

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat apply | grep roce

roce prio (roce_prio)                                    : 5

roce dscp (roce_dscp)                                    : 26

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_prio

0x6

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat cnp_dscp

0x30    <= 48 is HEX

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat roce_dscp

0x1a    <= 26 is HEX

root@MI300X-01:/sys/kernel/config/bnxt_re/bnxt_re0/ports/1/cc# cat roce_prio

0x5
```

NOTE: You need to make sure that not only `bnxt_setupcc.sh` is installed and executable, but also that at least one of the tools (`niccli` or `lldptool`) is installed.

The following example shows that `bnxt_setupcc.sh` and `niccli` are installed, but `lldptool` is not. It also shows examples of installing and using the `lldptool`.

```
root@MI300X-01:/# which bnxt_setupcc.sh

/usr/local/bin/bnxt_setupcc.sh
```

```
root@MI300X-01:/usr/local/bin# ls bnxt_setupcc.sh -l

-rwxr-xr-x 1 root root 14761 Jan 17 18:06 bnxt_setupcc.sh

root@MI300X-01:/$ which niccli

/usr/bin/niccli

root@MI300X-01:/usr/bin$ ls niccli -l

lrwxrwxrwx 1 18896 1381 18 Sep 25 18:52 niccli -> /opt/niccli/niccli

root@MI300X-01:/opt/niccli$ ls niccli -l

-rwxr-xr-x 1 18896 1381 609 Sep 25 18:52 niccli

root@MI300X-01:/$ which lldptool
```

The `lldptool` is used to check or modify the LLDP (Link Layer Discovery Protocol) settings. To enable LLDP you need to install `lldpad`, which also installs `lldptool` automatically.

To install `lldpad` and `lldptool` follow these steps:

1. Install required dependencies.
   Before installing lldpad, ensure that the necessary libraries are installed by running the following command:
   - `sudo apt install libconfig9 libnl-3-200`
   - libconfig9 – A configuration file processing library.
   - libnl-3-200 – A library for interacting with the Linux Netlink interface.

1. Install lldpad.
   Install lldpad by running the following command:
   - `sudo apt install lldpad`

   This package enables LLDP on the system, allowing it to exchange network topology information with other devices.

2. Enable lldpad.
   Enable lldp using systemctl:
   - `sudo systemctl enable lldpad`

This creates a systemd service that ensures lldpad is always running after a reboot.

3. Start the lldpad service
   Activate lldp using systemctl:

   - `sudo systemctl start lldpad`

   This activates lldpad immediately, allowing it to process LLDP packets.

   NOTE:

   To restart lldpad manually, use: `sudo systemctl restart lldpad`

   To disable lldpad from starting at boot, use: `sudo systemctl disable lldpad`

4. Verify the installation

Check the service status using systemctl

```
user@MI300X-01:/etc/apt$ sudo systemctl status lldpad

● lldpad.service - Link Layer Discovery Protocol Agent Daemon.

     Loaded: loaded (/usr/lib/systemd/system/lldpad.service; enabled; preset: enabled)

     Active: active (running) since Fri 2025-02-14 00:16:40 UTC; 2min 2s ago

TriggeredBy: ● lldpad.socket

       Docs: man:lldpad(8)

   Main PID: 695860 (lldpad)

      Tasks: 1 (limit: 629145)

     Memory: 1.3M (peak: 2.0M)

        CPU: 510ms

     CGroup: /system.slice/lldpad.service

             └─695860 /usr/sbin/lldpad -t
```

```
Feb 14 00:16:40 MI300X-01 systemd[1]: Started lldpad.service - Link Layer Discovery Protocol
Agent Daemon..
```

This ensures the tool is installed and ready to use. If everything is working properly, you should see an "active (running)" status.

You can use lldptool to enable or disable LLDP on an interface, and to check the LLDP status and the neighbors discovered on that interface. The lldptool -h shows you all the different options:

```
user@MI300X-01:/etc/apt$ lldptool -h

Usage:

  lldptool <command> [options] [arg]   general command line usage format

  lldptool                             go into interactive mode

           <command> [options] [arg]   general interactive command format

Options:

  -i [ifname]                          network interface

  -V [tlvid]                           TLV identifier

                                       may be numeric or keyword (see below)

  -c <argument list>                   used with get TLV command to specify

                                       that the list of configuration elements

                                       should be retrieved

  -d                                   use to delete specified argument from

                                       the configuration.  (Currently

                                       implemented for DCBX App TLV settings)

  -n                                   "neighbor" option for command
```

```
  -r                                  show raw message
  -R                                  show only raw messages
  -g                                   destination agent (may be one of):
                                               - nearestbridge (nb) (default)
                                               - nearestcustomerbridge (ncb)
                                               - nearestnontpmrbridge (nntpmrb)
Commands:
  license                             show license information
  -h|help                             show command usage information
  -v|version                          show version
  -p|ping                             ping lldpad and query pid of lldpad
  -q|quit                             exit lldptool (interactive mode)
  -S|stats                            get LLDP statistics for ifname
  -t|get-tlv                          get TLVs from ifname
  -T|set-tlv                          set arg for tlvid to value
  -l|get-lldp                         get the LLDP parameters for ifname
  -L|set-lldp                         set the LLDP parameter for ifname
TLV identifiers:
   chassisID    : Chassis ID TLV
   portID       : Port ID TLV
   TTL          : Time to Live TLV
```

```
    portDesc     : Port Description TLV

    sysName      : System Name TLV

    sysDesc      : System Description TLV

    sysCap       : System Capabilities TLV

    mngAddr      : Management Address TLV

    macPhyCfg    : MAC/PHY Configuration Status TLV

    powerMdi     : Power via MDI TLV

    linkAgg      : Link Aggregation TLV

    MTU          : Maximum Frame Size TLV

    LLDP-MED     : LLDP-MED Settings

    medCap       : LLDP-MED Capabilities TLV

    medPolicy    : LLDP-MED Network Policy TLV

    medLoc       : LLDP-MED Location TLV

    medPower     : LLDP-MED Extended Power-via-MDI TLV

    medHwRev     : LLDP-MED Hardware Revision TLV

    medFwRev     : LLDP-MED Firmware Revision TLV

    medSwRev     : LLDP-MED Software Revision TLV

    medSerNum    : LLDP-MED Serial Number TLV

    medManuf     : LLDP-MED Manufacturer Name TLV

    medModel     : LLDP-MED Model Name TLV

    medAssetID   : LLDP-MED Asset ID TLV

    CIN-DCBX     : CIN DCBX TLV
```

```
   CEE-DCBX     : CEE DCBX TLV
   evb          : EVB Configuration TLV
   evbcfg       : EVB draft 0.2 Configuration TLV
   vdp          : VDP draft 0.2 protocol configuration
   IEEE-DCBX    : IEEE-DCBX Settings
   ETS-CFG      : IEEE 8021QAZ ETS Configuration TLV
   ETS-REC      : IEEE 8021QAZ ETS Recommendation TLV
   PFC          : IEEE 8021QAZ PFC TLV
   APP          : IEEE 8021QAZ APP TLV
   PVID         : Port VLAN ID TLV
   PPVID        : Port and Protocol VLAN ID TLV
   vlanName     : VLAN Name TLV
   ProtoID      : Protocol Identity TLV
   vidUsage     : VID Usage Digest TLV
   mgmtVID      : Management VID TLV
   linkAggr     : Link Aggregation TLV
   uPoE         : Cisco 4-wire Power-via-MDI TLV
user@MI300X-01:/etc/apt$ sudo lldptool -S -i gpu0_eth
Total Frames Transmitted        = 0
Total Discarded Frames Received = 0
Total Error Frames Received     = 0
```

```
Total Frames Received           = 92

Total Discarded TLVs            = 0

Total Unrecognized TLVs         = 8

Total Ageouts                   = 0

user@MI300X-01:/etc/apt$ sudo lldptool -L -i gpu0_eth AMDinStatus=rxtx

AMDinStatus = rxtx

user@MI300X-01:/etc/apt$ sudo lldptool -S -i gpu0_eth

Total Frames Transmitted        = 5

Total Discarded Frames Received = 0

Total Error Frames Received     = 0

Total Frames Received           = 94

Total Discarded TLVs            = 0

Total Unrecognized TLVs         = 8

Total Ageouts                   = 0

user@MI300X-01:/etc/apt$ sudo lldptool -t -i gpu0_eth

Chassis ID TLV

        MAC: 7c:c2:55:bd:75:d0

Port ID TLV

        MAC: 7c:c2:55:bd:75:d0

Time to Live TLV

        120

IEEE 8021QAZ ETS Configuration TLV
```

```
        Willing: yes

        CBS: not supported

        MAX_TCS: 3

        PRIO_MAP: 0:0 1:0 2:0 3:0 4:0 5:0 6:0 7:0

        TC Bandwidth: 0% 0% 0% 0% 0% 0% 0% 0%

        TSA_MAP: 0:strict 1:strict 2:strict 3:strict 4:strict 5:strict 6:strict 7:strict
IEEE 8021QAZ PFC TLV

        Willing: yes

        MACsec Bypass Capable: no

        PFC capable traffic classes: 1

        PFC enabled: none

End of LLDPDU TLV
```

Check the Installing and Configuring Software Manually section of the Broadcom Ethernet Network Adapter User Guide or Installing the NICCLI Configuration Utility for more details.

Monitor interface and ECN/PFC operation:

Once you have the Broadcom name for a particular gpu as described at the beginning of this section, you can locate the directories where the interface's operation status, as well as RoCE traffic and Congestion Control statistics are located.

1. Navigate to the corresponding directory

/sys/class/infiniband/<Broadcom-interface-name>

EXAMPLE:

For gpu0_eth:

```
root@MI300X-01:/home/jnpr/SCRIPTS# cd /sys/class/infiniband/bnxt_re3

root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls

device  fw_ver  hca_type        hw_rev  node_desc

node_guid       node_type ports power subsystem sys_image_guid  uevent

root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls device/net/gpu3_eth/

addr_assign_type        address addr_len        broadcast       carrier

carrier_changes carrier_down_count      carrier_up_count        device  dev_id

dev_port        dormant duplex  flags   gro_flush_timeout

ifalias ifindex iflink  link_mode       mtu name_assign_type

napi_defer_hard_irqs    netdev_group    operstate       phys_port_id    phys_port_name

phys_switch_id  power   proto_down      queues  speed

statistics      subsystem       testing threaded        tx_queue_len

type    uevent
```

Here you can check attributes such as operational state, address, mtu, speed, and interface statistics (including transmit and received packets, dropped packets, as well as ECN-marked packets, CNP packets received and CNP packets transmitted):

```
root@MI300X-01:/sys/class/infiniband/bnxt_re3# cat device/net/gpu3_eth/operstate

up

root@MI300X-01:/sys/class/infiniband/bnxt_re3# cat device/net/gpu3_eth/address

7c:c2:55:bd:7e:20

root@MI300X-01:/sys/class/infiniband/bnxt_re3# cat device/net/gpu3_eth/mtu
```

```
9000

root@MI300X-01:/sys/class/infiniband/bnxt_re3# cat device/net/gpu3_eth/speed

400000

root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls device/net/gpu3_eth/statistics

collisions      multicast       rx_bytes        rx_compressed   rx_crc_errors

rx_dropped      rx_errors       rx_fifo_errors  rx_frame_errors rx_length_errors

rx_missed_errors        rx_nohandler    rx_over_errors  rx_packets      tx_aborted_errors

tx_bytes        tx_carrier_errors       tx_compressed   tx_dropped      tx_errors

tx_fifo_errors  tx_heartbeat_errors     tx_packets      tx_window_errors        tx_fifo_errors

rx_dropped      rx_frame_errors rx_nohandlertx_aborted_errors   tx_compressed   tx_window_errors

root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls ports/1

cap_mask        cm_rx_duplicates        cm_rx_msgs      cm_tx_msgs      cm_tx_retries

counters        gid_attrs       gids    hw_counters     lid

lid_mask_count  link_layer      phys_state      pkeys   rate

sm_lid  sm_sl   state



root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls ports/1/counters/ -m

excessive_buffer_overrun_errors link_downed     link_error_recovery

local_link_integrity_errors     port_rcv_constraint_errors      port_rcv_data

port_rcv_errors port_rcv_packets        port_rcv_remote_physical_errors

port_rcv_switch_relay_errors    port_xmit_constraint_errors     port_xmit_data
```

```
port_xmit_discards      port_xmit_packets       port_xmit_wait

symbol_error    VL15_dropped
```

To check ECN statistics, check the related counters for the specific interface:

```
root@MI300X-01:/sys/class/infiniband/bnxt_re3# ls ports/1/hw_counters/ -m

active_ahs      active_cqs      active_mrs      active_mws

active_pds      active_qps      active_rc_qps   active_srqs

active_ud_qps   bad_resp_err    db_fifo_register        dup_req

lifespan        local_protection_err    local_qp_op_err max_retry_exceeded

mem_mgmt_op_err missing_resp    oos_drop_count  pacing_alerts

pacing_complete pacing_reschedule       recoverable_errors      remote_access_err

remote_invalid_req_err  remote_op_err   res_cmp_err     res_cq_load_err

res_exceed_max  res_exceeds_wqe res_invalid_dup_rkey    res_irrq_oflow

resize_cq_cnt   res_length_mismatch     res_mem_err     res_opcode_err

res_rem_inv_err res_rx_domain_err       res_rx_invalid_rkey     res_rx_no_perm

res_rx_pci_err  res_rx_range_err        res_srq_err     res_srq_load_err

res_tx_domain_err       res_tx_invalid_rkey     res_tx_no_perm  res_tx_pci_err

res_tx_range_err        res_unaligned_atomic    res_unsup_opcode        res_wqe_format_err

rnr_naks_rcvd   rx_atomic_req   rx_bytes        rx_cnp_pkts

rx_ecn_marked_pkts      rx_good_bytes   rx_good_pkts    rx_out_of_buffer

rx_pkts rx_read_req     rx_read_resp    rx_roce_discards

rx_roce_errors  rx_roce_only_bytes      rx_roce_only_pkts       rx_send_req
```

```
rx_write_req    seq_err_naks_rcvd       to_retransmits  tx_atomic_req

tx_bytes        tx_cnp_pkts     tx_pkts tx_read_req

tx_read_resp    tx_roce_discards        tx_roce_errors  tx_roce_only_bytes

tx_roce_only_pkts       tx_send_req     tx_write_req    unrecoverable_err

watermark_ahs   watermark_cqs   watermark_mrs   watermark_mws

watermark_pds   watermark_qps   watermark_rc_qps        watermark_srqs

watermark_ud_qps

root@MI300X-01:/sys/class/infiniband#

for iface in /sys/class/infiniband/*/ports/1/hw_counters/rx_ecn_marked_pkts; do

    echo "$(basename $(dirname $(dirname $(dirname $(dirname "$iface"))))) : $(cat "$iface")"

done

bnxt_re0 : 0

bnxt_re1 : 1102

bnxt_re2 : 532

bnxt_re3 : 707

bnxt_re4 : 474

bnxt_re5 : 337

bnxt_re6 : 970

bnxt_re7 : 440

root@MI300X-01:/sys/class/infiniband#

for iface in /sys/class/infiniband/*/ports/1/hw_counters/tx_cnp_pkts; do

    echo "$(basename $(dirname $(dirname $(dirname $(dirname "$iface"))))) : $(cat "$iface")"
```

```
done

bnxt_re0 : 0

bnxt_re1 : 1102

bnxt_re2 : 532

bnxt_re3 : 707

bnxt_re4 : 474

bnxt_re5 : 337

bnxt_re6 : 970

bnxt_re7 : 440

root@MI300X-01:/sys/class/infiniband#

for iface in /sys/class/infiniband/*/ports/1/hw_counters/rx_cnp_pkts; do

    echo "$(basename $(dirname $(dirname $(dirname $(dirname "$iface"))))) : $(cat "$iface")"

done

bnxt_re0 : 0

bnxt_re1 : 830

bnxt_re2 : 0

bnxt_re3 : 375

bnxt_re4 : 734

bnxt_re5 : 23

bnxt_re6 : 2395

bnxt_re7 : 2291
```

To check PFC statistics use:

```
ethtool -s <InterfaceIndex> |egrep "pfc_frames|roce_pause"|more
```

EXAMPLE:

```
root@MI300X-01:/sys/class/infiniband# for iface in $(ls /sys/class/net/ | grep '^gpu'); do
    echo "$iface :"
    sudo ethtool -S "$iface" | egrep "pfc_frames|roce_pause"
done
gpu0_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 22598
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
gpu1_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 194626
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
gpu2_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 451620
     continuous_roce_pause_events: 0
```

```
     resume_roce_pause_events: 0
gpu3_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 492042
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
gpu4_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 407113
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
gpu5_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 290378
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
gpu6_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 228918
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
```

```
gpu7_eth :
     rx_pfc_frames: 0
     tx_pfc_frames: 477572
     continuous_roce_pause_events: 0
     resume_roce_pause_events: 0
root@MI300X-01:/sys/class/infiniband#
for iface in $(ls /sys/class/net/ | grep '^gpu'); do
    echo "$iface :"
    sudo ethtool -S "$iface" | grep cos | grep -v ": 0"
done
gpu0_eth :
     rx_bytes_cos0: 9529443988084
     rx_packets_cos0: 3319036491
     rx_bytes_cos4: 18230144638154
     rx_packets_cos4: 5955503873
     rx_discard_bytes_cos4: 3032625534
     rx_discard_packets_cos4: 736191
     tx_bytes_cos0: 27757371721830
     tx_packets_cos0: 9297694711
     tx_bytes_cos4: 604920
     tx_packets_cos4: 2628
gpu1_eth :
```

```
      rx_bytes_cos0: 27969554019118
      rx_packets_cos0: 9565740297
      rx_bytes_cos4: 4193860
      rx_packets_cos4: 47350
      tx_bytes_cos0: 27738638134736
      tx_packets_cos0: 9184463836
      tx_bytes_cos4: 619484
      tx_packets_cos4: 2686
      tx_bytes_cos5: 81548
      tx_packets_cos5: 1102
gpu2_eth :
      rx_bytes_cos0: 27961559203510
      rx_packets_cos0: 9438688373
      rx_bytes_cos4: 4134654
      rx_packets_cos4: 46526
      tx_bytes_cos0: 27177768852872
      tx_packets_cos0: 9028738664
      tx_bytes_cos4: 619444
      tx_packets_cos4: 2686
      tx_bytes_cos5: 39368
      tx_packets_cos5: 532
```

```
gpu3_eth :

      rx_bytes_cos0: 27886187894460

      rx_packets_cos0: 9394306658

      rx_bytes_cos4: 4161424

      rx_packets_cos4: 46910

      tx_bytes_cos0: 27963541263338

      tx_packets_cos0: 9314918707

      tx_bytes_cos4: 619624

      tx_packets_cos4: 2688

      tx_bytes_cos5: 52318

      tx_packets_cos5: 707

gpu4_eth :

      rx_bytes_cos0: 27760098268028

      rx_packets_cos0: 9493708902

      rx_bytes_cos4: 4190302

      rx_packets_cos4: 47275

      tx_bytes_cos0: 27943026331154

      tx_packets_cos0: 9175330615

      tx_bytes_cos4: 619068

      tx_packets_cos4: 2683

      tx_bytes_cos5: 35076

      tx_packets_cos5: 474
```

```
gpu5_eth :
     rx_bytes_cos0: 27742656661456
     rx_packets_cos0: 9603877462
     rx_bytes_cos4: 4136456
     rx_packets_cos4: 46558
     tx_bytes_cos0: 27862529155204
     tx_packets_cos0: 9053600792
     tx_bytes_cos4: 619318
     tx_packets_cos4: 2686
     tx_bytes_cos5: 24938
     tx_packets_cos5: 337
gpu6_eth :
     rx_bytes_cos0: 27204139187706
     rx_packets_cos0: 9417550449
     rx_bytes_cos4: 4309610
     rx_packets_cos4: 48912
     tx_bytes_cos0: 27939647032856
     tx_packets_cos0: 9122722262
     tx_bytes_cos4: 619248
     tx_packets_cos4: 2685
     tx_bytes_cos5: 71780
```

```
     tx_packets_cos5: 970

gpu7_eth :

     rx_bytes_cos0: 27985967658372

     rx_packets_cos0: 9636086344

     rx_bytes_cos4: 4303716

     rx_packets_cos4: 48823

     tx_bytes_cos0: 27949102839310

     tx_packets_cos0: 9149097911

     tx_bytes_cos4: 619138

     tx_packets_cos4: 2684

     tx_bytes_cos5: 32560

     tx_packets_cos5: 440

BCM57608> sudo niccli -i 2 listmap -pri2cos

--------------------------------------------------------------------------------

NIC CLI v231.2.63.0 - Broadcom Inc. (c) 2024 (Bld-94.52.34.117.16.0)

--------------------------------------------------------------------------------

Base Queue is 0 for port 0

----------------------------

Priority   TC   Queue ID

------------------------

 0         0    4

 1         0    4
```

```
2        0    4

3        1    0

4        0    4

5        0    4

6        0    4

7        2    5
```

Configuring the server to use the management interface for RCCL control traffic:

ROCm Communication Collectives Library (RCCL) creates TCP sessions to coordinate processes and exchange Queue Pair information for RoCE, GIDs (Global IDs), Local and remote buffer addresses, RDMA keys (RKEYs for memory access permissions)

NOTE: This traffic is separate from the RoCEv2 traffic (port 4791) and is used for synchronizing model parameters, partial results operations, etc.

These TCP sessions are created when the job starts and by default use one of the GPU interfaces (same interfaces used for RoCEv2 traffic).

Example:

```
jnpr@MI300X-01:~$ netstat -atn | grep 10.200 | grep "ESTABLISHED"

tcp        0      0 10.200.4.8:47932        10.200.4.2:43131        ESTABLISHED

tcp        0      0 10.200.4.8:46699        10.200.4.2:37236        ESTABLISHED

tcp        0      0 10.200.2.8:60502        10.200.13.2:35547       ESTABLISHED

tcp        0      0 10.200.4.8:37330        10.200.4.2:55355        ESTABLISHED

tcp        0      0 10.200.4.8:56438        10.200.4.2:53947        ESTABLISHED
```

```
---more---
```

It is recommended that the management interface connected to the (Frontend Fabric) is used. To achieve this, include the following when starting a job: **export NCCL_SOCKET_IFNAME="mgmt_eth".** The same environment variable applies to both NCCL and RCCL.

Example:

```
jnpr@MI300X-01:~$ netstat -atn | grep 10.10.1 | grep "ESTABLISHED"

tcp        0      0 10.10.1.0:44926         10.10.1.2:33149         ESTABLISHED

tcp        0      0 10.10.1.0:46705         10.10.1.0:40320         ESTABLISHED

tcp        0      0 10.10.1.0:54661         10.10.1.10:52452        ESTABLISHED

---more---
```

NOTE: ECN is enabled by default for these sessions; ***net.ipv4.tcp_ecn = 1*** , but can be disable with: ***sudo sysctl -w net.ipv4.tcp_ecn=0***

AMD Pollara DCQCN configuration for RDMA Traffic

For the AMD Pollara validation, DCQCN needs to be enabled and QOS has to be applied on the AMD NIC cards.

1. Configure QOS on the NICs using the script. The DSCP parameters are equivalent to the values suggested in Table 15. Server DCQCN configuration parameters.

```
jnpr@mi300-01:~$ cat /usr/local/bin/jnpr-setupqos.sh

#!/bin/bash

for i in $(sudo /usr/sbin/nicctl show port | grep Port | awk {'print $3'}); do sudo /usr/sbin/
nicctl update port -p $i --pause-type pfc --rx-pause enable --tx-pause enable; done

cts_dscp=48

cts_prio=2
```

```
data_dscp=26

data_prio=3

sudo nicctl update qos --classification-type dscp

sudo nicctl update qos dscp-to-priority --dscp $cts_dscp --priority $cts_prio

sudo nicctl update qos dscp-to-priority --dscp $data_dscp --priority $data_prio

sudo nicctl update qos pfc --priority $cts_prio --no-drop enable

sudo nicctl update qos pfc --priority $data_prio --no-drop enable

sudo nicctl update qos dscp-to-purpose --dscp $cts_dscp --purpose xccl-cts
```

2. Using AMD `nicctl` command line Utility below are the QOS parameters configured:

```
jnpr@mi300-01:~$ sudo nicctl show qos | more

NIC  : 42424650-4c32-3530-3130-313346000000 (0000:06:00.0)

Port : 0490812b-9860-4242-4242-000011010000

  Classification type         : DSCP

  DSCP-to-priority :

    DSCP bitmap               : 0xfffeffffbfffffff ==> priority : 0

    DSCP bitmap               : 0x0001000000000000 ==> priority : 2

    DSCP bitmap               : 0x0000000004000000 ==> priority : 3

    DSCP                      : 0-25, 27-47, 49-63 ==> priority : 0

    DSCP                      : 48 ==> priority : 2

    DSCP                      : 26 ==> priority : 3

  DSCP-to-purpose             : 48 ==> xccl-cts
```

```
  PFC :

    PFC priority bitmap       : 0xc

    PFC no-drop priorities    : 2,3

  Scheduling :

    -----------------------------------------

    Priority  Scheduling  Bandwidth Rate-limit

              Type        (in %age) (in Gbps)

    -----------------------------------------

    0         DWRR        0         N/A

    2         DWRR        0         N/A

    3         DWRR        0         N/A

NIC  : 42424650-4c32-3530-3130-313844000000 (0000:23:00.0)

Port : 0490812b-9fb0-4242-4242-000011010000

  Classification type         : DSCP

  DSCP-to-priority :

    DSCP bitmap               : 0xfffeffffbffffff ==> priority : 0

    DSCP bitmap               : 0x0001000000000000 ==> priority : 2

    DSCP bitmap               : 0x0000000004000000 ==> priority : 3

    DSCP                      : 0-25, 27-47, 49-63 ==> priority : 0

    DSCP                      : 48 ==> priority : 2

    DSCP                      : 26 ==> priority : 3
```

```
  DSCP-to-purpose             : 48 ==> xccl-cts

  PFC :

    PFC priority bitmap       : 0xc

    PFC no-drop priorities    : 2,3

--More--
```

3. The `rdma link` command can be used to check if the roce-devices association to the AMD Pollara NIC cards exist.

```
jnpr@mi300-01:~$ rdma link | grep gpu

link rocep9s0/1 state ACTIVE physical_state LINK_UP netdev gpu0_eth

link rocep38s0/1 state ACTIVE physical_state LINK_UP netdev gpu1_eth

link rocep70s0/1 state ACTIVE physical_state LINK_UP netdev gpu2_eth

link roceo1/1 state ACTIVE physical_state LINK_UP netdev gpu3_eth

link rocep137s0/1 state ACTIVE physical_state LINK_UP netdev gpu4_eth

link rocep166s0/1 state ACTIVE physical_state LINK_UP netdev gpu5_eth

link rocep198s0/1 state ACTIVE physical_state LINK_UP netdev gpu6_eth

link rocep233s0/1 state ACTIVE physical_state LINK_UP netdev gpu7_eth
```

The roce-devices are created when the ionic_rdma kernel module is loaded and should create the below roce-device file for each NIC card.

```
jnpr@mi300-01:/sys/class/infiniband$ find /sys/class/infiniband -type l

/sys/class/infiniband/rocep137s0

/sys/class/infiniband/rocep38s0

/sys/class/infiniband/rocep70s0
```

```
/sys/class/infiniband/roceo1

/sys/class/infiniband/rocep166s0

/sys/class/infiniband/rocep233s0

/sys/class/infiniband/rocep198s0

/sys/class/infiniband/rocep9s0
```

4. To configure DCQCN on the AMD Pollara NICs, run below script with appropriate parameters.

```
jnpr @mi300-01:~$ cat /usr/local/bin/jnpr-enable-dcqcn.sh

#!/bin/bash

TOKEN_BUCKET_SIZE=800000

AI_RATE=160

ALPHA_UPDATE_INTERVAL=1

ALPHA_UPDATE_G=512

INITIAL_ALPHA_VALUE=64

RATE_INCREASE_BYTE_COUNT=431068

HAI_RATE=300

RATE_REDUCE_MONITOR_PERIOD=1

RATE_INCREASE_THRESHOLD=1

RATE_INCREASE_INTERVAL=1

CNP_DSCP=48

ROCE_DEVICES=$(rdma link | grep gpu | awk '{ print $2 }' | awk -F/ '{ print $1 }' | paste -sd
" ")
```

```
for roce_dev in $ROCE_DEVICES

do

    sudo nicctl update dcqcn -r $roce_dev -i 1 \

    --token-bucket-size $TOKEN_BUCKET_SIZE \

    --ai-rate $AI_RATE \

    --alpha-update-interval $ALPHA_UPDATE_INTERVAL \

    --alpha-update-g $ALPHA_UPDATE_G \

    --initial-alpha-value $INITIAL_ALPHA_VALUE \

    --rate-increase-byte-count $RATE_INCREASE_BYTE_COUNT \

    --hai-rate $HAI_RATE \

    --rate-reduce-monitor-period $RATE_REDUCE_MONITOR_PERIOD \

    --rate-increase-threshold $RATE_INCREASE_THRESHOLD  \

    --rate-increase-interval $RATE_INCREASE_INTERVAL \

    --cnp-dscp $CNP_DSCP

Done
```

5. Using the nicctl command check the DCQCN profile for each roce-device.

```
jnpr@mi300-01:~$ sudo nicctl show dcqcn --roce-device rocep137s0 | more

ROCE device : rocep137s0

  DCQCN profile id                        : 7

  Status                                  : Disabled

  Rate reduce monitor period              : 100
```

```
Alpha update interval                   : 100

Clamp target rate                       : 0

Rate increase threshold                 : 1

Rate increase byte count                : 431068

Rate increase in AI phase               : 200

Alpha update G value                    : 50

Minimum rate                            : 1

Token bucket size                       : 4000000

Rate increase interval                  : 10

Rate increase in HAI phase              : 200

Initial alpha value                     : 64

DSCP value used for CNP                 : 48

DCQCN profile id                        : 5

Status                                  : Disabled

Rate reduce monitor period              : 100

Alpha update interval                   : 100

Clamp target rate                       : 0

Rate increase threshold                 : 1

Rate increase byte count                : 431068

Rate increase in AI phase               : 200

Alpha update G value                    : 50

Minimum rate                            : 1
```

```
  Token bucket size                       : 4000000
  Rate increase interval                  : 10
  Rate increase in HAI phase              : 200
  Initial alpha value                     : 64
  DSCP value used for CNP                 : 48
  DCQCN profile id                        : 3
  Status                                  : Disabled
  Rate reduce monitor period              : 100
  Alpha update interval                   : 100
  Clamp target rate                       : 0
  Rate increase threshold                 : 1
  Rate increase byte count                : 431068
  Rate increase in AI phase               : 200
  Alpha update G value                    : 50
  Minimum rate                            : 1
  Token bucket size                       : 4000000
  Rate increase interval                  : 10
  Rate increase in HAI phase              : 200
  Initial alpha value                     : 64
  DSCP value used for CNP                 : 48
--More—
```

6. Finally run the rccl_test.sh script as below. The example below shows the tests run for “All reduce”.

```
jnpr@mi300-01:/mnt/nfsshare/source/aicluster/rccl-tests$ ./run-rccl.sh

Running all_reduce, channels 64, qps 1  ...

Num nodes: 2

+ tee --append /mnt/nfsshare/logs/rccl/MI300-RAILS-ALL/06062025_18_03_35/test.log

+ /opt/ompi/bin/mpirun --np 16 --allow-run-as-root -H MI300-01:8,MI300-02:8 --bind-to numa -x
NCCL_IB_GID_INDEX=1 -x UCX_UNIFIED_MODE=y -x NCCL_IB_PCI_RELAXED_ORDERING=1 -x
NCCL_GDR_FLUSH_DISABLE=1 -x RCCL_GDR_FLUSH_GPU_MEM_NO_RELAXED_ORDERING=0 -x PATH=/opt/ompi/
bin:/opt/rocm/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/usr/local/games:/snap/bin -x LD_LIBRARY_PATH=/home/dbarmann/pollara/rccl-7961624/build/
release:/home/dbarmann/pollara/amd-anp-new/build:/opt/ompi/lib: -x
UCX_NET_DEVICES=gpu0_eth,gpu1_eth,gpu2_eth,gpu3_eth,gpu4_eth,gpu5_eth,gpu6_eth,gpu7_eth -x
NCCL_IB_HCA=rocep9s0,rocep38s0,rocep70s0,roceo1,rocep137s0,rocep166s0,rocep198s0,rocep233s0 --
mca btl '^vader,openib' --mca btl_tcp_if_include mgmt_eth -x NCCL_MIN_NCHANNELS=64 -x
NCCL_MAX_NCHANNELS=64 -x NCCL_IB_QPS_PER_CONNECTION=1 -x NCCL_TOPO_DUMP_FILE=/tmp/
system_run2.txt -x HSA_NO_SCRATCH_RECLAIM=1 -x NCCL_GDRCOPY_ENABLE=0 -x NCCL_IB_TC=106 -x
NCCL_IB_FIFO_TC=192 -x NCCL_IGNORE_CPU_AFFINITY=1 -x RCCL_LL128_FORCE_ENABLE=1 -x
NCCL_PXN_DISABLE=0 -x NCCL_DEBUG=INFO -x NET_OPTIONAL_RECV_COMPLETION=1 -x
NCCL_IB_USE_INLINE=1 -x NCCL_DEBUG_FILE=/mnt/nfsshare/logs/rccl/MI300-RAILS-ALL/
06062025_18_03_35/nccl-debug.log -x 'LD_PRELOAD=/home/dbarmann/pollara/amd-anp-new/build/
librccl-net.so /home/dbarmann/pollara/rccl-7961624/build/release/librccl.so' /mnt/nfsshare/
source/aicluster/rccl-tests/build/all_reduce_perf -b 1024 -e 16G -f 2 -g 1 -n 20 -m 1 -c 1 -w
5_# nThread 1 nGpus 1 minBytes 1024 maxBytes 17179869184 step: 2(factor) warmup iters: 5
iters: 20 agg iters: 1 validation: 1 graph: 0_#

rccl-tests: Version develop:b0a3841+

# Using devices

#  Rank  0 Group  0 Pid  18335 on   mi300-01 device  0 [0000:05:00] AMD Instinct MI300X

#  Rank  1 Group  0 Pid  18336 on   mi300-01 device  1 [0000:29:00] AMD Instinct MI300X

#  Rank  2 Group  0 Pid  18337 on   mi300-01 device  2 [0000:49:00] AMD Instinct MI300X

#  Rank  3 Group  0 Pid  18340 on   mi300-01 device  3 [0000:65:00] AMD Instinct MI300X
```

```
#  Rank  4 Group  0 Pid  18338 on   mi300-01 device  4 [0000:85:00] AMD Instinct MI300X

#  Rank  5 Group  0 Pid  18341 on   mi300-01 device  5 [0000:a9:00] AMD Instinct MI300X

#  Rank  6 Group  0 Pid  18342 on   mi300-01 device  6 [0000:c9:00] AMD Instinct MI300X

#  Rank  7 Group  0 Pid  18339 on   mi300-01 device  7 [0000:e5:00] AMD Instinct MI300X

#  Rank  8 Group  0 Pid  16249 on   mi300-02 device  0 [0000:05:00] AMD Instinct MI300X

#  Rank  9 Group  0 Pid  16251 on   mi300-02 device  1 [0000:29:00] AMD Instinct MI300X

#  Rank 10 Group  0 Pid  16250 on   mi300-02 device  2 [0000:49:00] AMD Instinct MI300X

#  Rank 11 Group  0 Pid  16254 on   mi300-02 device  3 [0000:65:00] AMD Instinct MI300X

#  Rank 12 Group  0 Pid  16255 on   mi300-02 device  4 [0000:85:00] AMD Instinct MI300X

#  Rank 13 Group  0 Pid  16253 on   mi300-02 device  5 [0000:a9:00] AMD Instinct MI300X

#  Rank 14 Group  0 Pid  16252 on   mi300-02 device  6 [0000:c9:00] AMD Instinct MI300X

#  Rank 15 Group  0 Pid  16256 on   mi300-02 device  7 [0000:e5:00] AMD Instinct MI300X

#

#                                                              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)

        1024           256     float     sum      -1    41.61    0.02    0.05      0
53.55    0.02    0.04      0

        2048           512     float     sum      -1    43.79    0.05    0.09      0
50.54    0.04    0.08      0

        4096          1024     float     sum      -1    45.75    0.09    0.17      0
45.21    0.09    0.17      0
```

```
        8192          2048     float     sum      -1    46.50    0.18    0.33      0
47.75    0.17    0.32      0

       16384          4096     float     sum      -1    60.52    0.27    0.51      0
48.90    0.34    0.63      0

       32768          8192     float     sum      -1    49.68    0.66    1.24      0
52.57    0.62    1.17      0

       65536         16384     float     sum      -1    53.75    1.22    2.29      0
52.74    1.24    2.33      0

      131072         32768     float     sum      -1    69.16    1.90    3.55      0
56.83    2.31    4.32      0

      262144         65536     float     sum      -1    69.31    3.78    7.09      0
63.17    4.15    7.78      0

      524288        131072     float     sum      -1    77.16    6.79   12.74      0
80.51    6.51   12.21      0

     1048576        262144     float     sum      -1    127.5    8.23   15.42      0
107.6    9.75   18.28      0

     2097152        524288     float     sum      -1    125.0   16.78   31.46      0
130.9   16.02   30.04      0

     4194304       1048576     float     sum      -1    149.5   28.06   52.61      0
148.4   28.26   52.99      0

     8388608       2097152     float     sum      -1    222.9   37.63   70.55      0
231.6   36.21   67.90      0

    16777216       4194304     float     sum      -1    321.3   52.21   97.90      0
326.2   51.43   96.43      0

    33554432       8388608     float     sum      -1    436.2   76.93  144.25      0
447.0   75.06  140.75      0

    67108864      16777216     float     sum      -1    678.9   98.85  185.35      0
684.6   98.02  183.79      0
```

```
   134217728      33554432     float     sum      -1   1164.6  115.25  216.10      0
1148.1  116.90  219.19      0

   268435456      67108864     float     sum      -1   1550.3  173.15  324.66      0
1563.9  171.65  321.84      0

   536870912     134217728     float     sum      -1   2979.9  180.16  337.81      0
2977.6  180.30  338.07      0

  1073741824     268435456     float     sum      -1   5824.8  184.34  345.64      0
5859.5  183.25  343.59      0

  2147483648     536870912     float     sum      -1    11596  185.20  347.25      0
11611  184.94  346.77      0

  4294967296    1073741824     float     sum      -1   520420    8.25   15.47      0
23190  185.21  347.27      0

  8589934592    2147483648     float     sum      -1    46157  186.10  348.94      0
46150  186.13  349.00      0

 17179869184    4294967296     float     sum      -1   568668   30.21   56.65      0
91823  187.10  350.81      0

# Errors with asterisks indicate errors that have exceeded the maximum threshold.

# Out of bounds values : 0 OK

# Avg bus bandwidth    : 117.077

#
```

VAST Storage Configuration

IN THIS SECTION

The VAST Data Platform is a modern, all-flash storage solution designed to support AI, machine learning (ML), and high-performance computing (HPC) workloads at any scale. By leveraging a disaggregated and shared everything (DASE) architecture, VAST eliminates traditional storage bottlenecks, delivering high throughput, low latency, and simplified data management.

By integrating VAST Data into the AI JVD design, we ensure that AI workloads have the performance, scalability, and resilience necessary to drive next-generation innovation. We selected the VAST Data Platform as part of the AI JVD design due to the following advantages:

- **High Performance:** Vast Data's architecture leverages all-flash storage for unparalleled data access speeds, making it ideal for AI/ML workloads, real-time analytics, and high-performance computing (HPC) environments.
- **Scalability:** With a disaggregated, shared-nothing architecture, Vast Data scales seamlessly from terabytes to exabytes without sacrificing performance. This design overcomes the limitations of traditional storage solutions and allows enterprises to grow efficiently.
- **Unified Storage:** Vast Data provides a single platform supporting multiple storage protocols, including NFS, S3, and SMB. Its ability to support both file and object storage offers flexibility in managing diverse data sets.

- **Data Resilience:** With advanced erasure coding and fault-tolerant architecture, Vast Data ensures data protection and high availability, even in the event of hardware failures.
- **Ease of Management:** The platform features a streamlined management experience, offering a single global namespace and real-time system monitoring for simplified operations and scalability.
- **Support for GPUs:** Vast Data's architecture is optimized for GPU-driven workloads, making it a perfect fit for environments focused on AI, machine learning, and other compute-intensive applications.
- **Low Latency:** The flash-optimized design provides ultra-low-latency data access, essential for real-time data processing and analytics tasks.

The Vast Data Platform empowers enterprises to unlock the full potential of AI-driven insights while maintaining a highly resilient, scalable, and manageable data storage environment.

VAST storage cluster components in the AI JVD lab

In VAST Data's architecture, a Quad Server Chassis or CBox (Compute Box) is a 2U high-density chassis that houses four dual-Xeon servers, each running VAST Server Containers. These servers, known as C-nodes, handle the compute services within a VAST cluster. The CBox works in conjunction with the DBox (Data Box), which provides the storage component of the system. DBoxes and CBoxes serve distinct roles:

- **DBox**: Provides storage capacity, housing NVMe SSDs that store data.
- **CBox**: Offers computational resources, running VAST's software services to manage and access the stored data. It delivers the necessary processing power to manage and access the stored data efficiently

Management of both DBoxes and CBoxes is centralized through VAST's management system, which provides a unified interface for configuring, monitoring, and maintaining the entire storage cluster.

The tested VAST solution for this JVD comprises a 7 Rack Unit that includes:

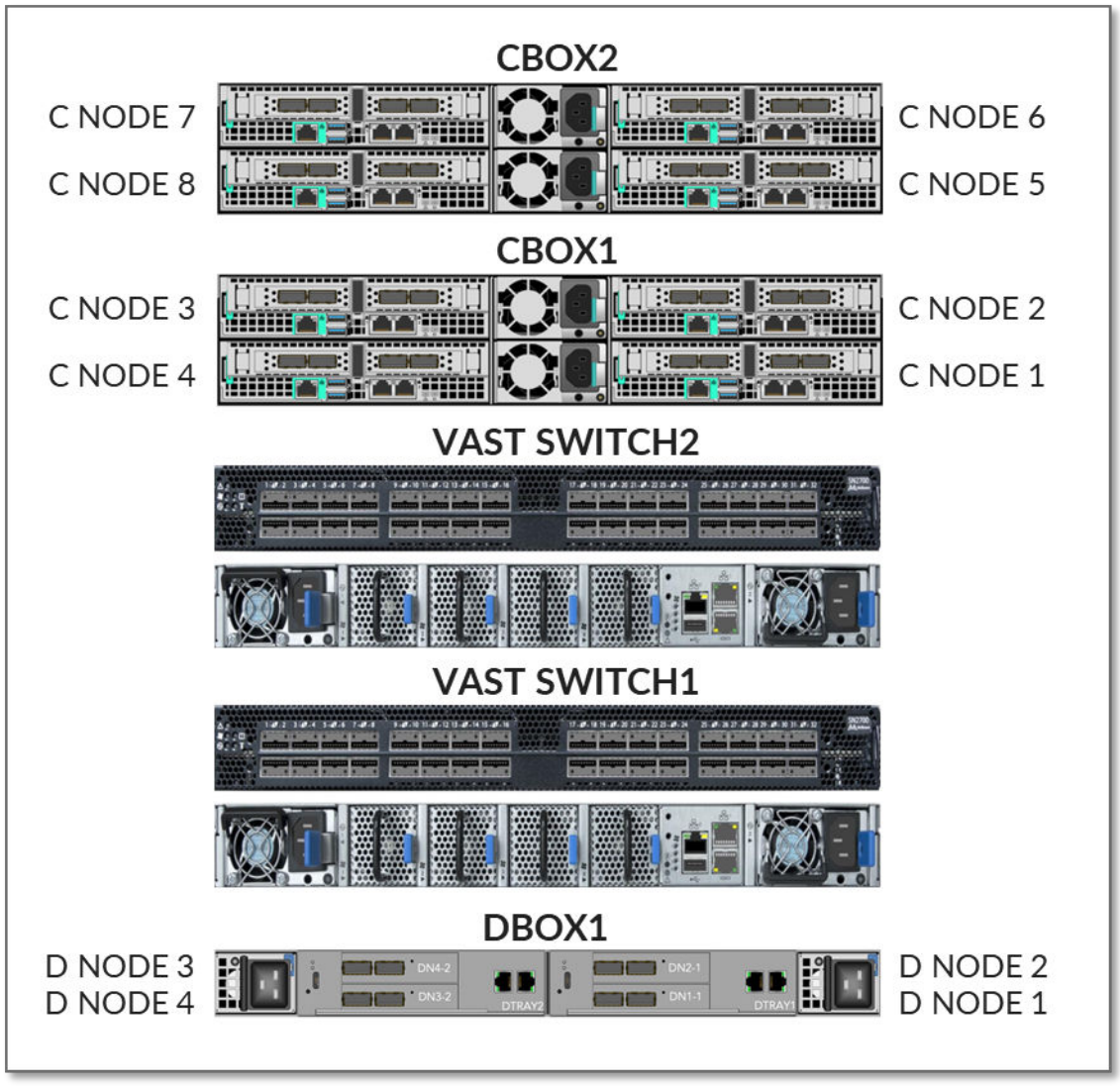
- 2 x CBOX (**QUAD-4N-IL-2NIC**) – 4 Slot Modular Chassis
  - Housing 8 C-nodes (4 C-nodes per C BOX)
  - Each C-node with
    - 2 x 100GE NICs &
    - 1 x Ice Lake Gen4
- 1 x Ceres DBOX (DF-3015) - 4 Slot Modular Chassis

- Houses 4 D-nodes
- Each D-node comes with 2 x 100GE NICs
- Houses 22 NVMe SSDs (15.363 TB each)

Total of 338TB of raw storage and 240TB of usable storage.

- 2 x Internal Mellanox switches (SN2100) for the VAST NVMe fabric

Figure 56: 2 x 1 VAST solution



VAST storage cluster components connections and management

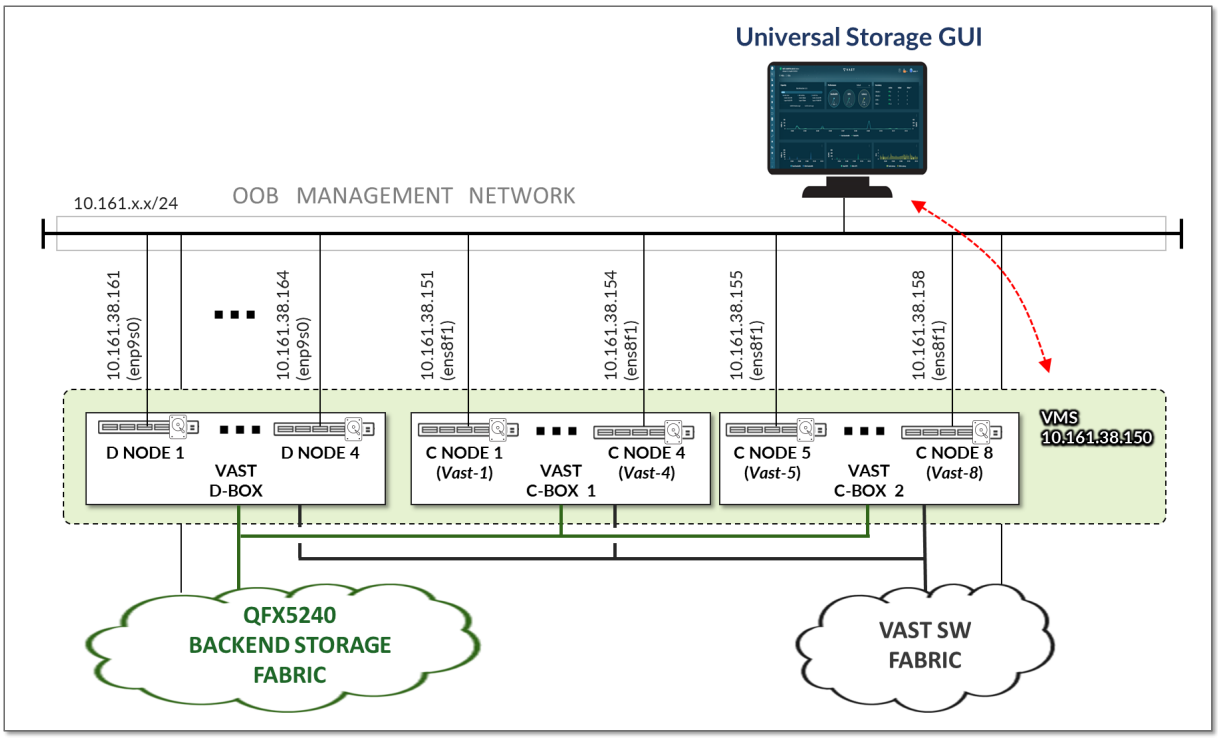
The VAST Storage cluster is managed by the Vast Control and Orchestration Layer (VMS), which oversees C-Nodes, D-Nodes, networking, security, monitoring, and overall system operations.

While it is possible to interact with VMS via CLI, VAST offers a powerful web-based UI (VAST Universal Storage GUI) that provides an intuitive way to configure, manage, and maintain the cluster—a key differentiator from traditional storage systems that are often CLI-driven.

All VMS management functions (from creating NFS exports to expanding the cluster) are exposed via a RESTful API. To simplify integration and usage, VMS publishes its API via Swagger, allowing for interactive documentation and code generation across different programming languages. Instead of directly controlling the system, both the GUI and CLI act as frontends consuming the RESTful API.

VMS runs on the C-Nodes, and the Universal Storage GUI is accessible from any web browser via a floating management IP assigned to the cluster as shown in Figure 57

Figure 57: VAST Universal Storage GUI connection



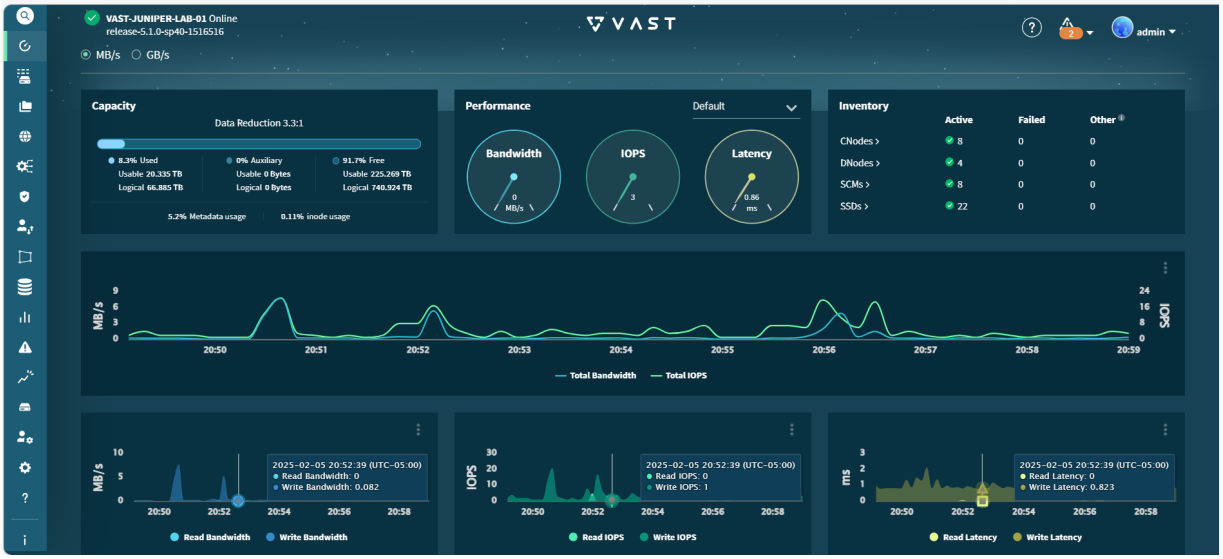
As also shown in Figure 57, all the cluster components are connected to the OOB management network, and the VMS IP address is assigned out of the same IP address range.

For more details about the Vast storage solution, you can review this whitepaper from Vast: The VAST Data Platform

VAST Universal Storage GUI

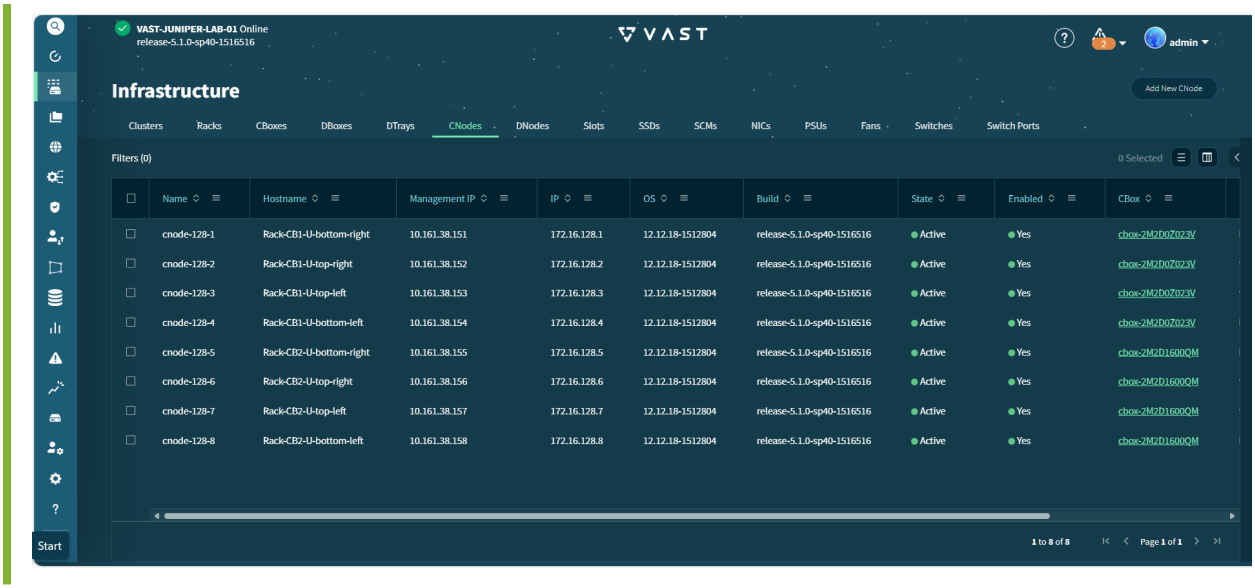
The dashboard of the GUI displays capacity, physical and logical usage, overall performance, read/write bandwidth, IOPS, and overall latency in a user-friendly graphical format as shown in Figure 58

Figure 58: VAST Universal Storage GUI Dashboard



Under the infrastructure tab, you can check the configuration and status of all the cluster components including cboxes and C-nodes, dboxes and D-nodes, SSDs, NICs, and so on.

Figure 59: VAST Universal Storage GUI Infrastructure details

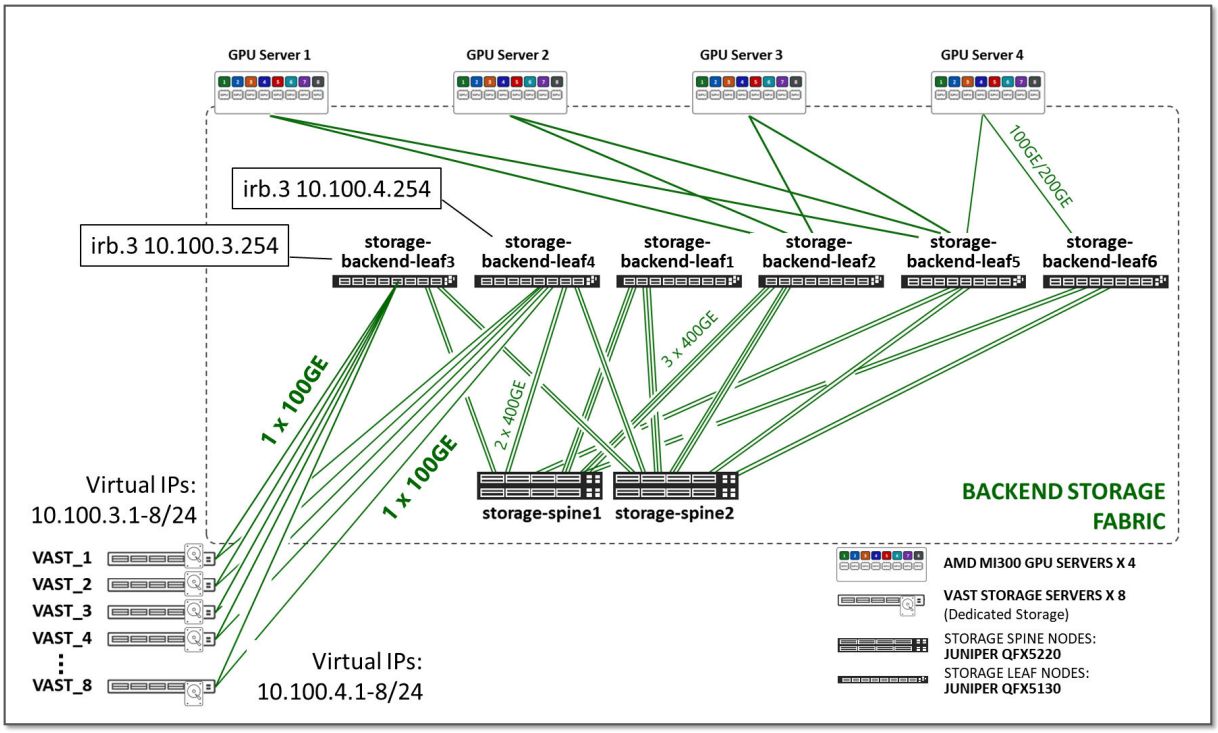


For more details about using the GUI check VAST Cluster 5.1 Administrator's Guide

Network Configuration for the Juniper Vast Storage Cluster

As described in the Storage Backend sections, the Vast servers are dual-homed, and connected to separate storage backend switches (*storage-leaf* 5 and *storage-leaf* 6) using 100GE ports

Figure 60: Vast to Storage fabric Connectivity



The storage leaf nodes are configured with IRB interfaces (irb.3 and irb.4), which have IP addresses 10.100.3.254 and 10.100.4.254, respectively.

The C-Nodes dynamically receive IP addresses from the Virtual IP (VIP) pool and take responsibility for responding to ARP requests for their assigned VIPs.

The system automatically rebalances VIPs and load distribution whenever a C-Node fails, is removed, or a new node is added.

- If a C-Node goes offline due to failure or intentional removal, the VIPs it was serving are redistributed across the remaining C-Nodes
- If a new C-Node is added, some VIPs are reassigned to integrate the node into the load-balancing mechanism.

To ensure optimal load balancing and fault tolerance, VAST recommends configuring each VIP pool with 2 to 4 times as many VIPs as there are C-Nodes. This ensures that in the event of a node failure, its load is distributed across multiple remaining C-Nodes rather than overburdening just one.

VAST NFS:

The Lab Servers use VAST's proprietary kernel module for higher performance. The kernel module allows for a 70% to 100% performance improvement.

fstab settings on the AMD GPU server:

10.100.3.1:/default /mnt/vast nfs nconnect=8,remoteports=10.100.3.1-10.100.3.8,rw,noatime,rsize=32768,wsize=32768,nolock,tcp,intr,fsc,nofail 0 0

Documentation about the Kernal Module and setup can be found here: https://vastnfs.vastdata.com/docs/4.0/Intro.html

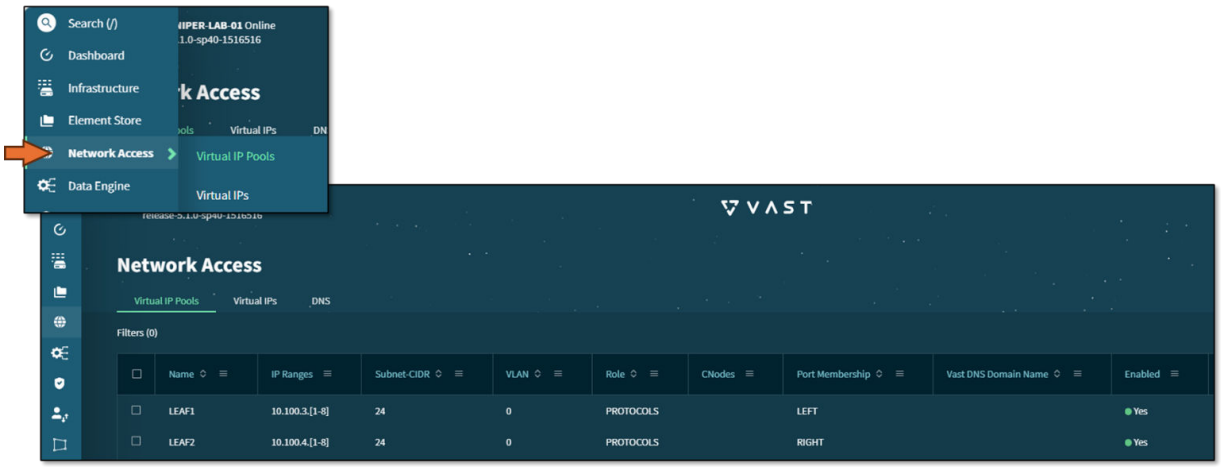
Common Setting and recommendations

When you order your VAST system, you will be asked to complete an initial site survey with IP addresses, and other details so your system can be configured accordingly, by Vast support. The Vast GUI will be installed and ready.

We recommend the following:

- Set the MTU to 9000
- VAST NFS Kernel on servers
- Enable multipathing this server fstab settings
- Configure the VIP pools with 2 to 4 times as many VIPs as there are C-Nodes

Figure 61: Storage Interface Connectivity



Network Connectivity Details (Reference Examples)

IN THIS SECTION

This section provides detailed examples for each fabric, as reference. It describes the IP connectivity across the common Frontend, and Storage Backend fabrics, and the GPU Backend fabric in Cluster 2.

Regardless of whether you are using Apstra with or without Terraform automation, the IP address Pools, ASN Pools, and interface addresses are automatically assigned and configured with little interaction from the administrator unless desired.

Notice that all the addresses shown in this section represent the IP addressing schema used in the Juniper lab to validate the design.

Frontend Network Connectivity

The Frontend fabric consists of two spine nodes (QFX5130-32CD) and two leaf nodes (QFX5130-32CD), where one leaf node is used to connect the GPU servers and Headend servers (*frontend-leaf1), and one is used to connect the Storage devices* (*frontend-leaf2).*

NOTE: Vast storage systems are NOT connected to the Frontend Fabric. Because this JVD is focused on Vast only, we will not cover the storage systems connectivity to the frontend.

IP addressing

The Frontend fabric is designed as a Layer 3 IP Fabric, with two 400GE links between each *frontend-leaf#* node and each *frontend-spine#* node as shown in Figure 62. These links are configured with /31 IP addresses, as shown in Table 16.

Figure 62: Frontend Spine to Leaf Nodes Connectivity

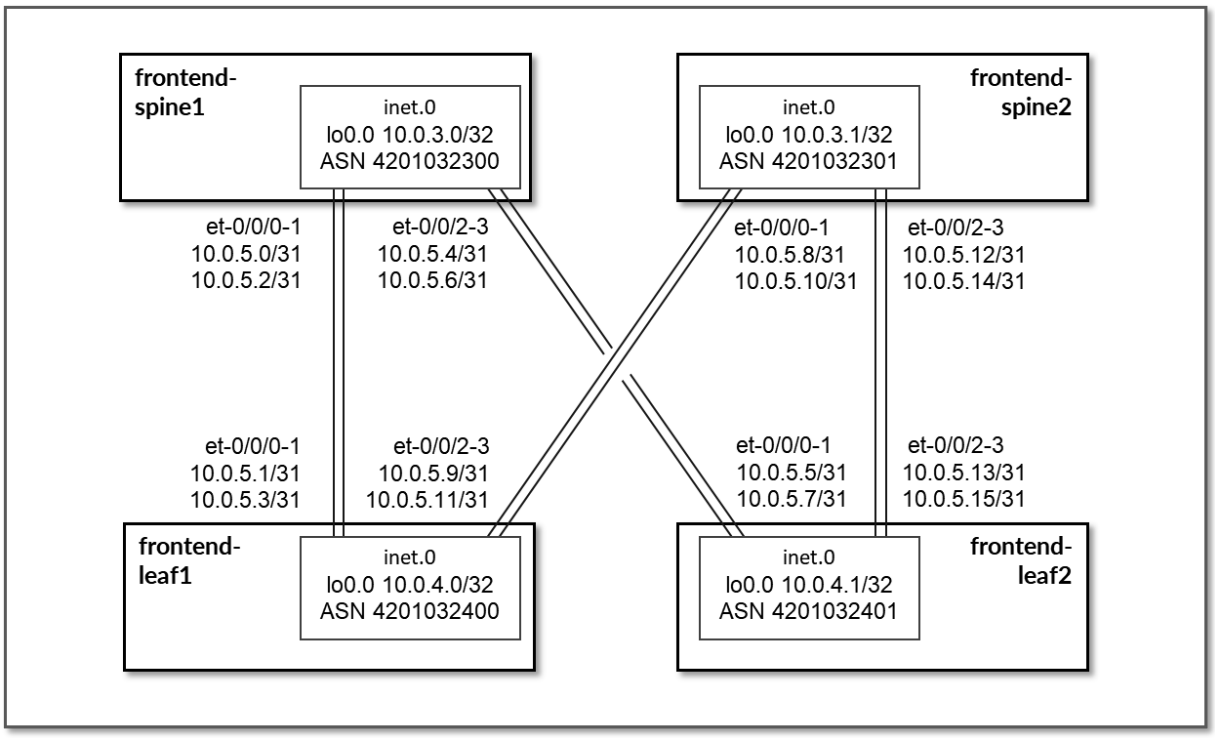


Table 16: Frontend Interface Addresses

Spine node	Leaf node	Spine IP address	Leaf IP address
frontend-spine1	*frontend-leaf1*	10.0.5.0/31 10.0.5.2/31	10.0.5.1/31 10.0.5.3/31
frontend-spine1	*frontend-leaf2*	10.0.5.4/31 10.0.5.6/31	10.0.5.5/31 10.0.5.7/31

(Continued)

Spine node	Leaf node	Spine IP address	Leaf IP address
frontend-spine2	*frontend-leaf1*	10.0.5.8/31 10.0.5.10/31	10.0.5.9/31 10.0.5.11/31
frontend-spine2	*frontend-leaf2*	10.0.5.12/31 10.0.5.14/31	10.0.5.13/31 10.0.5.15/31

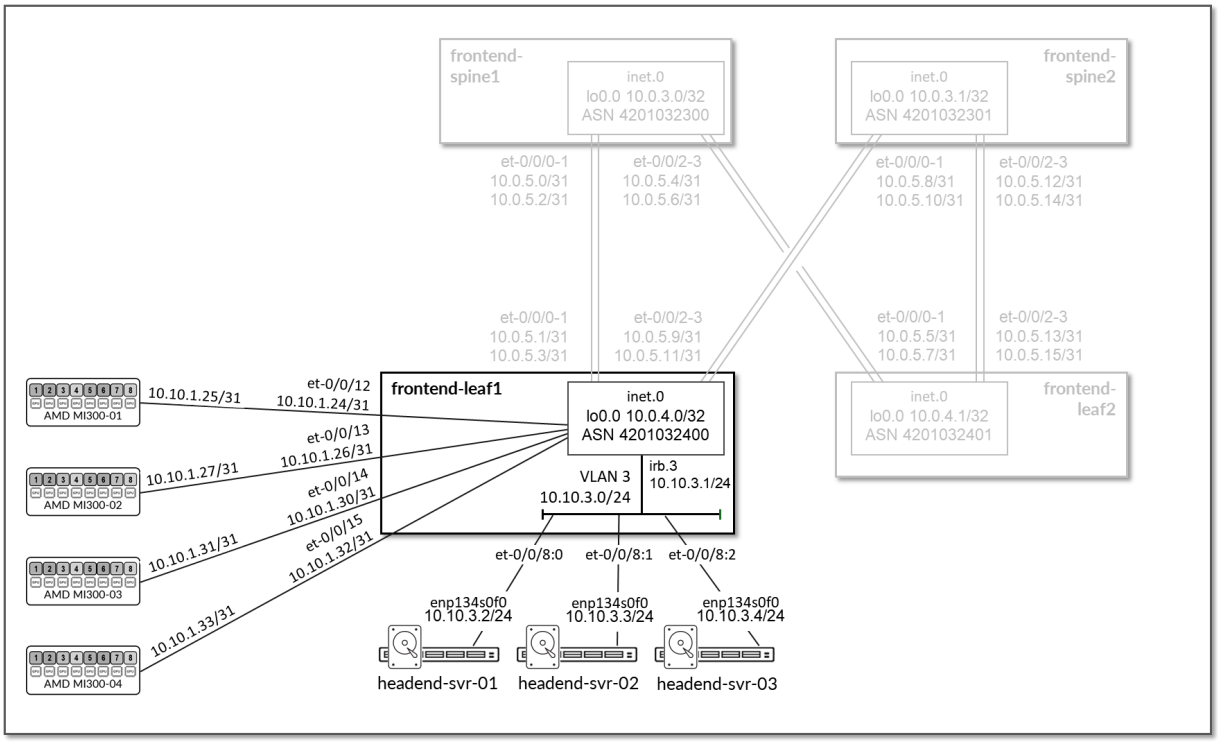
NOTE: All the Autonomous System numbers and IP addresses, including the devices loopback interface addresses (shown in table 17), are assigned by Apstra from predefined pools of resources and based on the defined intent.

Table 17: Frontend Loopback Addresses

Device	Loopback interface address
frontend-spine1	10.0.3.0/32
frontend-spine2	10.0.3.1/32
frontend-leaf1	10.0.1.0/32
frontend-leaf2	10.0.1.1/32

The MI300X GPU Servers and Headend servers are all connected to *frontend-leaf1* as shown in Figure 63.

Figure 63: Frontend Leaf Nodes to GPU Servers Connectivity



The links between these servers and *frontend-leaf1* are configured with /31 subnets from the 10.0.5.0/24 subnet.

The links between the Headend servers and *frontend-leaf1* do not have IP addresses assigned on the leaf node side. Layer 3 connectivity to the fabric is provided via an irb interface with address 10.10.3.1/24. The headend servers are configured with addresses out of 10.10.3.0/24 and use 10.10.3.1 as their default gateway. This is shown in table 18.

Table 18: Frontend Leaf Nodes to GPU Servers Interfaces Addresses

GPU Server	Leaf node	GPU Server IP address	Leaf IP address
MI300X GPU Server 1	***frontend-leaf1***	10.10.1.25/31	10.10.1.24/31
MI300X GPU Server 2		10.10.1.27/31	10.10.1.26/31
MI300X GPU Server 3		10.10.1.31/31	10.10.1.30/31
MI300X GPU Server 4		10.10.1.33/31	10.10.1.32/31
Headend Server 1		10.10.3.2/24	10.10.3.1/24 (irb.3)

(Continued)

GPU Server	Leaf node	GPU Server IP address	Leaf IP address
Headend Server 1		10.10.3.3/24	
Headend Server 1		10.10.3.4/24	

Routing information

EBGP is configured between the IP addresses assigned to the spine-leaf links. There are two EBGP sessions between each *frontend-leaf#* node and each *frontend-spine#*

Table 19: Frontend Sessions

Spine node	Leaf node	Spine	Leaf ASN	Spine IP address	Leaf IP address
frontend-spine1	*frontend-leaf1*	4201032300	4201032400	10.0.5.0/31 10.0.5.2/31	10.0.5.1/31 10.0.5.3/31
frontend-spine1	*frontend-leaf2*		4201032401	10.0.5.4/31 10.0.5.6/31	10.0.5.4/31 10.0.5.7/31
frontend-spine2	*frontend-leaf1*	4201032301	4201032400	10.0.5.8/31 10.0.5.10/31	10.0.5.9/31 10.0.5.11/31
frontend-spine2	*frontend-leaf2*		4201032401	10.0.5.12/31 10.0.5.14/31	10.0.5.13/31 10.0.5.15/31

NOTE: All the devices are configured to perform ECMP load balancing, as explained later in the document.

On the *frontend-leaf1* node, BGP policies are configured by Apstra to advertise the following routes to the spine nodes:

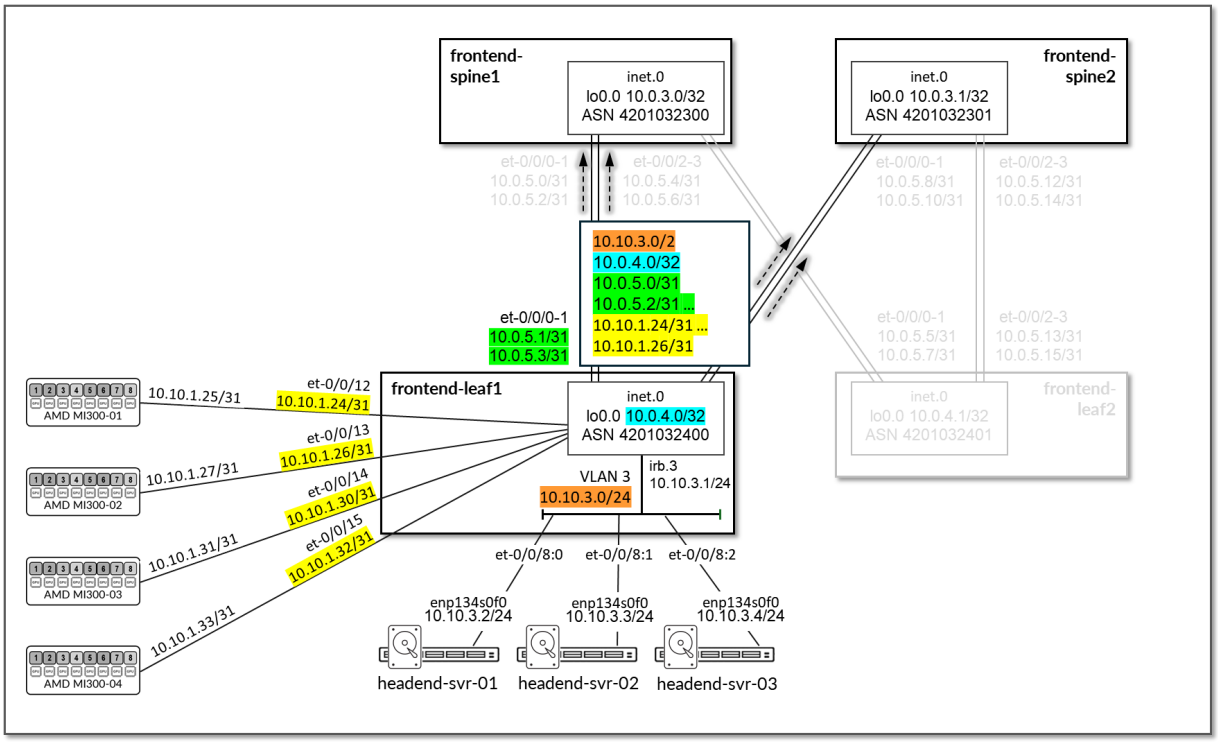
- *frontend-leaf1* node loopback interface address

- *frontend-leaf#* nodes *frontend-spine#* nodes links /31 subnet address
- *AMD MI300X-0#* to *frontend-leaf1* node links /31 subnet address
- *frontend-leaf1* irb interface subnets (connecting the Headend servers)

Table 20: Frontend Leaf to Frontend Spines advertised routes

Leaf Node	Peer(s)	Advertised routes		Adv. BGP Communities
frontend-leaf1	*frontend-spine1* & *frontend-spine2*	**Leaf1 Loopback:**10.0.4.0/32 **Leaf1-spines links:**10.0.5.0/31 10.0.5.2/31 10.0.5.8/31 10.0.5.10/31	**GPU servers <=> frontend spine links:**10.10.1.24/31 10.10.1.26/31 10.10.1.30/31 10.10.1.32/31 **Headend servers' subnet:** 10.10.3.0/24	3:20007 21001:26000

Figure 64: Frontend Leaf 1 to Frontend Spines advertised routers – routes to AMD MI300X servers and Headend Servers



On the Spine nodes, BGP policies are configured by Apstra to advertise the following routes to the *frontend-leaf2* leaf node:

- *frontend-spine#* node loopback interface address
- *frontend-leaf#* nodes loopback interface address
- *frontend-spine#* nodes *frontend-leaf#* nodes links /31 subnet address
- *AMD MI300X-0#* to *frontend-leaf1* node links /31 subnet address
- *frontend-leaf1* irb interface subnets (connecting the Headend servers)

Figure 65: Frontend Spines to Frontend Leaf 2 advertised routers – routes to AMD MI300X servers and Headend Servers

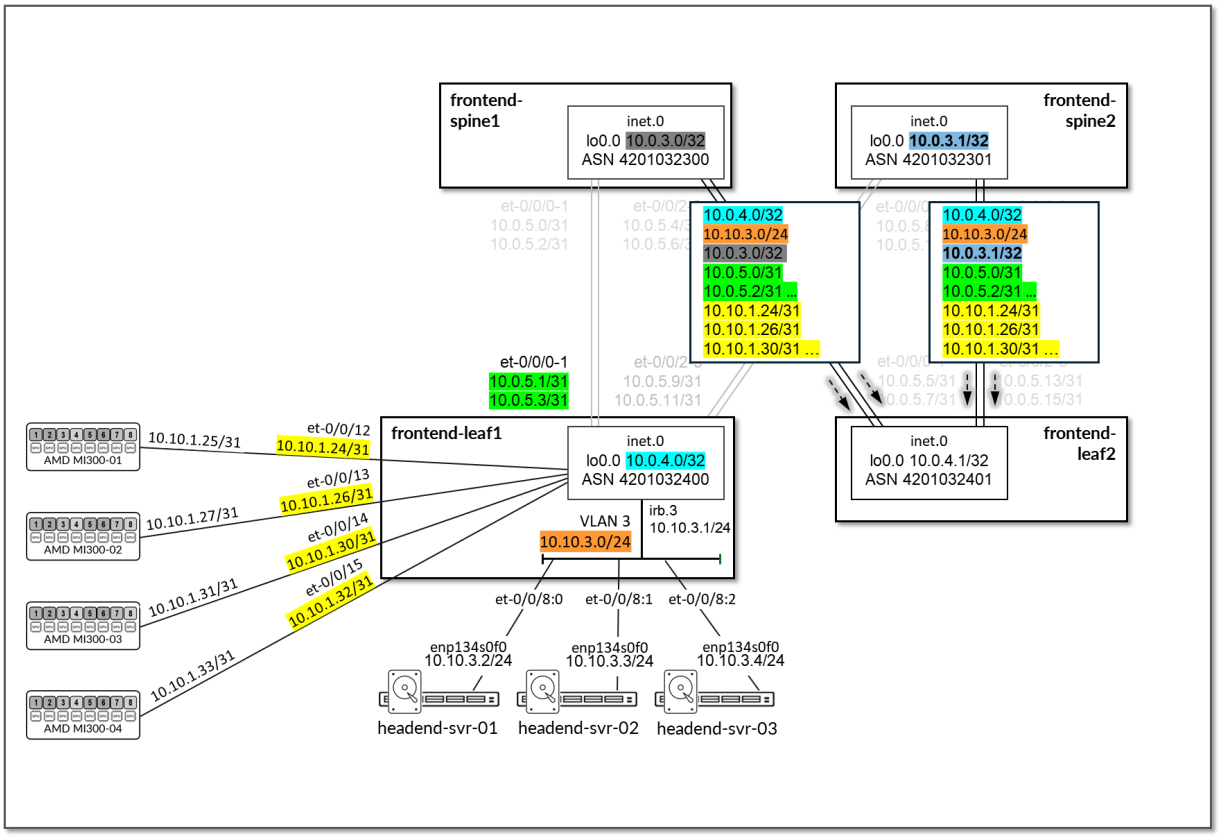


Table 21 Frontend Spine to Frontend Leaf Advertised Routes

Leaf Node	Peer(s)	Advertised Routes		BGP Communities
frontend-spine1	*frontend-leaf1*	**Loopbacks:**10.0.3.0/32 10.0.4.0/32 **Leaf1-spines links:**10.0.5.0/31 10.0.5.2/31 10.0.5.8/31 10.0.5.10/31 . . .	**GPU servers <=> frontend spine links:**10.10.1.24/31 10.10.1.26/31 10.10.1.30/31 10.10.1.32/31 **Headend servers' subnet:** 10.10.3.0/24	0:15 1:20007 21001:26000 Except for 10.0.4.1/32 (0:15 4:20007 21001:26000)

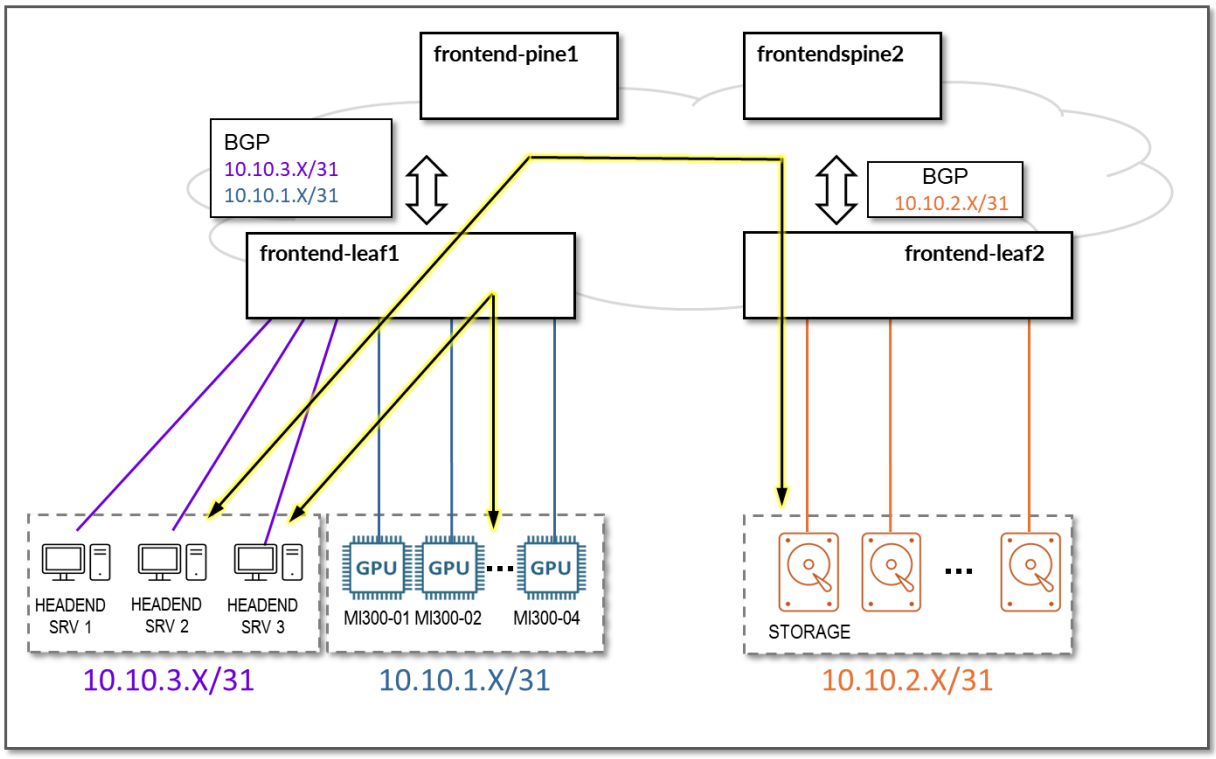
(Continued)

Leaf Node	Peer(s)	Advertised Routes		BGP Communities
frontend-spine2	*frontend-leaf1*	**Loopbacks:**10.0.3.1/32 10.0.4.0/32 **Leaf1-spines links:**10.0.5.0/31 10.0.5.2/31 10.0.5.8/31 10.0.5.10/31 . . .	**GPU servers <=> frontend spine links:**10.10.1.24/31 10.10.1.26/31 10.10.1.30/31 10.10.1.32/31 **Headend servers' subnet:** 10.10.3.0/24	0:15 2:20007 21001:26000 Except for 10.0.4.1/32 (0:15 4:20007 21001:26000)

NOTE: Equivalent policies are in place to advertise routes from *frontend-leaf2*

Advertising these subnets has the goal of allowing communication between the headend servers and the MI300X GPU server for AI job orchestration and monitoring, and between the headend servers and the storage devices connected to *frontend-leaf2*. Communication between the headend servers and storage devices is not discussed further in this document. Remember that the Vast storage devices are not connected to the frontend fabric.

Figure 66: Headend servers to GPU servers and storage devices across the frontend fabric.



GPU Backend Network Connectivity

The GPU backend fabric consists of four spine nodes (QFX5230-64CD) and eight leaf nodes (QFX5230-64CD) per stripe. Two AMD MI300X GPU servers are connected to each stripe, following the rail-optimized architecture described earlier in this document.

IP addressing

The GPU backend fabric is designed as a Layer 3 IP Fabric, with two 400GE links between each *gpu-backend-leaf#* node and each *gpu-backend-spine#* node as shown in Figure 67. These links are configured with /31 IP addresses, as shown in Table 22 and Figure 67.

Figure 67: GPU Backend Spine to GPU Backend Leaf Nodes Connectivity

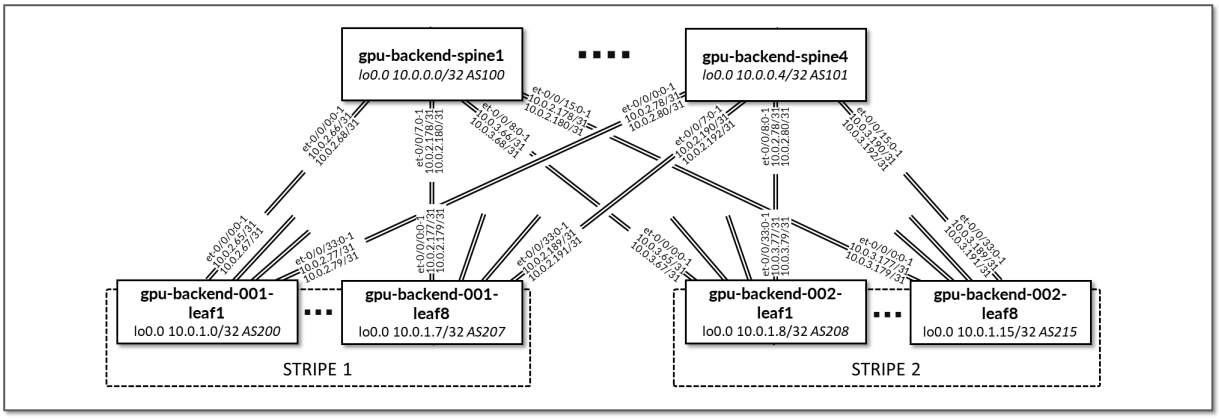


Table 22: GPU Backend Interface Addresses (sample)

Stripe #	Leaf node	Leaf node	Spine IP address	Leaf IP address
1	*gpu-backend-001-leaf1*	*gpu-backend-spine1*	10.0.2.65/31 10.0.2.67/31	10.0.2.1/31 10.0.2.3/31
1	*gpu-backend-001-leaf1*	*gpu-backend-spine2*	10.0.2.69/31 10.0.2.71/31	10.0.2.5/31 10.0.2.7/31
1	*gpu-backend-001-leaf1*	*gpu-backend-spine3*	10.0.2.73/31 10.0.2.75/31	10.0.2.9/31 10.0.2.11/31
1	*gpu-backend-001-leaf1*	*gpu-backend-spine4*	10.0.2.77/31 10.0.2.79/31	10.0.2.65/31 10.0.2.67/31
1	*gpu-backend-001-leaf2*	*gpu-backend-spine1*	10.0.2.81/31 10.0.2.83/31	10.0.2.69/31 10.0.2.71/31
1	*gpu-backend-001-leaf2*	*gpu-backend-spine2*	10.0.2.85/31 10.0.2.87/31	10.0.2.73/31 10.0.2.75/31
		...		

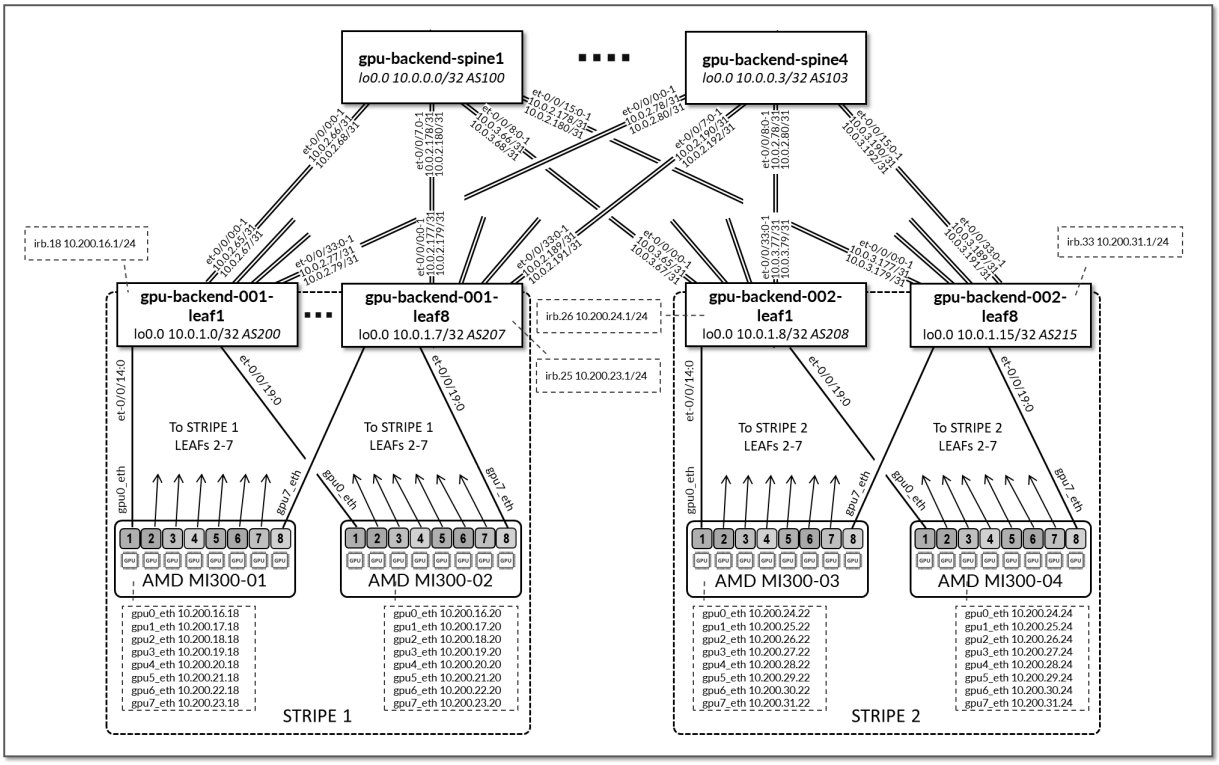
NOTE: All the Autonomous System numbers and IP addresses, including the devices loopback interface addresses (shown in table 23), are assigned by Apstra from predefined pools of resources and based on the defined intent.

Table 23: GPU Backend Loopback Addresses (sample)

Stripe #	Device	Loopback Interface Address
-	*gpu-backend-spine1*	10.0.0.0/32
-	*gpu-backend-spine2*	10.0.0.1/32
-	*gpu-backend-spine3*	10.0.0.2/32
-	*gpu-backend-spine4*	10.0.0.3/32
1	*gpu-backend-001-leaf1*	10.0.1.0/32
1	*gpu-backend-001-leaf2*	10.0.1.1/32
1	*gpu-backend-001-leaf3*	10.0.1.2/32
	. . .	

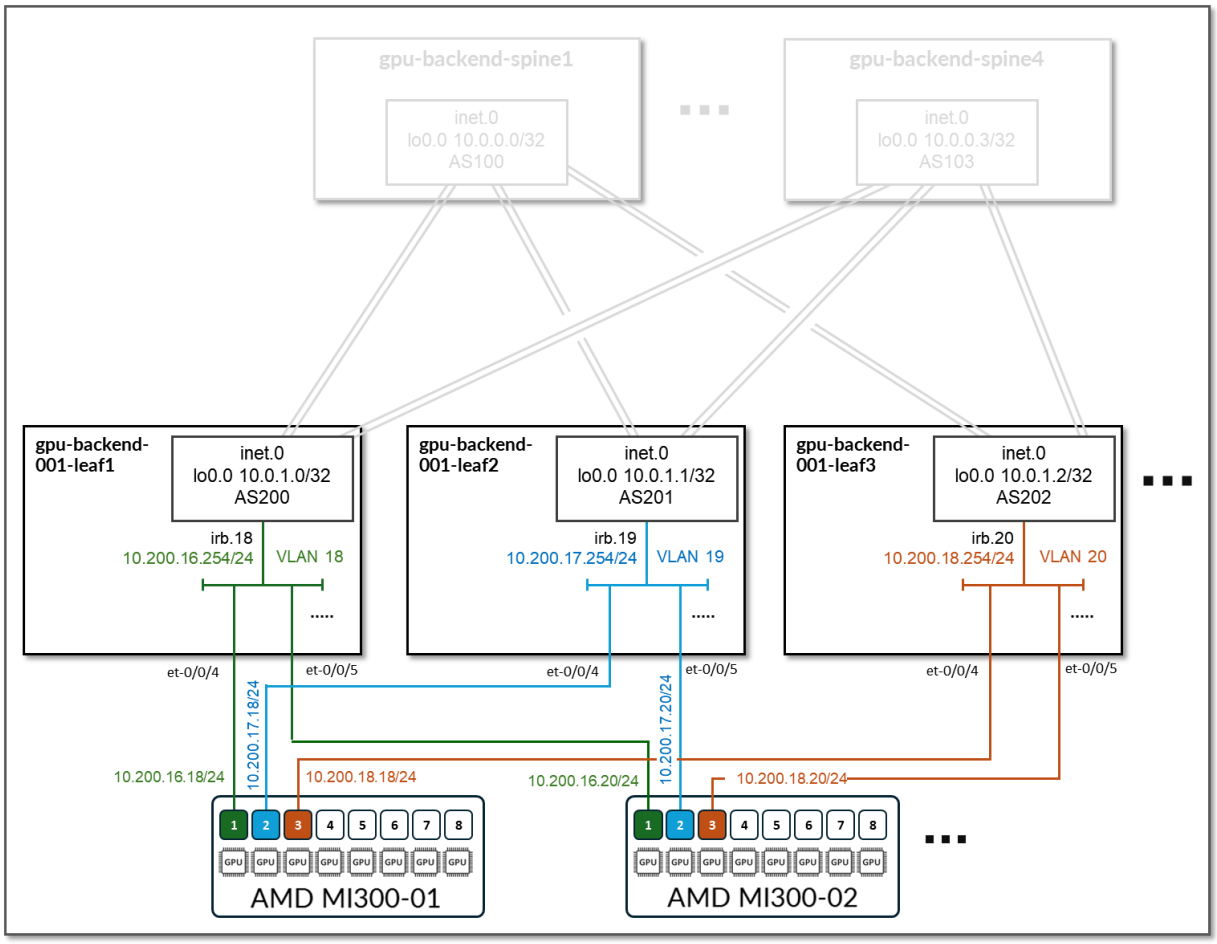
The AMD MI300X GPU Servers and Headend servers are all connected to *gpu-backend-00#-leaf#* nodes following the rail optimized architecture, as shown in Figure 68.

Figure 68: AMD MI300X GPU Servers connections to Storage Backend fabric



The links between the AMD MI300X GPU servers and the *gpu-backend-00#-leaf#* nodes do not have IP addresses assigned on the leaf node side. Layer 3 connectivity to the fabric is provided via irb interfaces with addresses 10.200.N.254/16, where N = 16, 17… (stripe/rail specific subnet), and the interfaces connected to the GPU servers are configured as L2 interfaces and are part of a unique vlan, as shown in Figure 69.

Figure 69: AMD MI300X GPU Servers rail optimized connections to Leaf nodes



Each rail in the cluster is mapped to a different /24 IP subnet. The GPU servers are configured with addresses out of 10.200.N.<server>/24 and use 10.200.N.254/14 as their default gateway. Where:

Server name	<server>	IP addresses
MI300X-01	18	10.200.N.18 24
MI300X-02	20	10.200.N.20 24
MI300X-03	22	10.200.N.22 24
MI300X-04	24	10.200.N.24 24

Table 24: GPU Backend Servers to Leaf Nodes Connectivity

Stripe #	Device	Rail #	VLAN #/ IRB #	Subnet	IRB on leaf	Connected device(s)
1	*gpu-backend-leaf*1	1	18	10.200.16.0/24	10.200.16.254	GPU 1 from the 2 GPU servers in the stripe
1	*gpu-backend-leaf*2	2	19	10.200.17.0/24	10.200.17.254	GPU 2 from the 2 GPU servers in the stripe
1	*gpu-backend-leaf*3	3	20	10.200.18.0/24	10.200.18.254	GPU 3 from the 2 GPU servers in the stripe
. . .						

Routing information

EBGP is configured between the IP addresses assigned to the spine-leaf links. There are two EBGP sessions between each *gpu-backend-leaf#* node and each *gpu-backend-spine#*.

Table 25: GPU Backend Sessions

Stripe #	Spine Node	Leaf Node	Spine ASN	Leaf ASN	Spine IP Address	Leaf IP Address
1	*gpu-backend-spine1*	*gpu-backend-001-leaf1*	4201032100	4201032200	10.0.2.0/31 10.0.2.2/31	10.0.2.1/31 10.0.2.3/31
1	*gpu-backend-spine1*	*gpu-backend-001-leaf2*		4201032201	10.0.2.4/31 10.0.2.6/31	10.0.2.5/31 10.0.2.7/31
1	*gpu-backend-spine1*	*gpu-backend-001-leaf3*		4201032202	10.0.2.8/31 10.0.2.10/31	10.0.2.9/31 10.0.2.11/31

(Continued)

Stripe #	Spine Node	Leaf Node	Spine ASN	Leaf ASN	Spine IP Address	Leaf IP Address
	. ..					
1	*gpu-backend-spine2*	*gpu-backend-001-leaf1*	4201032101	4201032208	10.0.2.64/31 10.0.2.66/31	10.0.2.65/31 10.0.2.67/31
1	*gpu-backend-spine2*	*gpu-backend-001-leaf2*		4201032209	10.0.2.68/31 10.0.2.70/31	10.0.2.69/31 10.0.2.71/31
1	*gpu-backend-spine2*	*gpu-backend-001-leaf3*		4201032210	10.0.2.72/31 10.0.2.74/31	10.0.2.73/31 10.0.2.75/31
	...					

NOTE: All the devices are configured to perform ECMP load balancing, as explained later in the document.

On the Leaf nodes, BGP policies are configured by Apstra to advertise the following routes to the spine nodes:

- *gpu-backend-000#leaf#* nodes loopback interface addresses
- *gpu-backend-000#leaf#* nodes to *gpu-backend-spine#* nodes links /31 subnet address
- *gpu-backend-00#-leaf#* irb interface subnets (connecting the GPU servers)

Figure 70: GPU Backend Leaf to GPU Backend Spine advertised routers – routes to Vast AMD devices

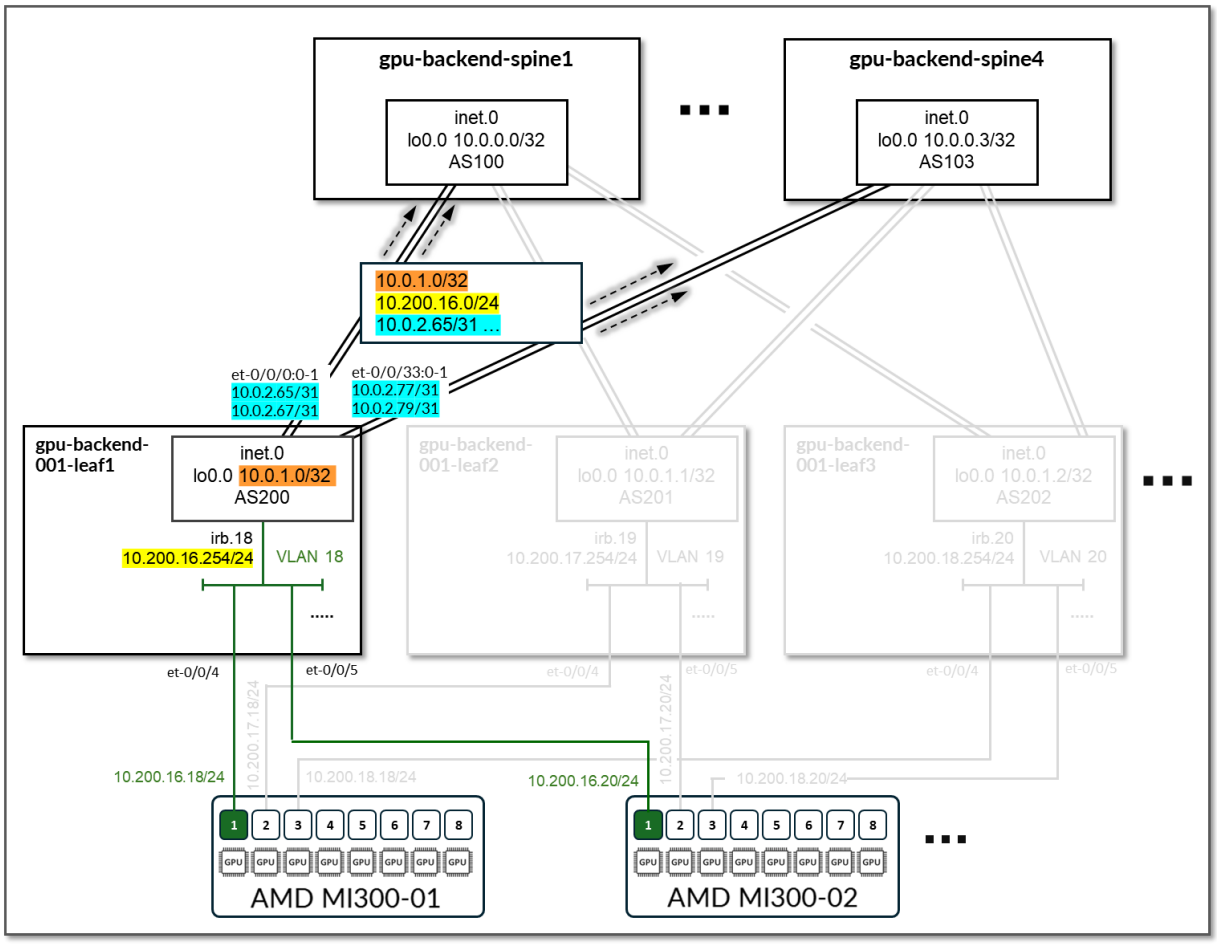


Table 26: GPU Backend Leaf Node Advertised Routes

Stripe #	Device	Advertised routes	BGP community
1	*gpu-backend-001-leaf* 1	10.0.1.0/32 10.200.16.0/24 10.0.2.64/31 10.0.2.65/31 10.0.2.68/31 10.0.2.70/31 10.0.2.72/31 10.0.2.74/31 10.0.2.76/31 10.0.2.78/31	3:20007 21001:26000
1	*gpu-backend-001-leaf2*	10.0.1.1/32 10.200.17.0/24 10.0.2.82/31 10.0.2.84/31 10.0.2.86/31 10.0.2.88/31 10.0.2.90/31 10.0.2.92/31 10.0.2.94/31 10.0.2.96/31	4:20007 21001:26000

(Continued)

Stripe #	Device	Advertised routes	BGP community
1	*gpu-backend-001-leaf3*	10.0.1.2/32 10.200.18.0/24 10.0.2.98/31 10.0.2.100/31 10.0.2.102/31 10.0.2.104/31 10.0.2.106/31 10.0.2.108/31 10.0.2.110/31 10.0.2.112/31	5:20007 21001:26000
	. . .		

On the Spine nodes, BGP policies are configured by Apstra to advertise the following routes to the *gpu-backend-00#-leaf#* nodes:

- *gpu-backend-spine#* node loopback interface address
- *gpu-backend-00#-leaf#* nodes loopback interface address
- *gpu-backend-spine#* nodes *gpu-backend-00#-leaf#* nodes links /31 subnet address
- *gpu-backend-00#-leaf#* irb interface subnets (connecting the GPU servers)

Figure 71: GPU Backend Spine to GPU Backend Leaf advertised routers – routes to Vast AMD devices

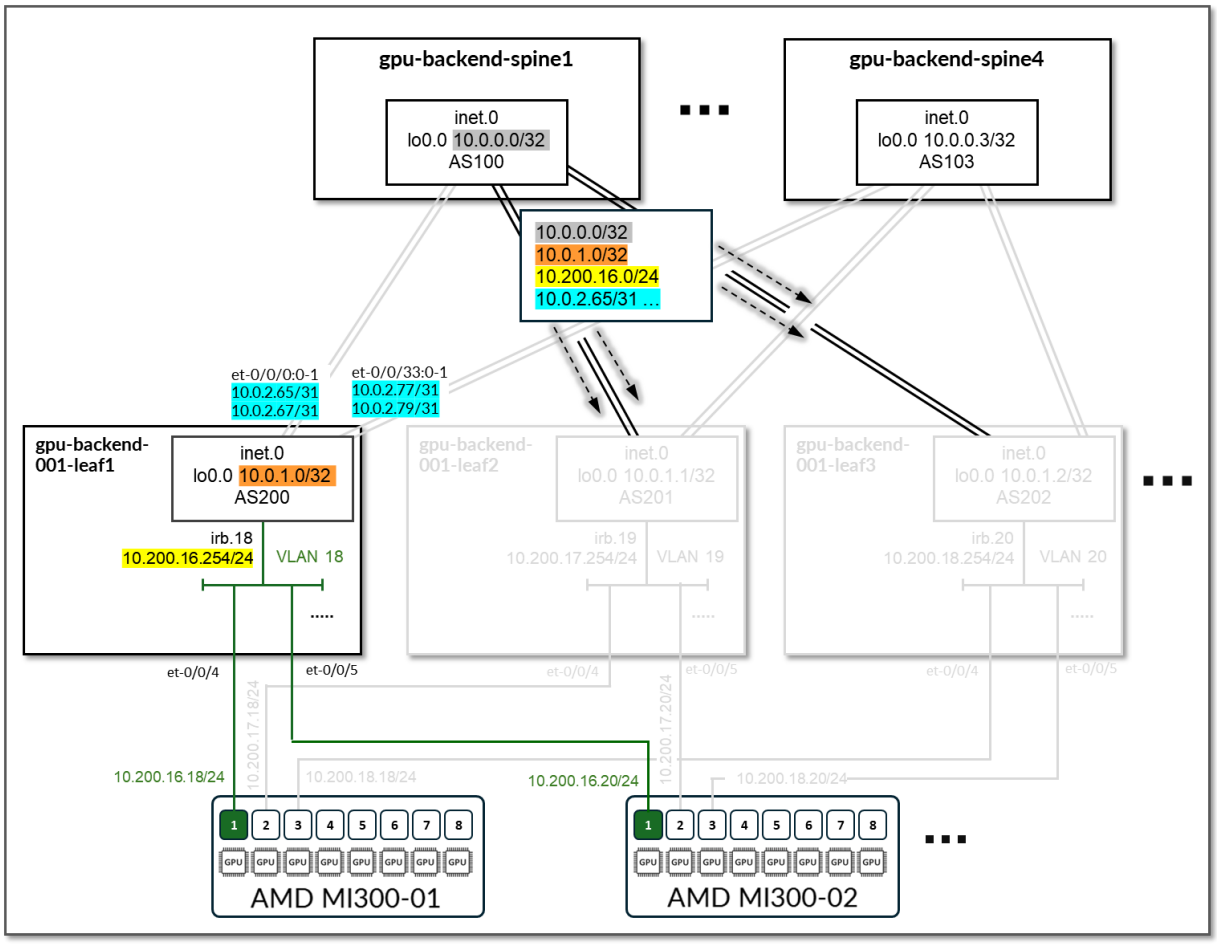
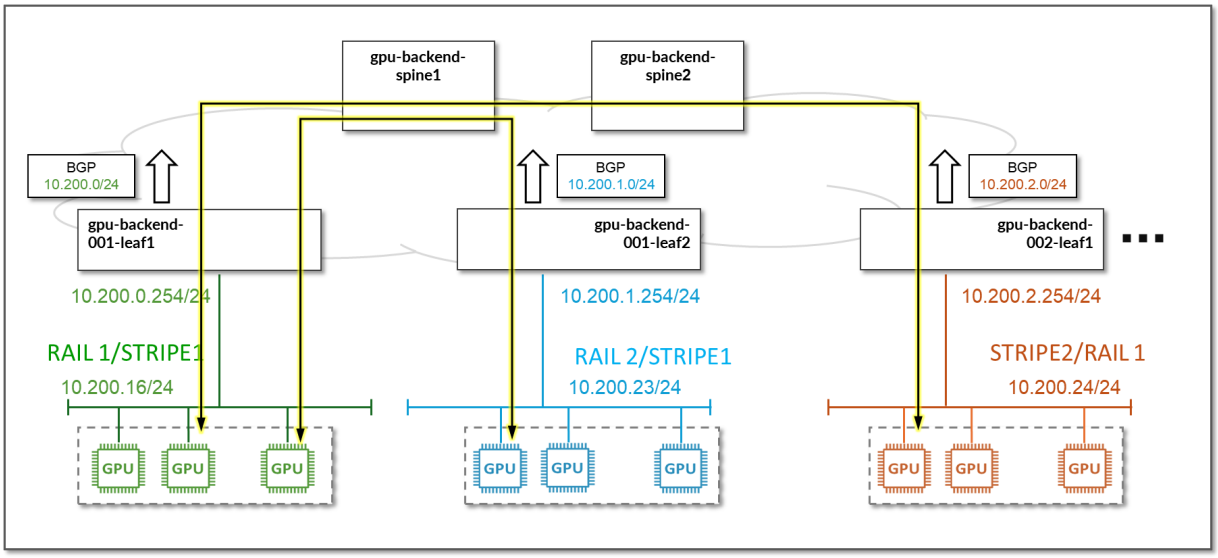


Table 27: GPU Backend Spine Node Advertised Routes

Stripe #	Spine Node	Advertised Routes	BGP Community
1	*gpu-backend-spine* 1	10.0.0.0/32 10.0.0.1/32 10.0.0.2/32 10.0.0.3/32 10.0.1.0/31 10.0.1.2/31 10.0.1.3/31 ... 10.200.16.0/24 10.200.17.0/24 ...	0:15 1:20007 21001:26000
1	*gpu-backend-spine* 2	10.0.0.0/32 10.0.0.1/32 10.0.0.2/32 10.0.0.3/32 10.0.1.0/31 10.0.1.2/31 10.0.1.3/31 ... 10.200.16.0/24 10.200.17.0/24 ...	0:15 2:20007 21001:26000
	...		

Advertising these subnets has the goal of allowing communication between the GPUs across all rails in stripes 1 and 2.

Figure 72: Communication Across Rails



Storage Backend Network Connectivity

The Storage Backend fabric consists of two spine nodes (QFX5220-32CD) and six leaf nodes (QFX5220-32CD), where two leaf nodes (*storage-backend-gpu-leaf3* and *storage-backend-gpu-leaf4*) provide connectivity to the Vast storage devices, and the other four provide connectivity to the GPU servers including the AMD MI300X servers (*storage-backend-gpu-leaf5* and *storage-backend-gpu-leaf6*) and other GPU servers in the lab.

IP addressing

The GPU Backend fabric is designed as a Layer 3 IP Fabric, with either two 400GE links between each *storage-backend-gpu-leaf#* node and each *storage-backend-spine#* node as shown in Figure 73. These links are configured with /31 IP addresses, as shown in Table 28.

Figure 73: Storage Backend Spine to Storage Backend GPU Leaf Nodes Connectivity

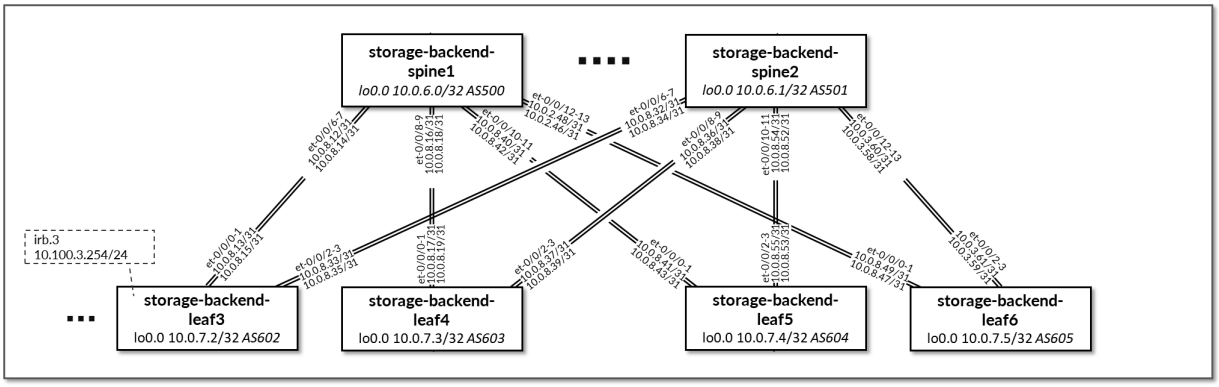


Table 28: Storage Backend Interface Addresses

Spine node	Leaf node	Spine IP Address	Leaf IP Address
storage-backend-spine 1	*storage-backend-gpu-leaf* 1	10.0.8.0/31 10.0.8.2/31 10.0.8.4/31	10.0.8.1/31 10.0.8.3/31 10.0.8.5/31
storage-backend-spine1	*storage-backend-gpu-leaf2*	10.0.8.6/31 10.0.8.8/31 10.0.8.10/31	10.0.8.7/31 10.0.8.9/31 10.0.8.11/31
storage-backend-spine1	*storage-leaf 1*	10.0.8.12/31 10.0.8.14/31	10.0.8.13/31 10.0.8.15/31
storage-backend-spine1	*storage-leaf 2*	10.0.8.16/31 10.0.8.18/31	10.0.8.17/31 10.0.8.19/31
storage-backend-spine2	*storage-backend-gpu-leaf1*	10.0.8.20/31 10.0.8.22/31 10.0.8.24/31	10.0.8.21/31 10.0.8.23/31 10.0.8.25/31

(Continued)

Spine node	Leaf node	Spine IP Address	Leaf IP Address
storage-backend-spine2	*storage-backend-gpu-leaf2*	10.0.8.26/31 10.0.8.28/31 10.0.8.30/31	10.0.8.27/31 10.0.8.29/31 10.0.8.31/31
storage-backend-spine2	*storage-leaf 1*	10.0.8.32/31 10.0.8.34/31	10.0.8.33/31 10.0.8.35/31
storage-backend-spine2	*storage-leaf 2*	10.0.8.36/31 10.0.8.38/31	10.0.8.37/31 10.0.8.39/31

NOTE: All the Autonomous System numbers and IP addresses, including the devices loopback interface addresses (shown in table 29), are assigned by Apstra from predefined pools of resources and based on the defined intent.

Table 29: Storage Backend Loopback Interfaces

Device	Loopback Interface Address
storage-backend-spine1	10.0.6.0/32
storage-backend-spine2	10.0.6.1/32
storage-backend-gpu-leaf3	10.0.7.2/32
storage-backend-gpu-leaf4	10.0.7.3/32
storage-backend-gpu-leaf5	10.0.7.4/32
storage-backend-gpu-leaf6	10.0.7.5/32

The Vast C-nodes and AMD MI300Xs are dual homed and connected to storage leaf nodes 3-4, and 5-6 respectively.

Table 30. Vast storage and GPU servers to Backend fabric connections

Device	Loopback Interface Address
Vast01--Vast08	*storage-backend-gpu-leaf3* *storage-backend-gpu-leaf4*
MI300X-01-- MI300X-04	*storage-backend-gpu-leaf5* *storage-backend-gpu-leaf6*

The links between the VAST C-nodes and the *storage-backend-leaf#* nodes do not have IP addresses assigned on the leaf node side. Layer 3 connectivity to the fabric is provided via the irb.3 and irb.4 interfaces with addresses 10.200.3.254/24 (*storage-backend-leaf3)* 10.200.4.254/24 (*storage-backend-leaf4)* respectively. The interfaces connected to the GPU servers are configured as L2 interfaces and are part of a unique vlan which the irb interfaces are part of. This is similar to the GPU Backend Servers to Leaf Nodes Connectivity described in the previous section.

On the Vast side of these links, the C-nodes are dynamically assigned responsibility for traffic destined to the addresses that are part of the Virtual IP address pools 10.100.3.0/24 and 10.100.4.0/24, (under Network Access in the WebUI), as shown in the following table.

Figure 74: AMD M300I servers and Vast Storage connectivity to the Storage Backend

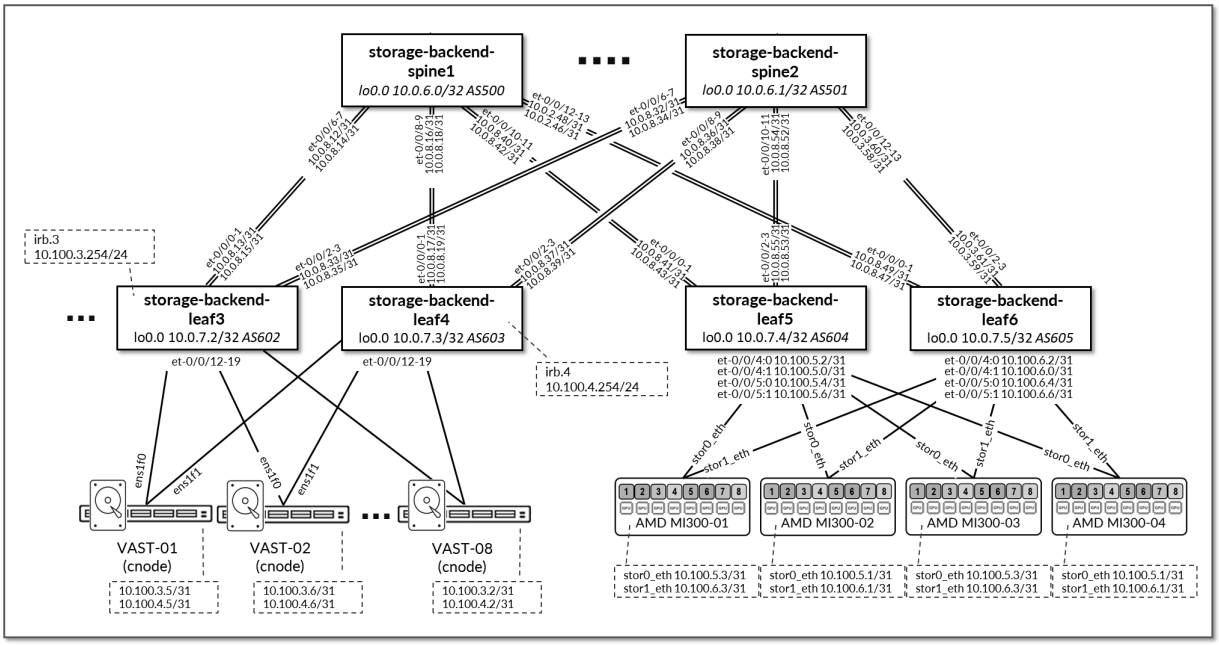


Table 31: Vast Storage to Leaf Nodes Interface Addresses

Vast node	Leaf Node	VAST C-node IP Address	Leaf IP Address
Vast C-node 1	*storage-backend-leaf 3*	10.100.3.5/24	10.100.3.254/24 (irb.3)
Vast C-node 1	*storage-backend-leaf 4*	10.100.4.4/24	10.100.4.254/24 (irb.4)
Vast C-node 2	*storage-backend-leaf 3*	10.100.3.6/24	10.100.3.254/24 (irb.3)
Vast C-node 2	*storage-backend-leaf 4*	10.100.4.2/24	10.100.4.254/24 (irb.4)
Vast C-node 3	*storage-backend-leaf 3*	10.100.3.4/24	10.100.3.254/24 (irb.3)
Vast C-node 3	*storage-backend-leaf 4*	10.100.4.5/24	10.100.4.254/24 (irb.4)
Vast C-node 4	*storage-backend-leaf 3*	10.100.3.1/24	10.100.3.254/24 (irb.3)
Vast C-node 4	*storage-backend-leaf 4*	10.100.4.8/24	10.100.4.254/24 (irb.4)
Vast C-node 5	*storage-backend-leaf 3*	10.100.3.3/24	10.100.3.254/24 (irb.3)
Vast C-node 5	*storage-backend-leaf 4*	10.100.4.3/24	10.100.4.254/24 (irb.4)
Vast C-node 6	*storage-backend-leaf 3*	10.100.3.8/24	10.100.3.254/24 (irb.3)
Vast C-node 6	*storage-backend-leaf 4*	10.100.4.1/24	10.100.4.254/24 (irb.4)
Vast C-node 7	*storage-backend-leaf 3*	10.100.3.7/24	10.100.3.254/24 (irb.3)
Vast C-node 7	*storage-backend-leaf 4*	10.100.4.6/24	10.100.4.254/24 (irb.4)
Vast C-node 8	*storage-backend-leaf 3*	10.100.3.2/24	10.100.3.254/24 (irb.3)
Vast C-node 8	*storage-backend-leaf 4*	10.100.4.7/24	10.100.4.254/24 (irb.4)

The links between the GPU servers and *storage-backend-gpu-leaf#* are configured with /31 subnets out of 10.100.1/24 by Apstra.

Table 32: GPU Servers to Storage GPU Backend Interface Addresses

GPU Server	Leaf Node	GPU Server IP Address	Leaf IP Address
MI300X-01	*storage-backend-gpu-leaf5*	10.100.5.1/31	10.100.5.0/31
MI300X-01	*storage-backend-gpu-leaf6*	10.100.6.1/31	10.100.6.0/31
MI300X-02	*storage-backend-gpu-leaf5*	10.100.5.3/31	10.100.5.2/31
MI300X-02	*storage-backend-gpu-leaf6*	10.100.6.3/31	10.100.6.2/31
MI300X-03	*storage-backend-gpu-leaf5*	10.100.5.5/31	10.100.5.4/31
MI300X-03	*storage-backend-gpu-leaf6*	10.100.6.5/31	10.100.6.4/31
MI300X-04	*storage-backend-gpu-leaf5*	10.100.5.7/31	10.100.5.6/31
MI300X-04	*storage-backend-gpu-leaf6*	10.100.6.7/31	10.100.6.6/31

Routing information

EBGP is configured between the IP addresses assigned to the spine-leaf links. There are two EBGP sessions between each *storage-backend-leaf#* node and each *storage-backend-spine#,* as shown in Figure 75

Table 33: Storage Backend Sessions

Spine Node	Leaf Node	Spine ASN	Leaf ASN	Spine IP Address	Leaf IP Address
storage-backend-spine1	*storage-backend-gpu-leaf3*	4201032500	4201032602	10.0.8.12/31 10.0.8.14/31	10.0.8.13/31 10.0.8.15/31
storage-backend-spine1	*storage-backend-gpu-leaf4*		4201032603	10.0.8.16/31 10.0.8.18/31	10.0.8.17/31 10.0.8.19/31
storage-backend-spine1	*storage-backend-gpu-leaf5*		4201032604	10.0.8.40/31 10.0.8.42/31	10.0.8.41/31 10.0.8.43/31
storage-backend-spine1	*storage-backend-gpu-leaf6*		4201032605	10.0.8.46/31 10.0.8.48/31	10.0.8.47/31 10.0.8.49/31
storage-backend-spine2	*storage-backend-gpu-leaf3*	4201032501	4201032602	10.0.8.32/31 10.0.8.34/31	10.0.8.33/31 10.0.8.35/31
storage-backend-spine2	*storage-backend-gpu-leaf4*		4201032603	10.0.8.36/31 10.0.8.38/31	10.0.8.37/31 10.0.8.39/31
storage-backend-spine2	*storage-backend-gpu-leaf5*		4201032604	10.0.8.52/31 10.0.8.54/31	10.0.8.53/31 10.0.8.55/31
storage-backend-spine2	*storage-backend-gpu-leaf6*		4201032605	10.0.8.56/31 10.0.8.58/31	10.0.8.57/31 10.0.8.59/31

On the Leaf nodes, BGP policies are configured by Apstra to advertise the following routes to the spine nodes:

- *storage-backend-leaf#* nodes loopback interface addresses
- *storage-backend-leaf#* nodes to *storage-backend-spine#* interfaces subnets
- *storage-backend-leaf3, storage-backend-leaf4* nodes irb interface (connection to Vast C-nodes)

- *storage-backend-leaf5, storage-backend-leaf6* nodes to *storage-backend-spine#* interfaces subnets (connection to MI300X servers)

Figure 75: Storage Backend Leaf to Storage Backend Spines advertised routers – routes to Vast Storage devices and MI300X servers

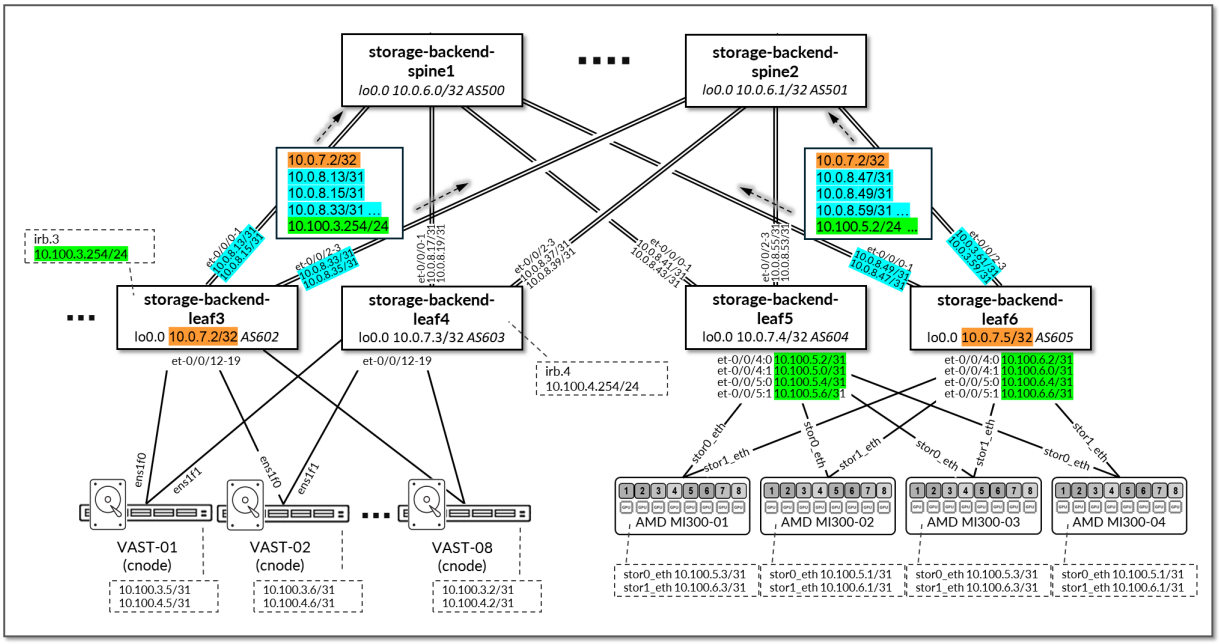


Table 34: Storage Backend Leaf Node Advertised Routes

Leaf Node	Peer	Advertised Routes		BGP Communities
storage-backend-leaf3	*storage-backend-spine1 &* *storage-backend-spine2*	10.0.7.2/32 10.0.8.12/31 10.0.8.14/31 10.0.8.32/31 10.0.8.34/31 10.100.3.0/24		5:20007 21001:26000

(Continued)

Leaf Node	Peer	Advertised Routes		BGP Communities
storage-backend-leaf4	*storage-backend-spine1 &* *storage-backend-spine2*	10.0.7.3/32 10.0.8.16/31 10.0.8.18/31 10.0.8.36/31 10.0.8.38/31 10.100.4.0/24		6:20007 21001:26000
storage-backend-leaf5	*storage-backend-spine1 &* *storage-backend-spine2*	10.0.7.4/32 10.0.8.40/31 10.0.8.42/31 10.0.8.52/31 10.0.8.54/31	10.100.5.0/31 10.100.5.2/31 10.100.5.4/31 10.100.5.6/31	7:20007 21001:26000
storage-backend-leaf6	*storage-backend-spine1 &* *storage-backend-spine2*	10.0.7.5/32 10.0.8.46/31 10.0.8.48/31 10.0.8.58/31 10.0.8.60/31	10.100.6.0/31 10.100.6.2/31 10.100.6.4/31 10.100.6.6/31	8:20007 21001:26000

On the Spine nodes, BGP policies are configured by Apstra to advertise the following routes to the leaf nodes:

- *storage-backend-leaf#* nodes loopback interface addresses
- *storage-backend-spine#* nodes loopback interface addresses
- *storage-backend-leaf#* nodes to *storage-backend-spine#* interfaces subnets
- *storage-backend-leaf3, storage-backend-leaf4* nodes irb interface (connection to Vast C-nodes)
- *storage-backend-leaf5, storage-backend-leaf6* nodes to *storage-backend-spine#* interfaces subnets (connection to MI300X servers)

Figure 76: Storage Backend Spine to Storage Backend Leafs advertised routers – routes to Vast Storage devices and MI300X servers

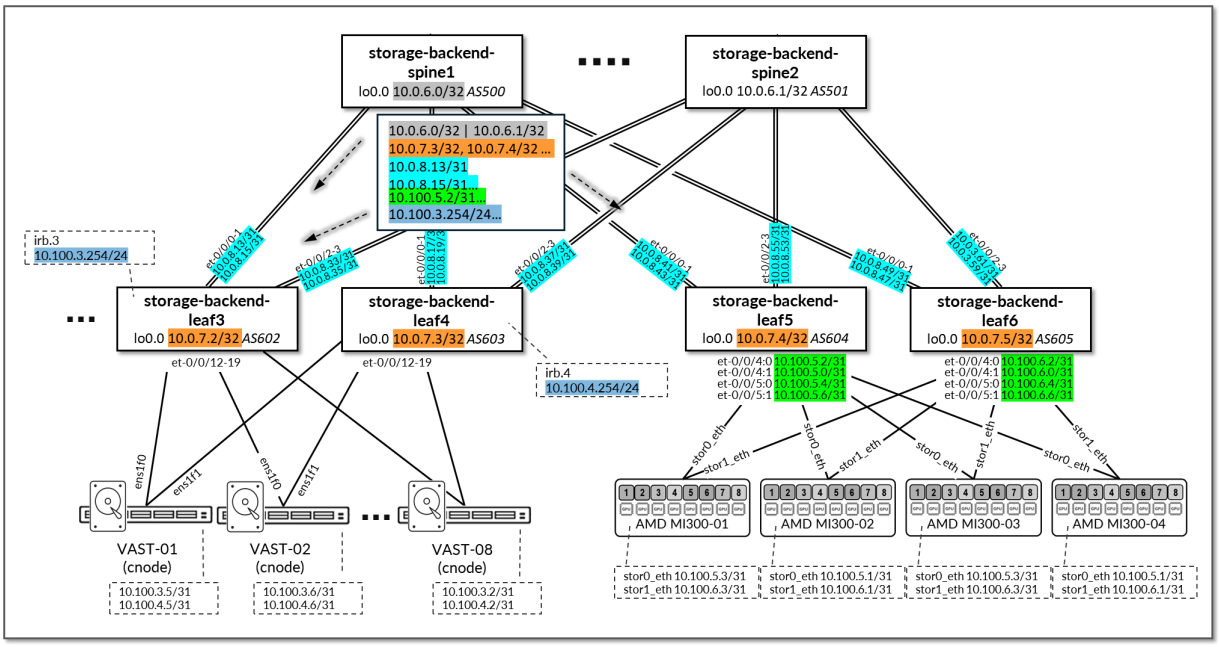


Table 35: Storage Backend Spine Node Advertised Routes

Leaf Node	Peer	Advertised Routes			BGP Communities
storage-backend-spine1	*storage-backend-leaf3*	10.0.6.0/32 10.0.7.0/32 10.0.7.1/32 10.0.7.3/32 10.0.7.4/32 10.0.7.5/32	10.100.4.0/24 10.100.5.0/31 10.100.5.2/31 10.100.5.4/31 10.100.5.6/31	10.100.6.0/31 10.100.6.2/31 10.100.6.4/31 10.100.6.6/31 10.0.8.33/31 ... 10.0.8.41/31 ... 10.0.8.49/31 ...	0:15 6:20007 21001:26000

(Continued)

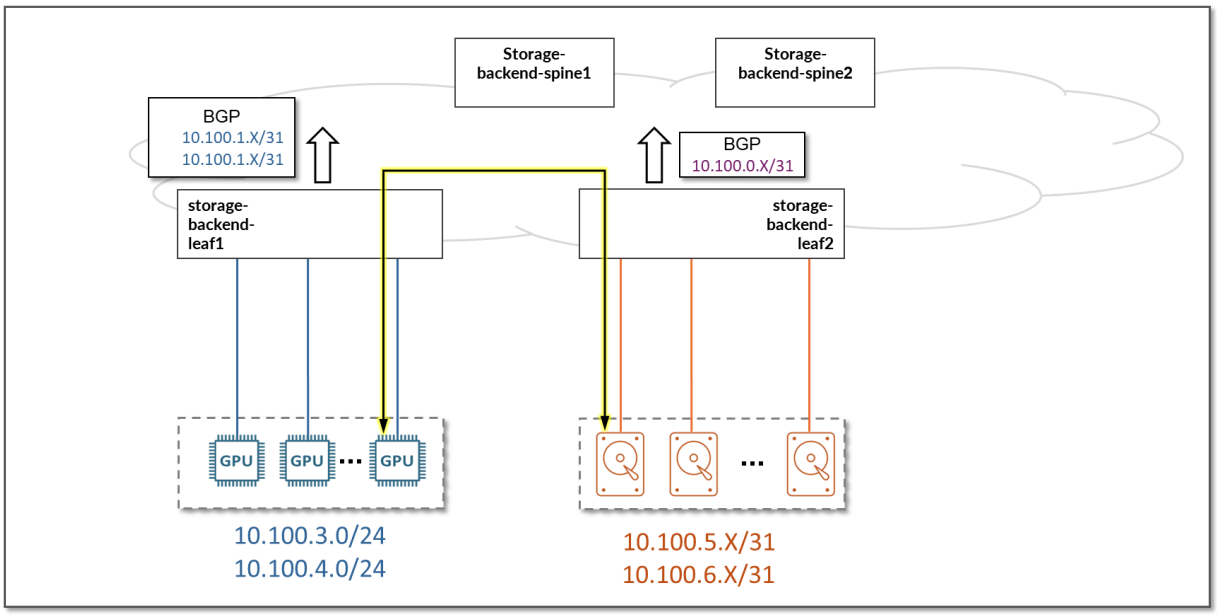
Leaf Node	Peer	Advertised Routes			BGP Communities
storage-backend-spine1	*storage-backend-leaf4*	10.0.6.0/32 10.0.7.0/32 10.0.7.1/32 10.0.7.2/32 10.0.7.4/32 10.0.7.5/32	10.100.3.0/24 10.100.5.0/31 10.100.5.2/31 10.100.5.4/31 10.100.5.6/31	10.100.6.0/31 10.100.6.2/31 10.100.6.4/31 10.100.6.6/31 10.0.8.13/31 ... 10.0.8.41/31 ... 10.0.8.49/31 ...	0:15 6:20007 21001:26000
storage-backend-spine1	*storage-backend-leaf5*	10.0.6.0/32 10.0.7.0/32 10.0.7.1/32 10.0.7.2/32 10.0.7.3/32 10.0.7.5/32	10.100.3.0/24 10.100.4.0/24	10.0.8.13/31 ... 10.0.8.33/31 ... 10.0.8.41/31 ...	0:15 6:20007 21001:26000
storage-backend-spine1	*storage-backend-leaf6*	10.0.6.0/32 10.0.7.0/32 10.0.7.1/32 10.0.7.2/32 10.0.7.3/32 10.0.7.4/32	10.100.3.0/24 10.100.4.0/24	10.0.8.13/31 ... 10.0.8.33/31 ... 10.0.8.49/31 ...	0:15 6:20007 21001:26000

(Continued)

Leaf Node	Peer	Advertised Routes			BGP Communities
storage-backend-spine2	*storage-backend-leaf5*	10.0.6.1/32 10.0.7.0/32 10.0.7.1/32 10.0.7.3/32 10.0.7.4/32 10.0.7.5/32	10.100.4.0/24	10.100.6.0/31 10.100.6.2/31 10.100.6.4/31 10.100.6.6/31 10.0.8.33/31 ... 10.0.8.41/31 ... 10.0.8.49/31 ...	0:15 4:20007 21001:26000
	...				

Advertising these subnets has the goal of allowing communication between the MI300X GPU servers and Vast Storage devices.

Figure 77: Communication between GPU servers and Storage devices



JVD Validation Framework

IN THIS SECTION

Platforms / Devices Under Test (DUT) on this JVD

To review the software versions and platforms on which this JVD was validated by Juniper Networks, see the Validated Platforms and Software section in this document.

NOTE:

QFX5220-64CD, and QFX5230-64CD acting as leaf nodes, as well as QFX5230-64CD and PTX10008 acting as spine nodes are covered in *AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD).* The same document also covers WEKA storage and NVIDIA GPUs servers.

JVD Validation Goals and Scope

IN THIS SECTION

Tests Objectives

The primary objectives of the JVD testing can be summarized as:

- Qualification of the complete AI fabric design functionality including the Frontend, GPU Backend, and Storage Backend fabrics, and connectivity between AMD GPUs and Vast Storage.
- Qualification of the deployment steps based on Juniper Apstra.
- Ensure the design is well-documented and will produce a reliable, predictable deployment for the customer.

The qualification objectives included validating:

- Validation of blueprint deployment, device upgrade, incremental configuration pushes/provisioning, Telemetry/Analytics checking, failure mode analysis, congestion avoidance and mitigation, and verification of host, storage, and GPU traffic.

Tests Scope

The AI JVD testing for the described network included the following:

- Design and blueprint deployment through Apstra of three distinct fabrics
- Fabric operation and monitoring through Apstra analytics and telemetry dashboard
- Congestion management with PFC and ECN, including failure scenarios
- End-to-end traffic flow, with Dynamic Load Balancing (DLB)
- System health, ARP, ND, MAC, BGP (route, next hop), interface traffic counters, and so on
- Software operation verification (no anomalies, or issues found)
- AI fabric with Juniper Apstra successfully performing under the following required scenarios (must):
  - Node failure (reboot)
  - Interface failures (interface down/up, Laser on/off):

Under these scenarios the following were evaluated/validated:

- Completion of AI Job models within MLCommons Training benchmarks
- Traffic recovery was validated after all failure scenarios.
- impact to the fabric and check anomalies reporting in Apstra.

Other features tested

- Broadcom 97608 THOR2 NICs
- Mellanox Connect-X NIC card default settings.
- DSCP and CNP configuration on the NICs
- Connectivity between fabric-connected hosts created by Apstra towards NSX-managed hosts.
- BERT/LLAMA3 test completion times
- Llama2 Inference against existing infrastructure.

Refer to the test report for more information.

Tested Optics

Table 37: Frontend Fabric Optics

Frontend Fabric				
Part number	Optics Name	Device Role	Device Model	Interface/NIC type
740-085351	QSFP56-DD-400GBASE-DR4	spine	QFX5130-32CD	QSFP-DD
740-085351	QSFP56-DD-400GBASE-DR4	leaf	QFX5130-32CD	QSFP-DD
740-061405	QSFP-100GBASE-SR4-T2	leaf	QFX5130-32CD	QSFP28
740-046565	QSFP+-40G-SR4 w/ 4x10G breakout cable.	leaf	QFX5130-32CD	QSFP+
AFBR-709SMZ	AVAGO 10GBASE-SR SFP+ 300m	Server	SuperMicro Headend Server	Intel X710

(Continued)

Frontend Fabric				
AFBR-89CDDZ	AVAGO 100GbE QSFP28 300m	GPU Server	AMD MI300Xx Dell XE96880	BCM97608 THOR2
AFBR-89CDDZ	AVAGO 100GbE QSFP28 300m	GPU Server	AMD MI300Xx SuperMicro AS-8125GS-TNMR2	ConnectX-7

Table 38: Backend Storage Fabric Optics

Backend Storage Fabric				
Part number	Optics Name	Device Role	Device Model	Interface/NIC type
740-085351	QSFP56-DD-400GBASE-DR4	spine	QFX5220-32CD	QSFP-DD
740-085351	QSFP56-DD-400GBASE-DR4	leaf	QFX5220-32CD	QSFP-DD
740-058734	QSFP-100GBASE-SR4	leaf	QFX5220-32CD	QSFP28
720-128730	QSFP56-DD-2x200GBASE-CR4-CU-2.5M w/ 400G DAC Breakout into 2X200G	leaf	QFX5220-32CD	QSFP-DD
740-061405	QSFP-100GBASE-SR4	leaf	QFX5220-32CD	QSFP28

(Continued)

Backend Storage Fabric				
740-159002	QSFP56-DD-2x200G-BOAOC-5M	GPU Server	AMD MI300Xx Dell XE9680	BCM97608 THOR2
740-159002	QSFP56-DD-2x200G-BOAOC-5M	GPU Server	AMD MI300Xx SuperMicro AS-8125GS-TNMR2	ConnectX-7
740-061405	QSFP-100GBASE-SR4	Storage	Vast Storage CBOX	ConnectX-6
740-061405	QSFP-100GBASE-SR4	Storage	Vast Storage DBOX	ConnectX-6

Table 39: Backend GPU Fabric

Backend GPU Fabric				
Part number	Optics Name	Device Role	Device Model	Interface/NIC type
740-174933	OSFP-800G-DR8	spine	QFX5240-64OD QFX5241-64OD	OSPF800
740-174933	OSFP-800G-DR8	leaf	QFX5240-64OD QFX5241-64OD	OSPF800
740-085351	QDD-400G-DR4	GPU Server	AMD MI300Xx Dell XE9680	BCM97608 THOR2
740-085351	QDD-400G-DR4	GPU Server	AMD MI300Xx SuperMicro AS-8125GS-TNMR2	BCM97608 THOR2

NOTE:

For optics tested on QFX5220-64CD, QFX5230-64CD, PTX10008, WEKA storage and NVIDIA GPUs servers check *AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD) Tested Optics Section.*

JVD Validation Test Results Summary and Analysis

For a detailed test results report, contact your Juniper representative.

Recommendations Summary

The AI Data Center Network with Juniper Apstra, AMD GPUs, and VAST Storage JVD follows an industry-standard dedicated IP Fabric design. Three distinct fabrics provide maximum efficiency while maintaining focus on AI model scale, expedited completion times, and rapid evolution with the advent of AI technologies.

To follow best practice recommendations:

- A minimum of 4 spines in each fabric is suggested.

Though the design for cluster 1 in this document only includes only 2 spines, we found that under certain dual failure scenarios, combined with congestion, the fabric becomes susceptible to PFC storms (not vendor-unique). We recommend deploying the solution with 4 spines as described for the QFX5240/QFX5241 fabric (cluster 2) even when using different switch models.

- Follow a rail-optimized fabric and maintain a 1:1 relation with bandwidth subscription and Leaf to GPU symmetry.
- Implement Dynamic Load Balancing (DLB) instead of traditional ECMP for optimal load distribution.
- Implement DCQCN (PFC and ECN) to ensure a lossless fabric in the GPU Backend Fabric, and possibly in the Storage Backend Fabric as required per vendor recommendation.
- Configure DCQCN (PFC and ECN) parameters on the AMD servers and change the NCCL_SOCKET interface to be the management (frontend) interface.
- The minimum recommended Junos OS releases for this JVD are:
  - Junos OS Release 23.4X100-D20 for the Juniper QFX5240-64CD

- Junos OS Release 23.4X100-D42 for the Juniper QFX5241-64CD

NOTE: For minimum software released for QFX5220-64CD, QFX5230-64CD, PTX10008, check the *AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD) Recommendations Section.*

The Juniper hardware listed in the Juniper Hardware and Software Components section are the best-suited switch platforms regarding features, performance, and the roles specified in this JVD.

Revision History

Table 40: Revision History

Date	Version	Description
Dec 2025	JVD-AICLUSTERDC-AIML-AMD-03-03	Added QFX5241, and GLB configuration. Update Rail Optimized Section.
June 2025	JVD-AICLUSTERDC-AIML-AMD-03-02	Added Pollara
February 2025	JVD-AICLUSTERDC-AIML-AMD-03-01	Initial Publish