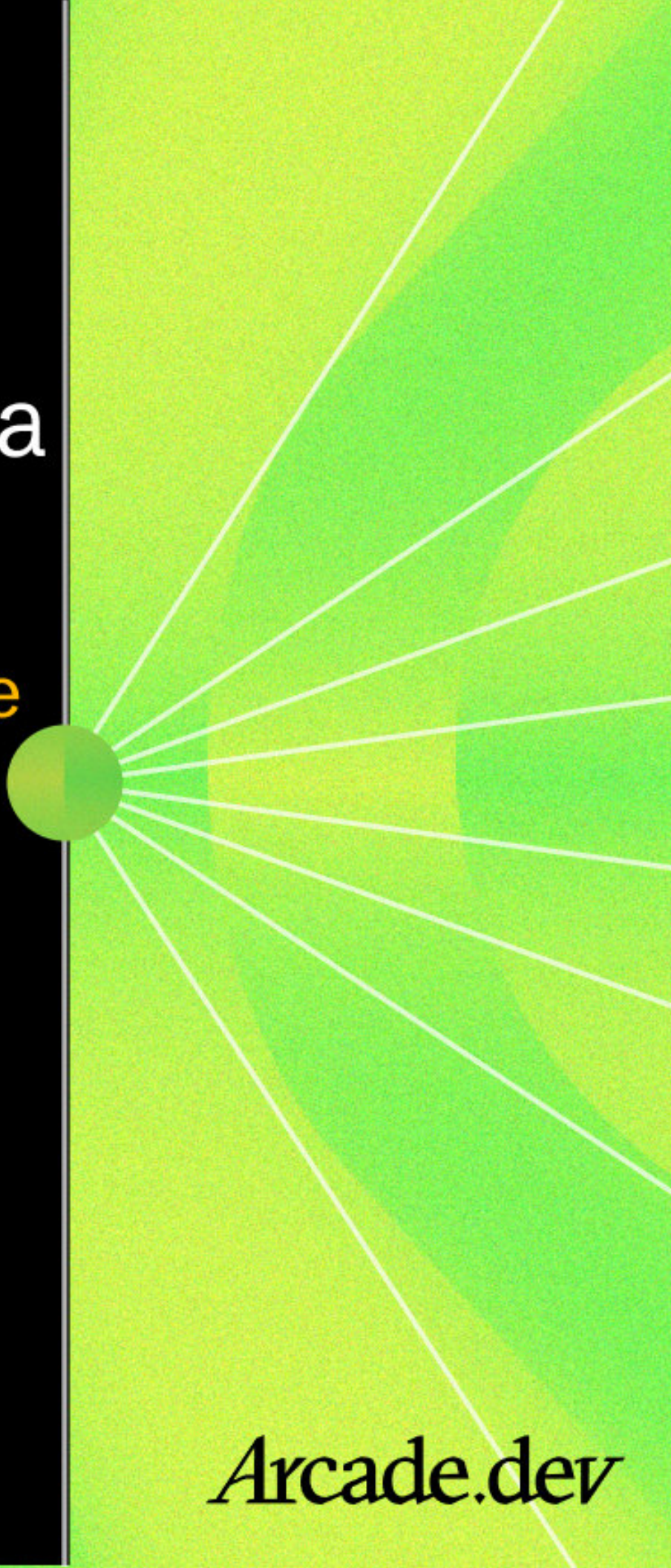
The End of the Gateway Era

Why MCP Runtimes Replace API Gateways for Agentic AI



Arcade.dev

EXECUTIVE SUMMARY

The enterprise software industry is experiencing a fundamental architectural shift. AI agents are autonomous software that reasons, plans, and takes action on behalf of users are rapidly moving from prototype to production. To connect these agents to enterprise systems, the industry has converged on the Model Context Protocol (MCP), an open standard originally developed by Anthropic in late 2024.

Predictably, incumbent API gateway vendors Kong, MuleSoft, and others are racing to claim this new category. Their pitch is familiar: "You already manage APIs through us. Now manage MCP through us too." It's a reasonable narrative. It's also wrong.

This paper argues that API gateways and MCP runtimes are fundamentally different architectural constructs, solving different problems at different layers of the stack. Bolting MCP support onto a gateway designed for stateless REST traffic is not a pivot, it's a patch. And enterprises that treat it as equivalent will pay the price in security gaps, integration complexity, and agents that don't work.

Arcade is the only runtime purpose-built for MCP from the ground up handling per-user authorization, reliable tool execution, and centralized governance as native capabilities, not plugins.

1. THE LAYER COLLAPSE: WHY AGENTS CHANGE THE ARCHITECTURE

Before we examine specific vendors, it's worth understanding the macro trend that makes this entire discussion inevitable.

In the traditional model, users interact with applications. Applications call APIs. A gateway sits between them routing, authenticating, rate limiting. The proxy is the control point because it's the choke point: all traffic flows through it.

In the agentic model, this topology inverts. **The agent is the proxy.** A user talks to an agent. The agent reasons, plans, and calls tools on the user's behalf. It already handles mediation; it already *is* the routing layer, the decision layer, the orchestration layer. Adding a traditional proxy in front of the tools the agent calls doesn't add a control point. It adds a redundant hop that cannot see into the execution context that matters: which user, which action, which permission, right now.

The control point in an agentic architecture is not the network layer between the agent and the tool. It's the **execution layer where the tool runs** where credentials are resolved, permissions are checked, and actions are taken on behalf of a specific human. That's the runtime.

This is why the "MCP gateway" framing is architecturally backwards. What you need is not another proxy. You need a runtime that governs what happens when the agent's request arrives at the tool, one that understands identity, enforces policy at the function level, and manages the credential lifecycle for every user and every action.

The gateway era was defined by the proxy as the control point. The agentic era is defined by the runtime as the control point. The layers have collapsed. The architecture must follow.

2. THE ARCHITECTURAL MISMATCH

With that topology in mind, let's examine the specific architectural assumptions that break when you try to serve agents with infrastructure designed for applications.

2.1 What API Gateways Were Built For

API gateways emerged in the microservices era to solve a specific problem: managing north-south HTTP traffic between external consumers and internal services. They excel at rate limiting, authentication at the perimeter, request routing, protocol translation (REST to gRPC), and observability for request-response patterns.

Kong, originally built on NGINX, and MuleSoft's Flex Gateway are optimized for this model. Every request is stateless. Every response is terminal. The gateway is a pass-through layer that applies policies to individual HTTP transactions.

This architecture was designed for a world where the caller is an application with structured inputs, deterministic behavior, and a known request schema. AI agents are none of these things.

2.2 What AI Agents Require

The shift to agentic AI introduces a fundamentally new consumer of enterprise services: autonomous software that reasons about which tools to use, executes multi-step workflows, and acts with delegated authority on behalf of individual users. The architectural requirements this creates are incompatible with the API gateway model.

Tool discovery and execution. Agents don't call fixed API endpoints; they discover available tools at runtime and select among them based on user intent. That selection only works if tools are built for it: rich semantic descriptions that help the model choose correctly, consistent schemas across services, and runtime-managed retry, pagination, and error handling that the agent never has to reason about. Most MCP servers are thin API wrappers that fail on all three counts. When a user says "update the Acme deal," the wrapper asks for `opportunity_id`, `owner_id`, `stage_enum`, `close_date`. The agent hallucinates or retries blindly. The gap between user intent and API parameter is where most agent failures occur and a proxy layer has no mechanism to close it.

Stateful, multi-step sessions. An agent's interaction with tools is conversational, not transactional. An agent may discover available tools, invoke one, process the result, invoke another based on that result, and iterate all within a single reasoning loop. The gateway model of "apply policy → forward request → return response" doesn't map to this flow.

Per-user, per-tool authorization. This is the hardest problem and the one gateways are least equipped to solve. In a traditional API, authorization happens at the endpoint level: does this API key have access to `/api/v1/messages`? In agentic systems, authorization must happen at the tool level, scoped to the individual user.

The failure modes here are instructive. Give an agent its own identity with broad permissions: an intern using that agent can now bypass their own access controls. Swing the other direction and let the agent inherit the user's full permissions: one prompt injection now cascades through every system that user can touch. The right answer sits at the intersection of what *is this agent allowed to do* AND *what is this user allowed to do* evaluated per action, at runtime. That evaluation cannot happen at the network layer.

Tool-level governance. Enterprises need fine-grained policies: "Sales agents can read Salesforce opportunities but not delete records." "Engineering agents can access GitHub repositories but not the production branch." This is per-tool, per-action, per-user policy not endpoint-level rate limiting or API key validation. Without it, security blocks production deployment entirely not out of caution, but because there is genuinely no way to verify what agents are doing, on whose behalf, with what permissions.

The Architectural Mismatch: Gateway vs. Runtime

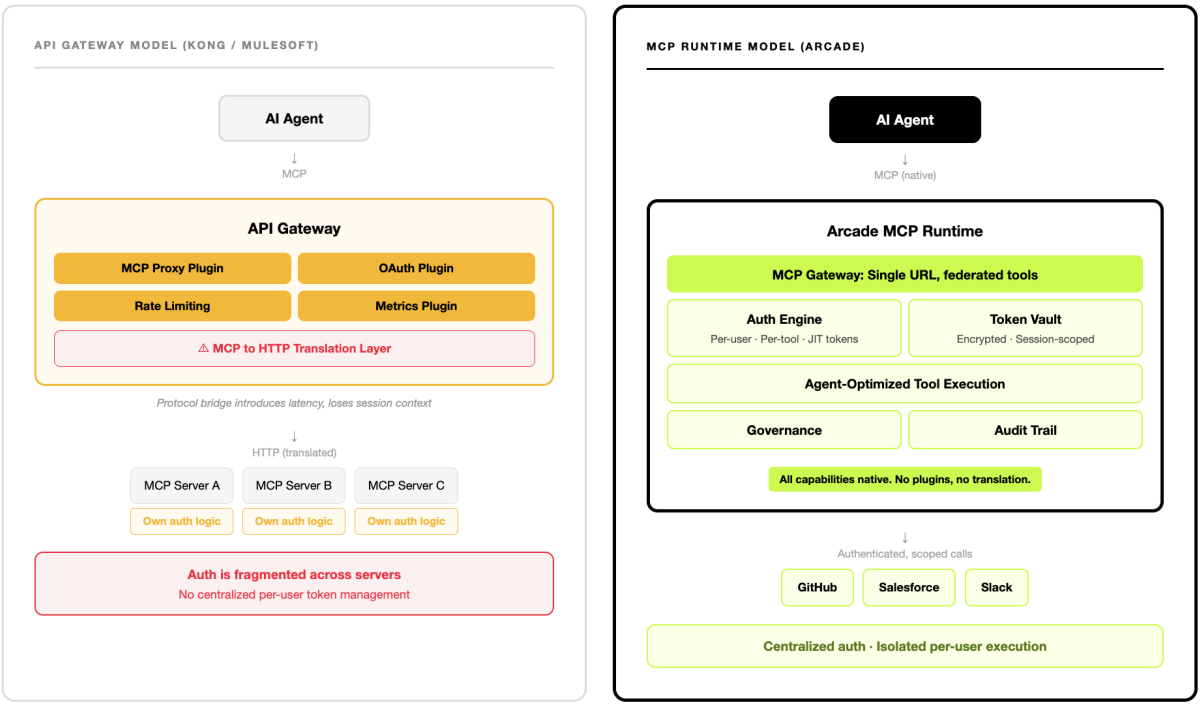


Figure 1: For agentic workloads, the gateway adds a translation hop and fragments authorization. The runtime centralizes both natively.

2.3 Why "MCP Gateway" Is the Wrong Category

When Kong adds an "AI MCP Proxy plugin" or MuleSoft's Flex Gateway adds protocol support for agent tools, they are bridging a protocol gap not solving the architectural problem.

The result: agent traffic hits the gateway, gets translated into HTTP, policies are applied using HTTP-era constructs (API keys, rate limits, endpoint ACLs), the request is forwarded to a tool server, and the response is translated back. This translation layer introduces latency, loses session context, and most critically cannot enforce per-user, per-tool authorization.

The gateway knows a request came in. It doesn't know which user the agent is acting for, what that user's specific permissions are for this specific tool, or whether the user has authorized this particular action.

Gateways are request routers. What agents need is an execution runtime. You can't solve a runtime problem with a proxy.

The actual work of agentic infrastructure acquiring per-user OAuth tokens just-in-time, enforcing function-level authorization, managing credential lifecycle, handling retry and error logic at the tool layer none of this happens at the proxy. All of it happens at the runtime. An enterprise that

deploys a proxy in front of its MCP servers and calls it an "MCP gateway" has deployed a network hop, not a control plane. The authorization problem remains unsolved. The credential management problem remains unsolved. The tool reliability and governance problem remains unsolved.

The runtime is the gateway.

Authorization Flow: Perimeter Auth vs. Per-User Runtime Auth

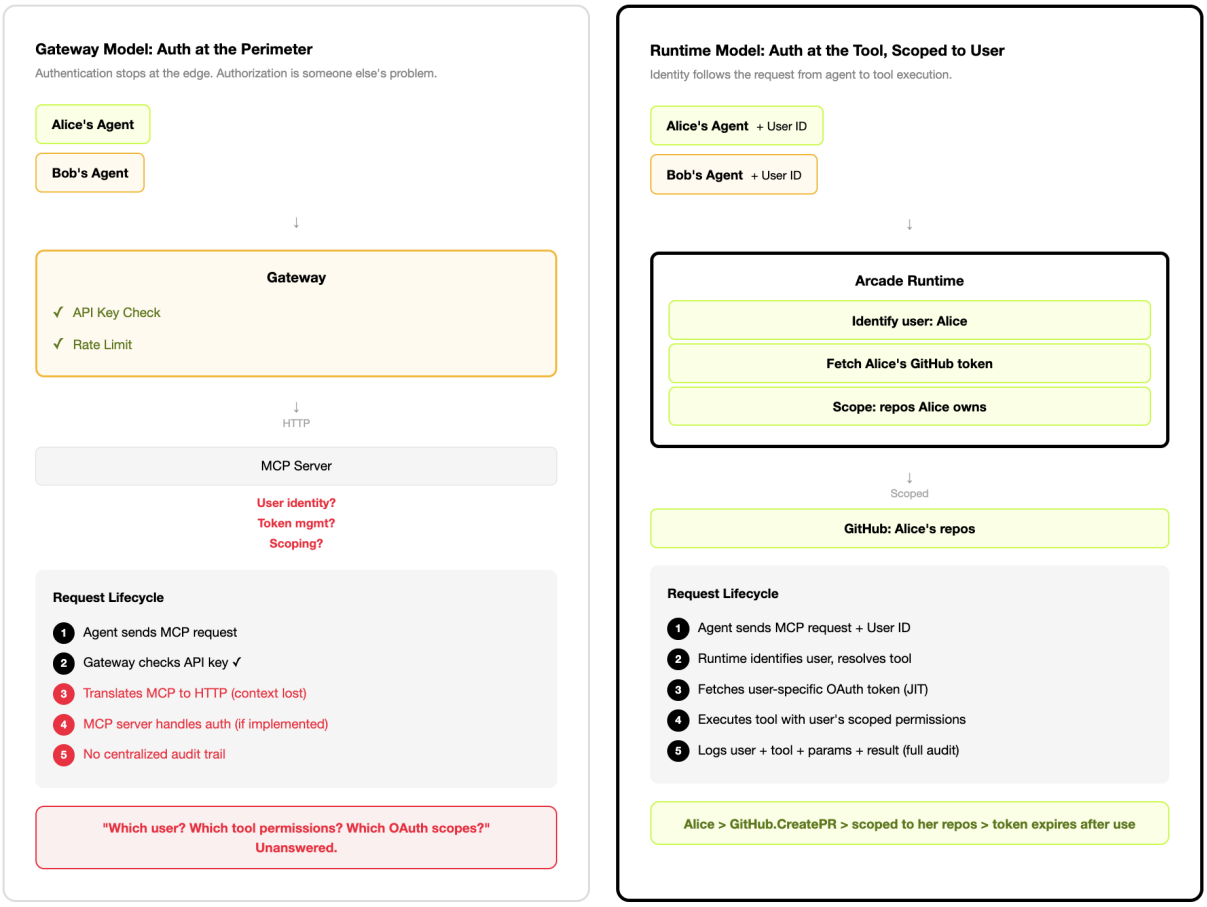


Figure 2: Gateway auth stops at the perimeter. Runtime auth follows the request through to per-user, per-tool execution.

3. WHERE THE INCUMBENTS ARE TODAY

Kong and MuleSoft are both adding MCP support. Both are doing it by extending architectures designed for stateless HTTP and both hit the same wall for the same reason.

3.1 Kong: The Stateless Policy Problem

Kong's MCP implementation (Gateway 3.12, October 2025; MCP Registry, February 2026) follows its established plugin model: the AI MCP Proxy plugin translates between MCP and HTTP, the AI MCP OAuth2 Plugin handles authentication, and the MCP Registry provides a centralized catalog within Kong Konnect.

The problem isn't the plugin model itself. It's that Kong's entire policy engine is stateless by design; it evaluates each request in isolation. That worked for REST, where every request is independent. It breaks for agents, where a request only makes sense in the context of the session carrying it. Kong can tell you a request arrived. It cannot tell you that it's step 3 of a 6-step agent workflow, acting on behalf of Alice, who authorized this specific scope 4 minutes ago. Without that context, you cannot enforce per-user, per-tool authorization the policy would need to understand identity, session state, and entitlements simultaneously.

This is what the authentication story reveals. Kong's MCP auth relies on Key Auth, OpenID Connect, and OPA perimeter constructs repurposed for an execution problem. They answer "is this API key valid?" The question that matters is: "does Alice have a valid, scoped OAuth token for this specific GitHub repository, and is she authorized to create a pull request in it right now?" No stateless proxy can answer that. It requires a runtime that participates in the execution.

3.2 MuleSoft: The Wrapping Problem

MuleSoft's pitch is straightforward: you have 1,000 APIs. The MCP Connector (GA July 2025) wraps them into MCP tool definitions. Agent Fabric (GA October 2025) adds discovery and cataloging. Flex Gateway handles MCP traffic. Your existing API estate becomes agent-accessible.

The problem is that wrapping an API in an MCP tool definition doesn't make it an agent tool. It makes it an API with a different interface. The agent still has to understand the API's mental model, its object IDs, its enums, its error codes, its pagination model. When a user says "update the Acme deal," the MuleSoft-wrapped Salesforce tool still asks for `opportunity_id`, `owner_id`, `stage_enum`, `close_date`. The translation failure doesn't disappear, it just moves. And because MuleSoft's entire tooling starts from the API catalog and works backward, every tool in the ecosystem inherits this by default. There is no path from "wrap your existing APIs" to "agent-optimized tools." Those are different things built from different starting points.

The authorization problem is structural for the same reason. When MuleSoft wraps Salesforce, the tool inherits Salesforce's auth model. When it wraps GitHub, it inherits GitHub's auth model. Each underlying API manages credentials independently because that's how integration platforms work. There is no unified per-user credential layer across tools, because MuleSoft was never designed to provide one. Adding MCP on top doesn't create it.

Flex Gateway's traffic policies PII detection, rate limiting, and spike protection operate at the message level. They can inspect a payload. They cannot enforce that User A's agent has read-only access to Salesforce while User B's agent has write access. That distinction lives in the execution context, not the traffic stream.

MuleSoft can make your existing APIs visible to agents. It cannot make them work reliably for agents, authorize them correctly per user, or govern them at the tool level. Those problems don't start at the API layer and an integration platform that starts there cannot work its way back to solving them.

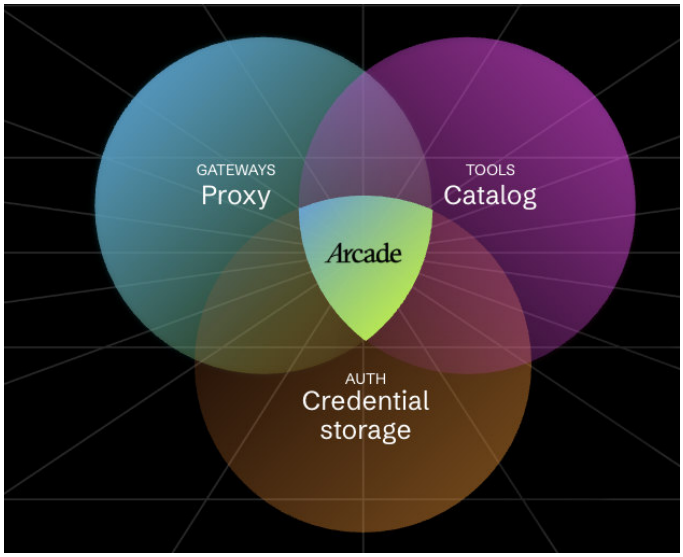
3.3 The Implication

Both vendors are extending proven architectures to cover a new protocol. But "covering MCP" and "solving the runtime problem" are not the same thing. You can proxy MCP traffic without ever touching the authorization, credential lifecycle, or tool reliability problems that actually determine whether agents work in production.

So what does a purpose-built alternative look like?

4. WHAT A PURPOSE-BUILT MCP RUNTIME LOOKS LIKE

Production agentic infrastructure requires three capabilities working in concert: **auth** (per-user credential management, entitlements, and least-privilege enforcement), **tools** (a catalog of agent-optimized, production-quality tools), and **governance** (centralized control over what runs, on whose behalf, with what permissions). Legacy gateway vendors solve one piece. Production requires all three, unified in a single runtime.



A gateway without tools is useless. It routes requests, it doesn't make them succeed. Community MCP tools have inconsistent quality, poor security posture, no SLA, and no roadmap you control. Routing traffic to unreliable tools is not infrastructure.

Tools without auth won't ship. MCP servers are not multi-user by default. Enterprise security teams will not approve agent deployments without delegated user authorization, entitlement integration, and least-privilege enforcement. Without these, every team building agent must first build a bespoke identity layer before their first agent runs a single tool in production.

Auth and tools without governance are blocked entirely. Shadow tool registries proliferate. Engineering effort is duplicated across teams. Approval processes are inconsistent. There is no audit baseline. Security blocks production not out of caution, but because there is genuinely no way to verify what agents are doing, on whose behalf, with what permissions.

Arcade was designed from inception as the runtime that unifies all three. The word "runtime" is deliberate. A gateway is a pass-through it routes traffic and applies policies at the perimeter. A runtime executes. It serves tools with tightly coupled primitives around authorization, token management, token refresh, retry logic, and error handling all specific to the tool layer, all operating within the execution context of the individual user and the individual action.

4.1 Per-User Authorization as a First-Class Primitive

Every MCP request in Arcade carries two identity layers: the project-level key (which application is calling) and the user-level identity (on whose behalf the action is taken). When Alice's agent calls GitHub.CreatePullRequest, Arcade authenticates Alice against GitHub using her specific OAuth tokens, scoped to her specific repositories and permissions. Tokens are acquired just-in-time, scoped to the specific tool and user, encrypted at rest, and never exposed to the LLM or the MCP client.

In the gateway model, authentication happens at the perimeter. Is this *a valid API key?* but authorization must be handled elsewhere: custom middleware, the MCP server itself, or the application layer. There is no centralized runtime managing which user, which tool, which action, which scope.

Critically, Arcade does not ask enterprises to redefine authorization policies inside Arcade. Enterprises already have entitlement systems and identity providers Okta, Entra, SailPoint. Arcade hooks into those existing systems, acquires scoped tokens at runtime, and enforces what the enterprise has already defined. The runtime delegates authorization; it doesn't duplicate it.

Every tool invocation is logged with user identity, tool name, parameters, and results in a full chain of custody for every agent action, exportable to SIEM via OTel. SOC 2 Type 2 certification validates these controls through independent audit.

4.2 Agent-Optimized Tools, Not API Wrappers

Arcade's tool catalog spanning GitHub, Linear, Slack, Salesforce, Google Workspace, and hundreds more is purpose-built for agent consumption. When a user says "make the intro paragraph in my Google Doc sound friendlier," Arcade's tools translate that intent into `segmentId=gz49hg56, index=350, text='your friendlier message'`. The agent never has to consider anything beyond "intro paragraph."

Tools provide rich semantic descriptions optimized for LLM tool selection. Schemas are consistent across all tools regardless of the underlying API. Error handling returns agent-interpretable messages, not HTTP status codes. Pagination, rate limiting, and retry logic are handled by the runtime, invisible to the agent.

This is the difference between wrapping an API and building a tool. A wrapper exposes the API's complexity to the agent. A tool abstracts it.

Tool Quality: API Wrappers vs. Agent-Optimized Tools

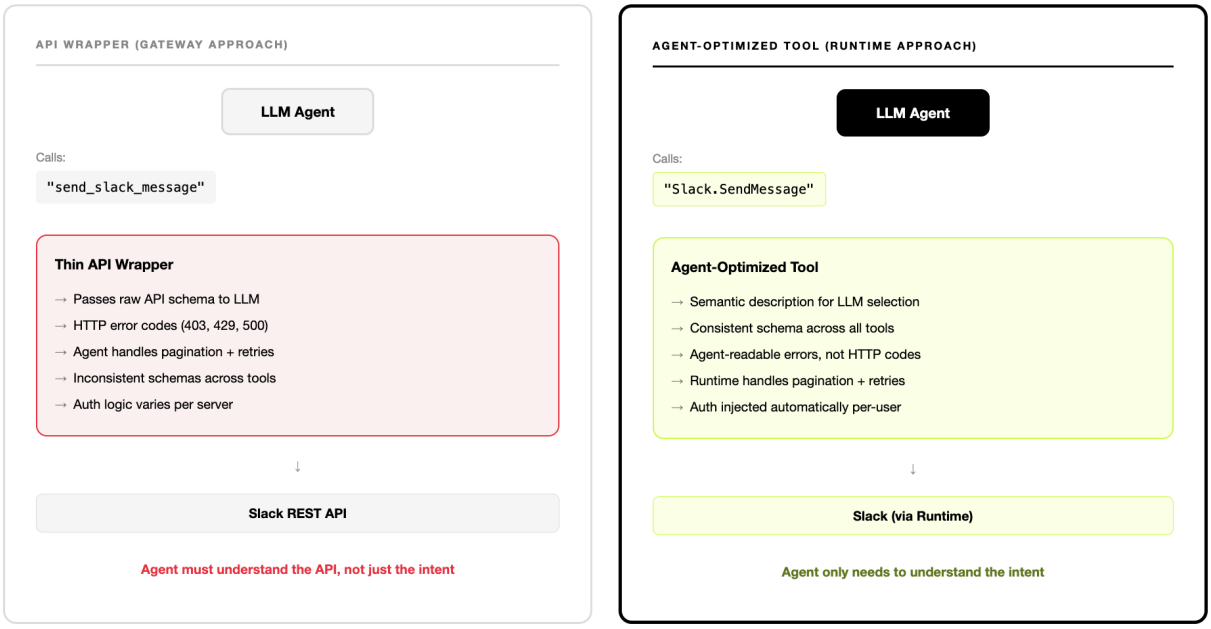


Figure 3: API wrappers expose complexity to the agent. Agent-optimized tools abstract it, improving reliability and reducing token waste.

4.3 Governance: Federated Gateways with Function-Level Control

Arcade's MCP Gateway provides a single URL that federates tools from multiple sources Arcade's catalog, custom-built tools, and third-party MCP servers into a curated collection. This is not a protocol bridge. It's a runtime-level composition layer.

An enterprise can create Gateways scoped by team, function, or security boundary: Engineering gets GitHub, Linear, and internal deployment tools. Sales gets Salesforce, Gong, and email tools. Support gets Zendesk, Slack, and knowledge base tools. Each Gateway is a single URL, usable in any MCP client Cursor, Claude Desktop, VS Code, ChatGPT, or custom applications.

Each Gateway can carry embedded LLM instructions skills and context that help the agent understand how to use the tools effectively. And each Gateway is identity-scoped: assignable to individual users, groups, service accounts, or non-human identities.

Governance is native: which tools appear in which Gateway, who can access them, and what actions they can perform are all managed centrally, all enforced by the runtime.

5. THE HISTORICAL PARALLEL

If this pattern feels familiar, it should. We've seen it before.

When the industry moved from monoliths to microservices, traditional load balancers tried to pivot into API gateways. They added routing rules, basic authentication, and protocol support. But they couldn't provide the developer experience, plugin ecosystem, and service mesh integration that purpose-built gateways offered. The market chose Kong, Apigee, and MuleSoft not F5 and HAProxy with plugins.

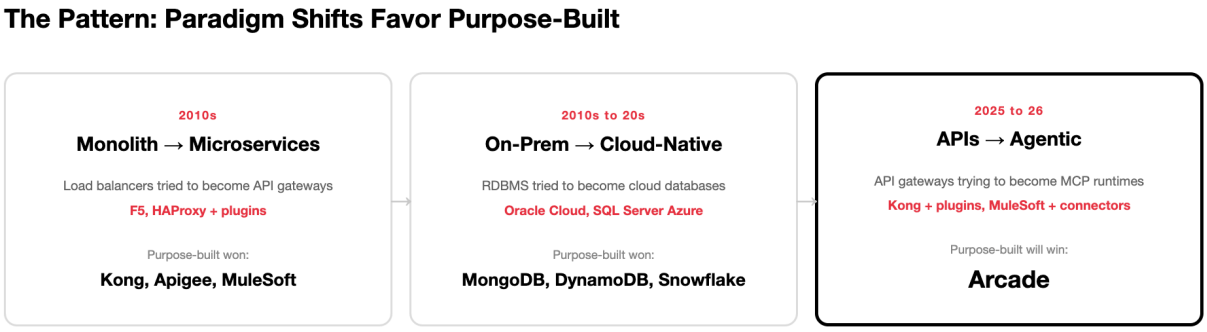


Figure 4: When paradigms shift, retrofitting loses to purpose-built. Every time.

When the industry moved from on-premise databases to cloud-native data, traditional RDBMS vendors tried to pivot with "cloud editions." But they couldn't match the scalability, developer ergonomics, and operational simplicity of purpose-built cloud databases. The market chose MongoDB and DynamoDB not Oracle Cloud.

The pattern is consistent: when the underlying architectural paradigm shifts, incumbents that retrofit new capabilities onto old architectures lose to purpose-built solutions that start from the new paradigm.

The shift from API management to agentic infrastructure is the same pattern. Gateways are retrofitting MCP support onto architectures designed for stateless HTTP. Arcade was built from

the ground up for the world where agents not applications are the primary consumers of enterprise services.

6. MAKING THE DECISION: A FRAMEWORK FOR EVALUATORS

For technical leaders evaluating infrastructure for production agentic systems, the key question is not "can my gateway handle MCP traffic?" It can, with plugins.

Decision Framework: Gateway vs. Runtime

CAPABILITY	API GATEWAY (KONG / MULESOFT)	MCP RUNTIME (ARCADE)
Protocol	REST-native + MCP bridge plugin	**MCP-native (JSON-RPC, SSE)**
Session model	Stateless (per-request)	**Stateful (per-session, per-user)**
Authorization	Perimeter auth + endpoint policies	**Per-user, per-tool, JIT tokens**
Tool quality	API wrappers (schema passthrough)	**Agent-optimized (semantic, consistent)**
Tool discovery	Static catalog / registry	**Dynamic catalog/registry, context-aware, gateway-scoped**
Token management	External (env vars, secrets managers)	**Built-in vault, zero LLM exposure**
Governance	Endpoint + traffic-level policies	**Tool + action + user-level policies**
Multi-user isolation	Requires custom implementation	**Native (session-scoped identity)**
Audit trail	Traffic logs (request/response)	**Full execution context (user, tool, params, result)**
Time to production	Weeks (custom auth + wrapper dev)	**Hours (catalog + gateway config)**
REST API management	**Mature, proven at scale**	Not the primary use case

Figure 5: For agentic workloads, the architectural gap is systemic, not feature-level. Gateways excel at what they were built for: REST traffic management.

The question is: **"Does my infrastructure understand what an agent is doing, on whose behalf, with what permissions, and can it enforce policy at that granularity?"**

To answer that concretely, evaluate against these criteria:

Per-user credential management. Does the platform acquire, scope, refresh, and revoke OAuth tokens for each individual user, for each individual tool, at the moment of invocation? Or does it rely on shared service accounts, static API keys, or credentials managed outside the platform?

Function-level authorization. Can you enable `process_claim` and `get_claim_status` while blocking `deny_claim` for a specific user, through a specific agent, at runtime? Or does authorization stop at the tool level, or worse, at the endpoint level?

Entitlement integration. Does the platform hook into your existing identity and governance systems Okta, Entra, SailPoint, or homegrown and enforce what you've already defined? Or does it require you to redefine policies in yet another system?

Tool quality. Are tools purpose-built for agent consumption with semantic descriptions, consistent schemas, and runtime-managed error handling? Or are they API wrappers that expose the underlying API's complexity to the LLM?

Audit and observability. Does every tool invocation produce a complete record of user identity, tool, function, parameters, result exportable to your SIEM? Or does the audit trail stop at the network perimeter?

If the answer to any of these requires custom middleware, bespoke authorization logic, or MCP server-level implementation you don't have a runtime. You have a proxy with aspirations. Which means your team is responsible for manually maintaining the stitched-together pieces to attempt to fill the gaps.

7. CONCLUSION

The API gateway served the industry brilliantly for a decade. It brought order to the chaos of microservices, standardized authentication and rate limiting, and gave platform teams a control plane for their API estate.

The risk for enterprises isn't choosing the wrong vendor. It's believed that proxying MCP traffic is equivalent to solving the runtime problem and building a production agent strategy on that assumption. That gap doesn't surface in a proof of concept. It surfaces when you try to deploy agents at scale: authorization exceptions you can't patch at the network layer, credential management living in custom middleware, tool failures the agent can't interpret, an audit trail that stops at the perimeter. At that point you're not evaluating infrastructure. You're doing incident response.

The agent is already the proxy. It mediates between the user and the infrastructure. The question is not whether to put another proxy in front of the tools, it's whether the execution layer where those tools run is governed, secured, and authorized at the granularity that production demands.

The gateway era gave us centralized control for stateless traffic. The runtime era demands centralized control for agentic execution. Those are not the same problem and the enterprises that recognize that distinction first are the ones that ship production agents while everyone else is still debugging their proxy configuration.

ABOUT ARCADE

Arcade.dev is the industry's first MCP runtime enabling AI to take secure, real-world actions. As the MCP runtime, Arcade is uniquely able to deliver secure agent authorization, high-accuracy tools, and centralized governance. Arcade helps teams at some of the largest organizations deploy multi-user AI agents that take actions across any system with granular permissions and complete visibility and no complex infrastructure required. Learn more and try it for free at www.arcade.dev.

REFERENCES

1. Kong, "Introducing Kong's Enterprise MCP Gateway for Production-Ready AI," konghq.com, 2025.
2. Kong, "Kong AI/MCP Gateway and Kong MCP Server Technical Breakdown," konghq.com, 2025.
3. Kong, "Kong Introduces MCP Registry in Kong Konnect," konghq.com, February 2026.
4. MuleSoft, "Introducing Governance for Agent Interactions With Support for A2A and MCP," blogs.mulesoft.com, 2025.
5. Salesforce, "New MuleSoft Innovations Enable Secure, Scalable AI Agent Orchestration," salesforce.com, 2025.
6. MuleSoft, "What's Next for MuleSoft: Q1 2026 Product Roadmap," blogs.mulesoft.com, 2026.
7. Palma.ai, "MCP vs API Gateways: Why AI Agents Need Governance," palma.ai, 2025.
8. Arcade, "MCP Gateways," docs.arcade.dev, 2026.
9. Arcade, "Arcade Tools for MCP Clients," blog.arcade.dev, 2025.
10. Anthropic, "Model Context Protocol Specification," modelcontextprotocol.io, 2024-2025.