BareStax

No layers. Just compute.

BareStax Technical Overview

Architecture, lifecycle, operations, security, deployment, and licensing for on-premises NVIDIA GPU clusters

Version 2026-11 · barestax.io

Contents

01 One operating model covers the full GPU cluster lifecycle

02 Cluster operations stay consistent from host configuration to job execution

03 Operators and researchers work from one consistent interface

04 Security controls define how the cluster operates

05 Deployment starts from your installed servers

06 Simple licensing for sovereign infrastructure

07 Built for institutions that own their GPUs

08 BareStax keeps its cluster boundaries explicit

01

One operating model covers the full GPU cluster lifecycle

BareStax turns an Ubuntu server into a verified BareStax Core node, then keeps its declared state and health under continuous control. The same architecture handles initial installation, daily operation, expansion, upgrades, recovery, and removal on connected or air-gapped networks.

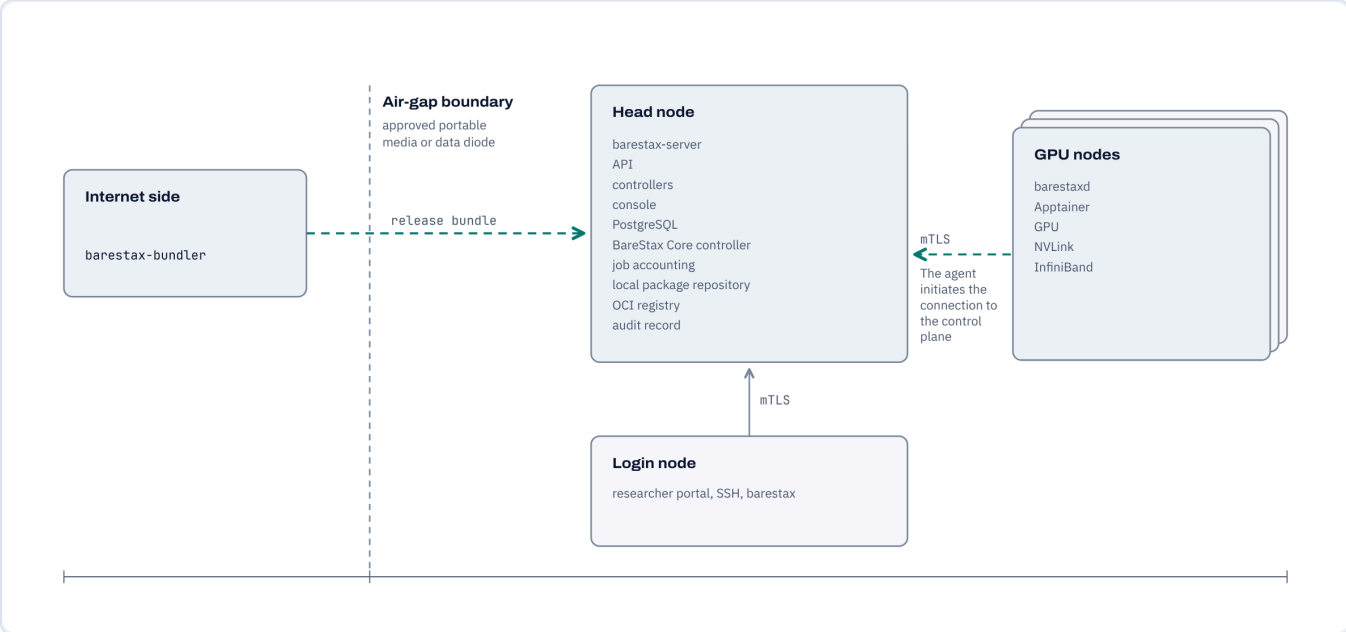
The control plane coordinates every cluster role

The head node runs `barestax-server` (API, controllers, and operator console in one binary), PostgreSQL, the BareStax Core controller, job accounting, the local package repository, the OCI registry, and the audit record. It can be a GPU-free server or a virtual machine on your institution's virtualization platform.

Each GPU node runs `barestaxd` and Apptainer. The agent initiates its connection to the control plane, so the control plane does not push commands into the node. A login node provides SSH, the researcher portal, and job submission.

On the connected side, `barestax-bundler` collects and signs release bundles for transfer into an isolated network. All internal cluster connections use mTLS.

BareStax is written in Go. Each component ships as a single binary: `barestax-server`, `barestaxd`, `barestaxctl`, `barestax`, and `barestax-bundler`. The console lives inside `barestax-server`.



The architecture separates the connected bundler, air-gap boundary, head node, login node, and GPU nodes linked by mTLS.

Day 0 establishes the control plane

The initial command creates the cluster identity and installs the control plane on the head node. You then connect LDAP, Active Directory, local accounts, OIDC, or SAML; configure built-in TOTP; and define network and NFS or S3 storage targets.

BareStax also checks the alert chain before compute nodes enter service. A test alert must reach the on-call path and return its acknowledgment code.

```
barestax-server init --cluster-name <name> --endpoint <fqdn> --key-backup <file>
barestaxctl alert test
barestaxctl doctor
```

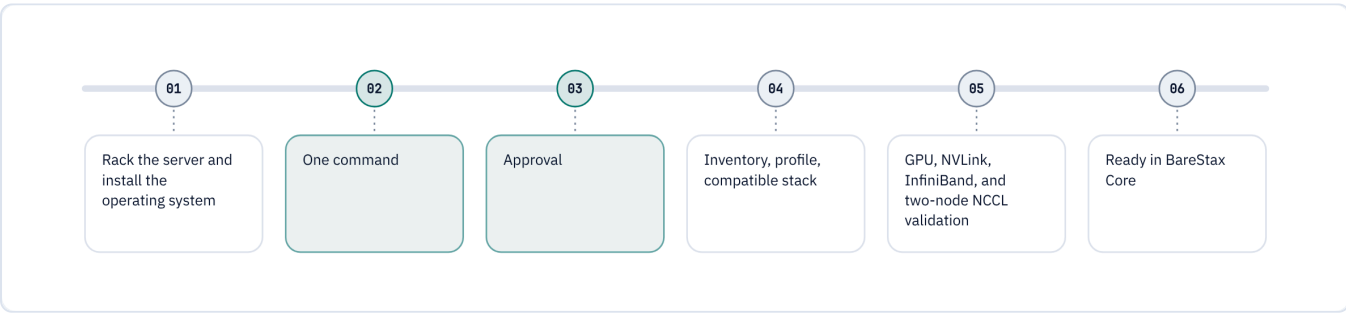
Day 1 turns installed servers into schedulable nodes

Preflight covers the firmware floor, disk, Secure Boot, and connectivity before registration. A join token lets the server register on a provisional basis. After approval, BareStax inventories GPU, NVLink, InfiniBand, PCIe, and NUMA topology and selects the matching profile.

The selected hardware profile installs a tested set of kernel, DOCA-OFED, NVIDIA driver, CUDA user libraries, Fabric Manager, DCGM, NCCL, Apptainer, and system settings. BareStax pins the kernel so unattended operating system updates cannot split the cluster across incompatible states.

GPU, NVLink, and InfiniBand checks plus two-node NCCL validation must pass before the node becomes ready in BareStax Core. A failed check leaves the node outside the work queue with a coded explanation.

```
barestaxctl token create
barestaxd join --token-file <path>
barestaxctl node approve <name>
barestaxctl get nodes -o wide
```



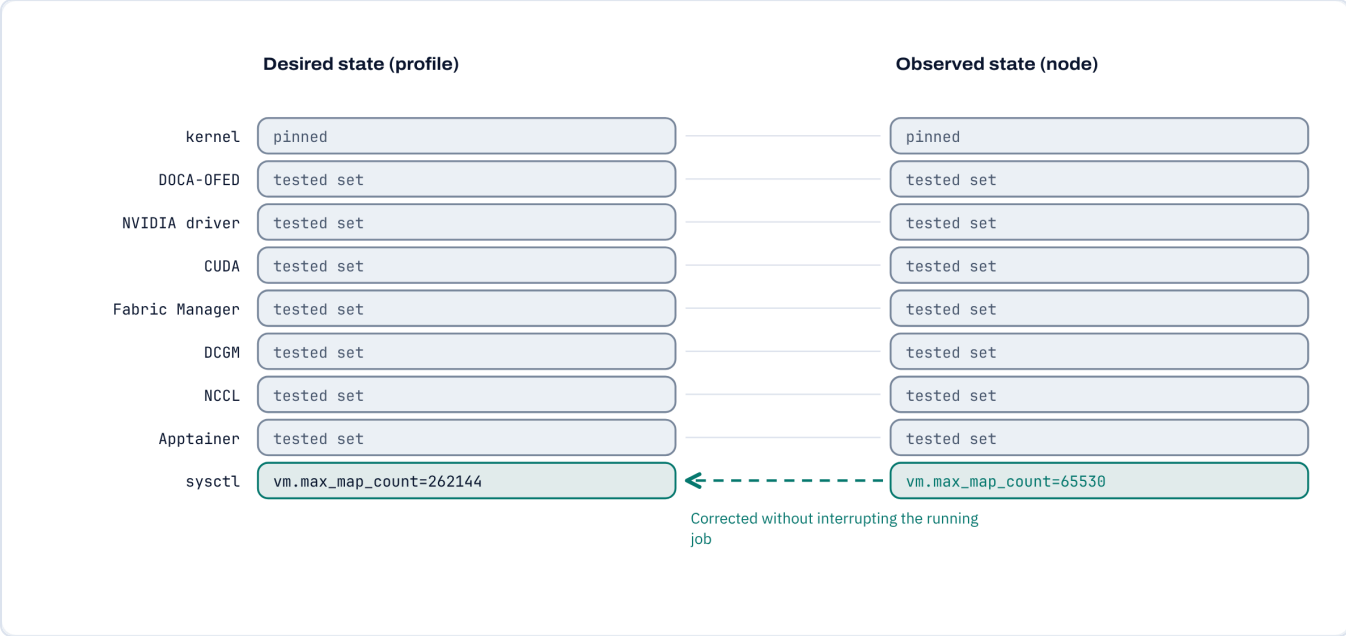
A server moves from operating system installation through one command, approval, profile installation, hardware validation, and BareStax Core readiness.

Day 2 keeps declared state and observed health aligned

BareStax compares each node with its assigned profile. Non-disruptive drift is corrected in service; a disruptive change is reported unchanged and waits for an approved maintenance window.

Health evaluation spans GPU, fabric, host, critical services, and shared storage. Stale telemetry becomes unknown rather than healthy.

A faulty node stops receiving new work while its active job continues. Policy controls staged recovery; a cluster-wide fault halts escalation.



The desired state in the profile and the observed state on the node appear side by side; one different row is corrected without interrupting the running job.

Upgrades respect running work and disruption budgets

Signed release bundles contain compatible component sets and pinned repository snapshots. Running jobs are preserved.

Rollout begins on one canary node, then continues within the configured disruption budget. The control plane follows the same signed path.

The API exposes desired state and observed status

Resources follow a declarative model: your team writes the specification, while BareStax writes status from the cluster. A live watch stream feeds the console and CLI. When the stream breaks, clients mark data as stale.

Plans show the proposed change before application, and dry-run checks it without modifying the cluster. The web console and command line call the same API, so an operation has the same policy and audit path in either interface.

```
barestaxctl plan -f <file>
barestaxctl apply -f <file>
barestaxctl operation list
```

Running jobs outlive a control plane interruption

GPU nodes retain their last known safe state if the control plane becomes unavailable. Local health gates continue to prevent a faulty node from receiving work, while jobs already running remain active.

Backups include identity, PKI, the database, and BareStax Core state. Recovery restores the control plane to clean hardware and prevents the former head node from answering afterward. Operators can inspect node, storage, job, tenant, and service impact during the recovery process.

02

Cluster operations stay consistent from host configuration to job execution

BareStax manages the layers that determine whether a GPU node can run production work: operating system settings, drivers, fabric, containers, job scheduling, health, storage access, and recovery.

The node stack is declared as a tested set

Profile contents

A node profile includes the kernel, DOCA-OFED, NVIDIA driver, CUDA user libraries, Fabric Manager, DCGM, NCCL, Apptainer, sysctl values, hostname, and time synchronization. BareStax distributes components only in combinations tested together.

Hardware comes before profile selection

The node agent inventories GPU, NVLink, InfiniBand, PCIe, and NUMA topology, then BareStax matches that inventory to the appropriate profile.

Preflight checks

Preflight rejects a node before registration if its firmware floor, disk, Secure Boot state, or connectivity fails. GPU, NVLink, and InfiniBand checks plus two-node NCCL validation still have to pass before BareStax Core accepts work.

Drift handling

Manual changes and unexpected package or setting changes appear as drift. Non-disruptive drift is corrected while the node remains in service. A disruptive change, such as a kernel change, is reported without modification and waits for an approved operation in a maintenance window.

BareStax Core configuration follows cluster resources and policy

Central configuration arrives with accounting

BareStax produces and centrally distributes the BareStax Core configuration with job accounting; queues follow GPU classes, with separate CPU queues, tenants, quotas, and fairshare.

CPU follows the GPU allocation

The scheduler assigns the CPU share automatically per GPU. Multi-node training receives the fabric configuration needed to reach the network's actual bandwidth.

Submission paths

Teams can submit directly from the command line, choose a `barestax run` template for batch work, array jobs, multi-node torchrun training, or interactive sessions, or use a form in the researcher portal.

Service and preemptible queues

Preemptible jobs are available, and jobs submitted through `barestax run` templates return safely to the queue after a node fault. Requeue behavior for jobs submitted directly from the command line remains the researcher's responsibility. Long-running model services use dedicated queues, service registration, and protection; the institution exposes them through its own load balancer.

The job owner always sees why a job is waiting

Researchers see the waiting reason and applicable quota while operators see the same queue and accounting state in the console.

Image policy controls what reaches compute nodes

Apptainer is the runtime on GPU nodes

Jobs run through Apptainer without Docker on GPU nodes; the full GPU is assigned directly to the job, without MIG or vGPU.

Image approval

The packaged OCI registry imports and scans an image. Two distinct people approve promotion, then the cluster key signs the approved artifact. GPU nodes reject images that lack an accepted signature.

Your registry can remain in the trust chain

BareStax supports an institution's own registry and counter-signature, while policy still decides which signed images can start on compute nodes.

Caches outlive the read-only image

The approved image remains read-only while package and model caches persist in the researcher's workspace across sessions.

Health policy protects work before it attempts recovery

Health signals

Signals include GPU XID and ECC events, row remapping, temperature, power throttling, NVLink, NVSwitch, InfiniBand, RoCE, PCIe AER, EDAC, clock quality, critical services, and shared storage access.

Stale data becomes an unknown state

When telemetry becomes stale, BareStax reports the state as unknown. It does not carry a former healthy result forward as current evidence.

Fault isolation and manual cordon

A faulty node stops accepting new jobs while its current job finishes. An operator can cordon a node for maintenance or deep diagnosis and return it to work afterward. Flap protection prevents unstable signals from repeatedly moving a node in and out of service.

Recovery escalates under policy

Policy advances from service restart to operating system restart and then power cycle. Each stage can run automatically or wait for approval. A storm brake halts recovery when many nodes show a systemic fault.

`barestaxctl node diagnose <name> --deep`

Operators can run deep GPU diagnosis from one command. The impact view links node or storage faults to affected jobs, tenants, and services, and distinguishes verified impact from possible impact.

```
barestaxctl impact node <name>
barestaxctl node diagnose <name> --deep
barestaxctl doctor
```

Release bundles make isolated operation repeatable

Offline runtime

Cluster services, repositories, licensing, and runbooks stay inside the institution's network. Security covers the external-connection boundary.

The release begins on the connected side

`barestax-bundler` collects required packages and metadata into a signed, content-addressed release bundle. It can export the bundle and verify it before transfer.

Import and promotion

Transfer uses the institution's approved media or data diode. Only a verified bundle reaches promotion; Deployment lists the checks and rollout order.

Repository snapshots preserve the release contents

Upstream repositories are pinned to timestamped snapshots. Their key fingerprints appear in the bundle's security section, so the same release produces the same bytes at each installation.

```
barestax-bundler export
barestax-bundler verify
barestaxctl release import <bundle>
barestax-server upgrade --offline
```

Security controls are part of every operation

Certificates and second factors establish identity

mTLS and the cluster PKI establish node identity. LDAP, Active Directory, local accounts, OIDC, SAML, and built-in TOTP establish user identity. Security carries the certificate lifecycle details.

Six roles separate tenants and approvals

Viewer, operator, approver, administrator, auditor, and tenant administrator duties remain distinct. Approval and tenant boundaries are detailed on the Security page.

Audit, secrets, and packages stay reviewable

Audit stays hash-linked; Security covers export and SIEM. Secrets are stored with envelope encryption and never shown in plain text. Package locks, SBOM output, a license allowlist, and reproducible builds make every software input reviewable.

Monitoring presents current evidence and data age

Exporters use standard Prometheus data

BareStax includes exporters for the head node and compute nodes, along with alert rules and prepared dashboards. It integrates with Prometheus and Alertmanager. An institution's existing Prometheus and Grafana installation can read the same exporter data.

BareStax dashboards

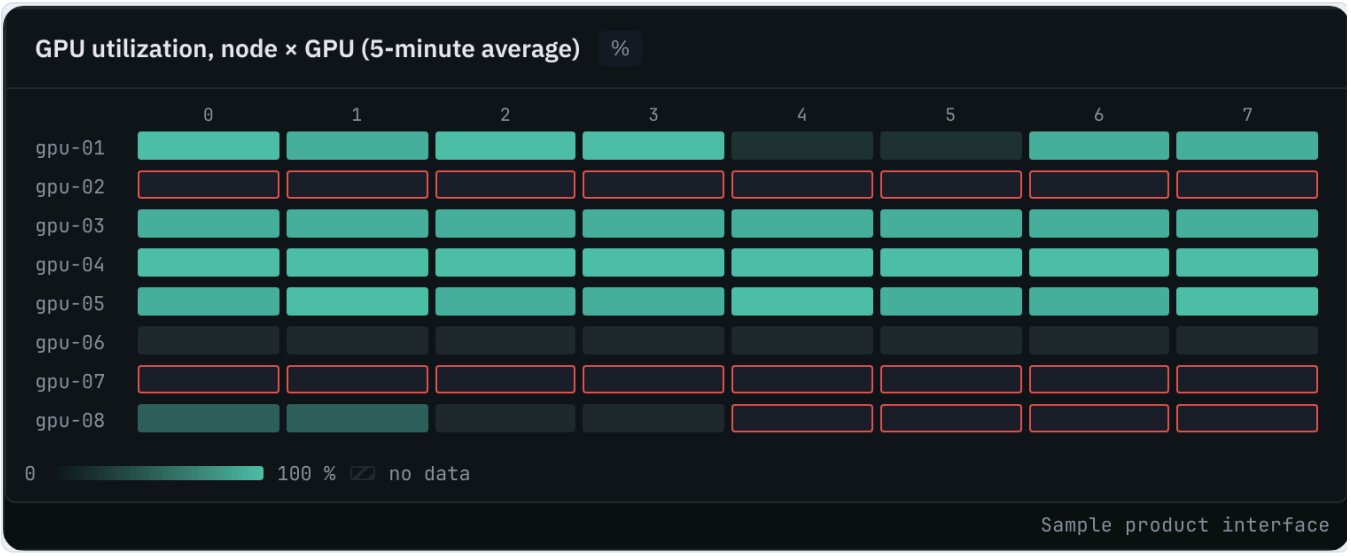
Operators see a node-by-GPU heat map plus utilization, memory, power, temperature, and network charts. The deeper dashboards are BareStax panels that open inside the console under the same authenticated session. They are not Grafana panels.

Alert-chain test

A test alert travels to the on-call recipient, who confirms its code. This verifies delivery rather than assuming that an accepted configuration reaches a person.

Idle GPUs are reported beside waiting work

Reports place idle GPUs beside waiting work, with queue state, quota, and health as context.



A node-by-GPU heat map shows current GPU state with utilization, memory, power, temperature, and network views.

Storage and BMC state enter the same operational model

NFS and S3 clients

BareStax manages the client side of NFS and S3 targets while the institution retains the storage server. Mount protection prevents a missing shared target from causing silent writes to local disk.

Storage faults carry job and tenant impact

The control plane checks storage access and shows which jobs, tenants, and services depend on a failing target. The impact view includes the age and certainty of that information.

Redfish-compatible BMCs

BareStax reads inventory, sensors, event logs, and power state. Power control follows the Security approval policy.

The CLI uses the same API as the console

Operator CLI

`barestaxctl` supports plan and dry-run, cluster health reports, support bundles, alert testing, node lifecycle actions, operations, and approvals.

Researcher CLI

`barestax` submits jobs, reports status and quota, reads logs, and cancels work, against the same state shown in the researcher portal.

Table and JSON output

CLI output is a readable table for people or JSON for automation.

```
barestax run -f job.yaml
barestax status
barestax quota
barestax logs <job>
```

03

Operators and researchers work from one consistent interface

The BareStax console connects cluster inventory, health, impact, operations, approvals, alarms, queues, quotas, image promotion, and runbooks.

The console runs on the head node inside the control plane. The researcher portal opens from the login node and follows the same design. Neither interface needs an external web service.

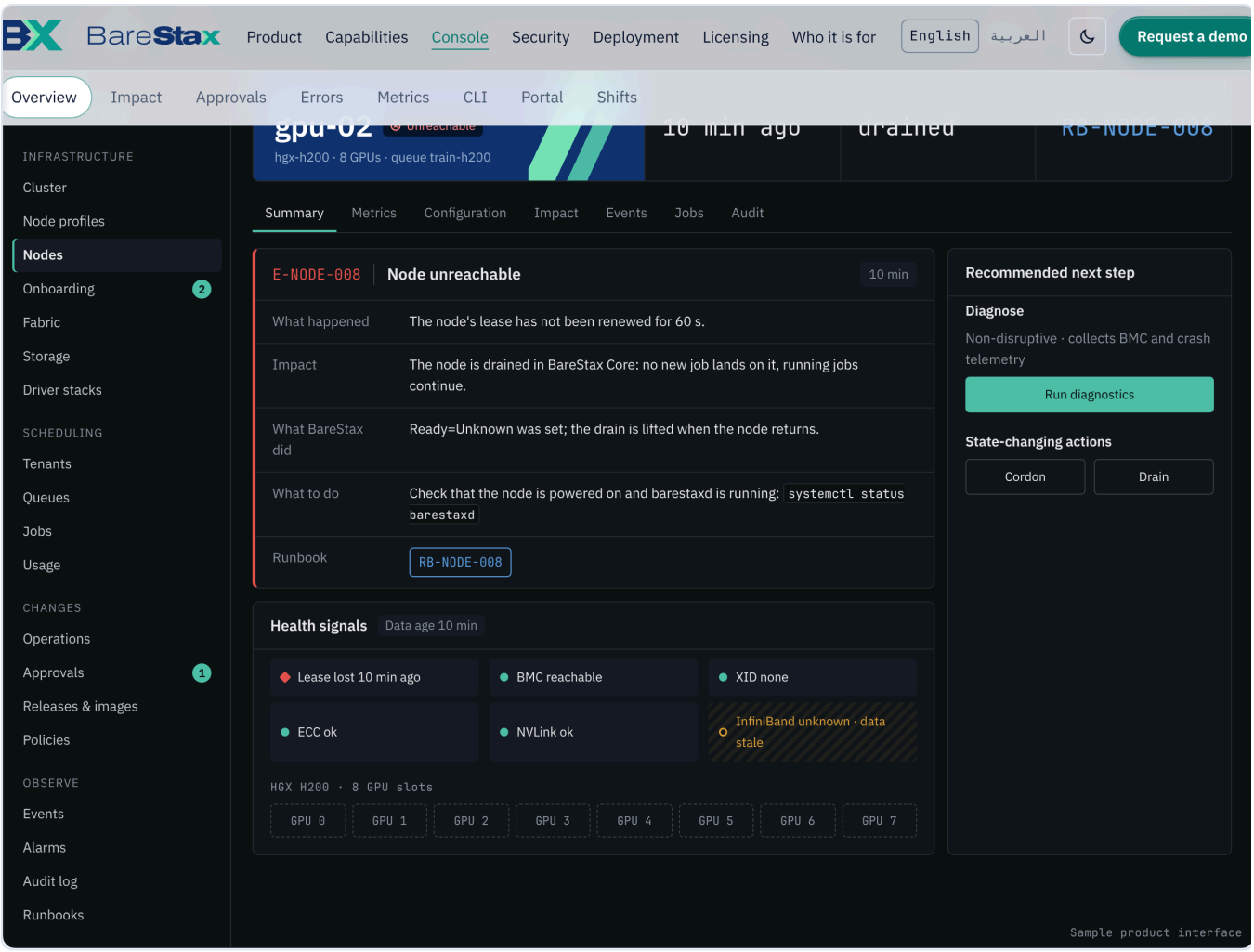
Cluster state comes first

The cluster summary shows node and GPU state, active work, alarms, pending operations, and queue conditions. Inventory connects each node to its hardware topology and assigned profile.

Object pages use the same tabs for Summary, Metrics, Configuration, Impact, Events, Jobs, and Audit. Operators can move from a node or storage target to its current work, recent events, declared settings, and recorded actions without reconstructing context across tools.

Operations and Approvals views list proposed, pending, running, and completed changes with their targets. Alarm, queue, quota, and image promotion views use the same resource context.

Live server changes appear immediately. If the connection to the update stream breaks, the page marks its data as stale.

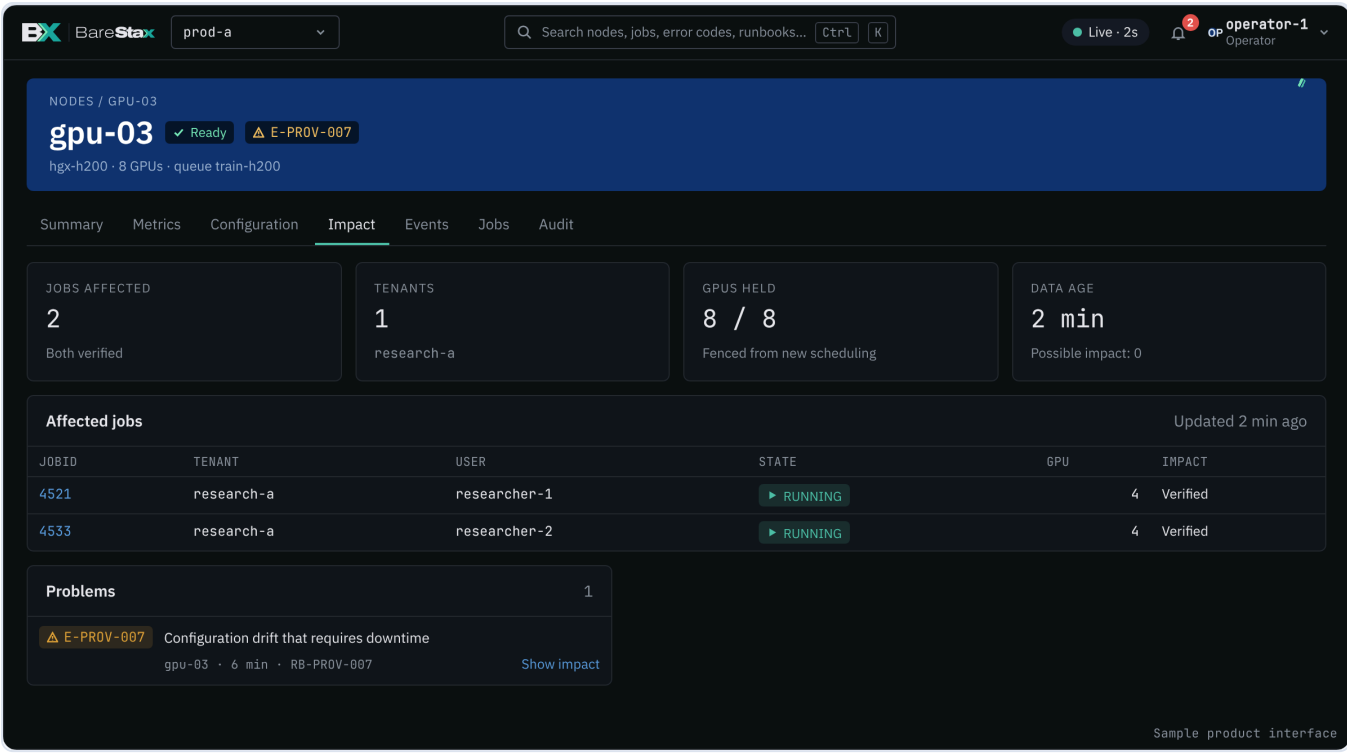


The node page combines summary, metrics, configuration, impact, events, jobs, and audit information.

The impact view connects an alarm to affected work

An alarm identifies a symptom. The impact view connects that symptom to affected jobs, tenants, and services. It separates verified impact from possible impact and displays the age of the underlying data.

From that view, the operator can open the relevant controlled operation. The workflow stays attached to the affected resource and its operational context.

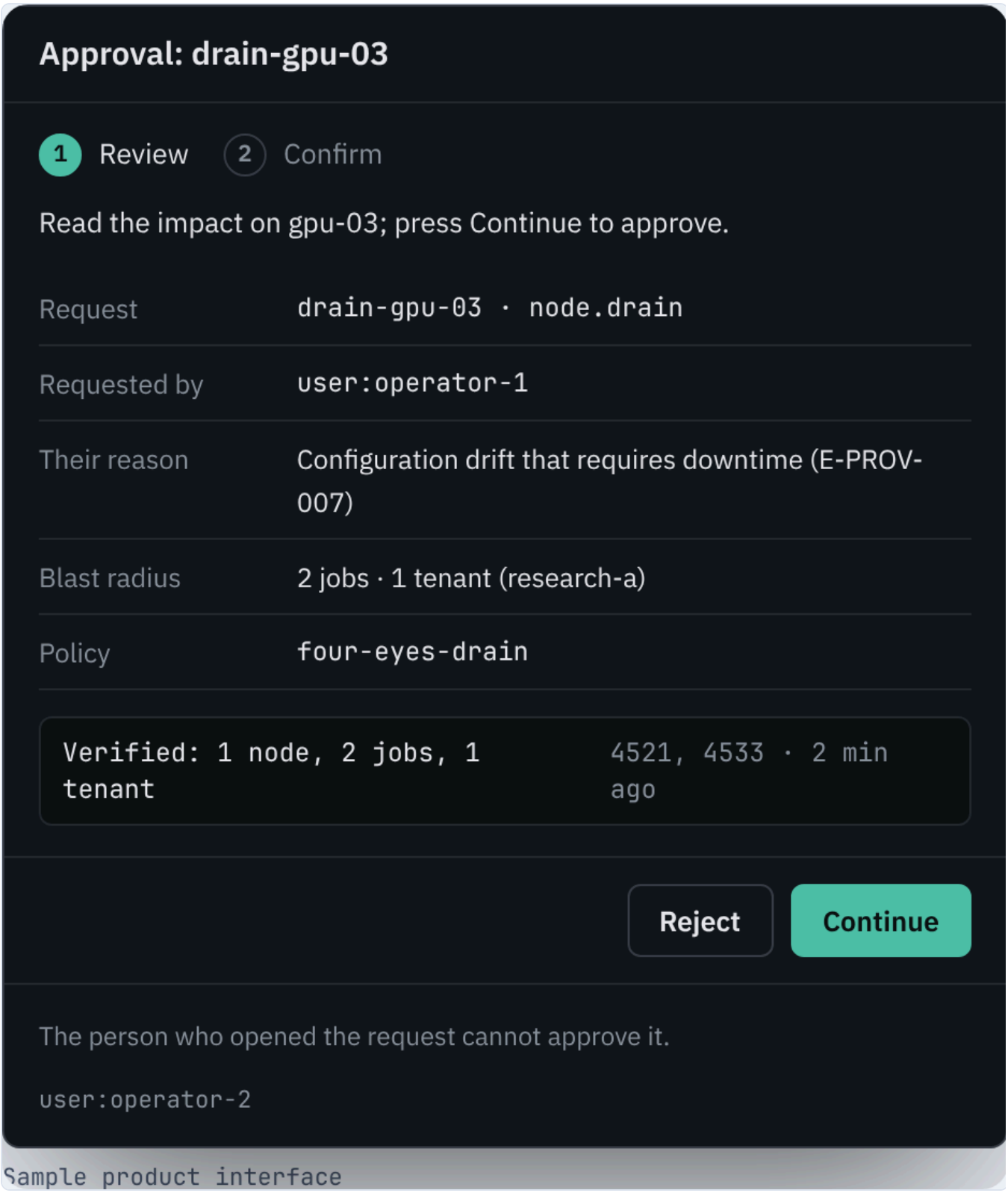


The impact view links a node alert to affected jobs, tenants, services, and an available controlled operation.

Critical actions require independent approval

Node removal, image promotion, restart, and other critical operations use four-eyes approval. The requester cannot approve the same request, and approval belongs to a person rather than a shared account.

Approval has two steps: impact review comes first, followed by the target name. The view shows the requested change, target, requester, and current decision state before execution.



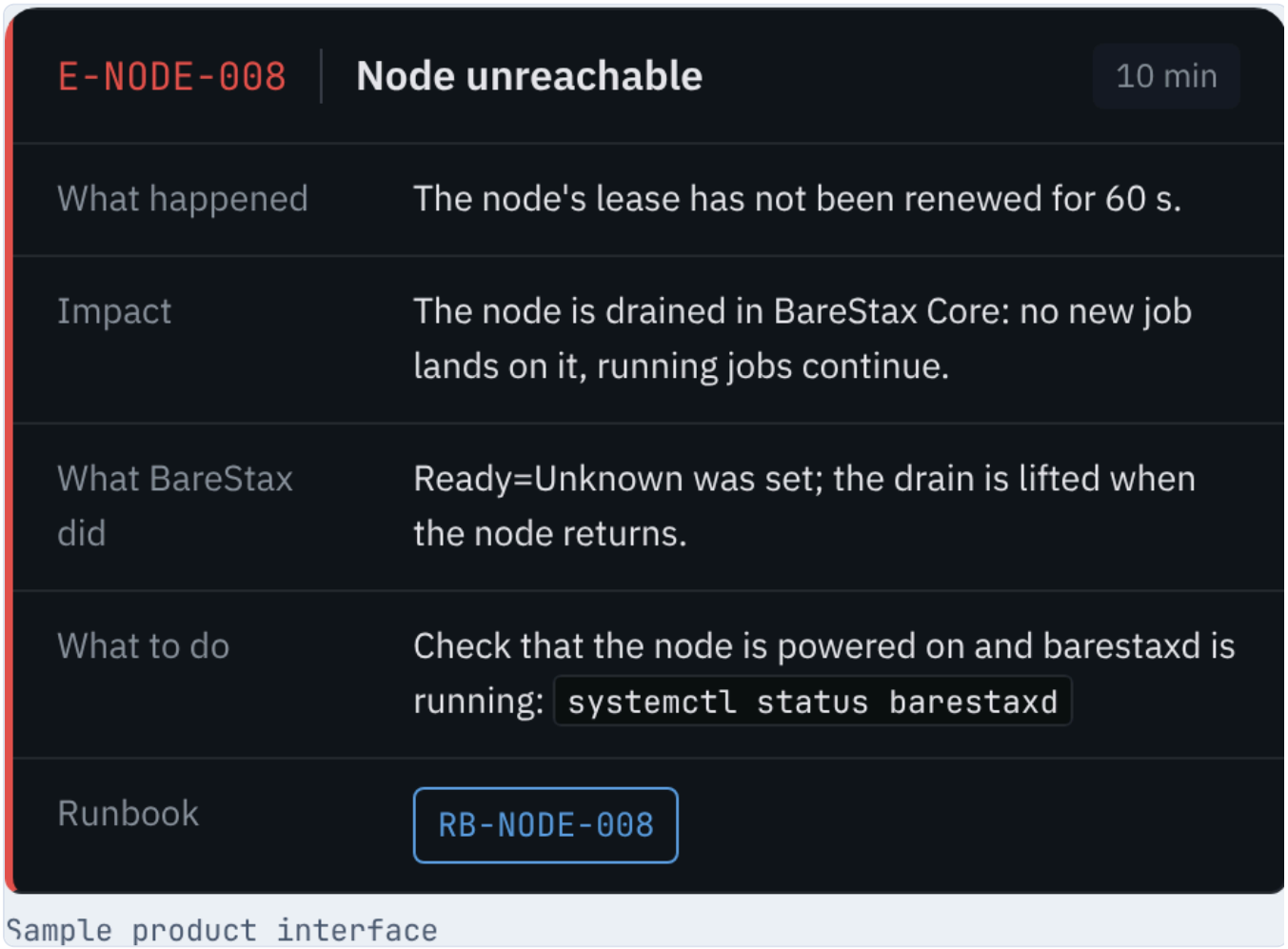
Sample product interface

The approval dialog asks a second authorized person to verify the target and review impact before approving.

Every error states what the operator can do

BareStax gives each error a code. The console explains what happened, the impact, what the product already did, what the operator should do, and which offline runbook applies.

The runbook reader is part of the console and works in an air-gapped network. Operators do not need an external documentation site while handling a fault.



Sample product interface

A coded error card presents the event, impact, product response, operator action, and linked offline runbook.

Cluster metrics stay connected to inventory and work

A node-by-GPU heat map shows state across the cluster. Related views cover GPU utilization, memory, power, temperature, and network behavior.

Deep monitoring dashboards open inside the console under the same authenticated session. Exporters, alert rules, and prepared dashboards include the head node as well as compute nodes. Stale measurements remain visible with their age shown.

Queue and quota state stays beside health data, so idle capacity can be read together with waiting work. Usage reports identify idle GPUs while jobs remain queued.

Every console operation has a CLI path

The console and CLI call the same declarative API. Policy, approval requirements, live status, and audit behavior therefore remain the same when an operator copies the command-line form of an action.

`barestaxctl` serves operators with plans, health reports, node actions, operations, and approvals. Output can be a readable table or JSON for automation.

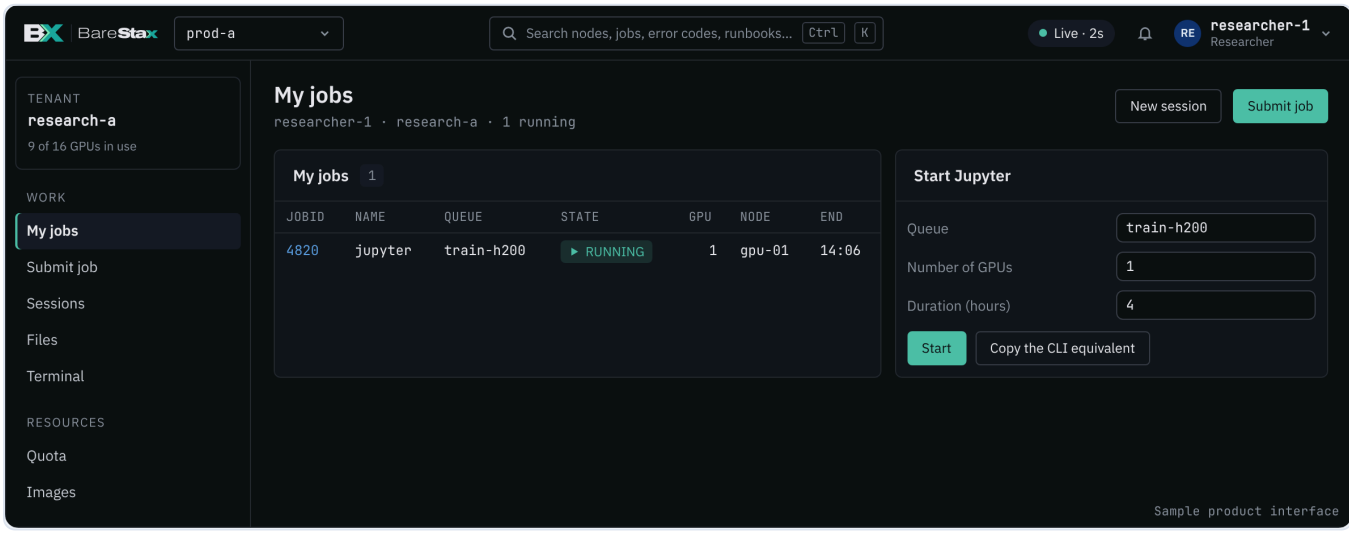
```
barestaxctl impact node <name>
barestaxctl operation list
barestaxctl operation approve <name>
barestaxctl node drain <name>
barestaxctl node remove <name>
```

The researcher portal exposes work without cluster authority

Researchers see their jobs, current state, waiting reason, quota, logs, and cancellation controls. A template form submits batch, array, multi-node training, or interactive work through BareStax Core.

Jupyter, VS Code, RStudio, and remote desktop sessions start as BareStax Core jobs and open in the browser. Only the session owner can enter. An idle session closes and releases its GPU.

The portal also provides a file manager and browser terminal. Actions run under the researcher's own Linux identity, while the persistent workspace keeps packages and environments between sessions.



The researcher portal shows personal jobs, queue reasons, quota, interactive sessions, files, and a terminal.

The interface works in English, by keyboard, in either theme

The product interface is English-only: console, portal, CLI messages, error explanations, and runbooks. Users can choose light or dark theme, operate the interface by keyboard, and open actions through the command palette.

04

Security controls define how the cluster operates

mTLS, cluster PKI, role checks, four-eyes approval, signed images, tenant boundaries, and hash-chained audit records apply through both interfaces on connected and fully air-gapped networks.

Every cluster connection has an authenticated identity

All internal connections use mutual TLS under a PKI owned by the cluster. Node identity is bound to its certificate. BareStax renews certificates and rejects revoked credentials as well as credentials from a former node identity.

The node agent initiates its connection to the control plane; no inbound command channel pushes shell commands into compute nodes.

User identity can come from LDAP, Active Directory, OIDC, SAML, or local accounts. Built-in TOTP remains available in an air-gapped network. BareStax asks for the second factor again when a user attempts a critical operation.

Stored secrets use envelope encryption, while passwords use a modern hash format rather than reversible form.

Six roles separate requests from approvals

Six roles define authority and keep observation, operation, approval, administration, audit, and tenant management separate.

Critical actions use four-eyes approval. Two distinct people participate, and the requester cannot approve the same request. Approval is tied to the person's identity rather than a shared account.

The approver first reads the impact, then enters the target name. BMC power actions run only as approved operations; policy decides which recovery steps run automatically and which require approval.

The console and CLI use the same API, so moving an action to the command line does not bypass its authorization or approval policy.

A hash chain protects the audit record

The immutable audit record links entries through a hash chain and supports export or delivery to a SIEM.

Tenant and workload boundaries remain explicit

Tenants cannot view one another's jobs, data, or queues. BareStax Core queues, quota, fairshare, and accounting preserve that boundary. It holds across the console, researcher portal, and CLI.

Compute-node SSH opens only for a user with a job there, apart from an explicit administrator exception.

Jobs run in Apptainer containers. Compute nodes accept only images promoted through the registry approval path and signed with an accepted cluster key. An unsigned image cannot start.

After import and scanning, two distinct people approve an image before the cluster key signs it for compute nodes.

Approved images remain read-only. User package and model caches stay in the user's persistent workspace. This separates the reviewed runtime artifact from changes a researcher retains between sessions.

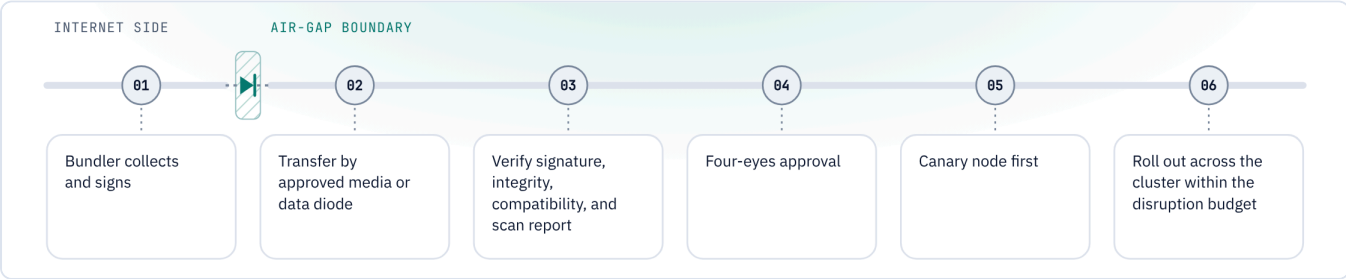
Release contents have a verifiable chain of custody

The connected-side bundler collects required software into a signed, content-addressed release bundle. Upstream repositories are pinned to timestamped snapshots, and repository key fingerprints appear in the bundle's security information. The same release therefore carries the same bytes into each installation.

Transfer uses approved portable media or a data diode. Import verifies the signature, integrity, compatibility, and attached scan report before promotion. Two distinct people approve that promotion.

Release signing keys are generated in an offline environment through a recorded key ceremony. The cluster installs packages from its local repository rather than reaching outside during node installation or upgrade.

Package locks, an SBOM, a license allowlist, and reproducible builds provide reviewable inputs for the software supply chain. Institutional OCI registries and institutional counter-signatures can participate in the same image trust path.



The release chain shows offline signing, controlled transfer, verification, independent approval, canary deployment, and budgeted cluster rollout.

Isolation does not reduce the operating model

No runtime path requires internet access. The control plane, node agents, console, researcher portal, CLI, local package repository, image registry, monitoring, error explanations, and runbooks remain available inside the isolated network.

Offline licensing adds no external service. BareStax sends no data outside the institution and connects to no remote server.

Identity and second-factor checks continue through local or connected institutional services. Signed release bundles cross the boundary on approved media or a data diode.

05

Deployment starts from your installed servers

Your institution racks the hardware, installs Ubuntu, and defines its network and storage environment. BareStax installs the control plane, joins and validates GPU nodes, configures BareStax Core, and keeps the cluster on a declared software state.

The same deployment model works on connected and fully air-gapped networks. No BareStax runtime component requires internet access.

The cluster starts from servers you have already racked

Area	Supported deployment
GPU nodes	NVIDIA data center and workstation GPUs, including HGX systems from Hopper and Blackwell generations and servers with PCIe GPUs. Validated hardware classes are listed in each release note.
Cluster size	Small profile: 2–4 GPU nodes. Standard profile: 5–128 GPU nodes, up to 1,024 GPUs, with a separate head node.
Network	InfiniBand and RoCE are supported, with NVLink and NVSwitch where present. These fabrics are optional; Ethernet-only servers can join.
Operating system	Ubuntu 24.04 LTS on amd64. BareStax preserves the institution's operating system installation and does not erase the disk.
Head node	A physical server without GPUs or a VM on the institution's existing virtualization platform. It hosts the control plane and cluster-local services.
Login node	The researcher entry point for SSH, the portal, and job submission.
Storage	NFS and S3 targets. BareStax configures clients, mount protection, health, and impact; it does not install the storage server.
BMC	Redfish-compatible BMCs. Inventory, sensors, event logs, and power state are visible; the Security approval policy governs power changes.

The full GPU is assigned directly to a job. BareStax does not use MIG or vGPU, and it does not place Kubernetes or a virtualization layer between BareStax Core and the GPU.

Installation puts the head node before compute capacity

Install `barestax-server` on the head node and create the cluster identity. The setup records the endpoint and writes the key backup to the specified file.

Connect LDAP, Active Directory, OIDC, SAML, or local identity, then configure built-in TOTP where required. Define cluster networking, NFS or S3 targets, the local package path, queues, tenants, quotas, and fairshare policy.

Test the alert chain before admitting compute capacity. The test confirms that an alert reaches the on-call recipient and that its code returns through the response path.

```
barestax-server init --cluster-name <name> --endpoint <fqdn> --key-backup <file>
barestaxctl alert test
barestaxctl doctor
```

Before a GPU node joins, BareStax checks firmware minimums, disk, Secure Boot, and connectivity. The server then registers with a token and waits for approval.

After approval, BareStax inventories the hardware, chooses the profile, installs the compatible stack, and runs GPU, NVLink, and InfiniBand checks plus two-node NCCL validation. Only a passing node becomes ready in BareStax Core.

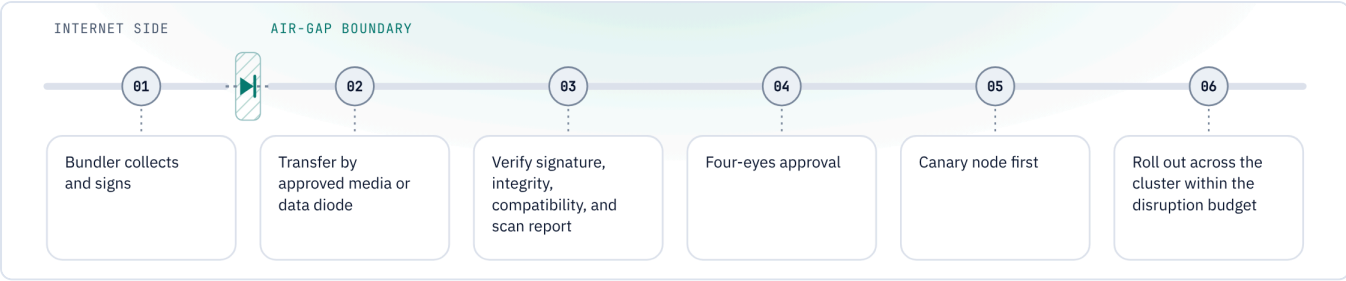
Air-gapped releases cross a controlled boundary

On a connected network, `barestax-bundler` collects required packages, repository metadata, signatures, scan information, and compatibility data. It produces a signed, content-addressed release bundle and verifies it before transfer.

Approved portable media or a data diode carries the bundle into the isolated network. Import checks the signature, integrity, compatibility, and scan report. Four-eyes approval then promotes the bundle to the cluster-local repository.

Rollout starts with one canary node. Running jobs are preserved as BareStax continues through the cluster within the configured disruption budget.

```
barestax-bundler export
barestax-bundler verify
barestaxctl release import <bundle>
barestax-server upgrade --offline
```



The release flow moves a signed bundle across approved media or a data diode, verifies it, requires independent approval, and deploys by canary.

Updates preserve compatibility and running work

BareStax distributes the kernel, DOCA-OFED, NVIDIA driver, CUDA libraries, Fabric Manager, DCGM, NCCL, and Apptainer only as tested sets. Timestamped repository snapshots and recorded key fingerprints fix the contents of each release.

The node rollout uses canary order and the configured disruption budget. The control plane upgrades through the same signed release process.

Each minor release line receives support for 24 months from its initial general release. Security fixes continue on that supported line, so an air-gapped institution can import the relevant bundle through the same controlled path.

06

Simple licensing for sovereign infrastructure

One unit, one annual subscription, one signed license file that works offline. Your cluster never depends on a connection to us.

The terms, in five lines

Priced per GPU node

Every GPU compute node is one license unit. Head, login, and build nodes are not counted. Two node classes, for nodes with 1–4 GPUs and nodes with 5–8 GPUs, with no difference by GPU generation. A license starts at two GPU nodes.

A subscription that never stops the cluster

The license is an annual subscription. If you do not renew, the last imported release and your licensed capacity stay in service, and jobs, draining, and recovery continue. Renewal covers new releases, support, and capacity changes.

A license file, not a license server

The license is a signed file installed offline. There is no license server, no outbound connection, and no telemetry.

24 months per release line

Each minor release line is supported for 24 months from its first general release. Security fixes arrive on that line as signed release bundles, through the same path you use in an air-gapped network.

Standard support included

Standard support is part of the subscription. Requests are handled at three priority levels, with response and resolution targets defined in the support agreement.

Demo first, then a paid pilot

Prove BareStax on your own hardware before you commit.

1. **Guided demo:** A remote session on the product, built around your team's questions.
2. **Technical session:** Your hardware, fabric, storage, identity, and transfer method.
3. **Paid pilot:** A separate head node and up to 4 GPU nodes for 90 days, starting when the hardware is ready.
4. **Acceptance drills:** Node join, two-node validation, a canary update, drain and recovery, an air-gapped import, audit export, and a control plane restore.

5. **Production order:** The pilot license is time-limited. If you order within 60 days, the full pilot fee counts toward the first year.

See the pilot acceptance checklist

How your quote is built

We quote after we have discussed your cluster scope and support needs. The quote is based on:

- the number of GPU nodes and their class, 1–4 or 5–8 GPUs;
- your support needs; and
- the pilot, if you start with one.

Your team can review the technical overview and both checklists before any commitment. Resources

07

Built for institutions that own their GPUs

BareStax is designed for organizations that run NVIDIA GPU clusters in their own data centers and need to keep control of data, keys, and operations: enterprise AI teams, research centers and universities, public institutions, and regulated sectors such as finance, health, telecom, and energy.

By sector

Public institutions

Clusters that stay inside the institution: no outbound connection, an offline license file, cluster keys and audit record kept in your data center, and two people on every critical action.

Research centers and universities

Shared GPUs for many groups: tenants, quotas, fair share, and accounting, plus a portal with Jupyter, VS Code, RStudio, and remote desktop sessions. Idle sessions close and free the GPU.

Finance, health, telecom, energy, and industry

Evidence your auditors can check: an exportable hash-chained audit record for your SIEM, signed and approved images only, an SBOM and package lock in the supply chain, and tenant isolation.

How BareStax compares

Approach	Strength	What stays with your team
Cluster manager and scripts	Mature installation tools	Upgrades, recovery, and air-gap procedures by hand; a script ordering error can leave a silent partial install.
Kubernetes with a GPU operator	Strong for service workloads	Extra scheduling and RDMA layers for batch training; OS, driver, fabric, and BMC remain outside.
Virtualization with PCIe passthrough	Familiar VM operations	GPUs stay pinned to VMs, idle when the VM is idle, with no sharing across jobs.
BareStax	One product from racked server to job queue; declarative and self-healing; air-gapped end to end	Rack the hardware and install Ubuntu.

What BareStax is not

Not a Kubernetes distribution, not a virtualization platform, and not a cloud service. It runs in your data center, sends no data outside, and does not install the operating system, autoscale, or balance load; your load balancer exposes long-running services.

08

BareStax keeps its cluster boundaries explicit

BareStax manages an installed Ubuntu server through job scheduling. Hardware ownership, storage servers, and external load balancing remain with the institution.

Infrastructure defines the cluster boundary

Does BareStax use Kubernetes?

No. BareStax is not a Kubernetes distribution and does not place Kubernetes between jobs and GPUs. It applies declarative reconciliation to system settings, kernel, drivers, fabric, BMC, and job scheduling.

Does BareStax virtualize GPUs?

No. BareStax assigns the full GPU directly to a job and does not use MIG or vGPU. There is no virtualization layer in the compute path.

Does BareStax install the operating system?

No. Your institution installs Ubuntu 24.04 LTS on amd64 before a server joins. BareStax preserves that installation, does not erase the disk, and configures the compatible stack above it.

Which GPUs does BareStax support?

BareStax supports NVIDIA data center and workstation-class GPUs. This includes HGX systems from Hopper and Blackwell generations and servers with PCIe GPUs. Each release note identifies the hardware classes validated for that release.

Which cluster networks are supported?

BareStax supports InfiniBand and RoCE, and records NVLink or NVSwitch where present. Neither fabric is required, so Ethernet-only hardware can join. Discovery also records PCIe and NUMA topology before profile selection.

What cluster sizes are covered?

The small profile covers 2–4 GPU nodes. Standard scale runs from 5–128 GPU nodes, with a ceiling of 1,024 GPUs and a separate head node. Different GPU classes keep their own queues.

What storage systems does BareStax manage?

NFS and S3 are client targets. When shared storage is missing, BareStax blocks fallback writes to local disk and leaves the storage server under institutional control. Capabilities carries the health and impact details.

Operations cover the node lifecycle

How are updates moved into an air-gapped network?

`barestax-bundler` collects and signs a content-addressed release bundle on a connected network. Approved portable media or a data diode transfers it into the isolated network, where import verifies signature, integrity, compatibility, and the scan report. Four-eyes approval precedes canary and budgeted rollout from the cluster-local repository.

Does any runtime function require internet access?

No. Runtime components stay on the institution's network without a remote-server connection. The Security page lists what remains available offline.

Can BareStax take over an existing cluster in place?

No. BareStax does not import an existing scheduler configuration in place; it generates its own BareStax Core configuration, accounting included. Servers join one by one with their Ubuntu installation intact. Queues, tenants, quotas, and fairshare are defined in BareStax, while researchers submit work from the command line, templates, or the portal. Your team and BareStax plan the sequence around the existing hardware and queue structure.

What happens if the control plane stops responding?

Running jobs continue, and compute nodes stay on their last known safe state. Local health gates keep a faulty node from accepting new work while the control plane is unavailable. Backups cover identity, PKI, the database, and BareStax Core state for restoration to clean hardware.

Can a researcher see why a job is waiting?

Yes. The portal gives each researcher the waiting reason, quota, logs, and controls for personal jobs. Operators see queue, fairshare, accounting, health, and capacity context in the console.

How does BareStax work with Prometheus and Grafana?

Exporters use standard Prometheus format, so an institution's existing Prometheus and Grafana installation can read the same data. BareStax also integrates with Alertmanager. The heat map and deep monitoring dashboards inside the console are BareStax panels, not Grafana. A test alert verifies that the alert chain reaches the on-call recipient.

How are faulty nodes handled?

BareStax withholds new placement from an unhealthy node and lets its current job finish. Health and recovery covers escalation, flap protection, and the storm brake.

Can BareStax expose long-running model services?

BareStax runs long-lived services in a dedicated BareStax Core queue with service registration and protection. External access comes through the institution's own load balancer. BareStax does not provide load balancing or automatic scaling.

Which languages are available in the product interface?

The product interface is in English. Arabic is available on the website, not in the product interface. Console details cover themes and keyboard use.

Security controls identity and software trust

Which identity providers can connect to BareStax?

BareStax supports LDAP, Active Directory, OIDC, SAML, and local accounts. Built-in TOTP provides a second factor in connected and air-gapped environments. Critical operations ask for that factor again.

How does BareStax decide which container images may run?

Jobs run through Apptainer. An image enters the packaged OCI registry, passes scanning, receives approval from two distinct people, and is signed by the cluster key. Compute nodes reject an image without an accepted signature, while institutional registries and counter-signatures can join the same trust path.

Commercial terms remain explicit

How long is a release line supported?

A minor release line stays supported for 24 months and continues to receive security fixes. The contract defines response and resolution times by severity.

How does licensing work in an isolated environment?

Licensing remains offline. Price depends on cluster scope and support level; Deployment explains the commercial boundary.