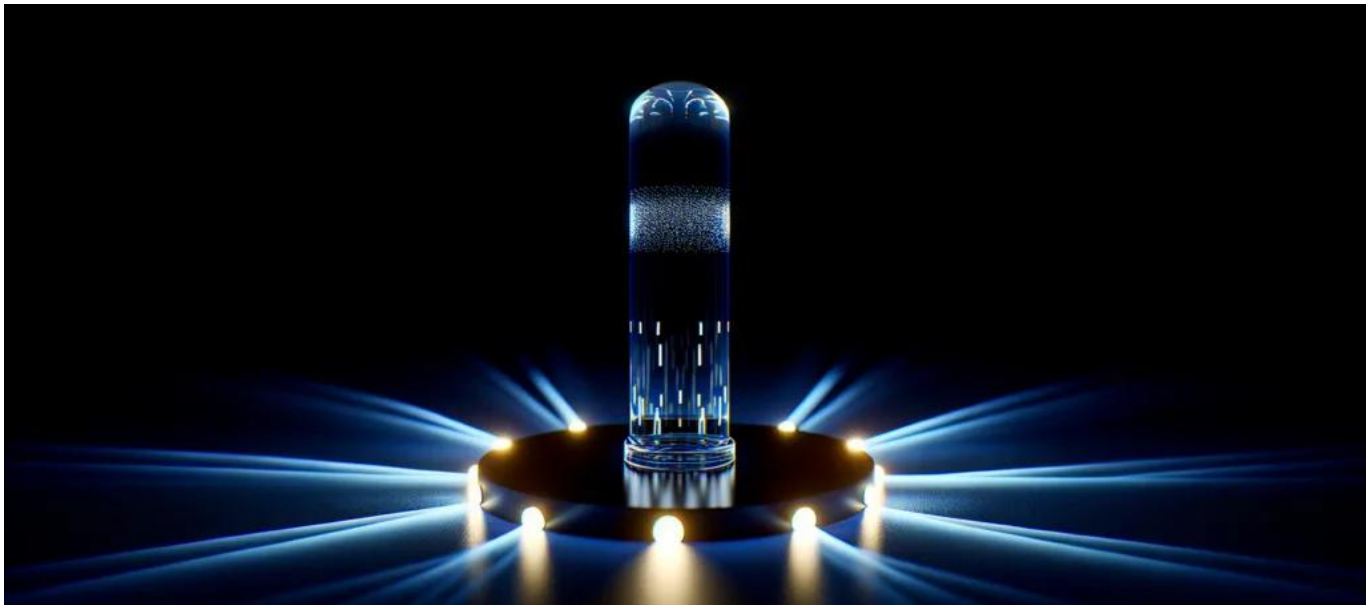


Anthropic’s Model Context Protocol Hits 97 Million Installs on March 25—MCP Transitions from Experimental to Foundation Layer for Agentic AI

Six months ago, MCP was a footnote in Anthropic’s documentation. Today it’s the plumbing underneath 97 million AI agent deployments, and most engineers still don’t know how it works.

The Numbers That Matter

On March 25, 2026, Anthropic’s Model Context Protocol crossed 97 million installs—a figure that represents the fastest adoption curve for any AI infrastructure standard in history. To put that in perspective: Kubernetes took nearly four years to reach comparable deployment density across enterprise environments.

The install count alone doesn't capture the shift. What matters is that every major AI provider now ships MCP-compatible tooling as a default, not an option. OpenAI, Google DeepMind, Cohere, Mistral—all of them integrated MCP support into their agent frameworks by mid-March. When your competitors all adopt the same standard within a 90-day window, you're not watching adoption. You're watching capitulation.

MCP isn't a model. It's not a framework. It's a protocol specification for how AI agents communicate with external tools and data sources. Think of it as the HTTP of agentic AI: boring, invisible, and utterly foundational.

What MCP Actually Does

Before MCP, connecting an AI agent to a database required custom integration code. Connecting it to a second database required different custom integration code. Every tool, every API, every data source demanded its own bespoke connector. Enterprise deployments routinely maintained hundreds of these integrations, each one a potential failure point.

MCP standardizes the handshake. An agent speaks MCP to a tool server. The tool server speaks MCP back. The agent discovers what capabilities the tool offers, what parameters it needs, and what responses to expect—all through a single protocol specification.

The architecture breaks down into three components:

Hosts are the AI applications—your Claude desktop app, your custom agent framework, your enterprise copilot. They initiate connections and maintain the overall session state.

Clients handle the actual protocol communication. One host can spawn multiple clients, each maintaining a connection to a different tool server.

Servers expose capabilities. A GitHub MCP server might offer repository search, file reading, and pull request creation. A Postgres MCP server offers query execution and schema inspection. The server advertises what it can do; the client decides what to use.

This separation matters for security. The server controls what capabilities get exposed. The client controls what requests get made. The host controls what the user actually

authorized. Three layers of permission boundaries instead of one monolithic integration.

Why March 2026 Was the Inflection Point

The timing wasn't accidental. Q1 2026 marked the quarter when Fortune 500 companies moved agentic AI from pilot programs to production deployment, according to enterprise AI trend analysis. Pilots can tolerate custom integrations. Production cannot.

When Block deployed MCP across their financial data infrastructure, they eliminated 340 custom connectors. Apollo integrated MCP into their go-to-market platform and cut integration maintenance overhead by 60%. Replit built their entire AI development environment on MCP primitives. These weren't experiments. They were architectural commitments.

The enterprise pressure created a coordination problem. No single vendor wanted to adopt a competitor's standard. But every vendor needed some standard or face the integration chaos their customers refused to accept. MCP offered a path: open specification, permissive licensing, and crucially, development by a company that wasn't trying to own the agent runtime layer.

Anthropic's strategic positioning made adoption palatable. They're a model company, not a platform company. MCP doesn't favor Claude over GPT-4 or Gemini. It's genuinely model-agnostic, which made it safe for OpenAI and Google to adopt without ceding competitive ground.

The Technical Architecture Under the Hood

MCP uses JSON-RPC 2.0 as its wire format, running over stdio for local connections or HTTP with Server-Sent Events for remote deployments. The choice is deliberately conservative. JSON-RPC is a 15-year-old specification that every language has mature implementations for. No new serialization formats to learn, no exotic transport protocols to debug.

The capability negotiation happens at connection time. When a client connects to a server, it receives a capability manifest describing:

- **Tools:** Executable functions the agent can invoke. Each tool has a name, description, and JSON schema defining its input parameters.
- **Resources:** Data sources the agent can read. Files, database contents, API responses—anything that can be represented as text or structured data.
- **Prompts:** Pre-defined instruction templates the server recommends for specific tasks.

The manifest format means agents can reason about tool capabilities before invoking them. An agent doesn't need hardcoded knowledge of what a Slack server can do. It reads the manifest, sees a "send_message" tool with parameters for channel and content, and understands the interface.

This dynamic discovery is what makes the 97 million install number meaningful. Each install doesn't represent a static integration. It represents a connection point that can adapt to new capabilities without code changes.

Anthropic's own infrastructure improvements accelerated adoption. Claude's tool use error rates dropped 40% in March 2026 through enhanced computer use capabilities. Better tool calling means more reliable MCP interactions. More reliable interactions mean more enterprise confidence. More confidence means more installs. The flywheel spins.

What Most Coverage Gets Wrong

The narrative around MCP has centered on convenience: fewer integrations to maintain, faster agent deployment, simplified architecture. That framing misses the strategic shift.

MCP doesn't just reduce integration work. It changes who controls the integration layer.

Before MCP, platform vendors owned the connector ecosystem. Salesforce controlled which AI agents could access Salesforce data and how. AWS controlled which agents could invoke Lambda functions. Each platform was a walled garden with proprietary access patterns.

MCP moves control to the tool server operator. If you run an MCP server for your internal APIs, any MCP-compatible agent can connect. The agent vendor doesn't need a

partnership agreement with your infrastructure provider. They just need to speak the protocol.

This shift hasn't fully registered with the incumbent platform vendors. Most are treating MCP as an additional access pattern alongside their proprietary APIs. That interpretation underestimates how quickly developers gravitate toward universal protocols once they exist. Nobody writes raw TCP socket code for web applications anymore. They use HTTP. In eighteen months, nobody will write custom agent integrations. They'll use MCP.

The second misunderstanding: MCP is "just" a protocol specification. In practical terms, the specification is maybe 20% of the value. The other 80% is the server ecosystem. Anthropic and the community have shipped reference implementations for:

- Filesystem access with configurable permission boundaries
- Git and GitHub operations
- Postgres, SQLite, and other databases
- Slack, Google Drive, and productivity tools
- Kubernetes cluster management
- AWS resource provisioning

These reference servers are production-quality, not toy examples. They implement the security boundaries, error handling, and edge cases that real deployments demand. The ecosystem, not just the spec, drives adoption.

Security Model: Where MCP Gets Serious

The obvious question with any universal agent-to-tool protocol: what stops malicious or compromised agents from wreaking havoc?

MCP's security model operates on explicit capability grants. A server only exposes the tools it chooses to expose. A host only authorizes the connections the user approves. The protocol has no mechanism for an agent to discover or access capabilities beyond what's been granted.

Server implementations typically add additional layers:

Authentication: Most remote MCP servers require OAuth tokens or API keys before accepting connections. The protocol doesn't mandate a specific auth mechanism, leaving that to server implementers who understand their security context.

Audit logging: Every tool invocation passes through the server, creating a natural audit point. Enterprise deployments hook this into their SIEM systems for compliance and anomaly detection.

Rate limiting: Servers can throttle requests per client, preventing runaway agents from overwhelming backend systems.

Sandboxing: The filesystem reference server, for example, accepts a list of allowed directories. Requests outside those paths fail at the server level, regardless of what the agent attempts.

The architecture assumes compromise happens. When (not if) an agent misbehaves, the blast radius is limited to what was explicitly granted. A compromised code assistant can't pivot to production databases if it was never given an MCP connection to that server.

That said, the security model depends entirely on server implementation quality. A poorly written MCP server can expose everything the underlying system can access. The protocol doesn't save you from yourself.

Practical Implementation: What You Should Actually Do

If you're running AI agents in production and haven't adopted MCP yet, here's the priority order:

First, inventory your current integrations. List every tool and data source your agents currently access. Note which use proprietary connectors versus custom code. This audit reveals your migration surface area.

Second, identify pre-built servers. Check the MCP server registry for existing implementations matching your tools. GitHub, Slack, major databases, cloud providers—there's probably already a server. Don't build what you can adopt.

Third, design your authorization model. Decide which agents get access to which servers. Map this to your existing IAM structure where possible. MCP connections should flow through your normal security governance.

Fourth, run a shadow deployment. Stand up MCP servers alongside your existing integrations. Route a percentage of agent traffic through MCP while maintaining the legacy path. Compare reliability, latency, and failure modes.

Fifth, sunset custom connectors. Once MCP shows equivalent or better performance, migrate fully and decommission the maintenance burden.

For teams building new agent systems, the calculus is simpler: start with MCP. The ecosystem has matured enough that custom integrations should require explicit justification.

Code-level, the client implementation in Python looks like:

```
from mcp import ClientSession
from mcp.client.stdio import stdio_client

async with stdio_client(server_path) as (read, write):
    async with ClientSession(read, write) as session:
        await session.initialize()
        tools = await session.list_tools()
        result = await session.call_tool("query_database",
            arguments={"sql": "SELECT * FROM users LIMIT 10"})
```

TypeScript and other language SDKs follow similar patterns. The learning curve is measured in hours, not weeks.

The Vendor Landscape Reshapes

MCP adoption creates clear winners and losers over the next twelve months.

Winners:

Infrastructure companies that ship MCP servers for their platforms gain stickiness. Every team using your MCP server represents an integration that persists across agent vendor switches. Datadog, Snowflake, and MongoDB all shipped official MCP servers in Q1 2026. They understand the game.

Agent framework developers benefit from reduced surface area. LangChain, CrewAI, and similar projects can focus on orchestration logic instead of maintaining connector libraries. The integration layer is no longer their problem.

Enterprise security vendors get a standardized interception point. Every MCP connection becomes an auditable boundary. Companies like Wiz and Snyk are already shipping MCP-aware security scanning.

Losers:

Platform vendors who relied on proprietary APIs as moats face commoditization pressure. When any agent can access your data through MCP, exclusive partnerships with specific AI vendors deliver less value.

Integration Platform as a Service (iPaaS) companies lose a major use case. Zapier, Make, and similar tools built businesses connecting disparate systems. If agents connect directly through MCP, the middleware layer thins significantly.

Companies that bet heavily on custom agent frameworks face rewrite costs. Technical debt in bespoke integration code becomes urgently problematic when the industry standardizes elsewhere.

What Comes Next

The 97 million install milestone marks adoption. The next twelve months determine whether MCP becomes truly foundational or just widely used.

Three developments to watch:

Streaming and real-time data. Current MCP excels at request-response patterns. Agentic workflows increasingly need streaming data—market feeds, log tails, real-time metrics. The

protocol specification includes primitive streaming support, but production implementations lag. Expect major iteration here by Q3 2026.

Multi-agent coordination. MCP defines client-server communication, not agent-to-agent protocols. As agentic systems grow more complex, orchestrating multiple agents with shared tool access requires coordination primitives MCP doesn't currently provide. Anthropic has published early drafts of a companion specification for multi-agent scenarios.

Formal verification. Enterprise security teams want mathematical guarantees about capability boundaries, not just implementation claims. Research groups are working on formally verified MCP server implementations. The first production-certified verified server will command a premium in regulated industries.

The longer-term trajectory points toward MCP as assumed infrastructure. Enterprise architecture diagrams in 2027 won't label MCP connections any more than they label HTTPS connections today. It becomes the default, unremarkable, invisible.

The Underhyped Angle

Everyone focuses on MCP connecting agents to external tools. The more significant application is MCP connecting agents to internal systems that were never designed for AI access.

Every enterprise has legacy databases, internal APIs, and custom systems that predate the AI era. These systems have no reasonable path to native AI integration. They expose SQL interfaces, REST endpoints, or worse.

An MCP server wraps these interfaces without modifying the underlying system. The legacy database remains untouched. The MCP server handles capability discovery, permission enforcement, and response formatting. The agent gets structured, safe access to data that would otherwise require custom integration per AI vendor.

This pattern—MCP as a translation layer for pre-AI infrastructure—represents the bulk of enterprise deployment value. Greenfield AI-native systems get the press coverage.

Brownfield integration pays the bills.

What This Means for Your Architecture

The MCP milestone forces a strategic question every technical leader should answer: what's your protocol layer strategy for agentic AI?

Three options exist:

Full adoption: Standardize on MCP for all agent-to-tool communication. Accept the ecosystem's direction and focus engineering resources on differentiation elsewhere.

Selective adoption: Use MCP where pre-built servers exist and quality is proven. Maintain custom integrations for critical paths where you need maximum control.

Protocol abstraction: Build an internal abstraction layer that could swap MCP for alternatives. Hedge against protocol evolution or fragmentation.

For most organizations, option one or two makes sense. Option three adds architectural complexity that rarely justifies itself. Protocol standards tend to entrench once they reach MCP's adoption level. The bet against MCP is a bet against considerable momentum.

The more productive question isn't whether to adopt MCP. It's how to adopt it without creating new security exposure or abandoning operational visibility you've built into existing integrations.

MCP's 97 million installs don't represent a technology trend—they represent the moment when AI agent infrastructure stopped being optional architecture and started being assumed plumbing.



WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting □