

NEURAL INVERSE

Security overview

Where Neural Inverse runs and who holds your data, the controls built into the product, how we attack our own software, how to verify a release, and measured performance. Printed from neuralinverse.com on 18 October 2026; the website is the current version.

Security and trust

Your code and data stay under your control.

How Neural Inverse protects what you build: where it runs, how it is isolated and recorded, and how to work with our security team.

Where it runs

You choose where your data lives.

Self-hosted

Runs on your servers. Your code, prompts and records stay on your network; we receive none of it.

In your cloud

Runs in your own AWS, Azure or Google Cloud account, under your keys and network controls.

Air-gapped

Installs from an offline bundle and runs with no internet connection, using models you host.

Hosted by us

We run it for you, with the same controls and audit record: shared in AWS US East (N. Virginia) or Europe (Frankfurt), or dedicated in the AWS, Azure or Google Cloud region you prefer.

Built-in controls

Security that is part of the product.

Tenant isolation

Every organisation's data is separated by the database itself, with row-level security that fails closed.

Encryption

Traffic is encrypted with TLS. Secrets, credentials and sensitive records are envelope-encrypted with keys in a key management service, and you can bring your own keys.

Tamper-evident audit

Every action by a person, agent or command is written to a hash-chained log that can be verified offline.

Identity

Single sign-on with SAML or OIDC, directory sync with SCIM and LDAP, passkeys and multi-factor authentication.

AI controls

Approved models only, data loss prevention on prompts and context, and approvals for risky agent actions.

Data residency

Bind your organisation to a region; its content and processing stay there. On the shared service, sign-in details are held in the United States; dedicated and self-hosted deployments keep them in your region.

Every model request goes through your controls.

One way to models	Engineers and agents reach models only through the gateway you control, with your approved list.
Checked before it leaves	Prompts, tool calls and context pass data loss prevention before they reach a model.
Attributed	Every model request and agent action is tied to a person or an agent identity.
Your keys, your providers	Use your own provider agreements or self-hosted models; provider keys stay in the vault.

How we build it

Security tested on every change.

Attack tests in CI

Security tests attack each defence directly, and each test is proven to fail when its defence is removed.

Cross-tenant tests

Automated tests try to read and write across organisations as the runtime database role, on every change.

Fails closed

If an isolation setting, policy check or audit write fails, the request is refused rather than allowed.

No outside calls

The platform makes no calls to external services at runtime, so it runs the same on a closed network.

Watches itself

Admission control, circuit breakers and supervised background jobs keep the platform stable under load and failure.

Standards

Evidence for your audits.

Govern checks your controls against rule packs for frameworks such as SOC 2, HIPAA, GDPR, ISO 42001 and the EU AI Act, and exports the evidence for your auditors. That evidence supports your certification; it is not a certification of your system or of Neural Inverse. Certification stays with your organisation and its assessor.

For your security review, ask us for our current security documentation and questionnaire responses.

Under a non-disclosure agreement, your team can review the source code, receive our capacity and benchmark reports, and run our benchmark tools on your own hardware.

Request security documents →

Report a vulnerability

Found a security issue?

Email security@neuralinverse.com. We acknowledge reports within 72 hours and work with you on a fix and coordinated disclosure.

Documents

Vulnerability disclosure policy →

How to report a vulnerability and what we commit to.

Security documents →

Request our security pack, questionnaire responses and DPA under NDA.

Software supply chain →

Verify a release: checksums, signatures, provenance and SBOMs.

Red-team testing →

The attackers we assume, the tests that play them, and what we do not claim.

Security advisories →

Fixed vulnerabilities in our products, and what to do.

Responsible AI →

The principles we build AI for regulated engineering by.

Sub-processors →

Who processes personal data for the service we host.

Privacy notice



How we handle personal information.

Security architecture



Technical detail for your security review, in the documentation.

Legal



Terms, cookies, accessibility and company information.

Security and trust

We attack our own products.

Regulated customers ask what someone could do if they set out to manipulate the platform. We answer with attacking tests: each one plays an attacker against the real code, and runs on every change.

Who we test against

Attackers who have read the source.

An insider with a modified client

Our source is not published, but we never rely on that: we assume an employee with a valid account, API key or agent token has read the source, runs a modified client and sends requests the real apps never would. Every control is enforced on the server.

An AI agent escaping its policy

An agent that writes to its own policy, hooks or audit folder, chains a forbidden command behind an allowed one, or follows hostile instructions in a repository file.

Another organisation

A signed-in user of one organisation trying to read, change or even detect the existence of another organisation's data.

A compromised extension or window

A malicious editor extension, or a window compromised through a crafted file, trying to reach audit records, approvals, policy or local sockets.

A local user without admin rights

Someone who edits files in the app's data folder or the repository, sets environment variables or starts the app with debugging flags.

Replay and rollback

Restoring an older policy bundle, audit ledger, signed checkpoint or approval to undo a decision.

How the tests work

Every defence has a test that tries to break it.

An attacking test per scenario

Each attack is written as an automated test that tries to break a control against the real code. On the server, the tests run as the same restricted database role the product uses, against PostgreSQL.

Proven to catch the attack

Each test is shown to fail when the defence it targets is removed, so a passing test means the defence works, not that the test is weak.

Run on every change

The suites run in continuous integration on changes to the code they protect, and a failure blocks the change.

Fixed, not filed

What the attacks found was fixed in the product, and each fix has its test. A module's security review maps every threat to its control and the test that proves it.

Neural Inverse Govern

What the server refuses.

Approvals and two-person rules

A requester cannot approve their own request, move an approval onto a different action, choose their approver, approve with an API key, or vote after a request expires.

Agent policy and the kill switch

API keys cannot loosen policy or release a kill switch; leaving a field out never loosens a rule; decisions from the edge are re-checked, not trusted as sent.

Tenant isolation

Tests try to read and write across organisations as the runtime database role; row-level security refuses them and fails closed.

Context and shared knowledge

Evidence is append-only; decisions such as approving, sharing and legal holds need a person, not an API key; a missing item and an item you may not see answer the same way.

The audit record

First-party apps can only record events for the signed-in person and their own organisation; a request naming anyone else is refused.

Neural Inverse Build

What the Editor, Agents app and CLI refuse.

Audit and approvals on the desktop

An extension or a crafted message tries to append, read or erase audit events, sign an approval or change policy. Decisions and keys never live in a window or an extension.

Rollback and tampering

A local user swaps in an older valid policy bundle, edits or restores the audit ledger, deletes the outbox or edits approval records in the repository. Each attempt is refused or detected and recorded.

Agents

An agent writes to its policy, hooks, git hooks or the audit folder, or chains commands behind an allowed one.

Sign-in, vault and context

Account files, sign-in location binding, vault commands and agent runs, context sharing, team memory and agent instruction files are each attacked in the Editor, the Agents app and the CLI.

What we do not claim

The limits, stated plainly.

Your own administrators

A local administrator can do anything on their own machine; we make tampering detectable, not impossible. A compromised administrator account can do what that account is allowed to do, and every action is recorded.

Whoever holds the keys

Someone with the database owner role or the audit signing keys can change records. Verifying the chain detects the change; it cannot prevent it. That is why your deployment's keys and database access matter.

What a person already copied

Content copied out by someone who was allowed to read it.

Instructions you approved

An approved instruction file that is hostile is labelled, delimited and recorded, but an agent may still follow it.

Two accounts, one person

Separation of duties works by account; the platform cannot know that two accounts belong to the same person.

Independent assessment

This is our own testing. It does not replace an assessment by your security team or an independent tester, which you can run against your own deployment under our vulnerability disclosure policy.

For your review

The threat models and tests,
under NDA.

Under a non-disclosure agreement, your security team can read our threat models, the security review of each module and the attacking tests themselves, and run them.

Request security documents →

Vulnerability disclosure policy →

Security and trust

Software supply chain

How to check that what you install is exactly what we built: checksums, signatures, build provenance, a bill of materials for every release, and security advisories your tools can read.

What each release carries

Product	Checksums	Signatures	Build provenance	Bill of materials
Govern	SHA-256 checksums file	Binaries and container images signed with Sigstore cosign	SLSA level 3	SPDX, signed
CLI	SHA256SUMS, checked by the installer	macOS binaries code-signed	Container image provenance	Container image SBOM
Editor and Agents app	[desktop app signing and SBOM: to be confirmed]			

Verifying Govern

1. Check each file against the release's SHA-256 checksums file.
2. Verify the signature on each binary and container image with cosign, against the signing identity we give you with the release. Signatures are keyless (Sigstore), so no long-lived signing key can leak.
3. Verify the SLSA provenance: a signed statement of the source commit and the build that produced the binary, from an isolated build service the build steps cannot alter.
4. Govern builds are reproducible: building the same commit with the published build script produces the same binary, so you can rebuild and compare.

Verifying the CLI

The installer downloads the release's SHA256SUMS, checks the binary against it, and refuses to install on a mismatch or a missing checksum. It does not edit your shell profile. macOS binaries are code-signed, and the container image is published with its build provenance and bill of materials. See [installing the CLI](#).

The Editor and the Agents app

Your administrator provides the installers for your organisation. How they are signed for each platform: [\[desktop app signing: to be confirmed\]](#).

Software bills of materials

Every Govern release has a software bill of materials in SPDX format, signed so you can check it came from our build. Customers receive the bill of materials for each version they install; ask through the [contact form](#) for any version.

Machine-readable advisories

Our [security advisories](#) are also published as CSAF 2.0 documents, the OASIS standard that vulnerability-management tools read. Point your tool at `/.well-known/csaf/provider-metadata.json` on this site. Our `security.txt` lists it too.

Closed networks

Checksums can be checked offline. Keyless signature checks consult a public transparency log, so on a closed network verify signatures and provenance on a connected machine before you transfer the release, and check the checksums again inside. Your administrator can host the release files and checksums on an internal mirror.

Our security key

To encrypt a vulnerability report to security@neuralinverse.com, use our PGP key at `/.well-known/pgp-key.txt`. Fingerprint: `[PGP fingerprint: to be confirmed]`.

Deployment

Deploy it where your data has to stay.

Run Neural Inverse Govern on your own servers, in your own cloud account, or on a network with no internet access. The controls and the audit record are the same in each.

Server

A single Go binary

Database

PostgreSQL 15 or later

Install with

Docker Compose or Helm on Kubernetes

Network

Designed to run with no internet access

Overview

Govern is built to run on infrastructure you control. The server is a single binary with PostgreSQL as its only required dependency, and it is designed to make no calls to external services at runtime. Choose where it runs; what it enforces does not change.

Self-hosted ↓

In your cloud ↓

Air-gapped ↓

Hosted by us ↓

Compare ↓

Self-hosted

On your own servers.

For a single host or your own Kubernetes clusters, on premises.

What you run

The Govern server, the web console and PostgreSQL, with Docker Compose on one host or the Helm chart on Kubernetes. Optional planes handle telemetry intake, data-plane traffic and vendor audit polling.

What we run

Nothing in your environment. We publish releases with their release notes.

Where data lives

In your PostgreSQL database and your object storage, either a shared directory or an S3-compatible bucket. Older audit entries can move to a bucket you own.

How updates arrive

You choose when. Back up, then upgrade: migrations run first, on every Docker Compose start or in a Helm pre-upgrade hook. To go back, restore the backup rather than migrating down.

Requirements

PostgreSQL 15 or later (the Docker Compose stack ships PostgreSQL 17). Docker 24.0 and Compose v2.20 or later with 2 GB of memory and 10 GB of disk, or Kubernetes 1.29 or later with Helm 3. A reverse proxy or Ingress for TLS.

Keys and secrets

You generate the deployment keys and keep them in your secret store. In production the server refuses to start with missing or placeholder values.

In your cloud

In your own cloud account.

On your Kubernetes cluster, next to a managed PostgreSQL service, your object storage and your secret store.

Kubernetes guide →

What you run

The Helm chart on your cluster. Each plane runs as its own workload, non-root with a read-only file system, spread across zones, with network policies that deny ingress by default.

What we run

Nothing in your account. We publish releases and the Helm chart.

Where data lives

In your account: a managed PostgreSQL database and your bucket, encrypted with your KMS key. The bucket is never public; downloads go through the API, which checks access and audits each one.

How updates arrive

You upgrade with Helm to a pinned release; the chart refuses an unpinned tag. Hooks run the migrations first, and pods roll over without taking capacity away.

Requirements

Kubernetes 1.29 or later and Helm 3. PostgreSQL 15 or later reachable from the cluster, with TLS required by default. For more than one server replica, a shared storage class such as EFS, Azure Files or Filestore, or an S3-compatible bucket.

Secrets

Secrets never go in chart values. Create them as Kubernetes Secrets, or sync them from AWS Secrets Manager with the External Secrets Operator.

Air-gapped

On a network with no internet access.

Everything Govern needs after installation, from container images to identity, mail and model servers, can live inside your boundary.

Air-gapped installation guide →

What you run

The same stack, installed from an offline bundle of images, Compose files and Helm charts with checksums. Your internal registry, identity provider, mail relay, model servers, DNS, time and certificate authority sit inside the boundary.

What we run

Nothing. We sign offline licences. Your installation holds only the public key to check them and refuses a licence it cannot verify.

Where data lives

Inside your network. Audit archiving works with an on-premises S3-compatible store that supports Object Lock, and audit verification needs no network access.

How updates arrive

A new bundle is built on a connected host, moved through your approved media process and checked against its checksums inside the gap. Images load with pulls turned off.

Requirements

PostgreSQL 15 or later. An internal registry, or images imported on every node. Object storage on a shared directory or an S3-compatible store inside the network.

Models

Self-hosted model servers inside networks you list. Restricted work can be limited to them, and workload identity accepts key sets you upload for issuers Govern cannot reach.

Verify before you rely on it

Run each release on a host with no route out and record every connection attempt. A strict air gap is enforced by your network controls; these tests are defence in depth.

Hosted by us

Or we run it for you.

The same software and controls, operated by Neural Inverse.

Talk to us about hosting →

What you run

Nothing to install. Your people sign in and use it.

Shared

Your organisation on our shared service at cloud.neuralinverse.com, in AWS US East (N. Virginia) or AWS Europe (Frankfurt). Your content stays in the region you choose; sign-in details (email, name, credentials, sessions) and billing records are held in the United States.

Dedicated

An installation for your organisation alone, on AWS, Microsoft Azure or Google Cloud, in the region set in your contract, with its own sign-in system so identity data stays in that region too. Use your own domain, keep encryption keys in your own cloud account, and choose a single region if you need one.

What we run

The Govern installation, its database and storage.

Where content lives

Code context, prompts, vault secrets, logs and audit content are created and kept in your region and never copied out of it. Every audit entry records the region that wrote it, and tokens from one region are refused by another.

Who controls it

We operate the service and process your data as your processor, under the data processing addendum. Our staff reach your data only through audited emergency access. To keep it entirely in your hands, run it yourself.

How updates arrive

We apply them.

Compare

The options side by side.

Operated by

Self-hosted: You

In your cloud: You

Air-gapped: You

Hosted by us: Neural Inverse

Runs on

Self-hosted: Your servers

In your cloud: Your cloud account

Air-gapped: Your isolated network

Hosted by us: Shared: AWS N. Virginia or Frankfurt. Dedicated: AWS, Azure or Google Cloud, region of your choice

Install method

Self-hosted: Docker Compose or Helm

In your cloud: Helm

Air-gapped: Offline bundle, then Compose or Helm

Hosted by us: None

Database

Self-hosted: Your PostgreSQL 15+

In your cloud: Your managed PostgreSQL 15+

Air-gapped: Your PostgreSQL 15+

Hosted by us: Operated by us

Updates

Self-hosted: When you choose

In your cloud: When you choose

Air-gapped: By transferred bundle

Hosted by us: Applied by us

Data location

Self-hosted: Your database and storage

In your cloud: Your cloud account

Air-gapped: Inside your network

Hosted by us: Content in your home region; on the shared service, sign-in details in the United States

Who holds your data

Self-hosted: Only you

In your cloud: Only you

Air-gapped: Only you

Hosted by us: Neural Inverse, as your processor

In every option

Where it runs changes. What it enforces does not.

The same software

Every option runs the same Govern code paths. A self-hosted installation is one region of its own.

Identity and policy

Single sign-on, roles, model access, agent policy and approvals behave the same wherever Govern runs.

Enforced on the server

Model requests, MCP tool calls and agent-to-agent calls are decided in Govern's gateways, and decisions fail closed.

Audit record

Every change and decision goes into the same hash-chained, append-only log, with signed exports you can verify offline.

Tenant isolation

Encryption

The database keeps organisations apart with row-level security, and the server refuses to start on a role that could bypass it.

Secrets are encrypted at rest under a key hierarchy, and policy delivered to machines is signed.

Guides

Install, operate and review.

Deploy and operate →

Requirements, planes, secrets, health checks and upgrades.

Docker Compose →

A single host, from first start to production.

Kubernetes →

Every plane with the Helm chart, on premises or in your cloud.

Air-gapped →

Build, transfer, install and verify a release offline.

Security architecture →

Isolation, encryption, audit and outbound controls.

Security and trust →

How we protect your code and data, and how to report a vulnerability.

Platform

Measured performance.

Throughput per core for Govern services, with the hardware and method behind every figure.

How we measure

Figures you can check.

Measured, not estimated

Every figure here was measured. We do not extrapolate from one service to another; components without a published result are not listed.

A floor, not a ceiling

Unless a section says otherwise, runs were on one shared 4-vCPU virtual machine (Intel Xeon, 2.1 GHz) with other load tests running at the same time, and with the load generator, the database and every component on the same cores. Dedicated hardware does better.

Per core

Where we can, we give throughput per CPU core, so you can size your own deployment from your hardware.

Check it yourself

Under a non-disclosure agreement, you receive the full reports and our benchmark tools, and can run them on your own hardware.

Telemetry

100,000 events a second, zero data loss, no Kafka.

Govern's own telemetry pipeline: intake writes each event to a write-ahead log on disk, ingest stores it in PostgreSQL, and older data is compacted into Parquet files on object storage or a local directory. No Kafka, separate analytics database or other streaming system is needed; Kafka is only an optional connector. Measured with every component and PostgreSQL sharing the same four cores, with events of about 550 bytes.

Measure	Result
Intake, per core	About 250,000 events a second
Write-ahead log	About 500,000 records a second
Sustained, 60 seconds	100,000 events a second, every event stored
Peak, 60 seconds	150,000 events a second, every event stored
Soak, 5 minutes	80,000 events a second, every event stored
From sending to stored, at 100,000 a second	32 ms median, 809 ms at the 99th percentile
Compaction to Parquet	277,000 rows a second, 25.7 times smaller
Query the archive: first 100 rows of an hour / a day	2.0 ms / 15.8 ms
Query the archive: every row of a day (48,000)	260 ms

Zero data loss: under load we killed intake and ingest processes, cut every database connection, filled the disk and restarted components. In every run, every event produced was stored exactly once.

Data loss prevention applies when telemetry is read: every string in a query result passes your organisation's DLP redaction. The figures above measure intake and storage.

Policy decisions

An agent's action is checked in about 10 microseconds.

Agent policy is decided where the agent runs, from signed policy bundles, and every decision is written to a local write-ahead log; there is no network round trip. Measured on one core with a realistic organisation of 3 guardrails and 12 agent policies, CPU only.

Measure	Result
Decision, including recording it	10.2 μ s; 97,800 a second per core
Latency, median / 99th percentile	5.7 μ s / 35 μ s
Evaluation alone	2.05 μ s; 487,000 a second per core

Audit log

Tens of thousands of audit entries a second, none lost.

Each organisation has its own hash-chained audit log. Three server replicas appended entries with every commit flushed to disk, and every acknowledged

entry was checked to be stored exactly once. One run per row; the database shared the same four cores.

Organisations, concurrent writers	Entries a second, 99th percentile
1 organisation, 64 writers	12,521; 11 ms
1 organisation, 1,024 writers	34,949; 114 ms
50 organisations, 64 writers	4,775; 39 ms
50 organisations, 1,024 writers	26,160; 111 ms
500 organisations, 64 writers	2,613; 80 ms
500 organisations, 1,024 writers	5,915; 488 ms

In all nine runs, every acknowledged entry was stored exactly once. An entry can be read as soon as it is acknowledged: about 1 ms at the median.

Offline verification of a chain ran at 98,000 to 143,000 entries a second. Searching 100,000 entries for the newest 100, one action or the last hour took 1.2 to 1.6 ms at the median.

Sign-in tokens

Over 130,000 token checks a second per core.

The cost of issuing and checking access tokens, measured on one core, CPU only; the database work of a sign-in is not included. Password checks are slow on

purpose, so single sign-on and passkeys are the fast path.

Operation	Result
Check an access token	7.6 μ s; 132,000 a second per core
Issue an access and refresh token pair	3.8 μ s; 264,000 a second per core

Sandboxes

A command in a sandbox answers in about 5 milliseconds.

Commands run in Firecracker microVMs and are measured end to end over the API, from request to result. Measured on one AWS m8i.2xlarge instance (8 vCPU) running the database, the API, the worker and 50 microVMs together.

Command, concurrent callers	Median / 99th percentile
Empty command, 1 caller	5.3 ms / 11.1 ms
Echo, 1 caller	5.4 ms / 6.7 ms
Empty command, 10 callers	22.2 ms / 42.7 ms
1 MB of output, 1 caller	42.9 ms / 90.4 ms
Empty command, through another server replica	6.3 ms / 8.2 ms

Output streams back as it is produced, and every result is stored exactly once, even when a connection drops. Files of up to 1 GiB move in pieces, checked end to end with SHA-256.

Directory sync

100,000 users synced in under two minutes.

Users created and updated through the SCIM service against PostgreSQL, 16 parallel streams as an identity provider pushes them, including each user's membership and log entry. The HTTP layer is not included.

Users	Time
10,000, created	14 to 22 seconds
100,000, created	97 seconds
100,000, updated in a full re-sync	117 seconds

DNS

Over 100,000 queries a second on one core.

The authoritative DNS server, pinned to one core, with the load generator on the other three. Signed answers cost about the same as unsigned ones, because the signature is cached with the answer.

Query	Answers per second, one core
Simple A record	114,926
DNSSEC-signed A record	107,502
Name that does not exist (never cached)	46,493

A record change was served by a replica within 2.6 ms at the median and 61.5 ms at the 99th percentile after the API answered, over 100 changes.

AI Gateway

Under a millisecond of overhead, with no database on the request path.

A dedicated AI Gateway, measured against a model stub that answers at once, so the figures are the gateway's own cost. Policy comes from signed bundles and every request is still checked, metered and recorded; the request path makes no database calls.

Measure	Result
Requests per core, end to end over TCP	3,380 to 3,460 a second
Requests per core, gateway work only	About 9,400 a second
Overhead, end to end at 2,000 to 3,000 requests a second	0.06 to 0.08 ms median, 0.28 to 2.29 ms at the 99th percentile
Overhead, gateway work only	0.03 ms median, 0.33 to 0.82 ms at the 99th percentile
Database statements on the request path	None
Memory per open stream	107 to 118 KiB; 10,000 streams held on 2 cores

Per core means per core of the gateway process's own CPU. Two in-process runs reached 2.06 ms and 4.82 ms at the 99th percentile while other load ran on the machine.

Streams

Over 270,000 records a second, on disk before they are acknowledged.

One Streams broker and its clients in one process, every write flushed to disk before it is acknowledged: 8 partitions, 1 KiB records, 64 concurrent producers.

Scenario	Result
Synchronous produce, 64-record batches	272,143 records a second (279 MB/s), 47 ms at the 99th percentile
Synchronous produce, 16-record batches	92,786 records a second (95 MB/s), 45 ms at the 99th percentile
Batching producer, single sends	166,447 records a second (170 MB/s), 14 ms at the 99th percentile
Three brokers, one copy	110,789 records a second
Two copies, acknowledged by all	61,547 records a second
Three copies, acknowledged by all	35,380 records a second
Endpoint authorizer overhead per request	About 44 μ s open; 99 to 125 μ s with JWT
Sensitive-data screening of record payloads, block mode	112 to 131 μ s per 10-record request, against 42 to 46 μ s without it

The replicated runs put all three brokers on one disk and the same four cores, so every copy is a flush to the same disk; on separate machines the copies proceed in parallel.

In failure tests we killed and paused brokers under load over 20 iterations with three copies: all 412,159 acknowledged records were present at their offsets, none was stored twice, and every copy was byte-identical at the end.

What you run

Built from our own source.

Our own code

Govern's services, including the vault, key management, DNS, streams, telemetry, sandboxes and cloud workspaces, are written by Neural Inverse. They are not copies or forks of open-source projects, and not resold or wrapped third-party services.

One database

The only other service they need is PostgreSQL. Long-term archives go to S3-compatible object storage or a local directory. Sandboxes and workspaces run in Firecracker microVMs, the open-source virtual machine runtime, on your compute hosts.

No extra licences

You need no separate licence for any of them. PostgreSQL and Firecracker are open source and need no paid licence. Our code uses open-source libraries, not open-source services.

No outside calls

The services make no calls to external services at runtime, so they run the same on a closed network.

Under load and failure

It keeps itself healthy, and tells you when it cannot.

Sheds load before it fails

An adaptive concurrency limit with request classes: health probes are never shed, sign-in keeps headroom, and exports and bulk jobs give way first. One organisation can hold only a share of capacity.

Repairs itself, within limits

Remediation playbooks recycle database connections, restart stalled jobs, release memory or shed batch traffic, then check that the repair worked. Disruptive actions such as draining a node wait for an operator's approval, and a kill switch stops them all.

Watches its own health

Baselines learnt per hour of the week, burn-rate rules on availability, and canaries of its own sign-in endpoints. Detection and alerting keep running from memory while the database is down.

Across every part

The same self-healing runs in the data plane, the compliance plane, the provisioner and the sandbox daemons.

Under NDA

Full reports, the code and the tools.

Under a non-disclosure agreement, your team can review the source code, receive our capacity and benchmark reports for the other services, and run our benchmark tools on your own hardware.

[Ask for the reports →](#)

[Deployment options →](#)