AHEAD



NEOCLOUD SECURITY:

Hardening Infrastructure with Zero Trust

A Zero Trust approach to neocloud security, from firmware to workloads

Neoclouds provide GPU capacity and specialized infrastructure for accelerated computing workloads across AI, high-performance computing, and other data-intensive use cases. But properly securing those services means first reviewing the layers that support the workload, including components you can't directly inspect or configure.

Secure architecture is critical to risk reduction. It's essential for limiting unauthorized access, cross-tenant exposure, and service disruption. Zero Trust shouldn't just be a checklist item in these discussions. As a discipline, its principles should guide these architecture and operating decisions. Hardening and the controls built around those decisions have to be evaluated by the protections they provide the workload.

Lightning AI and Voltage Park are a real example of what an integrated neocloud operating model looks like in practice. The two companies announced a January 2026 merger that combined Lightning AI's software with Voltage Park's owned GPU infrastructure into a single platform. The advice in the rest of this paper isn't assessing this deal, or really any named provider's implementation. Instead, consider it a broader conversation that applies across neocloud service models. Bringing software and infrastructure together is an opportunity to coordinate identity, provisioning, telemetry, and lifecycle controls across the service.



Define Neocloud Trust Boundaries

Let's start with the workload being considered, and **three questions** to kick things off:

1. What data and model assets need protection?
2. What disruption can the business tolerate?
3. What access does the workload require?

Answering these first helps determine which service model and isolation boundary can best support these requirements.

Bare-metal access, a virtual machine, a managed cluster, and an inference API each offer different levels of control. Those differences matter more to the review than the cloud label. GPU firmware, high-performance interconnects, orchestration, and management services can all remain provider-controlled even when the customer administers the host operating system.

> **Best advice:** Document which layers the customer controls and which the provider controls, and where those responsibilities overlap. For layers outside the customer's control, establish who's accountable, and what evidence the customer can obtain. Having a responsibility matrix helps to record ownership, while technical validation can determine whether the intended boundary is enforced.

Apply Zero Trust Principles to Neocloud Architecture

NIST SP 800-207 describes Zero Trust Architecture as the use of Zero Trust principles to plan infrastructure and workflows. Those principles emphasize resource protection, explicit authentication and authorization, and the absence of implicit trust based on network location or asset ownership.

As far as neoclouds go, these same Zero Trust principles can be used to define policies for customer access and the provider's internal control paths, with identity, network, workload, and data controls used to enforce those policies. Any access decision should be based on the requested action and resource, along with applicable policy, and available device or workload security context. Never make these decisions based on identity alone, because identity isn't a single category. It spans tenant users, provider operators, service accounts, orchestration components, and automated processes. All of these need to be distinguished from one another. Provider access shouldn't receive broad access to customer workloads just because it belongs to the provider. And customer root access to a host shouldn't confer authority over provider management systems either.

Instead, rely on centralized or federated identity, scoped credentials, role separation, phishing-resistant multifactor authentication for privileged human access, and time-bound privilege. Wherever the service can't support a required identity control, evaluate compensating controls through a documented exception process. Don't omit the requirement. Define emergency access separately, restrict and log its use, and test it so that an identity-service outage doesn't create an uncontrolled administrative bypass.

CISA's Zero Trust Maturity Model, Version 2.0 (April 2023) provides a structure for reviewing identity, devices, networks, applications and workloads, and data. It also incorporates visibility, automation, and governance across those areas. Use it to check review coverage instead of making a higher maturity rating the objective. Prioritize changes by workload risk and validate the effectiveness of the implemented controls.



Hardening the Neocloud Stack, Layer by Layer

Hardening means defining required functionality, removing unnecessary exposure, securing what remains, and then maintaining that configuration. These are requirements that have to be established before you select and add more security products. Don't just disable every feature or apply a benchmark without understanding the workload.

The October 2023 joint advisory, NSA and CISA Red and Blue Teams Share Top Ten Cybersecurity Misconfiguration, identifies recurring problems which include unsafe defaults, improper privilege separation, insufficient segmentation, and poor patch management. Admittedly, these observations are from assessments and operational work in government and private sector environments, not a specific neocloud study. But they still support examining configuration as part of the architecture stack.

So start by working from platform and firmware through the operating system, network, orchestration, applications, and data. Begin the review at the lowest relevant layer, even when the customer has to rely on provider evidence rather than direct access. Then, at each ensuing layer, define a versioned baseline, the responsible party, and a method for checking the resulting configuration. Use applicable vendor guidance along the way, with configuration benchmarks and documented adjustments for the platform and workload.



Platform, Firmware, and Management Interfaces

A real platform review means accounting for everything sitting beneath the workload: BIOS or UEFI, accelerator firmware and VBIOS, the BMC, SmartNICs, DPUs, and the hypervisor if one's in the mix. Once these are established, you can tackle them step by step.

First, define supported firmware versions and security settings, including boot and firmware-verification features where available. Those features do not, by themselves, establish tenant isolation. After that, separate platform administration from tenant workload access. Restrict BMC, IPMI, Redfish, switch, SmartNIC, and DPU management paths to their authorized operators.

Make sure to disable unnecessary management interfaces and remove or replace default credentials. Where tenant access is required, document its purpose and permitted scope. Finally, record the GPU allocation and sharing mode, its supported isolation properties, and any constraints relevant to the workload. Don't infer isolation from a dedicated-compute label.

Operating Systems, Drivers, and Runtimes

Treat the operating system and its accelerator dependencies as one supported configuration, rather than reviewing each package independently. Be sure to maintain a full inventory of:

> the kernel
>
> GPU and network drivers
>
> CUDA or equivalent runtime
>
> container runtime
>
> approved images, including versions and support status

From there, you have to narrow what the system can do. Strip out the services and packages nobody's using, restrict listening interfaces, and limit host privileges, device access, and filesystem mounts to what the workload actually needs. Nothing extra along for the ride.

Artifacts should come from approved sources, with provenance and integrity checked through the mechanisms those sources support. Include base images and runtime dependencies in the vulnerability and patch management process. Updates will still have to earn their way into broader deployment. Test for security requirements, component compatibility, and workload operation. A successful job is not evidence that the host remains correctly configured. When a required component can't be updated, that's not an automatic, indefinite exception. Be sure to document the exposure, compensating controls, accountable owner, and the condition under which it gets reviewed or retired.

Networks and High-Performance Fabrics

The review should include every network carrying traffic, whether that's customer, management, storage, monitoring, or the east-west links GPUs use to talk to each other. Frontend networking and backend interconnects often lean on different isolation mechanisms, so treat them as separate validations. Map out which paths are actually permitted between tenants, hosts, control planes, storage, and management services. Once that's complete, you can lock down everything else using controls the network implementation supports.

From there, configure tenant membership, routing, segmentation, and whatever fabric-specific isolation controls apply. This includes how memberships or keys get assigned and revoked, wherever the technology relies on them. Be sure to test for isolation from the tenant's own permitted access level. Can they reach other tenants? Can they reach provider management interfaces from the access level they're supposed to have? This is all the more reason for keeping testing inside an agreed scope and authorization. Nothing freelance.

Last but not least, document any shared dependencies and what breaks if they fail. A network diagram shows intended separation, but it doesn't prove enforcement.

Orchestration and Applications

Kubernetes APIs, GPU operators, schedulers, provisioning services, model registries, and inference endpoints can all change or expose the workload's security state. Keep control plane and management APIs restricted to authorized identities and paths, and keep tenant administration walled off from provider administration so the two don't blur together.

Once that's settled, you can move on to configuring workload identity, runtime restrictions, and network policies that match whatever tenancy model you're running. Namespace separation isn't enough to call mutually untrusted tenants isolated. Privileged workloads and host-resource access should be kept to a minimum; when an operator or runtime component genuinely needs elevated permissions, make sure to record and constrain who can invoke or modify it.

Validate authentication, resource authorization, and administrative exposure at model registries, inference APIs, and provisioning interfaces, along with testing application functionality. Where resources are shared, defined quotas, concurrency limits, or admission controls can all keep one tenant from starving the rest. Plus, disruption tolerance can be translated into recovery requirements, so it's clear what state needs to be restored or rebuilt, who owns that, and how to test recovery against agreed objectives.

Workloads, Data, and Telemetry

Data access requirements don't stop at the model. They have to cover datasets, prompts, embeddings, retrieval content, API responses, logs, infrastructure metadata — the works.

The good news is you're answering the same handful of **questions for each of these classes:**

What's permitted to collect, where does it live, and what's it used for?

Which identities are authorized, within what tenant scope, at the API, service, and storage layers?

How is it protected in transit and at rest? And who owns the keys and responsibilities that come with them?

How long does it stick around — AKA, when does it get deleted, and what gets logged afterwards?

As a hypothetical, imagine a tenant-specific dashboard, with the ability to retrieve records through a backend identity and access to every tenant. If that backend doesn't enforce authorization on each request, the tenant-specific display doesn't prevent cross-tenant disclosure. You have to test the underlying authorization path. Don't just rely on what the dashboard shows you.





Include Security Tools in the Architecture Review

Security tooling isn't exempt from your review either, as these tools need to be in line with everything they're meant to watch, whether that's a SIEM, an endpoint platform, an identity system, a vulnerability platform, or a monitoring service. Any of these might be sitting on privileged credentials and sensitive data, which means the same hardening requirements apply to their management interfaces and integrations too.

Look closely at a few things in particular, like how far the collection credentials reach, or who has the power to change detection or enforcement policy. Don't cut this corner, because one integration could end up accidentally opening a backdoor and expose another tenant's information. No security function makes your platform exempt from this type of review.

Maintain the Security Baseline Through Change and Reassignment

To be clear, configuration assurance is not the same as change control. Approval establishes that a change was authorized. Functional testing, meanwhile, establishes that the service works. But more importantly, neither one confirms that configurations are conforming to the intended security baseline.



To do so takes monitoring security-relevant configurations and changes wherever technically practical. If direct monitoring is unavailable, define alternative checks, along with their frequency and the resulting visibility gap. You should also record any deviations, investigate alerts, and verify remediation. This process should be applied to administrator, pipeline, orchestration, and vendor changes.

Reprovisioning and reassignment will also require validation. Define procedures by component and data type. Host images, device memory, persistent storage, firmware settings, credentials, and retained telemetry don't all have the same reset or disposal requirements. Support these procedures and record the outcome before returning the resource to tenant service.

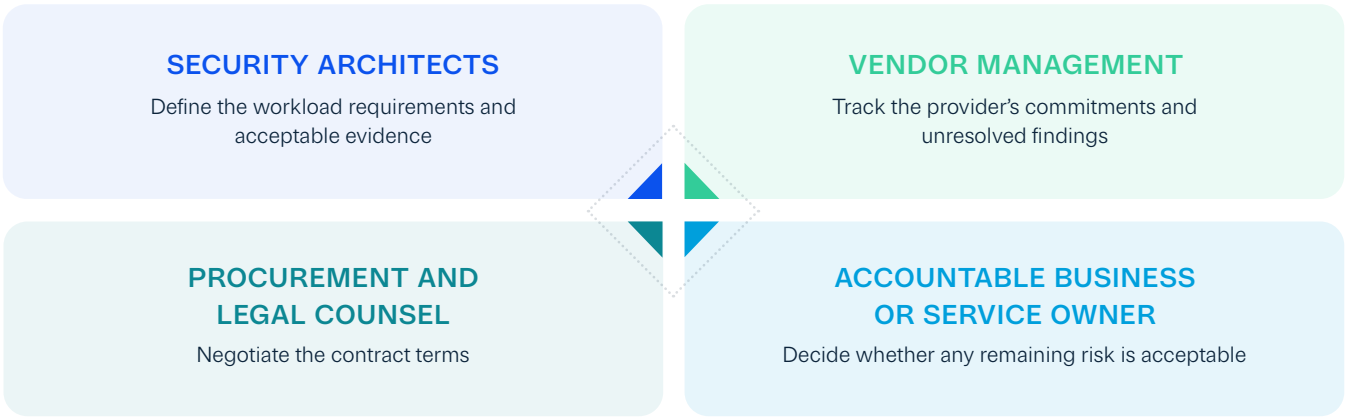
What if, for example, a GPU server has been rebuilt for a new customer, but a management API still authorizes the previous tenant to administer it? Re-imaging the operating system wouldn't correct that separate authorization. Reassignment checks should include revocation of prior tenant access, as well as the host and data-handling procedures.

For automated changes, separate interpretation, policy decisions, enforcement, and verification. Policy as code can evaluate versioned rules before deployment; post-change validation checks the resulting state. Protect the policy repository and enforcement service from unauthorized modification.

If AI proposes infrastructure changes, keep approval and enforcement independent of the proposing system. Require human approval for actions outside preauthorized scope. Define permitted actions and recovery procedures when policy services are unavailable or validation fails, including checks against restoring known-vulnerable configurations.

Build Provider Security Requirements into Vendor Management

All that technical work should feed into provider selection, contracting, onboarding, and ongoing vendor oversight.



Document the Requirements Before Committing

Push for a service-specific security schedule or an equivalent contractual attachment. Spell out the service, deployment model, relevant provider dependencies, and how the responsibilities get divvied up.

This will make it easier to negotiate:

- Which hardening baselines and versions apply, how updates get adopted, and how exceptions are approved and reviewed.
- How much configuration-monitoring coverage exists, how often it's validated, and how reporting and escalation work when something fails.
- Remediation timeframes tied to risk, what interim protections look like, and what evidence is required to close out a finding.
- What triggers a notification (including the incidents and changes that affect agreed isolation, administrative access, data handling, or control coverage).
- What evidence gets delivered, what assessment or audit rights are proportionate, and what rules protect both other tenants and the provider's own sensitive information.
- Availability and recovery commitments that suit the service model, including what the customer is responsible for and proof of recovery testing.
- Exit terms (including data return or deletion, credential revocation, decommissioning evidence, and what happens contractually in the event of unmet obligations).

Agree to measurable terms built around the workload rather than a universal deadline or blanket access to provider systems. Any contractual agreement should describe the required outcome and evidence; it can't establish whether a control is implemented or working.

Ask for Implementation and Operating Evidence

Map each requirement to evidence covering the actual service being purchased. Record the baseline version, assessment date or period, asset coverage, test method, exclusions, and unresolved findings. A useful evidence package should distinguish:

Design	architecture diagrams, responsibility assignments, and approved baselines that show the intended configuration and who owns it.
Implementation	configuration exports, policy results, or baseline-compliance reports show whether settings were applied across the covered assets.
Enforcement	authorized isolation tests, access tests, and reassignment checks show whether the intended boundaries hold under the tested conditions.
Continued operation	configuration-monitoring coverage reports, drift alerts, investigation records, and remediation retests show how deviations are detected and resolved over time.

Say a requirement restricts BMC access to provider administrators. It's important to ask for evidence of the configured access paths, an authorized test from the tenant network, and proof that a relevant configuration change triggers a reviewed alert or validation failure. Where continuous monitoring is unavailable, agree on an alternative check and record the remaining exposure.

Evidence can come through redacted records, a controlled review, or an independent assessment of the relevant service and controls. Just make sure to double-check the scope and exclusions. A policy statement, a single screenshot, or a certification on its own can't close out a requirement for demonstrated configuration and monitoring.

Carry the Review into Operations

Agree on an evidence-review cadence based on workload risk. Additional review should occur after any changes that affect the security boundary. Keep track of gaps in baseline coverage, overdue remediation, unresolved exceptions, and whether reports arrive on time. Then review it all again at renewal time.

If evidence isn't available or a requirement can't be met, it's important to record the uncertainty and its compensating controls, owner, and when you next review the condition. From there, it's the business or service owner's call whether to accept the exposure, restrict the workload, pick a different service model, or hold off on deployment. Vendor oversight will keep that decision on record, so an unanswered questionnaire doesn't become an assumed control.

Proving the Boundary Holds

Infrastructure hosted by a neocloud provider still needs hardening, full stop. The service model just determines which layers the provider has to secure and which remain the customer's responsibility.

Review needs to run from firmware and operating systems all the way through networks, orchestration, applications, and data, including all layers that can't be directly configured by the customer. Figure out the required functionality, the exposure you're willing to permit, and your security baseline before you do anything else.

Hardening cuts down on unnecessary exposure and keeps required configurations in place. Zero Trust principles should guide decisions about trust, access, and ongoing verification, while implemented controls enforce those decisions. A working service, an approved change, or a signed contract doesn't mean that the intended protections are effective; any of those can be true and the boundary can still fail. Configuration evidence, boundary testing, and monitoring prove that these protections are implemented and that they stay that way via updates, automation, and tenant reassignment.

In the end, it comes down to whether implemented protections reduce risk to an acceptable level while still meeting workload requirements. Assign ownership, document provider obligations, and maintain evidence throughout the service lifecycle. If evidence is incomplete, record that, and then decide what to do next, whether it's resolve the gap, restrict the workload, or explicitly accept the risk.



AHEAD

Combining cloud-native capabilities in software and data engineering with an unparalleled track record of modernizing infrastructure, we're uniquely positioned to help accelerate the promise of digital transformation.

Visit us at ahead.com.

Global Hub Offices

CHICAGO

444 W. Lake Street
Suite 3000
Chicago, IL 60606
United States

NEW YORK

500 5th Avenue
17th Floor
New York, NY 10010
United States

UNITED KINGDOM

310 Wharfedale Rd
Winnersh, Wokingham
RG41 5TP
United Kingdom

NETHERLANDS

Kerkenbos 10-123
6546 BJ Nijmegen
The Netherlands

INDIA

L-21, One Horizon Center
Golf Course Road, DLF Phase V
Sector 43, Gurugram 122002
India