RAFAY

WHITE PAPER

The NCP Capability Reference

What it takes to operate a multi-tenant AI cloud as an NVIDIA Cloud Partner



This document consolidates the capabilities an organization builds and operates in order to deliver AI cloud services as an NVIDIA Cloud Partner (NCP). It is organized layer by layer, from the physical foundation through to the orchestration and governance layer that makes capacity consumable

Contents

FRONT MATTER

Purpose and Scope 3
The Path to NCP Status 5
Audience and Key Roles 8

PART 1 — THE NCP MODEL

1 The NCP Model 9
2 The Physical Foundation 10

PART 2 — BUILDING THE INFRASTRUCTURE

3 Networking 12
4 Storage 15
5 Infrastructure-as-a-Service (IaaS) 18
6 Container-as-a-Service (CaaS) 21
7 AI Platform-as-a-Service (PaaS) 23

PART 3 — OPERATING AND SECURING

8 Operations 24
9 Security and Multitenancy 27

PART 4 — ORCHESTRATION AND GOVERNANCE

10 The Orchestration and Governance Layer 30
11 Rafay as a Reference Implementation 32

PART 5 — BRINGING IT TOGETHER

12 NCP Readiness Checklist 34
13 Validating Readiness 36
14 Glossary 37

APPENDIX

A Known Failure Modes 39

Purpose and Scope

An NVIDIA Cloud Partner operates a multi-tenant AI cloud built on the NCP hardware reference design. Delivering that service is a layered exercise: physical infrastructure is organized into standardized building blocks, networking and storage fabrics connect them, and successive software layers turn raw capacity into consumable services — Infrastructure-as-a-Service (IaaS), Container-as-a-Service (CaaS), and AI Platform-as-a-Service (PaaS).

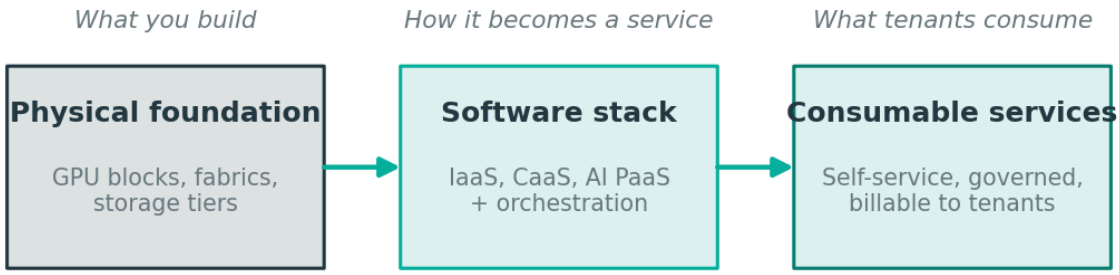


Figure 1. What an NVIDIA Cloud Partner is: a physical foundation turned into consumable services through a software stack.

This document states, layer by layer, the capabilities required to stand up and run that service.

What this document is

A capability reference. It enumerates what a multi-tenant AI cloud must be able to do at each layer so that an organization can assess its current stack against that set, locate gaps, and make informed build-versus-buy decisions. Part 5 consolidates the full set as a working checklist.

How to read it

Parts 1–2 cover what to know and what to build. Part 3 covers how to operate and secure it. Part 4 covers the orchestration and governance layer. Part 5 consolidates the capability set as a tiered checklist, sets out how readiness is validated, and records the failure modes worth designing against.

Readiness runs in parallel with hardware procurement, not after it: the software, networking, security, and operational capabilities in this guide are built alongside the physical build so the service is ready when capacity arrives.

Governing sources

This document is a synthesis. Where it abstracts, the authoritative detail lives in the sources below. NVIDIA publishes the NCP reference material at docs.nvidia.com/dsx/ncp.

Source	What it governs
NCP Hardware Reference Design	Physical building blocks, GPU and network topology, per-platform specifications, and sizing thresholds.
NCP Software Reference Guide	The layered IaaS / CaaS / AI PaaS capability model, break-fix architecture, telemetry, workload isolation, performance requirements, and the shared responsibility model.
NVIDIA Requirements for AI Clouds	Service-delivery expectations for partners supplying capacity: SLAs, compute and network provisioning, KaaS, security and identity, break-fix, telemetry, storage, transport and fabric visibility, and capacity and fleet management.

Source	What it governs
NVIDIA Inference Reference Architecture	Inference-specific deployment and serving patterns.
Certified-partner reference architectures	Storage and networking sizing validated against a specific vendor’s platform.

The Path to NCP Status

Becoming an NVIDIA Cloud Partner is a partner-program action, not solely an engineering outcome. NCPs are part of the NVIDIA Partner Network, and enrollment, capacity allocation, and designation are governed by NVIDIA through that program. What the engineering organization controls is readiness: whether the capabilities described in Parts 1–4 exist, are operable, and can be demonstrated on request.

The two tracks run in parallel rather than in sequence. Waiting for the program track to conclude before starting the capability track is the most common way an organization arrives at allocated capacity with nothing ready to run on it.

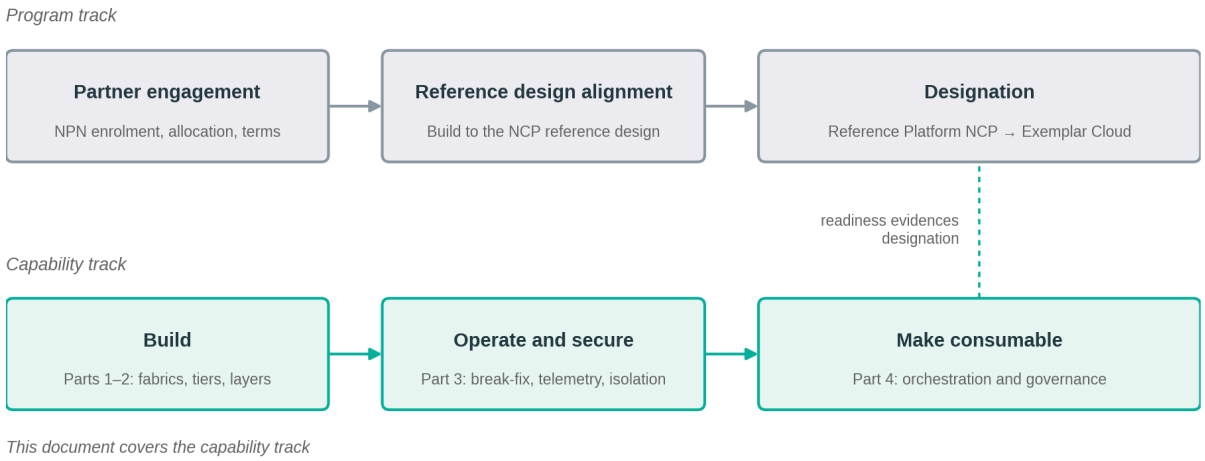


Figure 2. The program track and the capability track run in parallel; this document covers the second.

Sequencing the build

Readiness has a delivery date, and the date is not the day the racks land. NVIDIA publishes service-delivery expectations that place several capabilities well ahead of GPU delivery: transport established and the provisioning API demonstrably working roughly three months out, development capacity with that API integrated around six weeks out, and a bulk data path available a couple of weeks out so datasets can be staged before the cluster exists. Read backwards, those milestones say something uncomfortable: the software platform is due before the hardware.

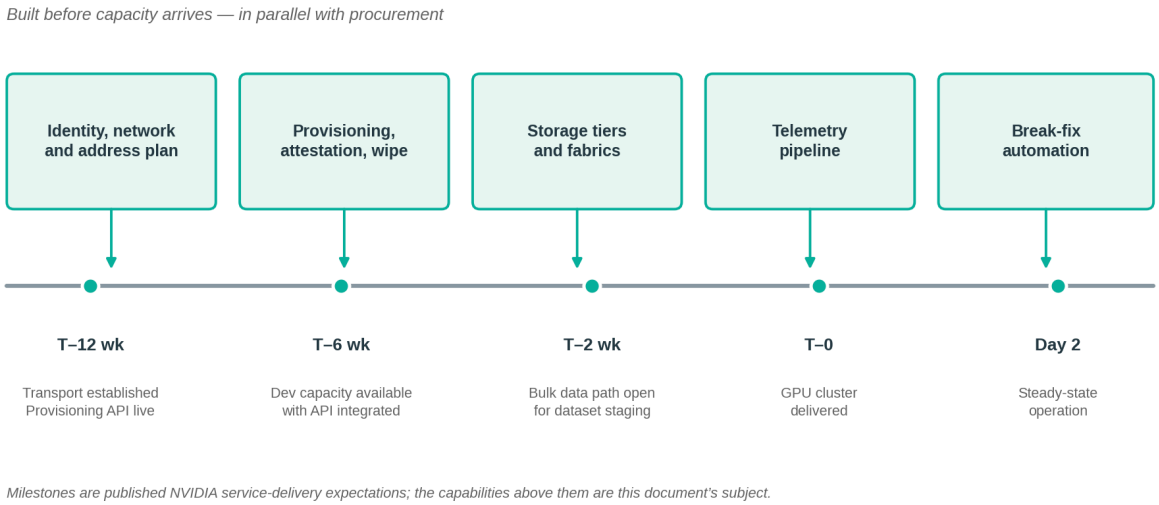


Figure 3. Readiness milestones run backwards from GPU delivery, not forwards from it.

The exact milestone definitions and any figures attached to them are set by NVIDIA and vary by engagement.

Within the capability track, the layers are not independent. Some things are cheap to build early and expensive to retrofit, and a few are hard prerequisites for capabilities that appear later in this document:

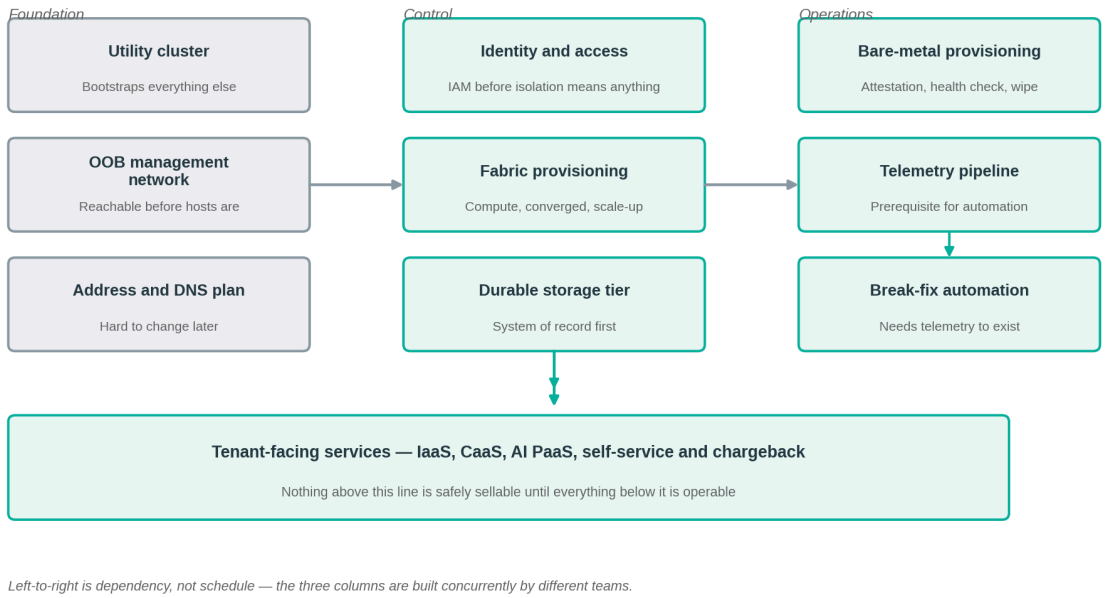


Figure 4. Build dependency order. Left to right is dependency, not schedule.

- **The utility cluster comes first** because it is what bootstraps everything else, including the systems used to provision the rest of the data center.
- **The address and DNS plan is effectively permanent.** Tenant address ranges, storage endpoints, and private service names are painful to renumber once tenants depend on them. Decide it before the first tenant network exists, not after.

- **Identity precedes isolation.** Tenant isolation, RBAC, quotas, and audit are all expressed in terms of identities. Building them against a placeholder identity model means rebuilding them.
- **Telemetry precedes automation.** Automated break-fix is a consumer of telemetry. An operator without a normalized, correlated signal pipeline can detect faults but cannot automate remediation, and will staff the gap with people instead.
- **The durable storage tier precedes the optional ones.** It is the system of record; acceleration and data-lake tiers extend it rather than substituting for it.

Nothing tenant-facing is safely sellable until the layers beneath it are operable. Selling ahead of that line is possible, it is also the most common source of the operational debt that shows up later as unpredictable performance.

Designations build on each other

“NCP” is not a single threshold. Several designations exist, each signalling something different to a prospective customer, and each established by a different mechanism.

Designation	What it signals	How it is established
NVIDIA Cloud Partner	Membership in the NVIDIA Partner Network as an AI cloud provider.	Program enrolment and certification requirement
Reference Platform NCP	The platform is built on the NCP reference design.	Alignment to the published hardware and software reference designs.
Preferred Cloud Service Provider	Elevated standing within the partner program.	NVIDIA program criteria.
NVIDIA Exemplar Cloud	Demonstrated, reproducible performance and resiliency on real AI workloads.	Benchmark validation against NVIDIA’s published recipes.

Exemplar Cloud: from declared to demonstrated

The capabilities in this document are declarative. An operator states that a control exists, and evidences it on request. Exemplar Cloud status is established by running NVIDIA’s published benchmarking recipes across representative workloads — inference, fine-tuning, and scaled pretraining — and producing results that can be replicated. Recipes and results are shared through NVIDIA’s DGX Cloud Benchmarking programme, which makes the comparison public rather than self-asserted.

A capability checklist gets an organization to a platform that works. Reaching Exemplar-grade requires that the platform also be tuned, instrumented, and stable enough to produce the same number twice, under load, on a different day.

The sections on performance-relevant design (§3.5, §5.5), observability (§8.2), and break-fix (§8.1) are where that difference is won or lost

> **Where the acceptance bar lives**
>
> Benchmark recipes, target figures, resiliency criteria, and the evaluation process for each designation are published and maintained by NVIDIA. They change between platform generations. Treat the capability set in this document as durable and the thresholds as versioned elsewhere.

Audience and Key Roles

This document is written for the organizations that build and operate AI clouds: neocloud operators standing up commercial GPU capacity, sovereign cloud builders delivering capacity under jurisdictional control, and enterprises building internal AI clouds for their own business units.

The architecture defines a set of roles that interact with the service. These roles recur throughout the guide and define who is responsible for each capability.

Role	Definition
Vendor	NVIDIA or a third party providing software, hardware, or services integrated into the NCP offering.
Operator	The entity that owns and operates the physical and virtual infrastructure. Responsible for IaaS, CaaS, and PaaS platform services, tenant isolation, security, and break-fix operations.
Tenant	A customer of the NCP that consumes resources within a cluster and either uses NCP-provided services directly or builds services on top of them.
Tenant Administrator	Manages the tenant's services on top of the NCP platform.
Tenant End User	Consumes the tenant's services. Includes AI practitioners who develop and deploy models and application users who consume AI-powered services.

PART 1

1 The NCP Model

Before building, establish the model the service follows: a layered stack viewed two ways, sitting on a standardized physical foundation.

1.1 The As-a-Service Stack

The architecture is layered. Each layer consumes the one beneath it and exposes a higher-level service to the tenant:

- **Bare Metal —** physical GPU and CPU hosts provisioned on demand.
- **Virtual Machines —** instances carved from hosts with hardware-accelerated access to GPU, network, and storage.
- **Managed Kubernetes —** container orchestration delivered as a managed experience on top of bare metal or VMs.
- **AI Platform —** tenant-facing training and inference services built on the layers below.

Compute, storage, and networking are allocated on demand rather than statically provisioned, enabling concurrent multi-tenant operation.

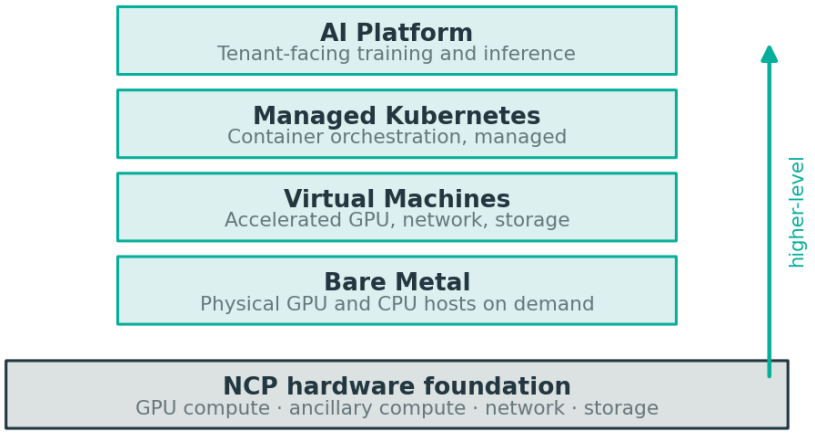


Figure 5. The layered as-a-service stack over the NCP hardware foundation.

1.2 The Tenant View

From the tenant's perspective, consumable compute is presented as the four layers above (Bare Metal, Virtual Machines, Managed Kubernetes, AI Platform), each backed by the underlying network, GPU compute, ancillary compute, and storage of the hardware reference design. A tenant selects the layer that matches its workload and isolation needs.

1.3 The Operator View

From the operator's perspective, the same stack is a set of services to run and manage: the workload-execution stack (the services tenants consume) and the control-and-management stack (the control planes that provision, isolate, monitor, and remediate). Two control planes are central to this view and are built in Part 2: the Software-Defined Networking (SDN) controller and the AI platform control planes.

2 The Physical Foundation

The service is built on standardized physical building blocks. Understanding them defines where each later capability runs.

2.1 Standardized Building Blocks

The data center is composed of standardized physical building blocks. GPUs are organized into dedicated GPU compute blocks. A separate core block hosts the supporting functions. A deployment scales by adding GPU compute blocks against a single core block.

2.2 GPU Compute Blocks

GPU compute blocks contain the accelerated compute. GPUs within a block are interconnected for high-performance, multi-GPU workloads. The GPU count per block and the number of supported blocks are defined in the applicable hardware reference design.

2.3 The Core Block

The core block provides the foundational functions that make GPU compute usable as a service:

Function	Role
Control Nodes	Compute for control planes and operator services.
General-Purpose Nodes	Non-GPU compute for user workloads and supporting services.
High-Speed Storage	Storage nodes or appliances serving the cluster.
Utility Cluster	Foundational element used to bootstrap the data center.
DC Edge Cluster	Networking interface to the outside world, including firewalls.

2.4 The Data Center View

Brought together, the data center is one or more GPU compute blocks attached to a core block, interconnected by the network fabrics defined in Section 3 and served by the storage tiers defined in Section 4. This is the canvas on which the software layers (Sections 5–7) are built.

2.5 Facility and Power Envelope

The software capabilities in this guide assume a facility that can physically host the platform. That assumption is worth making explicit, because for rack-scale GPU systems it is the binding constraint far more often than software is, and it is decided earliest.

- **Deployment size is gated by hall capacity.** The number of GPU compute blocks a site can host is set by available power and cooling, not by the software architecture. Configuration size is determined against hall power and cooling capacity, and the design is expected to start at a single scalable unit and grow without downtime.
- **Rack-scale systems require direct liquid cooling.** Current rack-scale GPU platforms are liquid cooled and deliver power over a DC busbar. This is a facility precondition, not a configuration option, and it extends to certified storage and networking racks that must comply with the same mechanical, power, and cooling reference design.
- **The scalable unit is a combined failure domain.** A scalable unit group is expected to include its own cooling and support infrastructure so that IT and facility failure domains align. Hot-air containment is aligned to row segments. Designing tenant placement and sparing without reference to those boundaries produces failure domains that cross physical ones.
- **Leak detection and thermal monitoring are operational signals.** Tray- and rack-level liquid leak detection and thermal state belong in the telemetry pipeline described in §8.2 and in the health model described in §8.1, not solely in a facilities dashboard.
- **Benchmark validation assumes uniform hardware at scale.** Performance validation is conducted against a uniform hardware cluster at a defined minimum scale. Heterogeneous or undersized clusters

cannot produce a comparable result, which makes fleet uniformity a procurement decision with downstream consequences for designation.

Where exact specifications live

Rack power draw, power-shelf configuration, cooling loop design, electrical distribution, thermal models, and the minimum cluster scale used for benchmark validation are published in the applicable NVIDIA hardware reference design and in the accompanying liquid-cooling, electrical-distribution, thermal-model, and controls-and-monitoring reference documents.

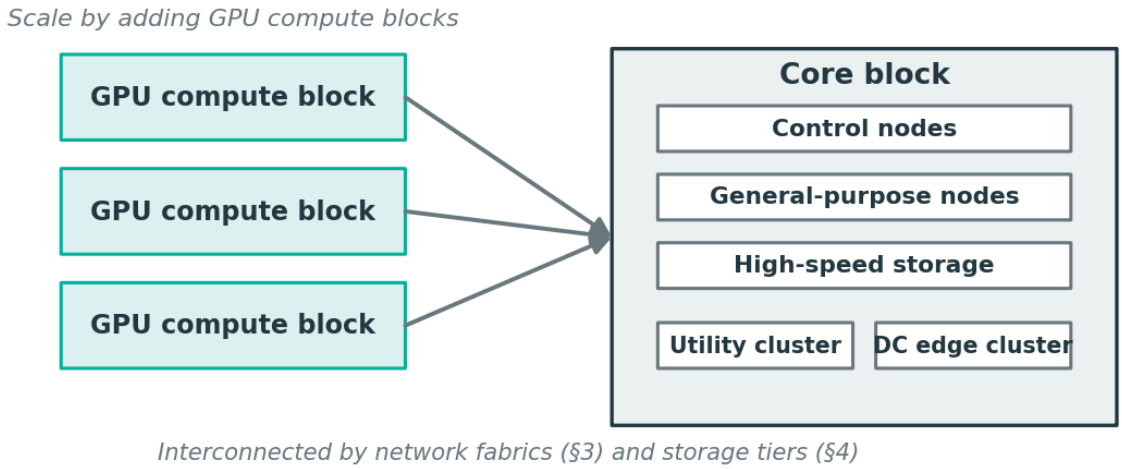


Figure 6. GPU compute blocks attach to a single core block that provides control, general-purpose, storage, utility, and edge functions.

PART 2

Building the Infrastructure

With the foundation defined, build the infrastructure in order: networking and storage first (the fabrics and tiers everything depends on), then the IaaS, CaaS, and PaaS software layers that turn capacity into services.

3 Networking

An AI data center runs several network fabrics, each with a defined purpose. Build network management to provision, configure, and isolate every fabric for multi-tenant use.

3.1 The Fabrics to Build

Fabric (by purpose)	Purpose	Typical Technology
Tenant access	North-south: tenant access to storage and external services.	Ethernet
Compute interconnect	East-west: GPU-to-GPU communication across nodes.	InfiniBand or Ethernet
Scale-up GPU interconnect	GPU-to-GPU communication within a rack.	Scale-up GPU fabric
Out-of-band management	Infrastructure management, separate from data paths.	Low-speed Ethernet
Shared services	Access to shared services such as registry and identity.	Logical overlay

The compute, converged (storage and in-band management), and out-of-band management networks are provisioned as distinct fabrics.

3.2 Required Network Management Capabilities

- **Ethernet fabric provisioning —** configure the leaf-spine fabric for tenant-access and Ethernet-based compute-interconnect traffic.
- **Ethernet visibility and telemetry —** monitor fabric health, validate changes, and troubleshoot.
- **InfiniBand fabric management —** configure the InfiniBand fabric for high-performance compute interconnect.
- **Scale-up domain management —** configure the rack-scale GPU interconnect.
- **Tenant network isolation —** create isolated tenant networks via overlays or fabric partitioning.
- **Virtual network lifecycle —** full lifecycle management of software-defined private networks, with stable private addressing and support for tenant-supplied address ranges where required.
- **Segmentation and filtering —** security-group or equivalent policy filtering with allow and deny rules scoped at workload, node, service, and tenant levels, with an auditable change trail.
- **Localized name resolution —** tenant-defined internal DNS so workloads resolve private endpoints such as storage, with network peering between tenant virtual networks where required.
- **Scale-up domain management —** management of the rack-scale GPU interconnect as a first-class fabric: a per-tray controller programming the switches, a telemetry agent per tray, and a centralized manager aggregating across racks, together with a fabric manager service that owns domain membership. Allocation, partitioning, and isolation within a scale-up domain are performed through that service's API.
- **Portable and reserved addressing —** movable addresses that survive instance replacement, and a stable, declared egress address pool so that tenant traffic leaving the platform presents a predictable identity to external services.
- **Link-layer encryption —** MACsec or equivalent protection on links that traverse infrastructure the operator does not physically control, including inter-site and carrier segments.

- **Geographic and policy filtering —** the ability to apply geolocation and sanctions or embargo policy to traffic, with a maintained threat and jurisdiction feed. For sovereign operators this is a compliance control rather than a security nicety.
- **Backend fabric visibility —** for each compute node, the ability to report programmatically which backend switches connect it toward the core, with every switch carrying a stable unique identifier. Without this, a consumer cannot reason about failure domains or placement, and the operator cannot correlate a fabric fault to the workloads it affected.
- **Unrestricted storage-plane reachability —** full-mesh layer-3 routing between all storage hosts and their clients, without restricted paths or hairpin routing. Parallel filesystems assume every client can reach every server directly; partial reachability degrades quietly rather than failing outright.

3.3 The Converged Network and Scaling Topologies

A converged network uses a central core to interconnect compute, storage, in-band management, and external connectivity. The topology scales in two stages:

- **Smaller deployments —** a two-tier (spine/leaf) Clos topology, with the core acting as the shared spine for compute, storage, and management.
- **Larger deployments —** a three-tier (core/spine/leaf) Clos topology, with dedicated spine/leaf networks per function and parallel core planes for redundancy.

The compute fabric is a rail-optimized, non-blocking fat-tree. Compute nodes are oversubscribed at the spine and sized to the storage-bandwidth delivery requirement.

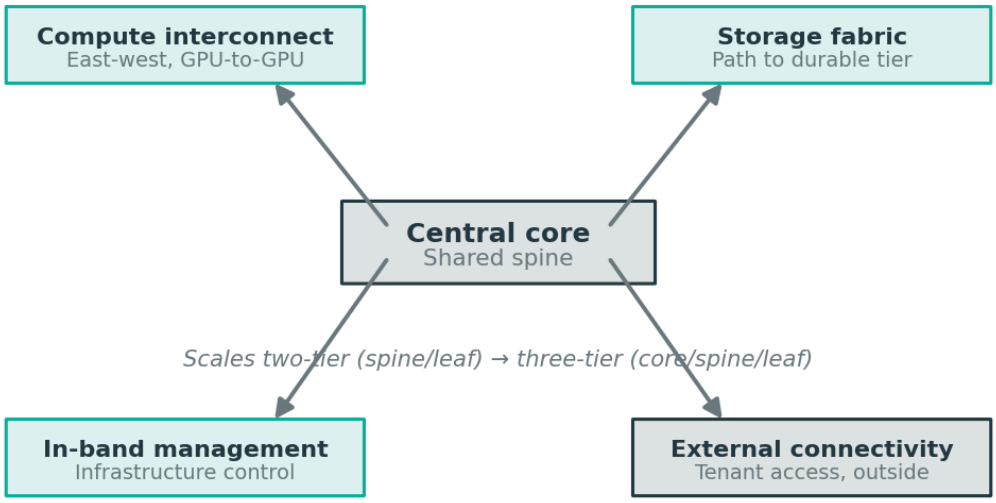


Figure 7. The converged network: a central core interconnecting compute, storage, management, and external connectivity.

> **Where exact specifications live**
>
> GPU-count thresholds for each topology, switch families and port counts, parallel-plane counts, and per-GPU bandwidth targets are published in the applicable NVIDIA networking and platform reference designs and in certified-partner guides.

3.4 InfiniBand and Ethernet Paths

The compute interconnect can be delivered over InfiniBand or Ethernet. InfiniBand fabrics use a centralized fabric manager to monitor health, detect congestion, and isolate tenants via partition keys. Ethernet fabrics use a schema-driven configuration model across the leaf-spine fabric and isolate tenants via VXLAN/EVPN overlays.

3.5 DPU-Accelerated Isolation

Offloading software-defined networking to a data processing unit (DPU) enforces tenant isolation without consuming host CPU cycles and establishes a hardware-rooted boundary between tenants and infrastructure. The DPU terminates overlay networking (VXLAN/EVPN) at the compute edge and applies per-tenant routing, segmentation, and policy in hardware before traffic reaches the host.

3.6 External Connectivity and Egress

A multi-tenant AI cloud is not a closed system. Tenants and the organizations consuming capacity need a private, high-bandwidth path into the platform, and the platform needs controlled paths out.

- **Private interconnect —** a dedicated private path between the platform and the networks of the organizations consuming capacity, sized to the data volumes involved and terminated at a defined point of presence rather than over the public internet.
- **Bulk data movement —** a capability to stage large datasets into the platform independently of, and ahead of, GPU capacity being available. This requires dedicated data-movement compute with high-performance networking, its own orchestration environment so the ingest stack can run before any GPU cluster exists, and — critically — mounts onto the same durable filesystem the GPU nodes will later use. An ingest path that lands data somewhere it has to be copied from again is not an ingest path.
- **Controlled egress —** outbound access from tenant clusters where required, from a declared address pool, subject to policy filtering, with no implicit public reachability inbound.
- **Corporate and administrative connectivity —** a separate, bandwidth-guaranteed path for operator administrative access, distinct from tenant data paths and from out-of-band management.

Operators supplying capacity to a specific large consumer will find these requirements stated concretely, including termination points and bandwidth expectations, in that consumer’s requirements document. The capability is general; the endpoints are not.

4 Storage

Storage is provided as an external, multi-tenant-capable service spanning several tiers. Build the durable tier first; add optional acceleration and onboarding tiers as workloads require.

4.1 The Storage Hierarchy

Storage is organized as a hierarchy of tiers describing the progression from on-package memory to shared storage. Performance is highest and capacity smallest at the top; capacity, sharing, and durability increase toward the bottom:

Tier	Description	Role
On-package GPU memory	High-bandwidth memory on the GPU.	Lowest latency, highest bandwidth, limited capacity.
Host memory	CPU DRAM.	Extends capacity beyond GPU memory at higher latency.
Node-local storage	NVMe SSDs attached to the compute host.	Greater capacity; scaling bounded by the node.
Shared acceleration tier (optional)	A shared, high-bandwidth tier between host and durable storage.	High-bandwidth shared access; trades durability for performance.
Durable shared storage (required)	Shared, durable system-of-record storage.	Durable storage for datasets and checkpoints.

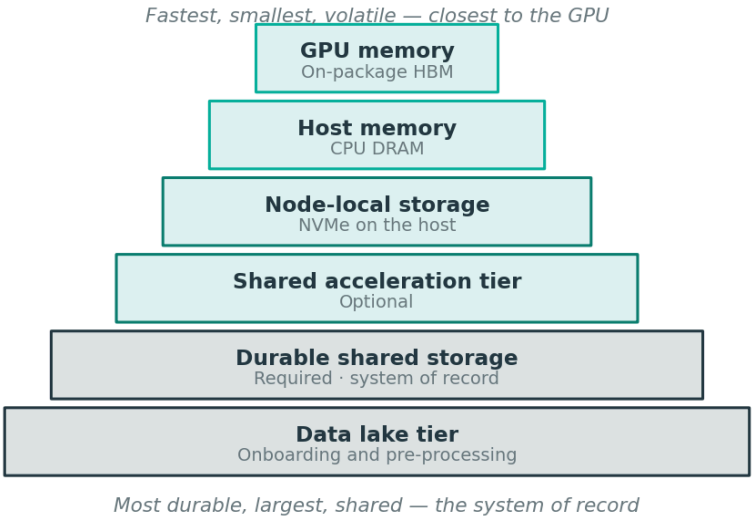


Figure 8. The storage tier hierarchy, from on-package memory down to durable shared storage and a data lake.

4.2 The Durable Storage Tier

A durable, high-performance shared storage tier is required for every deployment and serves as the system of record. It provides block, file, and object access for datasets and checkpoints. Throughput and capacity are determined per deployment by workload mix, concurrent-tenant count, per-tenant QoS, dataset size, model size, and checkpointing strategy. Certified storage partners publish reference architectures sized to standard performance guidance.

> **Where exact specifications live**
>
> Aggregate read/write performance guidance by GPU-system size, and the bandwidth, capacity, and power figures for the optional acceleration tier, are published in the applicable NVIDIA storage design guide and in certified-partner reference architectures.

4.3 The Optional Acceleration Tier

An optional shared acceleration tier may be added between host memory and the durable tier. It provides high-fan-out, low-latency access to shared state and is suited to inference patterns that reuse cached context. It is added only when bandwidth, concurrency, or persistence demands exceed what the durable tier sustains, and it extends the hierarchy without displacing the durable tier as the system of record. Specific implementation and integration requirements are defined in the applicable platform reference designs.

4.4 Data Lake Tier

A data lake tier supports onboarding new tenants and users, uploading and storing data, and pre-processing. It complements the durable and acceleration tiers rather than replacing them.

4.5 Storage Capabilities and Data Services

Capability	Purpose
Block Storage	Attach persistent volumes to instances.
File Storage	Shared filesystems for datasets and checkpoints.
Object Storage	S3-compatible storage for data lakes and artifacts.
Storage Tenant Isolation	Volume access control, namespace separation, per-tenant encryption.
Storage QoS	Enforce per-tenant bandwidth and IOPS limits to keep performance predictable.
High-Performance Data Path	Direct GPU-to-storage data movement that bypasses the CPU.
Storage Network Integration	Connect to NVMe-oF and RDMA-capable storage fabrics.
Multi-Node Coherency	Consistent view of shared filesystem state across many simultaneous client nodes.
Bulk Data Movement	Dedicated capability and compute for staging large datasets into the durable tier.
Stable Storage Addressing	Storage endpoints carry stable addresses so client mounts survive back-end change.
DPU-Attached Targets	Support for DPU-fronted storage enclosures where the design calls for it.

Data services to provide alongside the tiers: policy-enforced secure data access, encryption for data at rest and in transit, role-based access control for administrative functions, and non-disruptive upgrades to storage software and firmware.

Storage Service Management

Beyond the tiers themselves, durable and high-speed storage is delivered as a managed service. The capabilities below make that service operable, predictable, and safe for multi-tenant use:

Capability	Purpose
Provisioning API	Provision and manage storage through an API rather than manual configuration.
Quota Management	Soft and hard quotas by user, group, and project (directory), with usage accounting.
Parallel High-Speed Filesystem	A parallel or multi-path filesystem that scales to many simultaneous clients while sustaining requested performance.
Live Expansion	Non-disruptive expansion of capacity, inodes, and performance as the cluster grows.
High Availability	Tolerance of back-end component failure with continued client access, including client multipathing to storage servers.
Client Integration	Documented client support across the operating systems and kernels in use, including ARM and x86 hosts.

Capability	Purpose
Filesystem Semantics	Manageable root-squash, file locking (flock), and change-log auditing of file operations by user and group.
RDMA Memory Protection	Authorization-key-based memory protection for local and remote RDMA access.
Shared/Home-Directory Access	Shared filesystem access with POSIX semantics for identity- and group-based access control.
Multiple Fungible Filesystems	More than one filesystem provisionable within a total capacity envelope, so capacity can be divided between workloads or tenants rather than presented as one pool.
Fault Reporting Integration	Storage faults surface through the same break-fix interface as compute faults, so a consumer has one place to see infrastructure health.

5 Infrastructure-as-a-Service (IaaS)

IaaS provides bare metal and virtual-machine instances on demand. Build the IaaS layer by integrating software across the components below.

5.1 IaaS Components

Component	Description
Cloud Control Plane (CCP)	Tenant-facing API/UI to provision and manage compute, networking, and storage. Handles authentication, authorization, quota enforcement, and metering.
Software-Defined Networking (SDN)	Centralized, programmable network management for automation, scalability, and multi-tenant isolation.
Compute Service	Manages bare metal and VM instance lifecycle: provisioning, placement, monitoring, and sanitization between tenants.
Software-Defined Storage (SDS)	Provisions, pools, and scales block, file, and object storage.
Identity and Access Management (IAM)	Authentication and authorization for end users.

Across these components, the provisioning system is expected to be programmatically accessible — an API or CLI covering node lifecycle, network configuration, inventory and topology discovery, security configuration, and maintenance operations — with declarative interfaces for multi-step workflows. Resources carry stable, persistent identifiers and clear lifecycle states, and support user-defined tags and instance metadata.

Two interface details are worth stating explicitly because they are commonly left to default and are then difficult to change:

- **Out-of-band management protocol —** platform management should be exposed over a modern, authenticated, TLS-protected interface — in practice Redfish over TLS — with legacy IPMI disabled rather than merely firewalled. IPMI's authentication model cannot be secured to a standard appropriate for multi-tenant infrastructure.
- **Serial console —** console access for troubleshooting, at minimum read-only and preferably interactive, with console output captured and retained for a defined period. The retained output is what makes a boot failure diagnosable after the fact rather than only while someone is watching.

5.2 Software-Defined Networking

SDN separates the network control plane from the data plane so network behavior is configured programmatically through an SDN controller rather than by manual switch configuration. The controller translates tenant intent into hardware configuration, assigns VPCs, and programs accelerated NICs into instances.

A standard sequence for attaching hardware-accelerated networking to a VM:

1. The VM service makes an intent-based request to the SDN to add the VM to a VPC.
2. The SDN performs the mapping — assigns the VPC and configures resources.
3. The SDN triggers a PCIe hotplug event, exposing a new SR-IOV virtual function to the host kernel.
4. The SDN controller binds the virtual function to the VM, which then communicates with the NIC hardware directly.

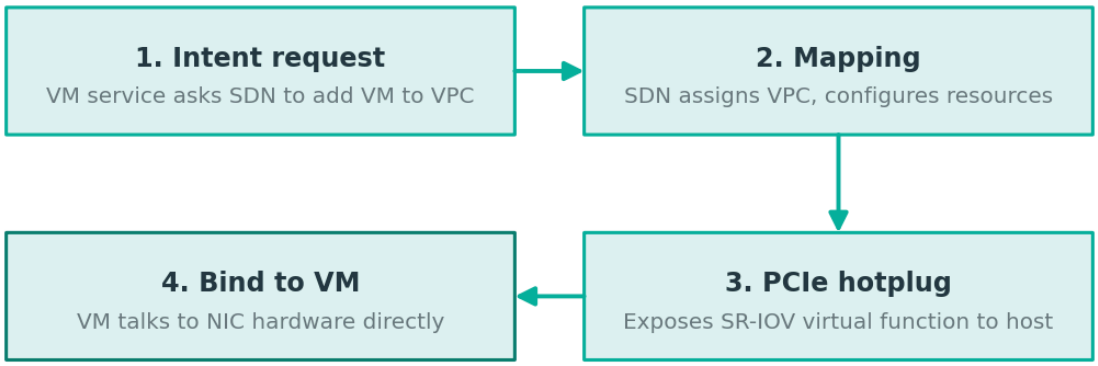


Figure 9. The SDN sequence for attaching hardware-accelerated networking to a VM.

5.3 Compute Service

The Compute Service manages the full instance lifecycle and must always deliver a known-good node to a tenant. Required capabilities:

Capability	Purpose
Machine Discovery	Discover and ingest physical machines into inventory.
Bare Metal Provisioning	PXE boot, firmware updates, and OS installation at scale.
Instance Lifecycle	Create, start/stop, terminate, reboot, and place instances.
Instance Management	Status, volume attach, IP assignment, and vNIC attach.
Hardware Attestation	Verify node and firmware integrity prior to allocation.
Node Health Check	Stress test to verify performance before allocation.
Node Sanitization	Securely wipe nodes between tenants.
GPU Virtualization	Share GPUs across VMs (time-sliced or MIG).
GPU Observability	Monitor GPU health, metrics, and diagnostics.
Rack-Scale Lifecycle	Power sequencing, cooling, and coordinated firmware.

Logical Components of the Compute Service

The capability table above states what the service does. A specific implementation may partition responsibility differently, but these components are usually present and are a useful decomposition when assigning ownership:

Component	Responsibility
Machine Lifecycle Manager	Discovery, ingestion, placement, provisioning, sanitization, firmware update, BMC management, address allocation, OS install.
Instance Database	Authoritative state for every physical machine and instance, including lifecycle state and stable identifier.
VM Control Plane	Receives provisioned bare-metal nodes and services VM requests: bin-packing, placement, event monitoring. Placement decisions apply at node level and again at rack and scalable-unit level to preserve bandwidth and latency.
Provisioning Boot Service	Responds to a booting node, loads a minimal OS and provisioning agent, then tests the node, updates firmware, and installs the bare-metal or hypervisor image.

Two allocation constraints deserve explicit treatment. Nodes must carry stable hardware identifiers that persist across reprovisioning, because break-fix, diagnostics, and maintenance history are keyed to them. And in rack-scale deployments, allocation must be aware of scale-up domain boundaries — a multi-node job placed across two domains will not perform like the same job placed within one, and a tenant allocation that straddles a domain boundary weakens the isolation guarantee the domain provides.

5.4 Software-Defined Storage

SDS exposes ephemeral and persistent storage to instances. Ephemeral storage uses local drives erased between tenants; persistent storage provides block, file, and object access. An SDS controller manages provisioning, pooling, and scaling and detects failures, recovery, and upgrades; storage targets and data-plane services (snapshots, imports) complete the layer.

5.5 GPU Exposure Approaches

To deliver optimal AI performance, give the instance direct access to GPU hardware. Select the exposure approach by use case, noting that isolation strength decreases down the table — an exclusive GPU is the strongest boundary, hardware partitioning is weaker than exclusive assignment, and time-slicing is the weakest. Assigning a full GPU to a VM, and letting that VM decide how to expose it to containers, is the appropriate default.

Use Case	Approach	Notes
Exclusive GPU for a VM	Device passthrough	Full GPU assigned to a single VM. Strongest isolation.
Exclusive GPU to a container	Runtime device access	GPU exposed to a container via the runtime.
Partitioned GPU for VM or container	Hardware partitioning	GPU divided into isolated instances. Weaker than exclusive assignment.
Fractional GPU	Time-slicing	A single GPU shared across consumers. Weakest isolation.

Density follows from this. Where a node’s GPUs are being served to tenants, the appropriate design is a single virtual machine per GPU node rather than several — subdividing a GPU host across multiple VMs reintroduces the contention and isolation problems that dedicating the node was meant to avoid, and costs performance on the interconnect. General-purpose CPU nodes are the opposite case: multiple instances per host is normal, with instance shape selected against memory and core count.

Two further performance constraints apply wherever workloads run inside virtual machines. First, the optimized PCIe and GPU interconnect topology of the underlying certified system must be preserved and exposed inside the VM rather than flattened by the hypervisor; vCPU pinning and NUMA-aligned memory placement belong to the same set of decisions. Second, the tenant overlay should traverse a single tunnel so payloads are not encapsulated twice, and the VM orchestrator should be topology-aware so that distributed jobs land in a placement that keeps collective operations local.

6 Container-as-a-Service (CaaS)

CaaS delivers Kubernetes as the workload-orchestration layer. Choose a delivery model, then build a GPU-optimized managed Kubernetes that meets the required capabilities.

6.1 Kubernetes Delivery Models

Model	Description
Provider-managed	The provider operates a dedicated cluster per tenant, handling control-plane lifecycle, upgrades, and scaling. Tenants receive cluster access credentials.
Platform-managed	A management platform provisions and manages clusters on provisioned infrastructure, providing operator and tenant portals and multi-tenant orchestration.
Customer-managed	The provider supplies bare metal or VMs; customers install and manage their own clusters for maximum flexibility.

6.2 Required Managed-Kubernetes Capabilities

- Abstracts control-plane nodes so consumers specify only the required high-availability and scalability for the control plane.
- Supports a Kubernetes version recent enough to enable dynamic resource allocation for flexible GPU sharing, together with the memory-export mechanism that allows addressing across a rack-scale scale-up domain. Both are version-gated; a defined minimum version applies and moves over time.
- Allows consumers to bring their own GPU-optimized node OS, or provides one integrated with managed services.
- Supports managed node groups and cluster node autoscaling.
- Provides standard integration with high-performance storage (CSI) and networking (CNI) optimized for the provider's services.
- Supports topology discovery for worker nodes in topology-aligned clusters, enabling topology-aware gang-scheduling for distributed training and disaggregated inference.
- Complies with CNCF Kubernetes certification and the emerging Cloud Native AI conformance initiative.
- Exposes cluster and node-group management through declarative interfaces, with autoscaling and GPU enablement delivered by operators rather than bespoke scripting.
- Supports several clusters within the same customer tenancy, including more than one cluster attached to the same virtual network, so that a customer can separate environments without being given a second account.
- Supports Kubernetes API extension mechanisms — custom resource definitions together with validating and mutating admission control — since the operators and policy tooling above depend on them.

Control-Plane Architecture and Assurance

The control plane is the part of a managed Kubernetes service the tenant cannot fix, which makes its construction the operator's problem in a way the data plane is not:

- **High availability with datastore separation —** a redundant control-plane architecture in which the cluster datastore — etcd in most distributions — is separated from the API server tier rather than co-located, so that datastore pressure and API load fail independently.
- **Encryption of control-plane state —** encryption at rest for the cluster datastore and for Kubernetes secrets specifically. This is a separate control from the disk-level encryption in §9.3; disk encryption does not protect secrets from anything with read access to the datastore.
- **Sized and bounded control planes —** the ability to size or pin a control plane to a defined load envelope, so that its capacity is a known quantity rather than an emergent one, and so a noisy tenant cannot degrade it by accident.

- **Demonstrated scale conformance —** the standard upstream Kubernetes performance and scalability test suite, run at the maximum cluster size the service offers. Conformance certification establishes correctness; the scalability suite is what establishes that the offered cluster size is real.

Cluster Operations

A managed Kubernetes service is defined as much by how it is operated as by its architecture. The operational capabilities below keep clusters current, available, and recoverable:

- **Lifecycle automation —** API- or CLI-driven create/update/delete and scaling for both the control plane and node pools, including default labels and taints, with a strong preference for declarative (e.g. Terraform) interfaces.
- **Version support policy —** a defined window of supported minor versions, timely availability of new upstream releases, automated control-plane security patching, and clear deprecation and end-of-life notice periods.
- **Non-disruptive upgrades —** minor-version control-plane upgrades without application downtime, and user-initiated rolling node upgrades that respect Pod Disruption Budgets.
- **Backup and disaster recovery —** recovery to defined, testable objectives.
- **Security-response participation —** participation in the Kubernetes security-response and disclosure process, with patches applied to the managed service ahead of public disclosure where guidance permits.
- **Control-plane observability —** API-server metrics in a standard scrapable format and viewable or exportable control-plane logs.

6.3 Cluster Operators for GPU Acceleration

Operators extend the cluster through Custom Resource Definitions and automate GPU-accelerated capabilities. The functions to provide as integration points:

Function	What it automates
GPU enablement	GPU driver, runtime, and device-plugin deployment.
Accelerated networking	RDMA, SR-IOV, and GPU-direct networking resources.
DPU management	DPU lifecycle, firmware, and runtime, coordinated with networking.
Inference microservice deployment	Deployment and lifecycle of inference microservices for generative AI.

6.4 Multitenancy and Isolation

Clusters that provide tenant API access are not shared between tenants; each tenant receives a dedicated control plane to preserve isolation. The physical worker hosts running those control-plane VMs can be shared to improve utilization when the isolation mechanism is sufficient. Bare metal provides the strongest node-level isolation; virtual machines fit the majority of needs at lower cost. Where multiple tenants share a node, a robust isolating mechanism (hardened hypervisor, DPU) is used, and configurable hypervisor protections guard GPU, CPU, memory, network, storage, and IP resources. When infrastructure passes between tenants, the operator sanitizes it — wiping memory, restoring firmware to a known-good state, and erasing ephemeral drives.

7 AI Platform-as-a-Service (PaaS)

The AI platform layer presents tenant-facing training and inference services. The reference architecture runs a wide range of industry AI platforms; select the platforms that match each tenant's deployment model.

Workload Type	Suited Approach	Description
Shared, multi-team GPU scheduling	Kubernetes-based orchestration platform	Maximizes utilization and provides fair resource allocation across teams via fractional sharing and queuing.
Serverless inference	Serverless inference platform	Deploys models as auto-scaling API endpoints, abstracting GPU infrastructure management.
Single-tenant HPC and large-scale training	HPC workload manager	Mature job queuing, prioritization, and GPU-aware scheduling for long-running distributed training.
Model customization and agent development	AI development platform	Data preparation, fine-tuning, RAG pipelines, guardrails, and evaluation, delivered as containers on Kubernetes or bare metal.

Containerized models packaged to open standards run on the GPU clusters provisioned by the layers below, with direct GPU, network, and storage access for optimal performance.

7.1 What the Platform Layer Has to Provide

Whichever platforms are selected, the operator is accountable for the substrate they run on. The requirements below are platform-independent:

- **Tenant-requestable primitives —** endpoints, models, Kubernetes capacity, GPU workers, and model storage requested through an interface, without the tenant needing to understand the physical layout of the data center.
- **Model artifact handling —** a registry path for signed, validated, attested container artifacts, with the storage tier sized for model pull at scale rather than as an afterthought.
- **Elastic serving behaviour —** scale-to-load for inference endpoints, including scale-down, since idle GPU behind a served endpoint is the dominant cost error in inference platforms.
- **Queueing and fair share —** for shared multi-team environments, admission control and fair allocation so one team cannot starve another.
- **Mixed-mode coexistence —** training and inference workloads on the same fleet have different placement, checkpointing, and interruption characteristics; the platform layer is where that is reconciled.

Inference deserves particular attention because it is where most operators' service revenue accrues and where the platform behaves least like a static cluster. NVIDIA publishes a dedicated Inference Reference Architecture covering serving topologies, disaggregated inference, and platform patterns, and a further requirements document specific to hosting inference service providers on rack-scale systems.

PART 3

Operating and Securing

With the layers built, operate and secure the service so tenants receive reliable, isolated, and verifiable infrastructure.

8 Operations

8.1 Break-Fix

The break-fix system keeps clusters available by detecting, triaging, remediating, and validating GPU nodes — automating remediation wherever possible to minimize human intervention. It is a day-two system: day two begins once the data center is online, validated, and ready for normal tenant operation.

Responsibility shifts with node assignment, and the architecture is organized around two logically separate domains — the tenant domain, covering operations performed while a node is in a tenant's possession, and the infrastructure operator domain, covering operations performed while the node is under operator control:

- **Tenant domain —** continuous in-band health checks run on a heartbeat, which requires the tenant to opt in and participate in the break-fix flow. Many faults are remediated in place without removing the node from circulation, for example by resetting a GPU. A node that cannot be remediated in place is returned to the operator domain, which may mean a replacement node or a temporarily reduced capability set for that tenant.
- **Operator domain —** initial node validation before first vending, so the service always hands a known-good node to a tenant; remediation of identified faults, as automated as possible; and return of the node to circulation.
- **Crossing the boundary —** continuous out-of-band checks are performed by the operator domain even while a node sits in the tenant domain. Thermal state queried through the management controller, DPU heartbeat, and log-file monitoring are typical examples. Keeping the two domains separate while allowing this one exception is the part most often implemented incorrectly.

Health checks run at the level where the needed signal is available, because some information exists only at a particular layer — many checks need direct access to the GPU driver, which may sit at the host OS or be passed through to a VM. The usual structure is a harness that runs local checks on a heartbeat and interfaces with the break-fix control plane, plus the check set itself, where some checks run every interval and others run rarely or only on request. No specific tooling is mandated, though a GPU-manager agent is generally assumed as the source of GPU-level health, metrics, and diagnostics.

Break-fix is usually built for compute and then never extended, which leaves storage faults reported through a different channel on a different cadence. A consumer holding one health view for GPUs and a vendor portal for storage cannot correlate the two during an incident. Storage faults belong in the same interface, with the same event schema.

The break-fix function is exposed programmatically so consumers can act on it and reason about it: lifecycle actions (power-cycle a node, reset a GPU, cordon a node, return a node or rack, request a replacement when health thresholds are breached), a queryable record of upcoming, current, and historical maintenance and repair events, and diagnostics that identify installed hardware by stable identifier and report firmware versions.

8.2 Observability and Telemetry

A unified observability solution provides visibility across the data center and supports tenant SLAs. It is built on three signal types, harvested across the data center, normalized, and correlated through a vendor-neutral telemetry pipeline:

- **Logs —** event records from applications, system services, and hardware.
- **Metrics —** measurements such as latency, throughput, and GPU utilization.
- **Traces —** the end-to-end path of requests across distributed systems.

Correlation by timestamp, resource ID, and trace ID relates signals to each other and attributes them to tenants or services.

Delivery matters as much as collection. Where telemetry is exported to a capacity consumer, the expected mechanism is a vendor-neutral wire protocol — in practice OpenTelemetry's protocol — under a bounded delivery latency, with the operator providing a written confirmation of its ability to deliver the specified metrics and logs once the consuming organization issues its specification. That written response is part of the requirement, not an administrative step around it.

Coverage is expected across every fabric the service runs:

- Tenant-access (north-south, front-end) network.
- Compute-interconnect (east-west, back-end GPU) network.
- Management network.
- Scale-up switching fabric, on rack-scale platforms.
- Host-level signal.

And across the log sources that actually reveal infrastructure health — scale-up fabric manager logs, subnet manager logs, fabric management platform event logs, general switch logs and syslogs, switch kernel logs, hardware event logs, and host system logs. An observability stack that covers the first list and not the second produces dashboards that look complete and cannot explain an outage.

Exact metric names, label schemas, latency ceilings, and delivery targets are agreed with the consumer and published in the applicable telemetry specification.

8.3 Lifecycle and Upgrades

Kubernetes nodes signal readiness through a schedulable state. Operators support break-fix procedures and sparing strategies for individual GPU nodes and for entire rack-scale fault domains, and apply firmware, software upgrades, and reboots through the break-fix flow to maintain continuity.

8.4 Operational Support

Operating an AI cloud is a support relationship as much as an infrastructure one. The capabilities below define how the operator supports the consumers of the service:

- **Named contact and escalation —** a dedicated technical point of contact and a defined escalation path for the infrastructure consumer.
- **Monitored support channel —** a real-time support channel monitored by a technical specialist.
- **Coverage and severity handling —** around-the-clock support aligned to defined incident-severity procedures.
- **Incident and maintenance communication —** proactive communication of service-impacting incidents and of planned and unplanned maintenance.
- **Scheduled-maintenance controls —** the ability for consumers to schedule or defer planned maintenance windows programmatically to avoid unexpected disruption.
- **Vulnerability remediation —** timely remediation of critical vulnerabilities with transparent disclosure.

Service-level commitments sit on top of this support model, and there is usually more than one of them. Operators supplying capacity at scale are typically asked for separate, independently measured commitments across at least four surfaces:

Commitment surface	What is committed
Managed Kubernetes control plane	Production availability, financially backed.
Home-directory storage	Availability for unplanned incidents, excluding scheduled maintenance, plus a durability commitment.
High-speed storage service	Availability measured against a rolling service-level objective, exclusive of maintenance, with non-disruptive upgrades and a consumer right to defer maintenance windows.
High-speed filesystem	Availability and durability expressed per unit of capacity.

Three properties matter more than the numbers. Each commitment must be measurable by an agreed method, not asserted. Scheduled maintenance is excluded from availability only if maintenance scheduling is itself controllable and communicable, which is why the scheduled-maintenance control above is a prerequisite rather than a convenience. And durability is a separate commitment from availability; conflating them is a common and expensive drafting error.

The numeric targets, measurement windows, and remedies are set in the agreement between operator and consumer, and NVIDIA publishes its own service-delivery expectations for partners supplying capacity. This document treats the capability to measure, report, and act against such commitments as in scope, and the specific figures as out of scope.

8.5 Capacity and Fleet Governance

At fleet scale, capacity must be governable as well as operable. The operator provides a consistent view of what capacity exists, its health, and how it is allocated:

- **Fleet-health metrics —** track capacity by state: delivered and available, healthy and meeting SLA, reserved, and currently in use, attributable to account, project, or tenant.
- **Programmatic resource discovery —** newly delivered capacity is discoverable through a centralized resource index rather than manual handoff, each resource carrying a stable identifier.
- **Capacity reservations —** the ability to group and pin compute, network, and storage to an account, and to allocate a topology block as a single unit so all resources share the same performance and security boundaries.
- **Unified health view —** a single source of truth for the health of physical hosts and logical primitives (cluster, node group, reservation) so broad failures can be localized quickly.
- **Failure-domain observability —** visibility into which resources share a physical failure domain, so that a correlated failure can be recognized as one event rather than investigated as many.

These are API capabilities, not reports, and the shape of the response is part of the requirement. A resource-governance interface is generally expected to return, per node: a stable node identifier, health state, instance identifier, creation timestamp, hardware type, GPU count, and the account and project the resource is attributed to. Capacity discovery is expected to work through a centralized resource index rather than through human handoff over chat or email. Health aggregation is expected at cluster, node-group, and reservation level as well as per host, covering GPU state, thermal status, and memory health.

Break-fix is subject to the same discipline. The interface is expected to support power-cycling a node or resetting an instance, resetting an individual GPU, marking a node unschedulable without disrupting running work, returning a node or rack for maintenance, querying upcoming, current and historical maintenance events, and querying retirement notices. Maintenance events themselves carry a schema — open, update and close timestamps, a stable hardware identifier, hardware category, fault description, and action taken — and a diagnostics interface exposes installed hardware by stable identifier, obfuscated where necessary, together with component firmware versions.

9 Security and Multitenancy

Secure multi-tenant infrastructure rests on a verifiable boot chain, enforced tenant isolation, protected data, and a clear division of responsibility.

9.1 Secure Boot and Attestation

Take a zero-trust approach to every processor that runs an OS in the data center, including BMCs, DPUs, and host processors. Implement a secure and measured boot solution that can attest to its state:

1. **Secure Boot —** establishes a chain of trust in which each privileged software stage is signed by a trusted vendor or operator and verified before the next loads, so only trusted code runs.
2. **Measured Boot —** generates a hash (digest) of each loaded image and extends it into a Trusted Platform Module, producing an audit trail that lets the control plane verify the expected software sequence loaded.
3. **Attestation —** uses that trail to confirm node and firmware integrity prior to allocating a node to a tenant.

A hardware root of trust (TPM 2.0) is present on all platforms, firmware is cryptographically signed and brought to a known-good state between tenants, and UEFI Secure Boot is enabled. Remote platform management follows secure defaults over encrypted transport.

9.2 Tenant Isolation

Keep nodes aligned to a single tenant where possible for the strongest isolation and most efficient GPU cooperation. Provide each tenant a dedicated Kubernetes control plane so a misbehaving tenant cannot impede another. When tenants share a node, enforce isolation with a hardened hypervisor or DPU and protect against resource-contention, memory, and cache-line attacks. Offloading network isolation to the DPU enforces per-tenant separation at the compute edge in hardware. Sanitize all infrastructure when it passes between tenants.

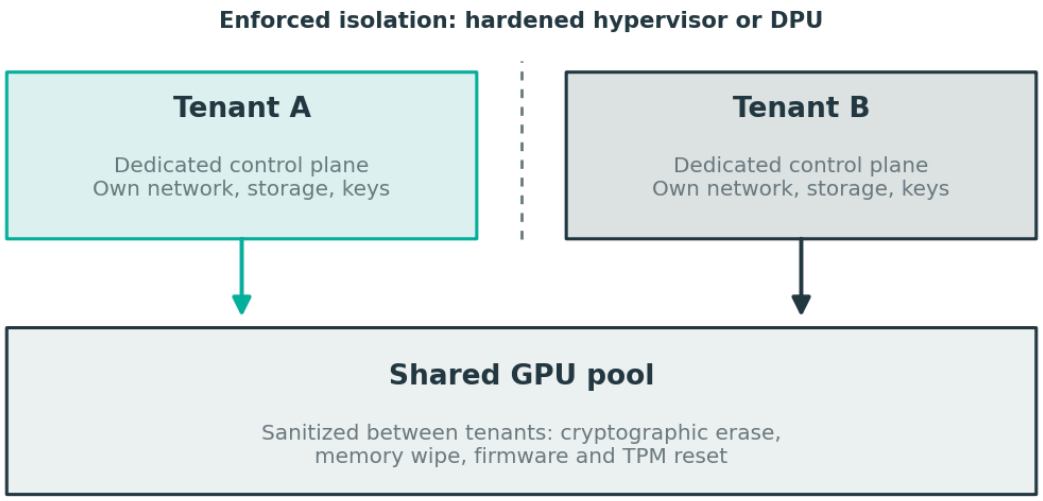


Figure 10. Two tenants isolated on a shared GPU pool, with sanitization when infrastructure passes between them.

Sanitization between tenants includes cryptographic erase of data drives, wiping of persistent and volatile memory, and resetting platform firmware and the TPM to a known-good state, with attestation that the node is clean before it is reallocated.

9.3 Data Security

Protect data with policy-enforced access controls, encryption at rest and in transit, and role-based access control for administrative functions. Containerized models packaged as signed, validated, and attested artifacts support software-supply-chain integrity, and policy-enforcement tooling keeps workloads running with least privilege.

Encryption at rest covers all data drives, including via self-encrypting drives; encryption and mutual authentication protect traffic in transit. Cryptographic keys and certificates follow a managed lifecycle — secure issuance, storage, rotation, and revocation — across platform services.

9.4 Identity and Access

Identity is the control point for a multi-tenant service. The platform provides a standards-based identity and access model spanning users, workloads, and services:

- **User authentication —** standards-based authentication (OIDC) with validation of issued tokens for identity and authorization decisions.
- **Workload and service identity —** authenticated in-cluster identities using short-lived credentials, authenticated out-of-cluster service accounts, and OIDC-based federation, including Kubernetes service accounts.
- **Authorization —** least-privilege, role-based access control with granular actions and scopes across managed services and infrastructure.
- **Directory integration —** integration with a directory service (LDAP) so identities and group membership resolve for dependent services, including identity- and group-based access control on storage.
- **Administrative protection —** multi-factor authentication on all administrative interfaces (UI, CLI, API).
- **Audit —** audit logs for security-relevant events — control-plane API calls, authentication events, and authorization decisions — retained and accessible to authorized operators.

9.5 Network Security Controls

The network perimeter and management plane follow defense-in-depth practices:

- **Private by default —** no public internet access by default; API endpoints, including the Kubernetes API server, are reached over private paths or restricted by firewall.
- **Out-of-band management isolation —** management interfaces (BMC) sit on a dedicated, restricted network reachable only through a hardened bastion, never directly from the public internet.
- **Traffic filtering —** security groups or user-customizable access-control lists filter traffic, with the enforcement technology and its placement in the packet path clearly specified.
- **Encryption in transit —** mutual TLS for platform and service traffic, east-west and north-south, with link-layer protection such as MACsec on segments crossing infrastructure the operator does not control.
- **Retained audit trail —** security-relevant events retained for a defined period and accessible to authorized operators and, where agreed, to the capacity consumer.

9.6 Compliance

Independent assurance underpins the security model. A recognized controls attestation (such as SOC 2, Type 1 or better) covering security, availability, and confidentiality across services and data-center infrastructure provides third-party evidence that the controls described in this section operate as stated. Operators serving regulated or sovereign customers layer jurisdiction-specific frameworks on top of that baseline rather than in place of it.

9.7 The Shared Responsibility Model

Security and compliance are shared across roles. The division is fixed and defines who owns each control:

Role	Responsibility
Operator	Everything from the host operating system and virtualization layer up through orchestration and platform services, plus network and firewall configuration, server-side encryption, and physical facility security. Ultimately responsible for tenant isolation and preventing cross-tenant data leakage.
Tenant Administrator	Privileged access for tenant users to each service; configuration of the tenant's own application software; integration with external parties; and the tenant's IT environment and regulatory compliance.

Role	Responsibility
Tenant End User	Application security, user authentication, and the security of data generated or used by the application.

PART 4

Orchestration and Governance

The layers in Parts 2–3 produce infrastructure capacity. They do not, by themselves, produce a service a customer can buy. The orchestration and governance layer is what converts capacity into repeatable, consumable, billable services — and it is the layer most operators underestimate, because none of it is visible in a hardware reference design.

This part states the capability requirements of that layer in the same terms as every other part, then describes Rafay as one reference implementation of them.

10 The Orchestration and Governance Layer

Sitting between the infrastructure layers and the customer, this layer answers a different question from the ones below it. Parts 2–3 ask whether capacity exists and is healthy. This layer asks who is allowed to consume it, on what terms, in what shape, and at what price.

10.1 Required Capabilities

Capability	Requirement
Tenant and customer administration	Create, configure, suspend, and remove customers and their sub-organizations as first-class objects, with delegated administration so a customer can manage its own users without operator involvement.
Self-service consumption	Tenants request environments, clusters, and services through a portal or API and receive them without an operator ticket in the path.
Service definition and cataloguing	Package compute, Kubernetes, storage, policy, and applications into named, versioned offerings that can be exposed selectively to different customers from one underlying stack.
Policy enforcement	Apply access, network, cluster, and workload policy consistently across tenants, with enforcement at admission rather than by audit after the fact.
Quota and entitlement control	Bound consumption by customer, project, and resource class, with soft and hard limits and predictable behaviour at the limit.
Environment segmentation	Separate customers, and separate a customer's own environments, without duplicating platform operations effort per environment.
Usage metering and attribution	Meter consumption and attribute it to account, project, and resource with sufficient fidelity to support billing and internal chargeback.
Chargeback and showback	Present cost or consumption back to the consuming organization in terms it can act on.
Activity audit	A durable, queryable record of who did what, across both operator and tenant actions.
Kubernetes fleet lifecycle	Create, upgrade, scale, and retire clusters across many customers as a fleet operation rather than a per-cluster one.
Underlay automation	Coordinate the layers below — bare metal, network, storage — so that a self-service request results in provisioned infrastructure without manual assembly.
Deployment flexibility	Operate as a hosted service or fully self-hosted, including disconnected and jurisdiction-constrained environments, without a functional cliff between the two.
White-labelling and embedding	Surface capability inside the operator's own commercial front end rather than forcing customers into a separately branded product.

10.2 Why This Layer Decides the Business Model

Two operators can hold identical hardware, identical fabrics, and identical Kubernetes, and sell entirely different businesses. The difference is expressed here. An operator without service definition sells raw capacity at

whatever the market rate is. An operator with it sells differentiated offerings at different margins from the same fleet.

The governance capabilities carry the same weight for a different reason: they are what makes multi-tenancy commercially safe. Quota control, policy enforcement, and attribution are the mechanisms by which one customer cannot consume another's capacity, exceed its entitlement unnoticed, or generate cost that cannot be assigned. Without them an operator is running a shared cluster and calling it a cloud.

10.3 Build or Buy

This layer is buildable. It is also, in the experience of most operators who have attempted it, the layer where internally built tooling ages worst — because its requirements grow with every new customer, every new service, and every new regulatory constraint, while the infrastructure layers beneath it remain comparatively stable. The relevant question is not whether an engineering team can build a tenant portal; it is whether the operator wants a standing internal product team maintaining one.

Section 11 describes one commercially available implementation. Operators evaluating alternatives, including building, should assess candidates against the capability table in §10.1 rather than against a feature list.

11 Rafay as a Reference Implementation

Rafay operates between NVIDIA-aligned infrastructure and customer-consumable GPU cloud services. It implements the orchestration and governance capabilities set out in §10.1.

11.1 Capability Mapping

Mapped against the requirements in §10.1. Capability maturity varies across the platform and moves between releases; the current status of any individual capability should be confirmed against product documentation rather than inferred from this table.

Requirement (§10.1)	How it is implemented
Tenant and customer administration	Provider-facing and customer-facing administration, with delegated customer admin and tenant portals.
Self-service consumption	Self-service workspaces and workflows for developers and data scientists, spanning infrastructure, Kubernetes, storage, networking, identity, and third-party AI services.
Service definition and cataloguing	Compute and application SKUs combining GPU and CPU resources, Kubernetes environments, AI applications, policies, and templates; application catalogs covering internally developed applications, NVIDIA services, third-party tools, and customer-specific templates.
Policy enforcement	Network policies, cluster policies, and RBAC applied across tenants.
Quota and entitlement control	Resource quotas by customer and project.
Environment segmentation	Multi-tenant environment segmentation across customers and teams.
Usage metering and attribution	Usage metrics and usage reporting by customer and team.
Chargeback and showback	Chargeback and usage views surfaced to the consuming organization.
Activity audit	Audit logging across operator and tenant activity.
Kubernetes fleet lifecycle	Kubernetes lifecycle management across multiple customers and clusters.
Underlay automation	Underlay automation coordinating bare metal, network, and storage provisioning.
Deployment flexibility	SaaS and self-hosted models, including air-gapped and jurisdiction-constrained operation without public-cloud dependencies.
White-labelling and embedding	White-label options and APIs allowing capability to be embedded in an existing commercial front end.

11.2 Sovereign and Regulated Deployment

For sovereign and regulated environments, control planes can be kept within a specific jurisdiction, operated in disconnected environments, and run under customer-defined security requirements. Self-hosted deployment gives the operator control over where the platform runs, who administers it, and how customer environments are governed — which is the point at which §9.6's jurisdiction-specific frameworks become implementable rather than aspirational.

PART 5

Bringing It Together

12 NCP Readiness Checklist

The capability set was consolidated as a gap assessment. It is tiered, because the rows are not equally urgent: some are the difference between operating and not operating, some are the difference between competitive and not, and some are what separates a good platform from a demonstrably excellent one.

12.1 Tier 1 — Required to Operate

Without these an operator cannot deliver isolated, known-good capacity to a paying tenant. There is no partial credit here; a gap in this tier is a gap in the service.

Capability	Layer	Typical owner
Facility power and cooling envelope, liquid cooling for rack-scale	Physical	Facilities / DC Eng
Standardized building blocks; core-block functions	Physical	DC Eng
Utility cluster and bootstrap path	Physical	Platform Eng
Compute, converged, and out-of-band fabrics provisioned as distinct networks	Networking	Network Eng
Address plan, private addressing, localized DNS	Networking	Network Eng
Tenant network isolation via overlay or fabric partitioning	Networking	Network Eng
Out-of-band management isolation behind a bastion	Networking / Security	Network Eng
Durable shared storage tier as system of record	Storage	Storage Eng
Block, file, and object access with per-tenant isolation	Storage	Storage Eng
Bare-metal provisioning at scale; firmware to known-good state	IaaS	Platform Eng
Instance lifecycle and stable hardware identifiers	IaaS	Platform Eng
Node sanitization and cryptographic erase between tenants	IaaS / Security	Platform Eng
Secure and measured boot with hardware root of trust and attestation	Security	Security Eng
Identity: OIDC authentication, least-privilege RBAC	Security	Security Eng
Encryption at rest and mutual TLS in transit	Security	Security Eng
No default public reachability; private API paths	Security	Network / Security
Managed Kubernetes with dedicated per-tenant control planes	CaaS	Platform Eng
Control-plane HA with datastore separation and secrets encryption	CaaS / Security	Platform Eng
Redfish over TLS for platform management; IPMI disabled	IaaS / Security	Platform Eng

Capability	Layer	Typical owner
Node health check before vending; basic break-fix loop	Operations	SRE / Ops
24x7 support with named contact and severity procedures	Operations	Support / Ops

12.2 Tier 2 — Required to Compete

These are what a sophisticated capacity consumer will ask for, and what published service-delivery expectations assume. An operator can technically run without them; it will not win an offtake agreement, and it will absorb the shortfall as headcount.

Capability	Layer	Typical owner
Programmatic provisioning across the full node lifecycle	IaaS	Platform Eng
Declarative interfaces for multi-step workflows	IaaS	Platform Eng
Virtual network lifecycle, security groups, movable addresses	Networking / SDN	Network Eng
Backend fabric visibility API; topology discovery	Networking	Network Eng
Full-mesh storage-plane reachability	Networking / Storage	Network Eng
Multiple clusters per tenancy; sized control planes; scale conformance	CaaS	Platform Eng
Storage faults reported through the break-fix interface	Storage / Ops	Storage Eng
Private interconnect to the consuming organization; controlled egress pool	Networking	Network Eng
MACsec on links crossing uncontrolled infrastructure	Networking / Security	Network Eng
Scale-up domain management and domain-aware allocation	Networking / IaaS	Platform Eng
Storage provisioning API, quotas, live expansion, HA and multipathing	Storage	Storage Eng
Bulk data movement path independent of GPU availability	Storage	Storage Eng
Storage QoS per tenant; high-performance GPU-to-storage data path	Storage	Storage Eng
Dynamic resource allocation and rack-scale scheduling support	CaaS	Platform Eng
CSI/CNI integration; topology-aware gang scheduling	CaaS	Platform Eng
Cluster operations: version policy, non-disruptive upgrades, backup and DR	CaaS	SRE / Ops
Unified telemetry over a vendor-neutral protocol, all fabrics and log sources	Operations	SRE / Ops
Resource governance, capacity discovery, and unified health APIs	Operations	Platform Eng
Break-fix API: remediation and maintenance lifecycle actions	Operations	SRE / Ops
Capacity reservations and atomic topology-block allocation	Operations	Platform Eng

Capability	Layer	Typical owner
Measurable, independently reported service-level commitments	Operations	Ops / Commercial
Scheduled-maintenance controls exposed to the consumer	Operations	SRE / Ops
Recognized controls attestation such as SOC 2	Security	Security / Compliance
Retained, queryable audit trail; directory integration; MFA on admin surfaces	Security	Security Eng
Tenant and customer administration; quotas; policy enforcement; audit	Orchestration	Platform / Product

12.3 Tier 3 — Differentiating

These do not make the service work; they make it demonstrably better than an equivalently equipped competitor, and they are where designation and margin are won.

Capability	Layer	Typical owner
Reproducible benchmark performance against published recipes	Operations	Perf Eng
Virtualized-path performance discipline: PCIe topology, NUMA, single-tunnel overlay	IaaS	Platform Eng
Automated remediation with minimal human intervention	Operations	SRE / Ops
Failure-domain observability and correlated-event recognition	Operations	SRE / Ops
Optional shared acceleration tier for context-reuse inference	Storage	Storage Eng
Data lake tier for onboarding and pre-processing	Storage	Storage / Data
Self-service consumption with service catalogues and SKU definition	Orchestration	Product / Platform
Usage metering, chargeback, and white-labelled customer portals	Orchestration	Product / Platform
Air-gapped and jurisdiction-constrained deployment	Orchestration	Platform / Security
Tenant-facing AI platform services beyond raw capacity	AI PaaS	Product / AI Eng
Geographic and sanctions policy filtering	Networking / Compliance	Security / Legal

13 Validating Readiness

A capability checklist is a claim. The question that follows it is how the claim is tested, and by whom.

13.1 Self-Assessment Against the Published Requirements

The first mechanism is a structured self-assessment. NVIDIA publishes its requirements for AI clouds openly, including a validation suite made available as part of an AI cloud-readiness initiative — a set of tests that can be run against an environment to evaluate it against the published requirements and specifications.

Reference tooling is published alongside the requirements for specific capabilities, network topology discovery being the clearest example. Where such tooling exists, aligning to it is cheaper than building an equivalent and considerably cheaper than being found non-conformant later.

13.2 What Counts as Evidence

For each capability in §12, following are some examples:

Evidence type	Applies to	What it looks like
Automated test result	Anything the published validation suite covers	A passing run, dated, against the current environment.
Interface demonstration	API and lifecycle capabilities	A live call returning the expected schema, not a screenshot of a console.
Operational record	Break-fix, maintenance, support, SLA	Historical events with timestamps, actions, and outcomes over a meaningful period.
Third-party attestation	Security and compliance controls	A current report from an independent assessor, scoped to the relevant services and facilities.
Benchmark result	Performance and resiliency	A reproducible run against published recipes, repeated on a different day under load.
Architecture documentation	Design properties not mechanically testable	A described and reviewed design, with the reviewer named.

14 Glossary

Term	Definition
CaaS	Container-as-a-Service: managed Kubernetes delivered to tenants.
CNI / CSI	Container Network Interface / Container Storage Interface: Kubernetes integration standards.
Concurrent multitenancy	More than one tenant active on a physical host at the same time.
Day 2	Operation after the data center is online, validated, and ready for normal tenant use.
Dedicated control plane	A Kubernetes control plane that serves the workloads of exactly one tenant.
DPU	Data Processing Unit: offloads networking, storage, and security from the host.
DRA	Dynamic Resource Allocation: flexible allocation of hardware resources such as GPUs in Kubernetes.
Exemplar Cloud	An NVIDIA designation established by reproducible benchmark validation of performance and resiliency.
Failure domain	A set of resources that can fail together, typically because they share power, cooling, or a switching tier.
Hot-air containment	A group of racks exhausting into a shared hot-air area, generally aligned to a pair of row segments.
IaaS	Infrastructure-as-a-Service: bare metal and VM provisioning.
IAM	Identity and Access Management.
Infrastructure operator domain	Break-fix operations performed while a node is under operator control.
KaaS	Kubernetes-as-a-Service: automated provisioning of clusters for end customers.
MACsec	Link-layer encryption for Ethernet segments.
MFA	Multi-Factor Authentication.
NCP	NVIDIA Cloud Partner: a partner that builds data centers on the NVIDIA platform and offers AI cloud services.
OIDC	OpenID Connect: a standards-based identity and authentication protocol.
OTLP	OpenTelemetry Protocol: a vendor-neutral wire format for metrics, logs, and traces.
PaaS	Platform-as-a-Service: tenant-facing AI training and inference.
Rail-aligned	A cluster topology in which node network attachment is aligned to fabric rails, enabling topology-aware scheduling.
RDMA	Remote Direct Memory Access: low-latency, CPU-bypassing data transfer.
Scalable unit (SU) group	A modular, repeatable building block with its own east-west fabric and, generally, its own cooling and support infrastructure.
Scale-up domain	The set of GPUs connected by a single multi-node scale-up fabric, within which memory can be addressed directly.
SDN / SDS	Software-Defined Networking / Storage.
Shared host	A worker host shared between two or more tenants.
SR-IOV	Single Root I/O Virtualization: hardware-accelerated NIC sharing.
Tenant domain	Break-fix operations performed while a node is in a tenant’s possession.
Topology block	A group of resources allocated as a single unit so that they share performance and security boundaries.
TPM	Trusted Platform Module: stores boot measurements for attestation.
Worker host	A bare metal server underlying control-plane and worker capacity.

Term	Definition
Worker node	A virtualized compute resource on a worker host, used by tenants to run workloads.

Appendix A Known Failure Modes

A capability list describes what to have. It does not describe what will go wrong. The failure modes below recur across multi-tenant GPU platforms, are largely absent from reference documentation because they are operational rather than architectural, and are cheaper to design against than to diagnose at three in the morning.

This appendix reflects field experience rather than a published source, and is offered as a starting list for an operator's own risk register.

Failure mode	How it presents	Where to design against it
Virtual function exhaustion	Instance creation begins failing at a utilization threshold with an error that points at the hypervisor rather than the NIC.	§5.2 — budget virtual functions per host against maximum instance density, and surface remaining capacity as a monitored resource.
Subnet manager failover behaviour	A fabric manager failover completes successfully but in-flight collective operations fail or hang, affecting tenants who were mid-job.	§3.4 — test failover under load, not at idle, and treat manager redundancy as a data-path concern.
Checkpoint storm	Synchronized checkpointing across many tenants saturates the storage fabric periodically, presenting as intermittent unexplained training slowdown.	§4.2 and §4.5 — size the durable tier for concurrent burst rather than average throughput, and enforce per-tenant QoS.
Partition and sharing interaction	GPU partitioning and dynamic resource allocation disagree about available resources; pods schedule onto capacity that is not really there.	§5.5 and §6.2 — pin the interaction between partitioning mode and the allocation mechanism, and validate at the version pair actually deployed.
Sustained-load thermal throttling	Benchmarks pass; a week-long training run degrades gradually as thermals rise across a containment area.	§2.5 and §8.2 — monitor thermals as a performance signal, and benchmark at duration rather than in short runs.
Firmware drift	Nodes that should be identical perform differently; the fleet slowly becomes heterogeneous through individual repairs.	§5.3 and §9.1 — enforce firmware to a known-good state on every return to circulation, and report versions through diagnostics.
NUMA and topology misplacement	Virtualized workloads reach perhaps two-thirds of bare-metal performance with no obvious fault.	§5.5 — preserve PCIe topology into the guest, pin vCPUs, align memory, and verify rather than assume.
Double encapsulation	Throughput on the tenant overlay is well below line rate and MTU-related failures appear under load.	§5.5 — ensure a single tunnel for the tenant overlay and validate effective MTU end to end.
Address plan collision	Tenant-supplied address ranges overlap with platform or another tenant's ranges; discovered when peering or storage access is requested.	§3.2 and Figure 4 — decide the address plan before the first tenant network, and validate supplied ranges at request time.
Sanitization gap	Infrastructure passes between tenants with residual state — an unwiped ephemeral drive, a firmware setting, a cached credential.	§9.2 — attest that a node is clean before reallocation rather than trusting that the wipe ran.
Telemetry blind spot	An incident cannot be explained because the signal that would have explained it was never collected from the fabric or switch logs.	§8.2 — cover the log sources, not only the metrics; test by attempting to reconstruct a past incident.
Correlated failure read as many	A single power or cooling event generates dozens of independent-looking node failures and the response fragments across teams.	§8.5 — expose failure-domain membership so correlated events collapse into one.