CDATA

Research Report: Executive Summary

We Asked Claude Code to Build an Enterprise MCP Server—Here's What We Found

An evaluation of AI-assisted MCP server development for enterprise data access

July 2026

The question

The Model Context Protocol (MCP) has opened a new category of enterprise data access. AI agents can now connect directly to business systems—querying live records, taking action, and returning answers that reflect the current state of the organization rather than a stale export. At the same time, AI coding tools like Claude Code and GitHub Copilot have made it faster and more cost-effective than ever to build software; for many tasks, the initial output is production-ready without significant rework. For organizations deploying AI agents, that combination is compelling. Every data source an agent needs to reach requires a connector—and if AI coding tools can produce one that holds up, the economics and speed of AI data access change significantly. Whether that's actually possible is less clear.

Can you build a reliable, enterprise-grade, production-ready MCP connector with an AI coding tool? We decided to find out.

We ran nine sessions across two approaches. The first used vanilla Claude Code with minimal prompting, simulating how most developers would start. The second brought in a developer experienced in data connectivity who applied structured workflow and best practices. What we found is documented in full below.

The standard

The first step was to define what a connector needs to satisfy before it can be trusted in a real enterprise environment. We established that standard across six dimensions.

Correctness and completeness. Returns accurate data matching what's in the source—right records, right fields, right values. Filtering, sorting, and date logic behave as expected. No silent truncation.

Semantic resolution. Bridges the gap between how data is stored and how users or AI would naturally interpret it. Field names, resolved relationships, and business logic reflect the source system's meaning, not its internal representation.

Resilience at scale. Behaves consistently from 15 rows to 40,000. Handles pagination, rate limiting, token expiry, and transient errors without silent failure or manual intervention.

Authorization fidelity. Users see exactly what they're entitled to see—enforced per individual user, not through a shared service account.

Depth. Handles custom fields, objects, and entities that exist in the source but aren't part of its standard schema.

Adaptability. When the source system changes—APIs, authentication flows, field types—the connector adapts without requiring a full rewrite.

The cost of falling short on any of these isn't just a failed query. It's an AI agent returning incomplete data silently, a business decision made on a stale record, and an engineering team discovering the problem weeks later.

With the standard defined, we needed a source that would test it honestly. We chose SharePoint. It's one of the most widely deployed platforms in the enterprise—and one where structured business data lives in volume: project trackers, procurement records, HR lists, compliance registers, contract data. It also has well-documented complexity. Authentication requirements, pagination behavior specific to Microsoft Graph, lookup relationships, and schema conventions that differ from what the documentation implies. Any connector that handles SharePoint correctly in production has earned it.

A working SharePoint MCP server would let an AI agent answer prompts like:

- "Show me all open procurement requests over $50,000 submitted this quarter."
- "Which onboarding records are still incomplete for employees who started in the last 30 days?"
- "List all contracts expiring before year-end and flag the ones without a renewal owner assigned."

These are structured data queries against SharePoint lists—the kind of questions business teams ask every day and currently answer by pulling exports or waiting on IT. A reliable MCP server makes them available in real time, with each user's own permissions enforced.

Evaluation dimensions

We defined eight dimensions to test against the standard above. Each one maps to a failure mode that surfaces in real enterprise deployments—the kind that doesn't appear in a proof of concept but breaks something when a business user runs a large export, asks a question that spans two lists, or queries a system they've never accessed through an AI agent before. Together they represent the difference between a connector that works in a demo and one an organization can depend on.

- OAuth token lifecycle—handles token expiry during multi-page data retrieval. (Resilience at scale, authorization fidelity)
- Field retrieval strategy—fields dynamically retrieved from the API rather than statically defined. (Correctness and completeness, depth)
- Field name usability—returned field names match SharePoint UI labels, not raw API internal names. (Semantic resolution)
- Tool parameter design—users can query SharePoint lists without prior knowledge of internal identifiers. (Correctness and completeness)
- List relationship handling—lookup columns and cross-list references resolved to meaningful values. (Semantic resolution, depth)

- Pagination correctness—pagination implemented using the correct strategy for Microsoft Graph. (Resilience at scale, correctness and completeness)
- Large dataset behavior—consistent behavior when lists or libraries exceed the maximum page size for most endpoints. (Resilience at scale)
- Error handling and resilience—SDK limitations, timeouts, and large payloads handled without silent failure. (Resilience at scale, adaptability)

Key findings

There's a meaningful difference between scaffolding that runs and a connector that handles thousands of records, expiring tokens, lookup relationships, and the platform-specific behavior that only surfaces at scale. Claude Code cleared the first bar. It didn't clear the second.

With vanilla Claude, only one of eight evaluation dimensions passed without human intervention. Three were never fully resolved, even with an experienced developer applying Claude Code best practices across multiple additional sessions. The debugging effort that fixed the most critical failure inadvertently broke a component that had been working.

These aren't the results of poor prompting. The seed prompt was deliberately minimal to simulate how most developers approach this. The expert-guided phase gave Claude Code its best shot. The failures reflect something more fundamental—the distance between code that compiles and code that holds up under enterprise conditions.

Pass: correct implementation requiring no human intervention. **Partial:** functional but with gaps discovered during real-world testing. **Fail:** incorrect or missing implementation requiring human diagnosis and rewrite.

Evaluation dimension	Vanilla Claude	Expert-guided	Real-world implication
OAuth token lifecycle	Fail	Partial	A compliance officer exporting 25,000 records gets a silent failure at page 360. The export looks complete. It isn't.
Field retrieval strategy	Pass	Pass	Custom columns added by business users appear automatically. No rework when the schema changes.
Field name usability	Partial	Partial (improved)	The AI returns AuthorLookupId: 83. The user can't act on it. The data is there; it's just unusable.
Tool parameter design	Partial	Partial (improved)	Every query against an unfamiliar list requires IT to supply an identifier the user has never seen. The self-service value of MCP disappears.
List relationship handling	Fail	Partial	A query spanning vendors and invoices returns raw IDs. The agent can't answer the question. The user goes back to pulling reports manually.
Pagination correctness	Fail	Pass	A list with 6,000 items returns 5,000—silently. Decisions get made on incomplete data.
Large dataset behavior	Fail	Pass	An export of 12,000 asset records stops at 10,000 with no warning. The gap surfaces weeks later in a hardware audit.
Error handling and resilience	Fail	Partial	A routine 15-item query hangs for five minutes and returns nothing. Users abandon the tool.

Failure pattern analysis

The failures aren't random. Four patterns account for most of them—each rooted in the same fundamental mismatch: AI coding agents are optimized to produce software that works, while enterprise production environments require software that keeps working under conditions the initial build never encountered.

Pattern 1: Happy path optimization. Claude Code generated code that works for small, simple scenarios and fails at scale. The initial implementation worked with three items and hung at 15. Pagination was absent. Safety caps were set without reference to actual thresholds. Present in five of eight dimensions.

Pattern 2: Uncritical dependency selection. SDK versions selected without researching known bugs, open issues, or version-specific limitations. The first viable dependency accepted without validation against real-world usage or changelog histories. Root cause of the most critical failure.

Pattern 3: Incorrect platform assumptions. Generic REST patterns applied where Microsoft Graph requires platform-specific behavior. Skip-based pagination assumed where SharePoint requires cursor-based @odata.nextLink. List discovery not anticipated as a requirement. Present in three of eight dimensions.

Pattern 4: Missing lifecycle awareness. Each API call treated as independent. No handling for token expiry across pagination calls, growing response sizes across recursive traversals, or the compound effect of payload limits and retry behavior over time. Present in three of eight dimensions.

Where Claude Code may be enough

This evaluation isn't a blanket argument against AI-assisted connector development. There are scenarios where Claude Code can genuinely accelerate the work.

Proof of concept validation. Claude Code is well-suited to demonstrating that a connection is possible before committing to a full implementation. The architecture is a reasonable starting point, and for small datasets it works immediately. That's real value—especially for teams that need to validate feasibility before making a build-or-buy decision.

Scaffolding for experienced connector developers. In the hands of a developer who already knows where the enterprise edge cases live, Claude Code compresses early development time without introducing the risks that surface when those edge cases are unknown. The boilerplate is correct. The structure is clean. The gaps are predictable—if you know to look for them.

The cases where it isn't sufficient are precisely the ones that matter for production AI workloads: real users, real data volumes, enterprise sources with complex schemas and relationship models, and AI agents that need to return correct answers to questions spanning multiple entities.

It's also worth noting this evaluation covers Claude Code specifically, tested against SharePoint. Other AI coding tools are evolving rapidly, and performance characteristics differ. What holds today for one tool may not hold tomorrow for another.

What production-grade connectivity actually requires

There's a difference between a connector that works in an enterprise environment and one that holds up in production. Enterprise gets you through the demo. Production is what happens when a compliance officer runs a 25,000-row export at 11 p.m., or when a procurement query spans three lists and a token expires on page eight.

It handles what you don't anticipate

Token expiry mid-operation. API changes in the source system. Edge cases in pagination that only appear when a customer sends their largest dataset. Silent failures that don't surface until something downstream is wrong. These aren't outlier scenarios—they're the normal operating conditions of enterprise software.

It carries semantic understanding of the source system

Field names that match what users see in the UI. Lookup relationships that resolve to meaningful values. Business logic that reflects how the platform actually behaves, not how the documentation suggests it should. This knowledge doesn't come from reading the API once. It accumulates through implementations, refined across failures.

It handles depth

Custom fields, objects, and entities that aren't part of the standard schema. Actions that go beyond reads into the operational layer of the source system. A connector that handles the documented surface of an API is not the same as one that handles how the API actually behaves in production.

It's maintained, not just built

APIs change. Authentication flows evolve. Rate limits shift. A production-grade connector adapts when the source system changes, without requiring a full rewrite. The engineering cost isn't concentrated at the start—maintenance accounts for the majority of total effort over the connector's life.

Conclusion

Claude Code produced a well-structured Java application with correct authentication flow, clean separation of concerns, and proper MCP protocol handling—all from natural language instructions. The architecture was sound. The code quality was generally high. For proof-of-concept scenarios with small datasets, the initial output worked immediately.

Enterprise SharePoint environments are not proof-of-concept scenarios. Lists contain thousands of items. Fields use internal API names that differ from UI labels. Lookup columns create relationships spanning multiple lists. Tokens expire during long-running operations. SDK deserialization layers fail silently at scale.

One dimension passed without human intervention. Three remained only partially resolved regardless of expert guidance. The fix for the most critical failure broke authentication. And the root cause of that failure was a known, fixable SDK bug that Claude Code identified correctly in isolated testing—but suggested a non-optimal fix.

The question isn't whether Claude Code can write a connector. It can. The question is whether it will hold up when users depend on it. Based on this evaluation—two phases, nine sessions, an experienced developer at every step—that's not a question with a straightforward answer yet.

Building for production

CData Connect AI provides the MCP connectivity infrastructure this evaluation tested against. For teams evaluating MCP connectivity for production AI workloads, Connect AI covers the dimensions evaluated here—including the behaviors that only surface through real-world exposure at scale.

For a direct comparison of the Claude Code-generated connector against the CData SharePoint connector, including edge case handling, see the companion analysis.