ENKIDU

€24 MIOVSKI GROUP €24

CONCEPTUAL ARCHITECTURE

Nusku

The control plane for distributed intelligence roll out models, monitor health, and respond to events across thousands of nodes, streaming only telemetry and signal upstream, never raw data. Structured by Anshar, governed by Kittu.

English · Complete edition

Version 0.1 · First Draft

4 July 2026

Prepared by MIOVSKI GROUP

Draft for Internal Review

Glossary

Terms used throughout this conceptual architecture. Code names are shown in bold. Terms defined in the Enlil, Anshar, and Trust Assurance documents are carried forward and not redefined except where Nusku extends them.

Nusku

The Enkidu control plane for distributed intelligence: the framework that enrols, configures, updates, observes, and remediates every compute element of a distributed AI fabric from the smallest edge node to the central fusion tier without ever carrying mission data.

Control plane / data plane separation

The load-bearing distinction in Nusku. The data plane is the mission fabric sensing, inference, fusion, and the operator picture, structured by Anshar and governed by Kittu. The control plane is Nusku: enrolment, desired state, artifact distribution, telemetry, and remediation. The two share the physical bearers but never share content: mission data never enters a Nusku channel, and Nusku never adjudicates a mission result.

Node twin

Nusku's authoritative record of a managed node: its identity, hardware tier, installed artifact versions, bound policies, current health, and desired state. The twin is what the fleet operator sees; the node is what exists in the field.

Desired state

The declared target configuration of a node or a fleet segment artifact versions, model roster, policy bundle, telemetry profile held centrally and signed. Nodes pull toward desired state and reconcile whenever connectivity permits; Nusku never assumes it can push.

Artifact

Any versioned, signed unit Nusku distributes: a model (with its adapter and quantization), a container image, a Anshar zone definition, an Kittu policy bundle, an ingestion connector, or a base-system (BSP/OS) image. Every artifact carries provenance and enters Kittu's chain of custody before it may be admitted.

Rollout wave

A staged deployment of an artifact across a fleet segment: canary node(s) -> hub cluster -> theatre-wide, with health gates between waves and automatic rollback on gate failure.

Telemetry envelope

The bounded, prioritized budget of control-plane traffic a node may send upstream per reporting interval, sized per tier and per bearer. Signal extraction and downsampling at the edge keep the fleet observable within the envelope.

Nusku Agent

The small-footprint control-plane process on every managed node: identity, reconciliation loop, artifact fetch and verification, local telemetry collection, and the local remediation executor.

Nusku Warden

The hub-tier control-plane service: a local sub-control-plane that caches desired state and artifacts for its cluster of nodes, aggregates and downsamples their telemetry, evaluates alert rules locally, and sustains fleet operations during central-link loss.

Nusku Core

The central-tier control plane: the fleet registry and node twins, the desired-state store, the rollout orchestrator, the telemetry lake, the event engine, and the fleet operator console.

Signal

The product of edge-side reduction: health scores, model-performance indicators, alert events, and summary statistics as opposed to the raw sensor data or inference payloads from which they derive. Nusku carries signal; the data plane carries data.

Abstract

A distributed intelligence fabric of the Enlil class succeeds or fails on a problem that is neither sensing nor inference: operating the fleet. Hundreds to thousands of edge nodes carried by teams, mounted on vehicles, embedded in autonomous platforms, or bolted to poles must be enrolled, configured, kept healthy, updated with new models and policies, and recovered when they fail, across intermittent, contested, and bandwidth-poor links. Doing this by hand does not scale past a handful of nodes; doing it with a conventional cloud device-management service violates the sovereignty, classification, and disconnection constraints these fabrics live under.

Nusku is Enkidu's answer: a sovereign, self-hosted control plane for distributed intelligence, and the third foundational framework of the Enkidu stack. Where Anshar gives a fabric its operational structure (zones, continuums, contractual handoffs) and Kittu gives it runtime trust (policy enforcement, trust-operation zones, chain of custody), Nusku gives it operability: fleet lifecycle, staged model and software rollout, health observability, and automated event response. Its defining discipline is the signal-not-data principle: Nusku streams only telemetry and signal upstream, never raw data, achieving fleet-scale visibility without fleet-scale bandwidth and without ever creating a second path by which mission data could leave a node.

Nusku follows the premise established across the Enkidu frameworks determinism where it matters, reasoning where it helps and adds its own corollary: declaration where it scales. Fleet state is declared, signed, and reconciled by pull, so a node that has been dark for a day converges to the same state as a node that never lost its link. The architecture deliberately adopts the strongest patterns proven in the NVIDIA edge-AI ecosystem Fleet Command's single-pane fleet orchestration [1][2], Jetson Platform Services' on-device foundation services [5][6], A/B fail-safe over-the-air update practice [8][9], Triton's live model management [11][12], DCGM-class GPU telemetry [14], Mission Control's autonomous recovery at the central tier [3][4], and GitOps pull-based reconciliation at extreme fleet scale [16][17] and recasts them behind a single governed, sovereign control plane that Kittu supervises and Anshar structures.

1. Purpose and Scope

1.1 What this document is

This is a conceptual architecture, not an implementation manual. It defines Nusku's structure, load-bearing decisions, and contracts with the other Enkidu frameworks enough to build against, assess for procurement, and extend without fixing every interface to a specific product or version. Where a concrete technology is named, it is named because it shapes the architecture or because Nusku deliberately adopts its pattern, not because it is the only valid choice.

Nusku is designed as a critical component of every Enkidu solution. The reference deployment described throughout is Enlil the distributed Fusion-in-a-Box platform with its blended mesh and hub-and-spoke fabric because Enlil presents the hardest version of the fleet-operations problem: four compute tiers, contested links, security classification, and autonomous hosts. The same Nusku architecture, with civilian bindings, is the control plane for Lamassu (urban sensing) and for fixed-site deployments such as smart health units. One control plane, many fabrics.

1.2 What Nusku is not

Nusku is not a data platform, not a fusion engine, and not a trust framework. It never ingests sensor telemetry, never carries observations or assessments, never renders a mission verdict, and never widens an agent's authority. Those functions belong to the data plane (Anshar-structured) and the trust plane (Kittu-governed). Nusku's refusal to touch mission data is not a limitation; it is the design.

1.3 The four capabilities Nusku must deliver

- Capability What it requires of the architecture
- Fleet lifecycle at scale Zero-touch enrolment, node twins, declared desired state, and pull-based reconciliation that converges thousands of nodes without per-node manual work (Section 6).
- Safe artifact distribution Signed, staged, health-gated rollouts of models, software, and policy with fail-safe A/B installation and automatic rollback, over constrained and intermittent links (Section 7).
- Fleet-scale observability Health and performance visibility across every tier, within a strict telemetry envelope: signal extraction at the edge, aggregation at the hub, correlation at the centre (Section 8).
- Autonomous event response Detection, classification, and bounded automated remediation of fleet events restart, isolate, roll back, recover with human escalation for anything beyond the runbook (Section 9).

2. Design Premise and Principles

2.1 The core premise

Nusku rests on one sentence: declare centrally, reconcile locally, report by exception. The centre declares what the fleet should be; every node pulls toward that declaration on its own schedule and survives any interruption; and the fleet reports upstream only the signal an operator needs health, deviation, and events never the data the mission produces.

2.2 The load-bearing invariants

Six invariants hold everywhere Nusku runs. They are deliberately parallel to Enlil's invariants, because Nusku must be as disciplined as the fabric it operates.

- # Invariant What it means in Nusku
- 1 Control never carries data No mission payload sensor data, observations, tracks, assessments ever transits a Nusku channel. Nusku telemetry schemas structurally cannot express mission content.
- 2 State is declared and pulled Desired state is signed at the centre; nodes reconcile by pull. Nusku never depends on a push reaching a node, so disconnection is a normal state, not a failure.
- 3 Every artifact is signed and admitted Nusku distributes artifacts; Kittu admits them. An unverified signature, broken provenance link, or policy mismatch stops installation at the node, regardless of what the centre ordered.
- 4 Updates are fail-safe by construction Every mutable install uses an A/B (or equivalent transactional) mechanism with automatic rollback, so a failed update degrades to the previous working state never to a bricked node [8][9][10].
- 5 Remediation is bounded Automated responses are drawn from a signed runbook of pre-approved actions with explicit blast-radius limits. Anything outside the runbook escalates to a human.
- 6 The control plane is itself governed The Nusku Agent, Warden, and Core are agents under Kittu: they hold trust-operation zones, act under deny-by-default policy, and write every action into the chain of custody. The operator of the fleet is as accountable as the fleet.

2.3 Why not a cloud device-management service

The commercial pattern for this problem is a vendor cloud: NVIDIA Fleet Command, for example, is a hybrid-cloud platform for securely deploying, managing, and scaling AI across up to thousands of edge devices from a single cloud-based control plane [1][2]. Nusku adopts much of what that pattern proved one console, one-touch provisioning, over-the-air lifecycle management, layered security, remote diagnostics [1][7] but rejects its deployment model. Enkidu fabrics operate under data-sovereignty, classification (e.g., Protected B in the Canadian context), and disconnected-operations constraints that a third-party cloud control plane cannot satisfy. Nusku is therefore self-hosted at the fabric's own central tier, inherits the fabric's accreditation boundary, and functions with the WAN entirely absent.

3. Position in the Enkidu Stack Anshar, Kittu, Nusku

The Enkidu stack now has three foundational frameworks, each answering a different question about the same fabric:

- Framework Question it answers Plane
- Anshar Where does work happen, and under what boundary conditions? Zones, continuums, entry/exit conditions, contractual handoffs, error zones. Structure
- Kittu May this action happen, and can we prove what happened? Trust-operation zones, deny-by-default policy, separation of duties, chain of custody. Trust
- Nusku Is the fabric itself healthy, current, and recoverable? Enrolment, desired state, rollout, telemetry, remediation. Operations

The three interlock rather than overlap:

Nusku is structured by Anshar. Fleet segments are Anshar zones; a rollout wave is a zone operation with entry conditions (health gates satisfied) and exit conditions (wave converged); a failed rollout enters a designed error zone quarantine, rollback, report rather than an ad-hoc scramble. Nusku's reporting levels map directly onto Anshar's nested reporting: node -> hub cluster -> zone -> fleet-wide program integrity.

Nusku is governed by Kittu. Every Nusku component is an agent with a trust-operation zone. Distributing an artifact is a proposal; Kittu's admission check signature, provenance, policy binding is the commit. Remediation actions are actuator authority in Kittu's terms: the most consequential automated action Nusku may take unaided is to reduce capability (isolate, roll back, restart); any action that widens capability, or that reaches beyond a node's own boundary, requires the Kittu verdict path and, where policy demands, a human.

Kittu and Anshar are operated by Nusku. Policy bundles and zone definitions are themselves versioned artifacts; Nusku is how a new Kittu policy or a revised Anshar zoning reaches ten thousand nodes safely staged, health-gated, and reversible.

This mirrors the separation-of-duties principle at the heart of the Trust Assurance Framework: no single component may both decide and grant. Nusku observes and executes; Kittu judges; Anshar bounds. A compromised control plane can, at worst, propose it cannot self-admit an artifact or widen an agent's authority.

4. Reference Topology Nusku on the Enlil Fabric

Enlil's fabric has four tiers: FIB nodes (Jetson Orin family Orin Nano, Orin NX, AGX Orin) carried by teams, vehicles, and autonomous platforms; FIB hubs (Jetson Thor T5000) acting as higher-inference fusion centres for node clusters; the FOB-IFC forward fusion centre (DGX Station GB300-class); and the rear Central IFC (GB200/GB300 NVL72-class data centre). Nodes exchange observations peer-to-peer over a self-healing mesh, cluster around hubs in a distributed hub-and-spoke overlay, and hubs reach back through the FOB-IFC to the Central IFC.

Nusku places one component at each tier and rides the same bearers as the data plane with its own, strictly smaller traffic class:

- Fabric tier Nusku component Responsibilities
- FIB node (Orin-class) Nusku Agent Cryptographic identity and enrolment; reconciliation loop against cached desired state; artifact fetch, verification, and A/B install; local telemetry collection (tegrastats-class board metrics, model and pipeline metrics); local runbook execution.
- FIB hub (Thor-class) Nusku Warden Sub-control-plane for its node cluster: desired-state and artifact cache (nodes never fetch past their hub); telemetry aggregation, downsampling, and local alert-rule evaluation; wave coordination within the cluster; out-of-band recovery hooks for its nodes.
- FOB-IFC (GB300 Station-class) Nusku Core (forward) Theatre-scope registry and twins; rollout orchestration across dozens of hubs; theatre telemetry lake and event engine; the forward fleet console; reconciliation upward to the central Core.
- Central IFC (NVL72-class) Nusku Core (central) Fleet-of-fleets registry; the authoritative desired-state store and artifact registry (co-located with the model-production pipeline); long-horizon telemetry and trend analysis; integration with data-centre infrastructure management (Section 12.4).

Two topology properties matter most. First, the hub is the unit of autonomy: a Warden holds everything its cluster needs desired state, artifact cache, alert rules so a cluster cut off from the FOB-IFC continues to reconcile, monitor, and remediate locally, exactly as the data plane continues to fuse locally. Second, distribution follows the fabric: a model artifact travels Central IFC -> FOB-IFC -> hub once, and fans out to nodes over the local mesh, so a 2 GB model update to a 40-node cluster costs the reachback link one transfer, not forty. Peer-assisted distribution across the mesh is a Phase 2 optimization (Section 14).

Figure 1 (to be produced in detailed design): the Nusku overlay on the Enlil fabric Agents on nodes, Wardens on hubs, forward and central Cores, with the control-plane traffic class shown strictly thinner than the data plane at every link.

5. The Three Planes of Nusku

Internally, Nusku decomposes into three cooperating planes, deliberately echoing the plane structure of the fabrics it manages:

1. State plane the node twins, desired-state store, and artifact registry. Deterministic, versioned, signed. This plane is the single source of truth for "what the fleet should be" and "what the fleet is."

2. Flow plane reconciliation, distribution, and telemetry transport. Pull-based, resumable, priority-classed, envelope-bounded. This plane is engineered for the worst link in the fleet, not the best.

3. Response plane the event engine, alert rules, runbooks, and escalation paths. Bounded automation with a designed human seam.

A fourth concern governance is intentionally not a Nusku plane: it is Kittu, wrapping all three, exactly as the trust plane wraps the data plane in Enlil and Lamassu.

6. Fleet Lifecycle Management

6.1 Enrolment: zero-touch, identity-first

A node's life in Nusku begins before deployment: at provisioning, the device receives its cryptographic identity (hardware-rooted where the tier supports it), its tier profile, and a baseline desired-state cache. In the field, first boot follows the pattern Enlil and Lamassu already commit to present identity, join the mesh, register through a zero-touch enrolment exchange when a Warden or Core is reachable and the pattern the commercial ecosystem has validated: Fleet Command-style one-touch provisioning that turns a system into a managed AI appliance without a technician at the site [1][7]. Because desired state and baseline models travel on the device, a node is useful before it is connected and managed the moment it is.

Enrolment creates the node twin. From then on, every reconciliation, install, health report, and remediation updates the twin, and the twin not ad-hoc queries to the field is what operators and the event engine reason over.

6.2 Desired state and reconciliation

Nusku manages the fleet the way large Kubernetes edge estates are managed: a declared, version-controlled desired state that lightweight agents pull and converge to. The GitOps pattern Git-style signed declarations as the source of truth, pull-based sync tolerant of dynamic addresses and unreliable connectivity, demonstrated at scales up to hundreds of thousands of clusters with minimal agent overhead [16][17] is adopted directly, because pull-based reconciliation is the only model that survives Enlil's deployment profiles (including infrastructure-denied MANET operation). On the hub and central tiers, where a container orchestrator is warranted, Nusku standardizes on a lightweight certified Kubernetes distribution of the K3s class a single small binary with ARM64 support, designed for unattended, resource-constrained, remote operation [16][18] while the smallest node tiers run the Agent against a plain container runtime, following the Jetson Platform Services model of docker-compose-deployed containerized service bundles [5][6].

Reconciliation is idempotent and resumable: a node that was dark for a week computes the diff between its actual and desired state and converges pulling only missing artifacts, from its hub's cache with no special "catch-up" path.

6.3 Decommissioning and loss

Nodes leave the fleet in one of three ways, each designed: graceful retirement (keys revoked, twin archived, artifacts scrubbed), suspected compromise (Kittu drops the node to quarantine; Nusku revokes its enrolment and instructs the mesh to shun it; its evidence is preserved), and physical loss (the twin's last-known state feeds the loss report; at-rest encryption and Kittu's key custody bound what a captured node can disclose). A fleet that plans only for adding nodes is half a fleet.

7. Artifact and Model Distribution

7.1 What Nusku distributes

Everything a node runs or obeys is an artifact: fine-tuned, quantized specialist models and their adapters (the LoRA/QLoRA products of the Enlil training discipline); container images for agents and services; Anshar zone and continuum definitions; Kittu policy bundles; ingestion connectors; and base-system images (JetPack/BSP-class). The artifact registry at the central Core is the sovereign analogue of the NGC-pattern private registry: a signed catalogue from which every deployable object and nothing else may be drawn [1][13][19].

The model-production pipeline feeds it directly: the TAO-pattern workflow select a pretrained base, fine-tune on mission data, distill and quantize, export in a deployable open format, validate against the gold-standard evaluation set terminates by publishing into the Nusku registry, with training provenance attached [13][19][20]. A model that cannot show its provenance cannot enter the registry; a model not in the registry cannot reach a node. This closes the loop the Trust Assurance Framework demands: models are policy encodings, versioned and governed like policy changes.

7.2 Staged rollout: waves, gates, rollback

No artifact reaches the whole fleet at once. A rollout declares its waves typically one canary node per hub cluster, then the cluster, then the theatre and the health gates between them: inference latency and accuracy indicators within bounds on the canary, no elevated error-zone entries, no Kittu trust-zone demotions attributable to the new artifact. A gate failure freezes the wave and triggers automatic rollback of already-updated nodes. Rollouts are Anshar operations: the wave's entry stake is the previous gate's pass, its error zone is the quarantine-and-rollback path, and its exit condition is fleet convergence measured, not assumed.

7.3 Fail-safe installation

At the node, installation is transactional. For system-level artifacts, Nusku mandates the full A/B partition strategy proven on the Jetson platform: the update installs to the inactive slot, the node boots into it, and only a successful health check commits so a failed or interrupted update can never produce a half-installed or bricked device [8][9]. For models, Nusku exploits the serving layer's native capability: Triton-class inference servers expose a model-control API through which models are loaded, unloaded, and replaced live, with new versions loaded in the background and swapped without loss of availability or application restart [11][12]. The Agent drives that API in explicit control mode dynamic repository access is a known arbitrary-code-execution surface, so model loading is restricted to the verified Agent path and never left open to polling by untrusted parties [12]. Delta transfer keeps constrained links viable: adapters and quantized weights ship as diffs against the base the node already holds.

7.4 Recovery beyond software

Hubs additionally carry out-of-band recovery capability for their nodes, following the pattern established by Jetson fleet-management practice: hardware-level power cycling and recovery-mode reflash that can revive a node whose operating system is unresponsive, without a site visit [9][10]. On the Enlil fabric this is a Warden function exposed over the local cluster's management links; its invocation is a runbook action under Kittu policy.

8. Telemetry, Health, and the Signal-Not-Data Principle

8.1 The principle

Nusku's observability is built backwards from its bandwidth constraint and its trust constraint, which happen to agree: the edge reduces; the fabric forwards signal; raw data never leaves the node on a control channel. This is what "fleet-scale visibility without fleet-scale bandwidth" means in practice — and it is simultaneously the strongest possible data-leak posture for the control plane, because Nusku telemetry schemas have no field in which a video frame, an RF capture, or an assessment could travel. The schemas structurally enforce Invariant 1.

8.2 What is measured, per tier

Collection follows the proven local pattern at every tier, then diverges from convention in what goes upstream.

On the node, the Agent runs the Jetson Platform Services-style monitoring stack in miniature: exporters exposing board metrics (the tegrastats utility's memory and processor reporting on Jetson-class devices), pipeline metrics (stream FPS, drop rates), and inference metrics (Triton's built-in utilization, throughput, and latency endpoints), scraped by a local Prometheus-pattern collector with local alert-rule evaluation [5][6][11]. On GPU-server tiers — FOB-IFC and Central IFC — the equivalent substrate is DCGM: NVIDIA's data-centre GPU telemetry suite, exposed through the DCGM-exporter's Prometheus endpoint and deployable per-node or as a daemonset [14][15].

Model health is a first-class metric class, not an afterthought: per-model confidence distributions, abstention rates, disagreement with the critique path, and drift indicators — the operational feed for Kittu's drift-monitoring requirement and the trigger for retraining requests back into the TAO-pattern pipeline.

Upstream, the Warden aggregates: full-resolution series stay on the hub with a bounded retention window; the hub sends the FOB-IFC downsampled series, health scores, and exceptions — threshold crossings, alert firings, state changes. The FOB-IFC sends the Central IFC less still. Each link carries an order of magnitude less than the one below it.

8.3 The telemetry envelope

Every node and hub holds a signed telemetry envelope: its per-interval budget, priority classes (events > health scores > downsampled series > logs-on-request), and degraded-mode behaviour (store-and-forward with watermarking when the link is poor; drop lowest class first when the buffer fills). Diagnostics that exceed the envelope — a full log pull, a trace capture — are pull-on-demand operations: an operator requests them, Kittu checks the requester's authority, and the transfer runs as a bounded, audited session, following the access-controlled, time-boxed remote-diagnostics pattern the commercial fleet platforms established [2][7].

8.4 Health as a score, alerts as events

The Warden computes a per-node health score from board, pipeline, and model metrics — one number an operator can rank a thousand nodes by — while the underlying series remain queryable at the tier that holds them. Alert rules run at the lowest tier that has the data to evaluate them (Prometheus/Alertmanager-pattern evaluation and routing, as in the Jetson Platform Services monitoring architecture [6]), so a node raises its own alarm even when it is alone in the dark, and the same alarm becomes an event in the fleet console the moment a path upstream exists.

9. Event Response and Automated Remediation

9.1 The event engine

Events alert firings, gate failures, enrolment anomalies, Kittu trust demotions, link-state changes flow into the Response plane's event engine at each Core. Events are classified by severity and blast radius, correlated across the fleet (one bad artifact looks like fifty simultaneous node events; the engine must see one cause), and matched against the runbook.

9.2 Runbooks: bounded autonomy

A runbook entry is a signed, versioned, pre-approved automated response with explicit preconditions and limits: restart service X on the affected node; if it fails twice, roll back to the previous artifact; if that fails, mark degraded and page the operator. The autonomy here is deliberately conservative restart, roll back, isolate, shed load, request recovery and asymmetric in the same direction as Kittu's trust zones: automated actions may reduce a node's activity or capability instantly, but nothing automated ever increases capability or scope. The pattern of self-healing platforms that automatically restart applications and migrate workloads at the onset of problems [2], and of autonomous recovery engines that diagnose and repair known failure modes without human intervention at the data-centre tier [3][4], is adopted inside those bounds.

9.3 The human seam

Everything outside the runbook escalates, with the twin, the correlated event chain, and the relevant telemetry already assembled into the operator's view. The fleet console is a Anshar reporting surface: node health for operations leads, rollout and policy compliance for governance, zone-level fleet status for program directors, cross-zone integrity for command the same four-level reporting discipline the Anshar framework defines, applied to the fleet itself.

10. Security and Governance Nusku Under Kittu

Nusku is the highest-value target in any fabric it manages: it touches every node and distributes executable artifacts. Its security posture is therefore not self-asserted; it is Kittu's, applied to Nusku as to any other agent system.

Identity and transport. Every Nusku link Agent<->Warden, Warden<->Core, Core<->Core is mutually authenticated and encrypted under Kittu's secured-communication regime, on the same footing as the data plane's chain-of-custody links.

Deny-by-default authority. The Agent's policy states exactly what it may install, execute, and report; the Warden's states which nodes it may command; the Core's states which fleet segments an operator role may touch. None can widen its own policy.

Admission over distribution. Nusku moves artifacts; Kittu admits them (Invariant 3). Signature verification and provenance validation run on the node, at the last moment before install so a compromised Warden cache or Core can, at worst, deliver an artifact the node then refuses.

Separation of duties, again. Assessment (telemetry and health), judgment (Kittu's verdict on any consequential action), and execution (the Agent's installer and runbook executor) are separate components that consume each other's signed outputs no Nusku component can both decide a node deserves a change and make it.

Total auditability. Every enrolment, rollout, install, telemetry-envelope change, and remediation is written to the chain of custody. "Who put this model on this node, when, under whose authority, and what did it replace?" is always answerable from the trace the fleet-operations analogue of the Trust Assurance Framework's provenance guarantee.

Classification discipline. Control-plane traffic carries its own data-sensitivity caveats (telemetry about a node's location and activity is itself sensitive), enforced at distribution exactly as the data plane's caveats are.

11. NVIDIA Toolset Alignment

Nusku does not reinvent what the NVIDIA edge-AI ecosystem has already proven; it adopts, hardens, and self-hosts. This section records the alignment explicitly, tier by tier, so detailed design can confirm each element against current releases.

- Concern Ecosystem precedent Nusku leverages Nusku posture
- Fleet orchestration pattern NVIDIA Fleet Command: single control plane for provisioning, deploying, and monitoring AI across thousands of edge systems; Kubernetes/Helm-based containerized deployment; OTA updates; alerts; APIs for integration [1][2][7] Adopt the operational pattern; reject the cloud-hosted deployment model in favour of a sovereign, self-hosted Core inside the fabric's accreditation boundary (Section 2.3).
- On-node foundation services NVIDIA Jetson Platform Services (JetPack 6): modular containerized foundation services monitoring, API gateway, IoT gateway, firewall, system bus with Prometheus/Grafana/Alertmanager-based monitoring and REST APIs, deployed as docker-compose bundles [5][6] The Nusku Agent composes with and manages JPS-class services on Jetson-tier nodes; the JPS monitoring stack is the node-local telemetry substrate.
- Fail-safe OS/BSP updates Mender + NVIDIA collaboration: full A/B system-update strategy for Jetson (L4T integration) eliminating half-installed or bricked devices; fleet-wide provisioning of platform services; monitoring add-ons [8][9] Mandated as the installation invariant for system-level artifacts (Invariant 4, Section 7.3).
- Out-of-band recovery Jetson fleet-management practice (Allxon-class): remote power cycling, hardware-level access to unresponsive systems, remote JetPack recovery-mode reflash; Peridio-class zero-downtime model/firmware rollout with automatic rollback [9][10] Warden-mediated OOB recovery for node clusters; rollback-by-default rollouts (Sections 7.2, 7.4).
- Live model serving and swap Triton Inference Server: framework-agnostic serving across cloud, data centre, edge, and embedded; model repository with versioning; model-control API for live load/unload/replace without service disruption; built-in health endpoints and utilization/throughput/latency metrics; shared-library embedding for the smallest tiers; documented security guidance restricting dynamic model loading to trusted entities [11][12] The Agent drives Triton-class serving in explicit model-control mode as the standard mechanism for model rollout on every tier that serves models (Section 7.3).
- Model production pipeline NVIDIA TAO: fine-tuning of 100+ pretrained vision and foundation models from NGC; distillation and quantization for edge-ready variants; export to open formats; deployment via DeepStream/Triton; jobs on local Docker, Slurm, or Kubernetes [13][19][20] The TAO-pattern pipeline at the Central IFC publishes exclusively into the Nusku artifact registry with training provenance attached (Section 7.1).
- GPU telemetry NVIDIA DCGM and DCGM-exporter: data-centre GPU metrics exposed at a Prometheus endpoint, runnable standalone or as a Kubernetes daemonset, with standard Grafana dashboards [14][15] Standard telemetry substrate at FOB-IFC and Central IFC tiers; the node-tier analogue is tegrastats-class board metrics through the JPS monitoring stack [5][6].
- Central-tier infrastructure management NVIDIA Mission Control for GB200/GB300 NVL72: Base Command Manager for provisioning and cluster administration; Run:ai and Slurm workload orchestration; UFM and NetQ fabric observability; an autonomous recovery engine that diagnoses and repairs known hardware issues and maintains training continuity [3][4] The Central IFC's internal management plane. Nusku Core integrates with it (consumes its health signal, schedules fleet-scope work through its orchestrators) rather than duplicating it: Mission Control runs the data centre; Nusku runs the fleet.
- Fleet-scale reconciliation K3s: certified lightweight Kubernetes (single small binary, ARM64, unattended remote operation) donated to CNCF; Rancher Fleet-class GitOps controllers: pull-based, Git-as-source-of-truth deployment across very large cluster counts with lightweight agents, tolerant of dynamic addressing and unreliable links [16][17][18] The reconciliation model of the Nusku flow plane; K3s-class orchestration at hub and central tiers (Section 6.2).

Two boundary notes. First, Mission Control and Nusku are complementary, not competing: Mission Control is the vendor's management plane inside one rack-scale system; Nusku is the mission's control plane across a heterogeneous, geographically distributed fleet of which that system is one tier. Second, where Kittu already builds on NVIDIA NeMo Guardrails for agent-level rails (per the Enlil architecture), Nusku does not duplicate that layer; it distributes and versions the rails configurations Kittu enforces.

12. Scalability and Failure Behaviour

12.1 Scale model

Nusku's scaling unit is the hub cluster. A Warden serves tens of nodes; a forward Core serves dozens of Wardens; the central Core serves multiple forward Cores. Because reconciliation is pull-based and telemetry is reduced at every tier, the centre's load grows with the number of hubs and exceptions, not the number of nodes the property that lets one console remain legible at ten thousand nodes. The underlying patterns are proven at and beyond this scale: pull-based GitOps controllers manage cluster counts in the hundreds of thousands with lightweight agents [17], and single-console edge platforms operate thousands of distributed locations [1][2].

12.2 Behaviour under loss

- Loss Fleet behaviour
- Node <-> hub link Node reconciles against cached desired state, evaluates its own alerts, buffers signal within its envelope; converges and uploads on reconnect. Mission inference continues on the data plane throughout.
- Hub <-> FOB-IFC link The cluster is autonomous: Warden continues local monitoring, remediation, and (already-staged) rollout waves within the cluster. New fleet-wide rollouts pause at the cluster boundary.
- FOB-IFC <-> Central IFC link Theatre operates on the forward Core. Registry writes queue; nothing already in the field is affected. On reconnect, twins and custody records reconcile upward.
- Nusku Core failure The fleet keeps working this is the decisive test. Nodes and Wardens hold their state; no reconciliation means no change, not no operation. Core recovery restores management, not the mission.
- Bad artifact escapes canary Health gates freeze the wave; automatic rollback restores prior versions; the event engine correlates the incident to one cause; the artifact is quarantined in the registry and the custody trail identifies every node that ran it.

12.3 The design test

Every Nusku mechanism must pass one question: does the mission degrade gracefully if this mechanism is absent? A control plane that the data plane cannot survive without has become a single point of failure wearing a management console. Nusku is built so that its total loss costs the fabric its future updates, visibility, recovery but never its present.

13. Maturity and Forward Path

Nusku delivery aligns with Enlil's phased roadmap:

- Phase Nusku scope
- Phase 1 Agent + Warden on the node/hub tiers; enrolment, twins, desired-state reconciliation; A/B system updates and Triton-mode model swaps; node-local monitoring with hub aggregation; manual-approval rollouts; runbook v1 (restart, roll back, isolate).
- Phase 2 Forward Core at the FOB-IFC; multi-hub wave orchestration with health gates; telemetry envelopes and degraded modes hardened against MANET profiles; peer-assisted artifact distribution over the mesh; drift-triggered retraining requests into the model pipeline.
- Phase 3 Central Core at the Central IFC; registry co-located with the TAO-pattern production pipeline; Mission Control integration for the data-centre tier; fleet-of-fleets federation across theatres; long-horizon trend analytics over the telemetry lake.

Open questions for detailed design, recorded here as commitments to resolve rather than gaps to hide: the concrete artifact-signing and key-custody scheme (with Kittu); telemetry schema freezing and its classification review; the envelope arithmetic per bearer class; and the runbook approval workflow's mapping onto the client's command structure.

14. Conclusion

Anshar made Enkidu fabrics legible. Kittu made them defensible. Nusku makes them operable and at fleet scale, operability is not a convenience but a precondition: a thousand ungoverned, unpatched, unobserved edge nodes are not a capability, they are a liability with antennas.

Nusku's contribution is a control plane disciplined enough to be trusted with the whole fleet: state declared and pulled rather than pushed and hoped; artifacts signed, staged, gated, and reversible; visibility built from signal rather than data, so the fleet can be seen without being leaked; remediation automated only within signed bounds, with the human seam designed in; and the entire plane itself standing under Kittu's judgment and inside Anshar's structure, so the operator of the fleet is held to the same standard as the fleet.

Declare centrally. Reconcile locally. Report by exception. And never let the control plane touch the data.

References

[1] NVIDIA. NVIDIA Fleet Command AI Lifecycle Management Solutions. https://www.nvidia.com/en-us/data-center/products/flee t-command/

[2] NVIDIA. Living on the Edge: New Features for NVIDIA Fleet Command Deliver All-in-One Edge AI Management, Maintenance for Enterprises. NVIDIA Blog. https://blogs.nvidia.com/blog/fleet-command-all-in-one -edge-ai-management/

[3] NVIDIA. NVIDIA Mission Control Software with GB200/GB300 NVL72 Systems Administration Guide (Overview; Software Stack; Release Notes). NVIDIA Docs. https://docs.nvidia.com/mission-control/

[4] NVIDIA. New NVIDIA Software for Blackwell Infrastructure Runs AI Factories at Light Speed. NVIDIA Blog. https://blogs.nvidia.com/blog/mission-control-software/

[5] NVIDIA. Jetson Platform Services Overview. NVIDIA Docs. https://docs.nvidia.com/jetson/jps/moj-overview.html

[6] NVIDIA. Jetson Platform Services Monitoring (Prometheus, Alertmanager, Grafana, exporters, tegrastats). NVIDIA Docs. https://docs.nvidia.com/jetson/jps/platform-services/m onitoring.html

[7] NVIDIA. Fleet Command FAQs. https://www.nvidia.com/en-us/data-center/products/flee t-command/faq/

[8] Mender / Northern.tech. How to Leverage Over-the-Air (OTA) Updates for NVIDIA Jetson Platform Services (A/B system updates, fleet provisioning, monitoring). https://mender.io/blog/how-to-leverage-over-the-air-ot a-updates-for-nvidia-jetson-platform-services

[9] Mender / Northern.tech. How to Use Over-the-Air (OTA) Updates & NVIDIA Jetson Microservices. https://mender.io/blog/how-to-leverage-over-the-air-ot a-updates-with-nvidia-microservices-for-jetson

[10] Allxon. Edge AI Device Fleet Management for NVIDIA Jetson (BSP OTA updates, out-of-band power control, remote JetPack recovery). https://www.allxon.com/jetson ; Peridio. Scalable Fleet Management for NVIDIA Jetson Edge AI. https://www.peridio.com/partners/nvidia

[11] NVIDIA. Triton Inference Server Architecture and User Guide (model repository, model management, metrics, edge shared-library deployment). NVIDIA Docs / GitHub. https://docs.nvidia.com/deeplearning/triton-inference- server/user-guide/docs/index.html ; https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/model_management.htm l

[12] NVIDIA. Simplifying AI Model Deployment at the Edge with NVIDIA Triton Inference Server (live model updates without restart); Triton Deployment Guide (security of dynamic model-repository updates). NVIDIA Technical Blog / GitHub. https://developer.nvidia.com/blog/simplifying-ai-model -deployment-at-the-edge-with-triton-inference-server/ ; https://github.com/triton-inference-server/server/blob /main/docs/customization_guide/deploy.md

[13] NVIDIA. TAO Toolkit Overview (fine-tuning 100+ pretrained models, distillation, quantization, deployment via DeepStream/Triton; jobs on local Docker, Slurm, Kubernetes). NVIDIA Docs. https://docs.nvidia.com/tao/tao-toolkit/latest/text/ov erview.html

[14] NVIDIA. DCGM-Exporter GPU Telemetry for Prometheus. GitHub / NVIDIA Docs. https://github.com/NVIDIA/dcgm-exporter ; https://docs.nvidia.com/datacenter/dcgm/latest/gpu-tel emetry/dcgm-exporter.html

[15] NVIDIA. NVIDIA GPU Telemetry Setting up Prometheus and Grafana with DCGM. NVIDIA Docs. https://docs.nvidia.com/datacenter/cloud-native/gpu-te lemetry/latest/kube-prometheus.html

[16] Rancher / SUSE. K3s Lightweight Kubernetes for Edge, IoT & ARM.

https://www.rancher.com/products/k3s ; https://www.suse.com/products/k3s/

[17] SUSE. Microservices at Edge with K3s and Fleet (GitOps pull-based management designed for very large cluster counts over unreliable links). https://www.suse.com/c/microservices-at-edge-with-k3s- and-fleet/

[18] CNCF / Rancher Labs. K3s announcement and CNCF donation (certified lightweight distribution for resource-constrained, unattended edge environments). https://www.rancher.com/products/k3s

[19] NVIDIA. TAO Toolkit NGC Catalog (containers and pretrained models; deployment to DeepStream and Triton). https://catalog.ngc.nvidia.com/orgs/nvidia/teams/tao/c ontainers/tao-toolkit

[20] NVIDIA. TAO Toolkit Developer Page (distillation and quantization for edge-ready models; DeepStream deployment skills). https://developer.nvidia.com/tao-toolkit

[21] Enkidu Enterprises (2026). Enlil Conceptual Architecture for a Distributed Fusion-in-a-Box Edge Platform, v0.14. (Fabric tiers, deployment profiles, Anshar zoning, Kittu trust harness, model-size ladder, fine-tuning discipline.)

[22] Enkidu Enterprises (2026). Anshar Overview, v1.0. (Zones, continuums, policies, conops, reporting levels, error zones, change management.)

[23] Enkidu Enterprises (2026). Trust Assurance Overview, v1.0. (Assurance cases, provenance and chain of evidence, Trust Harness, graduated trust zones, separation of duties.)

[24] Enkidu Enterprises (2026). Lamassu Conceptual Architecture, A Distributed Urban-Sensing Platform, v2.0. (Civilian edge-node tiers, rapid deployability, adaptive ingestion, policy framework.)