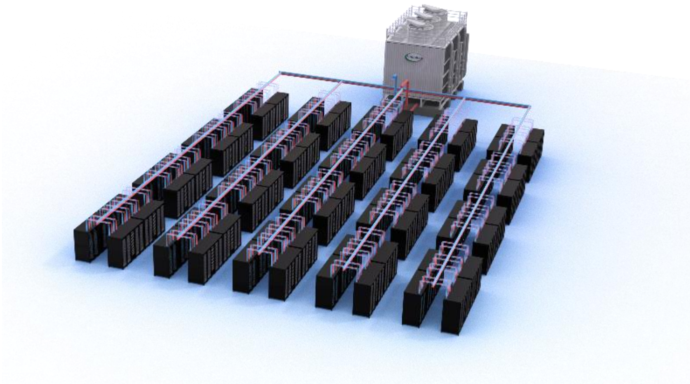
VALIDATED DESIGN

BUILDING A TENANT-AWARE AMD INSTINCT™ MI325X GPU CLUSTER WITH AN INTEGRATED SUPERMICRO SOLUTION

A Fully Validated Solution for Delivering Optimized Architecture for Providers and Customers Building an Infrastructure with AMD Instinct™ MI325X GPUs, AMD Pensando™ Pollara 400 AI NICs, and AMD EPYC™ CPUs

TABLE OF CONTENTS



Executive Summary

Supermicro and AMD have partnered as trusted technology leaders at the forefront of the data center industry. Together, we provide optimal Total Cost of Ownership (TCO) solutions across various domains, including Data Centers, AI, and HPC applications. Supermicro and AMD deliver leading systems for Generative AI through this strategic collaboration, working with partners across the technology stack. Customers trust this partnership for tested designs and solutions that simplify the complexity of these deployments. The components in these AI clusters deliver very high performance, with additional considerations beyond speeds and feeds to allow for optimal resource utilization in each solution.

This joint paper from Supermicro and AMD shares a pre-validated GenAI solution that can be replicated, expanded, shrunk, and operated in an optimal fashion. The scale of this solution can easily be adjusted to various cluster sizes, from

16/32/64/128 nodes up to a maximum of 1024 nodes with this validated design. This comprehensive full-stack solution can be deployed at a customer site or acquired pre-built with already connected and configured elements, burned-in before shipping for rapid deployment. The design is based on the AMD Instinct™ MI325X GPUs, AMD Pensando™ Pollara 400 AI NICs, and AMD EPYC™ CPUs integrated with Supermicro optimized servers, with all network cabling and switching configured for optimal performance upon delivery. This document addresses numerous design and engineering considerations to help readers understand the key elements of this Supermicro and AMD collaboration.

Glossary of Terms

Accelerator	The actual xPU performing calculations
SU	Scale Unit
RCCL	ROCm Collective Communications Library
Rail Optimized	Locality relating to accelerator rank on a system
DLC	Direct to chip Liquid Cooling
CDU	Coolant Distribution Unit
SLURM	Simple Linux Utility for Resource Management
NIC	Network Interface Card
AOC	Add On Card (Supermicro designation for NIC models)
RoCE	RDMA over Converged Ethernet
RNIC	RDMA-capable NIC
BIOS	Basic Input/Output System
GRUB	GNU Grand Unified Bootloader
PFC	Per-priority Flow Control
ECN	Explicit Congestion Notification
CNP	Congestion Notification Packet
ARS	Adaptive Routing and Switching
DLB	Dynamic Load Balancing
UDP	User Datagram Protocol
BGP	Border Gateway Protocol
AS	Autonomous System
PCIe	Peripheral Component Interconnect Express
DSCP	Differentiated Services Code Point
TC	Traffic Class
PG	Port Group
VRF	Virtual Routing and Forwarding
EVPN	Ethernet Virtual Private Network
VLAN	Virtual Local Area Network
NVMe	Non-Volatile Memory Express
ETS	Enhanced Transmission Selection
MTBF	Mean Time Between Failures
L12	Full Multi-Rack Level Solution Delivery

Foundations of AI Fabrics: RDMA, PCIe Switching, Ethernet, IP, and BGP

Traffic Characteristics within AI Training that Affect Fabric Design

In a typical AI training job, many sparse matrix calculations are performed on groups of accelerators within a larger cluster. This grouping shares a subset of the job (i.e., the dataset is parallelized, the model parameters are too large for any single accelerator to oversee, the parallelization occurs over time periods, and so on). During specific time periods, the accelerators compute their subset of the job. Once they complete, all parallel members share results with each other (and must acknowledge receipt of all peers' updated information – known as tail latency) before starting the next iteration of that job step. This data sharing between computation iterations causes large “elephant flows” to occur among those accelerators within that subset of the parallel working cluster (not every accelerator in the entire cluster). To minimize the time spent waiting on network I/O to complete, these types of cluster fabrics require lossless, low-latency, and predictable performance.

Characteristics of Elephant Flows on the Fabric

Examining traffic flows and load balancing on a cluster, we identify methods to allocate flows to links on a per-flow basis to prevent the risk of out-of-order delivery. Vendors introduce mechanisms to optimize this with per-packet balancing for improved utilization, if resources are available on the adapters to manage any out-of-order issues and reassemble the data for delivery to the application via RDMA. In this validated design, we aim to achieve a balance between per-packet and per-flow methods. When considering the large flows between accelerators during data exchange among that subset of parallel workers, we notice variability in the data transported within many smaller time windows within that elephant flow.

Remote Direct Memory Access (RDMA), and RDMA over Converged Ethernet Version 2

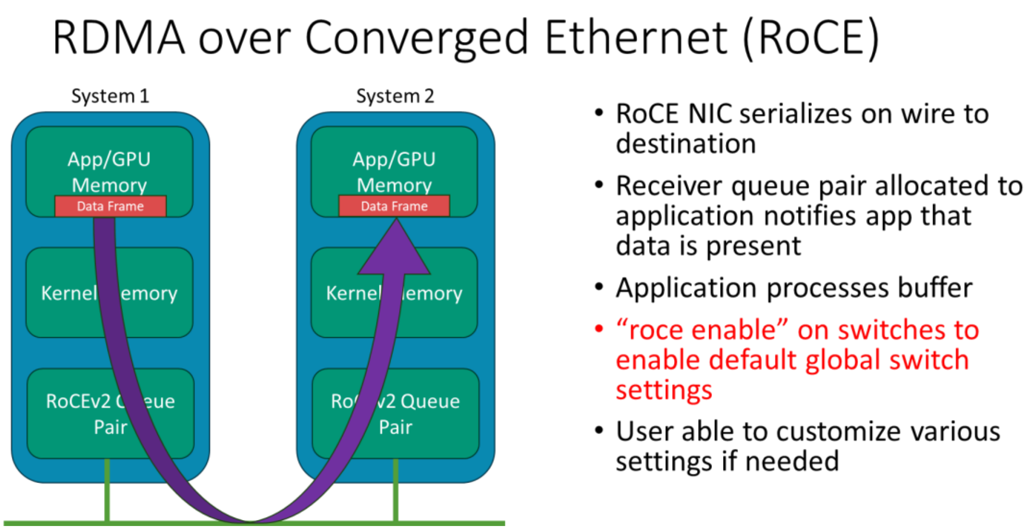


Figure 1 - Remote Direct Memory Access (RDMA)

Remote Direct Memory Access allows traffic to bypass the kernel. It saves the bottleneck of traversing any PCIe/PCIe bridges (if the RNIC and GPU connect on different bus/device/functions -i.e. B:D: F- or transiting CPU and IOMMU complexes) and send/receive directly into the High Bandwidth Memory (HBM3 in our case) in the accelerator.

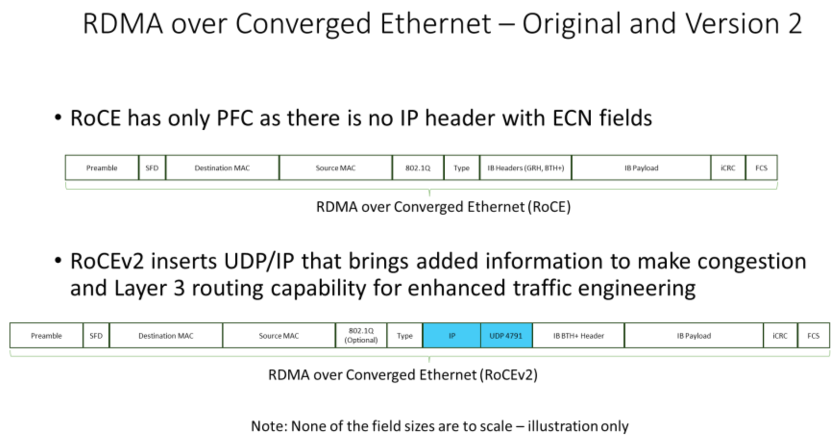


Figure 2 - RoCE and RoCEv2 Packet Structure

The original RDMA over Converged Ethernet – RoCE – transported the InfiniBand higher-level headers in an Ethernet layer 2 transport. To expand the scale and traffic engineering options, the industry went towards RoCE version 2 – RoCEv2 – to allow the encapsulation of those headers in a UDP/IP packet using a well-defined UDP port number of 4791.

Lossless Behavior, Congestion Avoidance, & Path Segmentation

The concepts of enabling lossless behavior and avoiding congestion on Ethernet involve a more extended discussion and exceed the scope of this document. To help the reader grasp some key points throughout this document, we will discuss several fundamental technology enablers essential for achieving lossless, low-latency, congestion-aware, and avoidance technology, which must be present in today's Ethernet fabrics.

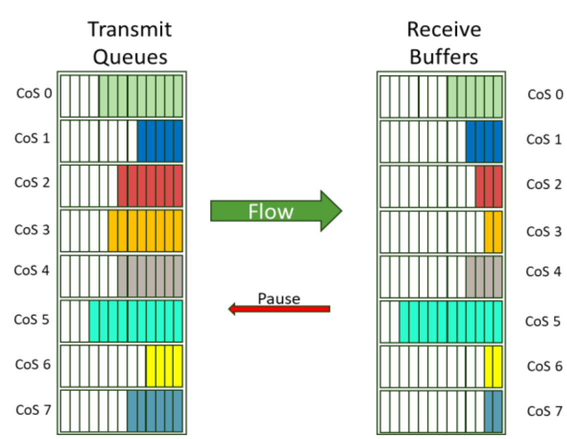


Figure 3 - Per-Priority Flow Control via Pause Frames

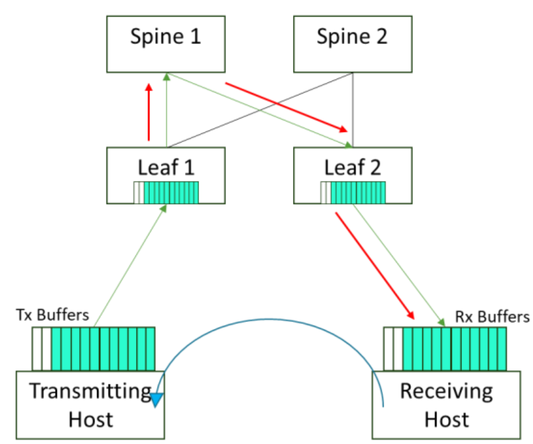


Figure 4 - Explicit Congestion Notification via CNP

Several core individual enablers exist to meet these goals, including Per-Priority Flow Control (PFC), as shown in Figure 4, which allows traffic to be paused by class of service. Another key aspect of managing congestion to ensure lossless behavior is depicted in Figure 3, where an Explicit Congestion Notification (ECN) is signaled by setting a Congestion Experienced (CE) value in the DSCP portion of the IP header on these intermediate switches. This setting enables the receiver to inform the sender to slow down, preventing traffic loss.

This is further extended with optimized buffering within the switch itself, traffic classification and handling methods to properly queue and schedule traffic, and present proper congestion avoidance mechanisms.

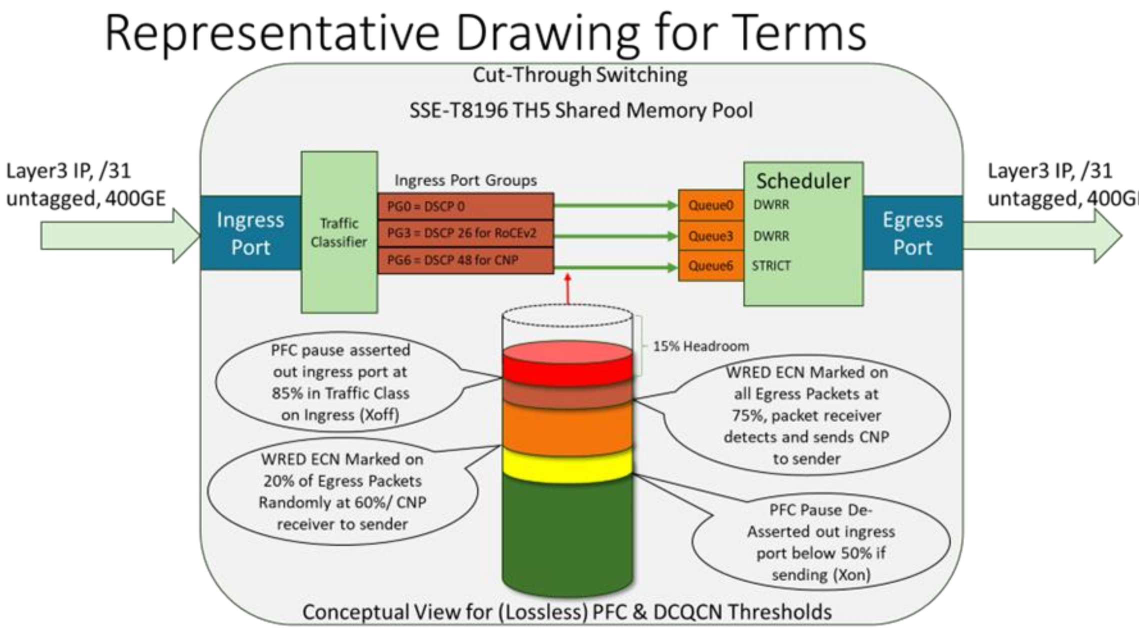


Figure 5 - View of Buffering, Classification, Queueing, and Scheduling in SSE-T8196S

The mechanisms to monitor, detect, and signal all of these are built into modern switching silicon and are already present in a RoCEv2 aware network fabric. As we delve into the precise element configurations for these elements, you can refer to a general representation for the shared buffering on the switch silicon.

Dynamic Load Balancing (DLB)

If we examine the flow, we see a distribution of traffic over a relatively long period at a high data rate. When we zoom in, there are busy transport periods alongside gaps in time between them. Traditional ECMP only uses hashing load mechanisms over multiple paths between endpoints by applying a “5-tuple” of header fields to select a path, which is optimized for the power of 2 link counts, and statically maps the entire flow for its entire duration to a single member link in a set of links. This can result in some links being congested while others are relatively lightly loaded.

The hardware in this solution extends significantly by measuring recent traffic trends and future patterns through the depth of queued traffic. This allows for the dynamic selection of ideal links to optimize flows at the overall flow level, while also accounting for individual packets to prevent out-of-order delivery risks to the destination. Achieving this requires analyzing the timing of end-to-end flows and identifying gaps within a given flow, enabling the movement of “flowlets” to dynamically link members with lighter loads. This approach is known as DLB(In SONiC UI, it will refer to Adaptive Routing and Switching) and is deployed in the underlying Tomahawk silicon found in the Ethernet switches of this design. These methods monitor congestion on specific links and serve as inputs to other protocols, controlling traffic on alternative links for optimal path segmentation and load balancing without necessitating more resource-intensive adapters on the hosts in the solutions.

Using Border Gateway Protocol (BGP) and Adaptive Routing and Switching (ARS) On the Fabric

Ethernet has employed Layer 2 mechanisms throughout its history (Spanning Tree and other more modern methods) to ensure a loop-free path in the fabric for hosts to communicate. When a link or node fails, the directly connected device removes entries from the MAC tables associated with the failed pathway. Traditionally, timers needed to expire for elements in the fabric to seek alternate paths (static routing). In addition to link or device failures, congestion scenarios were generally not addressed until PFC, ETS, DCQCN with ECN, WRED, and others on the Ethernet fabric were introduced. Many end users now utilize approaches like BGP routing, and the adaptive routing introduced with DLB to enhance recovery from both link and node failures while effectively adapting to congested links. Users apply these and other techniques within the data center to implement more active adjustments to the fabric. These mechanisms are highly effective at tracking and adjusting traffic patterns and re-establishing pathways without human intervention. Figure 6 below illustrates an example of the topology (from a public paper by Facebook) of an ideal BGP deployment for the scale and topology we will use. In this scenario, the leaf nodes advertise the reachability of the connected systems to the spines (and other leaf nodes also) so that effective balancing can occur when combined with information about link utilization and other metrics. With the EVPN type of BGP, we can interconnect multiple leaf-spine fabrics to extend the size of the clusters and allow direct connectivity between nodes as needed with Ethernet Spine and/or Super Spine Plane designs.

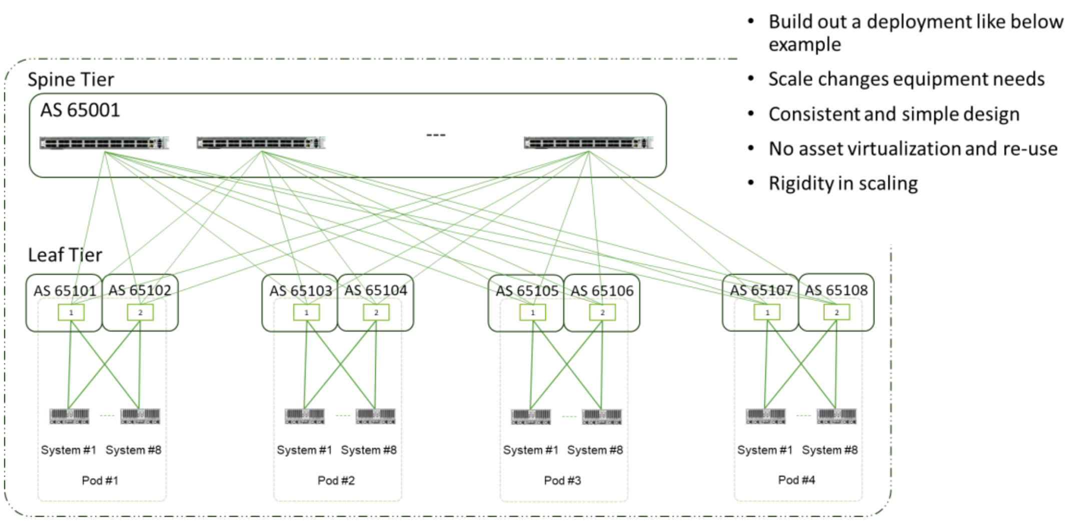


Figure 6 - BGP example from Meta for Ethernet Leaf-Spine architecture for maximum availability and traffic balancing

Hosting Multiple Lines of Business or Customers on a Virtualized Common Fabric for New Usage Models

One extension we see increasingly deployed is for providers to offer accelerator services to multiple entities, internal and external, to a company. Due to the perception of security isolation, early clusters were often built on a per-customer basis and were fully dedicated to that specific business need. Over time, it has become evident that a layer of virtualization is required to support multiple tenants (i.e., in cases where one tenant stops using services, goes out of business, or alters their resource needs) in a dynamic manner—while ensuring tenant isolation and full performance that matches that of a completely dedicated cluster. Many providers of accelerator services are grappling with how to offer services with minimized service length obligations and the fastest time to bring their customers online as key business differentiators. The critical factor here is achieving monetary break-even (and profit beyond that) points relative to the cost of dedicated clusters for each tenant under what terms. Much like the rapid growth of cloud computing offerings surpassing Infrastructure as a Service (IaaS) by utilizing virtualization and secure segmentation for compute, storage, network, security, and other components, a similar maturity in virtualization and secure segmentation for accelerator offerings needs to develop and is included natively in this validated design.

To that end, designs like the one shown in Figure 7 below are utilized to keep resources segmented even when deployed on a common accelerator utility base. Ethernet and IP offer built-in mechanisms that are widely used in industry to achieve these goals and have often been deployed in large enterprises and cloud solutions. This validated design document includes the topology to support this and some example schemas that can be deployed in real-world clusters today. Multiple methods exist to share individual accelerators among tenant customers via host virtualization and other means. Still, for this design and the scale of deployments we are exposed to, having granularity at the node level is most common, and this solution will assume that level of granularity. If the provider builds capacity ahead of demand trends, they can offer some portion (or all, if they buffer enough) on a very short lead time compared to the competition.

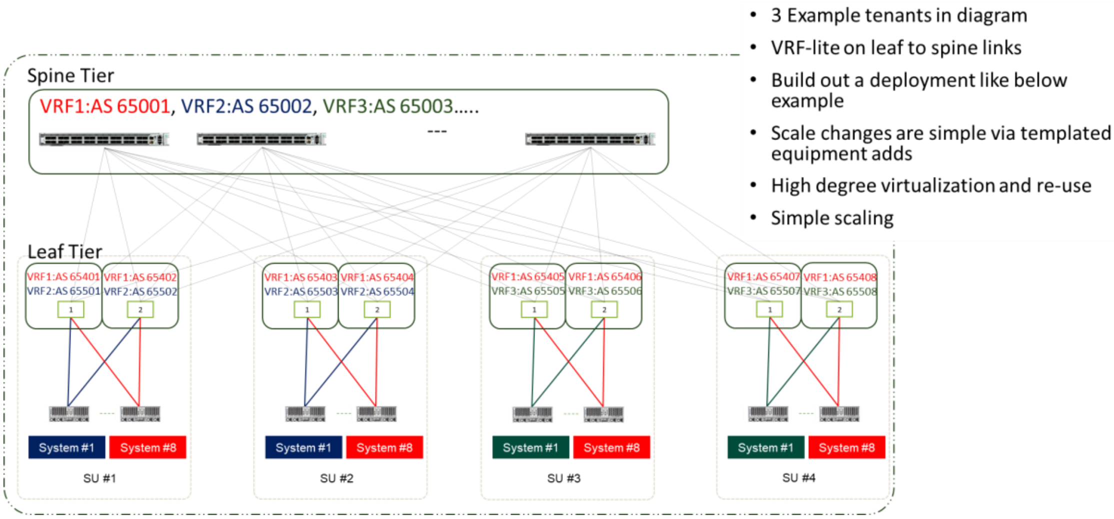


Figure 7 - BGP Sample Multi-Tenant Segmentation

Managing Incremental Growth to the Accelerator Cluster

Collaborating with the previous discussion on building a shared cluster utility, which facilitates rapid onboarding, offboarding, and adaptive business models for customers, there is frequently a goal of establishing just-in-time capacity or a limited set of reserved resource buffers. As services are consumed, the buffer is depleted, and providers can replenish it to stay ahead of demand. One way to scale is by replicating the dedicated clusters mentioned earlier; however, allowing tenants to scale clusters without the limitations imposed by inter-cluster boundaries necessitates the creation of a consolidated set of new resources to support seamless tenant growth. To achieve scalability, we must incorporate more service units (SUs) with nodes and their connected leaves. We then require a method to distribute traffic evenly across the existing and newly added spines in the design. If we “swing” some of the leaf uplinks from certain spines to the new spines, we may encounter challenges in establishing new subnets, configuring routing, load balancing, and other issues. A key principle of this validated solution is to utilize “BGP unnumbered” links, which eliminates the need for manual adjustments to any port IP or routing information. These link movements do not require taking down the clusters; however, the bandwidth between the SUs would be halved when reallocating a portion of those links to the new spines, as depicted in the red links in Figure 8 below. This process can be performed during periods of low cluster usage (to minimize the impact of reduced bandwidth during expansion) as often as necessary. Tools from Supermicro can assist engineering teams in creating detailed workflows to simplify this growth.

DOUBLING 32 TO 64 AMD INSTINCT MI300X/MI325X SYSTEMS (256 TO 512 ACCELERATOR)

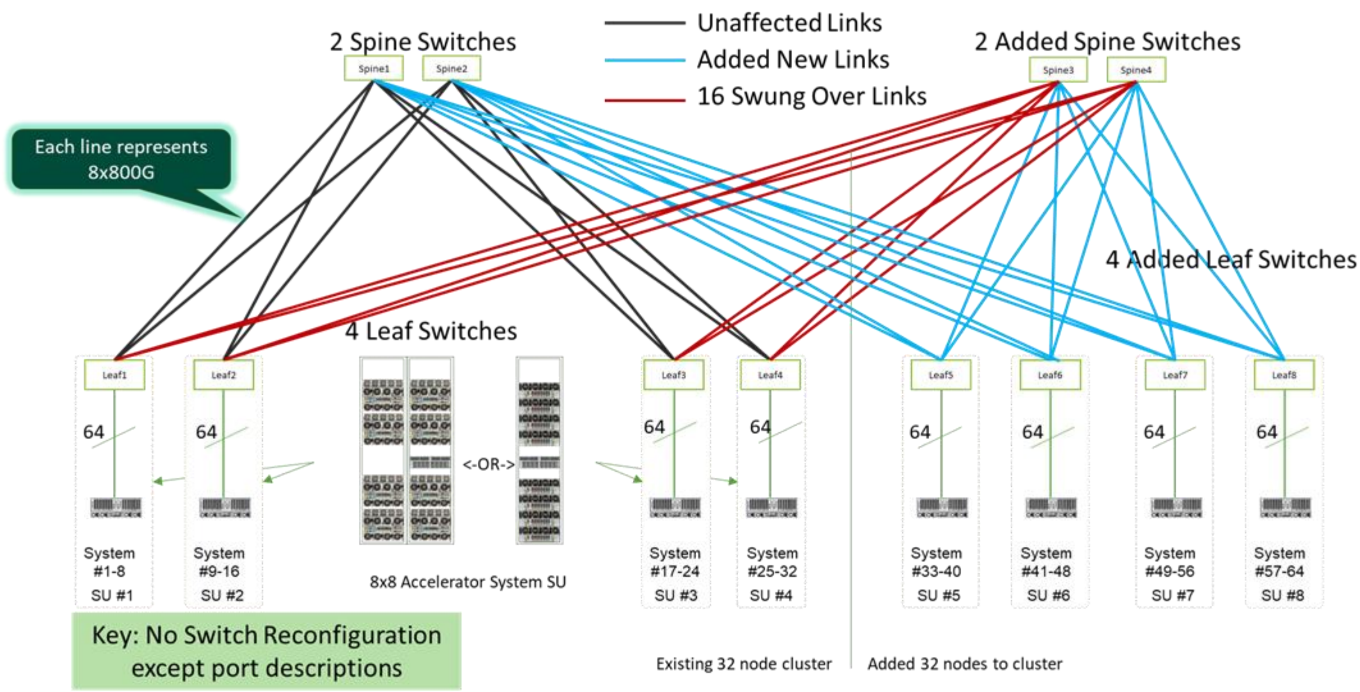


Figure 8 - Doubling the solution cluster to a 64-node example

Validated Design Equipment and Configuration

MI325X Server Utilized in a Validated Design

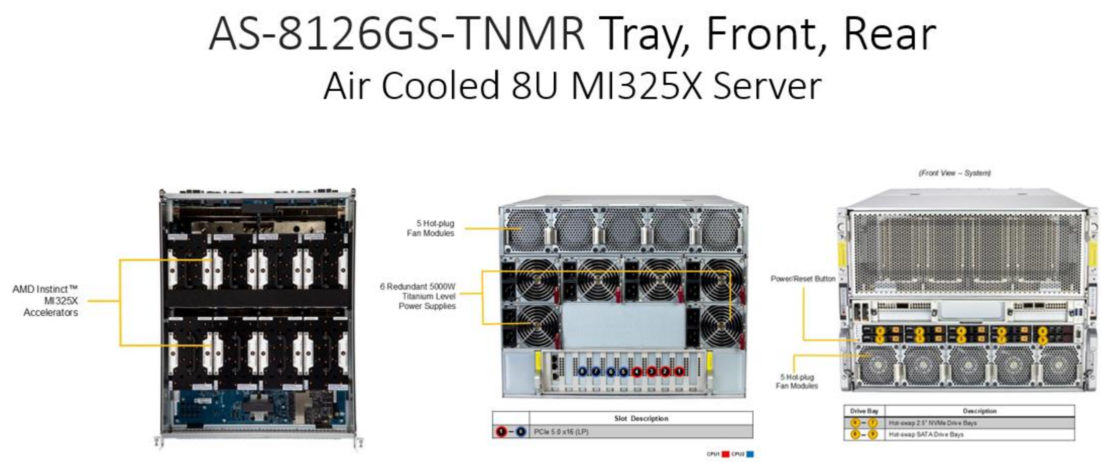


Figure 9 - Supermicro AS-8126GS-TNMR

Supermicro offers various server options compatible with many types of market accelerators, with Figure 9 above displaying our air-cooled 8 AMD MI325X solution. One key upfront design requirement for these servers is to facilitate optimized RDMA traffic with minimal distance, latency, and silicon between the RNIC and the accelerator. Below is a block diagram of the system, presented to outline some key performance considerations that occur well before traffic enters the scale-out fabric.

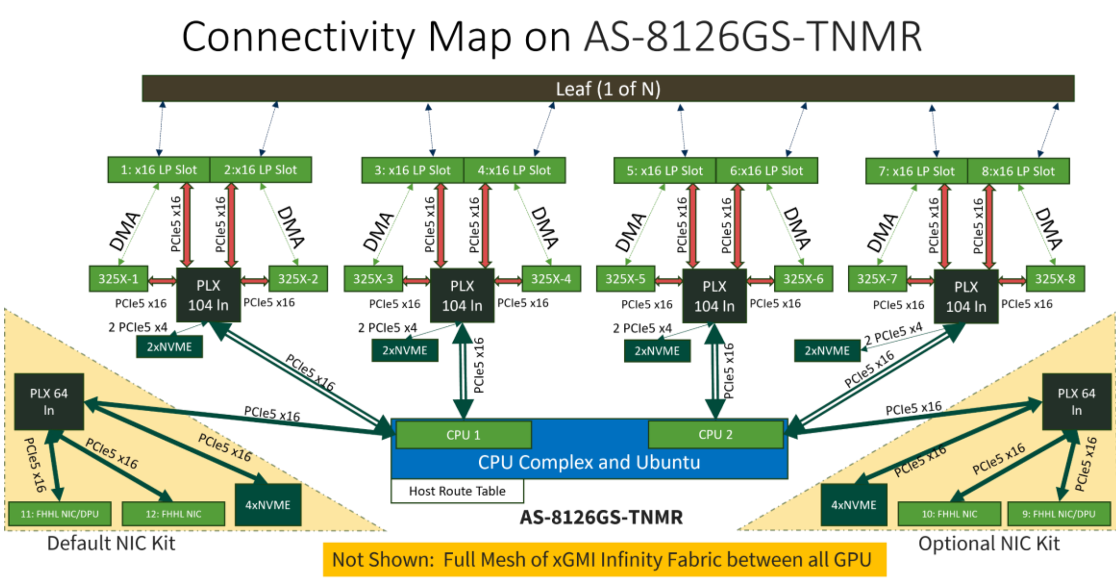
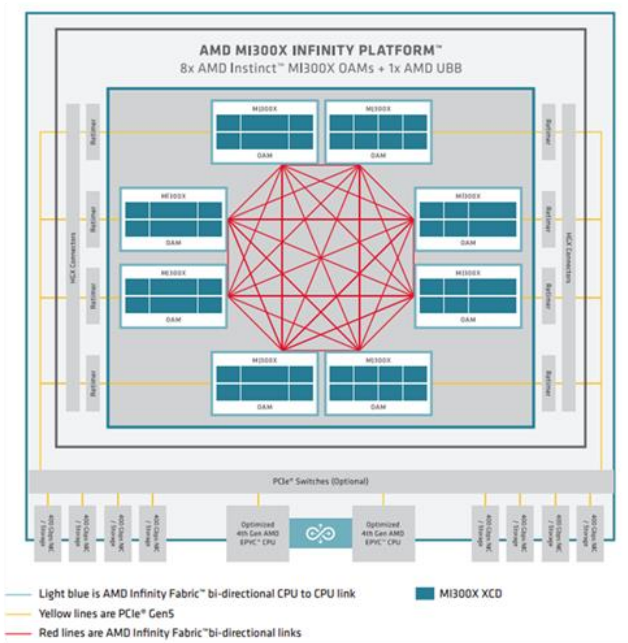


Figure 10 - Block Diagram of AS-8126GS-TNMR

Figure 10 illustrates the direct connections on dedicated PCIe switches, ensuring that RDMA traffic is localized for each accelerator. This DMA is also present from the accelerators to the local NVMe on the PLX switch for scratch drive usage, allowing data transfers without going through the CPUs. The NIC kits displayed support for 10/25GE links to external storage devices. Our validated design specifies 200GE storage access for each CPU to its connected accelerators, which minimizes inter-CPU transfers. Additionally, in-band networks permit firewalled internet access for tasks such as Linux package operations, driver updates, tenant access, etc. Note, the 1GE copper BMC access is not shown above.

AMD Instinct™ MI300X & MI325X GPU Internal Fabric Basic View



Gen5 PCIe on all. x16 means 500Gbps possible

Figure 11 - AMD Instinct™ MI300X & MI325X GPU Infinity Fabric™ Technology

Figure 11 above gives a simple view of the internal links and the MI325X OAM. All accelerators are directly meshed for optimal performance.

Scaling out the Accelerators with an Optimized Ethernet Fabric – Components and Configurations

AMD Pensando™ Pollara 400 Ethernet NIC for RoCEv2 Fabrics is shown below.

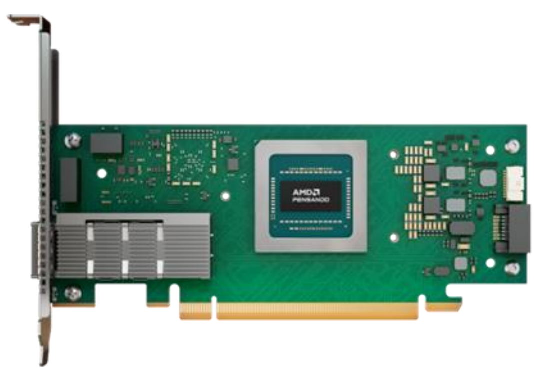


Figure 12 - AMD PensandoTM Pollara-1Q400P RNIC

The RDMA NIC (or just called RNIC here) is developed by AMD PensandoTM using their Pollara (Pollara-1Q400P) Silicon. This PCIe Gen5 x16 adapter (standup PCIe in this SVD, but an AIOM for OCP 3.0 TSFF Mezzanine is also offered) is plugged into 8 LP slots on the server above. These then have a direct set of 16 lanes of PCIe Gen5 to each accelerator.

SSE-T8196 Ethernet Switch

Supermicro designs for AI networks are consistent across many of our partner accelerator solution vendors. Supermicro designs, builds, and markets multiple 400GE and 800GE switches to support these designs. The foundational technical software features and functionality deployed differ across other network vendor solutions, but Supermicro uses the industry standard open Software for Open Networking in Cloud (SONiC). All the Ethernet features and functionality referenced in this paper are included in SONiC. To gain more familiarity with SONiC, we encourage tutorials and the deployment of a virtual switch environment. This document provides an optimized SONiC configuration for our example cluster. More customers and partners are deploying Supermicro switches due to the short lead times, desire for a fully validated solution rack from fewer component vendors inside, and the bench strength of all things AI that we provide. Brief overview below:

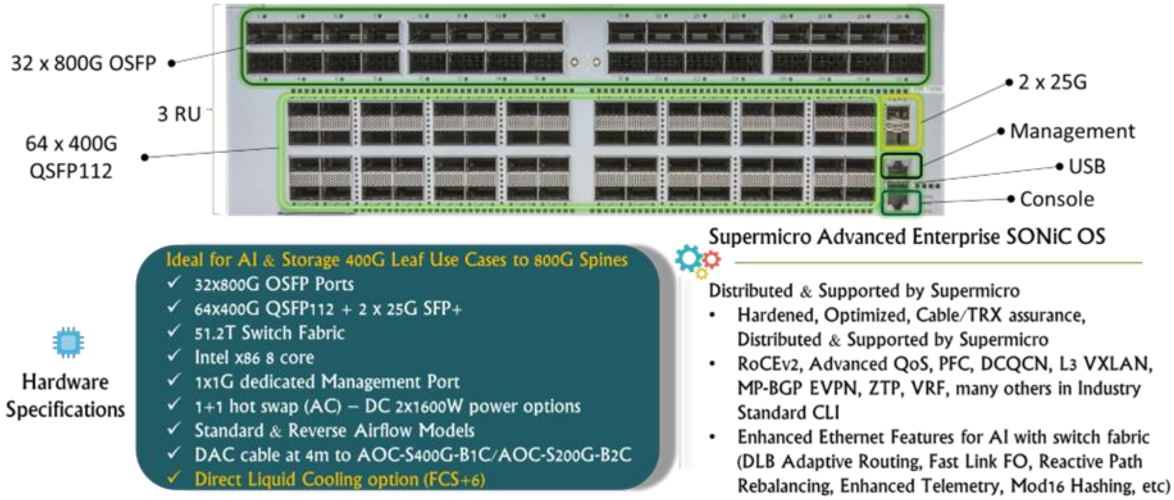


Figure 13 - Supermicro SSE-T8164 Ethernet Switch

The SMC SSE-T8196 Ethernet switch is the leaf in our AI network cluster design. In this design, Supermicro will use direct 400G QSFP112 links to all Pollara instances in our AMD Instinct™ MI325X systems. Supermicro has similar designs for other accelerator offerings, with a roadmap for regular updates to this and other designs. Please contact your Supermicro sales team to learn more. Each AMD Instinct MI325X GPU is connected to an AMD Pensando Pollara 400G NIC interface on the system PCIe fabric.

Design of the Scale Unit - Scaling Out the Cluster

All Direct Attached Copper Scale Unit Design

These designs are based on a fixed infrastructure unit that can scale out (SU) to accommodate a wide range of cluster sizes. The leaf devices are positioned in the middle of one of the two racks for this validated design of an air-cooled 2-rack SU. If DLC is an option, we can reduce the footprint to the center of the single rack that comprises this SU. Within this cluster are eight systems, with each rack having a maximum estimated power of approximately 80kW for our MI325X in the 4U server design. The rationale for the placement is to optimize the connectivity for copper connections while primarily reserving the more expensive fiber connections for the leaf-spine connections. Figure 16 below illustrates the air-cooled version of the SU. The Switch and RNIC depicted below feature an enhanced SERializer/DESerializer (SERDES), allowing a Direct Attached Copper (DAC) solution up to 4 meters in length. However, an important point to note is that for optimal performance, Supermicro recommends using the inner half of the OSFP ports to connect to the RNICs on the systems and the outer ports for the spines.

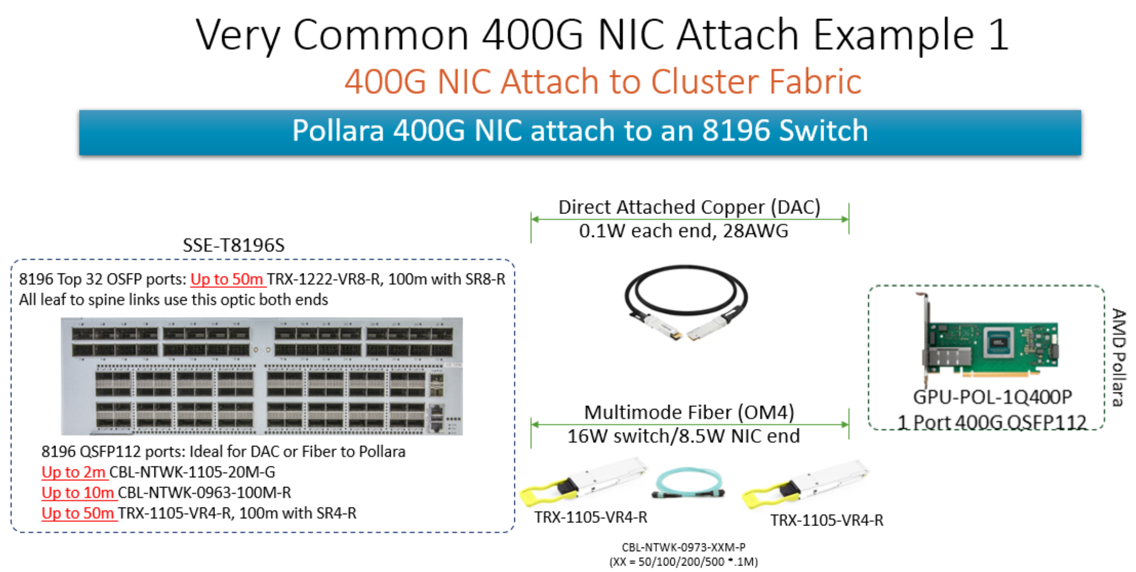


Figure 14 - Common DAC and Fiber leaf to RNIC Links for East-West Traffic

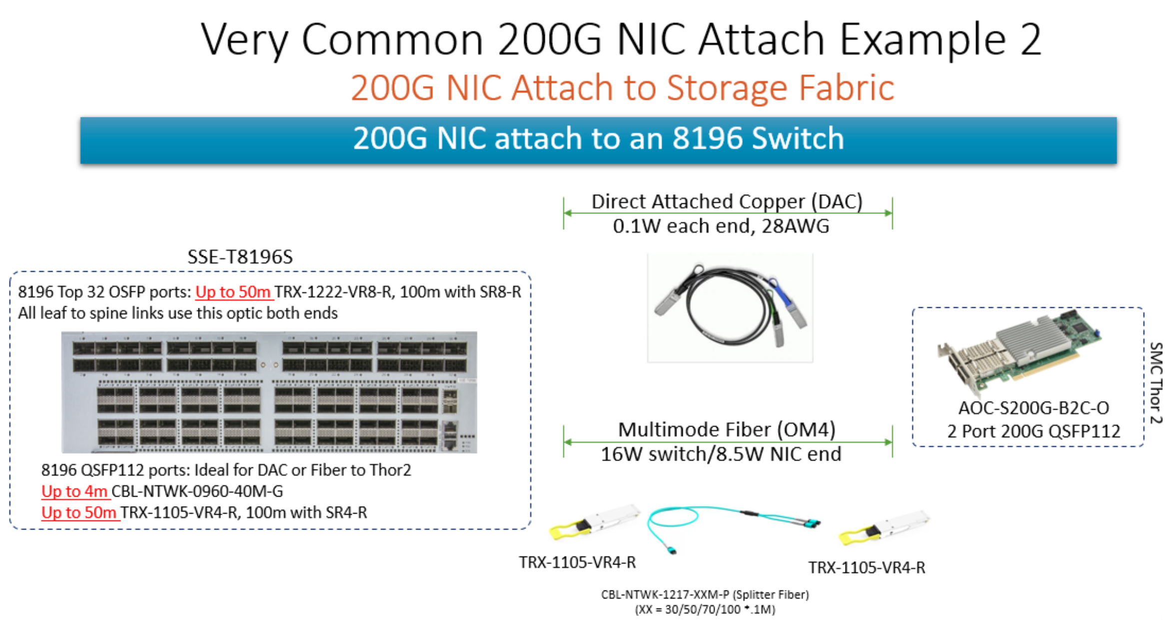


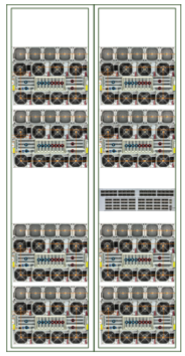
Figure 15 - Common DAC and Fiber Leaf to RNIC Links for North-South Storage Traffic

There are a few reasons to prefer passive copper for intra-SU links in this design, including:

- Cost (order of magnitude saved per link)
- Availability (500x increase in MTBF)
- Power (0.2W per link vs. typical 14-16W per optical link)

Air Cooled SU Layout and Cabling Inside

SMC Design for AMD Instinct MI300X/MI325X in 40kW Air Cooled Racks



2 Racks = 1 Scale Unit (SU)

- SU design allows for lower cost DAC cabling (2m)
- SU scale out for various cluster sizes
- SU is 2 racks
- Rack has 4 systems
- SU has 1 leaf, 1 optional storage switch, management (not shown), 2x optional storage servers
- 8 Example Systems: AS-8126GS-TNMR
- 8 x AMD Instinct MI325X per system (64 in SU)
- Per rack power: ~40kW
- 8 x Pollara 400G NIC per system

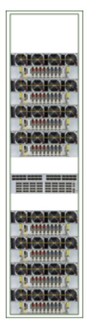
1. *Design Key is Power so 64 Accelerators/SU*
2. *Using NIC is 1 port Pollara-400-1Q400P with QSFP112 (AMD)*

Figure 16 - Supermicro Air Cooled AI SU with AMD Instinct MI300X/MI325X GPU Servers and Accelerator Fabric Switches

Figure 17 below illustrates the same SU design but becomes denser by implementing Direct Liquid Cooling for 8 hosts within a single rack with CDU. The core system design now requires only 4RU to accommodate 8 MI300X/325X Accelerators, while consistently performing alongside the air-cooled version of this SU.

Liquid Cooled SU Spacing and Cabling

SMC Design for AMD Instinct MI300X/MI325X in 80kW Liquid Cooled Racks



1 Rack = 1 Scale Unit (SU)

- SU is 1 rack with Direct Liquid Cooling
- Intra SU leaf to NIC all copper (using 2m)
- Design will assume linear racks for cable lengths
- Per rack power: ~80kW (1 CDU)
- Rack has MI300X/325X systems, management switch*, leaf, storage switch, storage
- 8 Systems per rack: AS-4126GS-TNMR-LCC
- 8 x AMD Instinct MI325X per system (64 in SU)
- Ethernet IO: 8 x Pollara 400G NIC per system

1. *Design Key is Power so 64 Accelerators/Rack*
2. *Using NIC is 1 port Pollara-400-1Q400P with QSFP112 (AMD)*

* Not Shown here for clarity

Figure 17 - Supermicro DLC AI SU with AMD Instinct MI325X Servers and Accelerator Fabric Switches

The connections between the leaf and the eight systems comprise eight sets of OSFP 800G in 2x400G linked to a pair of QSFP112 400G on the NIC. The leaf-spine connections feature OSFP 800G each, as illustrated in Figure 18 below. To scale out the number of systems in the cluster (the validated design depicted is a 32-system cluster for a 256 Accelerator cluster), we need to increase the spine count and add more SU's.

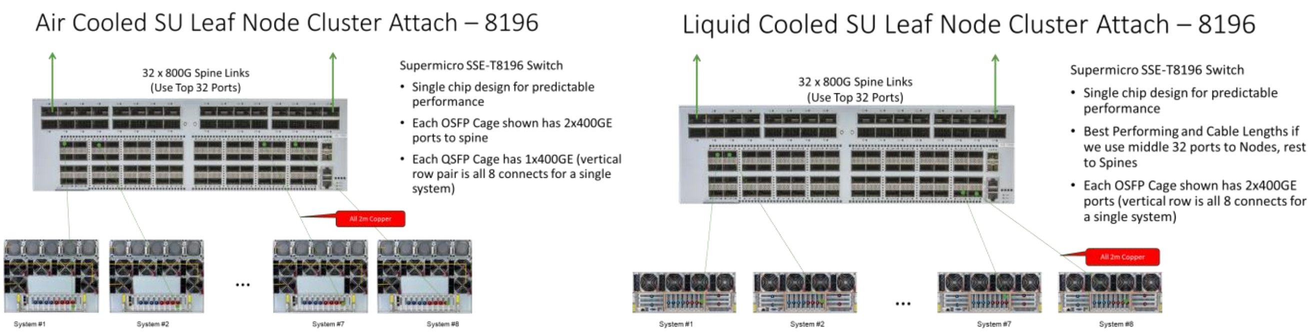


Figure 18 - System to Leaf Links for Air Cooled or Liquid Cooled MI325X Systems

Resource Management and Adding Locality into Work Placement (SLURM and Topology Optimization Including Concept of Rails)

As Figure 18 illustrates, all accelerators within the systems of a single SU connect to a single leaf. Availability considerations are managed at the resource manager layer, where job steps are checkpointed. If any accelerator or node fails, the workload is distributed among the surviving accelerators to resume from the last checkpoint. Consequently, the fabric has no inherent node-level redundancy concept, as each accelerator is linked to a single RNIC. This feature results in the optimal design of one leaf supporting eight systems in this configuration, utilizing a short copper cable.

One key consideration, however, is that if the resource manager controlling the job execution environment is unaware of the fabric, only two levels of adjacency are available. Suppose the job or step runs on a parallel set of accelerators lower than the count on a given system (eight accelerators in this solution). In that case, all collective communications are local to a single node, and traffic never needs to leave that node. If the resource manager utilizes more accelerators, which are very common in today's models, the only other option would be to use the east-west (or back-end) fabric. The issue is that this fabric may have a single silicon hop, three hops (leaf-spine-leaf), five hops (leaf-spine-superspine-spine-leaf), or even more if any of the switches are modular chassis, which typically have multiple hops within a single node. This leads to the potential for significant disparities in latencies for transporting collective communications, making these accelerators wait for a final data send (tail latency) to ensure all workers are in sync before moving to the next computation iteration.

Various customers utilize custom and open-source tools to provide a level of locality and segmentation to minimize this issue. Toolsets like the Simple Linux Utility for Resource Management (SLURM) are deployed in approximately 50% of today's supercomputer installations, offering various methods to achieve this outcome. For this discussion, we will focus solely on SLURM due to its open nature in describing the methods. One way to optimize flows is by using the immutable position of the accelerator within a given system (known as that accelerator's rank) and designing the fabric so that all ranks within a Service Unit (SU) connect to the same leaf. In this design, we will have a system of 64 accelerators, tightly controlling latency for optimized performance with more than 8 accelerators while reducing the variability across the entire cluster. This method of utilizing a rank is termed "rail-optimized," where the set of local system ranks collectively form a rail (i.e., all 64 nodes ranked 1 connect to a leaf for rail 1). These rails then merge into the spine tier for all accelerators to communicate together; SLURM ensures that workloads requiring fewer than 64 parallel members reside on that specific rail, keeping the rest of the fabric free of that traffic. For users, there are command line arguments like "--gres=gpu:1" and "--nodelist=/pathtofile/su1.nodelist" included in the sbatch commands to specify that rail within a particular SU.

Alternatives to Optimize Resource Management while Optimizing Connectivity

In these designs, however, having many nodes reach common leafs leads to lengthy cable runs, implying a need for fiber connections as copper often becomes cost-prohibitive. In this validated design, we not only use copper for the many advantages mentioned earlier in this document but also localize a set of nodes and their accelerators into a group not based on the rank of the accelerator on a node, but rather on all nodes and ranks under a given leaf. By not requiring many nodes to connect to each rail, we can keep the advantage of predictable performance and a single stage. We can signal the SLURM to treat the 8 nodes and 64 accelerators under a leaf as a single stage, not segmented by rank but by rack.

Suppose we utilize a Slurm topology/block plugin introduced in Slurm 23.11 and refined in 24.05. In that case, we can integrate this rack optimization and adjacency into resource scheduling to reap the benefits of a rail-optimized design, specifically implementing the rails at the rack or racks block level. The drawing below illustrates how this would appear for this validated design.

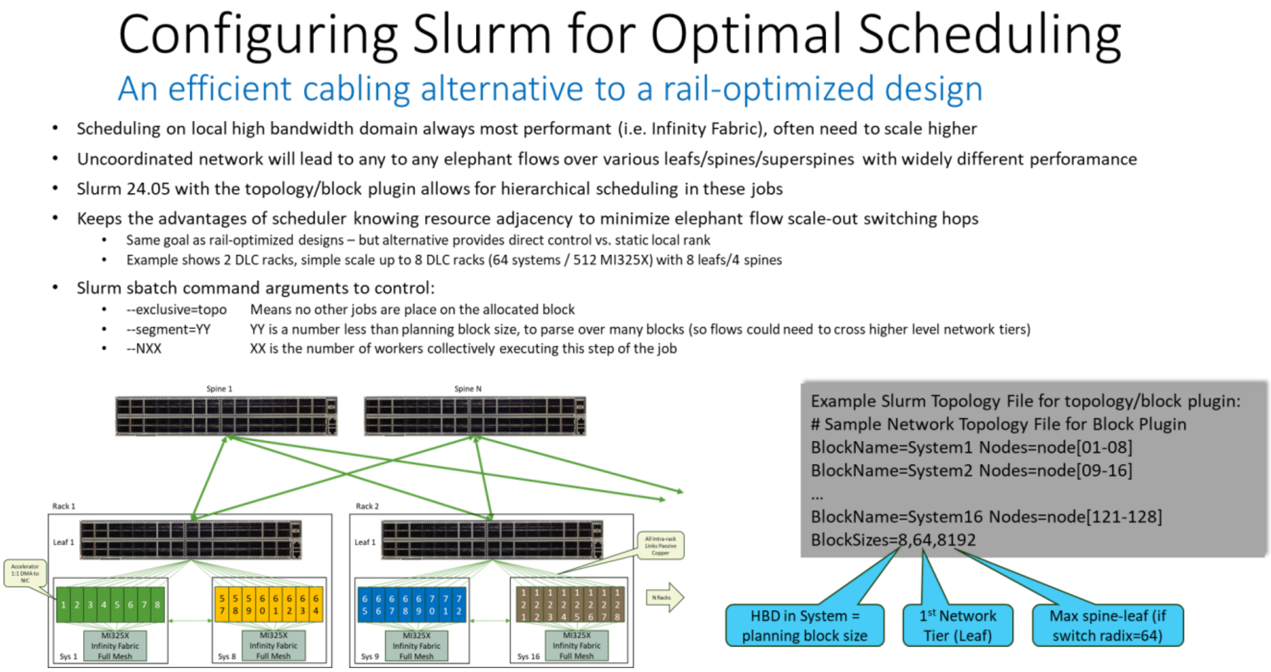


Figure 19 - Slurm Topology/Block Plugin

Supermicro Validated AMD Instinct MI325 Design: 4 Scale Units for 256 Instinct MI325X GPU

Figure 22 below illustrates the hardware layout of a 4 SU cluster interconnected by 2 spines. This 2-tier design assumes no over-subscription on the fabric. Depending on the power footprint, the individual scale units can be either 1 or 2 racks, and this design is highly adaptable, which we utilize in all our design scales. If we maintain the design constraint to a maximum of 2 tiers, this solution with this product set can scale up to a maximum of 8k MI325X accelerators in the cluster shown below.

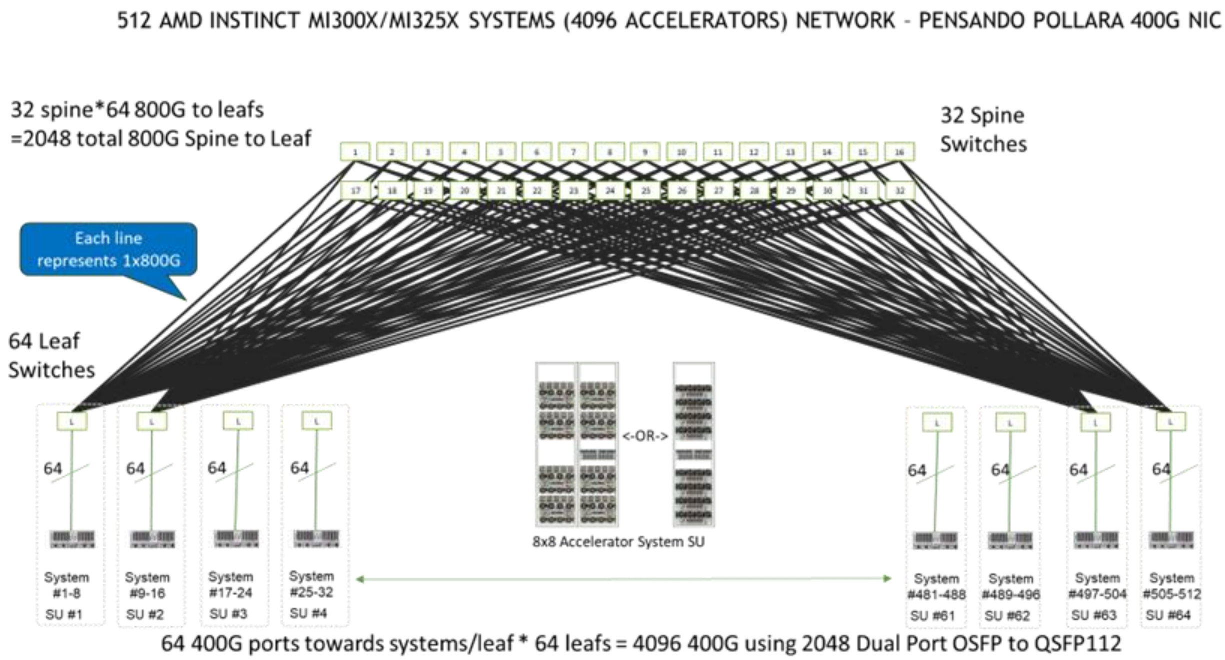


Figure 20 - Maximal 2 Tier Fabric Cluster for This Solution

If even higher scale numbers are needed, we can grow into a 3-tier topology using a superspine and retain full non-blocking performance on the cluster as shown below. We can go even further if needed, but adding more superspine planes.

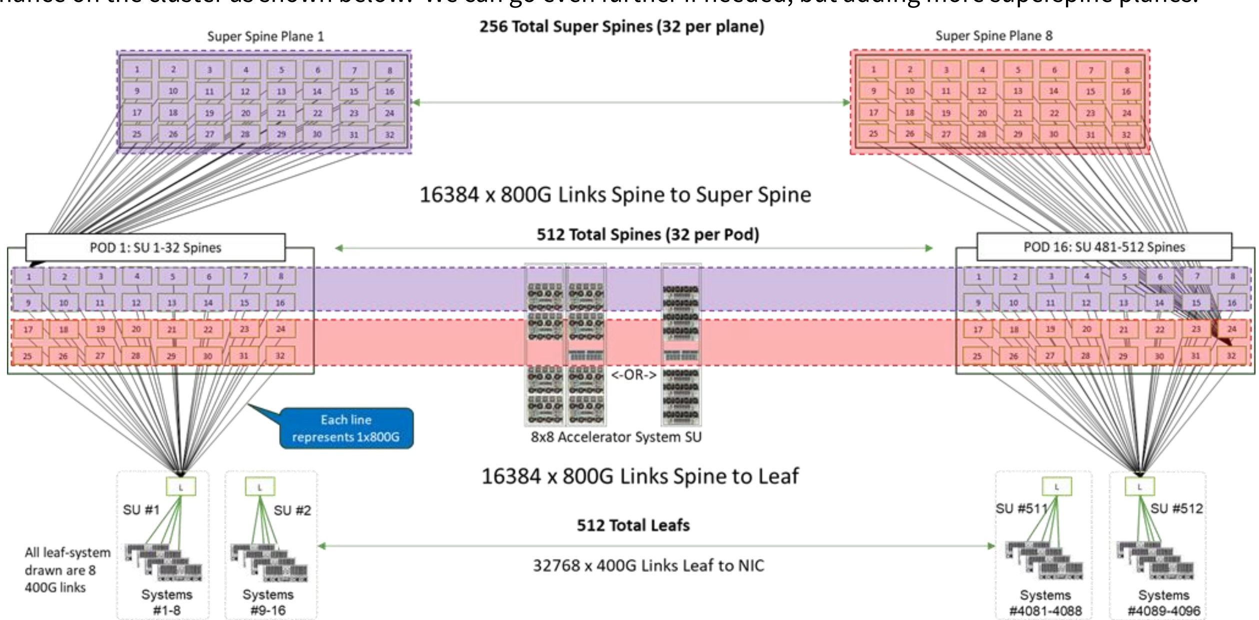


Figure 21 - Sample 3-Tier Fabric Cluster for this Solution (Can scale much higher)

Handling Transient Flow Hashes on Fabric

When accelerators complete a given calculation as part of the overall training process and start the process of collective communication to share its work, multiple large flows are established to the parallelized group of peers working together. These flows we classify as elephant flows for large sizes, but they are not steady and constant per the DLB section of this document. Flows will use a combination of initial hashing to determine links to use, which DLB optimizes for congestion based on the flowlets discussed earlier. This can show transient bandwidth overlaps, where many links may be utilizing a higher percentage of the bandwidth. Hence practitioners in the industry look at some alternative methods to account for these and internal Supermicro research has resulted in a preference in this area. Figure 22 below shows some common alternatives, and a method of using different bandwidths on the node to leaf, vs the leaf to spine, such that a lower percentage of these higher utilized flows will run out of bandwidth before they complete rebalancing with DLB. This delays the utilization of PFC and ECN signaling and results in overall better performance.

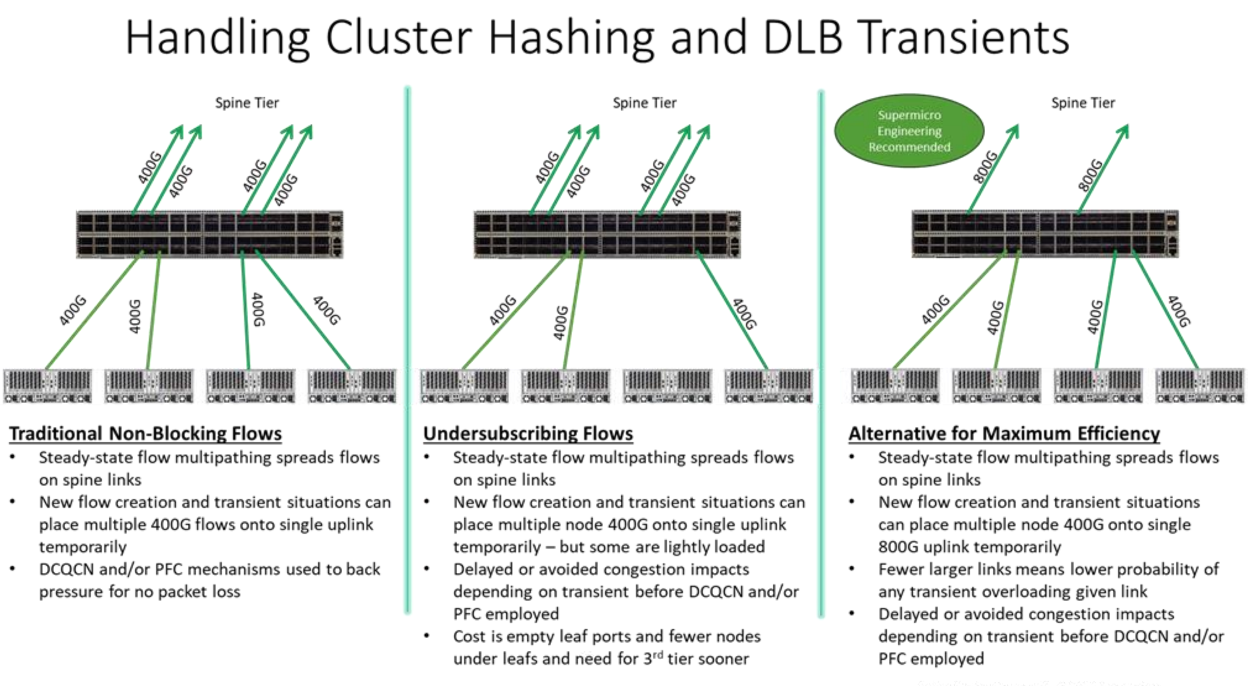


Figure 22 - Transients in Hashing and DLB adjustments

Grouping of Parallel Links from Leaf's to Spine's

There are multiple parallel links from each switch to each spine – and a network engineer may consider alternatives of a portchannel of those links, or a set of parallel BGP unnumbered links to achieve this connectivity. When we just use Equal Cost Multi-Pathing (ECMP) alone, often we desire a power of 2 on the number of spines such that the traffic can be evenly balanced over these links. In variations of these validated designs (i.e. to add some odd number of SU's, etc.) you would see cases where numbers of spines and links are not always guaranteed to be a power of 2, and with traditional switching silicon this leads to unbalanced flows (meaning some links can be carrying 2x the traffic of others). Also, with default ECMP only the entire flow is hashed to a member link. Fortunately, the Broadcom Tomahawk 5 silicon has a modulo 16 hashing that can produce an even set of probabilities over even non-power-of-2 links. While that helps, we can even go further with the deployment of Dynamic Load Balancing (DLB) we talk to above. This is an adaptive routing implementation which is referred to as Adaptive Routing and Switching (ARS) in the SONiC CLI. With this we not only get a per-flow hashing to any number of links and/or spines, but we also go more granular than a per-flow has to a "per-flowlet" hash we talk to above. This provides an intermediate solution between per flow hashing (to guarantee in-order delivery on any RNIC) and per-packet hashing (needing to reconstitute the sprayed packets on the receiving RNIC forcing more resource requirements on that RNIC) to allow for the per-flowlet hashing based on RTT measurements such that we can spread a slow over parallel links and still achieve in-order delivery on standard RNICs. This all results in a recommendation to not to a port channel from leafs to spines, rather do a parallel set of unnumbered BGP and let the DLB perform this optimal action.

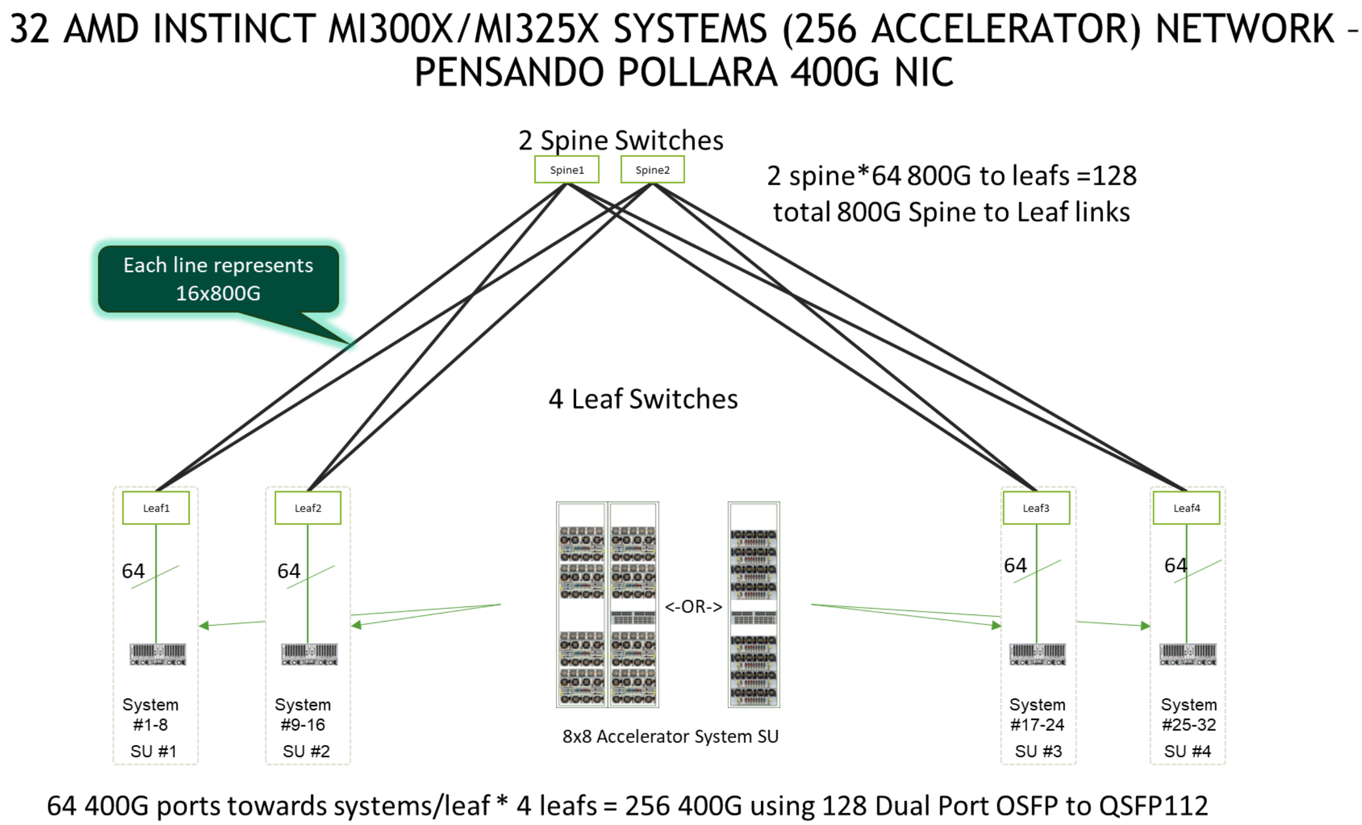


Figure 23 - 32 system 256 Accelerator design with 400G to each MI325X

Validated Design Architecture and Assumptions to Result in Detailed Configurations

- Example is for a 4 leaf, 2 spine cluster network (logic below can be scaled to much higher counts, however)
- Front end, north-south bonded interface pair (2x10GE) and its IP are not within the scope here as providers have well established methods to assign to various tenants for their use – but we will generally call out the hostname which maps to those IP's as carona_node1 through _node16, heineken_node1 through _node8, and budweiser_node1 through _node8
- Simple cluster fabric IP addressing illustrated (use 100.64.0.0/10 space RFC 7793 with 64+Tenant ID for 2nd Octet with none set via DHCP nor in DNS as these are dedicated to back-end east-west cluster traffic
- Design supports 64 max tenants, max nodecount of 32,768 per tenant (can adjust boundaries for ratios of these)
- We don't summarize on leaf boundary if tenant nodes can be anywhere (but the general goal is to keep tenant nodes together)
- Router-id's will be:
- 100.64.0.0/32-100.64.0.255/32 for super spines (256 max if 3-tier cluster)
- 100.64.1.0/32-100.64.2.255/32 for spines (512 max)
- 100.64.3.0/32-100.64.4.255/32 for leafs (512 max)
- vtep IP's will be on 100.64.5.0/32-100.64.6.255/32 on leafs (512 max)
- For ease of troubleshooting, I will just start above at base 1, however (i.e., leaf1, spine1)

- Each GPU then has a /31 route in the cluster table, where the IPv4 route scale of almost 1M will fit
- All RNICs are assumed Pollara-400-1Q400P at 400GE
- In AI we run L3 to each RNIC on the hosts, with vrf segments per tenant
- all GPUs talk to each other via L3 only whether under a single leaf or multiple
- 3 Tenants numbered 1 and up - Corona, Heineken, and Budweiser with vrf names to match
- although this configuration and example will fully work with just 1 tenant, also
- IP subnets for tenants
- Corona subnets are of 100.65.0.0/31 and up
- Heineken subnets are of 100.66.0.0/31 and up
- Budweiser subnets are of 100.67.0.0/31 and up
- VLANs 61 (Corona), 62 (Heineken), 63 (Budweiser) are the tenant VLANs assigned for transport in the L3 VXLAN (60+tenant ID)
- Corona has 128 Accelerators with 32 under each leaf, Heineken has 64 accelerators with 32 under leafs 1-2, and Budweiser has 64 accelerators with 32 under leafs 3-4 (but with design, they could be spread anywhere on the cluster)
- Leaf1 has both Corona and Heineken on multiple interfaces (Eth 1/33-1/64 for Corona nodes, Eth 1/65-1/96 for Heineken nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/16, while uplinks to Spine2 are Eth 1/17-1/32 all at 800G
- Leaf2 has both Corona and Heineken on multiple interfaces (Eth 1/33-1/64 for Corona nodes, Eth 1/65-1/96 for Heineken nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/16, while uplinks to Spine2 are Eth 1/17-1/32 all at 800G
- Leaf3 has both Corona and Budweiser on multiple interfaces (Eth 1/33-1/64 for Corona nodes, Eth 1/65-1/96 for Budweiser nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/16, while uplinks to Spine2 are Eth 1/17-1/32 all at 800G
- Leaf4 has both Corona and Budweiser on multiple interfaces (Eth 1/33-1/64 for Corona nodes, Eth 1/65-1/96 for Budweiser nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/16, while uplinks to Spine2 are Eth 1/17-1/32 all at 800G

Visual of the 3 Tenants on a virtualized GPU cloud service

Below is the simplified visual of the above example in this document:

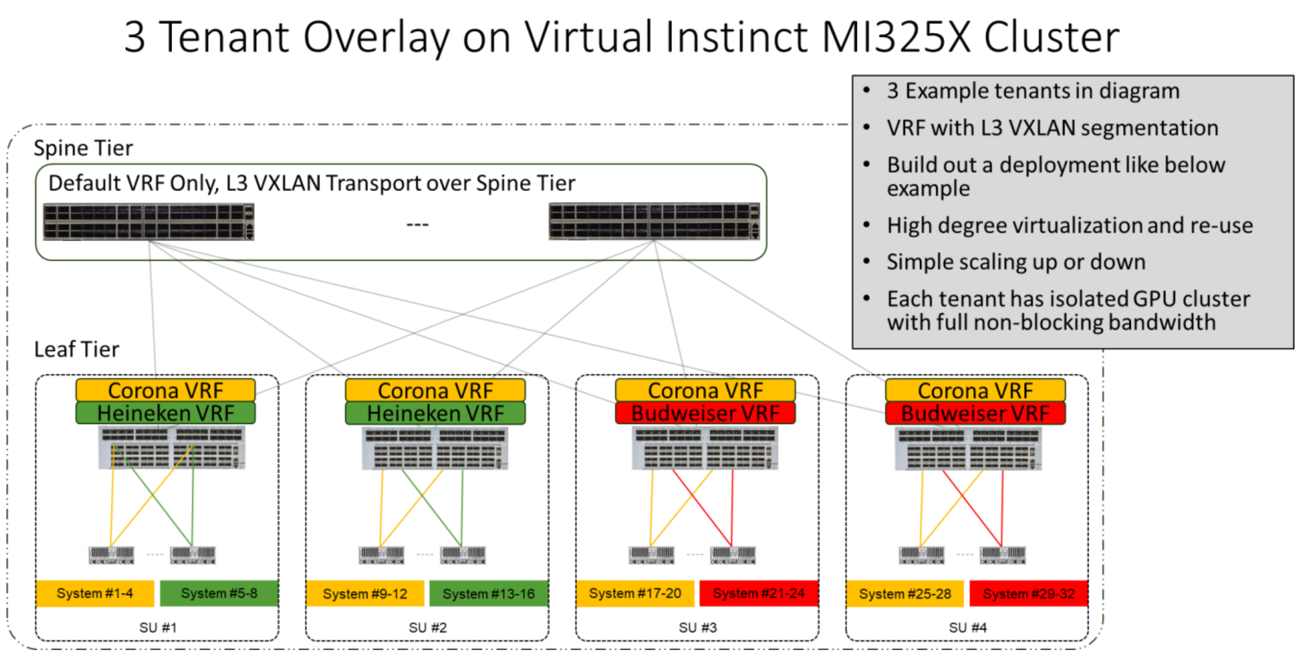


Figure 24 - 3 Tenant View on Cluster

Storage Network Validated Design

There are multiple alternatives in the storage portion of the cluster, with the goal of keeping up with the data to/from the accelerators as they function. One possible solution is for the customer to use a file system that employs the NVMe that is present on the AS-8126GS-TNMR, where the block diagram above shows the direct connection via the PLX switches.

While the SATA drives in slots 8 and 9 are ideal for the Ubuntu installation, the other 8 NVMe are optimized for parallel storage that is placed next to the accelerator and allows RDMA to the storage directly. Some solutions also allow for an adjacent set of flash local to each rack but to scale this we recommend a dedicated storage switch pair and Supermicro storage servers with a suitable high-performance stack like Weka, DDN, Vast, or many other excellent products available. A software solution to cluster this storage over the nodes that have tenant aware namespaces can be configured, and for extra security we could employ VRF technology again on this portion of the cluster. In Figure 23 below we show the connectivity from a north-south networking point of view.

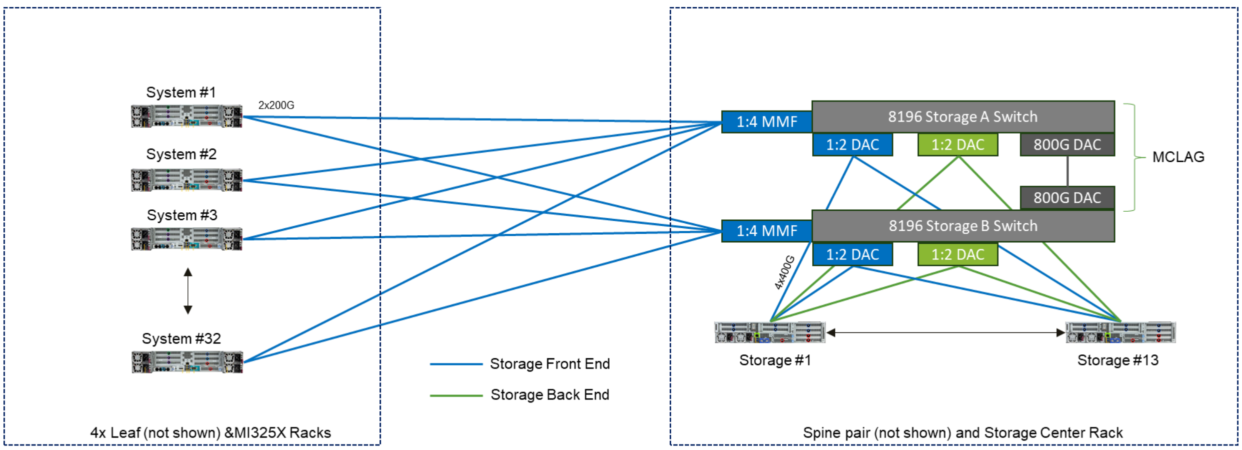


Figure 25 - Storage Fabric and Targets

As an aside, these solutions will have 2 more networks connected in the cluster. Firstly, we have an out of band management setup where 1GE Cat6 cable connects to the server BMC for routine setup and monitoring operation. It is on that connection we do the BIOS configurations mentioned below. Another network is the "front end" or again north-south but for the in-band access. This network connects as a redundant pair which in some cases is dedicated 10/25 GE that will have firewalled Internet and/or VPN reachability for apt-get/wget/rpm/tenant access/etc. operations, but in many cases, it is collapsed onto the storage network above. If that is the case, we would extend links to a set of border devices.

Importance of Automation of Fabric Configuration and Operations at any scale

If you look ahead to Appendix A with the device configuration for this relatively straightforward cluster of 4 leafs and 2 spines, you will see the scale and detail in the configuration for an Ethernet fabric for these designs. As we are looking at deployments that scale into 2/3/4 digits of switches and potentially tens of thousands of links – the probability of human error in many areas mandate the need for automation:

- Producing detailed architectural drawings of the cluster for proper equipment ordering and builds
- Automated output of detailed cabling maps with labeling at each end of the cables for these infrastructure elements from the RNICs to the leaf to the spine and perhaps even superspine to keep ahead of future operations
- Methods such as programmatic LLDP adjacency tests and/or IP Ping tests to validate proper cabling
- Methods for programmatic IP and BGP ASN assignment from a single source of truth database for the RNIC, switching, storage, front end, BMC networks in a coordinated manner goes beyond today's IPAM and DHCP capabilities

- Automated methods for server BIOS policies, RNIC policies, and distribution of policy to endpoints on when to signal interesting telemetry notifications and actions instead of polls and centralized processing of vast amounts of data
- Methods to coordinate QoS configurations of all elements in the infrastructure
- Methods to manage all device firmware lifecycle on all the constituent elements
- Automation of integrations of the topology with resource management tooling to provide efficient traffic management
- Performance optimization and acceptance testing along with diagnostic tooling from available telemetry
- Inventory Management

There are many more items on this list, and much activity in the industry for portions of the above. Many vendors including Supermicro are working towards the goal to bring all these tighter together and expect announcements forthcoming as we are all forced to tackle these tasks. Configuration of these elements from a centralized solution will relegate usage of individual device CLI's, API's, Web Interfaces directly.

An upcoming paper sharing a new Supermicro controller product in this space will show how this and other validated designs with multiple accelerator solutions can now be injected not only in the topology definition and deployment, but to include best practice optimizations based on that specific validated solution. Once the equipment arrives onsite the installation and cabling complete, the tooling can then validate the infrastructure and minimize the time to revenue. To make an even shorter-term deployment possible, the next section on having fully built and tested rack level infrastructure delivered is included.

How to Minimize Deployment Time – L12 Rack Pre-Built Solution from Supermicro

Total Rack Scale Solution

Supermicro's Rack Scale Solution Stack offers a fully integrated, end-to-end total solution that optimizes performance, efficiency, and scalability for AI, cloud, and enterprise workloads. As a total solution provider, Supermicro removes the complexity of multi-vendor integration by providing a pre-validated, high-density rack solution equipped with best-in-class servers, storage, networking, and power management, ensuring seamless deployment and faster time-to-value. By leveraging industry-leading energy efficiency, liquid and air-cooled designs, and global logistics capabilities, Supermicro delivers a cost-effective and future-proof solution designed to meet the most demanding IT requirements. Customers gain from direct manufacturer expertise, reduced operational overhead, and a single point of accountability, ensuring streamlined procurement, deployment, and support experience that maximize ROI.

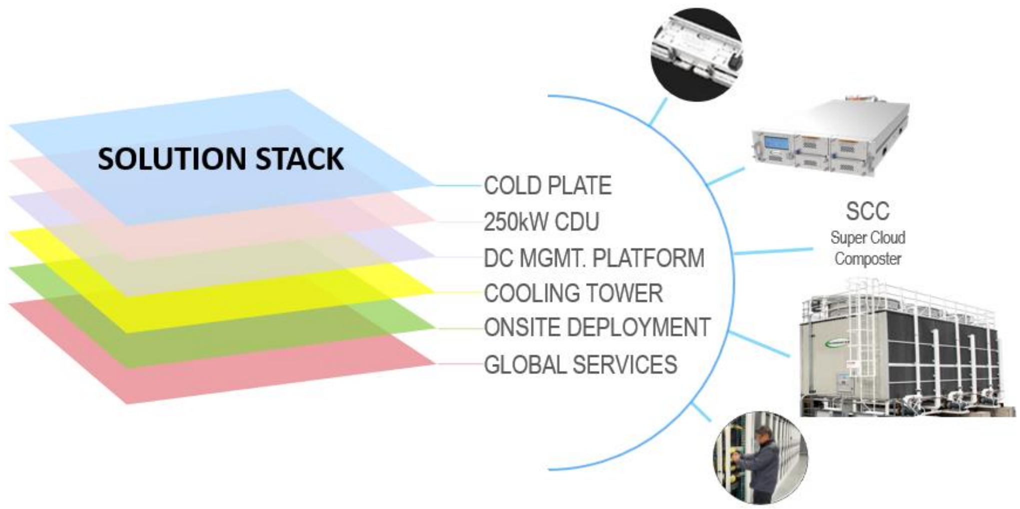
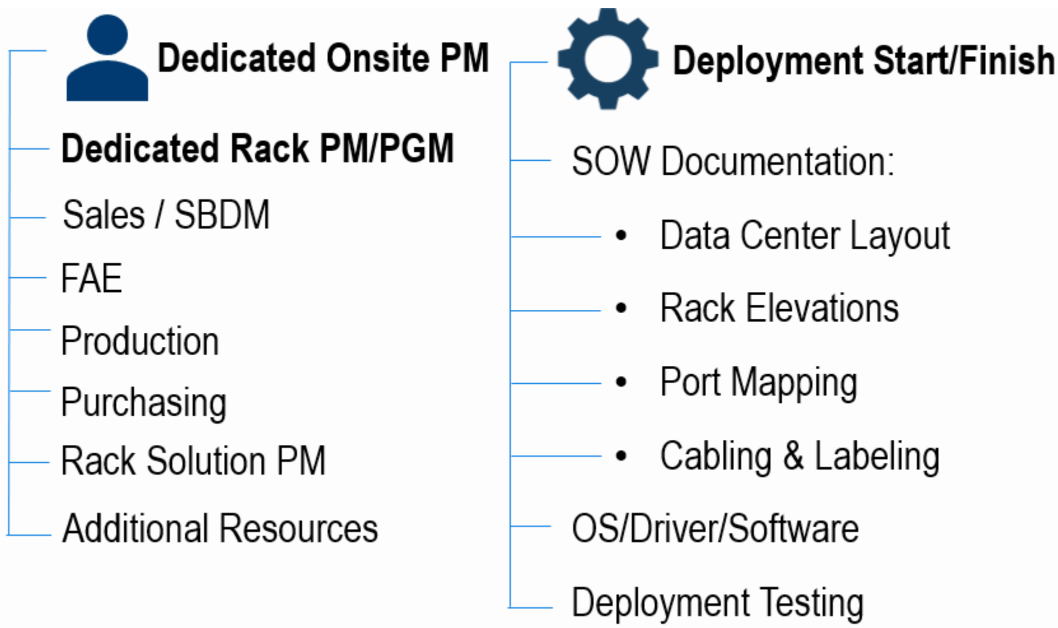


Figure 26 - Example Rack Solution Stack

Onsite Deployment

Supermicro’s Onsite Deployment Services ensure a seamless, end-to-end installation of AI and High-Performance Computing (HPC) clusters, accelerating time to production for enterprise applications, including LLMs, AI training, and mission-critical workloads. Our dedicated deployment team manages rack installation, cabling, labeling, network configuration, and testing to ensure optimal functionality and compliance with customer specifications. By leveraging Supermicro’s expertise and pre-validated deployment processes, customers reduce downtime, integration risks, and operational overhead, allowing IT teams to concentrate on performance tuning instead of infrastructure setup. With factory-trained professionals and global deployment capabilities, Supermicro provides a turnkey, fully optimized rack solution that is ready to run, helping organizations maximize efficiency, lower costs, and ensure long-term reliability.



Summary

This document provides an organized plan from start to finish that helps shorten the implementation time for clusters of various sizes (with a focus on 32 nodes to illustrate detailed concepts) while delivering business value in the shortest time. Additionally, further optimizations allow parts of the configuration and steps outlined here to be completed before the equipment even arrives onsite, especially if you are performing the installation yourself. Alternatively, Supermicro rack services can ensure the fastest time to value. Supermicro can share details of those possibilities in partnership with the organization executing the project.

Appendix A: Accelerator Fabric Detailed Leaf and Spine Configuration Steps

All Switches are running Enterprise Advanced 4.4.0 SONiC

Basic Starting Switch Preparation

To bootstrap the switches, you use a serial port (or USB to serial dongle) to access the console port and do the following steps to lay the base for applying the real configuration to the devices in subsequent operations. The console port and access defaults are below:

- 115,200 (8, N, 1)
- Default login: admin
- Default password: YourPaSsWoRd
- Disable ZTP using config ZTP disable -y
- Wait for 'System Ready'
- Enter the industry standard command line interface using sonic-cli
- To set the management IP (an IP on your existing management switch):
    - Leaf1# config terminal
    - Leaf1(config)# ip vrf mgmt
    - Leaf1(config)# username supermicro password testing123 role admin
    - Leaf1(config)# interface management 0
    - Leaf1(config-mgmt0)# ip address 10.1.1.101/24 gwaddr 10.1.1.1 <- your IP/subnet/dgw here
    - Leaf1(config-mgmt0)# exit
    - Leaf1(config)# exit
    - Leaf1# write memory
- Connect the management RJ45 1GE port into your network for remote access without the console requirement going forward

NOTE: All of the configuration examples below will be using an industry standard CLI that exists on the SSE-T8164S Supermicro Enterprise Advanced SONiC as this is the simplest descriptive method used by network administrators today. Supermicro highly recommends using some tooling to automate these configurations, where the built configurations are injected into the switch via methods like ZTP, Ansible, Puppet, gNMI and gRPC, and other tooling out there. These greatly reduce the probability of human error on data entry to these devices as we scale these solutions.

Leaf1

```
Leaf1# config terminal
Leaf1(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Leaf1(config)# roce enable
! Assign source VXLAN and Router ID addresses to loopback interfaces
Leaf1(config)# interface loopback 0
Leaf1(config-if-lo0)# description Router-id
Leaf1(config-if-lo0)# ip address 100.64.3.1/32
Leaf1(config-if-lo0)# exit
Leaf1(config)# interface loopback 1
Leaf1(config-if-lo1)# description Vtep
Leaf1(config-if-lo1)# ip address 100.64.5.1/32
Leaf1(config-if-lo1)# exit
! Setup the Adaptive Routing and switching globals
Leaf1(config)# ars profile default
Leaf1(config-ars-profile)# exit
Leaf1(config)# ars bind default
Leaf1(config)# ars port-profile default
Leaf1(config-ars-port-profile)# enable
Leaf1(config-ars-port-profile)# exit
Leaf1(config)# ars object default
Leaf1(config-ars-object)# exit
Leaf1(config)# route-map ars-map permit 10
Leaf1(config-route-map)# set ars-object default
Leaf1(config-route-map)# exit
Leaf1(config)# ip protocol any route-map ars-map
Leaf1(config)# route-map RM_SET_SRC permit 10
Leaf1(config-route-map)# set ars-object default
Leaf1(config-route-map)# exit
! Setup uplink interfaces to Spine1
! Note – do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Leaf1(config)# interface Eth 1/1
Leaf1(config-if-Eth1/1)# description Link to Spine1
Leaf1(config-if-Eth1/1)# speed 800000
Leaf1(config-if-Eth1/1)# unreliable-los auto
Leaf1(config-if-Eth1/1)# no shutdown
Leaf1(config-if-Eth1/1)# mtu 9100
Leaf1(config-if-Eth1/1)# ipv6 enable
Leaf1(config-if-Eth1/1)# ars bind default
Leaf1(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Leaf1(config)# interface Eth 1/16
Leaf1(config-if-Eth1/16)# description Link to Spine1
Leaf1(config-if-Eth1/16)# speed 800000
Leaf1(config-if-Eth1/16)# unreliable-los auto
Leaf1(config-if-Eth1/16)# no shutdown
Leaf1(config-if-Eth1/16)# mtu 9100
Leaf1(config-if-Eth1/16)# ipv6 enable
Leaf1(config-if-Eth1/16)# ars bind default
Leaf1(config-if-Eth1/16)# exit
! Setup uplink interfaces to Spine2
! Note – do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Leaf1(config)# interface Eth 1/17
Leaf1(config-if-Eth1/17)# description Link to Spine2
Leaf1(config-if-Eth1/17)# speed 800000
Leaf1(config-if-Eth1/17)# unreliable-los auto
Leaf1(config-if-Eth1/17)# no shutdown
Leaf1(config-if-Eth1/17)# mtu 9100
Leaf1(config-if-Eth1/17)# ipv6 enable
Leaf1(config-if-Eth1/17)# ars bind default
Leaf1(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Leaf1(config)# interface Eth 1/32
Leaf1(config-if-Eth1/32)# description Link to Spine2
Leaf1(config-if-Eth1/32)# speed 800000
Leaf1(config-if-Eth1/32)# unreliable-los auto
Leaf1(config-if-Eth1/32)# no shutdown
Leaf1(config-if-Eth1/32)# mtu 9100
Leaf1(config-if-Eth1/32)# ipv6 enable
Leaf1(config-if-Eth1/32)# ars bind default
Leaf1(config-if-Eth1/32)# exit
! Create tenant VRFs for a multi-tenant environment (NOTE:  Can just setup a single tenant/VRF only also)
```

```
Leaf1(config)# ip vrf Corona
Leaf1(config)# ip vrf Heineken
! No nodes for Budweiser here on leaf1, but for future if needed
Leaf1(config)# ip vrf Budweiser
! Assign /31 IP's to Corona's host interfaces Eth 1/33-1/64 (showing just first and last)
Leaf1(config)# interface Eth 1/33
Leaf1(config-if-Eth1/33)# speed 400000
Leaf1(config-if-Eth1/33)# mtu 9100
Leaf1(config-if-Eth1/33)# fec RS
Leaf1(config-if-Eth1/33)# standalone-link-training
Leaf1(config-if-Eth1/33)# unreliable-los auto
Leaf1(config-if-Eth1/33)# no shutdown
Leaf1(config-if-Eth1/33)# ip address 100.65.0.0/31
Leaf1(config-if-Eth1/33)# description Link to Corona Node 1 RNIC Slot 1 with IP 100.65.0.1/31
Leaf1(config-if-Eth1/33)# ip vrf forwarding Corona
Leaf1(config-if-Eth1/33)# exit
! Range of 30 links here 1/34-1/63
Leaf1(config)# interface Eth 1/64
Leaf1(config-if-Eth1/64)# speed 400000
Leaf1(config-if-Eth1/64)# mtu 9100
Leaf1(config-if-Eth1/64)# fec RS
Leaf1(config-if-Eth1/64)# standalone-link-training
Leaf1(config-if-Eth1/64)# unreliable-los auto
Leaf1(config-if-Eth1/64)# no shutdown
Leaf1(config-if-Eth1/64)# ip address 100.65.0.62/31
Leaf1(config-if-Eth1/64)# description Link to Corona Node 4 RNIC Slot 8 with IP 100.65.0.63/31
Leaf1(config-if-Eth1/64)# ip vrf forwarding Corona
Leaf1(config-if-Eth1/64)# exit
! Assign /31 IP's to Heineken's host interfaces Eth 1/65-1/96 (showing just first and last)
Leaf1(config)# interface Eth 1/65
Leaf1(config-if-Eth1/65)# speed 400000
Leaf1(config-if-Eth1/65)# mtu 9100
Leaf1(config-if-Eth1/65)# fec RS
Leaf1(config-if-Eth1/65)# standalone-link-training
Leaf1(config-if-Eth1/65)# unreliable-los auto
Leaf1(config-if-Eth1/65)# no shutdown
Leaf1(config-if-Eth1/65)# ip address 100.66.0.0/31
Leaf1(config-if-Eth1/65)# description Link to Heineken Node 1 RNIC Slot 1 with IP 100.66.0.1/31
Leaf1(config-if-Eth1/65)# ip vrf forwarding Heineken
Leaf1(config-if-Eth1/65)# exit
! Range of 30 links here 1/66-1/95
Leaf1(config)# interface Eth 1/96
Leaf1(config-if-Eth1/96)# speed 400000
Leaf1(config-if-Eth1/96)# mtu 9100
Leaf1(config-if-Eth1/96)# fec RS
Leaf1(config-if-Eth1/96)# standalone-link-training
Leaf1(config-if-Eth1/96)# unreliable-los auto
Leaf1(config-if-Eth1/96)# no shutdown
Leaf1(config-if-Eth1/96)# ip address 100.66.0.62/31
Leaf1(config-if-Eth1/96)# description Link to Heineken Node 4 RNIC Slot 8 with IP 100.66.0.63/31
Leaf1(config-if-Eth1/96)# ip vrf forwarding Heineken
Leaf1(config-if-Eth1/96)# exit
! Configure L3 VNI VLANs
Leaf1(config)# interface Vlan 61
Leaf1(config-if-Vlan61)# ip vrf forwarding Corona
Leaf1(config-if-Vlan61)# exit
Leaf1(config)# interface Vlan 62
Leaf1(config-if-Vlan62)# ip vrf forwarding Heineken
Leaf1(config-if-Vlan62)# exit
Leaf1(config)# interface Vlan 63
Leaf1(config-if-Vlan63)# ip vrf forwarding Budweiser
Leaf1(config-if-Vlan63)# exit
! Map VNIs to VLANs and L3 VNIs to VRFs
Leaf1(config)# interface vxlan vtep-1
Leaf1(config-if-vxlan-vtep-1)# source-ip 100.64.5.1
Leaf1(config-if-vxlan-vtep-1)# map vni 610 vlan 61
Leaf1(config-if-vxlan-vtep-1)# map vni 620 vlan 62
Leaf1(config-if-vxlan-vtep-1)# map vni 630 vlan 63
Leaf1(config-if-vxlan-vtep-1)# map vni 610 vrf Corona
Leaf1(config-if-vxlan-vtep-1)# map vni 620 vrf Heineken
Leaf1(config-if-vxlan-vtep-1)# map vni 630 vrf Budweiser
Leaf1(config-if-vxlan-vtep-1)# qos-mode uniform
Leaf1(config-if-vxlan-vtep-1)# exit
```

```
! setup underlay and overlay BGP
Leaf1(config)# router bgp 65101
Leaf1(config-router-bgp)# router-id 100.64.3.1
Leaf1(config-router-bgp)# address-family ipv4 unicast
Leaf1(config-router-bgp-af)# redistribute connected
Leaf1(config-router-bgp-af)# maximum-paths 64
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# address-family l2vpn evpn
Leaf1(config-router-bgp-af)# advertise-all-vni
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# peer-group SPINES
Leaf1(config-router-bgp-pg)# remote-as external
Leaf1(config-router-bgp-pg)# timers 3 9
Leaf1(config-router-bgp-pg)# advertisement-interval 5
Leaf1(config-router-bgp-pg)# bfd
Leaf1(config-router-bgp-pg)# capability extended-nexthop
Leaf1(config-router-bgp-pg)# address-family ipv4 unicast
Leaf1(config-router-bgp-pg-af)# activate
Leaf1(config-router-bgp-pg-af)# exit
Leaf1(config-router-bgp-pg)# address-family l2vpn evpn
Leaf1(config-router-bgp-pg-af)# activate
Leaf1(config-router-bgp-pg-af)# exit
Leaf1(config-router-bgp-pg)# exit
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 neighbors in this block (showing just first and last)
Leaf1(config-router-bgp)# neighbor interface Eth 1/1
Leaf1(config-router-bgp-neighbor)# description Link to Spine1
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/2-1/15
Leaf1(config-router-bgp)# neighbor interface Eth 1/16
Leaf1(config-router-bgp-neighbor)# description Link to Spine1
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Leaf1(config-router-bgp)# neighbor interface Eth 1/17
Leaf1(config-router-bgp-neighbor)# description Link to Spine2
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/18-1/31
Leaf1(config-router-bgp)# neighbor interface Eth 1/32
Leaf1(config-router-bgp-neighbor)# description Link to Spine2
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
Leaf1(config-router-bgp)# exit
Leaf1(config)# router bgp 65101 vrf Corona
Leaf1(config-router-bgp)# address-family ipv4 unicast
Leaf1(config-router-bgp-af)# redistribute connected
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# address-family l2vpn evpn
Leaf1(config-router-bgp-af)# advertise ipv4 unicast
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# exit
Leaf1(config)# router bgp 65101 vrf Heineken
Leaf1(config-router-bgp)# address-family ipv4 unicast
Leaf1(config-router-bgp-af)# redistribute connected
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# address-family l2vpn evpn
Leaf1(config-router-bgp-af)# advertise ipv4 unicast
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# exit
Leaf1(config)# router bgp 65101 vrf Budweiser
Leaf1(config-router-bgp)# address-family ipv4 unicast
Leaf1(config-router-bgp-af)# redistribute connected
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# address-family l2vpn evpn
Leaf1(config-router-bgp-af)# advertise ipv4 unicast
Leaf1(config-router-bgp-af)# exit
Leaf1(config-router-bgp)# exit
Leaf1(config)# exit
Leaf1# write memory
```

Leaf2

```
Leaf1# config terminal
Leaf2(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Leaf2(config)# roce enable
! Assign source VXLAN and Router ID addresses to loopback interfaces
Leaf2(config)# interface loopback 0
Leaf2(config-if-lo0)# description Router-id
Leaf2(config-if-lo0)# ip address 100.64.3.2/32
Leaf2(config-if-lo0)# exit
Leaf2(config)# interface loopback 1
Leaf2(config-if-lo1)# description Vtep
Leaf2(config-if-lo1)# ip address 100.64.5.2/32
Leaf2(config-if-lo1)# exit
! Setup the Adaptive Routing and switching globals
Leaf2(config)# ars profile default
Leaf2(config-ars-profile)# exit
Leaf2(config)# ars bind default
Leaf2(config)# ars port-profile default
Leaf2(config-ars-port-profile)# enable
Leaf2(config-ars-port-profile)# exit
Leaf2(config)# ars object default
Leaf2(config-ars-object)# exit
Leaf2(config)# route-map ars-map permit 10
Leaf2(config-route-map)# set ars-object default
Leaf2(config-route-map)# exit
Leaf2(config)# ip protocol any route-map ars-map
Leaf2(config)# route-map RM_SET_SRC permit 10
Leaf2(config-route-map)# set ars-object default
Leaf2(config-route-map)# exit
! Setup uplink interfaces to Spine1
! Note – do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Leaf2(config)# interface Eth 1/1
Leaf2(config-if-Eth1/1)# description Link to Spine1
Leaf2(config-if-Eth1/1)# speed 800000
Leaf2(config-if-Eth1/1)# unreliable-los auto
Leaf2(config-if-Eth1/1)# no shutdown
Leaf2(config-if-Eth1/1)# mtu 9100
Leaf2(config-if-Eth1/1)# ipv6 enable
Leaf2(config-if-Eth1/1)# ars bind default
Leaf2(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Leaf2(config)# interface Eth 1/16
Leaf2(config-if-Eth1/16)# description Link to Spine1
Leaf2(config-if-Eth1/16)# speed 800000
Leaf2(config-if-Eth1/16)# unreliable-los auto
Leaf2(config-if-Eth1/16)# no shutdown
Leaf2(config-if-Eth1/16)# mtu 9100
Leaf2(config-if-Eth1/16)# ipv6 enable
Leaf2(config-if-Eth1/16)# ars bind default
Leaf2(config-if-Eth1/16)# exit
! Setup uplink interfaces to Spine2
! Note – do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Leaf2(config)# interface Eth 1/17
Leaf2(config-if-Eth1/17)# description Link to Spine2
Leaf2(config-if-Eth1/17)# speed 800000
Leaf2(config-if-Eth1/17)# unreliable-los auto
Leaf2(config-if-Eth1/17)# no shutdown
Leaf2(config-if-Eth1/17)# mtu 9100
Leaf2(config-if-Eth1/17)# ipv6 enable
Leaf2(config-if-Eth1/17)# ars bind default
Leaf2(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Leaf2(config)# interface Eth 1/32
Leaf2(config-if-Eth1/32)# description Link to Spine2
Leaf2(config-if-Eth1/32)# speed 800000
Leaf2(config-if-Eth1/32)# unreliable-los auto
Leaf2(config-if-Eth1/32)# no shutdown
Leaf2(config-if-Eth1/32)# mtu 9100
Leaf2(config-if-Eth1/32)# ipv6 enable
Leaf2(config-if-Eth1/32)# ars bind default
Leaf2(config-if-Eth1/32)# exit
! Create tenant VRFs for a multi-tenant environment (NOTE:  Can just setup a single tenant/VRF only also)
```

```
Leaf2(config)# ip vrf Corona
Leaf2(config)# ip vrf Heineken
! No nodes for Budweiser here on leaf2, but for future if needed
Leaf2(config)# ip vrf Budweiser
! Assign /31 IP's to Corona's host interfaces Eth 1/33-1/64 (showing just first and last)
Leaf2(config)# interface Eth 1/33
Leaf2(config-if-Eth1/33)# speed 400000
Leaf2(config-if-Eth1/33)# mtu 9100
Leaf2(config-if-Eth1/33)# fec RS
Leaf2(config-if-Eth1/33)# standalone-link-training
Leaf2(config-if-Eth1/33)# unreliable-los auto
Leaf2(config-if-Eth1/33)# no shutdown
Leaf2(config-if-Eth1/33)# ip address 100.65.0.64/31
Leaf2(config-if-Eth1/33)# description Link to Corona Node 5 RNIC Slot 1 with IP 100.65.0.65/31
Leaf2(config-if-Eth1/33)# ip vrf forwarding Corona
Leaf2(config-if-Eth1/33)# exit
! Range of 30 links here 1/34-1/63
Leaf2(config)# interface Eth 1/64
Leaf2(config-if-Eth1/64)# speed 400000
Leaf2(config-if-Eth1/64)# mtu 9100
Leaf2(config-if-Eth1/64)# fec RS
Leaf2(config-if-Eth1/64)# standalone-link-training
Leaf2(config-if-Eth1/64)# unreliable-los auto
Leaf2(config-if-Eth1/64)# no shutdown
Leaf2(config-if-Eth1/64)# ip address 100.65.0.126/31
Leaf2(config-if-Eth1/64)# description Link to Corona Node 8 RNIC Slot 8 with IP 100.65.0.127/31
Leaf2(config-if-Eth1/64)# ip vrf forwarding Corona
Leaf2(config-if-Eth1/64)# exit
! Assign /31 IP's to Heineken's host interfaces Eth 1/65-1/96 (showing just first and last)
Leaf2(config)# interface Eth 1/65
Leaf2(config-if-Eth1/65)# speed 400000
Leaf2(config-if-Eth1/65)# mtu 9100
Leaf2(config-if-Eth1/65)# fec RS
Leaf2(config-if-Eth1/65)# standalone-link-training
Leaf2(config-if-Eth1/65)# unreliable-los auto
Leaf2(config-if-Eth1/65)# no shutdown
Leaf2(config-if-Eth1/65)# ip address 100.66.0.64/31
Leaf2(config-if-Eth1/65)# description Link to Heineken Node 5 RNIC Slot 1 with IP 100.66.0.65/31
Leaf2(config-if-Eth1/65)# ip vrf forwarding Heineken
Leaf2(config-if-Eth1/65)# exit
! Range of 30 links here 1/66-1/95
Leaf2(config)# interface Eth 1/96
Leaf2(config-if-Eth1/96)# speed 400000
Leaf2(config-if-Eth1/96)# mtu 9100
Leaf2(config-if-Eth1/96)# fec RS
Leaf2(config-if-Eth1/96)# standalone-link-training
Leaf2(config-if-Eth1/96)# unreliable-los auto
Leaf2(config-if-Eth1/96)# no shutdown
Leaf2(config-if-Eth1/96)# ip address 100.66.0.126/31
Leaf2(config-if-Eth1/96)# description Link to Heineken Node 8 RNIC Slot 8 with IP 100.66.0.127/31
Leaf2(config-if-Eth1/96)# ip vrf forwarding Heineken
Leaf2(config-if-Eth1/96)# exit
! Configure L3 VNI VLANs
Leaf2(config)# interface Vlan 61
Leaf2(config-if-Vlan61)# ip vrf forwarding Corona
Leaf2(config-if-Vlan61)# exit
Leaf2(config)# interface Vlan 62
Leaf2(config-if-Vlan62)# ip vrf forwarding Heineken
Leaf2(config-if-Vlan62)# exit
Leaf2(config)# interface Vlan 63
Leaf2(config-if-Vlan63)# ip vrf forwarding Budweiser
Leaf2(config-if-Vlan63)# exit
! Map VNIs to VLANs and L3 VNIs to VRFs
Leaf2(config)# interface vxlan vtep-2
Leaf2(config-if-vxlan-vtep-2)# source-ip 100.64.5.2
Leaf2(config-if-vxlan-vtep-2)# map vni 610 vlan 61
Leaf2(config-if-vxlan-vtep-2)# map vni 620 vlan 62
Leaf2(config-if-vxlan-vtep-2)# map vni 630 vlan 63
Leaf2(config-if-vxlan-vtep-2)# map vni 610 vrf Corona
Leaf2(config-if-vxlan-vtep-2)# map vni 620 vrf Heineken
Leaf2(config-if-vxlan-vtep-2)# map vni 630 vrf Budweiser
Leaf2(config-if-vxlan-vtep-2)# qos-mode uniform
Leaf2(config-if-vxlan-vtep-2)# exit
```

```
! setup underlay and overlay BGP
Leaf2(config)# router bgp 65102
Leaf2(config-router-bgp)# router-id 100.64.3.2
Leaf2(config-router-bgp)# address-family ipv4 unicast
Leaf2(config-router-bgp-af)# redistribute connected
Leaf2(config-router-bgp-af)# maximum-paths 64
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# address-family l2vpn evpn
Leaf2(config-router-bgp-af)# advertise-all-vni
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# peer-group SPINES
Leaf2(config-router-bgp-pg)# remote-as external
Leaf2(config-router-bgp-pg)# timers 3 9
Leaf2(config-router-bgp-pg)# advertisement-interval 5
Leaf2(config-router-bgp-pg)# bfd
Leaf2(config-router-bgp-pg)# capability extended-nexthop
Leaf2(config-router-bgp-pg)# address-family ipv4 unicast
Leaf2(config-router-bgp-pg-af)# activate
Leaf2(config-router-bgp-pg-af)# exit
Leaf2(config-router-bgp-pg)# address-family l2vpn evpn
Leaf2(config-router-bgp-pg-af)# activate
Leaf2(config-router-bgp-pg-af)# exit
Leaf2(config-router-bgp-pg)# exit
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 neighbors in this block (showing just first and last)
Leaf2(config-router-bgp)# neighbor interface Eth 1/1
Leaf2(config-router-bgp-neighbor)# description Link to Spine1
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/2-1/15
Leaf2(config-router-bgp)# neighbor interface Eth 1/16
Leaf2(config-router-bgp-neighbor)# description Link to Spine1
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Leaf2(config-router-bgp)# neighbor interface Eth 1/17
Leaf2(config-router-bgp-neighbor)# description Link to Spine2
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/18-1/31
Leaf2(config-router-bgp)# neighbor interface Eth 1/32
Leaf2(config-router-bgp-neighbor)# description Link to Spine2
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
Leaf2(config-router-bgp)# exit
Leaf2(config)# router bgp 65102 vrf Corona
Leaf2(config-router-bgp)# address-family ipv4 unicast
Leaf2(config-router-bgp-af)# redistribute connected
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# address-family l2vpn evpn
Leaf2(config-router-bgp-af)# advertise ipv4 unicast
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# exit
Leaf2(config)# router bgp 65102 vrf Heineken
Leaf2(config-router-bgp)# address-family ipv4 unicast
Leaf2(config-router-bgp-af)# redistribute connected
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# address-family l2vpn evpn
Leaf2(config-router-bgp-af)# advertise ipv4 unicast
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# exit
Leaf2(config)# router bgp 65102 vrf Budweiser
Leaf2(config-router-bgp)# address-family ipv4 unicast
Leaf2(config-router-bgp-af)# redistribute connected
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# address-family l2vpn evpn
Leaf2(config-router-bgp-af)# advertise ipv4 unicast
Leaf2(config-router-bgp-af)# exit
Leaf2(config-router-bgp)# exit
Leaf2(config)# exit
Leaf2# write memory
```

Leaf3

```
Leaf3# config terminal
Leaf3(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Leaf3(config)# roce enable
! Assign source VXLAN and Router ID addresses to loopback interfaces
Leaf3(config)# interface loopback 0
Leaf3(config-if-lo0)# description Router-id
Leaf3(config-if-lo0)# ip address 100.64.3.3/32
Leaf3(config-if-lo0)# exit
Leaf3(config)# interface loopback 1
Leaf3(config-if-lo1)# description Vtep
Leaf3(config-if-lo1)# ip address 100.64.5.3/32
Leaf3(config-if-lo1)# exit
! Setup the Adaptive Routing and switching globals
Leaf3(config)# ars profile default
Leaf3(config-ars-profile)# exit
Leaf3(config)# ars bind default
Leaf3(config)# ars port-profile default
Leaf3(config-ars-port-profile)# enable
Leaf3(config-ars-port-profile)# exit
Leaf3(config)# ars object default
Leaf3(config-ars-object)# exit
Leaf3(config)# route-map ars-map permit 10
Leaf3(config-route-map)# set ars-object default
Leaf3(config-route-map)# exit
Leaf3(config)# ip protocol any route-map ars-map
Leaf3(config)# route-map RM_SET_SRC permit 10
Leaf3(config-route-map)# set ars-object default
Leaf3(config-route-map)# exit
! Setup uplink interfaces to Spine1
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Leaf3(config)# interface Eth 1/1
Leaf3(config-if-Eth1/1)# description Link to Spine1
Leaf3(config-if-Eth1/1)# speed 800000
Leaf3(config-if-Eth1/1)# unreliable-los auto
Leaf3(config-if-Eth1/1)# no shutdown
Leaf3(config-if-Eth1/1)# mtu 9100
Leaf3(config-if-Eth1/1)# ipv6 enable
Leaf3(config-if-Eth1/1)# ars bind default
Leaf3(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Leaf3(config)# interface Eth 1/16
Leaf3(config-if-Eth1/16)# description Link to Spine1
Leaf3(config-if-Eth1/16)# speed 800000
Leaf3(config-if-Eth1/16)# unreliable-los auto
Leaf3(config-if-Eth1/16)# no shutdown
Leaf3(config-if-Eth1/16)# mtu 9100
Leaf3(config-if-Eth1/16)# ipv6 enable
Leaf3(config-if-Eth1/16)# ars bind default
Leaf3(config-if-Eth1/16)# exit
! Setup uplink interfaces to Spine2
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Leaf3(config)# interface Eth 1/17
Leaf3(config-if-Eth1/17)# description Link to Spine2
Leaf3(config-if-Eth1/17)# speed 800000
Leaf3(config-if-Eth1/17)# unreliable-los auto
Leaf3(config-if-Eth1/17)# no shutdown
Leaf3(config-if-Eth1/17)# mtu 9100
Leaf3(config-if-Eth1/17)# ipv6 enable
Leaf3(config-if-Eth1/17)# ars bind default
Leaf3(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Leaf3(config)# interface Eth 1/32
Leaf3(config-if-Eth1/32)# description Link to Spine2
Leaf3(config-if-Eth1/32)# speed 800000
Leaf3(config-if-Eth1/32)# unreliable-los auto
Leaf3(config-if-Eth1/32)# no shutdown
Leaf3(config-if-Eth1/32)# mtu 9100
Leaf3(config-if-Eth1/32)# ipv6 enable
Leaf3(config-if-Eth1/32)# ars bind default
Leaf3(config-if-Eth1/32)# exit
! Create tenant VRFs for a multi-tenant environment (NOTE:  Can just setup a single tenant/VRF only also)
```

```
Leaf3(config)# ip vrf Corona
! No nodes for Heineken here, but for future if needed
Leaf3(config)# ip vrf Heineken
Leaf3(config)# ip vrf Budweiser
! Assign /31 IP's to Corona's host interfaces Eth 1/33-1/64 (showing just first and last)
Leaf3(config)# interface Eth 1/33
Leaf3(config-if-Eth1/33)# speed 400000
Leaf3(config-if-Eth1/33)# mtu 9100
Leaf3(config-if-Eth1/33)# fec RS
Leaf3(config-if-Eth1/33)# standalone-link-training
Leaf3(config-if-Eth1/33)# unreliable-los auto
Leaf3(config-if-Eth1/33)# no shutdown
Leaf3(config-if-Eth1/33)# ip address 100.65.0.128/31
Leaf3(config-if-Eth1/33)# description Link to Corona Node 9 RNIC Slot 1 with IP 100.65.0.129/31
Leaf3(config-if-Eth1/33)# ip vrf forwarding Corona
Leaf3(config-if-Eth1/33)# exit
! Range of 30 links here 1/34-1/63
Leaf3(config)# interface Eth 1/64
Leaf3(config-if-Eth1/64)# speed 400000
Leaf3(config-if-Eth1/64)# mtu 9100
Leaf3(config-if-Eth1/64)# fec RS
Leaf3(config-if-Eth1/64)# standalone-link-training
Leaf3(config-if-Eth1/64)# unreliable-los auto
Leaf3(config-if-Eth1/64)# no shutdown
Leaf3(config-if-Eth1/64)# ip address 100.65.0.190/31
Leaf3(config-if-Eth1/64)# description Link to Corona Node 12 RNIC Slot 8 with IP 100.65.0.191/31
Leaf3(config-if-Eth1/64)# ip vrf forwarding Corona
Leaf3(config-if-Eth1/64)# exit
! Assign /31 IP's to Budweiser host interfaces Eth 1/65-1/96 (showing just first and last)
Leaf3(config)# interface Eth 1/65
Leaf3(config-if-Eth1/65)# speed 400000
Leaf3(config-if-Eth1/65)# mtu 9100
Leaf3(config-if-Eth1/65)# fec RS
Leaf3(config-if-Eth1/65)# standalone-link-training
Leaf3(config-if-Eth1/65)# unreliable-los auto
Leaf3(config-if-Eth1/65)# no shutdown
Leaf3(config-if-Eth1/65)# ip address 100.67.0.0/31
Leaf3(config-if-Eth1/65)# description Link to Budweiser Node 1 RNIC Slot 1 with IP 100.67.0.1/31
Leaf3(config-if-Eth1/65)# ip vrf forwarding Budweiser
Leaf3(config-if-Eth1/65)# exit
! Range of 30 links here 1/66-1/95
Leaf3(config)# interface Eth 1/96
Leaf3(config-if-Eth1/96)# speed 400000
Leaf3(config-if-Eth1/96)# mtu 9100
Leaf3(config-if-Eth1/96)# fec RS
Leaf3(config-if-Eth1/96)# standalone-link-training
Leaf3(config-if-Eth1/96)# unreliable-los auto
Leaf3(config-if-Eth1/96)# no shutdown
Leaf3(config-if-Eth1/96)# ip address 100.67.0.62/31
Leaf3(config-if-Eth1/96)# description Link to Budweiser Node 4 RNIC Slot 8 with IP 100.67.0.63/31
Leaf3(config-if-Eth1/96)# ip vrf forwarding Budweiser
Leaf3(config-if-Eth1/96)# exit
! Configure L3 VNI VLANs
Leaf3(config)# interface Vlan 61
Leaf3(config-if-Vlan61)# ip vrf forwarding Corona
Leaf3(config-if-Vlan61)# exit
Leaf3(config)# interface Vlan 62
Leaf3(config-if-Vlan62)# ip vrf forwarding Heineken
Leaf3(config-if-Vlan62)# exit
Leaf3(config)# interface Vlan 63
Leaf3(config-if-Vlan63)# ip vrf forwarding Budweiser
Leaf3(config-if-Vlan63)# exit
! Map VNIs to VLANs and L3 VNIs to VRFs
Leaf3(config)# interface vxlan vtep-3
Leaf3(config-if-vxlan-vtep-3)# source-ip 100.64.5.3
Leaf3(config-if-vxlan-vtep-3)# map vni 610 vlan 61
Leaf3(config-if-vxlan-vtep-3)# map vni 620 vlan 62
Leaf3(config-if-vxlan-vtep-3)# map vni 630 vlan 63
Leaf3(config-if-vxlan-vtep-3)# map vni 610 vrf Corona
Leaf3(config-if-vxlan-vtep-3)# map vni 620 vrf Heineken
Leaf3(config-if-vxlan-vtep-3)# map vni 630 vrf Budweiser
Leaf3(config-if-vxlan-vtep-3)# qos-mode uniform
Leaf3(config-if-vxlan-vtep-3)# exit
```

```
! setup underlay and overlay BGP
Leaf3(config)# router bgp 65103
Leaf3(config-router-bgp)# router-id 100.64.3.3
Leaf3(config-router-bgp)# address-family ipv4 unicast
Leaf3(config-router-bgp-af)# redistribute connected
Leaf3(config-router-bgp-af)# maximum-paths 64
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# address-family l2vpn evpn
Leaf3(config-router-bgp-af)# advertise-all-vni
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# peer-group SPINES
Leaf3(config-router-bgp-pg)# remote-as external
Leaf3(config-router-bgp-pg)# timers 3 9
Leaf3(config-router-bgp-pg)# advertisement-interval 5
Leaf3(config-router-bgp-pg)# bfd
Leaf3(config-router-bgp-pg)# capability extended-nexthop
Leaf3(config-router-bgp-pg)# address-family ipv4 unicast
Leaf3(config-router-bgp-pg-af)# activate
Leaf3(config-router-bgp-pg-af)# exit
Leaf3(config-router-bgp-pg)# address-family l2vpn evpn
Leaf3(config-router-bgp-pg-af)# activate
Leaf3(config-router-bgp-pg-af)# exit
Leaf3(config-router-bgp-pg)# exit
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 neighbors in this block (showing just first and last)
Leaf3(config-router-bgp)# neighbor interface Eth 1/1
Leaf3(config-router-bgp-neighbor)# description Link to Spine1
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
Leaf3(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/2-1/15
Leaf3(config-router-bgp)# neighbor interface Eth 1/16
Leaf3(config-router-bgp-neighbor)# description Link to Spine1
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
Leaf3(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Leaf3(config-router-bgp)# neighbor interface Eth 1/17
Leaf3(config-router-bgp-neighbor)# description Link to Spine2
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
Leaf3(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/18-1/31
Leaf3(config-router-bgp)# neighbor interface Eth 1/32
Leaf3(config-router-bgp-neighbor)# description Link to Spine2
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
Leaf3(config-router-bgp-neighbor)# exit
Leaf3(config-router-bgp)# exit
Leaf3(config)# router bgp 65103 vrf Corona
Leaf3(config-router-bgp)# address-family ipv4 unicast
Leaf3(config-router-bgp-af)# redistribute connected
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# address-family l2vpn evpn
Leaf3(config-router-bgp-af)# advertise ipv4 unicast
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# exit
Leaf3(config)# router bgp 65103 vrf Heineken
Leaf3(config-router-bgp)# address-family ipv4 unicast
Leaf3(config-router-bgp-af)# redistribute connected
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# address-family l2vpn evpn
Leaf3(config-router-bgp-af)# advertise ipv4 unicast
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# exit
Leaf3(config)# router bgp 65103 vrf Budweiser
Leaf3(config-router-bgp)# address-family ipv4 unicast
Leaf3(config-router-bgp-af)# redistribute connected
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# address-family l2vpn evpn
Leaf3(config-router-bgp-af)# advertise ipv4 unicast
Leaf3(config-router-bgp-af)# exit
Leaf3(config-router-bgp)# exit
Leaf3(config)# exit
Leaf3# write memory
```

Leaf4

```
Leaf4# config terminal
Leaf4(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Leaf4(config)# roce enable
! Assign source VXLAN and Router ID addresses to loopback interfaces
Leaf4(config)# interface loopback 0
Leaf4(config-if-lo0)# description Router-id
Leaf4(config-if-lo0)# ip address 100.64.3.4/32
Leaf4(config-if-lo0)# exit
Leaf4(config)# interface loopback 1
Leaf4(config-if-lo1)# description Vtep
Leaf4(config-if-lo1)# ip address 100.64.5.4/32
Leaf4(config-if-lo1)# exit
! Setup the Adaptive Routing and switching globals
Leaf4(config)# ars profile default
Leaf4(config-ars-profile)# exit
Leaf4(config)# ars bind default
Leaf4(config)# ars port-profile default
Leaf4(config-ars-port-profile)# enable
Leaf4(config-ars-port-profile)# exit
Leaf4(config)# ars object default
Leaf4(config-ars-object)# exit
Leaf4(config)# route-map ars-map permit 10
Leaf4(config-route-map)# set ars-object default
Leaf4(config-route-map)# exit
Leaf4(config)# ip protocol any route-map ars-map
Leaf4(config)# route-map RM_SET_SRC permit 10
Leaf4(config-route-map)# set ars-object default
Leaf4(config-route-map)# exit
! Setup uplink interfaces to Spine1
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Leaf4(config)# interface Eth 1/1
Leaf4(config-if-Eth1/1)# description Link to Spine1
Leaf4(config-if-Eth1/1)# speed 800000
Leaf4(config-if-Eth1/1)# unreliable-los auto
Leaf4(config-if-Eth1/1)# no shutdown
Leaf4(config-if-Eth1/1)# mtu 9100
Leaf4(config-if-Eth1/1)# ipv6 enable
Leaf4(config-if-Eth1/1)# ars bind default
Leaf4(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Leaf4(config)# interface Eth 1/16
Leaf4(config-if-Eth1/16)# description Link to Spine1
Leaf4(config-if-Eth1/16)# speed 800000
Leaf4(config-if-Eth1/16)# unreliable-los auto
Leaf4(config-if-Eth1/16)# no shutdown
Leaf4(config-if-Eth1/16)# mtu 9100
Leaf4(config-if-Eth1/16)# ipv6 enable
Leaf4(config-if-Eth1/16)# ars bind default
Leaf4(config-if-Eth1/16)# exit
! Setup uplink interfaces to Spine2
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Leaf4(config)# interface Eth 1/17
Leaf4(config-if-Eth1/17)# description Link to Spine2
Leaf4(config-if-Eth1/17)# speed 800000
Leaf4(config-if-Eth1/17)# unreliable-los auto
Leaf4(config-if-Eth1/17)# no shutdown
Leaf4(config-if-Eth1/17)# mtu 9100
Leaf4(config-if-Eth1/17)# ipv6 enable
Leaf4(config-if-Eth1/17)# ars bind default
Leaf4(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Leaf4(config)# interface Eth 1/32
Leaf4(config-if-Eth1/32)# description Link to Spine2
Leaf4(config-if-Eth1/32)# speed 800000
Leaf4(config-if-Eth1/32)# unreliable-los auto
Leaf4(config-if-Eth1/32)# no shutdown
Leaf4(config-if-Eth1/32)# mtu 9100
Leaf4(config-if-Eth1/32)# ipv6 enable
Leaf4(config-if-Eth1/32)# ars bind default
Leaf4(config-if-Eth1/32)# exit
! Create tenant VRFs for a multi-tenant environment (NOTE:  Can just setup a single tenant/VRF only also)
```

```
Leaf4(config)# ip vrf Corona
! No nodes for Heineken here, but for future if needed
Leaf4(config)# ip vrf Heineken
Leaf4(config)# ip vrf Budweiser
! Assign /31 IP's to Corona's host interfaces Eth 1/33-1/64 (showing just first and last)
Leaf4(config)# interface Eth 1/33
Leaf4(config-if-Eth1/33)# speed 400000
Leaf4(config-if-Eth1/33)# mtu 9100
Leaf4(config-if-Eth1/33)# fec RS
Leaf4(config-if-Eth1/33)# standalone-link-training
Leaf4(config-if-Eth1/33)# unreliable-los auto
Leaf4(config-if-Eth1/33)# no shutdown
Leaf4(config-if-Eth1/33)# ip address 100.65.0.192/31
Leaf4(config-if-Eth1/33)# description Link to Corona Node 13 RNIC Slot 1 with IP 100.65.0.193/31
Leaf4(config-if-Eth1/33)# ip vrf forwarding Corona
Leaf4(config-if-Eth1/33)# exit
! Range of 30 links here 1/34-1/63
Leaf4(config)# interface Eth 1/64
Leaf4(config-if-Eth1/64)# speed 400000
Leaf4(config-if-Eth1/64)# mtu 9100
Leaf4(config-if-Eth1/64)# fec RS
Leaf4(config-if-Eth1/64)# standalone-link-training
Leaf4(config-if-Eth1/64)# unreliable-los auto
Leaf4(config-if-Eth1/64)# no shutdown
Leaf4(config-if-Eth1/64)# ip address 100.65.0.254/31
Leaf4(config-if-Eth1/64)# description Link to Corona Node 16 RNIC Slot 8 with IP 100.65.0.255/31
Leaf4(config-if-Eth1/64)# ip vrf forwarding Corona
Leaf4(config-if-Eth1/64)# exit
! Assign /31 IP's to Budweiser host interfaces Eth 1/65-1/96 (showing just first and last)
Leaf4(config)# interface Eth 1/65
Leaf4(config-if-Eth1/65)# speed 400000
Leaf4(config-if-Eth1/65)# mtu 9100
Leaf4(config-if-Eth1/65)# fec RS
Leaf4(config-if-Eth1/65)# standalone-link-training
Leaf4(config-if-Eth1/65)# unreliable-los auto
Leaf4(config-if-Eth1/65)# no shutdown
Leaf4(config-if-Eth1/65)# ip address 100.67.0.64/31
Leaf4(config-if-Eth1/65)# description Link to Budweiser Node 5 RNIC Slot 1 with IP 100.67.0.65/31
Leaf4(config-if-Eth1/65)# ip vrf forwarding Budweiser
Leaf4(config-if-Eth1/65)# exit
! Range of 30 links here 1/66-1/95
Leaf4(config)# interface Eth 1/96
Leaf4(config-if-Eth1/96)# speed 400000
Leaf4(config-if-Eth1/96)# mtu 9100
Leaf4(config-if-Eth1/96)# fec RS
Leaf4(config-if-Eth1/96)# standalone-link-training
Leaf4(config-if-Eth1/96)# unreliable-los auto
Leaf4(config-if-Eth1/96)# no shutdown
Leaf4(config-if-Eth1/96)# ip address 100.67.0.126/31
Leaf4(config-if-Eth1/96)# description Link to Budweiser Node 8 RNIC Slot 8 with IP 100.67.0.127/31
Leaf4(config-if-Eth1/96)# ip vrf forwarding Budweiser
Leaf4(config-if-Eth1/96)# exit
! Configure L3 VNI VLANs
Leaf4(config)# interface Vlan 61
Leaf4(config-if-Vlan61)# ip vrf forwarding Corona
Leaf4(config-if-Vlan61)# exit
Leaf4(config)# interface Vlan 62
Leaf4(config-if-Vlan62)# ip vrf forwarding Heineken
Leaf4(config-if-Vlan62)# exit
Leaf4(config)# interface Vlan 63
Leaf4(config-if-Vlan63)# ip vrf forwarding Budweiser
Leaf4(config-if-Vlan63)# exit
! Map VNIs to VLANs and L3 VNIs to VRFs
Leaf4(config)# interface vxlan vtep-4
Leaf4(config-if-vxlan-vtep-4)# source-ip 100.64.5.4
Leaf4(config-if-vxlan-vtep-4)# map vni 610 vlan 61
Leaf4(config-if-vxlan-vtep-4)# map vni 620 vlan 62
Leaf4(config-if-vxlan-vtep-4)# map vni 630 vlan 63
Leaf4(config-if-vxlan-vtep-4)# map vni 610 vrf Corona
Leaf4(config-if-vxlan-vtep-4)# map vni 620 vrf Heineken
Leaf4(config-if-vxlan-vtep-4)# map vni 630 vrf Budweiser
Leaf4(config-if-vxlan-vtep-4)# qos-mode uniform
Leaf4(config-if-vxlan-vtep-4)# exit
```

```
! setup underlay and overlay BGP
Leaf4(config)# router bgp 65104
Leaf4(config-router-bgp)# router-id 100.64.3.4
Leaf4(config-router-bgp)# address-family ipv4 unicast
Leaf4(config-router-bgp-af)# redistribute connected
Leaf4(config-router-bgp-af)# maximum-paths 64
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# address-family l2vpn evpn
Leaf4(config-router-bgp-af)# advertise-all-vni
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# peer-group SPINES
Leaf4(config-router-bgp-pg)# remote-as external
Leaf4(config-router-bgp-pg)# timers 3 9
Leaf4(config-router-bgp-pg)# advertisement-interval 5
Leaf4(config-router-bgp-pg)# bfd
Leaf4(config-router-bgp-pg)# capability extended-nexthop
Leaf4(config-router-bgp-pg)# address-family ipv4 unicast
Leaf4(config-router-bgp-pg-af)# activate
Leaf4(config-router-bgp-pg-af)# exit
Leaf4(config-router-bgp-pg)# address-family l2vpn evpn
Leaf4(config-router-bgp-pg-af)# activate
Leaf4(config-router-bgp-pg-af)# exit
Leaf4(config-router-bgp-pg)# exit
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 neighbors in this block (showing just first and last)
Leaf4(config-router-bgp)# neighbor interface Eth 1/1
Leaf4(config-router-bgp-neighbor)# description Link to Spine1
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/2-1/15
Leaf4(config-router-bgp)# neighbor interface Eth 1/16
Leaf4(config-router-bgp-neighbor)# description Link to Spine1
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Leaf4(config-router-bgp)# neighbor interface Eth 1/17
Leaf4(config-router-bgp-neighbor)# description Link to Spine2
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
! Range of 14 links here 1/18-1/31
Leaf4(config-router-bgp)# neighbor interface Eth 1/32
Leaf4(config-router-bgp-neighbor)# description Link to Spine2
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
Leaf4(config-router-bgp)# exit
Leaf4(config)# router bgp 65104 vrf Corona
Leaf4(config-router-bgp)# address-family ipv4 unicast
Leaf4(config-router-bgp-af)# redistribute connected
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# address-family l2vpn evpn
Leaf4(config-router-bgp-af)# advertise ipv4 unicast
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# exit
Leaf4(config)# router bgp 65104 vrf Heineken
Leaf4(config-router-bgp)# address-family ipv4 unicast
Leaf4(config-router-bgp-af)# redistribute connected
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# address-family l2vpn evpn
Leaf4(config-router-bgp-af)# advertise ipv4 unicast
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# exit
Leaf4(config)# router bgp 65104 vrf Budweiser
Leaf4(config-router-bgp)# address-family ipv4 unicast
Leaf4(config-router-bgp-af)# redistribute connected
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# address-family l2vpn evpn
Leaf4(config-router-bgp-af)# advertise ipv4 unicast
Leaf4(config-router-bgp-af)# exit
Leaf4(config-router-bgp)# exit
Leaf4(config)# exit
Leaf4# write memory
```

Spine1

```
Spine1# config terminal
Spine1(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Spine1(config)# roce enable
! Assign Router ID addresses to loopback interface
Spine1(config)# interface loopback 0
Spine1(config-if-lo0)# description Router-id
Spine1(config-if-lo0)# ip address 100.64.1.1/32
Spine1(config-if-lo0)# exit
! Setup the Adaptive Routing and switching globals
Spine1(config)# ars profile default
Spine1(config-ars-profile)# exit
Spine1(config)# ars bind default
Spine1(config)# ars port-profile default
Spine1(config-ars-port-profile)# enable
Spine1(config-ars-port-profile)# exit
Spine1(config)# ars object default
Spine1(config-ars-object)# exit
Spine1(config)# route-map ars-map permit 10
Spine1(config-route-map)# set ars-object default
Spine1(config-route-map)# exit
Spine1(config)# ip protocol any route-map ars-map
Spine1(config)# route-map RM_SET_SRC permit 10
Spine1(config-route-map)# set ars-object default
Spine1(config-route-map)# exit
! Setup downlink interfaces to Leaf1
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Spine1(config)# interface Eth 1/1
Spine1(config-if-Eth1/1)# description Link to Leaf1
Spine1(config-if-Eth1/1)# speed 800000
Spine1(config-if-Eth1/1)# unreliable-los auto
Spine1(config-if-Eth1/1)# no shutdown
Spine1(config-if-Eth1/1)# mtu 9100
Spine1(config-if-Eth1/1)# ipv6 enable
Spine1(config-if-Eth1/1)# ars bind default
Spine1(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Spine1(config)# interface Eth 1/16
Spine1(config-if-Eth1/16)# description Link to Leaf1
Spine1(config-if-Eth1/16)# speed 800000
Spine1(config-if-Eth1/16)# unreliable-los auto
Spine1(config-if-Eth1/16)# no shutdown
Spine1(config-if-Eth1/16)# mtu 9100
Spine1(config-if-Eth1/16)# ipv6 enable
Spine1(config-if-Eth1/16)# ars bind default
Spine1(config-if-Eth1/16)# exit
! Setup downlink interfaces to Leaf2
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Spine1(config)# interface Eth 1/17
Spine1(config-if-Eth1/17)# description Link to Leaf2
Spine1(config-if-Eth1/17)# speed 800000
Spine1(config-if-Eth1/17)# unreliable-los auto
Spine1(config-if-Eth1/17)# no shutdown
Spine1(config-if-Eth1/17)# mtu 9100
Spine1(config-if-Eth1/17)# ipv6 enable
Spine1(config-if-Eth1/17)# ars bind default
Spine1(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Spine1(config)# interface Eth 1/32
Spine1(config-if-Eth1/32)# description Link to Leaf2
Spine1(config-if-Eth1/32)# speed 800000
Spine1(config-if-Eth1/32)# unreliable-los auto
Spine1(config-if-Eth1/32)# no shutdown
Spine1(config-if-Eth1/32)# mtu 9100
Spine1(config-if-Eth1/32)# ipv6 enable
Spine1(config-if-Eth1/32)# ars bind default
Spine1(config-if-Eth1/32)# exit
! Setup downlink interfaces to Leaf3
! Note - do all 1/33, 1/34, etc. to 1/47, 1/48 in this block (showing just first and last)
Spine1(config)# interface Eth 1/33
Spine1(config-if-Eth1/33)# description Link to Leaf3
Spine1(config-if-Eth1/33)# speed 800000
```

```
Spine1(config-if-Eth1/33)# unreliable-los auto
Spine1(config-if-Eth1/33)# no shutdown
Spine1(config-if-Eth1/33)# mtu 9100
Spine1(config-if-Eth1/33)# ipv6 enable
Spine1(config-if-Eth1/33)# ars bind default
Spine1(config-if-Eth1/33)# exit
! Range of 14 links here 1/34-1/47
Spine1(config)# interface Eth 1/48
Spine1(config-if-Eth1/48)# description Link to Leaf3
Spine1(config-if-Eth1/48)# speed 800000
Spine1(config-if-Eth1/48)# unreliable-los auto
Spine1(config-if-Eth1/48)# no shutdown
Spine1(config-if-Eth1/48)# mtu 9100
Spine1(config-if-Eth1/48)# ipv6 enable
Spine1(config-if-Eth1/48)# ars bind default
Spine1(config-if-Eth1/48)# exit
! Setup downlink interfaces to Leaf4
! Note - do all 1/49, 1/50, etc. to 1/63, 1/64 in this block (showing just first and last)
Spine1(config)# interface Eth 1/49
Spine1(config-if-Eth1/49)# description Link to Leaf4
Spine1(config-if-Eth1/49)# speed 800000
Spine1(config-if-Eth1/49)# unreliable-los auto
Spine1(config-if-Eth1/49)# no shutdown
Spine1(config-if-Eth1/49)# mtu 9100
Spine1(config-if-Eth1/49)# ipv6 enable
Spine1(config-if-Eth1/49)# ars bind default
Spine1(config-if-Eth1/49)# exit
! Range of 14 links here 1/50-1/63
Spine1(config)# interface Eth 1/64
Spine1(config-if-Eth1/64)# description Link to Leaf4
Spine1(config-if-Eth1/64)# speed 800000
Spine1(config-if-Eth1/64)# unreliable-los auto
Spine1(config-if-Eth1/64)# no shutdown
Spine1(config-if-Eth1/64)# mtu 9100
Spine1(config-if-Eth1/64)# ipv6 enable
Spine1(config-if-Eth1/64)# ars bind default
Spine1(config-if-Eth1/64)# exit
! Configure the underlay BGP
Spine1(config)# router bgp 65001
Spine1(config-router-bgp)# router-id 100.64.1.1
Spine1(config-router-bgp)# log-neighbor-changes
Spine1(config-router-bgp)# bestpath as-path multipath-relax
Spine1(config-router-bgp)# timers 60 180
Spine1(config-router-bgp)# address-family ipv4 unicast
Spine1(config-router-bgp-af)# redistribute connected
Spine1(config-router-bgp-af)# maximum-paths 64
Spine1(config-router-bgp-af)# exit
Spine1(config-router-bgp)# peer-group LEAFS
Spine1(config-router-bgp-pg)# remote-as external
Spine1(config-router-bgp-pg)# timers 3 9
Spine1(config-router-bgp-pg)# advertisement-interval 5
Spine1(config-router-bgp-pg)# bfd
Spine1(config-router-bgp-pg)# capability extended-nexthop
Spine1(config-router-bgp-pg)# address-family ipv4 unicast
Spine1(config-router-bgp-pg-af)# activate
Spine1(config-router-bgp-pg-af)# exit
! Setup all BGP neighbor interfaces to all Leafs (Note- just showing start and last 2 per leaf here)
Spine1(config-router-bgp)# neighbor interface Eth 1/1
Spine1(config-router-bgp-neighbor)# description Link to Leaf1
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/2-1/15
Spine1(config-router-bgp)# neighbor interface Eth 1/16
Spine1(config-router-bgp-neighbor)# description Link to Leaf1
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Spine1(config-router-bgp)# neighbor interface Eth 1/17
Spine1(config-router-bgp-neighbor)# description Link to Leaf2
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/18-1/31
Spine1(config-router-bgp)# neighbor interface Eth 1/32
```

```
Spine1(config-router-bgp-neighbor)# description Link to Leaf2
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Note - do all 1/33, 1/34, etc. to 1/47, 1/48 neighbors in this block (showing just first and last)
Spine1(config-router-bgp)# neighbor interface Eth 1/33
Spine1(config-router-bgp-neighbor)# description Link to Leaf3
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/34-1/47
Spine1(config-router-bgp)# neighbor interface Eth 1/48
Spine1(config-router-bgp-neighbor)# description Link to Leaf3
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Note - do all 1/49, 1/50, etc. to 1/63, 1/64 neighbors in this block (showing just first and last)
Spine1(config-router-bgp)# neighbor interface Eth 1/49
Spine1(config-router-bgp-neighbor)# description Link to Leaf4
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/50-1/63
Spine1(config-router-bgp)# neighbor interface Eth 1/64
Spine1(config-router-bgp-neighbor)# description Link to Leaf4
Spine1(config-router-bgp-neighbor)# peer-group LEAFS
Spine1(config-router-bgp-neighbor)# exit
Spine1(config-router-bgp)# exit
Spine1(config)# exit
Spine1# write memory
```

Spine2

```
Spine2# config terminal
Spine1(config)# lldp enable
! Now we are ready to initialize all default RoCE buffers and QoS under a single command
Spine2(config)# roce enable
! Assign Router ID addresses to loopback interface
Spine2(config)# interface loopback 0
Spine2(config-if-lo0)# description Router-id
Spine2(config-if-lo0)# ip address 100.64.1.2/32
Spine2(config-if-lo0)# exit
! Setup the Adaptive Routing and switching globals
Spine2(config)# ars profile default
Spine2(config-ars-profile)# exit
Spine2(config)# ars bind default
Spine2(config)# ars port-profile default
Spine2(config-ars-port-profile)# enable
Spine2(config-ars-port-profile)# exit
Spine2(config)# ars object default
Spine2(config-ars-object)# exit
Spine2(config)# route-map ars-map permit 10
Spine2(config-route-map)# set ars-object default
Spine2(config-route-map)# exit
Spine2(config)# ip protocol any route-map ars-map
Spine2(config)# route-map RM_SET_SRC permit 10
Spine2(config-route-map)# set ars-object default
Spine2(config-route-map)# exit
! Setup downlink interfaces to Leaf1
! Note - do all 1/1, 1/2, etc. to 1/15, 1/16 in this block (showing just first and last)
Spine2(config)# interface Eth 1/1
Spine2(config-if-Eth1/1)# description Link to Leaf1
Spine2(config-if-Eth1/1)# speed 800000
Spine2(config-if-Eth1/1)# unreliable-los auto
Spine2(config-if-Eth1/1)# no shutdown
Spine2(config-if-Eth1/1)# mtu 9100
Spine2(config-if-Eth1/1)# ipv6 enable
Spine2(config-if-Eth1/1)# ars bind default
Spine2(config-if-Eth1/1)# exit
! Range of 14 links here 1/2-1/15
Spine2(config)# interface Eth 1/16
Spine2(config-if-Eth1/16)# description Link to Leaf1
Spine2(config-if-Eth1/16)# speed 800000
Spine2(config-if-Eth1/16)# unreliable-los auto
Spine2(config-if-Eth1/16)# no shutdown
Spine2(config-if-Eth1/16)# mtu 9100
Spine2(config-if-Eth1/16)# ipv6 enable
Spine2(config-if-Eth1/16)# ars bind default
Spine2(config-if-Eth1/16)# exit
! Setup downlink interfaces to Leaf2
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 in this block (showing just first and last)
Spine2(config)# interface Eth 1/17
Spine2(config-if-Eth1/17)# description Link to Leaf2
Spine2(config-if-Eth1/17)# speed 800000
Spine2(config-if-Eth1/17)# unreliable-los auto
Spine2(config-if-Eth1/17)# no shutdown
Spine2(config-if-Eth1/17)# mtu 9100
Spine2(config-if-Eth1/17)# ipv6 enable
Spine2(config-if-Eth1/17)# ars bind default
Spine2(config-if-Eth1/17)# exit
! Range of 14 links here 1/18-1/31
Spine2(config)# interface Eth 1/32
Spine2(config-if-Eth1/32)# description Link to Leaf2
Spine2(config-if-Eth1/32)# speed 800000
Spine2(config-if-Eth1/32)# unreliable-los auto
Spine2(config-if-Eth1/32)# no shutdown
Spine2(config-if-Eth1/32)# mtu 9100
Spine2(config-if-Eth1/32)# ipv6 enable
Spine2(config-if-Eth1/32)# ars bind default
Spine2(config-if-Eth1/32)# exit
! Setup downlink interfaces to Leaf3
! Note - do all 1/33, 1/34, etc. to 1/47, 1/48 in this block (showing just first and last)
Spine2(config)# interface Eth 1/33
Spine2(config-if-Eth1/33)# description Link to Leaf3
Spine2(config-if-Eth1/33)# speed 800000
```

```
Spine2(config-if-Eth1/33)# unreliable-los auto
Spine2(config-if-Eth1/33)# no shutdown
Spine2(config-if-Eth1/33)# mtu 9100
Spine2(config-if-Eth1/33)# ipv6 enable
Spine2(config-if-Eth1/33)# ars bind default
Spine2(config-if-Eth1/33)# exit
! Range of 14 links here 1/34-1/47
Spine2(config)# interface Eth 1/48
Spine2(config-if-Eth1/48)# description Link to Leaf3
Spine2(config-if-Eth1/48)# speed 800000
Spine2(config-if-Eth1/48)# unreliable-los auto
Spine2(config-if-Eth1/48)# no shutdown
Spine2(config-if-Eth1/48)# mtu 9100
Spine2(config-if-Eth1/48)# ipv6 enable
Spine2(config-if-Eth1/48)# ars bind default
Spine2(config-if-Eth1/48)# exit
! Setup downlink interfaces to Leaf4
! Note - do all 1/49, 1/50, etc. to 1/63, 1/64 in this block (showing just first and last)
Spine2(config)# interface Eth 1/49
Spine2(config-if-Eth1/49)# description Link to Leaf4
Spine2(config-if-Eth1/49)# speed 800000
Spine2(config-if-Eth1/49)# unreliable-los auto
Spine2(config-if-Eth1/49)# no shutdown
Spine2(config-if-Eth1/49)# mtu 9100
Spine2(config-if-Eth1/49)# ipv6 enable
Spine2(config-if-Eth1/49)# ars bind default
Spine2(config-if-Eth1/49)# exit
! Range of 14 links here 1/50-1/63
Spine2(config)# interface Eth 1/64
Spine2(config-if-Eth1/64)# description Link to Leaf4
Spine2(config-if-Eth1/64)# speed 800000
Spine2(config-if-Eth1/64)# unreliable-los auto
Spine2(config-if-Eth1/64)# no shutdown
Spine2(config-if-Eth1/64)# mtu 9100
Spine2(config-if-Eth1/64)# ipv6 enable
Spine2(config-if-Eth1/64)# ars bind default
Spine2(config-if-Eth1/64)# exit
! Configure the underlay BGP
Spine2(config)# router bgp 65001
Spine2(config-router-bgp)# router-id 100.64.1.2
Spine2(config-router-bgp)# log-neighbor-changes
Spine2(config-router-bgp)# bestpath as-path multipath-relax
Spine2(config-router-bgp)# timers 60 180
Spine2(config-router-bgp)# address-family ipv4 unicast
Spine2(config-router-bgp-af)# redistribute connected
Spine2(config-router-bgp-af)# maximum-paths 64
Spine2(config-router-bgp-af)# exit
Spine2(config-router-bgp)# peer-group LEAFS
Spine2(config-router-bgp-pg)# remote-as external
Spine2(config-router-bgp-pg)# timers 3 9
Spine2(config-router-bgp-pg)# advertisement-interval 5
Spine2(config-router-bgp-pg)# bfd
Spine2(config-router-bgp-pg)# capability extended-nexthop
Spine2(config-router-bgp-pg)# address-family ipv4 unicast
Spine2(config-router-bgp-pg-af)# activate
Spine2(config-router-bgp-pg-af)# exit
! Setup all BGP neighbor interfaces to all Leafs (Note- just showing start and last 2 per leaf here)
Spine2(config-router-bgp)# neighbor interface Eth 1/1
Spine2(config-router-bgp-neighbor)# description Link to Leaf1
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/2-1/15
Spine2(config-router-bgp)# neighbor interface Eth 1/16
Spine2(config-router-bgp-neighbor)# description Link to Leaf1
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Note - do all 1/17, 1/18, etc. to 1/31, 1/32 neighbors in this block (showing just first and last)
Spine2(config-router-bgp)# neighbor interface Eth 1/17
Spine2(config-router-bgp-neighbor)# description Link to Leaf2
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/18-1/31
Spine2(config-router-bgp)# neighbor interface Eth 1/32
```

```
Spine2(config-router-bgp-neighbor)# description Link to Leaf2
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Note - do all 1/33, 1/34, etc. to 1/47, 1/48 neighbors in this block (showing just first and last)
Spine2(config-router-bgp)# neighbor interface Eth 1/33
Spine2(config-router-bgp-neighbor)# description Link to Leaf3
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/34-1/47
Spine2(config-router-bgp)# neighbor interface Eth 1/48
Spine2(config-router-bgp-neighbor)# description Link to Leaf3
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Note - do all 1/49, 1/50, etc. to 1/63, 1/64 neighbors in this block (showing just first and last)
Spine2(config-router-bgp)# neighbor interface Eth 1/49
Spine2(config-router-bgp-neighbor)# description Link to Leaf4
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
! Range of 14 Neighbors here 1/50-1/63
Spine2(config-router-bgp)# neighbor interface Eth 1/64
Spine2(config-router-bgp-neighbor)# description Link to Leaf4
Spine2(config-router-bgp-neighbor)# peer-group LEAFS
Spine2(config-router-bgp-neighbor)# exit
Spine2(config-router-bgp)# exit
Spine2(config)# exit
Spine2# write memory
```

Appendix B: Detail on how the Switch QoS will be setup

DSCP Values from RNIC

When using the roce enable command on the switches in this cluster, the following configuration will be set up.

DSCP Value:

26 will be set up for the AMD Pollara Pollara-1Q400P RoCEv2 traffic

- traffic-class 3, priority-group 3, queue 3, pfc-priority-group 3, pfc-priority-queue 3, no-drop enable

26 will be set up in case the customer wishes to attach Brocade Thor2 Adapters in the MI GPU systems

- traffic-class 3, priority-group 3, queue 3, pfc-priority-group 3, pfc-priority-queue 3

DSCP 48 will be set up for CNP handling

- traffic-class 6, priority-group 7, queue 0, pfc-priority-group 0, pfc-priority-queue 0

Scheduler on all Switches

Switch scheduling policy for queues

0 will be dwrr with a weight of 50
3 will be dwrr with a weight of 50
6 will be strict

ECN Configuration on all Switches

ECN is ideally the first method for congestion signaling and control, whereas the PFC configuration is used as a last resort to pause traffic and hop-by-hop backpressure to the sender. These settings are all assuming 400GE RNIC that are mapped 1:1 to MI325X GPU's via internal PLX switch for optimized RDMA without any PCI-PCI bridges (i.e. switched on identical bus)

qos wred-policy is green ECN with min threshold of 1000k Bytes, max of 3000k Bytes, and a drop probability of 20%

For interfaces 200GE and below, we make the min/max/drop values to 500kB/1500kB/20%

Buffer Configuration on all Switches

On a switch which has both per port dedicated and global shared buffers, ingress traffic can still come in upon the act of asserting a pause frame (IEEE 802.1x) to the device and the time the device actually pauses transmission. The size of buffers for this purpose is called the headroom and is set below to ~2.6MB.

shared-headroom-size set to size/mode of 2621440/dynamic

On the switch silicon inside the SSE-T8164S the default internal buffer allocations are not user configurable but are dynamically optimized when RoCEv2 is enabled on the switch, inclusive of accounting for the port speeds the traffic will ingress/egress. The configuration of the PFC pause operation thresholds are also optimized – and any modification requires expert level support – hence outside a discussion in this broader validated design document. Therefore, the internal sizing is not exposed to the level of the configuration file.

Overall Default QoS Sections of larger running-configuration on all Switches

```
!
roce enable
!
qos map dscp-tc ROCE
dscp 0-3,5-23,25,27-47,49-63 traffic-class 0
dscp 24,26 traffic-class 3
dscp 4 traffic-class 3
dscp 48 traffic-class 6
!
qos map tc-queue ROCE
traffic-class 0 queue 0
traffic-class 1 queue 1
traffic-class 2 queue 2
traffic-class 3 queue 3
traffic-class 4 queue 4
traffic-class 5 queue 5
traffic-class 6 queue 6
traffic-class 7 queue 7
!
qos map tc-pg ROCE
traffic-class 3 priority-group 3
traffic-class 4 priority-group 4
traffic-class 0-2,4-7 priority-group 7
!
qos map pfc-priority-queue ROCE
pfc-priority 0 queue 0
pfc-priority 1 queue 1
pfc-priority 2 queue 2
pfc-priority 3 queue 3
pfc-priority 4 queue 4
pfc-priority 5 queue 5
pfc-priority 6 queue 6
pfc-priority 7 queue 7
!
qos wred-policy ROCE
green minimum-threshold 1000 maximum-threshold 3000 drop-probability 20
ecn green
!
qos scheduler-policy ROCE
!
queue 0
type dwrr
weight 50
!
queue 3
```

```
type dwrr
weight 50
!
queue 4
type dwrr
weight 50
!
queue 6
type strict
! Showing one sample interface on leaf 1
interface Ethernet1/9/1
mtu 9100
speed 400000
ip address 100.65.1.0/31
description Link to Corona Node RNIC with IP 100.65.1.1/31
ip vrf forwarding Corona
unreliable-los auto
no shutdown
queue 3 wred-policy ROCE
scheduler-policy ROCE
qos-map dscp-tc ROCE
qos-map tc-queue ROCE
qos-map tc-pg ROCE
qos-map pfc-priority-queue ROCE
priority-flow-control priority 3
priority-flow-control watchdog action drop
priority-flow-control watchdog on detect-time 200
priority-flow-control watchdog restore-time 400
!
```

Appendix C: Connection maps to manage numbers of links

One important aspect of the deployment time is the interconnection of all these nodes, leafs, spines, and potentially superspines for a given cluster. Below is a shortened list of the interconnections in this validated design, with suggested labelling on the cables for ease in locating, replacing, troubleshooting, etc. as required. Ideally the tooling to perform the cluster design, can also output a connection map like these below. If the installer focuses on the label columns, they can place tags on the cable ends for simpler installation and any required operations later.

System to Leaf DAC Connection Map (with AOC-S400G-B1C in slots 1-8 on server)

Leaf	Leaf Port	Leaf End Cable Label	Cable Breakout Port	RNIC PCIe Slot	RNIC Port	Node	RNIC Port End Cable Label	Type	Length (m)
1	9	L1P9-N1PCI1_2	1	1	1	1	N1PCI1-L1P9	DAC	2
1	9		2	2	1	1	N1PCI2-L1P9	DAC	2
<…>	<…>	<…>	<…>	<…>	<…>	<…>	<…>	<…>	<…>
1	56	L1P56-N8PCI7_8	1	7	1	8	N8PCI7-L1P56	DAC	2
1	56		2	8	1	8	N8PCI8-L1P56	DAC	2
2	9	L2P9-N9PCI1_2	1	1	1	9	N9PCI1-L2P9	DAC	2
2	9		2	2	1	9	N9PCI2-L2P9	DAC	2

Entries continue per the leaf to node ports outlined in appendix a….

Leaf to Spine Fiber Connection Map

Leaf	Leaf Port	Leaf Sub Port	Leaf End Fiber Label	Spine	Spine Port	Spine Sub Port	RNIC Port End Cable Label	Type	Length (m)
1	1	1	L1P1-1-S1P1-1	1	1	1	S1P1-1-L1P1-1	VR8MMF	50
1	1	2	L1P1-2-S1P1-2	1	1	2	N1S3-L1P9-2	VR8MMF	50
<…>	<…>		<…>	<…>	<…>	<…>	<…>	<…>	<…>
1	64	1	L1P64-1-S2P16-1	2	16	1	N8S5-L1P56-1	VR8MMF	50
1	64	2	L1P64-2-S2P16-2	2	16	2	N8S7-L1P56-1	VR8MMF	50
2	1	1	L2P1-1-S1P17-1	1	17	1	N9S1-L1P9-1	VR8MMF	50
2	1	2	L2P1-2-S1P17-2	1	17	2	N9S3-L2P9-2	VR8MMF	50

Entries continue per the uplink ports outlined in appendix a…

After installation is complete, assuming you have installed the lldp daemon on each node per appendix E, on each leaf you can execute a 'show lldp neighbor' to confirm correct cabling in the entire cluster.

Appendix D: Server and RNIC Configuration Steps

All servers are running Ubuntu 24.04

BIOS & Grub Settings

Supermicro recommended BIOS settings for the AS-8126GS-TNMR (MI325X):

Here we will present our settings that are based on AMD recommendations at the link below. We do recommend the reader validate the entries below on that side, as sometimes recommendations are updated for a variety of reasons.

Link to AMD MI300X system documentation and steps (Identical settings to the MI325X solution):

https://rocm.docs.amd.com/en/latest/how-to/system-optimization/mi300x.html#mi300x-bios-settings

Some keys from that more exhaustive link are shared here as key elements in the table below:

BIOS setting location	Parameter	Value	Comments
Advanced / PCI subsystem settings	Above 4G decoding	Enabled	GPU large BAR support.
Advanced / PCI subsystem settings	SR-IOV support	Enabled	Enable single root IO virtualization.
AMD CBS / GPU common options	Global C-state control	Auto	Global C-states – do not disable this menu item).
AMD CBS / GPU common options	CCD/Core/Thread enablement	Accept	May be necessary to enable the SMT control menu.
AMD CBS / GPU common options / performance	SMT control	Disable	Set to Auto if the primary application is not compute-bound.
AMD CBS / DF common options / memory addressing	NUMA nodes per socket	Auto	Auto = NPS1. At this time, the other options for NUMA nodes per socket should not be used.
AMD CBS / DF common options / memory addressing	Memory interleaving	Auto	Depends on NUMA nodes (NPS) setting.
AMD CBS / DF common options / link	4-link xGMI max speed	32 Gbps	Auto results in the speed being set to the lower of the max speed the motherboard is designed to support and the max speed of the CPU in use.
AMD CBS / NBIO common options	IOMMU	Enabled	
AMD CBS / NBIO common options	PCIe ten bit tag support	Auto	
AMD CBS / NBIO common options / SMU common options	Determinism control	Manual	
AMD CBS / NBIO common options / SMU common options	Determinism slider	Power	
AMD CBS / NBIO common options / SMU common options	cTDP control	Manual	Set cTDP to the maximum supported by the installed CPU.
AMD CBS / NBIO common options / SMU common options	cTDP	400	Value in watts.
AMD CBS / NBIO common options / SMU common options	Package power limit control	Manual	Set package power limit to the maximum supported by the installed CPU.

BIOS setting location	Parameter	Value	Comments
AMD CBS / NBIO common options / SMU common options	Package power limit	400	Value in watts.
AMD CBS / NBIO common options / SMU common options	xGMI link width control	Manual	Set package power limit to the maximum supported by the installed CPU.
AMD CBS / NBIO common options / SMU common options	xGMI force width control	Force	
AMD CBS / NBIO common options / SMU common options	xGMI force link width	2	• 0: Force xGMI link width to x2 • 1: Force xGMI link width to x8 • 2: Force xGMI link width to x16
AMD CBS / NBIO common options / SMU common options	xGMI max speed	Auto	Auto results in the speed being set to the lower of the max speed the motherboard is designed to support and the max speed of the CPU in use.
AMD CBS / NBIO common options / SMU common options	APBDIS	1	Disable DF (data fabric) P-states
AMD CBS / NBIO common options / SMU common options	DF C-states	Auto	
AMD CBS / NBIO common options / SMU common options	Fixed SOC P-state	P0	
AMD CBS / security	TSME	Disabled	Memory encryption

For the RNIC in this solution, these settings are also needed:
Advanced -> PCIe/PCI/PnP Configuration -> Link Configuration area, set:

- Operational Link Speed: 400Gbps PAM4-112
- Link FEC: RS544
- Port Link Training: Enabled

Finally, to ensure valid connectivity on the cluster, add in the LLDP processes to each node with: “sudo apt-get install lldpd”

Recommended GRUB customization

GRUB settings pulled from the link above:

In any modern Linux distribution, the /etc/default/grub file is used to configure GRUB. In this file, the string assigned to GRUB_CMDLINE_LINUX is the command line parameters that Linux uses during boot.

It is recommended to append the following strings in GRUB_CMDLINE_LINUX.

pci=realloc=off

With this setting Linux is able to unambiguously detect all GPUs of the MI325X-based system because this setting disables the automatic reallocation of PCI resources. It’s used when Single Root I/O Virtualization (SR-IOV) Base Address Registers (BARs) have not been allocated by the BIOS. This can help avoid potential issues with certain hardware configurations.

iommu=pt

The iommu=pt setting enables IOMMU pass-through mode. When in pass-through mode, the adapter does not need to use DMA translation to the memory, which can improve performance.

IOMMU is a system specific IO mapping mechanism and can be used for DMA mapping and isolation. This can be beneficial for virtualization and device assignment to virtual machines. It is recommended to enable IOMMU support.

For a system that has AMD host CPUs add this to GRUB_CMDLINE_LINUX:

iommu=pt

Update GRUB

Update GRUB to use the modified configuration:

sudo grub2-mkconfig -o /boot/grub2/grub.cfg

Ubuntu 24.04 PCI BDF & netplan on a sample host for tenant Corona on Leaf1

The default PCI locations for these MI300X/325X & 8 RNICs on the server (corona-node1 as example) are:
root@carona-node1:~# lspci | grep -i -E "nvme|mi300|eth"
Shows the MI300X at:
05:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
26:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
46:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
65:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
85:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
a6:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
c6:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]
e5:00.0 Processing accelerators: Advanced Micro Devices, Inc. [AMD/ATI] Aqua Vanjaram [Instinct MI300X]

Shows the single port 400G GPU RDMA NICs at:
1f:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
41:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
52:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
63:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
a1:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
c1:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
d1:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller
e1:00.0 Ethernet controller: Pensando Systems DSC Ethernet Controller

Shows the 2 port 200G Storage RDMA NICs at:
2f:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
2f:00.1 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
ce:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
ce:00.1 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

AMD Instinct MI300X to RNIC RDMA Mapping:

MI300X Local Rank	PCIe Slot	PCIe Bus Device Function	Assigned Ubuntu 24.04 Interface	IP Address Assigned	Default Gateway
1	1	06:00.0	ens11np0	10.65.0.1/31	10.65.0.0
2	2	27:00.0	ens21np0	10.65.0.3/31	10.65.0.2
3	3	47:00.0	ens31np0	10.65.0.5/31	10.65.0.4
4	4	66:00.0	ens41np0	10.65.0.7/31	10.65.0.6
5	5	86:00.0	ens51np0	10.65.0.9/31	10.65.0.8
6	6	a7:00.0	ens61np0	10.65.0.11/31	10.65.0.10
7	7	c7:00.0	ens71np0	10.65.0.13/31	10.65.0.12
8	8	e6:00.0	ens81np0	10.65.0.15/31	10.65.0.14

Shown below is the relevant portion of one system's Netplan. We will not include the bonding of the 2 10G inband management interfaces, the setup of the customer north-south network with DHCP and those switching elements, as those are all very well understood and established in most customer environments. Here we will focus on the net new RNIC elements within a given node.

Ideally tooling to configure the IP and subnetting on the leafs, will produce these netplan sections once the slot and bus:device:function is determined from the lspci above (converting hex to decimal) for each RNIC in a standardized build, for insertion into automated node provisioning toolsets to scale. Slots 1-8 have cables 2:1 to leaf1 ports Eth 1/9/1-1/12/2 (physical 800G OSFP ports 9-12 on the switch).

```
network:
 ethernets:
   ens11np0:
     addresses:
     - 100.65.0.1/31
     match:
       macaddress: 7c:c2:55:b9:d0:70
     mtu: 9100
     nameservers:
       addresses:
       - 1.1.1.1
       search:
       - maas
     routes:
     -to: 100.65.0.0/16
      via: 100.65.0.0
      set-name: ens11np0
   ens21np0:
     addresses:
     - 100.65.0.3/31
     match:
       macaddress: 7c:c2:55:b9:d1:90
     mtu: 9100
     nameservers:
       addresses:
       - 1.1.1.1
       search:
       - maas
```

```
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.2
     set-name: ens21np0
  ens31np0:
    addresses:
    - 100.65.0.5/31
    match:
      macaddress: 7c:c2:55:b9:d2:a0
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.4
     set-name: ens31np0
  ens41np0:
    addresses:
    - 100.65.0.7/31
    match:
      macaddress: 7c:c2:55:b9:d3:00
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.6
     set-name: ens41np0
  ens51np0:
    addresses:
    - 100.65.0.9/31
    match:
      macaddress: 7c:c2:55:b9:d4:70
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
```

```
      via: 100.65.0.8
      set-name: ens51np0
  ens61np0:
    addresses:
    - 100.65.0.11/31
    match:
      macaddress: 7c:c2:55:b9:d5:90
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.10
     set-name: ens61np0
  ens71np0:
    addresses:
    - 100.65.0.13/31
    match:
      macaddress: 7c:c2:55:b9:d6:a0
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.12
     set-name: ens71np0
  ens81np0:
    addresses:
    - 100.65.0.15/31
    match:
      macaddress: 7c:c2:55:b9:d7:00
    mtu: 9100
    nameservers:
      addresses:
      - 1.1.1.1
      search:
      - maas
    routes:
    -to: 100.65.0.0/16
     via: 100.65.0.14
     set-name: ens81np0
```

AMD Pensando Pollara RoCEv2 Configuration

This is a concise, quick-start guide focused on what is required to run RCCL tests. It is not meant to replace the user guide – please refer to the user guide for additional options and configurations required for general performance improvements. BKC is vendor-dependent and outside the scope of this document. Please ensure that BKC is at the required level for your specific platform. Here, we will present some of the key steps on the host to enable all the AMD Pensando Pollara 400 RNIC and Instinct MI325X components with Ubuntu. For complete installation documentation, release notes, user guides, firmware, and drivers, please contact https://pensandosupport.amd.com/s/login/ to gain access to these resources.

Host Configuration

To ensure valid connectivity on the cluster, add the LLDP processes to each node with: “sudo apt-get install lldpd” Automatic NUMA balancing is a kernel feature that improves application performance on NUMA hardware by moving tasks closer to the memory they access and moving application data closer to the tasks that use it. When disabled (set to 0), it prevents unwanted memory unmapping and can reduce latency in certain workloads.

Check if auto-NUMA balancing is disabled with cat /proc/sys/kernel/numa_balancing or sysctl -a | grep 'kernel.numa_balancing' this will print 0 if disabled and 1 if enabled. Disable NUMA balancing on each participating host:

```
sysctl -w kernel.numa_balancing=0
```

This command only applies the setting temporarily (until reboot). To make this change permanent, you can either:

1. Add the setting to /etc/sysctl.conf:

```
echo "kernel.numa_balancing=0" | sudo tee -a /etc/sysctl.conf
```

1. Or modify GRUB settings by adding numa_balancing=disable to GRUB_CMDLINE_LINUX_DEFAULT in /etc/default/grub and then run sudo update-grub

To achieve optimal performance with Peer Memory Direct, ensure the following system configurations:

- Disable PCIe Access Control Services (ACS):
  ACS must be turned off on the PCIe switch that connects the network interface card (NIC) and the GPU. This allows direct peer-to-peer data transfers between the NIC and GPU over PCIe, bypassing the CPU root complex and maximizing throughput.
- Configure IOMMU for Best Performance:
  For the host system, the IOMMU should either be disabled or set to Pass Through (PT) mode. This minimizes overhead and ensures efficient peer-to-peer communication between devices.
- For Hosts with AMD CPUs:
  Set the IOMMU to PT mode using the kernel command line. This configuration ensures the IOMMU operates in passthrough mode, which is recommended for optimal Peer Memory Direct performance on AMD platforms.

These steps enable efficient PCIe peer-to-peer transfers between the NIC and GPU, ensuring the best possible performance for Peer Memory Direct workloads.

Ensure that ACS is disabled by running sudo lspci -vvv | grep -i "acsctl"; ensure that none of the lines show SrcValid+.

```
setpci -v -s ${BDF} ECAP_ACS+0x6.w=0000
sudo lspci -vvv | grep -i "acsctl"
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir-
UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk- ReqRedir- CmpltRedir-
UpstreamFwd- EgressCtrl- DirectTrans-
ACSCtl: SrcValid- TransBlk-
```

To help with debugging and troubleshooting, it is recommended to override the traditional PCIe BDF device IDs associated by the kernel and set deterministic names. This can be achieved using udev rules.

```
lshw -c network -businfo
Bus info            Device      Class      Description
==========================================================
pci@0000:06:00.0    enp6s0      network    DSC Ethernet Controller
pci@0000:25:00.3    enp37s0f3   network    DSC Ethernet Controller
pci@0000:45:00.3    enp69s0f3   network    DSC Ethernet Controller
pci@0000:65:00.3    enp101s0f3  network    DSC Ethernet Controller
pci@0000:86:00.0    enp134s0    network    DSC Ethernet Controller
pci@0000:a6:00.0    enp166s0    network    DSC Ethernet Controller
pci@0000:c6:00.0    enp198s0    network    DSC Ethernet Controller
pci@0000:e6:00.0    enp230s0    network    DSC Ethernet Controller
```

Create a custom udev rule in /etc/udev/rules.d.
- Set the KERNEL flag to the PCIe BDF.
- Use a name that is easy to identify.

```
cat /etc/udev/rules.d/61-persistent-net.rules
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:06:00.0"  NAME:="ai0"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:26:00.0", NAME:="ai1"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:46:00.0", NAME:="ai2"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:66:00.0", NAME:="ai3"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:86:00.0", NAME:="ai4"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:a6:00.0", NAME:="ai5"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:c6:00.0", NAME:="ai6"
ACTION=="add", SUBSYSTEM=="net", KERNELS=="0000:e6:00.0", NAME:="ai7"
```

Reload for this to take effect.

Configuring Pollara for RoCE Support

Update speed settings for all Pensando Pollara 400 cards in the host:

```
nicctl update port --speed 400g
```

Update FEC settings for all Pensando Pollara 400 cards in the host:

```
nicctl update port --fec-type rs|fc|none
```

Update MTU settings for all Pensando Pollara 400 cards in the host:

```
nicctl update port --mtu <bytes>
```

Note: Valid MTU ranges are 1500 to 9216.
When Pollara is configured for RoCE, it's important to configure the proper DCQCN settings:

- RoCE v2 packets are assigned a DSCP value of 26 and internally use Priority 3
- CNP packets are assigned a DSCP value of 48 and internally use Priority 7
- PFC is enabled specifically for Priority 3 traffic
- Three traffic classes are established:
    - TC0 for non-RoCE traffic
    - TC1 for RoCE traffic
    - TC2 for CNP traffic

RoCE and non-RoCE traffic each receive a 50% share of ETS (Enhanced Transmission Selection) bandwidth, with dynamic allocation—when one traffic type is absent, the other can utilize the full available bandwidth.

CNP traffic receives ETS Strict Priority treatment

Enabling PFC Classification and Priority Mappings
PFC is a flow-control mechanism used to prevent packet loss during congestion. PFC works by pausing the traffic on virtual-queues which are mapped/classified based on the PCP (3-bit Cos) in the VLAN header or DSCP (6-bits) in the IP header.

Enable PFC on the port by configuring --pause-type pfc and --rx\tx-pause enable on the port for the required card:

```
nicctl update port \
-port 04908126-83f8-4242-4242-000011010000 \
--pause-type pfc \
--rx-pause enable \
--tx-pause enable
```

To enable PFC on all ports in a system, use the following bash script.

```
for i in $(sudo nicctl show port | grep Port | awk {'print
$3'}); do sudo nicctl update port -p $i --pause-type pfc --rx-pause enable -
-tx-pause enable; done
```

Configure the QoS classification type to PCP:

```
nicctl update qos --classification-type pcp
```

Note: The classification type must be set before configuring priority mappings.

Configure the pcp-to-priority mappings:

```
nicctl update qos pcp-to-priority \
-port 04908126-83f8-4242-4242-000011010000 \
--pcp 3 \
--priority 3
```

Configure no-drop for the priorities required:

```
nicctl update qos pfc \
-port 04908126-83f8-4242-4242-000011010000 \
--priority 3 --no-drop enable
```

Note: Currently 2 no-drop priorities are supported.

Restart the port:

```
nicctl update port\
-port 04908126-83f8-4242-4242-000011010000 \
--admin-state up
```

Check the QoS configuration and verify the classification type, priority mappings and PFC priority bitmap:

```
nicctl show qos

    NIC  : 42424650-4c32-3433-3930-303630000000 (0000:61:00.0)

    Port: 04908126-83f8-4242-4242-000011010000

      Classification type : PCP

      PCP-to-priority :
        PCP : 00, 04, 05, 06, 07 ==> priority : 0
        PCP : 01 ==> priority : 1
        PCP : 02 ==> priority : 2
        PCP : 03 ==> priority :  3

      PFC priority bitmap : 0x8

      Scheduling :
------------------------------------------------------------------------------
        Priority    Scheduling  Bandwidth  Rate-limit
                    Type        (in %age)  (in Gbps)
------------------------------------------------------------------------------
         0          none        N\A        N\A
         1          none        N\A        N\A
         2          none        N\A        N\A
         3          none        N\A        N\A
```

Individual commands can also be used to verify the configuration details:

```
nicctl show qos --classification-type
----------------------------------------------------------------------------
-------------------------------------------
NIC                                         PCIe BDF
Port                                        Classification type
----------------------------------------------------------------------------
-------------------------------------------
2424650-4c32-3433-3930-303630000000     0000:61:00.0
04908126-83f8-4242-4242-000011010000    PCP

nicctl show qos pcp-to-priority

NIC  : 42424650-4c32-3433-3930-303630000000 (0000:61:00.0)
Port: 04908126-83f8-4242-4242-000011010000
pcp: 00, 04, 05, 06, 07 ==> priority: 0
cp: 01 ==> priority: 1
cp: 02 ==> priority: 2
cp: 03 ==> priority: 3
```

```
nicctl show qos pfc
------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------
NIC                                          PCIe BDF
Port                                         No-drop priorities
------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------
42424650-4c32-3433-3930-303630000000    (0000:61:00.0)
04908126-83f8-4242-4242-000011010000    3
```

You can also verify that the port is receiving pause frames and the Pause Type is PFC using:

```
nicctl show port --brief
```

Enabling DSCP Classification and Priority Mappings

Depending on the user traffic profile, the following steps can be used to

- Map multiple DSCP values to the same or different priority queues.
- Supports two priority queues as no-drop.

By enabling QOS classification-type to DSCP, all DSCP values (0-64) gets mapped to queue 0 by default. To allow any queue other than 0 to be no-drop, map the DSCP value to the queue before configuring it as no-drop, as displayed below.

The following example
• Maps DSCP value 26 to priority queue 3 and configures queue 3 as no-drop.
• Maps RDMA-ACK to different priority queues using the CLI displayed below.
Configure the classification-type to DSCP:

```
nicctl update qos --classification-type dscp
```

Note: The classification type must be set before configuring priority mappings.

Configure the dscp-to-priority mappings:

```
nicctl update qos dscp-to-priority --dscp 26 --priority 3
nicctl update qos dscp-to-purpose --dscp 26 --purpose data
```

Configure no-drop for the priorities required:

```
nicctl update qos pfc --priority 3 --no-drop enable
```

Configure RDMA-ACK to a different priority queue

```
nicctl update qos dscp-to-priority --dscp 48 --priority 7
nicctl update qos dscp-to-purpose --dscp 48 --purpose rdma-ack, xccl-cts
nicctl update qos pfc --priority 7 --no-drop enable
```

Note:
Currently, 2 no-drop priorities are supported.
DSCP to priority mapping is mandatory prior to enabling no-drop
Enabling a no-drop on a Specific Port

```
nicctl update port -p 0490812b-0218-4242-4242-000011010000 --pause-type
pfc --rx-pause enable --tx-pause enable ==> Port Level enable PFC both
RX and TX
nicctl update qos --classification-type dscp
nicctl update qos dscp-to-priority --dscp 26 --priority 3
nicctl update qos dscp-to-purpose --dscp 26 --purpose data
nicctl update qos pfc --priority 3 --no-drop enable
nicctl update qos dscp-to-priority --dscp 48 --priority 7
nicctl update qos dscp-to-purpose --dscp 48 --purpose rdma-ack, xccl-cts
nicctl update qos pfc --priority 7 --no-drop enable
```

DSCP configuration can be verified with nicctl show qos CLI. This shows the summary of the QoS configuration.
Following our configuration example above, we can confirm that DSCP 26 has been mapped to
priority queue 3 and PFC has been enabled for priority queue 3.

```
nicctl show qos

NIC : 42424650-4d32-3530-3830-303135000000 (0000:03:00.0)

Port : 0490812f-06a0-4242-4242-000011010000

Classification type         : DSCP

  DSCP-to-priority :
    DSCP bitmap                 : 0xfffefffffeffffff ==> priority : 0
    DSCP bitmap                 : 0x0001000000000000 ==> priority : 1
    DSCP bitmap                 : 0x0000000001000000 ==> priority : 3
    DSCP                        : 0-23, 25-47, 49-63 ==> priority : 0
    DSCP                        : 48 ==> priority : 1
    DSCP                        : 26 ==> priority : 3

  DSCP-to-purpose               : 26 ==> data
                                : 48 ==> rdma-ack, xccl-cts

  PFC :
    PFC priority bitmap         : 0xa
    PFC no-drop priorities      : 1,3

  Scheduling :
    --------------------------------------------
```

```
Priority  Scheduling  Bandwidth Rate-limit
          Type        (in %age) (in Gbps)
--------------------------------------------
0         DWRR        0         N/A
1         DWRR        0         N/A
3         DWRR        0         N/A
```

In this example configuration, priorities 0, 1 and 5 are DWRR queues with respective bandwidth percent of 70%, 10% and 20%. Priority 6 is strict priority with a rate limiter of 10 Gbps.

```
nicctl update qos scheduling \
--priority 0,1,5,6 \
--dwrr 70,10,20,0 \
--rate-limit 0,0,0,10
```

Recommended DCQCN Configuration

DCQCN Parameter	Description	Recommended Value
--profile-id	DCQCN profile id	1
--ai-rate	Rate increase in AI phase	5
--alpha-update-interval	Alpha update interval	55
--clamp-target-rate	Clamp target rate	disable
--rate-increase-threshold	Rate increase threshold	5
--rate-increase-byte-count	Rate increase byte count	131068
--rate-increase-interval	Rate increase in AI phase	5
--alpha-update-g	Alpha update G value	2
--min-rate	Minimum rate	1
--token-bucket-size	Token bucket size	8000000
--rate-increase-interval	Rate increase interval	5
--hai-rate	Rate increase in HAI phase	50
--initial-alpha-value	Initial alpha value	20
--cnp-dscp	DSCP value used for CNP	48
--rate-reduce-monitor-period	Rate reduce monitor period	50µs

Update Data Center Quantized Congestion Notification (DCQCN) configuration.

```
nicctl update dcqcn \
--roce-device rocep13s0 \
--profile-id 1 \
--alpha-update-interval 55 \
--token-bucket-size 8000000 \
--rate-increase-byte-count 131068 \
--alpha-update-g 2 \
```

```
--clamp-target-rate disable \
--rate-increase-threshold 5 \
--rate-increase-interval 5 \
--min-rate 1 \
--hai-rate 50 \
--initial-alpha-value 20 \
--cnp-dscp 48 \
--rate-reduce-monitor-period 50 \
--ai-rate 5
```

Perform the following final checks to confirm the software, firmware, tools, and settings are configured correctly for optimal Peer Memory Direct performance:

1. Verify the kernel modules ionic_driver, ionic_rdma, and ib_peer_mem.ko) are loaded and of the correct version.
   Note: Some Ubuntu kernels have built-in ib_peer_mem support and don't require AMD's module. The kernel driver Makefile will detect and build accordingly.
2. Ensure the AMD GPU driver (amdgpu.ko) is loaded.
3. Confirm PCIe Access Control Service (ACS) is disabled on the PCIe switch connecting the NIC and GPU. ACS must be disabled to enable PCIe peer-to-peer transactions between GPU and NIC, as enabling ACS can degrade performance.
4. Verify IOMMU is disabled or set to Pass-Through (PT) mode.
5. Confirm the following standard InfiniBand commands run successfully (included in the infiniband-diags package, available through your OS distro's package manager):
   - ibstatus
   - ibv_devinfo -vvv
   - ibdev2netdev
6. Confirm Pensando Pollara configuration settings for RDMA/RoCE defined above are complete, and PCIe Relaxed Ordering. (nicctl show qos)
7. Ensure the Pensando Pollara interface link is active and operating at the correct speed, verified using:
   - ibstatus
   - ethtool <ifname>

Confirm the NIC interface has an assigned IP address, visible as GID 3 (IPv4 or IPv6) through the commands:

Final Checks

Perform the following final checks to confirm the software, firmware, tools, and settings are configured correctly for optimal Peer Memory Direct performance:

1. Verify the kernel modules ionic_driver, ionic_rdma, and ib_peer_mem.ko) are loaded and of the correct version.
   Note: Some Ubuntu kernels have built-in ib_peer_mem support and don't require AMD's module. The kernel driver Makefile will detect and build accordingly.
2. Ensure the AMD GPU driver (amdgpu.ko) is loaded.
3. Confirm PCIe Access Control Service (ACS) is disabled on the PCIe switch connecting the NIC and GPU. ACS must be disabled to enable PCIe peer-to-peer transactions between GPU and NIC, as enabling ACS can degrade performance.
4. Verify IOMMU is disabled or set to Pass-Through (PT) mode.
5. Confirm the following standard InfiniBand commands run successfully (included in the infiniband-diags package, available through your OS distro's package manager):
   - ibstatus

- ibv_devinfo -vvv
- ibdev2netdev

6. Confirm Pensando Pollara configuration settings for RDMA/RoCE defined above are complete, and PCIe Relaxed Ordering. (nicctl show qos)
7. Ensure the Pensando Pollara interface link is active and operating at the correct speed, verified using:
   - ibstatus
   - ethtool <ifname>
8. Confirm the NIC interface has an assigned IP address, visible as GID 3 (IPv4 or IPv6) through the commands:
   - rdma link show
   - ibv_devinfo -vvv
   - dfadfasdfad
   - ibv_devinfo -vvv -d <roce_interface_name>

8. Set the host interface MTU size to 9000 bytes for maximum throughput.
9. Configure the Ethernet switch port connected to the Pollara with an MTU size of at least 9000 bytes.
10. Validate the PCIe slot for the Pollara reports the correct PCIe generation speed and lane width:
    - lspci -vvv -s <B:D:F>

12. Disable firewalls on communicating hosts if they block RDMA connection setup.
13. Confirm the Linux dmesg logs contain no NIC- or GPU-related errors.

For More Information:

https://www.supermicro.com/en/products/networking
https://www.supermicro.com/en/products/aplus

AMD

For more than 50 years AMD has driven innovation in high-performance computing, graphics and visualization technologies. Billions of people, leading Fortune 500 businesses and cutting-edge scientific research institutions around the world rely on AMD technology daily to improve how they live, work and play. AMD employees are focused on building leadership high-performance and adaptive products that push the boundaries of what is possible. For more information about how AMD is enabling today and inspiring tomorrow, visit the AMD (NASDAQ: AMD) website, www.amd.com

SUPERMICRO

As a global leader in high performance, high efficiency server technology and innovation, we develop and provide end-to-end green computing solutions to the data center, cloud computing, enterprise IT, big data, HPC, and embedded markets. Our Building Block Solutions® approach allows us to provide a broad range of SKUs, and enables us to build and deliver application-optimized solutions based upon your requirements.