

The Sovereign Agentic AI Operating System

An open-source reference architecture for auditable, federated, and self-improving agent infrastructure — engineered for European digital sovereignty

Sebastian Kister · Chief AI Architect, Open Source Program Office

White Paper · Souveränitäts-Kompass · July 2026

Table of Contents

Recursive · Self-Organising (Bounded) · Federated Context · Prompt → Organisation

Status: Internal review draft. Feasibility-assessed. License and project facts should be re-verified periodically against current sources.

1. Executive summary

Agentic AI is moving from demonstration to infrastructure. The organisations that adopt it will not merely run a chatbot; they will run *agent organisations* — software entities that read sensitive context, call tools, execute code, spend budget, and modify systems. The strategic question is no longer whether to adopt agentic AI, but on whose foundation. This paper argues that for a regulated European enterprise, that foundation must be open source, end-to-end governed, and federated — and it specifies an architecture that delivers all three.

The central thesis is a single sentence: sovereignty is not where the model is hosted; it is the entire chain — prompt, tool call, data access, response — and every link in that chain must be open, auditable, and under your control. A sovereign model behind a foreign-jurisdiction tool server is not sovereign. A self-hosted stack built on a restrictively licensed single-vendor component is not sovereign either. Sovereignty is a property of the whole system, and it is lost at the weakest link.

The architecture is organised as ten composable open-source layers plus a cross-cutting governance plane. It implements a bounded recursive agent hierarchy (META → EXEC → SPEC), a self-quality-assuring execution loop, and a federated context model that brings intelligence to the data rather than copying data into a central lake. It is deliberately engineered against three failure modes that defeat most agentic systems: unbounded reliability decay across long agent chains, unaccountable autonomy that cannot satisfy the EU AI Act, and operational sprawl that no team can sovereignly maintain.

This document is written to an engineering-honest standard. Where a capability is mature, it says so; where a capability is a research problem, it labels it as research and assigns a

technology-readiness level. That honesty is a strength, not a weakness: serious engineering investment is justified by solving hard, first-of-a-kind problems on a foundation of proven components. The mature layers here are integration; the genuine innovation is the reliable, auditable, self-improving agent fabric on top of them. Both are described plainly so the claim survives scrutiny.

2. The sovereignty problem

2.1 Sovereignty is the whole chain

The prevailing market narrative reduces AI sovereignty to model hosting: run the weights in-region and the problem is solved. It is not. A modern agentic request traverses many stages — it is parsed, decomposed, routed to a model, enriched with context fetched from internal systems, executed against tools, validated, and returned. Each stage is a data-processing event with its own jurisdiction, its own failure modes, and its own attack surface.

Sovereignty equals the entire chain: prompt → tool call → data access → response — not the location of the model.

Core thesis

Three myths are worth retiring explicitly. First, that sovereignty equals model location — when in reality it is the whole processing chain. Second, that the model is the principal security risk — when in practice stateful, multi-tenant tool servers are the highest-risk data processors, because they sit between the agent and the data and they persist. Third, that speed requires multi-tenant SaaS — when speed without governed infrastructure simply produces shadow IT that is fast precisely because it is ungoverned.

2.2 The foreign-jurisdiction tool-server threat

The Model Context Protocol (MCP) and similar tool interfaces are the connective tissue of agentic systems: they let an agent call an external capability and pull external context. They are also the most under-examined risk in the stack. A tool server that is stateful and multi-tenant, and that runs under a foreign jurisdiction, becomes a concentration point for an enterprise's most sensitive context — exactly the data the agent needs to be useful. If the agents that drive a transformation depend on such servers, the transformation is built on a compromised foundation, regardless of where the language model runs.

The architectural response is twofold: treat third-party tool servers as untrusted by default — sandboxed, with controlled egress and full attribution — and prefer connectors that query systems of record live rather than tool servers that hoard a copy. The connector is not a data sync; it is a governed, audited doorway to a source that remains authoritative.

2.3 Why open source is non-negotiable

Sovereignty and proprietary platforms are strategically incompatible. A closed agentic platform — however capable — reintroduces the dependency that sovereignty exists to remove: opaque data handling, unilateral pricing and policy changes, and an audit story that ends at the

vendor's boundary. For a system subject to the EU AI Act and the Cyber Resilience Act, auditability cannot stop at a black box.

But “open source” is not a single thing, and this is where most sovereignty strategies quietly fail. A component under a source-available but non-OSI-approved license — the SSPL class popularised by the MongoDB, Elastic, HashiCorp and Redis relicensing events — carries the same lock-in and legal-uncertainty risk that proprietary software does, often with less clarity. Worse, several of the most attractive young components in the agentic ecosystem are single-vendor projects that exist precisely because an upstream was deprecated or relicensed. Building the sovereignty layer on one of those reproduces the trap. The discipline this paper enforces is therefore stricter than “open source”: every component on the critical path must be OSI-approved and backed by a foundation or a genuine multi-vendor community. Section 9 makes that discipline concrete with a component-by-component map.

2.4 Jurisdiction is a trust axis — and trust is contingent

Framing sovereignty as jurisdiction assumes a stable answer to one question: do you trust a nation and its courts — today and for the lifetime of the system? Trust in the United States eroded from 2013 onward not because of lawbreaking but because the legal framework itself — FISA §702 and the CLOUD Act — authorises compelled access to data held by US-controlled providers. The instructive point is that a *lawful* regime, not an illegal one, is what defeats sovereignty-by-hosting.

The European legal space is stronger, but it is not static, and a paper that claimed otherwise would fail its own engineering-honesty test. Under the *ProtectEU* strategy and the “*Going Dark*” agenda, the European Commission is pursuing lawful-access-by-design: a 2026 encryption technology roadmap, next-generation decryption capability for Europol from 2030, revived data-retention rules reaching toward VPNs, and the recurring *Chat Control* proposals for client-side scanning. The fair counter-reading matters too: the Court of Justice (the *Schrems* rulings) and the European Court of Human Rights (which held that mandated encryption backdoors violate the Convention), together with an open and contested legislative process, demonstrate a functioning separation of powers — unlike the secret technical-capability notices issued in the United Kingdom. The honest conclusion is that jurisdiction is volatile and politically contingent: Europe remains the strongest available jurisdiction, but it cannot be locked to “sovereign” *by law*.

What *can* be locked is the technical layer. If the operator is excluded from cleartext by construction — confidential computing, keys held beyond the provider's reach, independent attestation — then even a future access law compels nothing, because no access-entitled party exists. This is why the architecture treats key sovereignty as the load-bearing control (Section 5.10 and the governance plane) and treats jurisdiction as a trust *input* to be minimised, not a guarantee to be relied upon.

3. Design principles

Six principles govern every decision in the architecture. They are stated as constraints, not aspirations, because each one rules options out.

Bring intelligence to the data, not data to the intelligence

The reflex of the last decade was to centralise: pull everything into a lake, harmonise schemas, then compute. For agentic context this is the wrong default. It creates a second system of record that is stale the moment a source changes, and it concentrates exactly the sensitive data that should stay put. The principle inverts it: keep data at source, maintain only a *graph of where things live and how they relate*, and fetch the actual data live when an agent needs it. The map is small and central; the territory is large and stays sovereign.

Sovereignty requires OSI-approved open source

Every component on the critical path must be OSI-approved and community- or foundation-governed. Source-available but restrictive licenses (SSPL and similar) and single-vendor projects are treated as lock-in risks and kept off the critical path. This principle directly determines component selection and is the reason this paper replaces certain popular choices with more neutral equivalents.

Engineering honesty over marketing language

Documents that face legal and financial scrutiny cannot contain aspirational claims dressed as facts. Where a property is achieved, the paper claims it; where it is a goal, the paper labels it a goal and assigns a maturity level. A reviewer who finds one overstated claim discounts the rest; honesty is the cheapest way to protect credibility.

Governance over autonomy

Autonomy and auditability pull in opposite directions, and where they conflict, governance wins. The system may propose changes — to its own prompts, tools, even model adapters — but nothing reaches production without a policy decision, a human gate where required, and a complete audit record. Autonomy proposes; humans and policy dispose; everything is logged.

Bounded reliability

Reliability in agent chains is multiplicative: independent steps each succeeding with probability p yield an end-to-end success of roughly p^n over n steps — and in practice errors correlate, so the real figure is worse. This single fact disciplines the entire agent design (Section 6). Unbounded recursion is therefore prohibited; depth is capped, each level carries an explicit error budget, and the original goal is re-asserted at every level to resist drift.

Design for Day 2, not Day 1

Day 1 — the launch — is a single moment. Day 2 — operation, upgrade, incident response, and the slow accretion of dependencies — is the rest of the system's life. A stack of fifty loosely-coupled projects can be assembled in a demo and is unmaintainable in production: every project is a version, a CVE stream, an upgrade treadmill, and a potential relicensing event. The architecture therefore consolidates deliberately — one inference runtime, one orchestration substrate, one GitOps engine, one isolation technology, two policy engines — accepting fewer moving parts as the price of sovereign operability. This also guards against the automation paradox: automating to cut cost, then spending more to maintain the automation than was ever saved.

4. Architecture at a glance

The platform is a stack of ten composable layers, from the human-facing surface down to the Kubernetes substrate, wrapped by a cross-cutting governance plane that every action passes through. The layers are independently deployable and individually replaceable; the only hard contract between them is typed interfaces, observable transitions, and cryptographic identity at every boundary.

#	Layer	What it does
01	Intake & Contract	Turns a prompt into a validated task contract plus an organisation plan. PII redaction and policy pre-check are deterministic; sizing and topology are schema-constrained LLM output.
02	Access	Five surfaces (web, IDE, CLI, voice, approvals) into one pipeline. Humans authenticate via OIDC; workloads carry SPIFFE identity.
03	LLM Gateway	One provider-agnostic API (LiteLLM) over one inference runtime (vLLM; Ollama for edge), with contract budgets enforced here, guardrails (Granite Guardian + OPA), and failover.
04	Agent Hierarchy	Bounded recursive META → EXEC → SPEC on a single LangGraph substrate; every spawn edge carries a validated task contract with depth allowance, error budget, and goal re-anchoring.
05	QA & Self-Healing	A verifier hierarchy — tests, schemas, and policies before any LLM critic — plus bounded retries and a codify-or-escalate loop. No in-loop fine-tuning.
06	Tool Factory	Codifies reliable patterns into

#	Layer	What it does
		deterministic tools and IaC — signed (Sigstore), SBOM-attested, human-gated — the primary cost-reduction mechanism.
07	Skills & Connectors	JIT skills catalog and governed connectors (MCP/OpenAPI) that query sources live, with caching and sandboxed third-party servers.
08	Federated Context	A Postgres-native relationship graph plus federated query (Trino) — no central data lake; sources stay authoritative.
09	Cache / Queue / DB	Valkey, Kafka (KRaft, via Strimzi), and PostgreSQL/pgvector with concerns separated; effectively-once messaging with idempotent consumers.
10	Execution	Sovereign Kubernetes: queue-driven scaling (KEDA), microVM isolation (Kata), GitOps (Argo CD), full observability.
—	Governance plane	SPIFFE + Keycloak identity, OpenFGA + OPA policy, OpenBao secrets, Sigstore/SLSA supply chain, OpenSearch audit, human-in-the-loop gates. Spans every layer.

Figure 1. The ten layers and the governance plane. Detailed component selection and rationale follow in Section 5; the governance plane is detailed in Section 10.

5. The layered architecture

Each layer is described by what it does, why it exists, and which open-source components deliver it. The descriptions are the corrected, feasible forms — they reflect the engineering constraints of Section 3 rather than an idealised diagram.

5.1 Intake and contract — from prompt to typed task contract

A natural-language prompt is unstructured; an agent organisation is structured. This layer is the translator, and it is the only layer the human directly touches. The only work that stays deterministic is the work that must be: PII redaction before any model contact (Presidio) and an OPA policy pre-check, so forbidden goals fail before budget is burned. Task-size estimation and organisational-topology planning, however, are irreducibly model-driven: deciding whether a request is atomic or needs forty sub-agents is a semantic judgment, not a parse. That judgment is therefore captured as a typed task contract — a Pydantic schema enforced through constrained decoding (xgrammar), not hoped for via prompt — with six fields: verbatim goal anchor, machine-checkable success criterion, budget, depth allowance, error budget, and return schema plus deadline. Because every downstream agent inherits the result, decomposition quality is the single largest determinant of end-to-end success — and no child is spawned without a validated contract.

5.2 Access layer

Different users live in different surfaces: a developer wants the IDE, an operator the CLI, a business user chat or voice. Forcing one surface excludes most of the audience. LibreChat, Continue.dev, a Typer/Click CLI, faster-whisper voice, and approval surfaces — thin views over orchestrator interrupts, not a separate low-code product — all route into the same ingestion pipeline: one brain, many mouths. (Open WebUI was replaced after it adopted a custom, non-OSI license with a branding clause and CLA in April 2025 — precisely the relicensing pattern Section 9 guards against.) Identity is handled correctly here: humans authenticate via Keycloak (OIDC); the workload acting on a human's behalf carries a SPIFFE identity plus a delegated, scoped user token. Attribution is therefore precise — which workload acted, on whose behalf, under which policy — and every interaction is traced end-to-end via OpenTelemetry's GenAI semantic conventions.

5.3 LLM gateway — provider-agnostic

Hard-coding one model provider into every agent makes the organisation hostage to that provider's pricing, availability, and policy. LiteLLM presents a single OpenAI-compatible API and routes by cost, latency, capability, and sovereignty. One inference runtime is chosen rather than several competing ones: vLLM for production serving, Ollama for development and edge. Constrained/structured output is a feature of that runtime (via xgrammar/outlines), not a separate engine. I/O guardrails run as an Apache-2.0 classifier plus policy engine at the gateway (Granite Guardian + OPA) — Llama Guard fails the OSI criterion, and NeMo Guardrails brings its own DSL and single-vendor weight. The gateway is also where task-contract budgets are technically enforced: children inherit shares, and LiteLLM counts and cuts. The honest performance lever is exact prefix caching in vLLM; a semantic cache stays off the critical path, opt-in per task class only, because a false cache hit is a silent error.

5.4 Agent hierarchy — bounded recursive

A single agent cannot solve arbitrarily complex tasks: context windows are finite and reasoning degrades with scope. Real organisations decompose work, and this layer mirrors that — META decides structure, EXEC drives a domain, SPEC performs atomic work. Crucially, because

reliability is multiplicative, recursion is depth-bounded (initially two to three levels), each level carries an error budget, and the original goal is re-asserted at every level to resist drift. One orchestration substrate is used — LangGraph, for its explicit state and checkpointing — rather than a stack of competing frameworks; Temporal was evaluated and rejected as a second orchestration system, with durability coming from checkpoints, idempotent tools, and the event log instead. The graph validates the task contract on every spawn edge: depth allowance and budget are decremented, not hoped for. A capable coding agent (OpenHands) is invoked as a sandboxed tool by an EXEC node; group-chat and crew patterns are reimplemented on the substrate rather than carried as separate runtimes. Every prompt runs under a hard cost ceiling with a fallback-to-human path.

5.5 QA loop and self-healing

Model output is stochastic, so the system never trusts a first answer. Verification follows a strict hierarchy: hard checkers first — executing tests, validating schemas, checking policy — then symbolic checks, and only where nothing harder exists an LLM critic (Reflexion pattern) that runs on a different model or prompt regime than the executor, because a checker correlated with the producer checks nothing. Retrieval tasks are additionally scored with RAGAS. The inline loop is deliberately bounded: critique, then a small number of reflective retries, then codify the result if it is reliable or escalate if it is not. Golden evaluation sets are versioned in Git and executed in CI — without reference evals, “improved” is an assertion, not a measurement. Persistent, systematic gaps are flagged for an offline improvement pipeline — never patched by retraining inside a live task, which is neither fast enough nor data-rich enough to work (Section 7).

5.6 Tool factory and codification

Agent inference is expensive; deterministic code is effectively free to invoke. When a pattern proves reliable across many invocations, paying for inference each time is waste. The Tool Factory captures earned reliability as infrastructure: it generates skills and agent definitions (skills.md and AGENTS.md — the open formats of the Agentic AI Foundation family), scripts, and infrastructure-as-code targeting OpenTofu (Terraform has been BUSL-licensed since 2023; OpenTofu is the Linux Foundation fork), versioned in Git. The distinction matters — a codified script or IaC module is genuinely inference-free and is where the token savings come from, whereas a codified skill is standardised but still drives a model. Agent-produced tools are a supply chain: every artefact is signed (Sigstore cosign), ships an SBOM (Syft) and SLSA provenance, and admission rejects the unsigned. All generated artefacts, and especially infrastructure code, additionally pass a mandatory human review gate before promotion. Over time the system shifts from reasoning about everything to reasoning only where reasoning is genuinely needed.

5.7 Skills engine and connectors

Codified tools need a runtime home, and agents need to discover capabilities without bloating every prompt with the full catalogue. The Skills Engine indexes skills by capability and serves them just-in-time from Git. Connectors reach external systems — Git, CI, ticketing, messaging, Kubernetes, BI — and are dual-trait: each both ingests context and acts. A connector is not a central data sync, but it does cache with explicit freshness policy, because querying every

system of record live on every step would overload them. The Model Context Protocol provides a universal tool interface; third-party MCP servers, being the highest-risk processors in the system, run sandboxed with egress enforced by eBPF network policy (Cilium) and full attribution.

5.8 Federated context — no data lake

The traditional way to give agents context is to centralise everything into a lake: schemas, ETL, harmonisation, governance overhead — and staleness the moment a source changes. This layer rejects that. It maintains a graph of where things live, what they mean, and how they relate; the actual data is fetched live when needed and cached under policy. The honest claim is precise: there is no central data lake and no second system of record — sources stay authoritative — but data does move transiently at query time and caches do hold derived copies with a freshness window. A relationship graph holds the map — by default as plain PostgreSQL relations queried with recursive CTEs (“boring first”), with Apache AGE as the in-footprint escalation when Cypher expressiveness is genuinely needed; OpenMetadata catalogues sources; Trino executes federated SQL across heterogeneous systems; LlamaIndex shapes retrieved context into model-sized envelopes; embedding models are served by the same vLLM runtime — no second serving system.

5.9 Cache, queue, and database

Agents are asynchronous: they emit events, wait, hand off work, and resume. They need ephemeral state and pub/sub (Valkey, the BSD-licensed Redis successor), a durable event and audit backbone (Apache Kafka in KRaft mode, operated by the Strimzi operator; effectively-once with idempotent consumers), and durable state. The LangGraph state object that survives a crash is checkpointed to PostgreSQL; episodic vector memories live in pgvector, with OpenSearch k-NN — already present as the audit store — as the scale path when volume genuinely demands it. State checkpoints, append-only audit, and vector search are run as separate concerns rather than overloading one engine, and CloudNativePG operates PostgreSQL declaratively with backups, replication, and failover. Row-level security is one isolation layer among several — paired with per-tenant credentials and the policy plane, not relied on alone.

5.10 Execution — sovereign Kubernetes

Agents need somewhere to run that is ephemeral, isolated, and observable, without creating vendor lock. This layer runs on sovereign Kubernetes — bare-metal, sovereign cloud, or a hyperscaler, by choice. KEDA scales queue-driven agent workers from zero on queue depth, with warm capacity via minimum replicas — Knative would bring an entire serving stack for what is a scaling problem; untrusted code runs under one isolation technology — Kata Containers microVMs, with a path to confidential computing — rather than two in parallel; gVisor remains the documented lighter alternative, not a second default. Container builds happen in CI with rootless Buildah, never at spawn time (Google archived Kaniko in June 2025; Buildah is part of the CNCF Podman tools family). Argo Workflows expresses multi-step task DAGs; OpenTelemetry — including its GenAI semantic conventions for model spans — with Jaeger and Prometheus makes every action visible end-to-end; and Argo CD alone makes every deployment a reviewed, audited Git commit — one GitOps engine, no imperative changes.

6. The recursive agent model and the reliability problem

The hierarchy's appeal is obvious: a prompt becomes an organisation that sizes itself to the task. Its danger is just as real and is frequently ignored. Reliability across an agent chain is multiplicative. If each step succeeds independently with probability p , a chain of n steps succeeds with probability about p^n . The numbers are sobering: at $p = 0.95$, ten steps succeed end-to-end roughly 60% of the time and twenty steps roughly 36%; at $p = 0.90$, ten steps fall to about 35%. And these are optimistic, because real agent errors correlate rather than occurring independently.

A recursive hierarchy that “matches any task size” would, on a complex task, decompose into dozens of leaf steps — and multiply its way to near-certain failure. Recursion is therefore an instrument to be bounded, not a feature to be celebrated.

The reliability constraint

Two mechanisms keep this in check. The first is structural and contractual: every spawn edge carries a typed task contract — verbatim goal anchor, machine-checkable success criterion, budget, depth allowance, error budget, return schema — validated before the child exists; depth and budget are decremented at each level, and the original goal is re-asserted (not re-derived) so that interpretation does not drift as the tree deepens. The second is corrective: the QA loop (Section 5.5) raises effective per-step reliability through verification and bounded retries. But correction has limits — the critic is itself stochastic, producing false passes and false failures, and every retry multiplies cost. The honest conclusion is that reliable, deeply-recursive agent decomposition is a research-grade problem, not a solved one. The architecture makes it tractable for shallow depth with strong QA; pushing it deeper is precisely the kind of hard R&D that warrants dedicated investment (Section 11).

There is also a cost dimension. Because a single prompt can fan out into many model-driven agents, naïve recursion makes the cost and latency of one request unbounded and non-deterministic — unacceptable for an enterprise that needs predictability. Hard per-request ceilings, depth limits, and codification (which removes repeated reasoning entirely) are the controls that turn an open-ended cost into a bounded one.

7. Self-improvement without compromising auditability

A system that learns from its own failures is attractive, but the naïve implementation — detect a repeated error, fine-tune a model adapter inside the task loop, and use it on the next call — does not work and must not ship. It fails on five independent grounds, and they are worth stating because the failure is instructive.

- **Timescale.** Fine-tuning, even with fast tooling, takes minutes to hours on a GPU. It cannot run inside a live task's retry loop; it is an offline, batch process.
- **Data volume.** A handful of failures is nowhere near enough to adapt a model without overfitting or catastrophic forgetting. Meaningful adaptation needs hundreds to thousands of curated examples.

- Serving complexity. Hot-swapping per-task-class adapters into a live serving path at fine granularity is operationally advanced and risk-laden.
- Auditability. Auto-promoting a self-trained model change with no human gate and no held-out evaluation is an unvalidated autonomous modification — incompatible with the EU AI Act and with this architecture’s own governance plane.
- Wrong layer. Most systematic agent errors are prompt, context, or tool problems, not model-weight problems. Fine-tuning is an expensive, risky fix for a defect that usually lives elsewhere.

The corrected design separates two timescales cleanly. Inline, the response to a repeated reliable pattern is codification — turn it into a deterministic tool (Section 5.6), which is cheaper, faster, and fully auditable. Offline, genuine model gaps feed a separate improvement pipeline: examples are accumulated and curated, an adapter is trained with Unsloth/PEFT, evaluated against the versioned golden sets in Git, approved by a human, versioned in MLflow, and only then deployed deliberately. The system still improves itself — but it does so on the right timescale, with evidence, and under human control. Codification is the primary mechanism; model adaptation is the rare, gated exception.

8. Federated context without a data lake

The federated context model deserves a precise statement, because it is both the architecture’s most distinctive claim and the easiest to overstate. The defensible claim is this: there is no central data lake and no second system of record; source systems remain authoritative. What is centralised is a small graph of relationships and a set of caches — not the data itself. What is explicitly not claimed is “zero data movement”: at query time data transits the network, lives transiently in query workers, and enters the model’s context, and caches hold derived copies with a freshness window.

Aspect	Data-lake model** (rejected)**	Federated model** (adopted)**
System of record	A new central copy	The source — unchanged
What is centralised	All the data	A relationship graph + caches
Freshness	Stale until next ETL	Live fetch; cache under TTL policy
Integration cost	Schemas + ETL + harmonisation	Catalogue + query federation
What moves	Bulk data, continuously	Query results, on demand, transiently
Sovereignty	Concentrated in the lake	Retained at each source

Figure 2. The honest distinction is system-of-record versus derived and transient copies — not the absence of movement.

A note on coverage, in the same honest spirit: federated query reaches the subset of sources that expose clean SQL or GraphQL surfaces. Trino federates across relational stores, object storage, and table formats. (GraphQL federation was removed from the critical path in the v2 review: a third federation mechanism next to Trino and MCP/OpenAPI is justified only where a GraphQL estate already exists.) The long tail of enterprise sources — files, streams, and bespoke systems — still requires governed connectors (Section 5.7). The model is powerful and it is not magic, and saying so is what makes it credible.

Component sovereignty matters most here, because this layer touches the most sensitive data. An earlier iteration placed a young, single-vendor graph database under a restrictive (SSPL-class) license at the centre of this layer — reproducing the very relicensing trap this paper warns against. It has been replaced with a relationship graph that runs inside the existing PostgreSQL footprint under a permissive license, removing both the restrictive-license exposure and a single point of vendor dependency, and reducing the operational surface at the same time.

9. Open-source sovereignty: a license and governance map

The sovereignty claim is only as strong as the licenses and governance behind each component. The following map records, for representative components on the critical path, the license posture and the governance model, and flags where a deliberate substitution was made to protect sovereignty. This is the backbone of the argument — without it, “sovereign” is an assertion. Licenses evolve; these should be re-verified periodically against current sources — the v2 review did exactly that, and three components fell out of the stack as a result (see the final rows).

Component	License	Governance	Sovereignty note
Kubernetes, Argo, KEDA, Cilium, OTel	Apache-2.0	CNCF	Graduated/ incubating; broad multi-vendor base. Foundation of the stack.
vLLM	Apache-2.0	PyTorch/LF ecosystem	Single inference runtime, also serving embeddings; hosted by the PyTorch Foundation since 2025.
LiteLLM	MIT	Vendor-led, OSS	Thin abstraction; low switching cost by design.
LangGraph	MIT	Vendor-led, OSS	Chosen as the single substrate; monitor governance; abstract where feasible.

Component	License	Governance	Sovereignty note
Valkey	BSD-3-Clause	Linux Foundation	The correct sovereign Redis successor after Redis's relicensing.
PostgreSQL, pgvector, Apache AGE	PostgreSQL / Apache-2.0	PostgreSQL GDG / ASF	Permissive, multi-vendor; graph runs in the Postgres footprint.
Trino	Apache-2.0	Trino Software Fdn.	Federation workhorse; broad connector ecosystem.
Kafka (KRaft) + Strimzi	Apache-2.0	ASF / CNCF	Event and audit backbone; ASF multi-vendor governance chosen after the 2025 NATS incident (final rows).
OpenTofu, OpenBao	MPL-2.0	Linux Foundation	The LF forks after HashiCorp moved Terraform and Vault to BUSL in 2023 — IaC target and secrets engine; the precedent this map exists to catch.
Buildah	Apache-2.0	CNCF (Podman tools)	Rootless, daemonless CI builds; entered the CNCF sandbox 01/2025. Replaces archived Kaniko.
LibreChat	MIT	Vendor-led, OSS	Web surface. Replaces Open WebUI after its April 2025 license change.
Granite Guardian	Apache-2.0	IBM (open weights)	I/O guardrail classifier under a real OSI license — Llama Guard's community license is not.
Sigstore, SLSA	Apache-2.0	OpenSSF / LF	Signing and provenance duty for agent-built artefacts; admission rejects the unsigned.

Component	License	Governance	Sovereignty note
OpenFGA	Apache-2.0	CNCF	Zanzibar-style authz; subsumes RBAC, so Casbin was dropped.
Keycloak, SPIFFE/SPIRE, OPA, OpenSearch	Apache-2.0	CNCF / Linux Foundation	Mature, foundation-governed identity, policy, and audit.
FalkorDB (graph)	SSPLv1 — rejected	Single vendor (~10–19 staff)	Not OSI-approved; relicensing-trap risk. Replaced by Apache AGE.
Open WebUI	Custom (non-OSI) — rejected	Single vendor + CLA	v0.6.6 (04/2025) added a branding-protection clause and CLA; replaced by LibreChat.
Kaniko	Apache-2.0 — archived	None (archived 06/2025)	Google archived the repository on 3 June 2025; replaced by Buildah.
NATS	Apache-2.0 — incident	CNCF; trademarks at LF (05/2025)	The 2025 relicensing/trademark attempt by its vendor was resolved, but documents concentration risk. Kept off the audit backbone; acceptable for light pub/sub with a documented exit path.

Figure 3. Representative critical-path components, including the options rejected in review and the documented 2025 governance incidents — each with its replacement.

10. Governance, identity, and regulatory alignment

Agents have power: they read sensitive context, execute code, spend budget, and modify infrastructure. Without governance, a single compromised or mis-spawned agent could exfiltrate or damage at machine speed. The governance plane spans every layer, and every action passes through it.

Identity is two-tiered and kept distinct. Workloads — every agent and service — receive a SPIFFE/SPIRE identity: a short-lived, attested cryptographic certificate. Humans authenticate through Keycloak (SSO, OIDC, multi-tenant realms). An agent acting for a person therefore

carries both its workload identity and a delegated user token, which is what makes attribution exact.

Authorisation is consolidated to two engines rather than three. OpenFGA provides Google-Zanzibar-style relationship-based authorisation — answering “can this workload, with these attributes, in this context, act on this resource?” — and it subsumes role-based access, which is why a separate RBAC engine was removed. OPA handles admission and infrastructure policy. Because authorisation sits on every action, decisions are cached with deny-by-default fallback and explicit latency budgets, so the policy plane is fast and fails safe rather than failing open. Secrets — the gap in the first draft — are served by OpenBao, the Linux Foundation fork of Vault after its 2023 BUSL relicensing, delivered into workloads through the External Secrets Operator. Artefact trust is enforced by Sigstore signatures and SLSA provenance; admission rejects the unsigned. Every action lands in an append-only audit store (OpenSearch, under the Linux Foundation since 2024) with full attribution, and human-in-the-loop approval gates — implemented as orchestrator state, with signed approval records — guard the actions that policy designates as high-risk. A second infrastructure control plane (Crossplane) was deliberately dropped from the default: next to OpenTofu and Argo CD it is justified only at multi-cluster self-service scale.

Regulatory alignment

The architecture is built to be auditable by construction, which is what the regulatory environment demands. Under the EU AI Act, the combination of end-to-end tracing, complete action attribution, and the prohibition on unsupervised self-modification means the system’s behaviour can be reconstructed and explained — and that autonomous changes cannot bypass evaluation and human oversight. Under the Cyber Resilience Act, the GitOps model (every change a reviewed commit), workload identity, sandboxed execution, and a software-bill-of-materials discipline across the open-source supply chain support the secure-by-design and vulnerability-management obligations. Auditability is not a feature bolted on at the end; it is a property of an architecture where every boundary has an identity and every action has a record.

11. Maturity assessment: integration versus innovation

Honesty about maturity is what makes an investment case defensible. The table below assigns an approximate technology-readiness level to each part of the system and states plainly whether it is commodity integration or genuine research. The pattern is the point: the lower layers are mature and are *not* where the novelty lies, while the reliable, auditable, self-improving agent fabric on top is the first-of-a-kind work.

Capability	Approx. TRL**	**Nature
Execution substrate (Argo, KEDA, Kata, OTel)	9	Mature integration
LLM gateway and serving (LiteLLM, vLLM)	8–9	Mature integration
Data, cache, messaging	9	Mature integration

Capability	Approx. TRL**	**Nature
(Valkey, Postgres, Kafka)		
Identity and policy tooling, individually	8–9	Mature integration
Skills, connectors, MCP	6–7	Applied development
Federated query (Trino)	9	Mature integration
Agent-driven federated context** (the integration)**	4–5	Research & development
Per-action agent authorisation** at scale	5–6	Research & development**
Tool-factory** auto-codification	3–4	Research & development**
Bounded reliable recursive hierarchy	3–4	Core R&D — funded innovation
Self-improving** / self-codifying loop, audit-safe**	2–3	Core R&D — funded innovation

Figure 4. TRLs are engineering judgments, not certifications, and should be validated against proofs-of-concept. The bolded rows are the genuine innovation; the rest is the proven foundation it stands on.

This is the case in one paragraph: the proven layers are assembled, not invented, and assembly is not where investment is warranted. The genuinely novel, first-of-a-kind work is the reliable and auditable agent fabric — bounded recursive decomposition with controlled error, a self-improving loop that satisfies the AI Act, and federated context for agents without a data lake — delivered on a fully open, foundation-governed stack with a documented sovereignty posture. Framed as a finished production system, the claim would not survive review. Framed as hard problems solved on proven foundations, it is exactly where focused investment belongs.

12. Deployment and Day-2 operations

The platform deploys on sovereign Kubernetes — bare-metal, a sovereign cloud, or a hyperscaler region — by deliberate choice, and the same manifests run in each because the abstraction is Kubernetes, not a provider. Everything is declarative and GitOps-managed: the cluster's desired state lives in Git, and every change is a reviewed, audited commit reconciled by Argo CD. There are no imperative, untracked changes to production.

Day-2 discipline is designed in, not improvised later. Observability is end-to-end from the first deployment (traces, metrics, logs across every agent action), so incidents are diagnosable rather than mysterious. The deliberate consolidation of the stack — one inference runtime, one orchestration substrate, one GitOps engine, one isolation technology, two policy engines — keeps the upgrade and CVE surface small enough for a focused team to own sovereignly. A software-bill-of-materials is generated and signed (Syft, Sigstore) across the open-source supply

chain, and the license-and-governance map (Section 9) is treated as a living document, because the principal Day-2 risk in an open-source strategy is not a bug but a relicensing event upstream. Watching for that, and keeping critical-path components abstractable, is part of operations.

Two organisational practices complete the picture. Codification (Section 6) is run as a continuous programme, not a one-off: every reliable pattern that becomes a deterministic tool permanently removes inference cost and a class of variability — directly countering the automation paradox by reducing, rather than accreting, the maintenance burden. And the human-in-the-loop gates are staffed and measured, because governance that exists only on paper is not governance.

13. Conclusion: toward a European reference implementation

The agentic era will be built on infrastructure, and the make-or-buy decision has shifted: with agentic AI, a small number of experts can self-manage an entire sovereign stack that would once have required a platform vendor. The opportunity for Europe is not to build a better model in isolation; it is to build better, sovereign plumbing — the open, governed, auditable fabric on which agentic AI can be trusted in regulated industry.

We don't just need better models. We need better, sovereign plumbing — and with agentic AI, a few experts can now self-manage an entire sovereign stack.

The make-or-buy decision has changed

This architecture is offered in that spirit: fully open source on the critical path, governed end-to-end, federated by design, and honest about what is built versus what remains to be solved. Its components are foundation-stewarded and multi-vendor wherever sovereignty is at stake; its hard problems are stated as hard problems. Pursued as a collaborative, upstream-first effort — contributed into the open-source commons rather than forked away from it — it can become a European reference implementation for sovereign agentic AI: a foundation that others build on, audit, and improve, rather than a dependency they inherit. That is what sovereignty, done properly, looks like.

End of white paper. This is an engineering-honest review draft: feasibility-assessed. Reliability and cost figures illustrate mechanisms rather than measured results; TRLs are judgments; and license and project facts should be re-verified against current sources.

Souveränitäts-Kompass Page 1

Appendix A. The sovereignty maturity model — thirteen axes, scored 0-5

Sovereignty is not a switch but a profile, and a profile is only as strong as its weakest link, not its average. The companion self-assessment — the *Souveränitäts-Kompass* — scores a deployment across thirteen axes rather than asserting a single label. Each axis runs from 0 (absent) through 1 (ad hoc), 2 (selective), 3 (defined), 4 (managed) to 5 (sovereign), against

explicit, engineering-honest criteria. Level 0 means the axis is not addressed at all; the criteria for levels 1–5 follow.

Axis 1 — Territorial Data Sovereignty

- **1 • Ad hoc** — Data location unknown or only contractually asserted — no evidence; backups, logs and telemetry uncontrolled.
- **2 • Selective** — Primary data in the intended region, but control plane, backups or telemetry partly reside elsewhere.
- **3 • Defined** — Region-pinning for primary data and backups documented; the main data flows are known.
- **4 • Managed** — All data at rest and in motion — incl. control plane, logs and metadata — provably in the target jurisdiction; audited regularly.
- **5 • Sovereign** — Territoriality technically enforced and independently verifiable (region constraints, attestation) — no data path leaves the jurisdiction, even under failure or support access.

Axis 2 — Open-Source Foundation & Upstream

- **1 • Ad hoc** — Proprietary or relicensable components dominate; open source only incidental, no license clarity.
- **2 • Selective** — Open source used, but purely as a consumer — no contributions, dependence on single vendors.
- **3 • Defined** — OSI-conformant licenses deliberately chosen and documented; first upstream contributions, relicensing risks understood.
- **4 • Managed** — Active contributor role in critical projects; foundation / multi-vendor governance preferred on the critical path.
- **5 • Sovereign** — Maintainer / steward role in key projects under neutral foundation governance; no single-vendor lock-in, roadmap co-shaped.

Axis 3 — Supply Chain & Integrity

- **1 • Ad hoc** — Artifact provenance unknown; no SBOM, no signatures, builds not traceable.
- **2 • Selective** — SBOM or scans spotty for individual components; toolchain integrity not addressed.
- **3 • Defined** — SBOM and vulnerability scans systematic; signature verification introduced, provenance partly provable.
- **4 • Managed** — Signing (Sigstore) and provenance (SLSA) mandatory; builds largely reproducible, toolchain hardened.
- **5 • Sovereign** — End-to-end signed, provenance-checked artifacts; nothing unsigned runs, reproducible builds and independent verification.

Axis 4 — Key Sovereignty & Confidential Computing

- **1 • Ad hoc** — Encryption left to the provider; keys held by the operator, “in use” unprotected.

- **2 • Selective** — “At rest” and “in transit” encrypted, but keys in the provider KMS (BYOK) — access remains delegated.
- **3 • Defined** — Own key management (HYOK) for core data; rotation and access logging defined.
- **4 • Managed** — “In use” protection for sensitive workloads via confidential computing (TEE) with attestation introduced.
- **5 • Sovereign** — Operator excluded from cleartext by design — confidential computing throughout, keys beyond provider reach, independently attested.

Axis 5 — Portability & Provider Switch

- **1 • Ad hoc** — Deep binding to proprietary services; no exit plan, switching cost unknown.
- **2 • Selective** — Data export possible in principle, but coupled to proprietary managed services; never rehearsed.
- **3 • Defined** — Open formats/standards for core data; exit path documented, switching cost roughly estimated.
- **4 • Managed** — Portability proven regularly via restore/migration drills; dependencies mapped.
- **5 • Sovereign** — Provider switch achievable and rehearsed within a defined time; no hard lock-ins, switching cost known in person-days.

Axis 6 — Operational Resilience & Day-2

- **1 • Ad hoc** — Operation hinges on vendor support; no recovery plan, an outage means standstill.
- **2 • Selective** — Backups exist, but restore untested; competence concentrated in a few people.
- **3 • Defined** — Recovery procedures documented and partly tested; monitoring and alerting established.
- **4 • Managed** — Self-healing, rehearsed recovery and redundant competence; Day-2 automation in place.
- **5 • Sovereign** — Provably operable independent of the vendor — rehearsed failures (region/vendor/support), automated self-healing, no competence single points.

Axis 7 — Legal Compliance & Jurisdiction

- **1 • Ad hoc** — Provider jurisdiction unclear or under foreign access law (e.g. CLOUD Act); no assessment.
- **2 • Selective** — EU nexus claimed, but parent company, support or operations fall under foreign jurisdiction.
- **3 • Defined** — Contracting party and data processing in the intended jurisdiction; legal position documented (DPA, TIA).
- **4 • Managed** — Jurisdiction secured throughout, foreign access rights excluded; regulatory risks (lawful access) monitored.
- **5 • Sovereign** — Trust made technically unnecessary — even a future access law compels no cleartext, because no access-entitled operator exists (cf. axis 4).

Axis 8 — Governance & Policy-as-Code

- **1·Ad hoc** — Rules informal or absent; actions are not traceable.
- **2·Selective** — Policies documented, but enforced manually and with gaps; audit incomplete.
- **3·Defined** — First machine-readable policies (OPA/Kyverno) on core paths; audit logs present.
- **4·Managed** — Policy-as-code bites at most actions; immutable, searchable audit logs with attribution.
- **5·Sovereign** — Every action policy-checked and logged immutably without gaps; no uncovered path, zero-trust enforced rather than documented.

Axis 9 — Ownership & Economic Sovereignty

- **1·Ad hoc** — Core technology and derived artifacts belong to the provider; high lock-in premium in TCO.
- **2·Selective** — Data is yours, but models, weights and embeddings arise in a foreign environment with usage rights unclarified.
- **3·Defined** — IP and usage rights to models, fine-tunes and embeddings contractually clarified; TCO transparent.
- **4·Managed** — Essential value creation (models, adapters, pipelines) owned and operated in-house; lock-in premium low.
- **5·Sovereign** — Full economic and legal sovereignty over core technology and artifacts; value-creation depth in-house, no structural lock-in.

Axis 10 — Model Sovereignty & LLM Gateway

- **1·Ad hoc** — Fixed coupling to one proprietary model or API; no switch without rebuilding the application.
- **2·Selective** — Several models usable, but via the provider's own gateway — switching stays costly.
- **3·Defined** — Self-controlled gateway (uniform API, routing, fallback) decoupled from providers.
- **4·Managed** — Critical paths runnable on open, self-hostable weights; model switch without rebuild rehearsed.
- **5·Sovereign** — Full model sovereignty — open weights self-operated, provenance and reproducibility known, no path necessarily tied to an external provider.

Axis 11 — Context & Federated Data

- **1·Ad hoc** — Data copied into a central lake for every use; residency and permissions break.
- **2·Selective** — Centralization dominates, federated access only as an exception; permissions not passed through.
- **3·Defined** — Context partly via graph and references instead of copies; catalogue and lineage present.

- **4 • Managed** — Federated context access is the default — intelligence comes to the data, permissions passed through.
- **5 • Sovereign** — End-to-end “bring intelligence to data”: data stays at source, purpose-bound joining under minimization, no silent copies.

Axis 12 — Agentic Abstraction Layer

- **1 • Ad hoc** — Agents run as unencapsulated scripts, fixed to one model/tool, without safeguards.
- **2 • Selective** — First agents in production, but switching model or provider requires a rebuild; actions barely traceable.
- **3 • Defined** — Models, providers and tools encapsulated; agent actions logged, first approval gates.
- **4 • Managed** — Stateful, isolated agents (e.g. microVMs) with approval gates; switching largely without a rebuild.
- **5 • Sovereign** — Fully governable agent layer — every action secured, traceable, within sovereign bounds; models/tools freely swappable, contract-bounded spawns.

Axis 13 — Reliability & AI Governance

- **1 • Ad hoc** — Individual LLM calls without checks; errors propagate through the chain unnoticed.
- **2 • Selective** — Spot-check manual review; no systematic verification, reliability unknown.
- **3 • Defined** — Automatic checks (critic/eval) on core steps; reliability metrics collected.
- **4 • Managed** — Decomposition into verifiable micro-steps with independent checking; self-QA and calibration established.
- **5 • Sovereign** — Reliability anchored architecturally — every chain built from individually verified steps, recursive self-correction, calibrated and demonstrable confidence rather than hope.