Taming the Agentic Supply Chain

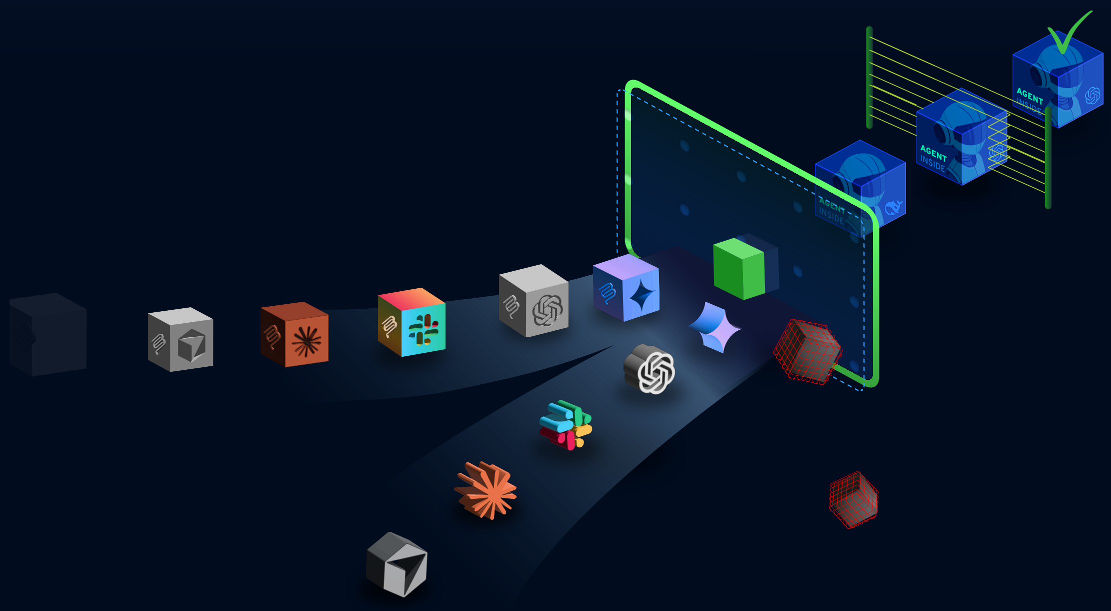
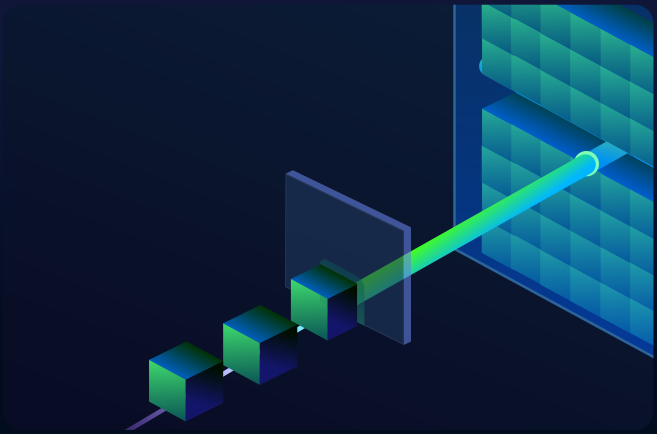


Table of Contents

01 — Introduction:

The Brain Gets Hands



Why AI is No Longer Just Watching

For the last two years, generative AI has been a "brain in a jar." It could think, reason, and write upon request, but it remained passive—it couldn't actually touch anything. That isolation is ending.

The shift from passive chat to active execution is driven by the Model Context Protocol (MCP), an open-source protocol that gives the AI "hands." With MCP, an AI model gains the ability to interact with the world; once this connectivity is established, developers can build Agents that perform autonomous actions. These agents can then reach into your file system, query production databases, and push code to GitHub—transforming the model from a passive advisor into an active participant in your infrastructure.

However, giving AI hands means your AI can now break things. By adopting MCP, you are essentially granting autonomous agents valid credentials to your internal infrastructure.
In this ebook, we'll explore the hidden risks of this new agentic supply chain and outline a framework for governing these AI "hands" with the same rigor, security, and visibility you apply to your traditional software.

02 — The Landscape:

Understanding MCP Servers

Before addressing the security of the agentic supply chain, we must understand the three primary ways these "hands" are attached to the model. Since its release by Anthropic in November 2024, the Model Context Protocol has rapidly become the "USB-C for AI," with adoption from every major provider including OpenAI, Google, Microsoft, and Amazon.

In late 2025, the protocol's long-term independence was secured when it was donated to the Agentic AI Foundation (AAIF) under the Linux Foundation. This shift from a vendor-led project to a neutral industry standard has fueled a massive ecosystem, which now includes over 36,000 public MCP servers across the following three categories:

- **These run directly on a developer's machine through AI-native IDEs (such as Claude Code, Cursor, GitHub CoPilot). A common example is a filesystem server that allows an agent to read local logs or analyze code in a specific directory. Because these run on localhost, they often operate entirely under the radar of corporate security.**

- These run directly on a developer’s machine through AI-native IDEs (such as Claude Code, Cursor, GitHub CoPilot). A common example is a filesystem server that allows an agent to read local logs or analyze code in a specific directory. Because these run on localhost, they often operate entirely under the radar of corporate security.
- Remote/Cloud Servers: These are hosted services that connect to third-party APIs. For instance, a Slack MCP server allows an agent to read messages or post updates to channels. These require managing API keys and external permissions.
- Custom-Made Servers: These are the most powerful and the most sensitive. They are internal tools built by your own engineering teams to expose proprietary business logic, private databases, or internal microservices to the AI.

03 — The Reality:

The "Wild West" of Agentic AI

AI agents gain their power through the specific MCP tools they are granted. The more tools an agent can access, the more potential harm it can cause to your infrastructure. At their core, local and custom MCP servers are simply software binaries—no different from the open-source packages or first-party code you already use. To prevent a "Wild West" of unmanaged AI sprawl, these tools must be governed, secured, and managed with the same rigorous practices as any other part of your software supply chain.

Management: The Sprawl

Developers are downloading MCP servers from unvetted community lists and running them on localhost without formal review and approval. They’re even building custom-made MCP servers to easily access internal proprietary systems with AI.

- **The Pain:** You have zero visibility. You can’t update a broken server, deprecate a risky one, or share a useful tool across the team. Developers keep reinventing the wheel and MCP servers don’t comply with organizational policies.
- **The Risk:** Operational debt. If a "Postgres MCP Server" version breaks, for example, dozens of developers stop working with no way to roll back centrally.

Governance: Tool-Level Complexity

An MCP Server is a package that contains multiple MCP Tools. For instance, a Database Server might contain read_table, write_table, and delete_table.

- **The Pain:** Governance isn't binary. You may want a developer to use an agent to read data, but you do not want their agent to have the delete tool. Currently, most systems only allow you to block the entire server upon execution or nothing at all.
- **The Risk:** Catastrophic accidents. A hallucinating agent paired with a delete_table tool can wipe a production database. This isn't a hack; it’s a feature used incorrectly.

Security: The Lethal Three

To secure the agentic supply chain, we must look beyond the AI model itself and focus on the execution layer. When an agent is given "hands" via MCP, it introduces three distinct categories of risk that traditional security perimeters are not yet equipped to handle.

A. The Supply Chain Risk: "Side-Loading" Unverified Binaries

Developers are under immense pressure to ship AI-enhanced features. To move fast, they often bypass IT approval processes entirely and are "side-loading" tools by running commands like npx -y @smithery/postgres-mcp or pulling unvetted Docker containers from personal GitHub repos.

- **The Pain:** This creates a "Shadow Supply Chain" where MCP servers run outside your CI/CD pipeline, bypassing existing Software Composition Analysis (SCA) tools. As a result, unverified binaries with no Software Bill of Materials (SBOM) or provenance run on developer laptops, potentially introducing malicious dependencies into your environment.
- **The Risk:** Network Lateral Movement. By allowing unvetted MCP servers, you have effectively introduced a compromised node inside your perimeter. Because the server runs under the developer's local environment, it inherits their full network privileges, providing a perfect jumping-off point for an attacker to move laterally through your internal systems.

B. The Access Risk: Over-Privileged Agents (Zero Trust Violation)

Most MCP servers are designed to prioritize "helpfulness" over security. When a developer connects an MCP server to a core system—like GitHub, AWS, or a production database—that server often inherits the developer's full user permissions by default.

- **The Pain:** There is no "Least Privilege" implementation. If a developer has Read/Write access to 50 sensitive repositories, the Agent now has Read/Write access to those same 50 repositories—even if it only needs to check a single file. There is no easy way to restrict the scope of what the AI tool can touch.
- **The Risk:** Blast Radius Expansion. In an agentic workflow, the "user" is no longer just a human; it's a model that can hallucinate or be manipulated via prompt injection. If an agent is compromised or makes a logical error, it isn't contained to a sandbox. It has the "keys to the kingdom," turning a minor hallucination into a major security incident.

C. The Operational Risk: Destructive Execution & Data Leakage

An Agent isn't a passive tool; it's an active participant that can read data, synthesize it, and post content. It operates with the speed of a machine but carries the authority of a human user.

- **The Pain:** Agents lack judgment. They follow the logic of the prompt and the capabilities of the tool. For example, a "Summarizer Agent" might read a confidential strategy document containing Personally Identifiable Information (PII) or secrets and then, following instructions to "provide an update," accidentally paste that sensitive summary into a public Slack channel or a Jira ticket visible to external contractors.
- **The Risk:** Automated Data Exfiltration. The danger isn't just malicious sabotage or deletion; it's the accidental, high-speed broadcasting of sensitive IP. Once an agent has the "hands" to move data between tools, the risk of data leakage scales from human-speed errors to machine-speed catastrophes.

04 — The Solution:

Governing AI Like Software

To tame the agentic supply chain, we must move beyond reactive security and total bans. Many organizations currently block all MCP servers and their public registries, yet developers inevitably find ways to bypass these restrictions to keep pace with modern development standards.
You need a trusted way to adopt MCP servers rather than ignoring the reality of their use. If developers will use these tools regardless of policy, it's time to pave an enterprise-grade path for doing so securely.
To move from this "Wild West" to a governed environment, your architectural approach must meet four key requirements.

Requirement 1:
Support for All MCP Types

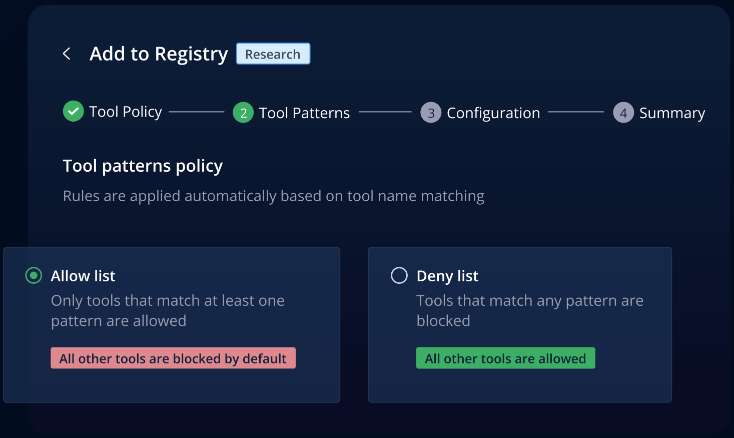
Governance is only as strong as its weakest link. Your solution can't just manage local CLI or IDE tools; it must provide a unified management layer for the entire spectrum: Local servers (filesystem access), Remote servers (external APIs), and Custom-built servers (proprietary internal logic).

- **Why it matters:** If you only govern local tools, developers will simply pivot to remote or custom servers to bypass restrictions. Comprehensive visibility across all three types is the only way to eliminate MCP servers that bypass your security perimeter.

Requirement 2:
Proactive Policy Enforcement (The "Digital Border")

Detection after the fact isn't enough. You must block unvetted or high-risk MCP servers before they enter your environment. This requires a "gatekeeper" that intercepts every download request and evaluates it against organizational policy, rejecting servers with restrictive licenses (like GPL), unverified publishers, or poor security reputations.

- **Why it matters:** Once a malicious or non-compliant binary lands on a developer's laptop, it's already too late. Security must start at the point of ingestion into the supply chain, creating a digital border that keeps the organization safe by default.


Add to Registry Research

✓ Tool Policy — 2 Tool Patterns — 3 Configuration — 4 Summary

Tool patterns policy

Rules are applied automatically based on tool name matching

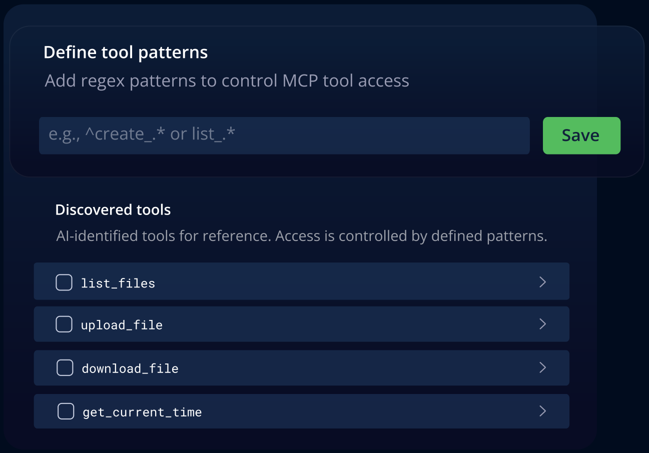
● Allow list
Only tools that match at least one pattern are allowed
All other tools are blocked by default

○ Deny list
Tools that match any pattern are blocked
All other tools are allowed

Requirement 3:
Granular "Tool-Level" Governance

Traditional security is often binary: you either allow a file or you block it. However, MCP servers are "containers" for multiple capabilities. A single database server might contain a safe read_data tool and a catastrophic delete_data tool. You need a system that understands these internal capabilities and allows you to toggle them individually.

- **Why it matters:** Security shouldn't slow you down. By enabling the safe parts of a tool while disabling the dangerous ones, you allow developers to stay fast without giving the AI the "hands" to cause a major incident.


Define tool patterns

Add regex patterns to control MCP tool access

e.g., ^create_.* or list_.*

Save

Discovered tools

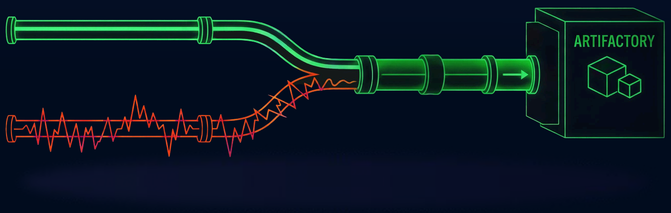
AI-identified tools for reference. Access is controlled by defined patterns.

- ☐ `list_files`
- ☐ `upload_file`
- ☐ `download_file`
- ☐ `get_current_time`

Requirement 4:
Contextual Unity (The "One Platform" Rule)

AI does not operate in a vacuum. An Agent uses an MCP Server (binaries), runs on a specific language (Python/Node.js), and is developed within a broader ecosystem of Docker images and NPM packages. To be effective, your MCP registry must live alongside these other artifacts in a single system of record.

- **Why it matters:** Managing AI tools separately from the rest of your software creates dangerous security gaps. When you unify your MCP servers with your existing software supply chain platform, you ensure that every part of the agentic workflow is governed by the same enterprise-grade security you already know and trust.



05 — The JFrog Solution:

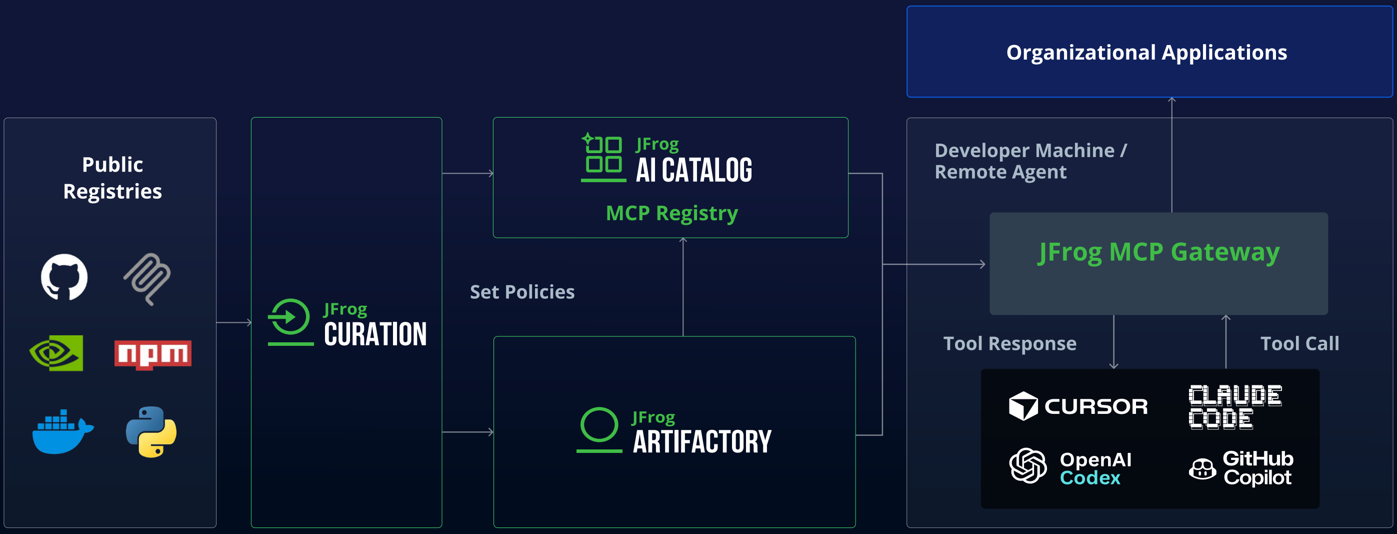
JFrog MCP Registry

The JFrog MCP Registry establishes the single source of truth for approved MCP servers and their respective tools, integrating directly into the JFrog AI Catalog to provide a unified, secure foundation for all AI and software assets.

- **Software Practices for AI:** We treat MCP servers as first-class artifacts, just like Docker images or Maven packages. This ensures the same enterprise-grade reliability—including high availability, replication, and rich metadata—for your AI tools as your production software. One place to look, one place to manage.

- **Permissions That Understand Reality:** JFrog allows you to define who can use specific capabilities through granular, tool-level control. You can grant senior architects access to the delete_table tool while restricting junior developers to read_only capabilities—even when they are using the exact same MCP server. This ensures you are governing the action, not just the file.

  - **The Power of the Platform:** The JFrog AI Catalog leverages the trusted security engines already protecting your environment, allowing you to extend your existing security shield to AI agents without implementing new, siloed tools.
  - **JFrog Curation:** Stops the "Shadow Supply Chain" at the gate by blocking malicious or non-compliant MCP servers before they can be downloaded.
  - **JFrog Artifactory:** Provides deep visibility by scanning MCP server binaries for vulnerabilities and license compliance issues post-download.



The Unified Advantage: Security at Scale

The JFrog MCP Registry closes critical security and governance gaps by ensuring every MCP server passes through the same rigorous, automated policies used for your traditional software artifacts. This proactive approach blocks unauthorized servers from ever entering your organization, allowing you to adopt agentic AI at scale without introducing new supply chain risks.

As the only MCP registry integrated into a trusted software supply chain platform, JFrog provides a unique differentiator: enterprise-grade control over all your binaries —software packages, AI models, and MCP servers—in one unified location.

Speed vs. Security: The Agentic Balance

The choice isn't whether to adopt Agentic AI or be secure; it's whether to use managed MCP servers or remain in the Wild West. To secure your organization, integrate your AI supply chain with your existing software supply chain.

YOUR AGENTIC READINESS CHECKLIST:

- [ ] Do you have a central inventory for all local and remote MCP servers?
- [ ] Can you block a specific Tool (capability) inside a server?
- [ ] Are your MCP servers scanned by the same engine protecting your production code?
- [ ] Is your AI supply chain unified with your SDLC platform?

Secure the hands of your AI. Start with the JFrog MCP Registry.