

Designing Data Centers for AI Clusters

Authors: Aninda Chatterjee, Vivek V

About this Document

This document is a generic design document for building network infrastructure for high-performance AI clusters.

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation.

Send your comments to design-center-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Table of Contents

| | |
|---|-----------|
| Introduction | 3 |
| AI/ML Workloads and Architecture | 3 |
| AI/ML Cluster Scale | 4 |
| Cluster Infrastructure | 5 |
| AI/ML Cluster Components | 7 |
| High-Level Design | 9 |
| Rail-Optimized Stripes | 9 |
| High-Level Front-end Network Design | 10 |
| High-Level Back-End Network Design | 11 |
| Compute-Design 1: A Rail-Optimized GPU Stripe with QFX5230-64CDs as the Spines and QFX5230-64CDs as the Leafs | 11 |
| Compute-Design 2: A Rail-Optimized GPU Stripe with PTX10008 as the Spines and QFX5230-64CDs as the Leafs | 12 |
| High-level Storage Network Design | 13 |
| Design and Implementation | 14 |
| Introduction | 14 |
| Leveraging an IP Fabric Design | 16 |
| Benefits | 16 |
| IP Services Deployed in an IP Fabric for AI Clusters | 16 |
| Dynamic Load Balancing | 17 |
| Data Center Quantized Congestion Notification (DCQCN) | 18 |
| Building Data Centers for an AI Cluster with Juniper Apstra | 19 |
| Juniper Apstra Overview | 19 |
| Back-End Network | 19 |
| Front-end Network | 30 |
| Storage Network | 33 |
| Summary | 34 |

Introduction

Artificial Intelligence (AI) has rapidly evolved over the past decade, and the demand for AI clusters to support research, development, and deployment has grown alongside it. AI clusters can be built in various sizes, tailored to specific needs and workloads. In this document, we explore the network infrastructure requirements of AI clusters delving into their workload paradigms and the Juniper Networks Data Center design approach for AI clusters.

AI/ML Workloads and Architecture

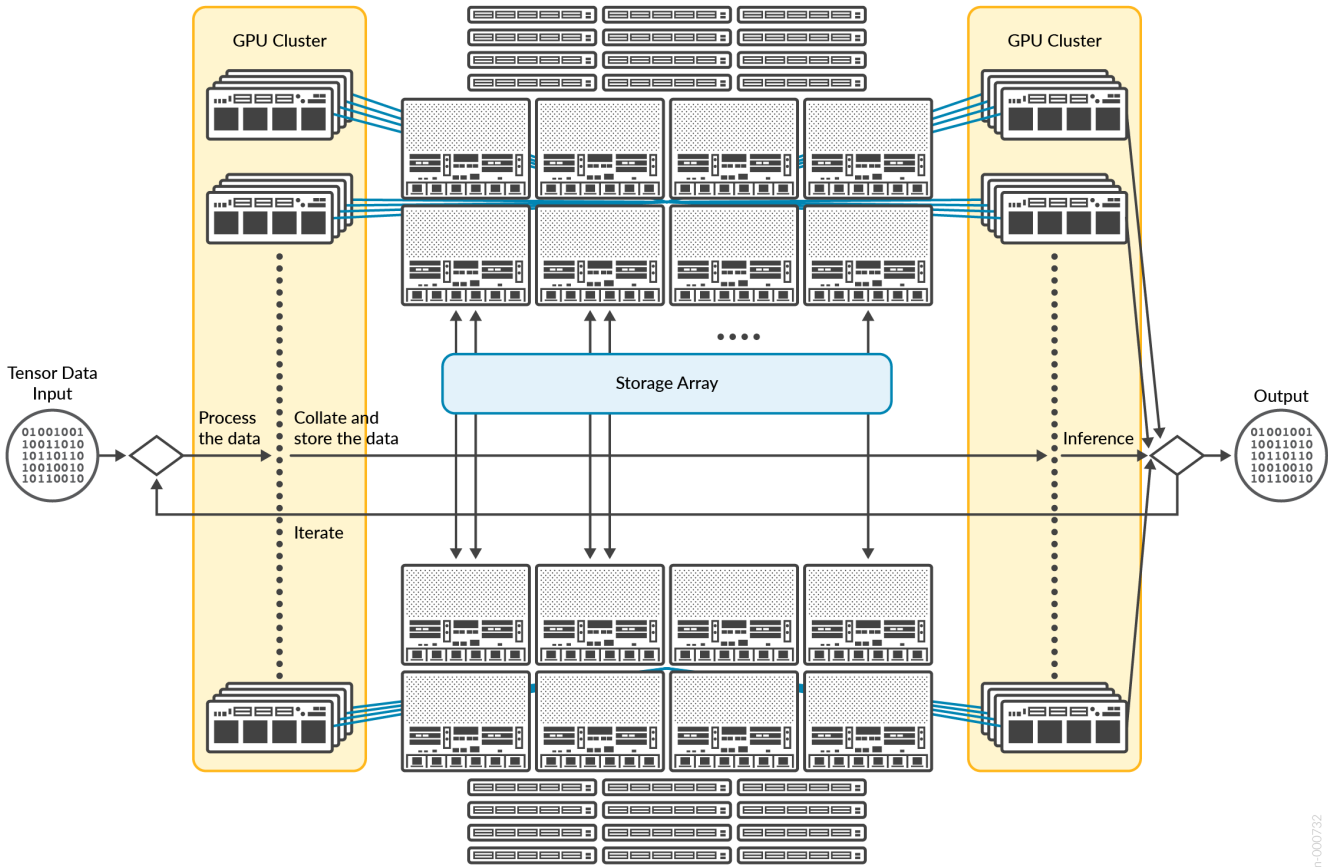
AI/ML workloads encompass a wide range of tasks and applications that leverage AI and Machine Learning (ML) techniques to analyze, understand, and make predictions from data. These workloads are at the heart of many modern technological advancements and applications, and are typically network, storage, and compute-intensive. Some of the more common workload types are:

- **Supervised Learning**—In supervised learning, models are trained using labeled datasets, where each input data point is associated with a known target or label.
- **Unsupervised Learning**—Unsupervised learning involves working with unlabeled data, where the model learns patterns and structures in the data without explicit guidance.
- **Reinforcement Learning**—Reinforcement learning involves training agents to make a sequence of decisions in an environment to maximize a reward signal.
- **Deep Learning**—Deep learning is a subset of ML that focuses on neural networks with many layers (deep neural networks).
- **Natural Language Processing (NLP)**—NLP workloads involve processing and understanding human language, enabling machines to interact with text or speech data.
- **Computer Vision**—Computer vision workloads deal with understanding and interpreting visual data, such as images and videos, to recognize objects, patterns, and scenes.
- **Time Series Analysis**—Time series analysis focuses on data that varies over time. It involves modeling and predicting future values based on historical data.
- **Recommendation Systems**—Recommendation systems use AI/ML to suggest items or content to users based on their preferences, behavior, or historical data.
- **Generative Models**—Generative models aim to generate new data that resembles existing data.
- **Anomaly Detection**—Anomaly detection workloads focus on identifying rare or unusual patterns in data that deviate from expected behavior.

The architecture of an AI/ML cluster network is counter-intuitive when compared to the conventional network architecture.

While [Figure 1](#) shows the entire cluster as one contiguous network segment, these are three separate network segments, each of which services only that aspect of the cluster. [Figure 1](#) represents the AI/ML data flow. Each GPU communicates with the back-end network through a GPU network port. The server hosting the GPUs has a dedicated storage network port that connects to external storage arrays over the storage network. Thus, the GPU/Compute nodes communicate with the storage nodes via the storage specific interfaces on the GPU server itself rather than hopping between networks. [Figure 1](#) shows further description of these segments in the respective sections.

Figure 1: High-Level Flow of GPU Training and Inference Process



Lambda is a trademark of Lambda Research Corporation, Nvidia is the trademark of Nvidia Corporation, Weka is a trademark of WekaIO Ltd.

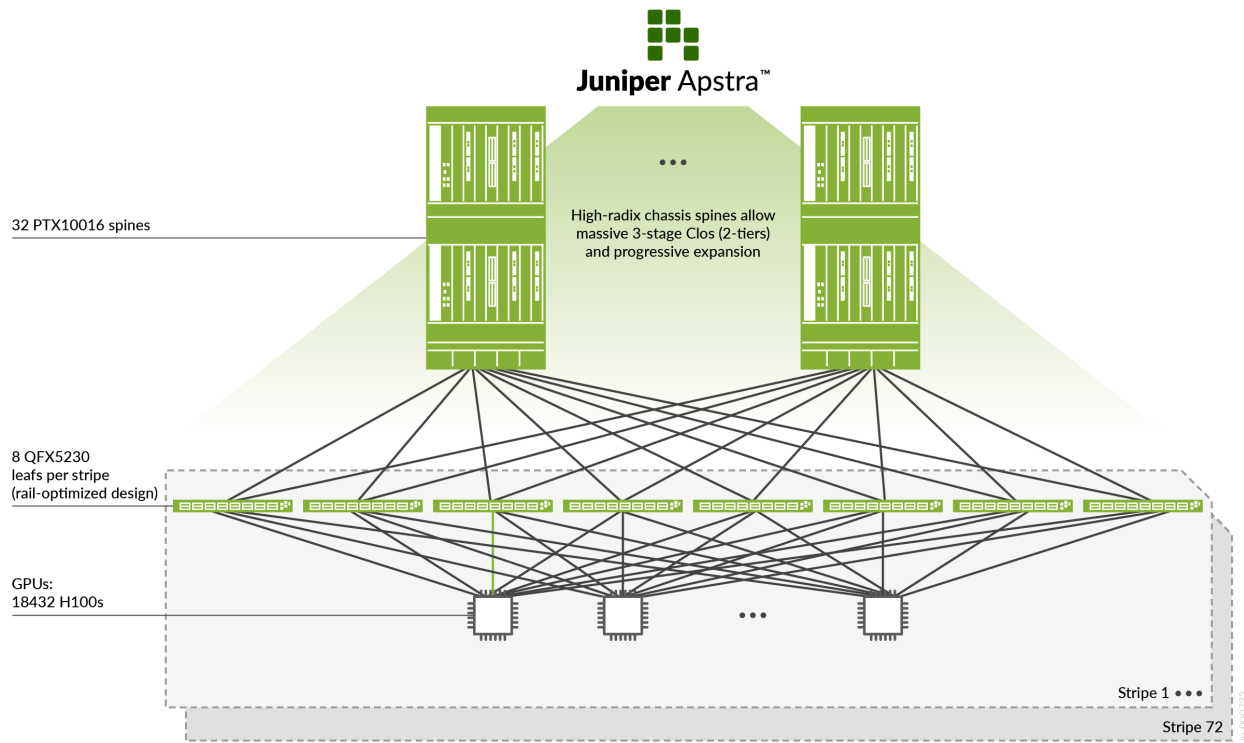
jp-000732

AI/ML Cluster Scale

The size of an AI/ML cluster varies significantly depending on the specific requirements of the workload and the scale that is necessary to meet these requirements. The number of nodes in an AI/ML cluster is influenced by factors such as the complexity of the machine learning models, the size of the datasets, the desired training speed, and the available budget. The number varies from a small cluster to a data center-wide cluster comprising of 1000s of compute, storage, and networking nodes. The scope of this document is limited to a scaled down version of an AI cluster.

The cluster size depends on the number of endpoint nodes that are present in the infrastructure. This document covers different design considerations and infrastructure elements to cater to different cluster sizes. [Figure 2](#) shows a large-scale cluster.

Figure 2: AI/ML Cluster Scale





Lambda is a trademark of Lambda Research Corporation. Nvidia is the trademark of Nvidia Corporation. Weka is a trademark of WekaIO Ltd.

Cluster Infrastructure

This section outlines the nodes (network, compute, and storage) used in this design guide. The design guide is based on Juniper QFX5220-32CD, QFX5230-64CD and PTX devices for the network infrastructure, with NVIDIA A100 and H100 GPUs for compute, and Weka storage nodes. [Table 1](#) provides a high-level overview of these nodes.

Table 1: Infrastructure Nodes Used by Juniper for the Cluster Design

| | |
|--|---|
| <p>QFX5220-32CD</p>  <p>IP fabric switch: leaf or spine scaling up to 12.8 Tbps.</p> | <p>Ports: 32 x 400GbE, 64 x 200GbE, 128 x 100GbE, 32 x 40GbE, 128 x 25GbE, 128 x 10 GbE</p> <p>Operating system: Junos OS Evolved</p> <p>Dimension: 1U rackmount, 17.26 x 1.72 x 21.1 in. (43.8 x 4.3 x 53.59 cm) W x H x D</p> |
| <p>QFX5230-64CD</p>  <p>IP fabric switch: leaf or spine scaling up to 25.6 Tbps</p> | <p>Ports: Up to 64 x 400GbE, 128 x 200GbE, 256 x 100GbE, 64 x 40GbE, 256 x 25GbE, 256 x 10GbE ports</p> <p>Operating system: Junos OS Evolved</p> <p>Dimensions: 2U rackmount, 17.4 x 3.43 x 25.6 in. (44.2 x 57.76 x 81.28 cm) W x H x D</p> |

PTX10008



Ports: Up to 288 x 400GbE, 1152 x 100GbE, 288 x 40GbE, 1152 x 25GbE, and 1152 x 10GbE

Operating system: Junos OS Evolved

Dimensions: 13U rackmount, 17.4 x 22.55 x 32 in. (44.2 x 57.76 x 81.28 cm) W x H x D

IP fabric switch: spine scaling up to 115.2 Tbps.

Lambda Hyperplane8 HGX-A100



Operating system: Ubuntu 22.04: Includes Lambda Stack for managing TensorFlow, PyTorch, CUDA, cuDNN, etc.

Processor: 2x AMD EPYC 7763: 64 cores, 2.45~3.5GHz, 256MB cache, PCIe 4.0

GPUs: 8x NVIDIA A100 (80GB) SXM4: HGX platform with NVLink and NVSwitch fabric

Standard networking: 1x NVIDIA ConnectX-6 Dx adapter card, 100GbE, dual-port QSFP28, AIOM PCIe 4.0 x16

OS drives: 2x 1.92 TB M.2 NVMe: Data center SSD, 1 DWPD, PCIe 4.0

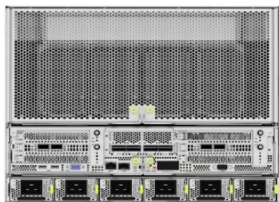
Storage Networking: 1x 200 Gbps NVIDIA ConnectX-6 VPI NIC: Dual-port QSFP56, HDR InfiniBand/Ethernet

Data drives: 6x 3.84 TB U.2 NVMe: Data center SSD, 1 DWPD, PCIe 4.0

GPU Direct RDMA Networking: 8x NVIDIA ConnectX-7 Adapter Card 200Gb/s NDR200 IB Single-port QSFP PCIe 4.0 x16

Dimensions: 4U rackmount, 6.9 x 17.6 x 35.4 in (174 x 446 x 900 mm) HxWxD

NVIDIA DGX H100



DGX H100 System: DGX H100 System, 80GB, 10 NIC, Standard Support, 3 Years

GPUs: 8x NVIDIA H100 80GB Tensor Core GPUs with NVLink and NVSwitch Fabric

Processor: Dual Intel Xeon Platinum 8480C Processors: 112 Cores total, 2.00 GHz (Base), 3.80 GHz (Max Boost) Memory: 2TB DDR5

Networking (GPUs): 4x OSFP ports serving 8x single-port NVIDIA ConnectX-7 VPI: 400 Gb/s InfiniBand/Ethernet

OS Storage: 2x 1.92TB M.2 NVMe drives

Internal storage: 8x 3.84TB U.2 NVMe drives

Networking (Storage): 2x dual-port NVIDIA ConnectX-7 VPI: 1x 400 Gb/s InfiniBand/Ethernet

Dimensions: 8U rackmount, 14 x 19.0 x 35.3 in (174 x 446 x 900 mm) HxWxD

Weka Storage node



Operating system: 1x Ubuntu | 22.04

Processor: 1x AMD EPYC 9454P (48-core, 2.75~3.8GHz, 256MB cache, 290W)

OS drives: 2x 1.92 TB M.2 NVMe: Data center SSD, 1 DWPD, PCIe 4.0

Data drives: 7x 7.68 TB U.2 NVMe: Data center SSD, 1 DWPD, PCIe 4.0

Networking (front-end): 1x NVIDIA ConnectX-6 Dx adapter card, 100GbE, dual-port QSFP28, PCIe 4.0 x16

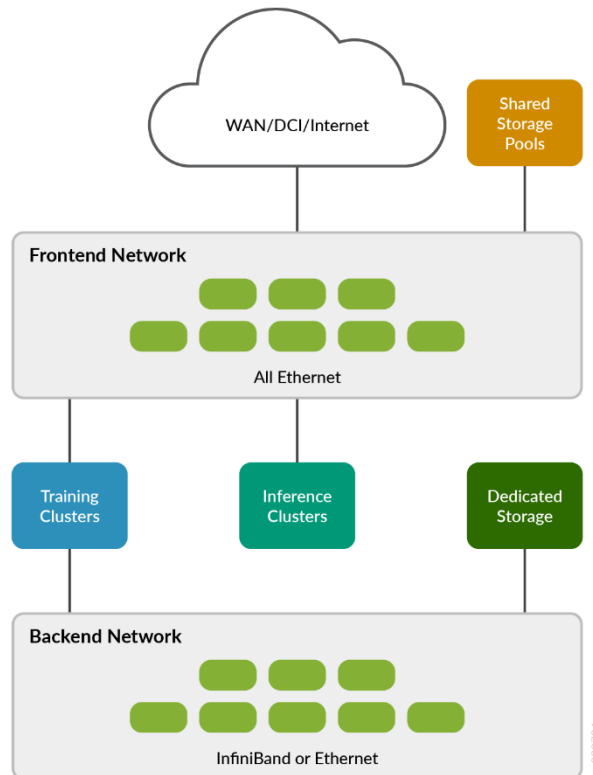
Software: 3-year Weka Flash tier license w/ Snapshot and high perf protocol services - POSIX + NFS-W + S3 + SMB-W

Networking (storage): 2x NVIDIA ConnectX-6 VPI adapter card, HDR IB (200Gb/s) and 200GbE, dual-port QSFP56, OCP 3.0

The components shown in [Table 1](#) are part of the overall AI/ML cluster anatomy, as shown in [Figure 3](#), wherein a front-end network connects to external users and data, and a back-end network supports the AI model training functions of the training clusters.

AI/ML Cluster Components

Figure 3: AI/ML Cluster Anatomy



Cluster Components:

- Training Cluster
- Inference Cluster
- Shared Storage Pools
- Dedicated Cluster Storage

Cluster Networks:

- Frontend:
 - Inference clusters use this network
 - Shared storage pools
 - Management network for training
- Backend:
 - GPU Compute Fabric
 - Dedicated Storage Fabric (may be converged with compute)

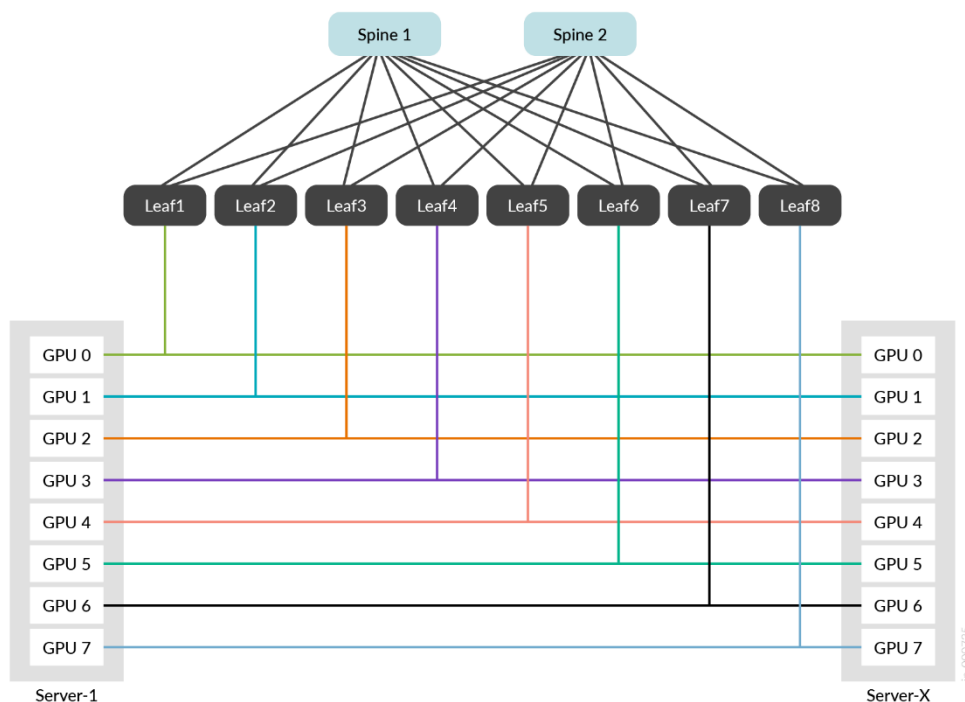
NOTE: A separate Intelligent Platform Management Interface (IPMI)/Out-of-Band (OOB) management network exists for the overall AI/ML cluster.

High-Level Design

The following sections lay out considerations of the AI Cluster design, focused on training clusters (and not inference clusters, whose overall design may vary in terms of GPU and storage nodes).

Rail-Optimized Stripes

Figure 4: Rail-Optimized Stripes



In designing the network infrastructure for an AI cluster, the key objectives are to provide maximum throughput, minimal latency, and minimal network interference for AI traffic flows. A Rail-Optimized Stripe, proposed by NVIDIA, is a design that extends from the compute nodes to the Top of Rack leaves, which takes the network requirements that are necessary for AI clusters into consideration.

From the perspective of the network infrastructure, this design is similar to a Layer 3 Clos fabric, that is common across most modern data center deployments. For example, for the DGX H100 compute servers (which has 8 GPUs and 8 NICs in the server), any-to-any GPU communication within the server is achieved via high throughput NVLink channels attached to a NVSwitch. In addition to this, each NIC in the server connects to a unique Top of Rack leaf (NIC1 to leaf1, NIC2 to leaf2, and so on), and the same methodology is followed on all servers, achieving a 'rail' design. [NVIDIA documentation](#) provides more information on such designs.

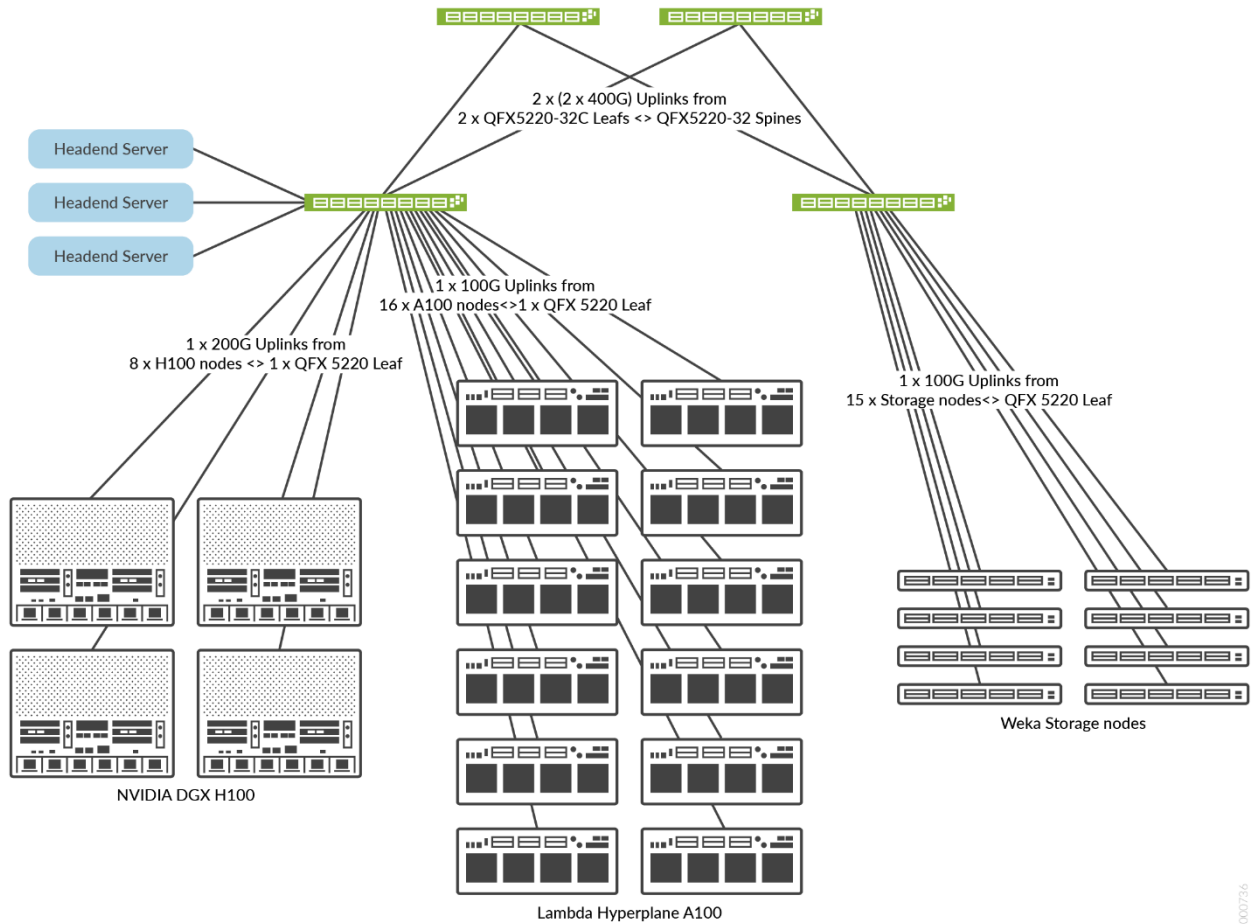
With such a design, along with optimizations such as PXN (PCIe x NVLink), network interference is minimized by moving data to a GPU on the same rail as the destination, and thus sending data to the destination without crossing rails, which minimizes the number of network hops required. This GPU/NIC connectivity for rail-optimized stripes is explored in more detail in the design and implementation section.

The high-level design for the back-end network uses a rail-optimized stripe as a basic building block and replicates the same design to scale up an AI cluster size.

High-Level Front-end Network Design

The front-end network for the AI Cluster provides the connections external to the cluster for users and data.

Figure 5: Frontend Network Architecture



Lambda is a trademark of Lambda Research Corporation, Nvidia is the trademark of Nvidia Corporation, Weka is a trademark of WekaIO Ltd.

jp-000736

Some of the design considerations for the front-end network are:

- The front-end network deals with elements such as orchestration of the training and possibly servicing inference if any GPUs are dedicated to inference, handling Application Programming Interface (API) calls from end users to deliver the inference from the model, telemetry, and so on.
- Thus, the front-end network is not expected to receive the same amount of network traffic and data flows as the back-end, which deals with the training methods and storage. Therefore, in the front-end network design, we chose to use the QFX5220-32CD devices as both the leaves and spines with the port mapping described below.
- To have an oversubscription ratio that is equal to 1, in this design, there are 16 x 100G links from the A100 servers and 8 x 200G links from the H100 servers, which at full capacity is a network load of 3200Gbps. The uplinks from the QFX5220-32CD leaves to the QFX5220-32CD spines at 2 x 2*400G uplinks per spine that brings the total upward bandwidth to 3200 Gbps, thus maintaining the oversubscription ratio at 1.

- The storage nodes in use (Weka storage nodes) have 1 x 100G links per node going to the leaf. For the purposes of parity in the design, the uplink from the storage leaf to the spine is also set at 3200 Gbps. Based on the requirements, this can either be brought down to 1600 Gbps by removing uplinks or provisioned for an additional 15 storage nodes maintaining the oversubscription ratio at 1.

High-Level Back-End Network Design

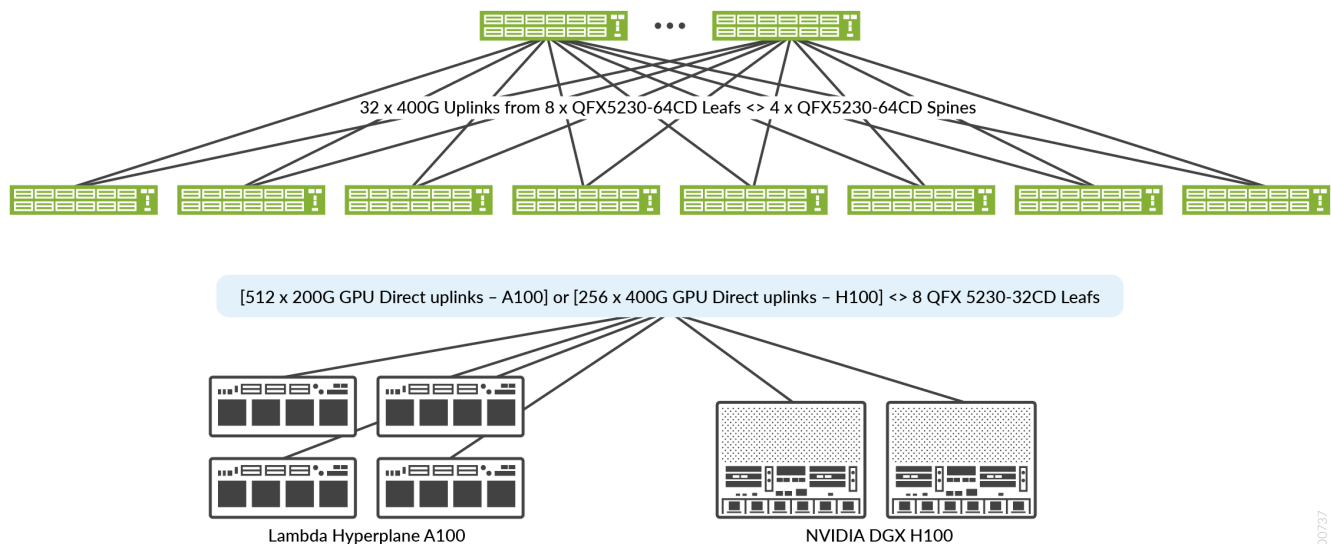
The design of the back-end network of the AI cluster as shown in [Figure 3](#), can be divided into two segments:

- Compute or GPU network.
- Storage network.

This document covers three variants of example topologies for the compute and storage networks that are differentiated by the devices used in various roles and the port density and capacity of the devices in use, that is, cluster size. These designs follow customer use cases and GPU vendor recommendations of 1:1 oversubscription. We work closely with our customers in understanding the specific network needs for their training workloads, including cluster size and possible oversubscription efficiencies.

Compute-Design 1: A Rail-Optimized GPU Stripe with QFX5230-64CDs as the Spines and QFX5230-64CDs as the Leafs

Figure 6: Rail-Optimized and Network Optimized Design with QFX5230-64CDs in Spine and Leaf Roles



Lambda is a trademark of Lambda Research Corporation. Nvidia is the trademark of Nvidia Corporation. Weka is a trademark of WekaIO Ltd.

jr-000737

While device capacity and ability to handle large traffic is an important design consideration, another important aspect is the cost of implementation. For smaller cluster sizes, high port density spines are not necessary. The design specifications of the above design are:

- In [Figure 6](#), each stripe has 64 x A100 GPUs OR 32 x H100 GPUs connected to a leaf. It provides a view of two different types of GPU clusters (A100-based cluster and H100-based cluster), enabling a more informed decision-making process, when choosing one type of cluster over the other.
- A100 GPU network port has a capacity of 200 Gbps, and the H100s have a GPU network port capacity of 400 Gbps.

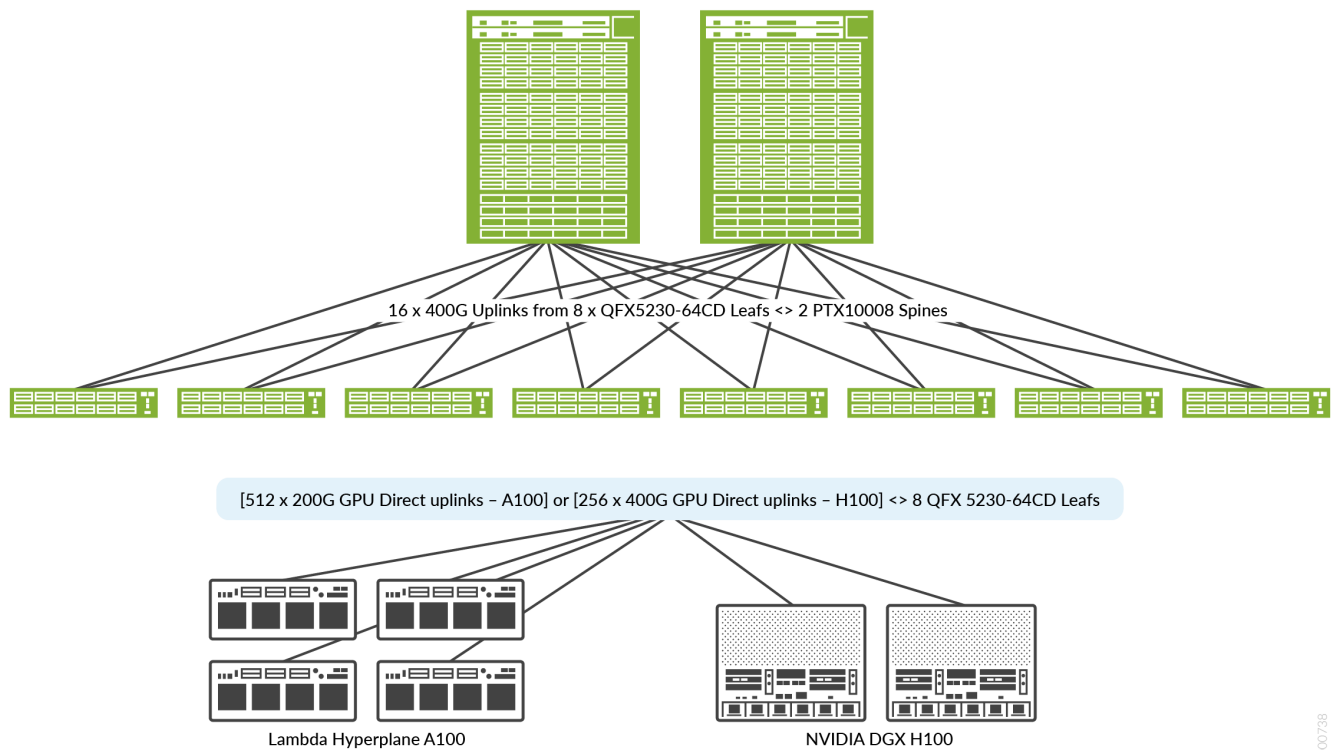
- Therefore, each leaf receives an ingress of 12.8 Tbps at maximum load. Hence, the leaf has 32x400 Gbps links going to each of the four spines resulting in an egress capacity of 12.8 Tbps, which maintains the oversubscription ratio at 1.

The scale calculation in this design is as follows:

- Each QFX5230-64CD spine has 64 x 400G ports.
- 64 ports on the leaf are split into 32 x 400G uplinks x 8 leaves = 256 uplinks divided among four spines and 32 x 400G downstream ports, which are further split into 64 x 200G ports supporting 64 A100 nodes each or 32 H100s each, or a mix of both.
- Thus, the cluster size is 64 A100 GPUs x 8 leaves = 512 A100 GPUs or 256 H100s GPUs per stripe.

Compute-Design 2: A Rail-Optimized GPU Stripe with PTX10008 as the Spines and QFX5230-64CDs as the Leafs

Figure 7: Rail-Optimized and Network Optimized Design with QFX5220-32CDs as Leafs and PTX10008s as Spines



Lambda is a trademark of Lambda Research Corporation, Nvidia is the trademark of Nvidia Corporation, Weka is a trademark of WekaIO Ltd.

jin-000738

The scale calculation in this design is as follows:

- In this case, a PTX10008 spine includes 8 slots with 36 x 400G line cards, which means the device has a total of 288 x 400G ports per spine.
- The leaves include 32 x 400G uplinks x 8 Leaves = 256 ports split among two spines, that is, 128 ports per spine.
- Based on the above calculation, the spines include 160 free ports. Since the PTX is a modular chassis, the number of slots are chosen to accommodate the appropriate number of stripes.

- Since the leaves are QFX5230-64CDs, the GPU nodes per stripe are 64 A100 nodes x 8 leaves = 512 A100 GPUs or 256 H100 GPUs per stripe.
- It is important to note that a modular spine allows for easier progressive growth by simply investing in additional line cards and new cabling as more stripes are added to the overall cluster. There is no re-cabling that is necessary.

High-level Storage Network Design

Figure 8: Back-end Storage Design with QFX5220-32CDs as Leaves and Spines

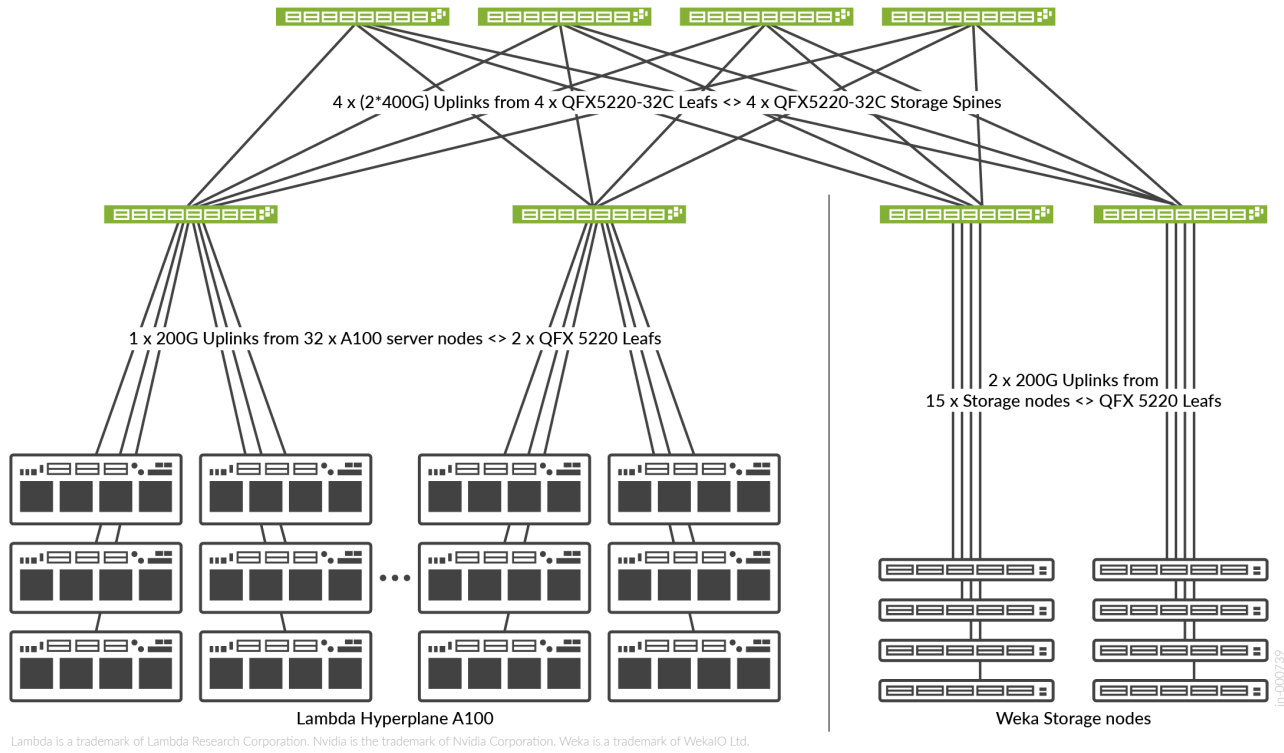


Figure 8 shows the storage network design comprising of two main segments:

- GPU storage node segment (Hyperplane8-A100 in the above example).
- Dedicated storage node segment (Weka AMD storage nodes in the above example).

Though there is no concept of rail-optimization in a storage cluster, it is recommended that the rail-optimized and network optimized stripe design is followed because there is a direct relationship between the number of minimum storage nodes required for a certain number of GPUs as shown in the Table 2. For more information regarding the storage nodes, see [AMD documentation](#).

Table 2: GPU Count to Storage Node Requirement Matrix (Source: AMD)

| GPU Count | Required Aggregate Throughput (GBytes/s) | Required Aggregate Throughput (Gbits/s) | Throughput per Storage Node (GBytes/s) | Storage Nodes Required |
|-----------|--|---|--|------------------------|
| 128 | 256 | 2048 | 36 | 8 |
| 256 | 512 | 4096 | 36 | 15 |
| 512 | 1024 | 8192 | 36 | 29 |

| GPU Count | Required Aggregate Throughput (GBytes/s) | Required Aggregate Throughput (Gbits/s) | Throughput per Storage Node (GBytes/s) | Storage Nodes Required |
|-----------|--|---|--|------------------------|
| 1024 | 2048 | 16,384 | 36 | 57 |
| 2048 | 4096 | 32,768 | 36 | 114 |

Since there are 32 A100 servers, that is, 256 GPUs in 1 stripe of our GPU cluster, the storage design includes 15 dedicated storage nodes. The specifications and design considerations are listed below:

- The corresponding GPU network design associated with this storage network are as shown in [Figure 6](#) and [Figure 7](#).
- Other designs that include more GPUs per stripe require a modification to the number of corresponding dedicated storage nodes.
- The storage network design, as shown in [Figure 10](#) has the following characteristics:
 - The Compute-Storage segment has 32 A100s each with a 200G link to the leaf nodes resulting in north bound traffic at a maximum capacity of 6.4T split amongst 2 x QFX5220-32CD leaf nodes (3.2T per leaf).
 - The dedicated storage segment has 15 x Weka storage nodes sending 2 x 200G links north-bound to the QFX5220-32CD leaf nodes (for the purpose of this design), which results in an overall load of $2 \times 200 \times 15 = 6000$ Gbps or 6T, again split amongst 2 x QFX5220-32CD leaf nodes.
 - All four QFX5220-32CD leaf nodes in this design have 8 x 400G northbound links (two per spine) bringing the overall egress capacity to 6.4T, thus maintaining an over-subscription ratio of 1:1.
 - In the dedicated storage segment, the leaf to spine configuration remains the same. Since the number of storage nodes is 15, the ingress to egress traffic capacity on the leaf nodes is 6Tbps: 6.4Tbps, making it slightly undersubscribed.
 - The QFX5220-32CD spines have 32 x 400G ports, and in this design, only eight of those 32 ports are in use. This is done with redundancy and expansion in mind, and these spines at full capacity support four times the scale shown in [Figure 8](#) and hence four times the number of end nodes as well.
 - Another variant of this design is with two spines instead of four, which still provides the same link level redundancy because it utilizes only 16 of the 32 ports but the node level redundancy changes, and it limits the expansion to two times the design instead of four.

Design and Implementation

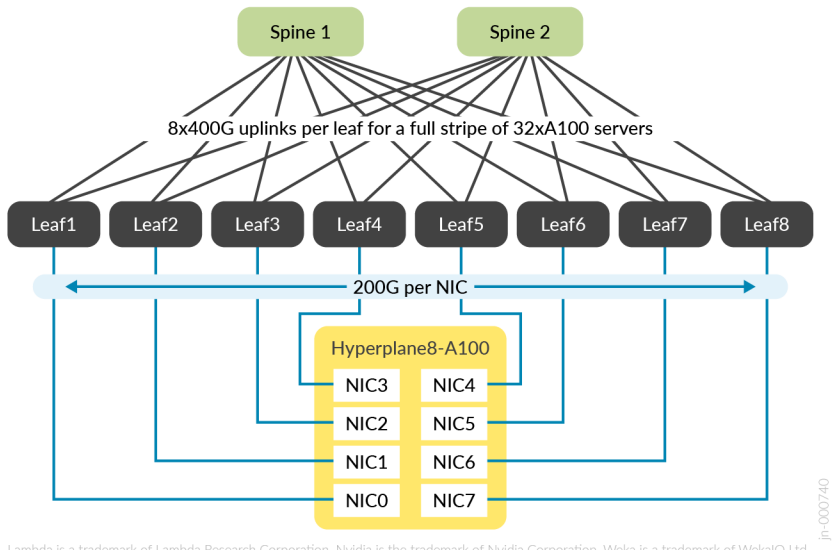
Introduction

In this section, an example design demonstrates how a network infrastructure is orchestrated with Juniper Apstra.

From the perspective of physical connectivity, the three networks (back-end, front-end, and storage) are each cabled to build a Layer 3 Clos fabric individually. This implies that every leaf connects to every spine via point-to-point Layer 3 links. The rail-optimized stripe design is followed, with the same NIC of every GPU cluster connecting to the same leaf in the back-end network.

Figure 9 shows a snippet of the back-end network, considering a port density of 32 x 400G per leaf. The focus is on one Hyperplane8-A100 server, which includes a total of 8 x A100 GPUs and 8 x 200G NICs. Each NIC connects to a unique leaf, following the rail-optimized design.

Figure 9: Hyperplane8-A100 Connectivity to Back-End Network

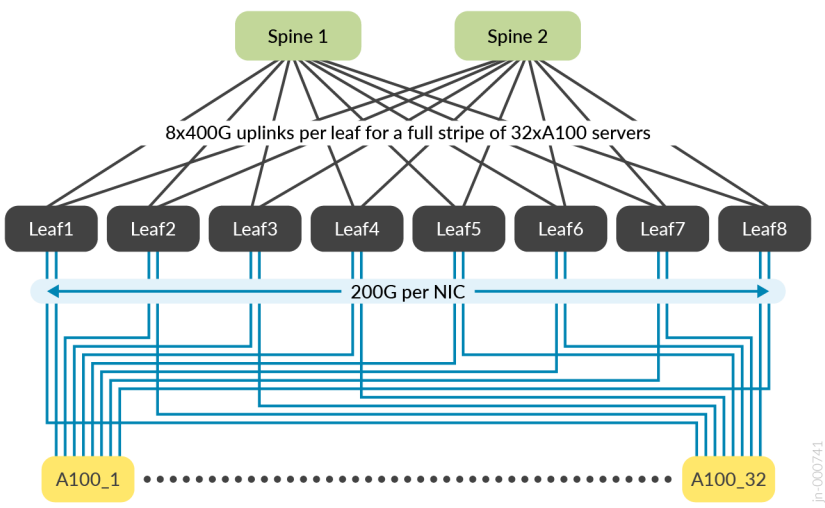


Lambda is a trademark of Lambda Research Corporation. Nvidia is the trademark of Nvidia Corporation. Weka is a trademark of WekaIO Ltd.

Each leaf receives 200G from one A100 server as only one 200G NIC maps to one GPU and connects to one leaf per server. Expanding this to a full stripe of 32 Hyperplane8-A100s, this implies that one leaf receives 32 x 200G. Since the oversubscription ratio must be 1, the uplinks must also match the same bandwidth. With two spines, each leaf must have 8 x 400G uplinks per spine to meet this requirement.

Thus, for our design and implementation, Figure 10 shows the back-end network topology that demonstrates how Juniper Apstra builds such a fabric.

Figure 10: Example Topology for Implementation of Back-End Network



Lambda is a trademark of Lambda Research Corporation. Nvidia is the trademark of Nvidia Corporation. Weka is a trademark of WekaIO Ltd.

For the leafs, QFX5220-32CDs are used, and for the spines, PTX10008s are used with 36x400G line cards, allowing for large scope of expansion.

Leveraging an IP Fabric Design

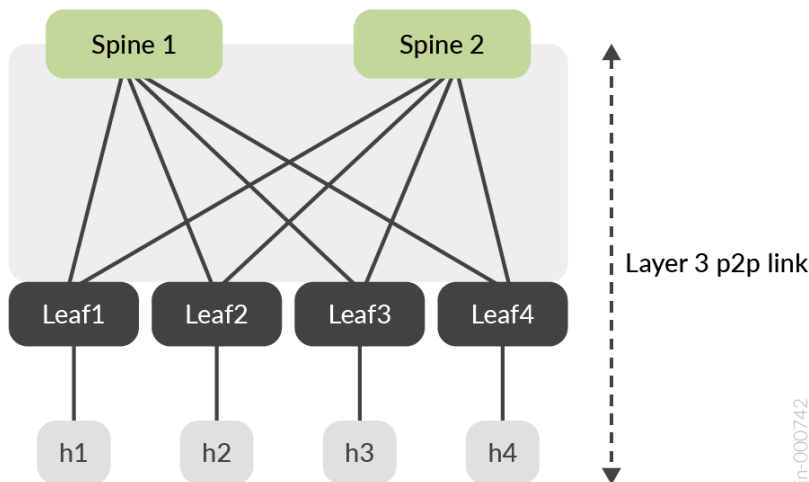
An IP fabric is a routed infrastructure, designed using a fat-tree, Clos type architecture. Such a design provides uniform capacity at each layer, eliminating the need for traditional Layer 2 technologies such as Spanning Tree Protocol (STP) by using Layer 3 links only. Instead, a routing protocol is used for injecting prefix information into the fabric. An IP fabric forms the underlay in modern data center designs and scales to large data center fabrics.

An IP fabric typically extends to the hosts, fully eliminating any Layer 2 links in the network and relying on Layer 3 routing protocols for convergence. As shown in [Figure 11](#), with routing protocols enabled over the Layer 3 point-to-point links between the leafs and the spines, each leaf has equal cost paths up to the spines, leveraging ECMP hashes to distribute packets across multiple links. A routing protocol or static routing is used between the leafs and the hosts.

Benefits

- Better utilization of all available paths through ECMP.
- Faster convergence.
- Elimination of Layer 2 links and the need for traditional Layer 2 protocols such as STP.

Figure 11: End to End L3 Clos Fabric Design



IP Services Deployed in an IP Fabric for AI Clusters

Typical data center fabrics do not require fine tuning of the fabric for traffic flows; however, AI clusters are unique in the kind of requirements that they need from the network infrastructure. The traffic pattern is high density, low entropy traffic which implies that the network infrastructure commonly sees elephant flows, with little variation in the flows themselves.

Due to these traffic characteristics, despite having equal cost paths from leafs to spines, the links might not get properly utilized since the default hash uses a combination of Layer 3 header information. As the variation in flows is not enough, it is possible that one link might be highly utilized, leading to these elephant flows causing drops of mice flows (low bandwidth flows) and possible flow collisions. For these reasons, Dynamic Load Balancing (DLB) is recommended to be configured on the leafs of an IP Fabric for AI clusters.

In addition, AI cluster generated traffic requires a lossless network. In this case, since the traffic on the back-end GPU network is RDMA over Converged Ethernet (RoCEv2), there is a need to configure congestion control methods. DCQCN (Data Center Quantized Congestion Notification) is used in this case.

Dynamic Load Balancing

Static load balancing uses packet headers only (such as information from the Layer 3 header) to determine the egress interface for a packet when multiple equal cost paths are available. This design causes a per-flow hash, where the packets from the same flow are always sent over the same interface amongst available equal cost paths. However, this can cause poor utilization of interfaces when dealing with traffic from AI clusters, which are high density, elephant flows with little variation in terms of the flows themselves, since static load balancing does not consider the usage of individual equal cost links.

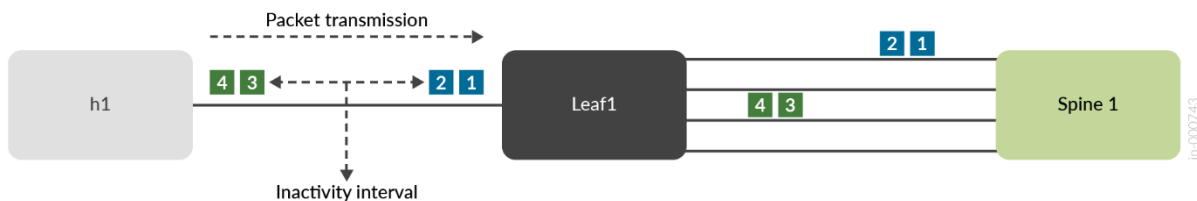
While it might be common to consider per-packet splitting of traffic across equal paths, such a design has large implications and causes packet reordering. This is where DLB helps. DLB can be initiated per flowlet, where flowlets burst of the same flow, separated by a period of inactivity. DLB works on the principle that if the transmission latency between equal cost paths is lower than the period between flowlets (called the inactivity interval), then the next packet in the flow is sent over an underutilized link, with the assurance that the packets for the overall flow arrive in the correct order, despite being sent over different links.

On a QFX5220-32CD, DLB is implemented using the following configuration:

```
*snip*
forwarding-options {
    enhanced-hash-key {
        ecmp-dlb {
            flowlet {
                inactivity-interval 16;
            }
            ether-type {
                ipv4;
            }
        }
    }
}
policy-options {
    policy-statement DLB {
        then {
            load-balance per-packet;
        }
    }
}
routing-options {
    forwarding-table {
        export DLB;
    }
}
*snip*
```

In this case, consider multiple packets from the same flow arrive at leaf1 of a data center network and need to be sent over four possible equal cost paths towards spine1.

Figure 12: DLB Packet Flow



Packets 1 through 4 are part of the same flow, however, there is a delay period (in microseconds) between packets 2 and 3. In this case, DLB treats packets 1 and 2 as part of the same flowlet, and packets 3 and 4 as part of another flowlet, based on the configured inactivity interval.

DLB, typically implemented as an engine in the Broadcom ASIC, is constantly fed information about the link usage of the equal cost paths, and each link is assigned a quality band. This is a dynamic assignment that constantly changes as the link utilization changes. DLB determines if the time taken to transmit packet 2 is lower than the inactivity interval between packet 2 and 3, then even if packet 3 is sent over a different link, it will arrive only after packet 2 has arrived. This eliminates any packet reordering issues, despite sending packets of the same overall flow over different links.

Data Center Quantized Congestion Notification (DCQCN)

Modern data centers for high performance computing (HPC) or AI clusters have requirements of high throughput at low latency. These requirements are not easily met by standard TCP/IP stacks as it causes a lot of CPU overhead and instead, leverages RoCEv2 (RDMA over Converged Ethernet v2). Unlike Infiniband, RoCEv2 can be easily integrated into existing Ethernet-based data centers, lowering infrastructure costs, and providing more flexibility.

RoCEv2 provides lossless network infrastructure that is needed for HPC and AI clusters, with the configuration of some additional features such as Priority Flow Control (PFC) and Explicit Congestion Notification (ECN). DCQCN is a combination of both PFC and ECN.

PFC works on receive buffers of an interface, by sending back Pause Frames when the receive buffer crosses a specific threshold. These Pause Frames indicate to the recipient to stop sending packets for a specific interval of time, avoiding buffer overflow by pausing data transmission. However, there are downsides to using PFC in the network – sustained Pause Frames cause ingress port congestion and dropped packets. Thus, PFC often leads to poor overall application and network performance due to head of line blocking (creating victim flows) and unfairness (also known as the parking lot problem). Additionally, PFC operates at interface level (queue level, specifically) and cannot distinguish between flows.

ECN, on the other hand, is an end-to-end congestion notification method between an ECN-enabled sender and an ECN-enabled receiver. The general methodology followed by DCQCN is to allow ECN to kick-in earlier than PFC, enabling flow control for ECN marked traffic by decreasing the transmission rate of the sender when it receives a notification to do so.

The correct operation of DCQCN requires balancing two conflicting requirements:

- Ensuring PFC is not triggered too early before giving ECN a chance to send congestion feedback to slow the flow.
- Ensuring PFC is not triggered too late, thereby causing packet loss due to buffer overflow.

Building Data Centers for an AI Cluster with Juniper Apstra

Juniper Apstra Overview

Juniper Apstra is a multivendor Intent Based Network System (IBNS), orchestrating data center deployments, and managing small to large scale data centers through Day-0 to Day-2 operations. It is an ideal tool to build data centers for AI clusters, providing invaluable Day-2 insights through monitoring and telemetry services.

Deploying a data center fabric through Juniper Apstra is a modular function, leveraging various building blocks to instantiate a fabric. These basic building blocks are as follows:

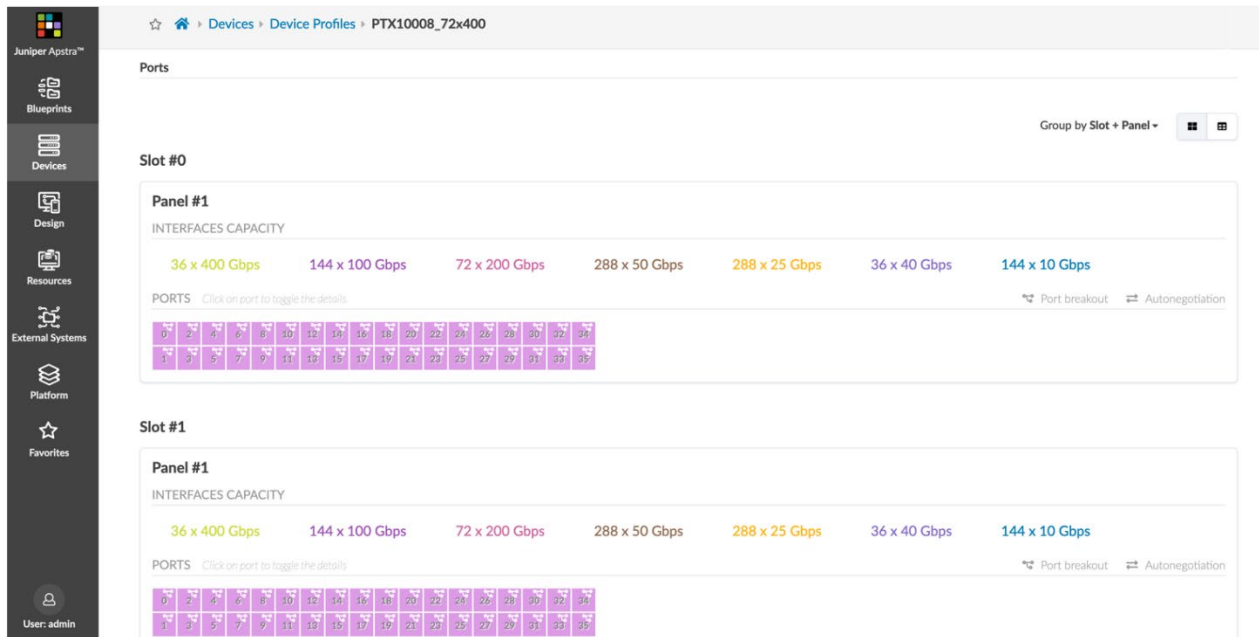
- **Logical Device** is a logical representation of the port density, speed, and possible breakout combinations of a switch. Since this is a logical representation, any hardware specifics are abstracted.
- **Device Profiles** provide hardware specifications of a switch that describe the hardware (such as CPU, RAM, type of ASIC and so on) and port organization. Juniper Apstra has several pre-defined Device Profiles that exist for common Data Center switches from different vendors.
- **Interface Map** binds together a Logical Device and a Device Profile, generating a port schema that is applied to the specific hardware and network operating system, which is represented by the Device Profile. By default, Juniper Apstra provides several pre-defined Interface Maps with the ability to create user-defined Interface Maps as needed.
- **Rack Types** define logical racks in Juniper Apstra, the same way a physical rack in a data center is constructed. However, in Juniper Apstra, this is an abstracted view of it, with links to Logical Devices that are used as leaves, the kind and number of systems connected to each leaf, any redundancy requirements (such as MLAG or ESI LAG) and how many links, per spine, for each leaf.
- **Template** takes one or more Rack Types as inputs and defines the overall schema/design of the fabric, with a choice between a 3-stage Clos fabric, a 5-stage Clos fabric or a collapsed spine design, and whether to build an IP fabric (with static VXLAN endpoints, if needed) or a BGP EVPN based fabric (with BGP EVPN as the control-plane).
- **Blueprint** is the instantiation of the fabric, taking a Template as its only input. A Blueprint requires additional user input to bring the fabric to life – this includes resources such as IP pools, ASN pools, Interface Maps. Additional virtual configuration is done, such as defining new virtual networks (VLANs/VNIs), building new VRFs, defining connectivity to systems such as hosts or WAN devices, and so on.

Back-End Network

The Back-end network is built using PTX10008 spines, with two PTX10K_LC1201_36CD line cards, providing 72 x 400G ports in total. For the purposes of this implementation, a single stripe of 32 x A100 servers are used, connected to 8 x QFX5220-32CD leafs.

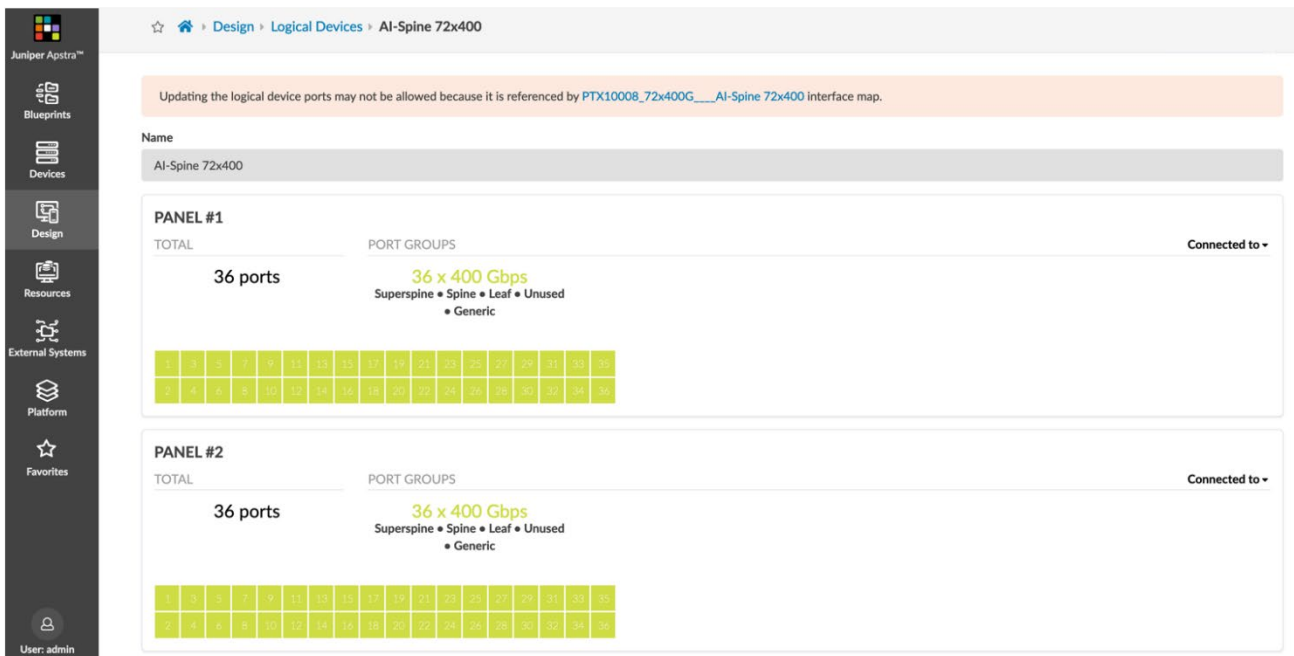
PTX spines are created as modular Device Profiles in Juniper Apstra, as shown below. This includes two 36 x 400G line cards, giving the spine a total of 72 x 400G interfaces for leaf connectivity.

Figure 13: Juniper Apstra Device Profile



The Logical Device is created as shown in Figure 14.

Figure 14: Juniper Apstra Logical Device



Finally, an Interface Map is created, which ties the Logical Device and the Device Profile together.

Figure 15: Juniper Apstra Interface Maps

The screenshot shows the Juniper Apstra interface for an Interface Map. The breadcrumb navigation is Design > Interface Maps > PTX10008_72x400G_AI-Spine 72x400. The main content area includes:

- Name:** PTX10008_72x400G_AI-Spine 72x400
- Logical device:** AI-Spine 72x400
- Device profile:** PTX10008_72x400
- Interface map preview:**
 - SUMMARY:** 72 x 400 Gbps, Superspine • Spine • Leaf • Unused • Generic
 - INTERFACES:** A grid of 72 numbered ports (1-72) arranged in three rows of 24.
 - MAPPING:**
 - Logical Device:** A grid of 72 ports, each with a checkmark icon.
 - Device Profile:** A grid of 72 ports, each with a checkmark icon.

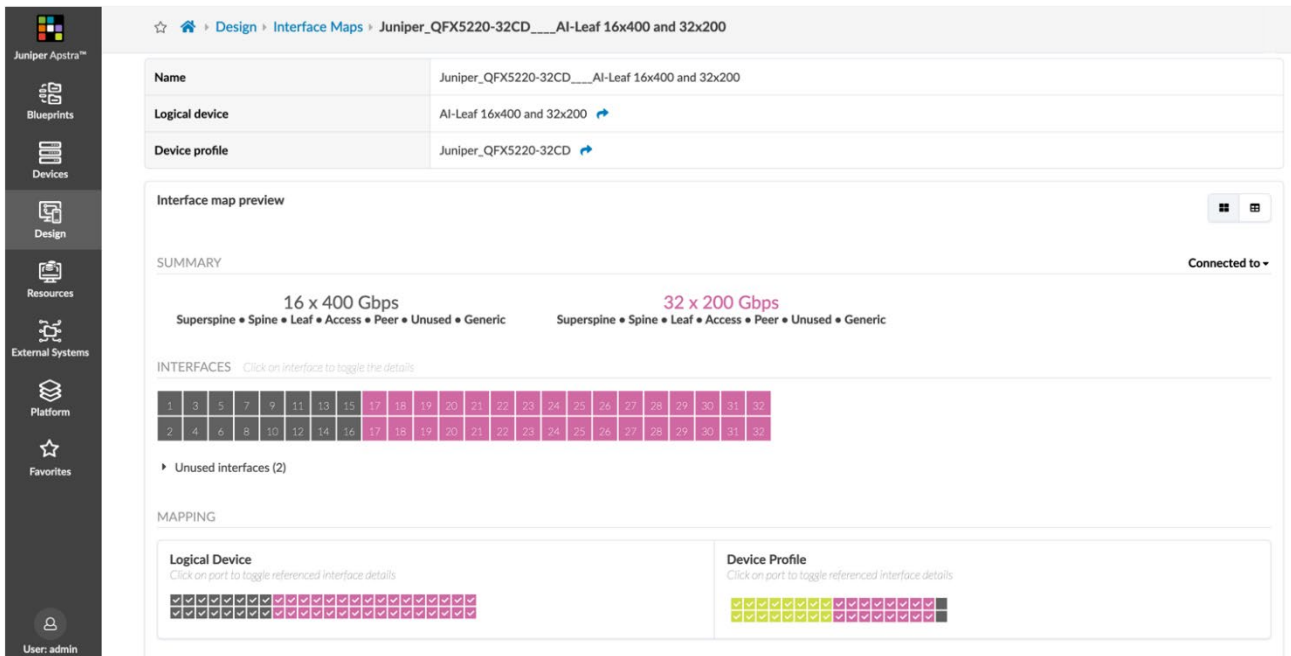
For the leafs, QFX5220-32CDs are used, with 16 x 400G and with the remaining 16 ports split into 2 x 200G per port, giving 32 x 200G in total. A new Logical Device and Interface Map is created as shown in Figure 16 and Figure 17.

Figure 16: Logical Device for QFX5220-32CD Leaf with 16 x 400G and 32 x 200G Ports

The screenshot shows the Juniper Apstra interface for a Logical Device. The breadcrumb navigation is Design > Logical Devices > AI-Leaf 16x400 and 32x200. The main content area includes:

- Name:** AI-Leaf 16x400 and 32x200
- PANEL #1:**
 - TOTAL:** 48 ports
 - PORT GROUPS:**
 - 16 x 400 Gbps:** Superspine • Spine • Leaf • Access • Peer • Unused • Generic
 - 32 x 200 Gbps:** Superspine • Spine • Leaf • Access • Peer • Unused • Generic
- Grid:** A grid of 48 numbered ports (1-48) arranged in two rows of 24.

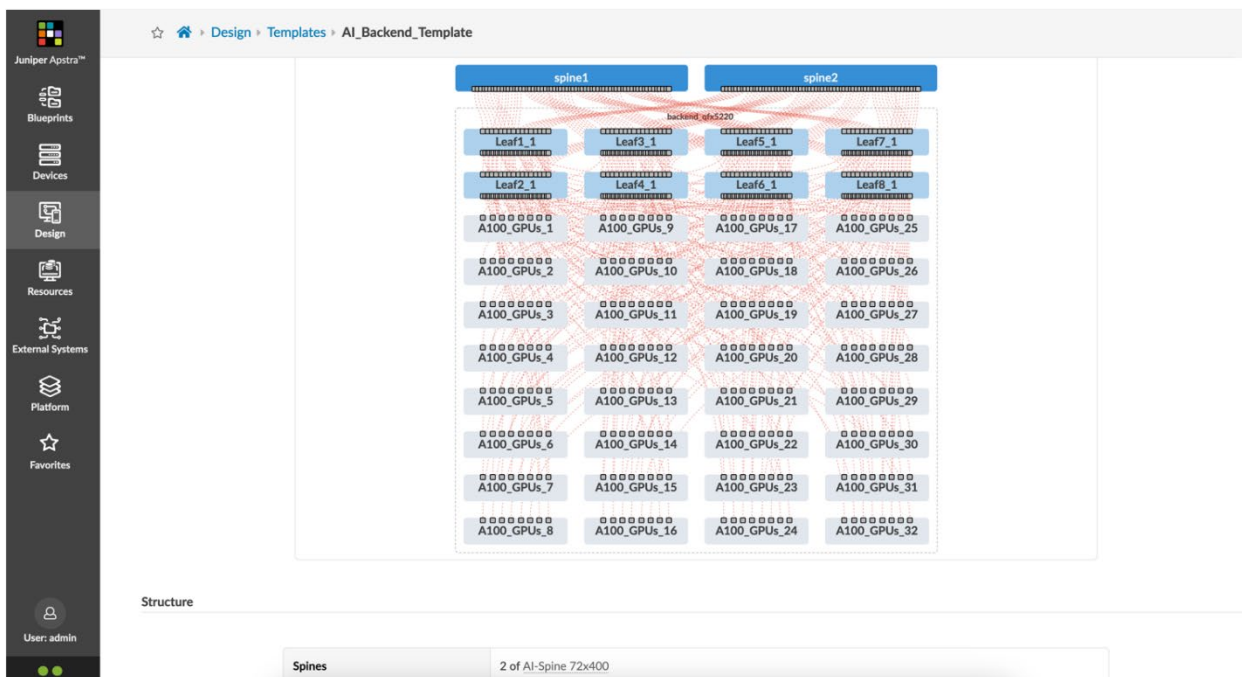
Figure 17: Interface Map for QFX5220-32CD with 16 x 400G and 32 x 200G Ports



To build the back-end network, first a Rack Type is created with the above Logical Devices as leaves. This includes 8 leafs, and 32 generic systems (32 Hyperplane8-A100s) connected to these leafs following the rail-optimized stripe design. Each leaf has 8 x 400G links to each spine. This Rack Type is used as an input into a Template for the back-end network. The Template is configured as a 3-stage Clos design, with static VXLAN (which essentially builds an IP fabric if no static VXLAN endpoints are defined by the user).

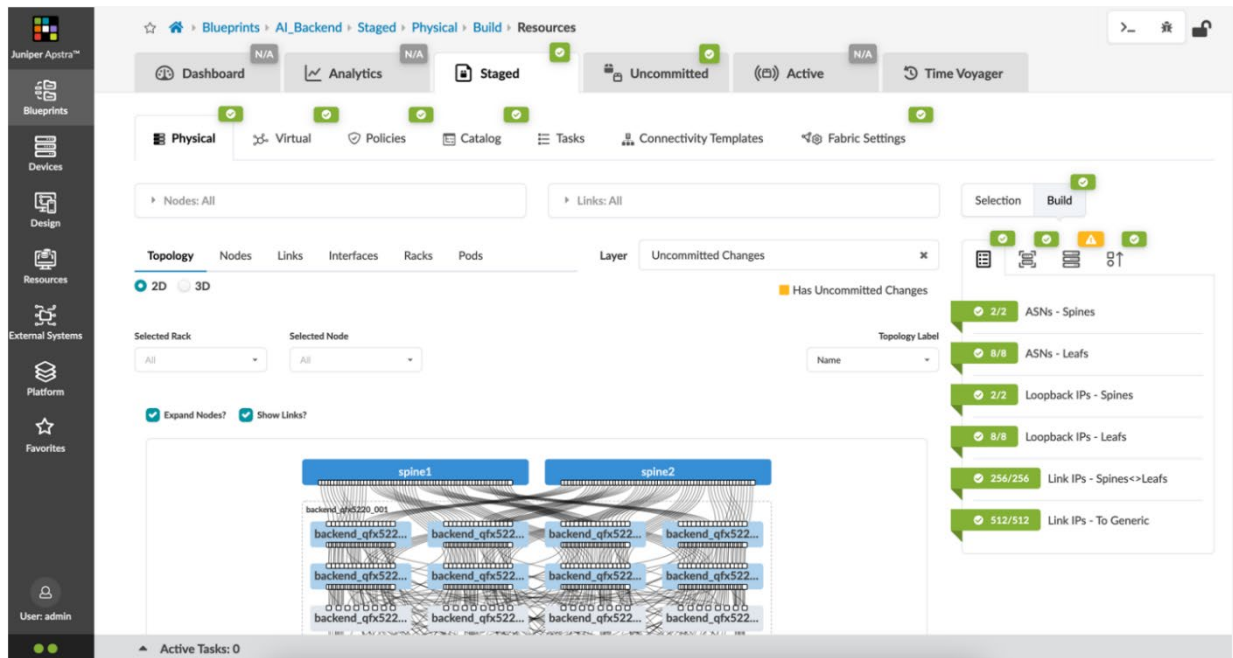
Figure 18 shows the template with the spines defined with the Logical Device that was created for the PTX10008 Device Profile.

Figure 18: Juniper Apstra Template for Backend Network



Finally, a Blueprint is instantiated with this Template.

Figure 19: Juniper Apstra Blueprint for Backend Network



There are several resources added into the Blueprint to start building the fabric – this includes the ASN pool for the leafs and spines, the spine and leaf loopbacks, and the point-to-point addresses between the leafs and the spines.

A common ASN pool and IP pool is defined for the ASNs and the loopbacks, as shown in [Figure 20](#) and [Figure 21](#).

Figure 20: Juniper Apstra ASN pool

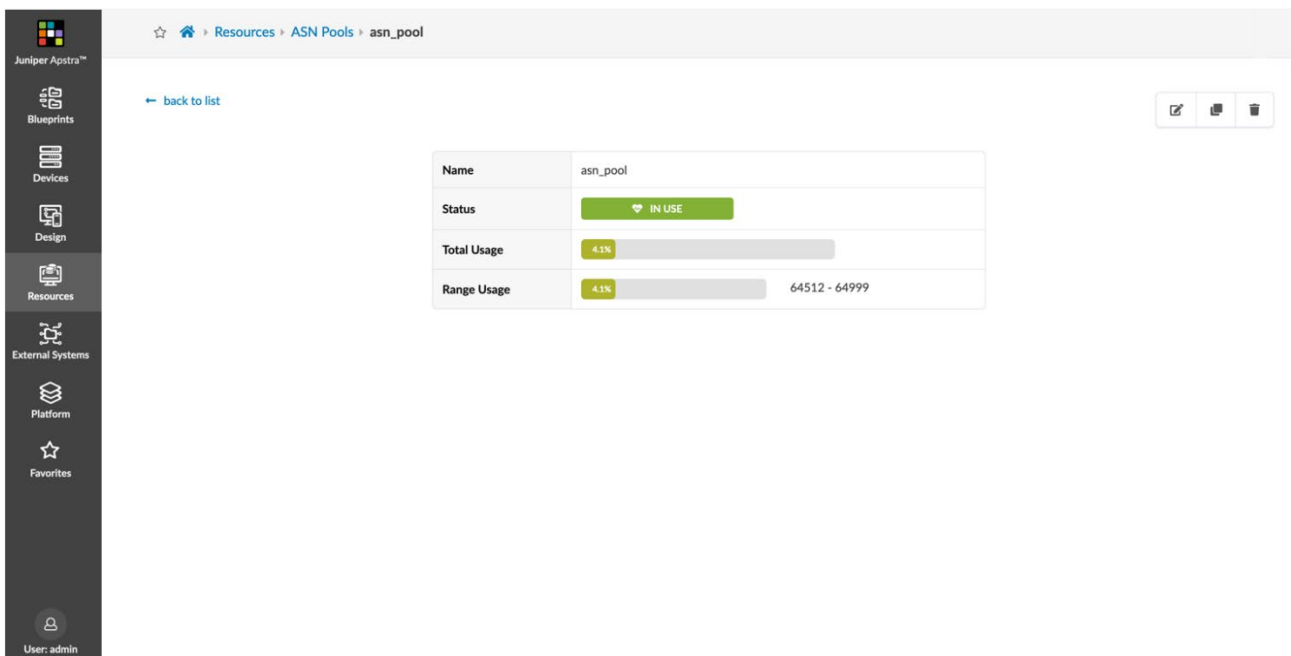
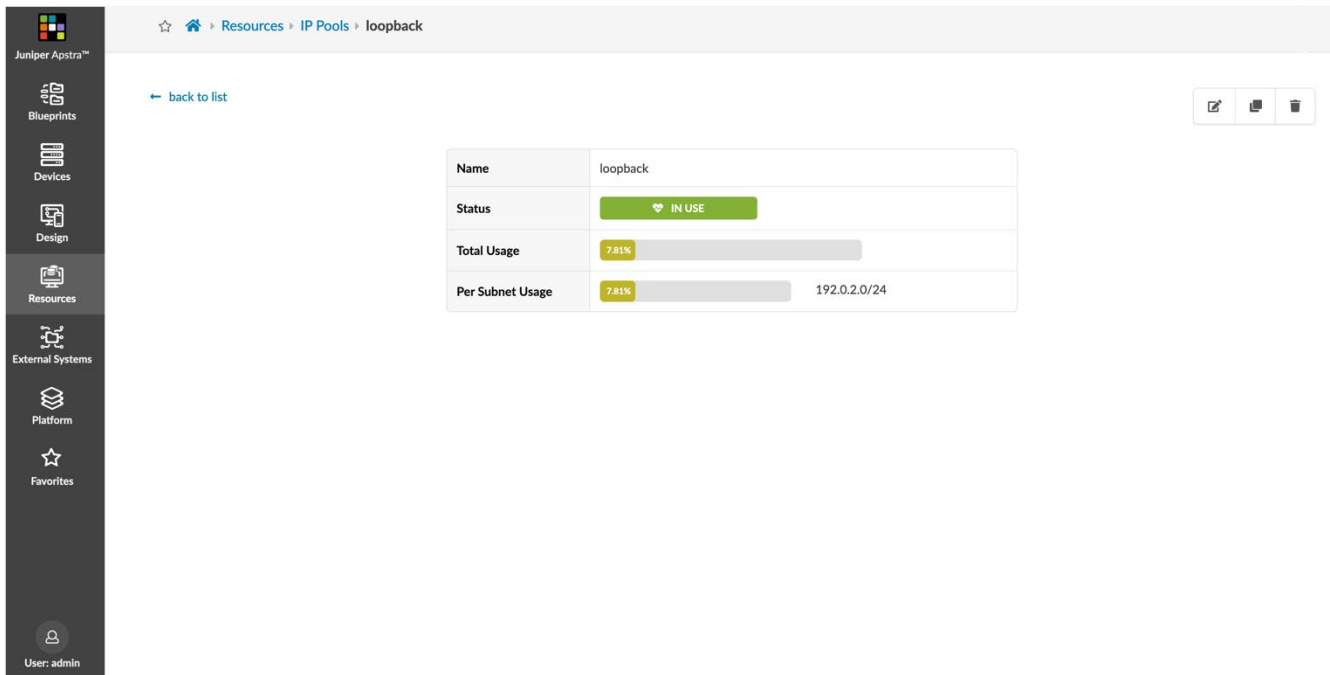
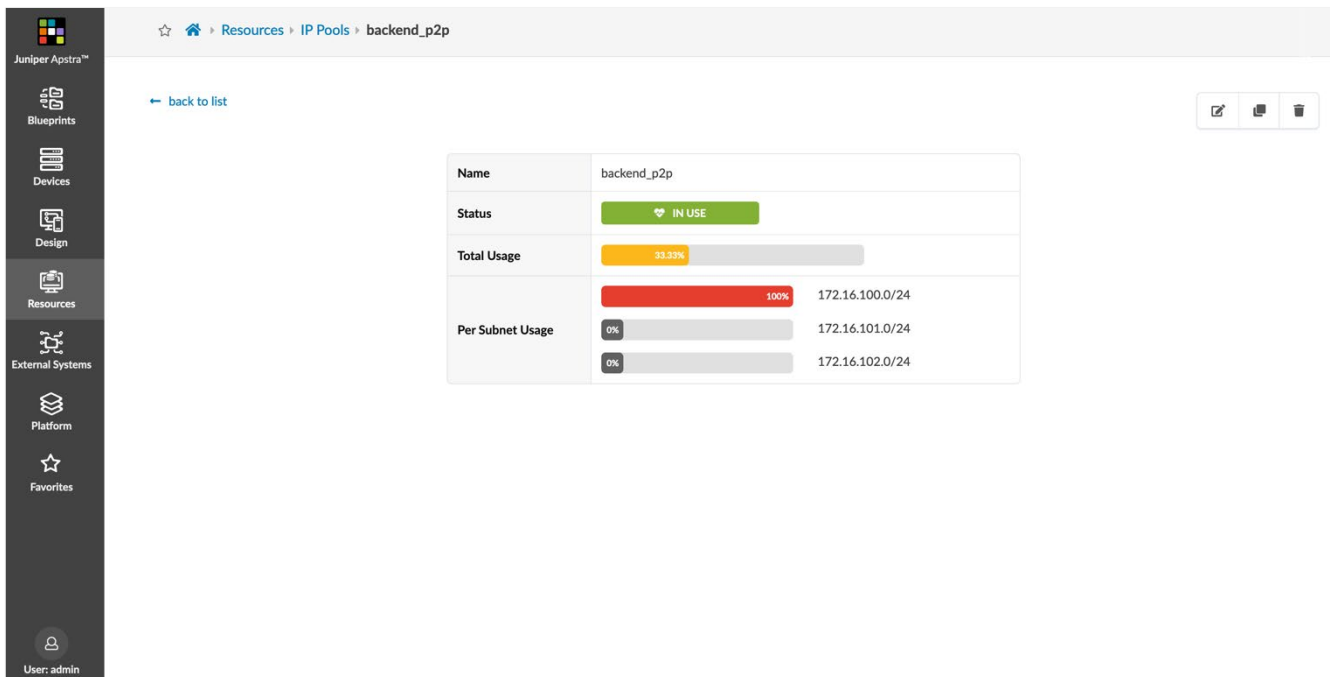


Figure 21: Juniper Apstra IP Pool for Loopbacks



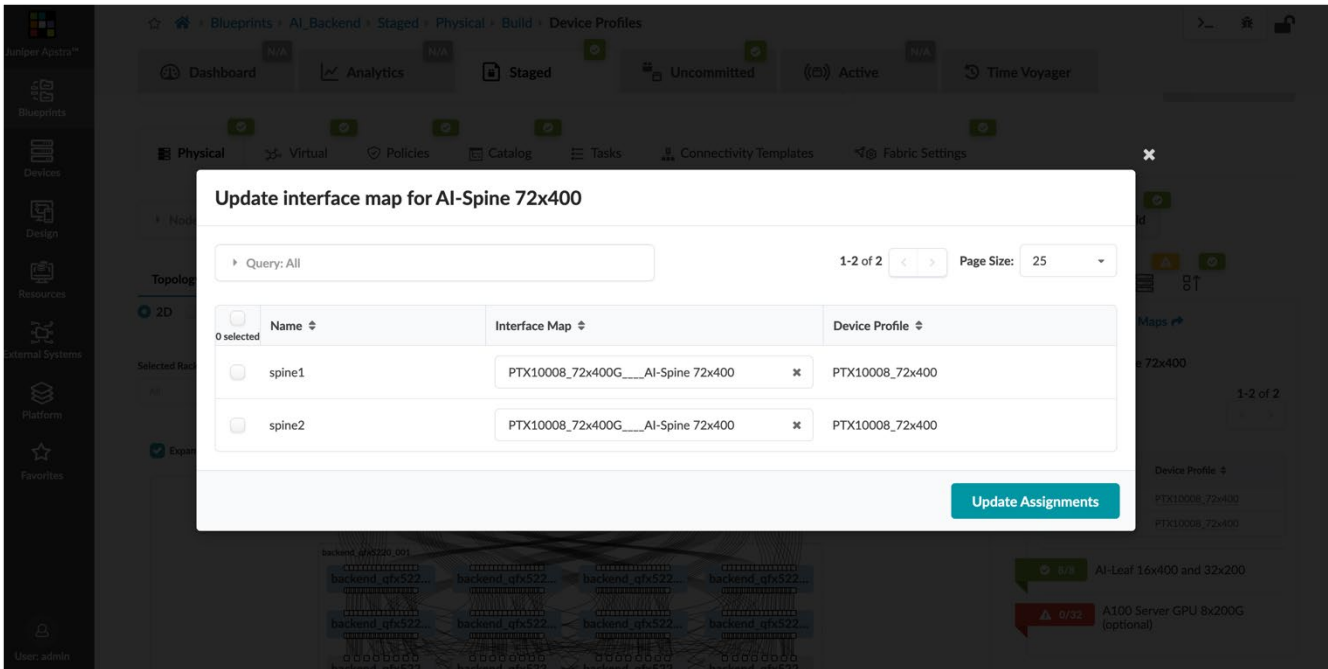
Since there are eight leafs, and each leaf has eight links to each spine, there is a requirement of 256 addresses for the point-to-point links between the leafs and the spines. For this, a separate group of pools are defined for the back-end network, as shown in [Figure 22](#).

Figure 22: Juniper Apstra IP pool for point-to-point links between leafs and spines



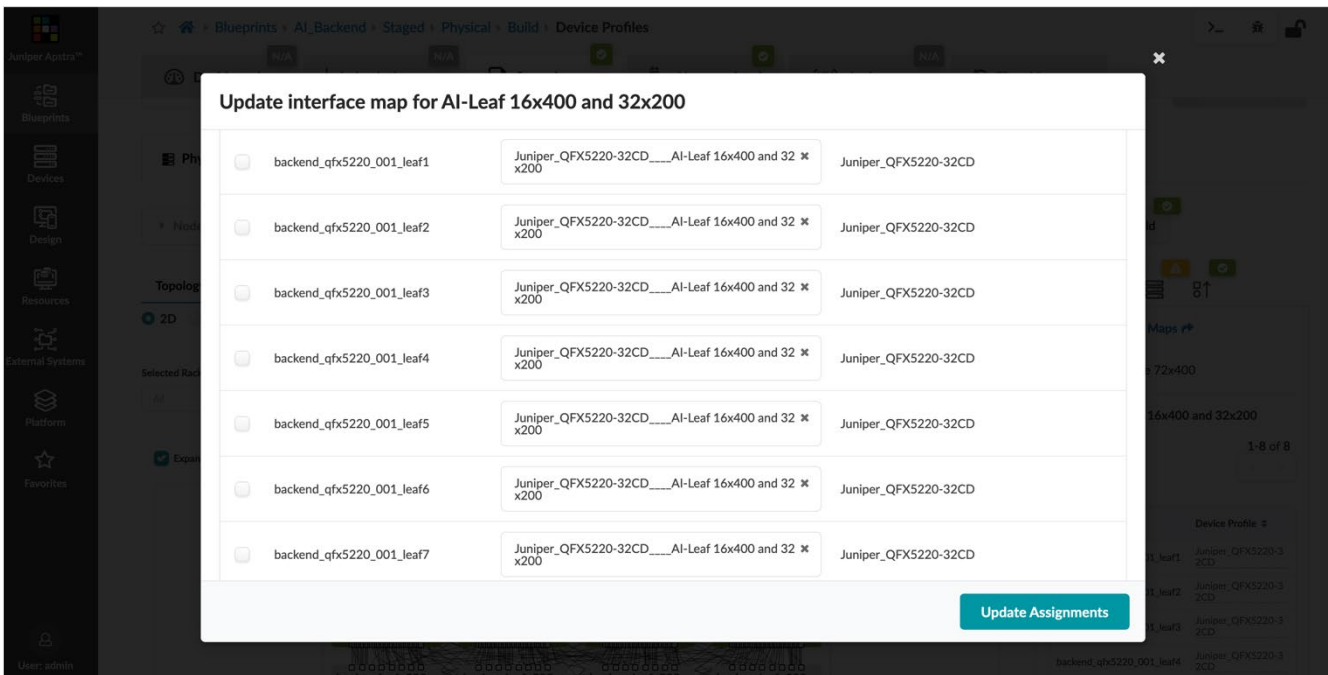
To start generating device specific configuration in Juniper Apstra, the Interface Maps must be mapped to each device in the fabric. For the spines, the Interface Map corresponding to the 72 x 400G PTX10008 device is used.

Figure 23: Interface Map to Device Mapping for spines



For the leafs, the 16 x 400G + 32 x 200G Interface Map corresponding to the QFX5220-32CD is used as shown in [Figure 24](#).

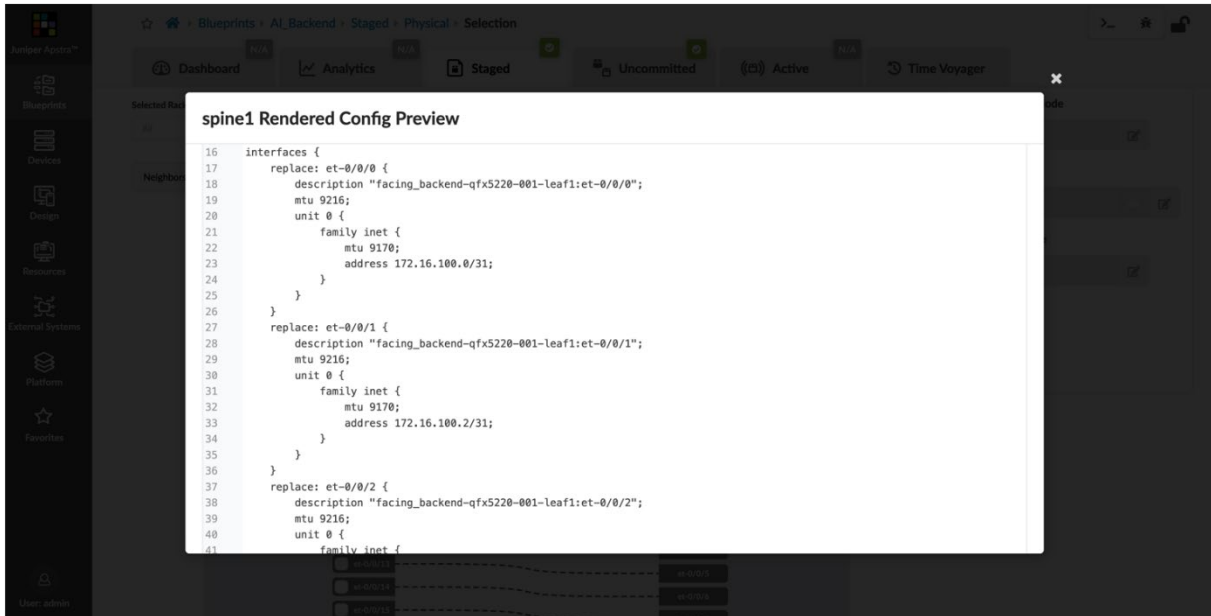
Figure 24: Interface Map to Device Mapping for Leafs



With these mapped to the respective devices, Juniper Apstra generates configuration that is used to build the IP fabric. To view rendered information specific to a device in the fabric, click **Rendered** under Config available on the right-side when in a device specific view of the topology.

For example, to confirm, we can see that the rendered configuration includes point-to-point /31 addresses on the interfaces to connect to each of the leaves, as shown in [Figure 25](#).

Figure 25: Juniper Apstra Rendered Configuration for Spine1



For the back-end network, the connectivity down to the GPU is also Layer 3. To do this, a Connectivity Template in Juniper Apstra is used, and this is attached to all the links on the leaves that connect to the Generic Systems (which logically represent the Hyperplane8-A100 servers). Thus, the application point is the interface itself.

This Connectivity Template is an IP Link primitive, using the default routing zone and a numbered IPv4 addressing type.

Figure 26: Juniper Apstra Connectivity Template for IP Link to GPUs

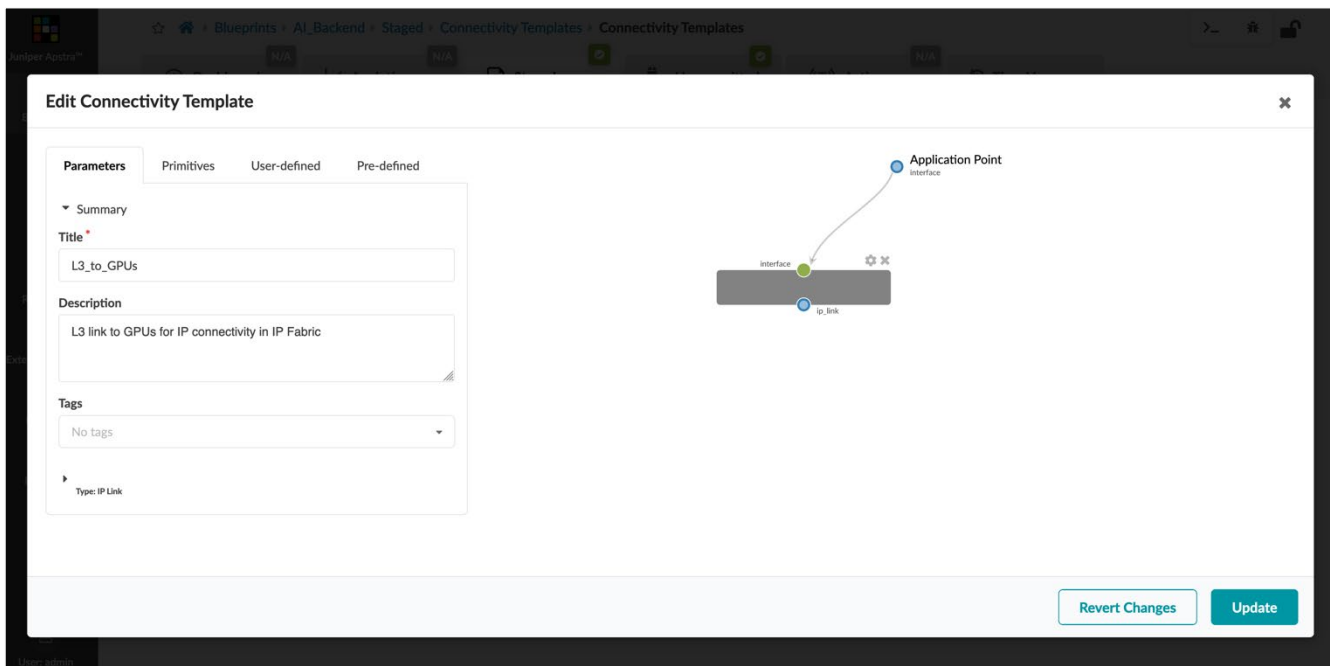
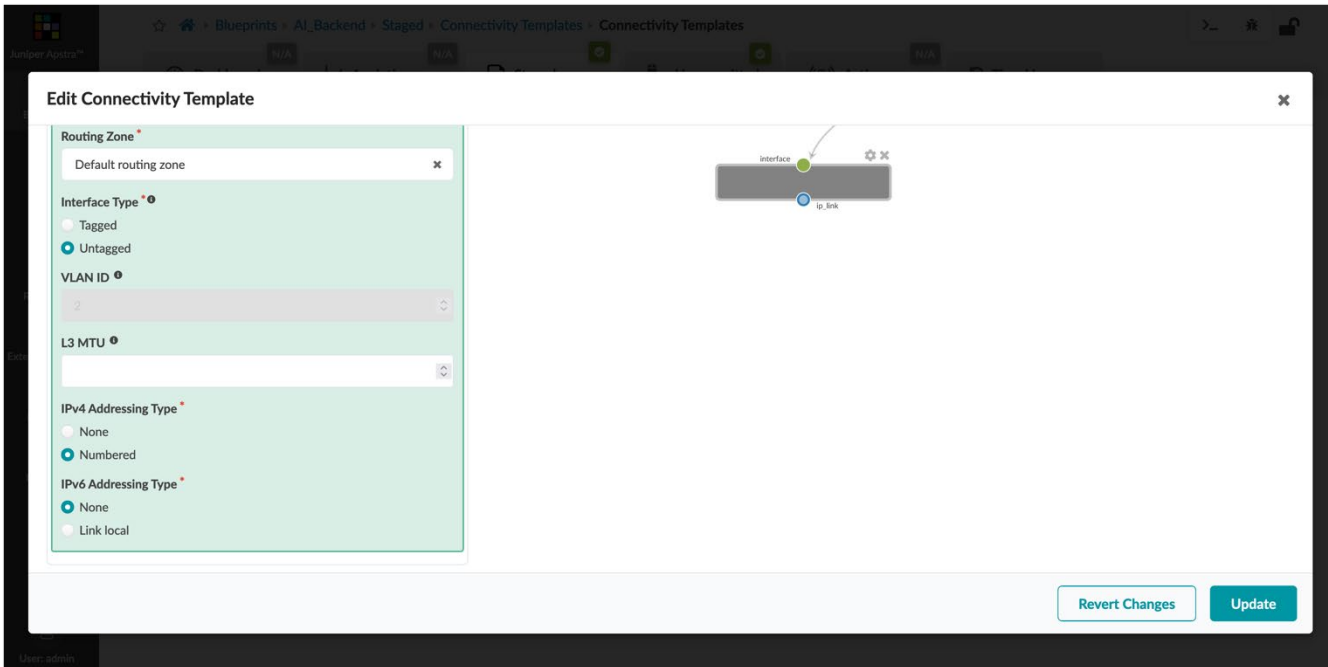
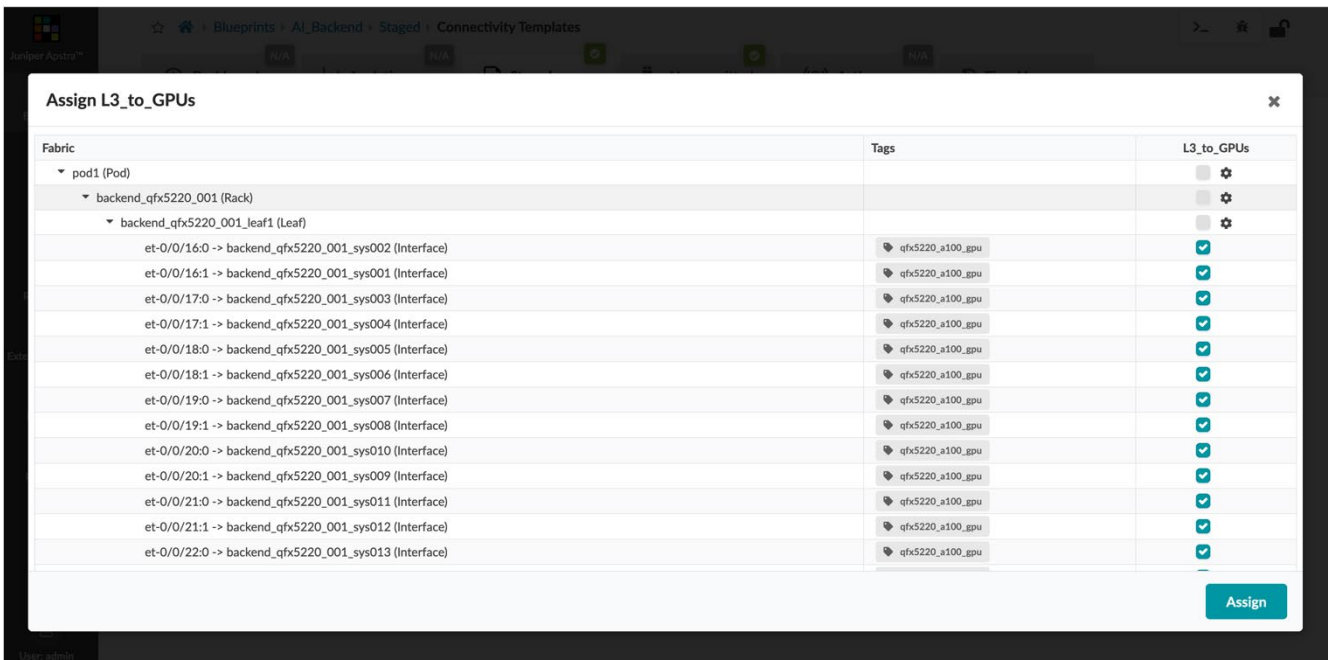


Figure 27: Configuration of Juniper Apstra Connectivity Template for IP Link to GPUs



This Connectivity Template is then attached to all the GPU facing interfaces on all leaves. A snippet is shown in [Figure 28](#).

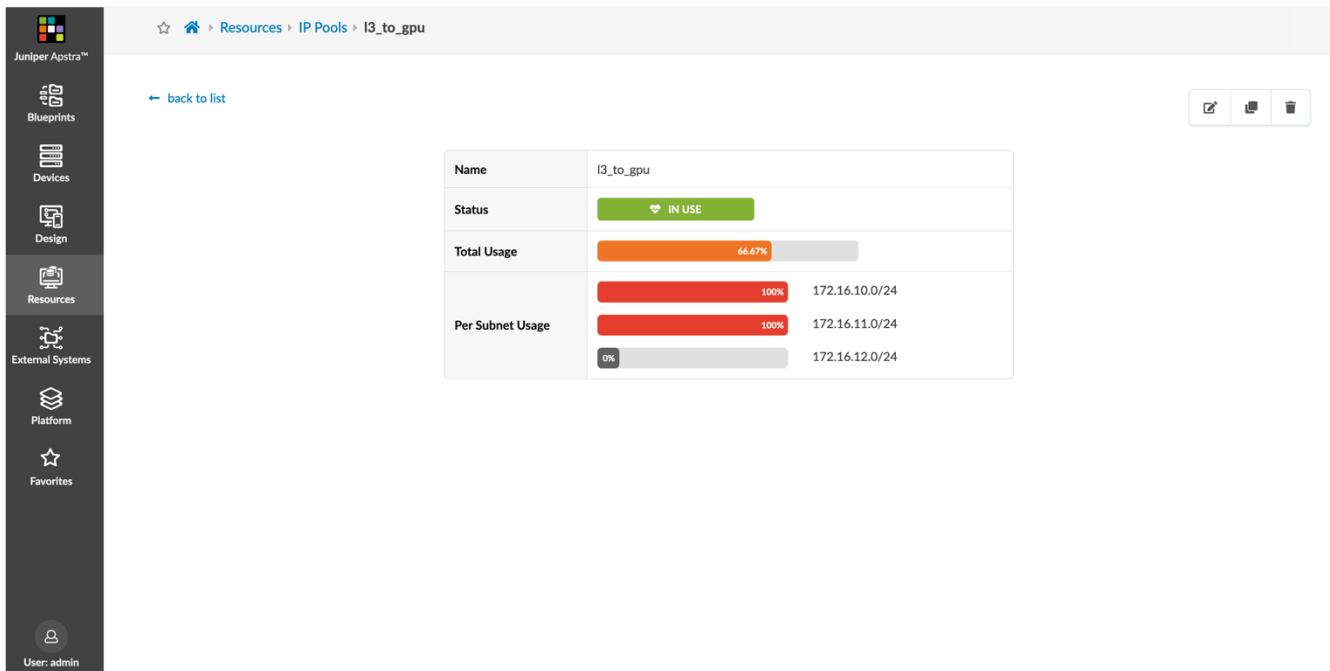
Figure 28: Attaching Connectivity Template to interfaces



Once this is attached to the interfaces, Juniper Apstra has an additional requirement of IP pools for these new point-to-point links. Since a single stripe has 32 x Hyperplane8-A100s and implies that there are 32 x 8 links to GPUs per leaf, there is a requirement of 512 addresses.

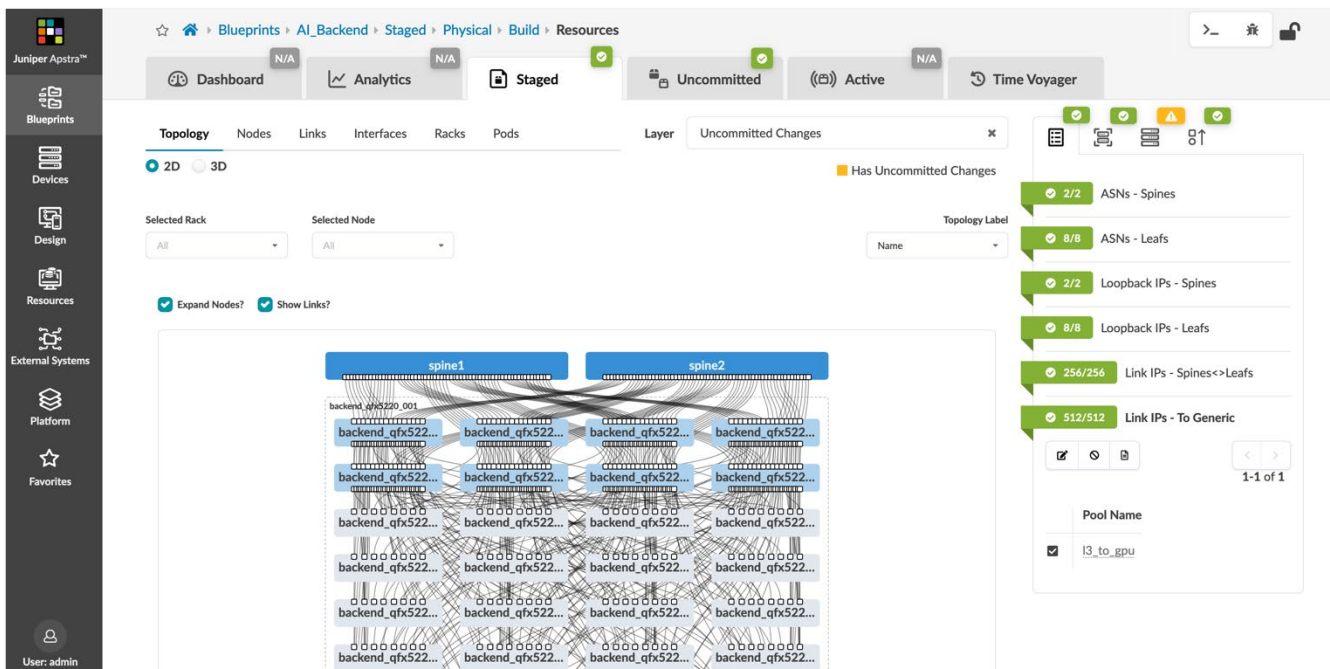
A dedicated group of IP pools are created and used as shown in [Figure 29](#).

Figure 29: Juniper Apstra IP Pool Created for IP Link Connectivity to GPUs



This pool is attached to the resource requirement in Juniper Apstra.

Figure 30: Juniper Apstra L3 Fabric IP Pool Assignment

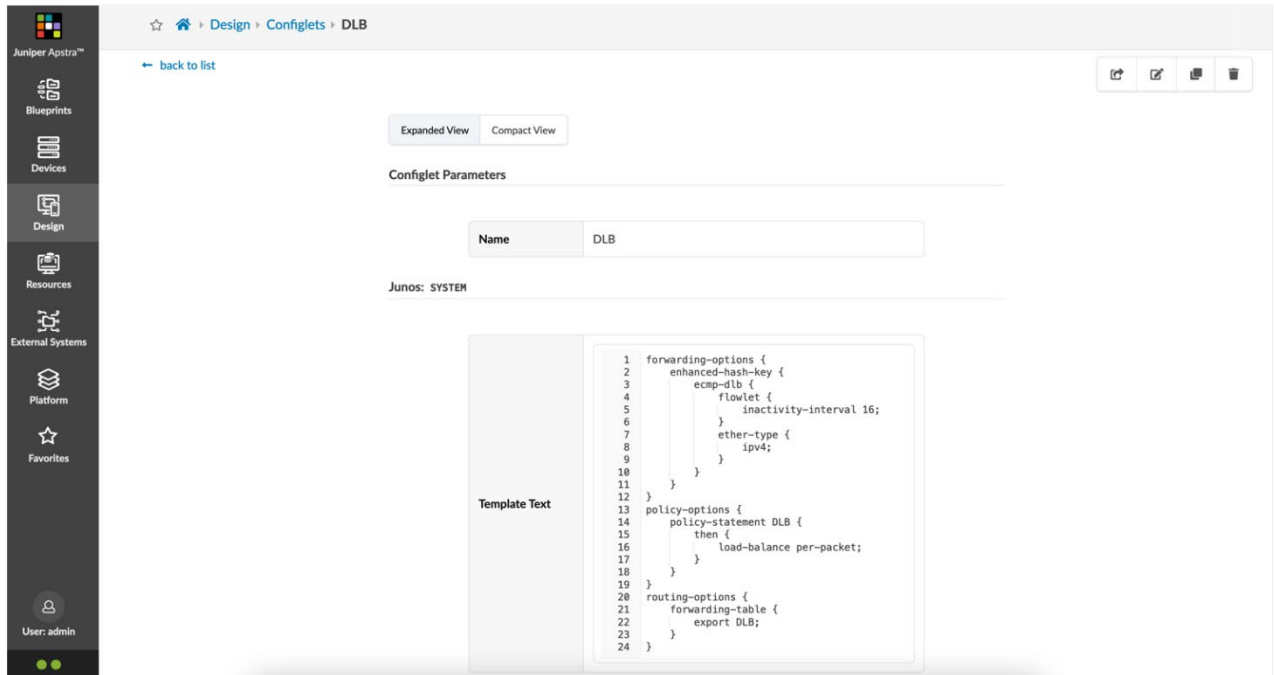


Juniper Apstra now generates configuration for these point-to-point GPU facing interfaces per leaf and redistributes these addresses into the IP fabric through BGP for reachability between all GPU servers.

Once the main fabric is configured, the two IP services (DLB and DCQCN) must be added using Configlets in Juniper Apstra.

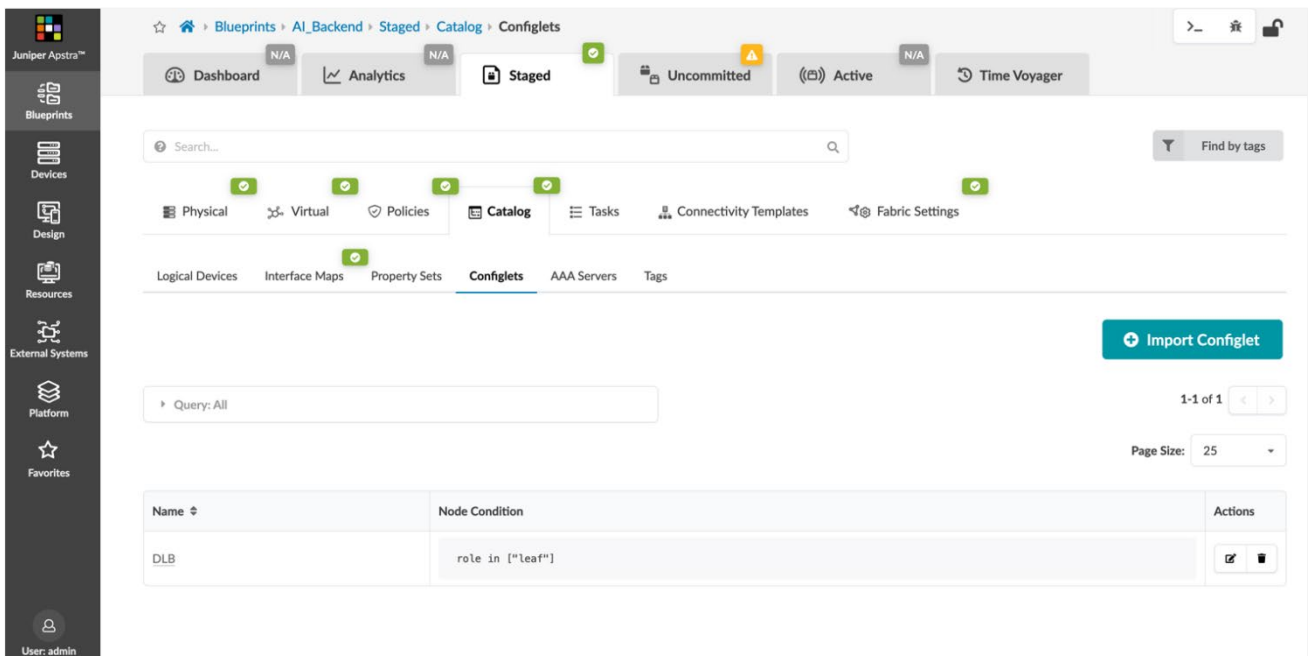
First, a Configlet is created for DLB as shown in **Figure 31**.

Figure 31: Configlet for DLB



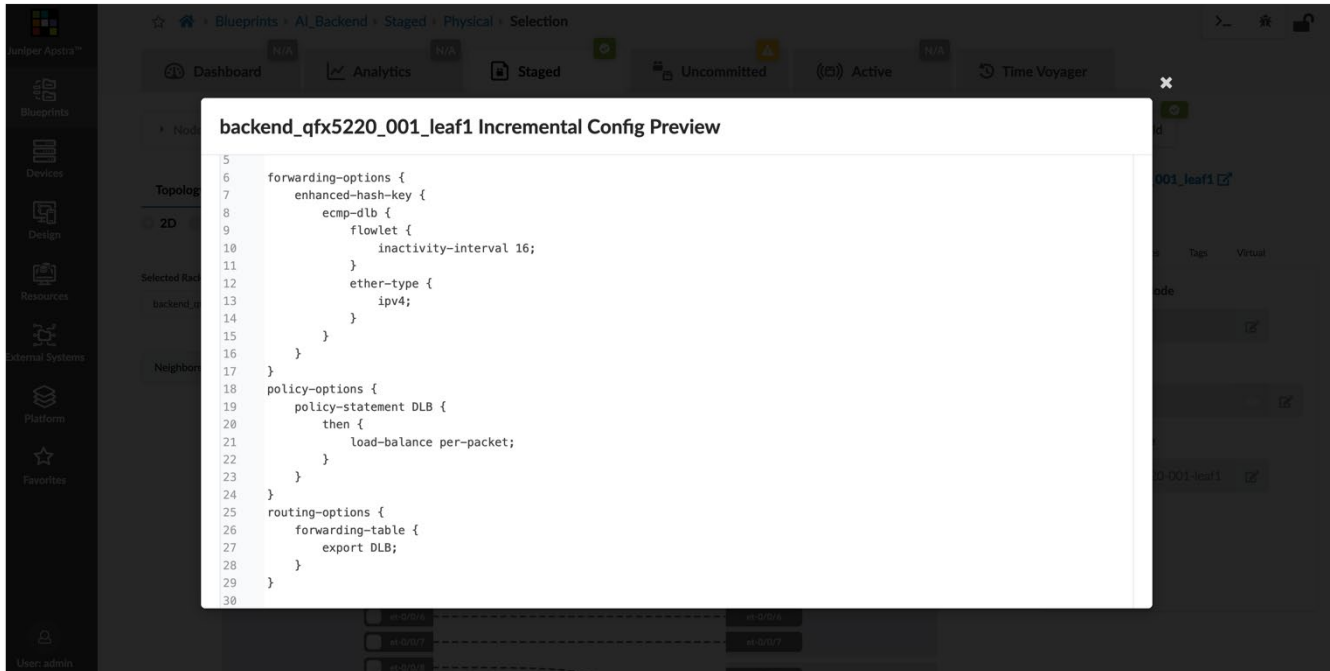
To use a Configlet in a Blueprint, it must be imported into the Blueprint Catalog. Specific conditions are used for its assignment to determine which devices receive this additional configuration as defined in the Configlet. For the purposes of DLB, we need all leafs to have this configuration, so the condition matches on the 'leaf' role.

Figure 32: Condition Match for DLB Configlet



Once this is added to the Blueprint, we can confirm that the leafs now have an incremental configuration added through this Configlet. An example for leaf1 is shown in [Figure 33](#).

Figure 33: Incremental Configuration Added to a Leaf When DLB Configlet is Successfully Imported



In the same way, relevant configuration for DCQCN is also added as a Configlet. Since the DCQCN configuration can get complicated, automating this via a tool like Terraform can provide reliability and the flexibility needed to accurately apply this on all necessary interfaces.

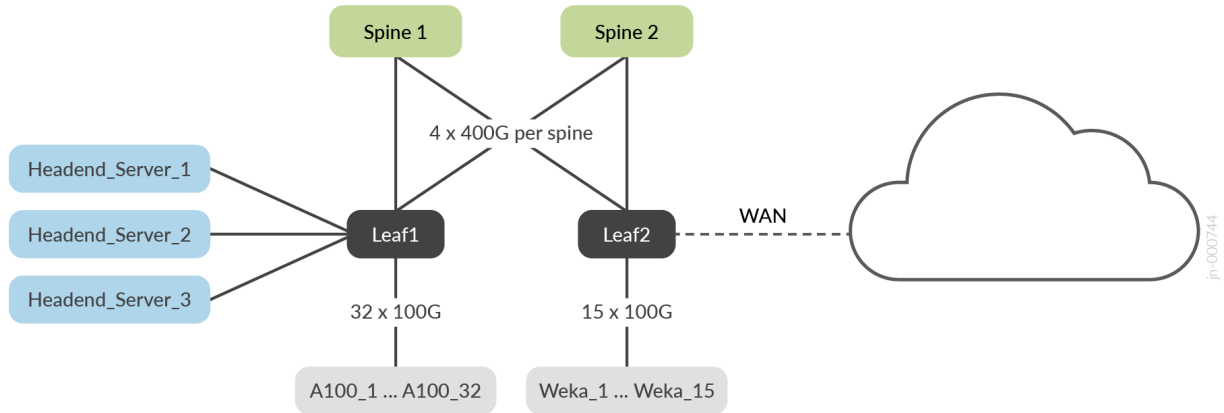
Front-end Network

The Front-end network uses QFX5220-32CD for both the spines and the leafs. The network requirement for this is not as strict as the back-end network in terms of the oversubscription ratio. To keep things consistent, the oversubscription ratio is still maintained as close to 1 as possible.

The standard networking port of the Hyperplane8-A100s are used to connect to the front-end network. Each server has only one of these ports, which means that 32 x 100G front-end interfaces need to be connected to this network. All the GPU server ports are connected to leaf1, while the storage front-end interfaces are connected to leaf2.

[Figure 34](#) shows a topology with a high-level overview of the front-end network for Juniper Apstra implementation.

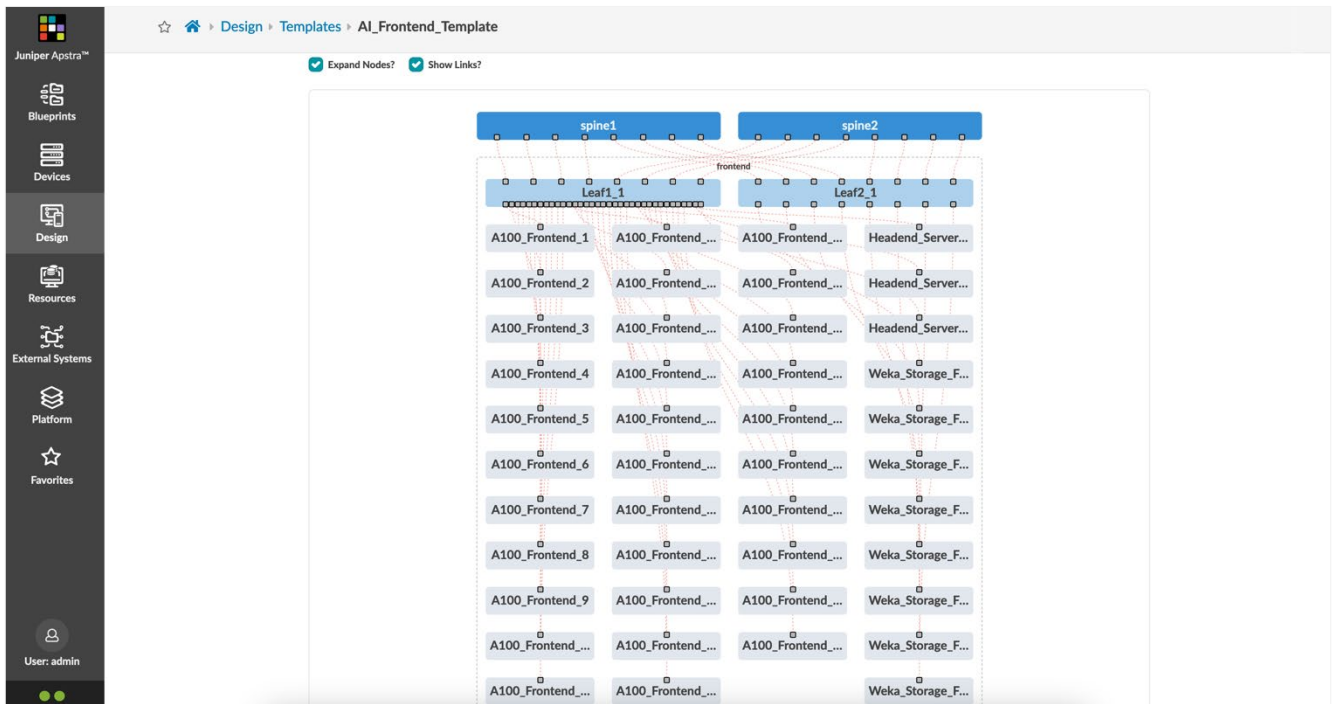
Figure 34: Topology for Implementation of Front-End Network in Juniper Apstra



Lambda is a trademark of Lambda Research Corporation. Nvidia is the trademark of Nvidia Corporation. Weka is a trademark of WekaIO Ltd.

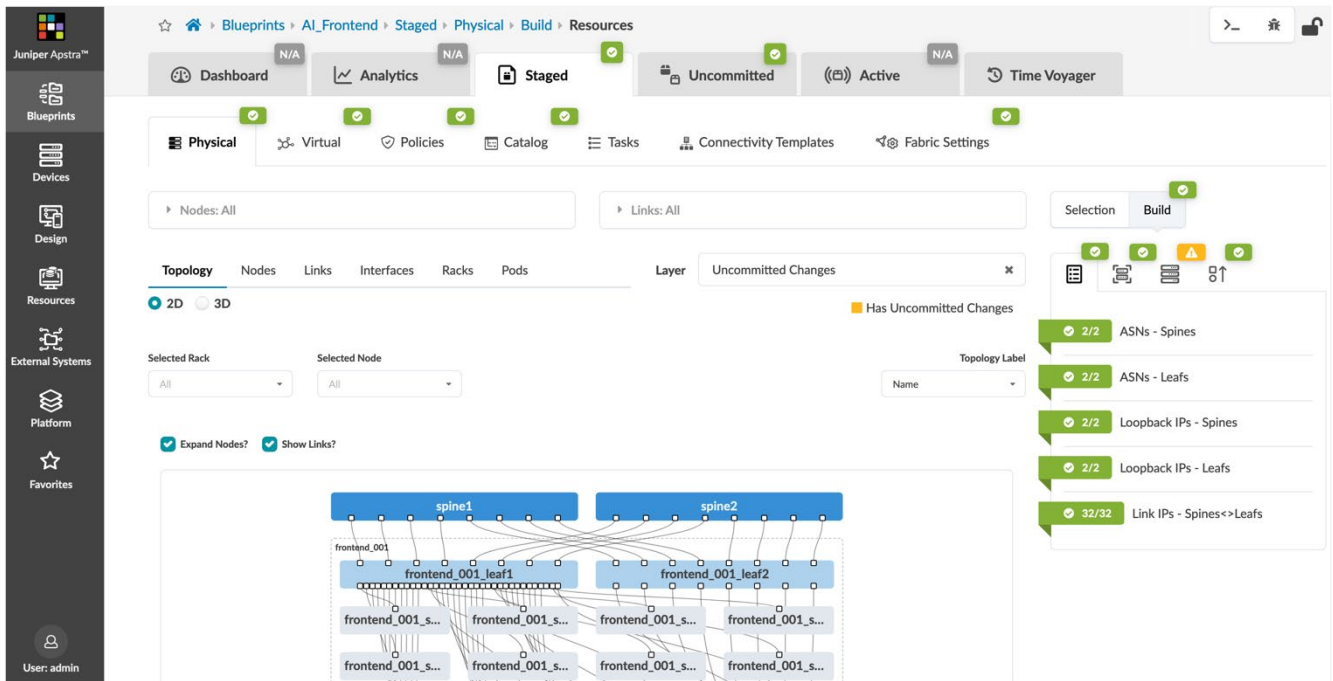
In Juniper Apstra, this Rack Type is added in a Template, as shown below. The Template is similar to the Template used for the back-end network. The front-end specific Rack Types are used instead.

Figure 35: Juniper Apstra Template for the Front-End Network



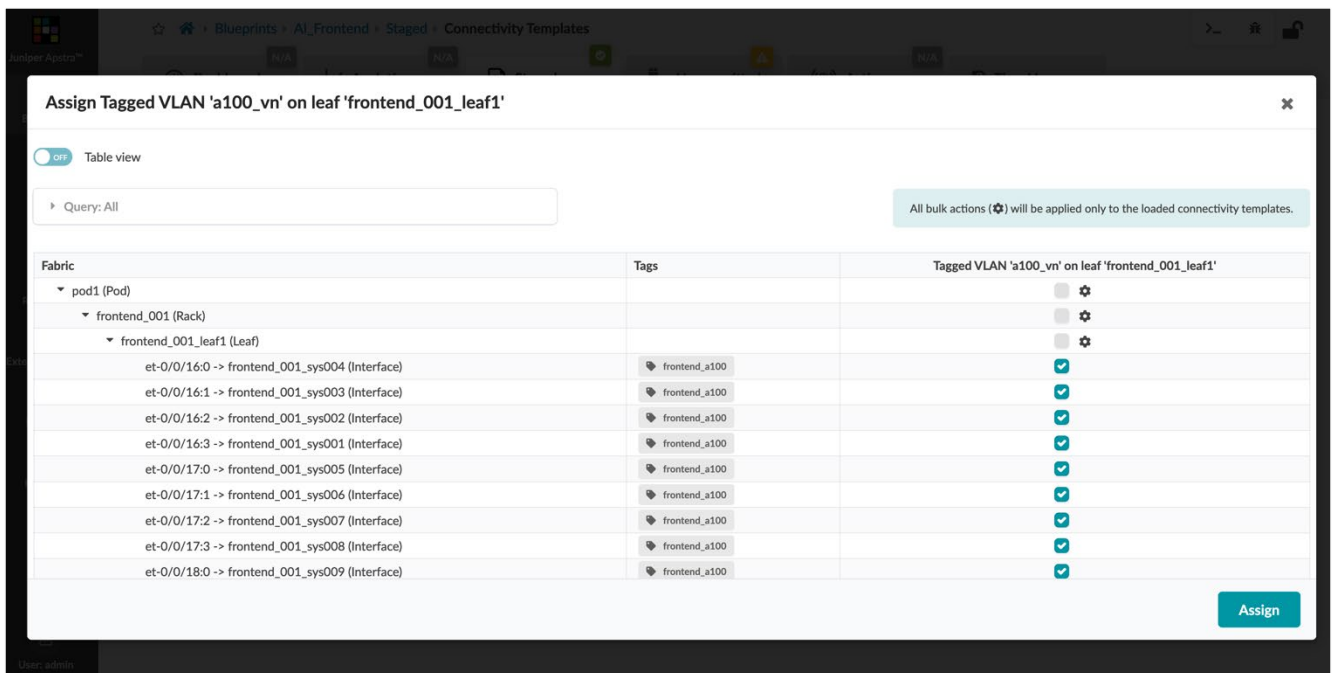
You can use this Template to instantiate a Blueprint specific to the Front-end network.

Figure 36: Juniper Apstra Blueprint for Front-End Network



Like any other fabric, the Blueprint includes resources such as ASN and IP pools, and Interface Maps for each of the devices in the fabric. For the front-end network, there is no requirement for Layer 3 point-to-point links to the servers, and general Layer 2 connectivity is used with untagged or tagged interfaces. This Connectivity Template is then assigned to all A100 server facing interfaces on leaf1.

Figure 37: Connectivity Template Attached to Interfaces of Leaf for Front-End Network



In the same way, another Layer 2 Connectivity Template is used for the Headend Servers, and a Layer 3 Connectivity Template for the WAN facing interface, allowing a default route to be received from the WAN edge.

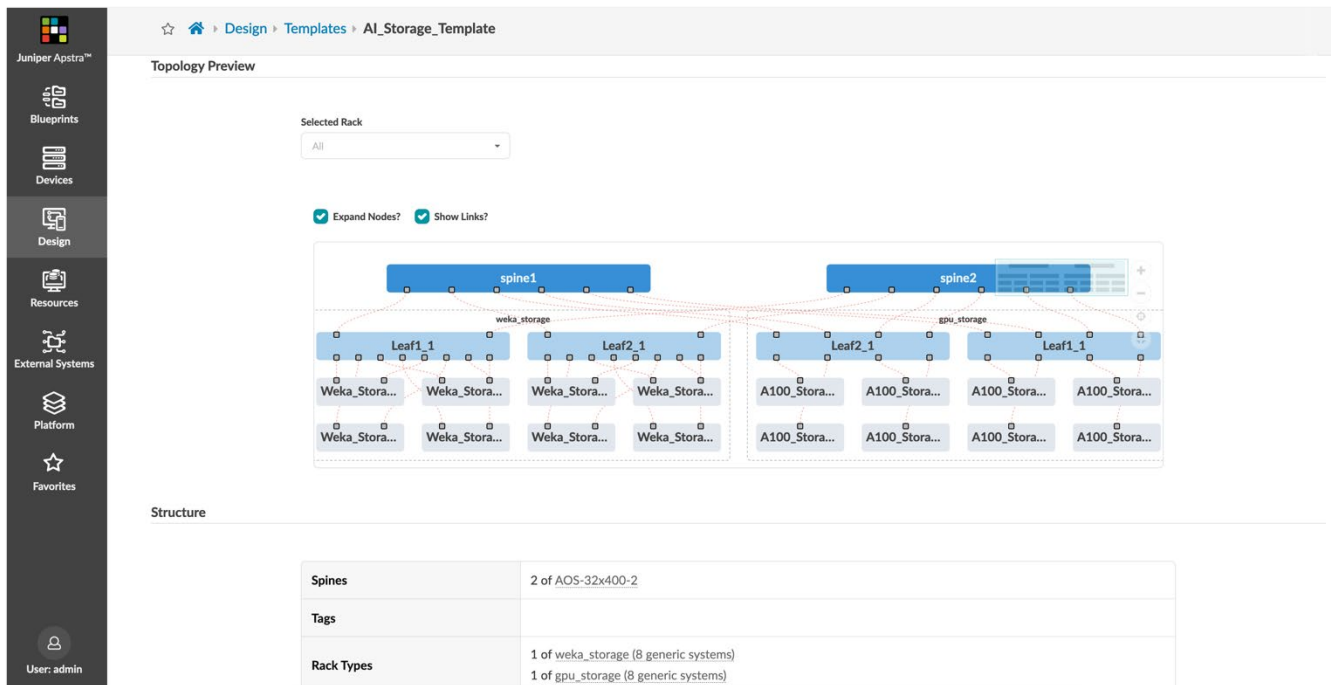
Storage Network

The storage network facilitates the connection of storage ports of the Hyperplane8-A100 servers, along with a dedicated storage cluster which includes Weka storage nodes. Two Rack Types are built for this, with each rack having 2 x QFX5220-32CD leafs. One Rack Type connects to the GPU storage ports and the other Rack Type connects to the dedicated storage nodes.

Each GPU server connects to a leaf through a dedicated 200G storage port. Since we have 32 servers, the cluster is divided into two, with 16 servers connecting to leaf1 and 16 servers connecting to leaf2. Thus, for the GPU storage rack, each leaf receives 16 x 200G inbound, implying that it must have 8 x 400G per leaf.

These Rack Types are added in the storage-specific Template in Juniper Apstra.

Figure 38: Juniper Apstra Template for Storage Network



You can use this Template to instantiate a Blueprint for the storage network, and various resources are tied to this Blueprint again. For the storage network, we recommended that Layer 3 links are used for the servers and the storage nodes. For this, a Layer 3 Connectivity Template is used, and it is attached to all the links to these nodes, similar to the back-end network.

At this stage, all three networks (back-end, front-end, and storage) are completely built with Juniper Apstra, with all necessary configuration ready to be pushed to the network devices of the fabric.

Summary

This document illustrates the various kinds of AI/ML cluster designs using Juniper Network devices and the corresponding implementation with Juniper Apstra. The design considerations maintained in the document are from the network perspective, and the performance of the GPU cluster and models are subject to change based on many external factors relative to the GPU hardware, NICs, congestion control mechanisms, algorithms in use and the workloads themselves as examples. Contact Juniper Networks representative for consultation on specific implementation.



Corporate and Sales Headquarters

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or +1.408.745.2000
Fax: +1.408.745.2100
www.juniper.net

APAC and EMEA Headquarters

Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
Phone: +31.207.125.700
Fax: +31.207.125.701

Copyright 2023 Juniper Networks, Inc. All rights reserved. Juniper Networks, the Juniper Networks logo, Juniper, Junos, and other trademarks are registered trademarks of Juniper Networks, Inc. and/or its affiliates in the United States and other countries. Other names may be trademarks of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Send feedback to: design-center-comments@juniper.net V1.0/231110/ai-clusters-data-center-design