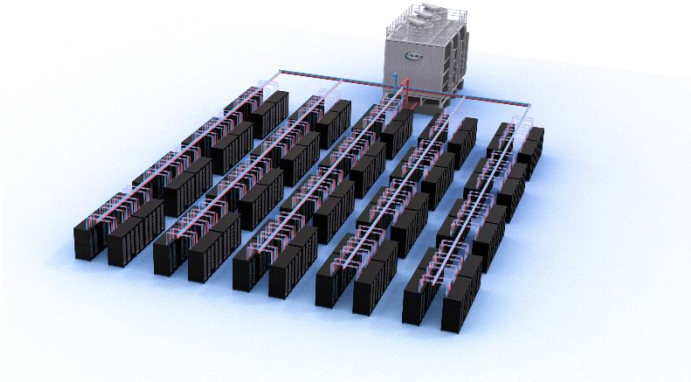
VALIDATED DESIGN

BUILDING A TENANT-AWARE AMD INSTINCT™ MI300X/MI325X/MI350X/MI355X CLUSTER FOR AI WORKLOADS WITH AN INTEGRATED SUPERMICRO SOLUTION

A Fully Validated Solution for Delivering an Optimized Architecture for Providers and Customers Building an Infrastructure with AMD Instinct™ MI300X-MI355X Accelerators

TABLE OF CONTENTS



Executive Summary

Supermicro is a trusted technology partner at the forefront of the data center industry. It offers optimal Total Cost of Ownership (TCO) solutions across various domains, including Data Centers, AI, and HPC applications. Additionally, Supermicro is a leading supplier of systems for Generative AI, collaborating with partners throughout the technology stack. As such, customers rely on Supermicro for tested designs and solutions that simplify the complexities of these deployments. While the components in the AI cluster typically deliver very high performance, other considerations extend beyond raw speeds and feeds to enable optimal use of these resources in each solution.

This paper will present a pre-validated GenAI solution that can be replicated, expanded, reduced, and operated optimally. By this, we mean that the scale of this solution can be easily adjusted to accommodate various cluster sizes. The solution discusses managing implementations of 16/32/64/128… up to a maximum of 1024 nodes with this validated design. This design is a full-stack solution from Supermicro that can be deployed at a customer site or obtained as a pre-built solution comprising a set of connected and configured elements, which are burned-in before shipping to the customer for rapid deployment. We base this design on the AMD MI300X-MI355X solution, utilizing Supermicro-optimized servers, with all network cabling and switching integrated and configured for optimal performance upon delivery. This document will address numerous design and engineering data points to help the reader understand the key components.

Glossary of Terms

Accelerator	The actual xPU performing calculations
SU	Scale Unit
RCCL	Radeon 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 handle, 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 handle 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

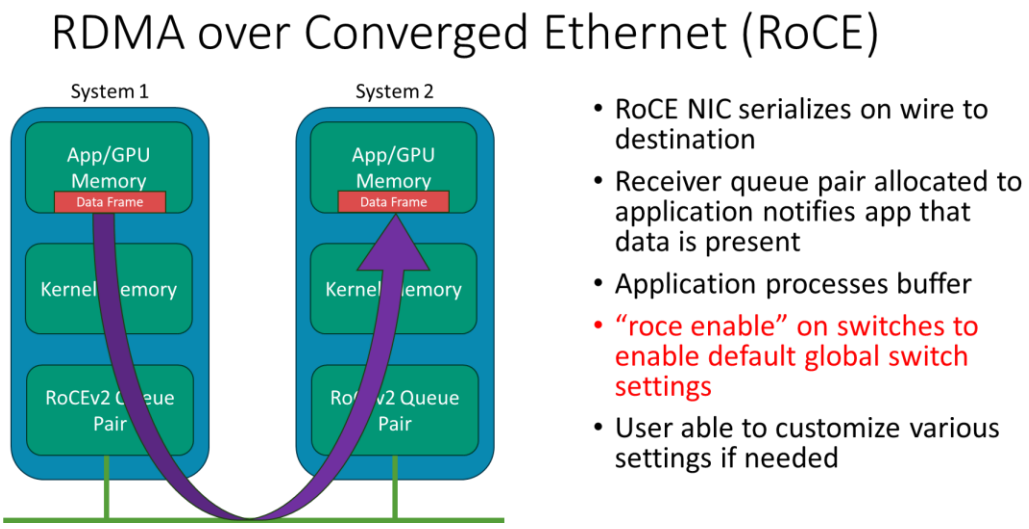


Figure 1 - Remote Direct Memory Access (RDMA)

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

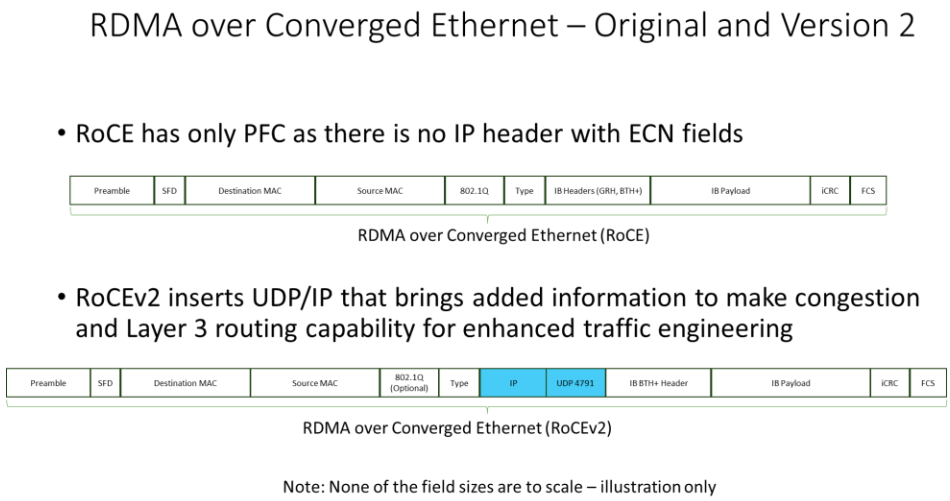


Figure 2 - RoCE and RoCEv2 Packet Structure

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.

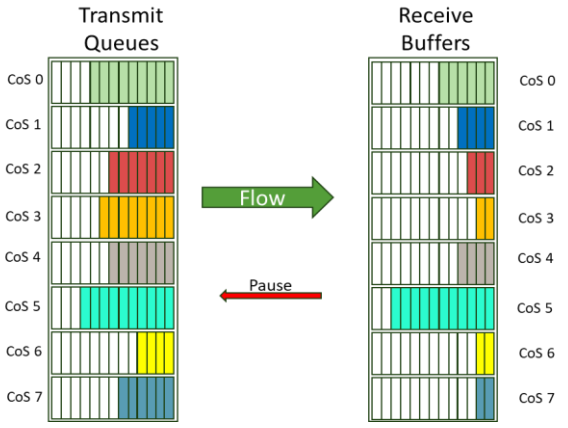


Figure 3 - Per-Priority Flow Control via Pause Frames

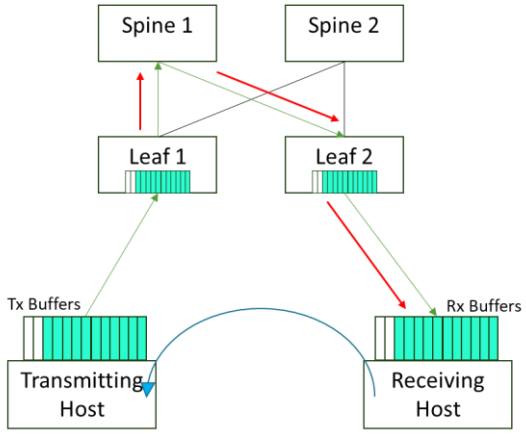


Figure 4 - Explicit Congestion Notification via CNP

Several core individual enablers exist to meet these goals, including Per-Priority Flow Control (PFC), as shown in Figure 3, which allows traffic to be paused by class of service. Another key aspect of managing congestion to ensure lossless behavior is depicted in Figure 4, 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.

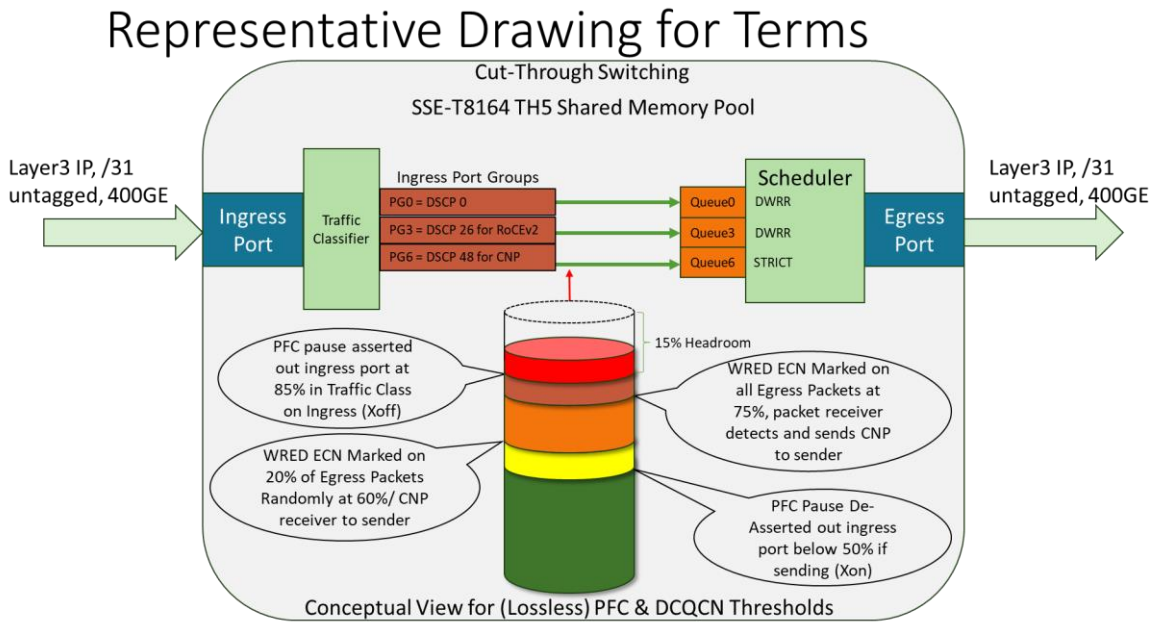


Figure 5 - View of Buffering, Classification, Queueing, and Scheduling in SSE-T8164S

The mechanisms to monitor, detect, and signal all of this 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 as a whole, 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 Dynamic Load Balancing (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 such as 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.

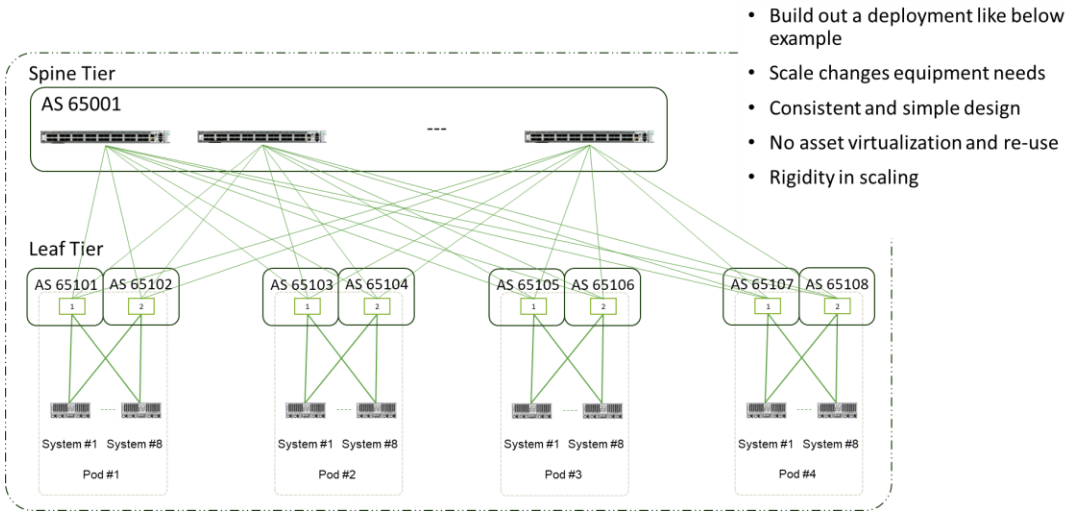


Figure 6 - BGP example from Meta for Ethernet Leaf-Spine architecture for maximum availability and traffic balancing

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 a monetary break-even point (and profit beyond that) 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, along with 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.

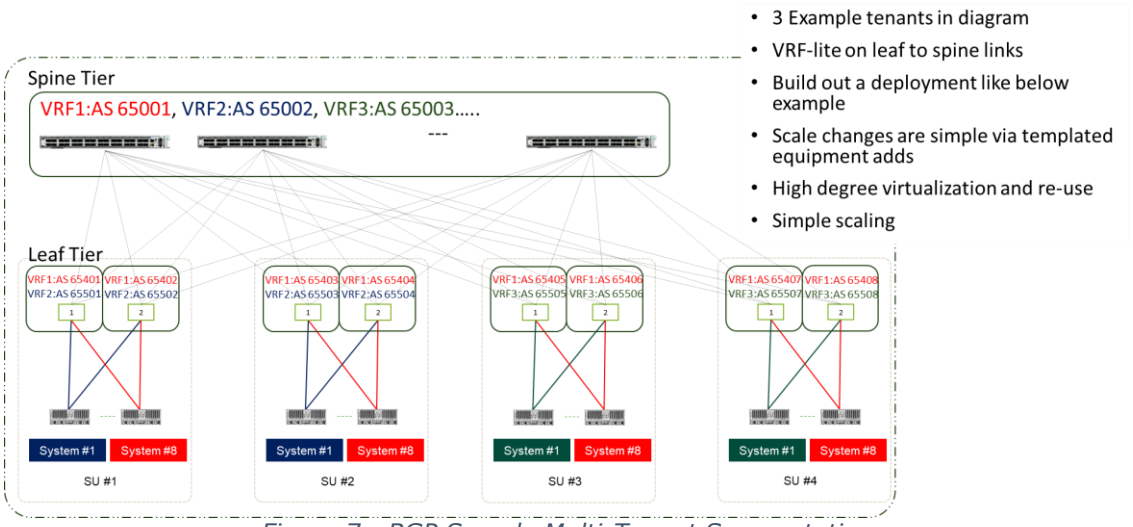


Figure 7 - BGP Sample Multi-Tenant Segmentation

Managing Incremental Growth to the Accelerator Cluster

Collaborating on the previous discussion about building a shared cluster utility, which facilitates rapid onboarding, offboarding, and adaptive business models for customers, there is often 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 Scale 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.

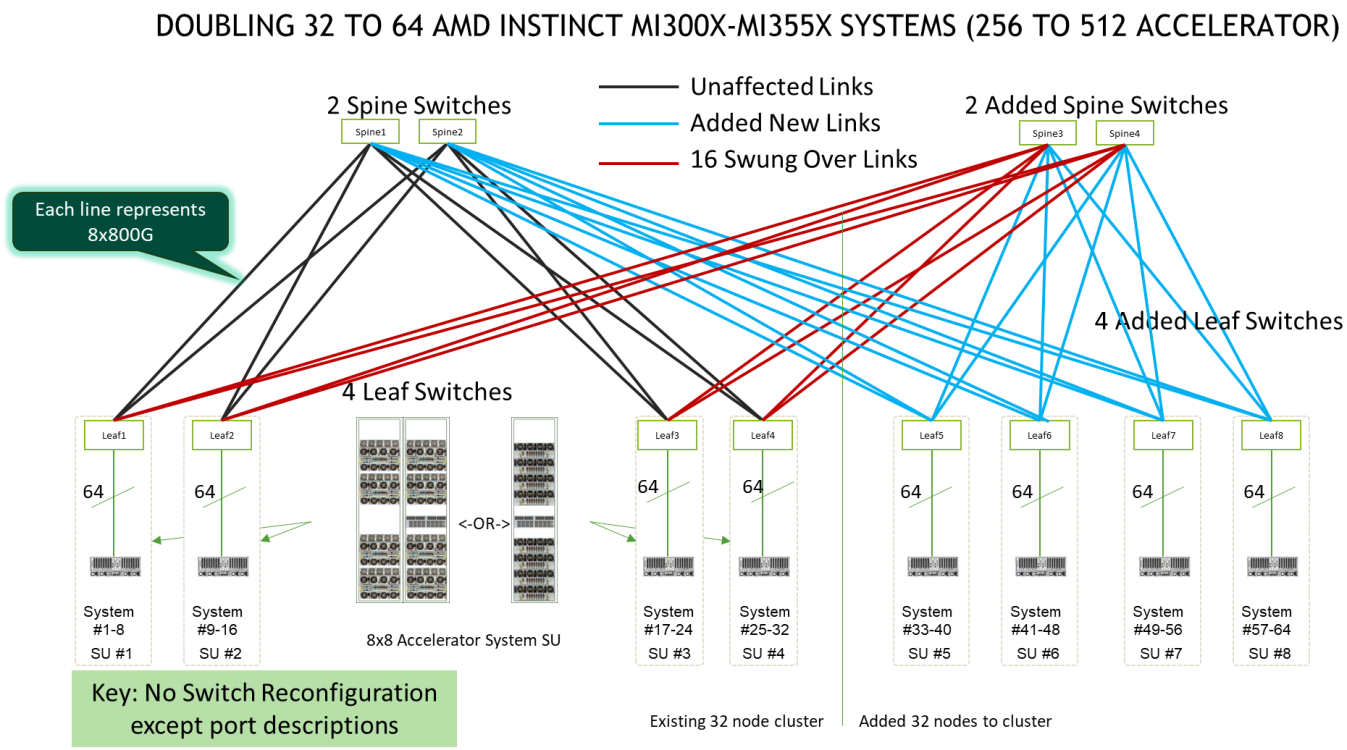


Figure 8 - Doubling the solution cluster to a 64-node example

Validated Design Equipment and Configuration

MI325X Server Utilized in a Validated Design

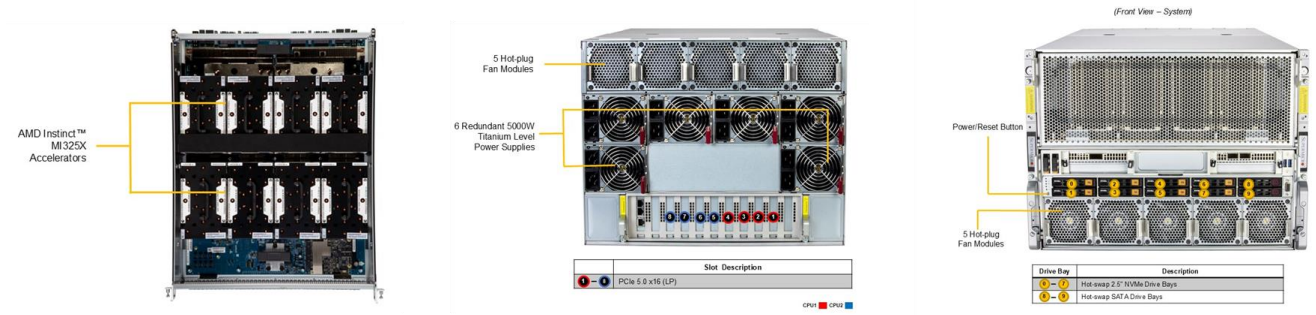


Figure 9 - Supermicro AS-8126GS-TNMR

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.

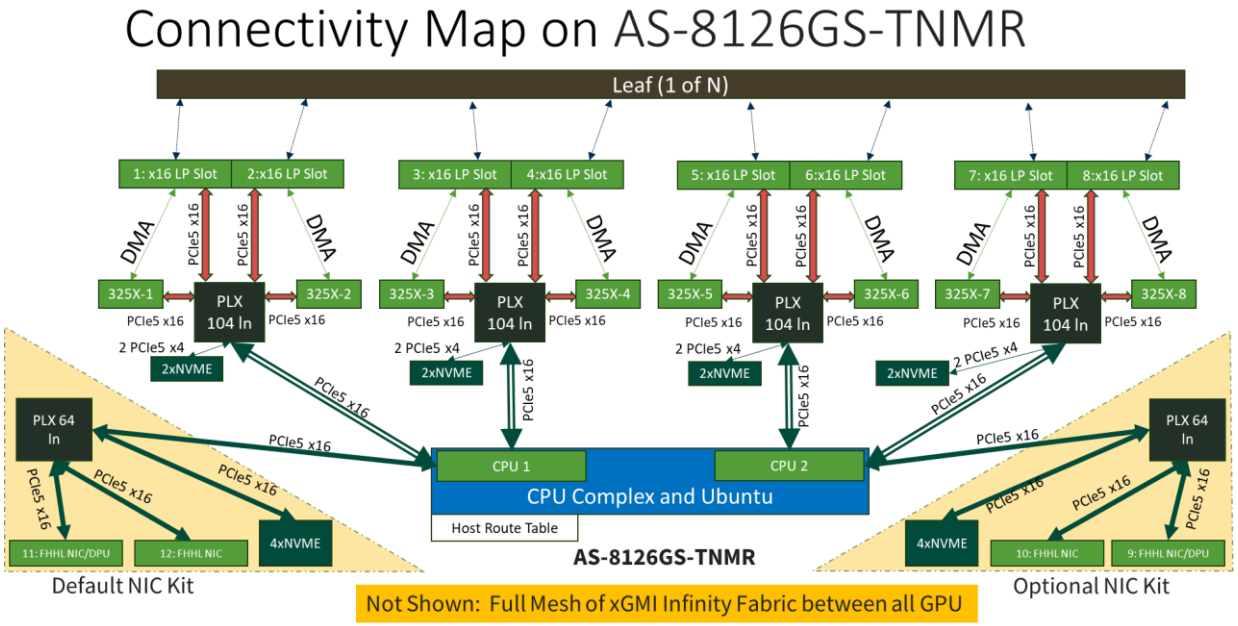
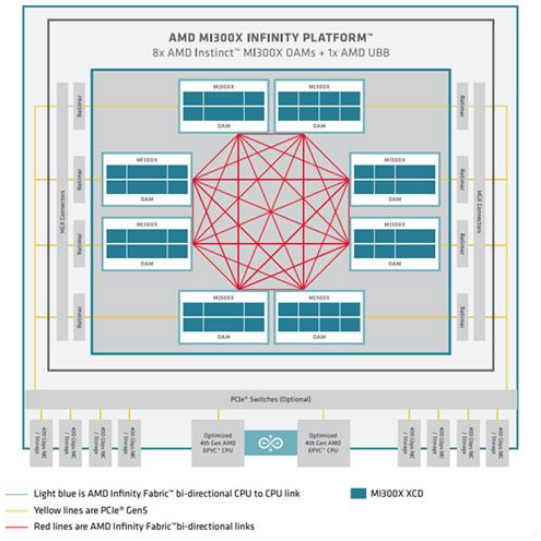


Figure 10 - Block Diagram of AS-8126GS-TNMR

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 generally permit firewalled internet access for tasks such as Linux package operations, driver updates, tenant access, etc. Notably, the 1GE copper BMC access is not shown in the above diagram.

MI300X-MI355X Internal Fabric Basic View



Gen5 PCIe on all. x16 means 500Gbps possible

Figure 11 - MI300X-MI355X Infinity Fabric

Figure 11 above gives a simple view of the internal links and the MI300X-MI355X OAM. All accelerators are directly meshed for optimal performance.

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

AOC-S400G-B1C RDMA Ethernet NIC for RoCEv2 Fabrics

Figure 12 - Supermicro AOC-S400G-B1C RNIC

Supermicro develops the RDMA NIC (or RNIC here) based on Broadcom Thor2 (BCM957608) silicon. This PCIe Gen5 x16 adapter (standup PCIe in this SVD, but an AIOM for OCP Mezzanine is also offered) is plugged into eight LP slots on the server above. These then provide a direct set of 16 lanes of PCIe Gen5 for each accelerator.

SSE-T8164 Ethernet Switch

Supermicro's designs for AI networks are consistent across many of our partner accelerator solution vendors. We design, build, and market multiple 400GE and 800GE switches to support these designs. The foundational technical software features and functionality vary among different network vendor solutions; however, Supermicro employs the industry-standard Software for Open Networking in the Cloud (SONiC). All the Ethernet features and functionalities referenced in this paper are included in SONiC. To become more familiar with SONiC, we encourage utilizing tutorials and deploying a virtual switch environment. This document provides an optimized SONiC configuration for our example cluster. Increasing numbers of customers and partners are deploying Supermicro switches due to the short lead times, the desire for a fully validated solution rack from fewer component vendors, and the strength in all things AI we offer. We will provide a very brief overview below:

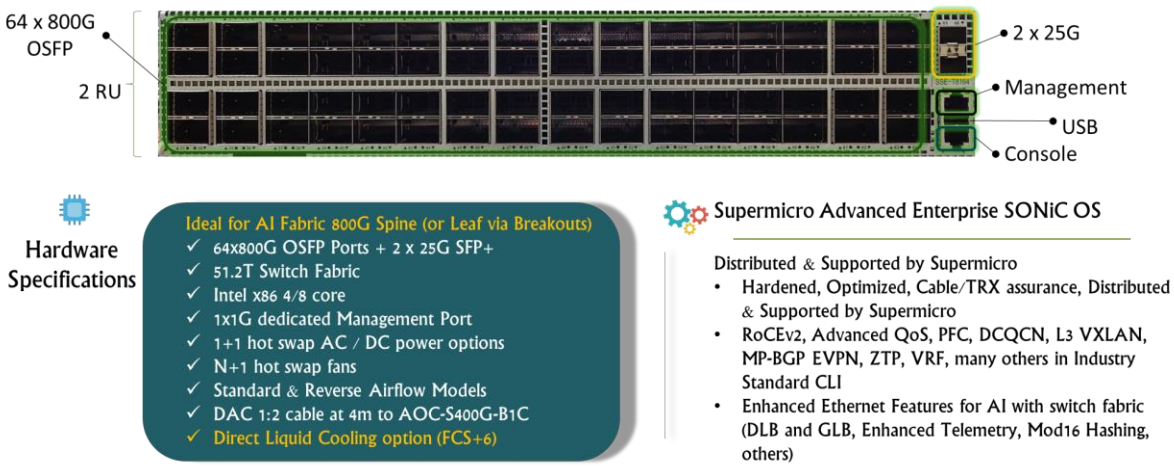


Figure 13 - Supermicro SSE-T8164 Ethernet Switch

The SMC SSE-T8164 Ethernet switch serves as both the leaf and spine in our AI network cluster design. In this design, Supermicro will utilize direct 800G presented as 2x400GE links between all elements via 400GE breakouts for connecting to the RNICs on our MI300X-MI355X systems. Supermicro has similar designs for other accelerator offerings and a roadmap for regular updates to this and different designs. Please contact your Supermicro sales team to learn more. Each AMD MI300X-MI355X connects to a 400G NIC interface on the 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 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 MI300X-MI355X 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 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.

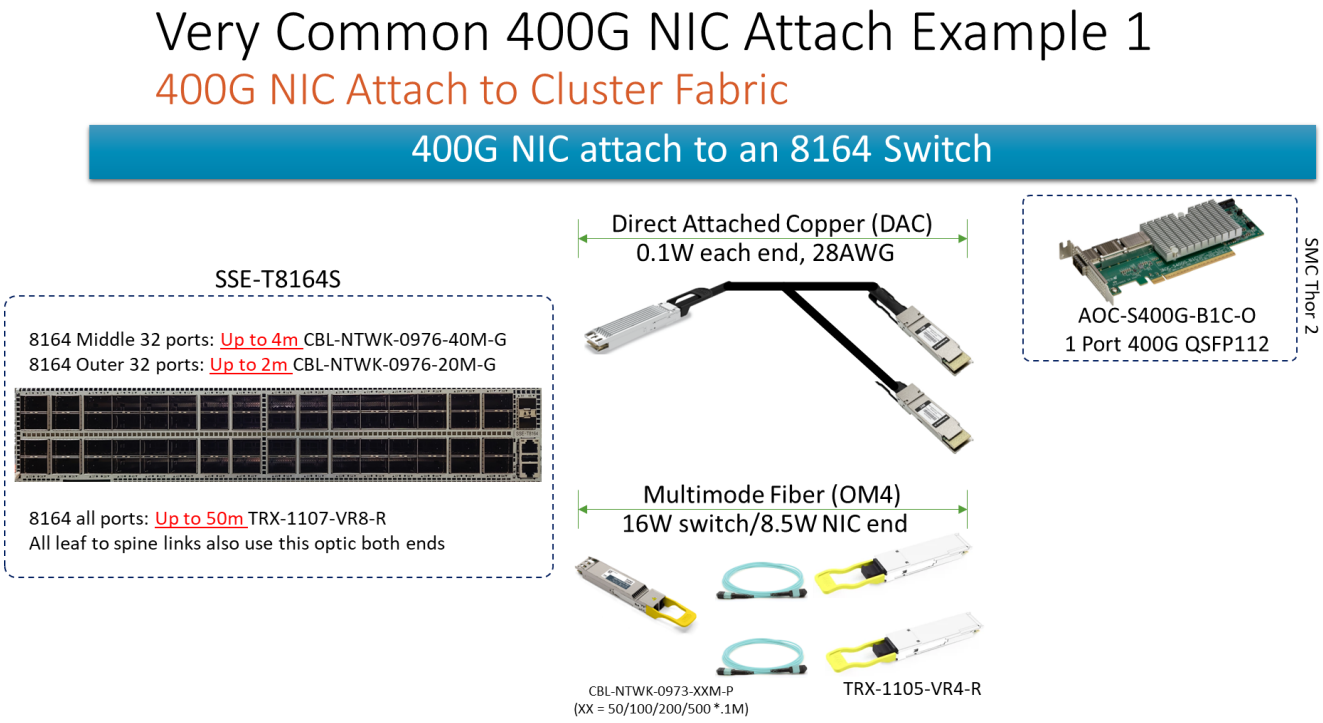


Figure 14 - Common DAC and Fiber leaf to RNIC Links for East-West Traffic

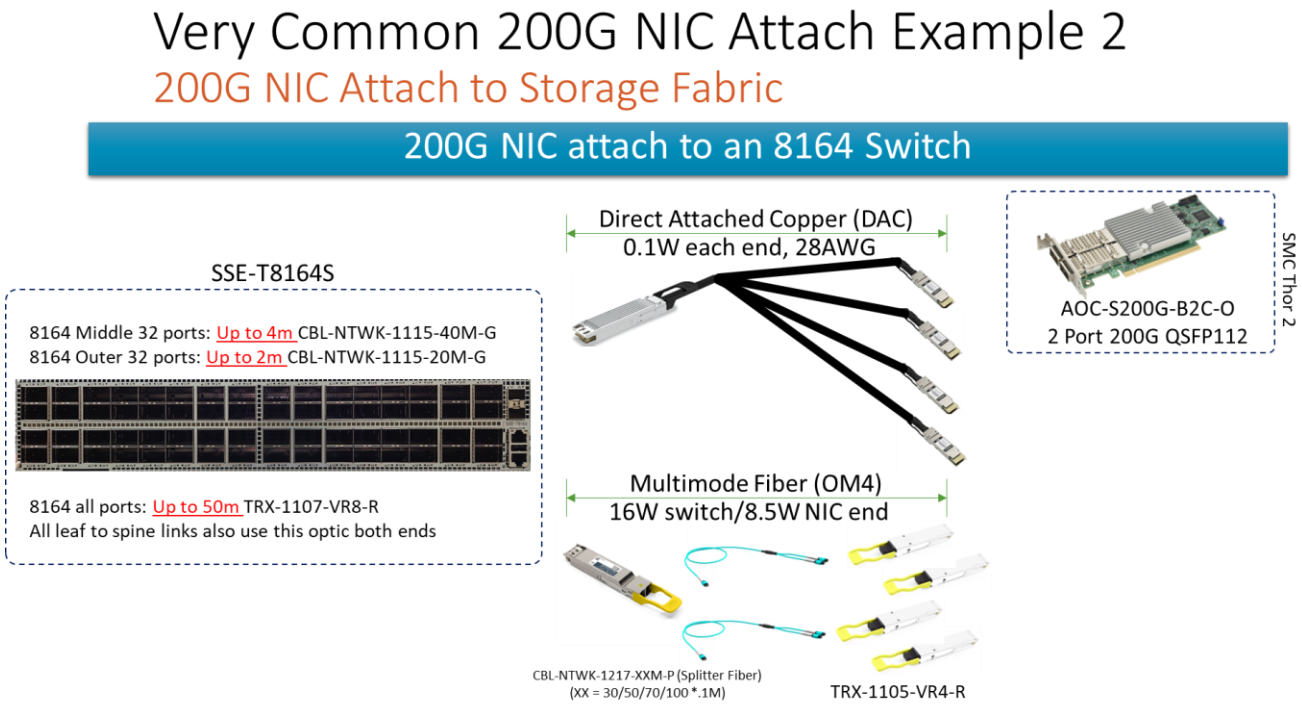


Figure 15 - Common DAC and Fiber Leaf to RNIC Links for North-South Storage Traffic

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)

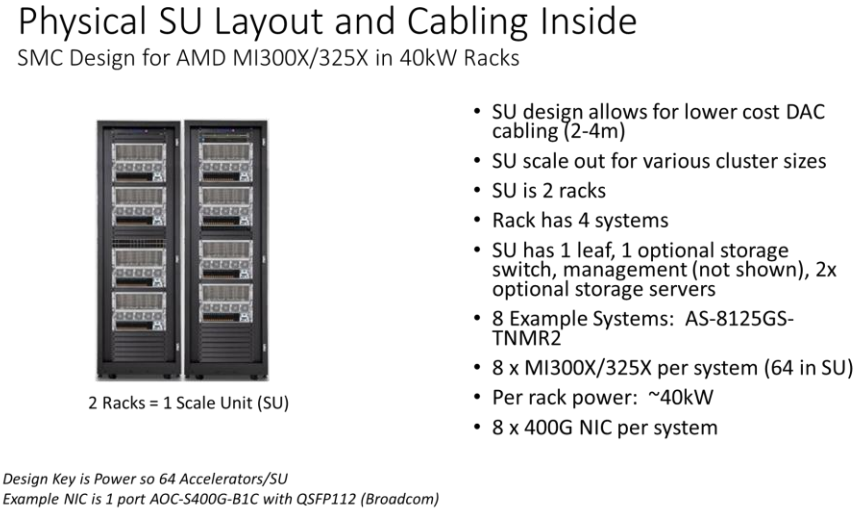


Figure 16 - Supermicro Air Cooled AI SU with MI300X-MI355X Servers and Accelerator Fabric Switches

Figure 17 below illustrates the same SU design, but it becomes denser by implementing Direct Liquid Cooling for 8 hosts within a single rack with a CDU. The core system design now requires only 4RU to accommodate 8 MI300X-MI355X Accelerators, while consistently performing alongside the air-cooled version of this SU.

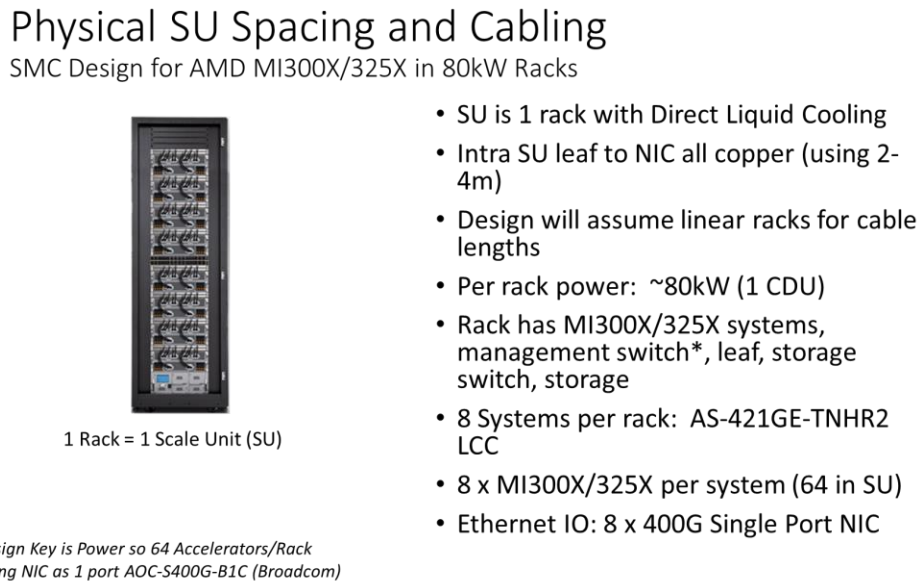


Figure 17 - Supermicro DLC AI SU with MI300X-MI355X Servers and Accelerator Fabric Switches

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.

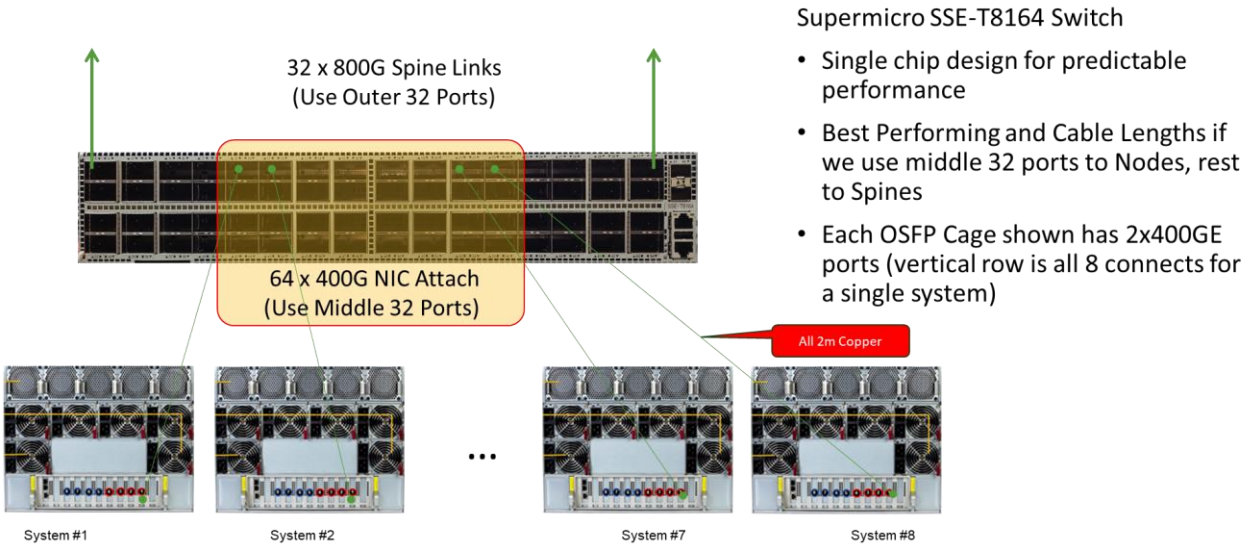


Figure 18 - System to Leaf Links

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, depending on whether 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, causing these accelerators to wait for a final data send (tail latency) to ensure all workers are in sync before proceeding 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 Scale Unit (SU) connect to the same leaf. In this design, we will utilize a system of 64 accelerators, tightly controlling latency to achieve optimized performance with more than 8 accelerators while reducing 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 utilize copper for the many advantages mentioned earlier in this document, but also localize a set of nodes and their accelerators into a group based not 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.

Configuring Slurm for Optimal Scheduling

An efficient cabling alternative to a rail-optimized design

- Scheduling on local high bandwidth domain always most performant (i.e. Infinity Fabric), often need to scale higher
- Uncoordinated network will lead to any to any elephant flows over various leafs/spines/superspines with widely different perforamance
- Slurm 24.05 with the topology/block plugin allows for hierarchical scheduling in these jobs
- Keeps the advantages of scheduler knowing resource adjacency to minimize elephant flow scale-out switching hops
  - Same goal as rail-optimized designs – but alternative provides direct control vs. static local rank
  - Example shows 2 DLC racks, simple scale up to 8 DLC racks (64 systems / 512 MI325X) with 8 leafs/4 spines
- Slurm sbatch command arguments to control:
  - --exclusive=topo Means no other jobs are place on the allocated block
  - --segment=YY YY is a number less than planning block size, to parse over many blocks (so flows could need to cross higher level network tiers)
  - --NXX XX is the number of workers collectively executing this step of the job

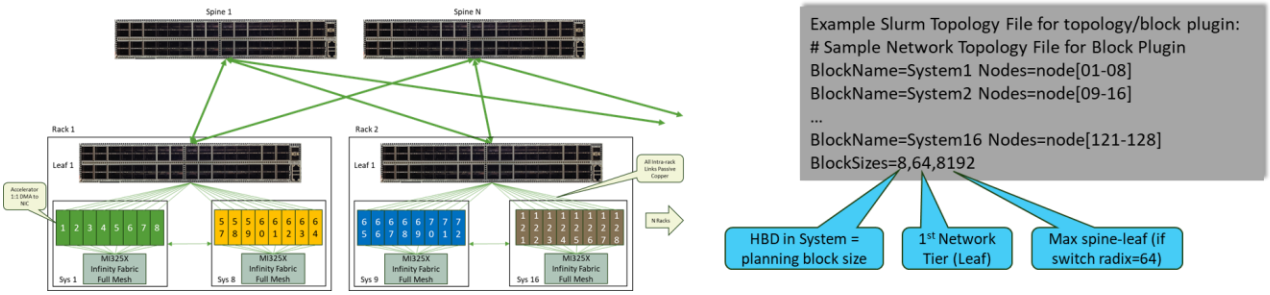


Figure 19 - Slurm Topology/Block Plugin

Supermicro Validated MI300X-MI355X Design: 4 Scale Units for 256 MI300X-MI355X

Figure 20 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. Suppose we maintain the design constraint to a maximum of two tiers. In that case, this solution with the specified product set can scale up to a maximum of 4,000 MI300X-MI355X accelerators in the cluster shown below.

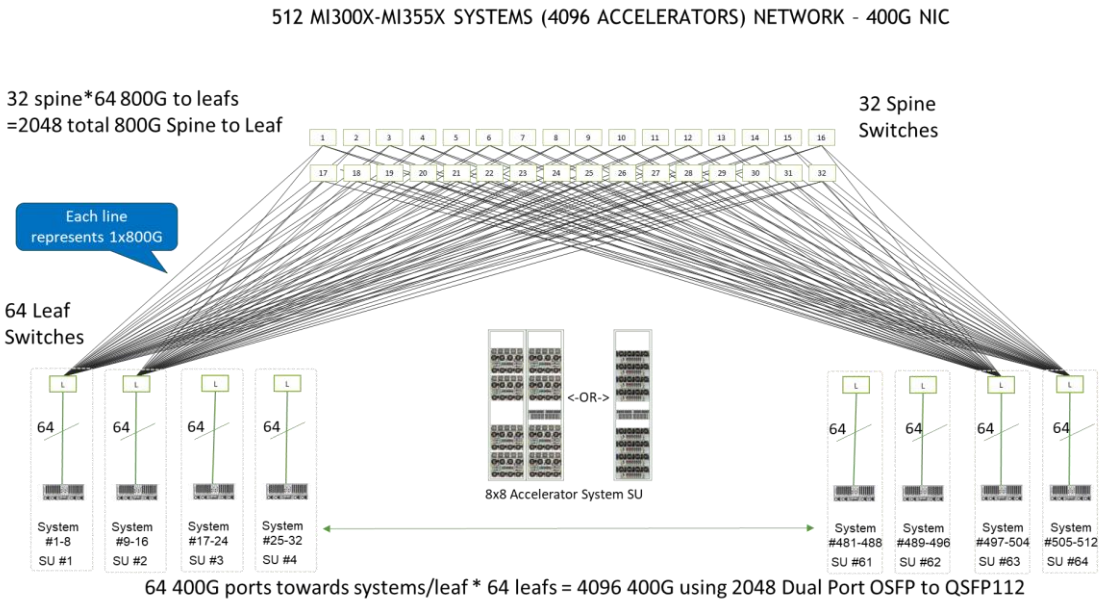


Figure 20 - Maximal 2 Tier Fabric Cluster or This Solution

If even higher-scale numbers are needed, we can scale up to a 3-tier topology using a superspine and retain full non-blocking performance on the cluster, as shown in Figure 21 below. We can go even further if needed by adding more superspine planes.

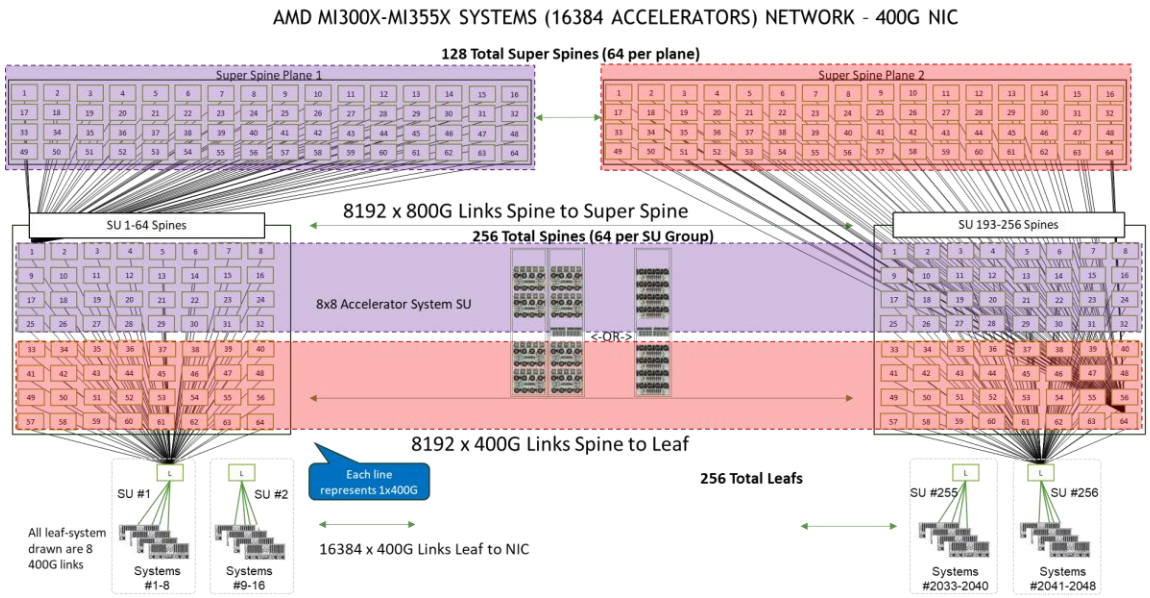


Figure 21 - Sample 3-Tier Fabric Cluster for this Solution (Can scale much higher)

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 their work, multiple large flows are established to the parallelized group of peers working together. These flows are classified as elephant flows for large sizes, but they are not steady and constant, as 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 illustrates some common alternatives and a method for utilizing different bandwidths on the node-to-leaf versus the leaf-to-spine paths, thereby reducing the likelihood that higher-utilized flows will exhaust their bandwidth before completing rebalancing with DLB. This delays the utilization of PFC and ECN signaling, resulting in overall better performance.

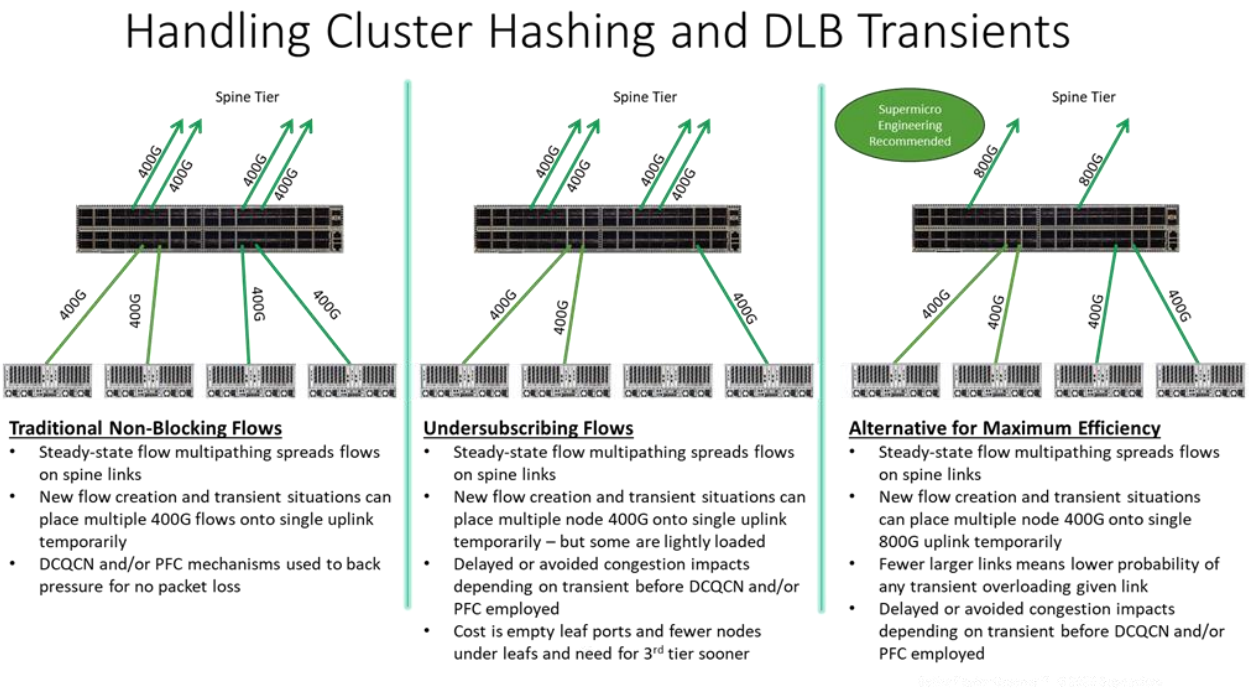


Figure 3 - Transients in Hashing and DLB adjustments

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 use ECMP alone, we often desire a power of 2 in the number of spines, allowing traffic to be evenly balanced over these links. In variations of these validated designs (i.e., adding an odd number of SU's, etc.), you would see cases where the numbers of spines and links are not always guaranteed to be a power of 2. 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 features modulo 16 hashing, which can produce an even set of probabilities over even non-power-of-2 links. While that helps, we can take it a step further with the deployment of Dynamic Load Balancing (DLB), which we discussed earlier. 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 hash to a "per-flowlet" hash we talked about 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 configure a port-channel from leafs to spines, rather configure a parallel set of unnumbered BGP and let the DLB perform this optimal action.

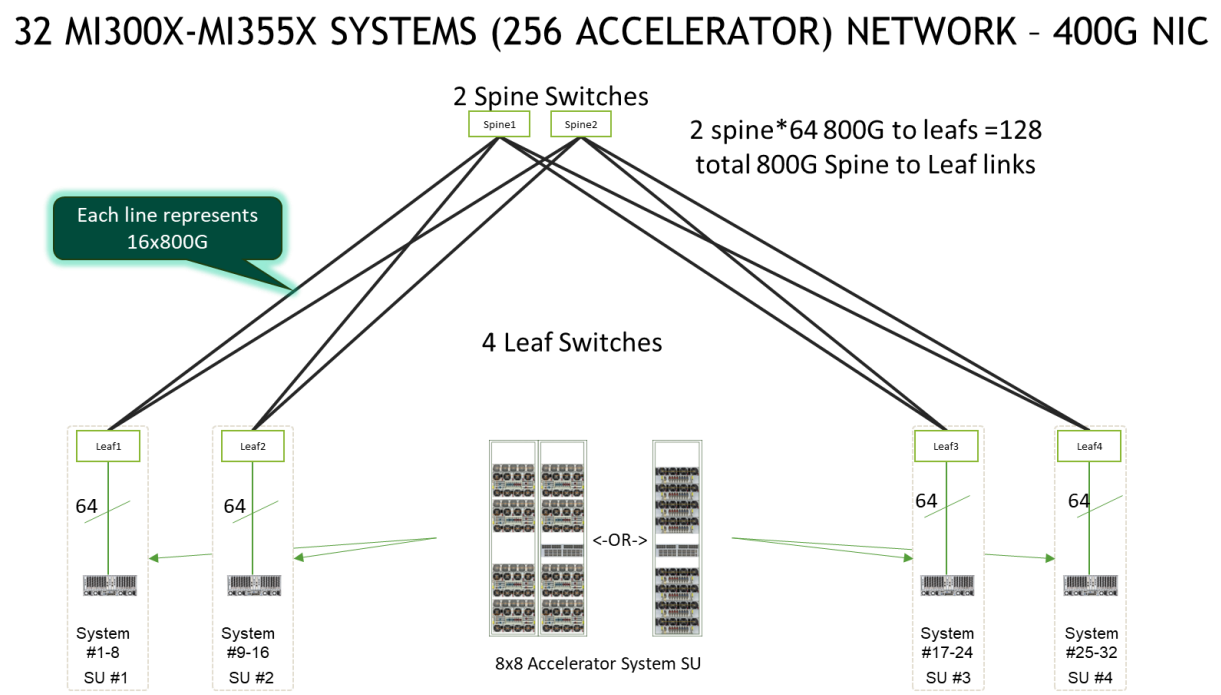


Figure 22 - 32 system 256 Accelerator design with 400G to each MI300X-MI355X

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 corona_node1 through _node16, heineken_node1 through _node8, and budweiser_node1 through _node8
- Simple cluster fabric IP addressing illustrates (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 the leaf boundary if tenant nodes can be anywhere (but the general goal is to keep tenants' 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 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 to be AOC-S400G-B1C 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
- VLAN 61 (Corona), 62 (Heineken), and 63 (Budweiser) are the tenant VLAN 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 1/9/1-1/24/2 for Corona nodes, Eth 1/41/1-1/56/2 for Heineken nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/7 & Eth1/25-1/32, while uplinks to Spine2 are Eth 1/33-1/40 & Eth 1/57-1/64 all at 800G
- Leaf2 has both Corona and Heineken on multiple interfaces (Eth 1/9/1-1/24/2 for Corona nodes, Eth 1/41/1-1/56/2 for Heineken nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/7 & Eth1/25-1/32, while uplinks to Spine2 are Eth 1/33-1/40 & Eth 1/57-1/64 all at 800G
- Leaf3 has both Corona and Budweiser on multiple interfaces (Eth 1/9/1-1/24/2 for Corona nodes, Eth 1/41/1-1/56/2 for Budweiser nodes) and these tenant nodes only see to each other local and over entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/7 & Eth1/25-1/32, while uplinks to Spine2 are Eth 1/33-1/40 & Eth 1/57-1/64 all at 800G
- Leaf4 has both Corona and Budweiser on multiple interfaces (Eth 1/9/1-1/24/2 for Corona nodes, Eth 1/41/1-1/56/2 for Budweiser nodes), and these tenant nodes only see each other locally and over the entire fabric L3. Uplinks to Spine1 are Eth 1/1-1/7 & Eth1/25-1/32, while uplinks to Spine2 are Eth 1/33-1/40 & Eth 1/57-1/64 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:

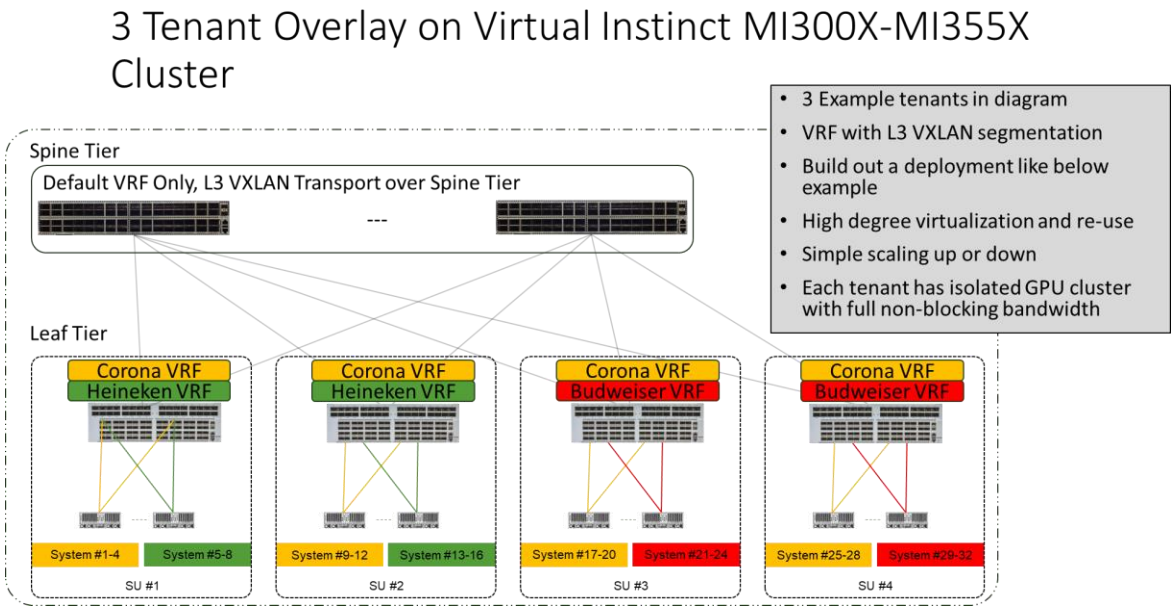


Figure 23 - 3 Tenant View on Cluster

Performance of a Validated Design

This section will present some sample RCCL testing numbers from the Supermicro Performance Analysis Center. One important note: while the design showcased here is validated with all components for correct operation, the performance tests were conducted on the available hardware at the time of this report. We performed these tests with four nodes, each equipped with eight MI300X, for a total of thirty-two. Two of the nodes were connected to one leaf, while the other two were connected to an alternate leaf. The two leafs were linked to a common spine to realize the three stages in this design.

Interpretation of These Results

With these standard testing toolsets for collective communications, a significant amount of information is displayed, including portions relevant to the network fabric. At the highest level is the meaning of out-of-place and in-place, which means we care about how the buffers are manipulated within the testing tool (in-place means same virtual address for both source and destination buffers, out-of-place means different virtual addresses – so some copying can be saved internally). There is typically little difference in those as the buffers become larger. Next, we determine the applicable size and element counts, which are crucial for extracting latency on the smallest element sizes. The elephant flows in our design are not as impacted by this, so we will not spend time there.

The algbw is where we start getting the most interest, and in the in-place, as more of the test is tied to the performance of the fabric. This value represents the algorithm's total data size divided by the time it takes to complete the collection. Lastly is the busbw, which is simply a mathematical calculation that then takes the total number of ranks in the test into account (i.e., this itself is not directly impacted by fabric usage, rank counts, or test tool load spread on the nodes – the algbw is, and this is

a derivative equation). This calculation is dependent on the collective and the most impactful metric to track overall fabric design performance. A couple of examples are below:

All_reduce: busbw = algbw * (2*(#ranks-1)/#ranks)
All_gather: busbw = algbw * (#ranks-1)/#ranks
All_to_all: busbw = algbw * (#ranks-1)/#ranks

Actual Performance Results

To help establish a baseline, we will start with the RCCL performance of various collectives to compare it to the scale-out fabric results.

Single Node Results

Nodes	Test	Size	# Ranks	Loop	Ranks/Node	MI300X NW busbw (GB/s)
1	All_reduce	16GB	1	100	8	322.38
	All_gather					318.29
	Alltoall					303.73
	Alltoallv					182.07
	Broadcast					321.30
	Gather					340.03
	Reduce					293.28
	Reduce_scatter					320.02
	Scatter					330.68
	Send_recv	8GB				48.08

2 Nodes Connected to Single Leaf Results

Nodes	Test	Size	# Ranks	Loop	Ranks/Node	MI300X NW busbw (GB/s)
2	All_reduce	16GB	16	100	8	280.08
	All_gather					261.69
	Alltoall					62.05
	Alltoallv					44.97
	Broadcast					247.2
	Gather					70.46
	Reduce					274.44
	Reduce_scatter					289.94
	Scatter					70.74
	Send_recv	8GB				12.02

2 Nodes Connected over a Spine Path Results

Nodes	Test	Size	# Ranks	Loop	Ranks/Node	MI300X NW busbw (GB/s)

2	All_reduce	16GB	16	100	8	278.88
	All_gather					261.77
	Alltoall					61.40
	Alltoallv					44.86
	Broadcast					246.53
	Gather					69.92
	Reduce					275.6
	Reduce_scatter					287.92
	Scatter					70.81
	Send_recv	8GB				11.97

4 Nodes Connected as 2 per Leaf Results

Nodes	Test	Size	# Ranks	Loop	Ranks/Node	MI300X NW busbw (GB/s)
4	All_reduce	16GB	32	100	8	277.7
	All_gather					261.79
	Alltoall					48.27
	Alltoallv					39.03
	Broadcast					241.8
	Gather					60.38
	Reduce					230.95
	Reduce_scatter					288.93
	Scatter					62.09
	Send_recv	8GB				11.99

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 utilize a file system that leverages the NVMe capabilities present on the AS-8126GS-TNMR, as shown in the block diagram above, which illustrates a direct connection via the PLX switches.

While the SATA drives in slots 8 and 9 are ideal for the Ubuntu installation, the other eight NVMe drives are optimized for parallel storage that is placed next to the accelerator, allowing RDMA to the storage directly. Some solutions also allow for an adjacent set of flash local to each rack. To scale this, we recommend a dedicated storage switch pair and Supermicro storage servers with a suitable high-performance stack, such as Weka, DDN, Vast, or many other excellent products available. A software solution can be configured to cluster this storage across nodes with tenant-aware namespaces. For added security, we could also employ VRF technology on this portion of the cluster. In Figure 24 below, we illustrate the connectivity from a north-south networking perspective.

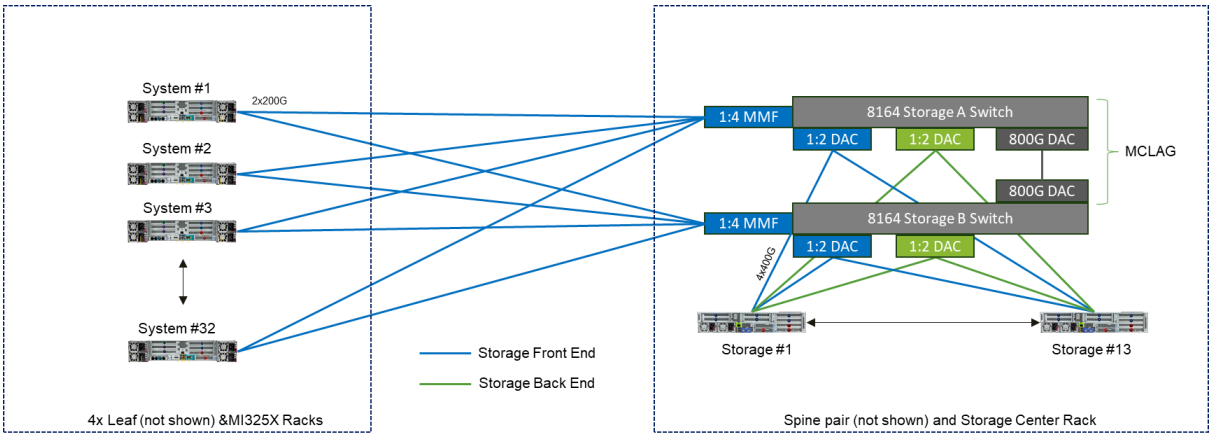


Figure 24 - Storage Fabric and Targets

As an aside, these solutions will have two more networks connected in the cluster. Firstly, we have an out-of-band management setup where a 1GE Cat6 cable connects to the server BMC for routine setup and monitoring operations. It is on that connection that 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, which outlines the device configuration for this relatively straightforward cluster of 4 leaves and two spines, you will see the scale and detail in the configuration for an Ethernet fabric in 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 mandates 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, and BMC networks in a coordinated manner go 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 of bringing all these together and expect announcements to follow as we are all

forced to tackle these tasks. Configuration of these elements from a centralized solution will eliminate the need for direct usage of individual device CLI's, APIs, and web interfaces.

An upcoming paper sharing a new Supermicro controller product in this space will demonstrate how this and other validated designs with multiple accelerator solutions can now be integrated not only into the topology definition and deployment, but also to include best practice optimizations based on that specific validated solution. Once the equipment arrives on-site, the installation and cabling are complete. The tooling can then validate the infrastructure and minimize the time to revenue. To facilitate an even shorter-term deployment, the next section, which includes 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 benefit from direct manufacturer expertise, reduced operational overhead, and a single point of accountability, ensuring a streamlined procurement, deployment, and support experience that maximizes ROI.

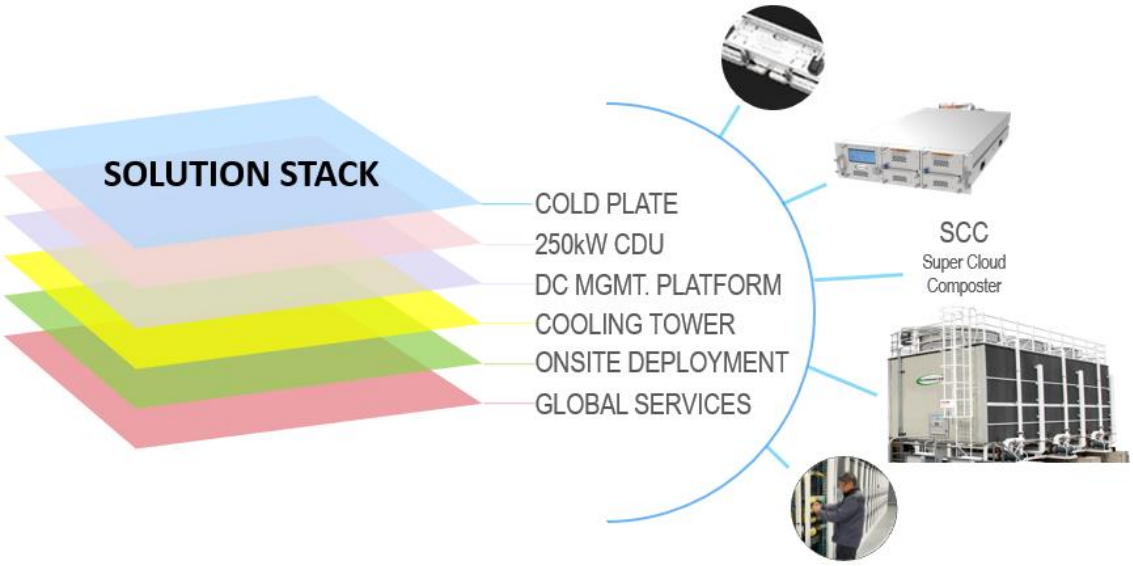
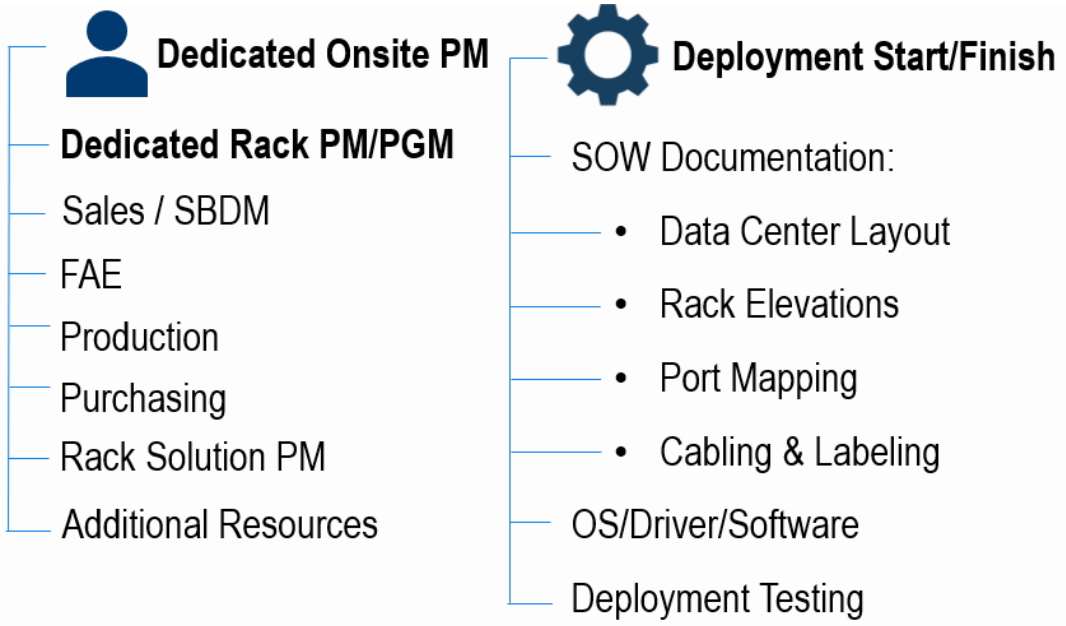


Figure 25 - Example Rack Solution Stack

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 utilize 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 tooling to automate these configurations, where the built configurations are injected into the switch via methods such as ZTP, Ansible, Puppet, gNMI, and gRPC, as well as other available tooling. These greatly reduce the probability of human error on data entry to these devices as we scale these solutions.

One important first step is to configure breakout ports on each switch before placing actual configurations there. Many elements will not be correct if we attempt to do the breakouts later. One good way is to configure all ports as 2x400G breakout mode via a script (or sequential manual command below):

SpineN/LeafN(config)# interface breakout port X/1 mode 2x400G (where X ranges from 1 to 64)

A Python script is available from the Supermicro switch team that can solicit the IP address of a new switch, gather login details, and configure the ports in a single operation. A screenshot of the script in operation is below, where the user selects the breakout mode desired (2x400G for this validated design) and the ports to apply it to (Eth1/9 through Eth1/24 and Eth1/41 through Eth1/56, as 2x400G is used for only leaf to systems).

```
admin@sonic:~$ show int stat
  Interface                                Lanes    Speed    MTU    FEC    Alias    Vlan    Oper    Admin    Type    Asym PFC
-----------  -------------------------------  -------  -----  -----  -------  ------  ------  -------  ------  ----------
  Ethernet0                  1,2,3,4,5,6,7,8     800G   9100     rs   Eth1/1  routed    down     down     N/A         N/A
  Ethernet8          17,18,19,20,21,22,23,24     800G   9100     rs   Eth1/2  routed    down     down     N/A         N/A
 Ethernet16          33,34,35,36,37,38,39,40     800G   9100     rs   Eth1/3  routed    down     down     N/A         N/A
 Ethernet24          49,50,51,52,53,54,55,56     800G   9100     rs   Eth1/4  routed    down     down     N/A         N/A
```

Figure 26 - Default Interface Lanes - 1x800GE

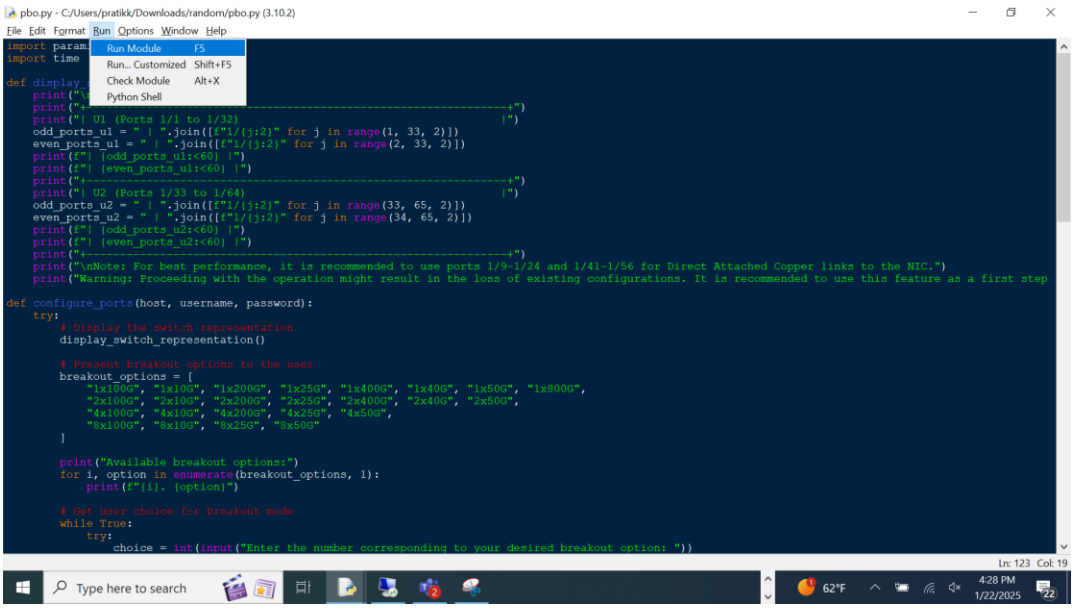


Figure 27 - Running Script from IDE

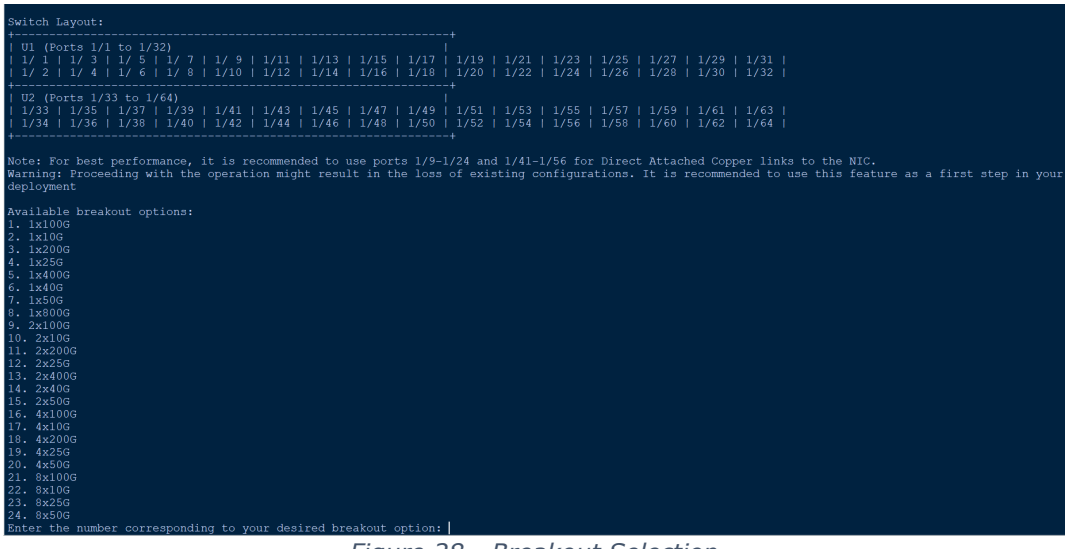


Figure 28 - Breakout Selection

```
admin@sonic:~$ show int stat
  Interface                            Lanes    Speed    MTU    FEC     Alias    Vlan    Oper    Admin    Type    Asym PFC
-----------  -------------------------------  -------  -----  -----  --------  ------  ------  -------  ------  ----------
  Ethernet0                        1,2,3,4     400G   9100     rs  Eth1/1/1  routed    down     down     N/A         N/A
  Ethernet4                        5,6,7,8     400G   9100     rs  Eth1/1/2  routed    down     down     N/A         N/A
  Ethernet8                    17,18,19,20     400G   9100     rs  Eth1/2/1  routed    down     down     N/A         N/A
 Ethernet12                   21,22,23,24     400G   9100     rs  Eth1/2/2  routed    down     down     N/A         N/A
 Ethernet16                   33,34,35,36     400G   9100     rs  Eth1/3/1  routed    down     down     N/A         N/A
 Ethernet20                   37,38,39,40     400G   9100     rs  Eth1/3/2  routed    down     down     N/A         N/A
 Ethernet24                   49,50,51,52     400G   9100     rs  Eth1/4/1  routed    down     down     N/A         N/A
 Ethernet28                   53,54,55,56     400G   9100     rs  Eth1/4/2  routed    down     down     N/A         N/A
```

Figure 29 - Resulting Breakouts for all Ports

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 to 1/8 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 6 links here 1/2-1/7
Leaf1(config)# interface Eth 1/8
Leaf1(config-if-Eth1/8)# description Link to Spine1
Leaf1(config-if-Eth1/8)# speed 800000
Leaf1(config-if-Eth1/8)# unreliable-los auto
Leaf1(config-if-Eth1/8)# no shutdown
Leaf1(config-if-Eth1/8)# mtu 9100
Leaf1(config-if-Eth1/8)# ipv6 enable
Leaf1(config-if-Eth1/8)# ars bind default
Leaf1(config-if-Eth1/8)# exit
! Now doing the next block of uplinks to spine 1
! Note – do all 1/25 to 1/32 in this block (showing just first and last)
Leaf1(config)# interface Eth 1/25
Leaf1(config-if-Eth1/25)# description Link to Spine1
Leaf1(config-if-Eth1/25)# speed 800000
Leaf1(config-if-Eth1/25)# unreliable-los auto
Leaf1(config-if-Eth1/25)# no shutdown
Leaf1(config-if-Eth1/25)# mtu 9100
Leaf1(config-if-Eth1/25)# ipv6 enable
Leaf1(config-if-Eth1/25)# ars bind default
Leaf1(config-if-Eth1/25)# exit
! Range of 6 links here 1/26-1/31
Leaf1(config)# interface Eth 1/32
Leaf1(config-if-Eth1/32)# description Link to Spine1
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
! Setup uplink interfaces to Spine2
! Note – do all 1/33 to 1/40 in this block (showing just first and last)
Leaf1(config)# interface Eth 1/33
Leaf1(config-if-Eth1/33)# description Link to Spine2
Leaf1(config-if-Eth1/33)# speed 800000
Leaf1(config-if-Eth1/33)# unreliable-los auto
Leaf1(config-if-Eth1/33)# no shutdown
Leaf1(config-if-Eth1/33)# mtu 9100
```

```
Leaf1(config-if-Eth1/33)# ipv6 enable
Leaf1(config-if-Eth1/33)# ars bind default
Leaf1(config-if-Eth1/33)# exit
! Range of 6 links here 1/34-1/39
Leaf1(config)# interface Eth 1/40
Leaf1(config-if-Eth1/40)# description Link to Spine2
Leaf1(config-if-Eth1/40)# speed 800000
Leaf1(config-if-Eth1/40)# unreliable-los auto
Leaf1(config-if-Eth1/40)# no shutdown
Leaf1(config-if-Eth1/40)# mtu 9100
Leaf1(config-if-Eth1/40)# ipv6 enable
Leaf1(config-if-Eth1/40)# ars bind default
Leaf1(config-if-Eth1/40)# exit
! Now doing next block of uplinks to spine 2
! Note – do all 1/57 to 1/64 in this block (showing just first and last)
Leaf1(config)# interface Eth 1/57
Leaf1(config-if-Eth1/57)# description Link to Spine2
Leaf1(config-if-Eth1/57)# speed 800000
Leaf1(config-if-Eth1/57)# unreliable-los auto
Leaf1(config-if-Eth1/57)# no shutdown
Leaf1(config-if-Eth1/57)# mtu 9100
Leaf1(config-if-Eth1/57)# ipv6 enable
Leaf1(config-if-Eth1/57)# ars bind default
Leaf1(config-if-Eth1/57)# exit
! Range of 6 links here 1/58-1/63
Leaf1(config)# interface Eth 1/64
Leaf1(config-if-Eth1/64)# description Link to Spine2
Leaf1(config-if-Eth1/64)# speed 800000
Leaf1(config-if-Eth1/64)# unreliable-los auto
Leaf1(config-if-Eth1/64)# no shutdown
Leaf1(config-if-Eth1/64)# mtu 9100
Leaf1(config-if-Eth1/64)# ipv6 enable
Leaf1(config-if-Eth1/64)# ars bind default
Leaf1(config-if-Eth1/64)# 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 1/9/1-1/24/2 (showing just first and last)
! SSE-8164 best practice to use the inner 32 ports for hosts
Leaf1(config)# interface Eth 1/9/1
Leaf1(config-if-Eth1/9/1)# speed 400000
Leaf1(config-if-Eth1/9/1)# mtu 9100
```

```
Leaf1(config-if-Eth1/9/1)# fec RS
Leaf1(config-if-Eth1/9/1)# standalone-link-training
Leaf1(config-if-Eth1/9/1)# unreliable-los auto
Leaf1(config-if-Eth1/9/1)# no shutdown
Leaf1(config-if-Eth1/9/1)# ip address 100.65.0.0/31
Leaf1(config-if-Eth1/9/1)# description Link to Corona Node 1 RNIC Slot 1 with IP 100.65.0.1/31
Leaf1(config-if-Eth1/9/1)# ip vrf forwarding Corona
Leaf1(config-if-Eth1/9/1)# exit
! Range of 30 links here 1/9/2-1/24/1
Leaf1(config)# interface Eth 1/24/2
Leaf1(config-if-Eth1/24/2)# speed 400000
Leaf1(config-if-Eth1/24/2)# mtu 9100
Leaf1(config-if-Eth1/24/1)# fec RS
Leaf1(config-if-Eth1/24/1)# standalone-link-training
Leaf1(config-if-Eth1/24/1)# unreliable-los auto
Leaf1(config-if-Eth1/24/1)# no shutdown
Leaf1(config-if-Eth1/24/2)# ip address 100.65.0.62/31
Leaf1(config-if-Eth1/24/2)# description Link to Corona Node 4 RNIC Slot 8 with IP 100.65.0.63/31
Leaf1(config-if-Eth1/24/2)# ip vrf forwarding Corona
Leaf1(config-if-Eth1/24/2)# exit
! Assign /31 IP's to Heineken's host interfaces Eth 1/41/1-1/56/2 (showing just first and last)
Leaf1(config)# interface Eth 1/41/1
Leaf1(config-if-Eth1/41/1)# speed 400000
Leaf1(config-if-Eth1/41/1)# mtu 9100
Leaf1(config-if-Eth1/41/1)# fec RS
Leaf1(config-if-Eth1/41/1)# standalone-link-training
Leaf1(config-if-Eth1/41/1)# unreliable-los auto
Leaf1(config-if-Eth1/41/1)# no shutdown
Leaf1(config-if-Eth1/41/1)# ip address 100.66.0.0/31
Leaf1(config-if-Eth1/41/1)# description Link to Heineken Node 1 RNIC Slot 1 with IP 100.66.0.1/31
Leaf1(config-if-Eth1/41/1)# ip vrf forwarding Heineken
Leaf1(config-if-Eth1/41/1)# exit
! Range of 30 links here 1/41/2-1/56/1
Leaf1(config)# interface Eth 1/56/2
Leaf1(config-if-Eth1/56/2)# speed 400000
Leaf1(config-if-Eth1/56/2)# mtu 9100
Leaf1(config-if-Eth1/56/1)# fec RS
Leaf1(config-if-Eth1/56/1)# standalone-link-training
Leaf1(config-if-Eth1/56/1)# unreliable-los auto
Leaf1(config-if-Eth1/56/1)# no shutdown
Leaf1(config-if-Eth1/56/2)# ip address 100.66.0.62/31
Leaf1(config-if-Eth1/56/2)# description Link to Heineken Node 4 RNIC Slot 8 with IP 100.66.0.63/31
Leaf1(config-if-Eth1/56/2)# ip vrf forwarding Heineken
Leaf1(config-if-Eth1/56/2)# 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 to 1/7 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 6 links here 1/2-1/7
Leaf1(config-router-bgp)# neighbor interface Eth 1/8
Leaf1(config-router-bgp-neighbor)# description Link to Spine1
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
Leaf1(config-router-bgp)# neighbor interface Eth 1/25
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 6 links here 1/26-1/31
Leaf1(config-router-bgp)# neighbor interface Eth 1/32
Leaf1(config-router-bgp-neighbor)# description Link to Spine1
Leaf1(config-router-bgp-neighbor)# peer-group SPINES
Leaf1(config-router-bgp-neighbor)# exit
Leaf1(config-router-bgp)# neighbor interface Eth 1/33
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 6 links here 1/33-1/41
Leaf1(config-router-bgp)# neighbor interface Eth 1/40
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)# neighbor interface Eth 1/57
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 6 links here 1/58-1/63
Leaf1(config-router-bgp)# neighbor interface Eth 1/64
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

```
Leaf2# 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 to 1/7 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 6 links here 1/2-1/7
Leaf2(config)# interface Eth 1/8
Leaf2(config-if-Eth1/8)# description Link to Spine1
Leaf2(config-if-Eth1/8)# speed 800000
Leaf2(config-if-Eth1/8)# unreliable-los auto
Leaf2(config-if-Eth1/8)# no shutdown
Leaf2(config-if-Eth1/8)# mtu 9100
Leaf2(config-if-Eth1/8)# ipv6 enable
Leaf2(config-if-Eth1/8)# ars bind default
Leaf2(config-if-Eth1/8)# exit
! Now doing next block of uplinks to spine 1
! Note – do all 1/25 to 1/32 in this block (showing just first and last)
Leaf2(config)# interface Eth 1/25
Leaf2(config-if-Eth1/25)# description Link to Spine1
Leaf2(config-if-Eth1/25)# speed 800000
Leaf2(config-if-Eth1/25)# unreliable-los auto
Leaf2(config-if-Eth1/25)# no shutdown
Leaf2(config-if-Eth1/25)# mtu 9100
Leaf2(config-if-Eth1/25)# ipv6 enable
Leaf2(config-if-Eth1/25)# ars bind default
Leaf2(config-if-Eth1/25)# exit
! Range of 6 links here 1/26-1/31
Leaf2(config)# interface Eth 1/32
Leaf2(config-if-Eth1/32)# description Link to Spine1
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
! Setup uplink interfaces to Spine2
! Note – do all 1/33 to 1/40 in this block (showing just first and last)
Leaf2(config)# interface Eth 1/33
Leaf2(config-if-Eth1/33)# description Link to Spine2
Leaf2(config-if-Eth1/33)# speed 800000
Leaf2(config-if-Eth1/33)# unreliable-los auto
Leaf2(config-if-Eth1/33)# no shutdown
Leaf2(config-if-Eth1/33)# mtu 9100
Leaf2(config-if-Eth1/33)# ipv6 enable
Leaf2(config-if-Eth1/33)# ars bind default
Leaf2(config-if-Eth1/33)# exit
! Range of 6 links here 1/34-1/39
Leaf2(config)# interface Eth 1/40
Leaf2(config-if-Eth1/40)# description Link to Spine2
Leaf2(config-if-Eth1/40)# speed 800000
Leaf2(config-if-Eth1/40)# unreliable-los auto
Leaf2(config-if-Eth1/40)# no shutdown
Leaf2(config-if-Eth1/40)# mtu 9100
Leaf2(config-if-Eth1/40)# ipv6 enable
Leaf2(config-if-Eth1/40)# ars bind default
Leaf2(config-if-Eth1/40)# exit
! Now doing next block of uplinks to spine 2
! Note – do all 1/57 to 1/64 in this block (showing just first and last)
Leaf2(config)# interface Eth 1/57
Leaf2(config-if-Eth1/57)# description Link to Spine2
Leaf2(config-if-Eth1/57)# speed 800000
Leaf2(config-if-Eth1/57)# unreliable-los auto
Leaf2(config-if-Eth1/57)# no shutdown
Leaf2(config-if-Eth1/57)# mtu 9100
Leaf2(config-if-Eth1/57)# ipv6 enable
Leaf2(config-if-Eth1/57)# ars bind default
Leaf2(config-if-Eth1/57)# exit
! Range of 6 links here 1/58-1/63
Leaf2(config)# interface Eth 1/64
Leaf2(config-if-Eth1/64)# description Link to Spine2
Leaf2(config-if-Eth1/64)# speed 800000
Leaf2(config-if-Eth1/64)# unreliable-los auto
Leaf2(config-if-Eth1/64)# no shutdown
Leaf2(config-if-Eth1/64)# mtu 9100
```

```
Leaf2(config-if-Eth1/64)# ipv6 enable
Leaf2(config-if-Eth1/64)# ars bind default
Leaf2(config-if-Eth1/64)# 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 1/9/1-1/24/2 (showing just first and last)
! SSE-8164 best practice to use inner 32 ports for hosts
Leaf2(config)# interface Eth 1/9/1
Leaf2(config-if-Eth1/9/1)# speed 400000
Leaf2(config-if-Eth1/9/1)# mtu 9100
Leaf2(config-if-Eth1/9/1)# fec RS
Leaf2(config-if-Eth1/9/1)# standalone-link-training
Leaf2(config-if-Eth1/9/1)# unreliable-los auto
Leaf2(config-if-Eth1/9/1)# no shutdown
Leaf2(config-if-Eth1/9/1)# ip address 100.65.0.64/31
Leaf2(config-if-Eth1/9/1)# description Link to Corona Node 5 RNIC Slot 1 with IP 100.65.0.65/31
Leaf2(config-if-Eth1/9/1)# ip vrf forwarding Corona
Leaf2(config-if-Eth1/9/1)# exit
! Range of 30 links here 1/9/2-1/24/1
Leaf2(config)# interface Eth 1/24/2
Leaf2(config-if-Eth1/24/2)# speed 400000
Leaf2(config-if-Eth1/24/2)# mtu 9100
Leaf2(config-if-Eth1/24/1)# fec RS
Leaf2(config-if-Eth1/24/1)# standalone-link-training
Leaf2(config-if-Eth1/24/1)# unreliable-los auto
Leaf2(config-if-Eth1/24/1)# no shutdown
Leaf2(config-if-Eth1/24/2)# ip address 100.65.0.126/31
Leaf2(config-if-Eth1/24/2)# description Link to Corona Node 8 RNIC Slot 8 with IP 100.65.0.127/31
Leaf2(config-if-Eth1/24/2)# ip vrf forwarding Corona
Leaf2(config-if-Eth1/24/2)# exit
! Assign /31 IP's to Heineken's host interfaces Eth 1/41/1-1/56/2 (showing just first and last)
Leaf2(config)# interface Eth 1/41/1
Leaf2(config-if-Eth1/41/1)# speed 400000
Leaf2(config-if-Eth1/41/1)# mtu 9100
Leaf2(config-if-Eth1/41/1)# fec RS
Leaf2(config-if-Eth1/41/1)# standalone-link-training
Leaf2(config-if-Eth1/41/1)# unreliable-los auto
Leaf2(config-if-Eth1/41/1)# no shutdown
Leaf2(config-if-Eth1/41/1)# ip address 100.66.0.64/31
Leaf2(config-if-Eth1/41/1)# description Link to Heineken Node 5 RNIC Slot 1 with IP 100.66.0.65/31
Leaf2(config-if-Eth1/41/1)# ip vrf forwarding Heineken
```

```
Leaf2(config-if-Eth1/41/1)# exit
! Range of 30 links here 1/41/2-1/56/1
Leaf2(config)# interface Eth 1/56/2
Leaf2(config-if-Eth1/56/2)# speed 400000
Leaf2(config-if-Eth1/56/2)# mtu 9100
Leaf2(config-if-Eth1/56/1)# fec RS
Leaf2(config-if-Eth1/56/1)# standalone-link-training
Leaf2(config-if-Eth1/56/1)# unreliable-los auto
Leaf2(config-if-Eth1/56/1)# no shutdown
Leaf2(config-if-Eth1/56/2)# ip address 100.66.0.126/31
Leaf2(config-if-Eth1/56/2)# description Link to Heineken Node 8 RNIC Slot 8 with IP 100.66.0.127/31
Leaf2(config-if-Eth1/56/2)# ip vrf forwarding Heineken
Leaf2(config-if-Eth1/56/2)# 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 to 1/7 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 6 links here 1/2-1/7
Leaf2(config-router-bgp)# neighbor interface Eth 1/8
Leaf2(config-router-bgp-neighbor)# description Link to Spine1
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
Leaf2(config-router-bgp)# neighbor interface Eth 1/25
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 6 links here 1/26-1/31
Leaf2(config-router-bgp)# neighbor interface Eth 1/32
Leaf2(config-router-bgp-neighbor)# description Link to Spine1
Leaf2(config-router-bgp-neighbor)# peer-group SPINES
Leaf2(config-router-bgp-neighbor)# exit
Leaf2(config-router-bgp)# neighbor interface Eth 1/33
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 6 links here 1/34-1/39
Leaf2(config-router-bgp)# neighbor interface Eth 1/40
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)# neighbor interface Eth 1/57
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 6 links here 1/58-1/63
Leaf2(config-router-bgp)# neighbor interface Eth 1/64
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 to 1/7 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 7 links here 1/2-1/7
Leaf3(config)# interface Eth 1/8
Leaf3(config-if-Eth1/8)# description Link to Spine1
Leaf3(config-if-Eth1/8)# speed 800000
Leaf3(config-if-Eth1/8)# unreliable-los auto
Leaf3(config-if-Eth1/8)# no shutdown
Leaf3(config-if-Eth1/8)# mtu 9100
Leaf3(config-if-Eth1/8)# ipv6 enable
Leaf3(config-if-Eth1/8)# ars bind default
Leaf3(config-if-Eth1/8)# exit
! Now doing next block of uplinks to spine 1
! Note – do all 1/25 to 1/32 in this block (showing just first and last)
Leaf3(config)# interface Eth 1/25
```

```
Leaf3(config-if-Eth1/25)# description Link to Spine1
Leaf3(config-if-Eth1/25)# speed 800000
Leaf3(config-if-Eth1/25)# unreliable-los auto
Leaf3(config-if-Eth1/25)# no shutdown
Leaf3(config-if-Eth1/25)# mtu 9100
Leaf3(config-if-Eth1/25)# ipv6 enable
Leaf3(config-if-Eth1/25)# ars bind default
Leaf3(config-if-Eth1/25)# exit
! Range of 6 links here 1/26-1/31
Leaf3(config)# interface Eth 1/32
Leaf3(config-if-Eth1/32)# description Link to Spine1
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
! Setup uplink interfaces to Spine2
! Note – do all 1/33 to 1/40 in this block (showing just first and last)
Leaf3(config)# interface Eth 1/33
Leaf3(config-if-Eth1/33)# description Link to Spine2
Leaf3(config-if-Eth1/33)# speed 800000
Leaf3(config-if-Eth1/33)# unreliable-los auto
Leaf3(config-if-Eth1/33)# no shutdown
Leaf3(config-if-Eth1/33)# mtu 9100
Leaf3(config-if-Eth1/33)# ipv6 enable
Leaf3(config-if-Eth1/33)# ars bind default
Leaf3(config-if-Eth1/33)# exit
! Range of 6 links here 1/34-1/39
Leaf3(config)# interface Eth 1/40
Leaf3(config-if-Eth1/40)# description Link to Spine2
Leaf3(config-if-Eth1/40)# speed 800000
Leaf3(config-if-Eth1/40)# unreliable-los auto
Leaf3(config-if-Eth1/40)# no shutdown
Leaf3(config-if-Eth1/40)# mtu 9100
Leaf3(config-if-Eth1/40)# ipv6 enable
Leaf3(config-if-Eth1/40)# ars bind default
Leaf3(config-if-Eth1/40)# exit
! Now doing next block of uplinks to spine 2
! Note – do all 1/57 to 1/64 in this block (showing just first and last)
Leaf3(config)# interface Eth 1/57
Leaf3(config-if-Eth1/57)# description Link to Spine2
Leaf3(config-if-Eth1/57)# speed 800000
```

```
Leaf3(config-if-Eth1/57)# unreliable-los auto
Leaf3(config-if-Eth1/57)# no shutdown
Leaf3(config-if-Eth1/57)# mtu 9100
Leaf3(config-if-Eth1/57)# ipv6 enable
Leaf3(config-if-Eth1/57)# ars bind default
Leaf3(config-if-Eth1/57)# exit
! Range of 6 links here 1/58-1/63
Leaf3(config)# interface Eth 1/64
Leaf3(config-if-Eth1/64)# description Link to Spine2
Leaf3(config-if-Eth1/64)# speed 800000
Leaf3(config-if-Eth1/64)# unreliable-los auto
Leaf3(config-if-Eth1/64)# no shutdown
Leaf3(config-if-Eth1/64)# mtu 9100
Leaf3(config-if-Eth1/64)# ipv6 enable
Leaf3(config-if-Eth1/64)# ars bind default
Leaf3(config-if-Eth1/64)# 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 1/9/1-1/24/2 (showing just first and last)
! SSE-8164 best practice to use inner 32 ports for hosts
Leaf3(config)# interface Eth 1/9/1
Leaf3(config-if-Eth1/9/1)# speed 400000
Leaf3(config-if-Eth1/9/1)# mtu 9100
Leaf3(config-if-Eth1/9/1)# fec RS
Leaf3(config-if-Eth1/9/1)# standalone-link-training
Leaf3(config-if-Eth1/9/1)# unreliable-los auto
Leaf3(config-if-Eth1/9/1)# no shutdown
Leaf3(config-if-Eth1/9/1)# ip address 100.65.0.128/31
Leaf3(config-if-Eth1/9/1)# description Link to Corona Node 9 RNIC Slot 1 with IP 100.65.0.129/31
Leaf3(config-if-Eth1/9/1)# ip vrf forwarding Corona
Leaf3(config-if-Eth1/9/1)# exit
! Range of 30 links here 1/9/2-1/24/1
Leaf3(config)# interface Eth 1/24/2
Leaf3(config-if-Eth1/24/2)# speed 400000
Leaf3(config-if-Eth1/24/2)# mtu 9100
Leaf3(config-if-Eth1/24/1)# fec RS
Leaf3(config-if-Eth1/24/1)# standalone-link-training
Leaf3(config-if-Eth1/24/1)# unreliable-los auto
Leaf3(config-if-Eth1/24/1)# no shutdown
Leaf3(config-if-Eth1/24/2)# ip address 100.65.0.190/31
Leaf3(config-if-Eth1/24/2)# description Link to Corona Node 12 RNIC Slot 8 with IP 100.65.0.191/31
```

```
Leaf3(config-if-Eth1/24/2)# ip vrf forwarding Corona
Leaf3(config-if-Eth1/24/2)# exit
! Assign /31 IP's to Budweiser host interfaces Eth 1/41/1-1/56/2 (showing just first and last)
Leaf3(config)# interface Eth 1/41/1
Leaf3(config-if-Eth1/41/1)# speed 400000
Leaf3(config-if-Eth1/41/1)# mtu 9100
Leaf3(config-if-Eth1/41/1)# fec RS
Leaf3(config-if-Eth1/41/1)# standalone-link-training
Leaf3(config-if-Eth1/41/1)# unreliable-los auto
Leaf3(config-if-Eth1/41/1)# no shutdown
Leaf3(config-if-Eth1/41/1)# ip address 100.67.0.0/31
Leaf3(config-if-Eth1/41/1)# description Link to Budweiser Node 1 RNIC Slot 1 with IP 100.67.0.1/31
Leaf3(config-if-Eth1/41/1)# ip vrf forwarding Budweiser
Leaf3(config-if-Eth1/41/1)# exit
! Range of 30 links here 1/41/2-1/56/1
Leaf3(config)# interface Eth 1/56/2
Leaf3(config-if-Eth1/56/2)# speed 400000
Leaf3(config-if-Eth1/56/2)# mtu 9100
Leaf3(config-if-Eth1/56/1)# fec RS
Leaf3(config-if-Eth1/56/1)# standalone-link-training
Leaf3(config-if-Eth1/56/1)# unreliable-los auto
Leaf3(config-if-Eth1/56/1)# no shutdown
Leaf3(config-if-Eth1/56/2)# ip address 100.67.0.62/31
Leaf3(config-if-Eth1/56/2)# description Link to Budweiser Node 4 RNIC Slot 8 with IP 100.67.0.63/31
Leaf3(config-if-Eth1/56/2)# ip vrf forwarding Budweiser
Leaf3(config-if-Eth1/56/2)# 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 to 1/7 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 6 links here 1/2-1/7
Leaf3(config-router-bgp)# neighbor interface Eth 1/8
Leaf3(config-router-bgp-neighbor)# description Link to Spine1
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
Leaf3(config-router-bgp-neighbor)# exit
Leaf3(config-router-bgp)# neighbor interface Eth 1/25
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 6 links here 1/26-1/31
Leaf3(config-router-bgp)# neighbor interface Eth 1/32
Leaf3(config-router-bgp-neighbor)# description Link to Spine1
Leaf3(config-router-bgp-neighbor)# peer-group SPINES
```

```
Leaf3(config-router-bgp-neighbor)# exit
Leaf3(config-router-bgp)# neighbor interface Eth 1/33
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 6 links here 1/34-1/39
Leaf3(config-router-bgp)# neighbor interface Eth 1/40
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)# neighbor interface Eth 1/57
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 6 links here 1/58-1/63
Leaf3(config-router-bgp)# neighbor interface Eth 1/64
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 to 1/7 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 6 links here 1/1-1/7
Leaf4(config)# interface Eth 1/8
Leaf4(config-if-Eth1/8)# description Link to Spine1
Leaf4(config-if-Eth1/8)# speed 800000
Leaf4(config-if-Eth1/8)# unreliable-los auto
Leaf4(config-if-Eth1/8)# no shutdown
Leaf4(config-if-Eth1/8)# mtu 9100
Leaf4(config-if-Eth1/8)# ipv6 enable
Leaf4(config-if-Eth1/7)# ars bind default
Leaf4(config-if-Eth1/7)# exit
! Now doing next block of uplinks to spine 1
! Note – do all 1/25 to 1/32 in this block (showing just first and last)
Leaf4(config)# interface Eth 1/25
Leaf4(config-if-Eth1/25)# description Link to Spine1
Leaf4(config-if-Eth1/25)# speed 800000
Leaf4(config-if-Eth1/25)# unreliable-los auto
Leaf4(config-if-Eth1/25)# no shutdown
Leaf4(config-if-Eth1/25)# mtu 9100
Leaf4(config-if-Eth1/25)# ipv6 enable
Leaf4(config-if-Eth1/25)# ars bind default
Leaf4(config-if-Eth1/25)# exit
! Range of 6 links here 1/26-1/31
Leaf4(config)# interface Eth 1/32
Leaf4(config-if-Eth1/32)# description Link to Spine1
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
! Setup uplink interfaces to Spine2
! Note – do all 1/33 to 1/40 in this block (showing just first and last)
Leaf4(config)# interface Eth 1/33
Leaf4(config-if-Eth1/33)# description Link to Spine2
Leaf4(config-if-Eth1/33)# speed 800000
Leaf4(config-if-Eth1/33)# unreliable-los auto
Leaf4(config-if-Eth1/33)# no shutdown
Leaf4(config-if-Eth1/33)# mtu 9100
Leaf4(config-if-Eth1/33)# ipv6 enable
Leaf4(config-if-Eth1/33)# ars bind default
Leaf4(config-if-Eth1/33)# exit
! Range of 6 links here 1/34-1/39
Leaf4(config)# interface Eh 1/40

```
Leaf4(config-if-Eth1/40)# description Link to Spine2
Leaf4(config-if-Eth1/40)# speed 800000
Leaf4(config-if-Eth1/40)# unreliable-los auto
Leaf4(config-if-Eth1/40)# no shutdown
Leaf4(config-if-Eth1/40)# mtu 9100
Leaf4(config-if-Eth1/40)# ipv6 enable
Leaf4(config-if-Eth1/40)# ars bind default
Leaf4(config-if-Eth1/40)# exit
! Now doing next block of uplinks to spine 2
! Note – do all 1/57 to 1/64 in this block (showing just first and last)
Leaf4(config)# interface Eth 1/57
Leaf4(config-if-Eth1/57)# description Link to Spine2
Leaf4(config-if-Eth1/57)# speed 800000
Leaf4(config-if-Eth1/57)# unreliable-los auto
Leaf4(config-if-Eth1/57)# no shutdown
Leaf4(config-if-Eth1/57)# mtu 9100
Leaf4(config-if-Eth1/57)# ipv6 enable
Leaf4(config-if-Eth1/57)# ars bind default
Leaf4(config-if-Eth1/57)# exit
! Range of 6 links here 1/58-1/63
Leaf4(config)# interface Eth 1/64
Leaf4(config-if-Eth1/64)# description Link to Spine2
Leaf4(config-if-Eth1/64)# speed 800000
Leaf4(config-if-Eth1/64)# unreliable-los auto
Leaf4(config-if-Eth1/64)# no shutdown
Leaf4(config-if-Eth1/64)# mtu 9100
Leaf4(config-if-Eth1/64)# ipv6 enable
Leaf4(config-if-Eth1/64)# ars bind default
Leaf4(config-if-Eth1/64)# 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 1/9/1-1/24/2 (showing just first and last)
! SSE-8164 best practice to use inner 32 ports for hosts
Leaf4(config)# interface Eth 1/9/1
Leaf4(config-if-Eth1/9/1)# speed 400000
Leaf4(config-if-Eth1/9/1)# mtu 9100
Leaf4(config-if-Eth1/9/1)# fec RS
Leaf4(config-if-Eth1/9/1)# standalone-link-training
Leaf4(config-if-Eth1/9/1)# unreliable-los auto
Leaf4(config-if-Eth1/9/1)# no shutdown
Leaf4(config-if-Eth1/9/1)# ip address 100.65.0.192/31
```

```
Leaf4(config-if-Eth1/9/1)# description Link to Corona Node 13 RNIC Slot 1 with IP 100.65.0.193/31
Leaf4(config-if-Eth1/9/1)# ip vrf forwarding Corona
Leaf4(config-if-Eth1/9/1)# exit
! Range of 30 links here 1/9/2-1/24/1
Leaf4(config)# interface Eth 1/24/2
Leaf4(config-if-Eth1/24/2)# speed 400000
Leaf4(config-if-Eth1/24/2)# mtu 9100
Leaf4(config-if-Eth1/24/1)# fec RS
Leaf4(config-if-Eth1/24/1)# standalone-link-training
Leaf4(config-if-Eth1/24/1)# unreliable-los auto
Leaf4(config-if-Eth1/24/1)# no shutdown
Leaf4(config-if-Eth1/24/2)# ip address 100.65.0.254/31
Leaf4(config-if-Eth1/24/2)# description Link to Corona Node 16 RNIC Slot 8 with IP 100.65.0.255/31
Leaf4(config-if-Eth1/24/2)# ip vrf forwarding Corona
Leaf4(config-if-Eth1/24/2)# exit
! Assign /31 IP's to Budweiser host interfaces Eth 1/41/1-1/56/2 (showing just first and last)
Leaf4(config)# interface Eth 1/41/1
Leaf4(config-if-Eth1/41/1)# speed 400000
Leaf4(config-if-Eth1/41/1)# mtu 9100
Leaf4(config-if-Eth1/41/1)# fec RS
Leaf4(config-if-Eth1/41/1)# standalone-link-training
Leaf4(config-if-Eth1/41/1)# unreliable-los auto
Leaf4(config-if-Eth1/41/1)# no shutdown
Leaf4(config-if-Eth1/41/1)# ip address 100.67.0.64/31
Leaf4(config-if-Eth1/41/1)# description Link to Budweiser Node 5 RNIC Slot 1 with IP 100.67.0.65/31
Leaf4(config-if-Eth1/41/1)# ip vrf forwarding Budweiser
Leaf4(config-if-Eth1/41/1)# exit
! Range of 30 links here 1/41/2-1/56/1
Leaf4(config)# interface Eth 1/56/2
Leaf4(config-if-Eth1/56/2)# speed 400000
Leaf4(config-if-Eth1/56/2)# mtu 9100
Leaf4(config-if-Eth1/56/1)# fec RS
Leaf4(config-if-Eth1/56/1)# standalone-link-training
Leaf4(config-if-Eth1/56/1)# unreliable-los auto
Leaf4(config-if-Eth1/56/1)# no shutdown
Leaf4(config-if-Eth1/56/2)# ip address 100.67.0.126/31
Leaf4(config-if-Eth1/56/2)# description Link to Budweiser Node 8 RNIC Slot 8 with IP 100.67.0.127/31
Leaf4(config-if-Eth1/56/2)# ip vrf forwarding Budweiser
Leaf4(config-if-Eth1/56/2)# 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 to 1/7 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 6 links here 1/1-1/7
Leaf4(config-router-bgp)# neighbor interface Eth 1/8
Leaf4(config-router-bgp-neighbor)# description Link to Spine1
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
Leaf4(config-router-bgp)# neighbor interface Eth 1/25
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 6 links here 1/26-1/31
Leaf4(config-router-bgp)# neighbor interface Eth 1/32
Leaf4(config-router-bgp-neighbor)# description Link to Spine1
Leaf4(config-router-bgp-neighbor)# peer-group SPINES
Leaf4(config-router-bgp-neighbor)# exit
Leaf4(config-router-bgp)# neighbor interface Eth 1/33
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 6 links here 1/34-1/39
Leaf4(config-router-bgp)# neighbor interface Eth 1/40
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)# neighbor interface Eth 1/57
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 6 links here 1/58-1/63
Leaf4(config-router-bgp)# neighbor interface Eth 1/64
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 to 1/16 in this block (showing just first and last)
```

```
Spine1(config)# interface Eth 1/1
Spine1(config-if-Eth1)# description Link to Leaf1
Spine1(config-if-Eth1)# speed 800000
Spine1(config-if-Eth1)# unreliable-los auto
Spine1(config-if-Eth1)# no shutdown
Spine1(config-if-Eth1)# mtu 9100
Spine1(config-if-Eth1)# ipv6 enable
Spine1(config-if-Eth1)# ars bind default
Spine1(config-if-Eth1/1)# exit
! Range of 13 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 to 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 13 links here 1/18-1/30
Spine1(config)# interface Eth 1/31
Spine1(config-if-Eth1/31)# description Link to Leaf2
Spine1(config-if-Eth1/31)# speed 800000
Spine1(config-if-Eth1/31)# unreliable-los auto
Spine1(config-if-Eth1/31)# no shutdown
Spine1(config-if-Eth1/31)# mtu 9100
Spine1(config-if-Eth1/31)# ipv6 enable
Spine1(config-if-Eth1/31)# ars bind default
Spine1(config-if-Eth1/31)# exit
! Setup downlink interfaces to Leaf3
! Note – do all 1/33 to 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 13 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 to 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 13 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 13 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
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 13 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
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 13 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
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 13 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 to 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 13 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 to 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 13 links here 1/18-1/30
Spine2(config)# interface Eth 1/31
Spine2(config-if-Eth1/31)# description Link to Leaf2
Spine2(config-if-Eth1/31)# speed 800000
Spine2(config-if-Eth1/31)# unreliable-los auto
Spine2(config-if-Eth1/31)# no shutdown
Spine2(config-if-Eth1/31)# mtu 9100
Spine2(config-if-Eth1/31)# ipv6 enable
Spine2(config-if-Eth1/31)# ars bind default
Spine2(config-if-Eth1/31)# exit
! Setup downlink interfaces to Leaf3
! Note – do all 1/33 to 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 13 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 to 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 13 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 13 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
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 13 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
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 13 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
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 13 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 below configuration will be setup.

DSCP Value:

26 will be setup for the Supermicro AOC-S400G-B1C (BCM957608 aka Thor2) RoCEv2 traffic

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

24 will be setup in case customer wished to attach Nvidia CX7 or BF3 Adapters, or AMD Pollara 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 setup 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 MI300X-MI355X 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 at 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 End Fiber Label	Spine	Spine Port	Spine End Fiber Label	Type	Length (m)
1	1	L1P1-S1P1	1	1	S1P1-L1P1	VR8MMF	50
1	2	L1P2-S1P2	1	2	S1P2-L1P2	VR8MMF	50
<…>	<…>	<…>	<…>	<…>	<…>	<…>	<…>
1	63	L1P63-S2P15	2	15	S2P15-L1P63	VR8MMF	50
1	64	L1P64-S2P16	2	16	S2P16-L1P64	VR8MMF	50
2	1	L2P1-S1P3	1	17	S1P17-L2P1	VR8MMF	50
2	2	L2P1-S1P4	1	18	S1P18-L2P2	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-MI355X 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.

BIOS setting location	Parameter	Value	Comments
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.
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 setting 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 MI300X-MI355X 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-MI355X & 8 RNICs on the server (corona-node1 as example) are:
root@corona-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:
06:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
27:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
47:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
66:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
86:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

a7:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
c7:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)
e6:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57608 10Gb/25Gb/50Gb/100Gb/200Gb/400Gb Ethernet (rev 11)

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)

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 systems Netplan. We will not include the bonding of the 2 10G inband management interfaces, the setup on 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
```

RNIC version 230.2.49.0, RoCE version 230.2.49.0, and ROCm version 6.4 on Ubuntu 24.04

Here we will present some of the key steps on the host to enable all the Thor2 RNIC components along with MI300X-MI355X components with Ubuntu but we first point the reader to the authoritative source that is documentation from Broadcom directly at the URL's below:

Link to the documentation and steps: https://docs.broadcom.com/doc/957608-AN2XX
Link to the software drivers and utilities: https://docs.broadcom.com/docs/BCM5760X_SW_231.2.63.0
Link to adapter cable solutions guide: https://docs.broadcom.com/doc/957608-AN1XX
Link to adapter user guide: https://techdocs.broadcom.com/us/en/storage-and-ethernet-connectivity/ethernet-nic-controllers/bcm957xxx/adapters/installation/connecting-the-network-cables/interconnect-compatibility-for-brcm9576xx-adapters-.html#concept.dita_48749323-898e-47bf-963a-be55755df979

Configuring RoCE Support

The NICs are default configured for RoCE/Peer Memory Direct. The following NVM CFG parameters control RoCE operation on an NIC and can be verified using the NICCLI tool.
To check the option value:
sudo niccli -dev <index|pci b:d:f> getoption -name support_rdma -scope <pf number>
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 getoption -name support_rdma -scope 0
To enable the option:
sudo niccli -dev <index|pci b:d:f> setoption -name support_rdma -scope <pf number> -value 1
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 setoption -name support_rdma -scope 0 -value 1
Enable RoCE Performance Profile on the NIC
The default performance profile on the NICs is non-ROCE. The default profile is optimized for scenarios where the majority of the traffic handled by the NIC is L2. If the NIC needs to handle a traffic mix where the RoCE traffic is greater than 50% of all traffic (as would be the case for Peer Memory Direct), then the performance profile should be changed to RoCE.
To check the option value:
sudo niccli -dev <index | pci b:d:f | mac> getoption -name performance_profile
# Highlighted item will change depending on the index or PCIe BDF or MAC
# Example:
sudo niccli -dev 1 getoption -name performance_profile
To enable the option:
sudo niccli -dev <index|pci b:d:f> setoption -name performance_profile -value 1
# Highlighted item will change depending on the index or PCIe BDF or MAC
#value 0: Default
#value 1: RoCE
# Example:
sudo niccli -dev 1 setoption -name performance_profile -value 1

Enable PCIe Relaxed Ordering on the NIC

PCIe Relaxed ordering allows PCIe transactions to be completed out of order and results in a performance boost for applications when enabled. However, care should be taken before Relaxed ordering is enabled as it can lead to data corruption for some applications.

To check the option value:
sudo niccli -dev <index | pci b:d:f | mac> getoption -name pcie_relaxed_ordering
# Highlighted item will change depending on the index or PCIe BDF or MAC
# Example:
sudo niccli -dev 1 getoption -name pcie_relaxed_ordering

To enable the option:
sudo niccli -dev <index|pci b:d:f> setoption -name pcie_relaxed_ordering -value 1
# Highlighted item will change depending on the index
# Example:
sudo niccli -dev 1 setoption -name pcie_relaxed_ordering -value 1

Firmware Based DCBx NVM CFG on NIC

The Broadcom RoCE driver (bnxt_re) configures the QOS defaults for the NIC upon loading. However, if the firwmware based DCBX or the FW-based LLDP is enabled via NVM CFG, the RoCE driver does not configure the QOS on the NIC interface.

NOTE: Firmware-based DCBX and FW-based LLDP should be disabled if the RoCE driver needs to configure the QOS for the interface.

To check the option values:
sudo niccli -dev <index | pci b:d:f | mac> getoption -name dcbx_mode -scope <pf number>
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 getoption -name dcbx_mode -scope 0
sudo niccli -dev <index | pci b:d:f | mac> getoption -name lldp_nearest_bridge -scope <pf number>
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 getoption -name lldp_nearest_bridge -scope 0
sudo niccli -dev <index | pci b:d:f | mac> getoption -name lldp_nearest_non_tpmr_bridge -scope <pf number>
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 getoption -name lldp_nearest_non_tpmr_bridge -scope 0
To disable the options:
sudo niccli -dev <index | pci b:d:f | mac> setoption -name dcbx_mode -scope <pf number> -value 0
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 setoption -name dcbx_mode -scope 0 -value 0
sudo niccli -dev <index | pci b:d:f | mac> setoption -name lldp_nearest_bridge -scope <pf number> -value 0
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF
# Example:
sudo niccli -dev 1 getoption -name lldp_nearest_bridge -scope 0 -value 0
sudo niccli -dev <index | pci b:d:f | mac> setoption -name lldp_nearest_non_tpmr_bridge -scope <pf number> -value 0
# First highlighted item will change depending on the index or PCIe BDF or MAC
# Second highlighted item will change depending on the PF

# Example:
sudo niccli -dev 1 getoption -name lldp_nearest_non_tpmr_bridge -scope 0 -value 0

QOS settings on the NIC

When the NIC is configured for RoCE on a host, the NIC is automatically configured for DCQCN-P along with the following settings.

RoCE v2 packets are marked with a DSCP value of 26 and use Priority 3 internally

CNP packets are marked with a DSCP value of 48 and use Priority 7 internally

PFC is enabled for Priority 3 traffic

Three Traffic classes are set up, TC0 for non-RoCE traffic, TC1 for RoCE traffic, and TC2 for CNP traffic

RoCE and non-RoCE traffic share ETS bandwidth of 50% each. The ETS bandwidth share applies only when the actual traffic is available to use the bandwidth share. In the absence of non-RoCE traffic, all the available bandwidth will be used by RoCE and vice-versa.

CNP traffic is treated as ETS Strict Priority

sudo niccli -dev 1 get_qos

IEEE 8021QAZ ETS Configuration TLV:

PRIO_MAP: 0:0 1:0 2:0 3:1 4:0 5:0 6:0 7:2

TC Bandwidth: 50% 50% 0% 0% 0% 0% 0% 0%

TSA_MAP: 0:ets 1:ets 2:strict 3:strict 4:strict 5:strict 6:strict 7:strict

IEEE 8021QAZ PFC TLV:

PFC enabled: 3

IEEE 8021QAZ APP TLV:

APP#0:

Priority: 7

Sel: 5

DSCP: 48

APP#1:

Priority: 3

Sel: 5

DSCP: 26

APP#2:

Priority: 3

Sel: 3

UDP or DCCP: 4791

TC Rate Limit: 100% 100% 100% 0% 0% 0% 0% 0%

sudo niccli -dev 1 dump pri2cos

Base Queue is 0 for port 0.

Priority TC Queue ID

---

0 0 4

1 0 4

2 0 4

3 1 0
4 0 4
5 0 4
6 0 4
7 2 5
sudo niccli -dev 1 get_dscp2prio
dscp2prio mapping:
priority:7 dscp:48,
priority:3 dscp:26,

Broadcom provides tools NICCLI and bnxtsetupcc.sh that allow changing the DSCP values, the Priority values, the ETS settings and the PFC settings.

NOTE: The important thing to note is that the settings on the NIC and the Ethernet switches match each other.

Section 2.7 BCM957608 Ethernet Networking Guide for AMD Instinct™ MI300X-MI355X GPU Clusters in the document lays out the necessary checks that needs to be performed to verify thor2 NICs have been provision with the right settings. Here we have captured the same with possible commands that can aid in this running this checklist just to highlight here.

Step 1 : Ensure the bnxt_en.ko, bnxt_re.ko, and ib_peer_mem.ko kernel modules are loaded and are the correct version
rmmod bnxt_re => remove the inbox driver (bnxt_re) if you need to
rmmod bnxt_en => remove the inbox driver (bnxt_re) if you need to
modprobe bnxt_en => install the new out of box bnxt_en
update-initramfs -u command for to update the kernel to use the new installed modules.

Step 2 : The AMD GPU driver amdgpu.ko is loaded
lsmod | grep amdgpu => to list the installed modules in the kernel and see if the amdgpu.ko is installed.

Step 3 : Appendix C in the document has the complete script to disable PCIe AXCS . This should be done every time the system is restarted , hence this script needs to be part of the startup script.

Step 4 : check if IOMMU is disabled or is in Pass Through (PT) mode.
dmesg | grep -i iommu (line with “default domain….” Denotes it is passthrough mode.

Step 5 : check the following Standard InfiniBand Commands listed below work correctly.
ibstatus
ibv_devinfo -vvv
ibv_devinfo –vvv -d < roce interface name>

Step 6 : NIC NVM Configuration (to enable RDMA, performance profile, and PCIe Relaxed Ordering) is set to enabled per above
niccli –list
niccli -i 1 nvm -getoption performance_profile
2.2.5.4 => commands to enable the performance on the NICs for RoCE (by default it is set to non-RoCE)
Sudo niccli – dev 1 nvm –setoption peformace_profle -value 1
Need to reboot for them to take effect. They are persistent and need not be applied upon each reboot

Step 7 : to check if the link is up and at the correct speed
Ibstatus
Ethtool <if name>

Step 8 : An IP address is assigned to the NIC interface and the IP address is visible as GID 3 for IPV4 address or IPv6 address in ibv_devinfo -vvv command below

Rdma link show
Ibv_devinfo -vvv
Ibv_devinfo -vvv -d <roce_inft_name)

Step 9 : Interface MTU size is set to 9100 bytes on the host for maximum throughput
Roce Max MTU is 4k , it should be set to a value more than 4K

Step 10 : The Ethernet switch port to which the NIC connects has its MTU set to 9100

Step 11 : The PCIe slot for the NIC shows correct PCIe GEN speed and width
lspci -vvv -s <B:D:F>

Step 12 : Firewall is disabled on the communicating hosts, in case it prevents RDMA
Should be good if the kernel level firewall utilities like iptables were not altered

Step 13 : Ensure there is no GPU/NIC related error
dmsg | grep bnxt_en (and check if there are any error messages)

Step 14 : For RCCL testing, NUMA balancing is disabled on each host participating in the RCCL test
echo 0 > /proc/sys/kernel/numa_balancing or sysctl -w kernel.numa_balancing=0

For More Information:

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

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.

BROADCOM

Broadcom Inc. (NASDAQ: AVGO) is a global technology leader that designs, develops, and supplies a broad range of semiconductor, enterprise software and security solutions. Broadcom's category-leading product portfolio serves critical markets including cloud, data center, networking, broadband, wireless, storage, industrial, and enterprise software. Our solutions include service provider and enterprise networking and storage, mobile device and broadband connectivity, mainframe, cybersecurity, and private and hybrid cloud infrastructure. Broadcom is a Delaware corporation headquartered in Palo Alto, CA. For more information, go to www.broadcom.com

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