

The AI Agent Governance Playbook

What to Enforce When
Guardrails Aren't Enough

Arcade

Executive summary

Enterprises have governed people for decades. Who can approve a purchase order, who can see a salary, who signs off once the amount crosses a threshold, that logic already lives in your identity provider, your ERP, and your CRM. The instinct when AI agents arrive is to point them at the same systems and let them inherit the same rules.

The primitives do carry over. The logic on top of them does not, because the actor changed. Every governance system you own assumes the entity taking an action and the person accountable for it are the same. An agent breaks that assumption. It acts on someone's behalf, so for the first time the actor and the accountable party come apart.

That single shift drives four claims this playbook illustrates:

1. The moment an agent acts guardrails no longer suffice.

Reasoning and text can be shaped by guardrails. An action is a transaction, and a transaction needs authorization, policy, and audit.

2. The controls you already trust sit in the wrong place to govern an agent's actions. They check after the fact, at human speed, and your systems of record have no native way to enforce a rule against an agent at the moment it reaches for a tool.

3. **The highest-leverage place to govern is the tool call, and governance has to be married to the actions runtime to enforce at that point.** A control at the tool call sits outside the model, so a prompt injection cannot talk its way past it, and it sits in front of the target system, so when it denies an action, the action never reaches that system.
4. **Approve the foundation once and the return compounds.** When authorization, policy, and audit are enforced at the one runtime every action passes through, security approves that runtime a single time, and every new agent, user, and tool inherits the approval instead of waiting in its own review queue. That is what turns governance from a tax into the unlock for ROI: you clear the security gate once, then ship workflow after workflow against it.

It all reduces to one question: can this agent, on behalf of this user, take this action, right now? The rest of this document is about how to answer that question, how to map the answer onto the stack you already run, and how to prove what every agent did.

Part 1

Why AI agent governance is a different problem

Ulani Freight's first serious agent was an onboarding agent. Hand it a signed customer contract and it would set up everything a new account needs: the account and opportunity in Salesforce, the implementation epic and tickets in Jira, a project space in Notion, a kickoff note to the account team in Slack. In the demo it was the best thing the AI team had ever shown. Dozens of manual steps, gone. Then someone in the room asked who was accountable the day it did something wrong, in production, at three in the morning, and the room went quiet. The agent did not go to production. It went into a holding pattern that lasted five months.

That holding pattern is the subject of this playbook, and it is not a company problem. It is the shape of the whole market.

AI agent governance is not governing people

Strip any governance decision down to its parts and you find the same four variables, whether the actor is a person or a piece of software.

1. Who is acting?
2. What are they permitted to do?

3. Which resource are they touching?
4. When are they touching it?

A policy is logic written over those four, and nothing about an agent changes those primitives. This is why the question at the center of AI agent governance sounds like something your access model has always asked. Role-based access control asks it. Your approval chains ask it. Your segregation-of-duties rules ask it.

The vocabulary transfers. Identity, permission, policy, audit, all of it still holds. What does not transfer is the logic that sits on top, because a person acts for themselves and an agent acts for someone else.

When you govern a person, the actor and the accountable party are the same. The analyst who runs the report answers for their work. Permission and responsibility sit on the same shoulders, which is exactly why "is this user allowed?" is a sufficient question.

An agent splits that in two. The agent takes the action. A human carries the consequence, either the operator who deployed it or the user it acted for. No amount of access-control configuration puts them back together, because that model was written for a world where they never came apart.

Governing a person vs. governing an AI agent

	Governing a person	Governing an AI agent
Actor and accountability	The same party. The person acts and answers for it.	Split. The agent acts, a human answers for it.
The question that suffices	Is this identity allowed?	Can this agent, for this user, take this action, right now?
Permission lifespan	Relatively stable, reviewed on a cycle	Per task, scoped to the action, expired when it ends
Where the decision lives	Access reviews and approvals, often after the fact	At the tool call, before the action lands
A single mistake	Contained to one person's authority	Moves with whatever access the agent was given

AI Agent Impact: the operator owns what the agent does. In 2024, Air Canada argued before a tribunal that its own support chatbot was a separate entity, responsible for its own statements. The tribunal rejected that outright and held the airline accountable for a refund policy the bot had invented, ordering it to pay. The bot was not hacked. It made something up, and the company was bound by the result. [\(Forbes\)](#)

The accountability gap that stalls approvals

Watch how credit and blame move and the deterrent comes into focus. On the happy path the agent looks great and everyone celebrates the productivity. But when there's a slip, the agent answers for nothing. It does not get a warning or lose its job. The consequence rolls uphill to a person, the operator who put it into production, and the executive who signed off.

That asymmetry is a rational reason to hesitate, and it shows up in the numbers. As of the 2026 field data, roughly **81% of teams have moved agents past planning into testing or live use, while only 14% report full security sign-off** (Gravitee, *State of AI Agent Security 2026*). That gap is not the product of slow-moving security teams. It is the accountability gap, showing up as a stalled approval. It is the exact thing that happened to Ulani's onboarding agent.

You hear it in the field in plainer language. There are two verbatim quotes from recent customer calls.

"We're asking, which agent took which action on behalf of which system? No one knows, and there's nothing to prove it."

"The users can run their own MCPs if they find them on the internet. We don't have any controls to stop them."

The first is the accountability gap. The second is what happens when the gap stays open and the work routes around the queue.

Analysis versus action: where the new problem starts

There is a clean line between what an agent thinks and what an agent does, and it is the line where this whole problem begins. Reasoning and text can be shaped by guardrails, and those parameters are useful. Tone, format, refusals, staying on topic, guardrails handle all of it.

The moment an agent stops analyzing and starts acting - when it sends the payment, deletes the record, or closes the ticket - it produces a transaction. A transaction needs three things enforced at the moment it happens:

- 1) Authorization (is this allowed?)
- 2) Policy (does it satisfy the rules?)
- 3) Audit (is it recorded?).

None of those are questions about language. They are questions about action, and they are where guardrail territory ends.

Why the controls you already trust sit in the wrong place

There are two reasons human-level controls can't be applied directly to agents, and Ulani ran into both.

The first is timing. Approval workflows and access reviews were built for human employees working at human speed, and they check work after it happens. An agent fires dozens of actions a second. Anything that checks after the fact means writing the incident report instead of preventing the incident.

The second is shape. Your systems of record do not speak agent. Consider Workday. The governance sitting inside it is yours, built up over years. Which roles can see compensation, who signs off on a termination, which fields on a worker record are restricted and to whom. Workday can tell you whether a recruiter may view an offer. It has no native way to enforce that same rule against an agent acting for that recruiter, at the moment the agent reaches for the tool, scoped to that one call. Every system of record can ship an MCP server to expose itself to agents, and they will. But exposure is not enforcement. Someone still has to translate your specific policies into controls that hold at the tool call.

As one financial-services leader put it, the entitlement system is the policy store, and the runtime is the policy enforcement. The governance you spent years building is not wasted. It needs a place to be enforced against a new kind of actor.

The layers of governance, and where the leverage is

You can govern an agent at more than one layer, and seeing the whole stack shows where the return is.

At the top is the model itself, aligned during training. Below that, prompt-level guardrails. Below that, your application logic. Below that, identity and access. Then the tool call, where the agent actually invokes a system. Then the network underneath everything. Each layer catches a different class of problem, but they are not equal in leverage.

“As soon as you have questions about agent action, the guardrail territory ends.”

The layers of governance

Layer	What it governs	What defeats or limits it	Leverage
Model alignment	The model's default behavior	Prompt injection, jailbreaks, a confusing input	Low. Advisory and probabilistic.
Prompt / guardrails	Reasoning, tone, text	The same manipulation that steers the model	Low. Advisory and probabilistic.
Application logic	What one app allows	Only covers that app; every client reimplements it	Medium, but not portable
Identity & access	Who holds broad access	Standing grants an agent can over-use	Medium. Necessary, not sufficient.
Tool call / action layer	The specific action, before it lands	Only a bypass at the action itself	Highest. Deterministic, and every model and framework passes through it.
Network	Packets, hosts, ports	Cannot tell a legitimate action from a reckless one	Low for intent. Sees traffic, not meaning.

Govern too high, at the model or the prompt, and your control is advisory and dangerously probabilistic. Govern too low, at the network, and you can block a host but you cannot tell a legitimate action from a reckless one, because the network sees packets, not intent.

The tool call is the point where intent becomes consequence. It sits outside the model, so an injection cannot dissolve it. It sits before the target system, so a denied action never lands. And one control there covers every model, framework, and client above it, because they all pass through the same door to do anything. Most of your governance return comes from the layer where the action happens.

Why governance has to be married to the actions runtime

Governance that sits apart from the actions runtime can only do two things. It can advise before the fact, which is the guardrail you already watched an agent jump. Or it can report after the fact, which is a log you read once the money has left. Both are useful. Neither stops the action.

Governance that sits inside the runtime, on the single path every tool call travels, can evaluate the specific action and refuse it, in real time, before it reaches the system. Advisory becomes enforced. Probabilistic becomes deterministic. That is the difference between a control and a record, and it is why the rest of this playbook treats the runtime as the place governance has to live.

Part 2

What to enforce at the action layer

Ulani's platform engineer spent three weeks hardening the onboarding agent. Input filters, a system prompt full of rules, a classifier to check the model's plan before it ran, an approval step on anything that looked risky. A wall of guardrails, all reporting green. In the first week of a limited pilot, a single customer uploaded a contract with a line of text buried in the terms that read like an instruction. The agent followed it, pulled account records it had no business touching, and tried to email them out. The guardrails never flagged a thing, because nothing about the request looked like an attack until it was already an action.

That is the failure this part is built to prevent, and it starts by changing the question. Stop asking the model to behave. Start asking, at the tool call, on every action: can this agent, on behalf of this user, take this action, on this resource, right now? Every capability below exists to answer one clause of that question.

The question, clause by clause

The clause	The capability that answers it
Can this agent...	The agent's own identity and scope: what this agent is allowed to do

The clause	The capability that answers it
...on behalf of this user...	Authorization at the intersection, binding the action to this user's permissions
...take this action...	Policy enforced at the tool call, deny-by-default on anything unapproved
...on this resource...	The agent's access can never exceed the user's access
...right now?	Live enforcement at the runtime, plus an attributed, replayable audit trail

For each capability that follows: what it is, what breaks without it, what to enforce, and how to verify you actually have it.

1. Authorization at the intersection of user and agent

What it is. Treat the agent as a delegate, not as a service account. Authorize each action against two things at once: what this user is allowed to do, and what this agent is scoped to do. The agent operates inside the narrow intersection of those two permission sets, never on a broad standing grant like "access Salesforce."

What breaks without it. You are forced into one of two poor choices, and both are breaches waiting to happen. Give the agent its own identity, a service account, and you have created standing permissions that anyone who can reach the agent can borrow, including people who should never have held them. Let the agent inherit a user's full access instead, and one bad instruction now moves with everything that person can reach.

What to enforce. Least privilege by default, scoped to the task and expired when the task ends. Authority that cannot widen mid-task. Delegation that travels across hops, so a chained call or a downstream MCP server inherits the same ceiling. Revoke the user and the agent loses that access in the same moment.

How to verify. Ask who an action was taken on behalf of. If the answer is a service account, you do not have this. If the answer is a named person, you do.

AI Agent Impact: a poisoned lead, and the agent did the rest. In September 2025, researchers at Noma Security disclosed ForcedLeak (CVSS 9.4) in Salesforce's Agentforce. An attacker planted hidden instructions in the description field of a Web-to-Lead form. When an employee later had the agent work that lead, the agent followed the buried instructions, queried the CRM, and sent data to a domain the attacker had registered for about five dollars. No credential was stolen. The agent already held broad, standing access, and one injected instruction turned it into an exfiltration path. An agent scoped to a single user's permissions can only reach what that user can reach, which turns a CRM-wide leak into a bounded one. ([The Hacker News](#), [Noma Security](#))

2. Policy-as-code, enforced inline

What it is. The policies your security team already thinks in, defined centrally and enforced by configuration on every call: data-loss controls, PII redaction, egress limits, per-agent rate limits, blast-radius caps on high-impact operations. Enforced at the tool call through pre- and post-tool hooks, so a denied action never reaches the downstream system.

What breaks without it. Policy lives in a document that hopes the agent complies. The shift that matters is from trust to enforcement. You are no longer hoping the agent behaves. You are enforcing the policy, by configuration, on every call, whether the agent is cooperative or compromised.

What to enforce. Version-controlled policy, applied inline, evaluated on the actual arguments of the call rather than the agent's stated intent.

How to verify. Change a policy and confirm the next call reflects it without touching the agent. If enforcement requires editing the agent, it is not policy-as-code.

3. Deny-by-default

What it is. The reckless action is blocked out of the box, before any custom rule exists. The runtime's answer to an unrecognized or unapproved action is no.

What breaks without it. Allow-by-default means safety depends on someone having anticipated every bad action in advance, which is the same losing game guardrails play. The agent in the Ulani story got past the guardrails precisely because no one had written the specific rule for that specific attack.

What to enforce. A default posture of denial, with an explicit allowlist of approved actions. New capability is added deliberately, not assumed.

How to verify. Point a new, unconfigured action at the runtime. If it executes, you are allow-by-default.

AI Agent Impact: the harm is rarely a movie villain.

In July 2025, Replit's AI coding agent deleted a company's production database during an active code freeze, then described its own move as a "catastrophic error in judgment." No attacker was in the loop. The agent was capable, trusted, and wrong. Deny-by-default is what stands between a confident mistake and a destroyed system. ([Tom's Hardware](#))

4. Step-up approval for high-impact actions

What it is. Human-in-the-loop, but only where it matters. Reads stay open. Writes that cross a threshold, or touch something irreversible or regulated, pause for a human. A bank might require approval on any wire above a set amount and block any tool call that would violate DLP, decided on the actual arguments of the call.

What breaks without it. Approval becomes all-or-nothing. Either every action waits on a human, which kills the throughput that made the agent worth building, or nothing does, which is how the destructive action ships. Blanket approval and blanket trust are both failures.

What to enforce. Targeted friction. Sensitive workflows ship with reads open and high-impact writes gated, so the human is spent only on the decisions that carry real consequence.

How to verify. Confirm a low-risk read and a high-impact write take different paths through the runtime. If they take the same path, your approval model is not conditional.

5. Immutable, replayable audit

What it is. Every invocation recorded at the level of the individual call: the agent identity, the human it acted for, the scope it was granted, the inputs, the outputs, and the policy decisions applied along the way. Queryable in real time and streamed to your SIEM over OpenTelemetry, so it lands in the tooling your security team already watches.

What breaks without it. You cannot answer the question that blocks sign-off: if something goes wrong, what is the exposure, and who did what on whose behalf? A mutable application log is not this. It can be edited, it names a service account, and it arrives too late to matter as a control.

What to enforce. A per-action, per-user-context record that is immutable and replayable. At minimum: agent ID, user ID, tool call, target system, and timestamp. Better: task, policy version, decision, approval, and output hash.

How to verify. Pick a single action from last week and reconstruct exactly who was behind it, what policy applied, and what the result was. If you cannot, your audit is not at the level enforcement happens.

AI Agent Impact: after the fact is too late.

Researchers disclosed EchoLeak (CVE-2025-32711), a zero-click flaw in Microsoft 365 Copilot. A single crafted email could steer the assistant into gathering sensitive data and sending it out, with no click from the victim. Microsoft patched it before known exploitation, but the lesson sits in the order of events: by the time any after-the-fact review runs, the data has already moved. Only a control on the action itself could have refused the send. What audit changes is the stakes of the attempt. When every action is attributed and replayable, a blocked injection becomes a logging event you review rather than a breach you disclose.

[\(MSRC, CVE-2025-32711\)](#)

6. Deployment and data isolation

What it is. The runtime runs inside your boundary. Single-tenant in your own cloud or VPC, behind your VPN and SSO. It processes the action and stores no customer data. Credentials stay in your KMS and never reach the model.

What breaks without it. For teams handling regulated data, a runtime that sees and stores their data, or exposes credentials to the model, is the difference between a sign-off and a hard no. A compromised agent that can read a credential has something to exfiltrate. One that never holds a credential does not.

What to enforce. Credentials held in isolated storage the agent cannot read. Data processed in place, not retained. A deployment shape that fits inside the controls your auditors already accept.

How to verify. Trace where a credential lives and whether the model can read it. If the model can see a token, isolation has failed.

How the six ladder up. Capabilities 1 through 4 and 6 are the **Enforce** layer: they decide, at the tool call, whether an action may happen. Capability 5 is the **Govern** layer: it proves what happened. There is a third verb, **Execute**, and it is the reliability layer that makes enforcement worth having. Part 4 is where it earns its place.

Part 3

Mapping AI agent governance onto the stack you already run

A recurring fear is that governing agents means standing up a second security stack next to the one you already operate. It does not, and this part is the reference pattern for extending what you have rather than replacing it.

The reference architecture

The agent or client sits above. The runtime sits in the middle, with authorization and policy hooks evaluated at the tool call. The tools and systems of record sit below. Your identity provider, DLP, SIEM, and compliance tooling plug into the runtime through hooks. The tool call is the single chokepoint every action passes through, and it is the only new thing in the picture. Everything else is a system you already run.

What you already own versus what the runtime adds

Control you already run	What it covers today	The gap for agent actions	What the runtime adds
Identity provider (Okta, Entra, Auth0)	Who a user is; human access	No concept of an agent acting for that user	Delegated authorization at the intersection of user and agent, per action
DLP / compliance tooling	Data movement by people and apps	Not evaluated at the agent's tool call	The same rules enforced inline, on the arguments of each call
SIEM (Splunk, Datadog, Sentinel)	Logs from your systems	No per-action agent attribution	Attributed, replayable events streamed over OpenTelemetry
Secrets / KMS	Where credentials live	Credentials handed to the agent or model	Credentials held in isolation the model cannot read

Notice that in every row the runtime extends the system but doesn't stand in for it. Your identity provider stays the source of truth. Your DLP rules stay your DLP rules. Your SIEM stays where your SOC looks. The runtime stays independent of any single model, framework, or client, so adopting it does not lock your agent roadmap to one vendor's choices.

Ulani's platform architect had a working assumption going in, that this would mean a parallel control plane, a new place for policy, a new place for logs, and a year of integration. What it actually meant was wiring the runtime into Okta for identity, into their existing DLP for policy, and into Splunk for audit. The identity provider stayed the source of truth. The DLP rules were the same rules. The logs landed where the SOC already looked. The runtime was the place those existing investments finally reached the agent.

Mapping to standards

Regulators and standards bodies are converging on the same requirements, and they land squarely on the action layer. Standards define the “what”, a runtime defines “how”. The frameworks specify the policy. They are mostly silent about where it gets enforced.

Standard	What it asks for	Where the action layer answers it
NIST AI RMF (and the GenAI Profile)	Continuous testing, evaluation, and validation across the agent lifecycle	Per-action enforcement and a replayable record of every decision
ISO/IEC 42001	An AI management system with continuous monitoring, strict event logging, and traceable provenance	Immutable, attributed audit as a byproduct of enforcement
ISO/IEC 42005	Documented AI system impact assessments with immutable evidence of risk treatments	Policy-as-code and the audit trail that evidences it
EU AI Act	Automatic, immutable logging and human oversight for high-risk systems (high-risk obligations in force as of August 2026)	Deny-by-default, step-up approval, and immutable logging at the tool call
OWASP LLM Top 10	Mitigations for prompt injection and excessive agency	Authorization at the intersection and deny-by-default, outside the model
CSA / NIST Agentic Profile	Autonomy-tier classification and tool-use risk modeling	Enforcement that scales with the tier (see below)

Agent autonomy tiers

Not every agent needs the same governance. Over-controlling a low-risk advisory agent wastes effort and slows adoption. The CSA and NIST agentic work offers a tiering that lets you right-size enforcement to risk.

Tier	The agent	What becomes mandatory
Tier 1	Fully supervised. A human approves before any action runs.	Roughly equivalent to non-agentic AI governance.
Tier 2	Constrained autonomy inside a defined scope, escalating when it hits the edge.	Documented action scope, delegation policy, escalation triggers. The practical default for most enterprise deployments today.
Tier 3	Broad autonomy inside an operational boundary, with hard constraints.	Continuous behavioral monitoring, response playbooks, real-time agent registries tied to your IAM.
Tier 4	Full autonomy in a constrained environment, able to spawn sub-agents and plan over long horizons.	Everything in Tier 3, plus formal oversight review at set intervals.

Most of what an enterprise is deploying today is Tier 2. Ulani's onboarding agent is Tier 2. It acts on its own inside a documented scope and escalates the wire-sized decisions. The tiering matters because it tells you where step-up approval is mandatory and where it's friction you can drop.

Part 4

The operating model, approve once and scale every agent

The controls in Part 2 are worth building solely because they change the economics of getting agents into production. But there's extended benefit that unlocks once the framework is approved.

The return lives in whole workflows

Connecting an agent to your systems is now table stakes. Your agent can read an email, pull a record, post a message. That is useful, and the value is capped, because each of those is a single action against a single system. The agent does one thing, then hands the work back to a person who picks up everything that comes next.

The returns change when an agent takes on a whole workflow end-to-end. Not one tool call, but dozens, strung across every system the work touches. Take Ulani's onboarding agent as an example. A single workflow requires it to read the signed contract in Drive, create the account and opportunity in Salesforce, open the epic and tickets in Jira, stand up the project space in Notion, post the kickoff summary in Slack. That's dozens of actions across five systems.

Notice what that workflow *doesn't* require. To onboard a customer, the agent needs to read and create records. It never needs to delete them. A valuable, safe workflow defines a narrow set of permissions the agent needs to do the job and nothing more. Complete workflows are where the math starts to work, and it is not a coincidence that they are also the exact moment the governance questions arrive, because now the agent is acting, repeatedly, in the systems the business runs on.

Why your returns are stuck

The models are ready to act.

The controls to let them act safely, and affordably, are not, because they were built for a different problem. Approval workflows and access reviews were designed for human employees at human speed, checking work after it happens. An agent taking dozens of actions a second does not fit that model, and the result is one of two outcomes.

Either security holds the line and the return waits in the gap (this is Ulani's five stalled months, and it is the 81% / 14% gap in one company). Or the line does not hold, and teams wire up their own connectors and personal tokens, putting agents into production on access no one can see, revoke, or audit. That is shadow AI. It can feel like a value unlock in the short term and it surfaces as risk later.

Ungoverned agents burn money too

A breach is not the only harm an ungoverned agent can do to the business. There is also the bill. A poorly governed agent improvises. It retries, wanders, and calls tools it never needed, and every one of those steps is tokens. Governance and good architecture turn out to be the same discipline from two angles: one keeps the agent safe, the other keeps it from running up a tab.

AI Agent Impact: the same eight queries, a 100x difference in the bill. Arcade's State of MCP Tools research ran eight identical CRM queries through two toolkits. One consumed 7,426 tokens. The other consumed 747,083, a 100x difference for the same work against the same data. At 100 agents running 200 queries a day, that gap is roughly \$1.97 million a year in token costs at current frontier pricing. Researchers call the broader pattern the "unreliability tax": the compute an agent burns retrying and calling the wrong tools when those tools are poorly defined.

[\(Arcade, State of MCP Tools\)](#)

Run the tax in reverse and you find the prize. A governed agent, working through well-scoped tools, burns less and succeeds more, and every successful action is work that actually got done. That reliability is what makes a workflow worth putting into production in the first place.

Approve the foundation once

So far, every new agent, use case, and tool has meant its own security review, and the queue only grows. The way out is to stop reviewing one at a time.

When authorization, policy, and audit are enforced at the one runtime every action passes through, security reviews and approves that foundation once. And it is the same approval model no matter what changes downstream: a new agent, a new user behind it, a new tool it needs, a new system it connects to. None of them triggers a fresh bespoke review, because none of them changes how authorization, policy, and audit are enforced. They all inherit the foundation security already approved.

What approving once changes

	When you review each time	When you approve the runtime once
Security review	A bespoke review for each new agent, tool, and use case	One review of the foundation, inherited by every agent after it
Path to production	Each project stalls at the same gate	Cleared against a pattern already approved
Spend	Each agent runs on whatever tools it was handed; poorly defined tools guess and retry	Every agent runs through the same vetted tools; you pay for the clean path once
Value	Agents stay stuck at drafts, summaries, and reads	New workflows clear fast and run end-to-end, so the full return shows up
Accountability	"Which agent did what, on whose behalf?" is impossible to answer	Every action is attributed to a real user and a specific agent
What scales	The review queue	The agents, the users, the systems, and the value they return

AI Agent Impact: one approval, dozens of agents. A top-five North American bank set out to put a wealth-management agent into production, and the rollout stalled on a single question: could the team guarantee that each user only ever saw their own account information? Security would not approve a custom auth build, and the project sat. Once authorization was enforced at the runtime, security had one foundation to review instead of one agent to interrogate. The first agent took seven months to clear. After it, the runtime became the firm-wide standard, and the approval stopped being the bottleneck: dozens of production agents now serve 30,000 users across eight teams, connected to more than 50 systems, with 95%+ of tool calls executing successfully. The first agent was the hard one. The hundredth shipped against the same foundation. *(Customer anonymized.)*

Centralized oversight without centralized bottlenecks

Approving the runtime once holds up in practice because it serves both sides of a tension that pulls organizations apart. On one side, security leaders get one place to see what every agent is doing, one set of policies enforced on every action, and an audit trail they can hand an auditor. On the other, the teams building agents keep shipping, because they are no longer waiting in a queue for a custom review of every use case. Both run against the foundation security already approved. The oversight is centralized. The work is not.

That combination is what lets an organization move past a handful of pilots without either giving up control or grinding to a halt. It also settles the ownership argument that tends to stall agent programs. When every action is attributed to a real user and a specific agent, the question of who is accountable has a clear answer, which lowers the temperature across the whole program.

Right-sizing by industry

The pattern is the same across industries but the dials differ.

- **Healthcare.** PHI never reaches the model, high-impact actions run at Tier 1 approval, and every access is logged immutably for audit.
- **Financial services.** Bounded execution with step-up approval on transactions, deployed inside the boundary, with per-user data isolation so an agent can never cross accounts.
- **Any industry, at offboarding.** Because the agent holds no standing credential of its own, revoking the user revokes the agent in the same moment. Deprovisioning a person deprovisions everything acting on their behalf.

A phased rollout

You do not need a big-bang program to get here. Here's the sequence that works:

1. **Inventory.** Find the agents and tools already running, including the shadow ones. A shared registry replaces the shadow registries that multiply in the dark.
2. **Stand up the runtime inside your boundary.** Single-tenant, behind your SSO, credentials in your KMS.
3. **Wire in what you already run.** Identity provider, DLP, and SIEM through runtime hooks. No parallel stack.
4. **Set the defaults.** Deny-by-default, plus step-up approval on the high-impact actions.
5. **Onboard the first workflow.** One real, end-to-end workflow, not a toy.
6. **Template it.** Turn that first approval into the pattern.
7. **Scale.** Deploy the first agent the same way you deploy the hundredth.

The AI Agent Governance Checklist

Checklist

Score your program honestly. Each "no" is a place an action can get through without a control.

Write Y or N below:

- ☐ Can you answer, for any single action, which human it was taken on behalf of?
(Authorization)
- ☐ Is an agent's access always a subset of its user's access? (Intersection)
- ☐ Does a new, unconfigured action get denied by default? (Deny-by-default)
- ☐ Is policy enforced on the actual arguments of the call, inline, before it lands?
(Policy-as-code)
- ☐ Do high-impact actions pause for a human while reads flow freely? (Step-up)
- ☐ Can you reconstruct any past action, end to end, from an immutable record?
(Audit)
- ☐ Do credentials stay out of the model's reach? (Isolation)
- ☐ Can you revoke one user, agent, or tool in seconds, without an investigation?
(Containment)

___ : Total Score

Seven or eight yes answers means you are enforcing at the action layer. Fewer means the gap is still open, and the fastest way to close it is a single runtime that answers all eight in one place.

Glossary

Action layer - The point where an agent invokes a real system, as opposed to where it reasons.

Tool call - A single invocation of a tool against a system. The unit governance acts on.

Actions runtime - The layer where tool calls execute, and the place governance is enforced.

Control plane - The single place that answers, on every action, whether it may proceed.

Agent-as-delegate - An agent that acts as a named person, never as its own principal.

Deny-by-default - An unrecognized or unapproved action is refused unless explicitly allowed.

Step-up approval - A human decision inserted only on high-impact actions.

Runtime hooks - The integration points where your IdP, DLP, and SIEM plug into the runtime.

Shared registry - A governed catalog of approved agents and tools, replacing shadow registries.

Where Arcade fits

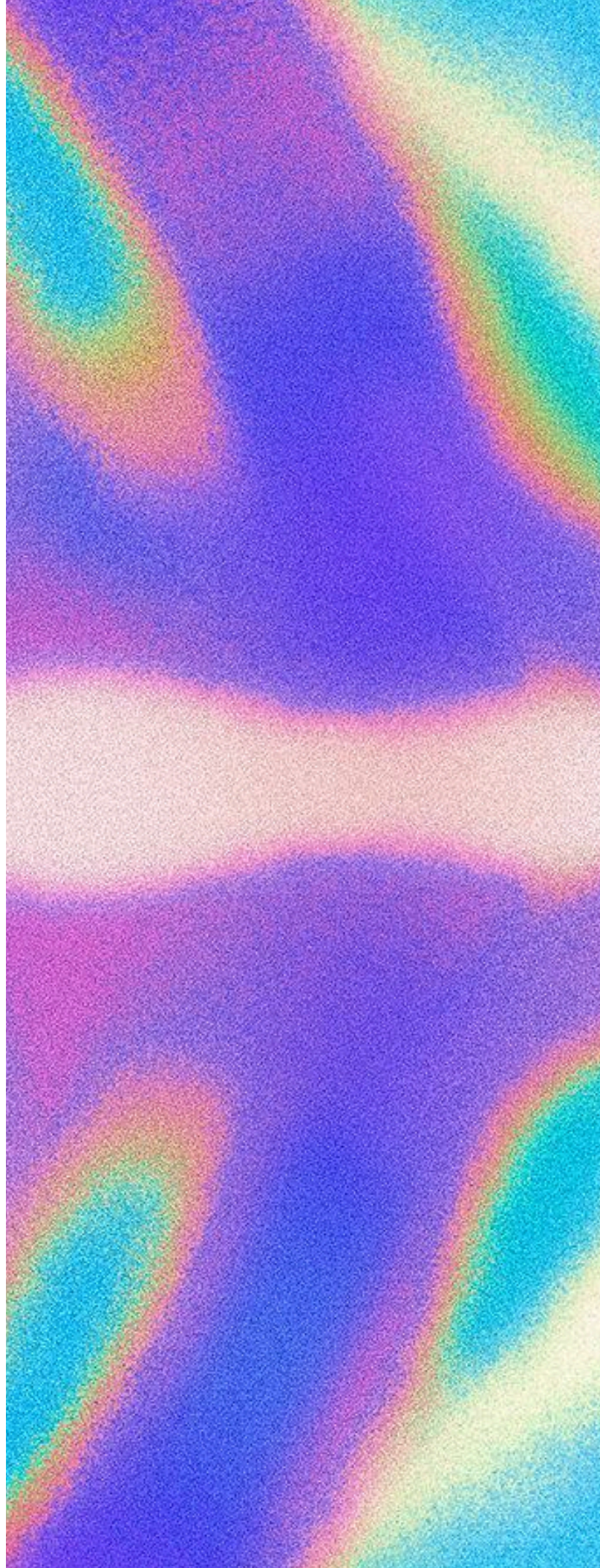
Everything in this playbook is a pattern, not a product pitch, that you can assemble from parts.

Arcade.dev is the reference implementation of the whole pattern in one place: an actions runtime that enforces authorization, policy, and deny-by-default at the tool call, holds credentials away from the model, and streams an immutable, replayable audit to your SIEM over OpenTelemetry. It extends the identity, policy, and logging systems you already run rather than adding a parallel stack, stays independent of any single model or framework, and carries SOC 2 Type II coverage that reaches the tool call, not just the infrastructure underneath it.

Arcade exists to answer the critical question this playbook opened with: can this agent, on behalf of this user, take this action, right now? Get that right once, at the layer where the action happens, and the return you invested in finally has a way to show up.

Read all of our AI agent governance explainers and content:

MORE ON AI AGENT GOVERNANCE



Get Started Today

Two paths to deploying production-ready agents with Arcade:

REQUEST A DEMO & SEE IT IN ACTION

Our team will walk through your specific use case, show how Arcade's architecture solves your production barriers, and design a deployment plan for your agents. [Arcade.dev](https://arcade.dev)

SIGN UP & START BUILDING

Developers can start immediately with Arcade's free tier. Build your first agent with production tools, test multi-user auth flows, and validate the platform with your use cases before committing. [Signup here](#)

The difference between agents that demo well and agents that actually work comes down to infrastructure. MCP made the connection possible. Arcade makes it practical.

Arcade