aklivity

ENTERPRISE AI ARCHITECTURE

The State of AI Gateways in 2026

From Model Routing to the Governance Layer for Autonomous AI

LLM Gateway → AI Gateway → Unified AI Gateway → Transaction Governance → AI Governance Runtime

CONTENTS

Executive Summary

The AI gateway is changing, and the definition that served the market a few years ago no longer captures the problem enterprises are trying to solve.

Early AI gateways were built for a relatively contained task. As organizations adopted models from multiple providers, they needed a shared layer for authentication, routing, rate limiting, cost management, and observability. In practical terms, the gateway acted as an intelligent proxy between an application and one or more models.

Agentic applications have widened that boundary. Agents do more than send prompts and return responses: they discover tools, call APIs, query enterprise data, delegate work to other agents, make decisions, and increasingly take actions with real-world consequences.

Adoption is still uneven, but the direction is clear. McKinsey's 2025 global survey found that 62% of respondents said their organizations were at least experimenting with AI agents, while 23% reported scaling an agentic AI system somewhere in the enterprise. At the level of an individual business function, however, no more than 10% reported scaling agents. Enterprise adoption is advancing, but operational maturity is lagging.

Governance is even less mature. Deloitte's 2026 research found that only 21% of surveyed organizations had a mature governance model for agentic AI. Gartner predicts that by 2027, 40% of enterprises will demote or decommission autonomous agents after governance gaps surface in production incidents.

62%	21%	40%
of organizations at least experimenting with AI agents McKinsey, 2025	report a mature governance model for agentic AI Deloitte, 2026	of enterprises predicted to demote or decommission agents by 2027 Gartner

Infrastructure vendors are responding. Microsoft's AI Gateway tier describes a governed runtime boundary for both models and Model Context Protocol (MCP) tools. Google has extended Apigee so existing APIs can be exposed as MCP tools to agentic applications. Kong's AI Gateway spans model routing and MCP traffic. Envoy AI Gateway reached version 1.0 in June 2026 with support for multiple LLM providers and an MCP gateway, while agentgateway is pursuing a unified data plane for HTTP, gRPC, LLM, MCP, and Agent2Agent traffic.

Together, these developments point to a broader shift: the first generation of AI gateways governed access to models; the emerging generation is beginning to govern access to everything an agent can touch, including models, tools, APIs, other agents, events, and enterprise data.

For enterprise architects, that shift raises three related questions. The first is one of scope: should the gateway remain a specialized model layer, extend an existing API-management platform, become an MCP or agent gateway, or provide a unified enforcement plane across all of these interactions? The second is an acquisition question, because building, buying, and adopting open-source infrastructure create very different long-term obligations around protocol evolution, security, extensibility, operations, and support. The third is more fundamental: even if a gateway governs every interaction, is that enough to govern the business activity those interactions collectively perform?

A contract renewal, customer refund, purchase order, incident response, or employee onboarding process may span several agents, models, tools, systems, approvals, and hours or days. No single HTTP request, MCP invocation, model trace, or agent-to-agent exchange represents the whole activity. The meaningful governance object is increasingly the **business transaction**.

That suggests a longer-term architectural progression:

`LLM Gateway` → `AI Gateway` → `Unified AI Gateway` → `Transaction Governance` → `AI Governance Runtime`

In this model, the gateway remains essential, but it becomes the enforcement foundation beneath a broader runtime responsible for identity, delegated authority, policy, approvals, durable state, reliability, observability, audit, and ultimately business outcomes. The gateway may be the beginning of enterprise AI governance rather than its final form.

SECTION 01

The Name Stayed the Same.The Architecture Did Not.

The earliest AI applications were straightforward from an infrastructure perspective. An application collected a prompt, called a model API, received a response, and returned it to the user. As organizations adopted multiple providers, infrastructure teams encountered familiar distributed-systems problems: credentials needed to be centralized, traffic controlled, failures handled, usage measured, costs allocated, and developers insulated from provider-specific differences.

The AI gateway emerged as the natural abstraction point:

```
Application  →  AI Gateway  →  Model
```

Instead of integrating separately with every provider, applications could call a single endpoint. The gateway handled routing, credential injection, usage controls, telemetry, and fallback behavior.

Those capabilities still matter, but they are becoming standard. LiteLLM, for example, provides an open-source LLM gateway with a common interface across more than 100 LLMs, along with cost tracking, budgets, authentication hooks, rate limiting, and retries or fallbacks. Cloudflare's AI Gateway supports spend limits, caching, dynamic routing, rate limiting, and provider fallback. Kong provides model-aware routing and provider abstraction through its AI Gateway. The original gateway problem is increasingly well understood and widely productized.

What has changed is the application architecture around it. An agent may ask a model what to do next, discover a tool through MCP, invoke an internal API, delegate part of a task to another agent, query a database, respond to an event, wait for a human decision, and then write to a system of record.

The architecture therefore looks less like:

```
Application  →  Model
```

and more like:

```
User  →  Agent  →  Models + Tools + APIs + Agents + Data
```

That difference is fundamental. The model is no longer the only resource that needs governance and, in many cases, it is not the highest-risk resource. The more consequential interaction may be the tool that changes a customer record, the API that creates a payment, the dataset containing regulated information, or the downstream agent that receives delegated authority to continue the task.

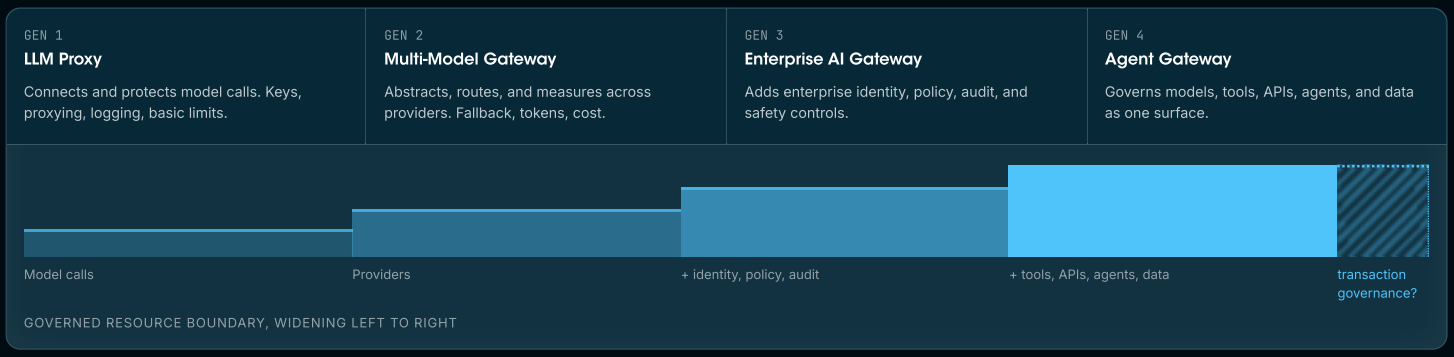
This is why the term “AI gateway” has become difficult to pin down in 2026. Products with very different architectural histories are converging on the same control problem, and enterprises must decide where that control should live.

SECTION 02

Four Generations of the AI Gateway

The category's evolution can be understood in four broad generations, each widening the resource boundary the gateway is expected to govern.

FIGURE 1 **Evolution of the gateway boundary**



The category has evolved by widening the resource boundary the gateway is expected to govern.

Generation One: The LLM Proxy

The first requirement was connectivity. Organizations wanted a central route to model providers without distributing provider credentials throughout application code. The gateway handled API-key management, proxying, basic request logging, simple rate limiting, and a common endpoint for developers.

The architectural boundary was narrow because the interaction itself was narrow: the application called a model, and the gateway governed that call.

Generation Two: The Multi-Model Gateway

As organizations experimented with more models and providers, abstraction became more important. The gateway took on provider normalization, model routing, retries, fallback, load balancing, token accounting, cost allocation, quotas, semantic caching, and model telemetry.

Products such as LiteLLM show how thoroughly these capabilities have been productized. Its gateway standardizes access to more than 100 LLMs while adding spend management, authentication, rate limits, and fallback behavior. Cloudflare similarly supports dynamic routing based on conditions and quotas, along with model fallback and dollar-denominated spend limits across AI traffic.

This generation answered a practical infrastructure question: how can model access be operated as shared enterprise infrastructure rather than repeated as application-specific integration code?

Generation Three: The Enterprise AI Gateway

Once AI moved beyond experimentation, connectivity began to intersect with identity, security, privacy, compliance, and platform operations. The gateway evolved from a development convenience into an enterprise control point, adding capabilities such as centralized identity and authorization, data-loss prevention, content safety, policy enforcement, audit, self-service access, enterprise networking, shared observability, and organizational budgets or quotas.

At this stage, the gateway no longer existed simply to hide provider differences. It became part of the organization's security and governance architecture.

Generation Four: The Agentic AI Gateway

Agentic AI expands the boundary again because the gateway now encounters traffic to resources other than models. Microsoft's Azure API Management documentation, for example, describes AI gateway capabilities across models, agents, remote MCP servers, and A2A agent APIs. Its dedicated AI Gateway tier, currently in public preview, is designed to publish, secure, govern, and observe access to both models and MCP tools.

Google is following a related path through Apigee. MCP became generally available in Apigee in March 2026, allowing existing APIs to be exposed as tools for agentic applications while retaining familiar API-management controls. Envoy AI Gateway 1.0 likewise combines access to multiple LLM providers with an MCP gateway that supports tool routing, filtering, authorization, and observability.

The center of gravity is moving from **governing the model request** to **governing the agent interaction**. That distinction may define the next phase of the market.

SECTION 03

What Does "AI Gateway" Mean in 2026?

There is no single answer yet. A more useful way to understand the market is to look at where different products began and which control surface they are expanding toward.

FIGURE 2 **The 2026 AI gateway taxonomy**

LLM Gateways	API Gateways Extended for AI	Cloud & Edge AI Gateways	MCP & Agent Gateways	Unified AI Gateways / Data Planes
BEGAN AT Model access	BEGAN AT API management	BEGAN AT Traffic & edge networking	BEGAN AT Agent connectivity	BEGAN AT Convergence of the other four
UNIT OF CONTROL `model request`	UNIT OF CONTROL `API call`	UNIT OF CONTROL `inference request`	UNIT OF CONTROL `tool invocation`	UNIT OF CONTROL `agent interaction`
INHERITED STRENGTH Provider abstraction, routing, spend and token control	INHERITED STRENGTH Identity, policy, lifecycle, enterprise networking	INHERITED STRENGTH Latency, availability, caching, resilience, cost control	INHERITED STRENGTH Tool federation, end-user identity, credential mediation	INHERITED STRENGTH One enforcement model across protocols and resources
OPEN QUESTION Can a model-first design govern actions?	OPEN QUESTION Can request abstractions express agent semantics?	OPEN QUESTION Will the same surface govern what agents do?	OPEN QUESTION Is protocol scope the same as governance scope?	OPEN QUESTION How much infrastructure must one plane own?
Governed AI Interaction · the enterprise enforcement boundary				

Products called "AI gateways" are converging on a common control problem from very different architectural starting points.

LLM Gateways

LLM gateways start with model access. Their center of gravity is provider abstraction, routing, resilience, token and spend management, rate limits, caching, guardrails, and observability. Their natural unit of control is the **model request**.

That focus remains valuable, especially for organizations managing a heterogeneous set of providers or offering internal development teams a standard AI endpoint. The limitation is architectural: these products begin from the assumption that the model is the primary resource being governed.

API Gateways Extended for AI

API-management vendors begin from a different foundation. Their historical strengths include identity, authentication, authorization, traffic policy, lifecycle management, enterprise networking, developer portals, rate limiting, and observability. From that perspective, AI is another class of traffic that must be secured and operated.

Microsoft explicitly positions AI gateway functionality as an extension of Azure API Management across LLM endpoints, MCP servers, and A2A agent APIs. Google describes Apigee as providing AI-gateway functions while turning existing services into discoverable tools for agents. Kong combines conventional API-gateway controls with provider abstraction, model-aware routing, guardrails, governance, and MCP integration.

The important question is no longer whether API gateways can carry AI traffic; they clearly can. The question is whether API-management abstractions can evolve far enough to represent agent-native concepts such as tool discovery, delegated authority, long-running activity, dynamic action selection, and agent-to-agent interaction.

Cloud and Edge AI Gateways

Cloud and edge platforms approach the problem through traffic management, distributed networking, and developer infrastructure. Their strengths naturally include routing, latency management, global availability, caching, resilience, telemetry, and cost control.

Cloudflare's AI Gateway, for example, combines caching, rate limiting, spend controls, dynamic model routing, and fallback behavior, making it a strong operational control point for inference traffic. The open question is whether the same enforcement surface will eventually govern the broader set of actions an agent can take.

MCP and Agent Gateways

A newer class begins with agent connectivity rather than model connectivity. Arcade's MCP Gateways, for example, federate tools from multiple MCP servers into governed collections and connect tool access to end-user identity and authorization. Its architecture can acquire and inject downstream OAuth credentials without exposing those tokens to the model or client.

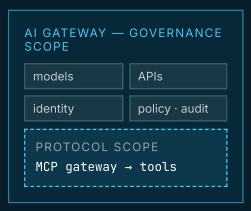
Agentgateway goes further by designing a unified data plane for HTTP, gRPC, LLM, MCP, and A2A traffic. Its premise is a meaningful reversal of the earlier model: **the agent is the new client**.

SIDEBAR 1

Is an MCP gateway an AI gateway?

An MCP gateway can be one form of AI gateway, but the terms are not interchangeable. MCP focuses on tool discovery and invocation. An enterprise AI gateway may also govern models, conventional APIs, agents, data, credentials, policy, and audit.

The useful question is not which protocol the product carries, but how much of the agent's interaction surface it can govern consistently.



Part versus whole: protocol scope is not the same as governance scope.

Unified AI Gateways and AI Data Planes

A fifth category is beginning to emerge from the convergence of the previous four. Its premise is that models, tools, APIs, agents, and data all participate in AI execution and should therefore share a common enforcement model.

Redpanda's Agentic Data Plane is one example. Its documentation describes an environment that combines an AI gateway, MCP servers, declarative agents, governance, access controls, and data infrastructure, with governance intended to sit above the choice of model, cloud, or agent framework. Agentgateway makes a similar case for one operational surface across conventional service traffic and agentic protocols.

These products do not solve the problem in the same way, but they point in the same direction. The AI gateway is becoming less about a specific protocol and more about **where the enterprise chooses to enforce control over AI-driven interactions**.

SECTION 04

The 2026 Capability Baseline

As the market matures, it is useful to distinguish capabilities that are becoming standard from those where differentiation is shifting.

FIGURE 3 **Capability maturity map**

	BECOMING BASELINE	ADVANCED TODAY	EMERGING DIFFERENTIATION
Model plane	Provider abstraction · routing · retries · fallback · rate limits · token accounting · telemetry	Semantic routing · task-aware model selection · prompt transformation · model guardrails	Policy-driven provider selection tied to data class and transaction budget
Tool & agent plane	MCP proxying · session and streaming handling · basic tool exposure	MCP federation · tool discovery and filtering · tool-level authorization · OAuth mediation	Policy over tool arguments and side effects · agent-to-agent governance
Enterprise controls	Authentication · quotas · cost allocation · audit logging	Enterprise identity · data-loss controls · content safety · compliance records	Delegated authority · authority narrowing across agents · approval as runtime state
Data & integration	API connectivity · request and response logging	Database and SaaS access · event streams · field-level masking of tool results	Transaction context carried across systems · observability tied to outcomes

Model-plane functions are becoming table stakes; differentiation is shifting toward tools, authority, data, and outcomes.

Model-Plane Capabilities

Multi-provider access, standardized APIs, routing, load balancing, retries, fallback, token accounting, budgets, rate limiting, caching, and request-and-response telemetry are increasingly expected. More advanced implementations add semantic routing, task-aware model selection, prompt transformation, content filtering, policy-driven provider selection, and model-specific guardrails.

These functions remain important, but they are unlikely to define the category on their own as comparable features appear across both commercial and open-source products.

Tool and Agent Capabilities

MCP is accelerating the move beyond model traffic. Gateways are beginning to proxy and federate MCP servers, support tool discovery and filtering, enforce tool-level authorization, mediate OAuth credentials, manage sessions and streaming, and connect agents to one another.

Envoy AI Gateway added fine-grained MCP authorization and tool filtering before reaching version 1.0. Microsoft's AI Gateway tier can federate remote MCP servers, convert selected OpenAPI operations into MCP tools, and expose SaaS connectors through a governed endpoint.

The gateway is no longer deciding only which model receives a prompt. It may also determine which tools an agent can discover, which credentials it can use, and which actions it may execute.

Enterprise-Control Capabilities

As AI moves deeper into business processes, the gateway begins to resemble security and governance infrastructure rather than application plumbing. It must account for human, workload, and agent identities; manage credentials and authorization; enforce policy and data-loss controls; maintain audit and compliance records; and integrate with catalogs, configuration systems, cost allocation, and organizational quotas.

The challenge is not simply to add each control independently, but to apply them consistently across every resource the agent can reach.

Data and Integration Capabilities

Agents need more than model access. They require the operational context held in APIs, databases, enterprise SaaS systems, event streams, and real-time data. Forrester has described integration as fundamental to agentic AI, noting that an agent without integration is little more than an answer engine. McKinsey's research reaches a related conclusion from the business side: AI high performers are nearly three times as likely as other organizations to report fundamentally redesigning workflows instead of merely layering AI onto existing processes.

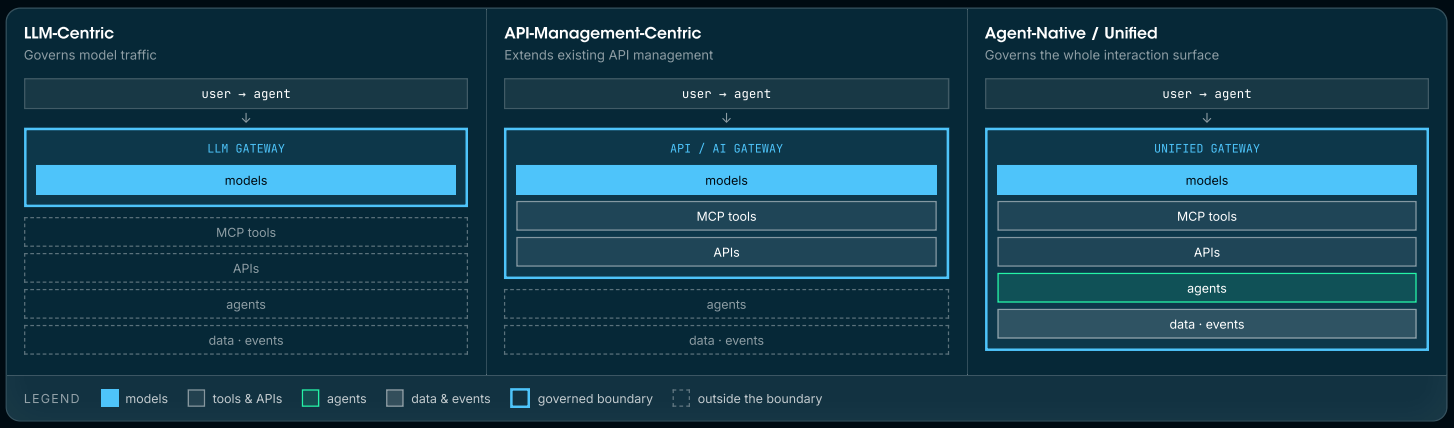
The implication is straightforward. As AI moves from answering questions to participating in workflows, the gateway has to understand more than models.

SECTION 05

Three Architectural Camps Are Emerging

The vendor landscape can be reduced to three broad architectural philosophies.

FIGURE 4 **Where each camp draws the governance boundary**



The three camps differ less in feature count than in where they draw the governance boundary.

Camp One: LLM-Centric

The first keeps the gateway centered on model traffic:

```
Agent or Application  →  LLM Gateway  →  Models
```

APIs and tools remain on separate infrastructure. This approach is simple and often appropriate when workloads are still primarily model-oriented. Its strength is specialization; its limitation is scope. The gateway can govern the agent's reasoning infrastructure while seeing little of what happens once that reasoning results in an action.

Camp Two: API-Management-Centric

The second extends existing API-management infrastructure:

```
Agent or Application  →  API/AI Gateway  →  Models + APIs + MCP
```

This approach benefits from years of investment in identity, policy, networking, lifecycle management, and operations. Microsoft and Google are both showing how conventional API-management platforms can expand in this direction by adding model and MCP capabilities. For many enterprises, this will be the most operationally natural path.

The unresolved issue is whether increasingly agent-native semantics can remain comfortably expressed through infrastructure designed around discrete API requests.

Camp Three: Agent-Native or Unified

The third camp starts with a wider boundary:

```
Agent or Application  →  Unified Gateway  →  Models + MCP + APIs + Agents + Data
```

Agentgateway's 2026 architecture, for example, brings HTTP, gRPC, MCP, A2A, and LLM traffic into the same control and data plane. The attraction is consistency: identity, policy, observability, and traffic management can potentially span the full interaction surface. The tradeoff is operational scope, because the gateway becomes responsible for substantially more infrastructure.

The strategic question is therefore larger than "Which AI gateway should we buy?" Enterprises must decide whether the control layer should evolve outward from an LLM gateway, inward from API management, or emerge as an agent-native layer of its own. The market has not yet settled on an answer.

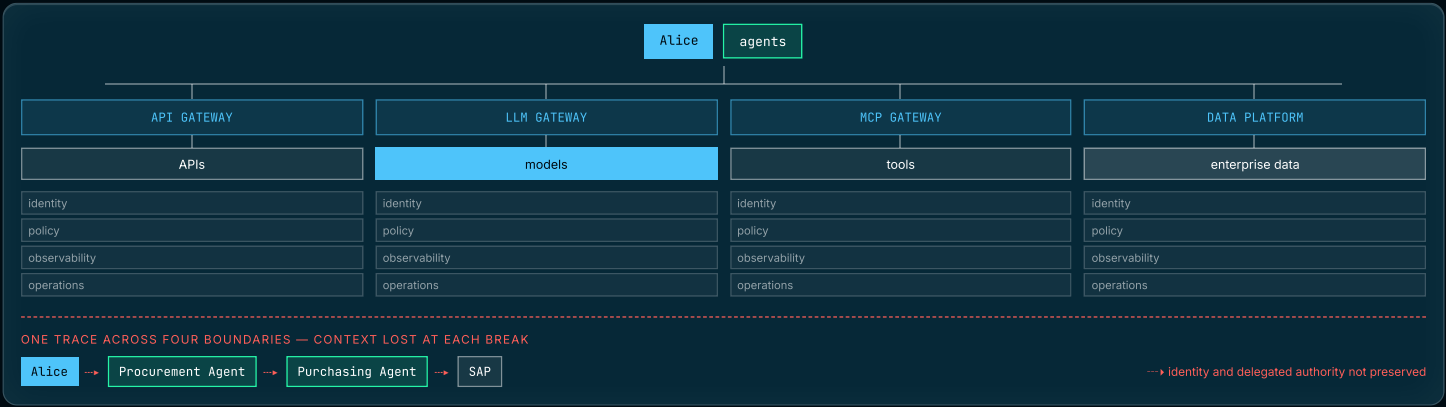
SECTION 06

The Three-Gateway Problem

Rapid adoption may leave many organizations with a fragmented architecture:

```
Applications  →  API Gateway  →  APIs
Agents  →  LLM Gateway  →  Models
Agents  →  MCP Gateway  →  Tools
Agents  →  Data Platform  →  Enterprise Data
```

FIGURE 5 **The Three-Gateway Problem**



Local controls can be strong while the end-to-end governance model remains fragmented.

Each choice can be rational on its own. Together, however, they can create new problems in identity, policy, observability, and operations.

Identity Fragmentation

Consider the chain:

```
Alice  →  Procurement Agent  →  Purchasing Agent  →  SAP
```

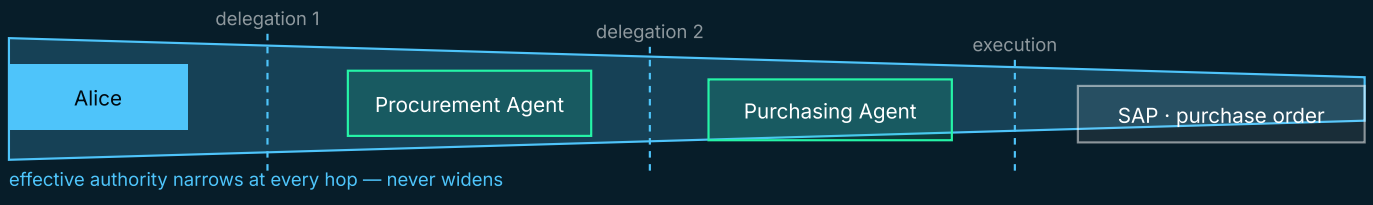
What identity should SAP see: Alice, the procurement agent, the purchasing agent, or a shared service account? If the second agent received only part of Alice's authority, where is that restriction represented and enforced?

Traditional infrastructure often treats authentication as the end of the problem. Agentic systems need a richer concept: **delegated authority**. Identity establishes who is involved; authority defines what that actor may do in this particular context.

SIDEBAR 2

Follow the identity, not just the prompt

When an agent changes a customer record or creates a payment, start with the action and trace authority upstream. Which human or workload initiated the task? Which agent received authority, and how was that authority narrowed at each delegation? A complete governance trace follows identity and effective authority across the entire chain, not merely the prompt and model response.



Policy Fragmentation

In a fragmented architecture, model policy may live in an AI gateway, tool policy in an MCP gateway, identity policy in the identity provider, API policy in an API gateway, data policy in the data platform, and workflow rules in application code. Every layer may be well controlled, yet the enterprise can still lack a coherent answer to a basic question:

What is this agent allowed to do, on behalf of this user, for this purpose, with this data?

That is a policy-composition problem rather than a missing individual control.

Observability Fragmentation

Telemetry fragments in much the same way. One system records the model call, another the tool invocation, another the API write or database query, and another the human approval. Each component may be observable while the overall task remains difficult to reconstruct.

An operator investigating an incident should be able to follow the activity as one trace:

```
User → Agent → Model → Agent → Tool → Data → Decision → Action
```

Without that view, incident response becomes an exercise in manually correlating several unrelated systems.

Operational Fragmentation

Every additional gateway can introduce another deployment model, policy language, upgrade cycle, credential store, dashboard, logging pipeline, certificate lifecycle, security surface, and operational owner. That does not make multiple gateways inherently wrong. Separate enforcement points may be justified by security zones, ownership boundaries, traffic classes, or deployment environments.

The real risk is **unintentional fragmentation**. Enterprises should aim for one coherent governance model even when that model is physically enforced by several runtime components.

SECTION 07

Build vs. Buy vs. Open Source

Once an organization decides it needs an AI gateway, the next question follows quickly: should it build one, buy one, or adopt an open-source platform? There is no universal answer, but the choice should be treated as an architectural commitment rather than a procurement checkbox.

Build

Building is attractive because the first version looks deceptively simple. A team can place a proxy in front of model APIs, add authentication, routing, and request logging, and have something useful in a short period of time.

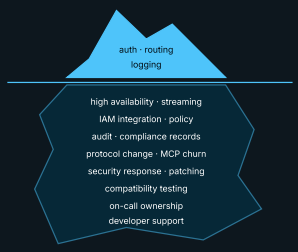
Then the requirements accumulate: a second provider, fallback behavior, budgets, streaming, token accounting, prompt logging, data-loss prevention, MCP support, OAuth, tool filtering, human identity, audit, high availability, dashboards, security patches, and protocol upgrades. At some point, the organization discovers that it has not built a proxy; it has built a product.

SIDEBAR 3

The thin-proxy trap

A thin proxy can be valuable, especially during experimentation. The mistake is assuming that the first useful version represents most of the work.

In production, the hidden surface area usually becomes larger than the proxy itself: protocol evolution, resilience, identity, policy, patching, observability, support, and operations. The prototype proves the need. It does not prove that the enterprise wants to own the product.



Forrester's 2025 predictions warned that “three out of four firms” attempting ambitious agentic architectures themselves would fail. That forecast should not be treated as measured failure-rate data, but it captures a genuine concern: complexity rises sharply when infrastructure moves from experimentation into enterprise operation.

Building can still be rational when the infrastructure creates strategic differentiation, latency or deployment requirements are unusual, custom protocols matter, internal policy requirements are highly specialized, the organization has a strong platform-engineering function, or commercial abstractions would impose unacceptable constraints. The underlying question is whether the enterprise genuinely wants to become its own AI-gateway vendor. If it does, building may be appropriate. If it does not, the apparent simplicity of the first implementation can be misleading.

Open Source

Open source offers transparency, extensibility, self-hosting, rapid access to emerging protocols, community development, and less dependence on proprietary interfaces. Projects such as LiteLLM, Envoy AI Gateway, and agentgateway occupy different points in that design space, from model-centric routing to MCP support and unified agent traffic.

But open source is not simply “buy, except free.” The enterprise still has to decide who will operate and patch the platform, respond to vulnerabilities, own upgrades, validate protocol compatibility, integrate IAM, diagnose production failures, and monitor the health and direction of the project itself.

The real decision is whether the organization is adopting open-source software or taking on responsibility for operating a software product. For a sophisticated platform team, that responsibility may be entirely acceptable. For others, commercial support around an open-source core may be the more practical middle ground.

SIDEBAR 4

Why open source is not the same as build

Open source can eliminate the need to create a gateway from scratch, but it does not eliminate ownership. The enterprise inherits a product it did not have to invent, then decides how much of its operation, integration, extension, and risk it will carry. Open source is therefore a separate acquisition model, not a softer synonym for build.

CREATE THE PRODUCT	OPERATE & EXTEND IT		CONSUME A SUPPORTED PRODUCT
Build invent, run, and support everything	**Open source** inherit the product, own operations	**Supported open source** shared operations and security response	**Commercial software** transfer most operational risk

Buy

Commercial products reduce a different set of risks. They can accelerate deployment and provide enterprise support, tested upgrades, operational tooling, identity integrations, compliance features, security response, and high-availability options.

The tradeoff is dependency. An organization is not only selecting a vendor; it is selecting that vendor's definition of the AI-gateway problem. In a category that is changing rapidly, this matters. A platform optimized entirely around LLM proxying may be excellent today and too narrow tomorrow, while a product built primarily around MCP may face the reverse problem.

The buying decision should therefore look beyond who offers the strongest implementation of today's gateway and consider whether the underlying architecture can evolve with the next one.

FIGURE 6 **Build vs. Buy vs. Open Source**

DECISION DIMENSION	BUILD	OPEN SOURCE	COMMERCIAL
Time to first value	Fast prototype, slow production	Fast start, integration effort remains	Fastest to a supported deployment
Architectural control	Complete	High, bounded by project direction	Bounded by the vendor's model
Customization	Unlimited, and entirely yours to maintain	Forks and extensions carry upgrade cost	Through the supported plugin model
Operational burden	Highest; platform team required	Deployment, upgrades, and tuning are yours	Lowest, subject to deployment model
Security response	Owned end to end	Community advisories, your patch cycle	Vendor obligation with SLAs
New protocols	As fast as your team can implement	Often earliest support in the ecosystem	Follows the vendor's roadmap
Support model	Internal on-call	Community, or a commercial backer	Contracted, with escalation paths
Switching risk	Internal lock-in to your own design	Lower, if extensions stay shallow	Commercial and contractual exit cost
Cost predictability	Understated early, grows with scope	No license, real engineering cost	Contracted, sensitive to consumption

Supported open source and self-hosted commercial data planes occupy the middle of every row above; few enterprises sit at one extreme.

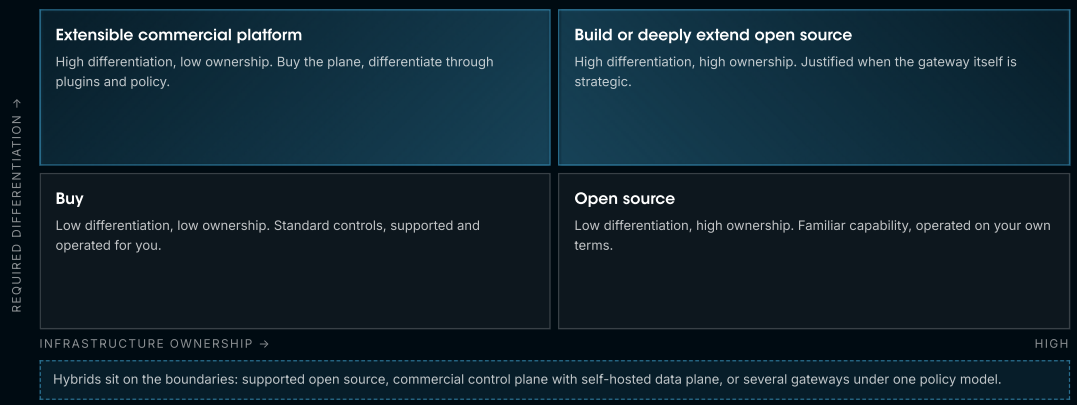
Acquisition strategy determines which responsibilities the enterprise keeps, shares, or transfers.

The Hybrid Reality

In practice, many enterprise deployments will combine these approaches. An organization may use an open-source core with commercial support, a commercial control plane with a self-hosted data plane, a commercial gateway extended through custom plugins, or an internal platform built around an open-source project. It may also retain several specialized gateways while unifying them through common identity and policy.

A useful framework considers two dimensions: how much differentiation the enterprise requires and how much infrastructure it wants to own. Low differentiation and low ownership generally favor buying. Low differentiation with a preference for operational control favors open source. High differentiation with limited ownership favors an extensible commercial platform, while high differentiation and a willingness to own the stack may justify building or deeply extending open source.

FIGURE 7 **Differentiation against ownership**



The right acquisition model depends on both how different the platform must be and how much infrastructure the enterprise wants to own.

The important point is that acquisition strategy should follow architecture, not substitute for it.

SECTION 08

The Economics Are Bigger Than License Cost

Build-versus-buy discussions often collapse into a comparison between license fees and engineering salaries. That is too narrow.

For a custom gateway, total cost can include initial development, integrations, provider maintenance, MCP evolution, protocol testing, security reviews, vulnerability response, observability, infrastructure, on-call engineering, upgrades, and developer documentation. An open-source deployment avoids some license expense but still requires platform engineering, deployment, configuration, patching, compatibility testing, version management, security ownership, and operational expertise. Commercial products bring subscriptions or consumption pricing, support, professional services, integration work, premium features, and potential switching costs.

These economics matter because organizations are becoming more demanding about AI returns. Gartner predicts that more than 40% of agentic-AI projects will be canceled by the end of 2027 because of escalating costs, unclear business value, or inadequate risk controls. Forrester's 2026 predictions similarly reported that only 15% of surveyed AI decision-makers had seen an EBITDA lift in the previous 12 months and forecast that enterprises would defer 25% of planned AI spending into 2027.

Those findings do not argue against AI infrastructure. They argue for measuring it honestly. The total cost of an AI gateway is better understood as:

```
software + engineering + operations + governance + organizational complexity + switching cost
```

SECTION 09

How to Evaluate an AI Gateway in 2026

Fast-moving markets invite feature-counting, but a long list of checkboxes can obscure the architectural decisions that matter most. The better evaluation starts with the platform's boundaries and assumptions.

FIGURE 8 **Architecture evaluation scorecard**

DIMENSION	ARCHITECTURAL QUESTION	EVIDENCE	SCORE	RED FLAG
Protocol architecture	Which protocols are first-class, and can a new one be absorbed without replacing the data plane?		/ 5	◆ Protocol support achieved only through brittle translation
Identity	Are human, application, workload, agent, and downstream identities distinguished across delegations?		/ 5	◆ No preservation of original human authority
Authorization	Can policy reach tools, tool arguments, datasets, destinations, and side effects — not just endpoints?		/ 5	◆ Policy limited to endpoint access
Governance	Is this one governance model, or several unrelated subsystems under one brand?		/ 5	◆ Separate policy languages and audit stores per surface
Observability	Can one trace follow the activity after the model responds?		/ 5	◆ Telemetry stops at the model call
Extensibility	How are plugins, policy hooks, transformations, and custom providers added and upgraded?		/ 5	◆ Extensions require replacing the data plane

◆ **Gating concern.** Any red flag marked here should override a high aggregate score rather than average out against it.

Evaluate the assumptions beneath the feature list, not only the number of features present.

Protocol Architecture

Begin by asking which protocols are genuinely first-class. Can model, MCP, A2A, HTTP, gRPC, and streaming traffic coexist? Is MCP implemented natively or through translation? Can existing APIs be exposed safely as agent tools? Can the data plane absorb a new protocol without being replaced?

The number of providers supported today is useful, but the ability to accommodate a protocol that becomes important tomorrow may prove more valuable.

Identity

A gateway should distinguish among human, application, workload, agent, and downstream identities. The harder test is whether original human authority remains attributable through several autonomous delegations.

Authentication answers who presented a credential. Enterprise governance also needs to establish who ultimately caused the action and under what delegated authority it was performed.

Authorization

Policy should reach beyond models to individual tools, tool arguments, APIs, datasets, agents, destinations, and side effects. A system that can decide whether Alice may access the finance API is useful. A system that can determine whether an agent acting for Alice may create a purchase order below $25,000—but may neither approve it nor release payment—is much more powerful.

Governance

Look for consistency across policy, configuration, audit, catalogs, developer access, and security operations. The central question is whether the platform creates one governance model or merely places several unrelated subsystems under the same brand.

Observability

The gateway should be able to follow activity across boundaries. “Which model was called?” is necessary but incomplete. The more revealing question is, “What did the agent do after receiving the model response?”

The eventual trace may need to span:

```
User → Agent → Model → Agent → Tool → Data → Action
```

Extensibility

The category is changing too quickly for a rigid architecture. Evaluate the quality of its plugin model, policy and protocol extensions, authentication hooks, transformations, custom-provider support, and observability integrations. In 2026, adaptability deserves to be treated as a first-class feature rather than a secondary implementation detail.

SECTION 10

The Competitive Frontier Is Moving From Model Governance to Action Governance

The original gateway question was, “Which model should handle this request?” The emerging enterprise question is, “What is this agent allowed to do on behalf of this user?”

The first problem is largely about routing, resilience, economics, and model policy. The second adds identity, authority, context, data sensitivity, tool risk, action semantics, approvals, audit, and rollback.

Gartner's May 2026 guidance illustrates this shift. It recommends governance that varies with agent autonomy, from read-only agents to agents that act with approval and, at the highest level, fully autonomous agents. Gartner warns that highly autonomous action can occur at a scale and speed that outpaces human oversight, and it recommends continuous monitoring, enforced guardrails, rapid rollback, circuit breakers, and clear ownership.

The significance is easy to overlook. Governance is no longer concerned only with what an AI system produces; it must increasingly control what that system is permitted to cause. A model can hallucinate a purchase recommendation. An autonomous agent can place the order. Those belong to different risk classes, and the infrastructure needs to recognize the difference.

Action semantics provide one useful way to do so. A read operation, such as searching a CRM, querying a database, or retrieving a document, may be allowed automatically. A write operation, such as modifying a customer record, creating a ticket, or updating a contract, may require stronger validation, audit, and idempotency. An irreversible or high-consequence action—transferring funds, deploying to production, deleting records, or sending legally significant communication—may require elevated authority, explicit approval, or a transaction checkpoint.

READ	WRITE	IRREVERSIBLE
Search a CRM, query a database, retrieve a document.	Modify a record, create a ticket, update a contract.	Transfer funds, deploy to production, delete records.
May be allowed automatically.	Stronger validation, audit, idempotency.	Elevated authority, approval, or a checkpoint.

At this point, AI gatewaying begins to overlap with a much larger concept: **runtime governance for autonomous software**.

SECTION 11

The AI Gateway Is Necessary—but Not Sufficient

Consider a seemingly simple enterprise instruction:

Renew the Acme contract for another year at no more than a 7% discount and send it for signature.

Completing that request could require an agent to retrieve the account and opportunity from Salesforce, ask a model to assess the renewal, delegate pricing analysis to another agent, call a pricing service, run a margin check, update Salesforce, generate a contract, evaluate legal policy, wait for executive approval, send the agreement through DocuSign, wait for the customer, and finally close the opportunity.

The process could involve several models, agents, tools, enterprise systems, and human approvals, and it might last two days. No single HTTP request, MCP invocation, model trace, or A2A exchange represents it. **The entire sequence is one enterprise transaction.**

That exposes a fundamental limitation of request-oriented governance. A gateway may govern every individual step and still fail to understand the activity as a whole. It may know that an agent was authorized to call `Salesforce.updateOpportunity`, yet not know that the call belonged to a contract renewal initiated by Alice, capped at a 7% discount, waiting for VP approval, constrained by a $3 AI budget, and expected to finish by Friday.

That broader context changes the policy decision, the audit record, the meaning of failure, and ultimately the object the enterprise wants to govern. It leads to a different question:

What if the correct unit of AI governance is not the request, but the transaction?

SECTION 12

From Request Governance to Transaction Governance

A transaction-oriented architecture makes the business activity itself a first-class object. Its durable record could capture the transaction ID, initiator, business intent, initiating agent, current state, start time and deadline, delegation chain, models and tools used, data classifications, budget and accumulated cost, policy decisions, approvals, and final outcome.

Unlike an HTTP request, the transaction persists. It can survive model failures, process restarts, expired credentials, human delays, deployment changes, and network interruptions. The unit of execution can remain active even while no compute is running.

That is materially different from request processing. The gateway can intercept and govern an interaction; a transaction layer can preserve the context needed to understand why the interaction exists, how it relates to earlier steps, and what outcome it is meant to produce.

This is where the gateway begins to look less like the final product and more like the **insertion point** for a broader governance runtime.

SECTION 13

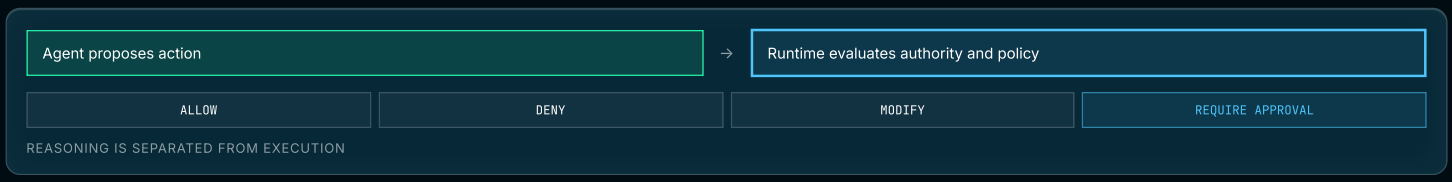
Reasoning Is Not Authority

One of the most important principles of agent governance is also one of the simplest: an agent deciding that something should happen does not mean it has authority to make it happen.

Suppose an agent concludes:

Transfer $100,000.

That conclusion is a proposal, not an authorization. A governance runtime can create a clear boundary between reasoning and execution:



This separation distinguishes intelligence from trust. The model should not be the security boundary. Neither should the agent framework, nor should every individual MCP server have to understand all of the business constraints surrounding a transaction. The trusted runtime becomes the decision point for consequential action.

Forrester's emerging “agent control plane” offers independent evidence that governance is moving outside individual agent-building and orchestration environments. As heterogeneous agents proliferate, Forrester argues that governance must sit outside both planes so it can provide independent oversight and consistent enforcement.

The architectural principle is straightforward: let agents reason wherever they are most effective, but route consequential actions through a trusted external control point.

SECTION 14

Identity Must Become an Authority Chain

Conventional identity systems ask, “Who are you?” Agentic systems must also ask, “On whose authority are you acting, for what purpose, and within what limits?”

Consider the chain:

```
Alice → Procurement Agent → Research Agent → Purchasing Agent → SAP
```

Passing only `user = Alice` is not enough. The purchasing agent should not automatically inherit everything Alice is allowed to do. Instead, the authority context might state that Alice delegated a laptop-replacement task to the procurement agent, which then delegated a narrower purchasing task. The purchasing agent may search suppliers, request quotes, and create a purchase order up to $25,000, but it may not approve the order or release payment, and its authority expires when the transaction ends.

Each delegation should be able to reduce authority, never expand it beyond what the parent possessed. Conceptually:

```
user authority
∩ agent authority
∩ delegated task authority
∩ transaction policy
─────────────────────────
= effective authority
```

This model is substantially safer than authenticating each agent independently and assuming that identity alone is sufficient. The distinction between identity and authority may become one of the defining enterprise architecture problems of agentic AI.

SECTION 15

Policy Must Span Models, Tools, Agents, and Data

Today's governance is often divided by technology domain. Model rules live in an AI gateway, tool access in an MCP gateway, identity rules in IAM, API policies in an API gateway, data controls in a DLP system or data platform, and workflow constraints in application code.

That division is understandable when each system operates independently. It becomes harder to manage when one autonomous transaction crosses all of them.

A future policy context may need to combine **user + agent + data + intent + action + risk + transaction**. That would make it possible to express rules such as: restricted data may be sent only to approved models; non-finance users cannot invoke finance tools; payments above a defined threshold require finance approval; support agents must not receive customer Social Security numbers; external sends require DLP inspection; AI spend is capped at the transaction level; and the number of external writes is limited for a given activity.

The important change is not simply to centralize every rule in one policy engine. It is to make policy aware of the meaning and context of the activity.

A tool call should not be evaluated only because its name is `createPayment`. The runtime should also understand that the action is financial, irreversible, high risk, and valued at $80,000. Policies can then be written against action semantics rather than duplicated for every implementation of a similar tool.

Redpanda's emerging MCP Data Policies illustrate part of this direction. Its July 2026 release added controls that can mask, redact, or filter fields from tool results before the model receives them. The broader opportunity is to apply that kind of contextual policy across every step of an AI-driven transaction.

SECTION 16

Human Approval Becomes Part of the Runtime

Human-in-the-loop control is often implemented today as application-specific logic: the agent reaches a sensitive step, the application sends a notification, someone approves, and the application resumes.

For consequential autonomous systems, approval should become a first-class execution state:

The revalidation step is essential. Approval may take twelve hours, and during that interval the user could lose access, the contract or price could change, credentials could expire, a data classification could be updated, or the transaction budget could be exhausted.

Approval should therefore not mean replaying an old request. It should mean resuming the transaction only after authority has been re-established and the relevant assumptions have been checked again.

Gartner's proportional-governance model points in the same direction. It distinguishes agents that act only with approval from more autonomous systems and emphasizes approval workflows, auditability, continuous controls, and rollback mechanisms as autonomy increases.

Human oversight is not merely a user-interface concern. It becomes part of the system's execution semantics.

SECTION 17

Reliability Moves From Requests to Outcomes

AI gateways already handle request-level reliability. If one model times out, the gateway can fall back to another:

```
Claude → timeout → GPT
```

The request still succeeds. Transaction-level reliability moves the problem one layer higher.

Consider customer onboarding:

WHAT A GATEWAY SEES	WHAT A TRANSACTION RUNTIME SEES
`Create customer` ✓ `Create invoice` ✓ `Charge payment` ✗ Three independent requests, one of which failed.	One business activity, partially complete — and a choice of recovery: retry the payment, use another processor, pause for finance, cancel the invoice, deactivate the customer, or escalate to an operator.

A conventional gateway sees three requests. A transaction runtime sees a business activity that is only partially complete.

That wider view allows the system to choose among several recovery strategies. It might retry the payment, use another processor, pause for finance, cancel the invoice, deactivate the customer, or escalate to an operator. The correct response depends on the state and intent of the overall transaction, not simply on the error code returned by the last request.

In this model, familiar reliability mechanisms expand beyond request retries to include step retries, alternate execution paths, compensation, transaction timeouts, escalation, partial-completion recovery, and end-to-end idempotency.

This is not merely LLM reliability. It is **agent reliability at the level of the business outcome**, which becomes increasingly important as agents begin to mutate enterprise systems rather than only generate information.

SECTION 18

Observability Becomes Accountability

Traditional AI observability answers important operational questions: which model was called, how many tokens were used, how long the request took, what the prompt contained, and how much the call cost.

Agent governance requires all of that, but it also needs to explain the complete activity. If every interaction belongs to a durable transaction, the system can potentially show who initiated it and why; which agents acted and received delegated authority; which models influenced the decision; which tools and data were used; what policy decisions were made; who approved consequential steps; how much the full transaction cost; and what outcome resulted.

The resulting trace is no longer just an engineering telemetry artifact. It becomes a shared record for engineering, security, compliance, finance, operations, and business owners. Observability shifts from asking only “What happened?” to explaining why the transaction happened, who authorized it, what it cost, which data it touched, and how AI influenced the result.

Over time, that record could become an **agent flight recorder**. After an incident, investigators could reconstruct the user, agent version, model, prompt version, tool schema, policy version, arguments, authorization decision, model output, approval state, and executed action.

That does not necessarily mean replaying irreversible side effects. It means recreating enough of the decision environment to determine whether the failure came from the model, prompt, tool definition, policy, user authority, agent logic, or an upstream service.

At that point, observability starts to look less like conventional AI monitoring and more like incident response for autonomous software.

SECTION 19

Events and Real-Time Data Become First-Class Inputs

Agents will not operate only because a person typed a prompt. They will increasingly respond to enterprise events: a failed payment in Kafka, a closed opportunity in Salesforce, a vulnerability reported by GitHub, a production alert from a monitoring system, a month-end schedule, or a direct instruction to investigate a customer account.

Any of these events may start a transaction, resume one that is waiting, or change the context of an activity already in progress. The execution model becomes:

```
Event → Transaction → Model → Agent → Tool → Approval → Wait → Tool → Event
```

rather than the much simpler:

```
Prompt → Model → Response
```

This matters because real-time enterprise context becomes part of AI governance. The system must determine not only whether an agent may consume a piece of data, but also how that data changes an ongoing transaction and which policies should apply as circumstances evolve.

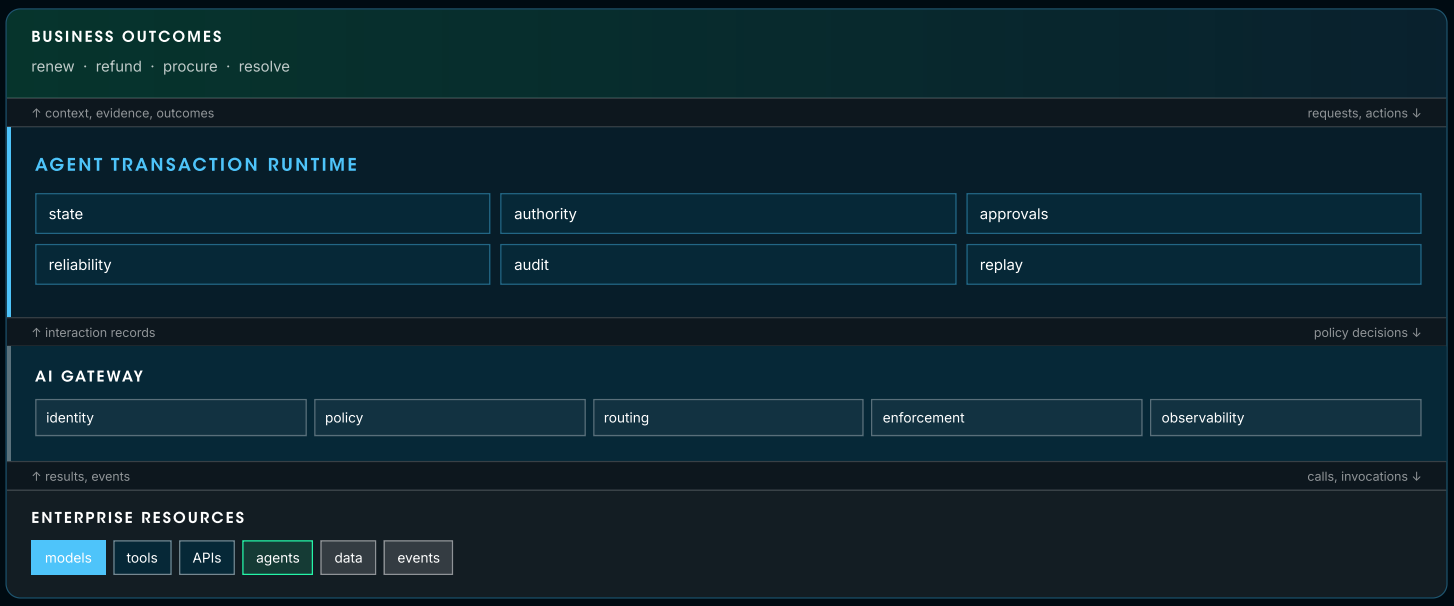
The internal transaction-runtime concept treats events as first-class inputs for precisely this reason. More broadly, AI governance is expanding to cover **models + tools + APIs + agents + identity + events + data**.

SECTION 20

A Broader AI Governance Stack

The architectural progression can be understood as a stack, with each layer answering a different governance question.

FIGURE 9 **The AI governance stack**



The AI gateway governs access. The transaction runtime governs outcomes.

The gateway remains the enforcement foundation; durable context is what makes outcomes governable.

Business Outcomes

At the top are the outcomes the enterprise actually cares about: renewing a contract, issuing a refund, procuring equipment, onboarding a customer, or resolving an incident. These are the activities against which value, risk, and accountability are ultimately measured.

Transaction Governance

Beneath those outcomes sits the durable execution context. This layer preserves state and intent, carries delegated authority, applies policy, manages approvals and budgets, coordinates reliability and recovery, and maintains the audit and replay history of the activity.

AI Gateway

The gateway provides the runtime enforcement boundary. It authenticates and authorizes actors, intercepts and routes traffic, mediates protocols, handles credentials, enforces policy, and records the interactions that make up the transaction.

Enterprise Resources

Below the gateway are the resources agents use: models, MCP tools, APIs, other agents, databases, event systems, enterprise applications, and real-time data.

The diagram makes the division of responsibility explicit while preserving the gateway as the foundation of the architecture.

This framing is consistent with the direction of the market without assuming that any single vendor has already solved the complete problem.

Forrester's emerging agent-control-plane category is particularly relevant. It distinguishes among systems used to build agents, systems used to orchestrate business processes, and a third independent plane that supervises heterogeneous agents and applies governance across them. Deloitte's research helps explain why such a layer is becoming necessary: only 21% of surveyed organizations currently report mature agentic-AI governance even as adoption accelerates.

The architecture is following the risk.

SECTION 21

When Unification Is—and Is Not—the Right Answer

None of this means every enterprise should immediately replace its gateways with one product. Specialization remains sensible in many environments.

Separate gateways may be appropriate when different teams own different platforms, security domains must remain isolated, an established API-management program is already operating successfully, AI workloads remain narrow, traffic classes have very different performance requirements, or organizational boundaries matter more than technical consolidation.

The goal should not be one physical gateway at any cost. A better objective is **one coherent governance model across however many enforcement points the architecture requires**.

Unification becomes more compelling when agents routinely cross models, tools, APIs, and data; end-user identity must remain attributable throughout the activity; policy needs to span systems; several agents participate in the same transaction; enterprise-wide audit and incident reconstruction are required; or operational duplication has become material.

The architecture should follow the governance requirement, not the other way around.

SECTION 22

Recommendations for Enterprise Architects

The AI gateway market will continue to change, so enterprises need evaluation principles that remain useful even as products and protocols evolve.

01 **Define the Boundary Before Selecting the Product**

Decide what the gateway is expected to govern before comparing vendors. Is the boundary limited to models, does it include models and tools, does it span the full agent interaction surface, or will it eventually need to govern complete agent-driven transactions? A comparison made before this question is answered risks placing products designed for different problems in the same category.

02 **Follow Identity End to End**

Do not stop after authenticating the agent. Determine whether every consequential downstream action can be attributed to the human or workload that ultimately caused it, and whether delegated authority remains constrained at each step.

03 **Treat MCP as Architectural, Not Merely as Another Connector**

MCP affects tool discovery, identity, authentication, authorization, credentials, sessions, audit, and policy. Adding MCP support is therefore not equivalent to connecting one more backend; it changes how agents discover and act on enterprise capabilities.

04 **Evaluate Action Governance, Not Just Model Governance**

Examine how the platform treats reads, writes, irreversible operations, financial activity, external communication, and production changes. The most consequential risk may occur several calls after the model produces its answer.

05 **Decide What You Actually Want to Own**

Building, buying, and open source can all be valid choices. Base the decision on strategic differentiation, internal platform capability, desired control, operational ownership, extensibility, protocol risk, and long-term economics rather than on the apparent simplicity of an initial implementation.

06 **Optimize for Change**

MCP will not be the final AI infrastructure protocol. Models, agent frameworks, standards, and deployment patterns will continue to change. The most valuable architectural attribute may be the ability to absorb new resources and protocols without rebuilding the governance plane.

07 **Plan for Durable Activity**

If agent tasks can last for hours or days, infrastructure designed entirely around stateless requests will eventually become limiting. State, approvals, timers, events, retries, and recovery need a durable home.

08 **Design Observability Around Outcomes**

Tracing model calls is necessary. Tracing the complete agent-driven business activity is more useful, particularly when operators, security teams, auditors, and business owners need to understand the same incident from different perspectives.

09 **Keep Governance Independent of the Agent Framework**

Enterprises are unlikely to standardize permanently on one model, runtime, framework, cloud, or tool implementation. Governance should survive changes in all of them. This principle aligns with Forrester's argument for an independent agent control plane outside the environments used to build and orchestrate agents.

SECTION 23

From AI Gateway to AI Governance Runtime

The AI gateway market in 2026 is best understood not as a mature category approaching its final form, but as an architectural boundary in motion.

The progression so far has moved from the **LLM Gateway**, focused on model access, abstraction, cost, and routing, to the **AI Gateway**, which adds enterprise governance. The **Agent Gateway** extends the boundary to tools, MCP, and agent connectivity, while the **Unified AI Gateway** brings models, tools, APIs, agents, and data under a common enforcement model.

The next step may be **Transaction Governance**, where intent, authority, durable state, approvals, reliability, and outcomes become first-class concerns. Beyond that lies the possibility of an **AI Governance Runtime**: an operating layer for policy, execution, security, reliability, observability, cost, audit, and forensic reconstruction across autonomous software.

The internal architecture explored for an Agent Transaction Runtime follows this progression explicitly. It begins by attaching transaction identity, policy, approvals, and tracing to gateway interactions, then expands into execution reliability, delegation, durable state, and eventually a broader layer for governance and execution.

This should be treated as a hypothesis about where enterprise AI infrastructure may go, not as an inevitability. Even so, the forces pushing in that direction are increasingly visible.

Gartner predicts that by 2028, 33% of enterprise software applications will include agentic-AI capabilities, up from less than 1% in 2024, and that at least 15% of day-to-day work decisions will be made autonomously. Gartner's 2026 research also argues that organizations will increasingly favor platforms organized around workflow outcomes rather than purely assistive AI, especially in approval-heavy and timing-sensitive processes.

If those trends continue, AI infrastructure will have to govern more consequential activity over longer periods of time. Request governance alone will not be enough.

CONCLUSION

The Gateway Is the Foundation, Not the Destination

The first AI gateways answered a relatively narrow question: how can applications connect safely to multiple models? The AI gateway of 2026 is being asked something much larger: how can autonomous software connect safely to the enterprise?

Answering that question means governing access not only to models, but also to tools, APIs, agents, events, and data. It requires identity that extends beyond authentication, a clear distinction between reasoning and authority, controls over actions rather than prompts alone, and consistent visibility across systems that were historically managed independently.

Enterprises evaluating gateways today therefore need to make three decisions together. They must define the gateway's scope, decide whether governance should remain specialized or operate through a unified enforcement model, and choose whether to build, buy, adopt open source, or combine those approaches.

A fourth question is now emerging: what happens when the object being governed is no longer a request, but a business transaction?

An AI gateway can determine whether an agent may call a model, discover a tool, or execute a tool call under a given policy. Those capabilities will remain essential. Yet the business ultimately cares about a different outcome: **did the agent safely accomplish what it was authorized to accomplish?**

Answering that requires durable knowledge of who initiated the activity, why it existed, how authority was delegated, what data was used, which decisions were made, which policies applied, which humans approved consequential actions, what failures occurred, and what outcome resulted.

That is a broader problem than gatewaying. The deeper architectural asset may not be an ever-larger proxy, but the **transaction graph and its policy history**: the record that connects human, agent, model, tool, data, decision, approval, and action.

From that foundation, new governance capabilities become possible, including risk scoring, anomaly detection, policy recommendations, adaptive approval thresholds, agent evaluation, cost optimization, failure prediction, incident reconstruction, and forensic replay.

The AI gateway is therefore likely to remain one of the most important control points in the agentic enterprise. But it may prove to be the foundation rather than the destination: the enforcement layer beneath a broader runtime for identity, authority, policy, approvals, reliability, observability, audit, and business outcomes.

The first generation of AI gateways governed model traffic. The next will govern agent actions. The infrastructure that follows may govern the transactions autonomous software ultimately performs on behalf of the enterprise.

aklivity

The gateway governs access. The transaction runtime governs outcomes. Enterprises need both, under one governance model.

Sources referenced in this paper include published research and product documentation from McKinsey, Deloitte, Gartner, Forrester, Microsoft, Google, Kong, Cloudflare, Envoy AI Gateway, agentgateway, Arcade, LiteLLM, and Redpanda. Vendor names appear as market evidence; no endorsement is implied.