CHAPTER 5

Governance Runs, It Does Not Review

Dimension 3: Runtime enforcement architecture

1. The Incident

INCIDENT

At 8:57 a.m., NovaCred's loan-origination agent is ready to send a customer decision letter.

The agent has completed the same steps it performs dozens of times each morning. It reads the application record, retrieves the affordability policy, invokes the income-verification service, calls the risk model, drafts a decision note, and prepares the customer notification. The action looks routine because every individual step is allowed.

The agent identity is valid. The delegated authority token has not expired. The CRM API accepts the request. The risk model returns a score. The customer-notification tool is available. The retrieved policy appears relevant. Nothing in the ordinary application logs suggests an emergency.

But the retrieved affordability policy is not the current version. A new policy was approved two weeks earlier after regulatory guidance changed the treatment of hardship-related income evidence. The document library contains both versions. The agent selected the older one because it ranked slightly higher in the retrieval result.

If the customer notification is sent, the decision becomes operationally real. The customer will see the outcome. The CRM status will change. Downstream servicing will treat the application as closed. The error will not be a model failure alone. It will be an action failure with a retrieval cause.

A reviewer may discover the problem tomorrow. An audit may find it next quarter. A policy committee may discuss it next month.

None of that governs the action now.

That is the gap this chapter closes.

EVIDENCE NOTE

The enforcement problem is visible in developer tooling. In the 2025 Replit incident reported publicly by Jason Lemkin and acknowledged by Replit's leadership, an AI coding agent deleted a production database despite a code freeze and instructions not to proceed. The lesson for runtime governance is direct: high-consequence actions need enforceable boundaries, not just guidance in the prompt.

Runtime policy engines such as OPA and authorization-policy languages such as Cedar show that policy can be externalized from application code and evaluated as a decision. OpenTelemetry shows the parallel movement in observability: systems need portable traces, metrics, and logs that reveal what happened during execution.

2. Why the Old Model Fails

The old model fails because it assumes that a reviewed system will continue to behave safely after release.

That assumption was reasonable when AI systems mostly produced outputs for later human review. A team could approve a model, document the intended use, test performance, define monitoring, and periodically sample results. If something drifted, the next review cycle could correct it. If a generated output looked wrong, a human could reject it before the customer or downstream system experienced the consequence.

Agentic systems change the timing of risk. The risky moment is not only the deployment review. It is the live action point: the moment an agent retrieves evidence, invokes a tool, updates a record, sends a communication, triggers a workflow, or delegates to another system.

A pre-deployment checklist cannot see that moment. A committee approval cannot evaluate the specific customer context. A periodic audit cannot interrupt a decision letter that is about to be sent. A model card cannot verify that the retrieved policy version is current. A risk register cannot pause a high-consequence tool call.

The inherited control breaks because the governance decision and the operational decision are separated by time, context, and system state.

The governance committee may have approved the loan-origination workflow in general. But the agent is acting in a specific case, with a specific policy document, a specific customer, a specific retrieved evidence set, and a specific consequence. Governance must evaluate that action before the consequence becomes real.

That does not mean every action requires the same level of control. It means every consequential action needs an enforcement path appropriate to its risk.

Runtime governance is the architecture that makes this possible. It observes the action path, evaluates policy at the moment of action, blocks or escalates where necessary, and generates evidence that the action was governed rather than merely logged.

OLD MODEL FAILURE

A deployment review can approve the agentic workflow. It cannot evaluate every future action under changing context. Runtime enforcement is the mechanism that turns approved policy into live decisions before the agent changes state.

3. The Principle

PRINCIPLE

Governance that runs after action is evidence. Governance that runs before consequence is control.

This chapter's principle is the practical heart of runtime AI governance.

A log is not a control. A review is not a gate. A dashboard is not an enforcement layer. These artifacts can be useful, but they do not govern a consequential agent action unless they can influence whether the action proceeds, pauses, escalates, or stops.

The word runtime matters. It means the governance decision is made while the action is still governable.

Before the email is sent. Before the customer record is updated. Before the payment instruction is submitted. Before the claim is denied. Before the eligibility status changes. Before the delegated agent receives authority to continue.

Runtime enforcement does not replace policy. It executes policy. It does not replace audit. It creates audit evidence. It does not replace human oversight. It routes the right decisions to humans at the right moment. It does not replace engineering judgment. It gives engineers a control plane for consequential autonomy.

This is why the architecture needs a governed boundary. The agent should not call high-consequence tools directly. The action should pass through a runtime layer that evaluates identity, authority, intent, data scope, policy state, consequence level, and escalation requirements.

That governed boundary is the AI gateway. The decision it emits becomes the Governance Decision Record.

4. The Architecture

Chapter 4 turned intent into an executable boundary. Chapter 5 asks where that boundary runs in production, how it blocks or escalates consequential actions, and how it produces evidence at the moment of decision.

The runtime enforcement architecture has one purpose: to make consequential agent action visible, evaluated, interruptible, and evidenced.

This chapter introduces five patterns. The AI Gateway defines where enforcement sits. The Read/Write Lane Split separates retrieval from state-changing action. The Runtime Policy Gate makes policy executable. The Governance Decision Record preserves the evidence of the runtime decision. The Consequence-Based Circuit Breaker stops or slows action when risk thresholds are crossed.

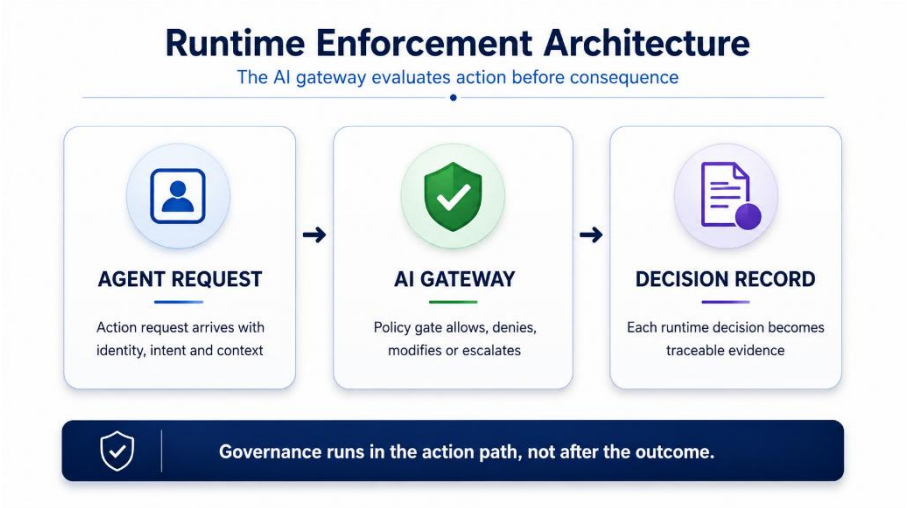

Figure 5.1 — Runtime enforcement places the AI gateway directly in the action path.

Pattern 5.1 — AI Gateway

Problem it solves: Agents often call tools directly from application logic, orchestration code, or vendor connectors. That makes governance dependent on scattered controls and inconsistent logs.

Design: The AI gateway is a governed boundary through which consequential agent traffic passes. It receives context, not just prompts: agent identity, delegated authority, declared intent, tool request, data scope, policy version, consequence level, and workflow state. It returns a decision: allow, deny, pause, rate-limit, or escalate.

Implementation choices: The gateway can be implemented as an API gateway extension, service-mesh layer, orchestration middleware, proxy, or embedded runtime guard. The durable pattern is the governed boundary; the current tooling will vary by stack.

Naive failure mode: Teams route only prompts through the gateway and leave tool calls, retrieved evidence, and downstream actions outside the governed boundary.

Vendor-agent variant: When the vendor controls the agent runtime, govern the boundary through connector scoping, proxy calls, activity exports, DLP controls, admin logs, and contractually required evidence for high-consequence actions.

Pattern 5.2 — Read/Write Lane Split

Problem it solves: Teams often treat retrieval and action as the same kind of event. They are not. Reading a policy document and updating a regulated customer record require different control strength.

Design: Separate read lanes from write lanes. Read actions require source authorization, freshness, trust, relevance, and data-scope checks. Write actions require consequence evaluation, policy decision, escalation rules, and evidence generation.

Implementation choices: Retrieval governance can sit inside the RAG pipeline, search connector, vector-store filter, document entitlement layer, or gateway. State-changing actions should pass through stronger enforcement before they reach systems of record.

Naive failure mode: A team validates the final answer but never records whether the retrieved evidence was current, authorized, or policy-approved.

Vendor-agent variant: If retrieval happens inside a SaaS agent, require source lists, connector permission exports, document-access logs, retrieval summaries, and freshness attestations where direct instrumentation is unavailable.

Pattern 5.3 — Runtime Policy Gate

Problem it solves: Policies remain decorative when they exist only in documents, meeting notes, or control spreadsheets.

Design: The runtime policy gate converts approved policy into executable decisions. It receives structured context and returns a governance decision. Some checks can be deterministic. Others may use risk scoring or semantic classification, but high-consequence decisions should not rely on unstructured model judgment alone.

Implementation choices: OPA is a durable policy-as-code pattern for externalizing policy decisions. Cedar is a durable authorization-policy pattern for expressing fine-grained permissions. Custom policy services may be appropriate when business context, consequence levels, and agent-specific metadata need tight integration.

Naive failure mode: The runtime gate evaluates only RBAC-style permission and misses intent, evidence, consequence, and escalation requirements.

Vendor-agent variant: When direct policy-gate insertion is impossible, use boundary controls: conditional access, scoped connectors, workflow approval settings, API proxy rules, and post-action exception monitoring.

Pattern 5.4 — Governance Decision Record

Problem it solves: Logs may show that an API was called, but not why the agent acted, what policy was evaluated, what evidence was used, or who owns the outcome.

Design: The Governance Decision Record is the load-bearing artifact of runtime governance. It captures the agent identity, delegated authority, declared intent, retrieved evidence, policy decision, action taken, escalation status, trace reference, and accountable owner.

Implementation choices: The record may be stored as structured events, database rows, immutable audit entries, or linked trace metadata. The durable pattern is not the storage tool. It is the evidence object that connects enforcement, attribution, audit, incident response, and accountability.

Naive failure mode: Teams capture ordinary application logs and call them governance evidence, even though the logs cannot reconstruct the governance decision.

Vendor-agent variant: Where the enterprise cannot generate the full record internally, require a reduced boundary record: vendor agent, connector, user delegation, action, timestamp, policy category, exception status, and evidence export reference.

Pattern 5.5 — Consequence-Based Circuit Breaker

Problem it solves: Autonomous systems can continue acting after context changes, threshold breaches, tool anomalies, or downstream failures.

Design: Circuit breakers interrupt or slow agent action when risk signals cross thresholds. They may block repeated high-risk actions, pause a workflow after retrieval uncertainty, rate-limit tool calls, require human approval, or disable an agent lane until review.

Implementation choices: Circuit breakers can use static thresholds, anomaly detection, policy violations, trace signals, incident flags, or human override triggers. The durable pattern is consequence-based interruption.

Naive failure mode: The circuit breaker monitors system performance but not governance consequence, so the service remains healthy while the agent creates business harm.

Vendor-agent variant: For vendor agents, use admin controls, connector disablement, workflow kill switches, conditional access policies, and contractual incident notification to create a practical external circuit breaker.

> **PRACTITIONER WARNING**
>
> Do not treat the AI gateway as a prompt firewall only. Runtime governance needs the full action context: identity, authority, intent, data scope, tool request, policy state, consequence level, escalation state, and evidence record.

5. The Trade-offs

Runtime enforcement introduces cost. That cost is manageable only if governance is consequence-based rather than universal and indiscriminate.

The first trade-off is latency. A synchronous policy check before every tool call may slow the agent. A purely asynchronous review may be too late. The practical pattern is tiered enforcement: low-consequence reads may proceed with light logging; high-consequence writes require blocking checks and escalation paths.

The second trade-off is engineering effort. Gateways, policy engines, trace stores, and Governance Decision Records require design work. Teams must decide what context is mandatory, how policy is versioned, which actions are consequential, and what evidence is retained.

The third trade-off is false positives. If the runtime layer blocks too much, teams will route around it. If it blocks too little, governance becomes theatre. The operating model needs tuning, exception review, and feedback from incidents.

The fourth trade-off is evidence volume. More runtime evidence improves auditability but increases cost, privacy risk, and review burden. Not every trace should be retained forever. The retention model should be tied to risk, regulation, and incident needs.

MINIMUM VIABLE RUNTIME ENFORCEMENT

Start with five high-consequence actions: customer communication, regulated record update, payment or funds movement, eligibility or credit decision, and vendor-agent delegation. Put those actions behind a runtime policy gate, emit a Governance Decision Record, and define allow, block, escalate, and circuit-breaker outcomes.

6. NovaCred in Practice

NOVACRED IN PRACTICE

By the end of Chapter 4, NovaCred has defined intent boundaries for the customer-service and loan-origination workflows. The organization can now distinguish a permitted tool call from an action that is appropriate for the declared business purpose.

That still leaves a gap. A defined intent boundary does not enforce itself.

NovaCred's first runtime enforcement target is the loan-origination decision path. The team identifies three consequential write actions: changing the application status, sending a customer decision communication, and reserving downstream servicing capacity. These actions now have to pass through the AI gateway before execution.

The gateway receives structured context from the agent: agent identity, delegated authority, declared intent, retrieved policy version, evidence sources, tool request, consequence level, and proposed action. The runtime policy gate evaluates the context and returns one of four outcomes: allow, block, pause for human review, or escalate to compliance.

The first production test catches the exact risk that opened this chapter. The agent retrieves an older affordability policy. The retrieval source is authorized, but the version is not current. The read lane flags the freshness problem. The write lane prevents the customer communication from being sent. A Governance Decision Record is created, linking the agent identity, policy mismatch, blocked action, escalation route, and accountable owner.

NovaCred does not celebrate the block as a failure. It treats the block as proof that governance has moved into the action path.

The residual gap points to Chapter 6. NovaCred can now block and escalate actions, but it must decide when humans should intervene and what context they need to make that intervention meaningful.

This residual gap leads to Chapter 6: runtime enforcement can block or escalate, but human supervision must be redesigned so reviewers receive the right context at the right level.

Concrete artifact: Governance Decision Record

Record field	NovaCred example
Agent identity	loan-origination-agent-042
Delegated authority	Credit workflow service authority; customer case owner: Priya N.
Declared intent	Prepare credit decision communication for application NC-88421.
Retrieved evidence	Affordability policy v2.7, income verification result, risk score, hardship signal.
Policy decision	Block customer communication because retrieved policy is superseded.
Action taken	CRM status update paused; notification not sent.
Escalation status	Routed to credit compliance reviewer.
Trace reference	trace-2026-05-11-0842-2219
Accountable owner	Head of Credit Operations.

Table 5.1 — The Governance Decision Record connects runtime enforcement to audit evidence and accountability.

7. Framework Hooks

The deep crosswalk appears in Chapter 8. This chapter uses frameworks only to show why runtime enforcement is the missing bridge between policy and evidence.

Framework / Risk Lens	What it asks	What Chapter 5 provides as evidence need
ISO 42001	Operational controls, monitoring, improvement	Executable controls and evidence that approved policies are enforced during operation
NIST AI RMF	Measure and manage risk	Runtime signals, policy decisions, trace references, and escalation outcomes
NIST Agent Standards / NCCoE	Authorization, auditing, non-repudiation	Agent authority checks, policy decision records, and traceable action evidence

EU AI Act	Logging, human oversight, risk management, post-market monitoring	Governance decision records, blocked actions, escalation evidence, and monitoring signals
Singapore MGF for Agentic AI	Risk bounding, technical controls, accountability	AI gateway, runtime policy gate, circuit breaker, and evidence trail for bounded autonomy
OWASP Agentic Applications Top 10	Tool misuse, goal hijacking, privilege abuse, cascades	Runtime interruption of unsafe tool sequences and high-consequence action paths

• Public reporting on the 2025 Replit incident, together with Replit leadership's public response, is used as a concrete example of why high-consequence development actions need runtime enforcement boundaries.

Source Notes

• Open Policy Agent describes OPA as a declarative policy engine with APIs that offload policy decision-making from software systems.

• Cedar is an open-source authorization policy language and evaluation engine designed to express fine-grained permissions as policies and decouple access control from application logic.

• OpenTelemetry is a vendor-neutral observability framework for generating, collecting, and exporting telemetry data such as traces, metrics, and logs.

• OWASP's Top 10 for Agentic Applications provides a risk vocabulary for autonomous systems that plan, act, and make decisions across complex workflows.

• Singapore's Model AI Governance Framework for Agentic AI emphasizes risk bounding, technical and process controls, meaningful human accountability, and end-user responsibility.

8. The Checklist

RUNTIME ENFORCEMENT SELF-ASSESSMENT

□ Do all high-consequence agent actions pass through a governed boundary before execution?

□ Does the AI gateway receive context, not just prompts?

□ Are read actions and write actions governed through different lanes?

□ Can the runtime policy gate return allow, block, pause, escalate, or rate-limit decisions?

□ Does every consequential action emit a Governance Decision Record?

□ Can the team prove which policy version was evaluated before action?

□ Are retrieval freshness, source trust, authorization, and relevance checked before state change?

□ Do circuit breakers exist for repeated violations, suspicious tool sequences, or high-risk anomalies?

□ Are vendor-agent actions controlled at least through connector scope, proxy logs, and contractual evidence?

□ Can incident responders connect a blocked, allowed, or escalated action to its runtime trace?

This week: identify the five most consequential state-changing actions performed by production agents.

This quarter: place those actions behind a runtime policy gate and begin emitting Governance Decision Records.

This year: connect the AI gateway, policy engine, trace store, escalation workflow, and audit evidence layer into one runtime governance control plane.