Engineering
Simplicity

AI Data Center Multitenancy with EVPN/VXLAN—Juniper Validated Design (JVD)

Published
2025-10-29

Table of Contents

AI Data Center Multitenancy with EVPN/VXLAN—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 that includes support for GPU multitenancy in the GPU backend fabric, using EVPN/VXLAN. This fabric is built based on AI-optimized Juniper Data Center QFX5240 series switches. The cluster includes Nvidia H100 DGX as well as AMD MI300X GPU servers, and Vast Storage systems.

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.

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's best practices, and leverages Juniper's extensive expertise in building high-performance data center networks.

The design employs a 3-stage Clos IP fabric architecture, utilizing Juniper QFX-series switches as leaf and spine nodes and multi-vendor GPU servers and storage devices.

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, 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.

AI Use Case and Reference Design

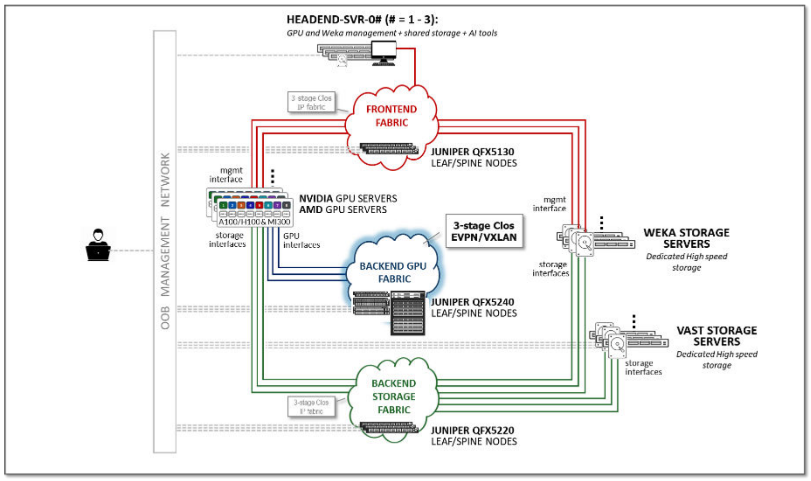
IN THIS SECTION

The **AI JVD Reference Design** covers a complete end-to-end ethernet-based AI infrastructure, which includes the Frontend fabric, GPU Backend 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 4 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 both NVIDIA Collective Communications Library (NCCL) and 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.
- **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.
- **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.
- **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 impact 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.

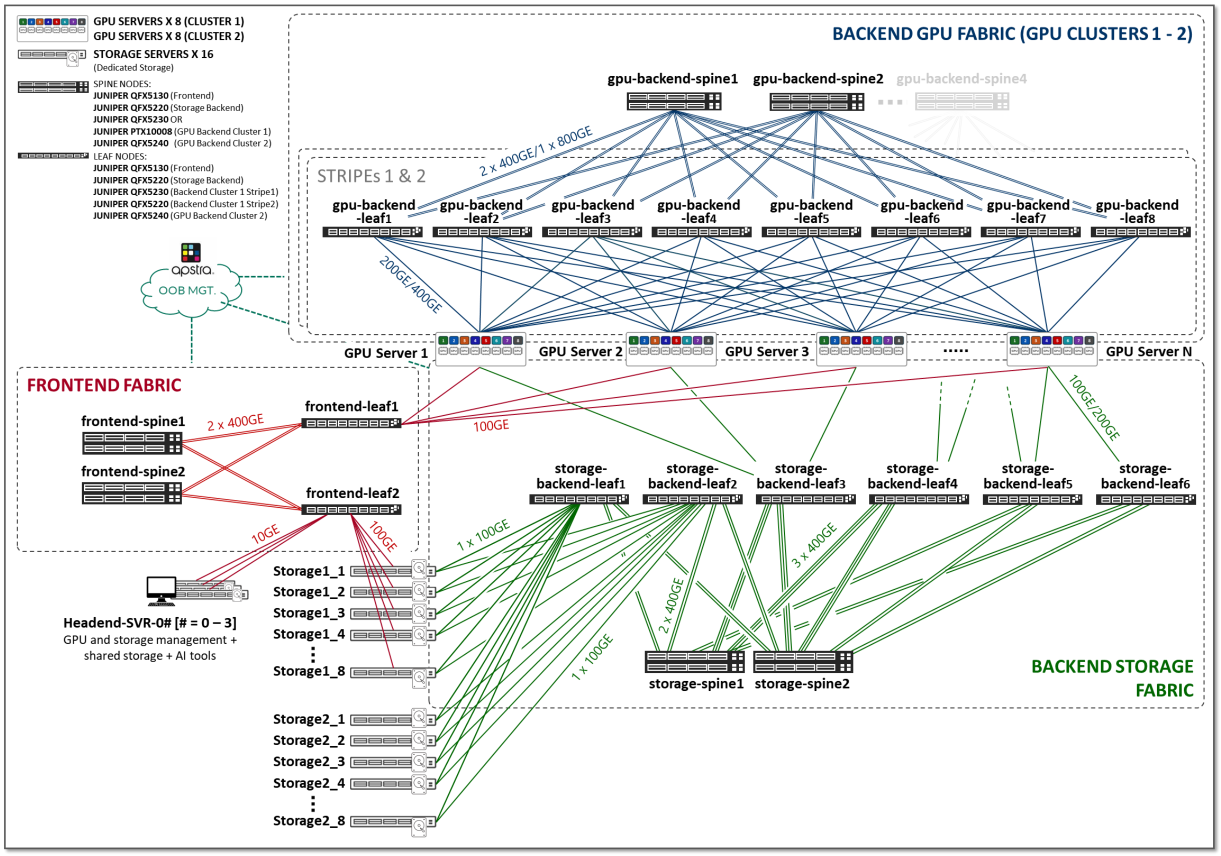
- **Storage Backend Fabric:** Routing and switching infrastructure that provides the connectivity between the GPU and the storage devices. This integration ensures that data can be efficiently transferred between storage and computational resources, optimizing overall AI workflow performance. The performance 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



Frontend Fabric

For details about connecting **Nvidia A100 and H100 GPU servers**, as well as **Weka Storage devices**, to the **Frontend Fabric**, see Frontend Fabric section of the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and Weka Storage—Juniper Validated Design (JVD).

For details about connecting **AMD MI300x GPU servers** to the **Frontend Fabric,** see Frontend Fabric section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage—Juniper Validated Design (JVD).

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, and have been tested in AI lab.

For details about connecting **Weka storage devices** to the **Storage Backend Fabric**, refer to the Storage fabric section of the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD) as well as the WEKA Storage Solution section in the same document.

For details about connecting **Vast storage devices** to the **Storage Backend Fabric**, refer to the Storage fabric section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage—Juniper Validated Design (JVD) as well as the VAST Storage Configuration section in the same document.

GPU Backend Fabric

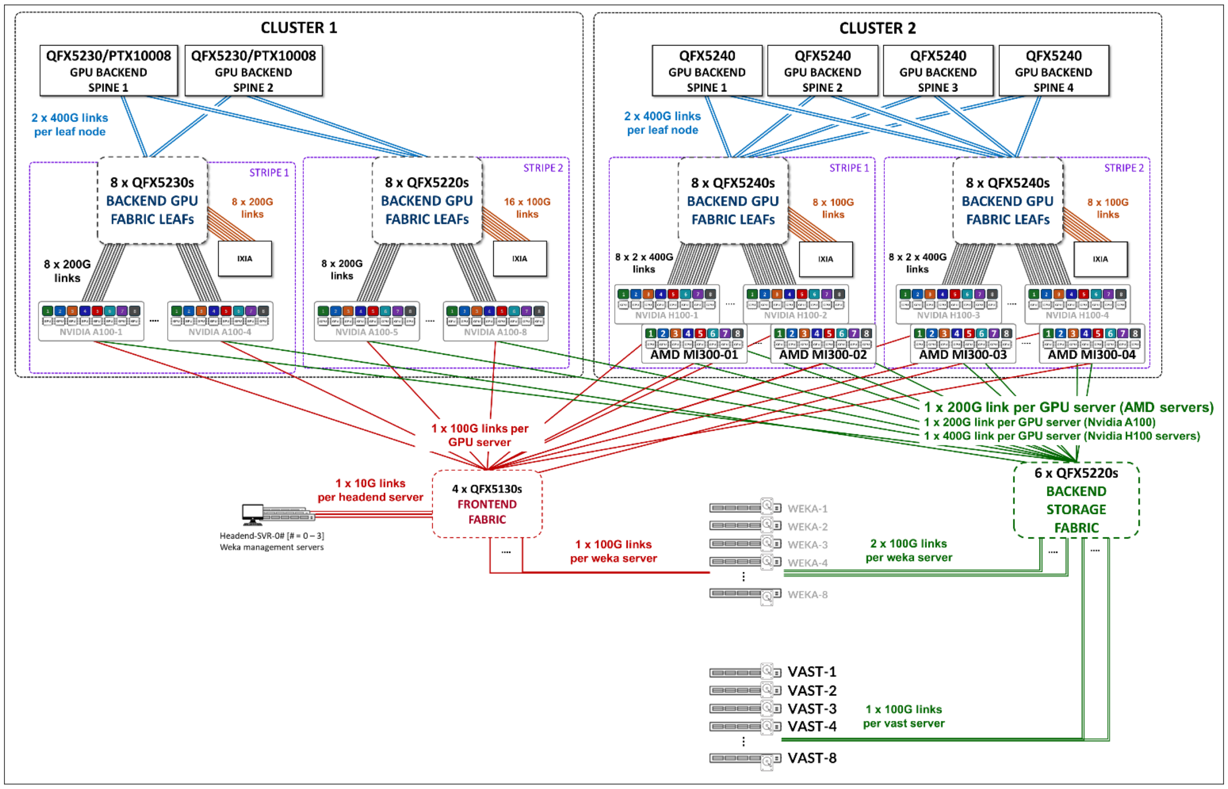
The GPU Backend fabric provides the infrastructure for GPUs to communicate with each other within a cluster, using RDMA over Converged Ethernet (RoCEv2). RoCEv2 enhances data center efficiency, reduces complexity, and optimizes data delivery across high-speed Ethernet networks.

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.

We have built two different Clusters, as shown in Figure 3, which share the " **Frontend fabric** " on page 9 and Storage Backend fabric but have separate " **GPU Backend fabrics** " on page 8. Each cluster is made of two stripes following the Rail Optimized Stripe Architecture , but include different switch models as Leaf and Spine nodes, as well as different GPU server models.

Figure 3: AI JVD Lab Clusters


CLUSTER 1
QFX5230/PTX10008 GPU BACKEND SPINE 1
QFX5230/PTX10008 GPU BACKEND SPINE 2
2 x 400G links per leaf node
8 x QFX5230s BACKEND GPU FABRIC LEAFs
STRIPE 1
8 x 200G links
IXIA
8 x 200G links
NVIDIA A100-1
NVIDIA A100-4
8 x QFX5220s BACKEND GPU FABRIC LEAFs
STRIPE 2
16 x 100G links
8 x 200G links
NVIDIA A100-5
NVIDIA A100-8
CLUSTER 2
QFX5240 GPU BACKEND SPINE 1
QFX5240 GPU BACKEND SPINE 2
QFX5240 GPU BACKEND SPINE 3
QFX5240 GPU BACKEND SPINE 4
2 x 400G links per leaf node
8 x QFX5240s BACKEND GPU FABRIC LEAFs
8 x 100G links
8 x 2 x 400G links
NVIDIA H100-1
NVIDIA H100-2
NVIDIA H100-3
NVIDIA H100-4
AMD MI300-01
AMD MI300-02
AMD MI300-03
AMD MI300-04
1 x 200G link per GPU server (AMD servers)
1 x 200G link per GPU server (Nvidia A100)
1 x 400G link per GPU server (Nvidia H100 servers)
1 x 100G links per GPU server
1 x 10G links per headend server
4 x QFX5130s FRONTEND FABRIC
Headend-SVR-0# [# = 0 – 3] Weka management servers
1 x 100G links per weka server
WEKA-1
WEKA-2
WEKA-3
WEKA-4
WEKA-8
2 x 100G links per weka server
6 x QFX5220s BACKEND STORAGE FABRIC
VAST-1
VAST-2
VAST-3
VAST-4
VAST-8
1 x 100G links per vast server

The **GPU Backend in Cluster 1** consists of Juniper **QFX5220** and **QFX5230** switches as leaf nodes, and either **QFX5230** switches or **PTX10008** routers as spine nodes, along with **NVIDIA A100** GPU servers. **QFX5230** and **PTX10008** devices have been validated independently as spine nodes while maintaining the same leaf configuration. The **GPU backend fabric** in this cluster follows a 3-stage Clos **IP fabric** architecture. Further details are available in the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage—Juniper Validated Design (JVD).

The **GPU Backend in Cluster 2** consists of Juniper **QFX5240** switches acting as both leaf and spine nodes, along with **AMD MI300X** and **NVIDIA H100** GPU servers. This cluster supports either a 3-stage IP fabric architecture or a 3-stage **EVPN/VXLAN fabric** architecture. Further details about the **IP Fabric** implementation are available in the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage—Juniper Validated Design (JVD).

The **EVPN/VXLAN-based** implementation is the focus of this document.

Solution Implementation

IN THIS SECTION

Frontend Fabric

For details about how to connect and deploy the Frontend Fabric, refer to the Vast Storage Configuration section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage: Frontend Fabric—Juniper Validated Design (JVD).

Storage Backend Fabric

For details about how to connect and deploy the Storage Fabric, refer to the Vast Storage Configuration section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage: Storage Backend Fabric—Juniper Validated Design (JVD).

Note that the **Frontend and Storage Backend fabrics** are not covered in detail here, as they remain unchanged and are fully documented in the JVDs referenced above.

GPU Backend Fabric

The remainder of this document will focus on the GPU Backend fabric implementation using the EVPN/VXLAN architecture.

EVPN/VXLAN GPU Backend Fabric – GPU Multitenancy

IN THIS SECTION

GPU Multitenancy (GPU as a Service – GPUaaS)

GPU as a Service (GPUaaS) is a model where GPU compute resources are provided on demand to users or applications, similar to other utility-style computing services. Rather than dedicating entire servers or clusters to a single team or purpose, GPUaaS allows resources to be dynamically allocated based on current workload requirements. Tenants can request specific numbers of GPUs, often across multiple servers, and use them for tasks such as AI training, data analytics, or visualization. The service abstracts the underlying infrastructure, providing users with a seamless and scalable experience while maintaining secure and efficient resource isolation. By combining flexibility with centralized management, GPUaaS enables better resource utilization and simplifies operations in environments where multiple teams or projects share the same data center.

GPU multitenancy is a resource management approach that allows multiple tenants to use GPU resources independently within a shared infrastructure. Instead of assigning all the GPUs in a server to a single tenant, GPU multitenancy enables more flexible allocation, where one or more GPUs on a server can be reserved for different tenants. This model improves efficiency by allowing organizations to match GPU resources to the specific needs of each workload, rather than over-provisioning entire servers. Each tenant operates in a logically isolated environment, with clear separation of compute resources, network paths, and associated configurations. This isolation ensures that tenants can run their applications without interference, while administrators maintain centralized control over GPU distribution and access.

GPU multitenancy and GPU as a Service (GPUaaS) are closely related concepts that, when combined, enable efficient and scalable use of GPU infrastructure in multi-tenant environments. GPU multitenancy provides the foundation by allowing GPU resources to be flexibly assigned to different tenants at a granular level, whether one GPU, several GPUs, or specific GPUs across different servers. This approach ensures that each tenant operates in a logically isolated environment, maintaining security and performance consistency even when physical infrastructure is shared.

Building on this, GPUaaS abstracts these capabilities into an on-demand service model. Instead of requiring users to manage physical servers or hardware configurations, GPUaaS delivers GPU resources dynamically as needed. It leverages the underlying multitenancy framework to allocate GPUs based on user requests, enforce isolation, and optimize usage across a diverse set of workloads. This allows data centers to support a wide range of teams or applications concurrently, without dedicating entire servers to each one.

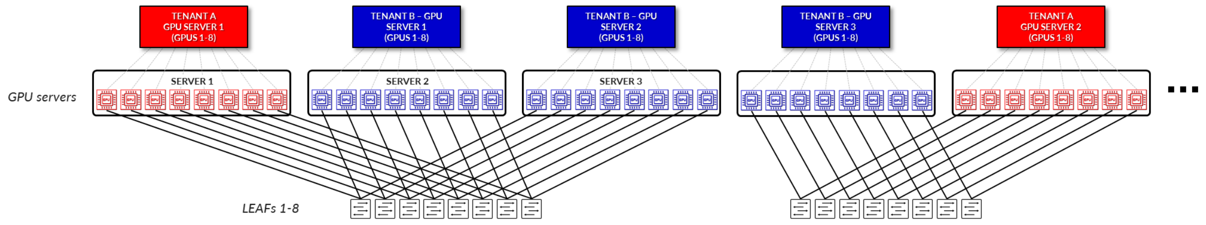
Together, GPU multitenancy and GPUaaS enable high efficiency, better resource utilization, and operational simplicity. While multitenancy handles the secure and flexible slicing of GPU resources, GPUaaS delivers these slices as consumable services, scaling compute capacity up or down as needed, and making GPU-powered computing more accessible and cost-effective for varied use cases.

Types of GPU multitenancy

SERVER ISOLATION:

In a server isolation model, each tenant is allocated one or more entire servers. All GPUs within those servers are exclusively dedicated to a single tenant, ensuring full physical and logical separation from other tenants. This model simplifies resource allocation and minimizes the risk of cross-tenant interference, making it well suited for workloads that require predictable performance and strict isolation. (Figure 4).

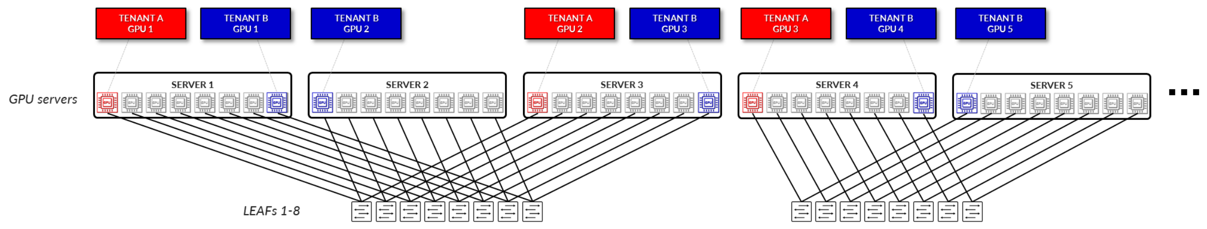
Figure 4: GPU as a Service – Server Isolation


TENANT A GPU SERVER 1 (GPUS 1-8)
TENANT B – GPU SERVER 1 (GPUS 1-8)
TENANT B – GPU SERVER 2 (GPUS 1-8)
TENANT B – GPU SERVER 3 (GPUS 1-8)
TENANT A GPU SERVER 2 (GPUS 1-8)
SERVER 1
SERVER 2
SERVER 3
GPU servers
...
LEAFs 1-8

GPU ISOLATION:

In a GPU isolation model, individual GPUs within a server are assigned to different tenants. This allows multiple tenants to securely share the same physical server, with each tenant accessing only the GPUs allocated to them. The underlying fabric provides logical separation and guarantees isolation at the GPU level, enabling greater flexibility and higher utilization of resources without compromising security or performance. (Figure 5).

Figure 5: GPU as a Service – GPU Isolation



GPU Backend Fabric for Multitenancy Architecture

The design of the GPU Backend Fabric for Multitenancy follows a 3-stage Clos, rail-optimized stripe architecture using EVPN/VXLAN. This approach enables high-performance communication between GPUs assigned to the same tenant while ensuring traffic isolation between tenants, for both Server Isolation and GPU Isolation. For more information on server isolation and GPU isolation, see "Rail Alignment and Local Optimization Considerations with GPU multitenancy" on page 27.

Figure 6: GPU Backend Fabric Architecture

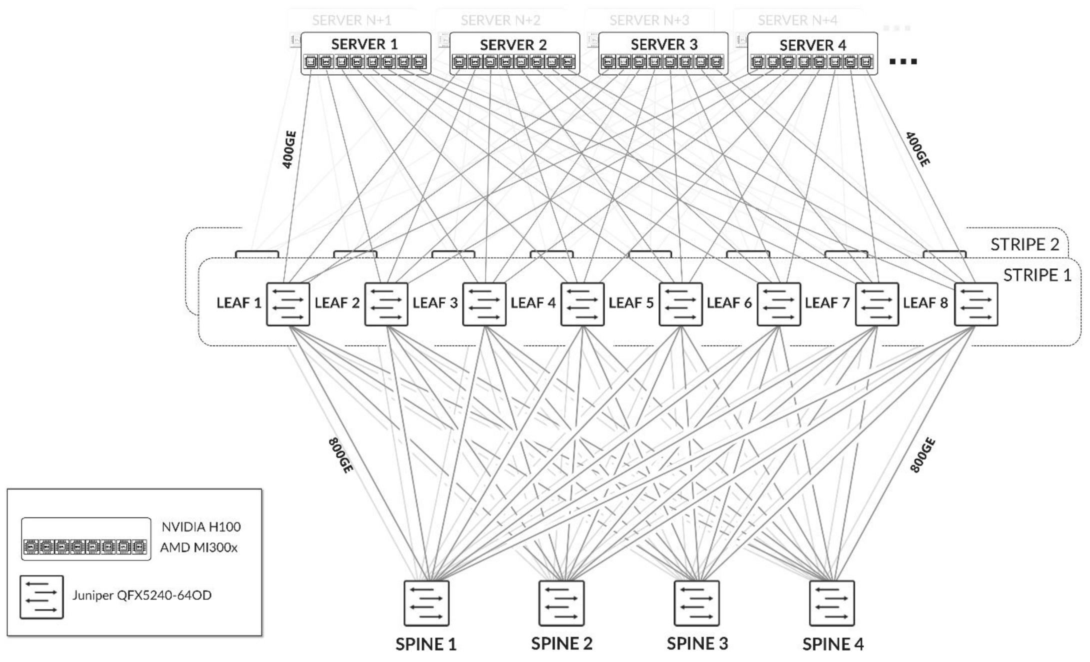


Figure 7: GPU Backend Fabric EVPN/VXLAN connectivity – Server Isolation

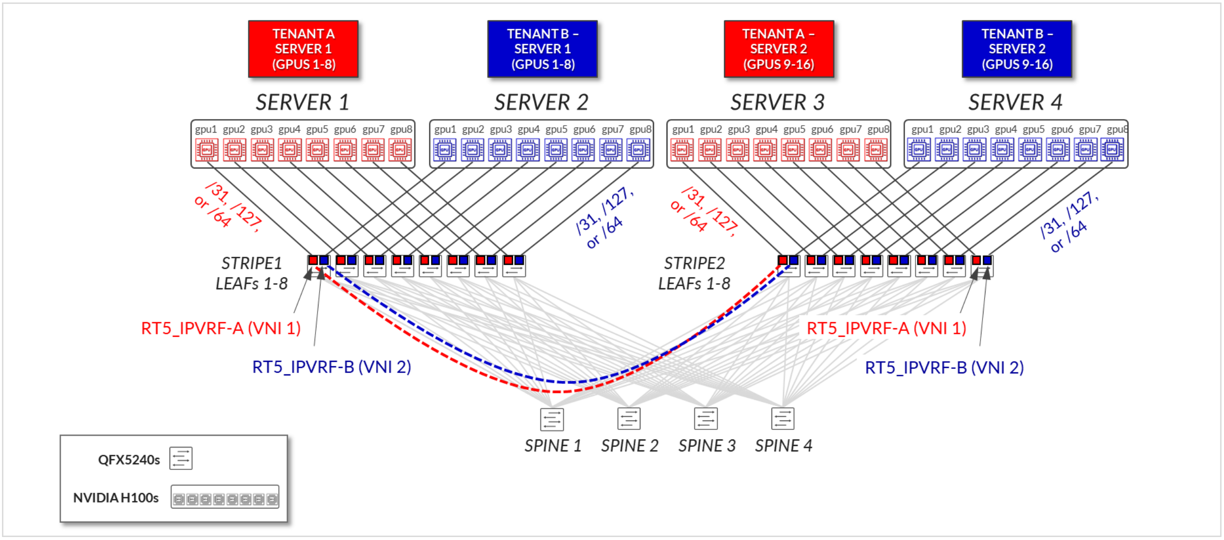
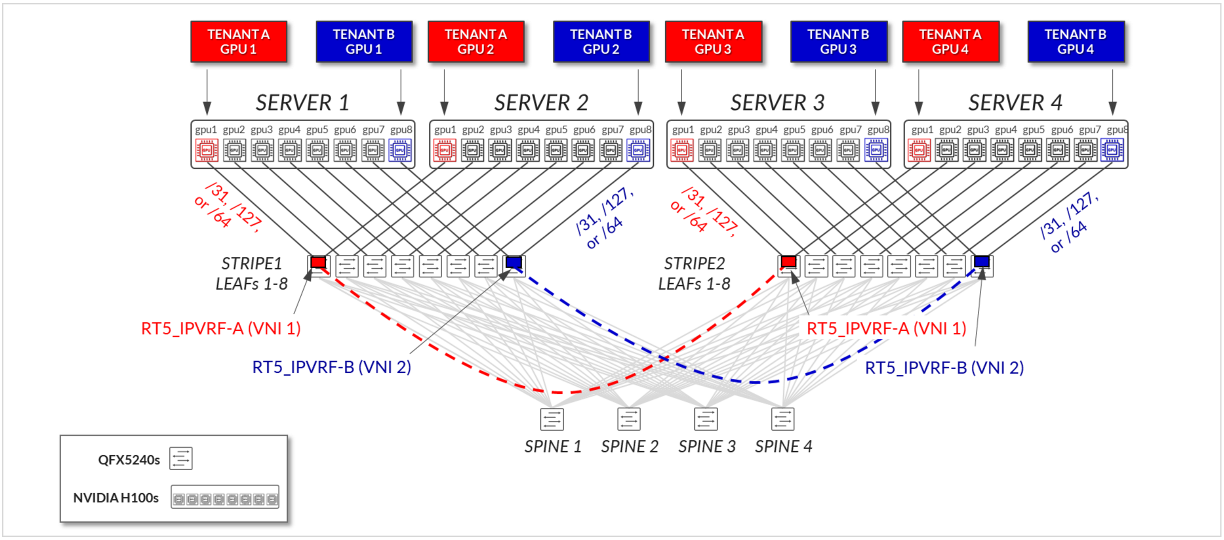


Figure 8: GPU Backend Fabric EVPN/VXLAN Connectivity – GPU Isolation



The devices that are part of the GPU Backend fabric in the AI Lab, and the connections between them, are summarized in Table 1 and Table 2:

Table 1: GPU Backend devices per Stripe

Stripe	GPU Servers	GPU Backend Leaf nodes switch model	GPU Backend Spine nodes switch model
1	MI300X x 2 (MI300X-01 & MI300X-02) H100 x 2 (H100-01 & H100-02)	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) H100 x 2 (H100-01 & H100-02)	QFX5240-64OD x 8 (gpu-backend-002_leaf#; #=1-8)	

All the Nvidia H100 and AMD MI300X GPU servers are connected to the GPU backend fabric using 400GE interfaces.

Table 2: 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	Total number of 400GE links between servers and leaf nodes = 8 (number of GPUs per server) x 1 (number of 400GE server to leaf links) x 4 (number of servers) = 32	Total number of 400GE links between GPU backend leaf nodes and spine nodes = 8 (number of leaf nodes) x 2 (number of 400GE links per leaf to spine connection) x 4 (number of spine nodes) = 64

(Continued)

Stripe	GPU Servers <=> GPU Backend Leaf Nodes	GPU Backend Leaf Nodes <=> GPU Backend Spine Nodes
2	Total number of 400GE links between servers and leaf nodes = 8 (number of GPUs per server) x 1 (number of 400GE server to leaf links) x 4 (number of servers) = 32	Total number of 400GE links between GPU backend leaf nodes and spine nodes = 8 (number of leaf nodes) x 2 (number of 400GE links per leaf to spine connection) x 4 (number of spine nodes) = 64

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 25.6 Tbps (Table 3), while the bandwidth available between the leaf and spine nodes is also 51.2 Tbps (Table 4). 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 3: Per stripe Server to Leaf Bandwidth

Server to Leaf Bandwidth per Stripe				
Stripe	Number of servers per Stripe	Number of 400 GE 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	4	8	400 Gbps	4 x 8 x 400 Gbps = 12.8 Tbps
2	4	8	400 Gbps	4 x 8 x 400 Gbps = 12.8 Tbps

(Continued)

Server to Leaf Bandwidth per Stripe				
			Total **Server <=> Leaf Bandwidth**	25.6 Tbps

Table 4: 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 800 GE leaf ó spine links per leaf node	Server <=> Leaf Link Bandwidth [Gbps]	Bandwidth Leaf <=> Spine Per Stripe [Tbps]
1	8	4	1	800 Gbps	8 x 4 x 1 x 800 Gbps = 25.6 Tbps
2	8	4	1	800 Gbps	8 x 4 x 1 x 400 Gbps = 25.6 Tbps
				Total **Leaf <=> Spine Bandwidth**	51.2 Tbps

GPU to leaf nodes connectivity follows the Rail-optimized architecture as described in Backend GPU Rail Optimized Stripe Architecture.

Backend GPU Rail Optimized Stripe Architecture

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

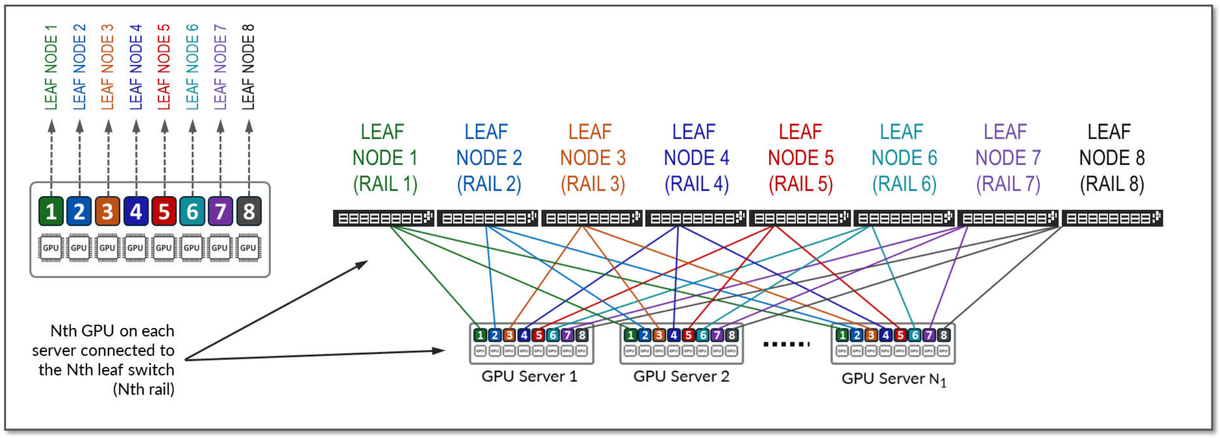
Optimized topology aims to maximize performance by providing minimal bandwidth contention, minimal latency, and minimal network interference, to provide this efficient data transfer.

In a Rail Optimized Stripe 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 9.

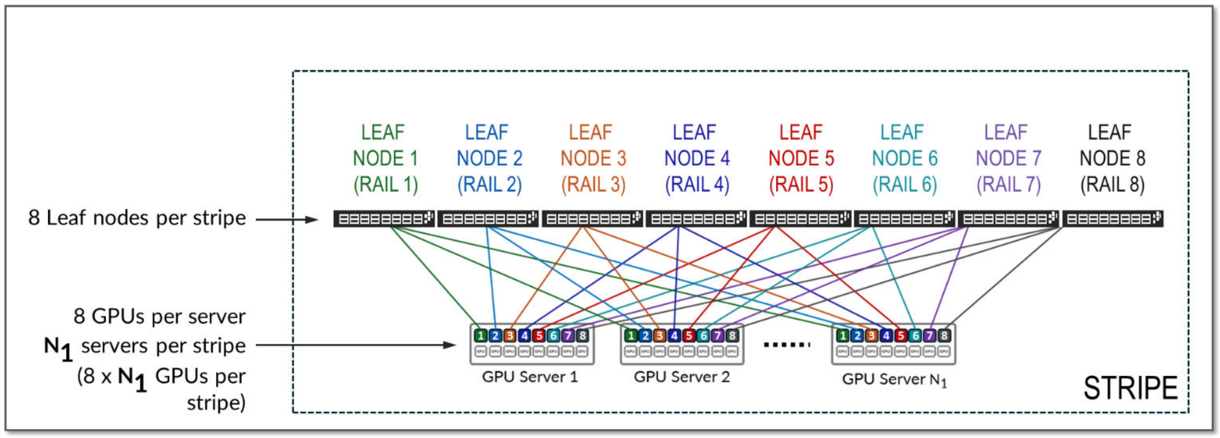
A rail connects GPUs of the same order across one of the leaf nodes in the fabric; that is, rail N connects GPUs in position N in all the servers to leaf node N.

Figure 9: Rails in a Rail Optimized Architecture



A **stripe** refers to a design module or building block consisting of a group of Leaf nodes and GPU servers, as shown in Figure 10. This module can be replicated to scale up the AI cluster.

Figure 10: Stripes in a Rail Optimized Architecture



The number of leaf nodes in a single stripe, and thus the number of rails in a single stripe, is always defined by the number of GPUs per server. Each GPU server typically includes 8 GPUs. Therefore, a single stripe typically includes 8 leaf nodes (8 rails).

In a rail optimized architecture, the maximum number of servers supported in a single stripe (N1 in Figure 7) is limited by the number and the speed of the interfaces supported by the Leaf node switch model. This is because 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, which is ideal.

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 total interfaces on each leaf node. Some examples are included in Table 5.

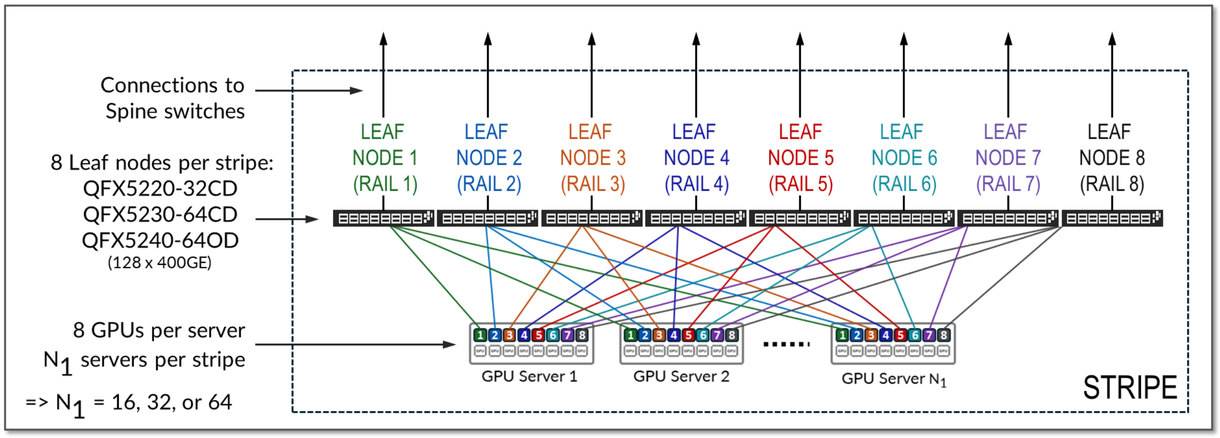
Table 5: Maximum number of GPUs supported per stripe

Leaf Node QFX switch Model	Maximum number of 400 GE interfaces per switch	Maximum number of servers supported per stripe (1:1 Subscription)	GPUs per server	Maximum number of GPUs supported per stripe
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). See Figure 11.

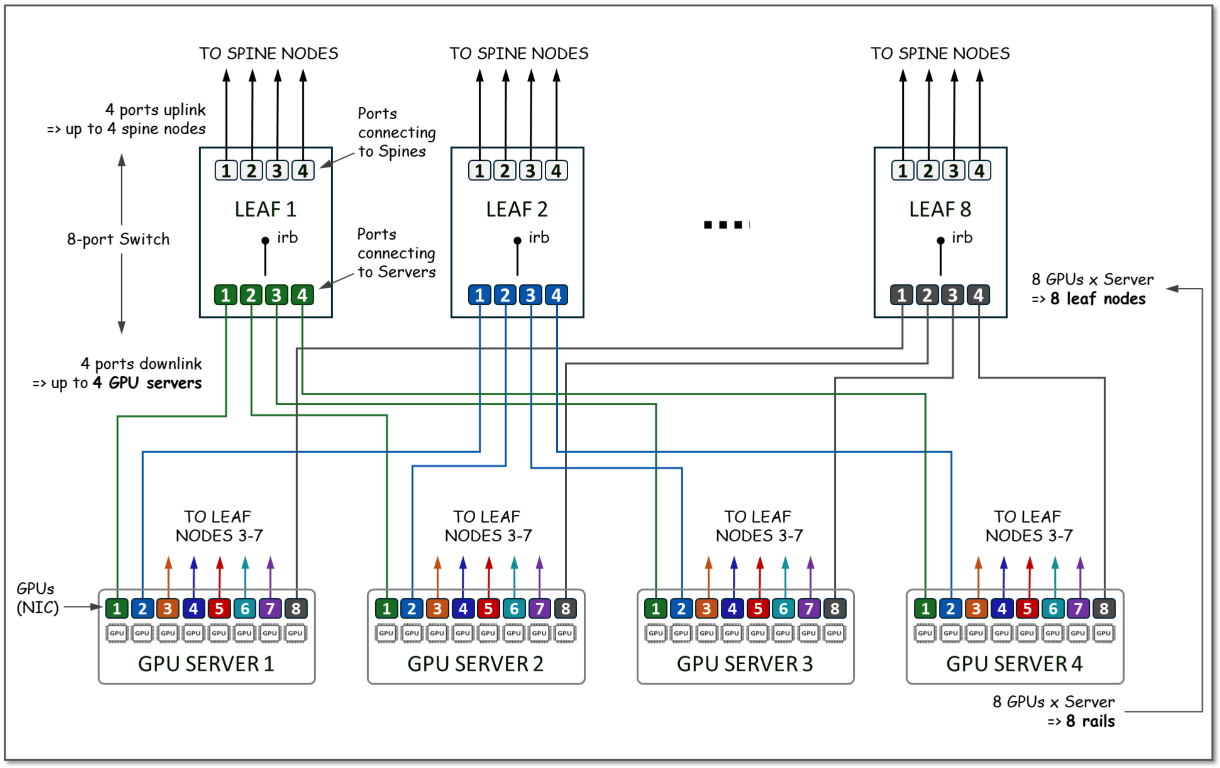
NOTE: QFX5240-64OD switches come with 64 x 800GE ports which can break out into 2x400GE ports, for a maximum of 128 400GE interfaces was shown in table 5.

Figure 11: Maximum number of Servers per Stripes in a Rail Optimized Architecture.



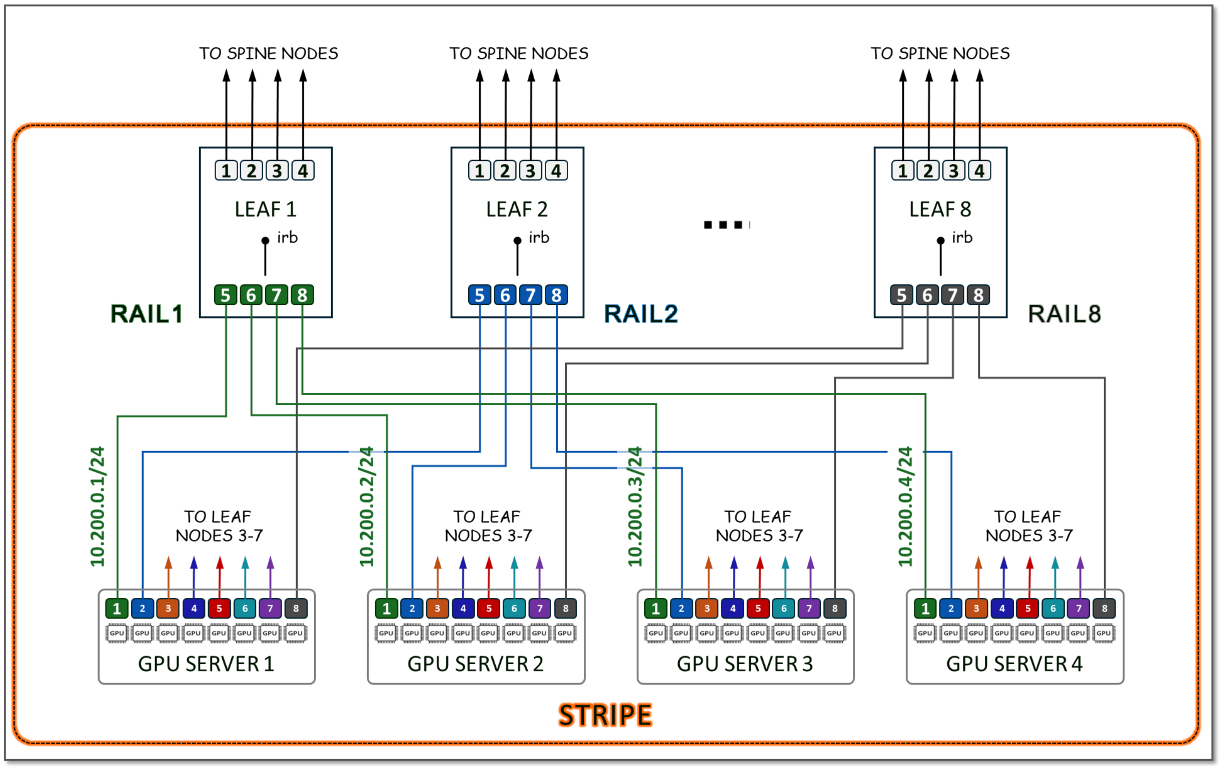
As an example of how to calculate the number of servers supported, and to reinforce the concepts of rail and stripe, consider a hypothetical switch with only 8 ports of the same speed, and GPU servers with 8 GPUs each, as shown in Figure 12.

Figure 12. Number of Servers Supported by 8-Port Switches as Leaf Nodes Example.


TO SPINE NODES
TO SPINE NODES
TO SPINE NODES
4 ports uplink => up to 4 spine nodes
Ports connecting to Spines
LEAF 1
LEAF 2
LEAF 8
irb
8-port Switch
Ports connecting to Servers
8 GPUs x Server => 8 leaf nodes
4 ports downlink => up to 4 GPU servers
TO LEAF NODES 3-7
GPUs (NIC)
GPU SERVER 1
GPU SERVER 2
GPU SERVER 3
GPU SERVER 4
8 GPUs x Server => 8 rails

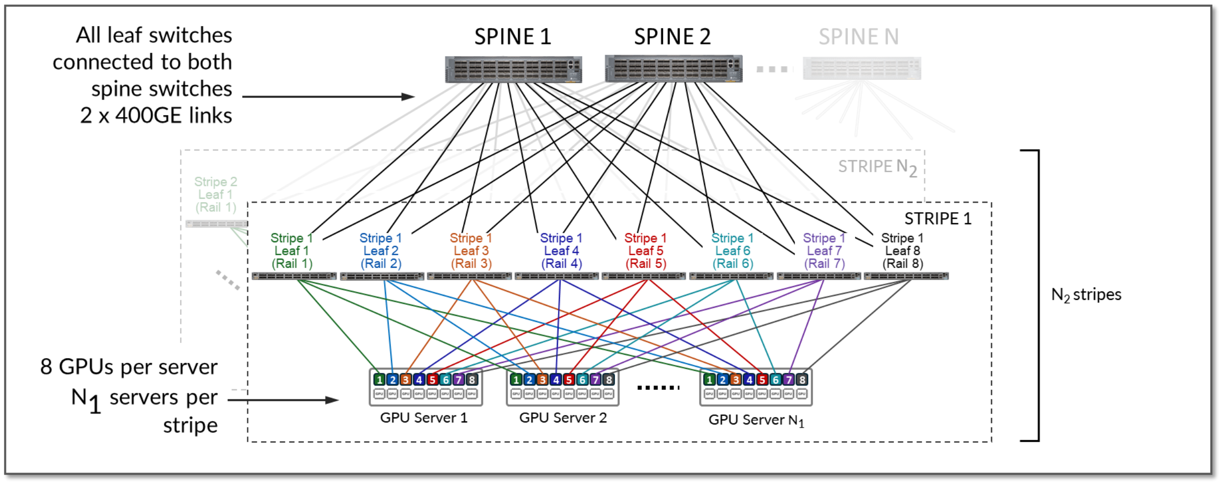
Because the GPU servers have 8 GPUs, the number of Leaf nodes will be 8. On each leaf node, 4 ports will be used to connect to the Spine Nodes (for scaling purposes as described in the next section), and 4 ports will be used to connect to the GPU servers. All the GPU numbered 1, will be connected to Leaf node 1, all the GPUs numbered 2 will be connected to Leaf node 2, and so on, with each group representing a RAIL (8 RAILS total), and the group of all 4 servers, and 8 switches together represent a STRIPE (with a total of 32 GPUs), as shown in Figure 13.

Figure 13. Stripe and Rails with 8 leaf (8-port switch) Nodes Example



To achieve larger scales, multiple stripes can be implemented. The stripes are connected using Spine switches which provide inter-stripe connectivity, as shown in Figure 14.

Figure 14: Multiple Stripes Connected via Spine Nodes



For example, assume that the desired number of GPUs is 16,000 and the fabric is using either QFX5230-64CD or QFX5240-64OD as leaf nodes:

- The QFX5240-64OD leaf nodes support up to 128 x 400Gbps ports

- The **Maximum Number of Servers Per Stripe (N_1)** is calculated by dividing the number of ports supported by the leaf node.

$N_1 = 128 \div 2 = 64$

- The **Maximum Number of GPUs supported per stripe** is calculated by multiplying the maximum number of servers per stripe (N_1) by the numbers of GPUs on each server:

$N_1 \times 8 = 64 \times 8 = 512$

- The **Required Number of Stripes (N_2)** is calculated by dividing the required number of GPUs by the maximum number of GPUs supported per stripe:

$N_2 = 16000/512 \approx 31.25$ stripes (rounded up to 32)

NOTE: With $N_2 = 64$ stripes & N_1 servers = 32, the cluster can provide 16,384 GPUs.If N_2 is increased to 72 & N_1 servers = 32, the cluster can provide 18432 GPUs.

The **stripes** in the AI JVD setup consist of wither 8 Juniper QFX5220-32CD, QFX5230-64CD or QFX5240-64OD switches depending on the cluster and stripe, as summarized in Table 6.

Table 6. Maximum number of GPUs supported per cluster in the JVD lab

Cluster	Stripe	Leaf Node QFX model	Maximum number of GPUs supported per stripe
1	1	QFX5230-64CD	32 servers x 8 GPUs/server = 256 GPUs
1	2	QFX5220-32CD	16 servers x 8 GPUs/server = 128 GPUs
Total number of GPUs supported by the cluster = 384 GPUs			
2	1	QFX5240-64OD	64 servers x 8 GPUs/server = 512 GPUs
2	2	QFX5240-64OD	64 servers x 8 GPUs/server = 512 GPUs
Total number of GPUs supported by the cluster = 1024 GPUs			

Local Optimization

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 same server when possible, the need to traverse spines or external links is reduced. This 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 on the same servers can be forwarded locally across the internal Server fabric (server architecture dependent). Traffic between GPUs in different servers happens across the GPU backend infrastructure, either within the same rail (intra-rail), or in different rails (inter-rail/inter-stripe).

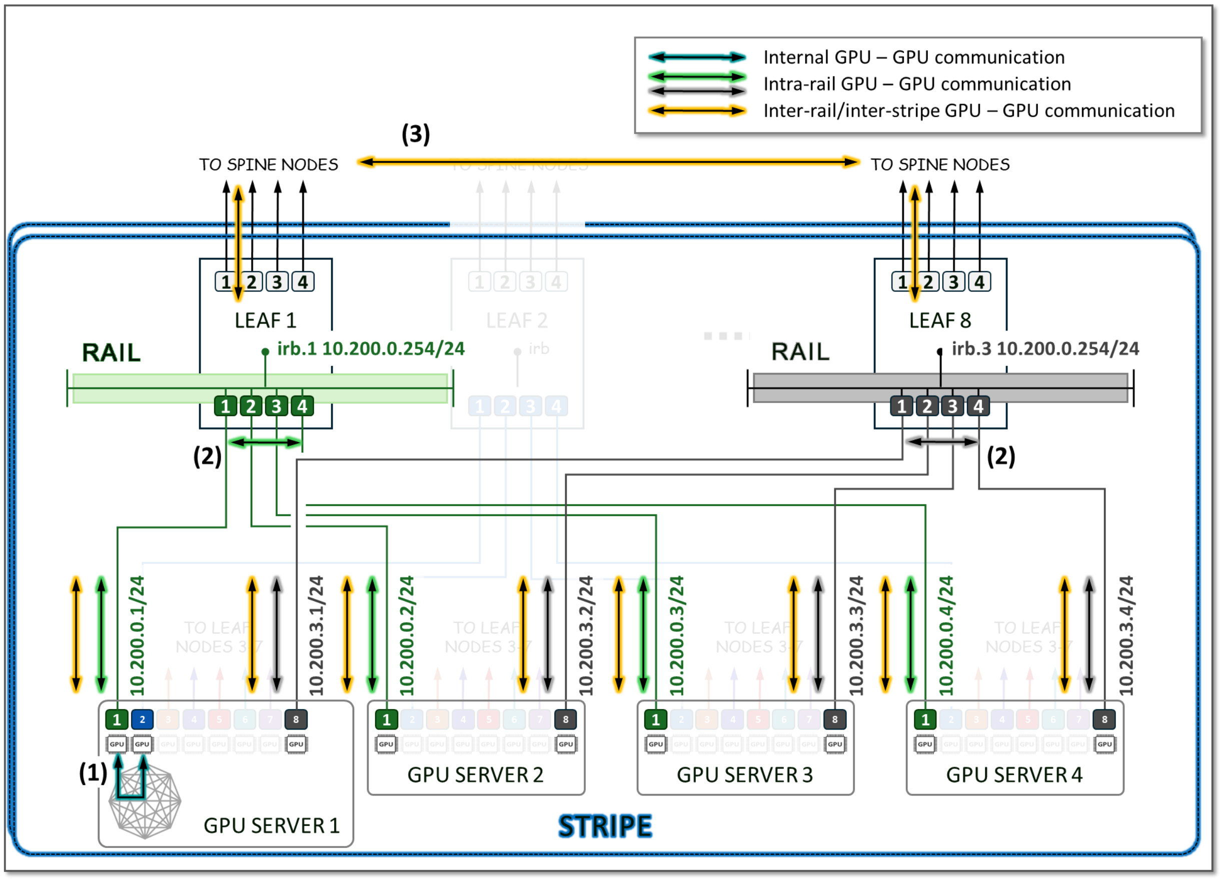
Intra-rail traffic is processed at the local leaf node. Following this design, data between GPUs on different servers (but in the same stripe) is always moved on the same rail and across one single switch, while data between GPUs on different rails needs to be forwarded across the spines.

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 GPU 1 in servers 1- 4, and between GPU 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 15.

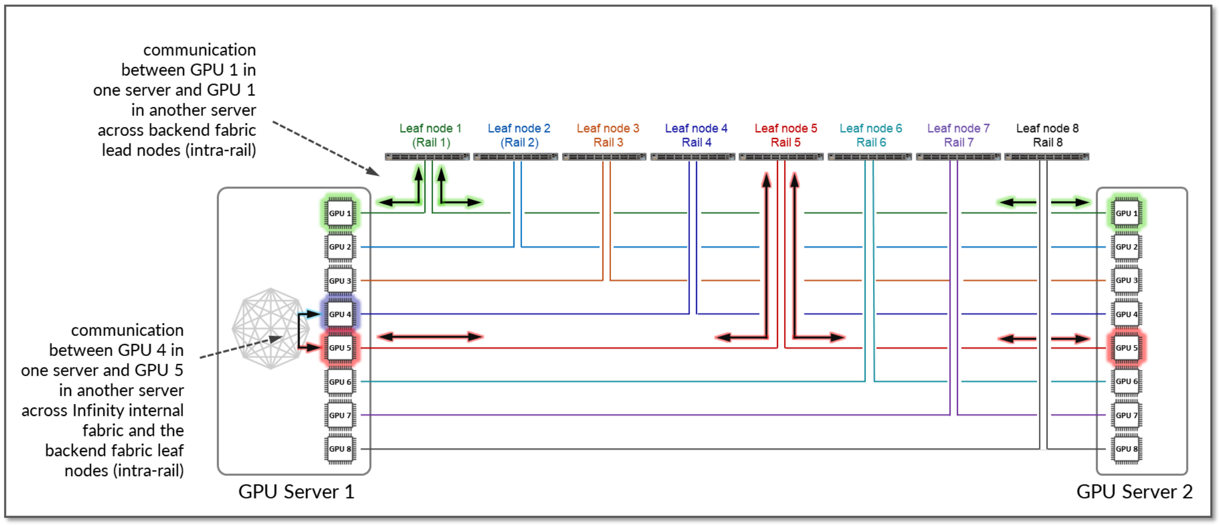
Figure 15: Inter-Rail vs. Intra-Rail GPU-GPU Communication



Most vendors implement **local optimization** to minimize latency for GPU-to-GPU traffic. Traffic between GPUs of the same number remains intra-rail. Figure 16 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.

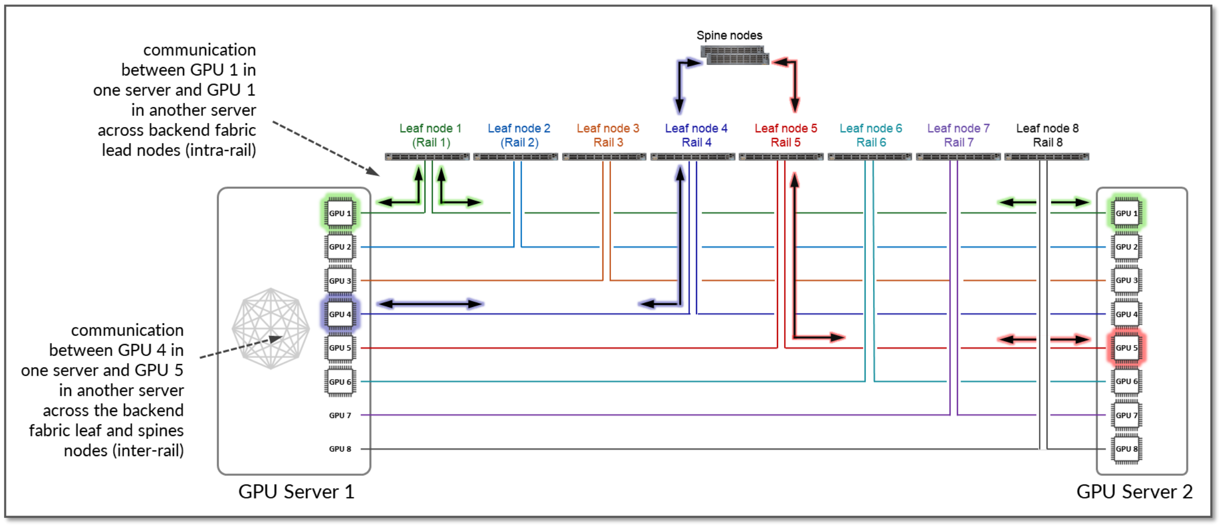
Additionally, a NCCL feature known as PXN can be enabled to leverage internal fabric connectivity between GPUs within a server, where data is first moved to a GPU on the same rail as the destination, then send it to the destination without crossing rails. For example, if GPU4 in Server 1 wants to communicate with GPU5 in Server 2, and GPU5 in Server 1 is available across the internal fabric, the traffic naturally prefers this path to optimize performance and keep GPU-to-GPU communication intra-rail.

Figure 16: GPU to GPU Inter-Rail Communication Between Two Servers with PXN.


communication between GPU 1 in one server and GPU 1 in another server across backend fabric lead nodes (intra-rail)
communication between GPU 4 in one server and GPU 5 in another server across Infinity internal fabric and the backend fabric leaf nodes (intra-rail)
Leaf node 1 (Rail 1)
Leaf node 2 (Rail 2)
Leaf node 3 Rail 3
Leaf node 4 Rail 4
Leaf node 5 Rail 5
Leaf node 6 Rail 6
Leaf node 7 Rail 7
Leaf node 8 Rail 8
GPU 1
GPU 2
GPU 3
GPU 4
GPU 5
GPU 6
GPU 7
GPU 8
GPU Server 1
GPU Server 2

If **this path** is not feasible because of workload or service constraints, or because PXN is disabled the traffic will 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 17.

Figure 17: GPU to GPU Inter-Rail Communication Between Two Servers without PXN.


communication between GPU 1 in one server and GPU 1 in another server across backend fabric lead nodes (intra-rail)
communication between GPU 4 in one server and GPU 5 in another server across the backend fabric leaf and spines nodes (inter-rail)
Spine nodes
Leaf node 1 (Rail 1)
Leaf node 2 (Rail 2)
Leaf node 3 Rail 3
Leaf node 4 Rail 4
Leaf node 5 Rail 5
Leaf node 6 Rail 6
Leaf node 7 Rail 7
Leaf node 8 Rail 8
GPU 1
GPU 2
GPU 3
GPU 4
GPU 5
GPU 6
GPU 7
GPU 8
GPU Server 1
GPU Server 2

While PXN is a NCCL (NVIDIA Collective Communication Library) it is also supported by AMDs ROCm Communication Collectives Library. To enable or disable PXN use the variable NCCL_PXN_DISABLE

Rail Alignment and Local Optimization Considerations with GPU multitenancy

When implementing multitenancy in GPU fabrics, additional considerations apply regarding how GPUs are assigned and how communication between GPUs is handled.

Server Isolation model

In the **server-isolation model**, all GPUs in a server are dedicated to a single tenant. In this model, direct communication between GPUs within the same server is both appropriate and desirable. Placing the network interfaces connecting servers assigned to different tenants into different VRFs on the leaf nodes is sufficient to keep tenants separated across the network, but GPU-to-GPU communication also needs to be consider. Local optimization ensures that GPU-to-GPU communication follows the most optimal internal path:

- • GPUs within the same server communicate using the server's internal mechanisms.
  - GPUs in different servers but connected to the same stripe can communicate across leaf nodes.
  - GPUs located in servers that connect to different stripes communicate through the spine layer, where traffic is encapsulated in VXLAN and routed across the EVPN/VXLAN fabric.

NOTE: The examples in this section show possible paths for data between GPUs. The actual path depends on collectives (All-Gather, All-Reduce, All-To-All, etc) and topology algorithm (ring, tree, etc.) selected. Also, when a job runs there might be multiple topologies at the same time (e.g. multiple rings) following different path, built to increase efficiency. The actual path can be found in the slurm logs as shown in the example:

•
```
jnpr@headend-svr-1:/mnt/nfsshare/logs/nccl/H100-RAILS-ALL/06102025_19_35_46$ cat
slurm-25432.out | egrep Channel

H100-01:3179628:3180857 [0] NCCL INFO Channel 00/16 :    0   1   2   3   4   5   6   7   8
9  10  11  12  13  14  15

H100-01:3179628:3180857 [0] NCCL INFO Channel 01/16 :    0   3   2   9  15  14  13  12   8
11  10   1   7   6   5   4

H100-01:3179628:3180857 [0] NCCL INFO Channel 02/16 :    0   3  10  15  14  13  12   9   8
11   2   7   6   5   4   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 03/16 :    0  11  15  14  13  12  10   9   8
3   7   6   5   4   2   1
```

```
H100-01:3179628:3180857 [0] NCCL INFO Channel 04/16 :    0   7   6   5  12  11  10   9   8
15  14  13   4   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 05/16 :    0   4   7   6  13  11  10   9   8
12  15  14   5   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 06/16 :    0   5   4   7  14  11  10   9   8
13  12  15   6   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 07/16 :    0   6   5   4  15  11  10   9   8
14  13  12   7   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 08/16 :    0   1   2   3   4   5   6   7   8
9  10  11  12  13  14  15

H100-01:3179628:3180857 [0] NCCL INFO Channel 09/16 :    0   3   2   9  15  14  13  12   8
11  10   1   7   6   5   4

H100-01:3179628:3180857 [0] NCCL INFO Channel 10/16 :    0   3  10  15  14  13  12   9   8
11   2   7   6   5   4   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 11/16 :    0  11  15  14  13  12  10   9   8
3   7   6   5   4   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 12/16 :    0   7   6   5  12  11  10   9   8
15  14  13   4   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 13/16 :    0   4   7   6  13  11  10   9   8
12  15  14   5   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 14/16 :    0   5   4   7  14  11  10   9   8
13  12  15   6   3   2   1

H100-01:3179628:3180857 [0] NCCL INFO Channel 15/16 :    0   6   5   4  15  11  10   9   8
14  13  12   7   3   2   1

H100-02:2723777:2725118 [2] NCCL INFO Channel 00/0 : 10[2] -> 11[3] via P2P/IPC

H100-02:2723779:2725122 [4] NCCL INFO Channel 00/0 : 12[4] -> 13[5] via P2P/IPC

H100-02:2723778:2725124 [3] NCCL INFO Channel 00/0 : 11[3] -> 12[4] via P2P/IPC
```

```
H100-02:2723780:2725121 [5] NCCL INFO Channel 00/0 : 13[5] -> 14[6] via P2P/IPC

H100-02:2723781:2725125 [6] NCCL INFO Channel 00/0 : 14[6] -> 15[7] via P2P/IPC

H100-02:2723776:2725123 [1] NCCL INFO Channel 00/0 : 9[1] -> 10[2] via P2P/IPC

H100-02:2723777:2725118 [2] NCCL INFO Channel 08/0 : 10[2] -> 11[3] via P2P/IPC

H100-02:2723775:2725119 [0] NCCL INFO Channel 00/0 : 7[7] -> 8[0] [receive] via NET/IBext/0/
GDRDMA

H100-02:2723779:2725122 [4] NCCL INFO Channel 08/0 : 12[4] -> 13[5] via P2P/IPC

H100-02:2723780:2725121 [5] NCCL INFO Channel 08/0 : 13[5] -> 14[6] via P2P/IPC

H100-02:2723782:2725120 [7] NCCL INFO Channel 00/0 : 15[7] -> 0[0] [send] via NET/IBext/0(8)/
GDRDMA

H100-02:2723775:2725119 [0] NCCL INFO Channel 08/0 : 7[7] -> 8[0] [receive] via NET/IBext/0/
GDRDMA

H100-02:2723775:2725119 [0] NCCL INFO Channel 00/0 : 8[0] -> 9[1] via P2P/IPC

--more---
```

where:

X[Y] -> A[B]:

- X Source GPU global index.
- Y Local GPU index (within the node).
- A Destination GPU global index.
- B Local GPU index.

[send] / [receive]: Direction from the perspective of the process writing the log.

NET/IBext/N or NET/IBext/N(P):

- N=InfiniBand interface index (N)
- P (in parentheses) = NIC port or peer rank.

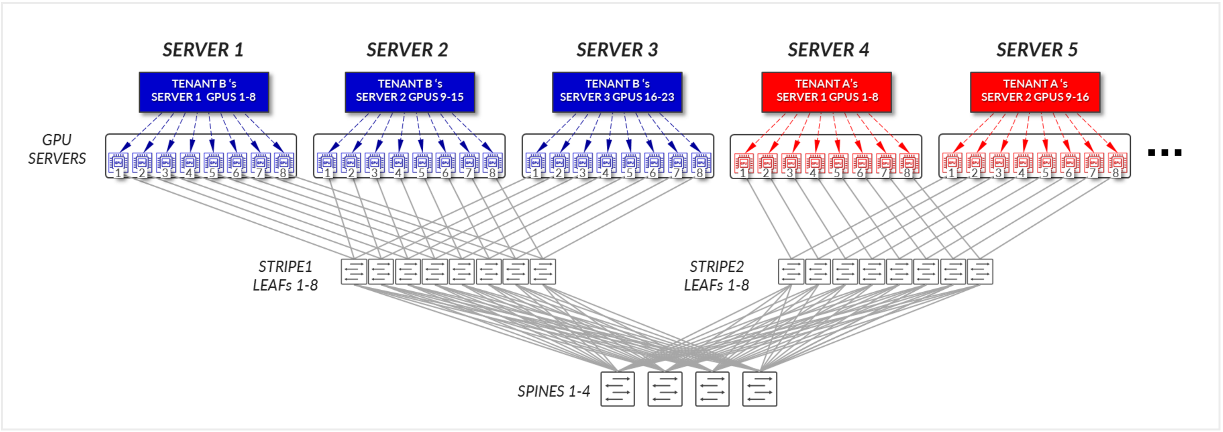
GDRDMA: GPUDirect RDMA, which means data goes directly between GPUs' memory over RDMA-capable NICs without CPU involvement. This is optimal for latency and bandwidth. Enables direct data exchange between the GPU and a third-party peer device using standard features of PCI Express. It is based on a kernel module called nv_peer_mem, which allows Mellanox and other RDMA-enabled NICs to directly read and write CUDA memory using NIC RDMA paths. NCCL provides routines optimized for high bandwidth and low latency over PCIe, NVLink, and NVIDIA Mellanox Network.

P2P/IPC: Point-to-Point (P2P) transport in the NVIDIA Collective Communications Library (NCCL). It enables GPUs to communicate directly with each other without going through the host CPU or network. NCCL provides inter-GPU communication primitives that are topology-aware and can be easily integrated into applications.

Example 1

Consider the example depicted in Figure 18, where Tenant A has been assigned SERVERS 4 and SERVER 5, in the same stripe and Tenant B has been assigned SERVER 1, SERVER 2, and SERVER 3, also in the same stripe.

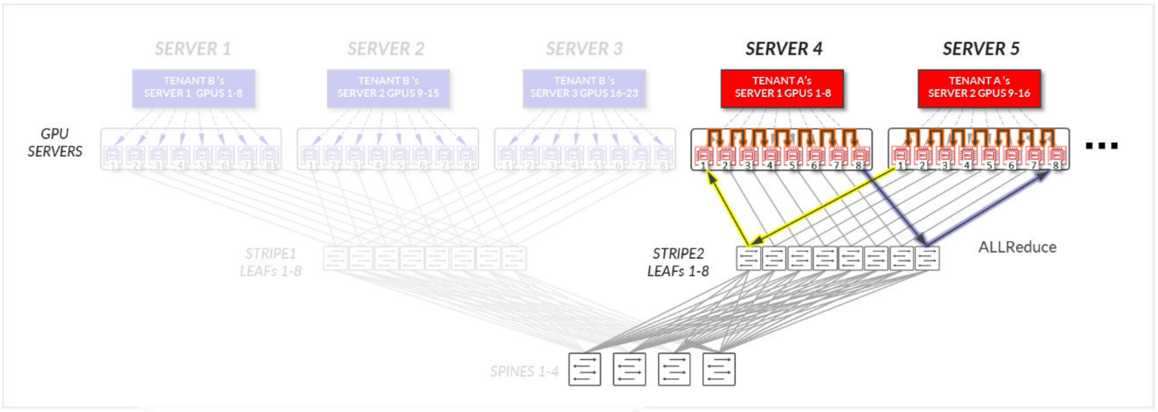
Figure 18: Server-isolation model GPU to GPU communication example 1


SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT B 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 2 GPUS 9-15
TENANT B 's SERVER 3 GPUS 16-23
TENANT A's SERVER 1 GPUS 1-8
TENANT A 's SERVER 2 GPUS 9-16
GPU SERVERS
1 2 3 4 5 6 7 8
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4

For Tenant A:

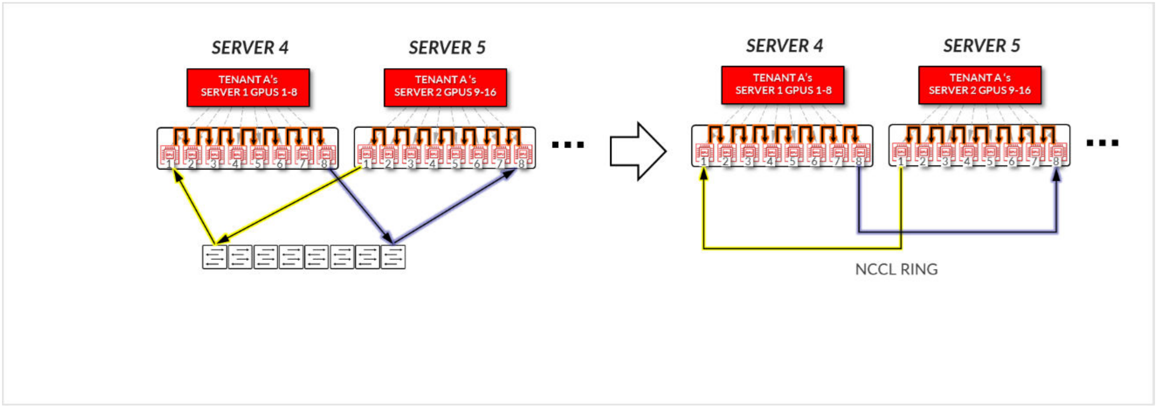
- GPUs 1-8 in SERVER 4, and GPUs 1-8 in SERVER 5 communicate internally within their respective servers, as explained in the section of "Local Optimization" on page 23.
- GPUs 1 and 8 in SERVER 4 communicate with GPUs 1 and 8 in SERVER 5 across the leaf and spine nodes - Intra-rail (traffic stays at the leaf node level).

Figure 19: Server-isolation model GPU to GPU communication example 1 – Tenant A


SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT B's SERVER 1 GPUS 1-8
TENANT B's SERVER 2 GPUS 9-15
TENANT B's SERVER 3 GPUS 16-23
TENANT A's SERVER 1 GPUS 1-8
TENANT A's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
ALLReduce
SPINES 1-4

- • You can see how a ring logical topology is established interconnecting the 16 GPUs assigned to Tenant A, without any traffic crossing the Spine nodes.

Figure 20: Server-isolation model GPU to GPU communication example 1 – Tenant A Ring topology



For Tenant B:

- GPUs 1-8 SERVER 1, GPUs 1-8 in SERVER 2, and GPUs 1-8 in SERVER3, communicate internally within their respective servers, as explained in the section of "Local Optimization" on page 23.
- GPUs 1 in SERVER 1 communicate GPUs 1 in SERVER 3 communicate with each other across the leaf nodes - Intra-rail (traffic stays at the leaf node level).
- GPUs 8 in SERVER 1 communicate GPUs 8 in SERVER 3 communicate with each other across the leaf nodes - Intra-rail (traffic stays at the leaf node level).
- GPUs 8 in SERVER 1 and GPUs 1 in SERVER 2 communicate across the leaf and spine nodes - Inter-rail. This is needed to complete the ring.

Figure 21: Server-isolation model GPU to GPU communication example 1 – Tenant B

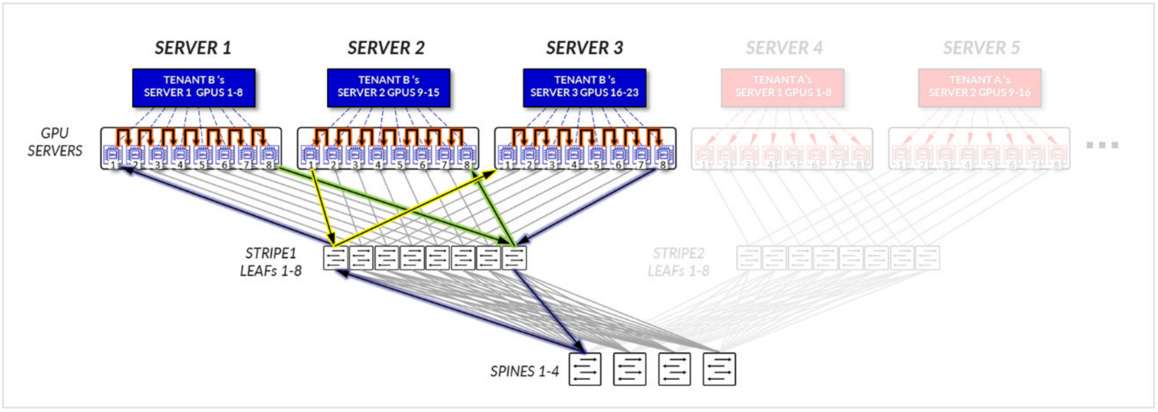
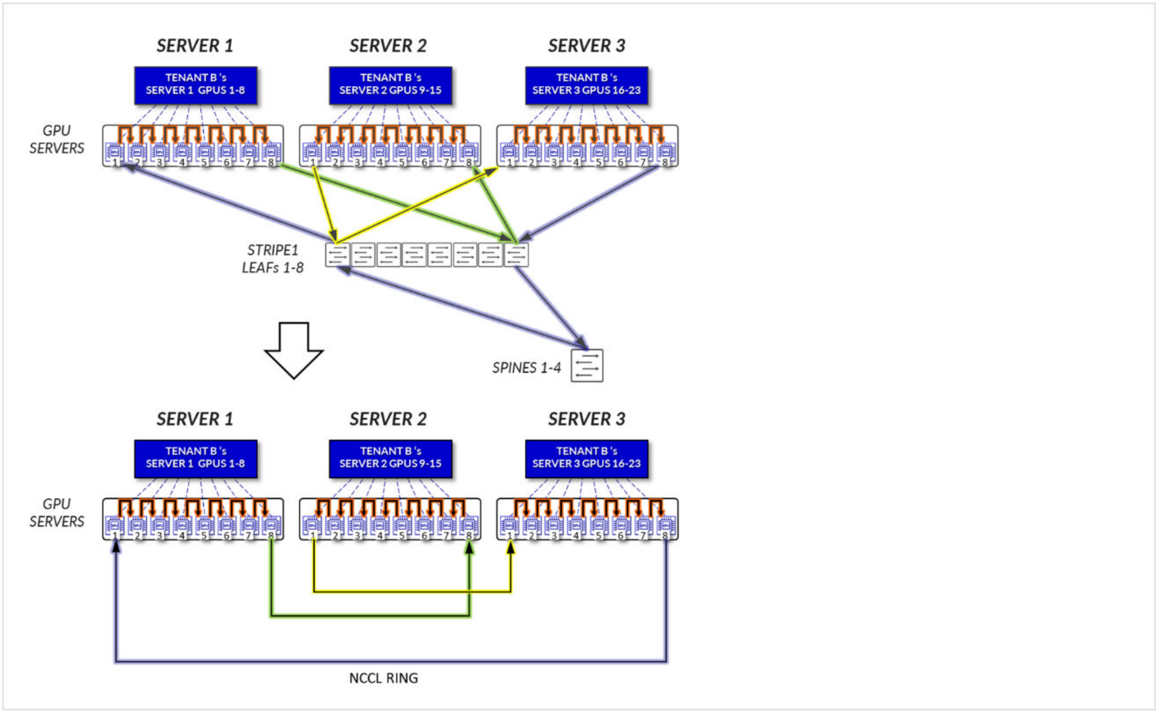

SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT B 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 2 GPUS 9-15
TENANT B 's SERVER 3 GPUS 16-23
TENANT A's SERVER 1 GPUS 1-8
TENANT A's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4

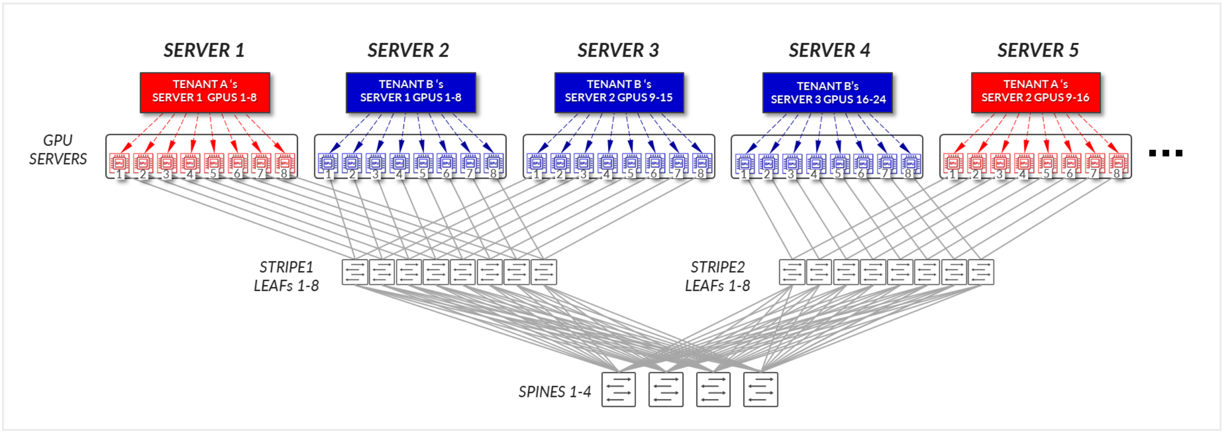
Figure 22: Server-isolation model GPU to GPU communication example 1 – Tenant B Ring topology


SERVER 1
SERVER 2
SERVER 3
TENANT B 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 2 GPUS 9-15
TENANT B 's SERVER 3 GPUS 16-23
GPU SERVERS
STRIPE1 LEAFs 1-8
SPINES 1-4
NCCL RING

Example 2

Now consider the example depicted in Figure 23, where Tenant A has been assigned Servers 1 and Server 5 in two different stripes, and Tenant B has been assigned Server 2, and Server 3, in the same stripe, and Server 4 in a different stripe.

Figure 23: Server-isolation model GPU to GPU communication example 2


SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT A 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 2 GPUS 9-15
TENANT B's SERVER 3 GPUS 16-24
TENANT A 's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4

For Tenant A:

- GPUs 1-8 in SERVER 1, and GPUs 1-8 in SERVER 5 communicate internally within their respective servers.
- GPUs 1 in SERVER 1 and GPUs 1 in SERVER 5 communicate across the leaf and spine nodes - Inter-stripe traffic.
- GPUs 8 in SERVER 1 and GPUs 8 in SERVER 5 communicate across the leaf and spine nodes - Inter-stripe traffic. This is needed to complete the ring.

Figure 24: Server-isolation model GPU to GPU communication example 2 – Tenant A

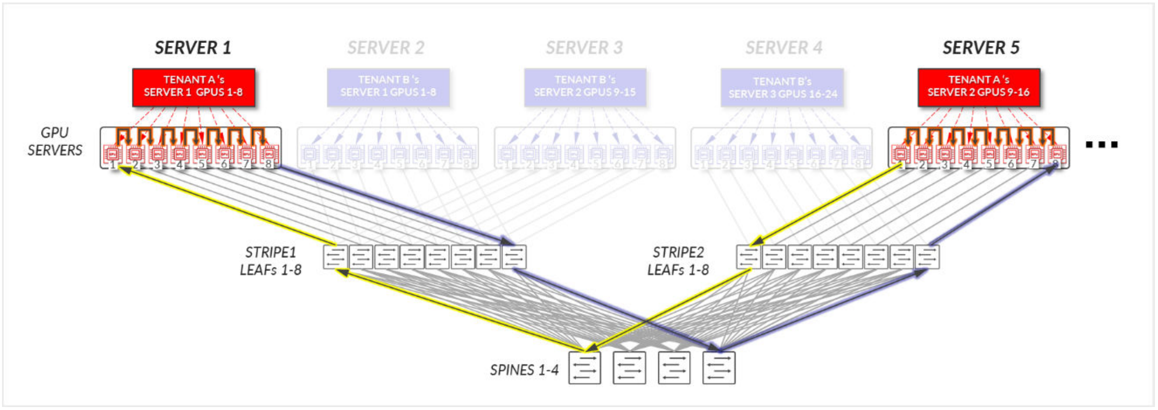
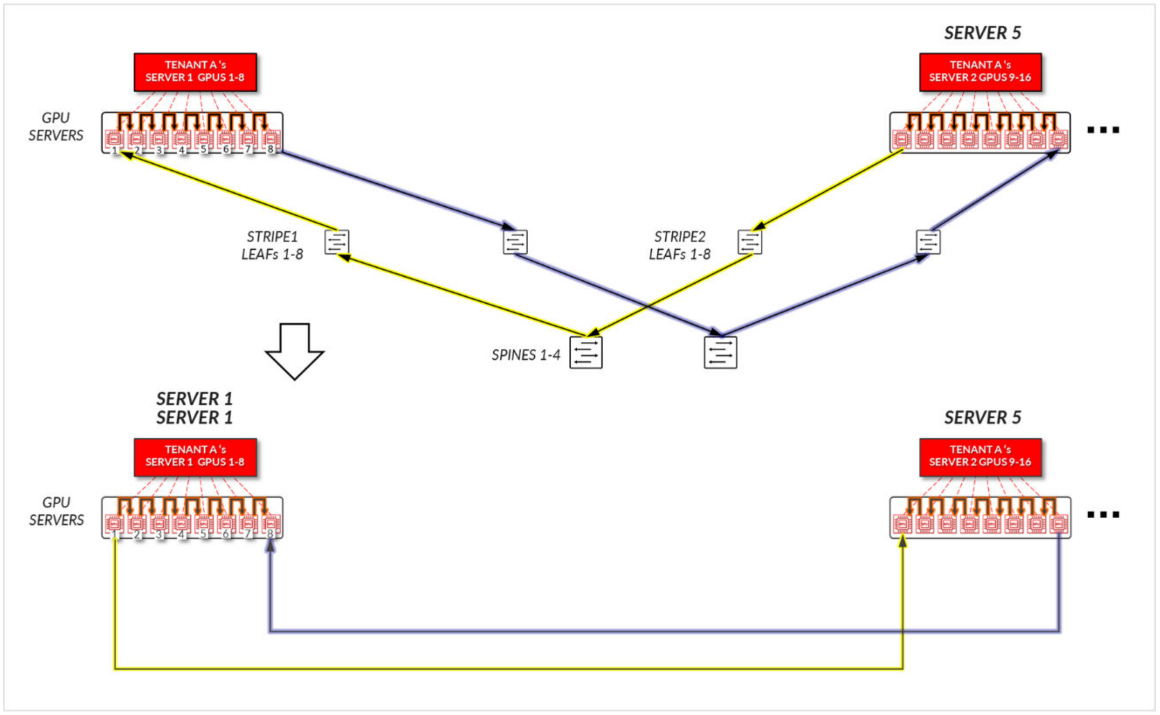

SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT A 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 1 GPUS 1-8
TENANT B 's SERVER 2 GPUS 9-15
TENANT B's SERVER 3 GPUS 16-24
TENANT A 's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4

Figure 25: Server-isolation model GPU to GPU communication example 2 – Tenant A Ring topology


SERVER 5
TENANT A's SERVER 1 GPUS 1-8
TENANT A's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4
SERVER 1
SERVER 1
SERVER 5
TENANT A's SERVER 1 GPUS 1-8
TENANT A's SERVER 2 GPUS 9-16
GPU SERVERS

For Tenant B:

- GPUs 1-8 SERVER 2, GPUs 1-8 in SERVER 3, and GPUs 1-8 in SERVER4, communicate internally within their respective servers.
- GPUs 1 in SERVER 2 and GPUs 1 in SERVER 4 communicate across the leaf and spine nodes - Inter-stripe traffic.
- GPUs 8 in SERVER 4 and GPUs 8 in SERVER 3 communicate across the leaf and spine nodes - Inter-stripe traffic.
- GPUs 1 in SERVER 3 and GPUs 8 in SERVER 2 communicate across the leaf and spine nodes – Inter-rail. This is needed to complete the ring.

Figure 26: Server-isolation model GPU to GPU communication example 2 – Tenant B

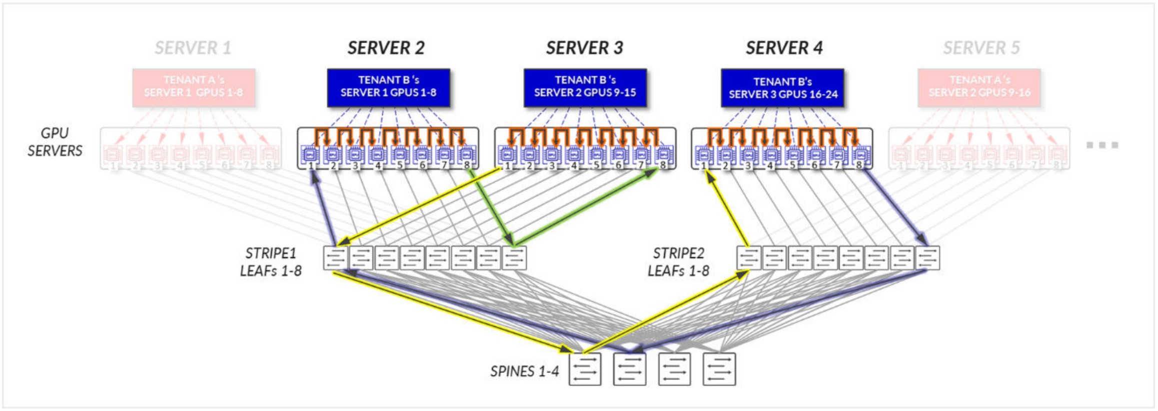
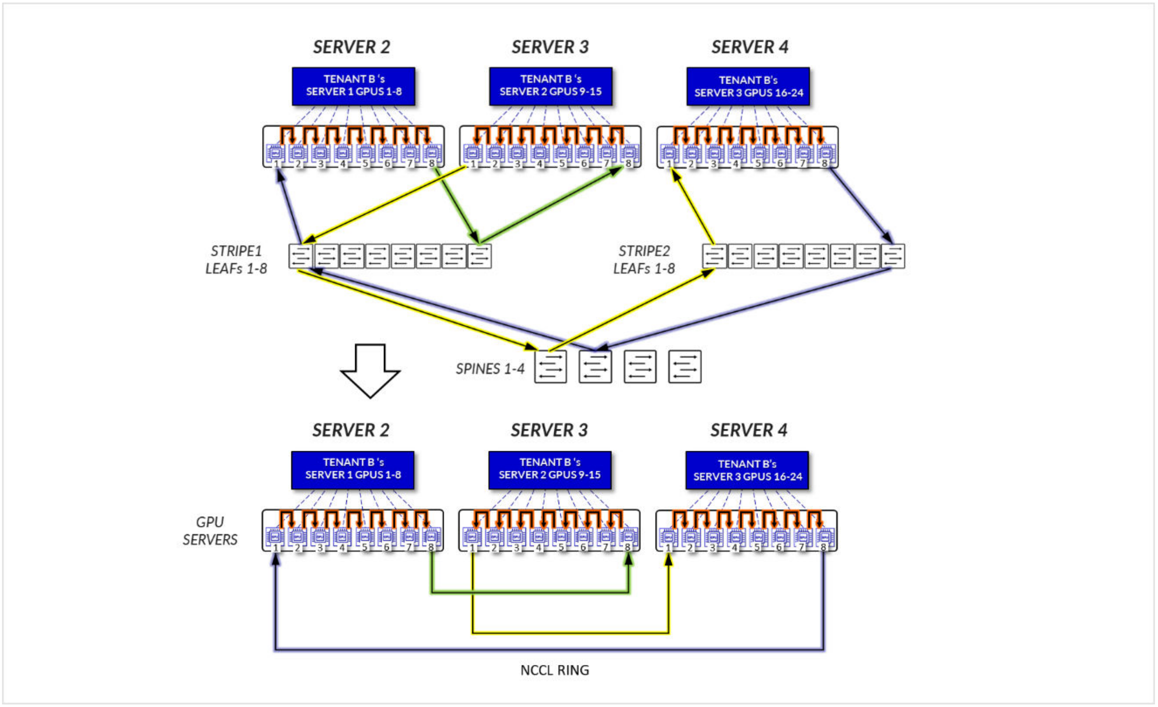
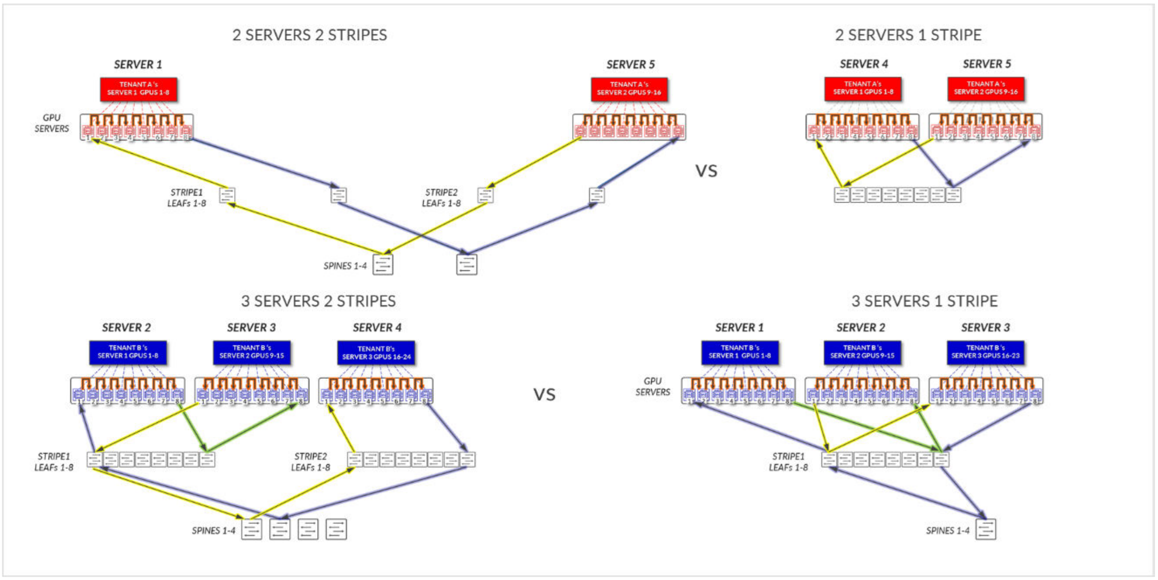

SERVER 1
SERVER 2
SERVER 3
SERVER 4
SERVER 5
TENANT A's SERVER 1 GPUS 1-8
TENANT B's SERVER 1 GPUS 1-8
TENANT B's SERVER 2 GPUS 9-15
TENANT B's SERVER 3 GPUS 16-24
TENANT A's SERVER 2 GPUS 9-16
GPU SERVERS
STRIPE1 LEAFs 1-8
STRIPE2 LEAFs 1-8
SPINES 1-4

Figure 27: Server-isolation model GPU to GPU communication example 2 – Tenant B Ring topology



Comparing the data flow in Examples 1 and 2, shows how the assignment of the Servers to a tenant could influence the performance of the jobs.

Figure 28: Server-isolation with servers in same stripe vs servers in different stripes



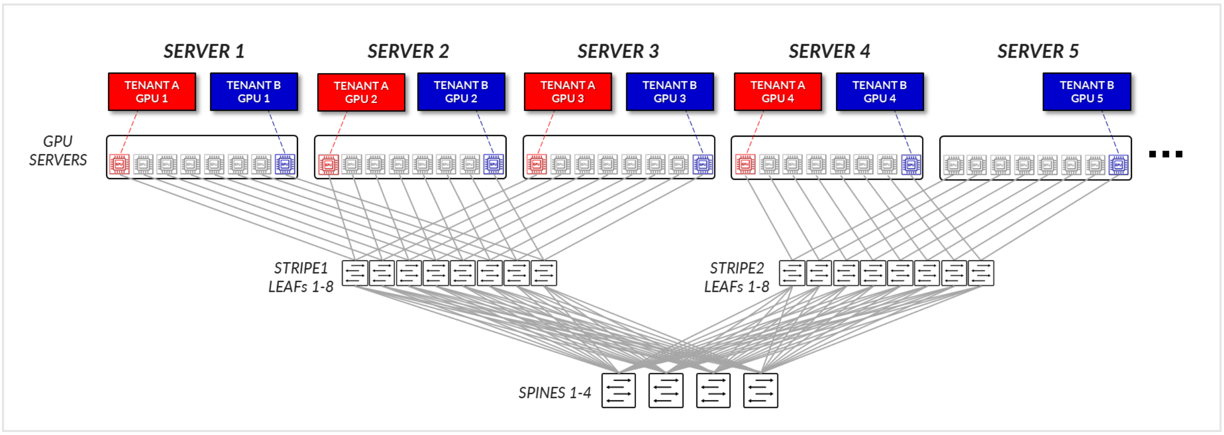
GPU Isolation model

In the **GPU-isolation model,** different GPUs in the same server can be assigned to different tenants. Also, a tenant might be assigned GPUs in multiple servers across multiple stripes. As for the server isolation model, where the assigned GPUs are located will affect the path and potentially the performance.

Example 1

Consider the example depicted in Figure 29, where Tenant A has been assigned GPU1 on SERVERs 1-4, and Tenant B has been assigned GPU8 on SERVERs 1-5.

Figure 29: GPU-isolation model GPU to GPU communication example 1



For Tenant A:

- Tenant A's GPUs 1, 2, and 3 communicate with each other across the leaf node where they are connected. (Intra-rail)

- Tenant A's GPUs 1, 2, and 3 communicate with GPU 4 communicate across the leaf and spine nodes.

Figure 30: GPU-isolation model GPU to GPU communication example 1 – Tenant A

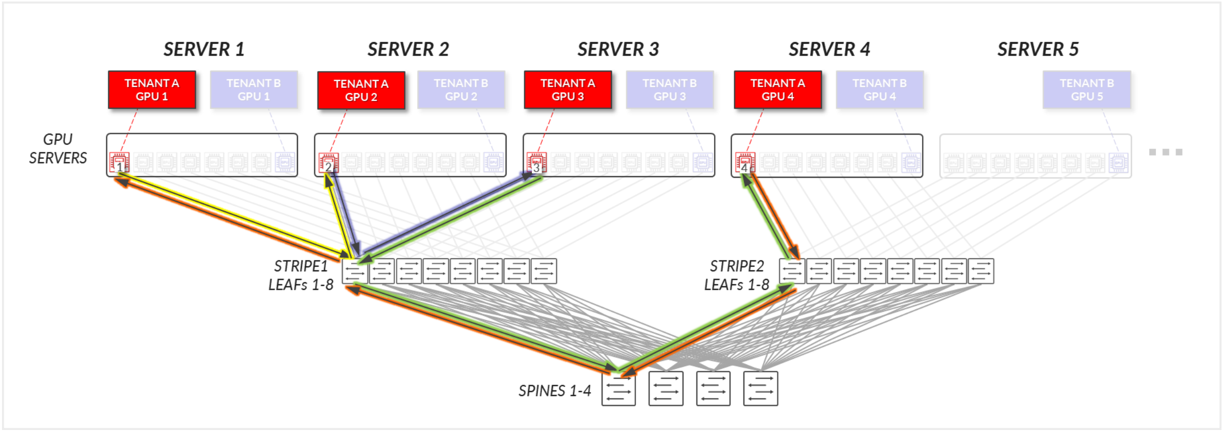
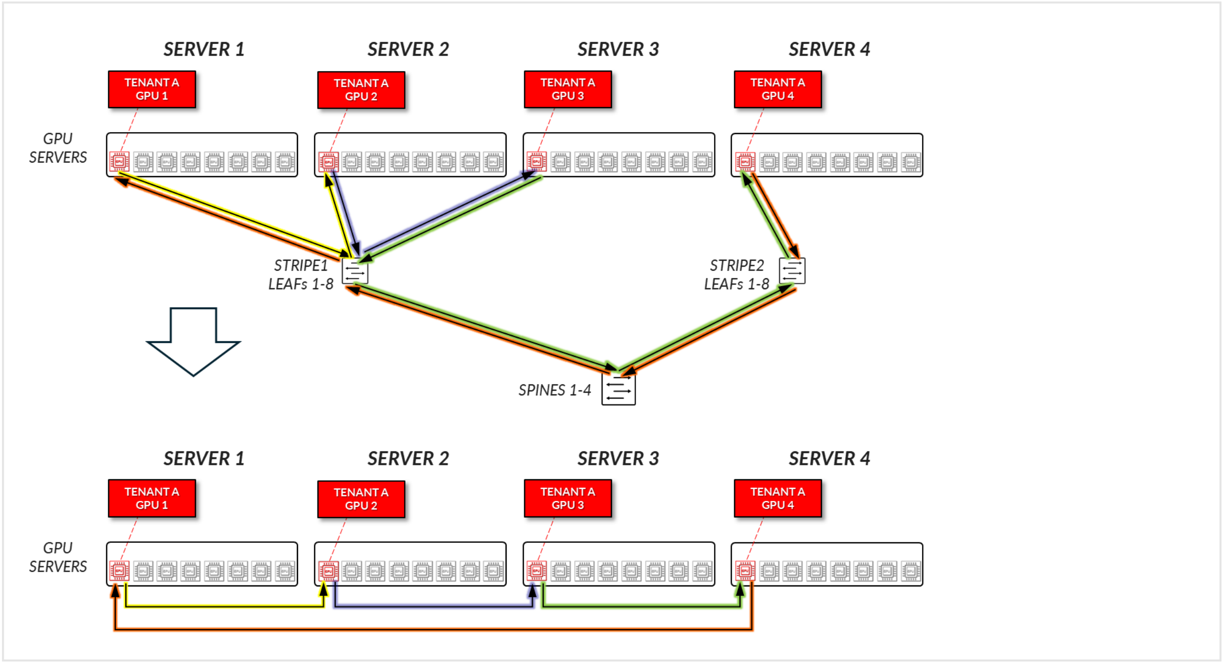


Figure 31: GPU-isolation model GPU to GPU communication example 1 – Tenant A ring topology

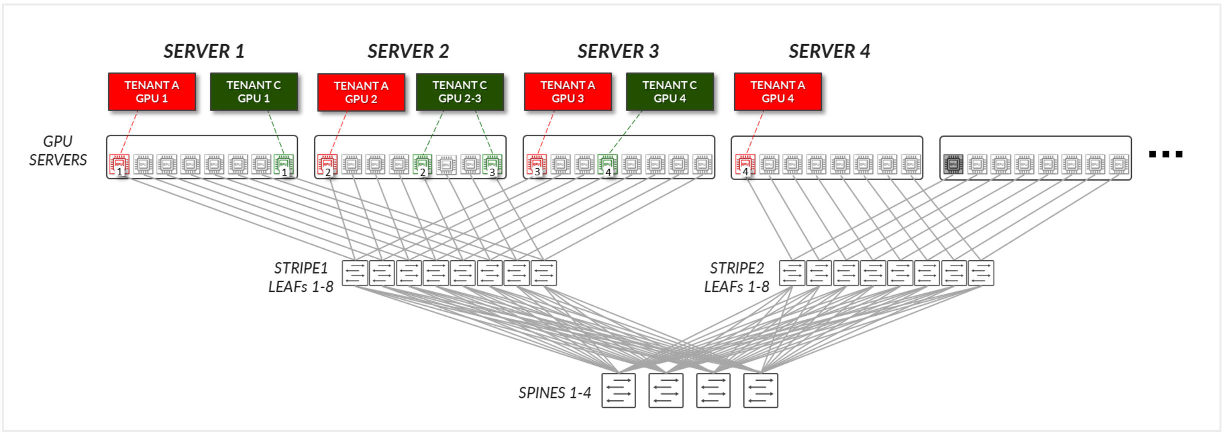


For Tenant B, a similar communication path is established.

Example 2

Now Consider the example depicted in Figure 32, where Tenant C has been assigned GPUs 8 on SERVER 1, GPUs 5 & 8 on SERVER 2, and GPU 4 on SERVER 3 (corresponding to Tenant C's GPUs 1-4 in the diagram).

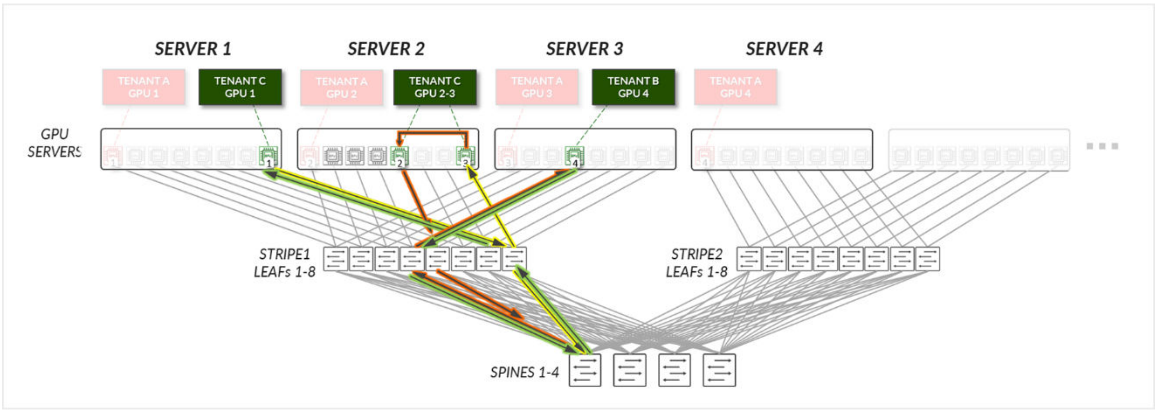
Figure 32: GPU-isolation model GPU to GPU communication example 2



For Tenant C:

- Tenant C’s GPUs 2 and 3 (on the same server), communicate internally within their server.
- Tenant C's GPU 3 (SERVER 2) and GPU 4 (SERVER 3) communicate across the leaf and spine nodes.
- Tenant C's GPU 4 (SERVER 3) and GPU 1 (SERVER 1) communicate across the leaf and spine nodes.
- Tenant C's GPU 1 (SERVER 1) and GPU 2 (SERVER 2) communicate across the leaf and spine nodes.

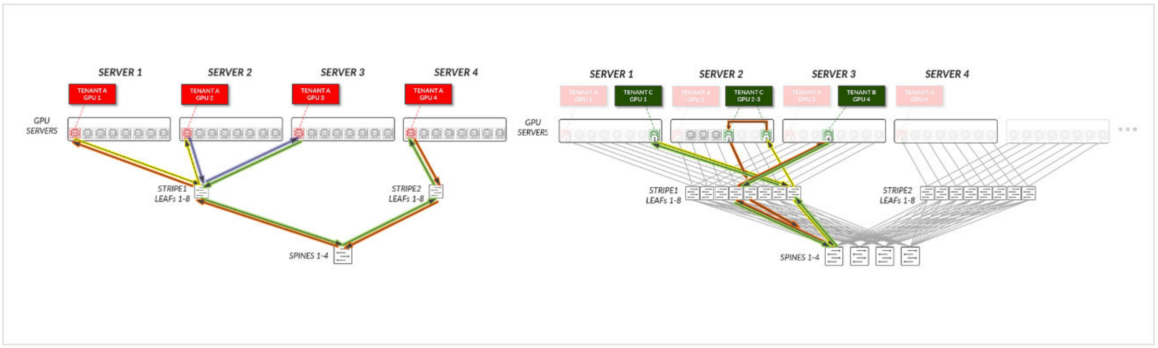
Figure 33: GPU-isolation model GPU to GPU communication example 2 – Tenant C



When comparing examples 1 and 2, it becomes clear how rail alignment and proper server or GPU assignment strategies are critical to achieving optimal GPU-to-GPU communication efficiency on a scale.

Tenant A in Example 1, has been assigned GPU0 on Servers 1-4, thus communication mostly stays at the leaf level. Tenant C in Example 2 has been assigned GPUs 8 on SERVER 1, GPUs 5 & 8 on SERVER 2, and GPU 4 on SERVER 3, so communication must go across the spines, introduces additional latency and potential congestion Both Tenant A and Tenant C have been assigned the same number of GPUs, but communication between their GPUs follows different paths, which could result in varying performance levels.

Figure 34: GPU-isolation with servers in same stripe vs servers in different stripes



EVPN/VXLAN GPU Backend Fabric for Multitenancy – Implementation Options

IN THIS SECTION

Implementing GPU multitenancy in a data center requires a network architecture that ensures strong isolation, high throughput, and low latency across the shared infrastructure. This involves architectural considerations not only for the **GPU backend fabric**, which provides connectivity between GPUs belonging to each tenant, but also for the **frontend fabric**, where user access, job submission, orchestration, and authentication are handled, and the **storage backend**, which is responsible for delivering datasets, model checkpoints, and results to and from the GPU infrastructure. These components each require their own design strategies to ensure end-to-end performance, security, and multitenancy across the entire AI platform stack.

This JVD focuses specifically on the GPU backend fabric, which handles east-west traffic between GPUs across servers and is subject to the strictest performance and isolation requirements. EVPN/VXLAN is commonly used as the foundation for scalable multitenant environments, supporting two main design approaches: **pure Type 5 services** with IP-VRFs only, and **VLAN-aware services** with MAC-VRFs and symmetric IRB.

The **pure Type 5** model follows the BGP EVPN IP Prefix Route specifications described in **RFC 9136** (IP Prefix Advertisement in Ethernet VPN - EVPN). Traffic forwarding across the fabric relies entirely on Layer 3 routing, avoiding MAC learning and simplifying both the control plane and IP address management. In contrast, the VLAN-aware model uses Layer 2 overlays to extend bridging and VLAN segmentation across the fabric. Both approaches use routed underlay designs with VXLAN encapsulation, enabling flexible resource allocation and tenant isolation across multiple physical servers.

These two approaches are summarized in table 7 for both GPU Isolation and Server Isolation.

Table 7. EVPN/VXLAN models comparison

FEATURES	**Pure RT5 EVPN/VXLAN (Recommended)**		**VLAN-Aware EVPN/VXLAN service with MAC-VRF**	
Multi-tenancy Type	GPU-Isolation (Per GPU multitenancy)	Server Isolation (Per-server multitenancy)	GPU-Isolation (Per GPU multitenancy)	Server Isolation (Per-server multitenancy)
GPU Assignment (Tenant Resource Allocation)	One or more GPU (but not all) per server assigned to multiple Tenants	All GPUs (8) per server assigned to a single Tenant	One or more GPU (but not all) per server assigned to multiple Tenants	All GPUs (8) per server assigned to a single Tenant
Tenant GPU Distribution	A tenant can have one or more (but not all) GPUs on one or more servers.	A tenant can have all the GPUs on one or more servers.	A tenant can have one or more (but not all) GPUs on one or more servers.	A tenant can have all the GPUs on one or more servers.
VLANs per serveró Leaf node Links	No VLANs	No VLANs	Each link is in a different VLAN and is assigned a different VNI.	Each link is in a different VLAN and is assigned a different VNI.
Interface configuration Mode and VLAN Mapping	Access-mode interfaces, server links in different RT5_IP-VRF	Access-mode interfaces, server links in different RT5_IP-VRF	Access-mode interfaces, server links in different MAC-VRF	Access-mode interfaces, server links in different MAC-VRF

(Continued)

FEATURES	**Pure RT5 EVPN/VXLAN (Recommended)**		**VLAN-Aware EVPN/VXLAN service with MAC-VRF**	
IP addressing per serveró Leaf node Links	Server links configured with: • /31 IPv4, • /127 IPv6 addresses or • /64 IPv6 addresses with SLAAC 8 x IP routed links	Server links configured with: • /31 IPv4, • /127 IPv6 addresses or • /64 IPv6 addresses with SLAAC 8 x IP routed links	Server links configured with: • /24 IPv4 or • /64 IPv6 addresses	Server links configured with: • /24 IPv4 or • /64 IPv6 addresses
VRF and Routing Instances per tenant	One RT5_IP-VRF only <u>No</u> MAC-VRF (on each leaf node where the GPUs assigned to tenant are connected)	One RT5_IP-VRF only <u>No</u> MAC-VRF (on each leaf node where the GPUs assigned to tenant are connected)	One RT5_IP-VRF & One MAC-VRF (on each leaf node where the GPUs assigned to tenant are connected)	One RT5_IP-VRF & One MAC-VRF (on each leaf node where the GPUs assigned to tenant are connected)
VNI Allocation per Tenant	Single VNI per tenant	Single VNI per tenant	8 x VNIs per tenant	8 x VNIs per tenant
Anycast Gateway Configuration	No Anycast Gateway (no IRB interfaces)	No Anycast Gateway (no IRB interfaces)	8 x Anycast IP Gateways (8 x IRB interfaces)	8 x Anycast IP Gateways (8 x IRB interfaces)
EVPN Service Type	Pure/Pure RT5 EVPN/VXLAN design	Pure/Pure RT5 EVPN/VXLAN	VLAN-Aware EVPN/ VXLAN service (with MAC-VRF)	VLAN-Aware EVPN/ VXLAN service (with MAC-VRF)
ERB Design	No ERB	No ERB	ERB design without ESI_LAG	ERB design without ESI_LAG

(Continued)

FEATURES	Pure RT5 EVPN/VXLAN (Recommended)		VLAN-Aware EVPN/VXLAN service with MAC-VRF	
Underlay BGP Configuration	Underlay IPv6 BGP Unnumbered	Underlay IPv6 BGP Unnumbered	Underlay IPv6 BGP Unnumbered	Underlay IPv6 BGP Unnumbered
IRB and Routing Strategy	Pure RT5 EVPN routing - no IRB interfaces	Pure RT5 EVPN routing - no IRB interfaces	Symmetric IRB – Type 5	Symmetric IRB – Type 5
Congestion Control (DCQCN Type)	Pure Type 5 DCQCN; VXLAN DCQCN	Pure Type 5 DCQCN; VXLAN DCQCN	Type 2 & 5 DCQCN; VXLAN DCQCN	Type 2 & 5 DCQCN; VXLAN DCQCN

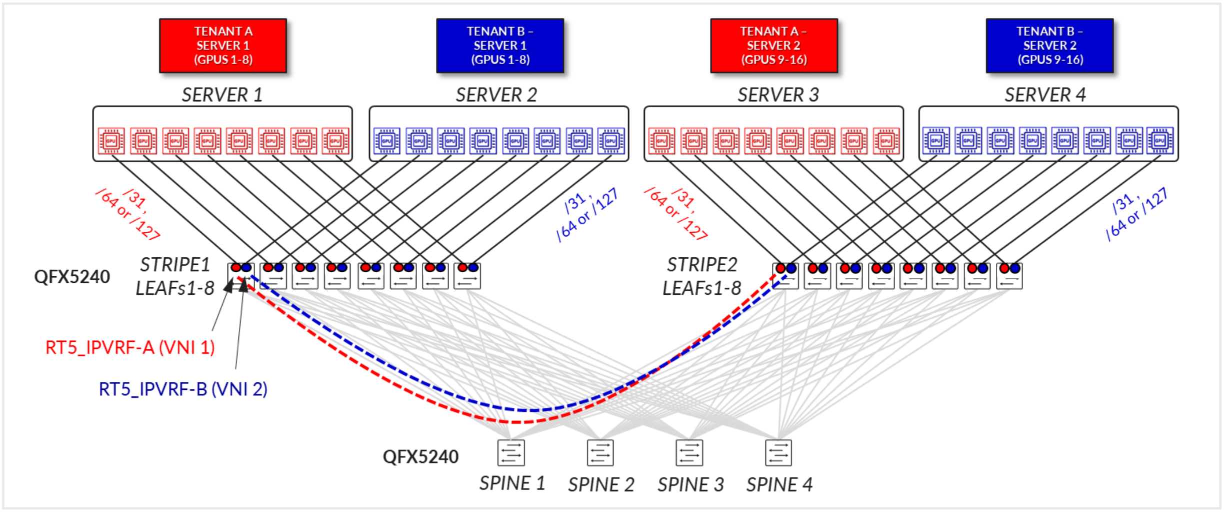
Pure RT5 EVPN/VXLAN - Server-Level Isolation (Per-Server Multitenancy)

In this design model, each physical server is dedicated entirely to a single tenant, meaning all GPUs (typically 8 per server) are assigned to one tenant only. This model simplifies resource allocation and isolation since there's no sharing of GPU resources between tenants on a single server. A tenant can span across multiple servers, each of which fully belongs to that tenant.

Server-to-leaf links are configured as L3 links, in access mode (no VLAN tagging), and are assigned unique IP addresses (/31 IPv4, /127 or /64 IPv6). The recommended solution in this document prescribes automatically assigning /64 IPv6 addresses using SLAAC (Stateless Address Autoconfiguration). This approach enables servers to self-configure their addresses without requiring manual edits to each server's netplan configuration. Configuration options for IPv4 are covered in the Appendix

Each server-facing link is associated with the same Tenant's RT5_IP-VRF routing instance across the leaf nodes within a stripe, according to Tenant's assignments, as shown in Figure 36.

Figure 35: Pure RT5 EVPN/VXLAN - Server-Level Isolation (Per-Server Multitenancy)



The fabric is configured as a pure EVPN/VXLAN Type 5 with no MAC-VRFs, IRBs, or anycast gateways involved.

BGP underlay sessions are established using IPv6 link-local addresses with automatic neighbor discovery, while overlay sessions are established between the IPv6 unicast addresses assigned to the loopback interfaces and advertised via the underlay. Congestion control is implemented using VXLAN-aware DCQCN, ensuring fairness and traffic stability.

NOTE: If the overlay is using IPv4 addresses, the underlay needs to be configured using RFC 5549 to advertise IPv4 routes with IPv6 next-hops. See "Appendix A – IPv4 Overlay Over IPv6 Underlay Fabric Implementation" on page 192 for more details.

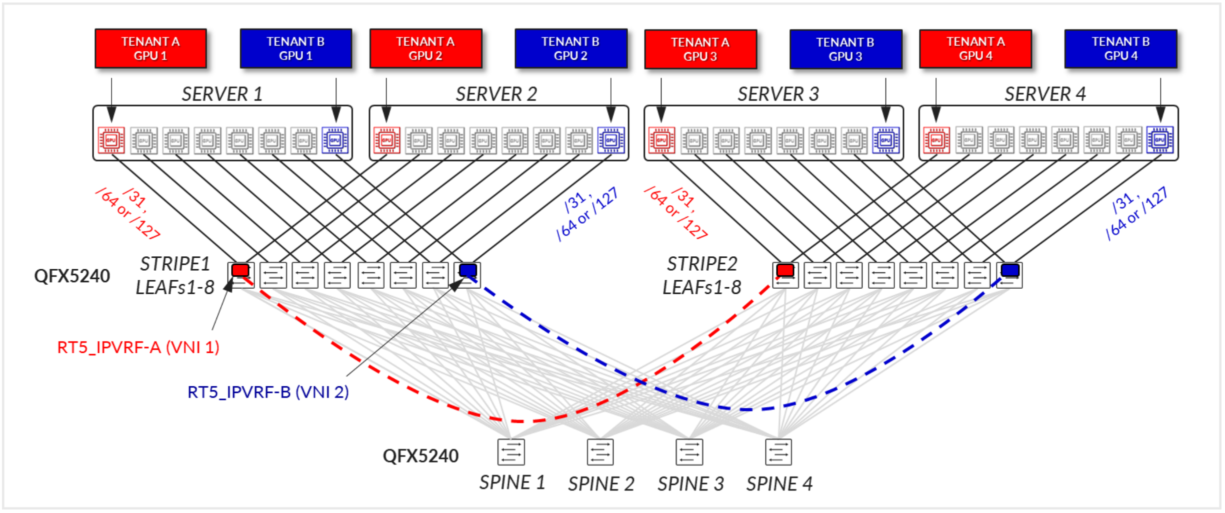
Pure RT5 EVPN/VXLAN - GPU-Level Isolation (Per-GPU Multitenancy)

This model introduces finer-grained resource sharing by allowing GPUs within the same server to be allocated to different tenants. A tenant may receive one or more GPUs across one or multiple servers, but not all GPUs on any given server unless explicitly assigned. This GPU-level partitioning allows for more efficient use of server resources and is well-suited for environments with dynamic or fractional GPU demands. Despite the increased resource-sharing granularity, the server to leaf node connectivity remains the same.

Server-to-leaf links are still configured as L3 links, in access mode (no VLAN tagging), and are assigned unique IP addresses (/31 IPv4, /127 or /64 IPv6). The recommended solution in this document prescribes automatically assigning /64 IPv6 addresses using SLAAC (Stateless Address Autoconfiguration). This approach allows servers to automatically configure their addresses, eliminating the need to manually edit each server's netplan configuration. Configuration options for IPv4 are covered in the Appendix.

Each link in a server is mapped to a different Tenant's RT5_IP-VRF routing instances across the leaf nodes within a stripe, according to Tenant's assignments, as shown in Figure 36.

Figure 36: Pure RT5 EVPN/VXLAN – GPU Level Isolation (Per-GPU Multitenancy)



The fabric is still configured as a pure EVPN/VXLAN Type 5 with no MAC-VRFs, IRBs, or anycast gateways involved.

BGP underlay sessions are established using IPv6 link-local addresses with automatic neighbor discovery, while overlay sessions are established between the IPv6 unicast addresses assigned to the loopback interfaces and advertised via the underlay. Congestion control is implemented using VXLAN-aware DCQCN, ensuring fairness and traffic stability.

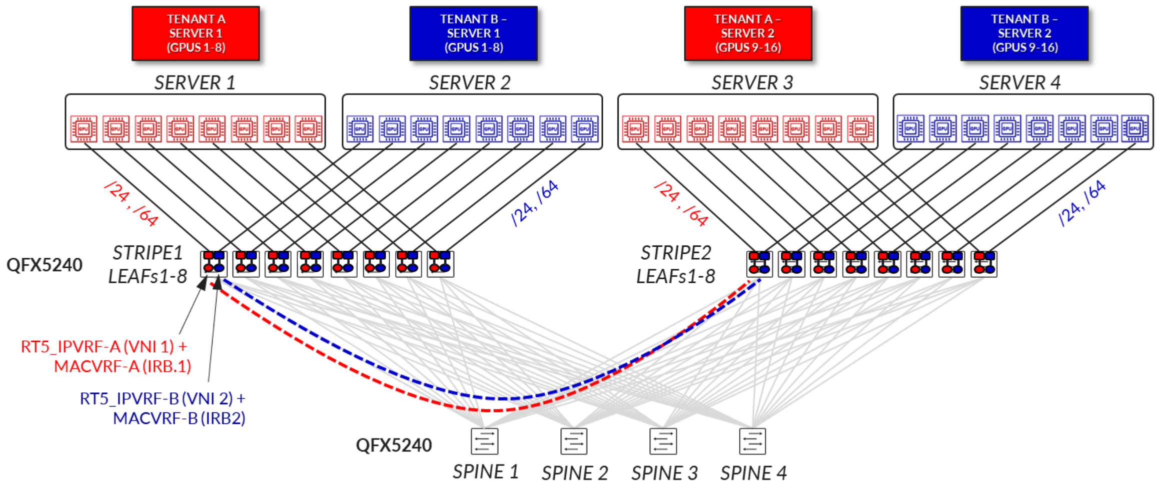
NOTE: If the overlay is using IPv4 addresses, the underlay needs to be configured using RFC 5549 to advertise IPv4 routes with IPv6 next-hops. See "Appendix A – IPv4 Overlay Over IPv6 Underlay Fabric Implementation" on page 192 for more details.

VLAN-Aware EVPN/VXLAN -Server-Level Isolation (Per-Server Multitenancy)

In this design model, each physical server is fully dedicated to a single tenant, meaning all GPUs (typically 8 per server) are assigned exclusively to that tenant. This approach simplifies resource allocation and ensures strong isolation, as there is no GPU resource sharing across tenants on the same server. A tenant may span multiple servers, each entirely allocated to that tenant.

Server-to-leaf links are configured as Layer 3 interfaces; each associated with a unique VLAN and VNI. The recommended solution in this document uses /64 IPv6 addresses automatically assigned via SLAAC (Stateless Address Autoconfiguration), eliminating the need for manual configuration of each server's netplan file. IP addressing is allocated from larger pools (e.g., /24 for IPv4 and /64 for IPv6), with each link receiving its own anycast gateway (IRB) interface, resulting in 8 IRB interfaces per server. Each server-facing link is associated with the tenant's MAC-VRF and IP-VRF routing instances across the leaf nodes within a stripe, according to the tenant's assignment, as shown in Figure 37.

Figure 37: VLAN-aware EVPN/VXLAN – Server Level Isolation (Per-Server Multitenancy)


TENANT A SERVER 1 (GPUS 1-8)
TENANT B – SERVER 1 (GPUS 1-8)
TENANT A – SERVER 2 (GPUS 9-16)
TENANT B – SERVER 2 (GPUS 9-16)
SERVER 1
SERVER 2
SERVER 3
SERVER 4
/24 , /64
/24, /64
/24 , /64
/24, /64
QFX5240
STRIPE1 LEAFs1-8
STRIPE2 LEAFs1-8
RT5_IPVRF-A (VNI 1) + MACVRF-A (IRB.1)
RT5_IPVRF-B (VNI 2) + MACVRF-B (IRB2)
QFX5240
SPINE 1
SPINE 2
SPINE 3
SPINE 4

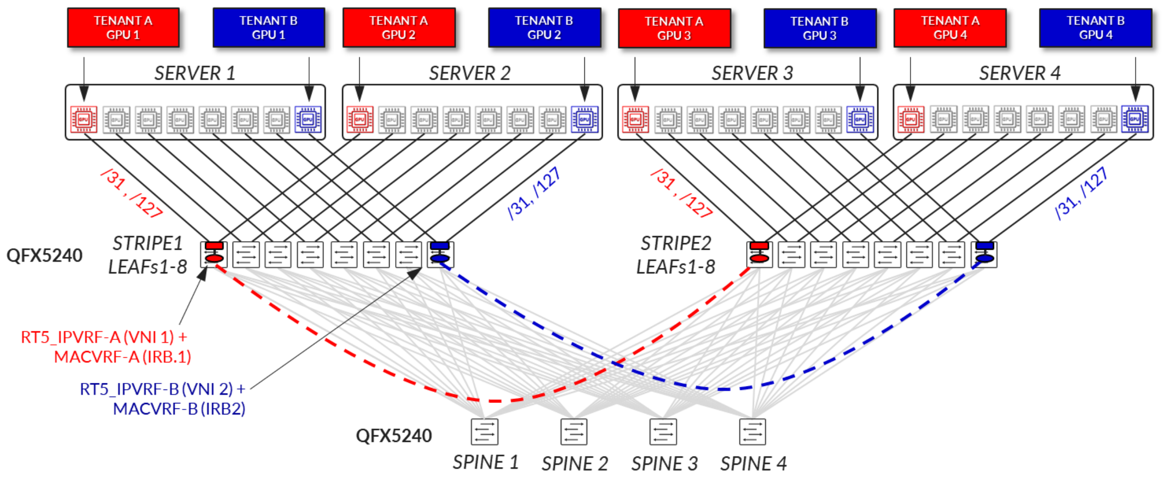
From a network design perspective, this use case relies on a VLAN-Aware EVPN/VXLAN service, with per-tenant separation using both MAC-VRFs and IP-VRFs. Each leaf switch hosting a tenant's servers maintains a pair of VRFs: a MAC-VRF for bridging and an RT5_IP-VRF for routing. The design follows a symmetric IRB model, supporting both EVPN Type 2 and Type 5 routes, and implements VXLAN-aware DCQCN for congestion management.

VLAN-Aware EVPN/VXLAN - GPU-Level Isolation (Per-GPU Multitenancy)

This design model enables finer-grained resource sharing by assigning individual GPUs within a server to different tenants. A single server can be shared across multiple tenants, each with access to a subset of GPUs rather than the entire server. This approach increases overall compute resource utilization while maintaining strong isolation between tenants.

Server-to-leaf links are configured as Layer 3 interfaces, each associated with a unique VLAN and VNI. IPv6 addresses are automatically assigned via SLAAC (/64 per interface), allowing the server to self-configure without requiring manual edits to its netplan file. IP addressing is allocated from larger pools (e.g., /24 for IPv4 and /64 for IPv6), with each GPU-facing link receiving its own anycast gateway (IRB) interface, resulting in 8 IRB interfaces per server. Each interface is associated with the tenant's MAC-VRF and IP-VRF routing instances across the leaf nodes, according to the tenant's assignment.

Figure 38: VLAN-aware EVPN/VXLAN - GPU-Level Isolation (Per-GPU Multitenancy)



From a network design perspective, this use case also relies on a VLAN-Aware EVPN/VXLAN service, with per-tenant separation using both MAC-VRFs and IP-VRFs. Each leaf switch hosting the server's GPU-assigned interfaces maintains a pair of VRFs: a MAC-VRF for bridging and an RT5_IP-VRF for routing. The design follows a symmetric IRB model, supports both EVPN Type 2 and Type 5 routes, and implements VXLAN-aware DCQCN to ensure fair and stable congestion control across the shared infrastructure.

Selecting the Best Approach

In the context of AI workloads such as training, inference, and GPU-as-a-Service (GPUaaS), the choice between a pure Type 5 and a VLAN-aware EVPN/VXLAN design can significantly impact operational efficiency. The **pure Type 5 model** is often better suited for large-scale **AI training environments**, where GPU resources are allocated in bulk, either per server or per tenant, and workloads are typically long running and tightly coupled. Its streamlined IP-based routing, stable addressing, and minimal control-plane overhead enable predictable performance and simplified automation across hundreds or thousands of servers. In contrast, the **VLAN-aware model** may be more appropriate for **GPUaaS platforms**, **inference workloads**, or **multi-purpose environments** where tenants run shorter, independent jobs and require granular isolation, dynamic L2 connectivity, or per-interface policy enforcement. The use of MAC-VRFs and anycast gateways provides flexibility for tenant-specific services, especially in use cases involving legacy applications, bare-metal workloads, or environments that need tenant-specific IP gateways. Ultimately, both models support GPU multitenancy, but the pure Type 5 design favors **scale and simplicity**, while the VLAN-aware design offers **flexibility and fine-grained control**.

NOTE: This JVD focuses on the Pure RT5 EVPN/VXLAN implementation. Thus, the rest of the document will cover all the details for the Pure RT5 EVPN/VXLAN - Server-Level Isolation (Per-Server Multitenancy) and Pure RT5 EVPN/VXLAN - GPU-Level Isolation (Per-GPU Multitenancy) options.

EVPN/VXLAN GPU Backend Fabric for Multitenancy – Type 5 EVPN/VXLAN Implementation

IN THIS SECTION

Tenant Separation

Preserving tenant separation requires careful design at two levels:

- **Fabric Tenant Separation** – isolation of traffic across the fabric
- **Internal Server Separation** – isolation of GPU access within each server

Fabric Tenant Separation

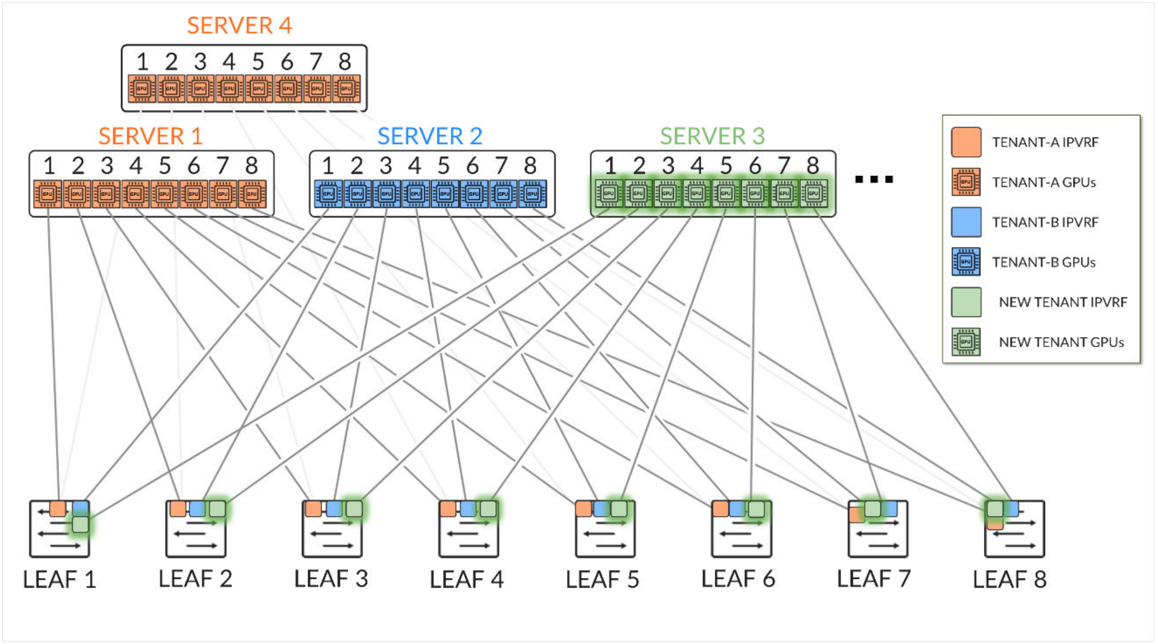
Across the fabric, separation is achieved by implementing EVPN/VXLAN pure Type 5, where the interfaces connecting the GPUs assigned to tenants are mapped to distinct IP-VRF routing instances on the leaf nodes. Fabric Tenant Separation is implemented slightly differently for the Server Isolation, and GPU Isolation models.

For **Server Isolation**:

When a new tenant is onboarded and assigned one or more servers, a dedicated IP-VRF routing instance is created for that tenant on each leaf node within a stripe. The interfaces of the assigned servers are then added to this VRF. Because GPU servers are connected in a rail-optimized topology, at least one interface on each leaf node is typically part of the new VRF, as illustrated in Figure 39.

In the example, Tenant A is assigned Servers 1 and 4, a VRF is instantiated on Leaf nodes 1 through 8. All the interfaces on these two servers are associated with the VRF based on the rail-aligned connectivity model, resulting in two interfaces per leaf node. Tenants B and C, are assigned to Servers 2 and 3 respectively, and each receive their own VRF, with one interface per leaf node.

Figure 39: Server Isolation Tenant Assignments



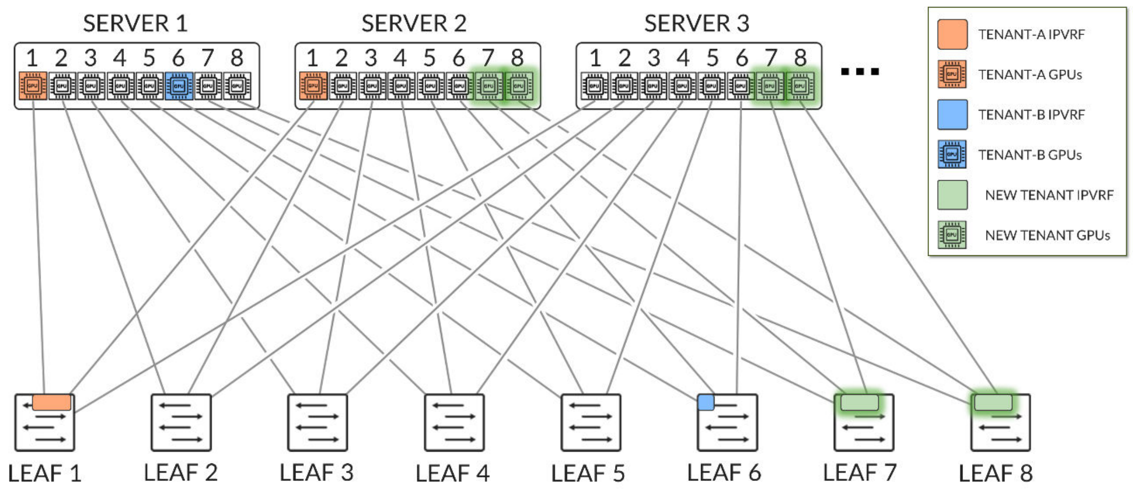
For **GPU Isolation**:

When a new tenant is onboarded and assigned one or more GPU, a dedicated IP-VRF routing instance is created for that tenant BUT *only* on the leaf nodes with physical connections to the GPUs assigned to that tenant, as shown in Figure 40

In the example, Tenant A is assigned GPU 0 on Servers 1 and 2, its VRF is created only on Leaf 1. No other leaf nodes are affected. Tenant B is assigned GPU 6 on Server 1, its VRF is created only on Leaf 6. Tenant C is assigned GPUs 7 and 8 on Servers 2 and 3, its VRF is created on Leaf 7 and Leaf 8

This selective placement of IP-VRFs ensures that only the required leaf nodes participate in each tenant's network, minimizing configuration overhead while maintaining strict isolation at the GPU level.

Figure 40: GPU Isolation Tenant Assignments



Internal Server Separation

Placing interfaces into different VRFs on the switch side is not sufficient for complete isolation. It is also necessary to isolate the GPUs within the servers. Although disabling local optimization or PXN may appear to prevent cross-GPU traffic, in reality it only prevents a GPU from using another GPU within the same server as a proxy to reach a GPU on a different rail in a different server, as described in the Local Optimization section. Additional mechanisms are therefore required to ensure true separation, including Kubernetes implementation, and Isolation using NCCL variables.

Kubernetes-Based Isolation:

Many organizations adopt Kubernetes for GPU multitenancy because of its ability to manage shared resources efficiently while isolating workloads across users or teams. Features such as namespaces, cgroups, and role-based access control (RBAC) provide secure, Tenant-1 ware environments that keep workloads isolated within a shared infrastructure. Kubernetes also integrates with vendor-supported GPU operators from NVIDIA and AMD, streamlining the deployment of drivers, device plugins, and monitoring components. This simplifies administration and enables accurate tracking of GPU usage per tenant.

While Kubernetes provides a robust framework for GPU multitenancy in production environments, it is not always practical or necessary for testing and validation.

Isolation with NCCL variables:

In lab setups or early development stages, multitenancy can be implemented without deploying a full Kubernetes stack by manually controlling resource visibility through environment variables. This lightweight approach allows administrators to isolate GPU and network resources per tenant using variables such as:

- **CUDA_VISIBLE_DEVICES** (for NVIDIA servers),
- **ROCR_VISIBLE_DEVICES** (for AMD servers),
- **UCX_NET_DEVICES**, and
- **NCCL_IB_HCA**.

By setting **CUDA_VISIBLE_DEVICES** (on NVIDIA servers) and **ROCR_VISIBLE_DEVICES** (on AMD servers) environment variables, administrators can restrict each tenant's applications to having visibility and access to only their assigned GPUs.

When set, they mask all other GPUs from the application's perspective, creating the appearance that only the assigned GPU(s) are available, preventing unwanted GPU-to-GPU communication. The exposed GPUs are then re-indexed starting from 0. Thus, for each tenant, the GPUs will be indexed starting at 0, regardless of the actual GPU number (rank).

For example, when running a NCCL test on an NVIDIA server:

- If a tenant is assigned GPU1, setting:
  export CUDA_VISIBLE_DEVICES=1

  ensures that only GPU1 is visible to the application. Internally, this GPU will appear to the application as cuda:0.

- Similarly, if a tenant is assigned GPU4, setting:
  export CUDA_VISIBLE_DEVICES=4

  ensures that only GPU4 is visible to the application. The GPU will also appear as cuda:0 to the application.

Understanding the remapping behavior of GPU visibility is essential for administrators managing multitenant environments. Because environment variables like CUDA_VISIBLE_DEVICES and ROCR_VISIBLE_DEVICES reindex visible GPUs starting from 0, administrators must track the logical-to-physical GPU mapping to ensure accurate monitoring, troubleshooting, and tenant-level usage accounting.

While CUDA_VISIBLE_DEVICES (for NVIDIA) and ROCR_VISIBLE_DEVICES (for AMD) effectively restrict GPU access within the local server, they do not control which **network interface** is used for inter-node communication. To maintain strict tenant isolation and avoid traffic leakage, additional environment variables must be set to control NIC selection. These include:

- UCX_NET_DEVICES
- NCCL_SOCKET_IFNAME
- NCCL_IB_HCA

These variables define the network interface(s) to be used by UCX and NCCL, ensuring that traffic remains within the tenant's routing instance and only uses the correct NICs.

The example shown in Figure 41 illustrates a multitenant configuration on a GPU server labeled **H100-01**, which contains eight GPUs (GPU0–GPU7) and eight corresponding NICs (NIC0–NIC7).

A **Tenant-1 NCCL job** is shown running on GPU0, isolated using the environment variable **CUDA_VISIBLE_DEVICES= GPU0**, ensuring the job only sees and accesses GPU0.

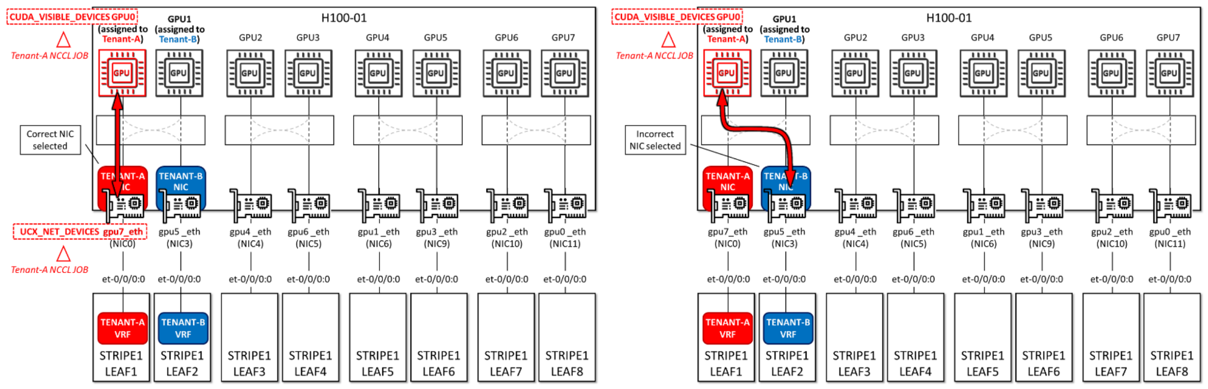
Because **GPUs 0 and 1 share NUMA locality with NIC6 and NIC8**, GPU0 can use either NIC6 or NIC8 to communicate with GPUs assigned to the same tenant on other servers. Without explicit control, it may select a NIC associated with a different tenant, violating traffic isolation. To prevent this, the job must also be restricted to **NIC6** (gpu0_eth) by setting: UCX_NET_DEVICES=gpu0_eth.

Failing to specify the correct NIC can result in communication failures or cross-tenant traffic leakage. In this example, NIC6 is connected to Tenant-1 VRF on the leaf node, while NIC NIC8 is connected to Tenant B's VRF.

The left side of Figure 41 shows a case where the correct NIC is selected, and therefore the traffic correctly exits on the interface connected to Tenant 1's routing instance.

The right side shows a case where the incorrect NIC is selected, and the traffic incorrectly exits on the interface connected to Tenant 2's routing instance.

Figure 41: GPU and NIC Isolation for Tenant-1 NCCL Job



For more details on NCCL and RCCL environment variables refer to the latest NVIDIA and AMD documentation. The latest at the time of this document's publication can be found here:

- Environment Variables — NCCL 2.27.5 documentation
- RCCL environment variables — RCCL 2.26.6 Documentation

Type 5 EVPN/VXLAN GPU Backend Fabric Implementation – Control Plane

IN THIS SECTION

The **underlay** serves as the IP transport between VXLAN Tunnel Endpoints (VTEPs), located at the leaf nodes, and provides IP reachability using **EBGP sessions**. These sessions are established between directly connected leaf and spine nodes and exchange unicast routes advertising the leaf nodes' loopback interfaces.

The **overlay** provides IP reachability between gpu-facing ethernet segments using multihop **EBGP sessions**. These sessions are established between the leaf and spine nodes using their loopback addresses and include the information required to encapsulate and forward tenant traffic across the fabric while maintaining traffic separation between customers.

EBGP is preferred in the overlay because it enforces loop-free, hop-by-hop forwarding without requiring route reflectors. By using unique ASNs per device, it aligns with Valley-Free Routing principles, ensuring traffic flows leaf to spine to leaf, avoiding loop and maintaining symmetry.

Fabric Underlay Control Plane Implementation Options

There are different options to implement the underlay in an EVPN/VXLAN fabric, depending on design goals, operational preferences, and hardware capabilities.

- IPv4 addresses (/31 subnet masks) numbered interfaces
- IPv6 addresses (/127 subnet masks) numbered interfaces
- IPv6 link-local addresses (unnumbered interfaces), with BGP neighbour auto-discovery based on IPV6 neighbour discovery (RFC 4861) and IPv4 route advertisement via IPv6 next-hops (RFC 5549) for IPv4 overlays.

Table 8. Comparison of Underlay Control Plane Implementation Options in EVPN/VXLAN Fabrics

Implementation Options	IPv4 /31	IPv6 /127	IPv6 Link-Local (RFC 4861) RECOMMENDED
Leaf to Spine Interface Addressing	Statically configured /31 IPv4 addresses	Statically configured routable (non-link local) /127 IPv6 addresses	**Automatically assigned Link-local IPv6 (no global addressing needed)**
BGP Peer Configuration	Explicit neighbor config per interface using IPv4 addresses of directly connected interface.	Explicit neighbor config per interface using routable IPv6 addresses of directly connected interface.	**No explicit neighbor config required** **Uses interface-scoped link-local discovery** **Link-local IPv6 + interface-scoped BGP config (fe80::1%et-0/0/0)**
Benefits	Simple Widely supported Low config overhead	Avoids IPv4 exhaustion IPv6-native underlay Aligns with modern fabrics	**Zero IP allocation needed** **Ideal for massive fabrics** **Minimal IPAM**
Drawbacks	No IPv6 ready	Needs dual-stack address planning	**Traceroute visibility reduced**

(Continued)

Implementation Options	IPv4 /31	IPv6 /127	IPv6 Link-Local (RFC 4861) <u>RECOMMENDED</u>
Use cases / Industry Trend	IPv4 /31 remains the most widely used in enterprise and service provider fabrics. For most enterprise and traditional data center fabrics, IPv4 /31 remains the recommended and most straightforward underlay option.	IPv6 /127 is gaining traction in dual-stack environments and in organizations preparing for IPv6 transitions. It conserves IPv4 space and offers a clean separation between infrastructure and tenant traffic.	IPv6 Link-Local / BGP auto-discovery is trending in hyperscaler, telco, and modern leaf-spine fabrics, especially where address scale or IPAM simplicity is critical. Many cloud providers (Azure, AWS internal fabrics, Meta, etc.) use unnumbered underlays. This option is becoming the preferred option for hyper scaling, IP-scarce, or automation-heavy fabrics with experienced ops teams, where managing IPs is a burden.

NOTE: This JVD focuses on the IPv6 link-local underlay, which is the preferred design choice. For details on implementing the underlay using statically configured /31 IPv4 addresses or statically configured routable (non-link-local) /127 IPv6 addresses, refer to "Appendix B – IPv4 Overlay over IPv4 Underlay Fabric Implementation" on page 217 and "Appendix C – IPv6 Overlay with Static Addresses Over IPv6 Underlay Fabric Implementation" on page 243, respectively.

In all three options, EBGP sessions are established between the leaf and spine nodes using the addresses assigned to the interfaces connecting the nodes (e.g. et-0/0/0.0). These sessions advertise the addresses assigned to the loopback interfaces, which will be used in the overlay.

Configuring **IPv6 with link-local addressing in the underlay** simplifies network design and reduces operational complexity, especially in large-scale fabrics where scalability and automation are key. By removing the need to assign and manage global IP addresses on leaf-to-spine links, this approach eliminates a major source of administrative overhead and prevents IPv4 address exhaustion. Each link automatically generates unique addresses, streamlining both deployment and ongoing maintenance. This is a significant advantage in environments with thousands of interfaces, where traditional IP management becomes a scaling bottleneck.

From a design and operational perspective, unnumbered underlays align perfectly with modern data center and AI fabric principles. They reduce configuration touch points, lower the chance of human error, and support dynamic, automation-friendly environments. For customers prioritizing agility, scalability, and simplified management, such as hyperscalers, telcos, and AI-driven infrastructures, IPv6 link-local fabrics not only meet today's technical requirements but also provide a future-proof foundation that supports growth without requiring continuous IP planning.

Control Plane Implementation with IPv6 Link-Local Underlay

The **underlay EBGP** sessions are set up using **unnumbered links,** also referred to as BGP auto-discovery or BGP auto-peering, which allows devices to dynamically discover directly connected neighbors and form BGP sessions using IPv6 link-local addresses. This design leverages Junos OS support for:

- RFC 4861: Neighbour Discovery for IP version 6 (IPv6)
- RFC 2462: IPv6 Stateless Address Autoconfiguration

Traditionally, BGP requires explicit configuration of neighbors, Autonomous System (AS) numbers, and routing policies to control route exchanges. With BGP unnumbered peering, neighbors are discovered dynamically, and BGP sessions are established automatically, eliminating the need for manual configuration and enabling faster, more scalable underlay deployments in EVPN/VXLAN data center fabrics.

The interfaces between leaf and spine nodes do not require explicitly configured IP addresses. It is sufficient to enable IPv6 (e.g. family inet6). Enabling IPv6 on an interface automatically assigns a link-local IPv6 address, which is then advertised through standard router advertisements as part of the IPv6 Neighbor Discovery process. This simplifies configuration and eliminates the need for manual IP addressing on leaf–spine links. All leaf and spine nodes are also configured with IPv6. addresses on the loopback interface (lo0.0).

Neighbor discovery uses standard IPv6 mechanisms to learn the link-local addresses of directly connected neighbors. These addresses are then used to automatically establish EBGP sessions.

The EBGP configuration for this model includes the local Autonomous System (AS) number, a list of accepted remote Autonomous System (AS) numbers, the list of interfaces where dynamic BGP neighbors with be accepted, and the export policy that allows the advertisement of routes to reach all the leaf and spine nodes in the fabric. These routes are standard IPv6 unicast advertising the IPv6 addresses assigned to the loopback interface (lo0.0). Peer-auto-discovery using IPv6-nd must also be enabled for the BGP group.

Although this approach requires some changes in the traditional way of configuring BGP, it offers significant operational advantages in highly scalable environments. By eliminating the need to assign and manage IP addresses on point-to-point links, this model simplifies IP planning and is ideal for large-scale, automated EVPN/VXLAN deployments.

Fabric Overlay Control Plane Implementation Options

Like the underlay, the overlay in an EVPN/VXLAN fabric can be implemented using either IPv4 or IPv6 addresses, depending on design goals, operational preferences, and hardware capabilities as summarized in table 9. Both options can be implemented over the recommended **IPv6 link-local addressing with BGP auto-discovery in the underlay.**

Table 9. Comparison of Overlay Control Plane Implementation Options in EVPN/VXLAN Fabrics

Implementation Options	**IPv4** **(RFC5549 with IPv6 underlay)**	**IPv6** **RECOMMENDED**
VTEP Tunnel Endpoint Addresses (leaf node loopback interface addresses)	Statically configured /32 IPv4 addresses	Statically configured routable (non-link local) /128 IPv6 addresses
Server to Leaf Nodes link prefix	/31 prefixes	/127 prefixes
Server addresses	Statically configured /31 IPv4 addresses	Statically configured /127 IPv6 addresses Autoconfigured /64 IPv6 addresses using SLAAC (RECOMMENDED)
Header Size (VXLAN + IP)	Lower overhead (UDP+IPv4 = ~28B)	Higher overhead (UDP+IPv6 = ~48B)
BGP Peer Configuration	Explicit neighbor config per interface using IPv4 addresses of directly connected interface. Requires	Explicit neighbor config per interface using routable IPv6 addresses of directly connected interface.

(Continued)

Implementation Options	IPv4 (RFC5549 with IPv6 underlay)	IPv6 RECOMMENDED
Benefits	Simple Widely supported	Avoids IPv4 exhaustion Scalability and future-proofing, especially pertinent to the demands of AI/ML data centers.

NOTE: This JVD focuses on the IPv6 overly, which is the preferred design choice.

Configuring **IPv6 in the overlay** ensures alignment with the IPv6-based underlay and avoids the operational complexity of running BGP sessions over mixed-protocol paths. When overlay loopbacks and EVPN sessions are IPv6-based, the routing model remains consistent across all layers, and control plane reachability does not require translation between IPv6 routes and IPv4 transport sessions. Choosing an IPv6 overlay also eliminates the need for dual-stack configurations, resulting in a cleaner deployment model and providing a foundation for extending IPv6-based services, such as SLAAC and per-tenant prefix assignment, across the fabric.

Using IPv4 in the overlay with IPv6 underlay requires the implementation of RFC5549 (Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop) which adds a layer of complexity of the solution. However, if IPv4 between the servers and leaf nodes is required, the details for this are covered in "Appendix B – IPv4 Overlay over IPv4 Underlay Fabric Implementation" on page 217.

Control Plane Implementation with IPv6 Overlay

EBGP sessions between leaf and spine nodes are established using the loopback IPv6 addresses advertised by the underlay, with BGP multihop enabled. These overlay sessions carry the IPv6 prefixes corresponding to the point-to-point links between GPU servers and leaf nodes.

On each leaf node, the interfaces connecting to the GPU servers are placed in tenant-specific IP VRFs. The associated IPv6 prefixes are then advertised as EVPN Type-5 routes, each containing a tenant-specific VNI. These routes provide Layer 3 reachability between GPUs assigned to the same tenant, even when distributed across multiple servers and racks, and are installed in the appropriate VRF routing tables.

EVPN Type 5 routes are used to advertise Layer 3 prefixes across the EVPN/VXLAN fabric without requiring destination MAC learning or IRB interfaces. These routes follow the BGP EVPN IP Prefix Route specifications described in RFC9136, and include the IPv6 prefix, route target, route distinguisher, and the VTEP next-hop information, enabling routed connectivity across the fabric. By decoupling routing from MAC learning, Type 5 routes simplify control plane operations and maintain clean tenant separation through BGP extended communities. Each route includes the information summarized in Table 10.

Table 10. EVPN Type 5 Route Fields Description

Field	Description
Route Type	IP Prefix Route (Type 5)
Route Distinguisher (RD)	RD of advertising PE (e.g., based on loopback IP) to make the route unique across the fabric
Ethernet Tag ID	0 (because it's not associated with a specific VLAN or MAC-VRF)
IP Prefix	The advertised IP prefix being advertised (e.g., FC00:1:1:1::/64)
Prefix Length	The length of the IP prefix (e.g. 64)
Label	VXLAN VNI (e.g., 1) identifying the virtual routing domain
Next hop	Loopback address of the advertising leaf node
Extended Community	**Route-target** to identify the associated tenant VRF (e.g., target:65000:1) **Router-mac** to identify the MAC address of the advertising VTEP,
Other BGP Attributes	BGP attributes like origin, AS-path, local-preference, etc.

Control Plane Summary

Table 11. Connections Summary

Option	GPU server to leaf node links	Leaf to spine node links	Leaf and spine nodes loopback interface addresses
IPv4 underlay and IPv4 overlay	Statically configured IPv4 address	Statically configured IPv4 addresses	Statically configured IPv4 addresses

(Continued)

Option	GPU server to leaf node links	Leaf to spine node links	Leaf and spine nodes loopback interface addresses
IPv6 underlay and IPv6 overlay	Statically configured IPv6 address	Statically configured IPv6 addresses	Statically configured IPv6 addresses
IPv6 Link-Local underlay and IPv4 overlay (RFC 5549)	Statically configured IPv4 address.	Automatically assigned IPv6 link local addresses	Statically configured IPv4 addresses
RECOMMENDED IPv6 Link-Local underlay and IPv6 overlay	**Dynamically assigned IPv6 address using SLAAC (Stateless Address Autoconfiguration)**	**Automatically assigned IPv6 link local addresses**	**Statically configured IPv6 address**

Table 12. EVPN/VXLAN options summary

Option	GPU server to leaf node links	Leaf to spine node links	Leaf and spine nodes loopback interface addresses	Underlay BGP sessions	Overlay BGP sessions
IPv4 underlay and IPv4 overlay	Statically configured IPv4 address	Statically configured IPv4 addresses	Statically configured IPv4 addresses	Statically configured IPv4 neighbors	Statically configured IPv4 neighbors using loopback interfaces.
IPv6 underlay and IPv6 overlay	Statically configured IPv6 address	Statically configured IPv6 addresses	Statically configured IPv6 addresses	Statically configured IPv6 neighbors	Statically configured IPv6 neighbors using loopback interfaces.
IPv6 Link-Local underlay and IPv4 overlay	Statically configured IPv4 address.	Automatically assigned IPv6 link local addresses	Statically configured IPv4 addresses	Automatically discovered IPv6 neighbors	Statically configured IPv4 neighbors using loopback interfaces.

(Continued)

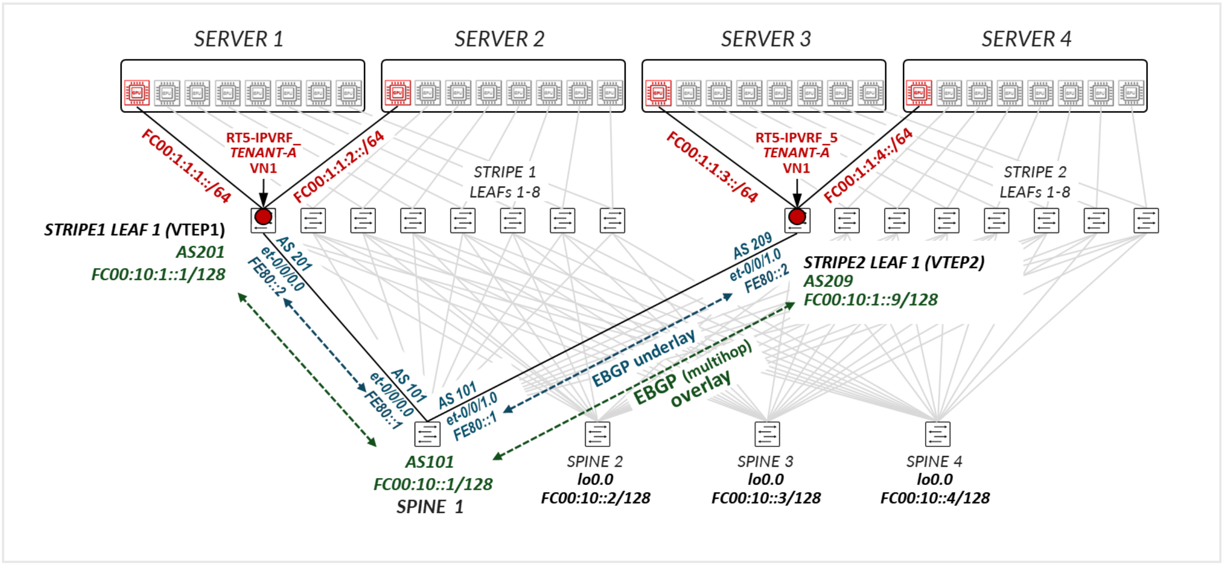
Option	GPU server to leaf node links	Leaf to spine node links	Leaf and spine nodes loopback interface addresses	Underlay BGP sessions	Overlay BGP sessions
RECOMMENDED IPv6 Link-Local underlay and IPv6 overlay	Dynamically assigned IPv6 address using SLAAC (Stateless Address Autoconfiguration)	Automatically assigned IPv6 link local addresses	Statically configured IPv6 address	Automatically discovered IPv6 neighbors using link local addresses.	Statically configured IPv6 neighbors using loopback interfaces.

Control Plane Implementation with IPv6 Link-Local IPv6 Underlay and IPv6 Overlay Example

Consider the example depicted in Figure 42.

For the underlay, STRIPE1 LEAF 1 in AS 201 automatically establishes an EBGP session with SPINE 1 in AS 101, over the directly connected link FE80::1 <=> FE80::2. Similarly, STRIPE2 LEAF 1 in AS 209 establishes an EBGP session with SPINE 1 over the link FE80::1 <=> FE80::2. These addresses are the link local addresses automatically assigned to the interfaces based on their MAC address, (shown here as FE80::1 and FE80::2 for simplicity), and are auto discovered by the BGP peers using standard IPv6 neighbor discover mechanisms.

Figure 42: IPv6 Link-Local Underlay and IPv6 Overlay Example



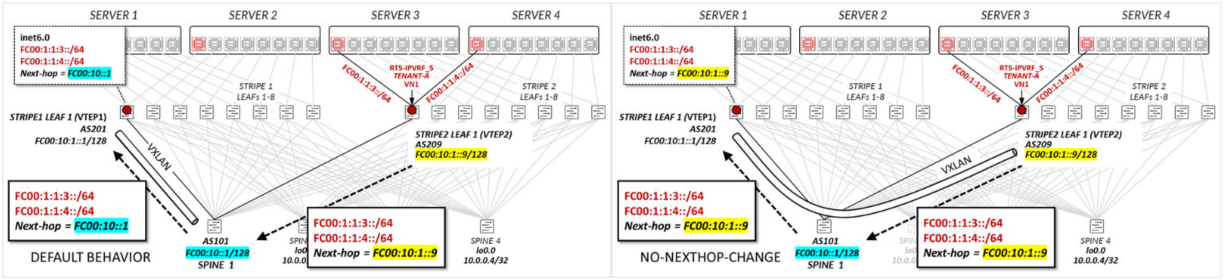
The underlay BGP sessions are configured to exchange IPv6 unicast routes and to advertise the addresses of the loopback interfaces (lo0.0) of STRIPE1 LEAF 1 (FC00:10::1:1), STRIPE2 LEAF 1 (FC00:10::1:9) and SPINE 1 (FC00:10::1). As a result, the leaf and the spine nodes have reachability to establish the EBGP overlay sessions. Once the overlay sessions are establish the leaf nodes, acting as VTEP, advertise the links facing the GPU servers as EVPN type 5 routes.

NOTE: Although it is not shown in the diagram, STRIPE1 LEAF 1 and STRIPE2 LEAF 1 will also establish EBGP sessions with SPINE 2, SPINE 3, and SPINE 4 to ensure multiple paths are available for traffic.

STRIPE1 LEAF 1 advertises the links connecting SERVER 1 GPU1 and SERVER 2 GPU1 (FC00:1:1:1::/64 and FC00:1:1:2::/64 respectively) to the spine nodes, which then advertise the routes to STRIPE2 LEAF 1. Similarly, STRIPE2 LEAF 1 advertises the links connecting SERVER 3 GPU1 and SERVER 4 GPU1 (FC00:1:1:3::/64 and FC00:1:1:4/64 respectively).

The spines are configured to maintain the next hop when advertising the routes received from the leaf nodes to other leaf nodes (**no-nexthop-change**). This allows VXLAN tunnels to be established between the leaf nodes, and not between the leaf and spine nodes.

Figure 43. Spine Route Readvertisements



Because all four GPUs in the example belong to the same tenant, their associated interfaces are mapped to the same VRF, RT5-IP-VRF_TENANT-1 which is configured on both STRIPE1 LEAF 1 and STRIPE2 LEAF 1 with the same VXLAN Network Identifier (VNI) and route targets.

STRIPE1 LEAF 1 advertises the prefixes FC00:1:1:1::/64 and FC00:1:12::/64 to SPINE 1 as EVPN Route Type 5, with its own loopback (FC00:10:1::1) as the next-hop VTEP. STRIPE2 LEAF 1 advertises FC00:1:1:3::/64 and FC00:1:1:4::/64 with its own loopback (FC00:10:1::9) as the next-hop.

When SERVER 1 GPU1 sends traffic to SERVER 3 GPU1, the destination addresses match the route FC00:1:1:4::/64 found in the VRF routing table on STRIPE1 LEAF 1 (Tenant-1 .inet.6). The route points to STRIPE2 LEAF 1 (VTEP FC00:10:1::9) as the protocol next-hop (which is resolved to the link local addresses of the spine nodes). The route also specifies VNI 1 as the VXLAN encapsulation ID. The packet is encapsulated with the VXLAN header and tunneled across the fabric to its destination.

Type 5 EVPN/VXLAN GPU Backend Implementation – Forwarding Plane

IN THIS SECTION

Each VTEP (VXLAN Tunnel Endpoint) is responsible for encapsulating and de-encapsulating traffic as it enters and exits the VXLAN fabric. In the context of a GPU multitenancy design, these VTEPs are located at the leaf nodes of the network, where tenant workloads (including GPU-accelerated compute instances) are hosted. Each VTEP maps tenant-specific traffic into the appropriate VXLAN segment, maintaining isolation and enabling east-west communication across the fabric.

The VTEPs are responsible for encapsulating and de-encapsulating traffic as it enters and exits the VXLAN fabric.

In the context of a GPU multitenancy design, these VTEPs are located at the leaf nodes of the network.

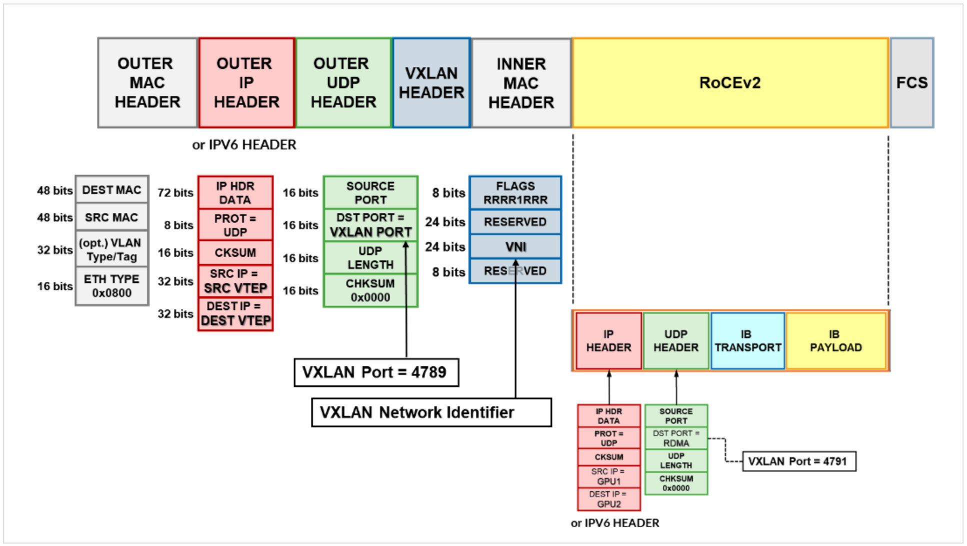
RoCEv2 traffic encapsulation

RoCEv2 (RDMA over Converged Ethernet version 2) traffic can be transported across an Ethernet-based IP network using VXLAN encapsulation, which allows RDMA workloads to operate across Layer 3 boundaries while preserving performance and scalability. In this model, the original RDMA payload is encapsulated inside a VXLAN packet, which is further wrapped in standard UDP and IP headers, enabling transport across IP-based fabrics.

The encapsulation begins with the original RoCEv2 payload, which consists of InfiniBand headers and data. This is encapsulated in a VXLAN header, where the VXLAN Network Identifier (VNI) uniquely identifies the Layer 2 segment associated with the RDMA flow. The VXLAN header is prepended by a UDP header (with a destination port typically set to 4789), allowing the packet to traverse standard IP networks without requiring special handling.

The outer IP header carries the source and destination IP addresses of the VTEPs (VXLAN Tunnel Endpoints), and the outer MAC header ensures correct delivery across the Ethernet fabric. **Importantly, the outer and inner IP headers are independent; each can be either IPv4 or IPv6, and they do not need to match.** For example, it is entirely valid to encapsulate an IPv6-based RoCEv2 flow within an IPv4 VXLAN tunnel, or vice versa, depending on the underlay and overlay configurations. All testing related to this JVD was completed using RoCEv2 over IPv6.

Figure 43: RDMA Encapsulation over IPv4/IPv6



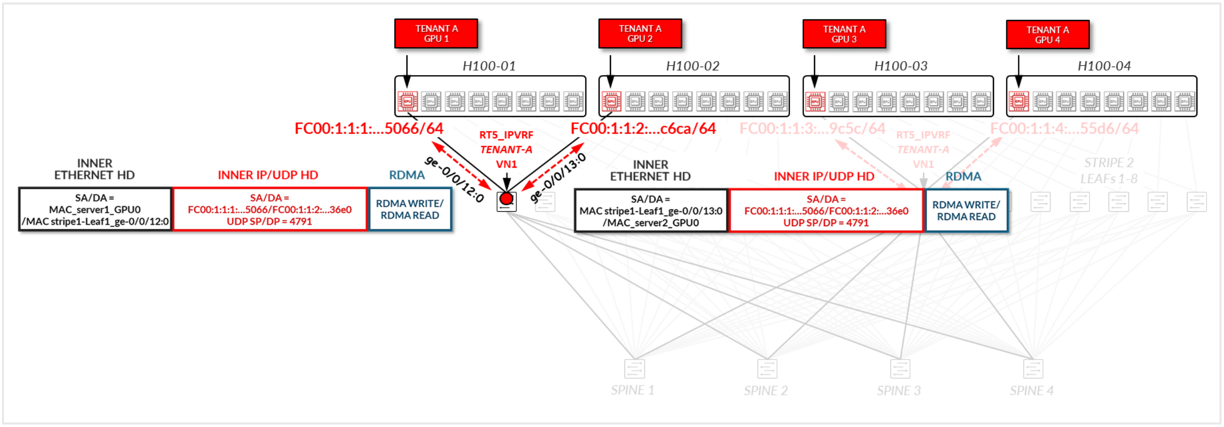
As described in the example in the previous section, the server-facing interfaces on the leaf nodes are configured as Layer 3 routed interfaces and are mapped into a tenant-specific IP-VRF. Tenant A has been assigned the first GPU on servers 1 through 4 (namely, GPU1 to GPU4). The interfaces connecting these GPUs are associated with an IP-VRF named Tenant A.

GPU1 and GPU2 (on servers 1 and 2) are connected to the same leaf node (Stripe 1, Leaf 1) and are mapped to the Tenant A VRF. Likewise, GPU3 and GPU4 (on servers 3 and 4) are connected to a different leaf node (Stripe 2, Leaf 1) and are also mapped to the same VRF. Communication between GPUs connected to the same leaf node occurs locally, while traffic between GPUs on different leaf nodes is routed across the fabric using the outer IP header added during VXLAN encapsulation, as described earlier.

RoCEv2 Traffic Flows Across the Fabric

Consider the example in Figure 44 which shows RoCEv2 traffic flows between 4 GPU,s on 4 different servers, that are assigned to Tenant-1

Figure 44: RoCEv2 Traffic Flow Across the Fabric



Traffic between Server H100-01 GPU0 (Tenant's GPU1) and **Server H100-02 GPU0 (Tenant's GPU2) is Switched Locally at the Leaf Node**

1. **Traffic Origination:**

   GPU 0 on Server 1 initiates a RoCEv2 **RDMA WRITE** targeting GPU 0 on Server 2.

   RoCEv2 packets are encapsulated in UDP over IP as any other IP traffic.

   The source and destination IP addresses are the autoconfigured IPv6 addresses associated with each GPU (FC00:1:1:1:a288:c2ff:fe3b:55d6 and FC00:1:1:2:5aa2:e1ff:fe46:c6ca), while the source and destination MAC addresses correspond to the MAC address of the NICs associated with GPU0 Server 1 and the MAC address of interface et-0/0/12.0 on Stripe1-leaf1.

2. **Leaf Forwarding/Delivery to Tenant's GPU 2:**

   The leaf node simply strips off the L2 Header, performs a route lookup in the tenants specific routing table (TENANT-1 _VRF), and re-encapsulates the packet with a new L2 header with source and destination MAC addresses corresponding to the Leaf's MAC addresses on interfaces et-0/0/13.0, and the MAC address of the NIC associated with GPU 0 on Server 2.

NOTE: Traffic between Server 1 and Server 4 (same tenant, same leaf) is handled in the same way.

Traffic Between Server H100-01 GPU0 (Tenant's GPU1) and **Server H100-03 GPU0 (Tenant's GPU3) has to be Encapsulated in VXLAN and Forwarded Across the Fabric**

1. **Traffic Origination:**

   GPU 0 on Server 1 initiates a RoCEv2 RDMA WRITE targeting GPU 0 on Server 2.

   RoCEv2 packets are encapsulated in UDP over IP as any other IP traffic.

   The source and destination IP addresses are the autoconfigured IPv6 addresses associated with each GPU (FC00:1:1:1:a288:c2ff:fe3b:55d6 and FC00:1:1:3:966d:aeff:fef5:9c5c), while the source and destination MAC addresses correspond to the MAC address of the NICs associated with GPU0 Server 1 and the MAC address of interface et-0/0/12.0 on Stripe1-leaf1.

2. **Source Leaf Forwarding:**

   Stripe1-Leaf 1 strips off the L2 Header, performs a route lookup in the tenants specific routing table (**TENANT-1 _VRF).** The route to the destination in this case, which was installed in this VRF via **EVPN Type 5 route advertisement**, points to loopback interface of Stripe2 leaf 1, and indicates the traffic needs to be encapsulated using VXLAN.

   The leaf re-encapsulates the packet in **VXLAN**, using a tenant-specific VNI, and the remote VTEP MAC address that was received with the EVPN type 5 route. An additional IP and UDP header (outer header) and a new L2 header are added to the packet. with the MAC addresses of Stripe 1 Leaf 1 and Spine1 as source and destination addresses.

   The source and destination IP addresses will be the loopback interface addresses of Stripe 1 Leaf 1 and Stripe 2 Leaf 1. The source and destination MAC addresses will be the MAC addresses of Stripe 1 Leaf 1 and Spine 1.

   NOTE: Spine 1 is used here as an example. Traffic will be load-balanced across all leaf–spine links in the fabric as will be reviewed later.

3. **Spine (Intermediate) Forwarding:**

   Spine 1 is not aware of VXLAN encapsulation and simply route the packets based on the outer IP header, in this case towards stripe 2 leaf 1. The outer header source and destination IP addresses are not modified.

4. **Destination Leaf Forwarding/Delivery to Tenant's GPU 3:**

   Stripe2-Leaf 1 receives the VXLAN packet and **decapsulates** the packet. It extracts the VNI number from the VXLAN header to determine the proper routing table for the arriving packet. Since VNI =1 is mapped to TENANT-1 _VRF, the leaf performs a router lookup in the corresponding table, which indicates that the destination is directly connected on interface et-0/0/12.0

The leaf node applies a new L2 header with source and destination MAC addresses corresponding to the Leaf and the NIC connected to the tenants GPU, and forwards the packet to the destination GPU.

NOTE: The forwarding process works the same way when servers NICs are configured with IPv4 addresses.

Type 5 EVPN/VXLAN GPU Backend Fabric, SLAAC IPv6 Overlay over IPv6 Link-Local Underlay - Configuration

IN THIS SECTION

This section outlines the configuration and verification steps to implement an **EVPN/VXLAN fabric** with:

- IPv6 GPU server NICs to Leaf Nodes connections using SLAAC
- IPv6 Leaf Nodes to Spine Nodes connections using link local addresses
- IPv6 GPU Backend Fabric underlay using BGP neighbor discovery
- IPv6 GPU Backend Fabric overlay
- Per Tenant IP-VRF Routing Instance

NOTE: Details on how to implement **IPv6 underlay/IPv4 overlay fabric** (RFC5549), **IPv4 underlay/IPv4 overlay fabric**, and **IPv6 underlay without SLAAC/IPv6 overlay fabric**, have been included in " **Appendix A** " on page 217, " **Appendix B** " on page 243, and " **Appendix C** " on page 243 respectively.

Consider the following scenario of a GPU-isolation implementation where:

TENANT	SERVER	ASSIGNED GPUS
Tenant-1	SERVER 1, SERVER 2, SERVER 3 SERVER 9, SERVER 10, SERVER 11	GPU0
Tenant-2	SERVER 1, SERVER 2, SERVER 3 SERVER 9, SERVER 10, SERVER 11	GPU1

Figure 45: Server-Isolation Example with Servers Across Multiple Stripes – Stripe 1

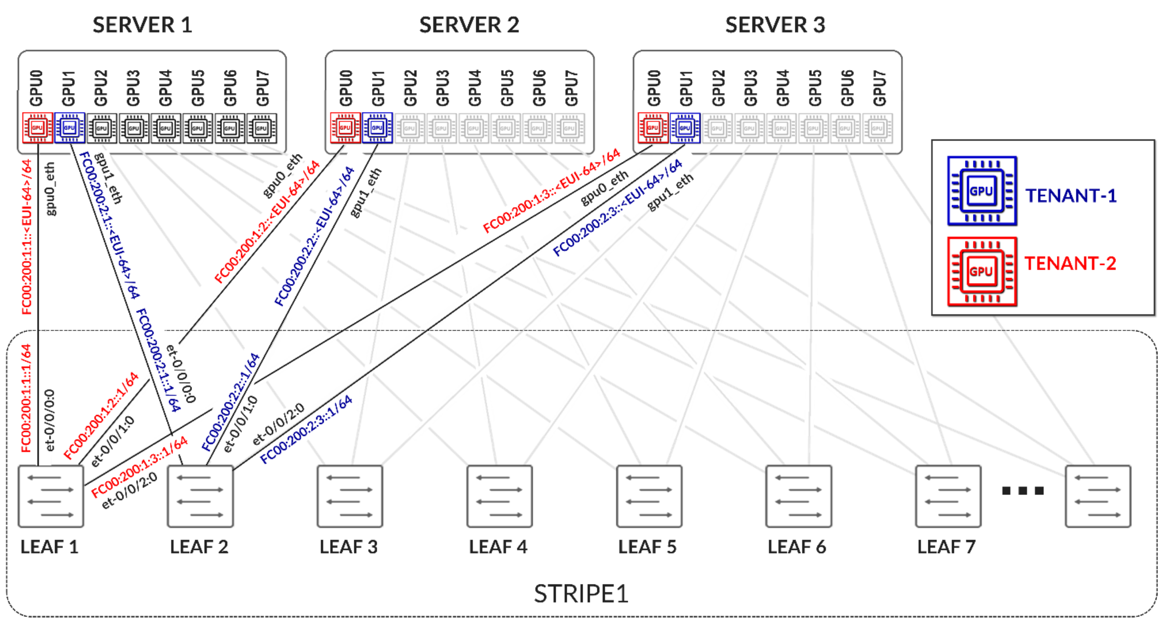
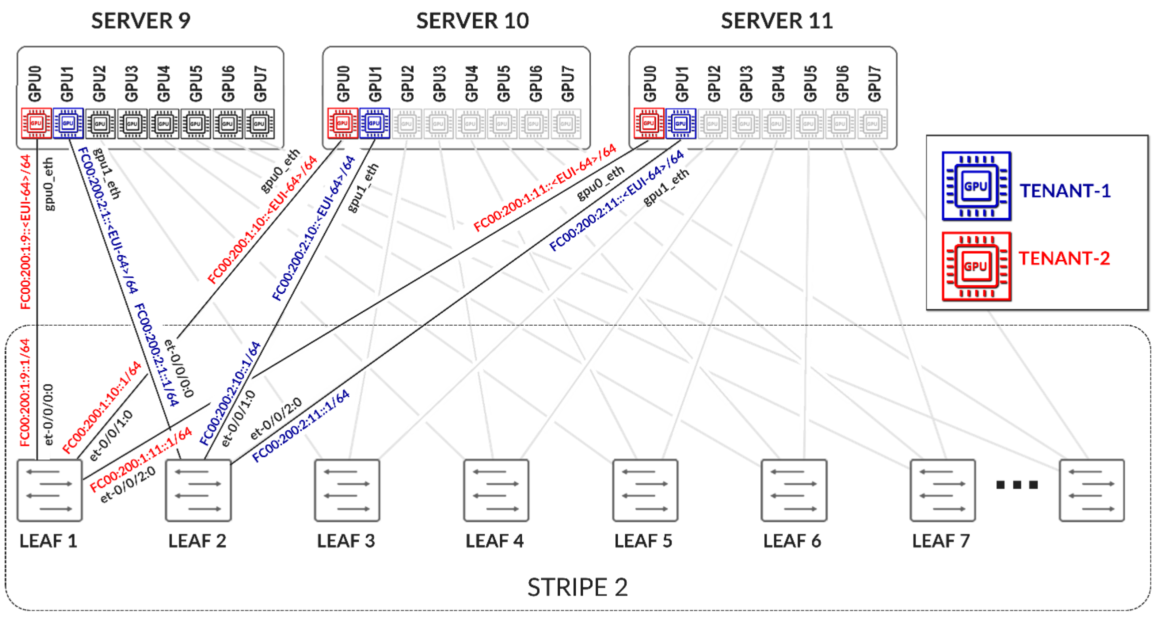


Figure 46: Server-Isolation Example with Servers Across Multiple Stripes – Stripe 2

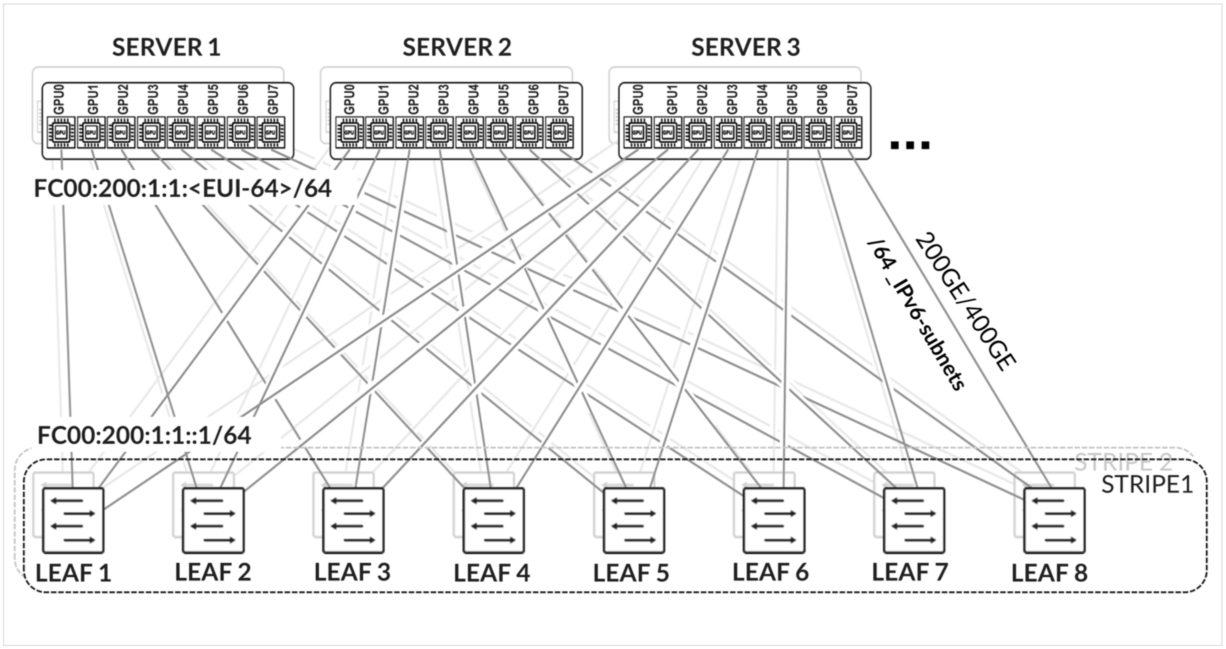


IPv6 GPU Server NICs to Leaf Nodes Connections Using SLAAC

This section describes the operation of SLAAC in the context of this solution, and then will the configuration and verification steps on both the servers and the Leaf nodes.

The GPU servers are connected to the leaf nodes following a rail-aligned architecture as described in the Backend GPU Rail Optimized Stripe Architecture section, where GPU 0 is connected to the first Leaf node, GPU 1 is connected to the second leaf node and so on. This is shown in Figure 47.

Figure 47. GPU Servers Rail-Aligned Connectivity



Each server to leaf node link is configured as an untagged L3 link and a statically configured /64 IPv6 address, while the server interface is autoconfigured using SLAAC to support scalable and automated IPv6 address assignment.

Each Tenant is assigned a /56 address, which will be used to derive /64 for each server to leaf node connection corresponding to the tenant. Tables 13 and 14 show the address assignment.

Table 13. Tenants /56 Prefixes Example

TENANT	/56 IPv6 Prefix
Tenant-1	FC00:200:1::/56
Tenant-2	FC00:200:2::/56
Tenant-3	FC00:200:3::/56
Tenant-4	FC00:200:4::/56
Tenant-5	FC00:200:5::/56
Tenant-6	FC00:200:6::/56
Tenant-7	FC00:200:7::/56

(Continued)

TENANT	/56 IPv6 Prefix
Tenant-8	FC00:200:8::/56
. . .	

Table 14. Tenants /64 Prefixes Example

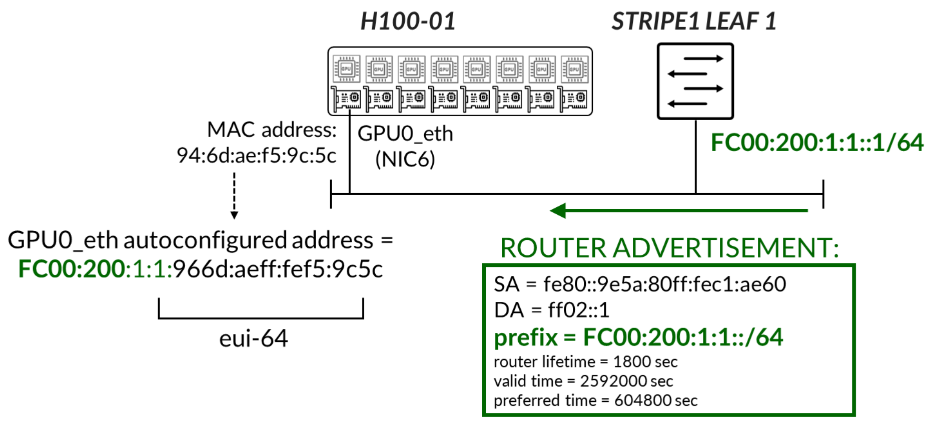
TENANT-1		TENANT-2		...
Server to leaf Link	/64 Prefix	Server to leaf Link	/64 Prefix	
SERVER 1 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:1:1::/64	SERVER 1 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:2:1::/64	
SERVER 2 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:1:2::/64	SERVER 2 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:2:2::/64	
SERVER 3 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:1:3::/64	SERVER 3 gpu0_eth ó Stripe 1 Leaf 1	FC00:200:2:3::/64	
. . .		. . .		
SERVER 9 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:1:9::/64	SERVER 9 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:2:9::/64	
SERVER 10 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:1:10::/64	SERVER 10 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:2:10::/64	
SERVER 11 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:1:11::/64	SERVER 11 gpu0_eth ó Stripe 2 Leaf 1	FC00:200:2:11::/64	

(Continued)

TENANT-1		TENANT-2		...
. . .		. . .		

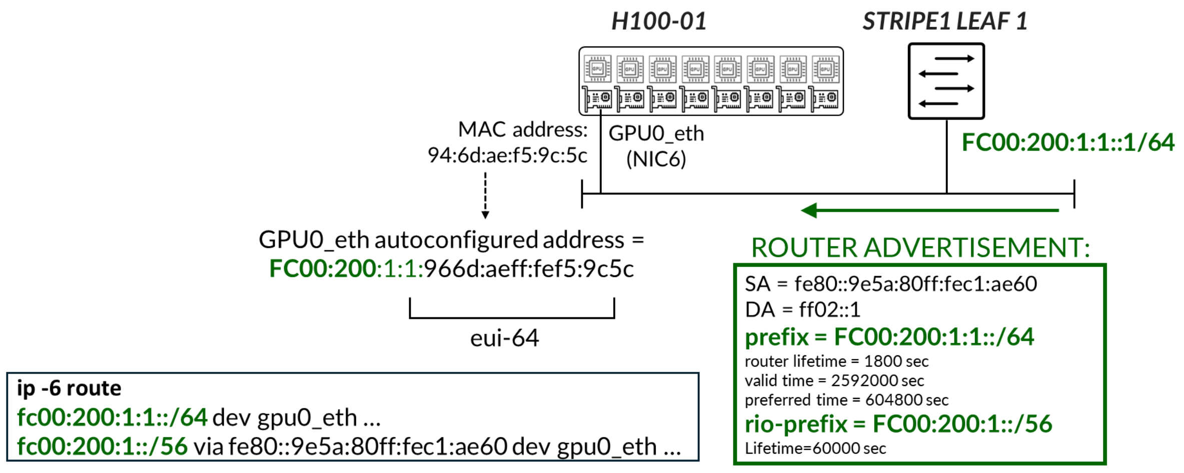
Each leaf node advertises a /64 IPv6 prefix, which is accepted by the server interface and used to automatically derive the interface's IPv6 address through its EUI-64 identifier (based on the interface's MAC address), as shown in Figure 48. This approach eliminates the need for DHCPv6 or manual configuration on the servers.

Figure 48: SLAAC – Stateless Address Autoconfiguration Operation Example – Tenant 1.


H100-01
STRIPE1 LEAF 1
MAC address: 94:6d:ae:f5:9c:5c
GPU0_eth (NIC6)
FC00:200:1:1::1/64
GPU0_eth autoconfigured address = FC00:200:1:1:966d:aeff:fef5:9c5c
eui-64
ROUTER ADVERTISEMENT:
SA = fe80::9e5a:80ff:fec1:ae60
DA = ff02::1
prefix = FC00:200:1:1::/64
router lifetime = 1800 sec
valid time = 2592000 sec
preferred time = 604800 sec

The leaf node must also advertise the tenant's /56 prefix using the Route Information Option (RIO) in IPv6 router advertisement messages as shown in figure 48. This provides the routing information required for a given GPU interface to communicate with remote GPU interfaces assigned to the same tenant.

Figure 49: SLAAC – Stateless Address Autoconfiguration with RIO-Prefix Operation Example – Tenant 1


H100-01
STRIPE1 LEAF 1
MAC address:
94:6d:ae:f5:9c:5c
GPU0_eth
(NIC6)
FC00:200:1:1::1/64
GPU0_eth autoconfigured address =
FC00:200:1:1:966d:aeff:fef5:9c5c
eui-64
ROUTER ADVERTISEMENT:
SA = fe80::9e5a:80ff:fec1:ae60
DA = ff02::1
prefix = FC00:200:1:1::/64
router lifetime = 1800 sec
valid time = 2592000 sec
preferred time = 604800 sec
rio-prefix = FC00:200:1::/56
Lifetime=60000 sec

```
ip -6 route
fc00:200:1:1::/64 dev gpu0_eth ...
fc00:200:1::/56 via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth ...
```

Without this option the server installs a default route pointing to the leaf node link local interface, for each RA received.

In the example shown in Figure 49, H100-01 and H100-02 provide GPU isolation for eight different tenants, with the following /56 prefix assignments:

Table 15. Tenants /56 Prefixes per Tenant Example

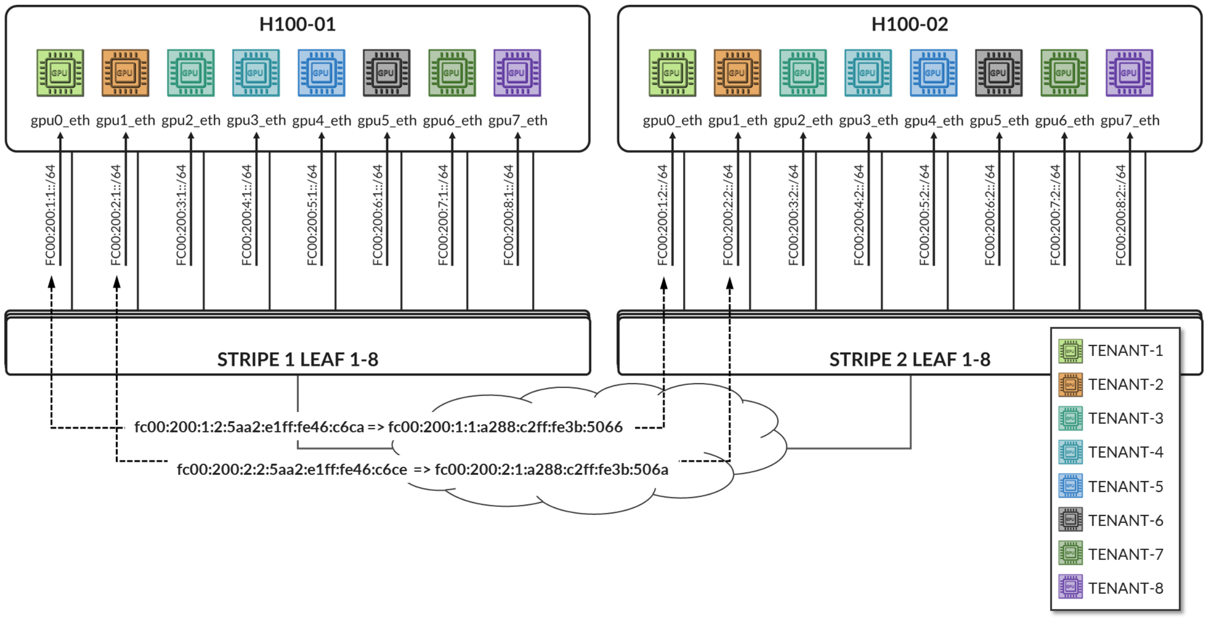
TENANT	/56 Prefix
TENANT-1	FC00:200:1::/56
TENANT-2	FC00:200:2::/56
TENANT-3	FC00:200:3::/56
TENANT-4	FC00:200:4::/56
TENANT-5	FC00:200:5::/56
TENANT-6	FC00:200:6::/56
TENANT-7	FC00:200:7::/56
TENANT-8	FC00:200:8::/56

(Continued)

TENANT	/56 Prefix
. . .	

All interfaces associated with Tenant-1 will have IPv6 addresses derived from FC00:200:1::/56, all interfaces associated with Tenant-2 will have IPv6 addresses from FC00:200:2::/56, and so on, as shown in Figure 50.

Figure 50. Multitenancy GPU-Isolation Example



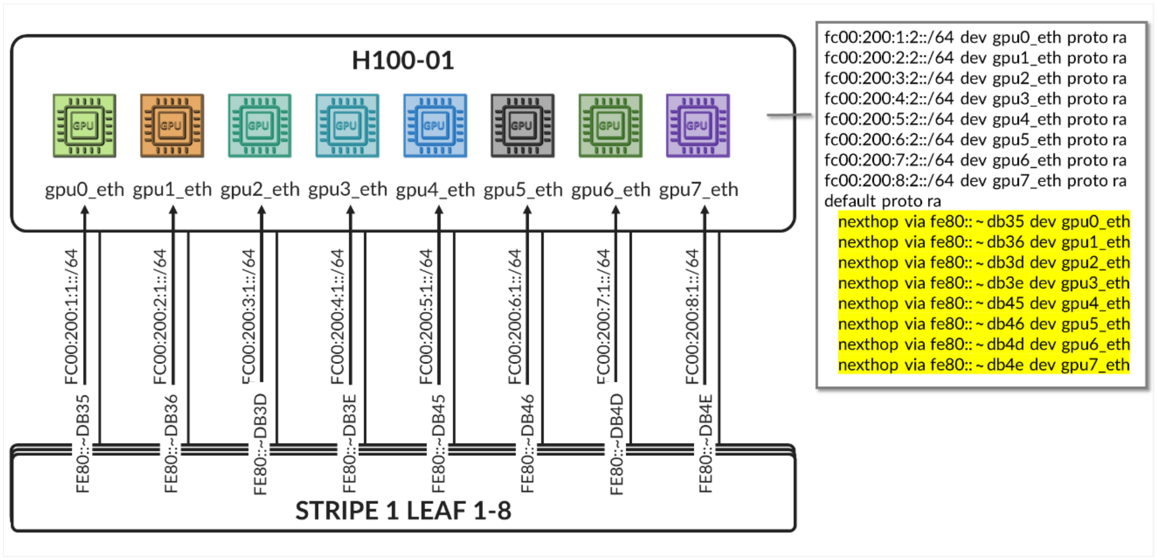
Initially, the leaf nodes are configured to advertise only the /64 prefixes. For example, Stripe 1 Leaf 1 advertises FC00:200:1:1::/64 to gpu0_eth server H100-01, while Stripe 2 Leaf 1 advertises FC00:200:1:2::/64 to gpu0_eth server H100-02. The two prefixes are derived from the FC00:200:1::/56 assigned to Tenant-1.

The servers automatically configure their IPv6 addresses from these advertised prefixes, as shown below:

```
jnpr@H100-01:~$ ifconfig | egrep "gpu|fc00"
gpu0_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:1:1:a288:c2ff:fe3b:5066 prefixlen 64  scopeid 0x0<global>
```

```
gpu1_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:2:1:a288:c2ff:fe3b:506a prefixlen 64  scopeid 0x0<global>
gpu2_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:3:1:a288:c2ff:fe3b:506e prefixlen 64  scopeid 0x0<global>
gpu3_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:4:1:a288:c2ff:fe3b:5072 prefixlen 64  scopeid 0x0<global>
gpu4_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:5:1:a288:c2ff:fe0a:7948 prefixlen 64  scopeid 0x0<global>
gpu5_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:6:1:a288:c2ff:fe0a:794c prefixlen 64  scopeid 0x0<global>
gpu6_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:6:1:a288:c2ff:fe0a:7940 prefixlen 64  scopeid 0x0<global>
gpu7_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:6:1:a288:c2ff:fe0a:7944 prefixlen 64  scopeid 0x0<global>
jnpr@H100-02:~$ ifconfig | egrep "gpu0|gpu1|fc00"
gpu0_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:1:2:5aa2:e1ff:fe46:c6ca prefixlen 64  scopeid 0x0<global>
gpu1_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:2:2:5aa2:e1ff:fe46:c6ce prefixlen 64  scopeid 0x0<global>
gpu2_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:1:2:5aa2:e1ff:fe46:c6d2  prefixlen 64  scopeid 0x0<global>
gpu3_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:2:2:5aa2:e1ff:fe46:c6d6  prefixlen 64  scopeid 0x0<global>
gpu4_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:1:2:5aa2:e1ff:fe46:c372  prefixlen 64  scopeid 0x0<global>
gpu5_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:2:2:5aa2:e1ff:fe46:c376 prefixlen 64  scopeid 0x0<global>
gpu6_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:1:2:5aa2:e1ff:fe46:c36a prefixlen 64  scopeid 0x0<global>
gpu7_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
        inet6 fc00:200:2:2:5aa2:e1ff:fe46:c36e prefixlen 64  scopeid 0x0<global>
```

At this time, the routing tables on both servers include default routes pointing to the link local addresses learned from the router advertisements.



```
fc00:200:1:2::/64 dev gpu0_eth proto ra
fc00:200:2:2::/64 dev gpu1_eth proto ra
fc00:200:3:2::/64 dev gpu2_eth proto ra
fc00:200:4:2::/64 dev gpu3_eth proto ra
fc00:200:5:2::/64 dev gpu4_eth proto ra
fc00:200:6:2::/64 dev gpu5_eth proto ra
fc00:200:7:2::/64 dev gpu6_eth proto ra
fc00:200:8:2::/64 dev gpu7_eth proto ra
default proto ra
    nexthop via fe80::~db35 dev gpu0_eth
    nexthop via fe80::~db36 dev gpu1_eth
    nexthop via fe80::~db3d dev gpu2_eth
    nexthop via fe80::~db3e dev gpu3_eth
    nexthop via fe80::~db45 dev gpu4_eth
    nexthop via fe80::~db46 dev gpu5_eth
    nexthop via fe80::~db4d dev gpu6_eth
    nexthop via fe80::~db4e dev gpu7_eth
```

```
jnpr@H100-01:~$ ip -6 route
::1 dev lo proto kernel metric 256 pref medium
fc00:200:1:1::/64 dev gpu0_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:2:1::/64 dev gpu1_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:3:1::/64 dev gpu2_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:4:1::/64 dev gpu3_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:5:1::/64 dev gpu4_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:6:1::/64 dev gpu5_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:7:1::/64 dev gpu6_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:8:1::/64 dev gpu7_eth proto ra metric 1024 expires 2591984sec pref medium
fe80::/64 dev stor0_eth proto kernel metric 256 pref medium
fe80::/64 dev mgmt_eth proto kernel metric 256 pref medium
fe80::/64 dev eno3 proto kernel metric 256 pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu1_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu2_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu3_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu4_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu5_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu6_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu7_eth proto kernel metric 256 pref medium
default proto ra metric 1024 expires 1749sec pref medium
        nexthop via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae61 dev gpu1_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae68 dev gpu2_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae69 dev gpu3_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae70 dev gpu4_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae71 dev gpu5_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae78 dev gpu6_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae88 dev gpu7_eth weight 1

jnpr@H100-02:~$ ip -6 route
::1 dev lo proto kernel metric 256 pref medium
fc00:200:1:2::/64 dev gpu0_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:2:2::/64 dev gpu1_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:3:2::/64 dev gpu2_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:4:2::/64 dev gpu3_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:5:2::/64 dev gpu4_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:6:2::/64 dev gpu5_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:7:2::/64 dev gpu6_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:8:2::/64 dev gpu7_eth proto ra metric 1024 expires 2591885sec pref medium
fe80::/64 dev mgmt_eth proto kernel metric 256 pref medium
fe80::/64 dev eno3 proto kernel metric 256 pref medium
fe80::/64 dev stor0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu1_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu2_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu3_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu4_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu5_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu6_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu7_eth proto kernel metric 256 pref medium
default proto ra metric 1024 expires 1685sec pref medium
       nexthop via fe80::5884:70ff:fe79:db35 dev gpu0_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db36 dev gpu1_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db3d dev gpu2_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db3e dev gpu3_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db45 dev gpu4_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db46 dev gpu5_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db4d dev gpu6_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db4e dev gpu7_eth weight 1
```

Traffic originating from **gpu0_eth** on H100-01 can successfully reach **gpu0_eth** on H100-02. However, traffic from **gpu1_eth** on H100-01 to **gpu1_eth** on H100-02 cannot. This is because, in both cases, the server selects the same default route, via **gpu0_eth**.

```
jnpr@H100-01:~$
ping fc00:200:1:2:5aa2:e1ff:fe46:c6ca -I fc00:200:1:1:a288:c2ff:fe3b:5066 -c 5
PING fc00:200:1:2:5aa2:e1ff:fe46:c6ca(fc00:200:1:2:5aa2:e1ff:fe46:c6ca) from
```

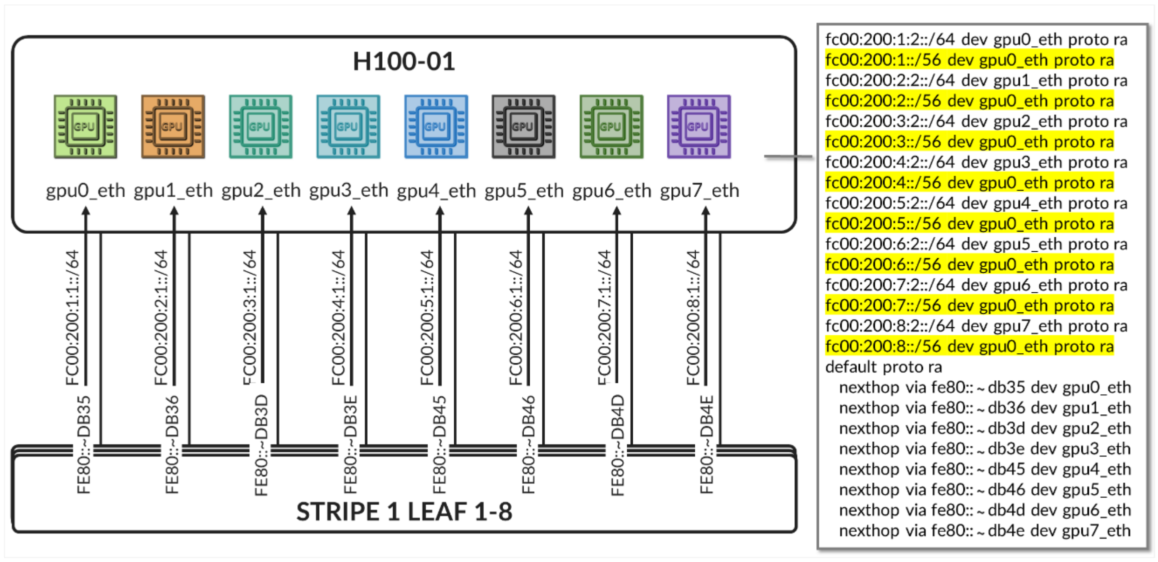
```
fc00:200:1:1:a288:c2ff:fe3b:5066 : 56 data bytes
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=1 ttl=63 time=0.231 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=2 ttl=63 time=0.310 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=3 ttl=63 time=0.322 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=4 ttl=63 time=0.344 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=5 ttl=63 time=0.275 ms
--- fc00:200:1:2:5aa2:e1ff:fe46:c6ca ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4102ms
rtt min/avg/max/mdev = 0.231/0.296/0.344/0.039 ms
jnpr@H100-01:~$
ping fc00:200:2:2:5aa2:e1ff:fe46:c6ce -I fc00:200:2:1:a288:c2ff:fe3b:506a -c 5
PING fc00:200:2:2:5aa2:e1ff:fe46:c6ce(fc00:200:2:2:5aa2:e1ff:fe46:c6ce) from
fc00:200:2:1:a288:c2ff:fe3b:506a : 56 data bytes
--- fc00:200:2:2:5aa2:e1ff:fe46:c6ce ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4090ms
```

After enabling **RIO-prefix** advertisements, the leaf nodes not only advertise the /64 prefixes that the servers will use to autoconfigure their addresses, but also the /56 prefix assigned to the tenant. As a result, the servers install the /56 prefixes in their routing tables, pointing to the correct interfaces, and use these routes instead of the default route to reach any destination within the /56 prefix.

In the example, Stripe-1 Leaf-1 and Stripe-2 Leaf-1 advertise **FC00:200:1:1::/64** and **FC00:200:1:2::/64** to **gpu0_eth** on H100-01 and H100-02, respectively, and advertise the **FC00:200:1::/56** prefix.

In the same way, Stripe-1 Leaf-1 and Stripe-2 Leaf-1 advertise **FC00:200:2:1::/64** and **FC00:200:2:2::/64** to **gpu1_eth** on H100-01 and H100-02, respectively, and advertise the **FC00:200:2::/56** prefix.

As a result, the servers also install /56 routes in their routing tables, pointing to the correct interface. The servers then use these routes instead of the default route to reach any destination corresponding to Tenant-1 (**FC00:200:1::/56) via gpu0_eth**, and any destination corresponding to Tenant-2 (**FC00:200:2::/56) via gpu1_eth**.



```
fc00:200:1:2::/64 dev gpu0_eth proto ra
fc00:200:1::/56 dev gpu0_eth proto ra
fc00:200:2:2::/64 dev gpu1_eth proto ra
fc00:200:2::/56 dev gpu0_eth proto ra
fc00:200:3:2::/64 dev gpu2_eth proto ra
fc00:200:3::/56 dev gpu0_eth proto ra
fc00:200:4:2::/64 dev gpu3_eth proto ra
fc00:200:4::/56 dev gpu0_eth proto ra
fc00:200:5:2::/64 dev gpu4_eth proto ra
fc00:200:5::/56 dev gpu0_eth proto ra
fc00:200:6:2::/64 dev gpu5_eth proto ra
fc00:200:6::/56 dev gpu0_eth proto ra
fc00:200:7:2::/64 dev gpu6_eth proto ra
fc00:200:7::/56 dev gpu0_eth proto ra
fc00:200:8:2::/64 dev gpu7_eth proto ra
fc00:200:8::/56 dev gpu0_eth proto ra
default proto ra
   nexthop via fe80::~db35 dev gpu0_eth
   nexthop via fe80::~db36 dev gpu1_eth
   nexthop via fe80::~db3d dev gpu2_eth
   nexthop via fe80::~db3e dev gpu3_eth
   nexthop via fe80::~db45 dev gpu4_eth
   nexthop via fe80::~db46 dev gpu5_eth
   nexthop via fe80::~db4d dev gpu6_eth
   nexthop via fe80::~db4e dev gpu7_eth
```

```
jnpr@H100-01:~$ ip -6 route
::1 dev lo proto kernel metric 256 pref medium
fc00:200:1:1::/64 dev gpu0_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:1::/56 via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:2:1::/64 dev gpu1_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:2::/56 via fe80::9e5a:80ff:fec1:ae61 dev gpu1_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:3:1::/64 dev gpu2_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:3::/56 via fe80::9e5a:80ff:fec1:ae68 dev gpu2_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:4:1::/64 dev gpu3_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:4::/56 via fe80::9e5a:80ff:fec1:ae69 dev gpu3_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:5:1::/64 dev gpu4_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:5::/56 via fe80::9e5a:80ff:fec1:ae70 dev gpu4_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:6:1::/64 dev gpu5_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:6::/56 via fe80::9e5a:80ff:fec1:ae71 dev gpu5_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:7:1::/64 dev gpu6_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:7::/56 via fe80::9e5a:80ff:fec1:ae78 dev gpu6_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:8:1::/64 dev gpu7_eth proto ra metric 1024 expires 2591984sec pref medium
fc00:200:8::/56 via fe80::9e5a:80ff:fec1:ae88 dev gpu7_eth proto ra metric 100 expires 59841sec
pref medium
fe80::/64 dev stor0_eth proto kernel metric 256 pref medium
```

```
fe80::/64 dev mgmt_eth proto kernel metric 256 pref medium
fe80::/64 dev eno3 proto kernel metric 256 pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu1_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu2_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu3_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu4_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu5_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu6_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu7_eth proto kernel metric 256 pref medium
default proto ra metric 1024 expires 1749sec pref medium
        nexthop via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae61 dev gpu1_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae68 dev gpu2_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae69 dev gpu3_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae70 dev gpu4_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae71 dev gpu5_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae78 dev gpu6_eth weight 1
        nexthop via fe80::9e5a:80ff:fec1:ae88 dev gpu7_eth weight 1

jnpr@H100-02:~$ ip -6 route
::1 dev lo proto kernel metric 256 pref medium
fc00:200:1:2::/64 dev gpu0_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:1::/56 via fe80::5884:70ff:fe79:db35 dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:2:2::/64 dev gpu1_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:2::/56 via fe80::5884:70ff:fe79:db36 dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:3:2::/64 dev gpu2_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:3::/56 via fe80::5884:70ff:fe79:db3d dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:4:2::/64 dev gpu3_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:4::/56 via fe80::5884:70ff:fe79:db3e dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:5:2::/64 dev gpu4_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:5::/56 via fe80::5884:70ff:fe79:db45 dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:6:2::/64 dev gpu5_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:6::/56 via fe80::5884:70ff:fe79:db46 dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fc00:200:7:2::/64 dev gpu6_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:7::/56 via fe80::5884:70ff:fe79:db4d dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
```

```
fc00:200:8:2::/64 dev gpu7_eth proto ra metric 1024 expires 2591885sec pref medium
fc00:200:8::/56 via fe80::5884:70ff:fe79:db4e dev gpu0_eth proto ra metric 100 expires 59841sec
pref medium
fe80::/64 dev mgmt_eth proto kernel metric 256 pref medium
fe80::/64 dev eno3 proto kernel metric 256 pref medium
fe80::/64 dev stor0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu1_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu2_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu3_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu4_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu5_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu6_eth proto kernel metric 256 pref medium
fe80::/64 dev gpu7_eth proto kernel metric 256 pref medium
default proto ra metric 1024 expires 1685sec pref medium
       nexthop via fe80::5884:70ff:fe79:db35 dev gpu0_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db36 dev gpu1_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db3d dev gpu2_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db3e dev gpu3_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db45 dev gpu4_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db46 dev gpu5_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db4d dev gpu6_eth weight 1
       nexthop via fe80::5884:70ff:fe79:db4e dev gpu7_eth weight 1
```

When sending traffic from **fc00:200:1:1:a288:c2ff:fe3b:5066** to **fc00:200:1:2:5aa2:e1ff:fe46:c6ca**, H100-01 selects the **fc00:200:1::/56** route via **fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth** instead of the default route. Similarly, when sending traffic from **fc00:200:2:1:a288:c2ff:fe3b:5066** to **fc00:200:2:2:5aa2:e1ff:fe46:c6ca**, H100-01 selects the **fc00:200:2::/56** route via **fe80::9e5a:80ff:fec1:ae81 dev gpu1_eth**. In both cases, the correct next-hop and interface is selected, and the traffic is forwarded successfully.

```
jnpr@H100-01:~$ ping fc00:200:1:2:5aa2:e1ff:fe46:c6ca -I fc00:200:1:1:a288:c2ff:fe3b:5066 -c 5
PING fc00:200:1:2:5aa2:e1ff:fe46:c6ca(fc00:200:1:2:5aa2:e1ff:fe46:c6ca) from
fc00:200:1:1:a288:c2ff:fe3b:5066 : 56 data bytes
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=1 ttl=63 time=0.598 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=2 ttl=63 time=0.555 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=3 ttl=63 time=0.552 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=4 ttl=63 time=0.594 ms
64 bytes from fc00:200:1:2:5aa2:e1ff:fe46:c6ca: icmp_seq=5 ttl=63 time=0.625 ms
--- fc00:200:1:2:5aa2:e1ff:fe46:c6ca ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4102ms
rtt min/avg/max/mdev = 0.552/0.584/0.625/0.027 ms
```

```
jnpr@H100-01:~$ ping fc00:200:2:2:5aa2:e1ff:fe46:c6ce -I fc00:200:2:1:a288:c2ff:fe3b:506a -c 5
PING fc00:200:2:2:5aa2:e1ff:fe46:c6ce(fc00:200:2:2:5aa2:e1ff:fe46:c6ce) from
fc00:200:2:1:a288:c2ff:fe3b:506a : 56 data bytes
64 bytes from fc00:200:2:2:5aa2:e1ff:fe46:c6ce: icmp_seq=1 ttl=63 time=0.330 ms
64 bytes from fc00:200:2:2:5aa2:e1ff:fe46:c6ce: icmp_seq=2 ttl=63 time=0.285 ms
64 bytes from fc00:200:2:2:5aa2:e1ff:fe46:c6ce: icmp_seq=3 ttl=63 time=0.290 ms
64 bytes from fc00:200:2:2:5aa2:e1ff:fe46:c6ce: icmp_seq=4 ttl=63 time=0.283 ms
64 bytes from fc00:200:2:2:5aa2:e1ff:fe46:c6ce: icmp_seq=5 ttl=63 time=0.286 ms
--- fc00:200:2:2:5aa2:e1ff:fe46:c6ce ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.283/0.294/0.330/0.017 ms
```

Server SLAAC Configuration:

The interfaces on the servers do not need to be configured with any IPv6 address. Disabling DHCPv6 is enough.

Example:

```
    gpu0_eth:
      match:
        macaddress: a0:88:c2:3b:50:66
      dhcp6: false
      mtu: 9000
      set-name: gpu0_eth
```

The servers must also be configured to **accept and process RA messages**, for IPv6 address autoconfiguration via Router Advertisements (RA) to work. In most cases, this will be enabled by default but the steps to enabled it are described here:

The configuration has two layers:

1. Interface-level RA policy in Netplan or systemd
2. Kernel-level sysctl parameters (accept_ra, autoconf)

Both must align to ensure proper RA behavior.

- If the system uses Netplan with systemd-networkd (common on Ubuntu Server):
  In the Netplan YAML file (e.g., /etc/netplan/01-netcfg.yaml), add the following under each interface:

  ```
  accept-ra: true
  ```

  ```
  IPv6-privacy: false
  ```

  Then apply the changes:

```
sudo netplan generate
```

```
sudo netplan apply
```

This ensures that Netplan renders a .network file for systemd-networkd with **IPv6AcceptRA=yes**, which enables RA-based autoconfiguration.

However, this alone is not enough. If the kernel is still configured to ignore RAs. You must also verify that the kernel is set to accept RAs at runtime. You can check using:

```
sudo sysctl net.IPv6.conf.<interface>.accept_ra
```

If the value is 0, RAs will be ignored regardless of Netplan settings. This can be temporarily corrected with:

```
sudo sysctl -w net.IPv6.conf.<interface>.accept_ra=1
```

To make it persistent across reboots, add the following to a sysctl configuration file (e.g., /etc/sysctl.d/99-accept-ra.conf):

```
net.IPv6.conf.<interface>.accept_ra = 1
```

And apply it with:

```
sudo sysctl --system
```

NOTE: Notice that parameters such as accept-ra can be enable or disable globally or on a per interface basis.

Table 17. Scope and Behavior of accept_ra Sysctl Parameters in IPv6 Configuration

Sysctl	Scope	Effect
net.IPv6.conf.all.accept_ra	Global (all current interfaces)	Applies immediately to **all existing** interfaces, but... **read-only if forwarding=1**
net.IPv6.conf.default.accept_ra	Global (for future interfaces)	Sets the **default value** used when a **new interface** comes up (e.g., plugged in or created later)
net.IPv6.conf.gpu0_eth.accept_ra	Per-interface	Controls RA processing for a specific active interface

If the interface is managed directly by the kernel (not using Netplan/systemd):

Enable RA acceptance and autoconfiguration by setting:

```
sudo sysctl -w net.IPv6.conf.<interface>.accept_ra=1
sudo sysctl -w net.IPv6.conf.<interface>.autoconf=1
sudo tee /etc/sysctl.d/99-IPv6-ra.conf > /dev/null <<EOF
net.IPv6.conf.<interface>.accept_ra = 1
net.IPv6.conf.<interface>.autoconf = 1
EOF
sudo sysctl --system
```

Follow the steps in AMD Configuration | Juniper Networks to configure the interfaces on AMD GPU servers or NVIDIA Configuration | Juniper Networks for NVIDIA GPU servers.

Leaf Node SLAAC Configuration

To enable SLAAC, the leaf nodes must be explicitly configured with IPv6 addresses on the interfaces facing the GPU servers.

Example:

```
jnpr@stripe1-leaf1# show interface et-0/0/0:0
description " Multitenancy Tenant-1 GPU0 Server 1";
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9140;
        address FC00:200:1:1::1/64;
    }
}
jnpr@stripe1-leaf1# show interface et-0/0/1:0
description "Multitenancy Tenant-1 GPU0 Server 2";
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9140;
        address FC00:200:1:2::1/64;
    }
}
jnpr@stripe1-leaf1# show interface et-0/0/2:0
description "Multitenancy Tenant-1 GPU0 Server 3";
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9140;
        address FC00:200:1:3::1/64;
    }
}
.
.
.
jnpr@stripe2-leaf1# show interface et-0/0/0:0
description "Multitenancy Tenant-1 GPU0 Server 9";
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9140;
        address FC00:200:1:9::1/64;
    }
}
jnpr@stripe2-leaf1# show interface et-0/0/0:0
description "Multitenancy Tenant-1 GPU0 Server 10";
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9140;
        address FC00:200:1:10::1/64;
    }
}
.
.
.
```

After assigning the IPv6 addresses, prefix advertisement must be enabled under the protocols router-advertisement hierarchy, as shown in the example below:

```
[edit protocols router-advertisement]
jnpr@stripe1-leaf1# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:1::1/64;
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:2::1/64;
}
```

```
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:3::1/64;
}
[edit protocols router-advertisement]
              jnpr@stripe1-leaf2# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:1::1/64;
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:2::1/64;
}
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:3::1/64;
}
```

The **retransmit-timer 10000** configures the retransmission frequency of neighbor advertisements in milliseconds.

Configuring router advertisements for a given prefix requires that the interface itself has an IPv6 address within that same prefix. If the prefix specified under router-advertisement is not also configured on the interface, the commit will fail with an error.

Example:

```
[edit interfaces et-0/0/0:0]
jnpr@stripe1-leaf1# show
unit 0 {
    family inet6 {
        address FC00:255:1:1::1/64;
    }
}
[edit protocols router-advertisement]
jnpr@stripe1-leaf1# show
interface et-0/0/0:0.0 {
    prefix FC00:200:1:1::1/64;
}
[edit protocols router-advertisement interface et-0/0/12:0.0]
jnpr@stripe1-leaf1# commit
[edit protocols router-advertisement interface]
```

```
    'et-0/0/0.0'
      Family inet6 should be configured on this interface
error: commit failed: (statements constraint check failed)
```

Also, configure the rio-prefix under protocol router-advertisement, as shown in the example:

```
[edit protocols router-advertisement]
jnpr@stripe1-leaf1# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:1::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:2::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:3::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
.
.
.
[edit protocols router-advertisement]
jnpr@stripe1-leaf2# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:1::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:2::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:3::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
.
.
.
[edit protocols router-advertisement]
jnpr@stripe2-leaf1# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:9::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:10::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:1:11::/64;
    rio-prefix fc00:200:1::/56 {
        rio-lifetime 1800;
    }
}
.
.
.
[edit protocols router-advertisement]
```

```
jnpr@stripe2-leaf2# show
interface et-0/0/0:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:9::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/1:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:10::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
interface et-0/0/2:0.0 {
    retransmit-timer 10000;
    prefix FC00:200:2:11::/64;
    rio-prefix fc00:200:2::/56 {
        rio-lifetime 1800;
    }
}
.
.
.
```

Notice that the lifetime is mandatory for the rio-prefix. In the example, this value is set **1800 seconds** (30 minutes). The rio-prefix must be the /56 prefix assigned to the tenant, as described in the previous section.

SLAAC Verification:

To verify that RA-based configuration is working and that the GPU interface has autoconfigured its IPv6 address, use: `ip -6 addr show dev <interface> or ifconfig <interface>`

The command should display the interface's link local address (FE80::<EUI-64>) and the global inet6 address generated by SLAAC (prefix::EUI-64). This global address will be marked as dynamic to indicate it was dynamically configured.

Example:

```
jnpr@H100-01:~$ ip -6 addr show dev gpu0_eth
17: gpu0_eth: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group default qlen
1000
```

```
  inet6 fc00:200:1:1:a288:c2ff:fe3b:5066/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 2591741sec preferred_lft 604541sec
  inet6 fe80::a288:c2ff:fe3b:5066/64 scope link
       valid_lft forever preferred_lft forever
jnpr@H100-01:~$ ifconfig gpu0_eth
gpu0_eth: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9000
  inet6 fe80::a288:c2ff:fe3b:5066  prefixlen 64  scopeid 0x20<link>
  inet6 fc00:200:1:1:a288:c2ff:fe3b:5066  prefixlen 64  scopeid 0x0<global>
  ether a0:88:c2:3b:50:66  txqueuelen 1000  (Ethernet)
  RX packets 67096  bytes 5792577 (5.7 MB)
  RX errors 0  dropped 0  overruns 0  frame 0
  TX packets 20886  bytes 3122514 (3.1 MB)
  TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

You can also observe incoming RA messages using tcpdump: `sudo tcpdump -i <interface> -vv icmp6 and 'ip6[40] == 134'`

Example:

```
jnpr@H100-01:~$ sudo tcpdump -i gpu0_eth -vv icmp6 and 'ip6[40] == 134'
tcpdump: listening on gpu0_eth, link-type EN10MB (Ethernet), snapshot length 262144 bytes
19:26:15.604130 IP6 (flowlabel 0xcbfef, hlim 255, next-header ICMPv6 (58) payload length: 72)
fe80::9e5a:80ff:fec1:ae60 > ip6-allnodes: [icmp6 sum ok] ICMP6, router advertisement, length 72
        hop limit 64, Flags [none], pref medium, router lifetime 1800s, reachable time 0ms,
retrans timer 0ms
          source link-address option (1), length 8 (1): 9c:5a:80:c1:ae:60
            0x0000:  9c5a 80c1 ae60
          prefix info option (3), length 32 (4): fc00:200:1:1::/64, Flags [onlink, auto], valid
time 2592000s, pref. time 604800s
            0x0000:  40c0 0027 8d00 0009 3a80 0000 0000 fc00
            0x0010:  0200 0001 0001 0000 0000 0000 0000
          route info option (24), length 16 (2):  fc00:200:1::/56, pref=medium, lifetime=60000s
            0x0000:  3800 0000 ea60 fc00 0200 0001 0000
19:26:31.605713 IP6 (flowlabel 0xcbfef, hlim 255, next-header ICMPv6 (58) payload length: 72)
fe80::9e5a:80ff:fec1:ae60 > ip6-allnodes: [icmp6 sum ok] ICMP6, router advertisement, length 72
        hop limit 64, Flags [none], pref medium, router lifetime 1800s, reachable time 0ms,
retrans timer 0ms
          source link-address option (1), length 8 (1): 9c:5a:80:c1:ae:60
            0x0000:  9c5a 80c1 ae60
          prefix info option (3), length 32 (4): fc00:200:1:1::/64, Flags [onlink, auto], valid
time 2592000s, pref. time 604800s
            0x0000:  40c0 0027 8d00 0009 3a80 0000 0000 fc00
```

```
             0x0010:  0200 0001 0001 0000 0000 0000 0000
           route info option (24), length 16 (2):  fc00:200:1::/56, pref=medium, lifetime=60000s
             0x0000:  3800 0000 ea60 fc00 0200 0001 0000
```

NOTE: If a new prefix needs to be advertised on an interface, reconfigure the router advertisements to age out the old address and to advertise the new one.

```
jnpr@H100-01:/etc/netplan$ ip -6 addr show dev gpu0_eth
17: gpu0_eth: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group default qlen
1000
 inet6 fc00:200:1:1:a288:c2ff:fe3b:5066/64 scope global dynamic mngtmpaddr noprefixroute
   valid_lft 2591988sec preferred_lft 604788sec
 inet6 fe80::a288:c2ff:fe3b:5066/64 scope link
   valid_lft forever preferred_lft forever
jnpr@H100-01:/etc/netplan$ ip -6 route | grep gpu0_eth
fc00:200:1:1::/64 dev gpu0_eth proto ra metric 100 expires 2591949sec pref medium
fc00:200:1::/56 via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth proto ra metric 100 expires 1749sec
pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
default via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth proto ra metric 100 expires 1749sec pref
medium
[edit protocols router-advertisement]
jnpr@stripe1-leaf1#
  interface et-0/0/0:0 {
    /* DEPRECATED IPv6 PREFIX */
    prefix fc00:200:1:1::/64 {
      valid-lifetime 0;
      preferred-lifetime 0;
    }
    rio-prefix fc00:200:1::/56 {
      rio-lifetime 0;
    }
    /* NEW IPv6 PREFIX */
    prefix fc00:200:100:100::/64;
    rio-prefix fc00:200:100::/56 {
      rio-lifetime 1800;
    }
  }
jnpr@H100-01:/etc/netplan$ ip -6 addr show dev gpu0_eth
17: gpu0_eth: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group default qlen
```

```
1000
    inet6 fc00:200:100:100:a288:c2ff:fe3b:5066/64 scope global tentative dynamic mngtmpaddr
noprefixroute
       valid_lft 2591999sec preferred_lft 604799sec
    inet6 fe80::a288:c2ff:fe3b:5066/64 scope link
       valid_lft forever preferred_lft forever
jnpr@H100-01:/etc/netplan$ ip -6 route | grep gpu0_eth
fc00:200:100::/56 via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth proto ra metric 100 expires 1797sec
pref medium
fc00:200:100:100::/64 dev gpu0_eth proto ra metric 100 expires 2591997sec pref medium
fe80::/64 dev gpu0_eth proto kernel metric 256 pref medium
default via fe80::9e5a:80ff:fec1:ae60 dev gpu0_eth proto ra metric 100 expires 1797sec pref
medium
```

If you need to manually flush any IPv6 address from the server interface you can use the following commands:

`sudo ip addr flush dev <interface>sudo ip link set <interface> down && sleep 1&& sudo ip link set <interface> up`

After bringing the interface back up, wait a few seconds and re-check the IPv6 address with:

`ip -6 addr show dev <interface>`

This ensures that stale addresses are removed, and fresh RAs are processed.

NOTE: All IPv6 settings can be found under: **/proc/sys/net/IPv6/conf**

To verify that router advertisements are being sent, you can use the following command:`show ipv6 router-advertisement interface <interface>`

Example:

```
jnpr@stripe1-leaf1> show IPv6 router-advertisement interface et-0/0/0:0
Interface: et-0/0/0:0.0
  Advertisements sent: 3, last sent 00:01:48 ago
  Solicits sent: 1, last sent 00:02:20 ago
  Solicits received: 0
  Advertisements received: 0
  Solicited router advertisement unicast: Disable
  IPv6 RA Preference: DEFAULT/MEDIUM
  Passive mode: Disable
  Upstream mode: Disable
  Downstream mode: Disable
  Proxy blackout timer: Not Running
  Route Information: fc00:200:1::/56
    IPv6 RA Preference: DEFAULT/MEDIUM
    Route lifetime: 60000 sec
```

You can also capture router advertisement packets on the interface using: `monitor traffic interface et-0/0/0:0.0 extensive matching "icmp6 and ip6[40] == 134"`

```
Example:
jnpr@stripe1-leaf1> monitor traffic interface et-0/0/0:0.0 extensive matching "icmp6 and ip6[40]
== 134"
18:05:50.344868 9c:5a:80:c1:ae:60 > 33:33:00:00:00:01, ethertype IPv6 (0x86dd), length 188:
(flowlabel 0x19976, hlim 255, next-header ICMPv6 (58) payload length: 56)
fe80::9e5a:80ff:fec1:ae60 > ff02::1: [icmp6 sum ok] ICMP6, router advertisement, length 56
        hop limit 64, Flags [none], pref medium, router lifetime 1800s, reachable time 0ms,
retrans timer 0ms
          source link-address option (1), length 8 (1): 9c:5a:80:c1:ae:60
            0x0000:  9c5a 80c1 ae60
          prefix info option (3), length 32 (4): fc00:200:1:1::/64, Flags [onlink, auto], valid
time 2592000s, pref. time 604800s
            0x0000:  40c0 0027 8d00 0009 3a80 0000 0000 fc00
            0x0010:  0200 0001 0001 0000 0000 0000 0000
          route info option (24), length 16 (2):  fc00:200:1::/56, pref=medium, lifetime=60000s
```

Notice that Router Advertisements are sent using the link local address of the leaf node interfaces as source, the IPv6 all-nodes multicast address (FF02::1), next-header ICMPv6 (58). The following are the most relevant attributes for these:

Table 18. Fields and Semantics in IPv6 Router Advertisement

PARAMETER	VALUE	DESCRIPTION
Flags	auto	Hosts can assume addresses in this prefix are on the local link. This prefix can be used for **SLAAC** (Stateless Address Auto Configuration).
Flags	On-link	tells hosts which destinations are directly reachable without going through a router.

(Continued)

PARAMETER	VALUE	DESCRIPTION
source link-address option	9c:5a:80:c1:ae:60	Tells the receiver the link-layer (MAC) address of the router sending the RA. The receiver knows the router's MAC address without having to send a separate Neighbor Solicitation.
prefix info option	fc00:200:1:1::/64	Advertises IPv6 prefixes that hosts can use to autoconfigured its IPv6 address.
route info option	fc00:200:1::/56	Carries routes to destinations other than the default. Routers can advertise more specific routes (beyond just "I'm the default gateway").
Valid Lifetime	2592000	Prefix is valid for 30 days (used for reachability).
Preferred Lifetime	604800	Preferred lifetime of 7 days (after which it becomes deprecated for new connections).
router lifetime	1800s	The router is considered a default gateway for 1800 seconds

After receiving the router-advertisement, the server's NIC interfaces will have autoconfigured their IPv6 addresses by concatenating the prefix advertise by the Leaf node, with the host portion of the address calculated using the EUI-64 address format (based on the interface's MAC address), as shown in Table 19.

Table 19. GPU to Leaf nodes IPv6 addresses

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	GPU NIC	GPU NIC MAC address	GPU NIC IPv6 ADDRESS
Stripe 1 Leaf 1 et-0/0/0:0	FC00:200:1:1::1/64	Server 1 - gpu0_eth	a0:88:c2:3b:50:66	FC00:200:1:1:a288:c2ff:fe3b:5066
Stripe 1 Leaf 1 et-0/0/1:0	FC00:200:1:2::1/64	Server 2 - gpu0_eth	58:a2:e1:46:c6:ca	FC00:200:1:2:a288:c2ff:fe3b:506a

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	GPU NIC	GPU NIC MAC address	GPU NIC IPv6 ADDRESS
Stripe 2 Leaf 1 et-0/0/2:0	FC00:200:1:3::1/64	Server 3 - gpu0_eth	a0:88:c2:3b:50:6e	FC00:200:1:3:a2:88:c2ff:fe3b:50:6e
. . .				

IPv6 Leaf Nodes to Spine Nodes Connections Using Link Local Addresses

When deploying the underlay using IPv6 Link-Local underlay, the interfaces between the leaf and spine nodes do not require explicitly configured IP addresses and are configured as untagged interfaces with only family inet6 to enable processing of IPv6 traffic as shown in Figure 50.

Figure 50: Leaf Nodes to Spine Nodes Connectivity

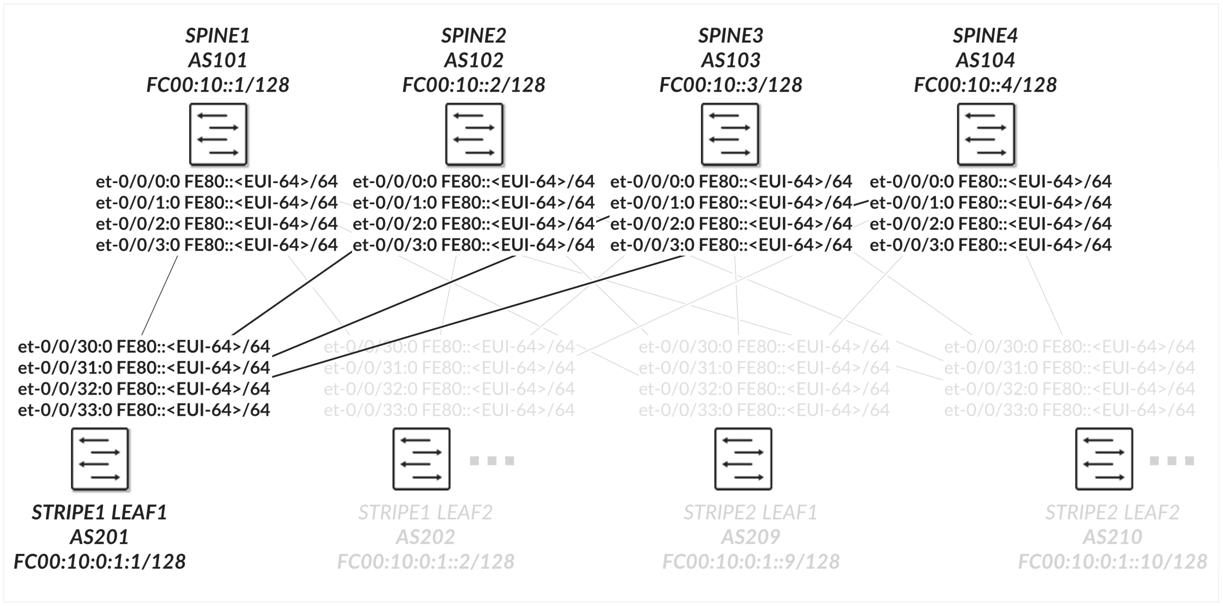


Table 20. Spine to Leaf Interface Configuration Example

STRIPE 1 LEAF 1 (et-0/0/0:30)	SPINE 1 (et-0/0/0:0)

```[edit interfaces et-0/0/30]```
```jnpr@stripe1-leaf1# show```
```description "Breakout et-0/0/30";```
```number-of-sub-ports 1;```
```speed 800g;```
 	
```[edit interfaces et-0/0/30:0]```
```jnpr@stripe1-leaf1# show```
```description facing_spine1:et-0/0/0:0;```
```mtu 9216;```
```unit 0 {```
```   family inet6 {```
```   mtu 9202;```
```   }```
```}```
	```[edit interfaces et-0/0/0]```
	```jnpr@spine1# show```
	```description "Breakout et-0/0/0";```
	```number-of-sub-ports 1;```
	```speed 800g;```
	
	```[edit interfaces et-0/0/0:0]```
	```jnpr@spine1# show```
	```description facing_Leaf1:et-0/0/0:0;```
	```mtu 9216;```
	```unit 0 {```
	``` family inet6 {```
	``` mtu 9202;```
	``` }```
	```}```

Enabling IPv6 on an interface automatically assigns a link-local IPv6 address. The switch autogenerates link local addresses for the interfaces using the EUI-64 address format (based on the interface's MAC address), as shown in Table 21.

Table 21. Spine and Leaf IPv6-Enabled Interface Link Local Addresses

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
Stripe 1 Leaf 1 - et-0/0/30:0	fe80::9e5a:80ff:fec1:ae00/64	Spine 1 – et-0/0/0:0	fe80::9e5a:80ff:feef:a28f/64
Stripe 1 Leaf 1 - et-0/0/31:0	fe80::9e5a:80ff:fec1:ae08/64	Spine 2 – et-0/0/0:0	fe80::5a86:70ff:fe7b:ced5/64
Stripe 1 Leaf 1 - et-0/0/32:0	fe80::9e5a:80ff:fec1:af00/64	Spine 3 – et-0/0/0:0	fe80::5a86:70ff:fe78:e0d5/64
Stripe 1 Leaf 1 - et-0/0/33:0	fe80::9e5a:80ff:fec1:af08/64	Spine 4 – et-0/0/0:0	fe80::5a86:70ff:fe79:3d5/64
Stripe 1 Leaf 2 - et-0/0/30:0	fe80::5a86:70ff:fe79:dad5/64	Spine 1 – et-0/0/1:0	fe80::9e5a:80ff:feef:a297/64
Stripe 1 Leaf 2 - et-0/0/31:0	fe80::5a86:70ff:fe79:dadd/64	Spine 2 – et-0/0/1:0	fe80::5a86:70ff:fe7b:cedd/64
Stripe 1 Leaf 2 - et-0/0/32:0	fe80::5a86:70ff:fe79:dbd5/64	Spine 3 – et-0/0/1:0	fe80::5a86:70ff:fe78:e0dd/64
Stripe 1 Leaf 2 - et-0/0/33:0	fe80::5a86:70ff:fe79:dbdd/64	Spine 4 – et-0/0/1:0	fe80::5a86:70ff:fe79:3dd/64

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
. . .			

These addresses need to be advertised through standard router advertisements as part of the IPv6 Neighbor Discovery process to allow the leaf and spine nodes to then establish BGP sessions between them. Router advertisement must be enabled on all the interfaces between the leaf and spine nodes as shown:

Table 22. IPv6 Router Advertisement on Leaf and Spine Interfaces

STRIPE 1 LEAF 1	SPINE 1

[edit protocols router-advertisement]
/* ROUTER ADVERTISEMENTS TO SPINE1 */
interface et-0/0/30:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE2 */
interface et-0/0/31:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE3 */
interface et-0/0/32:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE4 */
interface et-0/0/33:0.0;
}
	[edit]
	jnpr@spine1# show protocols router-advertisement
	/* ROUTER ADVERTISEMENTS TO LEAF1 */
	interface et-0/0/0:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF2 */
	interface et-0/0/1:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF3 */
	interface et-0/0/2:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF4 */
	interface et-0/0/3:0.0;
	.
	.
	.

To verify that router advertisements are being sent you can use:`show IPv6 router-advertisement interface <interface>` and `show IPv6 neighbors`

<u>Example</u>:

```
              jnpr@stripe1-leaf1> show IPv6 router-advertisement interface et-0/0/30:0
Interface: et-0/0/30:0.0
  Advertisements sent: 4, last sent 00:02:28 ago
  Solicits sent: 1, last sent 00:08:06 ago
  Solicits received: 0
  Advertisements received: 3
  Solicited router advertisement unicast: Disable
  IPv6 RA Preference: DEFAULT/MEDIUM
  Passive mode: Disable
  Upstream mode: Disable
  Downstream mode: Disable
```

```
    Proxy blackout timer: Not Running
    Advertisement from fe80::9e5a:80ff:feef:a28f, heard 00:01:57 ago
      Managed: 0
      Other configuration: 0
      Reachable time: 0 ms
      Default lifetime: 1800 sec
      Retransmit timer: 0 ms
      Current hop limit: 64
  jnpr@stripe1-leaf1> show IPv6 neighbors
  IPv6 Address     Linklayer Address  State       Exp   Rtr   Secure  Interface
  fe80::5a86:70ff:fe78:e0d5       58:86:70:78:e0:d5  reachable   11    yes   no
  et-0/0/31:0.0
  fe80::5a86:70ff:fe79:3d5        58:86:70:79:03:d5  reachable   23    yes   no
  et-0/0/33:0.0
  fe80::5a86:70ff:fe7b:ced5       58:86:70:7b:ce:d5  reachable   13    yes   no
  et-0/0/32:0.0
  fe80::9e5a:80ff:feef:a28f       9c:5a:80:ef:a2:8f  reachable   25    yes   no
  et-0/0/30:0.0
  Total entries: 4
```

The loopback interface IPv6 addresses and the Autonomous System numbers for all devices in the fabric are included in table 23:

Table 23. Spine and Leaf Loopback Addresses and ASNs

LEAF NODE INTERFACE	lo0.0 IPv6 ADDRESS	Local AS #
Stripe 1 Leaf 1	FC00:10:0:1::1/128	201
Stripe 1 Leaf 2	FC00:10:0:1::2/128	202
Stripe 1 Leaf 3	FC00:10:0:1::3/128	203
Stripe 1 Leaf 4	FC00:10:0:1::4/128	204
Stripe 1 Leaf 5	FC00:10:0:1::5/128	205
Stripe 1 Leaf 6	FC00:10:0:1::6/128	206
Stripe 1 Leaf 7	FC00:10:0:1::7/128	207

(Continued)

LEAF NODE INTERFACE	lo0.0 IPv6 ADDRESS	Local AS #
Stripe 1 Leaf 8	FC00:10:0:1::8/128	208
Stripe 2 Leaf 1	FC00:10:0:1::9/128	209
Stripe 2 Leaf 2	FC00:10:0:1::10/128	210
. . .		
SPINE1	FC00:10:0::1/128	101
SPINE2	FC00:10:0::2/128	102
SPINE3	FC00:10:0::3/128	103
SPINE4	FC00:10:0::4/128	104

Table 24. Spine and Leaf Loopback Address Configuration

STRIPE 1 LEAF 1	SPINE 1

`[edit interfaces lo0]`
`jnpr@stripe1-leaf1# show`
`unit 0 {`
`  family inet {`
`    address FC00:10:0:1::1/128;`
`  }`
`}`
	`[edit interfaces lo0]`
	`jnpr@spine1# show`
	`unit 0 {`
	`  family inet {`
	`    address FC00:10::1/128;`
	`  }`
	`}`

Recommended MTU

Configure the MTU consistently across the fabric and make sure that the MTU of the server->leaf links does not exceed the MTU of the leaf->spine links considering the extra overhead of the VXLAN encapsulation.

VXLAN Overhead Calculation

For IPv6, the MTU can also be calculated as:

Table 26 VXLAN Overhead Calculation

HEADER	BYTES
Outer Ethernet	14
Outer IP (IPv6)	40
UDP	8
VXLAN	8
Total	70 bytes

Recommended MTU Strategy

Table 27. Recommended MTU

LINK TYPE	MTU
Server ↔ Leaf	9000
Leaf ↔ Spine IPv6	> 9070

It is important to keep in mind that RoCEv2 message sizes are still limited by the RDMA MTU reported by ibv_devinfo

```
jnpr@MI300-01:~/SCRIPTS$ ibv_devinfo -d bnxt_re0
hca_id: bnxt_re0
        transport:                      InfiniBand (0)
        fw_ver:                         230.2.49.0
        node_guid:              7ec2:55ff:febd:75d0
        sys_image_guid:                 7ec2:55ff:febd:75d0
        vendor_id:                      0x14e4
        vendor_part_id:                 5984
        hw_ver:                         0x1D42
        phys_port_cnt:                  1
                port:   1
                        state:                  PORT_ACTIVE (4)
                         max_mtu:                       4096 (5)
                         active_mtu:                    4096 (5)
                        sm_lid:                         0
                        port_lid:                               0
```

```
                          port_lmc:               0x00
                          link_layer:                     Ethernet
```

Table 28. MTU Types: Ownership and Functional Role

MTU TYPE	OWNER	PURPOSE
Interface MTU (e.g. 9000) **ifconfig, ip**	Linux network stack	Defines the max L3/IP packet size
RDMA MTU (e.g. 4096) **ibv_devinfo**	RDMA stack	Defines the max RDMA message size per Work Queue Element (WQE)

The RDMA MTU can be configured at the verbs level, and it's negotiated during QP (Queue Pair) setup. You *cannot* override it by just setting the NIC's MTU to a higher value, but you would need to use low-level tools or RDMA apps.

Some performance tools such as ib_send_bw, ib_write_bw (via -m flag). For example:

ib_write_bw -m 1024 # sets RDMA MTU to 1024 bytes

ib_write_bw -m 4096 # sets RDMA MTU to 4096 (max allowed according to the output of ibv_devinfo shown before)

RDMA MTU must be ≤ Interface MTU – encapsulation overhead

IPv6 GPU Backend Fabric Underlay, using BGP Neighbor Discovery

Refer to Configure BGP Unnumbered EVPN Fabric | Juniper Networks for more information.

The **underlay EBGP sessions** are configured between the leaf and spine nodes to use **peer auto-discovery**, and are configured to advertise these loopback interfaces, as shown in the example between Stripe1 Leaf 1 and Spine 1 below:

Table 29. GPU Backend Fabric: BGP Underlay with Peer Auto-Discovery Configuration

STRIPE 1 LEAF 1

```
[edit routing-options]
jnpr@stripe1-leaf1# show
router-id 10.0.1.1;
autonomous-system 201;
graceful-restart;
forwarding-table {
  export PFE-LB;
  ecmp-fast-reroute;
}

[edit policy-options]
jnpr@stripe1-leaf1# show | match as-list
as-list discovered-as-list members 101-104;

[edit protocols bgp group l3clos-inet6-auto-underlay]
jnpr@stripe1-leaf1# show
  type external;
  family inet6 {
    unicast;
  }
  export (LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy);
  local-as 201;
  multipath {
    multiple-as;
  }
  bfd-liveness-detection {
    minimum-interval 3000;
    multiplier 3;
  }
  dynamic-neighbor underlay-dynamic-neighbors {
    peer-auto-discovery {
      family inet6 {
         ipv6-nd;
      }
      /* SPINE 1 */
      interface et-0/0/0:0.0;
      /* SPINE 2 */
      interface et-0/0/1:0.0;
      /* SPINE 3 */
      interface et-0/0/32:0.0;
      /* SPINE 3 */
      interface et-0/0/33:0.0;
    }
  }
  peer-as-list discovered-as-list;
```

SPINE 1

```
[edit routing-options]
jnpr@spine1# show
router-id 10.0.0.1;
autonomous-system 101;
graceful-restart;
forwarding-table {
  export PFE-LB;
  ecmp-fast-reroute;
}

[edit policy-options]
jnpr@stripe1-leaf1# show | match as-list
as-list discovered-as-list members 201-216;

[edit protocols bgp group l3clos-inet6-auto-underlay]
jnpr@stripe1-leaf1# show
  type external;
  family inet6 {
    unicast;
  }
  export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );
  local-as 101;
  multipath {
    multiple-as;
  }
  bfd-liveness-detection {
    minimum-interval 3000;
    multiplier 3;
  }
  dynamic-neighbor underlay-dynamic-neighbors {
    peer-auto-discovery {
      family inet6 {
         ipv6-nd;
      }
      /* LEAF1 1 */
      interface et-0/0/0:0.0;
      /* LEAF1 2 */
      interface et-0/0/1:0.0;
      /* LEAF1 3 */
      interface et-0/0/2:0.0;
      .
      .
      .
    }
  }
  peer-as-list discovered-as-list;
```

To configure peer auto discovery, the dynamic-neighbor named **underlay-dynamic-neighbors,** under **BGP group l3clos-inet6-auto-underlay,** specifies the interfaces where auto discovery is permitted. This replaces the neighbor a.b.c.d commands that would statically configure the neighbors.

The **family inet6 IPv6-nd** statement enables the use of **IPv6 Neighbor Discovery** to dynamically determine the addresses of neighbors with which to establish BGP sessions. To control and secure dynamic peer formation, a **peer-as-list** (discovered-as-list) is configured, restricting peering to neighbors whose autonomous system numbers fall within the defined range of **AS 101–104**.

The **family inet6 unicast** statements configure the sessions to advertise **IPv6** prefixes to support the IPv6 overlays.

The BGP sessions are also configured with **multipath multiple-as**, allowing multiple paths (even with different AS paths) to be considered for **ECMP (Equal-Cost Multi-Path)** routing. **BFD (Bidirectional Forwarding Detection)** is additionally enabled to accelerate convergence in case of link or neighbor failures.

You can check that the sessions have been established using:

```
show bgp summary group <group-name>
```

Example:

```
jnpr@stripe1-leaf1> show bgp summary group l3clos-inet6-auto-underlay
fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0         102        201        196       0       0
1:29:35 Establ
  inet6.0: 4/4/4/0
fe80::5a86:70ff:fe79:3d5%et-0/0/33:0.0          104        201        196       0       0
1:29:15 Establ
  inet6.0: 4/4/4/0
fe80::5a86:70ff:fe7b:ced5%et-0/0/32:0.0         103        201        196       0       0
1:29:21 Establ
  inet6.0: 4/4/4/0
fe80::9e5a:80ff:feef:a28f%et-0/0/30:0.0         101        202        197       0       0
1:29:30 Establ
  inet6.0: 4/4/4/0
```

Notice that when BGP sessions are established using link-local addresses Junos displays the neighbor address along with the interface scope (e.g. **fe80::5a86:70ff:fe78:e0d5%et-0/0/1:0.0**). The scope identifier (the part after the %) is necessary because the same link-local address (fe80::/10) could exist on multiple interfaces. The device must know which interface to use to send packets to that neighbor. Thus, after peer discovery is completed, the **`show bgp summary`** output lists the neighbor using the format: **`IPv6_link-local_address%interface-name`**.

You can check details about discovered neighbors using:

```
show bgp neighbor auto-discovered <peer-id>
```

Example:

```
jnpr@stripe1-leaf1> show bgp neighbor auto-discovered fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0
Peer: fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0+179 AS 102 Local:
fe80::9e5a:80ff:fec1:ae08%et-0/0/31:0.0+53984 AS 201
  Group: l3clos-inet6-auto-underlay Routing-Instance: master
  Forwarding routing-instance: master
  Type: External    State: Established    Flags: <Sync PeerAsList AutoDiscoveredNdp>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ (LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy) ]
  Options: <GracefulRestart AddressFamily Multipath LocalAS Refresh>
  Options: <MultipathAs BfdEnabled>
  Options: <GracefulShutdownRcv>
  Address families configured: inet6-unicast
```

```
Holdtime: 90 Preference: 170
Graceful Shutdown Receiver local-preference: 0
Local AS: 201 Local System AS: 201
Number of flaps: 0
Receive eBGP Origin Validation community: Reject
Peer ID: 10.0.0.2       Local ID: 10.0.1.1          Active Holdtime: 90
Keepalive Interval: 30         Group index: 0    Peer index: 0    SNMP index: 30
I/O Session Thread: bgpio-0 State: Enabled
BFD: enabled, up
Local Interface: et-0/0/1:0.0
NLRI for restart configured on peer: inet6-unicast
NLRI advertised by peer: inet6-unicast
NLRI for this session: inet6-unicast
Peer supports Refresh capability (2)
Restart time configured on the peer: 120
Stale routes from peer are kept for: 300
Restart time requested by this peer: 120
Restart flag received from the peer: Notification
NLRI that peer supports restart for: inet6-unicast
NLRI peer can save forwarding state: inet6-unicast
NLRI that peer saved forwarding for: inet6-unicast
NLRI that restart is negotiated for: inet6-unicast
NLRI of received end-of-rib markers: inet6-unicast
NLRI of all end-of-rib markers sent: inet6-unicast
Peer does not support LLGR Restarter functionality
Peer supports 4 byte AS extension (peer-as 102)
Peer does not support Addpath
NLRI(s) enabled for color nexthop resolution: inet6-unicast
Table inet6.0 Bit: 20000
  RIB State: BGP restart is complete
  Send state: in sync
  Active prefixes:              4
  Received prefixes:            4
  Accepted prefixes:            4
  Suppressed due to damping:    0
  Advertised prefixes:          1
Last traffic (seconds): Received 20   Sent 24   Checked 5788
Input messages:  Total 216    Updates 5       Refreshes 0     Octets 4535
Output messages: Total 212    Updates 1       Refreshes 0     Octets 4125
Output Queue[1]: 0            (inet6.0, inet6-unicast)
Trace options:  all
Trace file: /var/log//bgp size 131072 files 10
```

To verify the operation of BFD for the BGP sessions use:

```
show bfd session
```

Example:

```
jnpr@stripe1-leaf1> show bfd session
                                                  Detect   Transmit
Address                   State     Interface      Time     Interval  Multiplier
fe80::5a86:70ff:fe78:e0d5 Up       et-0/0/31:0.0   9.000     3.000        3
fe80::5a86:70ff:fe79:3d5  Up       et-0/0/33:0.0   9.000     3.000        3
fe80::5a86:70ff:fe7b:ced5 Up       et-0/0/32:0.0   9.000     3.000        3
fe80::9e5a:80ff:feef:a28f Up       et-0/0/30:0.0   9.000     3.000        3
8 sessions, 8 clients
Cumulative transmit rate 2.7 pps, cumulative receive rate 2.7 pps
```

To control the propagation of routes, and make sure the loopback interface addresses are advertised, export policies are applied to these EBGP sessions as shown in the example in Table 30.

Table 30. Export policy example IPv6 Underlay with auto discovery

LEAF	SPINE

[edit policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT]
jnpr@stripe1-leaf1# show
term LEAF_TO_SPINE_FABRIC_OUT-10 {
    from {
        protocol bgp;
        community FROM_SPINE_FABRIC_TIER;
    }
    then reject;
}
term LEAF_TO_SPINE_FABRIC_OUT-20 {
    then accept;
}
	[edit policy-options policy-statement SPINE_TO_LEAF_FABRIC_OUT]
	jnpr@spine1# show
	term SPINE_TO_LEAF_FABRIC_OUT-10 {
	    then {
	        community add FROM_SPINE_FABRIC_TIER;
	        accept;
	    }
	}

[edit policy-options community FROM_SPINE_FABRIC_TIER] **jnpr@stripe1-leaf1# show** members 0:15;	**[edit policy-options community FROM_SPINE_FABRIC_TIER]** **jnpr@spine1# show** **members 0:15;**

[edit policy-options policy-statement BGP-AOS-Policy]
jnpr@stripe1-leaf1# show
term BGP-AOS-Policy-10 {
    from policy AllPodNetworks;
    then accept;
}
term BGP-AOS-Policy-20 {
    from {
        protocol evpn;
        route-filter 0::0/0 prefix-length-range /128-/128;
    }
    then accept;
}
term BGP-AOS-Policy-100 {
    then reject;
}
	[edit policy-options policy-statement BGP-AOS-Policy]
	jnpr@spine1# show
	term BGP-AOS-Policy-10 {
	    from policy AllPodNetworks;
	    then accept;
	}
	term BGP-AOS-Policy-20 {
	    from protocol bgp;
	    then accept;
	}
	term BGP-AOS-Policy-100 {
	    then reject;
	}

[edit policy-options policy-statement AllPodNetworks]
jnpr@stripe1-leaf1# show
term AllPodNetworks-10 {
    from {
        family inet6;
        protocol direct;
        interface lo0.0;
    }
    then {
        community add DEFAULT_DIRECT_V6;
        accept;
    }
}
term AllPodNetworks-100 {
    then reject;
}
	[edit policy-options policy-statement AllPodNetworks]
	jnpr@spine1# show
	term AllPodNetworks-10 {
	    from {
	        family inet6;
	        protocol direct;
	        interface lo0.0;
	    }
	    then {
	        community add DEFAULT_DIRECT_V6;
	        accept;
	    }
	}
	term AllPodNetworks-100 {
	    then reject;
	}

[edit policy-options community DEFAULT_DIRECT_V6] **jnpr@stripe1-leaf1# show** members [5:20008 21001:26000];	**[edit policy-options community DEFAULT_DIRECT_V6]** **jnpr@spine1# show** members [1:20008 21001:26000];

These policies ensure loopback reachability without advertising unnecessary routes.

On the spine nodes, routes are exported only if they are accepted by both the *SPINE_TO_LEAF_FABRIC_OUT* and *BGP-AOS-Policy* export policies.

- The *SPINE_TO_LEAF_FABRIC_OUT* policy has no match conditions and accepts all routes unconditionally, tagging them with the **FROM_SPINE_FABRIC_TIER** community (0:15).
- The *BGP-AOS-Policy* accepts BGP-learned routes as well as any routes accepted by the nested *AllPodNetworks* policy.

- The *AllPodNetworks* policy, in turn, matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V6** community (1:20008 and 21001:26000 on Spine1).

As a result, each spine advertises both its directly connected routes (including its loopback interface) and any routes it has received from other leaf nodes.

You can verify that the expected routes are being advertised by the spine node using: `show route advertising-protocol bgp <peer-id> table inet6.0`

Example:

The following example shows the routes advertised to Stripe 1 Leaf 1 by Spine 1 which correspond to the loopback interface addresses of itself, as well as Stripe1 Leaf 2, Stripe 2 Leaf 1, and Stripe 2 Leaf 2.

```
jnpr@spine1> show route advertising-protocol bgp fe80::9e5a:80ff:fec1:ae00%et-0/0/30:0.0 table
inet6.0
inet6.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* fc00:10::1/128          Self                                    I
* fc00:10:0:1::2/128      Self                                    202 I
* fc00:10:0:1::9/128      Self                                    209 I
* fc00:10:0:1::10/128     Self                                    210 I
```

To verify routes are received by the Leaf nodes use: `show route receive-protocol bgp <peer-id> table inet6.0`

Example:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp fe80::5a86:70ff:fe78:e0d5%et-0/0/1:0.0 table
inet6.0
inet6.0: 14 destinations, 23 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* fc00:10::1/128          fe80::9e5a:80ff:feef:a28f               101 I
  fc00:10:0:1::2/128      fe80::9e5a:80ff:feef:a28f               101 202 I
  fc00:10:0:1::9/128      fe80::9e5a:80ff:feef:a28f               101 209 I
  fc00:10:0:1::10/128     fe80::9e5a:80ff:feef:a28f               101 210 I
```

On the **leaf nodes**, routes are exported only if they are accepted by both the *LEAF_TO_SPINE_FABRIC_OUT* and *BGP-AOS-Policy* export policies.

- The *LEAF_TO_SPINE_FABRIC_OUT* policy accepts all routes except those learned via BGP that are tagged with the **FROM_SPINE_FABRIC_TIER** community (0:15). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.

- The *BGP-AOS-Policy* accepts all routes allowed by the nested *AllPodNetworks* policy, which matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V4** community (5:20007 and 21001:26000 for Stripe1-Leaf1).
- As a result, leaf nodes will advertise only their directly connected interface routes, including their loopback interfaces, to the spines.

You can verify that the expected routes are being advertised by the spine node using: `show route advertising-protocol bgp <peer-id> table inet6.0`

Example:

The following example shows the routes advertised to Spine 1 by Stripe 1 Leaf 1.

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp fe80::5a86:70ff:fe78:e0d5%et-0/0/30:0.0
table inet6.0
inet6.0: 14 destinations, 23 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* fc00:10:0:1::1/128      Self                                    I
```

To verify routes are received by the spine node, use: `show route receive-protocol bgp <peer-id> table inet6.0`

Example:

```
jnpr@spine1> show route receive-protocol bgp fe80::9e5a:80ff:fec1:ae00%et-0/0/0:0.0 table
inet6.0
inet6.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* fc00:10:0:1::1/128      fe80::9e5a:80ff:fec1:ae00               201 I
```

IPv6 GPU Backend Fabric Overlay

When EVPN Type 5 is used to implement L3 tenant isolation across a VXLAN fabric, multiple routing tables are instantiated on each participating leaf node. These tables are responsible for managing control-plane separation, enforcing tenant boundaries, and supporting the overlay forwarding model.

Each routing instance (VRF) creates its own set of routing and forwarding tables, in addition to the global and EVPN-specific tables used for fabric-wide communication. These tables are listed in Table 31.

Table 31. Routing and Forwarding Tables for EVPN Type 5

TABLE	DESCRIPTON
bgp.evpn.0	Holds EVPN route information received via BGP, including Type 5 (IP Prefix) routes and other EVPN route types. This is the control plane source for EVPN-learned routes
<tenant-name>.evpn.0	The tenant-specific EVPN table.
<tenant-name>.inet.0	The tenant-specific IPv4 unicast routing table. Contains directly connected and EVPN-imported Type 5 prefixes for that tenant. Used for routing data plane traffic.

When routing instances are created for the tenants, separate routing domains (tenant-name.<tenant-name>.inet6.0) are created, providing full route and traffic isolation across the EVPN/VXLAN fabric.

The protocol next-hop (loopback interface or remote leaf) on each EVPN route is resolved in inet6.0. Then the route is added to the bgp.evpn.0 table. The routes are then imported into <tenant>.evpn.0 and <tenant>.inet6.0, based on route-targets.

The Overlay BGP Sessions between the leaf and spine nodes are statically configured (not auto discovered) using the loopback interfaces global IPv6 addresses, which were advertised by the Underlay BGP sessions.

As an example, consider the configuration between Stripe1 Leaf 1 and Spine 1.

Table 32. GPU Backend Fabric Overlay Using IPv6 Loopback Addresses

STRIPE 1 LEAF 1

```
[edit group l3clos-inet6-auto-overlay]
jnpr@stripe1-leaf1# show
  type external;
  multihop {
    ttl 1;
  }
  multipath {
    multiple-as;
  }
  bfd-liveness-detection {
    minimum-interval 3000;
    multiplier 3;
  }
  /* SPINE 1 */
  neighbor fc00:10::1 {
    description facing_spine1-evpn-overlay;
    local-address fc00:10:0:1::1;
    family evpn {
        signaling;
    }
    export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
    peer-as 101;
  }
  /* SPINE 2 */
  neighbor fc00:10::2 {
    description facing_spine2-evpn-overlay;
    local-address fc00:10:0:1::1;
    family evpn {
        signaling;
    }
    export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
    peer-as 102;
  }
  /* SPINE 3 */
  neighbor fc00:10::3 {
    description facing_spine3-evpn-overlay;
    local-address fc00:10:0:1::1;
    family evpn {
        signaling;
    }
    export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
    peer-as 103;
  }
  /* SPINE 4 */
  neighbor fc00:10::4 {
    description facing_spine4-evpn-overlay;
    local-address fc00:10:0:1::1;
    family evpn {
        signaling;
    }
    export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
    peer-as 104;
  }
  vpn-apply-export;
```

SPINE 1

```
[edit group l3clos-inet6-auto-overlay]
jnpr@spine1# show
  type external;
  multihop {
    ttl 1;
    no-nexthop-change;
  }
  multipath {
    multiple-as;
  }
  bfd-liveness-detection {
    minimum-interval 3000;
    multiplier 3;
  }
  /* LEAF 1 */
  neighbor FC00:10:0:1::1 {
    description facing_leaf1-evpn-overlay;
    local-address FC00:10::1;
    family evpn {
      signaling;
    }
    export ( SPINE_TO_LEAF_EVPN_OUT );
    peer-as 201;
  }
 /* LEAF 2 */
 neighbor FC00:10:0:1::1 {
    description facing_leaf1-evpn-overlay;
    local-address FC00:10::1;
    family evpn {
      signaling;
    }
    export ( SPINE_TO_LEAF_EVPN_OUT );
    peer-as 201;
  }
  /* LEAF 3 */
  neighbor FC00:10:0:1::1 {
    description facing_leaf1-evpn-overlay;
    local-address FC00:10::1;
    family evpn {
      signaling;
    }
    export ( SPINE_TO_LEAF_EVPN_OUT );
    peer-as 201;
  }
  /* LEAF 4 */
  neighbor FC00:10:0:1::1 {
    description facing_leaf1-evpn-overlay;
    local-address FC00:10::1;
    family evpn {
      signaling;
    }
    export ( SPINE_TO_LEAF_EVPN_OUT );
    peer-as 201;
  }
    .
    .
    .
  vpn-apply-export;
```

The sessions use **family evpn signaling** to enable EVPN route exchange. The **multihop ttl 1** statement allows EBGP sessions to be established between the loopback interfaces.

As with the underlay BGP sessions, these sessions are configured with **multipath multiple-as**, allowing multiple EVPN paths with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing. BFD (Bidirectional Forwarding Detection) is also enabled to improve convergence time in case of failures.

The **no-nexthop-change** knob on the spine nodes is used to preserve the original next-hop address, which is critical in EVPN for ensuring that the remote VTEP can be reached directly. The **vpn-apply-**

export statement is included to ensure that the export policies are evaluated for VPN address families, such as EVPN, allowing fine-grained control over which routes are advertised to each peer.

You can check that the sessions have been established using: `show bgp summary group <group-name>`

Example:

```
jnpr@stripe1-leaf1> show bgp summary group l3clos-inet6-auto-overlay
fc00:10:0:1::1          201        118        127       0       0       52:58 Establ
  bgp.evpn.0: 4/4/4/0
fc00:10:0:1::2          202        119        128       0       0       53:01 Establ
  bgp.evpn.0: 4/4/4/0
fc00:10:0:1::9          209        119        127       0       0       53:10 Establ
  bgp.evpn.0: 4/4/4/0
fc00:10:0:1::10         210         81         81       0       3       35:28 Establ
  bgp.evpn.0: 4/4/4/0
```

To verify the operation of BFD for the BGP sessions use: `show bfd session`

Example:

```
jnpr@stripe1-leaf1> show bfd session
                                                  Detect   Transmit
Address                  State     Interface      Time     Interval  Multiplier
fc00:10::1               Up                       9.000     3.000        3
fc00:10::2               Up                       9.000     3.000        3
fc00:10::3               Up                       9.000     3.000        3
fc00:10::4               Up                       9.000     3.000        3
8 sessions, 8 clients
Cumulative transmit rate 2.7 pps, cumulative receive rate 2.7 pps
```

You can check details about discovered neighbors using: `show bgp neighbor <peer-id>`

Example:

```
jnpr@stripe1-leaf1> show bgp neighbor fc00:10::1
Peer: fc00:10::1+48522 AS 101  Local: fc00:10:0:1::1+179 AS 201
  Description: facing_spine1-evpn-overlay
  Group: l3clos-inet6-auto-overlay Routing-Instance: master
  Forwarding routing-instance: master
  Type: External    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ (LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT) ]
  Options: <Multihop LocalAddress GracefulRestart Ttl AddressFamily PeerAS Multipath Rib-group
Refresh>
  Options: <VpnApplyExport MultipathAs BfdEnabled>
  Options: <GracefulShutdownRcv>
  Address families configured: evpn
  Local Address: fc00:10:0:1::1 Holdtime: 90 Preference: 170
  Graceful Shutdown Receiver local-preference: 0
  Number of flaps: 0
  Receive eBGP Origin Validation community: Reject
  Peer ID: 10.0.0.1        Local ID: 10.0.1.1          Active Holdtime: 90
  Keepalive Interval: 30         Group index: 2    Peer index: 3    SNMP index: 61
  I/O Session Thread: bgpio-0 State: Enabled
  BFD: enabled, up
  NLRI for restart configured on peer: evpn
  NLRI advertised by peer: evpn
  NLRI for this session: evpn
  Peer supports Refresh capability (2)
  Restart time configured on the peer: 120
  Stale routes from peer are kept for: 300
  Restart time requested by this peer: 120
  Restart flag received from the peer: Notification
  NLRI that peer supports restart for: evpn
  NLRI peer can save forwarding state: evpn
  NLRI that peer saved forwarding for: evpn
  NLRI that restart is negotiated for: evpn
  NLRI of received end-of-rib markers: evpn
  NLRI of all end-of-rib markers sent: evpn
  Peer does not support LLGR Restarter functionality
  Peer supports 4 byte AS extension (peer-as 101)
  Peer does not support Addpath
  NLRI(s) enabled for color nexthop resolution: evpn
  Table bgp.evpn.0 Bit: 40000
    RIB State: BGP restart is complete
    RIB State: VPN restart is complete
    Send state: in sync
    Active prefixes:              0
    Received prefixes:            12
    Accepted prefixes:            12
    Suppressed due to damping:    0
    Advertised prefixes:          4
  Table Tenant-1.evpn.0
```

```
      RIB State: BGP restart is complete
      RIB State: VPN restart is complete
      Send state: not advertising
      Active prefixes:              0
      Received prefixes:            4
      Accepted prefixes:            4
      Suppressed due to damping:    0
    Last traffic (seconds): Received 14   Sent 11   Checked 3980
    Input messages:  Total 158    Updates 16      Refreshes 0     Octets 6079
    Output messages: Total 146    Updates 1       Refreshes 0     Octets 3105
    Output Queue[3]: 0            (bgp.evpn.0, evpn)
```

To control the propagation of routes, export policies are applied to these EBGP sessions as shown in the example in Table 33.

Table 33. Export Policy example to advertise EVPN routes over IPv6 overlay

LEAF	SPINE

[edit policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT]
jnpr@stripe1-leaf1# show | display set relative
term LEAF_TO_SPINE_EVPN_OUT-10 {
    from {
        protocol bgp;
        community FROM_SPINE_EVPN_TIER;
    }
    then reject;
}
term LEAF_TO_SPINE_EVPN_OUT-20 {
    then accept;
}
	[edit policy-options policy-statement SPINE_TO_LEAF_EVPN_OUT]
	jnpr@spine1# show | display set relative
	term SPINE_TO_LEAF_EVPN_OUT-10 {
	    then {
	        community add FROM_SPINE_EVPN_TIER;
	        accept;
	    }
	}

[edit policy-options community FROM_SPINE_EVPN_TIER] **jnpr@stripe1-leaf1# show | display set relative** members 0:14;	**[edit policy-options community FROM_SPINE_EVPN_TIER]** **jnpr@spine1# show | display set relative** members 0:14;

[edit policy-options policy-statement EVPN_EXPORT]
jnpr@stripe1-leaf1# show | display set relative
term EVPN_EXPORT-10 {
    then accept;
}	

These policies are simpler in structure and are intended to enable end-to-end EVPN reachability between tenant GPUs, while preventing route loops within the overlay.

NOTE: Routes will only be advertised if EVPN routing-instances have been created, as described in the Per Tenant IP-VRF Routing Instances section.

On the **spine nodes**, routes are exported if they are accepted by the ***SPINE_TO_LEAF_EVPN_OUT*** policy.

- The ***SPINE_TO_LEAF_EVPN_OUT*** policy has no match conditions and accepts all routes. It tags each exported route with the **FROM_SPINE_EVPN_TIER** community (0:14).

As a result, the spine nodes export EVPN routes received from one leaf to all other leaf nodes, allowing tenant-to-tenant communication across the fabric.

You can verify that the expected routes are being advertised by the spine node using:`show route advertising-protocol bgp <peer-id> table bgp.evpn.0show route advertising-protocol bgp <peer-id> match-prefix <prefix>`

Example:

```
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:1::1 table bgp.evpn.0
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
*                         fc00:10:0:1::2                          202 I
  5:10.0.1.2:2002::0::fc00:200:2:1::::64/248
*                         fc00:10:0:1::2                          202 I
  5:10.0.1.2:2002::0::fc00:200:2:2::::64/248
*                         fc00:10:0:1::2                          202 I
  5:10.0.1.2:2002::0::fc00:200:2:3::::64/248
*                         fc00:10:0:1::2                          202 I
  5:10.0.1.9:2001::0::fc00:100:2:1::::64/248
*                         fc00:10:0:1::9                          209 I
  5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
*                         fc00:10:0:1::9                          209 I
  5:10.0.1.9:2001::0::fc00:200:1:10::::64/248
*                         fc00:10:0:1::9                          209 I
  5:10.0.1.9:2001::0::fc00:200:1:11::::64/248
*                         fc00:10:0:1::9                          209 I
  5:10.0.1.10:2002::0::fc00:100:2:2::::64/248
*                         fc00:10:0:1::10                         210 I
  5:10.0.1.10:2002::0::fc00:200:2:9::::64/248
*                         fc00:10:0:1::10                         210 I
  5:10.0.1.10:2002::0::fc00:200:2:10::::64/248
*                         fc00:10:0:1::10                         210 I
  5:10.0.1.10:2002::0::fc00:200:2:11::::64/248
*                         fc00:10:0:1::10                         210 I
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:1::1 match-prefix
5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
```

```
*                         fc00:10:0:1::2                          202 I
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:1::1 match-prefix
5:10.0.1.2:2002::0::fc00:100:1:2::::64/248 extensive
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
* 5:10.0.1.2:2002::0::fc00:100:1:2::::64/248 (1 entry, 1 announced)
 BGP group l3clos-inet6-auto-overlay type External
     Route Distinguisher: 10.0.1.2:2002
     Route Label: 20002
     Overlay gateway address: ::
     Nexthop: fc00:10:0:1::2
     AS path: [101] 202 I
     Communities: 0:14 5:20008 21002:26000 target:20002:1 encapsulation:vxlan(0x8) router-
mac:58:86:70:79:df:db
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:1::1 match-prefix
5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
*                         fc00:10:0:1::9                          209 I
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:1::1 match-prefix
5:10.0.1.9:2001::0::fc00:200:1:9::::64/248 extensive
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
* 5:10.0.1.9:2001::0::fc00:200:1:9::::64/248 (1 entry, 1 announced)
 BGP group l3clos-inet6-auto-overlay type External
     Route Distinguisher: 10.0.1.9:2001
     Route Label: 20001
     Overlay gateway address: ::
     Nexthop: fc00:10:0:1::9
     AS path: [101] 209 I
     Communities: 0:14 5:20008 21001:26000 target:20001:1 encapsulation:vxlan(0x8) router-
mac:58:86:70:7b:10:db
```

The leaf nodes receive the routes and first install them in the bgp.evpn.0 routing table which can be verified using:`show route receive-protocol bgp <peer-id> table bgp.evpn.0`

Example:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp fc00:10::1 table bgp.evpn.0
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
```

```
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:1::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:2::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:3::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.9:2001::0::fc00:100:2:1::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:10::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:11::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.10:2002::0::fc00:100:2:2::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:9::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:10::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:11::::64/248
                          fc00:10:0:1::10                         101 210 I
jnpr@stripe1-leaf1> show route receive-protocol bgp fc00:10::1 table bgp.evpn.0 match-prefix
5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
                          fc00:10:0:1::2                          101 202 I
jnpr@stripe1-leaf1> show route receive-protocol bgp fc00:10::1 table bgp.evpn.0 match-prefix
5:10.0.1.2:2002::0::fc00:100:1:2::::64/248 extensive
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248 (4 entries, 0 announced)
     Accepted
     Route Distinguisher: 10.0.1.2:2002
     Route Label: 20002
     Overlay gateway address: ::
     Nexthop: fc00:10:0:1::2
```

```
    AS path: 101 202 I
    Communities: 0:14 5:20008 21002:26000 target:20002:1 encapsulation:vxlan(0x8) router-
mac:58:86:70:79:df:db
```

On the leaf nodes, routes are exported if they are accepted by both the *LEAF_TO_SPINE_EVPN_OUT* and *EVPN_EXPORT* policies.

- The *LEAF_TO_SPINE_EVPN_OUT* policy rejects any BGP-learned routes that carry the *FROM_SPINE_EVPN_TIER* community (0:14). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.
- The *EVPN_EXPORT* policy accepts all routes without additional conditions.

As a result, the leaf nodes export only locally originated EVPN routes for the directly connected interfaces between GPU servers and the leaf nodes. These routes are part of the **tenant routing instances** and are used to establish reachability between GPUs belonging to the same tenant.

You can verify that the expected routes are being advertised by the Leaf nodes using: `show route advertising-protocol bgp <peer-id> table bgp.evpn.0` `show route advertising-protocol bgp <peer-id> match-prefix <prefix>`

Example:

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table bgp.evpn.0
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:1::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:2::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:3::::64/248
*                         Self                                    I
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table bgp.evpn.0 match-prefix
5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
*                         Self                                    I
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table bgp.evpn.0 match-prefix
```

```
5:10.0.1.1:2001::0::fc00:100:1:1::::64/248 extensive
bgp.evpn.0: 16 destinations, 52 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
* 5:10.0.1.1:2001::0::fc00:100:1:1::::64/248 (1 entry, 1 announced)
 BGP group l3clos-inet6-auto-overlay type External
     Route Distinguisher: 10.0.1.1:2001
     Route Label: 20001
     Overlay gateway address: ::
     Nexthop: Self
     Flags: Nexthop Change
     AS path: [201] I
     Communities: 5:20008 21001:26000 target:20001:1 encapsulation:vxlan(0x8) router-
mac:9c:5a:80:c1:b3:06
```

To verify routes are received by the spine nodes use: `show route receive-protocol bgp <peer-id> table bgp.evpn.0`

Example:

```
jnpr@spine1> show route receive-protocol bgp fc00:10:0:1::1 table bgp.evpn.0
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
*                         fc00:10:0:1::1                          201 I
  5:10.0.1.1:2001::0::fc00:200:1:1::::64/248
*                         fc00:10:0:1::1                          201 I
  5:10.0.1.1:2001::0::fc00:200:1:2::::64/248
*                         fc00:10:0:1::1                          201 I
  5:10.0.1.1:2001::0::fc00:200:1:3::::64/248
*                         fc00:10:0:1::1                          201 I
jnpr@spine1> show route receive-protocol bgp fc00:10:0:1::1 match-prefix
5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
*                         fc00:10:0:1::1                          201 I
jnpr@spine1> show route receive-protocol bgp fc00:10:0:1::1 match-prefix
5:10.0.1.1:2001::0::fc00:100:1:1::::64/248 extensive
bgp.evpn.0: 16 destinations, 16 routes (16 active, 0 holddown, 0 hidden)
Restart Complete
```

```
* 5:10.0.1.1:2001::0::fc00:100:1:1::::64/248 (1 entry, 1 announced)
     Accepted
     Route Distinguisher: 10.0.1.1:2001
     Route Label: 20001
     Overlay gateway address: ::
     Nexthop: fc00:10:0:1::1
     AS path: 201 I
     Communities: 5:20008 21001:26000 target:20001:1 encapsulation:vxlan(0x8) router-
mac:9c:5a:80:c1:b3:06
```

Tenants IP-VRF Routing Instances

Stripe 1 Leaf 1 and Stripe 1 Leaf 2 have been configured for Tenant-1 and Tenant-2 respectively as shown in Table 34. Stripe 2 Leaf 1 and Stripe 2 Leaf 2 are configured similarly.

Table 34. EVPN Routing-Instance for Tenant-1 and Tenant-2 Across Stripe 1 and Stripe 2

STRIPE 1 – LEAF 1	STRIPE 1 – LEAF 2
[edit routing-instances Tenant-1]	**[edit routing-instances Tenant-2]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe1-leaf2# show | display set relative**
instance-type vrf;	instance-type vrf;
routing-options {	routing-options {
rib Tenant-1.inet6.0 {	rib Tenant-2.inet6.0 {
multipath;	multipath;
}	}
graceful-restart;	graceful-restart;
multipath;	multipath;
}	}
protocols {	protocols {
evpn {	evpn {
ip-prefix-routes {	ip-prefix-routes {
advertise direct-nexthop;	advertise direct-nexthop;
encapsulation vxlan;	encapsulation vxlan;
vni 20001;	vni 20002;
export BGP-AOS-Policy-Tenant-1;	export BGP-AOS-Policy-Tenant-2;
}	}
}	}
}	}
/* SERVER 1 GPU0 TENANT-1 */	/* SERVER 1 GPU1 TENANT-2 */
interface et-0/0/0:0.0;	interface et-0/0/0:0.0;
/* SERVER 2 GPU0 TENANT-1 */	/* SERVER 2 GPU1 TENANT-2 */
interface et-0/0/1:0.0;	interface et-0/0/1:0.0;
/* SERVER 3 GPU0 TENANT-1 */	/* SERVER 3 GPU1 TENANT-2 */
interface et-0/0/2:0.0;	interface et-0/0/2:0.0;
interface lo0.1;	interface lo0.2;
route-distinguisher 10.0.1.1:2001;	route-distinguisher 10.0.1.1:2002;
vrf-target target:20001:1;	vrf-target target:20002:1;
[edit interfaces lo0]	**[edit interfaces lo0]**
jnpr@stripe1-leaf1# show unit 1	**jnpr@stripe1-leaf2# show unit 2**
family inet6 {	family inet6 {
address fc00:101:1:1::1/64;	address fc00:102:1:2::1/64;
}	}

Table 35. Policies Examples for Tenant-1 and Tenant-2

TENANT-1 POLICIES	TENANT-2 POLICIES

**[edit policy-options policy-statement BGP-AOS-Policy-
Tenant-1]**
jnpr@stripe1-leaf1# show | display set relative
term BGP-AOS-Policy-Tenant-1-10 {
    from policy AllPodNetworks-Tenant-1;
    then accept;
}
term BGP-AOS-Policy-Tenant-1-20 {
    from {
        protocol evpn;
        route-filter 0::0/0 prefix-length-range /128-/128;
    }
    then {
        community add TENANT-1_COMMUNITY_V6;
        accept;
    }
}
term BGP-AOS-Policy-Tenant-1-100 {
    then reject;
}
 	
**[edit policy-options policy-statement AllPodNetworks-
Tenant-1]**
jnpr@stripe1-leaf1# show | display set relative
term AllPodNetworks-Tenant-1-10 {
    from {
        family inet6;
        protocol direct;
    }
    then {
        community add TENANT-1_COMMUNITY_V6;
        accept;
    }
}
term AllPodNetworks-Tenant-1-100 {
    then reject;
}
 	
[edit policy-options community TENANT-A_COMMUNITY_V6]
jnpr@stripe1-leaf1# show | display set relative
members [5:20008 21001:26000];
	**[edit policy-options policy-statement BGP-AOS-Policy-
	Tenant-2]**
	jnpr@stripe1-leaf1# show | display set relative
	term BGP-AOS-Policy-Tenant-2-10 {
	    from policy AllPodNetworks-Tenant-2;
	    then accept;
	}
	term BGP-AOS-Policy-Tenant-2-20 {
	    from {
	        protocol evpn;
	        route-filter 0::0/0 prefix-length-range /128-/128;
	    }
	    then {
	        community add TENANT-2_COMMUNITY_V6;
	        accept;
	    }
	}
	term BGP-AOS-Policy-Tenant-2-100 {
	    then reject;
	}
	
	**[edit policy-options policy-statement AllPodNetworks-
	Tenant-2]**
	jnpr@stripe1-leaf1# show | display set relative
	term AllPodNetworks-Tenant-2-10 {
	    from {
	        family inet6;
	        protocol direct;
	    }
	    then {
	        community add TENANT-2_COMMUNITY_V6;
	        accept;
	    }
	}
	term AllPodNetworks-Tenant-2-100 {
	    then reject;
	}
	
	[edit policy-options community TENANT-B_COMMUNITY_V6]
	jnpr@stripe1-leaf1# show | display set relative
	members [5:20008 21002:26000];

Each routing instance is configured with the following key elements:

1. **Interfaces:**

   The interfaces listed under each tenant VRF (e.g. et-0/0/0:0.0 and et-0/0/1:0.0) are explicitly added to the corresponding routing table. By placing these interfaces under the VRF, all routing decisions and traffic forwarding associated with them are isolated from other tenants and from the global routing table. Assigning an interface that connects a particular GPU to the leaf node effectively maps that GPU to a specific tenant, isolating it from GPUs assigned to other tenants.

2. **Route-distinguisher (RD):**

   10.0.1.1:2001 and 10.0.1.1:2002 uniquely identify EVPN routes from Tenant-1 and Tenant-2, respectively. Even if both tenants use overlapping IP prefixes, the RD ensures their routes remain distinct in the BGP control plane. Although the GPU to leaf links use unique /32 prefixes, an RD is still required to advertise these routes over EVPN.

3. **Route target (RT) community:**

   VRF targets 20001:1 and 20002:1 control which routes are exported from and imported into each tenant routing table. These values determine which routes are shared between VRFs that belong to the same tenant across the fabric and are essential for enabling fabric-wide tenant connectivity, for example, when a tenant has GPUs assigned to multiple servers across different stripes.

4. **Protocols evpn parameters:**

   - The ip-prefix-routes controls how IP Prefix Routes (EVPN Type 5 routes) are advertised.

   - The advertise direct-nexthop enables the leaf node to send IP prefix information using EVPN pure Type 5 routes, which includes a router MAC extended community. These routes include a Router MAC extended community, which allows the remote VTEP to resolve the next-hop MAC address without relying on Type 2 routes.

   - The **encapsulation vxlan** indicates that the payload traffic for this tenant will be encapsulated using VXLAN. The same type of encapsulation must be used end to end.

   - The **VXLAN Network Identifier (VNI)** acts as the encapsulation tag for traffic sent across the EVPN/VXLAN fabric. When EVPN Type 5 (IP Prefix) routes are advertised, the associated VNI is included in the BGP update. This ensures that remote VTEPs can identify the correct VXLAN segment for returning traffic to the tenant's VRF.

   - Unlike traditional use cases where a VNI maps to a single Layer 2 segment, in EVPN Type 5 the VNI represents the **tenant-wide Layer 3 routing domain**. All point-to-point subnets, such as the /32 links between GPU servers and the leaf, that belong to the same VRF are advertised with the same VNI.

   In this configuration, VNIs 20001 and 20002 are mapped to the Tenant-1 and Tenant-2 VRFs, respectively. All traffic destined for interfaces in Tenant-1 will be forwarded using VNI 20001, and all traffic for Tenant-2 will use VNI 20002.

   Notice that the same VNI for a specific tenant is configured on both Stripe1-Leaf1 and Stripe2-Leaf1.

5. **Export Policy Logic**

   EVPN Type 5 routes from Tenant-1 are exported if they are accepted by the BGP-AOS-Policy-Tenant-1 export policy, which references a nested policy named ***AllPodNetworks-Tenant-1*** (and the equivalent policies for Tenant-2)

   - **Policy *BGP-AOS-Policy-Tenant-1*** controls which prefixes from this VRF are allowed to be advertised into EVPN. It accepts any route that is permitted by the ***AllPodNetworks-Tenant-1*** policy and explicitly rejects all other routes.

   - **Policy *AllPodNetworks-Tenant-1*** accepts directly connected IPv4 routes (family inet, protocol direct) that are part of the Tenant-1 VRF. It tags these routes with the **TENANT-1 _COMMUNITY_V4 (5:20007 21002:26000)** community before accepting them. All other routes are rejected.

   As a result, only the directly connected IPv4 routes from the Tenant-1 (/32 links between GPU servers and the leaf) are exported as EVPN Type 5 routes.

To verify that the interfaces have been assigned to the correct routing instance and installed in the correct tenant's routing table use: `show interfaces routing-instance <tenant-name> terse`

Example:

```
jnpr@stripe1-leaf1> show interfaces routing-instance Tenant-1 terse
Interface               Admin Link Proto     Local                 Remote
et-0/0/0:0.0             up    up   inet6    fc00:200:1:1::1/64
                                             fe80::9e5a:80ff:fec1:ae60/64
                                    multiservice
et-0/0/1:0.0             up    up   inet6    fc00:200:1:2::1/64
                                             fe80::9e5a:80ff:fec1:ae61/64
                                    multiservice
et-0/0/2:0.0             up    up   inet6    fc00:200:1:3::1/64
                                             fe80::9e5a:80ff:fec1:ae68/64
                                    multiservice
lo0.1                    up    up   inet6    fc00:100:1:1::1/64
                                             fe80::9e5a:80f0:c1:b2ff-->
jnpr@stripe1-leaf2> show interfaces routing-instance Tenant-2 terse
Interface                Admin Link Proto    Local                 Remote
et-0/0/0:0.0             up    up   inet6    fc00:200:2:1::1/64
                                             fe80::5a86:70ff:fe79:db35/64
                                    multiservice
et-0/0/1:0.0             up    up   inet6    fc00:200:2:2::1/64
                                             fe80::5a86:70ff:fe79:db36/64
                                    multiservice
et-0/0/2:0.0             up    up   inet6    fc00:200:2:3::1/64
                                             fe80::5a86:70ff:fe79:db3d/64
                                    multiservice
lo0.2                    up    up   inet6    fc00:100:1:2::1/64
                                             fe80::5a86:70f0:79:dfd4-->
```

You can also check the direct routes installed to the correspondent routing table using: `show route protocol direct table <tenant-name>.inet6.0`

Example:

```
jnpr@stripe1-leaf1> show route protocol direct table Tenant-1.inet6.0
Tenant-1.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
```

```
fc00:100:1:1::/64  *[Direct/0] 01:02:26
                    >  via lo0.1
fc00:200:1:1::/64  *[Direct/0] 01:10:04
                    >  via et-0/0/12:0.0
fc00:200:1:2::/64  *[Direct/0] 01:10:04
                    >  via et-0/0/12:1.0
fc00:200:1:3::/64  *[Direct/0] 01:10:04
                    >  via et-0/0/13:0.0
fe80::9e5a:80f0:c1:b2ff/128
                   *[Direct/0] 03:22:19
                    >  via lo0.1
jnpr@stripe1-leaf2> show route protocol direct table Tenant-2.inet6.0
Tenant-2.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:1:2::/64  *[Direct/0] 00:24:41
                    >  via lo0.2
fc00:200:2:1::/64  *[Direct/0] 00:24:41
                    >  via et-0/0/12:0.0
fc00:200:2:2::/64  *[Direct/0] 00:24:41
                    >  via et-0/0/12:1.0
fc00:200:2:3::/64  *[Direct/0] 00:24:41
                    >  via et-0/0/13:0.0
fe80::5a86:70f0:79:dfd4/128
                   *[Direct/0] 00:24:41
                    >  via lo0.2
jnpr@stripe2-leaf1> show route protocol direct table Tenant-1.inet6.0
Tenant-1.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:2:1::/64  *[Direct/0] 00:25:28
                    >  via lo0.1
fc00:200:1:9::/64  *[Direct/0] 00:25:17
                    >  via et-0/0/12:0.0
fc00:200:1:10::/64 *[Direct/0] 00:25:17
                    >  via et-0/0/12:1.0
fc00:200:1:11::/64 *[Direct/0] 00:25:17
                    >  via et-0/0/13:0.0
fe80::5a86:70f0:7b:10d4/128
                   *[Direct/0] 00:25:28
                    >  via lo0.1
```

```
jnpr@stripe2-leaf2> show route protocol direct table Tenant-2.inet6.0
Tenant-2.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:2:2::/64  *[Direct/0] 00:24:51
                    >  via lo0.2
fc00:200:2:9::/64  *[Direct/0] 00:24:40
                    >  via et-0/0/12:0.0
fc00:200:2:10::/64 *[Direct/0] 00:24:40
                    >  via et-0/0/12:1.0
fc00:200:2:11::/64 *[Direct/0] 00:24:40
                    >  via et-0/0/13:0.0
fe80::5a86:70f0:79:99d4/128
                   *[Direct/0] 00:24:51
                    >  via lo0.2
```

To verify evpn l3 contexts including encapsulation, VNI, router MAC address use: `show evpn l3-contextshow evpn l3-context <tenant-name> extensive`

Example:

```
jnpr@stripe1-leaf1> show evpn l3-context
L3 context Type  Adv Encap  VNI/Label  Router MAC/GW intf dt4-sid dt6-sid dt46-sid
Tenant-1         Cfg Direct   VXLAN  20001      9c:5a:80:c1:b3:06
jnpr@stripe1-leaf2> show evpn l3-context
L3 context Type  Adv Encap  VNI/Label  Router MAC/GW intf dt4-sid dt6-sid dt46-sid
Tenant-2         Cfg Direct   VXLAN  20002      58:86:70:79:df:db
jnpr@stripe2-leaf1> show evpn l3-context
L3 context Type  Adv Encap  VNI/Label  Router MAC/GW intf dt4-sid dt6-sid dt46-sid
Tenant-1         Cfg Direct   VXLAN  20001      58:86:70:7b:10:db
jnpr@stripe2-leaf2> show evpn l3-context
L3 context Type  Adv Encap  VNI/Label  Router MAC/GW intf dt4-sid dt6-sid dt46-sid
Tenant-2         Cfg Direct   VXLAN  20002      58:86:70:79:99:db
jnpr@stripe1-leaf1> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-1
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 9c:5a:80:c1:b3:06
  Encapsulation: VXLAN, VNI: 20001
  IPv6 source VTEP address: fc00:10:0:1::1
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-1
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
```

```
EXPORT-POLCIY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.1:2001
  Reference count: 9
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf2> show evpn l3-context Tenant-2 extensive
L3 context: Tenant-2
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 58:86:70:79:df:db
  Encapsulation: VXLAN, VNI: 20002
  IPv6 source VTEP address: fc00:10:0:1::2
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-2
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLCIY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.2:2002
  Reference count: 9
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf1> show evpn ip-prefix-database
L3 context: Tenant-1
IPv6->EVPN Exported Prefixes
Prefix                                       EVPN route status
fc00:100:1:1::/64                            Created
fc00:200:1:1::/64                            Created
fc00:200:1:2::/64                            Created
fc00:200:1:3::/64                            Created
EVPN->IPv6 Imported Prefixes
Prefix                                       Etag
fc00:100:2:1::/64                            0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.9:2001          20001                   58:86:70:7b:10:db  fc00:10:0:1::9
Accepted      n/a
fc00:200:1:9::/64                            0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.9:2001          20001                   58:86:70:7b:10:db  fc00:10:0:1::9
Accepted      n/a
fc00:200:1:10::/64                           0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
```

```
  10.0.1.9:2001         20001                  58:86:70:7b:10:db  fc00:10:0:1::9
Accepted      n/a
fc00:200:1:11::/64                         0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.9:2001         20001                  58:86:70:7b:10:db  fc00:10:0:1::9
Accepted      n/a
jnpr@stripe1-leaf2> show evpn ip-prefix-database
L3 context: Tenant-2
IPv6->EVPN Exported Prefixes
Prefix                                     EVPN route status
fc00:100:1:2::/64                          Created
fc00:200:2:1::/64                          Created
fc00:200:2:2::/64                          Created
fc00:200:2:3::/64                          Created
EVPN->IPv6 Imported Prefixes
Prefix                                     Etag
fc00:100:2:2::/64                          0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.10:2002        20002                  58:86:70:79:99:db  fc00:10:0:1::10
Accepted      n/a
fc00:200:2:9::/64                          0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.10:2002        20002                  58:86:70:79:99:db  fc00:10:0:1::10
Accepted      n/a
fc00:200:2:10::/64                         0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.10:2002        20002                  58:86:70:79:99:db  fc00:10:0:1::10
Accepted      n/a
fc00:200:2:11::/64                         0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.10:2002        20002                  58:86:70:79:99:db  fc00:10:0:1::10
Accepted      n/a
```

You can verify that the expected routes for each tenant are being advertised by the leaf nodes using: `show route advertising-protocol bgp <peer-id> table <tenant-name>.evpn.0`

Example:

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table Tenant-1.evpn.0
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:1::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:2::::64/248
*                         Self                                    I
  5:10.0.1.1:2001::0::fc00:200:1:3::::64/248
*                         Self                                    I
jnpr@stripe1-leaf2> show route advertising-protocol bgp FC00:10::1 table Tenant-2.evpn.0
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
*                         Self                                    I
  5:10.0.1.2:2002::0::fc00:200:2:1::::64/248
*                         Self                                    I
  5:10.0.1.2:2002::0::fc00:200:2:2::::64/248
*                         Self                                    I
  5:10.0.1.2:2002::0::fc00:200:2:3::::64/248
*                         Self                                    I
jnpr@stripe2-leaf1> show route advertising-protocol bgp FC00:10::1 table Tenant-1.evpn.0
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.9:2001::0::fc00:100:2:1::::64/248
*                         Self                                    I
  5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
*                         Self                                    I
  5:10.0.1.9:2001::0::fc00:200:1:10::::64/248
*                         Self                                    I
  5:10.0.1.9:2001::0::fc00:200:1:11::::64/248
*                         Self                                    I
jnpr@stripe2-leaf2> show route advertising-protocol bgp FC00:10::1 table Tenant-2.evpn.0
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.10:2002::0::fc00:100:2:2::::64/248
*                         Self                                    I
  5:10.0.1.10:2002::0::fc00:200:2:9::::64/248
*                         Self                                    I
  5:10.0.1.10:2002::0::fc00:200:2:10::::64/248
*                         Self                                    I
  5:10.0.1.10:2002::0::fc00:200:2:11::::64/248
*                         Self                                    I
```

You can verify that the expected routes for each tenant, are being received by the leaf nodes, and installed in the correct routing table use: `show route receive-protocol bgp <peer-id> table <tenant-name>.evpn.0show route table Tenant-1.inet6.0 protocol evpn`

Example:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp FC00:10::1 table Tenant-1.evpn.0
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.9:2001::0::fc00:100:2:1::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:9::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:10::::64/248
                          fc00:10:0:1::9                          101 209 I
  5:10.0.1.9:2001::0::fc00:200:1:11::::64/248
                          fc00:10:0:1::9                          101 209 I
jnpr@stripe1-leaf1> show route table Tenant-1.inet6.0 protocol evpn
Tenant-1.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:2:1::/64  *[EVPN/170] 00:20:14
                       to fe80::9e5a:80ff:feef:a28f via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0d5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:ced5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3d5 via et-0/0/33:0.0
fc00:200:1:9::/64  *[EVPN/170] 00:20:14
                       to fe80::9e5a:80ff:feef:a28f via et-0/0/30:0.0
                    >  to fe80::5a86:70ff:fe78:e0d5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:ced5 via et-0/0/32:0.0
                       to fe80::5a86:70ff:fe79:3d5 via et-0/0/33:0.0
```

```
fc00:200:1:10::/64 *[EVPN/170] 00:20:14
                       to fe80::9e5a:80ff:feef:a28f via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0d5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:ced5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3d5 via et-0/0/33:0.0
fc00:200:1:11::/64 *[EVPN/170] 00:20:14
                       to fe80::9e5a:80ff:feef:a28f via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0d5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:ced5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3d5 via et-0/0/33:0.0
jnpr@stripe1-leaf2> show route receive-protocol bgp FC00:10::1 table Tenant-2.evpn.0
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.10:2002::0::fc00:100:2:2::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:9::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:10::::64/248
                          fc00:10:0:1::10                         101 210 I
  5:10.0.1.10:2002::0::fc00:200:2:11::::64/248
                          fc00:10:0:1::10                         101 210 I
jnpr@stripe1-leaf2> show route table Tenant-2.inet6.0 protocol evpn
Tenant-2.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:2:2::/64  *[EVPN/170] 00:22:12
                       to fe80::9e5a:80ff:feef:a297 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0dd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cedd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3dd via et-0/0/33:0.0
fc00:200:2:9::/64  *[EVPN/170] 00:22:12
                       to fe80::9e5a:80ff:feef:a297 via et-0/0/30:0.0
                    >  to fe80::5a86:70ff:fe78:e0dd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cedd via et-0/0/32:0.0
                       to fe80::5a86:70ff:fe79:3dd via et-0/0/33:0.0
fc00:200:2:10::/64 *[EVPN/170] 00:22:12
                       to fe80::9e5a:80ff:feef:a297 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0dd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cedd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3dd via et-0/0/33:0.0
fc00:200:2:11::/64 *[EVPN/170] 00:22:12
```

```
                       to fe80::9e5a:80ff:feef:a297 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0dd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cedd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3dd via et-0/0/33:0.0
jnpr@stripe2-leaf1> show route receive-protocol bgp FC00:10::1 table Tenant-1.evpn.0
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::fc00:100:1:1::::64/248
                          fc00:10:0:1::1                          101 201 I
  5:10.0.1.1:2001::0::fc00:200:1:1::::64/248
                          fc00:10:0:1::1                          101 201 I
  5:10.0.1.1:2001::0::fc00:200:1:2::::64/248
                          fc00:10:0:1::1                          101 201 I
  5:10.0.1.1:2001::0::fc00:200:1:3::::64/248
                          fc00:10:0:1::1                          101 201 I
jnpr@stripe2-leaf1> show route table Tenant-1.inet6.0 protocol evpn
Tenant-1.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:1:1::/64  *[EVPN/170] 00:22:04
                       to fe80::9e5a:80ff:feef:a2af via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0f5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cef5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3f5 via et-0/0/33:0.0
fc00:200:1:1::/64  *[EVPN/170] 00:22:04
                       to fe80::9e5a:80ff:feef:a2af via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0f5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cef5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3f5 via et-0/0/33:0.0
fc00:200:1:2::/64  *[EVPN/170] 00:22:04
                       to fe80::9e5a:80ff:feef:a2af via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0f5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cef5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3f5 via et-0/0/33:0.0
fc00:200:1:3::/64  *[EVPN/170] 00:22:04
                       to fe80::9e5a:80ff:feef:a2af via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0f5 via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cef5 via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3f5 via et-0/0/33:0.0
jnpr@stripe2-leaf2> show route receive-protocol bgp FC00:10::1 table Tenant-2.evpn.0
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
```

```
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::fc00:100:1:2::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:1::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:2::::64/248
                          fc00:10:0:1::2                          101 202 I
  5:10.0.1.2:2002::0::fc00:200:2:3::::64/248
                          fc00:10:0:1::2                          101 202 I
jnpr@stripe2-leaf2> show route table Tenant-2.inet6.0 protocol evpn
Tenant-2.inet6.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
fc00:100:1:2::/64  *[EVPN/170] 00:22:16
                       to fe80::9e5a:80ff:feef:a2b7 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0fd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cefd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3fd via et-0/0/33:0.0
fc00:200:2:1::/64  *[EVPN/170] 00:22:16
                       to fe80::9e5a:80ff:feef:a2b7 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0fd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cefd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3fd via et-0/0/33:0.0
fc00:200:2:2::/64  *[EVPN/170] 00:22:16
                       to fe80::9e5a:80ff:feef:a2b7 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0fd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cefd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3fd via et-0/0/33:0.0
fc00:200:2:3::/64  *[EVPN/170] 00:22:16
                       to fe80::9e5a:80ff:feef:a2b7 via et-0/0/30:0.0
                       to fe80::5a86:70ff:fe78:e0fd via et-0/0/31:0.0
                       to fe80::5a86:70ff:fe7b:cefd via et-0/0/32:0.0
                    >  to fe80::5a86:70ff:fe79:3fd via et-0/0/33:0.0
```

Type 5 EVPN/VXLAN GPU Backend Fabric, SLAAC IPv6 Overlay over IPv6 Link-Local Underlay - IP Services

IN THIS SECTION

In this section, we describe the strategies employed to address traffic congestion and optimize load distribution within the Backend GPU fabric.

Congestion Management and Congestion Control Configuration

Congestion management and congestion control are implemented through a **VXLAN-aware Data Center Quantized Congestion Notification (DCQCN)** approach, ensuring traffic fairness and maintaining stability across the lossless fabric.

AI clusters impose unique demands on network infrastructure due to their **high-density** and **low-entropy** traffic patterns, characterized by frequent elephant flows and minimal flow variability. Moreover, most AI training workloads require uninterrupted, lossless packet delivery to be completed successfully. Consequently, when designing a network infrastructure for AI traffic flows, the key objectives include **maximizing throughput**, **minimizing latency**, and **minimizing network interference** while ensuring **lossless operation**. These requirements necessitate the deployment of effective congestion control mechanisms.

Data Center Quantized Congestion Notification (DCQCN) has become the industry-standard method for end-to-end congestion control in **RoCEv2** environments. DCQCN provides mechanisms to adjust traffic rates in response to congestion events without relying on packet drops, striking a balance between reducing traffic rates and maintaining ongoing traffic flow.

It is important to note that DCQCN is primarily required in the **GPU backend fabric**, where the majority of AI workload traffic resides. It is generally unnecessary in the Frontend and Storage Backend fabrics.

DCQCN combines two complementary mechanisms to implement flow and congestion control:

- Priority-based Flow Control (PFC)
- Explicit Congestion Notification (ECN)

Priority-Based Flow Control (PFC) mitigates data loss by pausing traffic transmission for specific traffic classes, based on **IEEE 802.1p priorities** or **DSCP markings** mapped to queues.

When congestion is detected, PFC operates by sending **PAUSE control frames** upstream, requesting the sender to halt transmission of traffic associated with a specific priority. The sender completely stops sending traffic for that priority until the congestion subsides or the PAUSE timer expires.

While PFC prevents packet drops and allows the receiver to catch up, it also impacts application performance for traffic using the affected queues. Furthermore, resuming transmission after a pause can lead to sudden traffic surges, potentially re-triggering congestion. For these reasons, PFC should be configured carefully so that it is used as a last resource.

Explicit Congestion Notification (ECN) offers a proactive congestion signaling mechanism, reducing transmission rates while allowing traffic to continue flowing during congestion periods.

When congestion occurs, ECN bits in the IP header are marked (11), prompting the receiver to generate **Congestion Notification Packets (CNPs)**, which inform the source to throttle its transmission rate. Unlike PFC, ECN aims to gradually reduce congestion without halting traffic completely or triggering packet drops.

When deploying ECN in a VXLAN overlay, it is essential to ensure that **ECN markings from the outer VXLAN/IP headers are copied into the inner payload headers.** This enables congestion signals detected in the transport layer (the VXLAN network) to correctly propagate to the inner RoCEv2 flows, ensuring that the devices generating the RoCEv2 traffic can be notified of congestion so they can respond accordingly. The QFX5240 switches will perform this function automatically without the need for any additional configuration.

Best Practice: Combining **PFC** and **ECN** provides the most effective congestion control strategy in a lossless IP fabric supporting RoCEv2. Their parameters must be carefully tuned so that ECN mechanisms are triggered before PFC.

For more detailed guidance, refer to *Introduction to Congestion Control in Juniper AI Networks* , which outlines best practices for building lossless fabrics for AI workloads using DCQCN (ECN and PFC) congestion control methods alongside Dynamic Load Balancing (DLB). The document is based on validation against DLRM training models and demonstrates how ECN thresholds, PFC parameters, input drops, and tail drops can be monitored and adjusted to optimize fabric performance for RoCEv2 traffic.

NOTE: While we provide general recommendations and lab-validated parameters, each AI workload may present distinct traffic patterns. Class of Service (CoS) and load balancing attributes might need to be further tuned to match the specific characteristics of a particular model and cluster environment.

This leaf and spines nodes in the JVD are configured with CoS parameters that were determined to provide the best performance.

The following configuration is applied uniformly across all devices in the fabric.

Traffic Classification

```
set class-of-service classifiers dscp fabric-dscp forwarding-class CNP loss-priority low code-points 110000
set class-of-service classifiers dscp fabric-dscp forwarding-class NO-LOSS loss-priority low code-points
011010
set class-of-service interfaces et-* unit * classifiers dscp fabric-dscp
```

Traffic classification is based on DSCP and implemented using the **fabric-dscp** classifier, which defines two forwarding classes: ***NO-LOSS*** and ***CNP***. This classifier is applied to all et-* unit * logical interfaces.

All incoming traffic with DSCP 011010 (26) is classified as **NO-LOSS**, while traffic marked with DSCP 110000 (48) is classified as **CNP**. All GPU servers are configured to mark **RoCEv2 traffic** with DSCP 26 and **Congestion Notification Packets (CNPs)** with DSCP 48.

NOTE: Refer to Configuring NVIDIA DCQCN – ECN section of the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and WEKA Storage JVD and the Configuring AMD DCQCN – ECN of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage JVD for details on how to configure DCQCN parameters on the Nvidia and AMD GPU servers.

```
set class-of-service forwarding-classes class CNP queue-num 3
set class-of-service forwarding-classes class NO-LOSS queue-num 4
set class-of-service forwarding-classes class NO-LOSS no-loss
set class-of-service forwarding-classes class NO-LOSS pfc-priority 3
```

CNP traffic is assigned to output queue 3, while *NO-LOSS* traffic is assigned to output **queue 4**.

Queue 4 is configured as a lossless using the **no-loss** attribute, and it mapped to PFC priority 3. Defining a queue as **lossless** ensures that packets mapped to this class are not dropped due to congestion, an essential requirement for **RoCEv2**. Configuring a forwarding class as lossless also impacts buffer allocation on the switch, reserving additional space to support flow control mechanisms such as PFC.

There are two types of buffers:

- Shared Buffer Pool: A global memory space dynamically shared by all ports. It is partitioned between lossy and lossless traffic types. Larger shared buffers help absorb traffic bursts.
- Dedicated Buffer Pool: A reserved portion of memory allocated per port which is then divided among the queues on that port. Though it can be tuned a minimum amount is always reserved by the system. A larger dedicated buffer pool means congestion on one port is less likely to affect traffic on another port because the traffic does not need to use as much shared buffer space. The larger the dedicated buffer pool, the less bursty traffic the switch can handle because there is less dynamic shared buffer memory.

The recommended values for the Shared and Dedicated Buffers in this JVD are as follows:

```
set class-of-service shared-buffer ingress buffer-partition lossless percent 66
set class-of-service shared-buffer ingress buffer-partition lossless dynamic-threshold 10set class-of-
service shared-buffer ingress buffer-partition lossless-headroom percent 24
set class-of-service shared-buffer ingress buffer-partition lossy percent 10
set class-of-service shared-buffer egress buffer-partition lossless percent 66
set class-of-service shared-buffer egress buffer-partition lossy percent 10
set class-of-service dedicated-buffer egress percent 30
set class-of-service dedicated-buffer ingress percent 15
```

Shared buffers:

- **Ingress lossless percent 66:** Reserves **66%** of the ingress shared buffer space for **lossless traffic** (e.g., RoCEv2).
- **Ingress lossless-headroom percent 24:** Carves out an additional **24%** of ingress buffer space specifically as **headroom** for burst absorption. This ensures that RoCEv2 flows have sufficient space to accommodate microbursts while waiting for PFC pause frames to take effect.
- **Ingress lossy percent 10:** Reserves **10%** of ingress shared buffer space for **lossy traffic**.

- **Ingress lossless dynamic-threshold 10:** Allows the lossless buffer pool to **dynamically expand** into unused lossy buffer space by up to **10%**, providing flexibility under heavy load./
- **Egress lossless percent 66:** Reserves **66%** of egress shared buffer space for **lossless** traffic.
- **Egress lossy percent 10:** Allocates **10%** for **lossy** traffic.

Dedicated Buffers (per-port or per-queue):

- Ingress percent 15: Allocates **15%** of the total ingress buffer capacity as **dedicated** buffers. These are not shared and are reserved for specific traffic classes or ports.
- Egress percent 30: Reserves **30%** of egress buffer space for dedicated use.

When this buffer space begins to fill, the PFC mechanism sends Ethernet PAUSE frames to the traffic source instructing it to temporarily halt transmission and prevent packet loss.

Since traffic classification is DSCP-based and interfaces between GPU servers and leaf nodes are **untagged**, the PFC implementation is DSCP based PFC. The congestion-notification-profile pfc, which is applied to all et-* interfaces, defines operation details for PFC.

```
set class-of-service interfaces et-* congestion-notification-profile pfc
set class-of-service congestion-notification-profile pfc pfc-watchdog
set class-of-service congestion-notification-profile pfc input dscp code-point 011010 pfc
set class-of-service congestion-notification-profile pfc output ieee-802.1 code-point 011 flow-control-queue
4
```

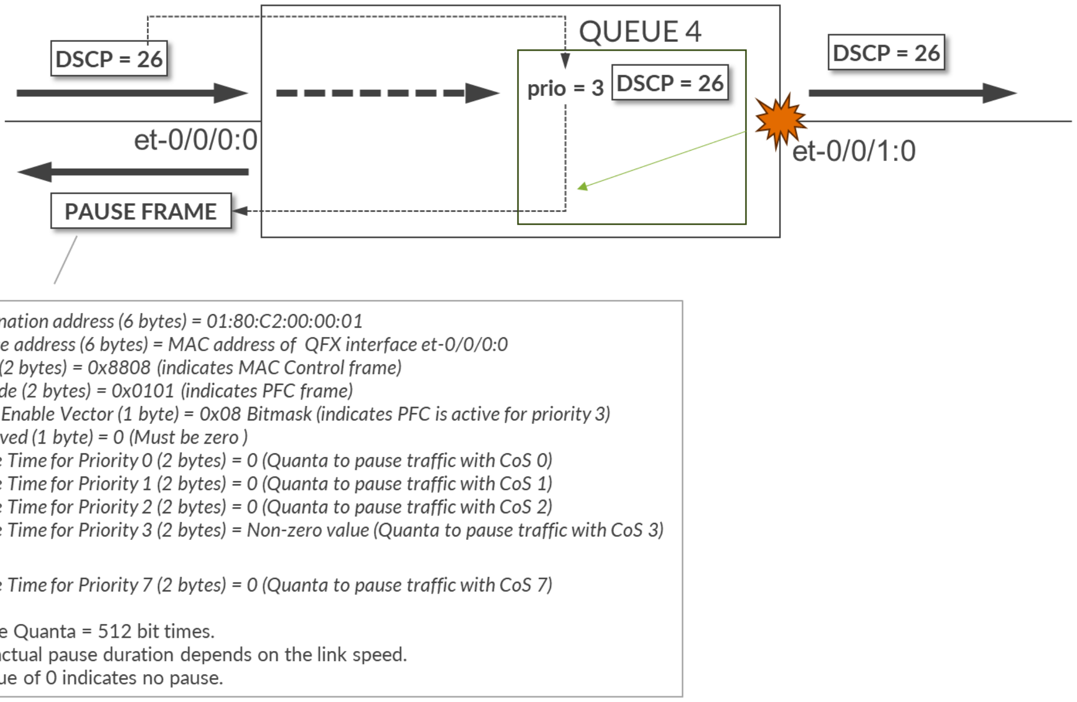
NOTE: The congestion-notification-profile might be interpreted as related to Congestion Notification Packets (ECN). congestion-notification-profile can also be found abbreviated as CNP in some documentation. However, this profile defines the behavior of PFC, not ECN.

The PFC watchdog function monitors for deadlock or stuck queues caused by persistent PFC pause conditions. If a queue remains paused for too long (indicating possible head-of-line blocking), the watchdog can take corrective actions to avoid traffic stall conditions.

The **input dscp code-point 011010 pfc** statement specifies that incoming traffic marked with DSCP value 011010 (decimal 26) should trigger PFC when congestion is detected. Essentially, if DSCP 26 (RoCEv2) traffic is experiencing congestion, PFC frames for priority 3 will be generated to pause upstream senders (PFC priority 3 mapped to code point 26). The pause frames will be generated for a priority 3 based on the forwarding-class NO-LOSS configuration previously described.

In the example below:

Figure 52: PFC Pause Frames Generation Example



Destination address (6 bytes) = 01:80:C2:00:00:01
Source address (6 bytes) = MAC address of QFX interface et-0/0/0:0
Type (2 bytes) = 0x8808 (indicates MAC Control frame)
Opcode (2 bytes) = 0x0101 (indicates PFC frame)
Class Enable Vector (1 byte) = 0x08 Bitmask (indicates PFC is active for priority 3)
Reserved (1 byte) = 0 (Must be zero)
Pause Time for Priority 0 (2 bytes) = 0 (Quanta to pause traffic with CoS 0)
Pause Time for Priority 1 (2 bytes) = 0 (Quanta to pause traffic with CoS 1)
Pause Time for Priority 2 (2 bytes) = 0 (Quanta to pause traffic with CoS 2)
Pause Time for Priority 3 (2 bytes) = Non-zero value (Quanta to pause traffic with CoS 3)
.
.
Pause Time for Priority 7 (2 bytes) = 0 (Quanta to pause traffic with CoS 7)

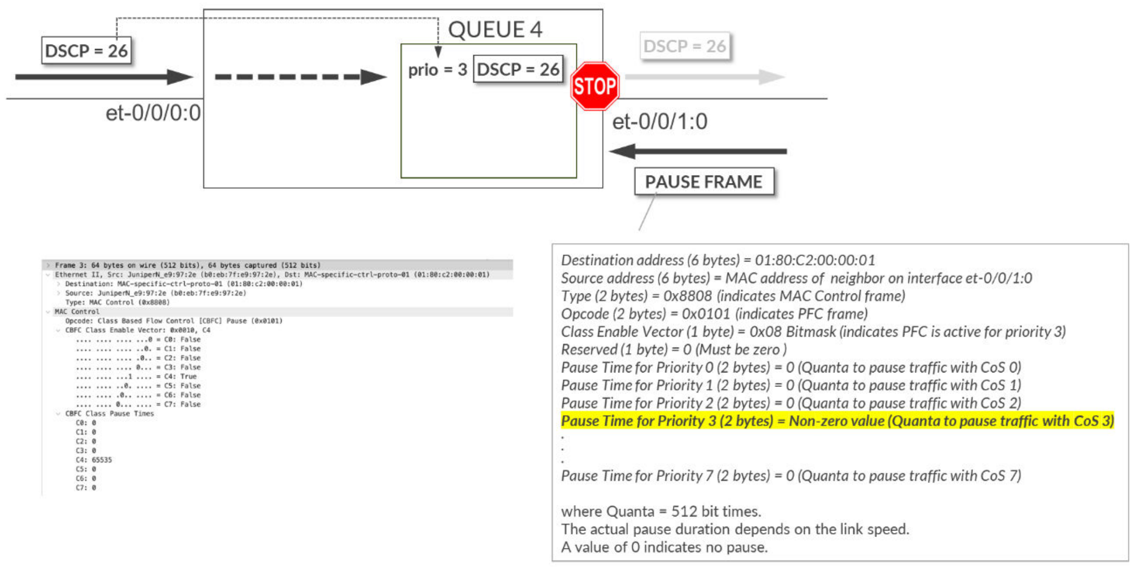
where Quanta = 512 bit times.
The actual pause duration depends on the link speed.
A value of 0 indicates no pause.

The combination of the following commands applied to interfaces et-0/0/0:0 and et-0/0/1:0, configures the device to classify all inbound traffic with DSCP 26 to the forwarding class NO-LOSS which is assigned to Queue 4, and mapped to pfc-priority 3, makes queue 4 a no-loss queue, and enables PFC for traffic with DSCP 26

```
set class-of-service classifiers dscp fabric-dscp forwarding-class NO-LOSS loss-priority low code-points
011010
set class-of-service forwarding-classes class NO-LOSS queue-num 4
set class-of-service forwarding-classes class NO-LOSS no-loss
set class-of-service forwarding-classes class NO-LOSS pfc-priority 3
set class-of-service congestion-notification-profile pfc input dscp code-point 011010 pfc
```

The **output ieee-802.1 code-point 011 flow-control-queue 4** statement specifies that when paused frames with priority 3 are received, traffic for queue 4 must stop.

Figure 53: PFC Received Pause Frames Behavior



```
> Frame 3: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
v Ethernet II, Src: JuniperN_e9:97:2e (b0:eb:7f:e9:97:2e), Dst: MAC-specific-ctrl-proto-01 (01:80:c2:00:00:01)
  > Destination: MAC-specific-ctrl-proto-01 (01:80:c2:00:00:01)
  > Source: JuniperN_e9:97:2e (b0:eb:7f:e9:97:2e)
    Type: MAC Control (0x8808)
v MAC Control
    Opcode: Class Based Flow Control [CBFC] Pause (0x0101)
  v CBFC Class Enable Vector: 0x0010, C4
      .... .... .... ...0 = C0: False
      .... .... .... ..0. = C1: False
      .... .... .... .0.. = C2: False
      .... .... .... 0... = C3: False
      .... .... ...1 .... = C4: True
      .... .... ..0. .... = C5: False
      .... .... .0.. .... = C6: False
      .... .... 0... .... = C7: False
  v CBFC Class Pause Times
      C0: 0
      C1: 0
      C2: 0
      C3: 0
      C4: 65535
      C5: 0
      C6: 0
      C7: 0
```

Destination address (6 bytes) = 01:80:C2:00:00:01
Source address (6 bytes) = MAC address of neighbor on interface et-0/0/1:0
Type (2 bytes) = 0x8808 (indicates MAC Control frame)
Opcode (2 bytes) = 0x0101 (indicates PFC frame)
Class Enable Vector (1 byte) = 0x08 Bitmask (indicates PFC is active for priority 3)
Reserved (1 byte) = 0 (Must be zero)
Pause Time for Priority 0 (2 bytes) = 0 (Quanta to pause traffic with CoS 0)
Pause Time for Priority 1 (2 bytes) = 0 (Quanta to pause traffic with CoS 1)
Pause Time for Priority 2 (2 bytes) = 0 (Quanta to pause traffic with CoS 2)
Pause Time for Priority 3 (2 bytes) = Non-zero value (Quanta to pause traffic with CoS 3)
.
.
.
Pause Time for Priority 7 (2 bytes) = 0 (Quanta to pause traffic with CoS 7)

where Quanta = 512 bit times.
The actual pause duration depends on the link speed.
A value of 0 indicates no pause.

Traffic Scheduling

```
set class-of-service interfaces et-* scheduler-map sm1
set class-of-service scheduler-maps sm1 forwarding-class CNP scheduler s2-cnp
set class-of-service scheduler-maps sm1 forwarding-class NO-LOSS scheduler s1
```

The scheduler map sm1 is applied to all et-* interfaces and defines how traffic for each forwarding class is scheduled.

Two schedulers are included:

- s1 for **NO-LOSS traffic** (queue 4)
- s2-cnp for **CNP traffic** (queue 3)

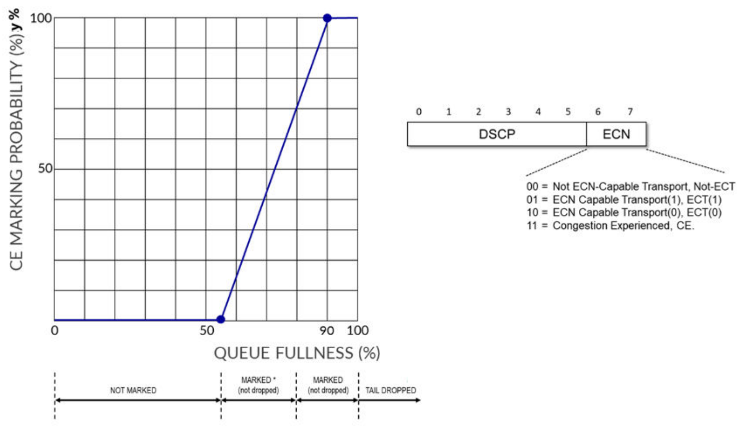
NO-LOSS Traffic Scheduling (Scheduler s1)

```
set class-of-service schedulers s1 drop-profile-map loss-priority any protocol any drop-profile dp1
set class-of-service schedulers s1 explicit-congestion-notification
set class-of-service drop-profiles dp1 interpolate fill-level 55
set class-of-service drop-profiles dp1 interpolate fill-level 90
set class-of-service drop-profiles dp1 interpolate drop-probability 0
set class-of-service drop-profiles dp1 interpolate drop-probability 100
```

Scheduler s1 controls how traffic in the ***NO-LOSS* forwarding class (queue 4)** is serviced. It applies the drop-profile **dp1** and enables **Explicit Congestion Notification (ECN)** marking using the explicit-congestion-notification statement.

NOTE: Drop profiles in Junos are commonly used to control how aggressively packets are dropped as the queue buffer fills up. However, when ECN is enabled, the profile is used to mark packets instead of dropping them. Marking packets means setting the Congestion Experienced (CE) bit in the IP header based on the configured thresholds.

Figure 54: ECN Profile Example



The profile dp1 defines a linear drop curve where:

- At **55% buffer fill**, packets are **not marked** (0% probability).
- At **90% buffer fill, all matching packets are marked** (100% probability).
- Between 55% and 90%, the **marking probability increases linearly** from 0% to 100%.

This approach ensures early congestion feedback to RoCEv2 endpoints while maintaining lossless delivery.

CNP Traffic Scheduling (Scheduler s2-cnp)

Scheduler s2-cnp specifies how *CNP* traffic in queue 3 is serviced. It assigns the queue **strict-high priority** and reserves **5%** of the interface's bandwidth:

```
set class-of-service schedulers s2-cnp

transmit-rate percent
5
set class-of-service schedulers s2-cnp

priority strict-high
```

Assigning **strict-high priority** along with a minimum bandwidth ensures that, during congestion, the **Congestion Notification Packets (CNPs)** required to trigger source-based rate reduction in **DCQCN** can be transmitted across the fabric.

NOTE: Strict-high priority queues are always serviced before any other queues, except for other high-priority queues, which could potentially starve lower-priority traffic. However, the risk of starvation in this case is minimal, because **CNP** traffic is generally very low volume. As a result, there is **no need to rate-limit** this queue.

Congestion Management and Congestion Control Verification

The `show class-of-service interface <interface>` command shows the scheduler-map, whether congestion-notification is enabled and the profile name, as well as the classifier applied to the interface.

```
jnpr@stripe1-leaf2> show class-of-service interface et-0/0/0:0
Physical interface: et-0/0/0:0, Index: 1292
Maximum usable queues: 12, Queues in use: 5
Exclude aggregate overhead bytes: disabled
Logical interface aggregate statistics: disabled
  Scheduler map: sm1
  Congestion-notification: Enabled, Name: cnp, Index: 1
  Logical interface: et-0/0/0:0.0, Index: 1256
Object                 Name                 Type                    Index
Classifier             fabric-dscp          dscp                        5
```

The `show class-of-service classifier <classifier-name>` command shows the mapping between DSCP values and forwarding classes and can be used to confirm correct assignments (CNP => 48, and NO-LOSS => 26)

```
jnpr@stripe1-leaf2> show class-of-service classifier name fabric-dscp
Classifier: fabric-dscp, Code point type: dscp, Index: 5
  Code point         Forwarding class                    Loss priority
  011010             NO-LOSS                             low
  110000             CNP                                 low
```

The `show class-of-service forwarding-class` command output shows the forwarding-classes to queue mapping. Can be used to confirm correct mapping (CNP => queue 3, and NO-LOSS => queue 4), as well and the No-loss status and PFC priority of the NO-LOSS queue.

```
jnpr@stripe1-leaf2> show class-of-service forwarding-class
Forwarding class                       ID      Queue  Policing priority  No-Loss   PFC
priority
  CNP                                  1         3        normal         disabled      0
  NO-LOSS                              2         4        normal         enabled       3
  best-effort                          0         0        normal         disabled      0
  mcast                                8         8        normal         disabled      0
  network-control                      3         7        normal         disabled      0
```

The `show class-of-service scheduler-map sm1` command output shows the **scheduler map sm1** and the **schedulers s1**, and **s2-cnp**, including their priority, assigned rate, and whether ECN is enabled.

```
jnpr@spine1> show class-of-service scheduler-map sm1
   Scheduler map: sm1, Index: 2
    Scheduler: s2-cnp, Forwarding class: CNP, Index: 7
    Transmit rate: 5 percent, Rate Limit: none, Buffer size: unspecified, Buffer Limit: none,
Buffer dynamic threshold:
    unspecified,
    Priority: strict-high
    Excess Priority: unspecified, Excess rate: unspecified, Explicit Congestion Notification:
disable, ECN pfc no assist:
    disable
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             any             0    default-drop-profile
      Medium high     any             0    default-drop-profile
      High            any             0    default-drop-profile
```

```
  Scheduler: s1, Forwarding class: NO-LOSS, Index: 6
    Transmit rate: unspecified, Rate Limit: none, Buffer size: unspecified, Buffer Limit: none, 
Buffer dynamic threshold: 
    unspecified,
    Priority: low
    Excess Priority: unspecified, Excess rate: unspecified, Explicit Congestion Notification: 
enable, ECN pfc no assist: mark
    Drop profiles:
      Loss priority   Protocol    Index    Name
      Low             any             0    dp1
      Medium high     any             0    dp1
      High            any             0    dp1
```

The `show interfaces queue <interface>` command combined with different options and output filter can help determine if there have been any packet drops, ECN marking, and PFC Pause frames.

```
jnpr@stripe1-leaf2> show interfaces queue et-0/0/0:0 forwarding-class CNP
Physical interface: et-0/0/0:0, up, Physical link is Up
  Interface index: 1292, SNMP ifIndex: 703
   Description: facing_spine1:et-0/0/1:0
Forwarding classes: 12 supported, 5 in use
Egress queues: 12 supported, 5 in use
Queue: 3, Forwarding classes: CNP
  Queued:
   Packets              :       0       0 pps
   Bytes                :       0       0 bps
  Transmitted:
    Packets              :      0       0 pps
    Bytes                :      0       0 bps
    Tail-dropped packets :      0       0 pps
    Tail-dropped bytes   :      0       0 bps
    RED-dropped packets  :      0       0 pps
    RED-dropped bytes    :      0       0 bps
    ECN-CE packets       :      0       0 pps
    ECN-CE bytes         :      0       0 bps
```

The output shows the number of CNP packets (DSCP 48) that have been queued. Increments in this value indicate congestion has been detected along the path and the receiver is sending CNP packets in response to packets with CE = 1.

```
jnpr@stripe1-leaf2> show interfaces queue et-0/0/0:0 forwarding-class NO-LOSS
Physical interface: et-0/0/0:0, up, Physical link is Up
  Interface index: 1292, SNMP ifIndex: 703
  Description: facing_spine1:et-0/0/1:0
Forwarding classes: 12 supported, 5 in use
Egress queues: 12 supported, 5 in use
Queue: 4, Forwarding classes: NO-LOSS
  Queued:
    Packets              :            1375227202        0 pps
    Bytes                :         4236817861328        0 bps
  Transmitted:
    Packets              :            1375227202        0 pps
    Bytes                :         4236817861328        0 bps
    Tail-dropped packets :                     0        0 pps
    Tail-dropped bytes   :                     0                0 bps
    RED-dropped packets  :                     0        0 pps
    RED-dropped bytes    :                     0                0 bps
    ECN-CE packets       :                     0                0 pps
                 ECN-CE bytes         :                     0                0 bps
```

The output shows the number of NO-LOSS packets (DSCP = 26) marked with CE=1. If this number is increasing that is an indication that congestion has been detected.

```
jnpr@stripe1-leaf2> show interfaces et-0/0/0:0 extensive | match ecn
    Resource errors: 0, ECN Marked packets: 0
```

The output shows the number of packets marked with CE=1 that have been seen on interface et-0/0/0:0.

```
jnpr@stripe1-leaf2> show interfaces et-0/0/0:0 extensive | find "MAC Priority Flow Control
Statistics"
  MAC Priority Flow Control Statistics:
    Priority :  0        0                0
    Priority :  1        0                0
    Priority :  2        0                0
    Priority :  3        0                0
    Priority :  4       0               0
    Priority :  5       0               0
    Priority :  6       0               0
    Priority :  7       0               0
```

The output shows the number of PFC pause frames that have been sent/received per priority on interface et-0/0/0:0.

```
jnpr@stripe1-leaf2> show interfaces et-0/0/0:0 extensive | find "  CoS information:"
  CoS information:
    Direction : Output
    CoS transmit queue
                Bandwidth               Buffer          Priority        Limit
                %       bps            %       usec
      3 CNP      5      20000000000    r         0       strict-high     none
                    4 NO-LOSS  r      r             r         0       low             none
```

The output shows bandwidth allocation, transmit rate, and queue priority for the forwarding classes CNP, and NO-LOSS on interface et-0/0/0:0.

```
jnpr@stripe1-leaf2> show interfaces queue buffer-occupancy et-0/0/0:0
Physical interface: et-0/0/0:0, Enabled, Physical link is Up
  Interface index: 1292, SNMP ifIndex: 703
Forwarding classes: 12 supported, 5 in use
Egress queues: 12 supported, 5 in use
            Queue: 0, Forwarding classes: best-effort
                Queue-depth bytes  :
                Peak               : 0
            Queue: 3, Forwarding classes: CNP
                Queue-depth bytes  :
                Peak               : 0
            Queue: 4, Forwarding classes: NO-LOSS
                Queue-depth bytes  :
                Peak               : 0
            Queue: 7, Forwarding classes: network-control
                Queue-depth bytes  :
                Peak               : 254
            Queue: 8, Forwarding classes: mcast
```

```
                  Queue-depth bytes  :
                  Peak               : 0
```

The output shows peak queue occupancy for each queue on interface et-0/0/0:0.

```
jnpr@stripe1-leaf2> show class-of-service shared-buffer
Ingress:
  Total Buffer     :  169207 KB
  Dedicated Buffer :  4627 KB
  Shared Buffer    :  143472 KB
    Lossless                    :  94691 KB
    Lossless Headroom           :  34432 KB
    Lossy                       :  14347 KB
    Lossless dynamic threshold :  10
    Lossy dynamic threshold    :  10
  Lossless Headroom Utilization:
  Node Device         Total          Used                 Free
  0                   34432 KB       29235 KB             5197 KB
  ITM0 Headroom Utilization:
  Total           Used            Free
  17216 KB        15260 KB        1956 KB
  ITM1 Headroom Utilization:
  Total           Used            Free
  17216 KB        13975 KB        3241 KB
Egress:
  Total Buffer     :  169207 KB
  Dedicated Buffer :  14162 KB
  Shared Buffer    :  143472 KB
    Lossless                  :  94691 KB
    Lossy                     :  14347 KB
    Lossy dynamic threshold  :  7
```

The output shows systems buffer allocations.

NOTE: Juniper ITM (Ingress Traffic Manager) is a component that manages packet buffering and queues.

Load Balancing Configuration

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.

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.

In this JVD, both the leaf and spine nodes are configured to Load Balance traffic using Dynamic Load Balancing flowlet-mode, applied to both IPv4 and IPv6 traffic.

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

The following example shows the configuration applied on all devices:

```
jnpr@gpu-backend-rack1-001-leaf2> show configuration forwarding-options | display set
set forwarding-options hash-key family inet layer-3
set forwarding-options hash-key family inet layer-4
set forwarding-options enhanced-hash-key ecmp-dlb flowlet inactivity-interval 128
set forwarding-options enhanced-hash-key ecmp-dlb flowlet flowset-table-size 2048
set forwarding-options enhanced-hash-key ecmp-dlb ether-type ipv4
set forwarding-options enhanced-hash-key ecmp-dlb ether-type IPv6
set forwarding-options enhanced-hash-key ecmp-dlb sampling-rate 1000000
```

This configuration defines how flows are identified and the conditions for reassigning them to alternate ECMP paths based on real-time congestion and flow characteristics.

The **`hash-key family inet layer-3`** and **`hash-key family inet layer-4`** statements configure the ECMP hashing function to include both IP addresses and TCP/UDP ports, ensuring granular distribution of IPv4 flows across ECMP paths.

The parameters under **enhanced-hash-key** modify the DLB hashing algorithm for ECMP traffic forwarding, enabling flowlet-based detection and intelligent reassignment. These include:

- **`ecmp-dlb flowlet inactivity-interval`**

Specifies the minimum inter-packet gap (in microseconds) used to detect the boundary between flowlets. A new flowlet is recognized when this threshold is exceeded.

The recommended value is 128 µsec.

- **`ecmp-dlb flowset-table-size`**

Defines the maximum number of flowset (macroflow) entries that can be stored in the DLB hash table. This controls how many active flows the device can track for dynamic reassignment. This value must be a multiple of 8.

The recommended value is 2048.

- **`sampling rate`** :

Defines the sampling rate to detect congestion by configuring the QFX forwarding ASIC to sample the port load on the egress ECMP members, and update quality scores.

The recommended value is 1,000,000, which means 1 in every million packets is sampled, balancing overhead and responsiveness.

`ether-type ipv4` and `ether-type IPv6`:

Enable enhanced ECMP DLB for both **IPv4** and **IPv6** packets

Load Balancing Verification

To verify the DLB parameters currently in use, you can use the operational command: `show forwarding-options enhanced-hash-key` . The output shows the values applied by the system for ECMP Dynamic Load Balancing (DLB), including flowlet behavior.

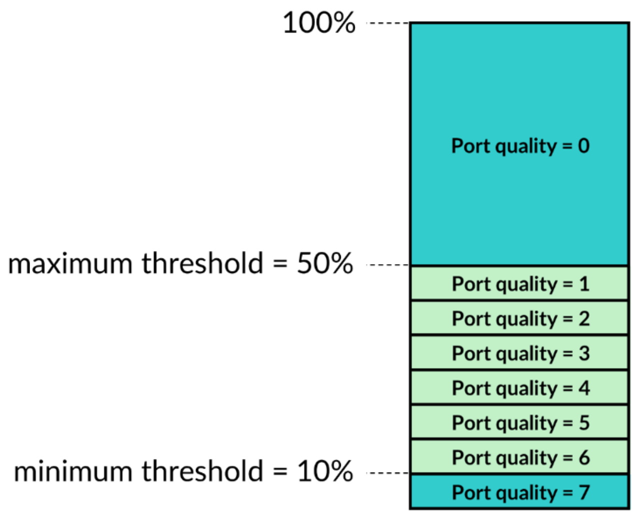
```
jnpr@stripe1-leaf1> show forwarding-options enhanced-hash-key
Current RTAG7 Settings
-------------------------
   Hash-Mode                  :layer2-payload
   Hash-Seed                  :112443776
inet  RTAG7 settings:
----------------------
inet packet fields
  protocol                   :yes
  Destination IPv4 Addr      :yes
  Source IPv4 Addr           :yes
  destination L4 Port        :yes
  Source L4 Port             :yes
  Vlan id                    :no
  RDMA Queue Pair            :yes
inet non-packet fields
  incoming port              :yes
inet6  RTAG7 settings:
----------------------
inet6 packet fields
  next-header                :yes
  Destination IPv6 Addr      :yes
  Source IPv6 Addr           :yes
  destination L4 Port        :yes
  Source L4 Port             :yes
  Vlan id                    :no
  RDMA Queue Pair            :yes
inet6 non-packet fields
  incoming port              :yes
Hash-Parameter Settings for ECMP:
------------------------------------
  Hash Function   = CRC16_BISYNC
  Hash offset base = 16
  Hash offset     = 5
  Hash preprocess = 0
```

```
Hash-Parameter Settings for LAG:
-----------------------------------
  Hash Function    = CRC16_CCITT
  Hash offset base = 0
  Hash offset      = 5
  Hash preprocess = 0
Ecmp Resilient Hash   = Disabled
ECMP DLB Load Balancing Options:
---------------------------------------------------
  Load Balancing Method              : Flowlet
  Inactivity Interval                : 128 (us)
               Flowset Table size                 : 2048 (entries per ECMP)
  Reassignment Probability Threshold : 0
  Reassignment Quality Delta         : 0
  Egress Port Load Weight            : 50
               EgressBytes Min Threshold          : 10
               EgressBytes Max Threshold          : 50
              Sampling Rate                      : 1000000
               Ether Type                         : IPv4 IPv6
```

The **Egress Port Load Weight** shown in the output defines the weights given to port load and port queue length when calculating the port quality score. The **EgressBytes Min** and **EgressBytes Max Thresholds** define quality bands. DLB assigns any egress port with a port load falling below this minimum to the highest quality band (7). Any port load larger than the maximum threshold falls into the lowest quality band (0). DLB divides the remaining port load quantities among quality bands 1 through 6.

We recommend maintaining the default values, **Egress Port Load Weight** (50) **EgressBytes Min** Threshold (10) and **EgressBytes Max Thresholds (50).** No configuration is needed to use these values.

Figure 55: DLB quality bands.



Servers and Storage Configuration

IN THIS SECTION

NVIDIA Configuration

For details about how to connect and configure the NVIDIA GPU servers, including Nvidia CX7 NICs, refer to the NVIDIA Configuration section of the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and Weka Storage—Juniper Validated Design (JVD).

AMD Configuration

For details about how to connect and configure the AMD GPU servers, including Broadcom Thor 2 NICs, and AMD Pensando™ Pollara 400 NICs, refer to the AMD Configuration section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage—Juniper Validated Design (JVD).

Weka Storage Configuration

For details about how to connect and configure the Weka Storage devices, refer to the Weka Storage Solution section of the AI Data Center Network with Juniper Apstra, NVIDIA GPUs, and Weka Storage—Juniper Validated Design (JVD). document.

VAST Storage Configuration

For details about how to connect and configure the Vast Storage devices, refer to the Vast Storage Configuration section of the AI Data Center Network with Juniper Apstra, AMD GPUs, and Vast Storage —Juniper Validated Design (JVD).

Fabric Devices Configuration

The configurations files for the QFX devices that were used for validating the features in this solution are posted in the following github repository:

https://github.com/Juniper/jvd/tree/main/Data%20Center/AIDC/backend/AI_ML_Multitenancy

Full configuration example (Stripe 1 Leaf 1):

-
```
set routing-options router-id 10.0.1.1

set routing-options autonomous-system 201

set routing-options graceful-restart

set routing-options forwarding-table export PFE-LB
set routing-options forwarding-table ecmp-fast-reroute

# LOAD BALANCING

set forwarding-options hash-key family inet layer-3

set forwarding-options hash-key family inet layer-4

set forwarding-options enhanced-hash-key ecmp-dlb flowlet inactivity-interval 256

set forwarding-options enhanced-hash-key ecmp-dlb flowlet flowset-table-size 2048

set forwarding-options enhanced-hash-key ecmp-dlb ether-type ipv4

set forwarding-options enhanced-hash-key ecmp-dlb sampling-rate 1000000

# CLASS OF SERVICE

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 drop-profiles dp1 interpolate fill-level 55

set class-of-service drop-profiles dp1 interpolate fill-level 90

set class-of-service drop-profiles dp1 interpolate drop-probability 0

set class-of-service drop-profiles dp1 interpolate drop-probability 100

set class-of-service shared-buffer ingress buffer-partition lossless percent 80

set class-of-service shared-buffer ingress buffer-partition lossless-headroom percent 10

set class-of-service shared-buffer ingress buffer-partition lossy percent 10

set class-of-service shared-buffer egress buffer-partition lossless percent 80

set class-of-service shared-buffer egress buffer-partition lossy percent 10

set class-of-service forwarding-classes class CNP queue-num 3
set class-of-service forwarding-classes class NO-LOSS queue-num 4

set class-of-service forwarding-classes class NO-LOSS no-loss

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

set class-of-service congestion-notification-profile cnp input dscp code-point 011010 pfc

set class-of-service congestion-notification-profile cnp output ieee-802.1 code-point 011
flow-control-queue 4

set class-of-service interfaces et-* congestion-notification-profile cnp

set class-of-service interfaces et-* scheduler-map sm1

set class-of-service interfaces et-* unit * classifiers dscp mydscp

set class-of-service scheduler-maps sm1 forwarding-class CNP scheduler s2-cnp

set class-of-service scheduler-maps sm1 forwarding-class NO-LOSS scheduler s1

set class-of-service schedulers s1 drop-profile-map loss-priority any protocol any drop-
profile dp1

set class-of-service schedulers s1 explicit-congestion-notification

set class-of-service schedulers s2-cnp transmit-rate percent 5

set class-of-service schedulers s2-cnp priority strict-high

# BGP UNDERLAY

set protocols bgp group l3clos-inet6-auto-underlay family inet6 unicast

set protocols bgp group l3clos-inet6-auto-underlay export ( LEAF_TO_SPINE_FABRIC_OUT && BGP-
AOS-Policy )

set protocols bgp group l3clos-inet6-auto-underlay local-as 201

set protocols bgp group l3clos-inet6-auto-underlay multipath multiple-as

set protocols bgp group l3clos-inet6-auto-underlay bfd-liveness-detection minimum-interval
```

```
3000

set protocols bgp group l3clos-inet6-auto-underlay bfd-liveness-detection multiplier 3

set protocols bgp group l3clos-inet6-auto-underlay dynamic-neighbor UNDERLAY peer-auto-
discovery family inet6 ipv6-nd

/* AUTODISCOVERED PEER SPINE 1 */

set protocols bgp group l3clos-inet6-auto-underlay dynamic-neighbor UNDERLAY peer-auto-
discovery interface et-0/0/30:0.0

/* AUTODISCOVERED PEER SPINE 2 */

set protocols bgp group l3clos-inet6-auto-underlay dynamic-neighbor UNDERLAY peer-auto-
discovery interface et-0/0/31:0.0

/* AUTODISCOVERED PEER SPINE 3 */

set protocols bgp group l3clos-inet6-auto-underlay dynamic-neighbor UNDERLAY peer-auto-
discovery interface et-0/0/32:0.0

/* AUTODISCOVERED PEER SPINE 4 */

set protocols bgp group l3clos-inet6-auto-underlay dynamic-neighbor UNDERLAY peer-auto-
discovery interface et-0/0/33:0.0

set protocols bgp group l3clos-inet6-auto-underlay peer-as-list discovered-as-list

set policy-options as-list discovered-as-list members 101-104

# BGP OVERLAY

set protocols bgp group l3clos-inet6-auto-overlay type external

set protocols bgp group l3clos-inet6-auto-overlay multihop ttl 1

set protocols bgp group l3clos-inet6-auto-overlay family route-target

set protocols bgp group l3clos-inet6-auto-overlay multipath multiple-as

set protocols bgp group l3clos-inet6-auto-overlay bfd-liveness-detection minimum-interval 3000
set protocols bgp group l3clos-inet6-auto-overlay bfd-liveness-detection multiplier 3

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::1 description
facing_spine1-evpn-overlay

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::1 local-address
fc00:10:0:1::1

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::1 family evpn signaling

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::1 export
( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT )

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::1 peer-as 101

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::2 description
facing_spine2-evpn-overlay

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::2 local-address
fc00:10:0:1::1

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::2 family evpn signaling

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::2 export
( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT )

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::2 peer-as 102

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::3 description
facing_spine3-evpn-overlay

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::3 local-address
fc00:10:0:1::1

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::3 family evpn signaling

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::3 export
( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT )

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::3 peer-as 103

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::4 description
facing_spine4-evpn-overlay
```

```
set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::4 local-address
fc00:10:0:1::1

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::4 family evpn signaling

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::4 export
( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT )

set protocols bgp group l3clos-inet6-auto-overlay neighbor fc00:10::4 peer-as 104

set protocols bgp group l3clos-inet6-auto-overlay vpn-apply-export

/* ROUTER ADVERTISEMENTS TO SPINE1 */

set protocols router-advertisement interface et-0/0/30:0.0 retransmit-timer 10000

/* ROUTER ADVERTISEMENTS TO SPINE2 */

set protocols router-advertisement interface et-0/0/31:0.0 retransmit-timer 10000

/* ROUTER ADVERTISEMENTS TO SPINE3 */

set protocols router-advertisement interface et-0/0/32:0.0 retransmit-timer 10000

/* ROUTER ADVERTISEMENTS TO SPINE4 */

set protocols router-advertisement interface et-0/0/33:0.0 retransmit-timer 10000

/* ROUTER ADVERTISEMENTS TO SERVER 1 GPU0 */

set protocols router-advertisement interface et-0/0/0:0.0 retransmit-timer 10000

set protocols router-advertisement interface et-0/0/0:0.0 prefix fc00:200:1:1::/64

set protocols router-advertisement interface et-0/0/0:0.0 rio-prefix fc00:200:1::/56 rio-
lifetime 1800

/* ROUTER ADVERTISEMENTS TO SERVER 2 GPU0 */

set protocols router-advertisement interface et-0/0/1:0.0 retransmit-timer 10000

set protocols router-advertisement interface et-0/0/1:0.0 prefix fc00:200:1:2::/64
set protocols router-advertisement interface et-0/0/1:0.0 rio-prefix fc00:200:1::/56 rio-
lifetime 1800

/* ROUTER ADVERTISEMENTS TO SERVER 2 GPU0 */

set protocols router-advertisement interface et-0/0/2:0.0 retransmit-timer 10000

set protocols router-advertisement interface et-0/0/2:0.0 prefix fc00:200:1:3::/64

set protocols router-advertisement interface et-0/0/2:0.0 rio-prefix fc00:200:1::/56 rio-
lifetime 1800

# TENANT ROUTING INSTANCES

set routing-instances Tenant-1 instance-type vrf

set routing-instances Tenant-1 routing-options rib Tenant-1.inet6.0 multipath

set routing-instances Tenant-1 routing-options graceful-restart

set routing-instances Tenant-1 routing-options multipath

set routing-instances Tenant-1 protocols evpn ip-prefix-routes advertise direct-nexthop

set routing-instances Tenant-1 protocols evpn ip-prefix-routes encapsulation vxlan

set routing-instances Tenant-1 protocols evpn ip-prefix-routes vni 20001

set routing-instances Tenant-1 protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-1

/* CONNECTION TO SERVER 1 */

set routing-instances Tenant-1 interface et-0/0/0:0.0

/* CONNECTION TO SERVER 2 */

set routing-instances Tenant-1 interface et-0/0/1:0.0

/* CONNECTION TO SERVER 3 */

set routing-instances Tenant-1 interface et-0/0/2:0.0
set routing-instances Tenant-1 interface lo0.1

set routing-instances Tenant-1 route-distinguisher 10.0.1.1:2001

set routing-instances Tenant-1 vrf-target target:20001:1

# ROUTING POLICIES

set policy-options policy-statement AllPodNetworks term AllPodNetworks-10 from family inet6

set policy-options policy-statement AllPodNetworks term AllPodNetworks-10 from protocol direct

set policy-options policy-statement AllPodNetworks term AllPodNetworks-10 from interface lo0.0

set policy-options policy-statement AllPodNetworks term AllPodNetworks-10 then community add
DEFAULT_DIRECT_V6

set policy-options policy-statement AllPodNetworks term AllPodNetworks-10 then accept

set policy-options policy-statement AllPodNetworks term AllPodNetworks-100 then reject

set policy-options policy-statement AllPodNetworks-Tenant-1 term AllPodNetworks-Tenant-1-10
from family inet6

set policy-options policy-statement AllPodNetworks-Tenant-1 term AllPodNetworks-Tenant-1-10
from protocol direct

set policy-options policy-statement AllPodNetworks-Tenant-1 term AllPodNetworks-Tenant-1-10
then community add TENANT-1_COMMUNITY_V6

set policy-options policy-statement AllPodNetworks-Tenant-1 term AllPodNetworks-Tenant-1-10
then accept

set policy-options policy-statement AllPodNetworks-Tenant-1 term AllPodNetworks-Tenant-1-100
then reject

set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-10 from policy
AllPodNetworks

set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-10 then accept

set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-20 from protocol evpn
set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-20 from route-filter
0::0/0 prefix-length-range /128-/128

set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-20 then accept

set policy-options policy-statement BGP-AOS-Policy term BGP-AOS-Policy-100 then reject

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-10
from policy AllPodNetworks-Tenant-1

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-10
then accept

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-20
from protocol evpn

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-20
from route-filter 0::0/0 prefix-length-range /128-/128

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-20
then community add TENANT-1_COMMUNITY_V6

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-20
then accept

set policy-options policy-statement BGP-AOS-Policy-Tenant-1 term BGP-AOS-Policy-Tenant-1-100
then reject

set policy-options policy-statement EVPN_EXPORT term EVPN_EXPORT-4095 then accept

set policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT term LEAF_TO_SPINE_EVPN_OUT-10
from protocol bgp

set policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT term LEAF_TO_SPINE_EVPN_OUT-10
from community FROM_SPINE_EVPN_TIER

set policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT term LEAF_TO_SPINE_EVPN_OUT-10
then reject

set policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT term LEAF_TO_SPINE_EVPN_OUT-20
then accept

set policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT term LEAF_TO_SPINE_FABRIC_OUT-10
```

```
from protocol bgp

set policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT term LEAF_TO_SPINE_FABRIC_OUT-10
from community FROM_SPINE_FABRIC_TIER

set policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT term LEAF_TO_SPINE_FABRIC_OUT-10
then reject

set policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT term LEAF_TO_SPINE_FABRIC_OUT-20
then accept

set policy-options policy-statement PFE-LB term 1 then load-balance per-packet

set policy-options policy-statement direct from protocol direct

set policy-options policy-statement direct then accept

set policy-options community TENANT-1_COMMUNITY_V6 members 5:20008

set policy-options community TENANT-1_COMMUNITY_V6 members 21001:26000

# TELEMETRY

set system services extension-service request-response grpc ssl port 32767

set system services extension-service request-response grpc ssl local-certificate aos_grpc

set system services extension-service request-response grpc max-connections 30

set system services extension-service request-response grpc routing-instance mgmt_junos

set system services extension-service request-response grpc skip-authentication
```

Telemetry and Monitoring

IN THIS SECTION

AI cluster networks demand lossless, high-throughput, and low-latency connectivity. A key component of maintaining performance is the collection and analysis of operational data to monitor congestion, system health, and traffic patterns. Junos OS telemetry enables detailed tracking of critical performance indicators, including thresholds, counters, and congestion metrics specific to AI workloads. Once collected, this data must be analyzed, structured, and visualized to support monitoring, decision-making, and continuous network optimization.

The following sections describe how to configure the devices to enable data collection and outline key performance metrics recommended for the AI EVPN/VXLAN fabric solution.

Configuring QFX Switches to Provide Telemetry Information

To implement telemetry collection the switches need to be configure to allow gPRC-based access as described in the OpenConfig and gRPC for Junos Telemetry Interface section of Junos Telemetry Interface User Guide.

The following configuration was used on all the leaf and spine node devices for this purpose:

```
user@spine1> show configuration system services extension-service
request-response {
    grpc {
        ssl {
            port 32767;
            local-certificate aos_grpc;
        }
        routing-instance mgmt_junos;
    }
}
```

Table 49. gRPC Configuration Commands for Junos OS

Command	Description
`extension-service request-response grpc`	Enables the gRPC interface under the extension service framework, used for APIs like Junos Telemetry Interface (JTI) or third-party integrations. The client issues a request and waits for a response from the Junos OS server.
`ssl port 32767`	Configures TCP port 32767 for communication using SSL encryption.
`local-certificate aos_grpc`	Configures authentication using a certificate named aos_grpc to secure the gRPC session. Follow the steps described in Configure gRPC Services to generate and install the necessary certificates.
`routing-instance mgmt_junos`	Binds the gRPC server to the mgmt_junos routing-instance, meaning it only listens on the out-of-band management interface.

To validate connectivity between the telemetry collector, use the `show system connections` command and search for the ssl port number configured.

```
jnpr@stripe2-leaf1> show system connections | match "Address|32767"
Proto Recv-Q Send-Q Local Address           Foreign Address     State       PID/Program name
tcp6       0      0 :::32767                :::*                LISTEN      11937/jsd
tcp6       0      0 10.161.33.38:32767      10.100.1.17:56634   ESTABLISHED 11937/jsd
tcp6       0      0 10.161.33.38:32767      10.100.1.20:53184   ESTABLISHED 11937/jsd
tcp6       0      0 10.161.33.38:32767      10.100.1.20:53170   ESTABLISHED 11937/jsd
```

The sample output shows connections from two collectors (10.100.1.17 and 10.100.1.20).

To confirm that the collectors are actively pulling data via gRPC/gNMI and see **what sensors are in use,** use:

- `show network-agent statistics`
- `show network-agent statistics detail`
- `show network-agent statistics subscription-paths <sensor-path>`
- `show network-agent statistics juniper`

- `show network-agent statistics gnmi`

Example:

```
jnpr@stripe2-leaf1> show network-agent statistics
Subscription Details :
    Subscription ID                          : 1
    Type                                     : juniper
    Client IP                                : IPv6:::ffff:10.100.1.17:56634
    Subscription Time (UTC)                  : Thu May  1 12:38:57 2025
    Sensor Statistics :
        Sensor Path                          : /network-instances/network-instance/mac-table/
entries/entry/
        Reporting Interval                   : 0
        Component(s)                         : l2aldTM,l2ald
        Child Sensor Statistics :
              Path                           : /network-instances/network-instance/mac-table/
entries/entry/
              Component                      : l2ald
              Component-ID                   : 65535
              Path                           : /network-instances/network-instance/mac-table/
entries/entry/
              Component                      : l2aldTM
              Component-ID                   : 65535
Subscription Details :
    Subscription ID                          : 2
    Type                                     : gnmi
    Client IP                                : IPv6:::ffff:10.100.1.17:56634
    GNMI mode                                : STREAM
    Subscription Time (UTC)                  : Thu May  1 12:38:57 2025
    Sensor Statistics :
        Sensor Path                          : /interfaces/interface/state/admin-status/
        Reporting Interval                   : 120
        Component(s)                         : re0/mib2d
        GNMI Sub Mode                        : SAMPLE
        Component ID                         : 65535
        Sensor Path                          : /interfaces/interface/state/oper-status/
        Reporting Interval                   : 120
        Component(s)                         : re0/mib2d,evo-pfemand
        GNMI Sub Mode                        : SAMPLE
        Child Sensor Statistics :
              Path                           : /interfaces/interface/state/oper-status/
              Component                  : evo-pfemand
              GNMI-SubMode               : SAMPLE
              Component-ID               : 0
              Path                       : /interfaces/interface/state/oper-status/
              Component                  : re0/mib2d
              GNMI-SubMode               : SAMPLE
              Component-ID               : 65535
    Sensor Statistics :
        Sensor Path                      : /interfaces/interface/subinterfaces/subinterface/
state/admin-status/
        Reporting Interval               : 120
        Component(s)                     : re0/mib2d
        GNMI Sub Mode                    : SAMPLE
        Component ID                     : 65535
        Sensor Path                      : /interfaces/interface/subinterfaces/subinterface/
state/oper-status/
        Reporting Interval               : 120
        Component(s)                     : re0/mib2d,evo-pfemand
        GNMI Sub Mode                    : SAMPLE
        Child Sensor Statistics :
              Path                       : /interfaces/interface/subinterfaces/subinterface/
state/oper-status/
              Component                  : evo-pfemand
              GNMI-SubMode               : SAMPLE
              Component-ID               : 0
              Path                       : /interfaces/interface/subinterfaces/subinterface/
state/oper-status/
              Component                  : re0/mib2d
              GNMI-SubMode               : SAMPLE
              Component-ID               : 65535
Subscription Details :
    Subscription ID                      : 3
    Type                                 : juniper
    Client IP                            : IPv6:::ffff:10.100.1.17:56634
    Subscription Time (UTC)              : Thu May  1 12:39:01 2025
    Sensor Statistics :
        Sensor Path                      : /junos/system/linecard/qmon-sw/
        Reporting Interval               : 5
        Component(s)                     : evo-pfemand
        Component ID                     : 0
Subscription Details :
    Subscription ID                      : 4
    Type                                 : gnmi
    Client IP                             : IPv6:::ffff:10.161.38.48:39588
    GNMI mode                             : STREAM
    Subscription Time (UTC)               : Thu May  1 12:39:15 2025
    Sensor Statistics :
        Sensor Path                       : /components/component/cpu/utilization/
        Reporting Interval                : 2
        Component(s)                      : re0/ehmd
        GNMI Sub Mode                     : SAMPLE
        Component ID                      : 65535
Subscription Details :
    Subscription ID                       : 5
    Type                                  : juniper
    Client IP                             : IPv6:::ffff:10.161.38.48:57182
    Subscription Time (UTC)               : Thu May  1 12:39:04 2025
    Sensor Statistics :
        Sensor Path                       : /junos/system/linecard/npu/memory/
        Reporting Interval                : 2
        Component(s)                      : evo-pfemand
        Component ID                      : 0
Subscription Details :
    Subscription ID                       : 6
    Type                                  : juniper
    Client IP                             : IPv6:::ffff:10.161.38.48:57182
    Subscription Time (UTC)               : Thu May  1 12:39:04 2025
    Sensor Statistics :
        Sensor Path                       : /junos/system/linecard/interface/
        Reporting Interval                : 2
        Component(s)                      : picd,evo-pfemand
        Child Sensor Statistics :
              Path                        : /junos/system/linecard/interface/
              Component                   : evo-pfemand
              Component-ID                : 0
              Path                        : /junos/system/linecard/interface/
              Component                   : picd
              Component-ID                : 0
Subscription Details :
    Subscription ID                       : 7
    Type                                  : juniper
    Client IP                             : IPv6:::ffff:10.161.38.48:57182
    Subscription Time (UTC)               : Thu May  1 12:39:04 2025
    Sensor Statistics :
        Sensor Path                       : /junos/system/linecard/qmon-sw/
        Reporting Interval                : 2
        Component(s)                  : evo-pfemand
        Component ID                  : 0
Subscription Details :
    Subscription ID                   : 8
    Type                              : juniper
    Client IP                         : IPv6:::ffff:10.161.38.48:57182
    Subscription Time (UTC)           : Thu May  1 12:39:04 2025
    Sensor Statistics :
        Sensor Path                   : /junos/system/linecard/interface/queue/
        Reporting Interval            : 2
        Component(s)                  : Not available
        Component ID                  : 65535
jnpr@stripe1-leaf1> show network-agent statistics detail
Subscription Details :
    Subscription ID                   : 1
    Type                              : gnmi
    Client IP                         : ipv6:::ffff:10.161.38.194:49526
    GNMI mode                         : STREAM
    Subscription Time (UTC)           : Wed Oct  1 14:39:38 2025
    Sensor Statistics :
        Sensor Path                                 : /components/component/cpu/utilization/
        Reporting Interval                          : 2
        Component(s)                                : re0/ehmd
        GNMI Sub Mode                               : SAMPLE
        Component ID                                : 65535
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 347630
        Packets Sent                                : 648
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 4536
        Initial Sync Packets Sent                   : 8
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 0
        Initial Sync Average Circular Buffer Used (%)  : 0
        Sensor Path                                 : /components/component/state/memory/
utilized/
        Reporting Interval                          : 30
        Component(s)                                : re0/hwdre/stack1,re0/hwdre,picd
        GNMI Sub Mode                               : SAMPLE
        Average iLatency (ms)                       : 11
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 34425
        Packets Sent                                  : 131
        Drops                                         : 0
        Initial Sync Bytes Sent                       : 7482
        Initial Sync Packets Sent                     : 29
        Initial Sync Drops                            : 0
        Initial Sync Average iLatency (ms)            : 11
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                               : /components/component/state/memory/utilized/
              Component                          : picd
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 0
              Average iLatency (ms)              : 13
              Bytes Sent                         : 31154
              Packets Sent                       : 116
              Drops                              : 0
              Initial Sync Bytes Sent            : 7220
              Initial Sync Packets Sent          : 28
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 12
              Path                               : /components/component/state/memory/utilized/
              Component                          : re0/hwdre
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 928
              Packets Sent                       : 5
              Drops                              : 0
              Initial Sync Bytes Sent            : 0
              Initial Sync Packets Sent          : 0
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 0
              Path                               : /components/component/state/memory/utilized/
              Component                          : re0/hwdre/stack1
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 2343
              Packets Sent                       : 10
              Drops                              : 0
              Initial Sync Bytes Sent            : 262
              Initial Sync Packets Sent          : 1
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 0
    Sensor Statistics :
        Sensor Path                                  : /components/component/state/temperature/
avg/
        Reporting Interval                           : 30
        Component(s)                                 : re0/hwdre/stack1,re0/hwdre
        GNMI Sub Mode                                : SAMPLE
        Average iLatency (ms)                        : 1
        Average Circular Buffer Used (%)             : 0
        Bytes Sent                                   : 51773
        Packets Sent                                 : 180
        Drops                                        : 0
        Initial Sync Bytes Sent                      : 9494
        Initial Sync Packets Sent                    : 34
        Initial Sync Drops                           : 0
        Initial Sync Average iLatency (ms)           : 1
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                               : /components/component/state/temperature/avg/
              Component                          : re0/hwdre
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 1
              Bytes Sent                         : 46300
              Packets Sent                       : 160
              Drops                              : 0
              Initial Sync Bytes Sent            : 8634
              Initial Sync Packets Sent          : 31
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 1
              Path                               : /components/component/state/temperature/avg/
              Component                          : re0/hwdre/stack1
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 5473
              Packets Sent                       : 20
              Drops                              : 0
              Initial Sync Bytes Sent            : 860
              Initial Sync Packets Sent          : 3
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 1
    Sensor Statistics :
        Sensor Path                                    : /components/component/state/used-power/
        Reporting Interval                             : 30
        Component(s)                                   : re0/hwdre/stack1,re0/hwdre
        GNMI Sub Mode                                  : SAMPLE
        Average iLatency (ms)                          : 2
        Average Circular Buffer Used (%)               : 0
        Bytes Sent                                     : 15636
        Packets Sent                                   : 65
        Drops                                          : 0
        Initial Sync Bytes Sent                        : 2615
        Initial Sync Packets Sent                      : 11
        Initial Sync Drops                             : 0
        Initial Sync Average iLatency (ms)             : 2
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                               : /components/component/state/used-power/
              Component                          : re0/hwdre
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 2
              Bytes Sent                         : 13453
              Packets Sent                       : 55
              Drops                              : 0
              Initial Sync Bytes Sent            : 2375
              Initial Sync Packets Sent          : 10
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 3
              Path                               : /components/component/state/used-power/
              Component                          : re0/hwdre/stack1
              GNMI-SubMode                       : SAMPLE
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 2183
              Packets Sent                       : 10
              Drops                              : 0
              Initial Sync Bytes Sent            : 240
              Initial Sync Packets Sent          : 1
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 2
Subscription Details :
    Subscription ID                      : 2
    Type                                 : juniper
    Client IP                            : ipv6:::ffff:10.161.38.194:45730
```

```
      Subscription Time (UTC)              : Wed Oct  1 14:40:14 2025
      Sensor Statistics :
          Sensor Path                                   : /junos/system/linecard/npu/memory/
          Reporting Interval                            : 2
          Component(s)                                  : evo-pfemand
          Component ID                                  : 0
          Average iLatency (ms)                         : 0
          Average Circular Buffer Used (%)              : 0
          Bytes Sent                                    : 532438
          Packets Sent                                  : 54
          Drops                                         : 0
          Initial Sync Bytes Sent                       : 0
          Initial Sync Packets Sent                     : 0
          Initial Sync Drops                            : 0
          Initial Sync Average iLatency (ms)            : 0
          Initial Sync Average Circular Buffer Used (%)  : 0
  Subscription Details :
      Subscription ID                      : 3
      Type                                 : juniper
      Client IP                            : ipv6:::ffff:10.161.38.194:45730
      Subscription Time (UTC)              : Wed Oct  1 14:40:14 2025
      Sensor Statistics :
          Sensor Path                                   : /interfaces/interface/state/
          Reporting Interval                            : 2
          Component(s)                                  : re0/mgmt-ethd,re0/mib2d,picd,evo-pfemand
          Average iLatency (ms)                         : 0
          Average Circular Buffer Used (%)              : 0
          Bytes Sent                                    : 3748322
          Packets Sent                                  : 329
          Drops                                         : 0
          Initial Sync Bytes Sent                       : 0
          Initial Sync Packets Sent                     : 0
          Initial Sync Drops                            : 0
          Initial Sync Average iLatency (ms)            : 0
          Initial Sync Average Circular Buffer Used (%)  : 0
          Child Sensor Statistics :
                Path                                    : /junos/system/linecard/interface/
                Component                               : evo-pfemand
                Component-ID                            : 0
                Average iLatency (ms)                   : 0
                Bytes Sent                              : 651076
                Packets Sent                            : 54
                Drops                                   : 0
```

```
              Initial Sync Bytes Sent           : 0
              Initial Sync Packets Sent         : 0
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 0
              Path                              : /junos/system/linecard/interface/
              Component                         : picd
              Component-ID                      : 0
              Average iLatency (ms)             : 0
              Bytes Sent                        : 771043
              Packets Sent                      : 55
              Drops                             : 0
              Initial Sync Bytes Sent           : 0
              Initial Sync Packets Sent         : 0
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 0
              Path                              : /interfaces/interface/state/
              Component                         : re0/mgmt-ethd
              Component-ID                      : 65535
              Average iLatency (ms)             : 0
              Bytes Sent                        : 34318
              Packets Sent                      : 55
              Drops                             : 0
              Initial Sync Bytes Sent           : 0
              Initial Sync Packets Sent         : 0
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 0
              Path                              : /interfaces/interface/state/
              Component                         : re0/mib2d
              Component-ID                      : 65535
              Average iLatency (ms)             : 0
              Bytes Sent                        : 2291885
              Packets Sent                      : 165
              Drops                             : 0
              Initial Sync Bytes Sent           : 0
              Initial Sync Packets Sent         : 0
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 0
Subscription Details :
    Subscription ID                      : 4
    Type                                 : juniper
    Client IP                            : ipv6:::ffff:10.161.38.194:45730
    Subscription Time (UTC)              : Wed Oct  1 14:40:14 2025
    Sensor Statistics :
        Sensor Path                                 : /junos/system/linecard/interface/queue/
        Reporting Interval                          : 2
        Component(s)                                : Not available
        Component ID                                : 65535
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 0
        Packets Sent                                : 0
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 0
        Initial Sync Packets Sent                   : 0
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 0
        Initial Sync Average Circular Buffer Used (%)  : 0
Subscription Details :
    Subscription ID                     : 5
    Type                                : juniper
    Client IP                           : ipv6:::ffff:10.161.38.194:45730
    Subscription Time (UTC)             : Wed Oct  1 14:40:14 2025
    Sensor Statistics :
        Sensor Path                                 : /junos/system/linecard/qmon-sw/
        Reporting Interval                          : 2
        Component(s)                                : evo-pfemand
        Component ID                                : 0
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 423086
        Packets Sent                                : 54
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 0
        Initial Sync Packets Sent                   : 0
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 0
        Initial Sync Average Circular Buffer Used (%)  : 0
Subscription Details :
    Subscription ID                     : 6
    Type                                : juniper
    Client IP                           : ipv6:::ffff:10.161.53.17:36216
    Subscription Time (UTC)             : Wed Oct  1 14:40:26 2025
    Sensor Statistics :
        Sensor Path                                 : /network-instances/network-instance/mac-
table/entries/entry/
        Reporting Interval                          : 0
        Component(s)                                : l2aldTM,l2ald
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 0
        Packets Sent                                : 0
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 2152
        Initial Sync Packets Sent                   : 2
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 0
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                               : /network-instances/network-instance/mac-table/
entries/entry/
              Component                          : l2ald
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 0
              Packets Sent                       : 0
              Drops                              : 0
              Initial Sync Bytes Sent            : 1953
              Initial Sync Packets Sent          : 1
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 1
              Path                               : /network-instances/network-instance/mac-table/
entries/entry/
              Component                          : l2aldTM
              Component-ID                       : 65535
              Average iLatency (ms)              : 0
              Bytes Sent                         : 0
              Packets Sent                       : 0
              Drops                              : 0
              Initial Sync Bytes Sent            : 199
              Initial Sync Packets Sent          : 1
              Initial Sync Drops                 : 0
              Initial Sync Average iLatency (ms) : 0
Subscription Details :
    Subscription ID                     : 7
    Type                                : gnmi
    Client IP                           : ipv6:::ffff:10.161.53.17:36216
    GNMI mode                           : STREAM
    Subscription Time (UTC)             : Wed Oct  1 14:40:26 2025
    Sensor Statistics :
        Sensor Path                                 : /interfaces/interface/state/admin-
status/
        Reporting Interval                          : 120
        Component(s)                                : re0/mib2d
        GNMI Sub Mode                               : SAMPLE
        Component ID                                : 65535
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 0
        Packets Sent                                : 0
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 21669
        Initial Sync Packets Sent                   : 87
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 4
        Initial Sync Average Circular Buffer Used (%)  : 0
        Sensor Path                                 : /interfaces/interface/state/oper-status/
        Reporting Interval                          : 120
        Component(s)                                : evo-pfemand,re0/mib2d
        GNMI Sub Mode                               : SAMPLE
        Average iLatency (ms)                       : 0
        Average Circular Buffer Used (%)            : 0
        Bytes Sent                                  : 0
        Packets Sent                                : 0
        Drops                                       : 0
        Initial Sync Bytes Sent                     : 41497
        Initial Sync Packets Sent                   : 168
        Initial Sync Drops                          : 0
        Initial Sync Average iLatency (ms)          : 3
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                                : /interfaces/interface/state/oper-status/
              Component                           : evo-pfemand
              GNMI-SubMode                        : SAMPLE
              Component-ID                        : 0
              Average iLatency (ms)               : 0
              Bytes Sent                          : 0
              Packets Sent                        : 0
              Drops                               : 0
              Initial Sync Bytes Sent             : 19943
              Initial Sync Packets Sent           : 81
              Initial Sync Drops                  : 0
              Initial Sync Average iLatency (ms)  : 3
              Path                             : /interfaces/interface/state/oper-status/
              Component                        : re0/mib2d
              GNMI-SubMode                     : SAMPLE
              Component-ID                     : 65535
              Average iLatency (ms)            : 0
              Bytes Sent                       : 0
              Packets Sent                     : 0
              Drops                            : 0
              Initial Sync Bytes Sent          : 21554
              Initial Sync Packets Sent        : 87
              Initial Sync Drops               : 0
              Initial Sync Average iLatency (ms) : 4
    Sensor Statistics :
        Sensor Path                                : /interfaces/interface/subinterfaces/
subinterface/state/admin-status/
        Reporting Interval                         : 120
        Component(s)                               : re0/mib2d
        GNMI Sub Mode                              : SAMPLE
        Component ID                               : 65535
        Average iLatency (ms)                      : 0
        Average Circular Buffer Used (%)           : 0
        Bytes Sent                                 : 0
        Packets Sent                               : 0
        Drops                                      : 0
        Initial Sync Bytes Sent                    : 31742
        Initial Sync Packets Sent                  : 91
        Initial Sync Drops                         : 0
        Initial Sync Average iLatency (ms)         : 12
        Initial Sync Average Circular Buffer Used (%)  : 0
        Sensor Path                                : /interfaces/interface/subinterfaces/
subinterface/state/oper-status/
        Reporting Interval                         : 120
        Component(s)                               : evo-pfemand,re0/mib2d
        GNMI Sub Mode                              : SAMPLE
        Average iLatency (ms)                      : 0
        Average Circular Buffer Used (%)           : 0
        Bytes Sent                                 : 0
        Packets Sent                               : 0
        Drops                                      : 0
        Initial Sync Bytes Sent                    : 63249
        Initial Sync Packets Sent                  : 180
        Initial Sync Drops                         : 0
        Initial Sync Average iLatency (ms)         : 4
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                              : /interfaces/interface/subinterfaces/
subinterface/state/oper-status/
              Component                         : evo-pfemand
              GNMI-SubMode                      : SAMPLE
              Component-ID                      : 0
              Average iLatency (ms)             : 0
              Bytes Sent                        : 0
              Packets Sent                      : 0
              Drops                             : 0
              Initial Sync Bytes Sent           : 30750
              Initial Sync Packets Sent         : 89
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 1
              Path                              : /interfaces/interface/subinterfaces/
subinterface/state/oper-status/
              Component                         : re0/mib2d
              GNMI-SubMode                      : SAMPLE
              Component-ID                      : 65535
              Average iLatency (ms)             : 0
              Bytes Sent                        : 0
              Packets Sent                      : 0
              Drops                             : 0
              Initial Sync Bytes Sent           : 32499
              Initial Sync Packets Sent         : 91
              Initial Sync Drops                : 0
              Initial Sync Average iLatency (ms) : 6
Subscription Details :
    Subscription ID                     : 8
    Type                                : juniper
    Client IP                           : ipv6:::ffff:10.161.53.17:36216
    Subscription Time (UTC)             : Wed Oct  1 14:40:29 2025
    Sensor Statistics :
        Sensor Path                                    : /junos/system/linecard/qmon-sw/
        Reporting Interval                             : 5
        Component(s)                                   : evo-pfemand
        Component ID                                   : 0
        Average iLatency (ms)                          : 0
        Average Circular Buffer Used (%)               : 0
        Bytes Sent                                     : 109920
        Packets Sent                                   : 15
        Drops                                          : 0
        Initial Sync Bytes Sent                        : 182314
        Initial Sync Packets Sent                      : 12
        Initial Sync Drops                             : 0
        Initial Sync Average iLatency (ms)             : 0
        Initial Sync Average Circular Buffer Used (%)  : 0
jnpr@stripe1-leaf1> show network-agent statistics subscription-paths /interfaces/interface/state/
oper-status/ detail
Subscription Details :
    Subscription ID                     : 2
    Type                                : gnmi
    Client IP                           : IPv6:::ffff:10.161.53.17:56132
    GNMI mode                           : STREAM
    Subscription Time (UTC)             : Thu May  1 14:49:53 2025
    Sensor Statistics :
        Sensor Path                                    : /interfaces/interface/state/oper-status/
        Reporting Interval                             : 120
        Component(s)                                   : re0/mib2d,evo-pfemand
        GNMI Sub Mode                                  : SAMPLE
        Average iLatency (ms)                          : 3
        Average Circular Buffer Used (%)               : 0
        Bytes Sent                                     : 2328768
        Packets Sent                                   : 8939
        Drops                                          : 0
        Initial Sync Bytes Sent                        : 40679
        Initial Sync Packets Sent                      : 165
        Initial Sync Drops                             : 0
        Initial Sync Average iLatency (ms)             : 4
        Initial Sync Average Circular Buffer Used (%)  : 0
        Child Sensor Statistics :
              Path                                 : /interfaces/interface/state/oper-status/
              Component                            : evo-pfemand
              GNMI-SubMode                         : SAMPLE
              Component-ID                         : 0
              Average iLatency (ms)                : 2
              Bytes Sent                           : 1087006
              Packets Sent                         : 4187
              Drops                                : 0
              Initial Sync Bytes Sent              : 19165
              Initial Sync Packets Sent            : 78
              Initial Sync Drops                   : 0
              Initial Sync Average iLatency (ms)   : 2
              Path                                 : /interfaces/interface/state/oper-status/
              Component                            : re0/mib2d
```

```
                GNMI-SubMode                      : SAMPLE
                Component-ID                      : 65535
                Average iLatency (ms)             : 5
                Bytes Sent                        : 1241762
                Packets Sent                      : 4752
                Drops                             : 0
                Initial Sync Bytes Sent           : 21514
                Initial Sync Packets Sent         : 87
                Initial Sync Drops                : 0
                Initial Sync Average iLatency (ms) : 5
```

To confirm the status of sensors, you can use: `show agents sensors`

```
jnpr@stripe1-leaf1> show agent sensors
Sensor Information :
    Name                                    : sensor_1000
    Resource                                : /network-instances/network-instance/mac-table/
entries/entry/
    Version                                 : 1.0
    Sensor-id                               : 562949953421313
    Subscription-ID                         : 1000
    Component(s)                            : re0/l2ald-agent
    Profile Information :
        Name                                : export_1000
        Reporting-interval                  : 0
        Payload-size                        : 5000
        Address                             : 0.0.0.0
        Port                                : 1000
        Timestamp                           : ntp
        Format                              : GPB
Sensor Information :
    Name                                    : sensor_1001
    Resource                                : /interfaces/interface/state/admin-status/
    Version                                 : 1.0
    Sensor-id                               : 562949953421443
    Subscription-ID                         : 1001
    Component(s)                            : re0/mib2d
    Profile Information :
        Name                                : export_1001
        Reporting-interval                  : 120
        Payload-size                        : 5000
        Address                             : 0.0.0.0
        Port                                  : 1000
        Timestamp                             : ntp
        Format                                : JSON
Sensor Information :
    Name                                      : sensor_1002
    Resource                                  : /interfaces/interface/state/oper-status/
    Version                                   : 1.0
    Sensor-id                                 : 562949953421314
    Subscription-ID                           : 1002
    Component(s)                              : re0/evoaft-jvisiond-brcm,re0/mib2d
    Profile Information :
        Name                                  : export_1002
        Reporting-interval                    : 120
        Payload-size                          : 5000
        Address                               : 0.0.0.0
        Port                                  : 1000
        Timestamp                             : ntp
        Format                                : JSON
Sensor Information :
    Name                                      : sensor_1003
    Resource                                  : /interfaces/interface/subinterfaces/subinterface/
state/admin-status/
    Version                                   : 1.0
    Sensor-id                                 : 562949953421444
    Subscription-ID                           : 1003
    Component(s)                              : re0/mib2d
    Profile Information :
        Name                                  : export_1003
        Reporting-interval                    : 120
        Payload-size                          : 5000
        Address                               : 0.0.0.0
        Port                                  : 1000
        Timestamp                             : ntp
        Format                                : JSON
Sensor Information :
    Name                                      : sensor_1004
    Resource                                  : /interfaces/interface/subinterfaces/subinterface/
state/oper-status/
    Version                                   : 1.0
    Sensor-id                                 : 562949953421316
    Subscription-ID                           : 1004
    Component(s)                              : re0/evoaft-jvisiond-brcm,re0/mib2d
    Profile Information :
        Name                                    : export_1004
        Reporting-interval                      : 120
        Payload-size                            : 5000
        Address                                 : 0.0.0.0
        Port                                    : 1000
        Timestamp                               : ntp
        Format                                  : JSON
Sensor Information :
    Name                                        : sensor_1005
    Resource                                    : /components/component/cpu/utilization/
    Version                                     : 1.0
    Sensor-id                                   : 562949953421450
    Subscription-ID                             : 1005
    Component(s)                                : re0/ehmd
    Profile Information :
        Name                                    : export_1005
        Reporting-interval                      : 2
        Payload-size                            : 5000
        Address                                 : 0.0.0.0
        Port                                    : 1000
        Timestamp                               : ntp
        Format                                  : GPB
Sensor Information :
    Name                                        : sensor_1006
    Resource                                    : /junos/system/linecard/npu/memory/
    Version                                     : 1.0
    Sensor-id                                   : 562949953421449
    Subscription-ID                             : 1006
    Component(s)                                : re0/evoaft-jvisiond-brcm
    Profile Information :
        Name                                    : export_1006
        Reporting-interval                      : 2
        Payload-size                            : 5000
        Address                                 : 0.0.0.0
        Port                                    : 1000
        Timestamp                               : ntp
        Format                                  : GPB
Sensor Information :
    Name                                        : sensor_1007
    Resource                                    : /junos/system/linecard/qmon-sw/
    Version                                     : 1.0
    Sensor-id                                   : 562949953421452
    Subscription-ID                             : 1007
    Component(s)                            : re0/evoaft-jvisiond-brcm
    Profile Information :
        Name                                : export_1007
        Reporting-interval                  : 2
        Payload-size                        : 5000
        Address                             : 0.0.0.0
        Port                                : 1000
        Timestamp                           : ntp
        Format                              : GPB
Sensor Information :
    Name                                    : sensor_1008
    Resource                                : /interfaces/interface/state/
    Version                                 : 1.0
    Sensor-id                               : 562949953421451
    Subscription-ID                         : 1008
    Component(s)                            : re0/evoaft-jvisiond-brcm,re0/mgmt-ethd,re0/mib2d
    Profile Information :
        Name                                : export_1008
        Reporting-interval                  : 2
        Payload-size                        : 5000
        Address                             : 0.0.0.0
        Port                                : 1000
        Timestamp                           : ntp
        Format                              : GPB
Sensor Information :
    Name                                    : sensor_1009
    Resource                                : /junos/system/linecard/qmon-sw/
    Version                                 : 1.0
    Sensor-id                               : 562949953421427
    Subscription-ID                         : 1009
    Component(s)                            : re0/evoaft-jvisiond-brcm
    Profile Information :
        Name                                : export_1009
        Reporting-interval                  : 5
        Payload-size                        : 5000
        Address                             : 0.0.0.0
        Port                                : 1000
        Timestamp                           : ntp
        Format                              : GPB
Sensor Information :
    Name                                    : sensor_1011
    Resource                                : /lldp/state/enabled/
    Version                                 : 1.0
    Sensor-id                                : 562949953421493
    Subscription-ID                          : 1011
    Component(s)                             : re0/l2cpd-agent
    Profile Information :
        Name                                 : export_1011
        Reporting-interval                   : 30
        Payload-size                         : 5000
        Address                              : 0.0.0.0
        Port                                 : 1000
        Timestamp                            : ntp
        Format                               : JSON
```

Recommended KPIs to Monitor

Table 50. Recommended KPIs for Monitoring GPU Backend Fabric with Junos Commands and Telemetry Sensors Paths

KPI	JUNOS COMMAND	SENSOR
Interface State	*show interfaces <interface> terse*	/interfaces/interface[name=<interface>]/state/oper-status /interfaces/interface[name=<interface>]/state/admin-status
Interface Description	*show interfaces <interface> extensive | match Description*	/interfaces/interface[name=<interface>]/state/description
Interface MTU	*show interfaces <interface> extensive | match MTU*	/interfaces/interface[name=<interface>]/state/mtu
Interface speed	*show interfaces <interface> extensive | match speed*	/interfaces/interface[name=<interface>]/state/high-speed
Interface input Drops	*show interfaces <interface> extensive | find "Input errors"*	/interfaces/interface[name=<interface>]/state/counters/in-discards
Interface output Drops	*show interfaces <interface> extensive | find "Output errors"*	/interfaces/interface[name=<interface>]/state/counters/out-discards

(Continued)

KPI	JUNOS COMMAND	SENSOR
Interface output Pkts	*run show interfaces <interface> extensive | match "Total Packets"*	/interfaces/interface[name=<interface>]/state/counters/out-pkts
Interface output unicast Pkts	*run show interfaces <interface> extensive | match Unicast*	/interfaces/interface[name=<interface>]/state/counters/out-unicast-pkts
Interface input Pkts	*run show interfaces <interface> extensive | match "Total Packets"*	/interfaces/interface[name=<interface>]/state/counters/in-pkts
Interface input unicast Pkts	*run show interfaces <interface> extensive | match Unicast*	/interfaces/interface[name=<interface>]/state/counters/in-unicast-pkts
Per interface ECN marked packets	*show interfaces <interface> extensive | match ecn*	/state/interfaces/interface[name=<interface>/counters/errors/out-ecn-ce-marked-pkts /junos/system/linecard/qmon-sw/ /cos/interfaces/interface/queues/queue/ecnMarkedPkts
Per interface per queue buffer-occupancy	*show interfaces queue buffer-occupancy <interface>*	/junos/system/linecard/qmon-sw/ /cos/interfaces/interface/queues/queue/peakBufferOccupancyPercent /cos/interfaces/interface/queues/queue/peakBufferOccupancy
Per Interface, Per forwarding class (queue) Tail Drops	*show interfaces queue <interface> forwarding-class <forwarding-class> | match "Tail"*	/junos/system/linecard/qmon-sw/ /cos/interfaces/interface/queues/queue/tailDropPkts

(Continued)

KPI	JUNOS COMMAND	SENSOR
Per Interface PFC Pause frames	*show interfaces <interface> extensive | math "Priority : <priority>"*	/interfaces/interface[name=<interface-name>]/ethernet/state/counters/in-pause-pkts /interfaces/interface[name=<interface-name>]/ethernet/state/counters/out-pause-pkts
EVPN l3-context	*show evpn l3-context extensive <context-name>* <context-name> = VRF routing-instance name = Tenant name	/junos/evpn/l3-context[context-name=<context-name>]/ <context-name> = VRF routing-instance name = Tenant name /junos/evpn/l3-context[context-name=<context-name>]/advertisement-mode /junos/evpn/l3-context[context-name=<context-name>]/encapsulation /junos/evpn/l3-context[context-name=<context-name>]/ip-prefix-database /junos/evpn/l3-context[context-name=<context-name>]/ip-prefix-database/route-status
IPv6 BGP Underlay Advertised routes	*show route advertised-routes protocol bgp <neighbor-address> extensive* <neighbor-address> = auto discovered link local address of directly connected EBGP neighbor	/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/IPv6-unicast/neighbors/neighbor/adj-rib-out-pre/routes/
IPv6 BGP Underlay Received routes	*show route received-routes protocol bgp <neighbor-address> extensive* <neighbor-address> = auto discovered link local address of directly connected EBGP neighbor	/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/IPv6-unicast/neighbors/neighbor/adj-rib-in-pre/routes/ /network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/IPv6-unicast/neighbors/neighbor/adj-rib-in-post/routes/

(Continued)

KPI	JUNOS COMMAND	SENSOR
EVPN BGP Advertised type-5 routes per tenant	*show route advertised-routes protocol bgp <neighbor-address> extensive* <neighbor-address> = IPv4 or IPv6 address of remote VTEP neighbor	/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/l2vpn-evpn/loc-rib/routes/route-distinguisher[route-distinguisher=<tenant-route-distinguisher>]/type-five-ip-prefix/type-five-route
EVPN BGP Received type-5 routes per tenant	*show route advertised-routes protocol bgp <neighbor-address> extensive table <Tenant-name.inet.0>* <neighbor-address> = IPv4 or IPv6 address of remote VTEP neighbor	/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/l2vpn-evpn/loc-rib/routes/route-distinguisher[route-distinguisher=<tenant-route-distinguisher>]/type-five-ip-prefix/type-five-route/paths/path/state/source-route-distinguisher
EVPN BGP Received type-5 routes per tenant per neighbor	*show route advertised-routes protocol bgp <neighbor-address> extensive table <Tenant-name.inet.0>* <neighbor-address> = IPv4 or IPv6 address of remote VTEP neighbor	/network-instances/network-instance/protocols/protocol/bgp/rib/afi-safis/afi-safi/l2vpn-evpn/loc-rib/routes/route-distinguisher[route-distinguisher=<tenant-route-distinguisher>]/type-five-ip-prefix/type-five-route/paths/path[peer-ip =<neighbor-IPv6-address>/state/source-route-distinguisher

Refer to Network Configuration Example: AI/ML - Telemetry Reference Guide for more details.

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

Table 51: Validated Devices and Positioning

Validated Devices and Positioning		
Fabric	Leaf Switches	Spine Switches
Frontend	QFX5130-32CD	QFX5130-32CD
GPU Backend	QFX5240-64OD	QFX5240-64CD
Storage Backend	QFX5220-32CD QFX5230-64CD QFX5240-64CD	QFX5220-32CD QFX5230-64CD QFX5240-64CD

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

Table 52: Platform Recommended Release

Platform	Role	Junos OS Release
QFX5240-64CD	GPU Backend Leaf	23.4X100-D31
QFX5240-64CD	GPU Backend Spine	23.4X100-D31

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

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.
- Ensuring the design is well-documented and will produce a reliable, predictable deployment for the customer.

The qualification objectives included:

- 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:

- 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.
- Software operation verification
- IPv6 Stateless Address Auto-configuration (SLAAC)
- Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop (RFC5549)
- BGP IPv6 link local neighbor autodiscovery

Under these scenarios the following were evaluated/validated:

- Completion of AI Job models within MLCommons Training benchmarks
- Traffic recovery after all failure scenarios.

Other Features Tested

- Broadcom 97608 THOR2 NICs
- Mellanox Connect-X NICs
- DSCP and CNP configuration on the NICs
- BERT/LLAMA3 test completion times
- Llama2 Inference against existing infrastructure.
- Refer to the test report for more information.

Features Not Included

- IPv4 DHCP/DHCP relay for tenants – Might be included in future version of this JVD
- IPv6 DHCP/DHCP relay for tenants – Might be included in future version of this JVD
- Multihomed – TBD
- Global Load Balancing (GLB) – Will be included in future JVD
- Storage Multitenancy – TBD
- Inference/Frontend Multitenancy – Will be included in future JVD
- IPv6 underlay/overlay deployment using Apstra – Will be included in future version of this JVD

Tested Optics

Table 54: 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 55: 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
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

(Continued)

Backend Storage Fabric				
740-061405	QSFP-100GBASE-SR4	Storage	Vast Storage DBOX	ConnectX-6

Table 56: Backend GPU Fabric Optics

Backend GPU Fabric				
Part number	Optics Name	Device Role	Device Model	Interface/NIC type
740-174933	OSFP-800G-DR8	spine	QFX5240-64OD	OSPF800
740-174933	OSFP-800G-DR8	leaf	QFX5240-64OD	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
Q112-400G-DR4	400G QSFP112 DR4 1310 nm	GPU Server	AMD MI300Xx SuperMicro AS-8125GS-TNMR2	POLLARA 1x400G QSFP112 (AMD Pensando™ Pollara 400 AI NIC)

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, see the Test Report Brief.

Recommendations Summary

Follow best practice recommendations:

- A minimum of 4 spines in each fabric is suggested.

NOTE: 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 QFX5240s 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 servers and change the NCCL_SOCKET interface to be the management (frontend) interface.
- The recommended Junos OS releases for this JVD is Junos OS Release 23.4X100-D31.6-EVO for the Juniper QFX5240-64CD

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 57: Revision History

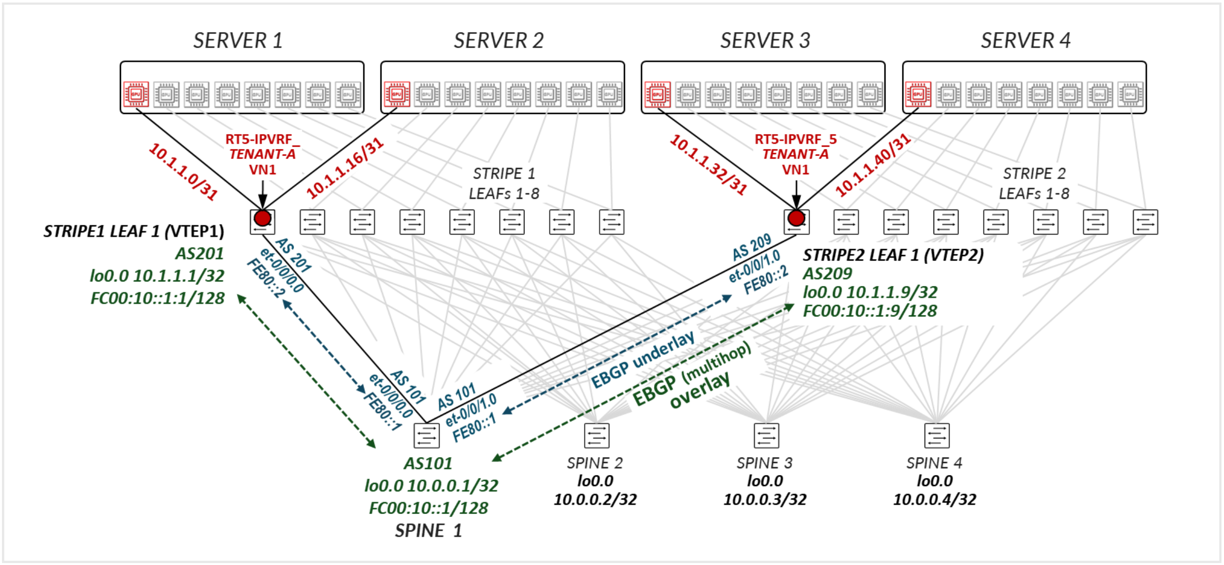
Date	Version	Description
Sep 2025	JVD-AICLUSTERDC-EVPNType5-01-04	Replaced the use of VRFs on the GPU servers with rio-prefix under IPv6 router advertisement. Moved IPv4 content to Appendix A.
Aug 2025	JVD-AICLUSTERDC-EVPNType5-01-03	Added Pollara NIC references and RCCL description in the "Tested Optics" on page 188 section.
June 2025	JVD-AICLUSTERDC-EVPNType5-01-02	New content on IPv6 SLAAC for GPU servers address assignment and how to run a job using IPv6, plus clarified content and improved examples.
May 2025	JVD-AICLUSTERDC-EVPNType5-01-01	Initial Publish

Appendix A – IPv4 Overlay Over IPv6 Underlay Fabric Implementation

When the underlay BGP sections use IPv6 and peer auto-discovery, and the overlay is IPv4, the overlay BGP sessions must be configured to advertise IPv4 routes with IPv6 next-hops as described in RFC 5549 (Advertising IPv4 Network Layer Reachability Information with an IPv6 Next Hop).

Consider the example depicted in Figure below.

Figure: IPv6 Link-Local underlay and IPv6 Overlay Example



IPv6 GPU Server NICs to Leaf Nodes Connections

The links between the GPU interfaces and the leaf nodes are statically configured with /31 IPv4 addresses as shown in the Table below No Router advertisements are sent by the leaf nodes, and SLAAC is not used in this case. All the IPv4 addresses in the example are subnets of 10.200/16 (with 10.200.0/24 being assigned to the links between the GPU servers and the leaf nodes in stripe 1, and 10.200.1/24 being assigned to the links between the GPU servers and the leaf nodes in stripe 2).

LEAF NODE INTERFACE	LEAF NODE IPv4 ADDRESS	GPU NIC	GPU NIC IPv4 ADDRESS
Stripe 1 Leaf 1 - et-0/0/0:0	10.200.0.0/31	Server 1 - gpu0_eth	10.200.0.1/31
Stripe 1 Leaf 2 - et-0/0/0:0	10.200.0.2/31	Server 1 - gpu1_eth	10.200.0.3/31
Stripe 1 Leaf 3 - et-0/0/0:0	10.200.0.4/31	Server 1 - gpu2_eth	10.200.0.5/31
Stripe 1 Leaf 4 - et-0/0/0:0	10.200.0.6/31	Server 1 - gpu3_eth	10.200.0.7/31
Stripe 1 Leaf 5 - et-0/0/0:0	10.200.0.8/31	Server 1 - gpu4_eth	10.200.0.9/31
Stripe 1 Leaf 6 - et-0/0/0:0	10.200.0.10/31	Server 1 - gpu5_eth	10.200.0.11/31
Stripe 1 Leaf 7 - et-0/0/0:0	10.200.0.12/31	Server 1 - gpu6_eth	10.200.0.13/31
Stripe 1 Leaf 8 - et-0/0/0:0	10.200.0.14/31	Server 1 - gpu7_eth	10.200.0.15/31

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv4 ADDRESS	GPU NIC	GPU NIC IPv4 ADDRESS
Stripe 1 Leaf 1 - et-0/0/1:0	10.200.0.16/31	Server 2 - gpu0_eth	10.200.0.17/31
Stripe 1 Leaf 2 - et-0/0/1:0	10.200.0.18/31	Server 2 - gpu1_eth	10.200.0.19/31
Stripe 1 Leaf 3 - et-0/0/1:0	10.200.0.20/31	Server 2 - gpu2_eth	10.200.0.21/31
Stripe 1 Leaf 4 - et-0/0/1:0	10.200.0.22/31	Server 2 - gpu3_eth	10.200.0.23/31
Stripe 1 Leaf 5 - et-0/0/1:0	10.200.0.24/31	Server 2 - gpu4_eth	10.200.0.25/31
Stripe 1 Leaf 6 - et-0/0/1:0	10.200.0.26/31	Server 2 - gpu5_eth	10.200.0.27/31
Stripe 1 Leaf 7 - et-0/0/1:0	10.200.0.28/31	Server 2 - gpu6_eth	10.200.0.29/31
Stripe 1 Leaf 8 - et-0/0/1:0	10.200.0.30/31	Server 2 - gpu7_eth	10.200.0.31/31
Stripe 1 Leaf 1 - et-0/0/2:0	10.200.0.32/31	Server 3 - gpu0_eth	10.200.0.33/31
Stripe 1 Leaf 2 - et-0/0/2:0	10.200.0.34/31	Server 3 - gpu1_eth	10.200.0.35/31
Stripe 1 Leaf 3 - et-0/0/2:0	10.200.0.36/31	Server 3 - gpu2_eth	10.200.0.37/31
Stripe 1 Leaf 4 - et-0/0/2:0	10.200.0.38/31	Server 3 - gpu3_eth	10.200.0.39/31
Stripe 1 Leaf 5 - et-0/0/2:0	10.200.0.40/31	Server 3 - gpu4_eth	10.200.0.41/31
Stripe 1 Leaf 6 - et-0/0/2:0	10.200.0.42/31	Server 3 - gpu5_eth	10.200.0.43/31
Stripe 1 Leaf 7 - et-0/0/2:0	10.200.0.44/31	Server 3 - gpu6_eth	10.200.0.45/31
Stripe 1 Leaf 8 - et-0/0/2:0	10.200.0.46/31	Server 3 - gpu7_eth	10.200.0.47/31
Stripe 2 Leaf 1 - et-0/0/0:0	10.200.1.0/31	Server 9 - gpu0_eth	10.200.1.1/31
Stripe 2 Leaf 2 - et-0/0/0:0	10.200.1.2/31	Server 9 - gpu1_eth	10.200.1.3/31

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv4 ADDRESS	GPU NIC	GPU NIC IPv4 ADDRESS
Stripe 2 Leaf 3 - et-0/0/0:0	10.200.1.4/31	Server 9 - gpu2_eth	10.200.1.5/31
Stripe 2 Leaf 4 - et-0/0/0:0	10.200.1.6/31	Server 9 - gpu3_eth	10.200.1.7/31
Stripe 2 Leaf 5 - et-0/0/0:0	10.200.1.8/31	Server 9 - gpu4_eth	10.200.1.9/31
Stripe 2 Leaf 6 - et-0/0/0:0	10.200.1.10/31	Server 9 - gpu5_eth	10.200.1.11/31
Stripe 2 Leaf 7 - et-0/0/0:0	10.200.1.12/31	Server 9 - gpu6_eth	10.200.1.13/31

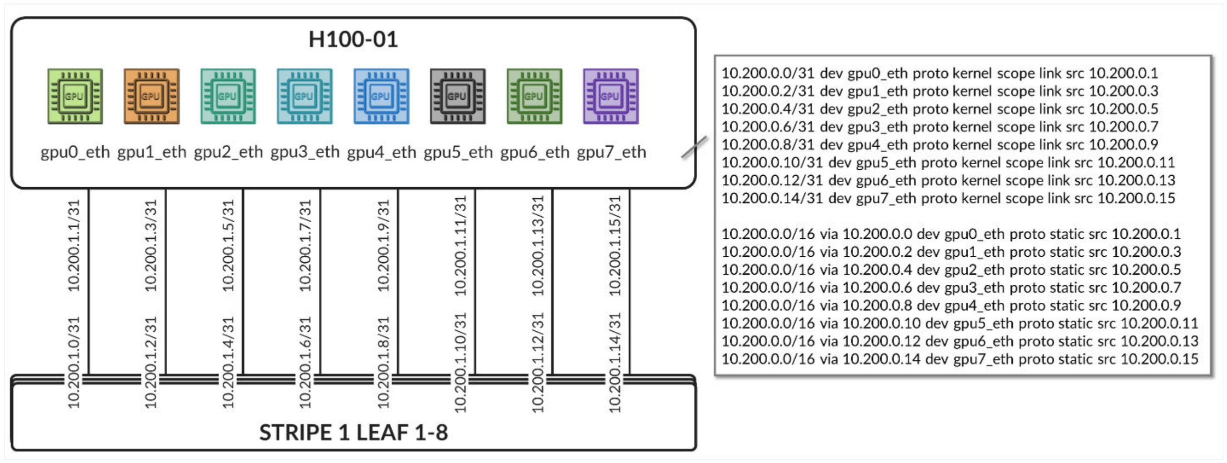
The following example shows the configuration of the interfaces on the leaf node. Only family IPv4 is enabled, with a /31 static IPv4 address.

```
[edit interfaces et-0/0/0]
jnpr@stripe1-leaf1# show
description "Breakout et-0/0/0";
number-of-sub-ports 2;
speed 400g;
[edit interfaces et-0/0/0:0]
jnpr@stripe1-leaf1# show
mtu 9216;
unit 2 {
    family inet {
      address 10.200.0.254/24;
    }
}
```

The following example shows the configuration of the interfaces on the server side. Only family IPv4 is enabled, with a /31 static IPv4 address.

```
gpu0_eth:
      match:
        macaddress: a0:88:c2:3b:50:66
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.0.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.0.254
          from: 10.200.0.10
      set-name: gpu0_eth
```

The netplan disables dhcp4 and configures a static IPv4 address on each of the gpu_eth interfaces. It also configures a static route for prefix 10.200/16, pointing to the address of the leaf node, for each gpu_eth. The route includes the address of the interface, which guarantees that the correct interface is used when sending traffic from a gpu_eth interface to a remote address belonging to the same tenant.



```
10.200.0.0/31 dev gpu0_eth proto kernel scope link src 10.200.0.1
10.200.0.2/31 dev gpu1_eth proto kernel scope link src 10.200.0.3
10.200.0.4/31 dev gpu2_eth proto kernel scope link src 10.200.0.5
10.200.0.6/31 dev gpu3_eth proto kernel scope link src 10.200.0.7
10.200.0.8/31 dev gpu4_eth proto kernel scope link src 10.200.0.9
10.200.0.10/31 dev gpu5_eth proto kernel scope link src 10.200.0.11
10.200.0.12/31 dev gpu6_eth proto kernel scope link src 10.200.0.13
10.200.0.14/31 dev gpu7_eth proto kernel scope link src 10.200.0.15

10.200.0.0/16 via 10.200.0.0 dev gpu0_eth proto static src 10.200.0.1
10.200.0.0/16 via 10.200.0.2 dev gpu1_eth proto static src 10.200.0.3
10.200.0.0/16 via 10.200.0.4 dev gpu2_eth proto static src 10.200.0.5
10.200.0.0/16 via 10.200.0.6 dev gpu3_eth proto static src 10.200.0.7
10.200.0.0/16 via 10.200.0.8 dev gpu4_eth proto static src 10.200.0.9
10.200.0.0/16 via 10.200.0.10 dev gpu5_eth proto static src 10.200.0.11
10.200.0.0/16 via 10.200.0.12 dev gpu6_eth proto static src 10.200.0.13
10.200.0.0/16 via 10.200.0.14 dev gpu7_eth proto static src 10.200.0.15
```

Netplan Example

```
jnpr@H100-01:/etc/netplan$ sudo cat 00-installer-config-type5_vrf.yaml
# This is the network config written by 'subiquity'
network:
  version: 2
  ethernets:
    mgmt_eth:
      match:
        macaddress: 6c:fe:54:48:2e:48
      dhcp4: false
      addresses:
        - 10.10.1.16/31
      nameservers:
        addresses:
        - 8.8.8.8
      routes:
        - to: default
```

```
          via: 10.10.1.17
      set-name: mgmt_eth
    gpu0_eth:
      match:
        macaddress: a0:88:c2:3b:50:66
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.0.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.0.254
          from: 10.200.0.10
      set-name: gpu0_eth
    gpu1_eth:
      match:
        macaddress: a0:88:c2:3b:50:6a
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.1.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.1.254
          from: 10.200.1.10
      set-name: gpu1_eth
    gpu2_eth:
      match:
        macaddress: a0:88:c2:3b:50:6e
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.2.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.2.254
          from: 10.200.2.10
      set-name: gpu2_eth
    gpu3_eth:
      match:
        macaddress: a0:88:c2:3b:50:72
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.3.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.3.254
          from: 10.200.3.10
      set-name: gpu3_eth
    gpu4_eth:
      match:
        macaddress: a0:88:c2:0a:79:48
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.4.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.4.254
          from: 10.200.4.10
      set-name: gpu4_eth
    gpu5_eth:
      match:
        macaddress: a0:88:c2:0a:79:4c
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.5.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.5.254
          from: 10.200.5.10
      set-name: gpu5_eth
    gpu6_eth:
      match:
        macaddress: a0:88:c2:0a:79:40
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.6.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.6.254
          from: 10.200.6.10
      set-name: gpu6_eth
    gpu7_eth:
      match:
        macaddress: a0:88:c2:0a:79:44
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.200.7.10/24
      routes:
        - to: 10.200.0.0/16
          via: 10.200.7.254
          from: 10.200.7.10
      set-name: gpu7_eth
    stor0_eth:
      match:
        macaddress: b8:3f:d2:63:e5:44
      dhcp4: false
      mtu: 9000
      addresses:
        - 10.100.1.13/31
      routes:
        - to: 10.100.0.0/21
          via: 10.100.1.12
      set-name: stor0_eth
```

Refer to the following documentation for details to configure the interfaces on AMD GPU servers or NVIDIA GPU servers respectively:

- AMD Configuration | Juniper Networks
- NVIDIA Configuration | Juniper Networks

All leaf and spine nodes are configured with IPv4 addresses under the loopback interface (lo0.0). The loopback and Autonomous System numbers for all devices in the fabric are included in Table 23:

Table 23. Spine and Leaf Loopback Addresses and ASNs

LEAF NODE INTERFACE	lo0.0 IPV4 ADDRESS	Local AS #
Stripe 1 Leaf 1	10.0.1.1/32	201
Stripe 1 Leaf 2	10.0.1.2/32	202
Stripe 1 Leaf 3	10.0.1.3/32	203

(Continued)

LEAF NODE INTERFACE	lo0.0 IPV4 ADDRESS	Local AS #
Stripe 1 Leaf 4	10.0.1.4/32	204
Stripe 1 Leaf 5	10.0.1.5/32	205
Stripe 1 Leaf 6	10.0.1.6/32	206
Stripe 1 Leaf 7	10.0.1.7/32	207
Stripe 1 Leaf 8	10.0.1.8/32	208
Stripe 2 Leaf 1	10.0.1.9/32	209
Stripe 2 Leaf 2	10.0.1.10/32	210
. . .		
SPINE1		101
SPINE2		102
SPINE3		103
SPINE4		104

IPv6 Leaf Nodes to Spine Nodes Connections Using Link Local Addresses

When deploying the underlay using IPv6 Link-Local underlay, the interfaces between the leaf and spine nodes do not require explicitly configured IP addresses and are configured as untagged interfaces with only family inet6 to enable processing of IPv6 traffic as shown in Figure 50.

Figure 50: Leaf nodes to spine nodes connectivity

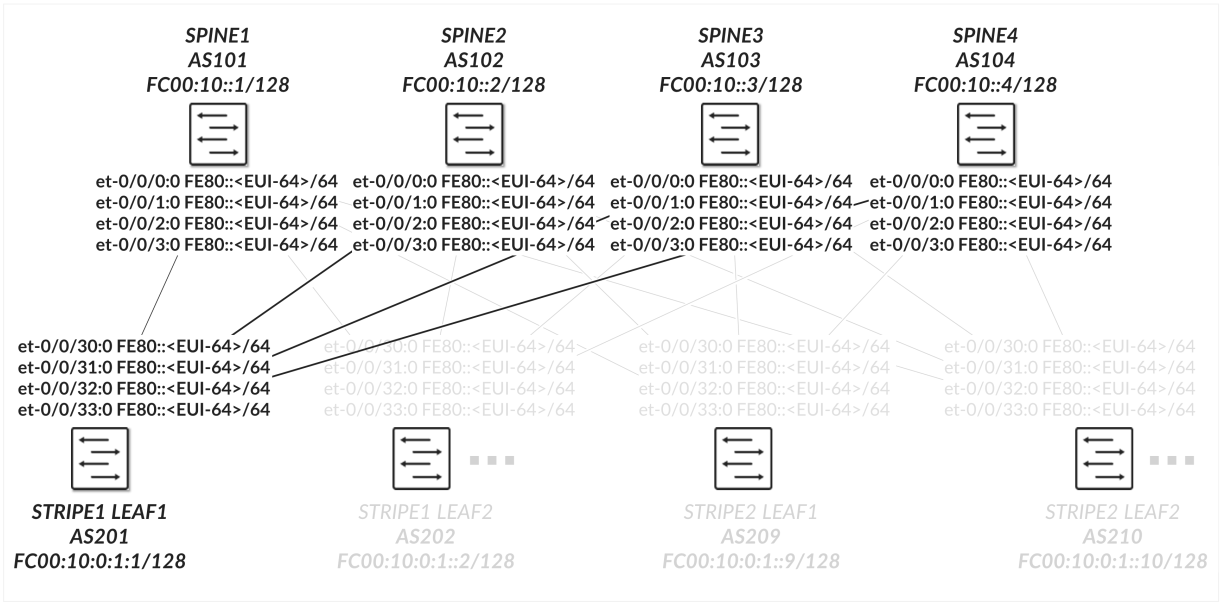


Table 24. Spine to Leaf Interface Configuration Example

STRIPE 1 LEAF 1 (et-0/0/0:30)	SPINE 1 (et-0/0/0:0)

```
[edit interfaces et-0/0/30]
jnpr@stripe1-leaf1# show
description "Breakout et-0/0/30";
number-of-sub-ports 1;
speed 800g;

[edit interfaces et-0/0/30:0]
jnpr@stripe1-leaf1# show
description facing_spine1:et-0/0/0:0;
mtu 9216;
unit 0 {
    family inet6 {
        mtu 9202;
    }
}
```
	```
	[edit interfaces et-0/0/0]
	jnpr@spine1# show
	description "Breakout et-0/0/0";
	number-of-sub-ports 1;
	speed 800g;

	[edit interfaces et-0/0/0:0]
	jnpr@spine1# show
	description facing_Leaf1:et-0/0/0:0;
	mtu 9216;
	unit 0 {
	    family inet6 {
	        mtu 9202;
	    }
	}
	```

Enabling IPv6 on an interface automatically assigns a link-local IPv6 address. The switch autogenerates link local addresses for the interfaces using the EUI-64 address format (based on the interface's MAC address), as shown in Table 25.

Table 25. Spine and Leaf IPv6-Enabled Interface Link Local Addresses

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
Stripe 1 Leaf 1 - et-0/0/30:0	fe80::9e5a:80ff:fec1:ae00/64	Spine 1 – et-0/0/0:0	fe80::9e5a:80ff:feef:a28f/64
Stripe 1 Leaf 1 - et-0/0/31:0	fe80::9e5a:80ff:fec1:ae08/64	Spine 2 – et-0/0/0:0	fe80::5a86:70ff:fe7b:ced5/64

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
Stripe 1 Leaf 1 - et-0/0/32:0	fe80::9e5a:80ff:fec1:af00/64	Spine 3 – et-0/0/0:0	fe80::5a86:70ff:fe78:e0d5/64
Stripe 1 Leaf 1 - et-0/0/33:0	fe80::9e5a:80ff:fec1:af08/64	Spine 4 – et-0/0/0:0	fe80::5a86:70ff:fe79:3d5/64
Stripe 1 Leaf 2 - et-0/0/30:0	fe80::5a86:70ff:fe79:dad5/64	Spine 1 – et-0/0/1:0	fe80::9e5a:80ff:feef:a297/64
Stripe 1 Leaf 2 - et-0/0/31:0	fe80::5a86:70ff:fe79:dadd/64	Spine 2 – et-0/0/1:0	fe80::5a86:70ff:fe7b:cedd/64
Stripe 1 Leaf 2 - et-0/0/32:0	fe80::5a86:70ff:fe79:dbd5/64	Spine 3 – et-0/0/1:0	fe80::5a86:70ff:fe78:e0dd/64
Stripe 1 Leaf 2 - et-0/0/33:0	fe80::5a86:70ff:fe79:dbdd/64	Spine 4 – et-0/0/1:0	fe80::5a86:70ff:fe79:3dd/64
. . .			

These addresses need to be advertised through standard router advertisements as part of the IPv6 Neighbor Discovery process to allow the leaf and spine nodes to then establish BGP sessions between them. Router advertisement must be enabled on all the interfaces between the leaf and spine nodes as shown:

Table 26. IPv6 Router Advertisement on Leaf and Spine Interfaces

STRIPE 1 LEAF 1	SPINE 1

[edit protocols router-advertisement]
/* ROUTER ADVERTISEMENTS TO SPINE1 */
interface et-0/0/30:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE2 */
interface et-0/0/31:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE3 */
interface et-0/0/32:0.0;
/* ROUTER ADVERTISEMENTS TO SPINE4 */
interface et-0/0/33:0.0;
}
	[edit]
	jnpr@spine1# show protocols router-advertisement
	/* ROUTER ADVERTISEMENTS TO LEAF1 */
	interface et-0/0/0:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF2 */
	interface et-0/0/1:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF3 */
	interface et-0/0/2:0.0;
	/* ROUTER ADVERTISEMENTS TO LEAF4 */
	interface et-0/0/3:0.0;
	.
	.
	.

To verify that router advertisements are being sent you can use: `show IPv6 router-advertisement interface <interface>` and `show IPv6 neighbors`

Example:

```
jnpr@stripe1-leaf1> show IPv6 router-advertisement interface et-0/0/30:0
Interface: et-0/0/30:0.0
  Advertisements sent: 4, last sent 00:02:28 ago
  Solicits sent: 1, last sent 00:08:06 ago
  Solicits received: 0
  Advertisements received: 3
  Solicited router advertisement unicast: Disable
  IPv6 RA Preference: DEFAULT/MEDIUM
  Passive mode: Disable
  Upstream mode: Disable
  Downstream mode: Disable
  Proxy blackout timer: Not Running
  Advertisement from fe80::9e5a:80ff:feef:a28f, heard 00:01:57 ago
    Managed: 0
    Other configuration: 0
    Reachable time: 0 ms
    Default lifetime: 1800 sec
    Retransmit timer: 0 ms
    Current hop limit: 64
jnpr@stripe1-leaf1> show IPv6 neighbors
IPv6 Address    Linklayer Address  State       Exp   Rtr  Secure  Interface
fe80::5a86:70ff:fe78:e0d5       58:86:70:78:e0:d5  reachable   11    yes  no
et-0/0/31:0.0
fe80::5a86:70ff:fe79:3d5        58:86:70:79:03:d5  reachable   23    yes  no
et-0/0/33:0.0
fe80::5a86:70ff:fe7b:ced5       58:86:70:7b:ce:d5  reachable   13    yes  no
et-0/0/32:0.0
fe80::9e5a:80ff:feef:a28f       9c:5a:80:ef:a2:8f  reachable   25    yes  no
et-0/0/30:0.0
Total entries: 4
```

The loopback interface IPv6 addresses and the Autonomous System numbers for all devices in the fabric are included in Table 26:

Table 26. Spine and Leaf Loopback Addresses and ASNs

LEAF NODE INTERFACE	lo0.0 IPv6 ADDRESS	Local AS #
Stripe 1 Leaf 1	FC00:10:0:1::1/128	201
Stripe 1 Leaf 2	FC00:10:0:1::2/128	202
Stripe 1 Leaf 3	FC00:10:0:1::3/128	203
Stripe 1 Leaf 4	FC00:10:0:1::4/128	204
Stripe 1 Leaf 5	FC00:10:0:1::5/128	205
Stripe 1 Leaf 6	FC00:10:0:1::6/128	206
Stripe 1 Leaf 7	FC00:10:0:1::7/128	207
Stripe 1 Leaf 8	FC00:10:0:1::8/128	208
Stripe 2 Leaf 1	FC00:10:0:1::9/128	209
Stripe 2 Leaf 2	FC00:10:0:1::10/128	210
. . .		
SPINE1	FC00:10:0::1/128	101
SPINE2	FC00:10:0::2/128	102
SPINE3	FC00:10:0::3/128	103
SPINE4	FC00:10:0::4/128	104

Table 27. Spine and Leaf Loopback Address Configuration

STRIPE 1 LEAF 1	SPINE 1

[edit interfaces lo0]
jnpr@stripe1-leaf1# show
unit 0 {
  family inet {
    address FC00:10:0:1::1/128;
  }
}
	[edit interfaces lo0]
	jnpr@spine1# show
	unit 0 {
	  family inet {
	    address FC00:10::1/128;
	  }
	}

Recommended MTU

Configure the MTU consistently across the fabric and make sure that the MTU of the server->leaf links does not exceed the MTU of the leaf->spine links considering the extra overhead of the VXLAN encapsulation.

VXLAN Overhead Calculation

For IPv6, the MTU can also be calculated as:

Table 28 VXLAN Overhead Calculation

HEADER	BYTES
Outer Ethernet	14
Outer IP (IPv6)	40
UDP	8
VXLAN	8
Total	70 bytes

Recommended MTU Strategy

Table 29. Recommended MTU

LINK TYPE	MTU
Server ↔ Leaf	9000
Leaf ↔ Spine IPv6	> 9070

It is important to keep in mind that RoCEv2 message sizes are still limited by the RDMA MTU reported by ibv_devinfo

```
jnpr@MI300-01:~/SCRIPTS$ ibv_devinfo -d bnxt_re0
hca_id: bnxt_re0
        transport:              InfiniBand (0)
        fw_ver:                 230.2.49.0
        node_guid:      7ec2:55ff:febd:75d0
        sys_image_guid:         7ec2:55ff:febd:75d0
        vendor_id:              0x14e4
        vendor_part_id:         5984
        hw_ver:                 0x1D42
        phys_port_cnt:          1
                port:   1
                        state:          PORT_ACTIVE (4)
                         max_mtu:               4096 (5)
                         active_mtu:            4096 (5)
                        sm_lid:                 0
                        port_lid:                       0
                        port_lmc:               0x00
                        link_layer:                     Ethernet
```

Table 30. MTU Types: Ownership and Functional Role

MTU TYPE	OWNER	PURPOSE
Interface MTU (e.g. 9000) **ifconfig, ip**	Linux network stack	Defines the max L3/IP packet size
RDMA MTU (e.g. 4096) **ibv_devinfo**	RDMA stack	Defines the max RDMA message size per Work Queue Element (WQE)

The RDMA MTU can be configured at the verbs level, and it’s negotiated during QP (Queue Pair) setup. You *cannot* override it by just setting the NIC's MTU to a higher value, but you would need to use low-level tools or RDMA apps.

Some performance tools such as ib_send_bw, ib_write_bw (via -m flag). For example:

ib_write_bw -m 1024 # sets RDMA MTU to 1024 bytes

ib_write_bw -m 4096 # sets RDMA MTU to 4096 (max allowed according to the output of ibv_devinfo shown before)

RDMA MTU must be ≤ Interface MTU – encapsulation overhead.

IPv6 GPU Backend Fabric Underlay, using BGP neighbor discovery

Refer to . [Configure BGP Unnumbered EVPN Fabric | Juniper Networks](#) for more information.

The underlay EBGP sessions are configured between the leaf and spine nodes to use peer auto-discovery, and are configured to advertise these loopback interfaces, as shown in the example between Stripe1 Leaf 1 and Spine 1 below:

Table 31. GPU Backend Fabric: BGP Underlay with Peer Auto-Discovery Configuration

STRIPE 1 LEAF 1	SPINE 1
`[edit routing-options]`	`[edit routing-options]`
`jnpr@stripe1-leaf1# show`	`jnpr@spine1# show`
`router-id 10.0.1.1;`	`router-id 10.0.0.1;`
`autonomous-system 201;`	`autonomous-system 101;`
`graceful-restart;`	`graceful-restart;`
`forwarding-table {`	`forwarding-table {`
`  export PFE-LB;`	`  export PFE-LB;`
`  ecmp-fast-reroute;`	`  ecmp-fast-reroute;`
`}`	`}`
`[edit policy-options]`	`[edit policy-options]`
`jnpr@stripe1-leaf1# show | match as-list`	`jnpr@stripe1-leaf1# show | match as-list`
`as-list discovered-as-list members 101-104;`	`as-list discovered-as-list members 201-216;`
`[edit protocols bgp group l3clos-inet6-auto-underlay]`	`[edit protocols bgp group l3clos-inet6-auto-underlay]`
`jnpr@stripe1-leaf1# show`	`jnpr@stripe1-leaf1# show`
`  type external;`	`  type external;`
`  family inet {`	`  family inet {`
`    unicast;`	`    unicast;`
`    extended-nexthop;`	`    extended-nexthop;`
`  }`	`  }`
`  export (LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy);`	`  export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );`
`  local-as 201;`	`  local-as 101;`
`  multipath {`	`  multipath {`
`    multiple-as;`	`    multiple-as;`
`  }`	`  }`
`  bfd-liveness-detection {`	`  bfd-liveness-detection {`
`    minimum-interval 3000;`	`    minimum-interval 3000;`
`    multiplier 3;`	`    multiplier 3;`
`  }`	`  }`
`  dynamic-neighbor underlay-dynamic-neighbors {`	`  dynamic-neighbor underlay-dynamic-neighbors {`
`    peer-auto-discovery {`	`    peer-auto-discovery {`
`      family inet6 {`	`      family inet6 {`
`         ipv6-nd;`	`         ipv6-nd;`
`      }`	`      }`
`      /* SPINE 1 */`	`      /* LEAF1 1 */`
`      interface et-0/0/0:0.0;`	`      interface et-0/0/0:0.0;`
`      /* SPINE 2 */`	`      /* LEAF1 2 */`
`      interface et-0/0/1:0.0;`	`      interface et-0/0/1:0.0;`
`      /* SPINE 3 */`	`      /* LEAF1 3 */`
`      interface et-0/0/32:0.0;`	`      interface et-0/0/2:0.0;`
`      /* SPINE 3 */`	`      .`
`      interface et-0/0/33:0.0;`	`      .`
`    }`	`      .`
`  }`	`    }`
`  peer-as-list discovered-as-list;`	`  }`
	`  peer-as-list discovered-as-list;`

To configure peer auto discovery, the dynamic-neighbor named **underlay-dynamic-neighbors,** under **BGP group l3clos-inet6-auto-underlay,** specifies the interfaces where auto discovery is permitted. This replaces the neighbor a.b.c.d commands that would statically configure the neighbors.

The family inet unicast and family inet6 unicast statements configure the sessions to advertise both **IPv4** to support the IPv4 overlay. When BGP sessions are established over IPv6 link-local addresses but carry IPv4 routes (IPv4 overlay), the **extended-nexthop** statement must be configured under family inet unicast. This allows IPv4 next-hops to be resolved across an IPv6 transport session, enabling correct installation of IPv4 prefixes in the routing table as described in RFC5549. Failing to include the **extended-nexthop** will result in hidden routes, as the protocol next-hop cannot be resolved.

The **family inet6 IPv6-nd** statement enables the use of **IPv6 Neighbor Discovery** to dynamically determine the addresses of neighbors with which to establish BGP sessions. To control and secure dynamic peer formation, a **peer-as-list** (discovered-as-list) is configured, restricting peering to neighbors whose autonomous system numbers fall within the defined range of **AS 101–104**.

The BGP sessions are also configured with **multipath multiple-as**, allowing multiple paths (even with different AS paths) to be considered for **ECMP (Equal-Cost Multi-Path)** routing. **BFD (Bidirectional Forwarding Detection)** is additionally enabled to accelerate convergence in case of link or neighbor failures.

You can check that the sessions have been established using: `show bgp summary group <group-name>`

Example:

```
jnpr@stripe1-leaf1> show bgp summary group l3clos-inet6-auto-underlay
fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0         102        201        196       0       0
1:29:35 Establ
  inet.0: 4/4/4/0
fe80::5a86:70ff:fe79:3d5%et-0/0/33:0.0          104        201        196       0       0
1:29:15 Establ
  inet.0: 4/4/4/0
fe80::5a86:70ff:fe7b:ced5%et-0/0/32:0.0         103        201        196       0       0
1:29:21 Establ
  inet.0: 4/4/4/0
fe80::9e5a:80ff:feef:a28f%et-0/0/30:0.0         101        202        197       0       0
1:29:30 Establ
  inet.0: 4/4/4/0
```

Notice that when BGP sessions are established using link-local addresses Junos displays the neighbor address along with the interface scope (e.g. **fe80::5a86:70ff:fe78:e0d5%et-0/0/1:0.0**). The scope identifier (the part after the %) is necessary because the same link-local address (fe80::/10) could exist on multiple interfaces. The device must know which interface to use to send packets to that neighbor. Thus, after peer discovery is completed, the **show bgp summary** output lists the neighbor using the format: **IPv6_link-local_address%interface-name.**

Even though, the sessions are established using the IPv6 link-local addresses the advertised routes are IPv4 and installed in the inet.0 routing table.

You can check details about discovered neighbors using: `show bgp neighbor auto-discovered <peer-id>`

Example:

```
jnpr@stripe1-leaf1> show bgp neighbor auto-discovered fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0
Peer: fe80::5a86:70ff:fe78:e0d5%et-0/0/31:0.0+179 AS 102 Local:
fe80::9e5a:80ff:fec1:ae08%et-0/0/31:0.0+53984 AS 201
  Group: l3clos-inet-auto-underlay Routing-Instance: master
  Forwarding routing-instance: master
  Type: External    State: Established    Flags: <Sync PeerAsList AutoDiscoveredNdp>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ (LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy) ]
  Options: <GracefulRestart AddressFamily Multipath LocalAS Refresh>
  Options: <MultipathAs BfdEnabled>
  Options: <GracefulShutdownRcv>
  Address families configured: inet-unicast
  Holdtime: 90 Preference: 170
  Graceful Shutdown Receiver local-preference: 0
  Local AS: 201 Local System AS: 201
  Number of flaps: 0
  Receive eBGP Origin Validation community: Reject
  Peer ID: 10.0.0.2        Local ID: 10.0.1.1          Active Holdtime: 90
  Keepalive Interval: 30         Group index: 0    Peer index: 0    SNMP index: 30
  I/O Session Thread: bgpio-0 State: Enabled
  BFD: enabled, up
  Local Interface: et-0/0/1:0.0
  NLRI for restart configured on peer: inet-unicast
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Restart time configured on the peer: 120
  Stale routes from peer are kept for: 300
  Restart time requested by this peer: 120
  Restart flag received from the peer: Notification
  NLRI that peer supports restart for: inet-unicast
  NLRI peer can save forwarding state: inet-unicast
  NLRI that peer saved forwarding for: inet-unicast
  NLRI that restart is negotiated for: inet-unicast
  NLRI of received end-of-rib markers: inet-unicast
  NLRI of all end-of-rib markers sent: inet-unicast
  Peer does not support LLGR Restarter functionality
  Peer supports 4 byte AS extension (peer-as 102)
  Peer does not support Addpath
  NLRI(s) enabled for color nexthop resolution: inet-unicast
  Table inet.0 Bit: 20000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:              4
    Received prefixes:            4
    Accepted prefixes:            4
    Suppressed due to damping:    0
    Advertised prefixes:          1
  Last traffic (seconds): Received 20   Sent 24   Checked 5788
  Input messages:  Total 216    Updates 5       Refreshes 0     Octets 4535
  Output messages: Total 212    Updates 1       Refreshes 0     Octets 4125
  Output Queue[1]: 0            (inet.0, inet-unicast)
  Trace options:  all
  Trace file: /var/log//bgp size 131072 files 10
```

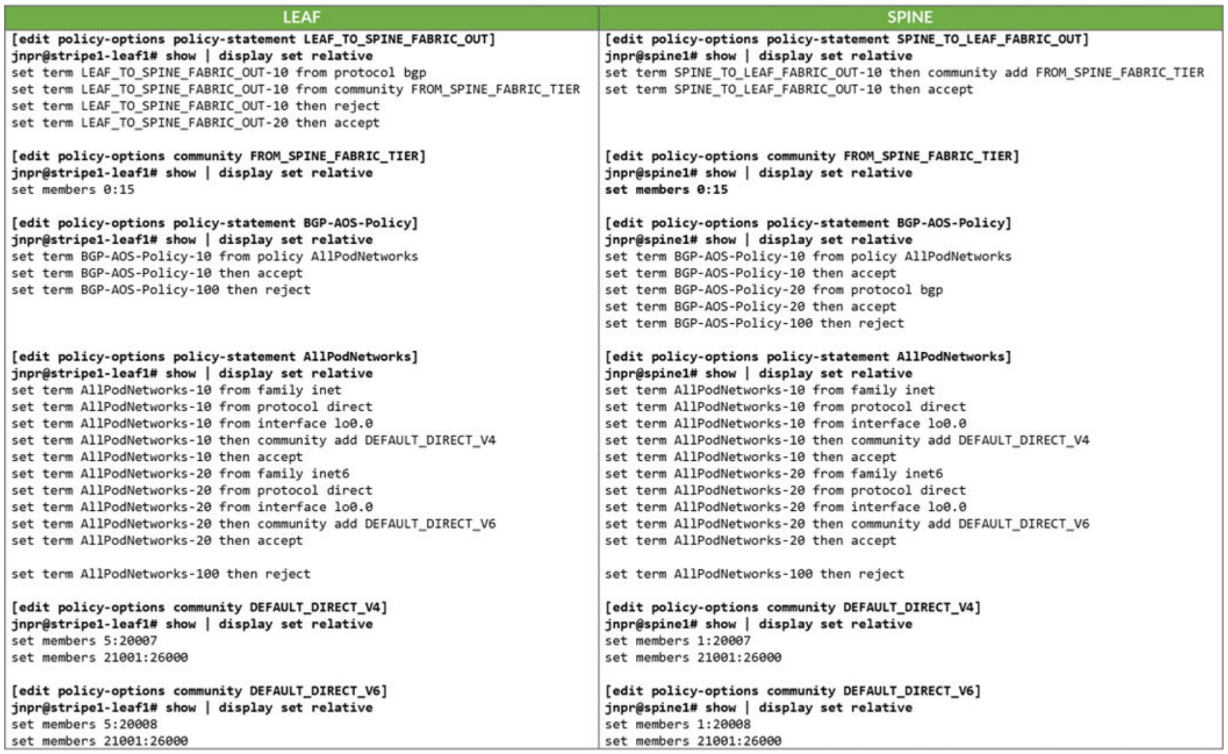
To verify the operation of BFD for the BGP sessions use: `show bfd session`

Example:

```
jnpr@stripe1-leaf1> show bfd session
                                                 Detect   Transmit
Address                  State     Interface      Time     Interval  Multiplier
fe80::5a86:70ff:fe78:e0d5 Up      et-0/0/31:0.0   9.000     3.000        3
fe80::5a86:70ff:fe79:3d5  Up      et-0/0/33:0.0   9.000     3.000        3
fe80::5a86:70ff:fe7b:ced5 Up      et-0/0/32:0.0   9.000     3.000        3
fe80::9e5a:80ff:feef:a28f Up      et-0/0/30:0.0   9.000     3.000        3
8 sessions, 8 clients
Cumulative transmit rate 2.7 pps, cumulative receive rate 2.7 pps
```

To control the propagation of routes, and make sure the loopback interface addresses are advertised, export policies are applied to these EBGP sessions as shown in the example in Table 32.

Table 32. Export policy example IPv4 Underlay with auto discovery



LEAF	SPINE
[edit policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT]	**[edit policy-options policy-statement SPINE_TO_LEAF_FABRIC_OUT]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term LEAF_TO_SPINE_FABRIC_OUT-10 from protocol bgp	set term SPINE_TO_LEAF_FABRIC_OUT-10 then community add FROM_SPINE_FABRIC_TIER
set term LEAF_TO_SPINE_FABRIC_OUT-10 from community FROM_SPINE_FABRIC_TIER	set term SPINE_TO_LEAF_FABRIC_OUT-10 then accept
set term LEAF_TO_SPINE_FABRIC_OUT-10 then reject	
set term LEAF_TO_SPINE_FABRIC_OUT-20 then accept	
[edit policy-options community FROM_SPINE_FABRIC_TIER]	**[edit policy-options community FROM_SPINE_FABRIC_TIER]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set members 0:15	**set members 0:15**
[edit policy-options policy-statement BGP-AOS-Policy]	**[edit policy-options policy-statement BGP-AOS-Policy]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term BGP-AOS-Policy-10 from policy AllPodNetworks	set term BGP-AOS-Policy-10 from policy AllPodNetworks
set term BGP-AOS-Policy-10 then accept	set term BGP-AOS-Policy-10 then accept
set term BGP-AOS-Policy-100 then reject	set term BGP-AOS-Policy-20 from protocol bgp
	set term BGP-AOS-Policy-20 then accept
	set term BGP-AOS-Policy-100 then reject
[edit policy-options policy-statement AllPodNetworks]	**[edit policy-options policy-statement AllPodNetworks]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term AllPodNetworks-10 from family inet	set term AllPodNetworks-10 from family inet
set term AllPodNetworks-10 from protocol direct	set term AllPodNetworks-10 from protocol direct
set term AllPodNetworks-10 from interface lo0.0	set term AllPodNetworks-10 from interface lo0.0
set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V4	set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V4
set term AllPodNetworks-10 then accept	set term AllPodNetworks-10 then accept
set term AllPodNetworks-20 from family inet6	set term AllPodNetworks-20 from family inet6
set term AllPodNetworks-20 from protocol direct	set term AllPodNetworks-20 from protocol direct
set term AllPodNetworks-20 from interface lo0.0	set term AllPodNetworks-20 from interface lo0.0
set term AllPodNetworks-20 then community add DEFAULT_DIRECT_V6	set term AllPodNetworks-20 then community add DEFAULT_DIRECT_V6
set term AllPodNetworks-20 then accept	set term AllPodNetworks-20 then accept
set term AllPodNetworks-100 then reject	set term AllPodNetworks-100 then reject
[edit policy-options community DEFAULT_DIRECT_V4]	**[edit policy-options community DEFAULT_DIRECT_V4]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set members 5:20007	set members 1:20007
set members 21001:26000	set members 21001:26000
[edit policy-options community DEFAULT_DIRECT_V6]	**[edit policy-options community DEFAULT_DIRECT_V6]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set members 5:20008	set members 1:20008
set members 21001:26000	set members 21001:26000

These policies ensure loopback reachability without advertising unnecessary routes.

On the spine nodes, routes are exported only if they are accepted by both the *SPINE_TO_LEAF_FABRIC_OUT* and *BGP-AOS-Policy* export policies.

- The *SPINE_TO_LEAF_FABRIC_OUT* policy has no match conditions and accepts all routes unconditionally, tagging them with the **FROM_SPINE_FABRIC_TIER** community (0:15).
- The *BGP-AOS-Policy* accepts BGP-learned routes as well as any routes accepted by the nested *AllPodNetworks* policy.
- The *AllPodNetworks* policy, in turn, matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V6** community (1:20008 and 21001:26000 on Spine1).

As a result, each spine advertises both its directly connected routes (including its loopback interface) and any routes it has received from other leaf nodes.

You can verify that the expected routes are being advertised by the spine node using: `show route advertising-protocol bgp <peer-id> table inet6.0`

Example:

The following example shows the routes advertised to Stripe 1 Leaf 1 by Spine 1 which correspond to the loopback interface addresses of itself, as well as Stripe1 Leaf 2, Stripe 2 Leaf 1, and Stripe 2 Leaf 2.

```
jnpr@spine1> show route advertising-protocol bgp fe80::9e5a:80ff:fec1:ae00%et-0/0/30:0.0 table
inet.0
inet4.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.1/32             Self                                    I
* 10.0.1.2/32             Self                                    202 I
* 10.0.1.9/32             Self                                    209 I
* 10.0.1.10/32            Self                                    210 I
```

To verify routes are received by the Leaf nodes use: `show route receive-protocol bgp <peer-id> table inet6.0`

Example:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp fe80::5a86:70ff:fe78:e0d5%et-0/0/1:0.0 table
inet.0
inet6.0: 14 destinations, 23 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.1/32           fe80::9e5a:80ff:feef:a28f               101 I
  10.0.0.2/32           fe80::9e5a:80ff:feef:a28f               101 202 I
  10.0.0.9/32           fe80::9e5a:80ff:feef:a28f               101 209 I
  10.0.0.10/32          fe80::9e5a:80ff:feef:a28f               101 210 I
```

On the leaf nodes, routes are exported only if they are accepted by both the LEAF_TO_SPINE_FABRIC_OUT and BGP-AOS-Policy export policies.

- The *LEAF_TO_SPINE_FABRIC_OUT* policy accepts all routes except those learned via BGP that are tagged with the **FROM_SPINE_FABRIC_TIER** community (0:15). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.

- The *BGP-AOS-Policy* accepts all routes allowed by the nested *AllPodNetworks* policy, which matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V4** community (5:20007 and 21001:26000 for Stripe1-Leaf1).

As a result, leaf nodes will advertise only their directly connected interface routes, including their loopback interfaces, to the spines.

You can verify that the expected routes are being advertised by the spine node using: `show route advertising-protocol bgp <peer-id> table inet6.0`

Example:

The following example shows the routes advertised to Spine 1 by Stripe 1 Leaf 1.

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp fe80::5a86:70ff:fe78:e0d5%et-0/0/30:0.0
table inet6.0
inet6.0: 14 destinations, 23 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.1/32             Self                                    I
```

To verify routes are received by the spine node, use: `show route receive-protocol bgp <peer-id> table inet6.0`

Example:

```
jnpr@spine1> show route receive-protocol bgp fe80::9e5a:80ff:fec1:ae00%et-0/0/0:0.0 table
inet6.0
inet6.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                  Nexthop              MED     Lclpref    AS path
* 10.0.0.1/32             fe80::9e5a:80ff:fec1:ae00               201 I
```

IPv6 GPU Backend Fabric Overlay

GPU Backend Fabric Overlay Using IPv4 The **overlay EBGP sessions** are configured between the leaf and spine nodes using the IPv4 addresses of the loopback interfaces, as shown in the example between Stripe1 Leaf 1/Stripe 2 Leaf 1 and Spine 1.

Table 33. GPU Backend Fabric Overlay Using IPv4 Loopback Addresses – Stripe 1 Example

STRIPE 1 LEAF 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet4-auto-overlay {
    type external;
    multihop {
        ttl 1;
    }
    family inet {
        unicast;
    }
    export direct;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.0.1 {
        description facing_spine1-evpn-overlay;
        local-address 10.0.1.1;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
edit]
jnpr@spine1# show protocols bgp
group l3clos-inet4-auto-overlay {
    type external;
    traceoptions {
        file bgp;
        flag all;
    }
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.1.1 {
        description facing_leaf1-evpn-overlay;
        local-address 10.0.0.1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 201;
    }
    .
    .
    .
    vpn-apply-export;
}
```

Table 34. GPU Backend Fabric Overlay Using IPv4 Loopback Addresses – Stripe 2 Example

STRIPE 2 LEAF 1

```
[edit]
jnpr@stripe2-leaf1# show protocols bgp
group l3clos-inet4-auto-overlay {
    type external;
    multihop {
        ttl 1;
    }
    family inet {
        unicast;
    }
    export direct;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.0.1 {
        description facing_spine1-evpn-overlay;
        local-address 10.0.1.9;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
edit]
jnpr@spine1# show protocols bgp
group l3clos-inet4-auto-overlay {
    type external;
    traceoptions {
        file bgp;
        flag all;
    }
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.1.9 {
        description facing_leaf1-evpn-overlay;
        local-address 10.0.0.1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 209;
    }
    .
    .
    .
    vpn-apply-export;
}
```

The overlay BGP sessions use family evpn signaling to enable EVPN route exchange. The **multihop ttl 1** statement allows EBGP sessions to be established between the loopback interfaces.

As with the underlay BGP sessions, these sessions are configured with **multipath multiple-as**, allowing multiple EVPN paths with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing. BFD (Bidirectional Forwarding Detection) is also enabled to improve convergence time in case of failures.

The **no-nexthop-change** knob on the spine nodes is used to preserve the original next-hop address, which is critical in EVPN for ensuring that the remote VTEP can be reached directly. The **vpn-apply-export statement** is included to ensure that the export policies are evaluated for VPN address families, such as EVPN, allowing fine-grained control over which routes are advertised to each peer.

To control the propagation of routes, export policies are applied to these EBGP sessions as shown in the example in table 33.

Table 35. Export Policy example to advertise EVPN routes over IPv4 overlay

LEAF	SPINE
[edit policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT] jnpr@stripe1-leaf1# show | display set relative set term LEAF_TO_SPINE_EVPN_OUT-10 from protocol bgp set term LEAF_TO_SPINE_EVPN_OUT-10 from community FROM_SPINE_EVPN_TIER set term LEAF_TO_SPINE_EVPN_OUT-10 then reject set term LEAF_TO_SPINE_EVPN_OUT-20 then accept **[edit policy-options community FROM_SPINE_EVPN_TIER]** **jnpr@stripe1-leaf1# show | display set relative** set members 0:14 **[edit policy-options policy-statement EVPN_EXPORT]** **jnpr@stripe1-leaf1# show | display set relative** set term EVPN_EXPORT-4095 then accept	**[edit policy-options policy-statement SPINE_TO_LEAF_EVPN_OUT]** **jnpr@spine1# show | display set relative** set term SPINE_TO_LEAF_EVPN_OUT-10 then community add FROM_SPINE_EVPN_TIER set term SPINE_TO_LEAF_EVPN_OUT-10 then accept **[edit policy-options community FROM_SPINE_EVPN_TIER]** **jnpr@spine1# show | display set relative** set members 0:14

These policies are simpler in structure and are intended to enable end-to-end EVPN reachability between tenant GPUs, while preventing route loops within the overlay.

Routes will only be advertised if EVPN routing-instances have been created. Example:

Table 36. EVPN Routing-Instances for a single tenant example across different leaf nodes.

STRIPE 1 – LEAF 1	STRIPE 2 – LEAF 1
[edit routing-instances Tenant-A] **jnpr@stripe1-leaf1# show | display set relative** set instance-type vrf set routing-options graceful-restart set routing-options multipath set protocols evpn ip-prefix-routes advertise direct-nexthop set protocols evpn ip-prefix-routes encapsulation vxlan set protocols evpn ip-prefix-routes vni 20001 set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A set interface et-0/0/12:0.0 set interface lo0.1 set route-distinguisher 10.0.1.1:2001 set vrf-target target:20001:1	**[[edit routing-instances Tenant-A]** **jnpr@stripe2-leaf1# show | display set relative** set instance-type vrf set routing-options graceful-restart set routing-options multipath set protocols evpn ip-prefix-routes advertise direct-nexthop set protocols evpn ip-prefix-routes encapsulation vxlan set protocols evpn ip-prefix-routes vni 20009 set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A set interface et-0/0/12:0.0 set interface lo0.1 set route-distinguisher 10.0.1.9:2009 set vrf-target target:20001:1

On the **spine nodes**, routes are exported if they are accepted by the ***SPINE_TO_LEAF_EVPN_OUT*** policy.

The ***SPINE_TO_LEAF_EVPN_OUT*** policy has no match conditions and accepts all routes. It tags each exported route with the **FROM_SPINE_EVPN_TIER** community (0:14).

As a result, the spine nodes export EVPN routes received from one leaf to all other leaf nodes, allowing tenant-to-tenant communication across the fabric.

Example:

```
jnpr@spine1> show route advertising-protocol bgp 10.0.1.1 | match 5:10.*2001.*31
  5:10.0.1.2:2001::0::10.200.0.2::31/248
  5:10.0.1.2:2001::0::10.200.0.34::31/248
  5:10.0.1.9:2001::0::10.200.1.0::31/248
  5:10.0.1.9:2001::0::10.200.1.32::31/248
  5:10.0.1.10:2001::0::10.200.1.2::31/248
  5:10.0.1.10:2001::0::10.200.1.34::31/248
jnpr@spine1>show route advertising-protocol bgp 10.0.1.1 match-prefix
5:10.0.1.9:2001::0::10.200.1.0::31/248
bgp.evpn.0: 378 destinations, 378 routes (378 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop             MED     Lclpref
AS path
  5:10.0.1.9:2001::0::10.200.1.0::31/248        * 10.0.1.9                             209 I
```

On the **leaf nodes**, routes are exported if they are accepted by both the ***LEAF_TO_SPINE_EVPN_OUT*** and ***EVPN_EXPORT*** policies:

- The ***LEAF_TO_SPINE_EVPN_OUT*** policy rejects any BGP-learned routes that carry the **FROM_SPINE_EVPN_TIER** community (0:14). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.
- The ***EVPN_EXPORT*** policy accepts all routes without additional conditions.

As a result, the leaf nodes export only locally originated EVPN routes for the directly connected interfaces between GPU servers and the leaf nodes. These routes are part of the **tenant routing instances** and are required to establish reachability between GPUs belonging to the same tenant.

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 table Tenant-1
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                          Nexthop         MED     Lclpref         AS path
  5:10.0.1.1:2001::0::10.200.0.0::31/248
*                                 Self                    I
  5:10.0.1.1:2001::0::10.200.0.16::31/248
*                                 Self                    I
jnpr@stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 table Tenant-2
```

```
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                        Nexthop MED     Lclpref    AS path
  5:10.0.1.1:2002::0::10.200.0.2::31/248
*                               Self                    I
  5:10.0.1.1:2002::0::10.200.0.18::31/248
*                               Self                    I
```

Appendix B – IPv4 Overlay over IPv4 Underlay Fabric Implementation

IN THIS SECTION

This section outlines the configuration components for an IPv4 underlay and IPv4 overlay deployment.

Control plane implementation with IPv4 underlay and IPv4 overlay

This model provides an IPv4 transport underlay and IPv4 EVPN/VXLAN transport in the overlay that can support IPv4-only devices communicating across the fabric. This model aligns with traditional IP fabric designs, where interface addressing is fully controlled and visible, neighbor relationships are explicitly defined, and support IPv4-only end devices.

The interfaces between leaf and spine nodes are configured with explicit /31 IPv4 addresses assigned from a pool of IPv4 addresses reserved for the underlay. Each device on the point-to-point link is configured with one of the two usable IPv4 addresses in the corresponding /31 subnet. This allows efficient address assignments for the point-to-point links between leaf and spine nodes. All leaf and spine nodes are also configured with IPv4 addresses on the loopback interface (lo0.0).

The **underlay EBGP** sessions are set up between the leaf and spine nodes, by explicitly configuring each neighbor, using the /31 IPv4 addresses assigned between them.

The EBGP configuration for this model includes each neighbor's IPv4 address and Autonomous System (AS) number, the local Autonomous System (AS) number, and the export policy that allows the advertisement of routes to reach all the leaf and spine nodes in the fabric. These routes are standard IPv4 unicast advertising the IPv4 addresses assigned to the loopback interface (lo0.0).

The **overlay EBGP** sessions are also set up by explicitly configuring each neighbor, using the IPv4 addresses of the loopback interfaces advertised by the underlay EBGP sessions, and are also established between the leaf and spine nodes.

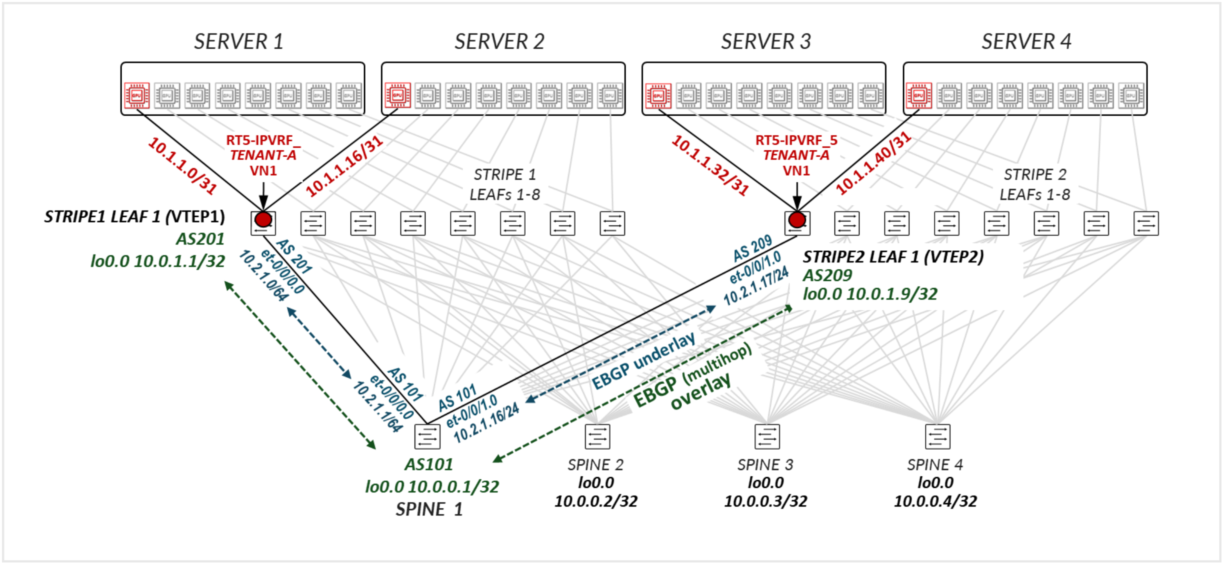
The leaf nodes act as VTEPs and advertise the IPv4 prefixes assigned to the links between the GPU servers and the leaf nodes using EVPN Type 5 routes.

Example:

Consider the example depicted in Figure 55.

For the underlay, STRIPE1 LEAF 1 in AS 201 establishes an EBGP session with SPINE 1 in AS 101, over the directly connected IPv4 link 10.2.1.2/31 <=> 10.2.1.1/31. Similarly, STRIPE2 LEAF 1 in AS 209 establishes an EBGP session with SPINE 1 over the link 10.2.9.2/31 <=> 10.2.9.1/31.

Figure 55: IPv4 Underlay and IPv4 Overlay Example



These sessions exchange IPv4 unicast routes advertising the address of the loopback interface (lo0.0) of STRIPE1 LEAF 1 (10.0.1.1), STRIPE2 LEAF 1 (10.0.1.9) and SPINE 1 (10.0.0.1).

NOTE: Although it is not shown in the diagram, STRIPE1 LEAF 1 and STRIPE2 LEAF 1 will also establish EBGP sessions with SPINE 2, SPINE 3, and SPINE 4 to ensure multiple paths are available for traffic.

EBGP sessions are established between the leaf nodes and SPINE 1 using their loopback addresses (10.0.1.1, 10.0.1.9, and 10.0.0.1, respectively).

The leaf nodes acting as VTEP advertise the links connecting the GPU servers and leaf nodes as /31 EVPN type 5 routes.

For example, STRIPE1 LEAF 1 advertises routes to the IPv4 addresses on the links connecting SERVER 1 GPU1 and SERVER 2 GPU1 to STRIPE1 LEAF 1 (10.1.1.0/31 and 10.1.1.16/31 respectively). Similarly, STRIPE2 LEAF 1 advertises router to the IPv4 addresses on the links connecting SERVER 3 GPU1 and SERVER 4 GPU1 (10.1.1.32/31 and 10.1.1.40/31 respectively).

Assuming all four GPUs in the example belong to the same tenant, their associated interfaces are mapped to the same VRF, RT5-IP-VRF_TENANT-1.

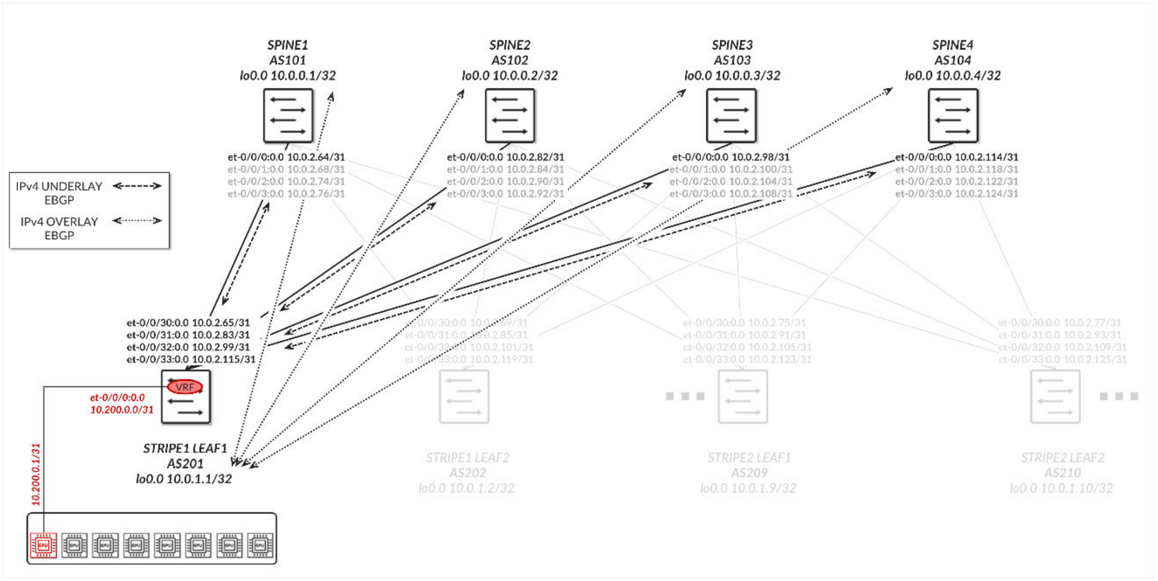
RT5-IP-VRF_TENANT-1 is configured on both STRIPE1 LEAF 1 and STRIPE2 LEAF 1 with the same VXLAN Network Identifier (VNI) and route targets. STRIPE1 LEAF 1 advertises the prefixes 10.1.1.0/31 and 10.1.1.16/31 to SPINE 1 as EVPN Route Type 5, with its own loopback (10.0.1.1) as the next-hop VTEP. STRIPE2 LEAF 1 advertises 10.1.1.32/31 and 10.1.1.40/31 with 10.0.1.9 as the next-hop.

When SERVER 1 GPU1 sends traffic to SERVER 3 GPU1, the destination addresses 10.1.1.32 for example, is found in the VRF routing table on STRIPE1 LEAF 1 (Tenant-1.inet.0). The route points to STRIPE2 LEAF 1 (VTEP at 10.0.1.9) as the protocol next-hop (which is resolved to the addresses of the spine nodes). The route also specifies VNI 1 as the VXLAN encapsulation ID. The packet is encapsulated with the VXLAN header and tunneled across the fabric to its destination.

Spine Nodes to Leaf Connections

The interfaces between the leaf and spine nodes do not require explicitly configured IP addresses and are configured as untagged interfaces with only family inet and family inet6 to enable processing of IPv4 and IPv6 traffic as shown in Figure 56.

Figure 56: IPv4 Underlay and IPv4 Overlay Configuration Example


SPINE1
AS101
lo0.0 10.0.0.1/32
SPINE2
AS102
lo0.0 10.0.0.2/32
SPINE3
AS103
lo0.0 10.0.0.3/32
SPINE4
AS104
lo0.0 10.0.0.4/32
et-0/0/0:0.0 10.0.2.64/31
et-0/0/1:0.0 10.0.2.68/31
et-0/0/2:0.0 10.0.2.74/31
et-0/0/3:0.0 10.0.2.76/31
et-0/0/0:0.0 10.0.2.82/31
et-0/0/1:0.0 10.0.2.84/31
et-0/0/2:0.0 10.0.2.90/31
et-0/0/3:0.0 10.0.2.92/31
et-0/0/0:0.0 10.0.2.98/31
et-0/0/1:0.0 10.0.2.100/31
et-0/0/2:0.0 10.0.2.104/31
et-0/0/3:0.0 10.0.2.108/31
et-0/0/0:0.0 10.0.2.114/31
et-0/0/1:0.0 10.0.2.118/31
et-0/0/2:0.0 10.0.2.122/31
et-0/0/3:0.0 10.0.2.124/31
IPv4 UNDERLAY EBGP
IPv4 OVERLAY EBGP
et-0/0/30:0.0 10.0.2.65/31
et-0/0/31:0.0 10.0.2.83/31
et-0/0/32:0.0 10.0.2.99/31
et-0/0/33:0.0 10.0.2.115/31
et-0/0/30:0.0 10.0.2.69/31
et-0/0/31:0.0 10.0.2.85/31
et-0/0/32:0.0 10.0.2.101/31
et-0/0/33:0.0 10.0.2.119/31
et-0/0/30:0.0 10.0.2.75/31
et-0/0/31:0.0 10.0.2.91/31
et-0/0/32:0.0 10.0.2.105/31
et-0/0/33:0.0 10.0.2.123/31
et-0/0/30:0.0 10.0.2.77/31
et-0/0/31:0.0 10.0.2.93/31
et-0/0/32:0.0 10.0.2.109/31
et-0/0/33:0.0 10.0.2.125/31
VRF
et-0/0/0:0.0
10.200.0.0/31
10.200.0.1/31
STRIPE1 LEAF1
AS201
lo0.0 10.0.1.1/32
STRIPE1 LEAF2
AS202
lo0.0 10.0.1.2/32
STRIPE2 LEAF1
AS209
lo0.0 10.0.1.9/32
STRIPE2 LEAF2
AS210
lo0.0 10.0.1.10/32
GPU

The interfaces between the leaf and spine nodes are configured with /31 addresses as shown in Table 58.

Table 58. IPv4 Address Assignments for Leaf-to-Spine Interfaces (/31 Subnetting)

LEAF NODE INTERFACE	LEAF NODE IPv4 ADDRESS	SPINE NODE INTERFACE	SPINE IPv4 ADDRESS
Stripe 1 Leaf 1 - et-0/0/30:0	10.0.2.65/31	Spine 1 – et-0/0/0:0	10.0.2.64/31
Stripe 1 Leaf 1 - et-0/0/31:0	10.0.2.83/31	Spine 2 – et-0/0/1:0	10.0.2.82/31
Stripe 1 Leaf 1 - et-0/0/32:0	10.0.2.99/31	Spine 3 – et-0/0/2:0	10.0.2.98/31
Stripe 1 Leaf 1 - et-0/0/33:0	10.0.2.115/31	Spine 4 – et-0/0/3:0	10.0.2.114/31
Stripe 1 Leaf 5 - et-0/0/30:0	10.0.2.69/31	Spine 1 – et-0/0/0:0	10.0.2.68/31
Stripe 1 Leaf 2 - et-0/0/31:0	10.0.2.85/31	Spine 2 – et-0/0/1:0	10.0.2.84/31
Stripe 1 Leaf 2 - et-0/0/32:0	10.0.2.101/31	Spine 3 – et-0/0/2:0	10.0.2.100/31
Stripe 1 Leaf 2 - et-0/0/33:0	10.0.2.119/31	Spine 4 – et-0/0/3:0	10.0.2.118/31
. . .			

These interfaces are configured as untagged interfaces, with family inet and static IPv4 addresses, as shown in the example for the link between Stripe 1 leaf 1 and Spine 1 below:

Table 59. Example Junos Configuration for Leaf-Spine IPv4 Interface

STRIPE 1 LEAF 1 (et-0/0/0:30)

```
[edit interfaces et-0/0/30]
jnpr@stripe1-leaf1# show | display set relative
set description "Breakout et-0/0/30"
set number-of-sub-ports 1
set speed 800g

[edit interfaces et-0/0/30:0]
jnpr@stripe1-leaf1# show | display set relative
set description facing_spine1:et-0/0/0:0
set mtu 9216
set unit 0 family inet mtu 9170
set unit 0 family inet address 10.0.2.64/31
```

SPINE 1 (et-0/0/0:0)

```
[edit interfaces et-0/0/0]
jnpr@spine1# show | display set
set interfaces et-0/0/0 description "Breakout et-0/0/0"
set interfaces et-0/0/0 number-of-sub-ports 1
set interfaces et-0/0/0 speed 800g

[edit interfaces et-0/0/0:0]
jnpr@spine1# show | display set relative
set description "To Leaf1"
set mtu 9216
set unit 0 family inet mtu 9170
set unit 0 family inet address 10.0.2.65/31
```

The loopback and Autonomous System numbers for all devices in the fabric are included in Table 60:

Table 60. Loopback IPv4 Addresses and Autonomous System Numbers for Fabric Devices

LEAF NODE INTERFACE	lo0.0 IPv4 ADDRESS	Local AS #
Stripe 1 Leaf 1	10.0.1.1/32	201
Stripe 1 Leaf 2	10.0.1.2/32	202
Stripe 1 Leaf 3	10.0.1.3/32	203
Stripe 1 Leaf 4	10.0.1.4/32	204
Stripe 1 Leaf 5	10.0.1.5/32	205
Stripe 1 Leaf 6	10.0.1.6/32	206
Stripe 1 Leaf 7	10.0.1.7/32	207
Stripe 1 Leaf 8	10.0.1.8/32	208
. . .		
SPINE1	10.0.0.1/32	101
SPINE2	10.0.0.2/32	102
SPINE3	10.0.0.3/32	103

(Continued)

LEAF NODE INTERFACE	lo0.0 IPv4 ADDRESS	Local AS #
SPINE4	10.0.0.4/32	104

Table 61. Example Junos Configuration for Loopback Interfaces and Routing Options

STRIPE 1 LEAF 1

```
[edit routing-options]
jnpr@stripe1-leaf1# show | display set relative
set router-id 10.0.1.1
set autonomous-system 201
set graceful-restart
set forwarding-table export PFE-LB
set forwarding-table ecmp-fast-reroute

[edit interfaces lo0]
jnpr@stripe1-leaf1# show |display set relative
set unit 0 family inet address 10.0.1.1/32
```

SPINE 1

```
[edit routing-options]
jnpr@spine1# show | display set
set router-id 10.0.0.1
set autonomous-system 101
set graceful-restart
set forwarding-table export PFE-LB
set forwarding-table ecmp-fast-reroute

[edit interfaces lo0]
jnpr@stripe1-leaf1# show |display set relative
set unit 0 family inet address 10.0.0.1/32
```

GPU Backend Fabric Underlay with IPv4

The underlay EBGP sessions are configured between the leaf and spine nodes using the IP addresses of the directly connected links, as shown in the example between Stripe1 Leaf 1 and the spine nodes below:

Table 62. EBGP Underlay Configuration Example: Stripe 1 Leaf 1 to Spine 1

STRIPE 1 LEAF 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor 10.0.2.64 {
        description facing_spine1;
        local-address 10.0.2.65;
        family inet {
            unicast;
        }
        export ( LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 108;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor 10.0.2.65 {
        description facing_stripe1-leaf1;
        local-address 10.0.2.64;
        family inet {
            unicast;
        }
        export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 208;
    }
    .
    .
    .
    vpn-apply-export;
}
```

Table 63. EBGP Underlay Configuration Example: Stripe 1 Leaf 1 to Spine 2

STRIPE 1 LEAF 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor 10.0.2.82 {
        description facing_spine1;
        local-address 10.0.2.83;
        family inet {
            unicast;
        }
        export ( LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 108;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 2

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor 10.0.2.83 {
        description facing_stripe1-leaf1;
        local-address 10.0.2.82;
        family inet {
            unicast;
        }
        export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 208;
    }
    .
    .
    .
    vpn-apply-export;
}
```

All the BGP sessions are configured with `multipath multiple-as`, which allows multiple paths (to the same destination) with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing, and with **BFD** to improve convergence in case of failures.

To control the propagation of routes, export policies are applied to these EBGP sessions as shown in the example in Table 64.

Table 64. Export policy example to advertise IPv4 routes over IPv4 Underlay

LEAF

```
[edit policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT]
jnpr@stripe1-leaf1# show | display set relative
set term LEAF_TO_SPINE_FABRIC_OUT-10 from protocol bgp
set term LEAF_TO_SPINE_FABRIC_OUT-10 from community FROM_SPINE_FABRIC_TIER
set term LEAF_TO_SPINE_FABRIC_OUT-10 then reject
set term LEAF_TO_SPINE_FABRIC_OUT-20 then accept

[edit policy-options community FROM_SPINE_FABRIC_TIER]
jnpr@stripe1-leaf1# show | display set relative
set members 0:15

[edit policy-options policy-statement BGP-AOS-Policy]
jnpr@stripe1-leaf1# show | display set relative
set term BGP-AOS-Policy-10 from policy AllPodNetworks
set term BGP-AOS-Policy-10 then accept
set term BGP-AOS-Policy-100 then reject


[edit policy-options policy-statement AllPodNetworks]
jnpr@stripe1-leaf1# show | display set relative
set term AllPodNetworks-10 from family inet
set term AllPodNetworks-10 from protocol direct
set term AllPodNetworks-10 from interface lo0.0
set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V4
set term AllPodNetworks-10 then accept
set term AllPodNetworks-100 then reject

[edit policy-options community DEFAULT_DIRECT_V4]
jnpr@stripe1-leaf1# show | display set relative
set members 5:20007
set members 21001:26000
```

SPINE

```
[edit policy-options policy-statement SPINE_TO_LEAF_FABRIC_OUT]
jnpr@spine1# show | display set relative
set term SPINE_TO_LEAF_FABRIC_OUT-10 then community add FROM_SPINE_FABRIC_TIER
set term SPINE_TO_LEAF_FABRIC_OUT-10 then accept


[edit policy-options community FROM_SPINE_FABRIC_TIER]
jnpr@spine1# show | display set relative
set members 0:15

[edit policy-options policy-statement BGP-AOS-Policy]
jnpr@spine1# show | display set relative
set term BGP-AOS-Policy-10 from policy AllPodNetworks
set term BGP-AOS-Policy-10 then accept
set term BGP-AOS-Policy-20 from protocol bgp
set term BGP-AOS-Policy-20 then accept
set term BGP-AOS-Policy-100 then reject

[edit policy-options policy-statement AllPodNetworks]
jnpr@spine1# show | display set relative
set term AllPodNetworks-10 from family inet
set term AllPodNetworks-10 from protocol direct
set term AllPodNetworks-10 from interface lo0.0
set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V4
set term AllPodNetworks-10 then accept
set term AllPodNetworks-100 then reject

[edit policy-options community DEFAULT_DIRECT_V4]
jnpr@spine1# show | display set relative
set members 1:20007
set members 21001:26000
```

These policies ensure loopback reachability is advertised cleanly and without the risk of route loops.

On the spine nodes, routes are exported only if they are accepted by both the *SPINE_TO_LEAF_FABRIC_OUT* and *BGP-AOS-Policy* export policies:

- The *SPINE_TO_LEAF_FABRIC_OUT* policy has no match conditions and accepts all routes unconditionally, tagging them with the **FROM_SPINE_FABRIC_TIER** community (0:15).
- The *BGP-AOS-Policy* accepts BGP-learned routes as well as any routes accepted by the nested *AllPodNetworks* policy.
- The *AllPodNetworks* policy, in turn, matches directly connected IPv4 routes and tags them with the **DEFAULT_DIRECT_V4** community (1:20007 and 21001:26000 on Spine1).

As a result, each spine advertises both its directly connected routes (including its loopback interface) and any routes it has received from other leaf nodes.

Example:

```
jnpr@spine1>  show route advertising-protocol bgp 10.0.2.65 | match /32
* 10.0.0.1/32             Self                                    I
* 10.0.1.2/32             Self                                    202 I
* 10.0.1.3/32             Self                                    203 I
---more---
jnpr@spine1> show route advertising-protocol bgp 10.0.2.65 10.0.0.1/32 extensive
inet.0: 85 destinations, 169 routes (85 active, 0 holddown, 0 hidden)
Restart Complete
* 10.0.0.1/32 (1 entry, 1 announced)
 BGP group l3clos-underlay type External
     Nexthop: Self
     AS path: [101] I
     Communities: 0:15 1:20007 21001:26000
jnpr@spine2> show route advertising-protocol bgp 10.0.2.65 10.0.1.2/32 extensive
inet.0: 85 destinations, 169 routes (85 active, 0 holddown, 0 hidden)
Restart Complete
* 10.0.1.2/32 (2 entries, 1 announced)
 BGP group l3clos-underlay type External
     AS path: [101] 202 I
     Communities: 0:15 6:20007 21001:26000
```

On the **leaf nodes**, routes are exported only if they are accepted by both the *LEAF_TO_SPINE_FABRIC_OUT* and *BGP-AOS-Policy* export policies:

- The *LEAF_TO_SPINE_FABRIC_OUT* policy accepts all routes except those learned via BGP that are tagged with the **FROM_SPINE_FABRIC_TIER** community (0:15). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.

- The *BGP-AOS-Policy* accepts all routes allowed by the nested *AllPodNetworks* policy, which matches directly connected IPv4 routes and tags them with the **DEFAULT_DIRECT_V4** community (5:20007 and 21001:26000 for Stripe1-Leaf1).

As a result, leaf nodes will advertise only their directly connected interface routes, including their loopback interfaces, to the spines.

```
jnpr@stripe1-leaf1>  show route advertising-protocol bgp 10.0.2.64 | match /32
* 10.0.1.1/32             Self                                    I
jnpr@stripe1-leaf1>  show route advertising-protocol bgp 10.0.2.64 10.0.1.1/32 extensive
inet.0: 48 destinations, 257 routes (48 active, 0 holddown, 0 hidden)
Restart Complete
* 10.0.1.1/32 (1 entry, 1 announced)
 BGP group l3clos-underlay type External
     Nexthop: Self
     AS path: [201] I
     Communities: 5:20007 21001:26000
```

GPU Backend Fabric Overlay with IPv4

The **overlay EBGP sessions** are configured between the leaf and spine nodes using the IPv4 addresses of the loopback interfaces, as shown in the example between Stripe1 Leaf 1 and Spines.

Table 65. EVPN Overlay EBGP Configuration Example: Stripe 1 Leaf 1 to Spine 1

STRIPE 1 LEAF 1	SPINE 1

STRIPE 1 LEAF 1

```
[edit protocols bgp]
jnpr@stripe1-leaf1# show
group l3clos-inet-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family evpn {
        signaling {
            loops 2;
        }
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.0.1 {
        description facing_spine1-evpn-overlay;
        local-address 10.0.1.1;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
[edit protocols bgp]
jnpr@spine1# show
group l3clos-inet-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family evpn {
        signaling {
            loops 2;
        }
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.1.1 {
        description facing_stripe1-leaf1-overlay;
        local-address 10.0.0.1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 201;
   }
    .
    .
    .
    vpn-apply-export;
}
```

Table 66. EVPN Overlay EBGP Configuration Example: Stripe 2 Leaf 1 to Spine 1

STRIPE 2 LEAF 1	SPINE 1

STRIPE 2 LEAF 1

```
[edit protocols bgp]
jnpr@stripe1-leaf1# show
group l3clos-inet-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family evpn {
        signaling {
            loops 2;
        }
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.0.1 {
        description facing_spine1-evpn-overlay;
        local-address 10.0.1.9;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
[edit protocols bgp]
jnpr@spine1# show
group l3clos-inet-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family evpn {
        signaling {
            loops 2;
        }
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor 10.0.1.9 {
        description facing_stripe1-leaf1-overlay;
        local-address 10.0.0.1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 209;
   }
    .
    .
    .
    vpn-apply-export;
}
```

The overlay BGP sessions use family evpn signaling to enable EVPN route exchange. The `multihop ttl 1` statement allows EBGP sessions to be established between the loopback interfaces.

As with the underlay BGP sessions, these sessions are configured with `multipath multiple-as`, allowing multiple EVPN paths with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing. BFD (Bidirectional Forwarding Detection) is also enabled to improve convergence time in case of failures.

The `no-nexthop-change` knob is used to preserve the original next-hop address, which is critical in EVPN for ensuring that the remote VTEP can be reached directly. The `vpn-apply-export` statement is included to ensure that the export policies are evaluated for VPN address families, such as EVPN, allowing fine-grained control over which routes are advertised to each peer.

To control the propagation of routes, export policies are applied to these EBGP sessions as shown in the example in Table 67.

Table 67. Export Policy Example to Advertise EVPN Routes over IPv4 Overlay

LEAF	SPINE
[edit policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT]	**[edit policy-options policy-statement SPINE_TO_LEAF_EVPN_OUT]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term LEAF_TO_SPINE_EVPN_OUT-10 from protocol bgp	set term SPINE_TO_LEAF_EVPN_OUT-10 then community add FROM_SPINE_EVPN_TIER
set term LEAF_TO_SPINE_EVPN_OUT-10 from community FROM_SPINE_EVPN_TIER	set term SPINE_TO_LEAF_EVPN_OUT-10 then accept
set term LEAF_TO_SPINE_EVPN_OUT-10 then reject	
set term LEAF_TO_SPINE_EVPN_OUT-20 then accept	
[edit policy-options community FROM_SPINE_EVPN_TIER]	
jnpr@stripe1-leaf1# show | display set relative	**[edit policy-options community FROM_SPINE_EVPN_TIER]**
set members 0:14	**jnpr@spine1# show | display set relative**
	set members 0:14
[edit policy-options policy-statement EVPN_EXPORT]	
jnpr@stripe1-leaf1# show | display set relative	
set term EVPN_EXPORT-4095 then accept	

These policies are simpler in structure and are intended to enable end-to-end EVPN reachability between tenant GPUs, while preventing route loops within the overlay.

Routes will only be advertised if EVPN routing-instances have been created. Example:

Table 68. EVPN Routing-Instances for a Single Tenant Example Across Different Leaf Nodes.

STRIPE 1 – LEAF 1	STRIPE 2 – LEAF 1
[edit routing-instances Tenant-A]	**[[edit routing-instances Tenant-A]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe2-leaf1# show | display set relative**
set instance-type vrf	set instance-type vrf
set routing-options graceful-restart	set routing-options graceful-restart
set routing-options multipath	set routing-options multipath
set protocols evpn ip-prefix-routes advertise direct-nexthop	set protocols evpn ip-prefix-routes advertise direct-nexthop
set protocols evpn ip-prefix-routes encapsulation vxlan	set protocols evpn ip-prefix-routes encapsulation vxlan
set protocols evpn ip-prefix-routes vni 20001	set protocols evpn ip-prefix-routes vni 20009
set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A	set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A
set interface et-0/0/12:0.0	set interface et-0/0/12:0.0
set interface lo0.1	set interface lo0.1
set route-distinguisher 10.0.1.1:2001	set route-distinguisher 10.0.1.9:2009
set vrf-target target:20001:1	set vrf-target target:20001:1

On the **spine nodes**, routes are exported if they are accepted by the ***SPINE_TO_LEAF_EVPN_OUT*** policy:

- The ***SPINE_TO_LEAF_EVPN_OUT*** policy has no match conditions and accepts all routes. It tags each exported route with the **FROM_SPINE_EVPN_TIER** community (0:14).

As a result, the spine nodes export EVPN routes received from one leaf to all other leaf nodes, allowing tenant-to-tenant communication across the fabric.

Example:

```
jnpr@spine1>  show route advertising-protocol bgp 10.0.1.1 | match 5:10.*2001.*31
  5:10.0.1.2:2001::0::10.200.0.2::31/248
  5:10.0.1.2:2001::0::10.200.0.66::31/248
  5:10.0.1.9:2001::0::10.200.1.0::31/248
  5:10.0.1.9:2001::0::10.200.1.64::31/248
  5:10.0.1.10:2001::0::10.200.1.2::31/248
  5:10.0.1.10:2001::0::10.200.1.66::31/248
jnpr@spine1>  show route advertising-protocol bgp 10.0.1.1 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248
bgp.evpn.0: 378 destinations, 378 routes (378 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop             MED     Lclpref
AS path
bgp.evpn.0: 20 destinations, 20 routes (20 active, 0 holddown, 0 hidden)
  5:10.0.1.2:2001::0::10.200.0.2::31/248
*  10.0.1.2                                             202 I
```

On the **leaf nodes**, routes are exported if they are accepted by both the ***LEAF_TO_SPINE_EVPN_OUT*** and ***EVPN_EXPORT*** policies:

- The ***LEAF_TO_SPINE_EVPN_OUT*** policy rejects any BGP-learned routes that carry the **FROM_SPINE_EVPN_TIER** community (0:14). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.

- The ***EVPN_EXPORT*** policy accepts all routes without additional conditions.

As a result, the leaf nodes export only locally originated EVPN routes for the directly connected interfaces between GPU servers and the leaf nodes. These routes are part of the **tenant routing instances** and are required to establish reachability between GPUs belonging to the same tenant.

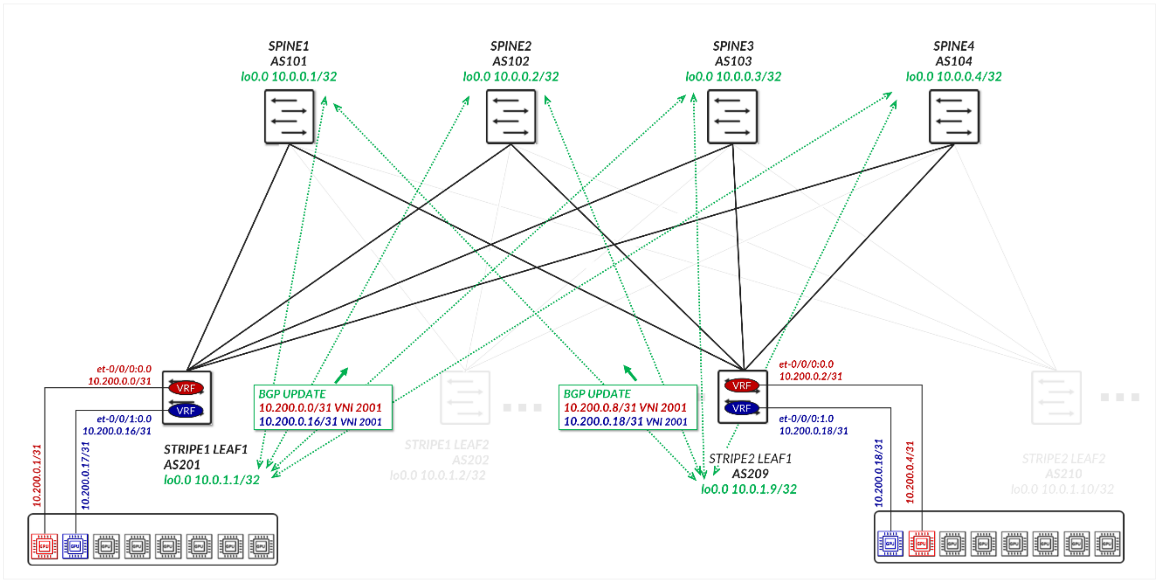
```
jnpr@stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 table Tenant-1
Tenant-1.evpn.0: 12 destinations, 39 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                        Nexthop         MED     Lclpref         AS path
  5:10.0.1.1:2001::0::10.200.0.0::31/248
* Self                      I
  5:10.0.1.1:2001::0::10.200.0.64::31/248
* Self                      I
  5:10.0.1.1:2001::0::192.168.11.1::32/248
```

```
* Self                  I
jnpr@stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 table Tenant-2
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                        Nexthop         MED     Lclpref         AS path
  5:10.0.1.1:2002::0::10.200.0.32::31/248
* Self                  I
  5:10.0.1.1:2002::0::10.200.0.96::31/248
* Self                  I
  5:10.0.1.1:2002::0::192.168.11.2::32/248
* Self                  I
```

Configuration and verification example

Consider the following scenario where Tenant-1 has been assigned GPU 0 on Server 1 and GPU1 on Server 2, and Tenant-2 has been assigned GPU 0 on Server 2 and GPU1 on Server 1 as shown in figure 57.

Figure 57: GPU Assignment Across Servers for Tenant-1 and Tenant-2



Both Stripe 1 Leaf 1 and Leaf 2 have been configured for Tenant-1 and Tenant-2 as shown below:

Table 69. EVPN Routing-Instance for Tenant-1 and Tenant-2 Across Stripe 1 and Stripe 2

STRIPE 1 – LEAF 1	STRIPE 2 – LEAF 1

[edit routing-instances Tenant-A]
jnpr@stripe1-leaf1# show | display set relative
`set instance-type vrf`
`set routing-options graceful-restart`
`set routing-options multipath`
`set protocols evpn ip-prefix-routes advertise direct-nexthop`
`set protocols evpn ip-prefix-routes encapsulation vxlan`
`set protocols evpn ip-prefix-routes vni 20001`
`set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A`
`set interface et-0/0/0:0.0`
`set interface lo0.1`
`set route-distinguisher 10.0.1.1:2001`
`set vrf-target target:20001:1`
	[[edit routing-instances Tenant-A]
	jnpr@stripe2-leaf1# show | display set relative
	`set instance-type vrf`
	`set routing-options graceful-restart`
	`set routing-options multipath`
	`set protocols evpn ip-prefix-routes advertise direct-nexthop`
	`set protocols evpn ip-prefix-routes encapsulation vxlan`
	`set protocols evpn ip-prefix-routes vni 20001`
	`set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A`
	`set interface et-0/0/0:0.0`
	`set interface lo0.1`
	`set route-distinguisher 10.0.1.9:2001`
	`set vrf-target target:20001:1`

[edit routing-instances Tenant-B]
jnpr@stripe1-leaf1# show | display set relative
`set instance-type vrf`
`set routing-options graceful-restart`
`set routing-options multipath`
`set protocols evpn ip-prefix-routes advertise direct-nexthop`
`set protocols evpn ip-prefix-routes encapsulation vxlan`
`set protocols evpn ip-prefix-routes vni 20002`
`set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A`
`set interface et-0/0/0:1.0`
`set interface lo0.2`
`set route-distinguisher 10.0.1.1:2002`
`set vrf-target target:20002:1`
	[[edit routing-instances Tenant-B]
	jnpr@stripe2-leaf1# show | display set relative
	`set instance-type vrf`
	`set routing-options graceful-restart`
	`set routing-options multipath`
	`set protocols evpn ip-prefix-routes advertise direct-nexthop`
	`set protocols evpn ip-prefix-routes encapsulation vxlan`
	`set protocols evpn ip-prefix-routes vni 20002`
	`set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A`
	`set interface et-0/0/0:1.0`
	`set interface lo0.2`
	`set route-distinguisher 10.0.1.9:2002`
	`set vrf-target target:20001:1`

The routing instances create separate routing spaces for the two tenants, providing full route and traffic isolation across the EVPN/VXLAN fabric. Each routing instance has been configured with the following key elements:

1. **Interfaces:**

   The interfaces listed under each tenant VRF (e.g. et-0/0/0:0.0 and et-0/0/1:0.0) are explicitly added to the corresponding routing table. By placing these interfaces under the VRF, all routing decisions and traffic forwarding associated with them are isolated from other tenants and from the global routing table. Assigning an interface that connects a particular GPU to the leaf node effectively maps that GPU to a specific tenant, isolating it from GPUs assigned to other tenants.

2. **Route-distinguisher (RD):**

   10.0.1.1:2001 and 10.0.1.1:2002 uniquely identify EVPN routes from Tenant-1 and Tenant-2, respectively. Even if both tenants use overlapping IP prefixes, the RD ensures their routes remain distinct in the BGP control plane. Although the GPU to leaf links use unique /127 prefixes, an RD is still required to advertise these routes over EVPN.

3. **Route target (RT) community:**

   VRF targets 20001:1 and 20002:1 control which routes are exported from and imported into each tenant routing table. These values determine which routes are shared between VRFs that belong to the same tenant across the fabric and are essential for enabling fabric-wide tenant connectivity, for example, when a tenant has GPUs assigned to multiple servers across different stripes.

4. **Protocols evpn parameters:**

   - The ip-prefix-routes controls how IP Prefix Routes (EVPN Type 5 routes) are advertised.

   - The advertise direct-nexthop enables the leaf node to send IP prefix information using EVPN pure Type 5 routes, which includes a router MAC extended community. These routes include a Router

MAC extended community, which allows the remote VTEP to resolve the next-hop MAC address without relying on Type 2 routes.

- The **encapsulation vxlan** indicates that the payload traffic for this tenant will be encapsulated using VXLAN. The same type of encapsulation must be used end to end.

- The **VXLAN Network Identifier (VNI)** acts as the encapsulation tag for traffic sent across the EVPN/VXLAN fabric. When EVPN Type 5 (IP Prefix) routes are advertised, the associated VNI is included in the BGP update. This ensures that remote VTEPs can identify the correct VXLAN segment for returning traffic to the tenant's VRF.

- Unlike traditional use cases where a VNI maps to a single Layer 2 segment, in EVPN Type 5 the VNI represents the tenant-wide Layer 3 routing domain. All point-to-point subnets, such as the /127 links between GPU servers and the leaf, that belong to the same VRF are advertised with the same VNI.

In this configuration, VNIs 20001 and 20002 are mapped to the Tenant-1 and Tenant-2 VRFs, respectively. All traffic destined for interfaces in Tenant-1 will be forwarded using VNI 20001, and all traffic for Tenant-2 will use VNI 20002.

Notice that the same VNI is configured for the tenant on both Stripe1-Leaf1 and Stripe2-Leaf1.

The export policy ***BGP-AOS-Policy-Tenant-1*** controls which prefixes from this VRF are allowed to be advertised into EVPN.

Table 70. Policies Examples for Tenant-1 and Tenant-2 Across Stripe 1 and Stripe 2

TENANT-A POLICIES	TENANT-B POLICIES
[edit policy-options policy-statement BGP-AOS-Policy-Tenant-A] **jnpr@stripe1-leaf1# show | display set relative** set term BGP-AOS-Policy-Tenant-A-10 from policy AllPodNetworks-Tenant-A set term BGP-AOS-Policy-Tenant-A-10 then accept set term BGP-AOS-Policy-Tenant-A-100 then reject	**[edit policy-options policy-statement BGP-AOS-Policy-Tenant-B]** **jnpr@stripe1-leaf1# show | display set relative** set term BGP-AOS-Policy-Tenant-B-10 from policy AllPodNetworks-Tenant-B set term BGP-AOS-Policy-Tenant-B-10 then accept set term BGP-AOS-Policy-Tenant-B-100 then reject
[edit policy-options policy-statement AllPodNetworks-Tenant-A] **jnpr@stripe1-leaf1# show | display set relative** set term AllPodNetworks-Tenant-A-10 from family inet set term AllPodNetworks-Tenant-A-10 from protocol direct set term AllPodNetworks-Tenant-A-10 then community add TENANT-A_COMMUNITY_V4 set term AllPodNetworks-Tenant-A-10 then accept set term AllPodNetworks-Tenant-A-100 then reject	**[edit policy-options policy-statement AllPodNetworks-Tenant-B]** **jnpr@stripe1-leaf1# show | display set relative** set term AllPodNetworks-Tenant-B-10 from family inet set term AllPodNetworks-Tenant-B-10 from protocol direct set term AllPodNetworks-Tenant-B-10 then community add TENANT-B_COMMUNITY_V4 set term AllPodNetworks-Tenant-B-10 then accept set term AllPodNetworks-Tenant-B-100 then reject
[edit policy-options community TENANT-A_COMMUNITY_V4] **jnpr@stripe1-leaf1# show | display set relative** set members 5:20007 set members 21002:26000	**[edit policy-options community TENANT-B_COMMUNITY_V4]** **jnpr@stripe1-leaf1# show | display set relative** set members 5:20007 set members 21003:26000

Export Policy Logic

EVPN Type 5 routes from Tenant-1 are exported if they are accepted by the BGP-AOS-Policy-Tenant-1 export policy, which references a nested policy named AllPodNetworks-Tenant-1.

- **Policy BGP-AOS-Policy-Tenant-1** accepts any route that is permitted by the ***AllPodNetworks-Tenant-1*** policy and explicitly rejects all other routes.

- **Policy AllPodNetworks-Tenant-1** accepts directly connected IPv6 routes (family inet6, protocol direct) that are part of the Tenant-1 VRF. It tags these routes with the **TENANT-1_COMMUNITY_V4 (5:20007 21002:26000)** community before accepting them. All other routes are rejected.

As a result, only the directly connected IPv6 routes from the Tenant-1 (/127 links between GPU servers and the leaf) are exported as EVPN Type 5 routes.

To verify the interface assignments to the different tenants, use: `show interfaces routing-instance <tenant-name> terse`.

```
jnpr@stripe1-leaf1> show interfaces routing-instance Tenant-1 terse
Interface       Admin   Link    Proto           Local                 Remote
et-0/0/0:0.0    up      up      inet            10.200.0.0/31
                              multiservice
lo0.1           up      up      inet            192.168.11.1        --> 0/0
jnpr@stripe1-leaf1> show interfaces routing-instance Tenant-2 terse
et-0/0/1:0.0    up      up      inet            10.200.0.16/31
                              multiservice
lo0.1           up      up      inet            192.168.11.2        --> 0/0
jnpr@stripe1-leaf2> show interfaces routing-instance Tenant-1 terse
Interface       Admin   Link    Proto           Local                 Remote
et-0/0/0:0.0    up      up      inet            10.200.0.2/31
                              multiservice
lo0.1           up      up      inet            192.168.12.1        --> 0/0
jnpr@stripe1-leaf2> show interfaces routing-instance Tenant-2 terse
et-0/0/1:0.0    up      up      inet            10.200.0.18/31
                              multiservice
lo0.1           up      up      inet            192.168.12.2        --> 0/0
```

You can also check the direct routes installed to the correspondent routing table:

```
jnpr@stripe1-leaf1> show route protocol direct table
Tenant-1.inet.0
Tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.0/31           *[Direct/0] 02:24:29
                         >  via et-0/0/12:0.0
192.168.11.1/32         *[Direct/0] 02:16:52
                         >  via lo0.1
jnpr@stripe1-leaf1> show route protocol direct table
Tenant-2.inet.0
Tenant-2.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
```

```
+ = Active Route, - = Last Active, * = Both
10.200.0.16/31          *[Direct/0] 02:24:29
                         >  via et-0/0/12:0.0
192.168.11.1/32         *[Direct/0] 02:16:52
                         >  via lo0.2
jnpr@stripe1-leaf2> show route protocol direct table
Tenant-1.inet.0
tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.2/31           *[Direct/0] 1d 17:42:33
                         >  via et-0/0/2:0.0
192.168.12.1/32         *[Direct/0] 02:16:52
                         >  via lo0.1
jnpr@stripe1-leaf2> show route protocol direct table
Tenant-2.inet.0
tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.18/31          *[Direct/0] 1d 17:42:33
                         >  via et-0/0/3:0.0
192.168.12.1/32         *[Direct/0] 02:16:52
                         >  via lo0.2
```

To verify evpn l3 contexts including encapsulation, VNI, router MAC address, use `show evpn l3-context`

`Use <tenant-name> extensive` for mode details.

```
jnpr@stripe1-leaf1> show evpn l3-context
L3 context              Type    Adv             Encap   VNI/Label       Router MAC/GW intf dt4-
sid                         dt6-sid                  dt46-sid
Tenant-1                                Cfg     Direct          VXLAN   20001
9c:5a:80:c1:b3:06
Tenant-2                                Cfg     Direct          VXLAN   20002
9c:5a:80:c1:b3:06
jnpr@stripe1-leaf2> show evpn l3-context
L3 context              Type    Adv             Encap   VNI/Label       Router MAC/GW intf dt4-
sid                         dt6-sid                  dt46-sid
Tenant-1                                Cfg     Direct          VXLAN   20001
58:86:70:79:df:db
```

```
Tenant-2                                Cfg     Direct          VXLAN   20002
58:86:70:79:df:db
jnpr@stripe1-leaf1> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-1
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 9c:5a:80:c1:b3:06
  Encapsulation: VXLAN, VNI: 20001
  IPv4 source VTEP address: 10.0.1.1
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-1
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.1:2001
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf1> show evpn l3-context Tenant-2 extensive
L3 context: Tenant-2
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 9c:5a:80:c1:b3:06
  Encapsulation: VXLAN, VNI: 20002
  IPv4 source VTEP address: 10.0.1.1
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-2
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.1:2002
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf2> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-1
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 58:86:70:79:df:db
  Encapsulation: VXLAN, VNI: 20001
  IPv4 source VTEP address: 10.0.1.2
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-1
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.2:2001
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf2> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-2
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 58:86:70:79:df:db
  Encapsulation: VXLAN, VNI: 20002
  IPv4 source VTEP address: 10.0.1.2
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-2
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.2:2002
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf1> show evpn ip-prefix-database
L3 context: Tenant-1
IPv4->EVPN Exported Prefixes
Prefix                                  EVPN route status
10.200.0.0/31                           Created
192.168.11.1/32                         Created
EVPN->IPv4 Imported Prefixes
Prefix                                  Etag
10.200.0.2/31                           0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2001          20001                  58:86:70:79:df:db  10.0.1.2
Accepted      n/a
192.168.12.1/32                         0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2001          20001                  58:86:70:79:df:db  10.0.1.2
Accepted      n/a
L3 context: Tenant-2
IPv4->EVPN Exported Prefixes
Prefix                                  EVPN route status
10.200.0.16/31                          Created
192.168.11.2/32                         Created
EVPN->IPv4 Imported Prefixes
Prefix                                  Etag
10.200.0.18/31                          0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
```

```
  10.0.1.2:2002         20002                  58:86:70:79:df:db  10.0.1.2
Accepted      n/a
192.168.12.2/32                            0
  Route distinguisher    VNI/Label/SID          Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2002         20002                  58:86:70:79:df:db  10.0.1.2
Accepted      n/a
```

When EVPN Type 5 is used to implement L3 tenant isolation across a VXLAN fabric, multiple routing tables are instantiated on each participating leaf node. These tables are responsible for managing control-plane separation, enforcing tenant boundaries, and supporting the overlay forwarding model. Each routing instance (VRF) creates its own set of routing and forwarding tables, in addition to the global and EVPN-specific tables used for fabric-wide communication. These tables are listed in Table 71.

Table 71. Routing and Forwarding Tables for EVPN Type 5

TABLE	DESCRIPTON
bgp.evpn.0	Holds EVPN route information received via BGP, including Type 5 (IP Prefix) routes and other EVPN route types. This is the control plane source for EVPN-learned routes
:vxlan.inet.0	Used internally for VXLAN tunnel resolution. Maps VTEP IP addresses to physical next-hops.
<tenant>.inet.0	The tenant-specific IPv6 unicast routing table. Contains directly connected and EVPN-imported Type 5 prefixes for that tenant. Used for routing data plane traffic.
<tenant>.evpn.0	The tenant-specific EVPN table.

The protocol next-hop is extracted from each EVPN route, is extracted and resolved in inet.0. The EVPN route is added to the bgp.evpn.0 table. The result is placed in :vxlan.inet.0.

The route-target community value is used to determine which tenant the route belongs to, and the route is placed in tenant.evpn.0. From there, IPv4 routes are imported into tenant.inet.0 to be used for route lookups when traffic arrives at the interfaces belonging to the VRF.

IPv4 EBGP sessions advertising evpn routes for Tenant-1 and Tenant-2 should be established. The routes should be installed in both the bgp.evpn.0 table and the <Tenant>.inet.0 table.

```
jnpr@stripe1-leaf1> show bgp summary | no-more
---more---
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.0.0.1                101          5          4       0       0          18 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
10.0.0.2                102          5          4       0       0          14 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
10.0.0.3                103          5          4       0       0          10 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
10.0.0.4                104          5          4       0       0           6 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
jnpr@stripe2-leaf1> show bgp summary | no-more
---more---
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.0.0.1                101        206        199       0       0     1:29:40 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
10.0.0.2                102        206        199       0       0     1:29:25 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
10.0.0.3                103        206        199       0       0     1:29:26 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
10.0.0.4                104        207        199       0       0     1:29:39 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
```

To check that evpn routes are being advertised, use `show route advertising-protocol bgp <neighbor>`. For a specific route, use the `match-prefix` option and include the entire evpn prefix as shown in the example below:

```
jnpr@stripe1-leaf1>  show route advertising-protocol bgp 10.0.0.1 table Tenant | match
5:10.0.1.1:2001 | match 31/248
  5:10.0.1.1:2001::0::10.200.0.0::31/248

jnpr@stripe1-leaf1>  show route advertising-protocol bgp 10.0.0.1 table Tenant | match
5:10.0.1.1:2002 | match 31/248
  5:10.0.1.1:2002::0::10.200.0.16::31/248
jnpr@stripe1-leaf2>  show route advertising-protocol bgp 10.0.0.1 table Tenant | match
5:10.0.1.2:2001 | match 31/248
  5:10.0.1.2:2001::0::10.200.0.2::31/248

jnpr@stripe1-leaf2>  show route advertising-protocol bgp 10.0.0.1 table Tenant | match
5:10.0.1.2:2002 | match 31/248
  5:10.0.1.2:2002::0::10.200.0.18::31/248
jnpr@ stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 match-prefix
5:10.0.1.1:2001::0::10.200.0.0::31/248 table Tenant-1
Tenant-1.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2001::0::10.200.0.0::31/248        *  Self                                    I
jnpr@ stripe1-leaf1> show route advertising-protocol bgp 10.0.0.1 match-prefix
5:10.0.1.1:2002::0::10.200.0.16::31/248 table Tenant-2
Tenant-2.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop              MED     Lclpref    AS path
  5:10.0.1.1:2002::0::10.200.0.16::31/248       *  Self                                    I
jnpr@stripe1-leaf2> show route advertising-protocol bgp 10.0.0.1 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 table Tenant-1
Tenant-1.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2001::0::10.200.0.2::31/248        *  Self                                    I
jnpr@stripe1-leaf2> show route advertising-protocol bgp 10.0.0.1 match-prefix
5:10.0.1.2:2002::0::10.200.0.18::31/248 table Tenant-2
Tenant-2.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
```

```
Restart Complete
  Prefix                                        Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::10.200.0.18::31/248       *  Self                                    I
```

The /248 prefixes represent EVPN route type 5 advertising each IPv4 prefix connecting the GPU servers and leaf nodes.

For example: 5:10.0.1.2:2001::0::10.200.0.0::31/248 is an EVPN route type 5 for prefix 10.200.0.0/31 where:

Table 72. EVPN Type 5 Route Advertisement Fields Description.

Name	Value	Description
Route type	5:	Indicates the route is a Type 5 (IP Prefix) route
Route Distinguisher	10.0.1.2:2001	Uniquely identifies the routes
Placeholder fields	::0::	For MAC address and other Type 2-related fields (not used here)
IP Prefix	10.200.0.4::31	The actual prefix being advertised
VNI	20001	VNI to push for traffic to the destination
Advertising router	10.0.0.1 (Spine 1)	Spine the route was received from.

To check that evpn routes are being received, use `show route receive-protocol bgp <neighbor>`. For a specific route, use the `match-prefix` option and include the entire evpn prefix as shown in the example below:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp 10.0.0.1 | match 5:10.0.1.2:2001 | match 31
  5:10.0.1.2:2001::0::10.200.0.2::31/248
jnpr@stripe1-leaf1> show route receive-protocol bgp 10.0.0.1 | match 5:10.0.1.2:2002 | match 31
  5:10.0.1.2:2002::0::10.200.0.18::31/248
jnpr@stripe1-leaf2> show route receive-protocol bgp 10.0.0.1 | match 5:10.0.1.1:2001 | match 31
  5:10.0.1.1:2001::0::10.200.0.0::31/248
jnpr@stripe1-leaf2> show route receive-protocol bgp 10.0.0.1 | match 5:10.0.1.1:2002 | match 31
  5:10.0.1.1:20021::0::10.200.0.16::31/248
```

The examples show routes received from Spine 1, but each route is received from all 4 spines nodes, which you can also confirm by entering:

```
jnpr@stripe1-leaf1>  show route table bgp.evpn.0 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 | match BGP
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
                   * [BGP/170] 11:31:33, localpref 100, from 10.0.0.1
                     [BGP/170] 11:31:21, localpref 100, from 10.0.0.2
                     [BGP/170] 11:31:14, localpref 100, from 10.0.0.3
                     [BGP/170] 11:31:10, localpref 100, from 10.0.0.4
jnpr@stripe1-leaf2>  show route table bgp.evpn.0 match-prefix
5:10.0.1.1:2001::0::10.200.0.0::31/248 | match BGP
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
                   * [BGP/170] 11:31:13, localpref 100, from 10.0.0.1
                     [BGP/170] 11:31:41, localpref 100, from 10.0.0.2
                     [BGP/170] 11:31:12, localpref 100, from 10.0.0.3
                     [BGP/170] 11:31:52, localpref 100, from 10.0.0.4
```

Additional information for a given route can be found using the extensive keyword:

```
jnpr@stripe1-leaf1> show route table bgp.evpn.0 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 active-path extensive
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
Restart Complete
5:10.0.1.2:2001::0::10.200.0.2::31/248  (4 entries, 0 announced)
     *BGP    Preference: 170/-101
             Route Distinguisher: 10.0.1.2:2001
             Next hop type: Indirect, Next hop index: 0
             Address: 0x55dfb9c305fc
             Next-hop reference count: 48
             Kernel Table Id: 0
             Source: 10.0.0.1
             Protocol next hop: 10.0.1.2
             Label operation: Push 20001
             Label TTL action: prop-ttl
             Load balance label: Label 20001: None;
             Indirect next hop: 0x2 no-forward INH Session ID: 0
             Indirect next hop: INH non-key opaque: (nil) INH key opaque: (nil)
             State: <Active Ext>
             Local AS:   201 Peer AS:   101
             Age: 7:54:49    Metric2: 0
```

```
               Validation State: unverified
               Task: BGP_109.10.0.0.1
               AS path: 109 210 I
               Communities: 0:14 7:20007 21002:26000 target:20001:1
               encapsulation:vxlan(0x8) router-mac:58:86:70:7b:10:db
                Import Accepted
                Route Label: 20001
                Overlay gateway address: 0.0.0.0
                ESI 00:00:00:00:00:00:00:00:00:00
                Localpref: 100
                Router ID: 10.0.0.1
                Secondary Tables: Tenant-1.evpn.0
                Thread: junos-main
                Indirect next hops: 1
               Protocol next hop: 10.0.1.2 ResolvState: Resolved
               Label operation: Push 20001
               Label TTL action: prop-ttl
               Load balance label: Label 20001: None;
               Indirect next hop: 0x2 no-forward INH Session ID: 0
               Indirect next hop: INH non-key opaque: (nil) INH key opaque: (nil)
               Indirect path forwarding next hops: 4
                                  Next hop type: Router
                                  Next hop: 10.0.2.64 via et-0/0/2:0.0
                                  Session Id: 0
                                  Next hop: 10.0.2.82 via et-0/0/3:0.0
                                  Session Id: 0
                                  Next hop: 10.0.2.98 via et-0/0/0:0.0
                                  Session Id: 0
                                  Next hop: 10.0.2.114 via et-0/0/1:0.0
                                  Session Id: 0
                                  10.0.1.2/32 Originating RIB: inet.0
                                    Node path count: 1
                                    Forwarding nexthops: 4
 ---(more)---
```

Table 73. EVPN Type 5 Route Advertisement Fields Description - Extensive

Name	Value	Description
Route type	5:	Indicates the route is a Type 5 (IP Prefix) route

(Continued)

Name	Value	Description
Route Distinguisher	10.0.1.2:2001	Uniquely identifies the routes
Placeholder fields	::0::	For MAC address and other Type 2-related fields (not used here)
IP Prefix	10.200.105.0::24	The actual prefix being advertised
VNI	20001	VNI to push for traffic to the destination
Advertising router	10.0.0.1	Spine the route was received from.
Protocol next hop	10.0.1.2 (Stripe 1 Leaf 2)	Router that originated the EVPN route (remote VTEP)
Encapsulation	Type: 0x08	standardized IANA-assigned value for VXLAN encapsulation in the EVPN Encapsulation extended community (RFC 9014)
Route target	target:20001:1	Identifies the route as belonging to Tenant-1

To check that the routes are being imported into the correspondent tenant's routing tables, use `show route table <tenant-name>.inet.0 protocol evpn`, as shown in the example below:

```
jnpr@stripe1-leaf1> show route table Tenant-1.inet.0 protocol evpn | match /31
10.200.0.2/31      *[EVPN/170] 04:02:04
jnpr@stripe1-leaf1> show route table Tenant-2.inet.0 protocol evpn | match /31
10.200.0.18/31     *[EVPN/170] 04:02:04
jnpr@stripe1-leaf2> show route table Tenant-1.inet.0 protocol evpn | match /31
10.200.0.0/31      *[EVPN/170] 04:02:04
jnpr@stripe1-leaf2> show route table Tenant-2.inet.0 protocol evpn | match /31
10.200.0.16/31     *[EVPN/170] 04:02:04
```

Appendix C – IPv6 Overlay with Static Addresses Over IPv6 Underlay Fabric Implementation

IN THIS SECTION

This section outlines the configuration components for an IPv6 underlay and IPv6 overlay deployment.

Control Plane Implementation with IPv6 Underlay and IPv6 Overlay

This model provides an IPv6 transport underlay and IPv6 EVPN/VXLAN transport in the overlay that can support both IPv4 and IPv6 devices communicating across the fabric. This model aligns with traditional IP fabric designs, where interface addressing is fully controlled and visible, and neighbor relationships are explicitly defined, while also supporting both IPv4-only and IPv6 only end devices.

The interfaces between leaf and spine nodes are configured with explicit /127 IPv6 addresses assigned from a pool of IPv6 addresses reserved for the underlay. These addresses can be global or site local routable IPv6 addresses. Each device on the point-to-point link is configured with one of the two usable IPv6 addresses in the corresponding /127 subnet. This allows efficient address assignments for the point-to-point links between leaf and spine nodes. All leaf and spine nodes are also configured with IPv6 addresses on the loopback interface (lo0.0).

The **underlay EBGP** sessions are set up between the leaf and spine nodes, by explicitly configuring each neighbor, using the /127 IPv6 addresses assigned between them.

The EBGP configuration for this model includes each neighbor's IPv6 address and Autonomous System (AS) number, the local Autonomous System (AS) number, and the export policy that allows the advertisement of routes to reach all the leaf and spine nodes in the fabric. These routes are standard IPv6 unicast advertising the IPv6 addresses assigned to the loopback interface (lo0.0).

The **overlay EBGP** sessions are also set up by explicitly configuring each neighbor, using the IPv6 addresses of the loopback interfaces advertised by the underlay EBGP sessions, and are also established between the leaf and spine nodes.

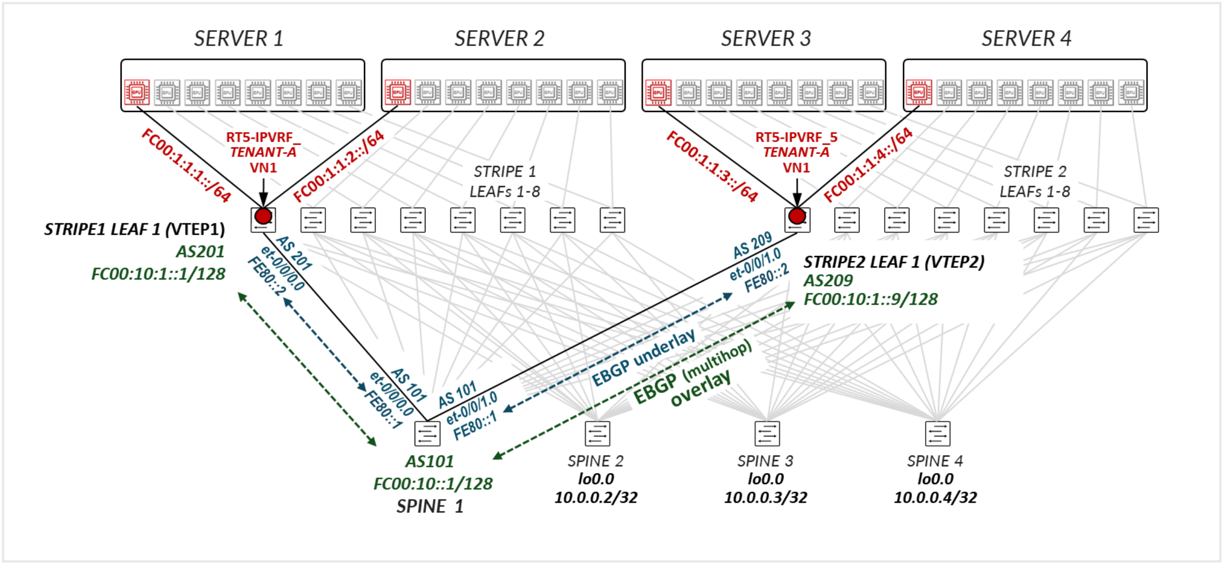
The leaf nodes act as VTEPs, and exchange EVPN Type 5 routes advertising the IPv4 prefixes or IPv6 prefixes assigned to the links between the GPU servers and the leaf nodes.

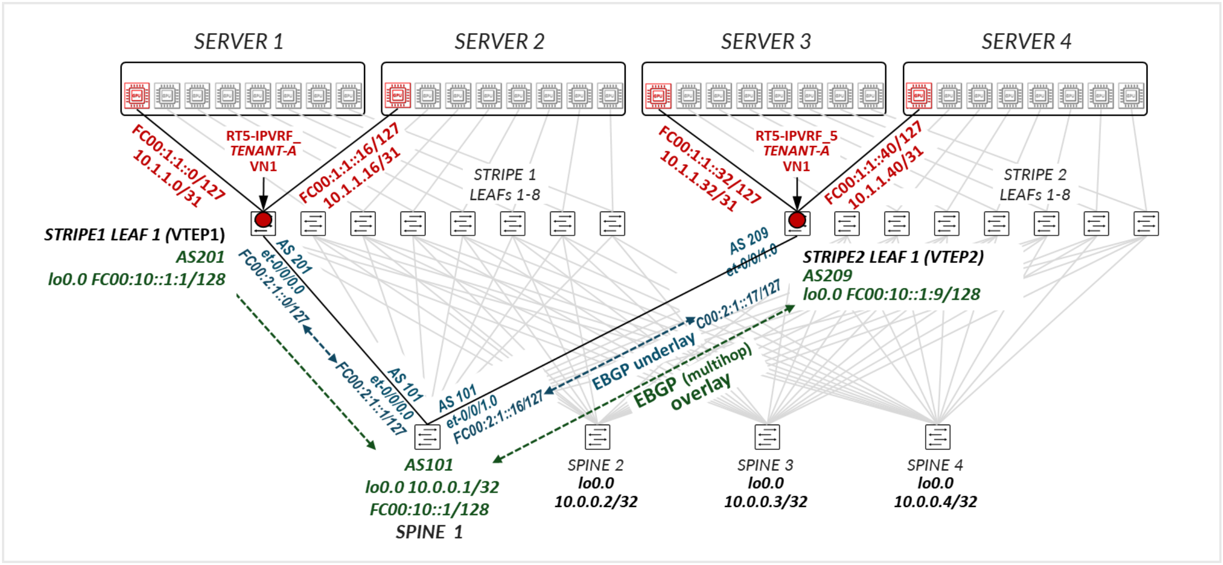
<u>Example:</u>

Consider the example depicted in Figure 58.

For the underlay, STRIPE1 LEAF 1 in AS 201 establishes an EBGP session with SPINE 1 in AS 101 over the directly connected IPv6 point-to-point link FC00:0:2:1::2/127 <=> FC00:0:2:1::1/127. Similarly, STRIPE2 LEAF 1 in AS 209 establishes an EBGP session with SPINE 1 over the link FC00:0:2:9::2/127 <=> FC00:0:2:9::1/127.

Figure 58: IPv6 Underlay and IPv6 Overlay Example





These sessions exchange IPv6 unicast routes advertising the address of the loopback interface (lo0.0) of STRIPE1 LEAF 1 (FC00:10::1:1), STRIPE2 LEAF 1 (FC00:10::1:9) and SPINE 1 (FC00:10::1).

Although it is not shown in the diagram, STRIPE1 LEAF 1 and STRIPE2 LEAF 1 will also establish EBGP sessions with SPINE 2, SPINE 3, and SPINE 4 to ensure multiple paths are available for traffic.EBGP sessions are established between the leaf nodes and SPINE 1 using their loopback addresses (FC00:10::1:1, FC00:10::1:9, and FC00:10::1 respectively).

The leaf nodes acting as VTEP advertise the links connecting the GPU servers and leaf nodes which in the example are configured with /31 IPv4 and /127 IPv6 addresses.

NOTE: The GPU servers and leaf nodes links are shown here with both IPv4 and IPv6 for demonstration purposes. It is not a requirement. The customer can choose which network layer address scheme to use.

The prefixes on the GPU servers and leaf nodes links are advertised using EVPN type 5 routes.

For example, STRIPE1 LEAF 1 advertises routes to the IPv4 and IPv6 addresses on the links connecting SERVER 1 GPU1 (10.1.1.0/31 and FC00:10:1:1::0/127 respectively) and SERVER 2 GPU1 to STRIPE1 LEAF 1 (10.1.1.16/31 and FC00:10:1:1::16/127 respectively).

Similarly, STRIPE2 LEAF 1 advertises router to the IPv4 addresses on the links connecting SERVER 3 GPU1 (10.1.1.32/31 and FC00:10:1:1::32/127 respectively) and SERVER 4 GPU1 to STRIPE1 LEAF 1 (10.1.1.40/31 and FC00:10:1:1::40/127 respectively).

Assuming all four GPUs in the example belong to the same tenant, their associated interfaces are mapped to the same VRF (RT5-IP-VRF_TENANT-1).

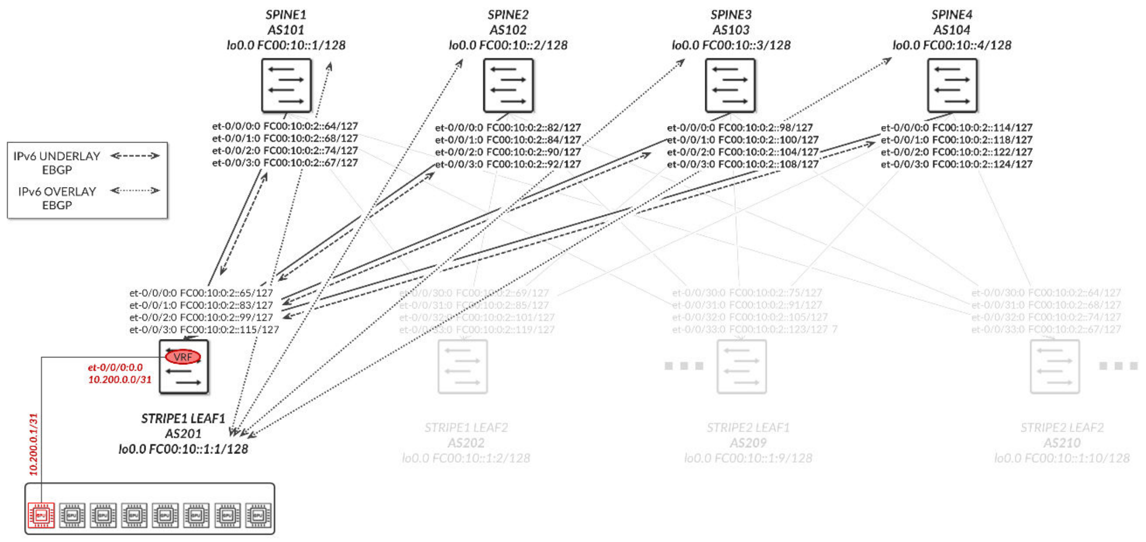
RT5-IP-VRF_TENANT-1 is configured on both STRIPE1 LEAF 1 and STRIPE2 LEAF 1 with the same VXLAN Network Identifier (VNI) and route targets. STRIPE1 LEAF 1 advertises the prefixes 10.1.1.0/31 and 10.1.1.16/31 (or their equivalent IPv6 prefixes) to SPINE 1 as EVPN Route Type 5, with its own loopback (10.0.1.1) as the next-hop VTEP. In the same way, STRIPE2 LEAF 1 advertises 10.1.1.32/31 and 10.1.1.40/31 (or their equivalent IPv6 prefixes) with 10.0.1.9 as the next-hop.

When SERVER 1 GPU1 sends traffic to SERVER 3 GPU1, the destination addresses 10.1.1.32 for example, is found in the VRF routing table on STRIPE1 LEAF 1 (Tenant-1.inet.0). The route points to STRIPE2 LEAF 1 (VTEP at 10.0.1.9) as the protocol next-hop (which is resolved to the addresses of the spine nodes). The route also specifies VNI 1 as the VXLAN encapsulation ID. The packet is encapsulated with the VXLAN header and tunneled across the fabric to its destination.

Spine Nodes to Leaf Connections

The interfaces between the leaf and spine nodes do not require explicitly configured IP addresses and are configured as untagged interfaces with only family inet and family inet6 to enable processing of IPv4 and IPv6 traffic as shown in Figure 59.

Figure 59: IPv6 underlay and IPv6 overlay configuration example


SPINE1
AS101
lo0.0 FC00:10::1/128
SPINE2
AS102
lo0.0 FC00:10::2/128
SPINE3
AS103
lo0.0 FC00:10::3/128
SPINE4
AS104
lo0.0 FC00:10::4/128
et-0/0/0:0 FC00:10:0:2::64/127
et-0/0/1:0 FC00:10:0:2::68/127
et-0/0/2:0 FC00:10:0:2::74/127
et-0/0/3:0 FC00:10:0:2::67/127
et-0/0/0:0 FC00:10:0:2::82/127
et-0/0/1:0 FC00:10:0:2::84/127
et-0/0/2:0 FC00:10:0:2::90/127
et-0/0/3:0 FC00:10:0:2::92/127
et-0/0/0:0 FC00:10:0:2::98/127
et-0/0/1:0 FC00:10:0:2::100/127
et-0/0/2:0 FC00:10:0:2::104/127
et-0/0/3:0 FC00:10:0:2::108/127
et-0/0/0:0 FC00:10:0:2::114/127
et-0/0/1:0 FC00:10:0:2::118/127
et-0/0/2:0 FC00:10:0:2::122/127
et-0/0/3:0 FC00:10:0:2::124/127
IPv6 UNDERLAY EBGP
IPv6 OVERLAY EBGP
et-0/0/0:0 FC00:10:0:2::65/127
et-0/0/1:0 FC00:10:0:2::83/127
et-0/0/2:0 FC00:10:0:2::99/127
et-0/0/3:0 FC00:10:0:2::115/127
VRF
et-0/0/0:0.0
10.200.0.0/31
10.200.0.1/31
STRIPE1 LEAF1
AS201
lo0.0 FC00:10::1:1/128
et-0/0/30:0 FC00:10:0:2::69/127
et-0/0/31:0 FC00:10:0:2::85/127
et-0/0/32:0 FC00:10:0:2::101/127
et-0/0/33:0 FC00:10:0:2::119/127
STRIPE1 LEAF2
AS202
lo0.0 FC00:10::1:2/128
et-0/0/30:0 FC00:10:0:2::75/127
et-0/0/31:0 FC00:10:0:2::91/127
et-0/0/32:0 FC00:10:0:2::105/127
et-0/0/33:0 FC00:10:0:2::123/127
STRIPE2 LEAF1
AS209
lo0.0 FC00:10::1:9/128
et-0/0/30:0 FC00:10:0:2::64/127
et-0/0/31:0 FC00:10:0:2::68/127
et-0/0/32:0 FC00:10:0:2::74/127
et-0/0/33:0 FC00:10:0:2::67/127
STRIPE2 LEAF2
AS210
lo0.0 FC00:10::1:10/128
GPU

The interfaces between the leaf and spine nodes are configured with /127 addresses as shown in Table 74.

Table 74. IPv6 Address Assignments for Leaf-to-Spine Interfaces (/127 Subnetting)

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
Stripe 1 Leaf 1 - et-0/0/30:0	FC00:10:0:2::65/127	Spine 1 – et-0/0/0:0	FC00:10:0:2::64/31
Stripe 1 Leaf 1 - et-0/0/31:0	FC00:10:0:2::83/127	Spine 2 – et-0/0/1:0	FC00:10:0:2::82/127
Stripe 1 Leaf 1 - et-0/0/32:0	FC00:10:0:2::99/127	Spine 3 – et-0/0/2:0	FC00:10:0:2::98/127
Stripe 1 Leaf 1 - et-0/0/33:0	FC00:10:0:2::155/127	Spine 4 – et-0/0/3:0	FC00:10:0:2::114/127
Stripe 1 Leaf 5 - et-0/0/30:0	FC00:10:0:2::69/127	Spine 1 – et-0/0/0:0	FC00:10:0:2::68/127
Stripe 1 Leaf 2 - et-0/0/31:0	FC00:10:0:2::85/127	Spine 2 – et-0/0/1:0	FC00:10:0:2::.84/127

(Continued)

LEAF NODE INTERFACE	LEAF NODE IPv6 ADDRESS	SPINE NODE INTERFACE	SPINE IPv6 ADDRESS
Stripe 1 Leaf 2 - et-0/0/32:0	FC00:10:0:2::101/127	Spine 3 – et-0/0/2:0	FC00:10:0:2::100/127
Stripe 1 Leaf 2 - et-0/0/33:0	FC00:10:0:2::119/127	Spine 4 – et-0/0/3:0	FC00:10:0:2::118/127
. . .			

These interfaces are configured as untagged interfaces, with family inet6 and static IPv6 addresses, as shown in the example for the link between Stripe 1 Leaf 1 and Spine 1 below:

Table 75. Example Junos Configuration for Leaf-Spine IPv6 Interface (/127 Subnet)

STRIPE 1 LEAF 1 (et-0/0/0:30)

```
[edit interfaces et-0/0/30]
jnpr@stripe1-leaf1# show | display set relative
set description "Breakout et-0/0/30"
set number-of-sub-ports 1
set speed 800g

[edit interfaces et-0/0/30:0]
jnpr@stripe1-leaf1# show | display set relative
set description facing_spine1:et-0/0/0:0
set mtu 9216
set unit 0 family inet mtu 9202
set unit 0 family inet address FC00:10:0:2::65/127
```

SPINE 1 (et-0/0/0:0)

```
[edit interfaces et-0/0/0]
jnpr@spine1# show | display set
set interfaces et-0/0/0 description "Breakout et-0/0/0"
set interfaces et-0/0/0 number-of-sub-ports 1
set interfaces et-0/0/0 speed 800g

[edit interfaces et-0/0/0:0]
jnpr@spine1# show | display set relative
set description "To Leaf1"
set mtu 9216
set unit 0 family inet mtu 9202
set unit 0 family inet address FC00:10:0:2::64/127
```

The loopback and Autonomous System numbers for all devices in the fabric are included in Table 76.

Table 76. Loopback IPv6 Addresses and Autonomous System Numbers

LEAF NODE INTERFACE	lo0.0 IPv6 ADDRESS	Local AS #
Stripe 1 Leaf 1	FC00:10:0:1::1/128	201
Stripe 1 Leaf 2	FC00:10:0:1::2/128	202
Stripe 1 Leaf 3	FC00:10:0:1::3/128	203

(Continued)

LEAF NODE INTERFACE	lo0.0 IPv6 ADDRESS	Local AS #
Stripe 1 Leaf 4	FC00:10:0:1::4/128	204
Stripe 1 Leaf 5	FC00:10:0:1::5/128	205
Stripe 1 Leaf 6	FC00:10:0:1::6/128	206
Stripe 1 Leaf 7	FC00:10:0:1::7/128	207
Stripe 1 Leaf 8	FC00:10:0:1::8/128	208
Stripe 2 Leaf 1	FC00:10:0:1::9/128	209
Stripe 2 Leaf 2	FC00:10:0:1::10/128	210
. . .		
SPINE1	FC00:10::1/128	101
SPINE2	FC00:10::2/128	102
SPINE3	FC00:10::3/128	103
SPINE4	FC00:10::4/128	104

Table 77. Example Junos Configuration for IPv6 Loopback Interfaces and Routing Options

STRIPE 1 LEAF 1

```
[edit]
jnpr@spine1# show routing-options
router-id 10.0.1.1;
autonomous-system 201;
graceful-restart;
forwarding-table {
    export PFE-LB;
    ecmp-fast-reroute;
}

[edit]
jnpr@stripe1-leaf1# show interfaces lo0
unit 0 {
  family inet {
     address FC00:10:0:1::1/128;
  }
}
```

SPINE 1

```
[edit]
jnpr@spine1# show routing-options
router-id 10.0.0.1;
autonomous-system 101;
graceful-restart;
forwarding-table {
    export PFE-LB;
    ecmp-fast-reroute;
}

[edit]
jnpr@spine1# show interfaces lo0
unit 0 {
  family inet {
     address FC00:10::1/128;
  }
}
```

GPU Backend Fabric Underlay with IPv6

The **underlay EBGP sessions** are configured between the leaf and spine nodes using the IP addresses of the directly connected links, as shown in the example between Stripe1 Leaf 1 and the spine nodes below:

Table 78. IPv6 EBGP Underlay Configuration Example: Stripe 1 Leaf 1 to Spine 1

STRIPE 2 LEAF 1

```
[edit protocols bgp]
jnpr@stripe1-leaf1# show
group l3clos-inet6-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor FC00:10:0:2::64 {
        description facing_spine1;
        local-address FC00:10:0:2::65;
        family inet6 {
            unicast;
        }
        export ( LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
[edit protocols bgp]
jnpr@spine1# show
group l3clos-inet6-underlay {
    type external;
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 1000;
        multiplier 3;
    }
    neighbor FC00:10:0:2::65 {
        description facing_stripe1-leaf1;
        local-address FC00:10:0:2::64;
        family inet6 {
            unicast;
        }
        export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );
        peer-as 201;
    }
     .
    .
    .
    vpn-apply-export;
}
```

Table 79. IPv6 EBGP Underlay Configuration Example: Stripe 1 Leaf 1 to Spine 2

STRIPE 2 LEAF 1	SPINE 2
[edit protocols bgp]	**[edit protocols bgp]**
jnpr@stripe1-leaf1# show	**jnpr@spine1# show**
`group l3clos-inet6-underlay {`	`group l3clos-inet6-underlay {`
`    type external;`	`    type external;`
`    multipath {`	`    multipath {`
`        multiple-as;`	`        multiple-as;`
`    }`	`    }`
`    bfd-liveness-detection {`	`    bfd-liveness-detection {`
`        minimum-interval 1000;`	`        minimum-interval 1000;`
`        multiplier 3;`	`        multiplier 3;`
`    }`	`    }`
`    neighbor FC00:10:0:2::82 {`	`    neighbor FC00:10:0:2::83 {`
`        description facing_spine1;`	`        description facing_stripe1-leaf1;`
`        local-address 2001:10:0:2::83;`	`        local-address 2001:10:0:2::82;`
`        family inet6 {`	`        family inet6 {`
`            unicast;`	`            unicast;`
`        }`	`        }`
`        export ( LEAF_TO_SPINE_FABRIC_OUT && BGP-AOS-Policy );`	`        export ( SPINE_TO_LEAF_FABRIC_OUT && BGP-AOS-Policy );`
`        peer-as 102;`	`        peer-as 201;`
`    }`	`    }`
`    .`	`     .`
`    .`	`    .`
`    .`	`    .`
`    vpn-apply-export;`	`    vpn-apply-export;`
`}`	`}`

All the BGP sessions are configured with **multipath multiple-as**, which allows multiple paths (to the same destination) with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing, and with **BFD** to improve convergence in case of failures.

To control the propagation of routes, `export policies` are applied to these EBGP sessions as shown in the example in Table 80.

Table 80. Export policy example to advertise IPv6 routes over IPv6 BGP Underlay

LEAF	SPINE
[edit policy-options policy-statement LEAF_TO_SPINE_FABRIC_OUT]	**[edit policy-options policy-statement SPINE_TO_LEAF_FABRIC_OUT]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term LEAF_TO_SPINE_FABRIC_OUT-10 from protocol bgp	set term SPINE_TO_LEAF_FABRIC_OUT-10 then community add FROM_SPINE_FABRIC_TIER
set term LEAF_TO_SPINE_FABRIC_OUT-10 from community FROM_SPINE_FABRIC_TIER	set term SPINE_TO_LEAF_FABRIC_OUT-10 then accept
set term LEAF_TO_SPINE_FABRIC_OUT-10 then reject	
set term LEAF_TO_SPINE_FABRIC_OUT-20 then accept	
[edit policy-options community FROM_SPINE_FABRIC_TIER]	**[edit policy-options community FROM_SPINE_FABRIC_TIER]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set members 0:15	**set members 0:15**
[edit policy-options policy-statement BGP-AOS-Policy]	**[edit policy-options policy-statement BGP-AOS-Policy]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term BGP-AOS-Policy-10 from policy AllPodNetworks	set term BGP-AOS-Policy-10 from policy AllPodNetworks
set term BGP-AOS-Policy-10 then accept	set term BGP-AOS-Policy-10 then accept
set term BGP-AOS-Policy-100 then reject	set term BGP-AOS-Policy-20 from protocol bgp
	set term BGP-AOS-Policy-20 then accept
	set term BGP-AOS-Policy-100 then reject
[edit policy-options policy-statement AllPodNetworks]	**[edit policy-options policy-statement AllPodNetworks]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set term AllPodNetworks-10 from family inet6	set term AllPodNetworks-10 from family inet6
set term AllPodNetworks-10 from protocol direct	set term AllPodNetworks-10 from protocol direct
set term AllPodNetworks-10 from interface lo0.0	set term AllPodNetworks-10 from interface lo0.0
set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V6	set term AllPodNetworks-10 then community add DEFAULT_DIRECT_V6
set term AllPodNetworks-10 then accept	set term AllPodNetworks-10 then accept
set term AllPodNetworks-100 then reject	set term AllPodNetworks-100 then reject
[edit policy-options community DEFAULT_DIRECT_V6]	**[edit policy-options community DEFAULT_DIRECT_V6]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@spine1# show | display set relative**
set members 5:20008	set members 1:20008
set members 21001:26000	set members 21001:26000

These policies ensure loopback reachability is advertised cleanly and without the risk of route loops.

On the spine nodes, routes are exported only if they are accepted by both the *SPINE_TO_LEAF_FABRIC_OUT* and *BGP-AOS-Policy* export policies.

- The *SPINE_TO_LEAF_FABRIC_OUT* policy has no match conditions and accepts all routes unconditionally, tagging them with the **FROM_SPINE_FABRIC_TIER** community (0:15).
- The *BGP-AOS-Policy* accepts BGP-learned routes as well as any routes accepted by the nested *AllPodNetworks* policy.
- The *AllPodNetworks* policy, in turn, matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V4** community (1:20007 and 21001:26000 on Spine1).

As a result, each spine advertises both its directly connected routes (including its loopback interface) and any routes it has received from other leaf nodes.

Example:

```
jnpr@spine1>  show route advertising-protocol bgp FC00:10:0:2::65 | match /32
* FC00:10::1/128                        Self       I
* FC00:10:0:1::2/128                    Self       202 I
* FC00:10:0:1::3/128                    Self       203 I
---more---
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:2::65 FC00:10::0/128 extensive
inet6.0: 36 destinations, 40 routes (36 active, 0 holddown, 0 hidden)
Restart Complete
* FC00:10::0/128 (1 entry, 1 announced)
 BGP group l3clos-inet6-underlay type External
     Nexthop: Self
     AS path: [101] I
     Communities: 0:15 1:20008 21001:26000
jnpr@spine1> show route advertising-protocol bgp FC00:10:0:2::65 FC00:10:0:1::2/128 extensive
inet6.0: 85 destinations, 169 routes (85 active, 0 holddown, 0 hidden)
Restart Complete
* FC00:10:0:1::2/128 (1 entry, 1 announced)
 BGP group l3clos-inet6-underlay type External
     AS path: [101] 202 I
     Communities: 0:15 5:20008 21001:26000
```

On the **leaf nodes**, routes are exported only if they are accepted by both the *LEAF_TO_SPINE_FABRIC_OUT* and *BGP-AOS-Policy* export policies.

The *LEAF_TO_SPINE_FABRIC_OUT* policy accepts all routes except those learned via BGP that are tagged with the **FROM_SPINE_FABRIC_TIER** community (0:15). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.

The ***BGP-AOS-Policy*** accepts all routes allowed by the nested ***AllPodNetworks*** policy, which matches directly connected IPv6 routes and tags them with the **DEFAULT_DIRECT_V4** community (5:20007 and 21001:26000 for Stripe1-Leaf1).

As a result, leaf nodes will advertise only their directly connected interface routes—including their loopback interfaces, to the spines.

```
jnpr@stripe1-leaf1>  show route advertising-protocol bgp FC00:10:0:2::64 | match /32
* FC00:10:0:1::1/128      Self                                    I
jnpr@stripe1-leaf1>  show route advertising-protocol bgp FC00:10:0:2::64 FC00:10:0:1::1/128
extensive
inet6.0: 48 destinations, 257 routes (48 active, 0 holddown, 0 hidden)
Restart Complete
* FC00:10:0:1::1/128  (1 entry, 1 announced)
 BGP group l3clos-inet6-underlay  type External
     Nexthop: Self
     AS path: [201] I
     Communities: 5:20007 21001:26000
```

GPU Backend Fabric Overlay with IPv6

The **overlay EBGP sessions** are configured between the leaf and spine nodes using the IPv4 addresses of the loopback interfaces, as shown in the example between Stripe1 Leaf 1/Stripe 2 Leaf 1 and Spine 1.

Table 81. IPv6 EVPN Overlay EBGP Configuration Example: Stripe 1 Leaf 1 to Spine 1

STRIPE 1 LEAF 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet6-overlay {
    type external;
    multihop {
        ttl 1;
    }
    family inet6 {
        unicast;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor FC00:10::1 {
        description facing_spine1-evpn-overlay;
        local-address FC00:10:0:1::1;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
edit]
jnpr@spine1# show protocols bgp
group l3clos-inet6-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family inet6 {
        unicast;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor FC00:10:0:1::1 {
        description facing_leaf1-evpn-overlay;
        local-address FC00:10::1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 201;
    }
    .
    .
    .
    vpn-apply-export;
}
```

Table 82. IPv6 EVPN Overlay EBGP Configuration Example: Stripe 2 Leaf 1 to Spine 1

STRIPE 2 LEAF 1

```
[edit]
jnpr@stripe1-leaf1# show protocols bgp
group l3clos-inet6-overlay {
    type external;
    multihop {
        ttl 1;
    }
    family inet6 {
        unicast;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor FC00:10::1 {
        description facing_spine1-evpn-overlay;
        local-address FC00:10:0:1::9;
        family evpn {
            signaling;
        }
        export ( LEAF_TO_SPINE_EVPN_OUT && EVPN_EXPORT );
        peer-as 101;
    }
    .
    .
    .
    vpn-apply-export;
}
```

SPINE 1

```
edit]
jnpr@spine1# show protocols bgp
group l3clos-inet6-overlay {
    type external;
    multihop {
        ttl 1;
        no-nexthop-change;
    }
    family inet6 {
        unicast;
    }
    multipath {
        multiple-as;
    }
    bfd-liveness-detection {
        minimum-interval 3000;
        multiplier 3;
    }
    neighbor FC00:10:0:1::9 {
        description facing_leaf1-evpn-overlay;
        local-address FC00:10::1;
        family evpn {
            signaling;
        }
        export ( SPINE_TO_LEAF_EVPN_OUT );
        peer-as 209;
    }
    .
    .
    .
    vpn-apply-export;
}
```

The overlay BGP sessions use family evpn signaling to enable EVPN route exchange. The `multihop ttl 1` statement allows EBGP sessions to be established between the loopback interfaces.

As with the underlay BGP sessions, these sessions are configured with `multipath multiple-as`, allowing multiple EVPN paths with different AS paths to be considered for ECMP (Equal-Cost Multi-Path) routing. BFD (Bidirectional Forwarding Detection) is also enabled to improve convergence time in case of failures.

The `no-nexthop-change` knob on the spine nodes is used to preserve the original next-hop address, which is critical in EVPN for ensuring that the remote VTEP can be reached directly. The `vpn-apply-export` statement is included to ensure that the export policies are evaluated for VPN address families, such as EVPN, allowing fine-grained control over which routes are advertised to each peer.

To control the propagation of routes, export policies are applied to these EBGP sessions as shown in the example in Table 83.

Table 83. Export Policy example to advertise EVPN routes over IPv6 BGP Overlay

LEAF	SPINE
[edit policy-options policy-statement LEAF_TO_SPINE_EVPN_OUT] jnpr@stripe1-leaf1# show | display set relative set term LEAF_TO_SPINE_EVPN_OUT-10 from protocol bgp set term LEAF_TO_SPINE_EVPN_OUT-10 from community FROM_SPINE_EVPN_TIER set term LEAF_TO_SPINE_EVPN_OUT-10 then reject set term LEAF_TO_SPINE_EVPN_OUT-20 then accept **[edit policy-options community FROM_SPINE_EVPN_TIER]** **jnpr@stripe1-leaf1# show | display set relative** set members 0:14 **[edit policy-options policy-statement EVPN_EXPORT]** **jnpr@stripe1-leaf1# show | display set relative** set term EVPN_EXPORT-4095 then accept	**[edit policy-options policy-statement SPINE_TO_LEAF_EVPN_OUT]** **jnpr@spine1# show | display set relative** set term SPINE_TO_LEAF_EVPN_OUT-10 then community add FROM_SPINE_EVPN_TIER set term SPINE_TO_LEAF_EVPN_OUT-10 then accept **[edit policy-options community FROM_SPINE_EVPN_TIER]** **jnpr@spine1# show | display set relative** set members 0:14

These policies are simpler in structure and are intended to enable end-to-end EVPN reachability between tenant GPUs, while preventing route loops within the overlay.

Routes will only be advertised if EVPN routing-instances have been created. Example:

Table 84. EVPN Routing-Instances for a single tenant example across different leaf nodes.

STRIPE 1 – LEAF 1	STRIPE 2 – LEAF 1
[edit routing-instances Tenant-A] **jnpr@stripe1-leaf1# show | display set relative** set instance-type vrf set routing-options graceful-restart set routing-options multipath set protocols evpn ip-prefix-routes advertise direct-nexthop set protocols evpn ip-prefix-routes encapsulation vxlan set protocols evpn ip-prefix-routes vni 20001 set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A set interface et-0/0/12:0.0 set interface lo0.1 set route-distinguisher 10.0.1.1:2001 set vrf-target target:20001:1	**[[edit routing-instances Tenant-A]** **jnpr@stripe2-leaf1# show | display set relative** set instance-type vrf set routing-options graceful-restart set routing-options multipath set protocols evpn ip-prefix-routes advertise direct-nexthop set protocols evpn ip-prefix-routes encapsulation vxlan set protocols evpn ip-prefix-routes vni 20009 set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A set interface et-0/0/12:0.0 set interface lo0.1 set route-distinguisher 10.0.1.9:2009 set vrf-target target:20001:1

On the **spine nodes**, routes are exported if they are accepted by the ***SPINE_TO_LEAF_EVPN_OUT*** policy.

- The ***SPINE_TO_LEAF_EVPN_OUT*** policy has no match conditions and accepts all routes. It tags each exported route with the **FROM_SPINE_EVPN_TIER** community (0:14).

As a result, the spine nodes export EVPN routes received from one leaf to all other leaf nodes, allowing tenant-to-tenant communication across the fabric.

Example:

```
jnpr@spine1>  show route advertising-protocol bgp FC00:10:0:1::1 | match 5:10.*2001.*31
  5:10.0.1.2:2001::0::10.200.0.2::31/248
  5:10.0.1.2:2001::0::10.200.0.34::31/248
  5:10.0.1.9:2001::0::10.200.1.0::31/248
  5:10.0.1.9:2001::0::10.200.1.32::31/248
  5:10.0.1.10:2001::0::10.200.1.2::31/248
  5:10.0.1.10:2001::0::10.200.1.34::31/248
jnpr@spine1>  show route advertising-protocol bgp FC00:10:0:1::1 match-prefix
5:10.0.1.9:2001::0::10.200.1.0::31/248
bgp.evpn.0: 378 destinations, 378 routes (378 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                        Nexthop        MED  Lclpref    AS path
  5:10.0.1.9:2001::0::10.200.1.0::31/248   * FC00:10:0:1::9                     209 I
```

On the leaf nodes, routes are exported if they are accepted by both the ***LEAF_TO_SPINE_EVPN_OUT*** and ***EVPN_EXPORT*** policies.

- The ***LEAF_TO_SPINE_EVPN_OUT*** policy rejects any BGP-learned routes that carry the ***FROM_SPINE_EVPN_TIER*** community (0:14). These routes are explicitly rejected to prevent re-advertisement of spine-learned routes back into the spine layer. As described earlier, spine nodes tag all routes they advertise to leaf nodes with this community to facilitate this filtering logic.
- The ***EVPN_EXPORT*** policy accepts all routes without additional conditions.

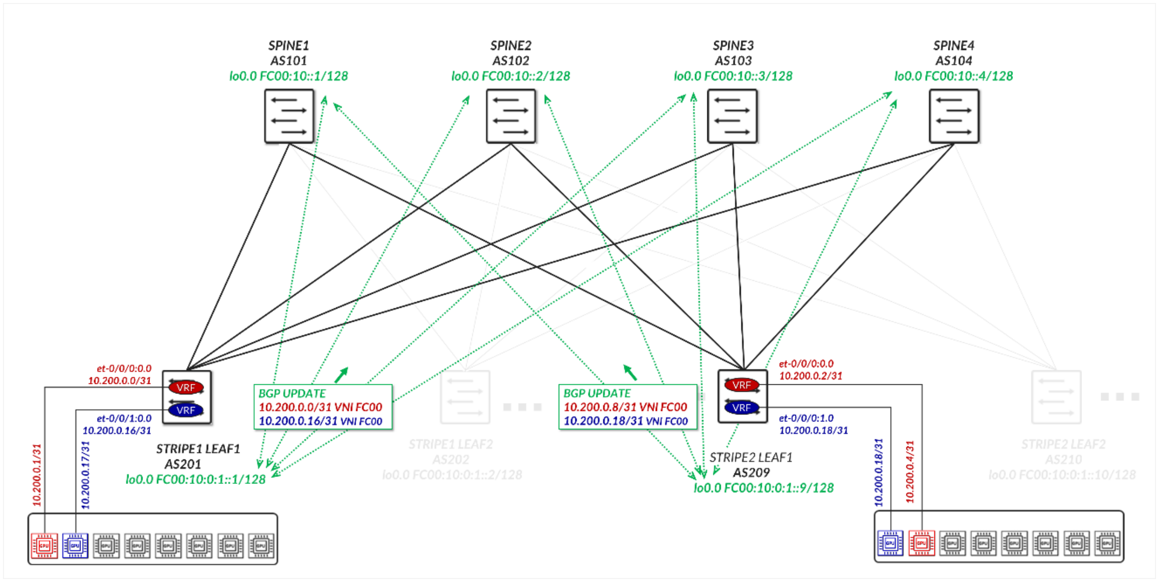
As a result, the leaf nodes export only locally originated EVPN routes for the directly connected interfaces between GPU servers and the leaf nodes. These routes are part of the **tenant routing instances** and are required to establish reachability between GPUs belonging to the same tenant.

```
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table Tenant-1
Tenant-1.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                        Nexthop         MED     Lclpref         AS path
  5:10.0.1.1:2001::0::10.200.0.0::31/248
*                               Self                    I
  5:10.0.1.1:2001::0::10.200.0.16::31/248
*                               Self                    I
jnpr@stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 table Tenant-2
Tenant-2.evpn.0: 8 destinations, 20 routes (8 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                        Nexthop MED     Lclpref    AS path
  5:10.0.1.1:2002::0::10.200.0.2::31/248
*                               Self                    I
  5:10.0.1.1:2002::0::10.200.0.18::31/248
*                               Self                    I
```

Configuration and Verification Example

Consider the following scenario where Tenant-1 has been assigned GPU 0 on Server 1 and GPU1 on Server 2, and Tenant-2 has been assigned GPU 0 on Server 2 and GPU1 on Server 1 as shown in Figure 60.

Figure 60: Overlay example with two tenants



Both Stripe 1 Leaf 1 and Leaf 2 have been configured for Tenant-1 and Tenant-2 as shown below:

Table 85. EVPN Routing-Instance for Tenant-1 and Tenant-2 Across Stripe 1 and Stripe 2

STRIPE 1 – LEAF 1	STRIPE 2 – LEAF 1
[edit routing-instances Tenant-A]	**[[edit routing-instances Tenant-A]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe2-leaf1# show | display set relative**
set instance-type vrf	set instance-type vrf
set routing-options graceful-restart	set routing-options graceful-restart
set routing-options multipath	set routing-options multipath
set protocols evpn ip-prefix-routes advertise direct-nexthop	set protocols evpn ip-prefix-routes advertise direct-nexthop
set protocols evpn ip-prefix-routes encapsulation vxlan	set protocols evpn ip-prefix-routes encapsulation vxlan
set protocols evpn ip-prefix-routes vni 20001	set protocols evpn ip-prefix-routes vni 20001
set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A	set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A
set interface et-0/0/0:0.0	set interface et-0/0/0:0.0
set interface lo0.1	set interface lo0.1
set route-distinguisher 10.0.1.1:2001	set route-distinguisher 10.0.1.9:2001
set vrf-target target:20001:1	set vrf-target target:20001:1
[edit routing-instances Tenant-B]	**[[edit routing-instances Tenant-B]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe2-leaf1# show | display set relative**
set instance-type vrf	set instance-type vrf
set routing-options graceful-restart	set routing-options graceful-restart
set routing-options multipath	set routing-options multipath
set protocols evpn ip-prefix-routes advertise direct-nexthop	set protocols evpn ip-prefix-routes advertise direct-nexthop
set protocols evpn ip-prefix-routes encapsulation vxlan	set protocols evpn ip-prefix-routes encapsulation vxlan
set protocols evpn ip-prefix-routes vni 20002	set protocols evpn ip-prefix-routes vni 20002
set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A	set protocols evpn ip-prefix-routes export BGP-AOS-Policy-Tenant-A
set interface et-0/0/0:1.0	set interface et-0/0/0:1.0
set interface lo0.2	set interface lo0.2
set route-distinguisher 10.0.1.1:2002	set route-distinguisher 10.0.1.9:2002
set vrf-target target:20002:1	set vrf-target target:20002:1

Table 86. Policies Examples for Tenant-1 and Tenant-2 Across Stripe 1 and Stripe 2

TENANT-A POLICIES	TENANT-B POLICIES
[edit policy-options policy-statement BGP-AOS-Policy-Tenant-A]	**[edit policy-options policy-statement BGP-AOS-Policy-Tenant-B]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe1-leaf1# show | display set relative**
set term BGP-AOS-Policy-Tenant-A-10 from policy AllPodNetworks-Tenant-A	set term BGP-AOS-Policy-Tenant-B-10 from policy AllPodNetworks-Tenant-B
set term BGP-AOS-Policy-Tenant-A-10 then accept	set term BGP-AOS-Policy-Tenant-B-10 then accept
set term BGP-AOS-Policy-Tenant-A-100 then reject	set term BGP-AOS-Policy-Tenant-B-100 then reject
[edit policy-options policy-statement AllPodNetworks-Tenant-A]	**[edit policy-options policy-statement AllPodNetworks-Tenant-B]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe1-leaf1# show | display set relative**
set term AllPodNetworks-Tenant-A-10 from family inet	set term AllPodNetworks-Tenant-B-10 from family inet
set term AllPodNetworks-Tenant-A-10 from protocol direct	set term AllPodNetworks-Tenant-B-10 from protocol direct
set term AllPodNetworks-Tenant-A-10 then community add TENANT-A_COMMUNITY_V4	set term AllPodNetworks-Tenant-B-10 then community add TENANT-B_COMMUNITY_V4
set term AllPodNetworks-Tenant-A-10 then accept	set term AllPodNetworks-Tenant-B-10 then accept
set term AllPodNetworks-Tenant-A-100 then reject	set term AllPodNetworks-Tenant-B-100 then reject
[edit policy-options community TENANT-A_COMMUNITY_V4]	**[edit policy-options community TENANT-B_COMMUNITY_V4]**
jnpr@stripe1-leaf1# show | display set relative	**jnpr@stripe1-leaf1# show | display set relative**
set members 5:20007	set members 5:20007
set members 21002:26000	set members 21003:26000

The routing instances create separate routing spaces for the two tenants, providing full route and traffic isolation across the EVPN/VXLAN fabric. Each routing instance has been configured with the following key elements:

- **Interfaces:**

The interfaces listed under each tenant VRF (e.g. et-0/0/0:0.0 and et-0/0/1:0.0) are explicitly added to the corresponding routing table. By placing these interfaces under the VRF, all routing decisions and traffic forwarding associated with them are isolated from other tenants and from the global routing table. Assigning an interface that connects a particular GPU to the leaf node effectively maps that GPU to a specific tenant, isolating it from GPUs assigned to other tenants.

- **Route-distinguisher (RD):**

10.0.1.1:2001 and 10.0.1.1:2002 uniquely identify EVPN routes from Tenant-1 and Tenant-2, respectively. Even if both tenants use overlapping IP prefixes, the RD ensures their routes remain distinct in the BGP control plane. Although the GPU to leaf links use unique /32 prefixes, an RD is still required to advertise these routes over EVPN.

- **Route target (RT) community:**

VRF targets 20001:1 and 20002:1 control which routes are exported from and imported into each tenant routing table. These values determine which routes are shared between VRFs that belong to the same tenant across the fabric and are essential for enabling fabric-wide tenant connectivity, for example, when a tenant has GPUs assigned to multiple servers across different stripes.

- **Protocols evpn parameters:**
  - The ip-prefix-routes controls how IP Prefix Routes (EVPN Type 5 routes) are advertised.
  - The advertise direct-nexthop enables the leaf node to send IP prefix information using EVPN pure Type 5 routes, which includes a router MAC extended community. These routes include a Router MAC extended community, which allows the remote VTEP to resolve the next-hop MAC address without relying on Type 2 routes.
  - The **encapsulation vxlan** indicates that the payload traffic for this tenant will be encapsulated using VXLAN. The same type of encapsulation must be used end to end.
  - The **VXLAN Network Identifier (VNI)** acts as the encapsulation tag for traffic sent across the EVPN/VXLAN fabric. When EVPN Type 5 (IP Prefix) routes are advertised, the associated VNI is included in the BGP update. This ensures that remote VTEPs can identify the correct VXLAN segment for returning traffic to the tenant's VRF.

Unlike traditional use cases where a VNI maps to a single Layer 2 segment, in EVPN Type 5 the VNI represents the **tenant-wide Layer 3 routing domain**. All point-to-point subnets, such as the /32 links between GPU servers and the leaf, that belong to the same VRF are advertised with the same VNI.

In this configuration, VNIs 20001 and 20002 are mapped to the Tenant-1 and Tenant-2 VRFs, respectively. All traffic destined for interfaces in Tenant-1 will be forwarded using VNI 20001, and all traffic for Tenant-2 will use VNI 20002.

Notice that the same VNI for a specific tenant is configured on both Stripe1-Leaf1 and Stripe2-Leaf1.

- **Export Policy Logic**

EVPN Type 5 routes from Tenant-1 are exported if they are accepted by the BGP-AOS-Policy-Tenant-1 export policy, which references a nested policy named AllPodNetworks-Tenant-1 (and the equivalent policies for Tenant-2)

- **Policy *BGP-AOS-Policy-Tenant-1*** controls which prefixes from this VRFs are allowed to be advertised into EVPN. It accepts any route that is permitted by the ***AllPodNetworks-Tenant-1*** policy and explicitly rejects all other routes.
- **Policy *AllPodNetworks-Tenant-1*** accepts directly connected IPv4 routes (family inet, protocol direct) that are part of the Tenant-1 VRF. It tags these routes with the **TENANT-1_COMMUNITY_V4 (5:20007 21002:26000)** community before accepting them. All other routes are rejected.

As a result, only the directly connected IPv4 routes from the Tenant-1 (/32 links between GPU servers and the leaf) are exported as EVPN Type 5 routes.

To verify the interface assignments to the different tenants, use `show interfaces routing-instance <tenant-name> terse`.

```
jnpr@stripe1-leaf1> show interfaces routing-instance Tenant-1 terse
Interface               Admin   Link    Proto           Local                 Remote
et-0/0/0:0.0            up      up      inet            10.200.0.0/31
                        multiservice
lo0.1                   up      up      inet            192.168.11.1        --> 0/0
jnpr@stripe1-leaf1> show interfaces routing-instance Tenant-2 terse
Interface               Admin   Link    Proto           Local                 Remote
et-0/0/1:0.0            up      up      inet            10.200.0.16/31
                        multiservice
lo0.1                   up      up      inet            192.168.11.2        --> 0/0
jnpr@stripe1-leaf2> show interfaces routing-instance Tenant-1 terse
Interface               Admin   Link    Proto           Local                 Remote
et-0/0/0:0.0            up      up      inet            10.200.0.2/31
                        multiservice
lo0.1                   up      up      inet            192.168.12.1        --> 0/0
jnpr@stripe1-leaf2> show interfaces routing-instance Tenant-2 terse
Interface               Admin   Link    Proto           Local                 Remote
et-0/0/1:0.0            up      up      inet            10.200.0.18/31
                        multiservice
lo0.1                   up      up      inet            192.168.12.2        --> 0/0
```

You can also check the direct routes installed to the correspondent routing table:

```
jnpr@stripe1-leaf1> show route protocol direct table Tenant-1.inet.0
Tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.0/31           *[Direct/0] 02:24:29
                         >  via et-0/0/12:0.0
192.168.11.1/32         *[Direct/0] 02:16:52
                         >  via lo0.1
jnpr@stripe1-leaf1> show route protocol direct table Tenant-2.inet.0
Tenant-2.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
```

```
+ = Active Route, - = Last Active, * = Both
10.200.0.16/31         *[Direct/0] 02:24:29
                        >  via et-0/0/12:0.0
192.168.11.1/32        *[Direct/0] 02:16:52
                        >  via lo0.2
jnpr@stripe1-leaf2> show route protocol direct table Tenant-1.inet.0
tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.2/31          *[Direct/0] 1d 17:42:33
                       >  via et-0/0/2:0.0
192.168.12.1/32        *[Direct/0] 02:16:52
                        >  via lo0.1
jnpr@stripe1-leaf2> show route protocol direct table Tenant-2.inet.0
tenant-1.inet.0: 14 destinations, 14 routes (14 active, 0 holddown, 0 hidden)
Restart Complete
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both
10.200.0.18/31         *[Direct/0] 1d 17:42:33
                       >  via et-0/0/3:0.0
192.168.12.1/32        *[Direct/0] 02:16:52
                        >  via lo0.2
```

To verify evpn l3 contexts including encapsulation, VNI, router MAC address, use `show evpn l3-context`

Use `<tenant-name> extensive` for more details.

```
jnpr@stripe1-leaf1> show evpn l3-context
L3 context             Type    Adv             Encap   VNI/Label       Router MAC/GW intf dt4-
sid                        dt6-sid                  dt46-sid
Tenant-1                               Cfg     Direct          VXLAN   20001
9c:5a:80:c1:b3:06
Tenant-2                               Cfg     Direct          VXLAN   20002
9c:5a:80:c1:b3:06
jnpr@stripe1-leaf1> show evpn l3-context
L3 context             Type    Adv             Encap   VNI/Label       Router MAC/GW intf dt4-
sid                        dt6-sid                  dt46-sid
Tenant-1                               Cfg     Direct          VXLAN   20001
58:86:70:79:df:db
Tenant-2                               Cfg     Direct          VXLAN   20002
58:86:70:79:df:db
```

```
jnpr@stripe1-leaf1> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-1
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 9c:5a:80:c1:b3:06
  Encapsulation: VXLAN, VNI: 20001
  IPv6 source VTEP address: FC00:10:0:1::1
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-1
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.1:2001
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf1> show evpn l3-context Tenant-2 extensive
L3 context: Tenant-2
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 9c:5a:80:c1:b3:06
  Encapsulation: VXLAN, VNI: 20002
  IPv6 source VTEP address: FC00:10:0:1::1
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-2
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.1:2002
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf2> show evpn l3-context Tenant-1 extensive
L3 context: Tenant-1
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 58:86:70:79:df:db
  Encapsulation: VXLAN, VNI: 20001
  IPv6 source VTEP address: FC00:10:0:1::2
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-1
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.2:2001
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf2> show evpn l3-context Tenant-1 extensive
```

```
L3 context: Tenant-2
  Type: Configured
  Advertisement mode: Direct nexthop, Router MAC: 58:86:70:79:df:db
  Encapsulation: VXLAN, VNI: 20002
  IPv6 source VTEP address: FC00:10:0:1::2
  IP->EVPN export policy: BGP-AOS-Policy-Tenant-2
  Flags: 0xc209 <Configured IRB-MAC ROUTING RT-INSTANCE-TARGET-IMPORT-POLICY RT-INSTANCE-TARGET-
EXPORT-POLICY>
  Change flags: 0x20000 <VXLAN-VNI-Update-RTT-OPQ>
  Composite nexthop support: Disabled
  Route Distinguisher: 10.0.1.2:2002
  Reference count: 5
  EVPN Multicast Routing mode: CRB
jnpr@stripe1-leaf1> show evpn ip-prefix-database
L3 context: Tenant-1
IPv4->EVPN Exported Prefixes
Prefix                                       EVPN route status
10.200.0.0/31                                Created
192.168.11.1/32                              Created
EVPN->IPv4 Imported Prefixes
Prefix                                       Etag
10.200.0.2/31                                0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2001          20001                   58:86:70:79:df:db  FC00:10:0:1::2
Accepted      n/a
192.168.12.1/32                              0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2001          20001                   58:86:70:79:df:db  FC00:10:0:1::2
Accepted      n/a
L3 context: Tenant-2
IPv4->EVPN Exported Prefixes
Prefix                                       EVPN route status
10.200.0.16/31                               Created
192.168.11.2/32                              Created
EVPN->IPv4 Imported Prefixes
Prefix                                       Etag
10.200.0.18/31                               0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2002          20002                   58:86:70:79:df:db  10.0.1.2
Accepted      n/a
```

```
192.168.12.2/32                              0
  Route distinguisher    VNI/Label/SID           Router MAC         Nexthop/Overlay GW/ESI
Route-Status  Reject-Reason
  10.0.1.2:2002          20002                   58:86:70:79:df:db  10.0.1.2
Accepted      n/a
```

When EVPN Type 5 is used to implement L3 tenant isolation across a VXLAN fabric, multiple routing tables are instantiated on each participating leaf node. These tables are responsible for managing control-plane separation, enforcing tenant boundaries, and supporting the overlay forwarding model. Each routing instance (VRF) creates its own set of routing and forwarding tables, in addition to the global and EVPN-specific tables used for fabric-wide communication. These tables are listed in Table 87.

Table 87. Routing and Forwarding Tables for EVPN Type 5

TABLE	DESCRIPTON
bgp.evpn.0	Holds EVPN route information received via BGP, including Type 5 (IP Prefix) routes and other EVPN route types. This is the control plane source for EVPN-learned routes
:vxlan.inet.0	Used internally for VXLAN tunnel resolution. Maps VTEP IP addresses to physical next-hops.
<tenant>.inet.0	The tenant-specific IPv4 unicast routing table. Contains directly connected and EVPN-imported Type 5 prefixes for that tenant. Used for routing data plane traffic.
<tenant>.evpn.0	The tenant-specific EVPN table.

When an EVPN route is received, the protocol next-hop is extracted and resolved in inet.0. The EVPN route is added to the bgp.evpn.0 table. The result is placed in :vxlan.inet.0.

The route-target community value is used to determine which tenant the route belongs to, and the route is placed in tenant.evpn.0. From there, IPv4 routes are imported into tenant.inet4.0 to be used for route lookups when traffic arrives at the interfaces belonging to the VRF.

IPv6 EBGP sessions advertising evpn routes for Tenant-1 and Tenant-2 should be established. The routes should be installed in both the bgp.evpn.0 table and the <Tenant>.inet.0 table.

```
jnpr@stripe1-leaf1> show bgp summary | no-more
---more---
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
FC00:10::1              101          5          4       0       0          18 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
FC00:10::2              102          5          4       0       0          14 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
FC00:10::3              103          5          4       0       0          10 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
FC00:10::4              104          5          4       0       0           6 Establ
  bgp.evpn.0: 0/4/4/0
  Tenant-1.evpn.0: 0/2/2/0
  Tenant-2.evpn.0: 0/2/2/0
jnpr@stripe2-leaf1> show bgp summary | no-more
---more---
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
FC00:10::1              101        206        199       0       0     1:29:40 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
FC00:10::2              102        206        199       0       0     1:29:25 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
FC00:10::3              103        206        199       0       0     1:29:26 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
FC00:10::4              104        207        199       0       0     1:29:39 Establ
  bgp.evpn.0: 4/4/4/0
  Tenant-1.evpn.0: 2/2/2/0
  Tenant-2.evpn.0: 2/2/2/0
```

To check that evpn routes are being advertised, use `show route advertising-protocol bgp <neighbor>`. For a specific route, use the `match-prefix` option and include the entire evpn prefix as shown in the example below.

```
jnpr@stripe1-leaf1>  show route advertising-protocol bgp FC00:10::1 table Tenant | match
5:10.0.1.1:2001 | match 31/248
  5:10.0.1.1:2001::0::10.200.0.0::31/248

jnpr@stripe1-leaf1>  show route advertising-protocol bgp FC00:10::1 table Tenant | match
5:10.0.1.1:2002 | match 31/248
  5:10.0.1.1:2002::0::10.200.0.16::31/248
jnpr@stripe1-leaf2>  show route advertising-protocol bgp FC00:10::1 table Tenant | match
5:10.0.1.2:2001 | match 31/248
  5:10.0.1.2:2001::0::10.200.0.2::31/248

jnpr@stripe1-leaf2>  show route advertising-protocol bgp FC00:10::1 table Tenant | match
5:10.0.1.2:2002 | match 31/248
  5:10.0.1.2:2002::0::10.200.0.18::31/248
jnpr@ stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 match-prefix
5:10.0.1.1:2001::0::10.200.0.0::31/248 table Tenant-1
Tenant-1.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                Nexthop             MED     Lclpref    AS path
  5:10.0.1.1:2001::0::10.200.0.0::31/248        *  Self                                  I
jnpr@ stripe1-leaf1> show route advertising-protocol bgp FC00:10::1 match-prefix
5:10.0.1.1:2002::0::10.200.0.16::31/248 table Tenant-2
Tenant-2.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                Nexthop             MED     Lclpref    AS path
  5:10.0.1.1:2002::0::10.200.0.16::31/248               *
Self                                    I
jnpr@stripe1-leaf2> show route advertising-protocol bgp FC00:10::1 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 table Tenant-1
Tenant-1.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                Nexthop             MED     Lclpref    AS path
  5:10.0.1.2:2001::0::10.200.0.2::31/248                *
Self                                    I
jnpr@stripe1-leaf2> show route advertising-protocol bgp FC00:10::1 match-prefix
```

```
5:10.0.1.2:2002::0::10.200.0.18::31/248 table Tenant-2
Tenant-2.evpn.0: 12 destinations, 54 routes (12 active, 0 holddown, 0 hidden)
Restart Complete
  Prefix                                Nexthop              MED     Lclpref    AS path
  5:10.0.1.2:2002::0::10.200.0.18::31/248              *
Self                                    I
```

The /248 prefixes represent EVPN route type 5 advertising each IPv4 prefix connecting the GPU servers and leaf nodes.

For example: 5:10.0.1.2:2001::0::10.200.0.0::31/248 is an EVPN route type 5 for prefix 10.200.0.0/31 where:

Table 88. EVPN Type 5 Route Advertisement Fields Description

Name	Value	Description
Route type	5:	Indicates the route is a Type 5 (IP Prefix) route
Route Distinguisher	10.0.1.2:2001	Uniquely identifies the routes
Placeholder fields	::0::	For MAC address and other Type 2-related fields (not used here)
IP Prefix	10.200.0.4::31	The actual prefix being advertised
VNI	20001	VNI to push for traffic to the destination
Advertising router	FC00:10::1 (Spine 1)	Spine the route was received from.

To check that evpn routes are being received, use `show route receive-protocol bgp <neighbor>`. For a specific route, use the **match-prefix** option and include the entire evpn prefix as shown in the example below:

```
jnpr@stripe1-leaf1> show route receive-protocol bgp FC00:10::1 | match 5:10.0.1.2:2001 | match
31
  5:10.0.1.2:2001::0::10.200.0.2::31/248
jnpr@stripe1-leaf1> show route receive-protocol bgp FC00:10::1 | match 5:10.0.1.2:2002 | match
31
  5:10.0.1.2:2002::0::10.200.0.18::31/248
jnpr@stripe1-leaf2> show route receive-protocol bgp FC00:10::1 | match 5:10.0.1.1:2001 | match
31
  5:10.0.1.1:2001::0::10.200.0.0::31/248
jnpr@stripe1-leaf2> show route receive-protocol bgp FC00:10::1 | match 5:10.0.1.1:2002 | match
```

```
31
  5:10.0.1.1:2002::0::10.200.0.16::31/248
```

The examples show routes received from Spine 1, but each route is received from all 4 spines nodes, which you can also confirm by entering:

```
jnpr@stripe1-leaf1>  show route table bgp.evpn.0 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 | match BGP
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
                   * [BGP/170] 11:31:33, localpref 100, from FC00:10::1
                     [BGP/170] 11:31:21, localpref 100, from FC00:10::2
                     [BGP/170] 11:31:14, localpref 100, from FC00:10::3
                     [BGP/170] 11:31:10, localpref 100, from FC00:10::4
jnpr@stripe1-leaf2>  show route table bgp.evpn.0 match-prefix
5:10.0.1.1:2001::0::10.200.0.0::31/248 | match BGP
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
                   * [BGP/170] 11:31:13, localpref 100, from FC00:10::1
                     [BGP/170] 11:31:41, localpref 100, from FC00:10::2
                     [BGP/170] 11:31:12, localpref 100, from FC00:10::3
                     [BGP/170] 11:31:52, localpref 100, from FC00:10::4
```

Additional information for a given route can be found using the extensive keyword:

```
jnpr@stripe1-leaf1> show route table bgp.evpn.0 match-prefix
5:10.0.1.2:2001::0::10.200.0.2::31/248 active-path extensive
bgp.evpn.0: 314 destinations, 1040 routes (314 active, 0 holddown, 0 hidden)
Restart Complete
5:10.0.1.2:2001::0::10.200.0.2::31/248  (4 entries, 0 announced)
        *BGP    Preference: 170/-101
                Route Distinguisher: 10.0.1.2:2001
                Next hop type: Indirect, Next hop index: 0
                Address: 0x55dfb9c305fc
                Next-hop reference count: 48
                Kernel Table Id: 0
                Source: FC00:10::1
                Protocol next hop: FC00:10:0:1::2
                Label operation: Push 20001
                Label TTL action: prop-ttl
                Load balance label: Label 20001: None;
                Indirect next hop: 0x2 no-forward INH Session ID: 0
                Indirect next hop: INH non-key opaque: (nil) INH key opaque: (nil)
```

```
                  State: <Active Ext>
                  Local AS:   201 Peer AS:   101
                  Age: 7:54:49    Metric2: 0
                  Validation State: unverified
                  Task: BGP_109.FC00:10::1
                  AS path: 109 210 I
                  Communities: 0:14 7:20007 21002:26000 target:20001:1
                  encapsulation:vxlan(0x8) router-mac:58:86:70:7b:10:db
                  Import Accepted
                  Route Label: 20001
                  Overlay gateway address: 0.0.0.0
                  ESI 00:00:00:00:00:00:00:00:00:00
                  Localpref: 100
                  Router ID: 10.0.0.1
                  Secondary Tables: Tenant-1.evpn.0
                  Thread: junos-main
                  Indirect next hops: 1
                    Protocol next hop: FC00:10:0:1::2 ResolvState: Resolved
                    Label operation: Push 20001
                    Label TTL action: prop-ttl
                    Load balance label: Label 20001: None;
                    Indirect next hop: 0x2 no-forward INH Session ID: 0
                    Indirect next hop: INH non-key opaque: (nil) INH key opaque: (nil)
                    Indirect path forwarding next hops: 4
                                  Next hop type: Router
                                  Next hop: FC00:10:0:2::144 via et-0/0/0:0.0
                                  Session Id: 0
                                  Next hop: FC00:10:0:2::150 via et-0/0/1:0.0
                                  Session Id: 0
                                  Next hop: FC00:10:0:2::158 via et-0/0/2:0.0
                                  Session Id: 0
                                  Next hop: FC00:10:0:2::176 via et-0/0/3:0.0
                                  Session Id: 0
                                  FC00:10:0:1::2/128 Originating RIB: inet6.0
                                    Node path count: 1
                                    Forwarding nexthops: 4
                                          Next hop type: Router
                                          Next hop: FC00:10:0:2::144 via et-0/0/0:0.0
                                          Session Id: 0
                                          Next hop: FC00:10:0:2::150 via et-0/0/1:0.0
                                          Session Id: 0
                                          Next hop: FC00:10:0:2::158 via et-0/0/2:0.0
                                          Session Id: 0
```

```
                                        Next hop: FC00:10:0:2::176 via et-0/0/3:0.0
                                        Session Id: 0
---(more)---
```

Table 89. EVPN Type 5 Route Advertisement Fields Description - Extensive

Name	Value	Description
Route type	5:	Indicates the route is a Type 5 (IP Prefix) route
Route Distinguisher	10.0.1.2:2001	Uniquely identifies the routes
Placeholder fields	::0::	For MAC address and other Type 2-related fields (not used here)
IP Prefix	10.200.105.0::24	The actual prefix being advertised
VNI	20001	VNI to push for traffic to the destination
Advertising router	FC00:10::1	Spine the route was received from.
Protocol next hop	10.0.1.2 (Stripe 1 Leaf 2)	Router that originated the EVPN route (remote VTEP)
Encapsulation	Type: 0x08	standardized IANA-assigned value for VXLAN encapsulation in the EVPN Encapsulation extended community (RFC 9014).
Route target	target:20001:1	Identifies the route as belonging to Tenant-1

To check that the routes are being imported into the correspondent tenant's routing tables, use `show route table <tenant-name>.inet.0 protocol evpn`, as shown in the example below:

```
jnpr@stripe1-leaf1> show route table Tenant-1.inet.0 protocol evpn | match /31
10.200.0.2/31   *[EVPN/170] 04:02:04
jnpr@stripe1-leaf1> show route table Tenant-2.inet.0 protocol evpn | match /31
10.200.0.18/31  *[EVPN/170] 04:02:04
jnpr@stripe1-leaf2> show route table Tenant-1.inet.0 protocol evpn | match /31
10.200.0.0/31   *[EVPN/170] 04:02:04
jnpr@stripe1-leaf2> show route table Tenant-2.inet.0 protocol evpn | match /31
10.200.0.16/31  *[EVPN/170] 04:02:04
```

Appendix D – How to Run NCCL Tests Using Autoconfigured IPv6 Address

IN THIS SECTION

To run a model or NCCL test using a global IPv6 addresses assigned either statically or automatically via SLAAC the value of the NCCL_IB_GID_INDEX variable must be adjusted.

NOTE: Starting with NCCL 2.21, the GID index no longer needs to be specified manually. It is automatically handled based on the NCCL_SOCKET_FAMILY setting. If NCCL_SOCKET_FAMILY is set to AF_INET6 and IPv6 connectivity between hosts is in place, RoCEv2 traffic over IPv6 should work as expected.

The NCCL_IB_GID_INDEX variable defines the Global ID index used by RoCE (RDMA) communication. The default value is -1, which means that NCCL will automatically select the correct GID index based on the active link layer of the InfiniBand device. If the link layer is Ethernet (RoCE), NCCL will use the GID index that returns a GID with RoCE v2 support (usually GID index 3, depending on driver/firmware).

For more details you can review Nvidia's Environment Variables documentation

To find the GID for the desired address, use the following command:

```
ibv_devinfo -vvv  -d <mellanox-interface-name> | grep GID
```

To find the mellanox interface name you can use the following script:

```
jnpr@H100-01:~/scripts$ cat nvidia_map_iface_to_mlx.sh
# Script to map network interfaces to Mellanox interfaces

echo "Network Interface to Mellanox Interface Mapping:"

# Loop through each network interface in /sys/class/net/
for iface in $(ls /sys/class/net/); do
```

```
    if [ -d /sys/class/net/$iface/device/infiniband_verbs ]; then
        # Find the Mellanox interface by reading the ibdev file
        mlx_iface=$(cat /sys/class/net/$iface/device/infiniband_verbs/*/ibdev)
        echo "$iface => $mlx_iface"
    fi
done
```

Example:

```
jnpr@H100-01:/etc/netplan$ ibv_devinfo -vvv  -d mlx5_6 | grep GID
   GID[  0]:  fe80:0000:0000:0000:a288:c2ff:fe3b:506a, RoCE v1
   GID[  1]:  fe80::a288:c2ff:fe3b:506a, RoCE v2
   GID[  2]:  0000:0000:0000:0000:0000:ffff:0ac8:010a, RoCE v1
   GID[  3]:  ::ffff:10.200.1.10, RoCE v2
   GID[  4]:  FC00:200:0000:0002:a288:c2ff:fe3b:506a, RoCE v1
   GID[  5]:  FC00:200:0:2:a288:c2ff:fe3b:506a, RoCE v2

jnpr@H100-01:~/scripts$ ./nvidia_map_iface_to_mlx.sh | egrep "gpu|Map"
Network Interface to Mellanox Interface Mapping:
gpu0_eth => mlx5_11
gpu1_eth => mlx5_6
gpu2_eth => mlx5_10
gpu3_eth => mlx5_9
gpu4_eth => mlx5_4
gpu5_eth => mlx5_3
gpu6_eth => mlx5_5
gpu7_eth => mlx5_0
```

NOTE: Make sure the GID matches in all nodes.

The easily find mapping information between the Mellanox interface names, the user assigned interface names (e.g. gpu0_eth), NICs, and the GPUs you can use the script **find_pxb_gpu_nic_pairs.py** which can be found under: https://github.com/Juniper/jvd/tree/main/Data%20Center/AIDC/backend/AI_ML_Multitenancy

Example:

```
jnpr@H100-01:~/SCRIPTS$ python3 find_pxb_gpu_nic_pairs.py
Running full GPU↔NIC mapping workflow...
```

```
Collecting GPU-NIC topology via nvidia-smi...
Converting raw_topo.txt → topo.csv ...
Parsing topo.csv and identifying PXB pairs...

Detected GPU ⇔ NIC (PXB) Pairs:
  GPU0  ↔  NIC0
  GPU1  ↔  NIC3
  GPU2  ↔  NIC4
  GPU3  ↔  NIC5
  GPU4  ↔  NIC6
  GPU5  ↔  NIC9
  GPU6  ↔  NIC10
  GPU7  ↔  NIC11

Saved to pxb_gpu_nic_pairs.txt

Building mlx5_X → NIC# mapping via mst status...

mlx5_X ⇔ NIC# Mapping:
  mlx5_0 ↔ NIC0
  mlx5_1 ↔ NIC1
  mlx5_2 ↔ NIC2
  mlx5_3 ↔ NIC3
  mlx5_4 ↔ NIC4
  mlx5_5 ↔ NIC5
  mlx5_6 ↔ NIC6
  mlx5_7 ↔ NIC7
  mlx5_8 ↔ NIC8
  mlx5_9 ↔ NIC9
Saved to mlx-to-nic-map.txt

Building gpuX_eth → NIC# mapping from PXB pairs...
gpu0_eth ↔ NIC0
gpu1_eth ↔ NIC3
gpu2_eth ↔ NIC4
gpu3_eth ↔ NIC5
gpu4_eth ↔ NIC6
gpu5_eth ↔ NIC9
gpu6_eth ↔ NIC10
gpu7_eth ↔ NIC11
Saved to gpu_eth-to-nic.txt
```

Once you have identified the GID you can run a NCCL test using:

TENANT=<TENANT#> GID=<GID> ./run-tenant.sh

which can also be found under: https://github.com/Juniper/jvd/tree/main/Data%20Center/AIDC/backend/AI_ML_Multitenancy

NOTE: The script was created for Tenants = 1-8.

Example:

```
jnpr@headend-svr-1:/mnt/nfsshare/source/aicluster/new-nccl$ TENANT=1 GID=5 ./run-tenant.sh
The RAIL partition directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/ already exists ...
Created SLURM logs directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10152025_19_16_31 ...
=== JOB SUMMARY =================================
TENANT            = 1
NODES             = 4
NODE_LIST         = H100-01,H100-02,H100-03,H100-04
PARTITION         = H100-RAILS-ALL
LOGDIR            = /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10152025_19_16_31
TEST              = all_reduce_perf  (bsize=16G, esize=16G, iters=10, agg_iters=1)
GID input         = 3(GID)  (GID_INDEX)
NCCL_IB_GID_INDEX = 3
UCX_IB_GID_INDEX  = 3
==================================================
Submitted batch job 29713

jnpr@headend-svr-1:/mnt/nfsshare/source/aicluster/new-nccl$ cat /mnt/nfsshare/logs/new-nccl/H100-
RAILS-ALL/10152025_19_16_31/slurm-29713.out|grep Avg
[1,0]<stdout>: # Avg bus bandwidth    : 47.669
```

To check if the correct GPU is being used when running a NCCL test use the following:

```
jnpr@H100-01:~/scripts$ cat show-gpu-procs.sh
#!/bin/bash
# show-gpu-procs.sh — display only the "Processes" section of nvidia-smi

set -euo pipefail

echo "| Processes:                                                                              |"
echo "|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |"
nvidia-smi | awk '/GPU   GI/{flag=1;next}/^$/{flag=0}flag'
```

Example:

```
jnpr@headend-svr-1:/mnt/nfsshare/source/aicluster/new-nccl$ TENANT=1 GID=5 ./run-tenant.sh
The RAIL partition directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/ already exists ...
Created SLURM logs directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10162025_15_24_33 ...
=== JOB SUMMARY =================================
TENANT            = 1
NODES             = 4
NODE_LIST         = H100-01,H100-02,H100-03,H100-04
PARTITION         = H100-RAILS-ALL
LOGDIR            = /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10162025_15_24_33
TEST              = all_reduce_perf  (bsize=16G, esize=16G, iters=10, agg_iters=1)
GID input         = 5(GID)  (GID_INDEX)
NCCL_IB_GID_INDEX = 5
UCX_IB_GID_INDEX  = 5
==================================================
Submitted batch job 29814
 jnpr@headend-svr-1:/mnt/nfsshare/source/aicluster/new-nccl$ TENANT=2 GID=5 ./run-tenant.sh
The RAIL partition directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/ already exists ...
Created SLURM logs directory /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10162025_15_24_36 ...
=== JOB SUMMARY =================================
TENANT            = 2
NODES             = 4
NODE_LIST         = H100-01,H100-02,H100-03,H100-04
PARTITION         = H100-RAILS-ALL
LOGDIR            = /mnt/nfsshare/logs/new-nccl/H100-RAILS-ALL/10162025_15_24_36
TEST              = all_reduce_perf  (bsize=16G, esize=16G, iters=10, agg_iters=1)
GID input         = 5(GID)  (GID_INDEX)
NCCL_IB_GID_INDEX = 5
UCX_IB_GID_INDEX  = 5
==================================================
Submitted batch job 29815

 jnpr@H100-01:~/scripts$ sudo ./show-gpu-procs.sh
| Processes:                                                                             |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|    4   N/A  N/A   4120128      C   nccl-tests/build/all_reduce_perf          51020MiB |
|    7   N/A  N/A   4119971      C   nccl-tests/build/all_reduce_perf          51020MiB |
+-----------------------------------------------------------------------------------------+
```

GPU–NIC Mapping and Topology Awareness

Make sure that the correct GPU and NIC are mapped to each Tenant. Maintaining **tight NUMA and PCIe alignment** between the assigned GPU and NIC ensures the best performance. Each tenant's GPU and NIC should be strategically **co-located within the same NUMA region and PCIe hierarchy** whenever possible.

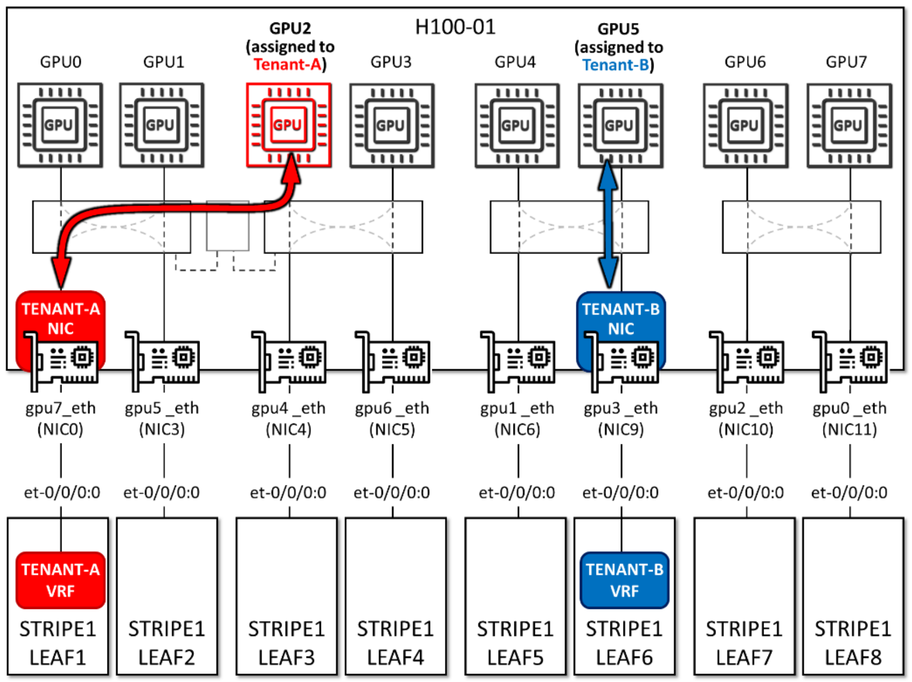
The nvidia-smi topo -m command displays the **interconnect topology** between GPUs, NICs, and CPUs in the system. The output is shown as a **matrix** where rows and columns represent devices, and each cell indicates the **connection type** (or "distance") between them. These connection types reveal how traffic flows across PCIe switches, host bridges, and CPU sockets, helping identify which GPU–NIC pairings deliver the best performance.

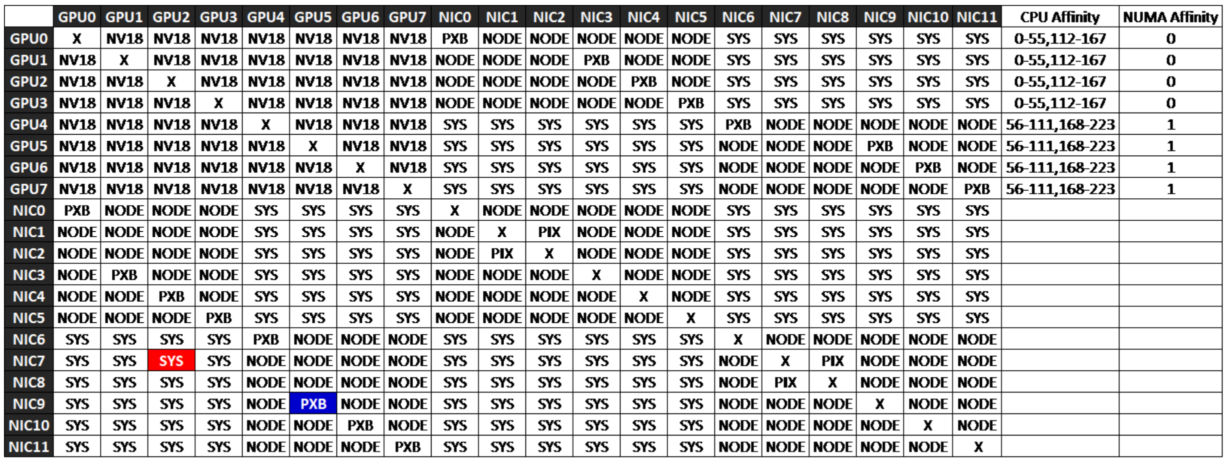
X	Same device (diagonal of the matrix)
PIX	**Single PCIe switch or bridge**. Shortest Path Fastest communication
PXB	**Multiple PCIe bridges** within the same root complex (NUMA node), but without traversing the PCIe Host Bridge. Slightly longer path and latency.
PHB	Crosses a **PCIe Host Bridge** (attached to CPU). May cross CPU boundaries. Lower performance.
SYS	Crosses **multiple PCIe Host Bridges** within the **same NUMA node**. More latency.
NODE	Crosses **NUMA nodes**, traversing **QPI/UPI interconnects** between CPU sockets. Slowest path — avoid for RDMA or latency-sensitive traffic.

For RDMA traffic, choose PXB or PIX paths for GPU↔NIC pairs to keep communication within the same NUMA domain and PCIe Host Bridge. Avoid SYS or NODE paths whenever possible, as they add unnecessary latency and reduce bandwidth efficiency.

As an example, consider a case where GPU2 and NIC0 are assigned to Tenant-A, and GPU5 and NIC9 are assigned to Tenant-B, as shown in Figure below. The nvidia-smi topo -m output in Figure ## indicates that traffic from GPU2→NIC0 must traverse multiple PCIe host bridges and cross NUMA domains, resulting in degraded performance for Tenant-A. In contrast, GPU5→NIC9 communicates through multiple PCIe bridges within the same root complex, avoiding CPU traversal and maintaining better performance for Tenant-B.

Figure 61. Tenants GPU and NIC assignment example





	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	NIC0	NIC1	NIC2	NIC3	NIC4	NIC5	NIC6	NIC7	NIC8	NIC9	NIC10	NIC11	CPU Affinity	NUMA Affinity
GPU0	X	NV18	NV18	NV18	NV18	NV18	NV18	NV18	PXB	NODE	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	0-55,112-167	0
GPU1	NV18	X	NV18	NV18	NV18	NV18	NV18	NV18	NODE	NODE	NODE	PXB	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	0-55,112-167	0
GPU2	NV18	NV18	X	NV18	NV18	NV18	NV18	NV18	NODE	NODE	NODE	NODE	PXB	NODE	SYS	SYS	SYS	SYS	SYS	SYS	0-55,112-167	0
GPU3	NV18	NV18	NV18	X	NV18	NV18	NV18	NV18	NODE	NODE	NODE	NODE	NODE	PXB	SYS	SYS	SYS	SYS	SYS	SYS	0-55,112-167	0
GPU4	NV18	NV18	NV18	NV18	X	NV18	NV18	NV18	SYS	SYS	SYS	SYS	SYS	SYS	PXB	NODE	NODE	NODE	NODE	NODE	56-111,168-223	1
GPU5	NV18	NV18	NV18	NV18	NV18	X	NV18	NV18	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	PXB	NODE	NODE	56-111,168-223	1
GPU6	NV18	NV18	NV18	NV18	NV18	NV18	X	NV18	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	PXB	NODE	56-111,168-223	1
GPU7	NV18	NV18	NV18	NV18	NV18	NV18	NV18	X	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	NODE	PXB	56-111,168-223	1
NIC0	PXB	NODE	NODE	NODE	SYS	SYS	SYS	SYS	X	NODE	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS		
NIC1	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	NODE	X	PIX	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS		
NIC2	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	NODE	PIX	X	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS		
NIC3	NODE	PXB	NODE	NODE	SYS	SYS	SYS	SYS	NODE	NODE	NODE	X	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS		
NIC4	NODE	NODE	PXB	NODE	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	X	NODE	SYS	SYS	SYS	SYS	SYS	SYS		
NIC5	NODE	NODE	NODE	PXB	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	NODE	X	SYS	SYS	SYS	SYS	SYS	SYS		
NIC6	SYS	SYS	SYS	SYS	PXB	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	X	NODE	NODE	NODE	NODE	NODE		
NIC7	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	NODE	X	PIX	NODE	NODE	NODE		
NIC8	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	NODE	PIX	X	NODE	NODE	NODE		
NIC9	SYS	SYS	SYS	SYS	NODE	PXB	NODE	NODE	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	X	NODE	NODE		
NIC10	SYS	SYS	SYS	SYS	NODE	NODE	PXB	NODE	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	X	NODE		
NIC11	SYS	SYS	SYS	SYS	NODE	NODE	NODE	PXB	SYS	SYS	SYS	SYS	SYS	SYS	NODE	NODE	NODE	NODE	NODE	X		