Kognoesis Platform: Security and AI Governance Architecture

This document outlines the architectural and security controls implemented by the Kognoesis platform, specifically tailored to address enterprise IT, Cyber Security, Data Privacy, and AI Compliance requirements.

---

1. Client Environment Integration and Security Controls

Client Environment Integration Kognoesis is designed as a cloud-native platform deployed exclusively via the **Kognoesis Operator** into the client's own Kubernetes (K8s) cluster. It integrates securely with existing enterprise infrastructure by hooking into corporate
Identity Providers (IdP) like Keycloak or Azure AD (Microsoft Entra) via OIDC, and respects the client's existing network topologies (VPCs, subnets).

Container and Application Isolation The platform is built on a distributed, modular architecture:

- **kognoesis-ui**: Frontend Next.js application exposed to end-users.
- **kognoesis-api**: The core FastAPI backend handling logic, access control, and routing.
- **kognoesis-engine-v2**: Isolated background workers processing the Agent Action Graphs.
- **JIT Code Sandbox**: A hardened runtime environment (executing within the worker pods) for safely running untrusted Python and .NET script nodes with restricted host and network access.
- **Optional Local AI Components**: Dedicated, isolated inference containers (such as **Ollama** for local LLMs, **LLM Guard** for input/output scanning, and **kognoesis-local-inference** for embeddings) ensuring sensitive data never leaves the cluster.

Each component runs in isolated containers with strict Pod Security Contexts (e.g., `runAsNonRoot: true`, `readOnlyRootFilesystem: true`, and all unnecessary capabilities dropped).

Separation Between Environments Environmental segregation (e.g., Dev, UAT, Prod) is natively enforced at the Kubernetes level using distinct Namespaces. The Kognoesis Operator manages deployments via `KognoesisEnvironment` Custom Resource Definitions (CRDs), ensuring configurations remain completely isolated across environments.

Integrations and APIs Kognoesis is API-first. It exposes secured HTTPS REST APIs and Webhooks for integration. Outbound integrations (e.g., enterprise databases or SaaS platforms) are mediated through secure Agent Nodes, which authenticate using credentials stored in the platform's encrypted Secrets Vault.

Resilience, Availability, and Backup Kognoesis utilizes **Temporal** as its underlying execution orchestrator. This provides "Durable Execution" capabilities: if a worker pod crashes or the network drops during an Agent's execution, the state is persisted in Postgres. When the system recovers, the workflow resumes exactly where it left off without data loss. Standard volume snapshots (for Postgres, etcd, and MinIO/S3) ensure point-in-time recovery capabilities.

---

2. Cyber Security Controls

Network Access and Ingress / Egress Controls The Operator enforces a **Zero-Trust Network Perimeter** via strict L3/L4 Kubernetes `NetworkPolicy` objects:

- **Default Deny:** All incoming lateral traffic from other namespaces is blocked.
- **Database Isolation:** Core stateful services (Postgres, Redis, Temporal, and the Internal System Milvus) only accept connections originating explicitly from the authorized `kognoesis-api` and `kognoesis-engine-v2` pods.
- **Restricted Egress:** Outbound traffic to the public internet can be restricted. Egress to internal enterprise subnets (RFC1918) is strictly blocked unless explicitly authorized via `allowedInternalEgressNamespaces` (e.g., for OIDC discovery).

Rate Limiting and DoS Prevention All external entry points are protected against abuse. Public authentication endpoints (like `/v2/auth/login`) are protected against brute-force attacks via rate-limiting. Agent triggers (like webhooks) enforce strict **Requests-Per-Minute (RPM) Rate Limits** (configured via the deployment budget). If a malicious or runaway external system blasts a webhook with thousands of requests, the API gateway enforces this limit, immediately rejecting the overflow with HTTP 429 (Too Many Requests). This guarantees a "noisy neighbor" cannot overwhelm the underlying Temporal workers or exhaust LLM rate limits.

Encryption in Transit and at Rest

- **In Transit (mTLS):** By enabling the `internalTLSEnabled` flag, the Operator uses `cert-manager` to bootstrap a dedicated, self-signed Internal Certificate Authority (CA) strictly for the Kognoesis namespace. The Operator globally injects standard CA environment variables (e.g., `PGSSLROOTCERT`) across all HTTP and database clients in the backend. This enforces strict certificate validation and fully encrypts Temporal orchestration payloads (requiring mTLS client-certificate authentication), Postgres/Redis connections (`sslmode=verify-full`), and internal Milvus/etcd/MinIO communications. Furthermore, because local AI components like **Ollama** and **LLM Guard** do not natively support mTLS, the Operator automatically injects secure, **mTLS-terminating NGINX sidecars** into their pods to ensure zero-trust encryption is maintained for all local inference traffic.
- **At Rest:** Data stored in Postgres and MinIO utilizes the underlying infrastructure's native AES-256 encryption-at-rest (e.g., AWS EBS volume encryption, Azure Disk Storage encryption, or DigitalOcean Block Storage/Spaces encryption).

Authentication and Least-Privilege Controls Authentication relies on your enterprise OIDC provider. Within the platform, Kognoesis uses a robust 4-tier Organizational Unit Role-Based Access Control (RBAC) model. Kubernetes ServiceAccounts attached to Kognoesis pods adhere to the Principle of Least Privilege, avoiding Cluster-Admin roles.

Secrets / Credential Management Kognoesis provides a flexible, dual-layered approach to secrets management:

1. **Native Encrypted Vault:** By default, Kognoesis ships with a built-in Secrets Vault backed by Postgres. When users input sensitive API keys or connection tokens in the UI, they are symmetrically encrypted using an AES-256 `SECRETS_ENCRYPTION_KEY` (injected securely at runtime by the Operator) before ever touching the disk.
2. **External Enterprise KMS (HashiCorp Vault / Azure Key Vault):** For enterprise compliance, the Secrets Vault is fully abstracted via a Provider Interface. Kognoesis can be configured to delegate secrets management entirely to an external KMS. In this mode, the platform only stores logical *pointers* (Vault URIs). The execution engines use their Kubernetes ServiceAccount identities (via OIDC federation) to request the raw credentials directly from the enterprise Vault exactly at the moment of execution, ensuring Kognoesis never permanently stores the plaintext keys.
3. **Operator-Managed Infrastructure Secrets:** Kognoesis does not rely on static default credentials (e.g., `minioadmin` or `root`/`Milvus`). The Operator automatically generates cryptographically secure, random credentials for Postgres,

Redis, MinIO, and Milvus during deployment. These are persisted exclusively in encrypted Kubernetes Secret objects and injected into the application pods at runtime via secretKeyRef mappings.

Dependency and Open-Source Security For strict supply chain security, the platform supports offline or air-gapped package management. Frontend (.npmrc), Python (pip.conf), and .NET (nuget.config) dependencies can be redirected to internal, pre-vetted package registries (e.g., JFrog Artifactory, Sonatype Nexus). Automated SAST scanning (npm audit, bandit) and Container scanning (Docker Scout) are integrated into our CI/CD pipelines, and all Docker base images are strictly pinned to immutable SHA256 digests.

---

3. Data and Privacy

Client Data Access and Data Federation (Knowledge Bases & Document Stores) Kognoesis only accesses data explicitly provided to it by the client. The platform utilizes a **Data Federation Architecture** to cleanly separate logic from enterprise data:

- **System Layer**: The Kognoesis Operator deploys a foundational *System Milvus* instance for core platform telemetry and default agent vectors, alongside a *System MinIO* for internal UI assets and temporary sandbox artifacts (both secured by Operator-injected credentials and TLS).
- **Federated Knowledge Bases**: When clients configure a new "Knowledge Base", they define a logical binding. This allows the client to connect Kognoesis to an entirely *external, client-managed* enterprise Milvus cluster (Vector Store) residing in a separate, highly secure VPC.
- **Federated Datasets**: Structured flat tables (e.g., CSV or JSON records) are managed as Datasets. Instead of storing these within the internal system database, clients configure a Postgres Data Connection pointing to their own external enterprise Postgres instance, ensuring full client ownership and isolation of the underlying structured data.
- **Federated Document Stores**: Enterprise files (PDFs, raw data blobs) are managed via Document Stores. The platform centralizes document metadata and indices securely in Postgres. For the actual blob storage, clients can utilize the in-built Kognoesis Document Store data connection (which stores blobs in the database), or they can achieve true data federation by creating their own custom Document Store data connection, binding it to an external S3 Bucket or Blob Storage.
- **Secure Runtime Binding**: The UI binds external Vector Stores and Document Stores to Kognoesis Secrets. The Agent only ever sees a logical "Knowledge Base ID" or "Document Store ID". At runtime, the Engine resolves these IDs to their target URIs and retrieves the required credentials from the Secrets Vault to authenticate securely via IAM or access keys.

Data Residency and Storage All data processing and storage happens **entirely within the client's own infrastructure** (Postgres for relational state, Milvus for vector embeddings, MinIO/S3 for object storage). Kognoesis (the vendor) does not have access to, nor retains, any client data.

Automated Retention and Deletion The platform enforces configurable Time-To-Live (TTL) policies via global settings (e.g., *Data*
Access Retention and *Execution Trace Retention* limits in days). This ensures that historical Agent execution logs, LLM traces, and Data Access Audit logs are automatically and systematically purged from the Postgres database to comply with GDPR/CCPA data minimization mandates without requiring manual intervention.

4. AI-Specific Controls

Permitted LLMs and Provider Controls Kognoesis is model-agnostic. Clients have absolute control over which models are permitted. Administrators configure approved models in the DependencyContainer. You can utilize:

- **Public APIs** (OpenAI, Anthropic) via Enterprise tiers with zero-data-retention agreements.
- **Private Cloud Deployments** (Azure OpenAI, AWS Bedrock) tied to your VPC.
- **Locally Deployed Models** (Llama 3, Mistral) running directly inside your cluster via the kognoesis-local-inference service, ensuring data *never* leaves your network.

Information Flow and Model Training Only the specific textual context explicitly retrieved by the Agent's workflow is sent to the LLM. If utilizing commercial LLM APIs, Kognoesis configures the connections to mandate zero-retention policies, ensuring your data is **never** used by the provider for model training.

AI Middleware (Guardrails & DLP) Kognoesis provides specialized Guardrail Nodes that can be injected into any agent workflow to enforce strict security boundaries around inputs and outputs:

- **Inbound Scanner (sys.guard.inbound)**: Operates as a security and compliance middleware for your AI agents. It scans incoming user prompts for malicious intent, prompt injection attacks, jailbreak attempts, or toxic content before the LLM ever processes them.
- **Outbound Scanner (sys.guard.outbound)**: Operates as a critical DLP (Data Loss Prevention) and security middleware. It scans the LLM's generated response and strips or redacts sensitive PII (Personally Identifiable Information), PCI, or PHI before it leaves your network and is returned to the user.

Human-in-the-Loop (HITL) and Interactive Workflows Kognoesis natively supports deep Human-in-the-Loop controls to ensure AI autonomy is strictly governed:

- **Approval Nodes**: High-risk actions (e.g., executing a financial transaction or sending an email) can be routed through a sys.tools.human_review node. This suspends the Agent's execution and pushes a pending task to an authorized user's Inbox, requiring explicit cryptographic approval to proceed.
- **Multi-Step Guided Workflows**: For complex business processes, the platform's Workflows feature enables the orchestration of multi-step agent pipelines punctuated by dynamic forms. This allows human operators to iteratively review agent outputs, modify data, and approve each individual step of a complex process via a centralized dashboard before the agent continues to the next phase.

Auditability and Agent Restrictions Every thought, tool call, and decision made by an Agent is deterministically logged in the
GraphExecutionState. This provides a complete, forensic audit trail of why an Agent took a specific action. Additionally, custom Python/agent scripts execute within a hardened JIT Python Sandbox that prevents unauthorized host system or network access, mitigating the risk of compromised or hallucinating agents attempting lateral movement.

Agent Identity, Node-Level Authorization, and LLM Secret Isolation Agents do not have blanket access to the Secrets Vault. They operate under strict **Node-Level Authorization**:

- Agents inherit the Authorization Scope of the Project or Organizational Unit they reside in.

- The JIT Sandbox enforces authorization via a cryptographically signed execution token (`KOGNOESIS_EXECUTION_TOKEN`). When a script requests a credential, the backend engine validates this token to ensure that *only* the specific secrets explicitly granted to that exact node by the builder can be resolved and injected at runtime. An agent cannot perform lateral movement to steal another department's API keys.
- **LLM Secret Isolation**: Crucially, sensitive credentials (like Salesforce API tokens or Database passwords) are resolved *exclusively by the backend execution engine* when making API calls or running sandbox scripts. **The LLM never sees the secrets**. It only receives the resulting contextual data (e.g., the JSON response from the API), ensuring secrets are never accidentally leaked into model context windows or prompt histories.

www.kognoesis.com

