The Missing Layer

Why Enterprise Agents Need an MCP Runtime

Arcade.dev

Executive Summary

Most enterprise agent projects stall not because the AI doesn't work, but because the infrastructure required to deploy agents safely in production does not yet exist. MCP standardized how agents discover and invoke tools, but it defines the communication layer, not the execution layer. It does not handle authorization, execution reliability, or governance at scale.

That gap is an MCP runtime: a shared execution layer that evaluates permissions at the moment each action is taken, ensures tools behave predictably under failure, and provides centralized visibility and control as agent usage grows. Existing enterprise infrastructure (API gateways, workflow engines, service meshes) cannot fill this role, as it was designed for static, deterministic workloads. Not for software that reasons dynamically and acts on behalf of users.

Arcade provides this runtime. This paper examines why agent projects stall, what an MCP runtime must handle, and how Arcade closes the gap.

Teams don't shelve agents because they don't work. They shelve them because deploying them safely inside real enterprise environments turns out to be harder than expected—often at the exact moment pressure to prove ROI is increasing.

Demos prove the value quickly. Workflows that once required hours of manual effort are automated, and systems that never talked to each other are suddenly coordinated through a single agent.

That momentum slows as soon as those agents need to operate against production data and real users. Authorization becomes ambiguous. Reliability under retries and partial failures starts to matter. What looked impressive in isolation begins to feel risky when the blast radius is no longer hypothetical.

Projects don't fail outright. They stall — usually right when expectations are highest. Timelines slip. Reviews drag on. Questions from security and platform teams linger without clean answers. Meanwhile, executive confidence in AI initiatives starts to erode. Not because the value isn't there, but because the systems can't yet be defended as production-ready.

At that point, the problem is no longer agent capability. It's the infrastructure surrounding it—and the growing gap between what agents promise and what the enterprise is prepared to support.

The reason enterprise agent projects stall is not model quality or orchestration, but the absence of a shared execution layer. As soon as agents are expected to act on behalf of real users, against real systems, with irreversible consequences - enterprises need infrastructure that MCP intentionally does not define. That layer is an MCP runtime.

MCP Is Necessary — and Insufficient

Model Context Protocol (MCP) represents an important step forward for teams building agents. By standardizing how agents discover and invoke tools, MCP removes a significant amount of coordination friction. Tools become easier to reuse, agents become easier to assemble, and development workflows become more repeatable. For teams moving beyond one-off demos, this standardization matters.

But coordination is not the constraint that ultimately stops agents from reaching production. Once agents are expected to act inside real enterprise environments—on behalf of multiple users, across systems of record, with irreversible consequences— a different class of problems emerges. At that point, the challenge is no longer how an agent formats a request, but how that request is executed safely, consistently, and under existing enterprise constraints.

MCP defines invocation. It does not define enforcement. It specifies how an agent calls a tool, but not how permissions are evaluated when that call is executed. It standardizes schemas, but not retries, partial failures, or the handling of side effects. It makes tools discoverable, but it does not provide centralized visibility, auditability, or control once agents operate at scale.

These gaps are easy to overlook in development environments. They often remain invisible in early demos, where agents operate under a single user, with tightly scoped permissions, and failures are corrected manually. The illusion holds, until agents move into production workflows that touch real data and real users.

In practice, teams reach the same conclusion. MCP makes it easier for agents to discover and invoke tools, but it does not address what happens when those invocations touch real systems, real users, and irreversible state. As agents move into production, questions of authorization, execution reliability, and control can no longer be handled implicitly or left to individual agents. At that point, coordination is no longer the constraint. Execution is.

Production Requirement	What MCP Provides	What You Must Build
Tool connectivity	Protocol specification	Actual integrations to every system
User authentication	Nothing—no user context support	Complete OAuth infrastructure, token management, refresh logic
Tool selection & reliability	Discovery and schema format	Selection logic, filtering, error handling, retries
Error prevention	Tool schema definitions	Parameter validation, intent confirmation, guardrails
Audit and compliance	Communication standard	Logging, permission tracking, audit trails, governance

What an MCP Runtime Must Handle

Authorization at Execution Time

Authorization failures are one of the first places agent deployments break in production — not because identity systems are inadequate, but because authorization is evaluated at the wrong moment.

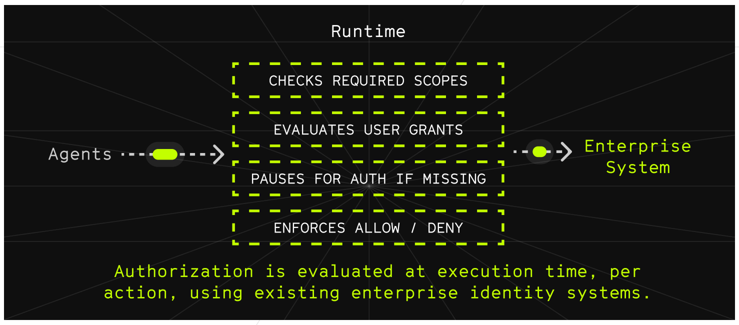
Agents do not operate through fixed interfaces or static sessions. They initiate actions dynamically, often on behalf of different users, across changing system state, and over extended periods of time. An action that is valid in one context may be invalid moments later, depending on permissions, role changes, or upstream system updates.

As a result, authorization cannot be treated as something that is resolved once and assumed thereafter. It must be evaluated at the moment an action is executed.

In early agent pilots, this distinction is easy to miss. Agents often run under a single user or test account, with permissions hard-coded or broadly scoped. Under those conditions, authorization appears to "just work." But as agents move into production—acting on behalf of many users, across multiple systems—that illusion collapses.

When authorization is not enforced at execution time, teams fall back on fragile workarounds: shared credentials, over-privileged service accounts, or implicit trust baked into tool calls. These approaches may keep agents running, but they undermine security, obscure accountability, and create real risk once agents touch production data.

Execution-time authorization closes that gap. Each action is evaluated in context, aligned to the correct user identity, and enforced consistently regardless of which agent initiates it or which tool is invoked. Permissions remain explicit.
Actions remain attributable. Revocation and policy changes take effect immediately.
At scale, this is not an optimization. It is a requirement.


Runtime
CHECKS REQUIRED SCOPES
EVALUATES USER GRANTS
PAUSES FOR AUTH IF MISSING
ENFORCES ALLOW / DENY
Agents
Enterprise System
Authorization is evaluated at execution time, per action, using existing enterprise identity systems.

**Security Implication:
Prompt Injection Can't Expose Credentials**

One of the biggest risks teams worry about with agents is prompt injection — an attacker manipulating the model into leaking secrets or escalating privileges.

In a runtime-mediated architecture, that failure mode is structurally eliminated. Sensitive credentials never enter the model's context.

Tokens are injected only at execution time, permissions are revalidated on every action, and revocation is enforced centrally.
Even if the prompt or model behavior is manipulated, there are no credentials to leak and no additional authority to exploit.
This isn't a prompt-level defense — it's a consequence of where execution authority lives.

Reliable Action Execution

Authorization determines whether an agent is allowed to act. Execution determines whether that action actually happens correctly once it does.

This is the second major fault line agent deployments encounter in production.

Agents operate on semantic intent, while enterprise systems require structured, deterministic operations. Bridging that gap means translating high-level instructions into actions with clear inputs, predictable side effects, and well-defined failure behavior.
In practice, that translation happens through tools.

Most MCP tools today are thin wrappers around existing APIs. They expose endpoints and parameters designed for human developers writing deterministic code, not for reasoning systems deciding how to act dynamically across changing contexts. As a result, agents are forced to reason through execution details the underlying systems were never designed to absorb.

The breakdown usually isn't dramatic at first.
One team builds a Slack integration for an early agent pilot. It works. A second team later needs Slack access for a different agent and rebuilds the integration from scratch. Error handling differs. Authorization is implemented slightly differently. Logging lives somewhere else.

As usage grows, these small inconsistencies start to matter. API changes break one agent but not another. Retries amplify failures instead of resolving them. Partial updates leave systems in inconsistent states. What appeared reliable in a demo becomes fragile once actions span multiple systems or must execute repeatedly under real load.

Reliable execution depends on more than calling the right endpoint. It requires tools with explicit contracts: clearly defined schemas, deterministic side effects, and structured error signaling that distinguishes between retryable failures, missing context, and fatal conditions. Without these guarantees, agents are left to infer execution semantics at runtime, which quickly becomes brittle as complexity increases.

Teams often discover that making agents reliable quietly turns into building and maintaining a parallel execution and tool-management layer.
Retries, error normalization, and compensating logic accumulate around brittle integrations, pulling focus away from improving agent behavior and recreating the same infrastructure across teams.

In practice, this means senior teams spend their time rebuilding tool execution and management infrastructure, rather than focusing on higher-leverage work like improving agent orchestration, refining context and decision boundaries, and making agents more effective at the tasks that actually matter.

An MCP runtime absorbs this complexity. It provides a shared execution layer where tools behave predictably, failures are handled consistently, and agents are insulated from the low-level mechanics required to safely operate enterprise systems.

Execution at Scale Requires a Runtime-Controlled Access Layer

As execution logic centralizes, teams quickly encounter a second-order problem: controlling how agents access tools as integrations and MCP servers multiply. Allowing agents to connect directly to individual servers recreates the same fragmentation and inconsistency that made execution brittle in the first place.

In production deployments, this leads teams to introduce a centralized access layer inside the runtime that brokers all tool execution.

This makes it possible to expose tools deliberately, manage versioning and failures centrally, and evolve integrations without rewriting agents or duplicating execution logic across teams.

Arcade's MCP Gateway is designed to serve this role as part of the runtime, providing a single, governed execution surface for agent tool access.

Learn more about MCP Gateways →

Control and Visibility at Scale

Authorization and execution failures are painful early. At scale, they become unmanageable.

As agent usage expands beyond a handful of pilots, teams begin to lose visibility first. It becomes unclear which agents can access which tools, under what conditions, and on whose behalf actions are being taken. Execution paths fragment across MCP servers and integrations. Failures surface inconsistently, with limited context about what happened or why.

These are not agent-level problems. They are system-level ones — and they tend to surface only after enough people depend on the system.

Agents operate locally. They reason about individual tasks, tools, and outcomes. They do not have a global view of execution across users, environments, or time. As a result, governance logic ends up scattered across prompts, tools, custom wrappers, and ad hoc monitoring systems. Standards drift. Behavior diverges. Control becomes hardest to assert precisely when it matters most.

In practice, teams attempt to compensate by rebuilding control infrastructure alongside their agents. Logging pipelines appear. Dashboards follow. Execution policies and review processes are layered on incrementally. This work grows quietly but relentlessly, often becoming a bottleneck long before agents reach meaningful scale.

At the same time, operational requirements accumulate. Execution behavior must be consistent across environments. Failures must be traceable across authorization, tools, and downstream systems. Retries, timeouts, secrets management, storage, and telemetry stop being edge cases and become shared defaults that must be enforced uniformly.

An MCP runtime provides a centralized control plane where execution can be observed, constrained, and audited consistently across agents, tools, and environments. It creates a single place to understand what actions are being taken, under whose authority, and with what outcomes—without pushing operational complexity into every agent or team.

At scale, this layer is not optional. It is what allows agent systems to remain trusted and controllable as they grow.

Why Existing Infrastructure Can't Play This Role

When agent deployments stall, teams naturally look to existing enterprise infrastructure to fill the gaps.

API gateways. Workflow engines. iPaaS platforms. Service meshes. These systems are mature, trusted, and deeply embedded in production environments. It's reasonable to ask whether they can be extended to support agents at scale.

In practice, they fall short for a simple reason: they were designed for a different class of workload.

Traditional enterprise infrastructure assumes static identities, deterministic execution paths, and well-defined service boundaries. Requests are expected to be well formed. Authorization is evaluated ahead of execution. Failures are handled predictably within known control planes.

Agent-driven workloads violate these assumptions. Agents initiate actions dynamically, often on behalf of different users, across multiple systems, and in response to probabilistic reasoning rather than predefined flows. Execution unfolds over time. Context changes. The same instruction may result in different actions depending on permissions, system state, and prior outcomes.

Existing infrastructure is not built to govern this mode of execution.

API gateways can validate schemas and enforce rate limits, but they do not evaluate user-aligned authorization at execution time. Workflow engines can orchestrate predefined paths, but they assume static credentials and deterministic steps. Service meshes provide observability between services, not governance over software acting on behalf of users.
As teams attempt to adapt these systems, a familiar pattern emerges.

Authorization logic is pushed into agents. Execution semantics are duplicated across tools.
Observability is stitched together from fragmented logs. Custom wrappers and glue code accumulate. Complexity grows, but control never fully materializes.

At some point, teams realize they are no longer extending existing infrastructure. They are rebuilding a new execution layer alongside it. This is not a failure of engineering effort. It is a mismatch between the problem and the tools being applied.
Agents represent a new workload class—one that requires infrastructure designed specifically for dynamic, delegated, multi-step execution.

An MCP runtime exists to meet that need. It does not replace existing enterprise systems. It complements them by providing a shared execution layer where authorization, execution behavior, and control can be enforced consistently across agents and tools.

Without this layer, teams solve the same problems repeatedly and locally. With it, those concerns move into shared infrastructure, where they can be addressed once and relied on everywhere.

The MCP Runtime, in Practice

As agents move from demos into production, teams quickly discover that reasoning alone is not the hard part. The challenge is operating agents safely inside real enterprise environments, where actions must be authorized correctly, executed reliably, and governed consistently across users and systems.

Arcade was built to solve this problem directly.

Rather than embedding authorization, execution safeguards, and governance logic inside individual agents or MCP servers, Arcade provides a centralized MCP runtime that sits between agents and enterprise systems. Agents reason and decide what to do. Arcade determines whether and how those actions are allowed to execute.

When an agent invokes a tool through Arcade, execution is mediated by the runtime. Required permissions are evaluated at execution time using existing enterprise identity systems. If authorization is missing or incomplete, Arcade handles the authorization flow before allowing execution to proceed. Tokens are managed centrally, permissions are revalidated on every action, and revocation is respected automatically.

Execution behavior is enforced consistently. Arcade validates inputs against explicit schemas, classifies failures deterministically, and manages retries, idempotency, and partial execution as first-class concerns. Agents are insulated from the mechanics required to safely operate production systems, allowing them to focus on reasoning rather than defensive execution logic.

Arcade also serves as the control plane for agent activity. Every action taken by an agent is attributable to a user, observable end-to-end, and auditable by default. Rather than scattering governance across prompts, wrappers, and ad hoc monitoring systems, Arcade centralizes visibility and enforcement in a single execution layer.

This architecture changes how teams deploy agents.

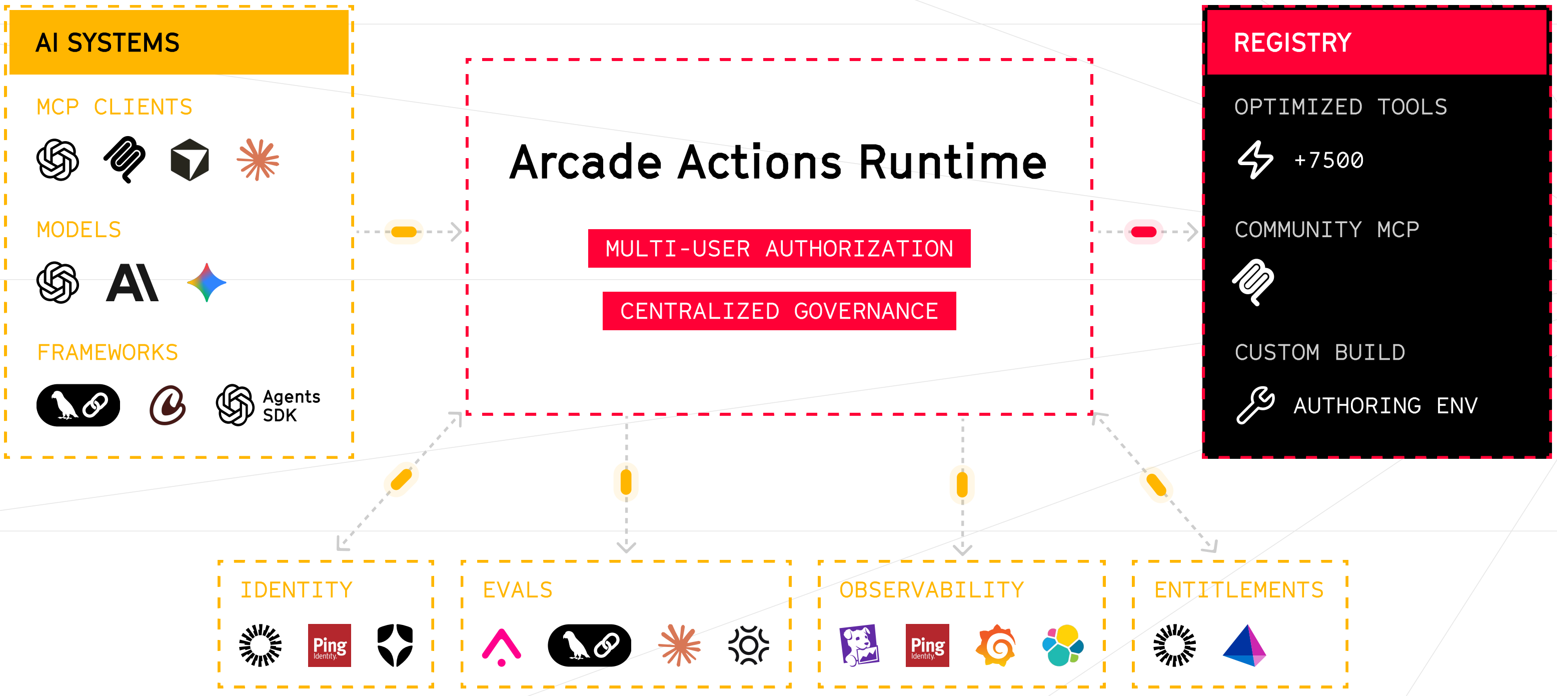
Instead of treating each agent as a bespoke integration, teams can reuse tools safely across workflows, apply consistent policies across environments, and move from pilot to production without rebuilding infrastructure each time. Security reviews accelerate because responsibility boundaries are explicit. Platform teams regain confidence because behavior is predictable and controlled. AI/ML teams spend less time hardening execution paths and more time improving agent capability.

Arcade does not replace existing enterprise systems. It makes them usable by agents at scale. By providing a production-grade MCP runtime that handles authorization, execution, and governance centrally, Arcade allows agents to operate as first-class participants in enterprise workflows—without undermining security, reliability, or trust.

Arcade's Agent-Optimized Tools

Arcade provides the largest catalog of agent-optimized tools available today. These thousands of tools let teams quickly build AI agents that connect to enterprise systems and take actions with better reliability and lower costs, without spending months building and maintaining custom integrations for every system.

For teams that need custom tools, the MCP Framework makes it straightforward to build high-quality tools that automatically integrate with the runtime.



Closing the Production Gap

Organizations experimenting with agents are already discovering that intelligence is not the limiting factor. Execution is. The question is not whether agents can reason through work, but whether enterprise systems are prepared to let software act—reliably, securely, and at scale.

That gap does not close on its own.
As agent usage grows, teams either invest in shared execution infrastructure or accumulate increasingly fragile workarounds. Authorization logic spreads. Reliability becomes situational. Governance turns into a patchwork of policies and manual checks. Progress slows, not because the opportunity disappears, but because the foundation can no longer support it.

This is the inflection point.

Faced with these constraints, many teams initially try to build this layer themselves — stitching together credentials, execution logic, retries, and guardrails inside their agent framework or application code.
In practice, this quickly becomes a drain on senior engineering time.

Maintaining a secure, reliable execution layer requires constant attention: evolving permission models, rotating credentials, handling partial failures, enforcing idempotency, and adapting as new tools and workflows are introduced. These are foundational infrastructure problems, not differentiating product work.

Teams that scale successfully recognize this early. Instead of having their most experienced engineers rebuild execution infrastructure, they direct that effort toward the workflows and agent behaviors that actually create value. A runtime designed to absorb execution complexity by default is what makes that shift possible.

By providing a production-grade MCP runtime, Arcade allows organizations to move beyond experimentation and operate agents as first-class participants in enterprise workflows. It is designed for the moment when agent projects stop being impressive demos and start needing to work—every day, across users, with real consequences.

The transition from demos to production is not a question of ambition.
It is a question of standardizing the right architectural pieces.

If your organization is serious about moving agents into production, the next step is to make the execution layer explicit.

Arcade's Forward-Deployed Engineers work directly with AI/ML, platform, and security teams to review existing agent architectures, identify where execution breaks down, and map a path to production-grade deployment.

Start an architecture review with an Arcade FDE

Arcade.dev