— The hard part was never the GPU.

Zero to GPU

The Infrastructure That Makes AI Run

Meet Andani

Proudly sponsored by Rafay

Preface

Everyone can see the GPU. It's on the keynote slides, in the earnings calls, and at the center of almost every conversation about AI. What almost nobody sees is everything around it.

A GPU on its own doesn't train a model or serve a request. Before it can do either, someone has to rack it, boot it, and connect it to its neighbors at hundreds of gigabytes per second. Someone has to decide who is allowed to use it, keep that user's data away from everyone else's, notice when it fails, and replace it without taking a week-long training run down with it. That work is invisible when it goes well, and it's where most of the real engineering in AI infrastructure lives.

The engineers who do this work go by many titles: infrastructure engineers, platform engineers, cloud and network engineers. What they share is the stack they work across, from bare metal and firmware, through high-speed networks and isolation, up to Kubernetes and the platforms that other teams build on.

For a long time, this layer was the concern of a small number of teams at hyperscalers and research labs. ChatGPT launched in November 2022, and almost overnight, every company wanted GPUs. Since then, the scale has changed completely. A single rack, such as NVIDIA's GB200 NVL72, now holds 72 GPUs that behave like one computer, each with 1.8 TB/s of NVLink bandwidth to the others. Neoclouds, enterprises, and national AI programs are building their own GPU infrastructure instead of only renting it. And because GPUs are scarce and expensive, every idle hour is money lost, which puts the infrastructure around them under more pressure than ever.

I came to GPU infrastructure from networking and security, a path I tell in the Afterword. Today I build GPU infrastructure at Rafay. When I made that move, I couldn't find one place that explained how all the pieces fit together, from bare metal and interconnects to multi-tenancy and scheduling. So I started writing it down, one post at a time, as the #ZeroToGPU series on LinkedIn.

This book grew out of that series, and out of the questions, corrections, and conversations it started. Much of what's in it I learned from the engineers I've worked alongside at Juniper Networks, Citrix, and Rafay, who taught me how to build systems that have to be fast, secure, and reliable at the same time.

I also learned by asking questions: of colleagues, of people who know each part of the stack far better than I do, and of anyone patient enough to explain it twice. Piecing it all together took a long time. This book is my attempt to save you that time, by putting as much of it as I can in one place.

The book follows the stack from the physical accelerator outward: how GPU capacity is exposed through virtual machines and bare metal, how GPUs talk to each other inside a server and across a datacenter, how tenants share hardware safely, how fleets are provisioned, observed, and repaired, and how all of it comes together in rack-scale AI factories. The last chapters look at where the field is heading and at the problems that are still unsolved.

It's written for AI/ML engineers, infrastructure engineers, platform teams, and cloud operators, and for anyone who wants to understand what actually runs AI. You don't need a background in GPUs to start. Much of this field is still being figured out, and there are far fewer people working on it than the demand for AI suggests. That makes it a very good time to learn it.

The hard part was never the GPU. It's everything around it.

Thank you for reading, and welcome to everything around the GPU.

Meet Andani
Bengaluru, India

About the Author



Meet Andani is a Senior Software Engineer at Rafay Systems, where he builds the platform layer that turns raw GPU hardware into self-service, production-ready AI infrastructure. His work spans GPU PaaS and Bare Metal as a Service (BMaaS), from provisioning and managing rack-scale systems for AI factories to multi-tenancy, GPU orchestration, and the networking that keeps tenants isolated.

He has spent nearly seven years building scalable, reliable systems software and distributed, cloud-native microservice platforms across networking, security, and AI infrastructure. His career began at Juniper Networks, where he worked on Layer 3 routing protocols, including OSPF, IS-IS, and segment routing, and built OpenConfig support for OSPF using YANG models and gNMI streaming telemetry on Junos and Junos Evolved. At Citrix he moved into security, working on Secure Private Access and zero-trust network access (ZTNA), where he improved application launch performance, built failure visibility and multi-hop latency measurement, and delivered real-time session termination. Different layers, the same problem underneath: how to let many people safely share one system.

Meet writes about infrastructure and engineering on LinkedIn. This book began as #ZeroToGPU, his 25-post series on how GPU infrastructure actually works. He continues with #GPUtoPlatform, on everything between having GPUs and having a platform other people can use, and #BetweenTheCommits, on the craft of engineering itself.

He holds a Bachelor of Technology in Information Technology from Nirma University, Ahmedabad. Originally from Rajkot, Gujarat, he lives in Bengaluru, India. Outside of work, he spends time with friends and family and pursues spiritual practices that keep him grounded.

About the Editor

Haziqa Sajid is a data scientist and technical writer specializing in AI, machine learning, data, and cloud technologies. She holds a Master's degree in Data Science, earned a gold medal for academic distinction, and has more than five years of experience working with AI and data companies. Her work combines hands-on technical experience with developing research-driven technical content for both engineering and executive audiences.

Her technical experience spans generative AI, machine learning, MLOps, cloud infrastructure, data platforms, and AI applications. She has developed and deployed AI/ML projects in cloud environments including Amazon Web Services (AWS) and Google Cloud Platform (GCP), and has worked with AI and cloud technology companies including Nebius, gaining exposure to AI infrastructure, cloud computing, GPU workloads, and the systems that support modern AI applications.

Her writing and editorial work includes technical eBooks, whitepapers, guides, research-driven articles, and long-form technical content. As editor of *Zero to GPU*, she brings her data science background, practical experience with cloud-based AI/ML systems, and technical writing expertise to ensure the material's technical accuracy, clarity, and consistency throughout the book.

Copyright

Written by **Meet Andani**.

Edited by **Haziqa Sajid**.

Published by **Rafay**.

Preface 3
About the Author 5
About the Editor 6
Copyright 7
Chapter 0: The Hard Part Was Never the GPU 11
PART I: Building the GPU Infrastructure Stack 14
Chapter 1: GPU on a VM vs GPU on Bare Metal 15
GPU on a Virtual Machine 15
GPU on Bare Metal 16
Chapter 2: Operationalizing Bare Metal at Scale 17
What Bare Metal Provisioning at Scale Requires 17
Why Bare Metal Is Hard to Operationalize 18
Chapter 3: Bare Metal as a Service (BMaaS) 19
Buying Hardware, Colocation, and BMaaS 19
What Makes Bare Metal a Service 19
Why It Matters for AI Workloads 19
Chapter 4: What Is GPU PaaS? 21
What GPU PaaS Takes Off the AI Team's Plate 21
Why This Matters for AI Teams 21
Chapter 5: The AI Infrastructure Stack 23
Layer 1: Hardware 23
Layer 2: Bare Metal Platform 24
Layer 3: GPU PaaS 24
Layer 4: AI Runtime 24
Layer 5: Model 24
How the Layers Depend on Each Other 24
Chapter 6: Why GPU Infrastructure Is Different From Regular Cloud 26
1. Network Sensitivity 26
2. Failure Blast Radius 26
3. Resource Granularity and Topology 26
4. Multi-Tenancy Complexity 27
Chapter 7: PCIe, NVLink, and NVSwitch 28
PCIe: The General-Purpose Interconnect 28
What Is NVLink? 28
What Is NVSwitch? 28
PCIe vs NVLink vs NVSwitch 29
Chapter 8: RDMA Over InfiniBand 30
Conventional Network Path 30
What RDMA (Remote Direct Memory Access) Changes 30
InfiniBand: The Network Fabric for RDMA at Scale 31
Chapter 9: BlueField DPU and Bare Metal Networking 33

What the BlueField DPU Handles....33
Bare Metal and the Trust Boundary....33
DPU-Based Policy Enforcement....34
Chapter 10: GPU Multi-Tenancy and Network Isolation....35
Compute Isolation....35
Memory Isolation....36
Network Isolation....36
Performance Isolation....36
Security and Trust....37
PART II: Operating and Scaling GPU Infrastructure....38
Chapter 11: Fleet-Wide Software Consistency and Zero-Touch Lifecycle Automation....39
Version Drift Across GPU Fleets....39
Defining a Reproducible GPU Software State....39
Zero-Touch Provisioning Across the Server Workflow....40
Failure Handling in a Zero-Touch Workflow....40
Chapter 12: GPU Observability and Failure Recovery....41
The GPU Metrics That Matter....41
The GPU Observability Tooling Layer....41
Observability Is Also About Utilization....42
Why GPU Failures Are Different....42
Failure Signals to Watch....42
The GPU Recovery Flow....42
Chapter 13: How Kubernetes Manages GPU Workloads....44
The NVIDIA Device Plugin....44
The GPU Stack on Kubernetes....45
Scheduling a GPU vs Scheduling the Right GPU....45
Chapter 14: GB200 and the AI Factory in a Rack....46
What Makes GB200 Different....46
Scaling Beyond the Rack With Spectrum-X....46
Chapter 15: NVIDIA's Moat Is More Than the Chip....48
CUDA: The Original Moat....48
DOCA: The Software Layer for the DPU....48
NIM: Packaging Inference....48
Chapter 16: Sovereign AI and National GPU Infrastructure....50
What Sovereign AI Means....50
Why Countries Care About Sovereign AI....50
What It Takes to Build a Sovereign AI Cloud....50
Where Sovereign AI Is Heading....51
Chapter 17: The Hardest Problems Still Unsolved in GPU Infrastructure....52
1. GPU Utilization....52

2. Dynamic Multi-Tenancy 52
3. Failure at Fleet Scale 52
4. The Tail, Not the Average 53
Hardware Advances, Infrastructure Catches Up 53
Chapter 18: From GPU Infrastructure to Consumable AI Services 54
From GPU Capacity to a Service 54
Afterword: From Routing Protocols to GPU Infrastructure 55
Appendix A: GPU Infrastructure Glossary 57
Appendix B: Recommended Reading 61
Acknowledgements 63
About Rafay 64

Chapter 0: The Hard Part Was Never the GPU

Access to GPUs is only the first step in building AI infrastructure. A GPU is a component, not a product. On its own, it's a very fast chip sitting in a server. Before anyone can train a model or serve a request on it, a lot has to happen around it.

For CPUs, most of that work was solved long ago. Virtualization, cloud APIs, and decades of operating experience mean a developer can get a working CPU instance in seconds without thinking about the hardware underneath. GPUs are different. They're scarce and expensive, they're often used directly on bare metal rather than behind a hypervisor, and the largest workloads tie hundreds or thousands of them together so tightly that a single slow link or failed card affects all of them.

The work around the GPU falls into five jobs:

- **Provision:** turning racked hardware into a ready, consistent machine, with the right operating system, drivers, and firmware, without anyone logging in by hand.
- **Connect:** letting GPUs talk to each other, inside a server, across a rack, and across a datacenter, fast enough that communication doesn't become the bottleneck.
- **Share:** letting many teams or tenants use the same infrastructure safely and fairly, with real isolation between them.
- **Operate:** seeing what every GPU is doing, catching failures early, and recovering without losing work.
- **Deliver:** wrapping all of it into a platform, so the people who use GPUs can ask for capacity and get it, without needing to understand any of the above.

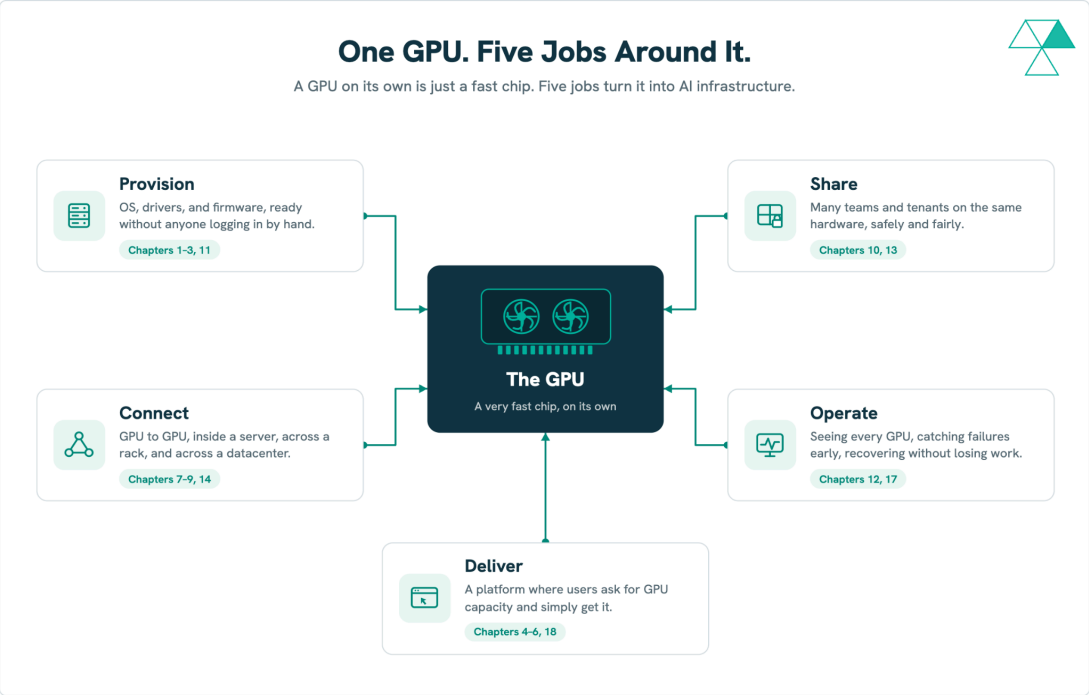


Figure 0: The GPU at the center, and the five jobs around it.

These jobs depend on each other. Isolation is only as strong as the network that enforces it. Utilization is only as good as the scheduler that places work and the observability that shows where capacity sits idle. A self-service platform is only as reliable as the provisioning and failure handling underneath it. That's why this book covers all five, rather than treating any one of them on its own.

How the problems change with scale

The nature of the work changes as a GPU footprint grows:

- **A single server:** the challenge is getting the GPUs working at all, with the right drivers, firmware, and software stack.
- **A cluster:** the challenge becomes sharing, scheduling work, isolating tenants, and keeping expensive GPUs from sitting idle.
- **A fleet** of hundreds or thousands of GPUs: failure becomes the default. Something is always broken, and consistency and automation matter more than any single machine.
- **Rack scale:** the unit of compute is no longer a server but a whole rack, and topology, power, and fault domains shape every decision.

Most organizations move through these stages faster than they expect, and the infrastructure has to keep up.

Understanding that system starts with the infrastructure closest to the GPU. This includes how workloads access the accelerator and how physical GPU servers are provisioned and exposed for use.

How this book is organized

Part I, Building the GPU Infrastructure Stack, starts with how GPU capacity is exposed and delivered. Chapter 1 compares GPUs on virtual machines and on bare metal. Chapter 2 covers what it takes to operate bare metal at scale, Chapter 3 introduces Bare Metal as a Service, and Chapter 4 explains GPU PaaS. Chapters 5 and 6 step back to show the full AI infrastructure stack and why GPU infrastructure is different from regular cloud. Chapters 7 to 9 cover how GPUs connect: PCIe, NVLink, and NVSwitch inside a server, RDMA over InfiniBand between servers, and BlueField DPUs at the edge of each server. Chapter 10 covers how tenants share GPU infrastructure safely.

Part II, Operating and Scaling GPU Infrastructure, turns to running all of this in production. Chapter 11 covers keeping a fleet consistent with zero-touch lifecycle automation, and Chapter 12 covers observability and failure recovery. Chapter 13 explains how Kubernetes schedules GPU workloads. Chapter 14 looks at GB200 and the rack as the new unit of compute, and Chapter 15 at the software ecosystem around NVIDIA hardware. Chapter 16 covers sovereign AI, Chapter 17 the hardest problems still unsolved, and Chapter 18 how all of it becomes a consumable AI service.

The book closes with an Afterword on the lessons that run through all of it, followed by a glossary of terms (Appendix A) and recommended reading for going deeper (Appendix B).

A note on timing

This book was finished in OCT 2026. GPU infrastructure moves quickly, and some specifics, such as product generations, bandwidth figures, and software versions, will change. The problems underneath them, and the principles for solving them, change much more slowly.

PART I: Building the GPU Infrastructure Stack

A GPU server can deliver the computing power, but turning that compute into real-world use requires an operating architecture around the hardware. That architecture governs how workloads access GPUs, how bare metal servers are provisioned and maintained, and how compute, networking, and security isolation are orchestrated across the environment.

This part moves from how GPUs are accessed and delivered as a service, down to the interconnects, networking, and isolation underneath.

These chapters address the engineering question of what has to exist between a physical GPU and a customer being able to consume it as part of an AI workload.

Chapter 1: GPU on a VM vs GPU on Bare Metal

Most people think GPU cloud means GPU access. Not quite. The same GPU can be consumed through two different infrastructure models, either inside a virtual machine (VM) or directly on a bare metal server.

In the VM model, a hypervisor separates the workload from the physical hardware. With bare metal, that virtualization layer is removed, and the workload runs directly on the physical system.

This architectural difference affects latency, memory bandwidth, and access to the server's GPU interconnects.

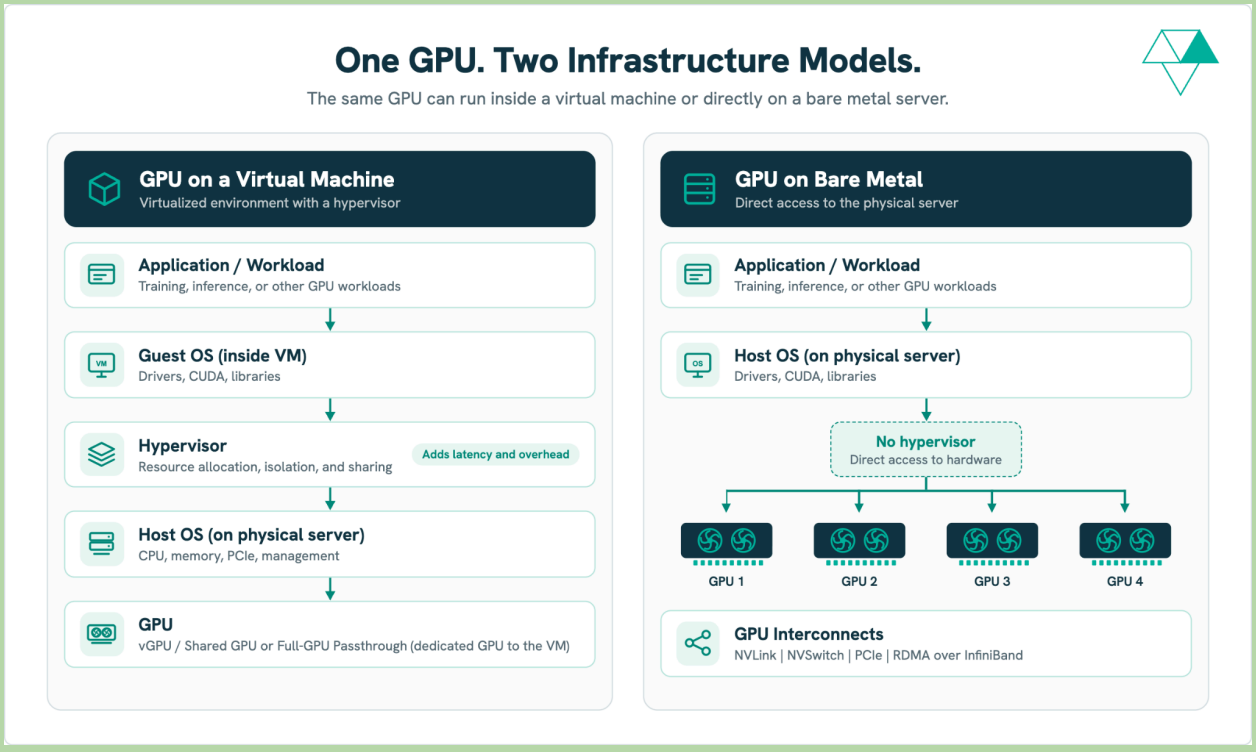


Figure 1. GPU on a virtual machine vs. GPU on bare metal.

GPU on a Virtual Machine

The GPU workload runs inside a VM in a virtualized environment with the hypervisor between the workload and the underlying hardware. This adds another infrastructure layer to the execution path.

Virtualization can add overhead and make it harder to expose the server's GPU topology, peer-to-peer paths, and NVLink or RDMA connectivity to the workload. When a GPU is shared across VMs using vGPU, noisy-neighbor effects can also appear. So, the impact depends on the virtualization model and the workload. Light inference workloads can operate on virtualized GPUs because they may not depend on sustained communication across multiple accelerators.

Multi-GPU training places different demands on the system. Large models distribute computation across several GPUs, requiring repeated exchanges of gradients, activations, and other data during execution. As GPU communication increases, memory bandwidth and interconnect performance have a greater effect on overall training efficiency.

GPU on Bare Metal

A bare metal GPU removes the hypervisor from the workload path, exposing the physical GPU server directly. Workloads can tap into the GPU topology and interconnect capabilities available in that server without the additional virtualization layer.

This direct access is the most straightforward path for multi-GPU setups using NVLink and NVSwitch. NVLink provides high-throughput communication between GPUs, while NVSwitch scales that connectivity across larger GPU domains. For distributed workloads spanning multiple servers, RDMA over InfiniBand provides the high-speed communication path between nodes.

So, bare metal exposes more of the physical system architecture to the workload. This includes GPU compute, memory bandwidth, GPU-to-GPU interconnects, and the network path used for distributed communication.

> **ENGINEERING TAKEAWAY**
>
> No hypervisor. No virtualization tax. You get the actual silicon.

However, the performance comes with operational complexity. Manual setup may work for an isolated physical machine. Operating hundreds or thousands of bare metal GPU servers for multiple tenants requires repeatable discovery, provisioning, configuration, lifecycle management, and isolation.

Chapter 2: Operationalizing Bare Metal at Scale

Bare metal GPU sounds simple: just give me the hardware, skip the hypervisor. Running it at scale is a completely different engineering problem.

Preparing one bare metal GPU server manually is manageable. An engineer can provision the operating system, configure networking and storage, install the required drivers, and prepare the machine for use.

At fleet scale, those steps have to be executed consistently across hundreds of physical servers and multiple tenants. Server state, software versions, network configuration, and tenant isolation must remain consistent as machines are provisioned and reprovisioned.

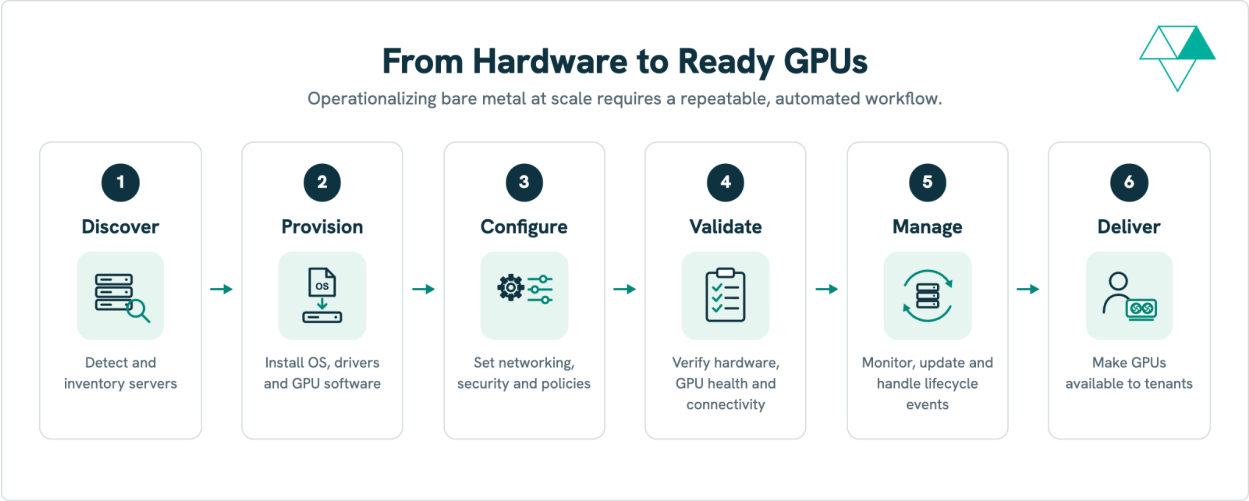


Figure 2. Automated workflow from bare metal discovery to tenant-ready GPU capacity.

What Bare Metal Provisioning at Scale Requires

A repeatable bare metal provisioning workflow requires several capabilities:

- **Zero-touch discovery:** When new hardware enters the rack, the provisioning system needs to detect it without requiring an operator to manually add each server to the workflow.
- **OS imaging:** The system must install operating systems across the fleet. The same imaging process must be repeatable as new servers are added or existing machines are reprovisioned.

- **Driver and firmware management:** GPU drivers and firmware versions need to be managed across the fleet. Updating servers individually increases operational work and can leave nodes running different software or firmware versions.
- **Network isolation:** In a multi-tenant environment, network configuration must prevent one tenant from accessing another tenant's traffic. So, isolation needs to be applied as part of provisioning rather than configured manually after the server is assigned.
- **Tenant self-service:** GPU capacity needs to be requestable without an infrastructure engineer manually preparing the physical server for every request. Provisioning automation connects the infrastructure request to the operations required to make the server available.

Why Bare Metal Is Hard to Operationalize

Virtualized infrastructure places a hypervisor between workloads and the underlying hardware. Bare metal removes that abstraction, leaving power, networking, storage, operating systems, drivers, and firmware as infrastructure components to coordinate.

This increases the number of states that a provisioning workflow has to manage. A server may be physically available but unusable because its network interface card (NIC) is misconfigured, its GPU driver is stale, or its network configuration does not enforce the expected tenant boundary. Across a large fleet, these inconsistencies create configuration drift and make provisioning outcomes less predictable.

So, production bare metal operations depend on executing discovery, imaging, software configuration, networking, and isolation consistently across the fleet.

> **ENGINEERING TAKEAWAY**
>
> The hardware gives you raw performance. The platform layer gives you the ability to use it safely, at scale, and on demand.

Once these operations are automated and repeatable, dedicated physical servers can be exposed through a cloud consumption model. Bare Metal as a Service (BMaaS) applies that model to physical infrastructure.

Chapter 3: Bare Metal as a Service (BMaaS)

Bare Metal as a Service (BMaaS) provides dedicated physical infrastructure through an on-demand consumption model. The workload runs on the physical server without a hypervisor, while hardware acquisition and physical infrastructure operations remain outside the customer's environment.

This changes how customers acquire and receive dedicated GPU capacity. Instead of purchasing GPU servers and waiting for hardware procurement, customers consume physical capacity that has already been deployed and made available as a service.

Buying Hardware, Colocation, and BMaaS

Purchasing GPU infrastructure requires upfront capital and introduces hardware lead times before capacity is available. The organization owns the servers and assumes responsibility for deploying and operating the physical infrastructure.

With colocation, the organization owns the hardware, but the hardware runs in a third-party data center.

With BMaaS, the provider operates the physical GPU infrastructure and makes dedicated servers available to customers on demand.

What Makes Bare Metal a Service

BMaaS combines dedicated physical infrastructure with automated, on-demand delivery. Its key service characteristics include:

- **Dedicated physical hardware:** Each allocation provides access to a physical GPU server with direct access to its compute, memory, and interconnect capabilities.
- **On-demand provisioning:** GPU servers are provisioned from available infrastructure in minutes, reducing lead time for hardware acquisition and deployment.
- **Preloaded software environment:** Operating systems, CUDA, and GPU drivers can be installed as part of server preparation, delivering a configured environment with the allocated hardware.
- **Usage-based consumption:** Usage-based billing, often per hour or per-second, maps physical GPU capacity to a consumption-based infrastructure model without requiring upfront server acquisition

Delivering these capabilities repeatedly requires automated server discovery, OS imaging, driver and firmware management, network isolation, and provisioning workflows across the bare metal fleet.

Why It Matters for AI Workloads

Large-model training can require the performance characteristics of the physical GPU system, including GPU memory throughput, NVLink bandwidth, and no virtualization overhead.

BMaaS gives AI teams access to those hardware capabilities while removing the need to purchase and operate the underlying physical infrastructure.

The result is dedicated GPU capacity provisioned through an on-demand service model. The customer receives the physical compute environment required for the workload. The provider handles the operational work of preparing and delivering that server before the capacity becomes available.

> **ENGINEERING TAKEAWAY**
>
> Cloud simplicity, bare metal performance.

BMaaS still exposes the server and its infrastructure environment to the user. GPU PaaS moves the abstraction boundary higher by taking on more of the infrastructure operations.

Chapter 4: What Is GPU PaaS?

Everyone wants to run AI workloads. Nobody wants to manage GPU infrastructure. AI teams need GPU compute, but GPU compute comes with infrastructure work. Hardware has to be provisioned. Drivers need updates. Operating systems need patching. Networks have to be configured. Security and tenant isolation have to be maintained.

That is the problem GPU Platform as a Service, or GPU PaaS, is designed to solve. It moves those responsibilities behind a managed platform so the AI team interacts with the compute rather than the infrastructure that delivers it.

What GPU PaaS Takes Off the AI Team's Plate

GPU PaaS builds on the bare metal infrastructure and abstracts its provisioning, lifecycle management, networking, and multi-tenancy behind a managed platform. AI teams consume GPU resources through the platform without directly managing the infrastructure operations used to prepare and maintain them.

Several infrastructure responsibilities move behind the platform at this layer:

- **Hardware provisioning:** The platform provisions and prepares physical GPU capacity for workload execution.
- **Driver updates:** The platform manages GPU driver maintenance across the underlying infrastructure.
- **OS patching:** Operating-system maintenance remains part of the platform lifecycle.
- **Network configuration:** Network connectivity is configured alongside the compute environment.
- **Security and isolation:** Infrastructure and tenant isolation are maintained below the workload layer.

The resulting interface exposes GPU compute without requiring users to coordinate each of these infrastructure operations. This allows users to focus on the model, training job, or inference endpoint.

Why This Matters for AI Teams

Data scientists and ML engineers often need to move between model development, training, and inference without adding infrastructure preparation to each workflow. GPU PaaS separates these workload activities from the underlying provisioning and lifecycle operations.

A self-service request can initiate the required infrastructure workflow while provisioning, networking, lifecycle management, and multi-tenancy remain within the platform layer. The user consumes the resulting GPU environment without operating those components directly.

> **ENGINEERING TAKEAWAY**
>
> AI teams are not infrastructure teams. They shouldn't have to be. One click, GPU ready, no ticket to the infra team. Building that one click is where the real complexity lives.

To understand where that abstraction sits relative to the hardware, orchestration, AI runtime, and model, the next chapter maps the full AI infrastructure stack.

Chapter 5: The AI Infrastructure Stack

Everyone talks about AI models. Nobody talks about what actually runs them. An AI workload sits at the top of a five-layer stack: hardware, bare metal platform, GPU PaaS, AI runtime, and model.

Each layer provides a different part of the execution environment, from GPU compute and interconnects to resource orchestration and model execution.

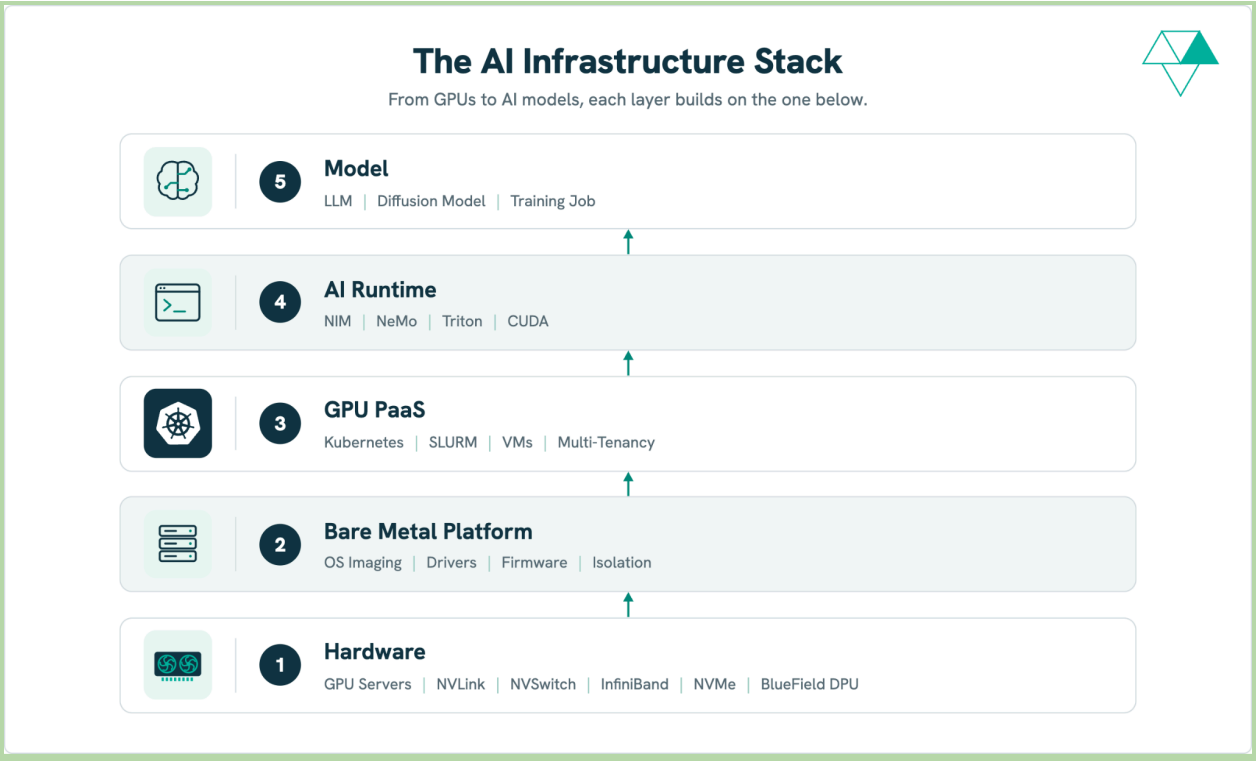


Figure 3. The five layers of the AI infrastructure stack, from GPU hardware to AI models.

Layer 1: Hardware

The hardware layer contains the physical components of the GPU infrastructure:

- GPU servers provide the accelerated compute capacity.
- NVLink provides high-bandwidth communication between GPUs.
- NVSwitch connects multiple GPUs through an all-to-all switching fabric.
- InfiniBand provides high-speed networking for communication across GPU servers.
- NVMe storage provides high-speed storage within the infrastructure.
- BlueField DPUs provide dedicated processing for networking, storage, and security functions.

These components define the available compute capacity and the communication paths within and between GPU systems.

Layer 2: Bare Metal Platform

The bare metal platform manages the configuration and lifecycle of the physical servers. Its responsibilities include OS imaging, driver management, firmware updates, network isolation, and zero-touch lifecycle operations.

OS images and GPU drivers establish the software environment on each server, while firmware management maintains the underlying hardware configuration. Network isolation applies the required connectivity boundaries, and zero-touch lifecycle operations repeat these configurations for the physical fleet.

Layer 3: GPU PaaS

GPU PaaS introduces the orchestration and consumption layer above bare metal. It includes Kubernetes, SLURM, VM orchestration, multi-tenancy, and self-service APIs

Kubernetes and SLURM provide orchestration and scheduling environments for GPU workloads, while VM orchestration supports virtualized GPU consumption. Multi-tenancy separates infrastructure resources across users or tenants, and self-service APIs expose those resources through programmatic workflows.

Layer 4: AI Runtime

The AI runtime provides the software components used to execute AI workloads on the GPU infrastructure, such as NVIDIA NIM microservices, NeMo, Triton Inference Server, and CUDA libraries.

CUDA provides the GPU computing software layer, while NIM, NeMo, and Triton provide runtime components for model development and inference. These components execute against the compute resources exposed through the infrastructure and platform layers.

Layer 5: Model

The model layer contains the AI workloads running on the stack. These can be an LLM generation, diffusion model inference, or a model training job.

At this layer, the workload consumes the compute, communication, orchestration, and runtime capabilities provided by the layers below it. Its execution characteristics depend on how those underlying resources are configured and delivered.

How the Layers Depend on Each Other

The five layers form a dependency chain. Hardware provides compute, while GPU interconnects and network fabrics determine how that compute communicates within and across servers. The bare metal and platform layers provision, configure, orchestrate, and isolate those resources. The AI runtime provides the software execution environment, and the model consumes the resulting system.

These dependencies also define how infrastructure issues surface at the workload layer. For example, a training workload showing poor scaling may be constrained by the communication layer.

So, model behavior is only one part of diagnosing an AI workload. GPU compute, interconnects, networking, provisioning, isolation, orchestration, and runtime configuration all contribute to how the workload executes. Mapping the full stack helps trace an issue to the infrastructure layer responsible for it.

> **ENGINEERING TAKEAWAY**
>
> Most AI conversations happen at Layer 5. Most AI problems happen at Layers 1–3.

This interconnected architecture makes GPU infrastructure different from regular cloud environments, particularly as workloads span multiple accelerators and servers.

Chapter 6: Why GPU Infrastructure Is Different From Regular Cloud

GPU infrastructure is not simply “regular cloud with a GPU attached.”The difference shows up in how compute, networking, resource topology, and failures affect workload execution.

In a regular cloud, CPU workloads are largely independent. Capacity can be expanded by adding VMs, and workloads can be rescheduled when a node fails. In contrast, GPU infrastructure supports parallel workloads that can span multiple GPUs and servers. These systems rely on NVLink, NVSwitch, and InfiniBand topology to move data between accelerators, creating tighter compute-communication dependencies.

Four differences follow from that architecture.

1. Network Sensitivity

In typical cloud environments, the network connects workloads and services but is not the primary compute bottleneck.

Distributed GPU workloads continuously exchange data between participating accelerators. Network latency, bandwidth, and congestion affect how quickly GPUs can synchronize and continue computation. A congested switch can slow communication across the distributed job, leaving GPUs waiting for data.

Low-latency, high-bandwidth fabrics and RDMA over InfiniBand or Ethernet (RoCE) reduce this communication overhead across GPU nodes.

2. Failure Blast Radius

Node failures have different consequences for tightly coupled distributed workloads. A regular cloud workload can be rescheduled onto another node when its original node goes down.

But a distributed training job can span hundreds of GPUs that participate in the same execution. If one node fails, the failure can interrupt the entire training job. For example, in a synchronous 256-GPU training job, one failed node can stall all 256 GPUs until the job restarts from its last checkpoint.

So, failure detection and recovery need to be managed at the level of the entire distributed job.

3. Resource Granularity and Topology

Regular cloud infrastructure primarily allocates resources such as CPU cores and RAM. GPU infrastructure introduces additional resource dimensions, including GPUs, memory bandwidth, PCIe connectivity, NVLink domains, NVSwitch connectivity, and InfiniBand topology.

GPU placement determines which communication paths a workload can use. Adding GPU capacity without accounting for these relationships can place participating accelerators across different interconnect or network paths. This increases communication latency and reduces synchronization efficiency between GPUs.

So, resource scheduling must account for both GPU capacity and accelerator topology.

4. Multi-Tenancy Complexity

GPU multi-tenancy requires parallel isolation across various infrastructure layers, including compute, GPU memory, network fabric, DPU-enforced policies, and InfiniBand PKeys.

A tenant boundary may span the GPU, its memory, and the communication fabric linking workloads, with each resource needing a dedicated isolation mechanism.

> **ENGINEERING TAKEAWAY**
>
> GPU infrastructure isn't harder because GPUs are hard. It's harder because everything around the GPU has to operate at a completely different level of precision. Not a subset of clouds. A superset.

Among these differences, communication sensitivity affects how efficiently multiple GPUs operate as a single workload. The next chapter examines the intra-server communication architecture through PCIe, NVLink, and NVSwitch.

Chapter 7: PCIe, NVLink, and NVSwitch

You have eight GPUs in a server. Now make them work as one. That requires moving data between those accelerators fast enough to keep computation moving. Multi-GPU training repeatedly exchanges and synchronizes gradients, activations, and other data across GPUs. When communication takes longer, GPUs spend more time waiting for synchronization and less time executing compute.

This links GPU compute utilization with interconnect bandwidth, with PCIe, NVLink, and NVSwitch functioning at different points in this communication architecture.

PCIe: The General-Purpose Interconnect

PCI Express (PCIe) connects devices within a server, including GPUs, NICs, and storage. It provides the general-purpose system fabric for communication among these devices and other server components.

PCIe Gen5 x16 provides approximately 128 GB/s of bidirectional bandwidth. For single-GPU inference and general device communication, PCIe provides the required system connectivity.

Multi-GPU training introduces heavier GPU-to-GPU traffic. As gradients and activations move repeatedly between accelerators, the bandwidth available between GPUs becomes part of training performance. PCIe remains part of the server architecture, but it provides general-purpose device connectivity.

What Is NVLink?

NVLink provides high-bandwidth communication between NVIDIA GPUs. It creates GPU-to-GPU communication paths without routing those transfers through the CPU.

For H100 SXM GPUs, NVLink 4.0 provides up to 900 GBps of total bidirectional bandwidth.

Higher GPU-to-GPU bandwidth reduces the time spent moving this data. Raw FLOPS define available compute capacity, while interconnect bandwidth affects how much of that capacity a multi-GPU workload can use during synchronized execution.

What Is NVSwitch?

NVLink provides point-to-point GPU connectivity, but the number of direct connections increases as more GPUs are added. An eight-GPU system would require 28 direct connections for every GPU to have a dedicated link to every other GPU.

NVSwitch provides a high-bandwidth switching fabric for NVLink-connected GPUs. GPUs connect through the NVSwitch fabric, enabling all-to-all communication without requiring an individual physical connection between every GPU pair.

In an H100 SXM5 system, four NVSwitches support all-to-all communication among eight GPUs, with the GPU fabric operating at NVLink bandwidth.

NVSwitch can also extend the NVLink domain beyond a single server. For example, GB200 NVL72 uses an NVSwitch fabric to connect 72 GPUs across multiple nodes, extending high-bandwidth GPU communication to a rack-scale architecture.

PCIe vs NVLink vs NVSwitch

The three technologies serve different roles within the GPU system:

- **PCIe:** Connects GPUs, NICs, storage, and other devices through the general-purpose system fabric.
- **NVLink:** Provides high-bandwidth, direct GPU-to-GPU communication.
- **NVSwitch:** Switches NVLink connections to support all-to-all communication across larger GPU domains.

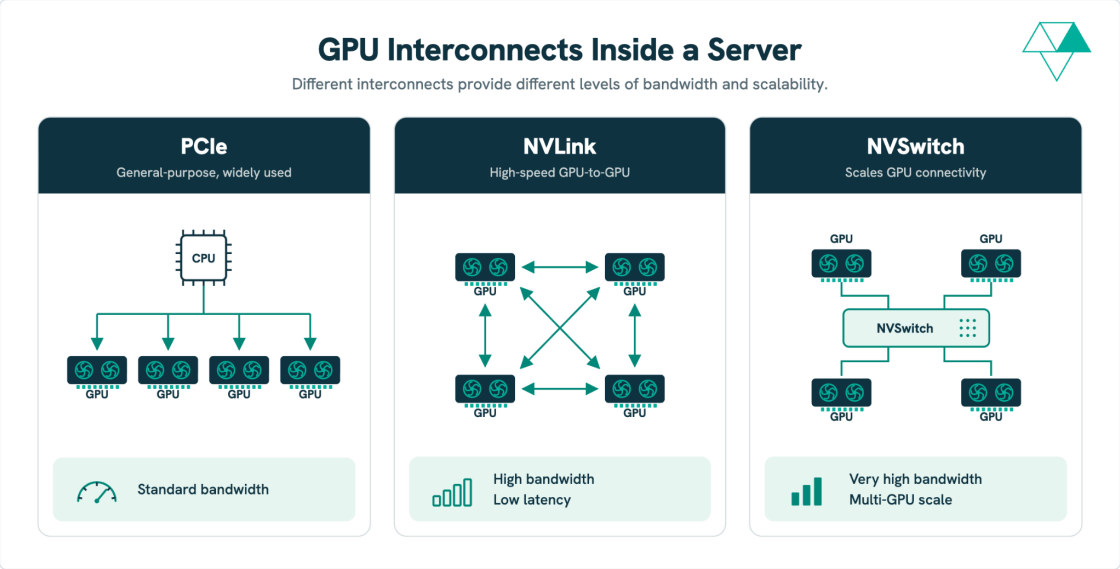


Figure 4. PCIe, NVLink, and NVSwitch connectivity within a multi-GPU server.

ENGINEERING TAKEAWAY

PCIe makes the GPU usable. NVLink makes GPUs work as a team. And NVSwitch is the highway interchange that lets that team scale to a whole rack.

PCIe, NVLink, and NVSwitch address communication within the server and tightly coupled GPU domains. Distributed GPU clusters require a scale-out fabric for communication between servers, which brings RDMA and InfiniBand into the architecture.

Chapter 8: RDMA Over InfiniBand

NVLink connects GPUs inside a server. NVSwitch connects GPUs inside a rack. But what connects hundreds of GPU nodes across the datacenter?

Remote Direct Memory Access (RDMA) provides the communication mechanism for moving data between GPU nodes with reduced CPU and kernel involvement. RDMA can operate over InfiniBand or Ethernet through RoCE (RDMA over Converged Ethernet). This chapter focuses on RDMA over InfiniBand, while Ethernet-based scale-out networking is discussed later in the book.

Conventional Network Path

With a conventional TCP/IP data path, network transfers pass through the host networking stack. The CPU participates in processing the transfer, data moves through kernel space, and the network stack handles communication between nodes.

For distributed GPU workloads, this processing occurs during repeated exchanges between participating nodes. CPU and kernel involvement adds latency to the communication path and increases host processing overhead.

As more GPUs participate in a distributed job, these communication operations span more nodes during synchronization.

What RDMA (Remote Direct Memory Access) Changes

RDMA allows data to move directly between the memory of two systems over the network (memory-to-memory transfers).

With GPUDirect RDMA, the network adapter can read and write GPU memory directly, so data moves from a GPU on one node to a GPU on another without staging through host memory.

So, RDMA reduces CPU involvement in network transfers and provides a lower-latency path for data exchanged during distributed computation.

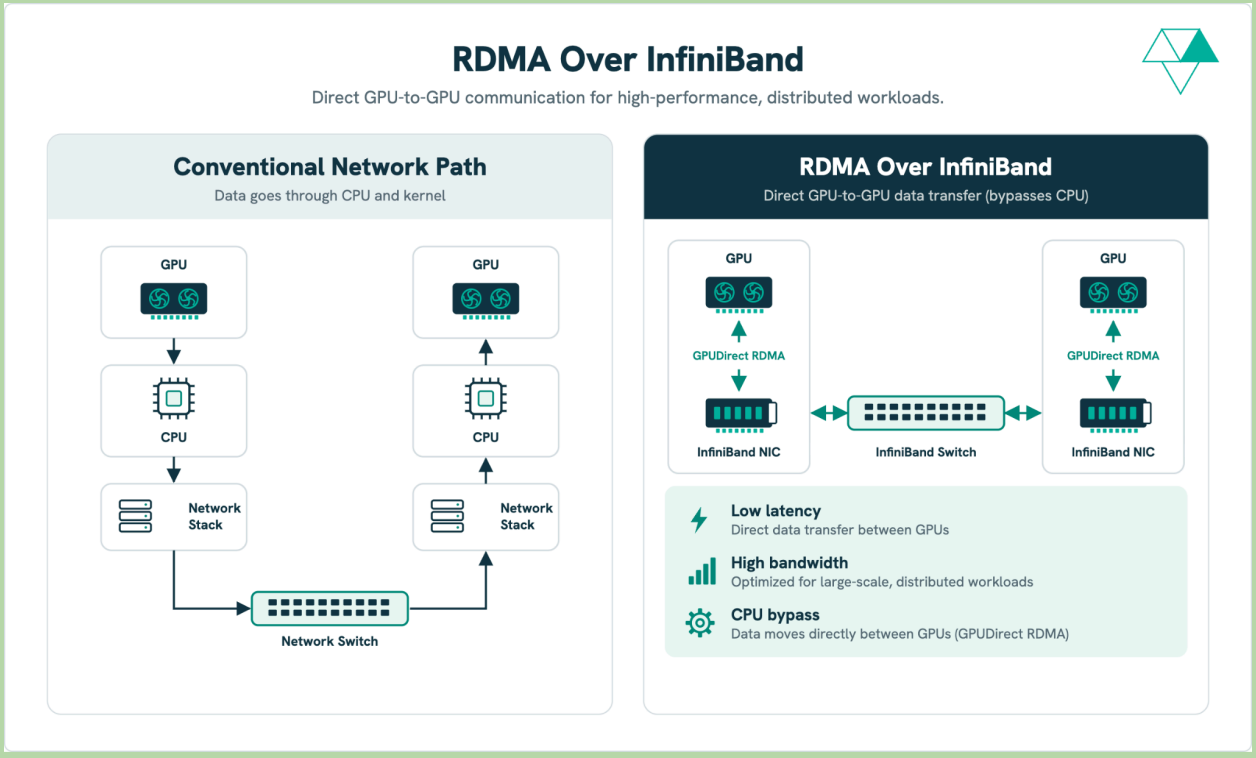


Figure 5. Conventional network communication compared with RDMA over InfiniBand.

InfiniBand: The Network Fabric for RDMA at Scale

InfiniBand provides a high-bandwidth and low-latency network fabric for RDMA communication across GPU nodes. The infrastructure can include:

- **HDR InfiniBand:** 200 Gbps per port.
- **NDR InfiniBand:** 400 Gbps per port.
- **XDR InfiniBand:** 800 Gbps per port.

Combined with RDMA, InfiniBand provides very low latency, around a microsecond end-to-end, and high bandwidth for collective communication such as all-reduce. AllReduce synchronizes data across the GPUs in a distributed training job and makes communication performance important as GPU and node counts grow

The communication hierarchy spans both intra-system and scale-out connectivity. NVLink and NVSwitch provide high-bandwidth communication within tightly coupled GPU systems, while RDMA over InfiniBand carries data between GPU nodes across the cluster.

ENGINEERING TAKEAWAY

> NVLink handles the conversation inside the server. RDMA handles the conversation between servers. Together they let thousands of GPUs train as if they were one.

The next architectural requirement is maintaining networking, storage, security, and isolation below the tenant-controlled host OS. BlueField DPUs move these infrastructure functions onto a dedicated processor.

Chapter 9: BlueField DPU and Bare Metal Networking

Everyone knows the CPU. Everyone knows the GPU. But there's a third processor quietly running modern AI data centers: the DPU. A GPU server can use three processor types with different responsibilities:

- **CPU:** General-purpose compute
- **GPU:** Accelerated parallel compute
- **DPU:** Infrastructure processing and offload

In a GPU server without a DPU, the CPU has to handle infrastructure work alongside the application it runs. This work includes packet processing and routing, managing storage, encryption, security enforcement, and virtualization-related processing. Every CPU cycle spent on these tasks is not spent on the workload. A Data Processing Unit (DPU) takes over these functions with its own dedicated processing resources, so the CPU and GPU can stay focussed on application and accelerated compute.

What the BlueField DPU Handles

NVIDIA BlueField combines dedicated processor cores, network connectivity, and hardware accelerators for infrastructure processing.

BlueField-3 includes 16 Arm cores and up to 400 Gb/s of network throughput, along with hardware acceleration for networking, storage, and security. It runs its own operating system independently of the host operating system.

This architecture separates selected infrastructure operations from the CPU and host OS. The DPU can run networking, storage, and security functions even when the tenant controls the operating environment on the server.

Bare Metal and the Trust Boundary

In a bare metal environment, the tenant can control the physical server's host OS and have root access. Network isolation cannot depend on controls running inside that same operating system because root access gives the tenant administrative control over the host.

Network policy therefore needs an enforcement point outside the tenant-controlled OS.

Switch-based networking keeps network control outside the host, but ties configuration to the external switching infrastructure. Host-based networking moves configuration onto the server but places that control within the tenant-managed environment.

A DPU introduces an infrastructure processing layer between the host and the network.

DPU-Based Policy Enforcement

DPU-based networking shifts network policy, routing, and tenant isolation to the DPU. It has its own processor cores and operating environment, enabling these controls to function independently of the host operating system.

The resulting control boundary separates host administration from infrastructure networking. The tenant controls the host OS, while the provider controls networking and isolation through the DPU

This allows the tenant to retain root access to the physical server without giving that host control over the networking policies used to isolate tenants. For this separation to hold, administrative control of the DPU must remain outside the tenant-controlled host, preventing the host from reconfiguring the networking and isolation policies enforced on the DPU.

> **ENGINEERING TAKEAWAY**
>
> CPU runs applications. GPU runs compute. DPU runs infrastructure. The DPU redraws the trust boundary: everything above it is the tenant's domain, everything on and below it is the provider's.

However, moving network-policy enforcement below the tenant-controlled host addresses the networking component of tenant isolation. A complete GPU tenant boundary must also isolate compute, GPU memory, network traffic, performance, and security.

Chapter 10: GPU Multi-Tenancy and Network Isolation

Running one GPU workload is straightforward. Running many tenants' workloads on shared GPU infrastructure, safely, fairly, and with full isolation, is one of the hardest problems in the field.

Multi-tenancy is well established in regular cloud environments. Hypervisors divide physical servers into virtual machines, giving each tenant an isolated environment while enforcing boundaries between workloads.

GPU infrastructure adds complexity, with tenants sharing accelerators, GPU memory, and network fabrics, sometimes providing root access in bare metal setups. So, isolation must extend beyond the VM boundary across compute, GPU memory, network fabric, performance, and security. Each dimension requires different controls to maintain the tenant boundary.

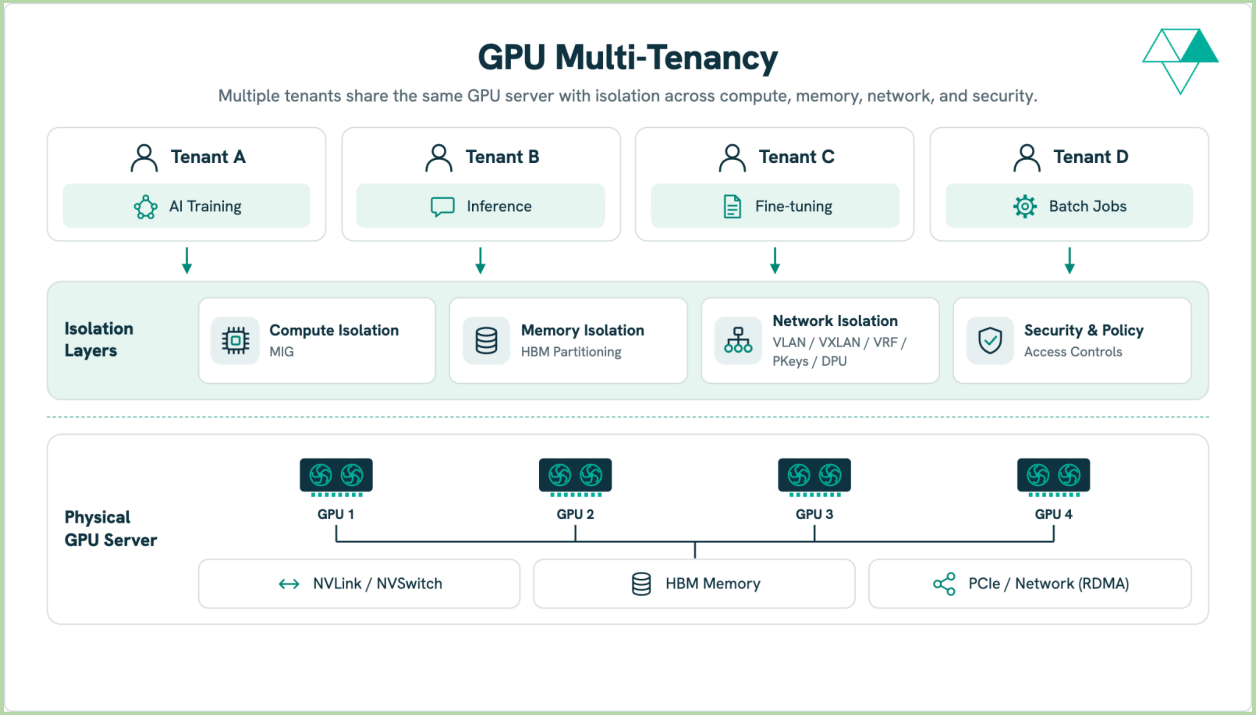


Figure 6. GPU multi-tenancy with isolation across compute, memory, networking, and security.

Compute Isolation

Compute isolation prevents workloads from different tenants from interfering with GPU execution.

Multi-Instance GPU (MIG) can partition a GPU into separate instances, allowing GPU compute resources to be allocated across workloads. MIG is available on specific NVIDIA GPUs and uses predefined instance profiles, so it does not fit every workload.

MIG partitions compute, memory, and cache resources into hardware-isolated instances. Other sharing methods, such as time-slicing and MPS, provide more flexible GPU sharing but do not provide the same memory or fault isolation. Networking, performance, and security still require separate controls.

Memory Isolation

GPU multi-tenancy also requires separation of high-bandwidth memory (HBM). Memory assigned to one tenant must remain inaccessible to other tenants sharing the infrastructure and must be cleared before the GPU is reassigned so that data from the previous tenant is not exposed to the next tenant.

This boundary prevents cross-tenant access to data stored in GPU memory. Compute and memory isolation must be maintained together when sharing GPU resources.

Network Isolation

Multiple tenants can share the same physical switches, cables, NICs, and InfiniBand infrastructure while requiring separate network paths. Network isolation can be applied at several layers:

- **VLAN:** Provides Layer 2 segmentation, with 4,094 usable VLAN IDs in the 802.1Q identifier space.
- **VXLAN:** Creates overlay networks and supports approximately 16 million network segments.
- **VRF:** Maintains separate routing tables for Layer 3 isolation.
- **InfiniBand PKeys:** Partition access within the InfiniBand fabric.

These mechanisms operate at different parts of the network stack. VLAN and VXLAN provide network segmentation, VRFs separate Layer 3 routing, and PKeys partition the InfiniBand fabric. A multi-tenant architecture can apply several of these controls across the same infrastructure.

Enforcement also depends on who controls the host. Bare metal environments may give tenants root access to the operating system, which makes host-level network controls part of the tenant-controlled environment.

DPU-based enforcement moves network policy and isolation below the host OS. The tenant retains control of the server, while the DPU enforces which networks the host can access.

Performance Isolation

Tenant workloads can remain logically isolated while competing for shared infrastructure resources. A compute-intensive workload can consume resources, reducing the performance available to other tenants.

Performance isolation addresses this noisy-neighbor effect by preventing one workload from starving another workload of the resources required for execution.

Security and Trust

Bare metal multi-tenancy introduces a trust boundary in which the tenant can control the host OS while infrastructure policy must remain under provider control.

Isolation must be enforced across GPU compute, HBM, network fabric, performance, and security. Controls that depend on the host OS are insufficient for infrastructure policies that must remain outside the tenant's administrative boundary.

> **ENGINEERING TAKEAWAY**
>
> CPU multi-tenancy is one dimension. GPU multi-tenancy is compute, memory, network, performance, and security, all at once. Compute isolation keeps workloads apart. Network isolation keeps data apart. And in a multi-tenant cloud, a network leak isn't a bug. It's a breach.

DPU-based policy enforcement places selected networking and security controls below the tenant-controlled host, while GPU partitioning and network segmentation apply isolation at their respective resource layers.

PART II: Operating and Scaling GPU Infrastructure

So far, we've covered the architecture beneath GPU workloads, including bare metal infrastructure, platform layers, GPU interconnects, scale-out networking, DPUs, multi-tenancy, and network isolation.

Operating that architecture brings requirements for consistency across thousands of GPU nodes, effective hardware failure detection, and a deep understanding of accelerator topology.

At the same time, the concept of infrastructure is expanding. Kubernetes schedules individual GPU resources, while GB200 NVL72 extends high-bandwidth GPU domains to the rack. NVIDIA's software stack connects the GPU, DPU, and inference layers, while sovereign AI projects implement these requirements at national scale.

This part moves from keeping a fleet consistent and observable, up to scheduling, rack-scale systems, and the problems still unsolved.

These chapters address the engineering question of how to keep that architecture consistent, observable, and reliable as it grows.

Chapter 11: Fleet-Wide Software Consistency and Zero-Touch Lifecycle Automation

The least glamorous part of GPU infrastructure is also one of the most likely to break everything.

A GPU node requires an operating system, a compatible NVIDIA driver and CUDA toolkit, libraries such as cuDNN and NCCL, and the correct firmware versions. These components form the software and firmware state in which GPU workloads execute.

Across hundreds of GPU servers, that state must remain consistent as nodes are deployed and re-imaged. A driver and CUDA mismatch can cause workload failures, while firmware inconsistencies can cause otherwise similar nodes to behave differently.

Version Drift Across GPU Fleets

GPU nodes can diverge as software and firmware versions change over time. Because of this, individual nodes may end up running different NVIDIA drivers, CUDA releases, supporting software libraries, or firmware revisions.

These differences affect reproducibility. A workload that runs successfully on one node can fail on another when the underlying software stacks do not match.

Maintaining a consistent GPU fleet requires a defined software baseline that can be reproduced across nodes. Rather than updating components piecemeal, orchestrate the operating system, NVIDIA drivers, CUDA toolkits, system libraries, and hardware firmware as a single synchronized configuration.

Defining a Reproducible GPU Software State

Several mechanisms can be combined to maintain a consistent GPU software environment.

- **Golden images** package the operating system, NVIDIA driver, and CUDA environment into a version-controlled image. The same approved combination can then be applied across GPU nodes.
- **PXE boot** provides an automated mechanism for network-booting that image onto bare metal. A node can be re-imaged from the approved software baseline without manually installing the operating system and GPU stack.
- **Declarative configuration** defines the expected system state and uses automation to converge nodes toward that configuration.

- **Immutable infrastructure** replaces in-place patching with clean re-imaging. When the software baseline changes, nodes can be rebuilt from the updated image.

The objective is to create a consistent software environment across nodes with the same configuration. This reduces configuration drift and makes the execution environment reproducible across the fleet.

Zero-Touch Provisioning Across the Server Workflow

Chapter 2 introduced the stages of zero-touch bare metal provisioning, from server discovery and imaging to network configuration and tenant delivery. At fleet scale, that workflow also provides a repeatable way to apply the approved software and firmware state across GPU nodes.

After the server is racked and powered on, BMC and Redfish can discover hardware without manual registration. The system inventories and health-checks GPUs, NICs, and DPUs. The approved golden image is PXE-booted, applying the required drivers, CUDA environment, and firmware.

Network configuration then programs the DPU, applies tenant isolation, and connects the server to the required fabric. Once those operations complete successfully, the server registers, credentials are issued, and the capacity is made available for tenant use.

No operator login, manual configuration step, or infrastructure ticket is required between physical installation and delivery.

Failure Handling in a Zero-Touch Workflow

Automating the successful provisioning path does not cover nodes that fail validation or configuration.

- A **failed GPU health check** should quarantine the affected node and prevent delivery.
- A **failed firmware update** can trigger a retry and flag the node when the update cannot complete.
- A **network configuration conflict** requires rollback so that the server is not left with a partially applied network state.

These checks keep failed or partially configured nodes out of the available GPU pool. Zero-touch provisioning therefore includes both executing the provisioning workflow and automatically handling conditions that prevent a server from reaching the required configuration.

The engineering work shifts to automation that performs, validates, retries, quarantines, and rolls back infrastructure operations.

> **ENGINEERING TAKEAWAY**
>
> No human logs in. No manual step. No ticket. Zero-touch isn't about removing effort. It's about moving all the effort upfront, into the automation, so the runtime path needs zero humans.

Once servers can be configured consistently and provisioning failures can be handled automatically, the next requirement is visibility into GPU health, utilization, memory, interconnect activity, and hardware failures.

Chapter 12: GPU Observability and Failure Recovery

You can't fix what you can't see. Conventional infrastructure monitoring tracks CPU utilization, memory, disk, and network activity, but GPU clusters require additional telemetry from the accelerators, GPU memory, and interconnects.

A server can report high CPU utilization while its GPUs remain idle. GPU observability measures accelerator utilization, memory activity, interconnect throughput, power and thermal conditions, and hardware errors across the fleet.

The GPU Metrics That Matter

GPU monitoring combines compute, memory, interconnect, and hardware-health metrics:

- **GPU utilization:** The percentage of the sampling period during which one or more kernels were executing on the GPU. It can report 100% even when those kernels are not fully using the GPU's available compute resources.
- **GPU memory usage:** The amount of HBM allocated or in use.
- **Memory bandwidth:** Data movement through the GPU memory subsystem, which can expose memory-bound workloads.
- **Temperature and power:** Thermal and power conditions that can affect GPU operation.
- **NVLink and PCIe throughput:** Traffic across GPU interconnects and system connectivity.
- **ECC errors:** Correctable and uncorrectable memory errors.
- **SM activity and SM occupancy:** SM activity measures how often the streaming multiprocessors have active warps, while occupancy measures the number of resident warps relative to the maximum supported.
- **Tensor core usage:** Activity on the tensor cores used by supported AI operations.

GPU utilization percentage provides only one view of accelerator activity. Memory bandwidth, SM occupancy, tensor core usage, and NVLink or PCIe throughput provide additional visibility into how the workload is using the GPU and its surrounding data paths.

The GPU Observability Tooling Layer

NVIDIA Data Center GPU Manager (DCGM) provides GPU telemetry used for monitoring accelerator health and operation.

Operators can collect and store that telemetry with Prometheus and visualize it in Grafana. Node-level exporters stream metrics from GPUs across the fleet, allowing operators to inspect accelerator state without accessing each server individually.

This gives operators a centralized view of GPU utilization, memory activity, interconnect throughput, thermal conditions, and hardware errors across the fleet.

Observability Is Also About Utilization

Monitoring GPUs helps identify idle or underutilized capacity. Even when a GPU runs at low utilization, it still takes up valuable infrastructure resources. In a large fleet, consistently low utilization indicates that expensive GPU resources are available but not contributing enough compute work.

Utilization data makes usage visible at both the individual GPU and fleet level. Operators can identify accelerators spending significant periods idle and distinguish them from GPUs running steady workloads.

For GPU infrastructure, observability covers hardware health and how much of the installed GPU capacity is actually being used.

Why GPU Failures Are Different

GPU failures can occur independently within a multi-GPU server. For example, in an eight-GPU node, one GPU can fail while the other GPUs and the host system continue to operate normally.

Distributed training increases the risk of failures affecting the entire workload. A job may coordinate hundreds of GPUs across multiple nodes, meaning that a single GPU failure can disrupt the distributed task.

Additionally, some GPU failures can be silent data corruption (SDC). In such cases, the workload may continue to execute but produce incorrect results instead of failing with an obvious error.

Simply restarting the server or process does not address all failure scenarios. Distributed jobs may need to checkpoint the workload state, isolate the affected GPU or node, and reschedule tasks to use only healthy resources.

Failure Signals to Watch

GPU telemetry provides several signals for identifying accelerator and interconnect failures:

- **Xid errors:** NVIDIA GPU error codes associated with hardware, driver, or execution faults.
- **ECC errors:** Correctable or uncorrectable GPU memory errors.
- **Thermal events:** Conditions that can cause GPU throttling or shutdown.
- **GPU fallen off the bus:** A condition where the operating system can no longer communicate with the GPU.
- **NVLink errors:** Errors or degradation in GPU-to-GPU communication.

The GPU Recovery Flow

Failure recovery can be organized into six steps:

1. **Detect:** Identify the fault through DCGM telemetry or health checks.
2. **Isolate:** Cordon the affected node so that no new workloads are scheduled onto it.
3. **Drain:** Move existing workloads away from the node.
4. **Diagnose:** Determine whether the failure is transient or represents a persistent hardware fault.
5. **Remediate:** Perform the appropriate action, ranging from a GPU reset to reseating or RMA.
6. **Return:** Validate the repaired node before allowing it to rejoin the fleet.

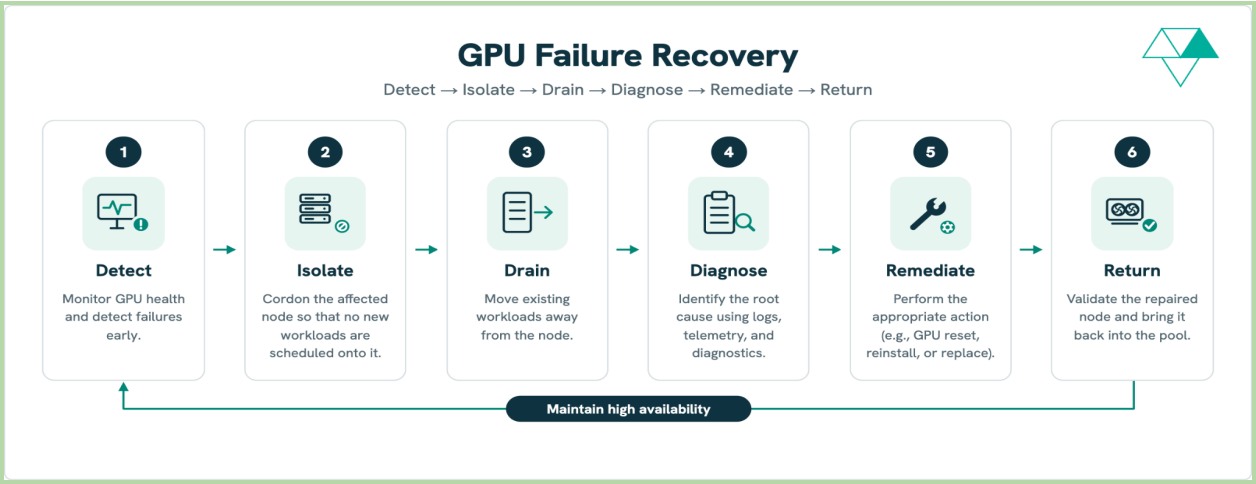


Figure 7. GPU failure recovery workflow from detection and isolation through remediation and return to service.

ENGINEERING TAKEAWAY

A GPU at 30% utilization is money on fire. Idle GPUs are the most expensive thing in the datacenter. And at fleet scale, a GPU failing isn't an edge case. It's a Tuesday. A good GPU cloud isn't defined by how it runs when everything works, but by how gracefully it handles things breaking.

Monitoring and recovery establish the GPU fleet's health and utilization. For containerized GPU workloads, Kubernetes must discover available GPUs, expose them as resources, and decide where to schedule GPU workloads.

Chapter 13: How Kubernetes Manages GPU Workloads

Kubernetes was built to schedule CPUs and memory. GPUs were an afterthought.

Kubernetes schedules CPU and memory as native resources. GPU workloads require additional components to expose accelerators to the scheduler, provide GPU access inside containers, and manage GPU sharing.

Kubernetes does not natively identify an H100 as an accelerator resource, account for GPU memory, or provide GPU isolation. NVIDIA's Kubernetes components add these capabilities to the cluster.

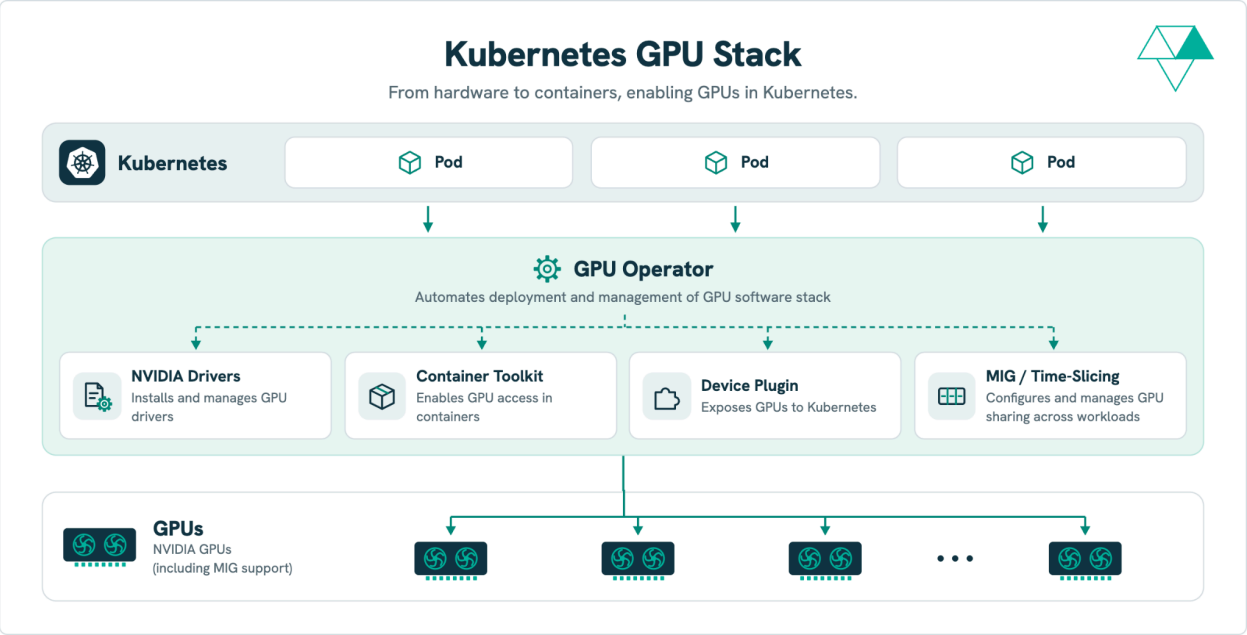


Figure 8. Kubernetes GPU stack for exposing, managing, accessing, and sharing GPU resources.

The NVIDIA Device Plugin

The NVIDIA Device Plugin runs on GPU nodes and exposes the installed accelerators to Kubernetes. It performs four main functions:

- **Discovers GPUs** available on each node.
- **Advertises GPUs** to Kubernetes as schedulable resources.
- **Processes GPU** resource requests from pods, such as `nvidia.com/gpu: 1`.
- **Makes the allocated GPU device** available inside the workload container.

BA pod can declare its GPU requirement in its resource configuration, and Kubernetes can allocate an advertised GPU to that workload. Newer Kubernetes releases also support Dynamic Resource Allocation (DRA), which lets workloads request devices with richer attributes and constraints than a simple GPU count. But GPU allocation is only one part of scheduling. Multi-GPU workloads can also depend on the physical topology of the assigned accelerators. .

The GPU Stack on Kubernetes

GPU support in Kubernetes uses several NVIDIA components for device discovery, runtime access, software management, partitioning, and sharing.

- **NVIDIA Device Plugin** handles GPU discovery and resource advertisement.
- **NVIDIA GPU Operator** automates the GPU software stack, including drivers, runtime configuration, and monitoring.
- **NVIDIA Container Toolkit** provides containers with access to NVIDIA GPU devices.
- **Multi-Instance GPU (MIG)** partitions a physical GPU into multiple GPU instances.
- **Time-slicing** allows multiple pods to share a GPU when the workload does not require the isolation provided by dedicated GPU allocation or MIG partitioning.
- **Multi-Process Service (MPS)** allows multiple processes to execute GPU workloads concurrently, with configurable resource limits but weaker isolation than MIG.

Scheduling a GPU vs Scheduling the Right GPU

For multi-GPU workloads, accelerator placement affects the communication path between the GPUs assigned to the job. Scheduling needs to account for GPU topology in addition to GPU availability.

Three considerations are important:

- **Topology awareness:** Identifies which GPUs share high-bandwidth interconnects such as NVLink. GPUs connected through the same high-bandwidth topology are preferable when the workload requires frequent GPU-to-GPU communication.
- **NUMA alignment:** Keeps the assigned GPU close to the CPU and system memory the workload uses.The Kubernetes Topology Manager can coordinate this alignment on each node.
- **Multi-GPU placement:** Places the GPUs assigned to the same workload according to the best available interconnect between them.

For example, a node may have enough available GPUs to satisfy a workload's request, but those GPUs may not share the best available interconnect. For communication-intensive

workloads, the scheduler should consider how the selected GPUs are connected, not just how many are available.

> **ENGINEERING TAKEAWAY**
>
> Making GPUs work on Kubernetes takes a device plugin. Making them work well takes topology-aware scheduling. The gap between the two is where real platform engineering lives.

Kubernetes manages GPU resources through the software scheduling layer. The next chapter moves to the underlying hardware architecture, where GB200 NVL72 extends the high-bandwidth GPU domain from a multi-GPU server to a 72-GPU rack.

Chapter 14: GB200 and the AI Factory in a Rack

H100 gave us the fastest GPU. GB200 asked a different question: what if the whole rack was the computer?

GPU infrastructure has expanded from individual accelerators to multi-GPU servers and now to rack-scale systems. H100 systems connect up to eight GPUs within a server. GB200 NVL72 extends the high-bandwidth GPU domain to 72 GPUs in a rack.

The result is a rack-scale system in which 72 GPUs communicate over NVLink, moving the tightly connected compute domain beyond the individual server.

What Makes GB200 Different

GB200 combines Grace CPUs and Blackwell GPUs using NVLink-C2C. Each GB200 superchip pairs one Grace CPU with two Blackwell GPUs, giving an NVL72 rack 36 Grace CPUs and 72 Blackwell GPUs. The NVL72 configuration connects all 72 Blackwell GPUs within a single NVLink domain, creating a much larger NVLink domain than the eight-GPU domains in H100 system.

Each Blackwell GPU delivers 1.8 TB/s of NVLink bandwidth. Within the NVL72 configuration, NVLink serves as the high-bandwidth communication fabric, linking the GPUs throughout the rack.

Because communication between GPUs inside the rack utilizes the NVLink fabric, InfiniBand is not needed for GPU-to-GPU communication within the NVL72 domain.

This changes the physical scope of the high-bandwidth GPU fabric. In H100 systems, the communication domain is limited to eight GPUs within a single server. In contrast, the GB200 NVL72 configuration expands this domain to encompass 72 GPUs across an entire rack.

Scaling Beyond the Rack With Spectrum-X

Large AI clusters extend beyond a single NVL72 rack, and connecting multiple racks requires a separate scale-out network, either Ethernet, such as NVIDIA Spectrum-X, or InfiniBand.

Spectrum-X offers NVIDIA's Ethernet-based networking for communication between racks. It is designed for the low-latency, lossless network behavior required as AI clusters scale across larger numbers of GPUs.

This system operates at two levels.

- NVLink provides high-bandwidth GPU communication within the NVL72 rack.
- Spectrum-X serves as the Ethernet scale-out fabric connecting different racks.

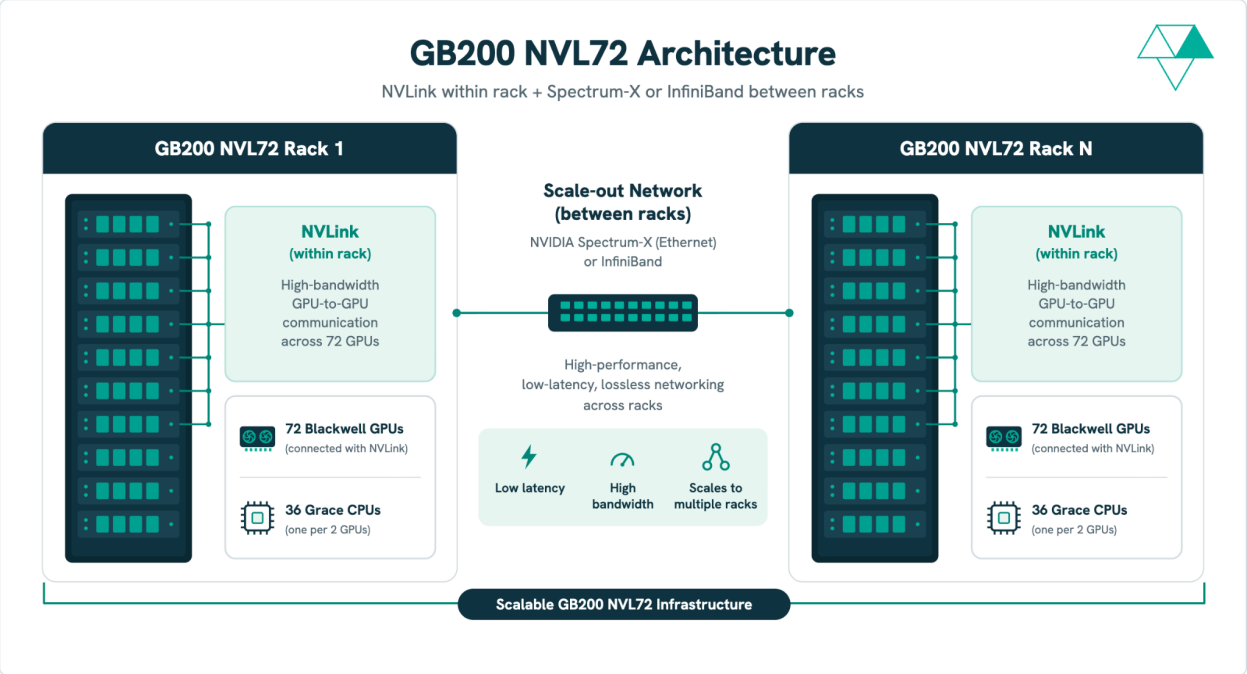


Figure 9. GB200 NVL72 architecture with NVLink within the rack and Spectrum-X connectivity between racks.

> ENGINEERING TAKEAWAY
>
> The hard part was never the GPU. It's making thousands of them behave as one.

As deployments grow to include thousands of GPUs, the infrastructure must coordinate multiple rack-scale GPU domains through that network. GB200 expands NVIDIA's hardware architecture from multi-GPU servers to rack-scale compute. A larger NVLink domain is a big win for performance, but it also means a single fault can affect a much larger group of GPUs, so failure handling matters more, not less. The next layer is software, including CUDA for GPUs, DOCA for DPUs, and NIM for inference.

Chapter 15: NVIDIA's Moat Is More Than the Chip

NVIDIA's position in AI infrastructure extends beyond GPU hardware. Its software ecosystem spans GPU programming, DPU infrastructure processing, and model inference.

Three software layers are central to this ecosystem:

- CUDA for GPU computing
- DOCA for BlueField DPUs
- NIM for model inference

CUDA: The Original Moat

CUDA is NVIDIA's parallel computing platform for programming GPUs. Over nearly two decades, NVIDIA has built an ecosystem of libraries, frameworks, tools, and developer expertise around it. That ecosystem includes cuDNN for deep learning primitives, NCCL for multi-GPU and multi-node communication, and TensorRT for inference optimization.

AI frameworks and applications use these CUDA-based libraries to execute GPU workloads. Developers have also spent years building applications and workflows around the CUDA programming model. Competing with NVIDIA also involves competing with the libraries, framework integrations, tooling, and developer expertise built around CUDA.

DOCA: The Software Layer for the DPU

DOCA is the software framework for programming NVIDIA BlueField DPUs.

BlueField offloads infrastructure functions such as networking, storage, and security from the host CPU. DOCA provides the APIs, libraries, and development framework used to program and control those functions on the DPU.

CUDA targets computation on the GPU, while DOCA targets infrastructure processing on the BlueField DPU.

NIM: Packaging Inference

NVIDIA NIM packages model inference into containers with a pre-optimized inference engine and standard APIs, including OpenAI-compatible endpoints for language models.NIM supports deployment across different infrastructure environments.

This gives NVIDIA a software layer at the inference level in addition to CUDA at the GPU level and DOCA at the DPU level.

> **ENGINEERING TAKEAWAY**
>
> Hardware gets the headlines. Software builds the lock-in. The real moat isn't the chip, it's the ecosystem around it

This hardware and software stack matters beyond individual companies. As governments build domestic AI capacity, control over the underlying compute, data, and infrastructure becomes a question of sovereignty.

Chapter 16: Sovereign AI and National GPU Infrastructure

For most of computing history, where your servers ran didn't matter much. For AI, it's becoming a national priority.

As AI infrastructure gains strategic significance, countries are building GPU capacity within their own borders. They are retaining local control over the data and models that run on this infrastructure.

This moves GPU infrastructure beyond individual enterprise or cloud deployments. Building national AI capacity involves establishing local compute resources, data governance, infrastructure management, and the ability to run AI workloads independently of foreign hyperscalers.

What Sovereign AI Means

Sovereign AI refers to a country's ability to build and run AI using infrastructure and data under its own control and located within its borders.

This includes three key resources: GPU compute, data, and models. GPU infrastructure provides the compute capacity, national or organizational datasets remain under local control, and models are trained or operated within that environment.

The physical location of the infrastructure is part of the deployment model. GPU servers, data, and the systems used to operate AI workloads remain within the required national boundary.

Why Countries Care About Sovereign AI

Several requirements drive sovereign AI infrastructure.

- **Data sovereignty:** Sensitive data may need to remain and be processed within national borders.
- **Strategic independence:** AI infrastructure is becoming a critical asset alongside systems such as power and telecommunications. Locally operated GPU capacity reduces dependence on external infrastructure providers.
- **Economic value:** Building AI infrastructure locally keeps more infrastructure investment and AI activity within the country's economy.
- **Regulation:** Local laws can require control over where data is stored, where workloads execute, and who is accountable for operating the infrastructure.

What It Takes to Build a Sovereign AI Cloud

Building a sovereign AI cloud requires the same infrastructure capabilities covered throughout this book, but deployed under national or regional control. These include:

- **Bare metal provisioning** to deploy and configure GPU servers consistently.
- **Multi-tenancy** to support multiple organizations or workloads on shared infrastructure.
- **High-speed networking** for distributed workloads spanning multiple GPUs and servers.
- **Compute and network isolation** to maintain tenant boundaries.
- **GPU observability** to monitor utilization, memory, interconnects, and hardware health.
- **Failure handling** to detect, isolate, recover, and return failed capacity to service.

These capabilities must operate across national or regional GPU infrastructure. Provisioning must remain repeatable across the fleet, tenant boundaries must span compute and networking, and GPU failures must be detected and recovered as deployed capacity grows.

Where Sovereign AI Is Heading

National GPU clouds are expected to involve technology vendors such as NVIDIA, telecommunications companies, and local cloud providers working together to build the underlying infrastructure. These organizations contribute the GPU infrastructure, connectivity, and local cloud capacity required to build and operate sovereign AI environments.

National control over AI depends on the infrastructure that runs it. Ownership and operation of the GPU infrastructure determine where AI workloads run, where data is processed, and who controls the underlying compute capacity.

> **ENGINEERING TAKEAWAY**
>
> Compute is becoming a form of sovereignty.

Scaling that infrastructure does not resolve the engineering problems underneath it. GPU utilization, dynamic multi-tenancy, failure recovery, and provisioning latency become harder as fleets grow. This challenge leads to the remaining infrastructure problems covered in the next chapter.

Chapter 17: The Hardest Problems Still Unsolved in GPU Infrastructure

GPU hardware continues to advance, but operating large GPU fleets still involves unresolved engineering problems around utilization, dynamic multi-tenancy, failure recovery, and provisioning latency.

Four challenges become more significant as GPU fleets scale.

1. GPU Utilization

Provisioning GPU capacity does not guarantee high utilization. GPUs can remain allocated while spending significant periods idle, leaving expensive accelerator capacity unused.

Maintaining high utilization across a fleet requires keeping available GPUs consistently occupied with workloads. At scale, even short idle periods can translate into substantial unused GPU capacity.

The infrastructure challenge is to keep more of the installed GPU capacity actively computing. Adding more GPUs does not solve the problem if existing capacity remains idle.

2. Dynamic Multi-Tenancy

GPU demand changes over time, while static partitioning assigns resources according to predefined boundaries.

Dynamic multi-tenancy requires sharing GPU capacity across tenants as demand changes, without compromising fairness or isolation. Resource allocation must adapt as workloads enter, leave, or change their GPU requirements.

The difficulty is maintaining tenant isolation and fair access while reallocating capacity on actively shared infrastructure.

3. Failure at Fleet Scale

In a fleet with thousands of GPUs, hardware failures are recurring. Individual GPUs, nodes, or interconnects can fail while workloads are running across the rest of the infrastructure.

Distributed jobs increase the impact of those failures. A single failed GPU can stall a synchronized job spanning hundreds of accelerators.

Fleet operations must detect the failure, isolate the affected resource, and recover capacity without requiring an operator for every event. Recovery also needs to avoid terminating a large distributed job unnecessarily when the failed resource can be isolated and replaced, for example, through elastic training or a fast restart from the last checkpoint.

4. The Tail, Not the Average

Average provisioning time does not show how long the slowest nodes take to become ready. A fleet may provision most nodes quickly while a small percentage take much longer.

For a large distributed job, that difference matters. If the job requires 100 nodes, it cannot start when 99 are ready. It must wait for the last node.

This is why p99 provisioning latency matters for large GPU workloads. It shows what happens at the slow end of the provisioning process, where a small number of delayed nodes can hold up an entire distributed job.

The same pattern appears at runtime: in synchronous distributed training, the slowest GPU determines the pace of each training step, while in inference, high p99 latency can affect the user experience even when average response times remain low.

Hardware Advances, Infrastructure Catches Up

GPU hardware is evolving faster than the infrastructure software that runs it. For example, the shift from H100 to GB200 expands the high-bandwidth GPU domain from eight GPUs within a server to 72 GPUs across a rack.

That increase in scale changes the requirements placed on provisioning, scheduling, networking, observability, and failure handling. New hardware can solve an existing constraint while introducing another at the infrastructure layer.

The engineering challenge is keeping the software and operational systems aligned with each new generation of GPU architecture.

> **ENGINEERING TAKEAWAY**
>
> The GPU was never the hard part. Everything around it, utilization, sharing, failure, latency, is where the real, unsolved engineering lives. That's exactly what makes this field worth working in right now.

Chapter 18: From GPU Infrastructure to Consumable AI Services

Production GPU infrastructure spans the entire operating stack around the accelerator. Bare metal provisioning prepares servers, networks connect workloads, isolation separates tenants, schedulers allocate GPU resources, and observability and recovery systems maintain operation. The final layer turns that operational infrastructure into services that users can request and consume.

From GPU Capacity to a Service

Turning GPU infrastructure into a service requires a layer between users and the underlying infrastructure operations. Providers can define standardized compute configurations and expose them through self-service portals or APIs.

A user request can then trigger the required provisioning, networking, isolation, and lifecycle operations without requiring separate infrastructure requests or manual coordination.

For multi-tenant environments, this service layer also needs RBAC, quotas, policy controls, usage visibility, and metering. The underlying GPU capacity can then be delivered through different consumption models, including bare metal, virtual machines, Kubernetes, SLURM, and higher-level AI environments.

The same model also applies to sovereign AI infrastructure, where GPU services must operate within a defined national boundary for infrastructure, data, and operational control.

> **ENGINEERING TAKEAWAY**
>
> The user sees the service. The platform handles what sits underneath it: provisioning, networking, isolation, policy, lifecycle, and operations. Turning raw GPU capacity into a repeatable consumption model is the final infrastructure layer.

Afterword: From Routing Protocols to GPU Infrastructure

I didn't start in AI infrastructure. I got here through networking and security, and looking back, that path makes complete sense.

Where it started: networking

My career began at Juniper Networks, working on Layer 3 routing: OSPF, segment routing, the low-level networking that moves packets across the internet. The work was in C, C++, and Linux, on systems that had to be correct and fast, because everything above them depended on it.

That's where I learned that the network is never "just plumbing."

Then: security

Next came Citrix, and security: Secure Private Access and zero-trust network access. Isolation, access control, and trust boundaries at scale. It was still systems programming in C, C++, and Go, but the question changed. It was no longer only "is it fast?" It was "is it secure, and can you trust the boundary?"

Networking taught me speed. Security taught me trust.

Now: GPU infrastructure

Today I build GPU infrastructure at Rafay: bare metal, rack-scale systems, multi-tenancy, and the platform layer that turns raw hardware into AI compute.

What surprised me is that it's all the same problems. Network isolation. Trust boundaries. Performance at scale. Multi-tenancy. Failure handling. GPU infrastructure is where networking, security, and systems programming converge, with the most demanding workloads the industry has ever seen.

Careers rarely look like a straight line. Mine looked like three different fields. But the skills compounded, and every layer built on the last. The networking never left. The security never left. They just found a bigger problem to solve.

Six lessons

While writing the series, and then turning it into this book, I kept coming back to a few lessons.

- **The GPU was never the hard part.** Everything around it, networking, isolation, provisioning, and failure handling, is where the real engineering lives.
- **The network is the computer.** At GPU scale, interconnect decides performance more than raw FLOPS. NVLink, InfiniBand, and topology are everything.
- **Idle GPUs are the most expensive thing in the datacenter.** Utilization isn't a nice-to-have. It's the whole economic game.
- **Failure isn't an edge case. It's the default.** At scale, something is always broken. How gracefully you handle that is the platform.
- **Isolation is a hardware problem, not a software promise.** Real multi-tenancy needs boundaries a tenant can't cross, which is why DPUs and hardware partitioning matter so much.
- **The unglamorous work is the actual work.** OS imaging, driver versions, zero-touch provisioning. Nobody posts about it, but it's what makes everything else possible.

What comes next

This field moves fast. Some of the specifics in this book will date quickly: new GPUs, new interconnects, new schedulers. The problems underneath them won't. Isolation, utilization, failure, and trust were the hard parts when I worked on routers and access gateways, and they're the hard parts now.

If there's one thread through all of it, it's this: when it works, nobody notices. That's the job, and it's exactly what makes GPU infrastructure such a fascinating place to build.

— Meet Andani

Appendix A: GPU Infrastructure Glossary

AI Factory: A large-scale infrastructure environment designed to run AI training and inference workloads using tightly integrated GPU compute, networking, storage, and software systems.

Bare Metal: A physical server used directly without a hypervisor separating the workload from the underlying hardware.

Bare Metal as a Service (BMaaS): A service model that provides dedicated physical servers through on-demand provisioning while the provider manages the underlying physical infrastructure.

Blackwell GPU: NVIDIA's GPU architecture used in GB200 systems. The GB200 NVL72 configuration connects 72 Blackwell GPUs within a single NVLink domain for high-bandwidth GPU-to-GPU communication.

BlueField DPU: NVIDIA's Data Processing Unit used to offload and isolate infrastructure functions such as networking, storage, and security from the host CPU.

BMC (Baseboard Management Controller): A management controller that provides out-of-band access to a physical server for operations such as hardware discovery, power management, and remote administration.

CUDA: NVIDIA's parallel computing platform and programming model for executing workloads on NVIDIA GPUs.

DCGM (Data Center GPU Manager): NVIDIA software for monitoring GPU health, utilization, errors, and other accelerator telemetry across data-center environments.

Declarative Configuration: An infrastructure-management approach in which the desired system state is defined, and automation continually works to bring resources into that state.

DOCA: NVIDIA's software framework for developing and managing networking, storage, and security functionality on BlueField DPUs.

DPU (Data Processing Unit): A processor designed to handle infrastructure functions such as networking, security, storage, and data movement independently of the host CPU.

ECC (Error-Correcting Code): A memory-protection mechanism capable of detecting and correcting certain types of memory errors. GPU ECC telemetry is commonly monitored as a hardware-health signal.

GB200 NVL72: An NVIDIA rack-scale system that connects 72 Blackwell GPUs within a high-bandwidth NVLink domain.

Golden Image: A version-controlled operating-system image containing an approved software configuration such as GPU drivers, CUDA components, and supporting libraries.

GPU PaaS (GPU Platform as a Service): A managed platform layer that exposes GPU compute while handling underlying provisioning, software lifecycle, networking, isolation, and infrastructure operations.

Grace CPU: NVIDIA's Arm-based CPU used in GB200 systems. Each GB200 superchip pairs one Grace CPU with two Blackwell GPUs through NVLink-C2C.

HBM (High-Bandwidth Memory): High-throughput memory located close to the GPU and used to store data required by accelerated workloads.

InfiniBand: A high-bandwidth, low-latency network fabric commonly used for communication between nodes in large distributed GPU clusters.

Kubernetes Device Plugin: A Kubernetes mechanism that allows hardware resources such as GPUs to be discovered and advertised to the scheduler. The NVIDIA Device Plugin implements this mechanism for NVIDIA GPUs.

MIG (Multi-Instance GPU): An NVIDIA technology that partitions a physical GPU into multiple isolated GPU instances that can be assigned to separate workloads.

MPS (Multi-Process Service): An NVIDIA GPU-sharing mechanism that allows multiple processes to execute GPU workloads concurrently, with configurable resource limits and weaker isolation than MIG.

Multi-Tenancy: An infrastructure model in which multiple users or organizations share physical resources while maintaining defined boundaries across compute, memory, networking, security, and resource usage.

NCCL (NVIDIA Collective Communications Library): A CUDA library optimized for collective communication operations across multiple GPUs and nodes.

NVIDIA GPU Operator: A Kubernetes operator that automates the deployment and management of NVIDIA GPU software components, including drivers, the Container Toolkit, Device Plugin, and MIG configuration.

NIM (NVIDIA Inference Microservices): Containerized inference software that packages optimized model-serving components behind standardized APIs.

NUMA (Non-Uniform Memory Access): A system architecture in which CPU and memory access latency depends on the physical location of the resources within the server. NUMA alignment can affect GPU workload placement.

NVLink: NVIDIA's high-bandwidth interconnect for direct communication between GPUs and, in some architectures, other processors.

NVLink-C2C: A chip-to-chip interconnect used to connect processors such as Grace CPUs and Blackwell GPUs with high-bandwidth communication.

NVSwitch: A switching fabric that connects multiple NVLink-enabled GPUs and supports high-bandwidth all-to-all communication across a GPU domain.

PCIe (Peripheral Component Interconnect Express): A general-purpose system interconnect used to connect GPUs, NICs, storage devices, and other components to the host system.

PKey (Partition Key): An InfiniBand mechanism used to separate access to different logical partitions on a shared network fabric.

PXE Boot: A network-based boot mechanism commonly used to automatically provision or re-image physical servers.

RBAC (Role-Based Access Control): An authorization model that assigns permissions according to defined user or system roles.

RDMA (Remote Direct Memory Access): A networking mechanism that transfers data directly between memory locations on separate systems while reducing CPU and kernel involvement in the data path.

Redfish: A standards-based API for managing and automating physical server hardware through out-of-band management interfaces.

RoCE (RDMA over Converged Ethernet): A networking technology that enables RDMA communication over Ethernet, allowing data transfers with reduced CPU involvement.

SLURM: A workload manager and scheduler widely used in HPC and large compute environments for allocating resources and scheduling distributed jobs.

SM (Streaming Multiprocessor): A GPU execution unit containing compute resources used to execute parallel workloads. SM activity and occupancy are metrics used to understand GPU utilization.

Silent Data Corruption: An error in which data or computation results become incorrect without an immediately detected hardware or software failure. In GPU workloads, such errors can affect training or inference results without stopping execution.

Spectrum-X: NVIDIA's Ethernet networking platform designed for scale-out AI infrastructure and communication between large GPU systems.

Tensor Core: Specialized GPU compute units optimized for matrix operations used heavily in AI training and inference.

Time-Slicing: A GPU-sharing mechanism that allows multiple workloads to access the same GPU at different execution intervals. Unlike MIG, time-slicing does not provide memory or fault isolation between workloads.

Topology-Aware Scheduling: A scheduling approach that considers physical relationships among GPUs, CPUs, memory, and interconnects when assigning resources to workloads.

VLAN (Virtual Local Area Network): A Layer 2 network segmentation mechanism used to separate traffic on shared networking infrastructure.

VRF (Virtual Routing and Forwarding): A Layer 3 isolation mechanism that maintains separate routing tables on shared networking infrastructure.

VXLAN (Virtual Extensible LAN): An overlay networking technology that provides large-scale logical network segmentation across an underlying IP network.

Xid Error: An NVIDIA GPU error identifier used to report hardware, driver, memory, or execution-related faults.

Zero-Touch Provisioning: An automated provisioning model in which servers progress from discovery and validation through imaging, network configuration, and delivery without requiring manual operator intervention.

Appendix B: Recommended Reading

This book is an introduction to everything around the GPU. Almost every chapter could be a book of its own, and there's always more depth to explore.

The resources below are the documentation, technical articles, and papers I'd point you to next, grouped by topic.

GPU infrastructure moves fast. I keep writing about it in the #GPUtoPlatform series on LinkedIn, which picks up where this book leaves off.

Hardware and Interconnects

- NVIDIA NVLink and NVSwitch by NVIDIA: This technical resource explains the GPU interconnect technologies used for high-bandwidth communication within multi-GPU systems.
- NVIDIA GB200 NVL72 Delivers Trillion-Parameter LLM Training and Real-Time Inference by NVIDIA: This technical article examines the GB200 NVL72 rack-scale architecture, including Grace CPUs, Blackwell GPUs, NVLink, and high-bandwidth communication across 72 GPUs.

Networking and DPUs

- NVIDIA NCCL Documentation by NVIDIA: This documentation explains collective communication operations, including all-reduce, and how NCCL supports communication across multiple GPUs and nodes in distributed AI workloads.
- NVIDIA GPUDirect RDMA Documentation by NVIDIA: This documentation explains how network adapters can read and write GPU memory directly, enabling GPU-to-GPU communication across nodes without staging data through host memory.
- RFC 7348: Virtual eXtensible Local Area Network (VXLAN) by IETF: This standard defines VXLAN overlays, which are widely used to create isolated tenant networks on shared infrastructure.
- NVIDIA DOCA Programming Guide by NVIDIA: This guide introduces the APIs, libraries, and programming framework used to develop networking, storage, and security applications on NVIDIA BlueField DPUs.

GPU Sharing and Isolation

- NVIDIA MIG User Guide by NVIDIA: This guide explains GPU partitioning, instance profiles, and the isolation capabilities of Multi-Instance GPU technology.

- NVIDIA Multi-Process Service (MPS) Documentation by NVIDIA: This documentation explains how multiple processes can share a GPU concurrently, and how this approach differs from hardware partitioning with MIG.

Kubernetes and Scheduling

- Schedule GPUs by Kubernetes: This official guide explains how Kubernetes uses device plugins to expose GPU resources and allows workloads to request accelerators through pod resource configurations.
- NVIDIA GPU Operator Documentation by NVIDIA: This documentation explains how the GPU Operator deploys and manages drivers, the container toolkit, the device plugin, and MIG configuration on Kubernetes clusters.
- Dynamic Resource Allocation by Kubernetes: This official guide describes how workloads can request devices such as GPUs with richer parameters than a simple device count.
- Enabling Multi-Node NVLink on Kubernetes for GB200 NVL72 and Beyond by NVIDIA: This technical article examines how Kubernetes can manage multi-node NVLink environments and support GPU orchestration across rack-scale systems.
- Slurm Workload Manager by SchedMD: This documentation introduces Slurm's resource allocation, job scheduling, and cluster management capabilities.

Operations and Reliability

- Redfish Standard by DMTF: This standard defines interfaces for managing physical server hardware, including discovery, monitoring, and power management.
- NVIDIA Data Center GPU Manager (DCGM) Documentation by NVIDIA: This technical reference covers GPU telemetry, health monitoring, diagnostics, and observability for managing GPU infrastructure at scale.
- NVIDIA Xid Errors by NVIDIA: This reference describes the GPU error codes reported by the NVIDIA driver, which are used to detect and diagnose hardware and software failures.

Research

- Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism by Mohammad Shoeybi et al. (2019): This research paper explores model parallelism for training large language models and demonstrates the importance of GPU communication and distributed execution.

Acknowledgements

This book started as a series of LinkedIn posts, and it became a book because people believed in it before it was one. Thank you to Haseeb Budhani, Rafay's CEO, and Hanumantha (Hemanth) Kavuluru, Rafay's CTO, who first noticed the #ZeroToGPU posts and encouraged the team to take a closer look.. Thank you to Venu Prasad, Robbie Gill, and Chaitanya V for their support and for believing in this from the start. And thank you to Arjoyita Roy, who led this project, and to Simone Morellato, who saw a book in a series of posts and made it happen. Thank you to Larry Carillo for supporting this project.

Thank you to Haziqa Sajid for editing this book and shaping a series of posts into chapters. Any errors that remain are my own. Thank you to all my teammates at Rafay, and to my colleagues at Citrix and Juniper Networks, past and present.

Much of what I learned came from documentation written by people I'll never meet: the engineers behind NVIDIA's documentation, the Kubernetes and Linux communities, and the authors of the open standards this field runs on. Thank you for writing it all down. And thank you to everyone who read, commented on, and corrected the #ZeroToGPU posts. This book is better because of you.

To Shree Swaminarayan Bhagwan, for His grace, and to my family and friends, for their constant support.

And finally, thank you, the reader. Without you, this would just be twenty-five posts in a folder.

About Rafay

Rafay turns GPU and CPU infrastructure into governed, self-service AI platforms. The Rafay Platform helps neoclouds, sovereign AI cloud providers, telecommunications companies, and enterprises provision, manage, and expose bare metal, virtual machines, Kubernetes, and Slurm environments through standardized services.

Through self-service portals and APIs, providers can offer approved compute environments while maintaining centralized control over multi-tenancy, RBAC, quotas, policies, and infrastructure operations. Lifecycle automation supports consistent fleet management, while observability and usage metering provide visibility into infrastructure health and resource consumption.

The #ZeroToGPU series behind this book was written by Meet Andani, who builds GPU infrastructure at Rafay.

To see how the Rafay Platform works, request a demo or talk to the Rafay team.

Learn more about Rafay

- Bare Metal GPUs as a Service: Provision dedicated GPU infrastructure through an on-demand service model.
- Rafay Self-Service: Deliver governed, self-service access to GPU and compute infrastructure.
- Monetizing AI Infrastructure: Explore infrastructure consumption models, resource utilization, and monetization.
- Managed Kubernetes as a Service: Explore Kubernetes provisioning and lifecycle management.

— Everything around the GPU.

The hard part was never the GPU. It's everything around it.

How raw GPUs become infrastructure that runs AI, from bare metal and networking to multi-tenancy and the AI factory.



By Meet Andani

Proudly sponsored by Rafay