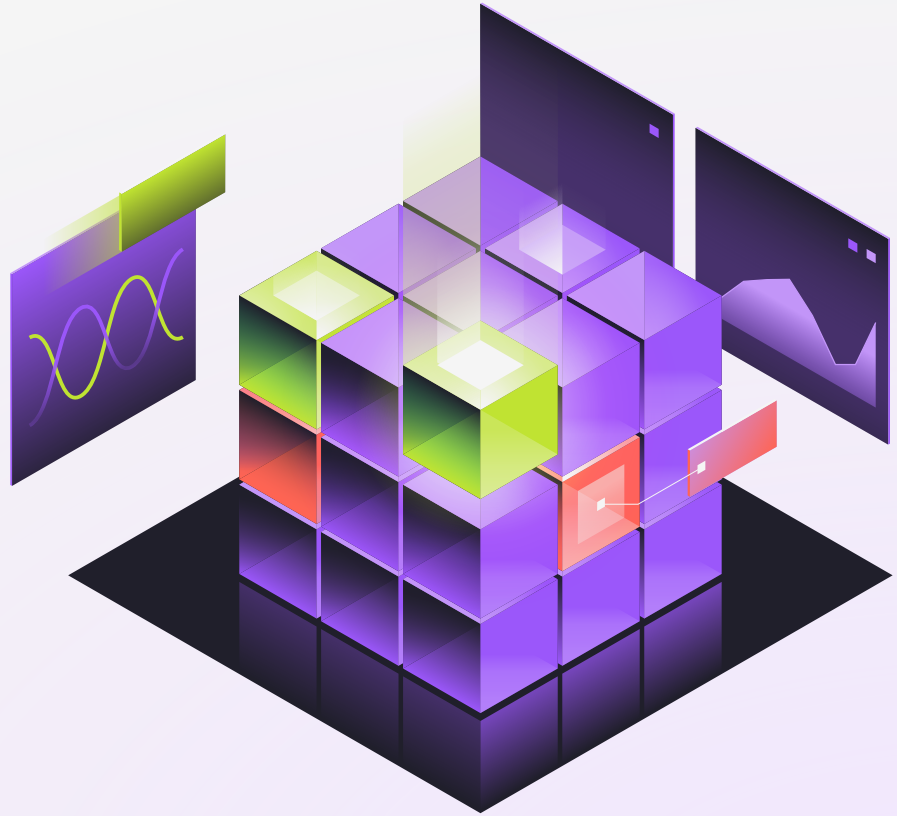
earthmover

MCP BASICS

how agents reach data and tools



Joe Hamman — Earthmover

Responsible Gen-AI for NASA Earthdata · Seattle · August 2026

👋 HELLO 👋

scientist turned software developer

PATH

UW (PhD) → NCAR (post-doc) → CarbonPlan → Earthmover

OPEN SOURCE

Xarray · Pangeo · Pangeo-Forge · Zarr · Xpublish

TODAY

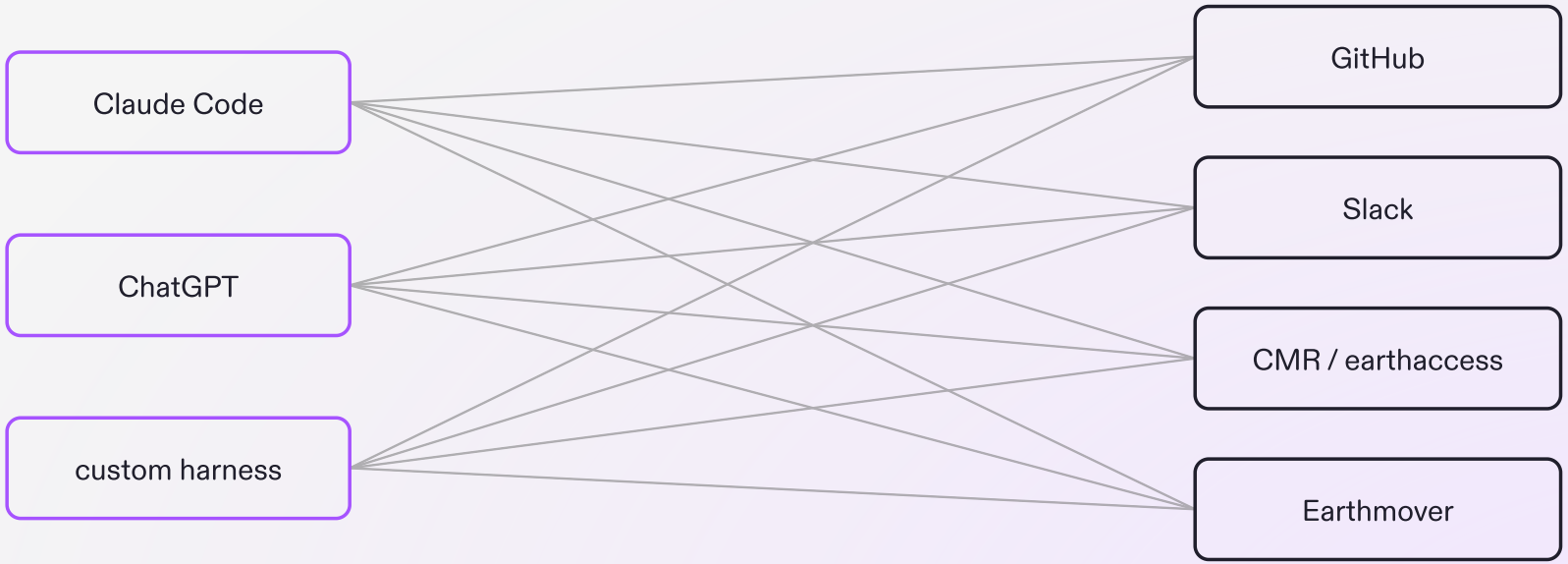
CTO at Earthmover — the cloud platform for scientific data teams

PART ONE

the problem that mcp solves

THE PROBLEM

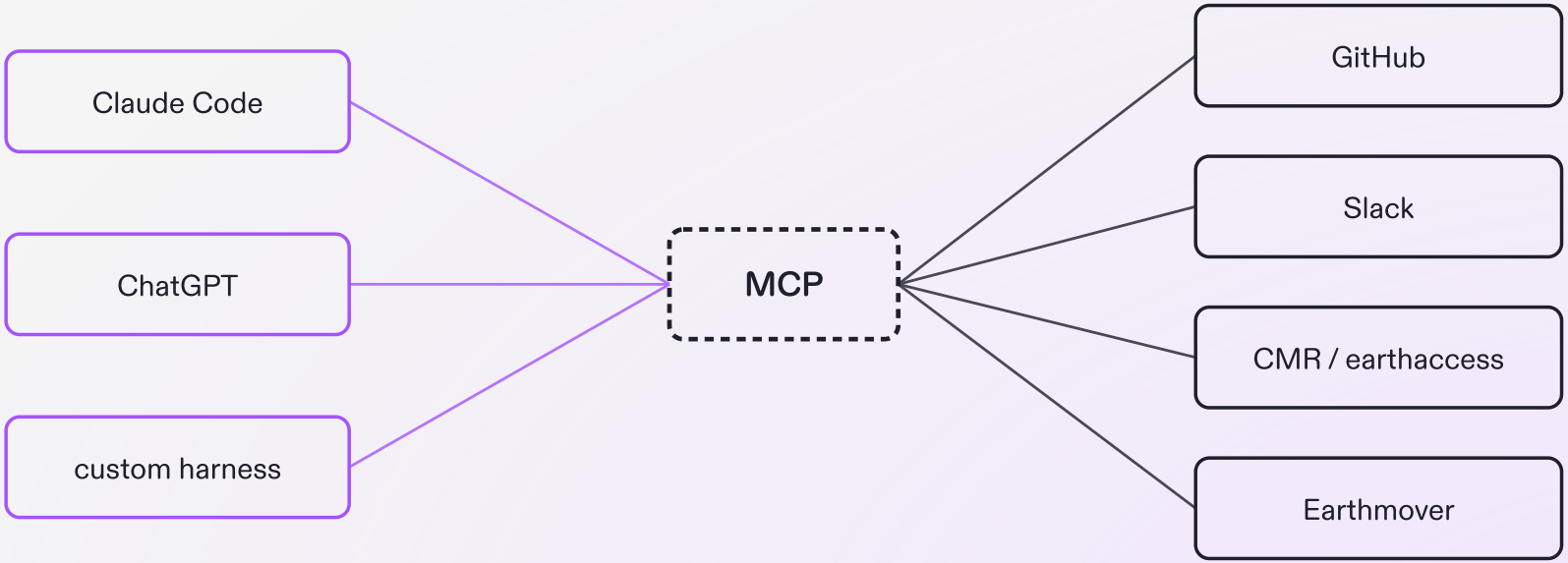
everybody writes the same wrapper, and nobody can share it



the same adapter, written again in every client, and shared by nobody

THE FIX

one protocol replaces the custom integrations



one dependable interface for every agent

ORIENTATION

what is mcp?

MCP is an open protocol. It connects agents to tools and to data. The specification is public. SDKs exist for most languages.

MCP is a wire format. It sends JSON-RPC messages over a transport.

MCP is a client-side capability. Your harness implements it.

> **Note**
>
> MCP is **not** a model feature. No model learned MCP in training.
>
> MCP is **not** an API gateway, and **not** an authentication layer.
>
> MCP does **not** make the agent use your tool well.

SHOW OF HANDS

who has used an mcp server against a remote system?

Which system?

What did the agent do with it?

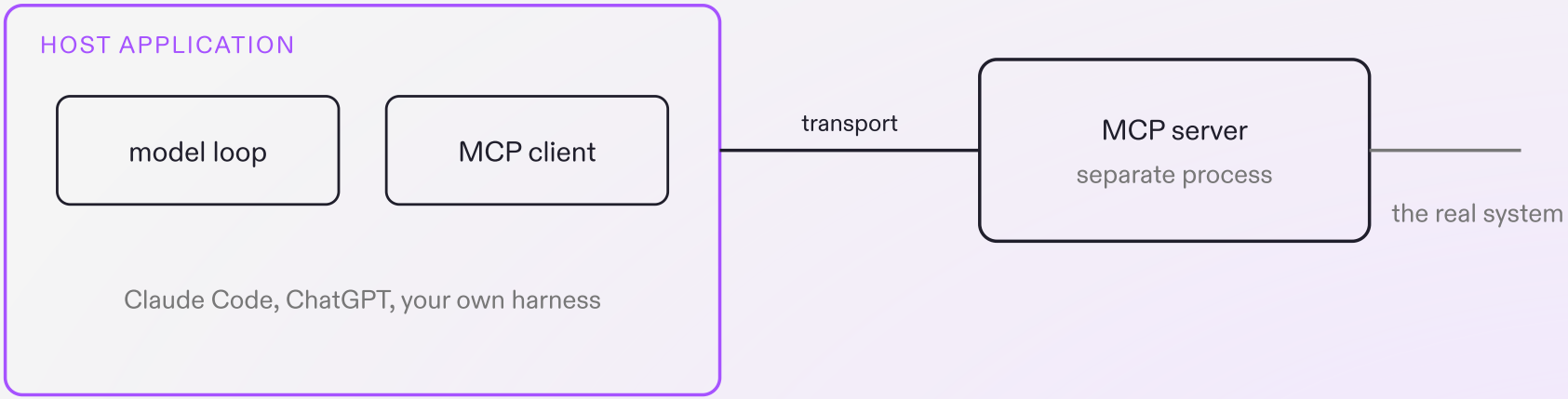
Why did the server help, versus an API call you wrote yourself?

PART TWO

how mcp works

ARCHITECTURE

host, client, server



one client per mcp server — the host may connect to many at once

TRANSPORTS

two ways the client reaches the server

stdio

The client starts the server as a **subprocess**. The two talk over stdin and stdout.

- The server runs on your machine.
- The server runs as you. It uses your credentials and your files.
- You set up no network.
- Use stdio for a server on your laptop.

streamable HTTP

The server runs as a **service** at a URL. Many clients share it.

- The server runs remote, for many users.
- You need real authentication (OAuth).
- The team that owns the system can run the server.
- Use HTTP for a team or an agency deployment.

Examples:

- https://api.githubcopilot.com/mcp/
- https://cmr.earthdata.nasa.gov/mcp/v1
- https://app.earthmover.io/mcp

PRIMITIVES

three things an mcp server can offer

tools

Tools are functions that the agent calls. **The model decides** when to call one.

```
get_granules(collection, time_range)
```

You will use tools for 90% of the work.

resources

A resource is a document that the server offers for reading: a file, a record, a metadata page. A URI names it. Nothing runs when you read it.

`file:///data/era5/README.md`

You attach it to the conversation like a file. **The user or the client picks it**, not the model.

prompts

Prompts are templates that the user starts. **The user decides.** The client shows them as slash commands.

`/summarize-dataset`

A prompt packages expertise. It adds no capability.

DISCOVERY

how an agent learns what a server offers

The client asks each configured server at startup. The answers become part of the model context.

```
client → server   initialize
client → server   tools/list
server → client   [ { name: "get_granules",
                      description: "Search NASA CMR granules for a specific parent
                        collection [...] IMPORTANT — without temporal and/or spatial
                        filters, results represent ALL granules ever archived [...]",
                      inputSchema: { type: "object", required: ["collection_concept_id"],
                                     properties: { collection_concept_id, temporal_start_date,
                                                   spatial_wkt_geometry, ... } } },
                    ... 6 more tools ]
```

> **Important**
>
> Tool descriptions **are** prompt engineering — the model knows nothing else, and pays for the text every turn.

PART THREE

unpacking mcp calls

CONFIGURATION

declaring a server

```
// Claude Code:    ./.mcp.json  (project)   ·   ~/.claude.json  (user)
// Claude Desktop: ~/Library/Application Support/Claude/claude_desktop_config.json
{
  "mcpServers": {
    "earthmover": {
      "type": "http",
      "url": "https://app.earthmover.io/mcp"
    },
    "everything": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-everything"]
    }
  }
}
```

- The **project** file is checked in, so collaborators get the same servers. The **user** file applies to every project.
- Remote servers log in with OAuth; local ones start as a process. No secret lives in the file.

ADD A SERVER

using the claude cli

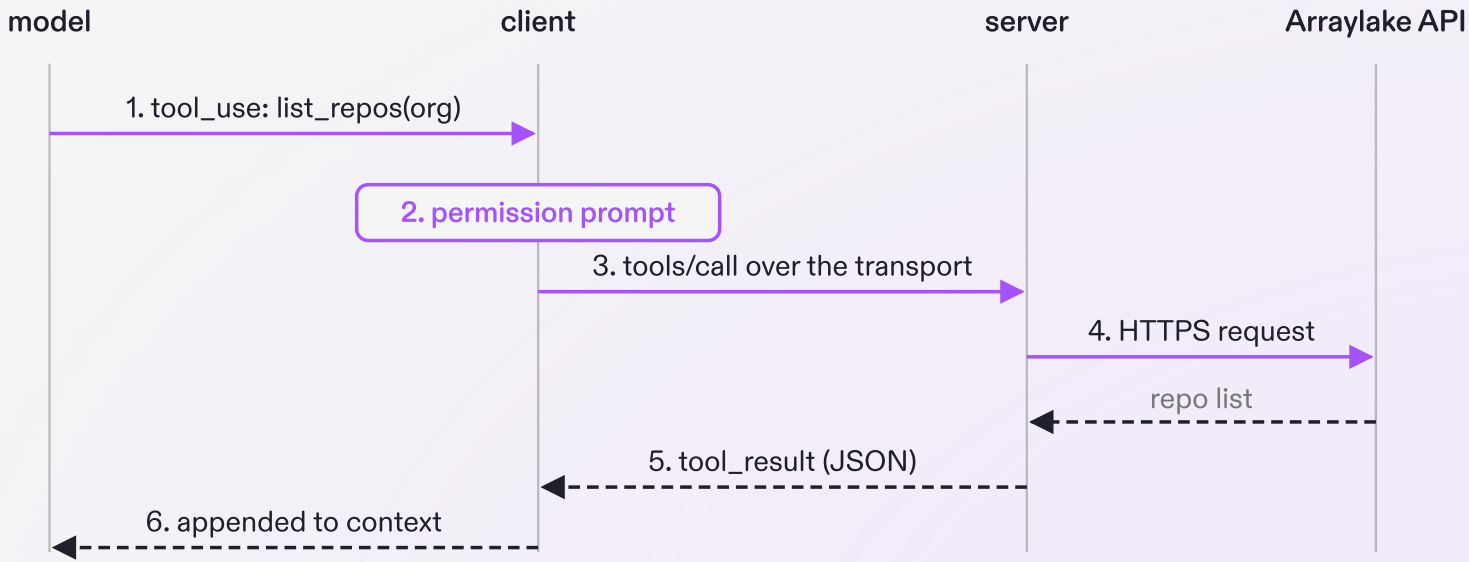
earthmover | claude-code-docs | earthdata | github

```
$ claude mcp add --transport http earthmover https://app.earthmover.io/mcp
Added HTTP MCP server earthmover with URL: https://app.earthmover.io/mcp to local config

$ claude mcp list
earthmover: https://app.earthmover.io/mcp (HTTP) - ✔ Connected

$ claude
> What repos do I have?
```

- **Authentication:** OAuth — run /mcp once
- Add `–scope user` for every project, or `–scope project` to write `.mcp.json` for your team.
- Local server instead? `claude mcp add playwright – npx –y @playwright/mcp@latest`

ROUND TRIP

tool call flow



THE HUMAN IN THE LOOP

what you see and what you approve

The client stops between step 2 and step 3. It asks you.

- **Per call** — you see the exact arguments before the tool runs.
- **Allowlist** — "always allow this tool" writes a rule to your configuration.
- **Read and write differ** — approve `list_repos` freely. Think harder about `delete_repo` 😉.
- Some clients let you change the arguments before you approve.

> **Caution**
>
> "Always allow" gives a standing permission to a program. That program reads model output as instructions.
>
> Give the permission per tool, not per server.

PART FOUR

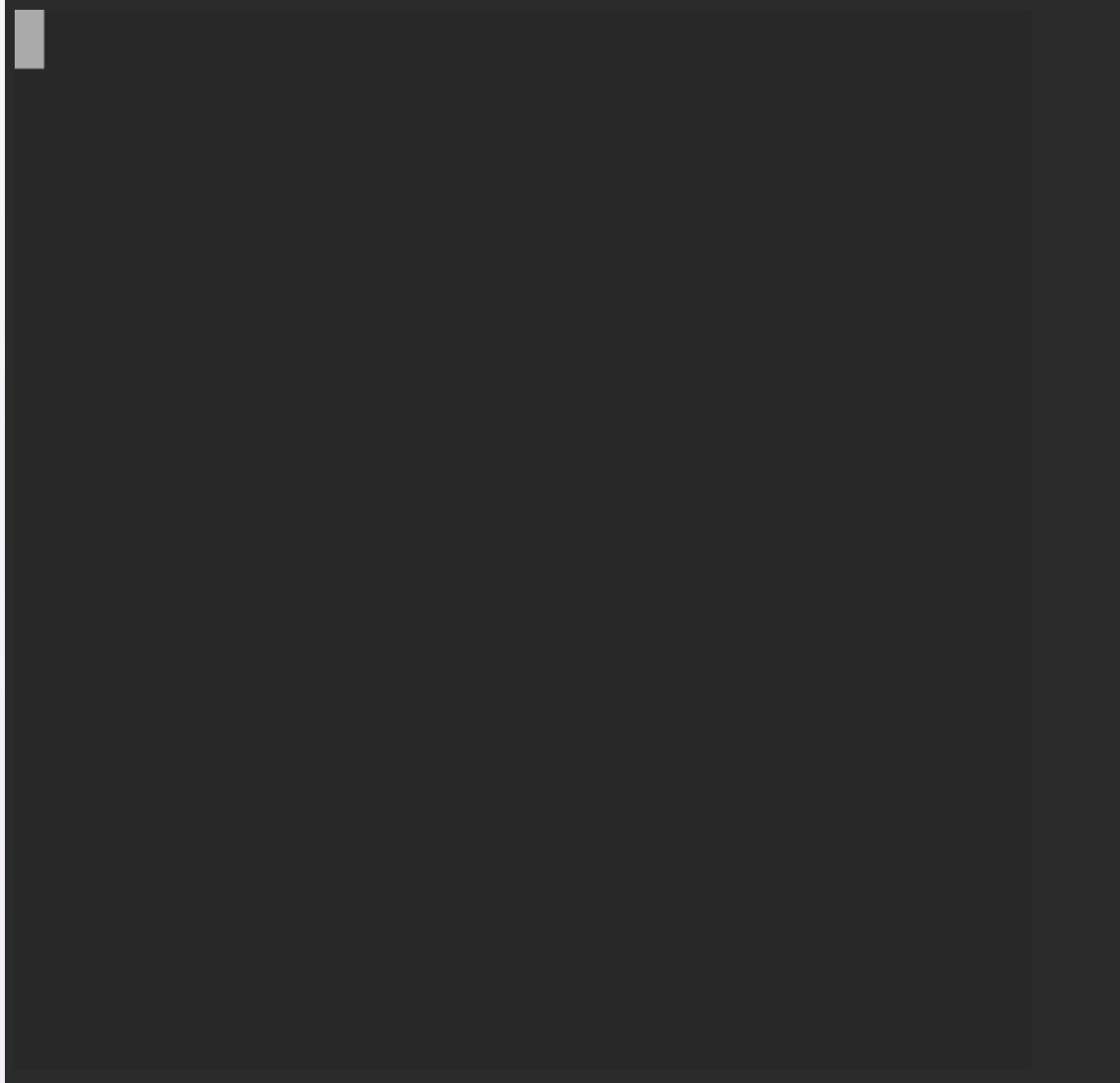
in practice

LIVE DEMO

the agent

1. `earthmover` is in the config. `/mcp` shows it connected; `/context` shows what the model has available.
2. Ask: *"What repos do I have, and what is in the sea surface temperature one?"*
3. Watch: `list_repos` → `get_dataset_info` → `render_dataset_map`.
4. Approve the permission prompt. A map lands in the transcript.

No code, no credentials. The server holds the data; the agent manages the conversation.



DEBUGGING

the mcp inspector

The Inspector is a browser interface. It speaks MCP directly to your server. **It uses no agent and no model.**

```
npx @modelcontextprotocol/inspector
# then connect: https://app.earthmover.io/mcp
```

- List the tools. Read the schemas that the model reads.
- Call a tool by hand. Choose the arguments yourself.
- Watch the raw JSON-RPC messages.

> **Tip**
>
> If the tool fails in the Inspector, the fault is in your server.
>
> If the tool works in the Inspector but the agent calls it wrong, the fault is in your tool descriptions.

LIVE DEMO

the inspector, by hand

1. Connect **earthmover** (Streamable HTTP) — the same server the agent just used, with no model in front of it.
2. **List Tools** — the same tools the agent saw. Open `list_repos`: this text is all that the model sees.
3. Call `list_repos` by hand — the same answer the agent got, **no model, no agent.**
4. If time: connect **everything** (stdio) — tools, resources, prompts, and the other transport.

open the inspector ↗

PART FIVE

where mcp fits

NEIGHBORS

three ways to equip an agent, and when to reach for each

AGENTS.md

Context. *Reach for it when the agent needs to know how this project works.*

- Plain Markdown, one file per repo: commands, conventions, what not to touch.
- Loaded at startup and kept in context — keep it short.
- 60,000+ repos. Claude Code, Codex, Copilot, Cursor all read it.

Skills

Procedure. *Reach for it when a task needs a repeatable method.*

- A folder: `SKILL.md` plus scripts and references.
- Name and description at startup; the full text loads on demand.
- Text and scripts that you own, portable across clients.

MCP

Capability. *Reach for it when the agent must reach a system.*

- A running server — yours, or the provider's.
- Authentication, rate limits, and the data stay server-side.
- The only one of the three that can act on the outside world.

SERVER-SIDE

the provider ships the workflow, not just the api

the owner writes it

The people who run the system define what a correct call looks like. You get their knowledge without reverse-engineering their docs.

Fix the server once, and every agent gets the fix on the next connect.

a narrow, fast path

A small set of tools keeps the agent on supported APIs instead of improvising with `curl`.

One `create_release` call replaces six low-level ones. The server also trims the response — Anthropic measured about **⅓ the tokens**.

what a text file cannot do

Authentication, scopes, and rate limits stay on the server.

Heavy work happens next to the data: filter 2 TB, return the answer.

A server can pause a call to ask you for a parameter, or run for an hour and call back.

GOING DEEPER

things to try this week

1. install one server

Pick a system with a server that exists today: GitHub, a filesystem, or a data catalog. Put it in your client configuration. Check that the tools appear.

2. run the inspector against it

Call one tool by hand. Read the schema that the model reads. Then ask yourself: can you choose the right arguments from that description alone?

3. wrap one of your own scripts

Pick the function that you always explain to a collaborator. Fifteen lines with the Python SDK make it callable by any agent.

WRAP UP

questions and discussion

Where does an MCP server help your workflow? What do you refuse to let it touch?

EXTRAS

reference material

CONFUSIONS

four things that trip everyone up early

the server runs separately from the model

The server is an ordinary process on your machine, or a service that you deploy. You start it. You own it. You can put a `print()` in it.

mcp is a client feature, not a model feature

The model sees only a list of tools. Change the model, and MCP still works. Change to a client without MCP support, and MCP stops.

an available tool is not a tool the agent uses well

Tool choice is a prompting problem. Bad names and vague descriptions make the agent skip your tool, or call it wrong.

more tools do not make a better agent

Each tool description fills context on every turn. Ten sharp tools beat sixty vague ones.

TRUST

you give an mcp server your credentials

- **Supply chain** — `npx some-mcp-server` runs code that you did not read. Read it. Pin the version.
- **Prompt injection** — the client puts tool results into context as text. A poisoned README becomes an instruction.
- **Scope** — someone will ask a server that can write to your bucket to write to your bucket.
- **Audit** — log every call. You will want the record.

> **Important**
>
> The protocol gives you a checkpoint. It gives you no guarantee.
>
> What you connect the agent to decides how the agent behaves.

SOURCES

where this talk comes from

Specification and tools

- modelcontextprotocol.io — specification, SDKs, and server list
- Introducing MCP — Anthropic, the original problem statement
- The 2026-07-28 specification — tasks, elicitation, extensions
- MCP Inspector — the debugging interface
- Connect to MCP servers — Claude Code, the `claude mcp add` flow

Design and practice

- Writing effective tools for AI agents — Anthropic, the token measurements
- A practical guide to the GitHub MCP server — GitHub
- GitHub toolsets — dynamic tool discovery
- MCP servers vs. skills — Red Hat
- AGENTS.md and Agent Skills — the two neighbors