AVIATRIX

SECURITY ARCHITECTURE BRIEF

Secure Enterprise AI Chat

Threat model, enforcement architecture, and compliance evidence for LibreChat on Kubernetes

Powered by Aviatrix — LibreChat on Kubernetes

EXECUTIVE SUMMARY

Enterprises stand up self-hosted AI chat so sensitive data stays in their environment. LibreChat, the most widely deployed open-source enterprise chat platform, delivers that promise at the application layer – single sign-on, role-based access control, customer-managed infrastructure. It does not ship with an enforcement layer beneath the application, and Kubernetes networking provides none by default. The result: any large language model (LLM) provider an administrator configures can be called immediately, any Model Context Protocol (MCP) tool server can reach any internet destination, and any compromised open-source dependency can open an outbound socket to an attacker-controlled host.

The Aviatrix Validated Containment Architecture for LibreChat is a policy overlay that attaches to an existing Aviatrix-secured cluster and enforces default-deny Communication Governance across the chat stack – LLM provider egress locked to approved providers, each MCP server scoped to its authorized interfaces, the conversation data store isolated, and container image pulls restricted to known registries. Enforcement occurs at the network, below the application trust boundary, so a compromised dependency cannot disable it. The network policy is derived from LibreChat's own configuration by a continuous integration (CI) shim and reviewed in the same pull request, so it cannot drift behind the application.

The strategic outcome is a bounded Blast Radius: a compromised chat stack can reach only what its own configuration permits. This brief gives the security architect what is needed to approve the control before procurement: the full threat model, the kill chain and point of intervention, how enforcement works, how it deploys, what the control explicitly does not do, and the compliance evidence it produces.

Threat Model

The threat is the LibreChat stack reaching a destination it should not: to send conversation data to an unapproved LLM provider, to let an MCP tool server pivot to the open internet, or to let a compromised dependency exfiltrate to an attacker-controlled host. The application is the vehicle; the reachable destination is the target.

Threat property	Why it defeats application-layer controls
Provider sprawl by configuration	The set of providers the deployment can reach is whatever anyone with admin access has configured. There is nothing between that configuration and the open internet.
Unconstrained MCP egress	An MCP tool server can reach any destination by default. A single tool server becomes a path to the rest of the internet.
Open-source supply chain	LibreChat has a large, community-maintained dependency tree. A tampered image, poisoned package, or hijacked chart gets code execution inside the stack and beacons outbound.
No layer beneath the app	Single sign-on, role-based access control, and the application's own controls all sit at or above the application. None of them governs what a compromised component reaches on the network.

WHY THIS MATTERS

A compromise below the application does not announce itself – it gets code execution inside the chat stack and attempts to reach an attacker-controlled host. No application-layer control catches a compromise beneath itself. The control that holds evaluates the destination rather than the caller's identity, so it blocks exfiltration regardless of which process initiated it. That control has to live at the network, beneath the application the attacker may already be inside.

Attack Scenario and Kill Chain

The kill chain below traces a supply-chain compromise of the LibreChat stack and identifies where containment intervenes.

Kill Chain Stage	What Happens	Control
Dependency compromise	A tampered container image or poisoned transitive dependency is pulled into the chat stack through the normal supply chain.	Out of scope for egress control – the registry allow-list limits pulls to known registries; containment assumes a compromise may still occur.

Code execution in the stack	The compromised component runs inside a LibreChat pod with the pod's privileges.	Not prevented – containment acts on reachability, not on local execution.
Outbound beacon	The component opens a socket to an attacker-controlled host to exfiltrate data or receive instructions.	DENIED. Default-deny egress blocks any destination not on the approved list, before the socket completes, and logs it.
Unapproved provider call	An administrator adds, or a component invokes, an LLM provider that is not approved.	DENIED. Only providers declared in the configuration are reachable; the call is blocked and surfaced.
Audit trail	Responders need to know what was attempted and whether it was stopped.	CoPilot logs every allowed and denied connection with source pod identity, destination, and timestamp; URL paths when decryption is active.

POINT OF INTERVENTION

Aviatrix intervenes at the outbound connection – the one stage where the compromised component must traverse the network to succeed. LibreChat pods are scoped by a SmartGroup keyed to pod identity; their permitted destinations are defined as WebGroups; the Spoke Gateway evaluates every outbound connection against a default-deny posture before it leaves the cluster. A beacon to an attacker-controlled host or a call to an unapproved provider hits a deny rule regardless of how the compromise entered. The component runs; the exfiltration does not.

Enforcement Architecture

Enforcement is Communication Governance at the network layer, built on the Aviatrix Distributed Cloud Firewall (DCF). The architecture is a policy overlay on an existing Aviatrix-secured cluster; no new network infrastructure is provisioned and no application code changes are required.

Policy derived from configuration

A CI pipeline shim reads LibreChat's configuration file – which already declares the LLM providers and MCP servers the deployment uses – and generates the exact Aviatrix Kubernetes custom resource definition (CRD) manifests that encode those declarations as policy: WebGroups for approved providers, per-server allow-lists for MCP servers, a registry allow-list, and a default-deny base. Adding a provider produces a new rule in the same pull request; removing one removes the rule. Policy and configuration stay in lockstep instead of drifting as two separately owned files.

Identity-based, below the application

SmartGroups are keyed to Kubernetes pod identity – labels and namespaces – so a rule reads "LibreChat pods may reach approved providers" and survives pod restarts, scaling, and MCP sidecar churn without manual updates. Because enforcement is at the network, beneath the application trust boundary, a compromised dependency inside a pod cannot disable it.

Transparent decryption and the inspection insertion point

On the current Controller release, the gateway decrypts permitted LLM provider traffic inline by loading the Aviatrix certificate into the LibreChat container through a configuration mount – no image rebuild – giving Uniform Resource Locator-path visibility in CoPilot. Decryption is scoped only to permitted connections. That decrypted path is the insertion point for Aviatrix AgentGuard inspection (prompt-injection detection, output classification, tool-argument validation), enabled by configuration as those capabilities ship, with no architectural change.

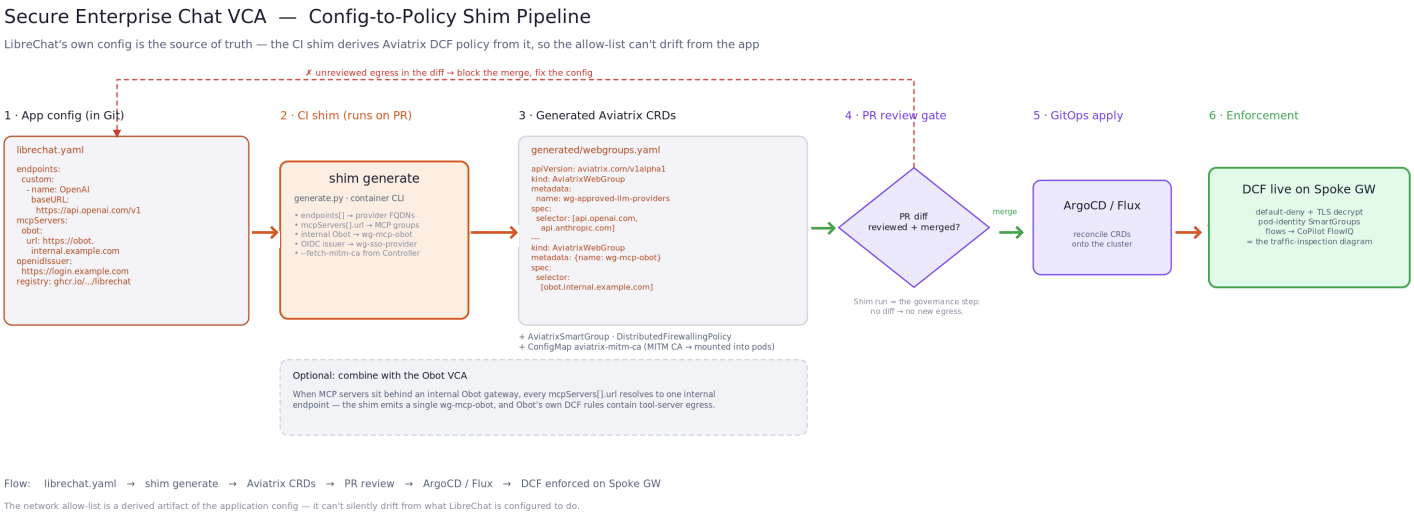
> **WHY THIS IS DIFFERENT FROM AN AI GATEWAY OR NETWORK POLICY**
>
> A software-as-a-service AI gateway governs model calls at the application layer but cannot contain what a compromised dependency reaches at the network layer – and it adds a new external dependency in the egress path, undercutting the reason the enterprise chose self-hosted chat. Hand-rolled Kubernetes network policy is address-based, does not survive pod restarts, and has no domain-aware filtering. This architecture is pod-identity-aware, domain-filtered, default-deny policy built from primitives the team already controls, kept honest by the CI shim.

Deployment Paths

The architecture deploys as a policy overlay on a standard Aviatrix Distributed Cloud Firewall footprint, with no new network infrastructure and no application code changes. It attaches to an existing Aviatrix-secured cluster on any major cloud:

- Amazon, Azure, or Google Kubernetes (EKS, AKS, or GKE) – the policy layer is cloud-agnostic Aviatrix custom resource definitions and Terraform; the only cloud-specific parameter is the container registry allow-list.
- Existing Aviatrix Spoke Gateway – the overlay attaches to a cluster already under Aviatrix coverage; no new gateways are provisioned.

The shim runs at continuous integration time and generates the policy as infrastructure-as-code, reviewed in the same pull request as the application change. Server Name Indication-based domain filtering is available on the current general-availability Controller; transparent Transport Layer Security decryption and the AgentGuard insertion point are available on the latest release. Deployment completes in under twenty minutes; step-by-step prerequisites and policy structure are in the companion Technical Brief.



Architectural Boundaries

Containment governs reachability – what the chat stack can reach on the network. Stating what it does not do is part of approving it.

Out of Scope	What governs it instead / status
Prompt and output content inspection	Prompt-injection detection, output classification, and tool-argument validation are delivered by Aviatrix AgentGuard, which plugs into the decryption insertion point this architecture installs as those capabilities ship.
URL-path enforcement	Transparent decryption provides URL-path visibility today; URL-path enforcement (permitting one path while denying another on the same domain) is pending Uniform Resource Locator WebGroup general availability.
Application-layer access control	Single sign-on and role-based access control remain the application's responsibility. This control governs the network beneath them.
In-cluster lateral traffic semantics	The architecture governs egress and component isolation; deep east-west application semantics remain with the application and platform controls.

Compliance Evidence

For SOC 2, HIPAA, PCI-DSS, FedRAMP, and NIST Secure Software Development Framework environments, auditors require architectural proof of enforcement – not a policy document. This architecture produces evidence as a by-product of how it runs.

Evidence Artifact	What It Proves
Default-deny policy definition	Chat-stack egress is restricted to explicitly approved providers and registries (PCI-DSS outbound restriction; FedRAMP least-functionality).

Per-connection flow logs	Every allowed and denied connection is logged in CoPilot with source pod identity and destination, plus URL paths when decryption is active (SOC 2 audit logging; FedRAMP continuous monitoring).
Per-component isolation	MCP servers are scoped to authorized interfaces and the conversation data store is isolated from the internet (HIPAA segmentation for protected data).
Config-derived policy history	The shim derives DCF objects from the LibreChat configuration, reviewed in the same pull request – an auditable, drift-resistant approval trail (SOC 2 change management; NIST Secure Software Development Framework).
Supply-chain mitigation record	The registry allow-list plus default-deny is a documented control against compromised dependencies beaconing outbound.

The proof of enforcement is the architecture itself – not a statement about the architecture.

Next Steps

Request a 30-minute architecture review.

We run the shim against your actual LibreChat configuration, show you the network policy it derives, and surface any provider or tool path you did not expect to be there.

About Aviatrix

Aviatrix® is pioneering the Cloud Native Security Fabric – the architecture the Containment Era requires. The Cloud Native Security Fabric governs every workload communication path across every cloud, every VPC, every Kubernetes cluster, and every serverless function, from a single policy plane. One rule. Universal propagation. Enforced at the workload, not at a chokepoint. Trusted by more than 500 of the world's leading enterprises. For more information, visit aviatrix.com.