WHITEPAPER · TECHNICAL ARCHITECTURE

Enterprise AI Security Architecture

A reference architecture for Zero Trust identity governance of AI agents and non-human identities, from local pilot to multi-cloud production.

Contents

01 Executive Summary

This whitepaper describes a reference security architecture for governing AI agents and non-human identities (NHIs) at enterprise scale, based on the Zero-Agent AI platform. It covers system context, container and module structure, runtime behavior, data architecture, integration boundaries, and the security architecture itself — including the STRIDE-based threat model — along with the deployment path from a local reference implementation to a multi-cloud production footprint.

The architecture is built around five goals: run the full governance lifecycle at minimal cost and dependency footprint; be cloud-portable so promotion to AWS, Azure, or GCP is a configuration change rather than a rewrite; be secure-by-default through Zero Trust and default-deny enforcement; keep every integration swappable behind a stable interface; and remain demonstrable and testable end to end.

> **WHO SHOULD READ THIS**
>
> Security architects, platform and infrastructure engineering leads, and technical reviewers evaluating the platform for a production security posture.

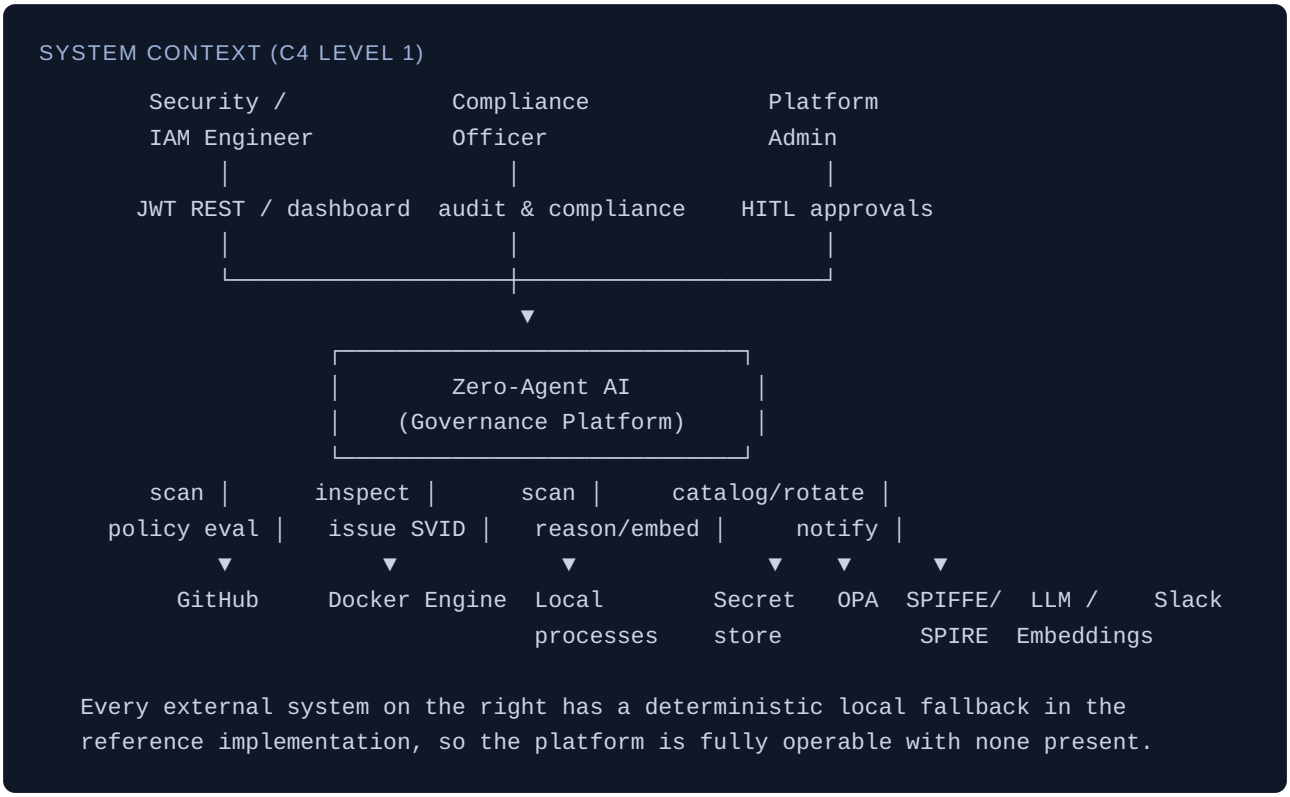
02 Architecturally Significant Requirements

The requirements below shaped every major structural decision in the architecture.

ID	REQUIREMENT	ARCHITECTURAL RESPONSE
ASR-1	Discover AI agents/NHIs from heterogeneous sources	Pluggable connector pattern
ASR-2	Explainable Zero Trust authorization	Policy-as-code (OPA) plus an explainable, additive risk model
ASR-3	Immutable, tamper-evident audit written before action	Append-only, hash-chained log; write-before-act discipline
ASR-4	Human approval for high-blast-radius actions	HITL queue with a tiered approval state machine
ASR-5	Zero external dependency required to run	Deterministic local fallback behind every integration
ASR-6	Cloud portability	12-factor configuration; interface/factory boundaries
ASR-7	Sub-100ms policy evaluation, sub-300ms API (p95)	Async I/O; policy caching; co-located data
ASR-8	Least-privilege access for humans and workloads	RBAC for humans; SPIFFE-based identity for workloads

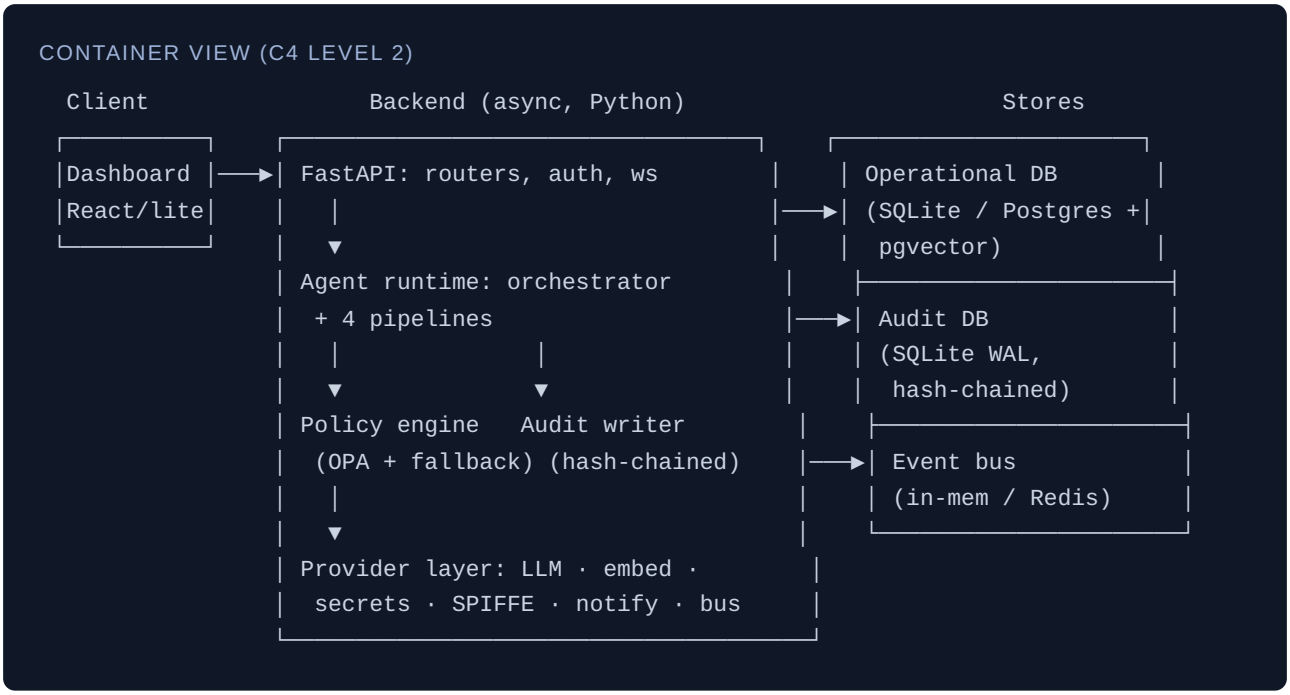
03 System Context

At the highest level, the platform sits between the humans who operate it and the systems it governs.



04 Container View

Inside the platform boundary, responsibilities are split across a small number of well-defined containers.



- **Dashboard** — a single-page application (full React build or a no-build "lite" variant) communicating over JWT-secured REST and WebSocket.
- **FastAPI application** — the HTTP/WebSocket surface: authentication, authorization, request-ID middleware, and route handlers.
- **Agent runtime** — an orchestrator chaining four pipelines (Discovery, Lineage, Policy, Remediation). Runs in-process in the reference build; the design maps directly onto distributed task workers (e.g. Celery) in production.
- **Policy engine** — an OPA REST client paired with a behaviorally identical local fallback evaluator.
- **Audit writer** — an append-only, hash-chained store, physically separate from operational data.
- **Provider layer** — the single boundary through which every external integration is accessed.

05 Module and Layering View

Dependencies point in one direction only — downward — which keeps the security-sensitive lower layers free of upward coupling to presentation concerns.

LAYER	RESPONSIBILITY	REPRESENTATIVE MODULES
L5 — Surface	API and dashboard surface	API routers, security/deps, websocket, dashboard app
L4 — Policy	Authorization and identity issuance	Policy engine (OPA client + Rego), secrets provider, workload identity provider
L3 — Data	Persistence and eventing	Database models/session, audit writer, event bus
L2 — Intelligence	Agent reasoning and orchestration	Orchestrator, pipeline, discovery/lineage/policy/remediation agents, LLM/embedding providers, guardrails
L1 — Discovery	External inventory collection	Connectors: source control, container runtime, process, secret catalog

Layering rule: dependencies point downward (Surface → Intelligence → Data/Policy → Providers). The provider layer is the only place aware of whether an integration is a real service or a local fallback — no business logic branches on environment.

06 Runtime View: Key Scenarios

Governance pipeline (discovery request)

A discovery request flows through Discovery → Lineage → Policy → Remediation, with audit written before every state-changing step:

```
SEQUENCE — FULL DISCOVERY RUN

API ──▶ Orchestrator: run_full_discovery()
Orchestrator ──▶ Connectors: parallel scan
Connectors ──▶ Orchestrator: raw entities
Orchestrator ──▶ DB: upsert agents (by entity hash)
Orchestrator ──▶ Audit: discover_agent (per agent)
Orchestrator ──▶ Event bus: discovery:completed
Orchestrator ──▶ DB: embeddings + inferred ownership
Orchestrator ──▶ Audit: infer_ownership
Orchestrator ──▶ Policy engine: evaluate(agent, request)
Policy engine ──▶ Orchestrator: allow / needs_approval / deny + reason
Orchestrator ──▶ Audit: evaluate_authorization   (BEFORE the finding is recorded)
Orchestrator ──▶ DB: risk_findings ; Event bus: risk:findings
Orchestrator ──▶ Audit: remediation:*            (BEFORE the action executes)
Orchestrator ──▶ API: summary { discovered, governed, findings }
```

Policy enforcement (authorization request)

Map the requesting agent to a policy-engine input (including its inferred role) → evaluate the decision → produce a guardrailed, human-readable rationale → write the audit record → return the decision. Audit is written first in every case, including for automatic allows.

Remediation and human-in-the-loop

Automatic remediation actions execute inline once approved by policy. Actions requiring human sign-off create a tiered approval record, publish to the HITL queue, trigger a notification, and resume execution once approved.

07 Data Architecture

STORE	PURPOSE	LOCAL IMPLEMENTATION	PRODUCTION TARGET
Operational store	Organizations, owners, agents, machine identities, policies, risk findings, HITL approvals, users	SQLite	Postgres with pgvector
Vector store	Behavioral embeddings for similarity and anomaly detection	JSON-stored vectors, Python cosine similarity	pgvector IVFFlat cosine index
Audit store	Append-only, tamper-evident governance evidence	Separate SQLite database, WAL mode	Same logical model on a managed append-only store
Event bus	Inter-agent signaling across six logical streams (discovery, risk findings, remediation, audit, HITL)	Bounded in-memory queue	Redis Streams or a managed equivalent

Secrets are never persisted as raw values in the operational store — only path references into the secret manager are recorded, keeping the blast radius of a database compromise limited to references rather than live credentials.

08 Integration Architecture

All external integrations are mediated by a single provider/adapter boundary, so every integration has exactly one place where "real" and "local fallback" implementations are selected.

CAPABILITY	INTERFACE	LOCAL IMPLEMENTATION	PRODUCTION IMPLEMENTATION
Reasoning (LLM)	Risk explanation provider	Deterministic mock	Production LLM (configurable provider)
Embeddings	Embedding provider	Deterministic hash-based mock (1536-d)	Production embedding model
Secrets	Secrets provider	In-memory key/value store	HashiCorp Vault (KV v2)
Workload identity	Workload identity provider	Mock SVID issuance	SPIFFE/SPIRE
Notifications	Notification provider	Local ring buffer	Slack webhook or equivalent
Event bus	Event bus interface	In-memory queue	Redis Streams
Policy	Policy client	Python fallback evaluator	OPA server

Enterprise IGA/PAM/Zero Trust connectors (e.g. SailPoint, Saviynt, Okta, Entra, CyberArk, Zscaler) plug into this same boundary as additional discovery connectors and provider adapters, without requiring changes to the orchestration or policy layers.

09 Security Architecture

Authentication and authorization

- Human authentication via OAuth2 password grant issuing a JWT (HS256 for local reference deployments, RS256 recommended in production), with an algorithm allowlist that never accepts "none," expiry enforcement, and bcrypt password hashing at cost factor 12.
- Role-based access control enforced per route (platform_admin, security_engineer, iam_engineer, compliance_officer, ai_platform_engineer roles).
- Every governed agent is issued a workload identity (SPIFFE SVID in production); default-deny policy requires identity, an owner, and an acceptable risk score to allow any action.

Audit integrity

Every audit record is written before the action it describes executes; records are append-only, SHA-256 hash-chained, and stored in write-ahead-log mode with restrictive file permissions. A chain-verification routine can independently detect tampering across the full history.

Secrets hygiene

Only secret-manager path references are persisted in the operational database — never raw values. Secret rotation generates new material server-side and is never returned in an API response.

Trust boundaries

Connector and LLM output are treated as untrusted input: guardrails sanitize and validate model output before use, and neither connector data nor model output is ever passed directly into SQL, system, or secret-manager calls.

Network

An explicit CORS origin allowlist governs browser access; internal services are intended to run on a private network in production, fronted by a load balancer / WAF and reverse proxy.

STRIDE threat model summary

THREAT CATEGORY	PRIMARY MITIGATION
Tampering	SHA-256 hash chain over the audit log
Repudiation	Per-actor audit records for every decision
Information disclosure	RBAC enforcement; no raw secret storage
Elevation of privilege	Default-deny policy plus per-route role checks

Denial of service	Async I/O; rate limiting at the reverse proxy layer in production

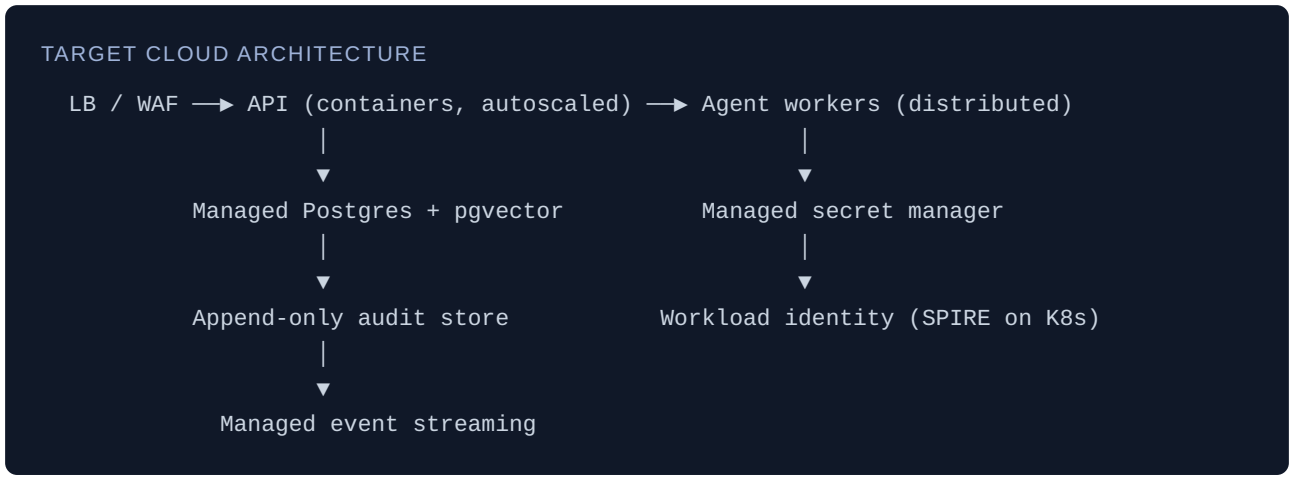
REFERENCE IMPLEMENTATION CAVEAT

The identity, secret, and cryptography providers in the local reference build are deterministic mocks, not hardened production services. Authentication, cryptography, IAM integration, and remediation code paths require a full security review before any deployment that touches real credentials or regulated data.

10 Deployment Architecture

Reference (local) deployment

Target cloud deployment



LOCAL COMPONENT	AWS	AZURE	GCP
SQLite / Postgres	RDS Postgres	Azure Database for Postgres	Cloud SQL
Event bus (Redis)	SQS / SNS	Service Bus	Pub/Sub
Audit store	DynamoDB	Cosmos DB	BigQuery
Distributed workers	ECS Fargate	Container Apps	Cloud Run Jobs
Secrets manager	Secrets Manager	Key Vault	Secret Manager
Workload identity	SPIRE / EKS	SPIRE / AKS	SPIRE / GKE
Edge / load balancing	ALB + WAF	App Gateway	Cloud Armor + LB

11 Quality Attributes and Architecture Decisions

Quality-attribute scenarios

ATTRIBUTE	SCENARIO	TACTIC	TARGET
Performance	Policy evaluation under load	Async execution, caching, local fallback	< 100ms p95
Performance	Full discovery scan	Parallel connector fan-out	< 60s p95
Security	A tampered audit row	Hash-chain verification	Detected 100% of the time
Availability	An external dependency is unreachable	Deterministic local fallback	Platform stays operable
Portability	Moving to a cloud environment	Configuration-only swap	No code change required
Auditability	Reconstructing a past decision	Write-before-act evidence	100% coverage
Modifiability	Adding a new discovery connector	Implement the connector contract	Localized change, no core rewrite

Key architecture decisions

- **Provider/factory boundary for every integration.** Enables zero-dependency local operation and configuration-only cloud promotion, at the cost of a thin indirection layer that must be kept behavior-faithful between real and mock implementations.
- **A local policy evaluator mirroring the production policy engine.** Preserves enforcement semantics without requiring a running policy server locally; the two implementations are kept in sync through shared unit tests.
- **A dedicated, hash-chained audit store, separate from operational data.** Isolates compliance evidence from application data by design.
- **An explainable, additive risk model over a black-box classifier.** Trades some potential predictive accuracy for auditability and defensibility under explainability-focused regulation.

12 Risks and Mitigations

RISK	IMPACT	MITIGATION
Local and production policy evaluators drift apart	Inconsistent authorization decisions	Shared unit tests across both implementations; a single source of truth for policy weights
Mock providers mistaken for production-secure	Security gap in a real deployment	Explicit caveats; mandatory human security review before go-live
In-process orchestration limits throughput at scale	Ceiling on discovery/ enforcement volume	Design maps directly to distributed task workers for horizontal scale-out
Local vector similarity search scales linearly	Latency at large agent counts	Migrate to pgvector with an approximate-nearest-neighbor index in production
Secret-handling implementation errors	Credential leakage	Path-reference-only invariant enforced at the data-model level; no raw values ever surface via the API

13 Technology Summary

A representative technology stack for this architecture: an async-first web framework (e.g. FastAPI), an ORM with async support (e.g. SQLAlchemy 2.0), structured JSON logging, a policy-as-code engine (OPA/Rego), a relational database with vector-search support (Postgres + pgvector) or a local file-based equivalent, a streaming event bus (Redis Streams or a managed equivalent), workload identity via SPIFFE/SPIRE, a secrets manager (HashiCorp Vault or a cloud-native equivalent), JWT-based authentication with bcrypt password hashing, and a modern frontend stack (React/TypeScript) for the operator dashboard.

14 Summary

This architecture is designed so that the same governance logic — discover, own, enforce, remediate, audit — runs unchanged from a single-laptop pilot to a multi-cloud production deployment. The provider/adapter boundary, the write-before-act audit discipline, and the explainable policy engine are the three structural decisions that make that possible.

> **NEXT**
>
> See the companion *AI Identity Governance Framework* whitepaper for the governance model this architecture implements, and *AI Agent Governance Best Practices* for an operational rollout playbook.