

REFERENCE ARCHITECTURE

Observational Governance

On the KanTec Continuum. How an enterprise designs, builds and deploys a closed loop in which policy is true while systems and agents act.

Senior IT architecture · KanTec Platform Engineering · September 2026

1. Governing thought

Observational Governance is the control architecture of Continuum. Continuum is not a catalogue of tools. It is the operating model that makes policy observable in production.

Situation. Enterprises already own policy documents, GRC platforms, identity, pipelines and monitoring. Agents and models now act inside those estates.

Complication. Policy lives in documents. Operations live in clusters. Almost nobody can prove that intent is true at the moment of action. That gap is the constraint on regulated AI, not model quality.

Question. What architecture makes governance observable — design, build, deploy — without standing up a second estate?

Answer. One control plane, one evidence contract, one eight-step loop, delivered through Continuum's six domains, deployed as a beachhead then cloned. Partner products are replaceable implementations of that contract.

2. Scope and non-goals

In scope

- Logical and control-plane architecture for Observational Governance.
- Object model: system, model, agent, tool, policy, control, outcome, evidence, exception.
- Design / build / deploy method on Continuum.
- Acceptance tests that an architect can fail a programme against.

Out of scope

- SKU-level bills of materials or list prices.
- Agentic Opportunity Engine (revenue orchestration). Different product.
- Estate-wide transformation programme design.
- Legal opinion on any single regulation.

3. Architectural definition

Observational Governance: a property of a production system in which (a) every consequential actor is registered, (b) policy is executable at the point of action, (c) action emits evidence, and (d) evidence can change policy without a separate audit project.

If any of those four clauses is missing, the organisation has oversight theatre. Continuum exists to make all four true on one path, then copy the path.

4. Logical architecture — five planes

Planes are not organisation charts. They are contracts. A component may sit in two planes; it may not sit in none.

Plane	Contract
Authority	Appetite, stop rights, exception rights. Named humans. Not a committee minute.
Control plane	Registry, policy-as-code, PDP/PEP, exception bus, evidence store, decision UI.
Capability plane	Continuum six domains. Each domain may only emit outcomes the observability contract can see.
Runtime plane	Applications, models, agents, pipelines, infrastructure. Subjects of policy, not exceptions to it.
Evidence / decision	Outcomes measured, risk evaluated, insights issued, policy evolved. The loop closes here.

5. The loop is the sequence diagram

Governance defines policy → policy drives controls → controls drive operations → operations generate outcomes → observability measures outcomes → risk and compliance evaluated → insights inform decisions → governance evolves.

Architecture rule: each arrow is a system interface with a schema. If the interface is email or a quarterly workshop, the loop is broken. Owner, system of record and SLA sit on every arrow.

6. Object model

These objects are mandatory on the beachhead. If an object is missing, do not scale.

- Actor — user, service account, model, agent, tool. First-class identity and scope.
- Asset — system, dataset, model artefact, prompt pack, pipeline.
- Policy — intent record + executable artefact, same version.
- Control — binding of policy to a PEP (admission, gateway, tool-firewall, pipeline gate).
- Outcome — runtime event: allow, deny, exception, drift.
- Evidence — immutable tuple of outcome + artefact hashes + actor + policy version.
- Exception — time-boxed override with owner and expiry. Expiry is a deny.

Agents are not a special class of application. They are actors that call tools. Tool calls without an allowed scope fail closed.

7. Continuum capability mapping

Domains implement the contract. They do not replace it. Reference implementations below are patterns, not mandated vendors.

Domain	Pattern	Observational duty
AI Governance	watsonx.gov-class	Register models and agents. Risk tier. Human gate on consequence.
Integration	webMethods-class	Every interface is attested. No dark integration.
Platform Engineering	OpenShift-class	Policy in the developer path. No bypass cluster for 'AI'.
Secure delivery	Chainguard / OX-class	Signed images, SBOMs, versioned prompts and agents.

Sovereign infrastructure	Jurisdiction-class	Residency and operator access are controls, not footnotes.
Observability	Instana-class	Traces, SLOs and policy-deny events in one query surface.

8. Design — seven decisions that must be taken before build

- Policy dual-form: natural language and executable artefact share a version. One cannot ship without the other.
- Identity of everything: models and agents join IAM. Service accounts are not a hiding place.
- Fail closed on consequence: payments, identity grants, external send, model promotion.
- No silent path: no trace + no attestation = no deploy.
- Evidence by construction: the audit pack is a query over the evidence store.
- Exception is an object: owner, scope, expiry. Calendar expiry is a control.
- Evolve from production: a policy merge without a linked outcome is rejected.

9. Build — minimum control plane

Do not build a platform programme. Build these eight components on the beachhead.

- Registry — authoritative list of actors and assets.
- Policy repository — Git, signed releases, reviewable diffs.
- PDP / PEP — admission controllers, API gateways, agent tool-firewalls.
- Pipeline gates — build, sign, SBOM, promotion. Unsigned never leaves.
- Runtime telemetry — application traces and agent spans in one backend.
- Evidence store — append-only control results and artefact hashes (schema: Appendix A).
- Exception bus — break-glass as data, not a Slack thread.
- Decision UI — operator workbench plus an executive surface that reads the same store.

Reference deployment for most Continuum beachheads is the OpenShift golden path: policy lint in CI, cluster PEP, signed supply chain, unified deny+trace store, evidence query, policy PR linked to a spike.

10. Deploy — beachhead method

Wave one is one AI system, one regulated process, or one platform path. Not the estate.

- Day 0 — name stop authority and exception owner. If unnamed, stop.
- Days 1–30 — registry populated for the beachhead; policy dual-form for that path only.
- Days 31–60 — PEP live; unsigned artefact cannot land; agent tool-call without scope fails closed.
- Days 61–90 — weekly govern review; evidence query produces the pack; first policy merge cites a production signal.

Scale rule: clone the path. A second control plane ‘for AI’ is an architecture defect.

11. Operating model

Board / CRO owns the objective and stop right. CISO / CDO owns policy intent. Platform engineering owns the control plane and golden path. SRE owns the telemetry contract. Risk owns continuous test queries. Product and AI owners own registration. Nobody owns ‘governance’ as a slide.

12. Architecture acceptance tests

Fail the programme if any test is red on the beachhead at day 90.

- Every beachhead workload has registry ID, owner and risk tier.
- Policy deny events join application traces in the same store within an agreed SLO.
- Unsigned artefact cannot reach the beachhead runtime.
- Unscoped agent tool-call fails closed.
- Expired exception is a deny.
- Audit pack is generated from the evidence store in one query.
- Every policy merge in the quarter cites at least one production signal.
- Bypass rate off the golden path is measured weekly and falling.

13. Implications for KanTec

Continuum is the language and the capability map. Observational Governance is the architecture those capabilities must satisfy. Platform Engineering builds the path. Observability proves the path. Partners supply implementations. If a partner product cannot emit the evidence contract, it does not sit on the beachhead.

Commission the beachhead. Do not commission another framework document.

Detail for the evidence store and PEP/PDP contract is Appendix A.

Appendix A. Evidence schema and policy enforcement engines

This appendix is normative for the beachhead. Schema and engines are one design viewed from storage and from runtime.

A.1 Why these two pieces

Observational Governance fails in two ways. Either the enforcement point is missing, so policy is a PDF. Or the evidence record is missing, so last quarter's workshop is the audit pack.

A.2 Evidence schema — design rules

- Append-only. Corrections are new records that supersede.
- Content-addressed artefacts. Hashes in the hot row; files in object storage.
- Policy version is mandatory. An allow without policy_version is not evidence.
- Trace join key is mandatory.
- One schema for humans, services, models and agents. Actor type is a field.

A.3 Canonical evidence_event

```
evidence_event {
  evidence_id      UUID
  recorded_at      timestampz    // store clock, UTC
  occurred_at      timestampz    // actor clock, UTC
  action           enum          // allow | deny | exception | drift | promote
  actor { id, type, idp, scopes[] } // user|service|model|agent|tool
  asset { id, type, owner, risk_tier }
```

```

policy { id, version, hash, dual_form_uri }
control { id, pep_id, pep_class } // admission|gateway|tool_fw|pipeline
decision { result, reason_code, input_hash }
exception { id, owner, expires_at }
artefacts[] { kind, hash, uri }
trace { trace_id, span_id, vendor }
tenant / residency
}

```

input_hash is the hash of the PDP input document. Replay the decision without storing raw PII in the evidence row.

A.4 Projections

- Control register — latest pass/fail per control_id.
- Actor timeline — all events for one agent or model.
- Exception watch — live, time-to-expiry, expired-but-still-allowing (must be zero).
- Policy lineage — evidence_ids cited by a policy merge.
- Audit pack — time-bounded query: asset + policy_version + artefacts + denials.

Hot path: append log with idempotent evidence_id. Cold path: object store. Index: warehouse. Do not make the SIEM the system of record.

A.5 PDP and PEP

PDP evaluates {actor, asset, action, context} and returns allow, deny, or allow-with-obligation. PEP interposes on the real path and forces the decision. A logger is not a PEP. Consequential action plus PDP timeout equals deny.

Class	Where	Forces
Pipeline	CI / CD / registry	Unsigned image, missing SBOM, unregistered agent artefact cannot promote.
Admission	Cluster / scheduler	Workload without registry ID or policy label cannot land.
Gateway	API / integration edge	Unattested interface or out-of-scope consumer is rejected.
Tool-firewall	Agent runtime	Tool call outside allowed scopes fails closed.
Data	Query / retrieval layer	Residency and class controls on what a model or agent may read.

A.6 Decision contract

```

decision_request { actor, asset, action, context, policy_pin? }
decision_response { result, policy_version, reason_code, obligations[], input_hash, ttl_ms }

```

policy_pin evaluates the version that will ship. Human-gate is an obligation. Break-glass is an exception object on the exception bus, not a cluster-admin kubeconfig.

Continuum does not require one engine product. It requires one decision contract and one evidence write from every PEP class.

A.7 Lock and beachhead order

Every PEP write is an evidence_event. A policy PR must list evidence_ids that justified the change.

- Week 1-2 — freeze evidence_event; stand append log and hash store.
- Week 3-4 — pipeline PEP + admission PEP on one OpenShift path.

- Week 5–6 — gateway PEP; join traces.
- Week 7–8 — tool-firewall on the first agent; prove the audit query.

A.8 Appendix acceptance tests

- Replay: input_hash + policy_version returns the same PDP result.
- Join: 99% of beachhead denials have a resolvable trace_id.
- Expiry: synthetic expired exception is denied within PEP TTL.
- Pack: one query returns 30 days of beachhead evidence with artefact hashes.

If replay or join fails, Observational Governance is not deployed. Monitoring is.

— KanTec Continuum · Observational Governance reference architecture —