Architecting a GenAI-Powered Digital Social Support Platform

1. Executive Summary & Strategic Vision

The government has launched a strategic initiative to transform its social support services, moving from a slow, manual, and paper-based system to a citizen-centric, AI-driven digital platform. The core vision is to provide rapid, fair, and transparent financial and economic enablement to residents in need. The current application process, which takes up to **20 working days**, is a significant bottleneck that erodes public trust and delays critical support.

As a lead architect, you are tasked with designing the enterprise-grade solution architecture for this new platform. Your mission is to create a secure, scalable, and resilient system that leverages Generative AI to automate **up to 99% of social support application decisions**, reducing the processing time to mere minutes. This is not just an IT project; it is a flagship digital transformation initiative with high public visibility, aimed at setting a new global standard for public sector service delivery.

2. Current State: The "As-Is" Architecture

The existing process is plagued by deep-seated architectural and operational issues, leading to significant inefficiencies and poor citizen experience.

- **Data Silos & Manual Ingestion:** Data is fragmented across physical documents, handwritten forms, and various disconnected government offices. The process relies on manual data entry from scanned documents, which is error-prone and time-consuming.
- **Lack of Integrated Validation:** The system performs only basic field validations. Critical data consistency checks—such as verifying addresses against credit reports or income across different documents—are performed manually by case officers, leading to delays.
- **Fragmented & Inefficient Workflows:** Applications undergo multiple, sequential reviews across different departments. This linear and disjointed process creates bottlenecks and significantly extends the review cycle.
- **Inconsistent & Biased Decision-Making:** The reliance on manual assessment introduces human subjectivity and bias, leading to inconsistent and potentially unfair outcomes for applicants with similar circumstances.

3. The Challenge: Your Architectural Mandate

Your task is to design a comprehensive, end-to-end "To-Be" architecture for the new AI-powered Social Support Platform. Your design must address the pain points of the current state and meet the functional and non-functional requirements of a modern, enterprise-grade system.

3.1. Core Functional Requirements

The solution must be able to:

- **Ingest Multimodal Data:** Process a variety of data sources including interactive web forms and user-uploaded attachments like IDs, bank statements (PDFs), resumes (DOCX/PDF), and asset/liability declarations (Excel files).
- **Perform AI-Driven Eligibility Assessment:** Automatically assess applications against key criteria: income level, employment history, family size, wealth, and demographic profile.
- **Generate Decision Recommendations:** Provide clear, explainable recommendations to a human case officer to **approve** or **soft decline** applications for financial support.
- **Provide Economic Enablement Pathways:** For all applicants, recommend personalized economic support such as upskilling courses, job matching, and career counseling services.
- **Enable Interactive Citizen Experience:** Allow applicants to interact with the system via an intelligent, conversational chatbot for status updates and queries.

3.2. Critical Non-Functional Requirements (NFRs)

As an architect, your design must explicitly address the following NFRs:

- **Scalability:** The platform must be designed to handle a 10x increase in application volume over the next two years without performance degradation.
- **Performance:** The AI decision-making workflow must complete within 2 minutes for 95% of applications. Chatbot responses should be near real-time (<2 seconds).

- **Security & Compliance:** The architecture must ensure end-to-end security. All sensitive citizen data must be encrypted at rest and in transit. Implement role-based access control (RBAC) and design for compliance with national data protection regulations.
- **Reliability & Availability:** The system must be highly available (99.9% uptime) with a clear strategy for disaster recovery and business continuity.
- **Maintainability & Extensibility:** The solution must be modular, allowing for easy updates and the addition of new features (e.g., new support types, integration with new government services) without significant rework.
- **Observability:** The design must include a comprehensive monitoring and observability strategy using tools like **Langfuse** to track system health, API performance, and the end-to-end behavior of AI agents.

3.3. Enhanced Architectural Dimensions

Your architecture must provide a detailed perspective on the following four key areas:

A. Infrastructure, Deployment, and Operations (DevSecOps)

Your design must be deployable on a **hybrid cloud environment** (on-premises for secure data processing and public cloud for scalable, non-sensitive workloads).

- **Containerization & Orchestration:** Define a strategy using **Docker** and **Kubernetes** for packaging, deploying, and managing microservices.
- **CI/CD Pipeline:** Design a robust CI/CD pipeline (e.g., using GitLab CI/CD or Jenkins) that automates testing, security scanning (SAST/DAST), and deployment across environments.
- **Infrastructure as Code (IaC):** Specify how you would use **Terraform** or **Ansible** to provision and manage all infrastructure components for consistency and repeatability.
- **Monitoring & Logging:** Detail a centralized logging and monitoring solution using the **ELK Stack (Elasticsearch, Logstash, Kibana)** or **Prometheus/Grafana** to provide real-time insights into system health and performance.

B. Enterprise Security (Zero Trust Architecture)

Adopt a **Zero Trust** security model. Assume no implicit trust and continuously validate every stage of digital interaction.

- **Identity and Access Management (IAM):** Propose an IAM solution using **Keycloak** or integration with a federal identity service for single sign-on (SSO) and federated identity.
- **Data Security & Governance:** Define a comprehensive data classification scheme (e.g., Public, Internal, Confidential, Restricted). Specify encryption standards (e.g., AES-256 for data at rest) and key management practices using a tool like **HashiCorp Vault**.
- **API Security:** Secure all north-south and east-west API traffic. Your design must include an **API Gateway** (e.g., Kong, Tyk) to enforce authentication, authorization, rate limiting, and request validation.
- **LLM & AI Security:** Address specific AI security threats, including **prompt injection**, **model inversion**, and **data poisoning**. Propose mitigation strategies such as input sanitization, output filtering, and continuous model monitoring for adversarial attacks.

C. Advanced Data Integration

The solution must integrate seamlessly and securely with a variety of internal and external systems.

- **Real-time & Batch Integration:** Design patterns for both real-time API-based integration (e.g., with the National Credit Bureau) and asynchronous batch processing (e.g., nightly reconciliation with the Department of Economic Development).
- **Event-Driven Architecture:** Propose the use of a message broker like **Apache Kafka** or **RabbitMQ** to decouple microservices and enable a resilient, event-driven flow of information between the ingestion, validation, and decisioning components.
- **External Data Providers: Architect the integration with third-party services for:**
    - **Identity Verification:** National ID validation service.
    - **Financial Verification:** Secure integration with bank data aggregators (using Open Banking APIs).
    - **Economic Opportunities:** APIs from job portals and national upskilling platforms.

- **Data Lineage and Provenance:** Your design must ensure full data lineage. For any AI-generated decision, it must be possible to trace back to the exact source data and model version that influenced it.

D. Advanced AI & Agentic Solutioning

Go beyond a basic agentic workflow and design a sophisticated, explainable, and self-improving AI system.

- **Multi-Agent Collaboration: Design a sophisticated agentic system where specialized agents collaborate. For example:**
    - **Document Intelligence Agent:** Uses OCR and layout-aware models to extract structured data from complex PDFs and images.
    - **Fact-Checking Agent:** Cross-references extracted claims (e.g., income, address) against integrated data sources (credit bureau, bank statements) to identify discrepancies.
    - **Reasoning & Deliberation Agent:** Uses a **Plan-and-Solve (PaS)** or **Reflexion** framework. When discrepancies are found, this agent deliberates on the conflicting evidence, reasons about the most likely truth, and flags ambiguities for human review.
- **AI Explainability (XAI):** The recommendation from the AI must not be a "black box." Your architecture must produce human-readable explanations for its decisions, referencing the specific criteria and data points that led to an approval or decline recommendation.
- **Human-in-the-Loop (HITL) & Model Finetuning:** Design a workflow where low-confidence decisions or flagged discrepancies are routed to a human case officer via the **Streamlit** dashboard. The officer's validated corrections should be captured in a structured format and used as training data to continuously fine-tune the local ML/LLM models, creating a virtuous cycle of improvement.
- **Knowledge Graph for Reasoning:** Justify how a **Graph Database (Neo4j/ArangoDB)** will be used to build a knowledge graph of applicants, their families, assets, and relationships. Explain how this graph will be leveraged by the AI agents to uncover complex, non-obvious patterns and ensure consistent information across the entire application.

4. Your Deliverables

Your submission will be presented and defended in a one-hour session with the architecture review board. You must provide a **Solution Architecture Document**.

Solution Architecture Document (Max 10 pages, PDF)

This document is the primary deliverable and must include all previously mentioned sections, now enhanced with detailed designs for the four new dimensions:

* High-Level Architecture Diagram (C4 Model): Must now include infrastructure components, security boundaries, and key integration points.

* Infrastructure & DevSecOps Plan: Diagrams and explanations for your Kubernetes deployment, CI/CD pipeline, and IaC strategy.

* Zero Trust Security Architecture: Detailed diagram and explanation of the security measures, including IAM, data encryption, API gateway placement, and LLM firewalls.

* Data and Integration Architecture: A comprehensive diagram showing the flow of data via the event bus (e.g., Kafka) and integrations with all internal and external systems.

* Advanced AI Agentic Workflow: A detailed diagram illustrating the collaboration between the specialized agents, the reasoning framework (e.g., Reflexion), the role of the knowledge graph, and the Human-in-the-Loop feedback mechanism.

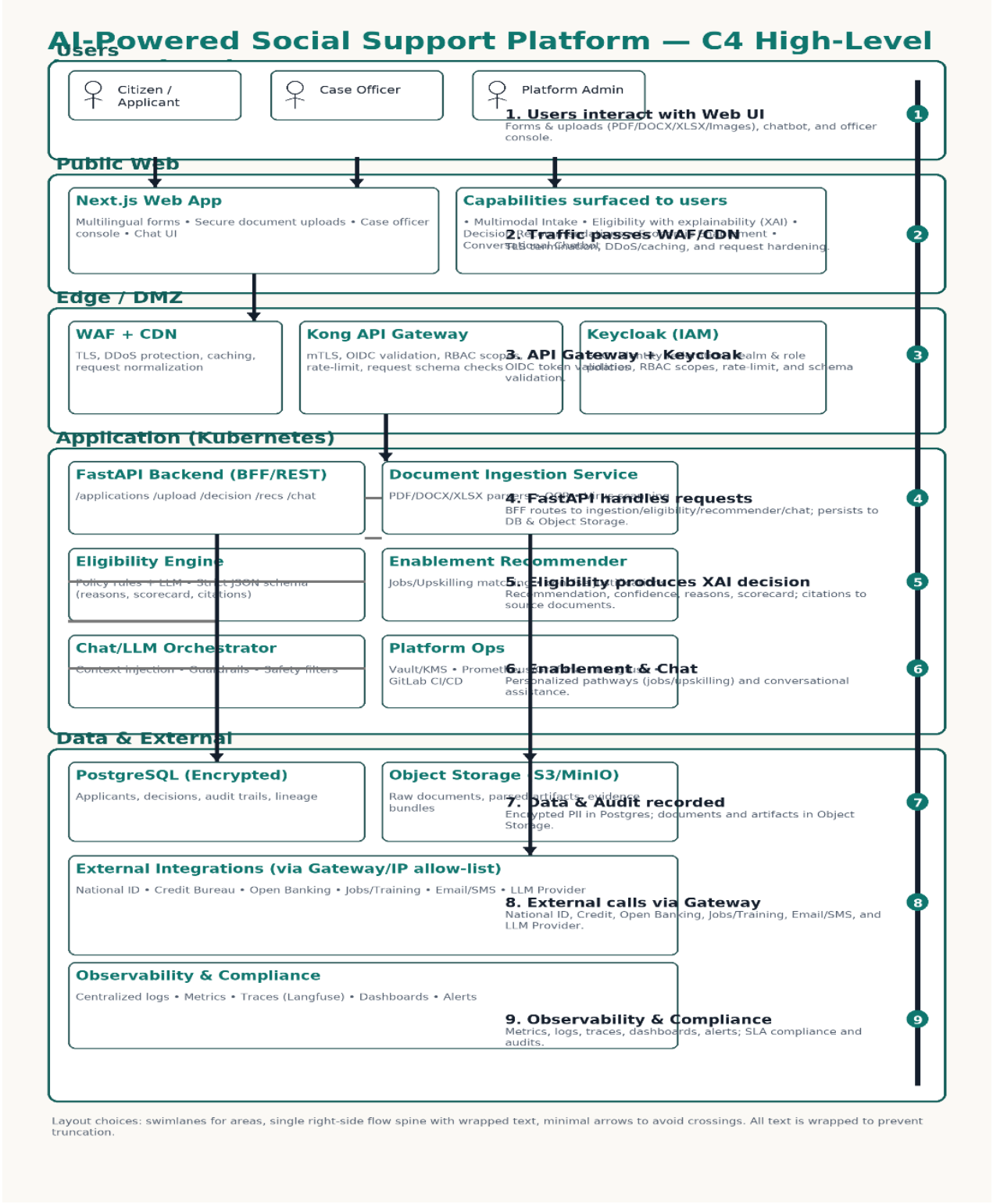
5. Evaluation Criteria

Your submission will be evaluated on the depth and rigor of your architectural thinking.

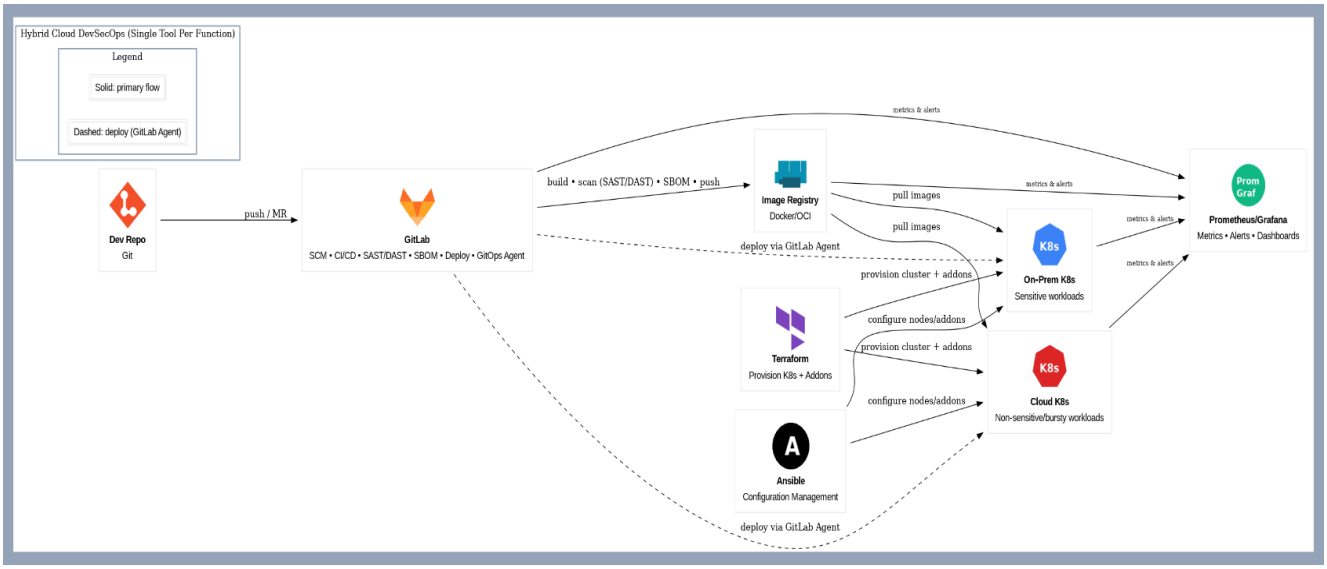
- **Architectural Design & Rigor:** Is the proposed architecture well-reasoned, scalable, secure, and aligned with the problem's complexity? Does it demonstrate a profound understanding of modern AI/ML principles and system design?
- **Technical Justification:** How well do you justify your technology choices? Do your decisions reflect a deep understanding of their trade-offs in terms of performance, cost, and maintainability?

- **Functionality & Completeness:** Does your proposed solution address all the core functional requirements and constraints outlined in the challenge?
- **Integration & Scalability:** Are the API designs and data pipelines effective and built for future growth?
- **Problem-Solving & Innovation:** How effectively did you address potential challenges (e.g., handling messy data, ensuring AI explainability, securing LLMs)?
- **Clarity & Communication:** Is your documentation clear, concise, and professional? Is the user interface for the prototype intuitive?

High-Level Architecture Diagram (C4 Model):



Infrastructure & DevSecOps Plan:



CI/CD pipeline (single system: GitLab)

Purpose: take code → secure, tested image → deployed to clusters.

1. **Trigger:** Dev pushes or opens a Merge Request → GitLab runs pipelines.
2. **Build:** Docker build → versioned tag (app:gitsha) → sign SBOM (e.g., Syft/CycloneDX).

3. Security gates:

- **SAST** on the repo (Python/JS/etc.).
- **DAST** against a review app or staging URL.
- **Image scan** (Trivy/Grype) on the built image + SBOM policy checks.
- Pipeline fails if policy thresholds are exceeded.

4. **Publish:** Push the vetted image to the **Image Registry** (Docker/OCI).

5. Deploy (GitOps via GitLab Agent):

- GitLab updates the env manifest/Helm values (image tag, replicas, config).
- **GitLab Agent** in each cluster applies the change (dashed lines in the diagram).
- Rollouts use **K8s deployments** with health probes and progressive rollout (canary/blue-green optional).

Kubernetes deployment (On-Prem + Cloud)

Purpose: run the same workload in two footprints with clear data boundaries.

- **On-Prem K8s** (blue badge): hosts **sensitive workloads** and data-adjacent services.
- **Cloud K8s** (red badge): runs **non-sensitive/bursty** services and scale-outs.

Common deployment shape

- **Namespace per env/app** with **NetworkPolicies**.
- **Deployment** + **Service** + **Ingress/IngressClass** (or Gateway API).
- **Config & Secrets:** ConfigMaps, encrypted Secrets (SOPS/Sealed-Secrets/External-Secrets).
- **Autoscaling:** HPA (CPU/requests per second); optionally VPA for rightsizing.
- **Probes:** liveness/readiness/startup for safe rollouts.
- **Rollback:** kubectl rollout undo or Git revert → Agent reapplies.

Image flow: clusters **pull images** from the registry using **imagePullSecrets**; tags are immutable per release.

IaC strategy

Terraform (provisioning – "what exists"):

- Creates clusters (or attaches existing), node pools, storage classes, Ingress controllers, GitLab Agent, and baseline add-ons (metrics server, cert manager, ExternalDNS if cloud).

- Manages environment separation and labels/taints for scheduling.

Ansible (configuration – "how it's configured"):

- Post-provision host/node config (sysctls, containerd config, CSI drivers), bootstrap cluster add-ons, or configure on-prem dependencies (NFS, reverse proxies, firewalls).

Image Registry

- **Single source** for OCI images, with repo-level policies (immutable tags, retention).
- Integrated vulnerability scan (and enforced in pipeline).
- Geo-replication/caching recommended for on-prem pull performance.

Observability (Prometheus + Grafana)

- **Prometheus** scrapes: app metrics (/metrics), kube-state-metrics, node exporter.
- **Alertmanager** routes SLO/SLA alerts (latency, error rate, saturation).
- **Grafana** dashboards for CI/CD, registry pulls, and both clusters.
- Pipelines and clusters emit **metrics & alerts** to this stack (see arrows).

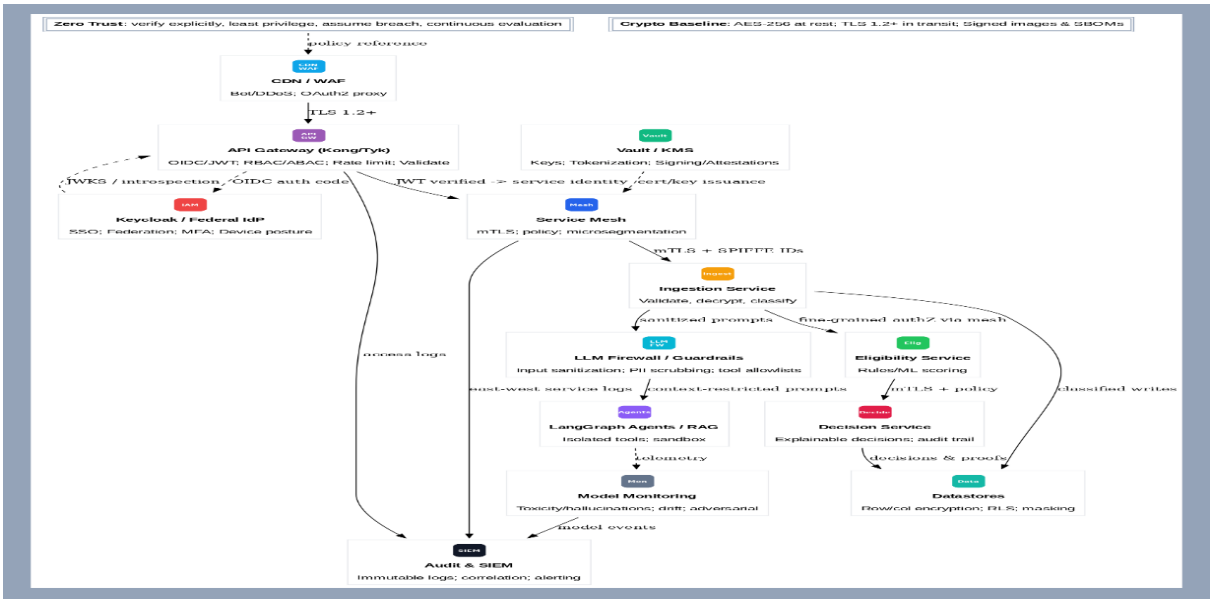
Security & compliance touchpoints (visible in the diagram)

- **SAST/DAST/SBOM** gating in the pipeline (shift-left).
- **Signed images** and SBOM stored per build (attestations with Cosign optional).
- **Runtime controls** (recommended): Pod Security Standards (restricted), network policies, read-only root FS, non-root UID, resource limits/requests.
- **Secrets** from a KMS/secret manager (avoid inline secrets).

End-to-end flow (what happens on a change)

1. **Dev push/MR** → GitLab pipeline starts.
2. **Build & scan** → SBOM, image scans; if clean, **push to registry**.
3. **Terraform** (periodic/when infra changes) ensures clusters/add-ons exist.
4. **Ansible** applies/updates node & add-on configs (esp. on-prem).
5. **Deploy**: GitLab Agent updates K8s manifests/Helm → controlled rollout.
6. **Run**: K8s pulls the image, autoscaling as needed; Prometheus/Grafana collect metrics/alerts.
7. **Rollback**: revert Git change or rollout undo; pipeline re-applies desired state.

Zero Trust Security Architecture:



1) Zero-Trust banner (top)

Principles enforced everywhere: verify explicitly (strong identity), least privilege (scoped tokens & policies), assume breach (mTLS, segmentation), and continuous evaluation (telemetry & revocation).

2) Edge / North–South entry

CDN/WAF

- Stops junk at the edge: DDoS & bot mitigation, geo/IP allow/deny, basic OWASP rules.
- Optional OAuth2 proxy for simple browser apps (adds user session before they reach APIs).

API Gateway (Kong/Tyk)

- **AuthN:** OIDC/JWT with Keycloak/Federal IdP.
- **AuthZ:** RBAC/ABAC (e.g., roles + attributes like tenant, clearance).
- **Threat controls:** schema validation (OpenAPI), request size limits, IP/geo/rate limiting, mTLS to the mesh.
- **Why:** central choke point for north–south traffic; anything that fails here never reaches services.

Key configurations

- OIDC client in API GW; validate JWT iss, aud, exp; fetch JWKS for key rotation.
- Global policies: 429 on rate-limit, 413 on payload too large, reject unknown content-types.

3) Identity & Secrets

Keycloak / Federal IdP

- SSO + federation (SAML/OIDC), MFA, device posture claims.
- Emits **short-lived** JWTs with minimal claims (subject, tenant, roles).

Vault / KMS

- Central key management: encryption keys, signing/attestation keys, tokenization (PAN/SSN).
- Issues **mTLS certs** (SPIFFE IDs) to workloads; rotates certs & seals secrets.

Key configurations

- JWT lifetime ≤ 15 min; use refresh tokens only at trusted frontends.
- Vault PKI role per namespace; cert TTL hours, automated renewal via sidecars/mesh.

4) Platform / East–West (inside the cluster)

Service Mesh (mTLS, policy, micro-segmentation)

- Every service-to-service call is **authenticated** (workload identity) and **encrypted** (mTLS).
- Authorization by **policies**: "who can talk to whom" + request-level checks (method/path).
- Rate-limits and retries at mesh, not in app code.

Business services

- **Ingestion**: validates payloads, decrypts fields, classifies data.
- **Eligibility**: rule/ML scoring (stateless where possible).
- **Decision**: final decision + full audit trail.

Data Security & Governance (alongside services)

- **Classification:** Public, Internal, Confidential, Restricted (stored with the record).
- **Storage policy per class:**
  - *Public*: standard storage.
  - *Internal/Confidential*: AES-256 at rest, key from Vault; masking in non-prod.
  - *Restricted*: field-level encryption + tokenization; access via break-glass workflow only.
- **Row/Column security:** RLS (per tenant) + column masks for PII/PHI.

Key configurations

- Mesh authorization: "ingestion → eligibility" allowed, **everything else denied by default**.
- Mutual TLS enforced namespace-wide; reject plaintext.
- Secrets mounted via CSI/sidecar; never in environment variables.

5) AI / LLM Security

LLM Firewall / Guardrails (before the model)

- **Input sanitization:** strip secrets, normalize Unicode, enforce schema (JSON only), remove tool-calling directives unless allowed.

- **Prompt canonicalization:** escape/quote user content; prepend trusted system prompts.
- **PII scrubbing:** deterministic masking/tokenization; allow-list tools/functions.

Agents / RAG

- Each tool/skill runs in a **sandbox** with a minimal permission set (filesystem, network egress, external APIs).
- Retrieval (RAG) constrained to approved indexes/tenants; documents pre-redacted by classification.

Model Monitoring (after the model)

- Classifiers & rules for **toxicity, leakage, hallucinations**, high perplexity, and jailbreak signatures.
- Stores **model telemetry** (prompt+response hashes, scores, version, source docs) → attaches to audit trail.

Threat → control mapping

- **Prompt injection** → input sanitizer + tool allow-lists + output filters.
- **Model inversion/leakage** → PII scrubbing, response redaction, egress controls, strict retrieval scope.
- **Data poisoning** → signed data pipelines, dataset versioning, anomaly checks pre-train, canary evals.
- **Abuse** → rate-limits, content policies, human review queues for flagged outputs.

6) Storage / Audit

Datastores

- AES-256 at rest; field-level encryption for sensitive fields; tamper-evident logs.
- Backups encrypted; restore tested; KMS-bound keys (no app-owned keys).

Audit & SIEM

- Ingests **access logs** (API GW), **east-west logs** (mesh), and **model events** (monitor).
- Correlates user identity, workload identity, and data class → detections & alerts.
- Feeds compliance reports (who read what, when, and why).

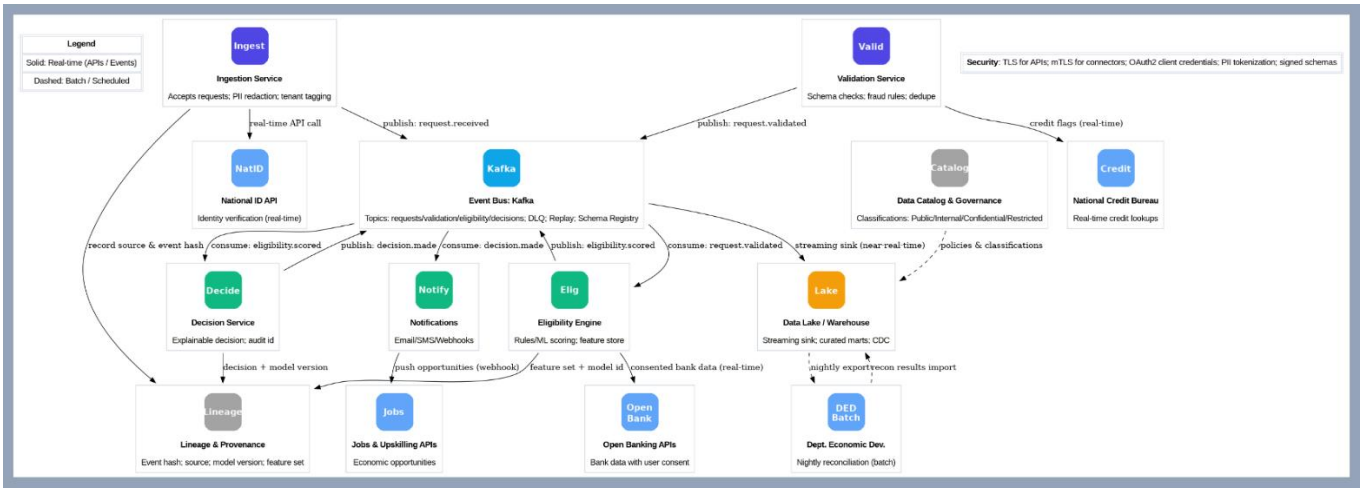
7) Crypto Baseline (callout box)

- **In transit:** TLS 1.2+ end-to-end, mTLS inside mesh.
- **At rest:** AES-256; keys from Vault/KMS; rotation & revocation policies.
- **Supply chain:** signed artifacts & SBOMs; verify before deploy.

8) How the request actually flows (arrow-by-arrow)

1. **Client → CDN/WAF:** edge filters; suspicious traffic dropped.
2. **CDN/WAF → API Gateway:** TLS; gateway enforces OIDC login.
3. **API GW ↔ Keycloak:** OIDC code flow; JWKS is fetched for signature verify.
4. **API GW → Service Mesh:** forwards request with validated JWT; mesh establishes **mTLS** to **Ingestion** using workload identities.
5. **Ingestion → Eligibility → Decision:** each hop allowed by mesh policy; tokens narrowed to least privilege; classification tags travel with the payload.
6. **Ingestion → LLM Firewall:** input sanitized; PII scrubbed; allowed tools only.
7. **Guardrails → Agents:** constrained RAG/tools; outputs checked; telemetry to **Model Monitoring**.
8. **Services → Datastores:** writes use class-based encryption and masking rules.
9. **Everywhere → SIEM:** API access logs, east-west logs, and model security events flow to SIEM for correlation/alerts.

Data and Integration Architecture:



1) Core services & event bus

Ingestion Service

- Accepts requests (API/gateway) and **normalizes + tags** payloads (tenant, source, correlationId).
- **PII handling:** tokenize/mask sensitive fields on entry; keep raw only when justified.
- Publishes request.received to **Kafka** with an **idempotency key** (eventId).

Validation Service

- Consumes request.received, performs **schema & business validation**, dedup/fraud checks.
- Writes corrections/flags and publishes request.validated (or sends to **DLQ** with reason).

Kafka (with Schema Registry, DLQ, Replay)

- Topics (example):
  requests.received, requests.validated, eligibility.scored, decision.made.
- **Schema Registry** enforces Avro/JSON schema compatibility; events are **signed** and versioned.
- **DLQ** per topic for poison messages; **Replay** for backfills or recovery.

2) Decisioning lane

Eligibility Engine

- Consumes requests.validated; enriches with **consented bank data** (Open Banking) + **credit flags**.
- Scores features/rules; publishes eligibility.scored.

Decision Service

- Consumes eligibility.scored; applies policy/ML decision with explainability.
- Emits decision.made with **auditId** and **modelVersion** references.

Notifications

- Consumes decision.made; pushes email/SMS/webhook; can also call job/upskilling APIs for opportunities.

3) External providers (real-time & batch)

- **National ID API (real-time)** from Ingestion (identity verification).
- **National Credit Bureau (real-time)** from Validation (credit flags/score).
- **Open Banking (real-time, consented)** from Eligibility (account/transaction summaries).
- **Jobs/Upskilling (webhook)** from Notifications (match opportunities).
- **Department of Economic Development (batch)**: nightly **export** of decisions and **import** of reconciliation results.

4) Data platform, governance & lineage

Data Lake/Warehouse

- Near-real-time sink from Kafka (streaming/CDC) plus batch loads.
- Zones: **raw** (immutable events), **curated** (conformed), **marts** (analytics).

Data Catalog & Governance

- Classification labels: **Public, Internal, Confidential, Restricted** drive storage/masking/retention.

- Access via policies (role + purpose + data-class).

Lineage & Provenance Store

- For each decision, stores: eventHash, sourceSystem, schemasUsed, featureSetId, modelVersion, and **pointers to the exact events** used.
- Guarantees you can answer: *"Which data and which model produced this decision?"*

How a request flows (step-by-step)

1. **Client → Ingestion (API)**: payload arrives with Idempotency-Key. PII tokenized.
2. **Ingestion → Kafka**: publish requests.received (schema-validated, signed).
3. **Validation** consumes → enriches with **Credit Bureau** → publish requests.validated.
4. **Eligibility** consumes → fetches **Open Banking** (via consent) → compute features → publish eligibility.scored.
5. **Decision** consumes → evaluate policy/ML → publish decision.made (with auditId, modelVersion).
6. **Notifications** consumes → send comms / call **Jobs & Upskilling**.
7. **Streaming sink** writes all events to **Data Lake**; **Lineage** captures hashes/versions; **Nightly batch** reconciles with **DED** both ways.

Real-time vs batch patterns

- **Real-time (solid lines):** synchronous API calls to NID/Credit/Open Banking with **circuit breakers**, **timeouts**, and **retries (exponential backoff)**; fallback to partial scoring if non-critical.
- **Batch (dashed lines):** nightly S3/Blob exports to **DED**, checksum-verified; results re-ingested as reconciliation.completed.

Security controls (applied end-to-end)

- **Transport:** TLS for external APIs; **mTLS** for internal connectors.
- **Auth:** OAuth2 client-credentials for outbound; short-lived tokens; per-provider scopes.
- **Data:** PII tokenization at ingest; field-level encryption for Restricted data.
- **Events:** schemas signed; producers/consumers authorized per-topic (ACLs).
- **Secrets:** pulled at runtime from Vault/KMS (no secrets in env).

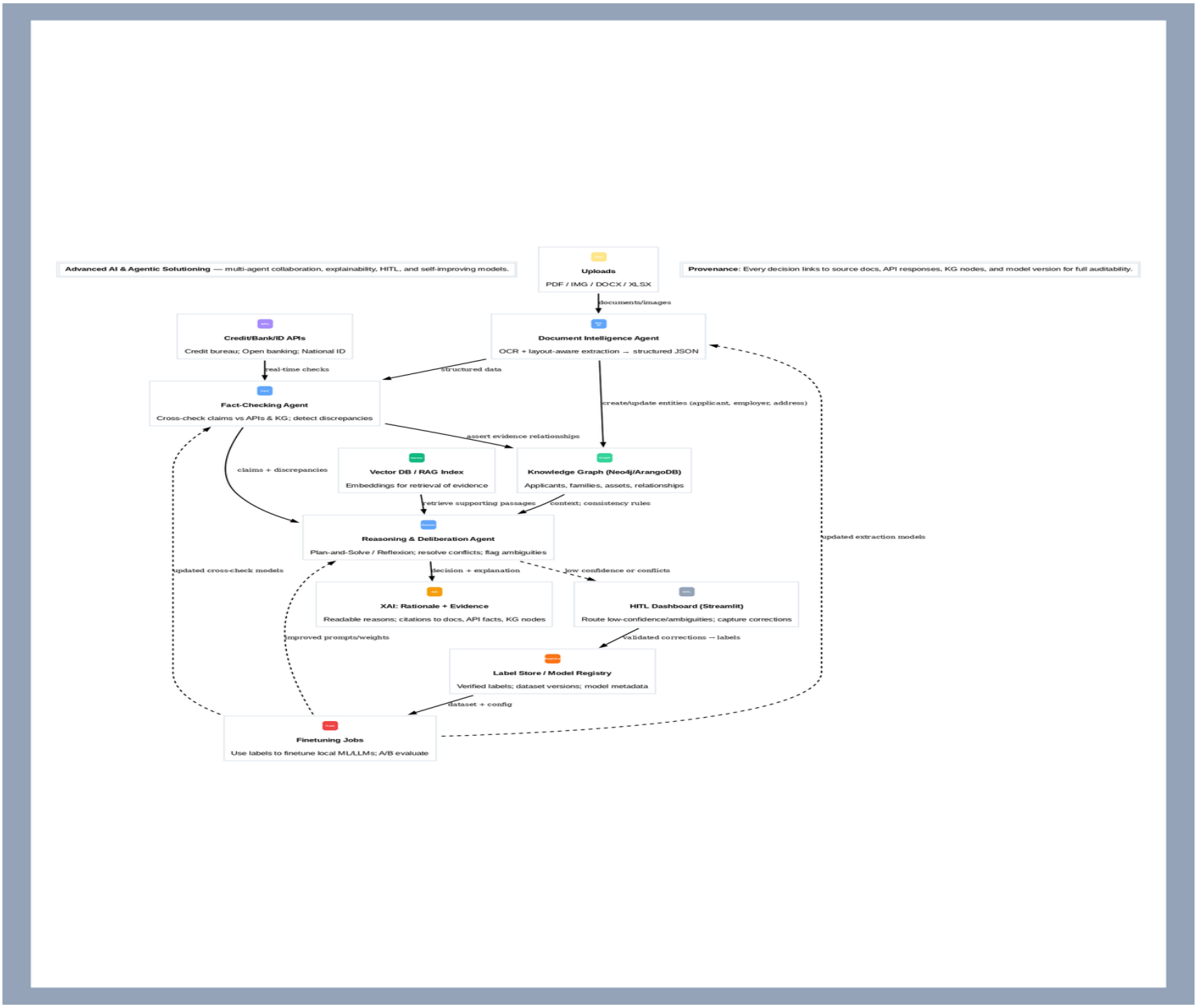
Failure handling & resilience

- **DLQ** on every critical topic with structured error (schema violation, auth, transform).
- **Replay** from a timestamp or offset to re-process with a new model/policy.
- **Outbox pattern** for services that need both DB write and event publish (avoid dual-write issues).
- **Backpressure**: consumer groups with autoscaling; max-in-flight per partition.

Monitoring & SLOs

- **Lag & throughput** per topic/partition; consumer error rate; DLQ rate.
- **External API** p95 latency/error budgets; retry counts; circuit-breaker opens.
- **Data quality** checks: schema violations, null-rates, drift on key features.
- **Lineage coverage**: % of decisions with complete provenance.

Advanced AI Agentic Workflow:



1) Multi-Agent Collaboration (who does what)

Document Intelligence Agent (DocAI)

- **Goal:** turn messy PDFs/images (forms, bills, statements, IDs) into clean, typed JSON.
- **How:** OCR + layout-aware models (table/field detection, key-value pair extraction, signature/stamp detection).
- **Output:** structured fields (e.g., name, address, declared income), plus **per-field confidence** and **source pointers** (page, bounding box).
- **Writes to:**
    - **Knowledge Graph (KG)**: new/updated entities (Applicant, Employer, Address) and relationships (works_at, lives_at).
    - **Vector DB/RAG**: chunks of the source doc (for grounding later explanations).

Fact-Checking Agent

- **Goal:** verify extracted facts and find discrepancies.
- **How:** calls **external APIs** (e.g., national ID, credit bureau, open banking aggregators) and **queries the KG** to cross-reference prior facts (historical addresses, past income, employer status).

- **Output:** per-field **veracity status** (matched / mismatched / unknown), supporting evidence (API responses, KG nodes), and **discrepancy list**.

Reasoning & Deliberation Agent

- **Goal:** produce a **reasoned recommendation** (approve/decline/needs-review) with a clear explanation.
- **How: a Plan-and-Solve (PaS) or Reflexion loop:**
    1. **Plan** the verification steps based on discrepancies.
    2. **Execute** targeted checks (query KG, ask Fact-Checker for additional facts, retrieve doc snippets from the Vector DB).
    3. **Reflect** on conflicts; resolve with simple rules (freshness > stale records, primary evidence > secondary, multiple corroborations > single source).
- **Output:** decision + **explanation package** (see XAI below); routes low-confidence cases to **HITL**.

2) Knowledge Layer (how we keep the facts straight)

Knowledge Graph (Neo4j/ArangoDB)

- **Nodes:** Applicant, FamilyMember, Employer, Address, Asset, Account, Document.
- **Edges:** LIVES_AT, WORKS_AT, OWNS, RELATED_TO, SUPPORTED_BY (Document → Claim), VERIFIED_BY (API → Claim).
- **Why a graph:** captures **relationships and history** (e.g., an applicant moved, changed jobs), enabling **consistency checks** across documents and claims. It also helps detect **non-obvious patterns** (shared addresses/accounts across different applicants → potential fraud/risk).

Vector DB (RAG index)

- Stores **embeddings** of document passages/snippets, API snippets, and prior decisions.
- Lets the Reasoning agent **retrieve precise evidence** to support (or refute) a claim and **quote** it in the explanation.

3) Data Flow (what happens to a case)

1. **Upload/Intake** → *DocAI* ingests PDFs/images; extracts fields with confidence + source coords.
2. *DocAI* writes structured facts to **KG**, embeds document snippets to **Vector DB**.
3. *Fact-Checker* pulls key claims (e.g., income, employer, address) →
    - Queries **National ID** (identity match, date of birth),
    - Queries **Credit Bureau** (score, delinquencies),
    - Pulls **Open Banking** (consented transactions/income patterns),
    - Queries **KG** for historical consistency.
4. *Fact-Checker* emits a **discrepancy report** (e.g., “declared income 45k vs bank-observed 30-35k trend; address mismatch vs ID”).
5. *Reasoning* runs a **PaS/Reflexion** loop to reconcile conflicts, retrieve evidence from Vector DB, and produce:
    - **Recommendation:** approve / decline / needs human review,
    - **Confidence score**,
    - **XAI explanation** (bullet reasons + citations).
6. If **confidence < threshold** or **high-impact discrepancy** → route to **HITL** (Streamlit Dashboard). Otherwise, decision is ready for downstream systems (with full rationale and provenance).

4) XAI (explainability that isn’t a black box)

What the system produces for each decision

- **Summary:** 1–3 bullet reasons written in simple, human language.
- **Citations:** clickable references to **exact doc regions** (page/box), **API fields**, and **KG nodes** used as evidence.
- **Scorecard:** key criteria and pass/fail status (e.g., “Minimum tenure: met; Income ≥ threshold: unmet; Address verified: yes.”).
- **Model metadata:** model/version used for extraction and reasoning; features referenced; decision policy version.

Why this works

- Every claim is backed by **explicit artifacts** (doc excerpt, API field, KG relationship), so auditors and case officers see **exactly** why the system concluded what it did.

5) HITL (Human-in-the-Loop) & Finetuning Loop

When a case is routed to HITL

- Low **confidence** or conflicting facts (e.g., income mismatch).
- **Ambiguous** documents (blurry scans, multiple addresses).
- **Edge cases** not covered by policy.

What the case officer does in Streamlit

- Sees the **explanation** + **citations**.
- Marks fields as **correct/incorrect**, uploads better evidence, or selects the **right value**.
- Confirms final decision; adds comments.

How feedback improves the system

- Corrections are stored in a **Label Store / Model Registry** with exact references (fields, spans, evidence).
- **Nightly/weekly Finetuning Jobs:**
  - Update **DocAI** (better OCR/layout extraction on difficult layouts),
  - Refine **Fact-Checker** rules/models (new patterns),
  - Adjust **Reasoning** prompts/weights (reduce hallucinations, improve deliberation).
- A/B evaluation ensures **new models outperform** before promotion.

6) Data Lineage & Provenance (full traceability)

For every automated or human-assisted decision, the system stores:

- **Source artifacts**: document file/version + bounding boxes; API response hashes; KG node IDs/version.
- **Model metadata**: which **model/version** produced each extraction, verification, reasoning step.
- **Prompt/parameters** (summarized/hashes for LLM steps) to reproduce reasoning.
- **Decision policy** version (thresholds, rules).
- **Time-ordered event log** to replay a case end-to-end.

7) Security & Governance (baked in)

- **PII protections:** tokenize/mask sensitive fields in working stores; limit plaintext to secure enclaves.
- **Access control:** least-privilege roles across agents, KG, Vector DB, APIs.
- **Audit logs:** immutable logs for doc access, API calls, KG changes, model inferences.
- **Secrets:** fetched at runtime from a vault/KMS; no embedded secrets.
- **Policy guardrails:** rate limits, schema validation, and output filters on the Reasoning agent (to prevent leakage).

8) Quality, Monitoring & SLOs

- **Extraction quality:** field-level F1 by document type; confidence calibration charts.
- **Verification quality:** match/mismatch accuracy vs ground truth; coverage of cross-checks.
- **Reasoning quality:** decision accuracy vs adjudicated ground truth; **explanation fidelity** (does it cite the same evidence humans would?).
- **HITL metrics:** % of cases routed to review; turnaround time; re-adjudication rate.
- **Continuous learning:** uplift from new finetunes; rollback if regressions detected.