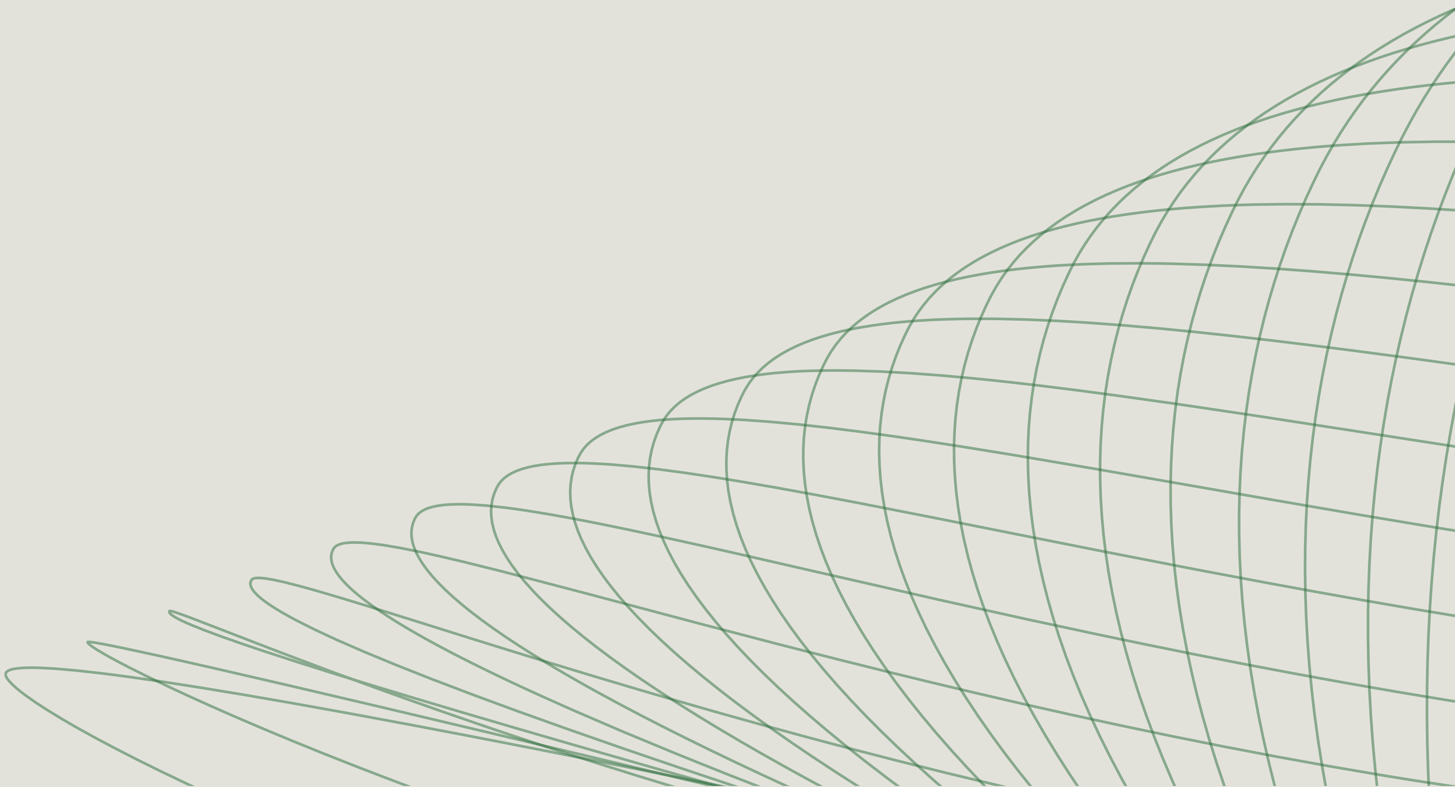
STACKLOK

The MCP Platform Buyer's Guide for AI Enablement Teams



Contents

Over the past few months, we’ve worked closely with more than forty AI enablement teams at large enterprises. This buyer’s guide is based on the critical capabilities they required from an MCP platform; we’ve summarized their priorities and questions to help you leave no stone unturned as you identify the right path forward for your enterprise.

Unlocking Return on Agent Investments with Model Context Protocol

Your organization has invested in AI agents. Significantly. There is executive visibility, board-level pressure, and a clear expectation that these investments will translate into measurable productivity gains. You have the use cases. You have developer enthusiasm.

And yet utilization is lower than it should be. Agents that look promising in demos produce inconsistent results in production. Developers who try them once and hit a wall do not always come back. Security reviews slow rollouts considerably. And the context window problems of too many tools, conflicting descriptions, and bloated payloads quietly degrade the quality of every interaction.

The gap between "we have AI agents" and "our AI agents are generating real return" is often not a model problem. It is an infrastructure problem, and the Model Context Protocol is a potential solution.

MCP is the layer that connects your agents to the systems they need: GitHub, Jira, Databricks, Salesforce, internal APIs, knowledge bases, and more. When that layer is well-governed, properly authenticated, and optimized for agent performance, your agents work reliably and developers adopt them broadly. When it is improvised (i.e. API keys in config files, ungoverned community servers, no audit trail, security teams blocking rollouts) your agents underperform and the business gets harder to justify.

This guide is for the AI leader who is responsible for making the agents work and making the case that they are working.

What Is Actually Holding Your Agents Back

AI enablement teams consistently encounter the same set of obstacles when trying to scale AI agent adoption. It’s critical to understand how MCP can relieve some of the pain.

Context window bloat is quietly destroying agent performance

When an AI agent is presented with 50 tools it does not need, the model's ability to select the right tool degrades and it burns tokens on irrelevant context. A Director of AI we’ve been working with described a particular, commercial MCP server as "borderline disrespectful" in token consumption and chose to absorb the cost rather than fix the underlying problem because the fix was too complex. That is a failure mode that scales badly.

Security teams are blocking rollout, not enabling it

AI enablement leaders frequently describe a dynamic where they have organizational alignment and developer enthusiasm but cannot move because they are stuck in security review. Security teams are not wrong to be concerned; MCP gives AI agents the ability to read and write across internal systems in ways that require real governance answers.

Agents lack the right context at the right time

The best agents are connected to the right tools, with the right descriptions, tuned for the specific task at hand. Most early MCP implementations expose a generic set of tools to every agent, regardless of what the agent is doing. The result is inconsistent performance that is hard to diagnose and harder to fix without rebuilding the integration from scratch.

Adoption is lower than the headline numbers suggest

Developers who experience friction with AI tooling do not escalate, they quietly stop using it. While usage concentrates in a small cohort of enthusiastic early adopters, the broader developer population stays on the sideline. Without observability into who is using what, and without a governed path that makes adoption easy, you cannot diagnose why utilization is low or demonstrate that it is improving.

Reusable patterns do not exist yet

Every team is reinventing the same integrations. The same Jira connection, the same Confluence lookup, the same internal API wrapper rebuilt five times across five teams, with five different quality levels and five different security postures. There is no mechanism to define a pattern once and make it available to everyone. That is not a developer problem. That is a platform problem.

The Five Jobs AI Enablement Teams Need to Get Done

Before evaluating any platform, it is useful to be explicit about what success looks like. AI enablement teams are accountable for five distinct outcomes, and the right MCP platform needs to advance all five.

1. **Prove ROI.** You need agents that perform reliably enough, and adoption broad enough, to construct a credible return-on-investment narrative. That requires usage telemetry, performance instrumentation, and the ability to attribute outcomes to specific agent deployments. You cannot prove ROI for something you cannot measure.

2. **Pass the security review.** Every agent rollout that matters will eventually encounter a security review. You need a platform that gives you credible answers to questions about identity, access control, audit trails, and data handling.

3. **Scale adoption beyond the early adopters.** Broad adoption requires a path that is simple enough for a developer who has never heard of MCP to get value within an hour, without needing to understand OAuth flows or container runtimes.

4. **Improve agent performance over time.** Agents that work in demo conditions but degrade in production are liabilities. You need the infrastructure to tune tool selection, optimize context usage, and iterate on agent performance based on real usage data.

5. **Build reusable organizational capability.** The skills, workflows, and integration patterns your best engineers build should be available to every agent across the organization. An MCP platform should be the mechanism for distributing that institutional knowledge.

The Capability Checklist: What to Require From Any MCP Platform

Section 1: Agent Performance and Context Optimization

This is where AI enablement teams should start because it is where the ROI gap is most directly addressed. Better-performing agents drive adoption. Adoption drives measurable return.

Capability	Why it Matters	What to Require
Context optimizer / smart tool selection	When agents are presented with too many tools, model performance degrades and token costs rise. Context optimization dynamically surfaces the right tools for the task at hand, reducing context noise and improving selection accuracy.	Require semantic-search-based tool selection that reduces the effective tool set presented to the model at inference time. Ask for specific data on token reduction and tool call accuracy improvement. Benchmark claims against your own use cases.
Virtual MCP server composition	Rather than exposing a raw MCP server with every available tool, the platform should allow you to define task-specific views optimized for a particular agent or workflow.	Require the ability to define virtual servers that filter, rename, and tune underlying tools without modifying the underlying server implementation. This is a meaningful differentiator; most lightweight platforms do not support it.
Tool description override and curation	Auto-generated tool descriptions from OpenAPI specs are often thin, ambiguous, or poorly suited to the way a model interprets them. The ability to override descriptions without touching server code is a direct lever on agent performance.	Require the ability to override tool names and descriptions at the platform layer, independent of the underlying server implementation. Verify this is available to platform admins without requiring server code changes.

Composite tool and workflow definitions	Many high-value agent tasks require sequencing multiple tool calls in a defined order. Leaving this to the LLM's judgment produces inconsistent results.	Require declarative composite tool support that chains multiple underlying tool calls into a single, repeatable workflow with structured output and defined transaction semantics.
Reusable skills library	Your organization's coding conventions, incident response procedures, architecture patterns, and domain knowledge should be packaged as skills that any agent can invoke.	Require a first-class skills concept that allows packaged, reusable agent capabilities to be published to the registry and consumed across teams without custom development per team.

Section 2: Developer Adoption and Time-to-Value

The platforms that drive broad adoption are the ones that make the governed path the easy path.

Capability	Why it Matters	What to Require
Zero-configuration developer onboarding	If a developer needs to understand OAuth flows, edit YAML configuration files, or install and configure Docker before they can use an MCP server, most of them will not get there.	Require a desktop application or one-click install path that allows a developer to discover and connect to approved MCP servers without prerequisite technical knowledge. Verify this works without the platform team needing to pre-configure each developer's machine.
Automatic IDE client configuration	Requiring developers to manually configure VS Code, Cursor, or Claude Code for each MCP server is adoption-killing friction.	Require automatic client configuration: when a developer gains registry access to a server, their IDE should be configured automatically. Verify this works across all clients your organization has approved.
Self-service discovery portal	Developers should be able to find, evaluate, and connect to approved MCP servers without filing a ticket or waiting for an admin.	Require a searchable, human-readable catalog that developers can access without admin assistance. The portal should show what servers are available, what tools they expose, and what teams have access.
Open source evaluation path	Developers who can evaluate a platform without a procurement conversation become internal advocates.	Require an Apache 2.0 (or equivalent permissive) licensed core that developers can download, run, and evaluate independently.

Section 3: Security and Governance (Unlocking Adoption, Not Blocking It)

The goal of governance is not to restrict what developers can do, it is to create the conditions under which security teams can say yes to broader adoption.

Capability	Why it Matters	What to Require
Enterprise IdP integration (Okta, Entra ID, Google)	Developers should authenticate through your existing identity provider. Separate credentials for MCP access create security gaps and adoption friction simultaneously.	Require native OIDC/OAuth SSO with your existing IdP. Developers should authenticate once and receive scoped access to all MCP servers their role permits, with no manual credential management.
Per-user identity passthrough to downstream systems	When an agent calls Databricks or Salesforce on behalf of a developer, those systems should log that developer's identity, not a shared service account. This is the audit trail that security and compliance teams require.	Require token exchange flows (OAuth on-behalf-of / RFC 8693) that preserve and propagate end-user identity to downstream systems.
Client-side enforcement hooks	Governance that only exists at the server layer can be bypassed by a developer who configures their own MCP server locally. Enforcement at the IDE client layer closes that gap.	Require hooks-based enforcement for your approved IDE clients. Ask specifically which clients are supported today and get explicit confirmation on any gaps.
Supply chain security for community MCP servers	Developers will want to use community-sourced MCP servers from GitHub and the public ecosystem. Some of those servers are fine. Some are not.	Require CVE scanning and SBOM generation for servers in the registry. Ask whether the vendor maintains hardened versions of high-usage community servers (GitHub, Atlassian, Databricks, etc.) with ongoing security patch commitments.
Shadow MCP server discovery	Before you can govern what is running, you need to know what is running. Most organizations have more ungoverned MCP activity than they realize.	Ask whether the platform offers any form of MCP estate scanning or shadow server detection. The ability to surface ungoverned deployments is the first step toward bringing them into compliance.

Section 4: Observability and ROI Measurement

You cannot prove value you cannot measure. Observability is the foundation of your ROI case.

Capability	Why it Matters	What to Require
Per-developer and per-team usage telemetry	Board-level AI investment gets justified by measurable adoption and productivity gains. Neither is demonstrable without granular usage data.	Require telemetry that attributes tool invocations to individual users and teams, not just aggregate server-level metrics. You should be able to show adoption trends over time, by team, by tool, and by use case.
Tool call accuracy and performance instrumentation	Knowing that a tool was called tells you about activity. Knowing whether it succeeded, how long it took, and how often the model selected it correctly tells you about performance.	Require per-tool success/failure rates, latency metrics, and tool selection frequency data. This is the data you need to identify which agents are performing well and which need optimization.
OpenTelemetry-native export	You already have an observability stack. Your MCP telemetry should flow into it, not create a parallel system you need to maintain separately.	Require OTLP-compatible traces, metrics, and logs following official OpenTelemetry MCP semantic conventions. Verify export to your existing backend is supported out of the box.
Audit trail for compliance and security review	A defensible audit trail turns a security review from a blocker into a checkbox.	Require complete, tamper-evident audit logs for all tool invocations, covering identity, arguments, outcomes, and timestamps. Verify logs can be exported to your SIEM and retained for the periods your compliance framework requires.

Section 5: Deployment and Integration Flexibility

AI enablement teams do not control the infrastructure, but will be held responsible for adoption outcomes that depend on deployment decisions. Make sure the platform supports your organization's real environment.

Capability	Why it Matters	What to Require
Kubernetes-native deployment	Production MCP infrastructure needs to run in the same environment as your other production workloads with the same operational maturity, scaling behavior, and observability integration.	Require a native Kubernetes operator with CRD-based lifecycle management. Verify support for your specific Kubernetes distribution and cloud provider.

Self-hosted, on-premises deployment	AI agents that touch internal systems and potentially sensitive data cannot always route through external SaaS infrastructure. Many regulated environments require full on-premises control.	Confirm the platform is fully self-hostable with no required SaaS components; this is a deal-breaker for most large enterprises.
Local development option	Developers experimenting with agent patterns need to be able to run MCP servers locally before a centralized deployment exists.	Require that local execution modes enforce registry policies and emit telemetry to your central observability stack. The deployment model should not determine the governance model.
Integration with existing policy engines	Your organization may already have an OPA/Rego-based policy framework, a Cedar deployment, or an existing authorization service. The MCP platform should augment those investments, not replace them.	Require explicit confirmation of policy engine compatibility. Verify that your existing policy framework can be plugged in, and that migration is supported if you eventually want to consolidate.
Connector portfolio depth	The value of an MCP platform scales with the breadth of integrations available. Building every connector from scratch defeats the purpose.	Evaluate the vendor's pre-built connector library (both breadth and quality). Ask which connectors are maintained by the vendor with ongoing security patch commitments versus community-sourced.

Section 6: Engagement Model and Time-to-Value

For AI enablement teams operating under pressure to show results, the engagement model is as important as the product. It's critical to partner with someone who is going to implement the solution with you.

Capability	Why it Matters	What to Require
Forward deployed engineering support	The most common failure mode for MCP platform deployments is not technical, it is that no one has the bandwidth to drive implementation alongside their day job. Embedded vendor engineering support changes that dynamic.	Ask specifically about Forward Deployed Engineer (FDE) availability. What does the engagement look like? Is there a defined sprint structure? Will engineers write code alongside your team, or just advise?
Defined POC structure with success criteria	A 30 or 60-day paid POC with defined success criteria is a more useful evaluation structure than an open-ended free trial. It creates shared accountability and produces a clear decision point.	Ask the vendor to propose a POC structure with specific deliverables, deployment milestones, and success metrics. Use those criteria to hold both parties accountable during the evaluation.

Open source core with commercial enterprise tier	Open source lets you build internal confidence and developer familiarity before a procurement conversation. It also de-risks the vendor relationship — if the core is Apache 2.0, you are not entirely dependent on the vendor's commercial trajectory.	Require a permissively licensed open source core that is genuinely production-capable. Commercial tiers should add hardening, support SLAs, and enterprise convenience features — not gatekeep foundational governance capabilities.

Putting It Together: Prioritizing Your Evaluation

The capabilities in this guide are not equally important for every organization. Use the following framework to sequence your evaluation based on your current situation.

If your primary challenge is agent underperformance

Start with context optimization, virtual MCP composition, and tool description curation. These capabilities have the most direct impact on the accuracy and reliability of agent outputs.

If your primary challenge is security review blockage

Start with identity passthrough, IdP integration, client-side enforcement hooks, and audit log completeness. These are the capabilities that allow you to clear the security review.

If your primary challenge is low developer adoption

Start with the zero-configuration onboarding path, the self-service portal, and the open source evaluation path. Developers who can get value quickly become the internal advocates who drive broader rollout.

If your primary challenge is scaling from pilots to production

Start with the Kubernetes operator, the reusable skills library, and composite workflow support. The patterns that work for five agents need to hold for fifty.

What the MCP Platform Landscape Actually Looks Like

The MCP platform market is early, and the vendor landscape reflects that. Several solutions have entered the space, and it is worth being clear-eyed about where different categories of vendors focus and where they fall short against enterprise AI enablement requirements.

Developer-tooling platforms that have added MCP support often treat governance as a secondary concern. They are optimized for individual developer productivity, not organizational deployment at scale. Their observability stories are thin, their identity integrations are limited, and their policy models are designed for convenience rather than enterprise security posture.

Gateway-focused tools can proxy MCP traffic and provide basic routing, but they typically lack the runtime management, context optimization, and supply chain security that production AI enablement requires. They are useful for proof-of-concept work. They are not adequate for governing an MCP estate that needs to demonstrate ROI at scale.

Cloud provider offerings address deployment but lock you into a single-cloud architecture, lack the vendor-neutral flexibility that heterogeneous enterprise environments require, and typically do not address the developer adoption and agent performance challenges that AI enablement teams actually face.

The right platform will be purpose-built to fully leverage MCP, with the observability, identity, governance, and performance optimization capabilities that turn AI agent investments into measurable business outcomes.

The questions in this guide will help you tell the difference.