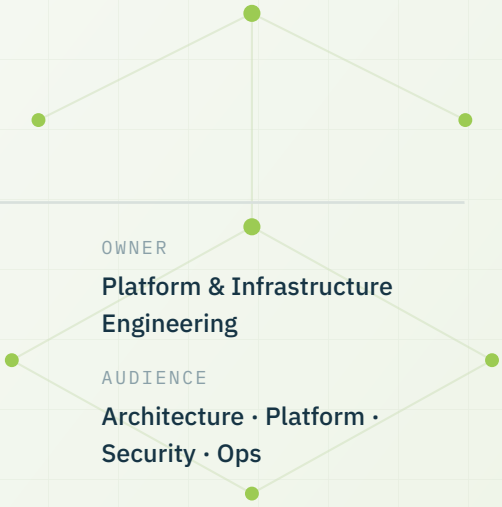
TECHNICAL SPECIFICATION

On-Premises NVIDIA AI Factory

A multi-tenant platform for R&D, large-scale AI training, and governed inference services for custom tenant workflows

An air-gap-capable reference design built on NVIDIA AI Enterprise. It unifies accelerated compute, GPU-aware orchestration, model customization, and optimized inference behind a self-service, policy-governed multi-tenant control plane — engineered for sovereign and regulated environments.

NVIDIA AI Enterprise · NIM Microservices · Triton Inference Server · TensorRT-LLM · NeMo · NeMo Guardrails · Run:ai / KAI Scheduler · vLLM · GGUF · Quantized · Air-Gapped



DOCUMENT
AI Factory Platform Specification

VERSION
1.0 — Draft for Review

DATE
26 June 2026

CLASSIFICATION
Confidential — Internal Use

OWNER
Platform & Infrastructure Engineering

AUDIENCE
Architecture · Platform · Security · Ops

Document Control

Purpose of this document

This specification defines the target architecture, components, interfaces, and operating model for an on-premises NVIDIA AI factory: a consolidated platform that serves three workload classes on shared accelerated infrastructure — (1) interactive and batch **R&D**, (2) large-scale distributed **model training and customization**, and (3) production **inference services** exposed to internal and external tenants through governed, self-service workflows. It is written to be buildable: every layer names concrete products, deployment patterns, and reference configurations, and is designed to operate in fully **air-gapped** sites.

Revision history

Version	Date	Author	Summary of change
0.1	2026-05-30	Platform Engineering	Initial outline and component selection.
0.6	2026-06-14	Platform Engineering	Added orchestration, inference, and air-gap lifecycle sections; first sizing tiers.
1.0	2026-06-26	Platform Engineering	Draft for review — full architecture, security, observability, roadmap, and BOM.

Reviewers & approvers

Role	Responsibility	Status
Chief Architect	Architecture sign-off	Pending
Head of Platform Engineering	Build & operability	Pending
CISO / Security Architecture	Security & compliance	Pending
Data Center / Facilities Lead	Power, cooling, space	Pending
FinOps	Capacity & cost model	Pending

CONVENTIONS

The keywords **MUST**, **SHOULD**, and **MAY** are used in the sense of RFC 2119. Bracketed values such as `‹ORG›`, `‹SITE-A›`, or `‹domain›` are site-specific placeholders to be resolved during detailed design. Requirement identifiers (e.g. `FR-07`) are referenced throughout for traceability. Product capabilities reflect the NVIDIA AI Enterprise stack as of Q2 2026 and SHOULD be re-validated against current release notes at implementation time.

Contents

1 **Executive Summary** 5

2 **Goals, Scope & Requirements** 7

2.1 Functional Requirements 7

2.2 Non-Functional Requirements 7

2.3 Scope Boundaries 8

2.4 Personas & Tenancy Model 8
2.5 Assumptions & Constraints 9

3 Architecture Overview 10
3.1 Design Principles 10
3.2 Layered Reference Architecture 10
3.3 Component-to-Capability Map 11
3.4 Environment & Network Domains 12

4 Physical Infrastructure & Hardware 13
4.1 Accelerated Compute Options 13
4.2 Network Fabrics 13
4.3 Storage Architecture 14
4.4 Power, Cooling & Facilities 15
4.5 Reference Pod Sizing Tiers 15

5 Platform Software Foundation (NVAIE) 16
5.1 NVAIE Entitlement & Licensing 16
5.2 Host OS, Drivers & Container Runtime 16
5.3 Kubernetes, GPU & Network Operators 17
5.4 Private Registry & Artifact Management 17
5.5 Air-Gapped Bootstrap & GitOps 18

6 GPU Orchestration & Multi-Tenancy 19
6.1 Run:ai vs. KAI Scheduler Selection 19
6.2 Scheduling: Queues, Fair-Share & Gang 19
6.3 GPU Partitioning: MIG, Slicing, Fractional 20
6.4 Topology-Aware Scheduling & DRA 21
6.5 Tenant Isolation Model 21
6.6 Quota, Metering & Showback 21

7 AI Training & Model Customization (NeMo) 22
7.1 The NeMo Customization Pipeline 22
7.2 Distributed Training & Parallelism 22
7.3 Fine-Tuning, PEFT & Alignment 23
7.4 Experiment, Data & Lineage 23
7.5 Scheduling Training on Run:ai 23

8 Inference Serving Plane 25
8.1 NIM Microservices Architecture 25
8.2 Triton Inference Server 25
8.3 TensorRT-LLM 26
8.4 vLLM 26
8.5 NIM Operator on Kubernetes 26
8.6 Disaggregated Serving (Dynamo / Grove) 27
8.7 Model Gateway & Tenant Catalog 27

9 Quantization & Air-Gapped Model Lifecycle 29

9.1 Quantization Strategy Matrix 29
9.2 GPU-Optimized Path (TensorRT-LLM) 29
9.3 Portable / Sovereign Path (GGUF & vLLM) 30
9.4 Air-Gapped Artifact Pipeline 30
9.5 Model Provenance & Registry Promotion 30

10 Safety, Guardrails & Responsible AI 32
10.1 Guardrails Architecture & Rail Types 32
10.2 NemoGuard Safety Models 32
10.3 Per-Tenant Policy Configuration 32
10.4 Placement in the Request Path 33

11 Multi-Tenant Workflow Platform 34
11.1 Tenant Onboarding Lifecycle 34
11.2 Self-Service & Quotas 34
11.3 Custom Tenant Workflow Patterns 34
11.4 Data Isolation & Residency 35
11.5 Tenant API Gateway & Metering 35

12 Security Architecture 36
12.1 Zero-Trust & Network Segmentation 36
12.2 Identity, RBAC & Secrets 36
12.3 Supply-Chain Security & Admission 36
12.4 Confidential Computing 37
12.5 Audit, Logging & Compliance 37

13 Observability & Operations 38
13.1 Metrics & GPU Telemetry 38
13.2 Logging & Tracing 38
13.3 Health, Faults & Mission Control 38
13.4 SLOs & Alerting 39

14 Reliability, Availability & DR 40
14.1 High-Availability Design 40
14.2 Training Resiliency & Checkpointing 40
14.3 Backup & Restore 40
14.4 Disaster Recovery (Air-Gapped) 41

15 Capacity Planning & Sizing 42
15.1 Workload Characterization 42
15.2 Sizing Methodology 42
15.3 Reference Tier Bill of Materials 43

16 Deployment Roadmap 44

17 Risk Register 45

18 Appendices 46

CHAPTER 01

Executive Summary

This document specifies an on-premises **AI factory** — a single, shared, GPU-accelerated platform that an organization owns and operates to run research, train and customize models, and deliver inference as an internal product to many tenants. It is built end-to-end on **NVIDIA AI Enterprise (NVAIE)** and is designed to run with no dependency on the public internet, so it can be deployed in air-gapped, sovereign, or regulated facilities.

The core problem this platform solves is consolidation without contention. GPUs are the most expensive and most contended resource in any modern enterprise, and the default pattern — static, per-team clusters — strands capacity: industry case studies routinely show accelerated clusters running at 15–35% utilization because whole GPUs are pinned to workloads that use a fraction of them. The AI factory replaces that pattern with a pooled, policy-governed fabric in which research notebooks, multi-node training runs, and latency-sensitive inference all draw from the same capacity under fair-share scheduling, GPU partitioning, and hard tenant isolation.

1.1 What the platform delivers

- **One pool, three workload classes.** Interactive R&D, distributed training/customization, and production inference share the same accelerated infrastructure, scheduled by **NVIDIA Run:ai** (built on the open-source **KAI Scheduler**) with gang scheduling, fractional GPUs, MIG, and topology awareness for Blackwell-class hardware.
- **Inference as a product.** Models are served as **NVIDIA NIM** microservices — each a container bundling the model, an optimal runtime (**TensorRT-LLM**, **vLLM**, or SGLang) selected for the target GPU, and an OpenAI-compatible API — fronted by a model gateway that gives tenants self-service access to a curated catalog. **Triton Inference Server** underpins multi-framework and ensemble serving.
- **Customization in-house.** The **NeMo** framework provides the full path from data curation through pre-training, supervised fine-tuning, parameter-efficient tuning (LoRA/QLoRA), and alignment (DPO/RLHF), with distributed training on Megatron-Core.
- **Governed by default. NeMo Guardrails** and the NemoGuard safety models enforce per-tenant input, dialog, retrieval, and output rails in the request path; supply-chain controls, signing, RBAC, and admission policy gate everything that runs.
- **Sovereign and quantized.** A dual model-delivery model supports a GPU-optimized path (TensorRT-LLM FP8/FP4 engines) and a portable path (**GGUF** single-file quantized weights served via **vLLM**/llama.cpp) for fully disconnected, hardware-flexible deployment.

1.2 Architecture at a glance

The platform is layered: accelerated hardware and high-bandwidth fabric at the base; a hardened host, container runtime, and Kubernetes with the NVIDIA GPU and Network Operators above it; NVAIE platform services and a

private registry; Run:ai for orchestration and multi-tenancy; the NeMo training plane and the NIM/Triton inference plane as the AI services tier; and a self-service tenant control plane on top. Security and observability are cross-cutting rails that span every layer. Figure 3.1 in §3.2 details this.

1.3 Key decisions

Decision	Choice	Rationale
Software foundation	NVIDIA AI Enterprise (NVAIE)	Single supported, security-maintained stack spanning operators, NIM, Triton, NeMo; licensed for air-gapped use.
Orchestration	Run:ai (enterprise) over KAI Scheduler	Enforced GPU memory isolation, MIG automation, memory swap, multi-cluster control, RBAC/SSO on top of the open KAI engine.
Inference packaging	NIM microservices on Triton + TensorRT-LLM	One-command, pre-optimized, OpenAI-compatible serving with enterprise support and air-gap caching.
Flexible / sovereign serving	vLLM + GGUF as the secondary path	Serve models outside the NIM catalog and run on heterogeneous or disconnected hardware.
Safety	NeMo Guardrails + NemoGuard NIMs	Programmable, per-tenant rails enforced in the request path, independent of the underlying model.
Operating posture	Air-gap-first, GitOps-driven	Reproducible, auditable, and portable across connected and disconnected sites.

1.4 Delivery approach

Implementation is phased over five stages (Chapter 16): a hardware and platform Foundation; the Orchestration and multi-tenancy control plane; the Inference service plane; the Training and customization plane; and Multi-Tenant GA with full self-service, metering, and guardrails. Each phase has explicit exit criteria so the platform delivers usable capability early and hardens progressively. Three reference sizing tiers — **Pilot**, **Production**, and **Scale** — are defined in §4.5 and §15.3 to anchor procurement.

> **OUTCOME**
>
> A single accelerated platform that raises GPU utilization from the typical 15–35% toward 70%+ through pooling and partitioning, shortens the path from research to served model to days, and exposes governed AI services to tenants — all operable inside an air-gapped boundary with a clear, supported upgrade path.

CHAPTER 02

Goals, Scope & Requirements

This chapter states what the platform must do and the constraints it operates under. Requirements are identified for traceability and are referenced from the design chapters that satisfy them.

2.1 Functional Requirements

ID	Requirement	Detail & acceptance
FR-01	Shared accelerated pool	Serve R&D, training, and inference from one GPU pool with policy-based allocation; no static per-team partitioning required.
FR-02	Interactive R&D	Provide on-demand notebooks/IDEs and dev workspaces with fractional or whole-GPU attach, backed by shared datasets.
FR-03	Distributed training	Run multi-node, multi-GPU training/fine-tuning with gang scheduling, NeMo/Megatron parallelism, and resumable checkpointing.
FR-04	Model customization	Support SFT, LoRA/QLoRA, and alignment (DPO/RLHF), with evaluation gates before promotion.
FR-05	Optimized inference	Serve LLM, embedding, reranking, vision, and speech models as NIM/ Triton endpoints with OpenAI-compatible APIs.
FR-06	Model catalog	Curated, versioned catalog of approved models; tenants discover and consume via self-service.
FR-07	Custom tenant workflows	Tenants compose RAG, agentic, and batch pipelines against platform model and retrieval services.
FR-08	Guardrails	Per-tenant programmable input/dialog/retrieval/output rails enforced in the request path.
FR-09	Quantized & portable models	Support FP8/FP4 GPU engines and GGUF/AWQ/GPTQ portable weights for disconnected/heterogeneous serving.
FR-10	Self-service & quota	Tenants request projects, quotas, API keys, and endpoints through a portal/API with approval workflow.
FR-11	Metering & showback	Per-tenant accounting of GPU-hours and tokens for chargeback/ showback.
FR-12	Air-gapped operation	All install, model, and update artifacts importable and operable with no internet egress.

2.2 Non-Functional Requirements

ID	Attribute	Target / constraint
NFR-01	Inference latency	Interactive LLM endpoints meet a per-tenant SLO, e.g. TTFT ≤ 300 ms and inter-token latency ≤ 50 ms at defined concurrency for the reference model class.

ID	Attribute	Target / constraint
NFR-02	Inference availability	Production endpoints ≥ 99.5% monthly; rolling upgrades with no dropped in-flight requests.
NFR-03	GPU utilization	Cluster-wide allocated-GPU utilization sustained ≥ 65–75% under steady load.
NFR-04	Scheduling fairness	Guaranteed-quota workloads start within a bounded queue time; over-quota use is reclaimable.
NFR-05	Scalability	Scale from a single Pilot pod to multi-rack Scale tier without architectural change.
NFR-06	Tenant isolation	Network, namespace, data, and (where required) hardware isolation between tenants; no cross-tenant data path by default.
NFR-07	Security & compliance	Zero-trust segmentation, signed/scanned images, full audit trail; mappable to the org's control framework.
NFR-08	Recoverability	Training resumable from checkpoint; platform state (etcd, registry, models, configs) backed up with tested restore.
NFR-09	Operability	Single observability plane for GPU, node, workload, and service health; declarative, GitOps-managed configuration.
NFR-10	Supportability	Components covered by NVAIE enterprise support and a defined LTS/upgrade cadence.

2.3 Scope Boundaries

In scope	Out of scope (this revision)
Accelerated compute, fabric, and storage reference design	Detailed facility civil/electrical engineering drawings
NVAIE platform, orchestration, training, and inference planes	Application-level tenant product code
Multi-tenant control plane, guardrails, metering	Procurement contracts and commercial licensing negotiation
Security architecture and air-gapped lifecycle	Foundation-model pre-training data sourcing/licensing
Observability, HA/DR, sizing, and roadmap	Org-specific HR/training and change-management programs

2.4 Personas & Tenancy Model

The platform serves three actor groups, each with a distinct contract:

- **Platform operators** — own the cluster, fabric, NVAIE stack, scheduler policy, registry, and guardrail defaults. They define departments, quotas, and the model catalog.
- **R&D / ML engineers** — consume interactive and batch compute to explore, train, and customize models; promote artifacts through the model registry.
- **Tenant application teams** — consume governed inference and retrieval services to build custom workflows (chat assistants, RAG, agents, batch enrichment) within their quota and policy.

A **tenant** is the unit of isolation and accounting. It maps to one or more Kubernetes namespaces, a Run:ai project/queue with a GPU quota, a network segment, dedicated object-storage buckets, and a set of API

credentials. Two isolation grades are offered — soft (namespace + network policy + scheduler quota, default) and hard (adds dedicated node pools and/or MIG-isolated GPUs and stricter network controls) — selected per tenant by data sensitivity. See §6.5 and Chapter 11.

2.5 Assumptions & Constraints

Area	Assumption / constraint
Connectivity	The production environment is air-gapped or strongly egress-controlled; a separate, internet-connected staging enclave exists for artifact acquisition and curation.
Licensing	Valid NVAIE entitlements are available, including an air-gapped license-server option; NIM/NGC artifacts are accessible from the staging enclave for import.
Facilities	Sufficient power and cooling exist for the chosen tier, including liquid cooling for any GB200 NVL72 rack-scale deployment.
Skills	The operating team is fluent in Kubernetes, GitOps, and Linux GPU operations; NVIDIA professional services may assist with bring-up.
Identity	An enterprise IdP (OIDC/SAML) is reachable from the management plane for SSO, or a local IdP is provisioned in disconnected sites.
Standardization	Workloads are containerized; bespoke bare-metal exceptions are handled case-by-case and discouraged.

CHAPTER 03

Architecture Overview

The AI factory is a layered platform with two cross-cutting concerns — security and observability — that pass through every layer. Each layer presents a stable contract to the one above it, so capabilities can be upgraded independently and the same architecture scales from a single pod to many racks.

3.1 Design Principles

- **Air-gap-first.** Every artifact — container images, model weights, Helm charts, drivers, licenses — is importable and operable with no internet egress. Connected convenience features are optional, never load-bearing.
- **Pool, then partition.** GPUs are a single fabric-wide pool. Isolation is imposed by scheduler policy and partitioning (MIG, fractional, node pools), not by carving the fleet into static silos.
- **Declarative & reproducible.** Cluster, platform, and tenant state are expressed as version-controlled manifests reconciled by GitOps. The running system is always a function of Git plus the registry.
- **Defense in depth.** Identity, network, admission, runtime, and data controls are layered so no single failure exposes a tenant.
- **Standard interfaces.** OpenAI-compatible inference APIs, Kubernetes CRDs, and S3-compatible storage keep tenants and tooling portable and avoid lock-in within the platform.
- **Separation of planes.** Control, training, and serving planes are logically and operationally distinct, with independent scaling and failure domains.

3.2 Layered Reference Architecture

Figure 3.1 shows the full stack. Reading bottom-up: accelerated hardware and fabric; a hardened host and Kubernetes with the NVIDIA operators; NVAIE platform services and the private registry; Run:ai orchestration and multi-tenancy; the NeMo training plane and the NIM/Triton inference plane; and the tenant self-service control plane. Security and observability span all layers.

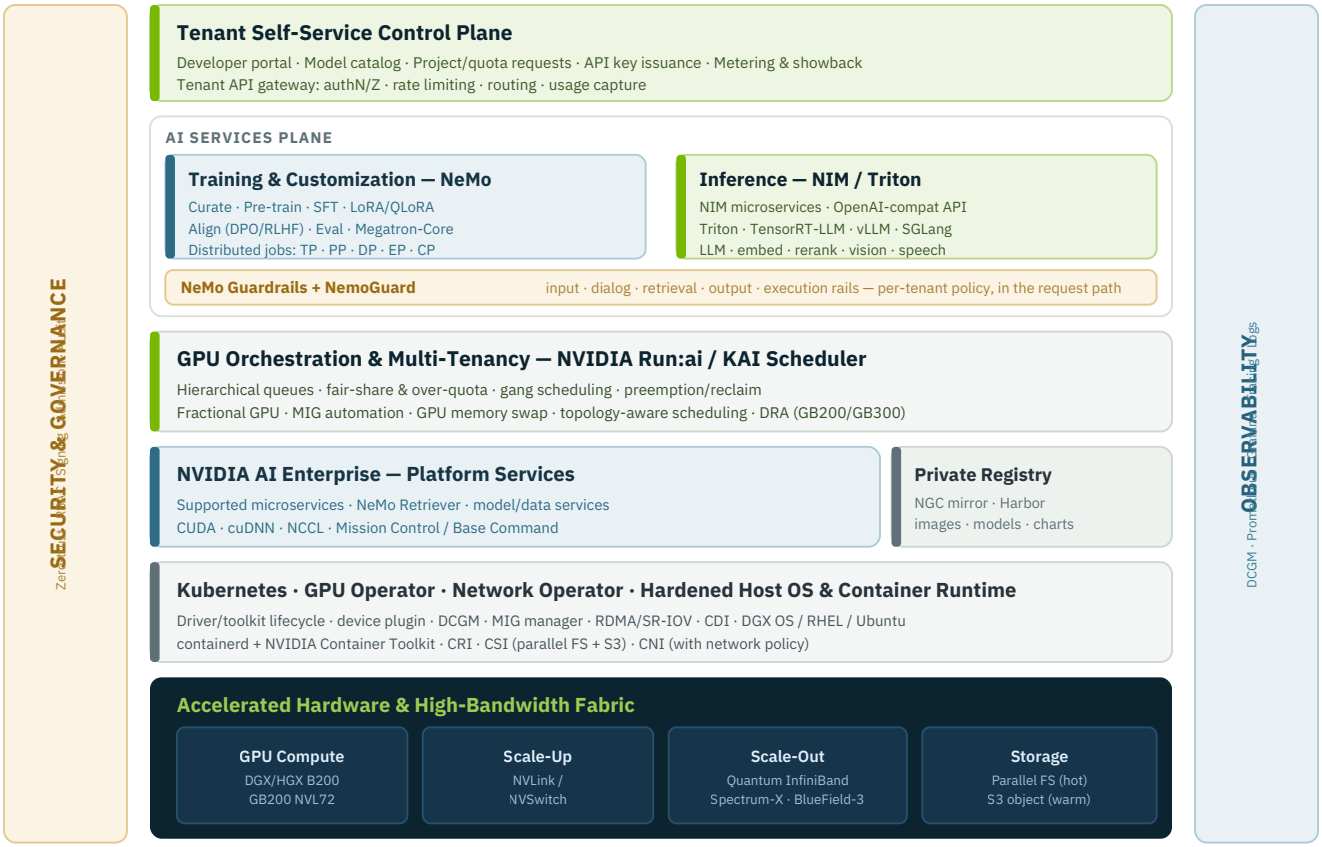


Figure 3.1 — Layered reference architecture of the on-premises NVIDIA AI factory. Security/governance and observability are cross-cutting rails spanning all layers.

3.3 Component-to-Capability Map

The named technologies in the brief map onto the layers as follows. This table is the index used by the detailed chapters.

Component	Layer	Primary role	Chapter
NVIDIA AI Enterprise	Platform foundation	Supported, security-maintained software stack and entitlement	5
Run:ai / KAI Scheduler	Orchestration	GPU-aware scheduling, partitioning, multi-tenant quota	6
NeMo	AI services — training	Data curation, training, fine-tuning, alignment, eval	7
NIM microservices	AI services — inference	Packaged, optimized, OpenAI-compatible model endpoints	8
Triton Inference Server	AI services — inference	Multi-framework serving, ensembles, dynamic batching	8

Component	Layer	Primary role	Chapter
TensorRT-LLM	AI services — inference	Compiled, quantized, high-throughput LLM engines	8
vLLM	AI services — inference	Flexible serving; off-catalog and quantized weights	8
GGUF (quantized)	Model lifecycle	Portable single-file weights for sovereign/heterogeneous serving	9
NeMo Guardrails	AI services — safety	Programmable per-tenant rails in the request path	10

3.4 Environment & Network Domains

The platform spans up to three logical environments. The **staging enclave** is the only environment with controlled outbound internet access; it is where artifacts are acquired, scanned, and signed before a one-way transfer into the production boundary. The **production environment** is air-gapped (or strongly egress-controlled) and hosts the live platform. An optional **management environment** hosts Git, CI, secrets, and the observability backends that operate the production cluster from within the trusted boundary.

Environment	Connectivity	Contents
Staging enclave	Controlled egress	NGC/Hugging Face mirroring, model conversion/quantization, vulnerability scanning, signing, SBOM generation, transfer staging.
Production (air-gapped)	No egress	GPU cluster, NVAIE, Run:ai, NIM/Triton, NeMo jobs, private registry, tenant gateway, guardrails.
Management	Internal only	GitOps repo + reconciler, CI runners, Vault/secrets, IdP, Prometheus/ Grafana/Loki, backup targets.

The crossing between staging and production is a controlled, auditable, one-way artifact transfer (see §9.4 and §12.1). No production workload ever reaches out to the staging enclave or the internet at request time.

CHAPTER 04

Physical Infrastructure & Hardware

The hardware layer provides accelerated compute, a high-bandwidth scale-up/scale-out fabric, and tiered storage. The reference design is expressed as repeatable pods so the platform grows by adding pods rather than by redesign. Three named tiers — Pilot, Production, Scale — anchor procurement.

4.1 Accelerated Compute Options

Three Blackwell-generation building blocks cover the spectrum from a cost-effective x86 node to trillion-parameter rack-scale training. The platform standardizes on one primary block per site; mixed fleets are supported and handled by topology-aware scheduling and DRA (§6.4).

Building block	Composition	Best for	Notes
HGX B200 node (x86)	8× Blackwell B200, 5th-gen NVLink within the node; ConnectX-7/8 + BlueField-3	Inference at scale, mixed R&D, single-node and small multi-node training	Air-cooled or rear-door heat-exchanger viable; integrates into standard OEM x86 servers. Most flexible entry point.
DGX B200 system	8× B200, NVLink, full NVIDIA software stack, ~14.3 kW	Turnkey develop-to-deploy; BasePOD/ SuperPOD foundation	Validated appliance; ~3× training and ~15× inference vs. DGX H100 (NVIDIA figures). Predictable bring-up.
GB200 NVL72 rack	36 Grace CPU + 72 B200 in one 72-GPU NVLink domain; 13.5 TB HBM3e; liquid-cooled	Frontier training and trillion-parameter, low-latency inference	Acts as a single large GPU; requires liquid cooling and high-density power as a hardware prerequisite. (GB300/Blackwell Ultra is the successor.)

COMPUTE SELECTION GUIDANCE

Default to **HGX/DGX B200** nodes for the Pilot and Production tiers: they cover R&D, inference, and most fine-tuning, and fit conventional or lightly retrofitted facilities. Introduce **GB200 NVL72** only at the Scale tier when frontier training or very large MoE inference justifies the facility investment in liquid cooling and 60–130 kW-class racks.

4.2 Network Fabrics

Four physically or logically separated networks serve distinct roles. The **compute fabric** is the performance-critical, rail-optimized, lossless network for distributed training collectives (NCCL all-reduce, etc.); it MUST NOT be shared with storage or management traffic.

Network	Technology	Speed (per port)	Purpose
Compute (east-west)	NVIDIA Quantum-2 / Quantum-X800 InfiniBand (or Spectrum-X Ethernet)	400–800 Gb/s	GPU-to-GPU collectives across nodes; rail-optimized, lossless, with SHARP in-network reduction.

Network	Technology	Speed (per port)	Purpose
Storage	InfiniBand or RoCE (Spectrum-X) on dedicated ports	200–400 Gb/s	High-throughput access to the parallel filesystem and object store.
In-band management	Ethernet via BlueField-3 DPU	2×200 Gb/s	Kubernetes, provisioning, service traffic, control plane.
Out-of-band (OOB)	1 GbE BMC / NVOS	1 Gb/s	Lights-out hardware management, firmware, console, telemetry.

The fabric is a non-blocking (or controlled over-subscription) leaf-spine topology. **BlueField-3 DPUs** offload networking, storage, and security functions from the host CPU and provide an isolation boundary between tenant traffic and the infrastructure control plane.

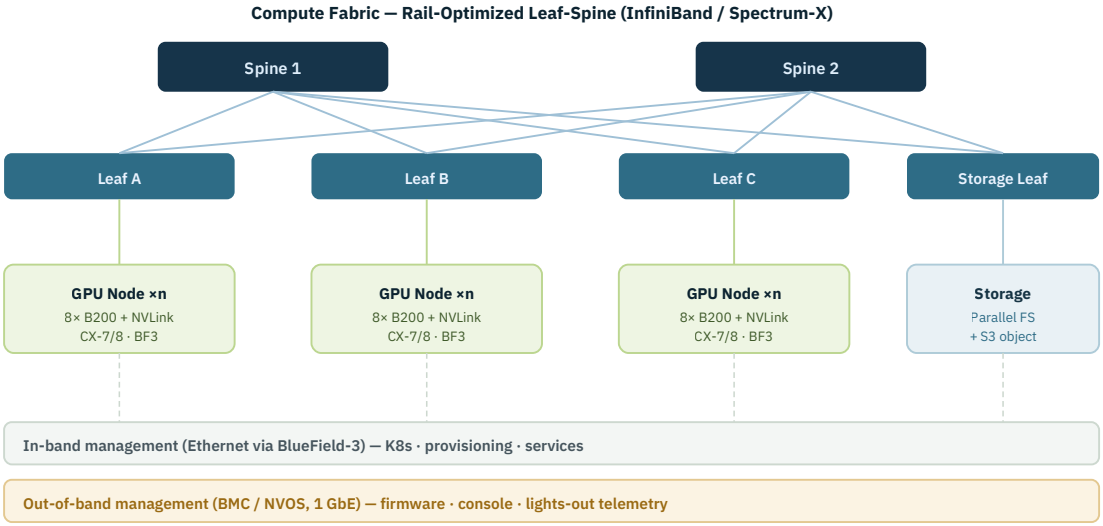


Figure 4.1 — Reference fabric: a dedicated rail-optimized compute network for GPU collectives, separate storage access, and isolated in-band and out-of-band management planes.

4.3 Storage Architecture

Training and inference impose different storage demands, addressed by a two-tier design with a clear contract for each.

- **Hot tier — parallel filesystem.** A high-throughput, POSIX, parallel filesystem (e.g. a certified offering on NVMe with InfiniBand/RoCE access) feeds training data and checkpoints at the multi-GB/s-per-node rates large jobs require. It is the working set for active training and the backing store for model engine caches.
- **Warm tier — S3-compatible object store.** An on-prem object store holds the canonical model registry artifacts, datasets, curated corpora, and tenant data buckets. It is the system of record and the source for promotion between environments.

- **Local NVMe.** Per-node NVMe (e.g. RAID-0 E1.S on DGX) provides scratch, dataset shards, and fast container/model cache. NIM model caches and TensorRT-LLM engine builds are persisted to RWX volumes (§8.5).

Datasets and models are versioned and content-addressed; tenant buckets are isolated with per-tenant credentials and, where required, encryption keys (§11.4).

4.4 Power, Cooling & Facilities

Blackwell density changes facility requirements materially. An air-cooled HGX/DGX B200 server draws on the order of 14 kW; a populated rack can reach 60–100 kW, beyond the 10–30 kW envelope of conventional enterprise racks. A GB200 NVL72 rack is liquid-cooled by design and pulls substantially more per square foot.

Aspect	B200 nodes (HGX/DGX)	GB200 NVL72
Cooling	Air, or rear-door heat exchanger at higher density	Direct liquid cooling — prerequisite, not optional
Rack power	~14 kW/server; plan 30–100 kW/rack by density	High-density; provision facility power and CDU accordingly
Facility impact	Fits many existing data halls, possibly with retrofit	Liquid-cooling plant + structural/electrical upgrades
Redundancy	N+1 PSUs and PDUs; A/B power feeds	Redundant power shelves; CDU redundancy

> **FACILITIES NOTE**
>
> Retrofitting liquid cooling into an air-cooled hall is a significant capital item. If the Scale tier is anticipated, validate power, weight, and cooling headroom during the Pilot so the facility path is known before frontier hardware is ordered. Engage facilities and NVIDIA infrastructure specialists for GB200 site planning.

4.5 Reference Pod Sizing Tiers

A pod is the unit of growth: GPU nodes, fabric, storage, and the platform stack sized to operate together. Tiers are indicative starting points; §15.3 expands them into a bill of materials.

Tier	Compute	Fabric	Intended use
Pilot	2–4× HGX/DGX B200 nodes (16–32 B200)	Single IB leaf pair; 1 storage leaf	Prove the platform: R&D, first inference services, one or two tenants.
Production	8–16 B200 nodes (64–128 B200)	Non-blocking leaf-spine; redundant storage	Steady-state multi-tenant inference plus routine training/fine-tuning.
Scale	Multiple B200 racks and/or 1+ GB200 NVL72	Multi-spine; SuperPOD-class design	Frontier training, large MoE inference, many tenants, multi-site DR.

CHAPTER 05

Platform Software Foundation (NVAIE)

The factory runs on NVIDIA AI Enterprise (NVAIE) — the supported, secure software layer that turns bare GPU servers into a governed platform. This chapter defines the host, cluster, registry, and bootstrap foundations on which every later capability depends.

5.1 NVAIE Entitlement & Licensing

NVAIE is the licensed distribution that bundles GPU Operator, Network Operator, NIM, NeMo, Triton, the inference microservices, and enterprise support with security maintenance and CVE patching. Crucially for this design, the NVAIE entitlement is what makes **air-gapped deployment** a supported configuration rather than a workaround.

Licensing is enforced through NVIDIA License System (NLS). In a disconnected facility the platform MUST run a **local DLS (Delegated License Service)** instance, served from a VM or container inside the management domain, with license entitlements pre-staged. GPU clients lease licenses from the local DLS; no call to NVIDIA's cloud licensing is made at runtime.

> **AIR-GAP REQUIREMENT**
> Provision the DLS appliance and load entitlements during staging, before the production network is sealed. Validate license lease and renewal entirely within the air-gapped enclave. Track entitlement expiry as an operational SLO — an expired lease degrades supported features and must never surprise the platform team.

Entitlement count is sized to deployed GPUs plus headroom for the Scale tier. The post-GTC 2026 NIM access changes (a free tier via the NVIDIA Developer Program for up to 16 GPUs, with broader self-hosting rights) lower the barrier for the Pilot, but production multi-tenant operation SHOULD carry full NVAIE entitlement for support and security maintenance.

5.2 Host OS, Drivers & Container Runtime

Each GPU node runs a supported host OS: **DGX OS** on DGX systems, or certified **Ubuntu LTS / RHEL** on HGX and partner servers. The host carries the minimum to be a Kubernetes node and a GPU host; all application logic ships in containers.

Layer	Component & role
Base OS	DGX OS (Ubuntu-derived) or certified Ubuntu 22.04+/RHEL 9; hardened CIS baseline, immutable where possible.
GPU driver	Data Center Driver, lifecycle-managed by the GPU Operator (driver containers) rather than hand-installed.
Container runtime	`containerd` with the **NVIDIA Container Toolkit** and CDI (Container Device Interface) for GPU injection.

Layer	Component & role
Fabric stack	OFED / DOCA for InfiniBand and RoCE; GPUDirect RDMA and GPUDirect Storage enabled for collective and I/O paths.
Telemetry agent	DCGM on every node, exporting GPU health and utilization (consumed in §13.1).

Driver, toolkit, and firmware versions are pinned to an NVAIE-validated combination and rolled forward deliberately. Host configuration is delivered by image (golden node image) plus a configuration-management pass, never by manual SSH — reproducibility is a hard requirement for an air-gapped estate.

5.3 Kubernetes, GPU & Network Operators

Kubernetes is the substrate. Two operators make it GPU-aware:

The **NVIDIA GPU Operator** automates the full GPU software lifecycle on each node: driver, container toolkit, device plugin, DCGM exporter, Node Feature Discovery, MIG manager, and the DCGM-based health checks. It reconciles this state declaratively, so a new node joins the cluster and becomes schedulable for GPU work without manual steps.

The **NVIDIA Network Operator** manages the high-performance fabric: the OFED driver, the RDMA shared-device or SR-IOV device plugins, and secondary network attachment via Multus, so pods can attach to the InfiniBand/RoCE fabric for multi-node collectives and to storage networks. It pairs with the GPU Operator to enable GPUDirect RDMA end to end.

> **DECISION — KUBERNETES-NATIVE FROM DAY ONE**
>
> All compute — training and inference — is scheduled on Kubernetes. This yields one control plane, one RBAC model, one GitOps workflow, and one place to enforce admission policy. Slurm-style HPC scheduling is explicitly out of scope; gang scheduling and topology awareness are provided by the GPU scheduler in Chapter 6 instead.

Modern GPU partitioning and Blackwell multi-node devices are exposed through **Dynamic Resource Allocation (DRA)**, which reached GA in Kubernetes 1.34. DRA replaces the older device-plugin model for advanced cases — fractional GPUs, MIG profiles requested at submission time, and GB200/GB300 multi-node NVLink domains (§6.4).

5.4 Private Registry & Artifact Management

An air-gapped factory cannot pull from NGC, Docker Hub, or PyPI at runtime. Every image, model, Helm chart, and Python wheel is served from inside the perimeter.

Artifact class	Internal system	Source / sync path
Container images	Harbor (OCI registry)	Mirrored from NGC via staged transfer; signed and scanned on ingest.
Helm / OCI charts	Harbor OCI	e.g. `oci://ghcr.io/nvidia/kai-scheduler/kai-scheduler` mirrored to local path.

Artifact class	Internal system	Source / sync path
Model artifacts	S3 object store + NIM cache	NIM download-to-cache output and NeMo checkpoints (§9.4).
Python / OS packages	Local PyPI & APT mirror	Curated wheelhouse and package snapshots, version-pinned.
Git / config	Internal Git (GitOps source of truth)	Platform and tenant manifests; see §5.5.

Harbor provides project-level RBAC, Notation/Cosign signature verification, Trivy vulnerability scanning, and immutable tags. Admission control (§12.3) rejects any image not originating from the internal registry, closing the door on accidental external dependencies.

5.5 Air-Gapped Bootstrap & GitOps

The platform is defined as code and reconciled by a GitOps controller (Argo CD or Flux) running inside the perimeter. The internal Git repository is the single source of truth; the controller continuously converges cluster state to the committed manifests. Humans change the platform by merging a reviewed commit, not by `kubectl apply` against production.

Bringing up a sealed site follows a repeatable bootstrap:

Step	Action
1	Stage all artifacts in the connected staging enclave: mirror NGC images, NIM model caches, charts, wheels; sign and scan everything.
2	Transfer the validated artifact set across the air gap via approved media or a one-way data diode into the production registries.
3	Provision hosts from golden images; install Kubernetes; seat the GPU and Network Operators from the local registry.
4	Seat the local DLS, registry, and GitOps controller; point the controller at the internal Git repo.
5	Let GitOps converge the platform layer (scheduler, observability, gateway, guardrails); validate with automated conformance checks.

PROMOTION, NOT CONNECTION

The production enclave is never connected to the internet to "make it work." New software and models enter only as versioned artifacts that have passed staging gates and crossed the gap through the controlled transfer mechanism (§9.4). This single rule governs the entire lifecycle.

CHAPTER 06

GPU Orchestration & Multi-Tenancy

Pooling expensive accelerators and then partitioning them fairly across tenants and workloads is the economic heart of the factory. This layer is built on NVIDIA's scheduler lineage: the open-source KAI Scheduler engine, optionally wrapped by the Run:ai enterprise platform.

6.1 Run:ai vs. KAI Scheduler Selection

NVIDIA acquired Run:ai in late 2024 and in April 2025 open-sourced its scheduling core as the **KAI Scheduler** (Apache 2.0, a CNCF Sandbox project). Understanding the split is key to the build-vs-buy decision.

Capability	KAI Scheduler (open source)	Run:ai platform (enterprise)
Gang scheduling	Yes — via PodGrouper	Yes
Hierarchical queues & fair-share	Yes	Yes, with UI & policy
Fractional GPU / GPU sharing	Yes (soft, compute-level)	Yes
Bin-pack / spread / consolidation	Yes	Yes
Topology-aware scheduling & DRA	Yes	Yes
Enforced GPU memory isolation	No — relies on cooperative limits	**Yes — hard memory isolation**
Automated MIG management	Manual / Operator	**Automated**
GPU memory swap to host RAM	No	**Yes**
Multi-cluster management, RBAC, SSO	No	**Yes**
Support model	Community	NVAIE enterprise support

DECISION

Adopt **Run:ai** for production multi-tenancy. Hard GPU-memory isolation, automated MIG, GPU-memory swap, and multi-cluster RBAC/SSO are the precise controls a governed multi-tenant estate needs, and they are not in the open-source engine. The Pilot MAY start on KAI Scheduler alone to validate scheduling behavior at low cost, then layer Run:ai before tenant GA. Both share the same scheduling core, so the migration is additive rather than a re-platform.

6.2 Scheduling: Queues, Fair-Share & Gang

Capacity is organized as a tree of **queues**. Each tenant or project maps to a queue with a guaranteed deserved share, a fair-share weight, and an optional burst limit. Idle capacity is lent to busy queues and reclaimed by preemption when an owning queue submits work, which is how utilization climbs from the typical 15–35% toward the 65–75% NFR target while still honoring guarantees.

Distributed training requires **gang scheduling**: every worker of a job starts together or none does, preventing partially-placed jobs from deadlocking the cluster while holding GPUs. A representative queue definition:

```
apiVersion: scheduling.run.ai/v2
kind: Queue
metadata:
  name: tenant-acme-training
spec:
  parentQueue: research
  resources:
    gpu:
      quota: 16          # guaranteed (deserved) GPUs
      limit: 32          # hard burst ceiling
      overQuotaWeight: 2 # fair-share weight for idle capacity
    cpu:    { quota: 256, limit: 512 }
    memory: { quota: 2Ti, limit: 4Ti }
  preemptible: true       # may lend idle GPUs, reclaimed on demand
```

Scheduling policy per queue selects **bin-pack** (consolidate onto fewest nodes — best for leaving room for large jobs and powering down idle nodes) or **spread** (distribute for latency-sensitive inference). Priorities distinguish `build`, `train`, and `inference-production` classes, with production inference made non-preemptible.

6.3 GPU Partitioning: MIG, Slicing, Fractional

One Blackwell GPU is too large a unit for many tasks — a notebook, a small embedding model, a CI job. Three partitioning mechanisms right-size the allocation:

Mechanism	Isolation	Best for
MIG (Multi-Instance GPU)	Hardware-partitioned compute & memory	Guaranteed QoS slices for inference and untrusted tenants.
Time-slicing	Temporal sharing, no memory isolation	Bursty dev/interactive work tolerant of contention.
Fractional GPU (Run:ai)	Memory-fraction limits, enforced under Run:ai	Packing many small models with predictable memory bounds.

Run:ai adds **GPU-memory swap**, paging idle workloads' GPU memory to host RAM so more models can be resident than raw VRAM allows — valuable for serving a long tail of intermittently-hit tenant models. MIG geometry (e.g. `1g.24gb` through `7g.180gb` on B200-class parts) is automated by Run:ai based on queue demand rather than statically pre-cut.

> **ISOLATION GUIDANCE**
>
> For untrusted or strongly-isolated tenants, prefer **MIG** (hardware boundary) or whole-GPU allocation. Reserve time-slicing for cooperative, same-trust-domain workloads — it provides no memory protection between co-tenants. Map the isolation grade from §2.4 to the partitioning mechanism explicitly in each tenant's profile.

6.4 Topology-Aware Scheduling & DRA

On multi-GPU and multi-node jobs, where GPUs are placed dominates performance. NVLink-connected GPUs within a node, and NVLink/NVSwitch domains within a GB200 NVL72 rack, deliver far higher bandwidth than crossing the InfiniBand spine. **Topology-Aware Scheduling** places a job's workers to maximize intra-domain links — packing within an NVLink domain first, then within a leaf, before spanning the spine.

Dynamic Resource Allocation (DRA), GA in Kubernetes 1.34, is the mechanism for requesting these structured resources. Workloads express a need ("8 GPUs in one NVLink domain", or "a `2g.48gb` MIG slice") through a ResourceClaim, and the scheduler binds matching devices. DRA is also how GB200/GB300 multi-node NVLink domains are represented as schedulable units rather than opaque single GPUs.

6.5 Tenant Isolation Model

Isolation is layered, with each tenant pinned to a grade from §2.4:

Dimension	Control
Namespace	One or more Kubernetes namespaces per tenant; all quota and policy scoped to them.
Scheduling	Dedicated queue subtree with guaranteed share and burst ceiling.
Compute	MIG / whole-GPU for hard isolation; fractional for soft, cooperative tenants.
Network	Default-deny NetworkPolicy; explicit east-west allowlists (§12.1).
Storage	Per-tenant buckets/volumes with distinct credentials and optional keys (§11.4).
Identity	Tenant-scoped RBAC bound to SSO groups; no cross-tenant API visibility.

6.6 Quota, Metering & Showback

Every GPU-second is attributable. DCGM and scheduler accounting record per-namespace, per-queue consumption — GPU-hours, MIG-slice-hours, memory, and energy where instrumented — feeding dashboards and a showback/chargeback export (§11.5). Quotas are enforced at admission (a tenant cannot exceed its queue limit) and surfaced to tenants in self-service views so they see remaining headroom before submitting. This accounting is the evidence base for capacity planning in Chapter 15 and the commercial model for the platform.

CHAPTER 07

AI Training & Model Customization (NeMo)

The R&D and training mission is served by the NVIDIA NeMo framework — an end-to-end toolkit for curating data, training and fine-tuning models at scale, and aligning them to intended behavior. NeMo runs as containerized jobs on the same Run:ai-scheduled pool as inference.

7.1 The NeMo Customization Pipeline

Most tenant value comes not from pre-training from scratch but from **customizing** capable base models on proprietary data. NeMo structures this as a pipeline, each stage a schedulable job:

Stage	NeMo component	Function
Data curation	NeMo Curator	Dedup, filter, quality-score, and PII-handle large text/ multimodal corpora at GPU scale.
Training / fine-tune	Megatron-Core	Tensor/pipeline/data/expert/context parallelism for efficient large-model training.
Parameter-efficient tune	NeMo PEFT	LoRA / QLoRA / p-tuning adapters at a fraction of full-fine-tune cost.
Alignment	NeMo Aligner	SFT, DPO, RLHF, and reward-model training to shape behavior.
Evaluation	NeMo Evaluator / harnesses	Benchmark and regression-test candidates before promotion.
Export	TensorRT-LLM / NIM	Compile and package the result for serving (Ch 8, Ch 9).

The pipeline terminates at the model registry: a validated checkpoint, versioned and signed, ready for the quantization and serving paths.

7.2 Distributed Training & Parallelism

Large models exceed a single GPU and often a single node. Megatron-Core composes several parallelism strategies, which the platform must place correctly via topology-aware scheduling (§6.4):

Parallelism	What it splits
Data (DP)	The batch — replicas process different shards, gradients all-reduced.
Tensor (TP)	Individual layers across GPUs — bandwidth-hungry, kept within an NVLink domain.
Pipeline (PP)	Layers into sequential stages across nodes — tolerant of slower links.
Expert (EP)	MoE experts across GPUs — essential for large mixture-of-experts models.
Context (CP)	The sequence dimension — enables very long context windows.

PLACEMENT MATTERS

Tensor parallelism MUST stay inside a high-bandwidth NVLink domain; pipeline parallelism can cross the InfiniBand spine. The scheduler's topology awareness is what keeps a poorly-placed TP group from collapsing throughput. GB200 NVL72's 72-GPU NVLink domain is what makes very large TP groups practical (§4.1).

7.3 Fine-Tuning, PEFT & Alignment

The everyday tenant workflow is parameter-efficient fine-tuning. **LoRA/QLoRA** train small adapter weights over a frozen base, cutting GPU and storage cost by orders of magnitude and enabling many tenant-specific variants over one shared base model — which NIM can serve as hot-swappable LoRA adapters (§8.1). Full fine-tuning and continued pre-training remain available for tenants who need them. **Alignment** (SFT then DPO/RLHF via NeMo Aligner) shapes tone, format, and refusal behavior, and is where intended-use policy is baked into weights — complemented at runtime by Guardrails (Chapter 10).

7.4 Experiment, Data & Lineage

R&D needs reproducibility. Every training run records its inputs and outputs so any served model can be traced to its origin:

- **Experiment tracking** — hyperparameters, metrics, and curves captured per run (e.g. MLflow/W&B-class tooling hosted on-prem).
- **Data versioning** — datasets are content-addressed and immutable; a run pins exact dataset versions.
- **Model lineage** — base model, dataset versions, code commit, and config are bound to each checkpoint, producing an auditable chain from raw data to served endpoint (feeds provenance in §9.5 and audit in §12.5).

7.5 Scheduling Training on Run:ai

Distributed training is submitted as a gang-scheduled, topology-aware job against the tenant's queue. A representative multi-node manifest using the training operator:

```
apiVersion: kubeflow.org/v1
kind: PyTorchJob
metadata:
  name: acme-llm-sft
  namespace: tenant-acme
  labels:
    runai/queue: tenant-acme-training   # bind to fair-share queue
spec:
  pytorchReplicaSpecs:
    Master: { replicas: 1, template: {spec: &gpuspec
      schedulerName: runai-scheduler
      containers:
        - name: trainer
          image: registry.internal/nvaie/nemo:24.09
          command: ["torchrun", "sft.py", "--tp=8", "--pp=2"]
          resources:
            limits: { nvidia.com/gpu: 8 }
      } }
    Worker: { replicas: 3, template: {spec: *gpuspec} }  # 4 nodes x 8 = 32 GPUs
```

The job lands only when all 32 GPUs can start together (gang), placed to honor TP locality (topology), and is preemptible against higher-priority production inference per queue policy. Checkpoints write to the hot parallel filesystem on the cadence defined for training resiliency (§14.2).

CHAPTER 08

Inference Serving Plane

The inference plane turns trained models into governed, low-latency services for tenant workflows. It is built on NVIDIA NIM microservices — each a self-contained, OpenAI-compatible endpoint wrapping an optimized backend — orchestrated on Kubernetes and fronted by a model gateway.

8.1 NIM Microservices Architecture

A **NIM** (NVIDIA Inference Microservice) packages model weights, an inference engine, and a standard API into one container. On startup it inspects the GPU and selects an optimized backend automatically — **TensorRT-LLM** on supported GPUs for peak performance, falling back to **vLLM** (or SGLang) where a pre-built engine is unavailable. It exposes an OpenAI-compatible REST interface, so tenant code targets a stable contract regardless of the backend underneath.

NIM is part of NVAIE and is the standard unit of deployment for inference here. Key properties the platform relies on:

- **Backend abstraction** — the same NIM image runs optimized or portable depending on hardware and available engine, easing the air-gapped and quantized paths in Chapter 9.
- **LoRA multi-adapter serving** — one base-model NIM can serve many tenant LoRA adapters concurrently, the serving counterpart to PEFT (§7.3).
- **Air-gap support** — models are pre-staged to a local cache via download-to-cache; no external pull at runtime (§9.4).
- **Profiles** — latency- vs throughput-optimized engine profiles, including `trtllm_buildable` profiles compiled on-site when a prebuilt engine is not shipped.

8.2 Triton Inference Server

Inside many NIMs, and available directly for custom and non-LLM models, is **Triton Inference Server** — NVIDIA's production serving runtime. Triton supports multiple frameworks (TensorRT, PyTorch, ONNX, Python), dynamic batching, concurrent model execution, and model ensembles, all behind HTTP/gRPC with a metrics endpoint. A minimal LLM-engine model configuration:

```
# config.pbtxt
name: "llama-t"
backend: "tensorrtllm"
max_batch_size: 64
model_transaction_policy { decoupled: true }   # streaming tokens
input  [ { name: "input_ids",  data_type: TYPE_INT32, dims: [-1] } ]
output [ { name: "output_ids", data_type: TYPE_INT32, dims: [-1] } ]
instance_group [ { count: 1, kind: KIND_GPU } ]
dynamic_batching { max_queue_delay_microseconds: 1000 }
```

For custom tenant models (rerankers, embedders, classifiers, vision) that are not LLMs, Triton is the direct serving path; for standard generative models, NIM (which embeds Triton/TRT-LLM) is preferred for its turnkey packaging.

8.3 TensorRT-LLM

TensorRT-LLM is the high-performance LLM backend. It compiles a model into a hardware-specific engine and applies the optimizations that make Blackwell economical at scale:

Technique	Effect
In-flight (continuous) batching	New requests join the batch as others finish — high throughput under mixed load.
Paged KV cache	Non-contiguous KV memory cuts fragmentation, raising concurrency per GPU.
FP8 / FP4 quantized kernels	Blackwell-native low precision for large speedups at controlled accuracy (Ch 9).
Speculative decoding	A draft model proposes tokens a target model verifies — lower latency.
Tensor parallelism	Splits very large models across GPUs for models that exceed one device.

Engines are built per target GPU and precision. In the air-gapped flow they are either pre-built in staging and shipped as artifacts, or compiled on-site from `trtllm_buildable` NIM profiles (§9.2).

8.4 vLLM

vLLM is the portable, flexible serving path. Its PagedAttention KV management delivers strong throughput without a per-GPU engine-compile step, so it serves new or custom model architectures immediately and runs the same model across heterogeneous hardware. It also reads quantized weights directly — AWQ, GPTQ, FP8, and (experimentally) GGUF — which makes it the engine of the sovereign/portable tier (§9.3).

Prefer TensorRT-LLM when…	Prefer vLLM when…
Peak throughput / lowest latency on NVIDIA GPUs is the goal	Flexibility, fast bring-up, or portability matters more
The model + precision has a buildable engine path	Architecture is new/custom or no engine is available
Workload is stable and high-volume (worth compiling)	Serving GGUF/AWQ/GPTQ or running the CPU/portable tier

8.5 NIM Operator on Kubernetes

The **NIM Operator** manages inference services declaratively through custom resources, so model serving is GitOps-managed like everything else. Three resources do the work: `NIMCache` pre-stages model artifacts into a local cache (the air-gap primitive), `NIMService` runs a model as a scalable, autoscaling endpoint, and `NIMBuild` compiles an optimized engine on-site. A service backed by a pre-populated cache:

```
apiVersion: apps.nvidia.com/v1alpha1
kind: NIMCache
metadata: { name: llama3-8b-cache, namespace: inference }
spec:
  source:
    ngc: { modelPuller: "registry.internal/nim/llama-3-8b-instruct:latest" }
  storage: { pvc: { create: true, size: 60Gi, storageClass: rwx-nfs } }
---
apiVersion: apps.nvidia.com/v1alpha1
kind: NIMService
metadata: { name: llama3-8b, namespace: inference }
spec:
  image: { repository: registry.internal/nim/llama-3-8b-instruct, tag: latest }
  storage: { nimCache: { name: llama3-8b-cache } }   # no external pull
  replicas: 2
  resources: { limits: { nvidia.com/gpu: 1 } }
  scale: { enabled: true, minReplicas: 2, maxReplicas: 8 }   # HPA on GPU/latency
  expose: { service: { type: ClusterIP, port: 8000 } }
```

Endpoints scale horizontally on GPU utilization and latency; the cache reference guarantees the air-gapped service never reaches outside the perimeter.

8.6 Disaggregated Serving (Dynamo / Grove)

At Scale, the most demanding models benefit from **disaggregated serving** — separating the compute-bound prefill phase from the memory-bandwidth-bound decode phase onto different GPU pools, each scaled independently. NVIDIA **Dynamo** provides this datacenter-scale inference orchestration, with **Grove** handling the multi-component scheduling on Kubernetes. This is an optimization for high-volume frontier endpoints, not the default for routine tenant services, and is introduced when volume justifies the added topology.

8.7 Model Gateway & Tenant Catalog

A single **model gateway** fronts all inference. It presents a unified OpenAI-compatible API, authenticates the caller, resolves the requested model from a per-tenant **catalog** (only models a tenant is entitled to are visible), applies rate limits and quotas, routes to the correct NIM/Triton backend, and emits metering and audit records. Guardrails (Chapter 10) attach at this boundary as input and output rails. The consolidated request path is shown in Figure 8.1.

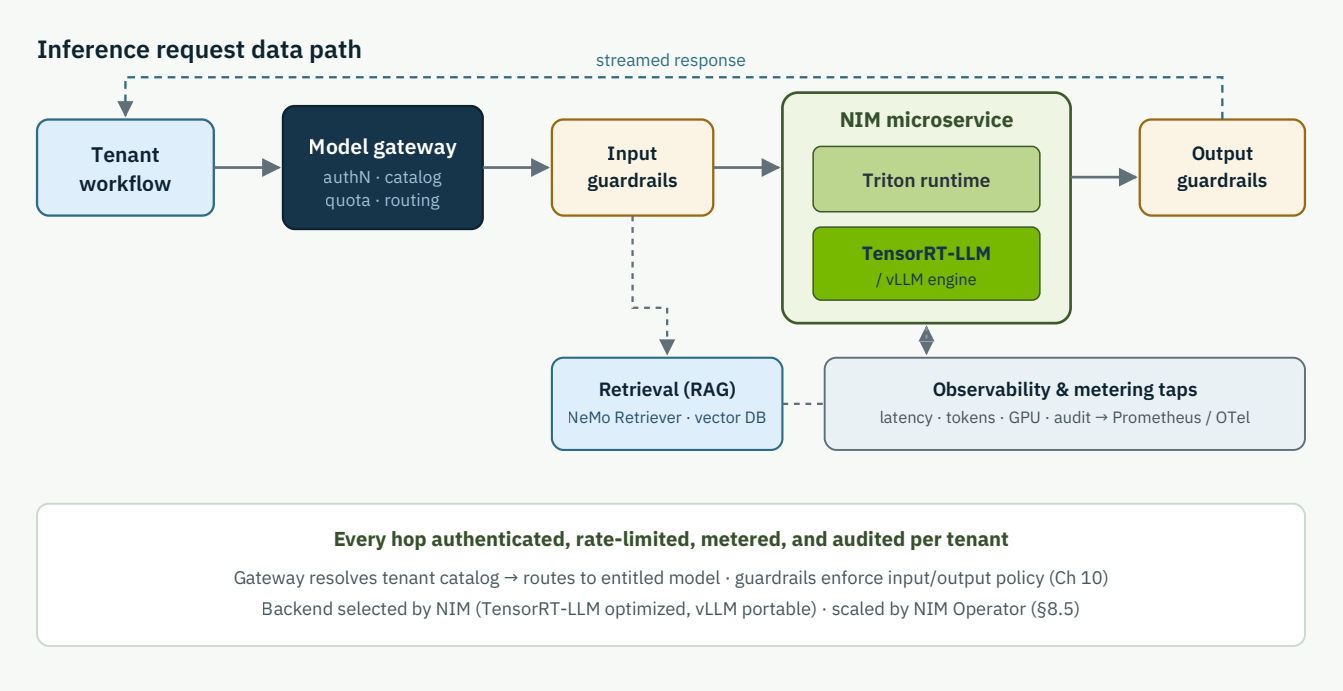


Figure 8.1 — Consolidated inference request path. A request flows through the gateway, input rails, and a NIM-served engine, with an optional retrieval branch; responses pass output rails back to the tenant. Observability and metering tap every stage.

CHAPTER 09

Quantization & Air-Gapped Model Lifecycle

Two concerns converge in this chapter: making models small and fast enough to serve economically (quantization), and moving them into a sealed facility safely (the air-gapped lifecycle). Together they define how a model gets from a training checkpoint to a trusted production endpoint with no internet in the loop.

9.1 Quantization Strategy Matrix

Quantization reduces weight and activation precision to cut memory and increase throughput, trading a controlled amount of accuracy. The platform supports a spectrum and selects per workload:

Format	Precision	Best for	Engine
FP8	8-bit float	Blackwell-native; near-lossless, large speedup	TensorRT-LLM, vLLM
FP4 / NVFP4	4-bit float	Maximum Blackwell throughput for huge models	TensorRT-LLM
INT4 AWQ	4-bit weight	Strong accuracy retention at 4-bit; GPU serving	TensorRT-LLM, vLLM
INT4 GPTQ	4-bit weight	Widely-available 4-bit checkpoints	vLLM
GGUF (Q4–Q8)	2–8-bit	Portable / CPU / sovereign tier, single-file	llama.cpp, vLLM*

*GGUF support in vLLM is functional but considered experimental; llama.cpp is the reference runtime for GGUF.

DECISION — TWO PRECISIONS, TWO PATHS

Default production serving on Blackwell uses **FP8** (accuracy-safe, big gains), stepping to **FP4/INT4-AWQ** for the largest models where memory or throughput demands it and evaluation confirms acceptable quality. The **GGUF** path exists for portability and sovereignty, not peak GPU performance. Every quantized model MUST pass the evaluation gate (§7.1) before promotion — quantization is never applied blindly to a production endpoint.

9.2 GPU-Optimized Path (TensorRT-LLM)

For maximum performance on the factory's own GPUs, models take the **TensorRT-LLM** path. The NVIDIA **Model Optimizer** applies post-training quantization (FP8/FP4/INT4) and calibration, then TensorRT-LLM compiles a hardware-specific engine with paged KV cache and in-flight batching (§8.3). Engines are precision- and GPU-specific, so the build target matches the deployment hardware exactly. In the air-gapped estate, engines are either built in the staging enclave and shipped as artifacts, or compiled on-site from `trtllm_buildable` NIM profiles via a `NIMBuild` resource (§8.5).

9.3 Portable / Sovereign Path (GGUF & vLLM)

Some requirements outweigh peak throughput: running on heterogeneous or CPU hardware, full sovereignty over a single self-contained model file, edge or DR sites without Blackwell, or rapid bring-up of a community model. The **GGUF** format — a single portable file from the llama.cpp ecosystem — serves these via **llama.cpp** or vLLM. This tier accepts lower performance in exchange for portability and independence from compiled engines, and underwrites the "sovereign fallback" posture: the factory can serve critical models with no NVIDIA-specific engine at all if it must. **vLLM** additionally serves AWQ/GPTQ/FP8 weights directly without a compile step, bridging the optimized and portable worlds.

9.4 Air-Gapped Artifact Pipeline

No model reaches production by being downloaded there. Artifacts are pulled and validated in a connected **staging enclave**, then promoted across the air gap as immutable, signed packages. NIM's download-to-cache populates a local model cache in staging; that cache, with its engine profiles and manifests, becomes the transfer unit.

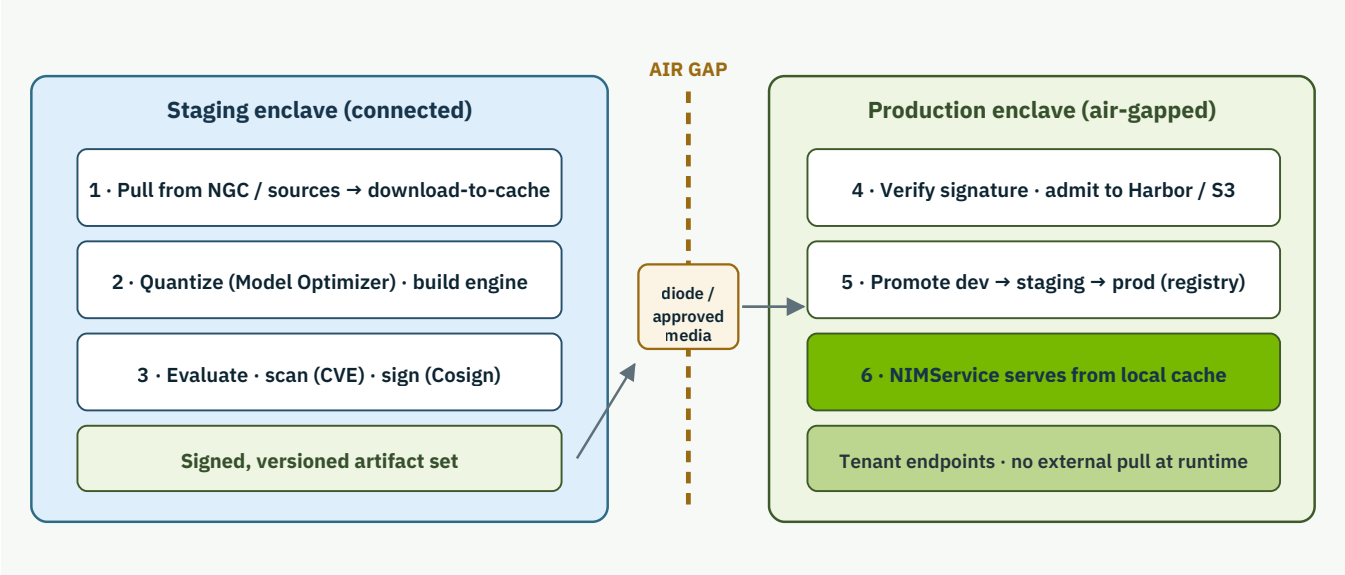


Figure 9.1 — Air-gapped model promotion pipeline. Models are pulled, quantized, evaluated, scanned, and signed in the connected staging enclave, then crossed through a controlled transfer into the sealed production registry, where signature verification gates admission before serving.

9.5 Model Provenance & Registry Promotion

Trust in a served model rests on provenance. Each artifact carries a signed manifest binding it to its lineage (§7.4) — base model, datasets, training config, code commit, quantization recipe, and evaluation results. Promotion is gated:

Stage	Gate to advance
dev	Builds and serves; smoke tests pass.

Stage	Gate to advance
staging	Evaluation thresholds met; CVE scan clean; signature attached.
prod	Signature verified on admission; approval recorded; immutable tag pinned.

Signing uses Cosign/Notation; admission control (§12.3) refuses any model or image lacking a valid signature from the trusted key. The result is an auditable chain: every production endpoint resolves to a specific, verified artifact whose entire history — data to engine — can be reconstructed for security review or incident response.

CHAPTER 10

Safety, Guardrails & Responsible AI

Governed inference means more than uptime: tenant-facing AI services must enforce safety, topic, and security policy on every request and response. NeMo Guardrails and the NemoGuard safety models provide this programmable control layer at the serving boundary.

10.1 Guardrails Architecture & Rail Types

NeMo Guardrails is a programmable layer that sits between the application and the model, evaluating policy at defined points in the request lifecycle. Policies — rails — are expressed in **Colang**, a modeling language for conversational flows and constraints. Five rail types cover the lifecycle:

Rail	Acts on
Input	The user prompt — block jailbreaks, off-policy or unsafe requests before the model sees them.
Dialog	Conversation flow — steer or constrain multi-turn behavior along permitted paths.
Retrieval	RAG context — filter or redact retrieved chunks before they enter the prompt.
Output	The model response — check for unsafe, off-topic, or non-compliant content before return.
Execution	Tool/function calls — authorize and bound actions the model may invoke.

10.2 NemoGuard Safety Models

Rails invoke purpose-built **NemoGuard** models, themselves served as NIMs on the same platform, for high-quality classification rather than brittle keyword rules:

- **Content Safety** — classifies prompts and responses against a safety taxonomy (e.g. the Aegis-trained guard model).
- **Topic Control** — keeps a tenant assistant within its sanctioned domain, declining out-of-scope subjects.
- **Jailbreak Detection** — recognizes prompt-injection and guard-circumvention attempts.

Because the guards are NIMs, they inherit the platform's scaling, observability, and air-gap properties — no external moderation API is called, which is mandatory in the sealed enclave.

10.3 Per-Tenant Policy Configuration

Each tenant gets a Guardrails configuration matched to its risk profile and domain — a research sandbox may run permissive rails, while an externally-facing tenant assistant runs strict content, topic, and jailbreak rails. Configuration is declarative and version-controlled alongside the tenant's other manifests. A representative config binding guard models to rails:

```
# config.yml — tenant guardrails
models:
  - type: main
    engine: nim
    model: registry.internal/nim/llama-3-8b-instruct
rails:
  input:
    flows: [ self check input, jailbreak detection ]
  output:
    flows: [ self check output, content safety check ]
  config:
    content_safety:
      model: registry.internal/nim/nemoguard-content-safety
    jailbreak_detection:
      model: registry.internal/nim/nemoguard-jailbreak-detect
```

```
# rails.co — Colang: refuse off-policy input
define flow self check input
  $allowed = execute content_safety_check(text=$user_message)
  if not $allowed
    bot refuse politely
    stop
```

10.4 Placement in the Request Path

Guardrails attach at the model gateway (§8.7) as the input and output rails shown in Figure 8.1, so every tenant request is mediated regardless of the calling application — policy cannot be bypassed by a misbehaving client. Rail evaluations are logged to the audit trail (§12.5) and metered, giving operators visibility into block rates and triggers. Guardrails complement, and do not replace, alignment baked into weights during training (§7.3): the model is shaped to behave well, and the rails enforce policy deterministically at the boundary. Latency cost of the rails is budgeted within the end-to-end TTFT NFR (§2.2); guard NIMs are sized so safety checks do not dominate the response path.

CHAPTER 11

Multi-Tenant Workflow Platform

The factory's third mission — beyond R&D and training — is to deliver AI services for custom tenant workflows. This chapter defines how tenants are onboarded, what they self-serve, the workflow patterns supported, and how their data and consumption are isolated and accounted.

11.1 Tenant Onboarding Lifecycle

A tenant is a bounded unit of identity, quota, isolation, and billing. Onboarding is automated and GitOps-driven so a new tenant is a reviewed commit, not a manual build:

Step	Action
1	Register tenant: SSO group mapping, isolation grade (§2.4), and quota tier agreed.
2	Provision namespace(s), scheduler queue (§6.2), and default-deny network policy.
3	Create storage buckets/volumes with tenant-scoped credentials and keys (§11.4).
4	Bind RBAC roles; publish entitled models to the tenant catalog (§8.7).
5	Apply Guardrails policy (§10.3); issue gateway API credentials.
6	Enable metering/showback and self-service dashboards.

11.2 Self-Service & Quotas

Tenants operate within their boundary without platform-team tickets: submit training and fine-tuning jobs to their queue, deploy and scale entitled model endpoints, manage datasets in their buckets, and view consumption against quota. The platform enforces guardrails on this autonomy — queue limits cap GPU burst, admission policy constrains what can run, and namespace scoping prevents any cross-tenant visibility. Self-service raises utilization and removes the platform team as a bottleneck while keeping hard controls in force.

11.3 Custom Tenant Workflow Patterns

The platform supports recurring patterns rather than bespoke one-offs:

Pattern	Composition
RAG application	Embedding + reranker NIMs (NeMo Retriever) + vector DB + generative NIM, mediated by guardrails.
Agentic workflow	LLM NIM with tool/function calling, bounded by execution rails (§10.1).
Fine-tune-as-a-service	Tenant data → NeMo PEFT job → LoRA adapter → served on a shared base NIM (§8.1).
Batch inference	Scheduled, preemptible jobs over tenant datasets for bulk scoring/extraction.
Custom model serving	Tenant-supplied model on Triton/vLLM where it meets platform packaging and scan gates.

11.4 Data Isolation & Residency

Data isolation is foundational to multi-tenancy and to many tenants' compliance obligations. Each tenant's data lives in dedicated buckets/volumes with distinct credentials; cross-tenant access is impossible by construction, not by convention. Where required, per-tenant encryption keys (envelope encryption via the platform secrets/KMS in §12.2) ensure even platform operators cannot read tenant data without authorization. Retrieval indexes, fine-tuned adapters, and logs are equally tenant-scoped. Because the entire factory is on-premises and air-gapped, data residency is inherent — tenant data never leaves the facility, which is often the decisive reason an organization builds this platform rather than using a public API.

> **RESIDENCY BY CONSTRUCTION**
>
> On-prem + air-gapped means tenant prompts, documents, embeddings, and fine-tunes physically remain in the controlled facility. This satisfies sovereignty and regulated-data constraints that public inference cannot, and is a primary justification for the build (§1.1).

11.5 Tenant API Gateway & Metering

Tenant workflows reach inference exclusively through the model gateway (§8.7), which authenticates per-tenant credentials, enforces rate limits and quotas, routes to entitled models, and records consumption. Metering captures requests, input/output tokens, GPU-time, and endpoint uptime per tenant, feeding a showback/chargeback export and the capacity baseline (Chapter 15). The gateway is also the single point where guardrails, audit, and quota converge — one boundary to secure, observe, and bill, which is what makes multi-tenancy operationally tractable at scale.

CHAPTER 12

Security Architecture

Security is cross-cutting, not a layer — it spans hardware, cluster, supply chain, data, and runtime. The posture is zero-trust and defense-in-depth, appropriate to a multi-tenant platform handling proprietary models and regulated data inside an air-gapped perimeter.

12.1 Zero-Trust & Network Segmentation

No component is trusted by location. The network is segmented into the domains of §3.4 — management, production (air-gapped), and the staging enclave — with controlled transfer between them. Inside the cluster, the default posture is **deny**: every namespace starts with a default-deny `NetworkPolicy`, and east-west traffic is permitted only by explicit allowlist. A service mesh provides mutual TLS for service-to-service traffic where required. The air gap itself is the outermost control: no inbound or outbound internet path exists in production, eliminating entire classes of exfiltration and command-and-control risk.

12.2 Identity, RBAC & Secrets

Identity flows from the enterprise IdP via SSO (OIDC/SAML) into Kubernetes RBAC and Run:ai roles. Access is least-privilege and tenant-scoped: a tenant's users see and act only within their namespaces and queue. Platform, operator, and tenant roles are distinct, and privileged operations require approval and are audited. **Secrets** — credentials, tokens, encryption keys — are held in a secrets manager (e.g. HashiCorp Vault) rather than in manifests or images, with dynamic leasing and rotation; tenant data keys are managed for envelope encryption (§11.4). Secret material is never committed to Git; GitOps references secrets by pointer, resolved at deploy time.

12.3 Supply-Chain Security & Admission

In an air-gapped estate, the supply chain is the primary attack surface — what crosses the gap must be trustworthy. Controls:

- **Signing** — all images and model artifacts signed (Cosign/Notation) in staging (§9.5).
- **SBOM & scanning** — software bills of materials generated and vulnerability-scanned (Trivy) before promotion; failing artifacts do not cross.
- **Admission control** — a policy engine (Kyverno or OPA Gatekeeper) rejects non-compliant workloads at the API boundary.

A representative admission rule requiring signed images from the internal registry only:

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata: { name: require-internal-signed-images }
spec:
  validationFailureAction: Enforce
  rules:
    - name: registry-allowlist
      match: { any: [ { resources: { kinds: [ Pod ] } } ] }
      validate:
        message: "Images must come from registry.internal"
        pattern:
          spec:
            containers:
              - image: "registry.internal/*"
    - name: verify-signature
      match: { any: [ { resources: { kinds: [ Pod ] } } ] }
      verifyImages:
        - imageReferences: [ "registry.internal/*" ]
          attestors:
            - entries: [ { keys: { publicKeys: "{{ cosign_pubkey }}" } } ]
```

12.4 Confidential Computing

For the highest-sensitivity tenants, Blackwell's **confidential computing** capabilities extend protection to data in use. GPU Confidential Computing establishes a hardware trusted execution environment with attestation, so model weights and tenant data are protected in GPU memory even from a privileged host. This is an optional control for tenants whose threat model or regulatory regime demands in-use protection, applied selectively given its operational overhead rather than enabled platform-wide.

12.5 Audit, Logging & Compliance

Every privileged and tenant-facing action is recorded to a tamper-evident audit trail: Kubernetes API audit logs, gateway request logs, guardrail decisions (§10.4), model-promotion approvals (§9.5), and access to secrets and tenant data. Logs are centralized (§13.2), retained per policy, and queryable for investigation. The lineage and provenance chains make it possible to answer, for any output, which model version produced it from which data — the evidentiary backbone for compliance audits, incident response, and tenant assurance.

CHAPTER 13

Observability & Operations

An expensive GPU estate must be observable to be operated and to be paid for. The platform instruments GPUs, infrastructure, and services into a unified telemetry stack, with NVIDIA's operational tooling for fleet health and a clear SLO and alerting model.

13.1 Metrics & GPU Telemetry

The metrics backbone is Prometheus, fed by exporters across the stack. The **DCGM Exporter** (deployed by the GPU Operator) is central, surfacing per-GPU utilization, memory, power, temperature, SM occupancy, NVLink and PCIe throughput, and ECC/XID error events. Combined with node, Kubernetes, and gateway metrics, this gives the full picture from silicon to service:

Source	Signals
DCGM exporter	GPU util, memory, power, temp, NVLink/PCIe, XID errors.
Node exporter	CPU, host memory, disk, network.
Kubernetes / kube-state	Pod, queue, scheduling, and capacity state.
Gateway / NIM / Triton	Request rate, latency (TTFT, inter-token), tokens, batch, queue depth.
Scheduler (Run:ai)	Per-queue allocation, fair-share, preemption, wait times.

Grafana provides dashboards by audience: a fleet view for operators, per-tenant consumption for showback, and per-service latency for SLO tracking. GPU utilization — the headline efficiency metric — is tracked against the 65–75% NFR target (§2.2).

13.2 Logging & Tracing

Logs are aggregated centrally (e.g. Loki or an ELK/OpenSearch stack) from containers, the control plane, the gateway, and guardrails, with tenant labels for scoped queries and retention by policy. Distributed tracing via OpenTelemetry follows a request across gateway, guardrails, retrieval, and model backends, making latency contributors and failures diagnosable hop by hop. Audit logs (§12.5) are segregated with stricter retention and access controls than operational logs.

13.3 Health, Faults & Mission Control

At fleet scale, hardware faults are routine and must be detected and remediated automatically. DCGM health checks and the GPU Operator's validation surface unhealthy GPUs; the scheduler cordons and drains affected nodes so jobs reschedule rather than fail silently. For larger deployments, **NVIDIA Mission Control** (with Base Command capabilities) provides cluster-level operations — provisioning, health, workload, and autonomous

recovery — purpose-built for DGX/GB200 estates, complementing the open observability stack with vendor fleet management. XID error patterns feed predictive maintenance, pulling suspect GPUs before they fail mid-job.

13.4 SLOs & Alerting

Service levels are explicit and tied to the NFRs (§2.2):

SLO domain	Indicator	Target (reference)
Inference latency	TTFT / inter-token	≤300 ms / ≤50 ms (model-class dependent)
Service availability	Endpoint uptime	≥99.5% production inference
Cluster efficiency	GPU utilization	≥65–75% sustained
Scheduling	Job queue wait	Within tier-defined bounds

Alerts are defined on SLO burn and on infrastructure faults (GPU XID, thermal, fabric link down, capacity exhaustion), routed by severity to on-call with runbooks. Alerting favors symptom-based, actionable signals over raw thresholds to limit fatigue.

CHAPTER 14

Reliability, Availability & DR

The platform must survive component failure without losing tenant services or long-running training, and must recover from site-level events — all within an air-gapped constraint that rules out cloud failover. This chapter defines HA, training resiliency, backup, and disaster recovery.

14.1 High-Availability Design

No single failure should take down the platform. Control and data planes are made redundant:

Layer	HA approach
Kubernetes control plane	Multiple control-plane nodes; quorum etcd (§14.3).
Inference endpoints	≥2 replicas per production NIM, spread across nodes; gateway load-balances.
Model gateway	Multiple replicas behind a virtual IP / load balancer.
Storage	Redundant parallel-FS controllers; replicated/erasure-coded object store.
Fabric	Redundant leaf-spine paths; multi-port host attach (§4.2).
Power	N+1 PSUs/PDUs, A/B feeds; CDU redundancy for liquid-cooled racks.

Production inference endpoints are non-preemptible (§6.2) so capacity pressure from training never degrades tenant services.

14.2 Training Resiliency & Checkpointing

Large training jobs run for days across many GPUs, where a single GPU or node fault would otherwise waste enormous compute. Resiliency rests on disciplined **checkpointing**: model and optimizer state written to the hot parallel filesystem at an interval balancing overhead against re-compute risk, so a failed job resumes from the last checkpoint rather than from zero. NeMo and the training operators support automatic restart on the surviving allocation; gang scheduling (§6.2) re-places the job when healthy capacity is available, and faulted nodes are cordoned (§13.3). Checkpoints are also the basis of model lineage and rollback.

14.3 Backup & Restore

Platform state and tenant assets are backed up on defined schedules with tested restores:

Asset	Protection
etcd (cluster state)	Periodic snapshots, off-node retention; restore drills.
Registry (Harbor) & models	Replicated object storage; signed artifacts re-verifiable.
GitOps repositories	Mirrored; the source of truth for full platform rebuild.

Asset	Protection
Tenant data & checkpoints	Snapshots/replication per tenant policy and residency rules.
Secrets / KMS	Backed up and recoverable under strict control (§12.2).

Because the platform is declarative, GitOps plus artifact and data backups allow reconstruction of the entire stack onto fresh hardware — configuration is recovered by re-converging from Git, not by restoring imperatively-built servers.

14.4 Disaster Recovery (Air-Gapped)

Cloud failover is unavailable by design, so DR is on-premises. The reference posture is a second site (or a segregated hall) holding mirrored registries, GitOps repositories, model artifacts, and replicated tenant data, able to stand up services from backups within the recovery objectives. RPO is governed by replication/checkpoint cadence and RTO by rebuild-and-converge time; both are set per tier (§4.5) and validated by exercises. The portable/GGUF tier (§9.3) is a deliberate DR asset: critical models can be served on modest or non-Blackwell hardware at a recovery site when full GPU capacity is unavailable, preserving essential tenant services through degraded conditions.

DECISION — DECLARATIVE RECOVERY

DR strategy rests on the platform being fully reproducible from Git plus artifact/data backups. This makes recovery a convergence operation rather than a manual rebuild, shrinking RTO and removing human error from the critical path. Recovery procedures MUST be exercised on the cadence set in Chapter 16.

CHAPTER 15

Capacity Planning & Sizing

Sizing the factory means translating workload intent into GPUs, memory, storage, and fabric. This chapter gives a methodology for both training and inference and expands the reference tiers of §4.5 into indicative bills of materials.

15.1 Workload Characterization

Capacity follows demand, so the first step is to characterize it. The platform's three missions size differently:

Workload	Driving parameters
Training / fine-tune	Model size, dataset tokens, target wall-clock, parallelism, checkpoint cadence.
Inference (online)	Model size/precision, concurrent requests, context length, TTFT/inter-token SLO.
Inference (batch)	Volume, deadline, acceptable preemption.
R&D / interactive	Number of users, burstiness, tolerance for sharing (MIG/fractional).

Real consumption from metering (§11.5) feeds back into this characterization, so sizing becomes evidence-based after the Pilot rather than purely projected.

15.2 Sizing Methodology

Training compute. A widely-used first-order estimate for transformer pre-training is `C ≈ 6 · N · D` FLOPs, where `N` is parameter count and `D` is training tokens. Dividing by aggregate sustained GPU throughput (per-GPU peak FLOPs × realistic utilization, typically 0.3–0.5 MFU) and the GPU count yields wall-clock; the planner inverts this to find GPUs needed for a target time. Fine-tuning is a small fraction of this; LoRA smaller still.

```
GPUs ≈ (6 · N · D) / (peak_FLOPs · MFU · target_seconds)
# e.g. N=7e9, D=1e12, B200 dense FP8 throughput, MFU≈0.4, target≈3 days
#      → solve for GPUs; round up to a topology-friendly multiple (8, 16, …)
```

Inference memory. Two terms dominate GPU memory: model weights and the KV cache. Weights ≈ parameters × bytes/param (e.g. 1 byte at FP8). The **KV cache** grows with concurrency and context and often sets the real concurrency ceiling:

```
KV_bytes ≈ 2 · layers · 2 · n_kv_heads · head_dim · seq_len · batch · bytes_per_elem
# the leading 2 = keys+values; second 2 = bytes if FP16 (use 1 for FP8 KV)
```

Throughput (QPS) is then governed by per-request latency under in-flight batching at the chosen precision; the planner sizes replicas to meet peak concurrent demand within the TTFT/inter-token SLO, adding headroom for the availability target. Paged KV and quantization (Ch 9) directly raise achievable concurrency per GPU.

METHOD, NOT GUARANTEE

These first-order formulas size the estate to a defensible starting point. They are refined with vendor performance data, model-specific benchmarks, and measured production telemetry. Always validate a candidate sizing against a representative benchmark on the actual target GPU and precision before committing capital.

15.3 Reference Tier Bill of Materials

Indicative BOMs for the three tiers of §4.5. Quantities are starting points to be tuned by the methodology above and by procurement and facilities realities.

Element	Pilot	Production	Scale
GPU compute	2–4× HGX/DGX B200 (16–32 B200)	8–16× B200 nodes (64–128 B200)	B200 racks &/or GB200 NVL72
Compute fabric	1 IB leaf pair (Quantum-2/ X800)	Non-blocking leaf-spine	Multi-spine SuperPOD-class
Storage fabric / DPUs	Shared storage leaf; BlueField-3	Dedicated storage network; BF3	Rail-optimized; BF3 at scale
Hot tier (parallel FS)	~100–250 TB NVMe	~0.5–1 PB NVMe	Multi-PB NVMe
Warm tier (object)	~0.5 PB	1–3 PB	10 PB+
Cooling	Air (RDHx optional)	Air + rear-door HX	Direct liquid + CDU plant
Control plane	3 mgmt nodes	3+ HA mgmt nodes	HA + Mission Control fleet ops
Software	NVAIE; Run:ai or KAI	NVAIE; Run:ai	NVAIE; Run:ai multi-cluster
Indicative concurrency	R&D + 1–2 tenants	Many tenants, steady inference	Frontier train + large-scale serve

Each tier is a coherent unit that can be deployed and then grown into the next; the architecture does not change between tiers, only its scale (§3.1, pool-then-partition). Detailed part numbers are fixed with NVIDIA and the chosen OEM/storage partners at procurement time against the then-current Blackwell/Blackwell-Ultra lineup.

CHAPTER 16

Deployment Roadmap

The factory is delivered in five phases, each building on the last and gated by explicit exit criteria. Phasing de-risks a large capital programme: value is proven early on small capacity before frontier hardware and full multi-tenancy are committed.

16.1 Phased Delivery

Phase	Theme	Scope & exit criteria
1	Foundation	Facility, Pilot hardware, fabric, host OS, Kubernetes, GPU/Network Operators, NVAIE entitlement & local DLS, registries, GitOps. Exit: GPU nodes schedulable; air-gapped bootstrap reproducible; observability baseline live (Ch 5).
2	Orchestration	Run:ai/KAI scheduler, queues, fair-share, gang scheduling, MIG/fractional, topology awareness, quota & metering. Exit: multi-queue sharing demonstrated; utilization climbing toward target; showback reporting (Ch 6).
3	Inference	NIM Operator, first NIM endpoints, Triton/TensorRT-LLM, model gateway, guardrails, quantization & air-gapped model pipeline. Exit: governed inference meeting latency SLOs; signed models promoted through the gap (Ch 8, Ch 9, Ch 10).
4	Training	NeMo pipeline, distributed training & parallelism, PEFT/alignment, experiment & lineage tracking, checkpoint resiliency. Exit: end-to-end fine-tune → evaluate → serve on platform; training resiliency verified (Ch 7, §14.2).
5	Multi-Tenant GA	Tenant onboarding automation, self-service, per-tenant isolation/guardrails/ metering, DR exercises, scale-out toward Production/Scale tiers. Exit: tenants self-serving within isolation and quota; DR drill passed; SLOs sustained (Ch 11, Ch 14).

Each phase ends with a go/no-go review against its exit criteria and the requirements of Chapter 2. Hardware scale-out is pulled forward only as utilization evidence and tenant demand justify, keeping capital aligned to proven need.

CHAPTER 17

Risk Register

Principal risks and their mitigations. The register is living; it is reviewed at each phase gate and as the threat and technology landscape evolves.

17.1 Principal Risks & Mitigations

ID	Risk	Sev.	Mitigation
R-01	Facility cannot meet GB200 power/cooling, blocking Scale tier.	High	Validate liquid-cooling & power headroom during Pilot; default to air-cooled B200; engage facilities early (§4.4).
R-02	Low GPU utilization undermines the platform's economics.	High	Pooling + fair-share queues, MIG/fractional, showback transparency; track utilization as an SLO (Ch 6, §13.4).
R-03	Air-gap operational burden slows updates and model delivery.	Med	Automated staging → sign → transfer pipeline; GitOps convergence; batched promotion windows (§9.4).
R-04	Cross-tenant data or compute leakage.	High	Namespace + queue + MIG isolation, default-deny network, per-tenant keys, admission policy (§6.5, Ch 12).
R-05	Supply-chain compromise via artifacts crossing the gap.	High	SBOM, scanning, signing, signature-verifying admission; internal-registry-only policy (§12.3).
R-06	Unsafe or off-policy model outputs to tenants.	Med	NeMo Guardrails rails + NemoGuard models at the gateway; alignment in training (Ch 10).
R-07	Quantization degrades accuracy below tenant needs.	Med	Mandatory evaluation gate before promotion; FP8 default; portable fallback (§9.1).
R-08	Hardware faults waste long training runs.	Med	Checkpointing, auto-restart, node cordon/drain, predictive maintenance (§14.2, §13.3).
R-09	Vendor/version churn (Blackwell→Ultra, Run:ai, NIM) erodes the design.	Med	Standard interfaces (K8s, OpenAI API), declarative config, NVAIE support; tier-agnostic architecture (§3.1).
R-10	Site-level disaster with no cloud failover.	Med	On-prem DR site, mirrored registries/Git/data, portable-tier serving, exercised RTO/RPO (§14.4).
R-11	Skills/operational maturity gap for a complex GPU platform.	Med	Phased rollout, runbooks, Mission Control fleet ops, NVIDIA support & enablement (Ch 16).

CHAPTER 18

Appendices

A Acronyms & Glossary

Term	Definition
AWQ	Activation-aware Weight Quantization — a 4-bit weight quantization method with strong accuracy retention.
BlueField-3	NVIDIA DPU offloading networking, storage, and security from the host CPU.
CDI	Container Device Interface — standard for exposing devices (GPUs) to containers.
CDU	Coolant Distribution Unit — supplies and manages liquid cooling to racks.
Colang	The modeling language used by NeMo Guardrails to define rails and flows.
DCGM	Data Center GPU Manager — NVIDIA's GPU telemetry and health stack.
DGX / HGX	NVIDIA's integrated GPU system (DGX) and OEM baseboard platform (HGX).
DLS	Delegated License Service — local NVIDIA license server for disconnected sites.
DPO	Direct Preference Optimization — an alignment method.
DRA	Dynamic Resource Allocation — Kubernetes API (GA 1.34) for structured device requests.
Dynamo	NVIDIA datacenter-scale inference orchestration for disaggregated serving.
FP8 / FP4	8- and 4-bit floating-point formats; Blackwell-native low-precision compute.
GB200 NVL72	Rack-scale system: 36 Grace CPUs + 72 Blackwell GPUs in one NVLink domain, liquid-cooled.
GGUF	Single-file portable model format from the llama.cpp ecosystem.
GPTQ	A post-training 4-bit weight quantization method.
Grove	Multi-component inference scheduling on Kubernetes (part of Dynamo).
KAI Scheduler	NVIDIA's open-source (Apache 2.0) GPU scheduling engine; core of Run:ai.
KV cache	Key/Value attention cache; a primary driver of inference GPU memory.
LoRA / QLoRA	Low-Rank Adaptation — parameter-efficient fine-tuning; QLoRA adds quantization.
MFU	Model FLOPs Utilization — fraction of peak FLOPs realized in training.
MIG	Multi-Instance GPU — hardware partitioning of one GPU into isolated instances.
NeMo	NVIDIA's framework for training, customizing, and aligning models.
NIM	NVIDIA Inference Microservice — containerized, OpenAI-compatible model endpoint.
NVAIE	NVIDIA AI Enterprise — the supported, secure software suite underpinning the platform.
NVLink / NVSwitch	High-bandwidth GPU-to-GPU interconnect and switch fabric.
PEFT	Parameter-Efficient Fine-Tuning (e.g. LoRA, p-tuning).
RAG	Retrieval-Augmented Generation — grounding generation in retrieved context.

Term	Definition
RDMA / RoCE	Remote Direct Memory Access; RoCE = RDMA over Converged Ethernet.
RLHF	Reinforcement Learning from Human Feedback — an alignment method.
Run:ai	NVIDIA's enterprise GPU orchestration platform layered on KAI.
SBOM	Software Bill of Materials — inventory of components for supply-chain assurance.
SHARP	Scalable Hierarchical Aggregation and Reduction Protocol — in-network collective reduction.
TP / PP / DP / EP / CP	Tensor / Pipeline / Data / Expert / Context parallelism strategies.
TTFT	Time To First Token — a key inference latency metric.
TensorRT-LLM	NVIDIA's compiled high-performance LLM inference backend.
Triton	NVIDIA Triton Inference Server — multi-framework production serving runtime.
Trtllm_buildable	NIM profile permitting on-site compilation of a TensorRT-LLM engine.
vLLM	Open-source inference engine with PagedAttention; portable serving path.
XID	NVIDIA GPU error identifier reported via the driver/DCGM.

B Reference Configuration Index

Illustrative configurations included in this document. These are reference snippets to convey intent and structure; production manifests are maintained in the GitOps repository (§5.5) and pinned to validated versions.

Section	Artifact	Purpose
§6.2	Run:ai `Queue` (scheduling.run.ai/v2)	Tenant fair-share queue with quota, limit, weight.
§7.5	`PyTorchJob` (kubeflow.org/v1)	Gang-scheduled multi-node distributed training.
§8.2	Triton `config.pbtxt`	TensorRT-LLM model with dynamic batching & streaming.
§8.5	`NIMCache` + `NIMService` (apps.nvidia.com)	Air-gapped, autoscaling inference endpoint.
§10.3	Guardrails `config.yml` + Colang	Per-tenant input/output rails bound to guard NIMs.
§12.3	Kyverno `ClusterPolicy`	Internal-registry allowlist + image signature verification.
§15.2	Sizing formulas	Training-GPU and KV-cache memory estimation.

C References

This specification is grounded in NVIDIA's published platform documentation and reference architectures, current as of Q2 2026. Primary references for implementation:

- NVIDIA AI Enterprise — product documentation, support matrix, and air-gapped deployment guidance.
- NVIDIA NIM — microservice documentation, NIM Operator, and offline/air-gap caching.
- NVIDIA Triton Inference Server & TensorRT-LLM — serving and optimization guides.
- NVIDIA NeMo & NeMo Guardrails — framework, Megatron-Core, Aligner, and Colang documentation.

- Run:ai & KAI Scheduler — orchestration documentation and the open-source KAI project (CNCF Sandbox).
- NVIDIA Blackwell / DGX B200 / GB200 NVL72 & networking (Quantum InfiniBand, Spectrum-X, BlueField) — reference architectures.
- NVIDIA GPU Operator, Network Operator, and Mission Control / Base Command — platform operations.
- Kubernetes Dynamic Resource Allocation (DRA) — upstream documentation (GA in 1.34).

> **VERSIONING NOTE**
>
> NVIDIA's Blackwell-generation hardware and AI software evolve rapidly (e.g. Blackwell Ultra/GB300, NIM profile and licensing changes through 2026). Validate exact versions, support matrices, and part numbers against current NVIDIA documentation and your NVAIE entitlement at implementation time. The architecture in this document is deliberately tier- and version-agnostic (§3.1) so it remains valid as components advance.

End of specification · On-Premises NVIDIA AI Factory · v1.0 · Confidential — Draft for Review