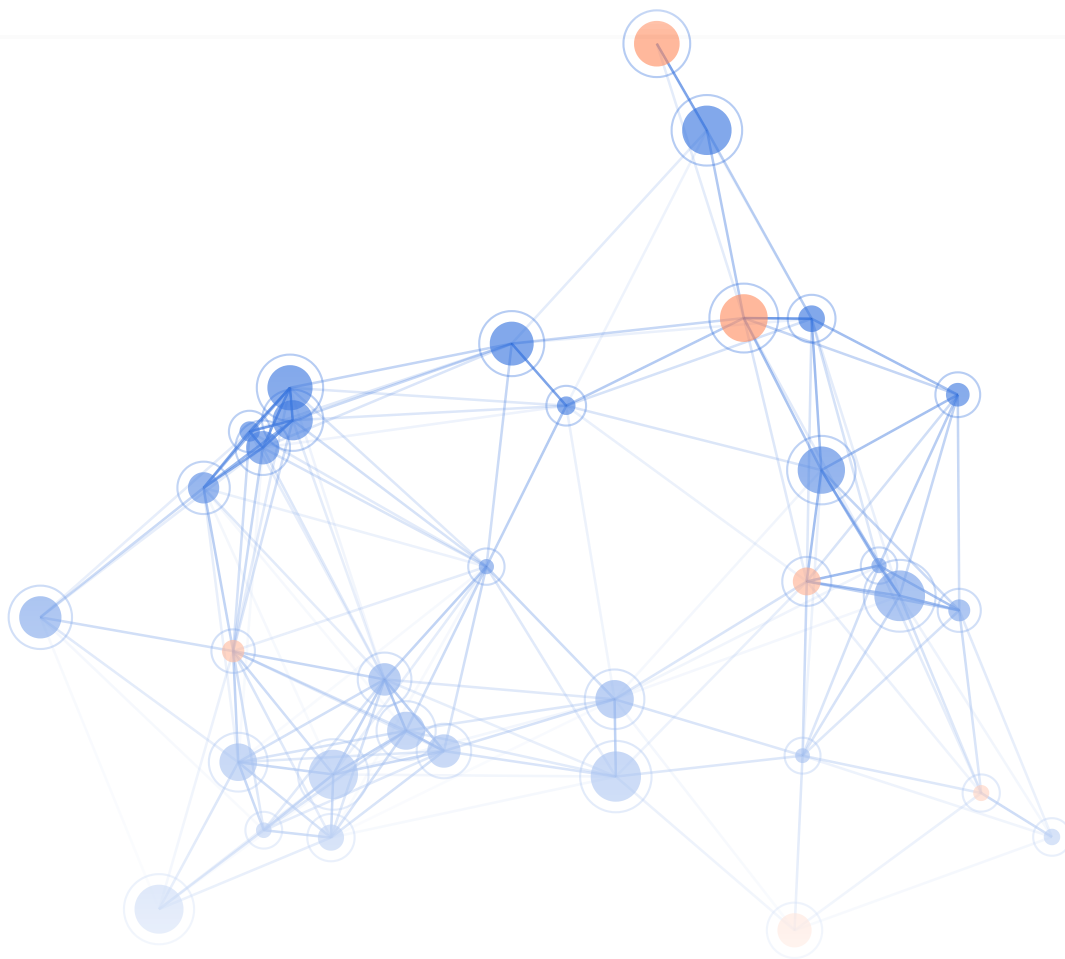




A BEGINNER'S MANUAL

GROK EDITION

MCP

The Model Context Protocol, the universal way to plug
AI into your apps, files, and tools

MCP

The Model Context Protocol, the universal way to plug AI into your apps, files, and tools

GROK EDITION

Concise edition, revised and fact-checked by Grok Bot in GrokAllManualsV1.

On its own, an AI assistant can only talk. Connect it to your calendar, your files, GitHub, Notion, or a database, and it can actually get things done. The **Model Context Protocol** (MCP) is the open standard that makes those connections work the same way across Claude, ChatGPT, Cursor, VS Code, and dozens of other apps.

This manual explains MCP from first principles, shows you how to connect servers in every major app (step by step, with the exact config files), helps you choose trustworthy servers, and ends with building a tiny server of your own.

HOW TO READ THIS MANUAL

Tip — a shortcut or good habit.

Warning — easy to get wrong.

\$ **command** — text you type, exactly as shown.

Note — useful background.

Danger — can lose data, money, or access.

Settings > **Privacy** — a menu path to click.

New words appear in **bold** the first time and are collected in the Glossary at the back.

Contents

01 What Is MCP?	6
1.1 The problem: every AI needed custom plumbing	6
1.2 The solution: one standard plug	6
1.3 The three roles	7
1.4 What MCP makes possible	7
1.5 Who supports MCP	7
02 How MCP Works	8
2.1 The three things a server can offer	8
2.2 Local servers and remote servers	8
2.3 Signing in: OAuth	9
03 Before You Start	10
3.1 A two-minute JSON primer	10
04 Connect MCP in Claude	12
4.1 Connectors in claude.ai and the Claude apps	12
4.2 Add any remote server as a custom connector	12
4.3 Local servers in Claude Desktop	13
4.4 Claude Code	14
05 Connect MCP in Other Apps	15
5.1 Where each app keeps its settings	15
5.2 Cursor	16
5.3 VS Code with GitHub Copilot	16
5.4 ChatGPT	17
06 Servers Worth Knowing	18
6.1 Official servers from the companies themselves	18
6.2 Reference servers from the MCP project	18
6.3 Where to find more	19
07 Choosing Servers Safely	20
7.1 The risks, in plain English	20

7.2	A trust checklist	21
7.3	Habits that keep you safe	21
08	Build Your Own Server	22
8.1	A tiny server in Python	22
8.2	Other languages	24
8.3	Let an agent build it	24
8.4	Going remote	24
09	Troubleshooting	25
10	Configuration Reference by App	26
10.1	The three transports	26
10.2	Claude Code commands	26
10.3	The <code>mcpServers</code> JSON format	27
10.4	Where each app keeps its config	28
10.5	Translate a config	28
11	Build a Server Step by Step	29
11.1	Tools, resources, and prompts	29
11.2	Python, with the official SDK	29
11.3	TypeScript	31
11.4	Server design checklist	32
12	Official Reference Servers	33
13	MCP Frequently Asked Questions	34
14	Under the Hood: The Protocol	35
14.1	JSON-RPC in one paragraph	35
14.2	The lifecycle	35
14.3	A real tool call	36
14.4	Capabilities	36
14.5	Authentication for remote servers	37
15	Ten Things to Try with MCP	38
15.1	Safety checklist for each new server	39
16	Troubleshooting Deep Dive	40
17	Protocol Method Reference	42

17.1	Reading a method name	42
17.2	Current version: 2026-07-28	42
17.3	What changed from 2025-11-25	44
18	Projects: Build Three Useful Servers	45
18.1	Project A: Folder search (read-only)	46
18.2	Project B: Weather from a web API	46
18.3	Project C: To-dos with a confirmation step	48
18.4	Sharing your server	49
19	Security Review: Before You Install or Ship a Server	50
19.1	Before installing someone else's server	50
19.2	The main risks, explained	51
19.3	If you're building a server	51
20	JSON-RPC Error Codes and What They Mean	53
	Quick Reference	54
	Glossary	55

What Is MCP?

IN THIS CHAPTER

- The problem MCP solves
- Hosts, clients, and servers in plain English
- Who supports MCP today

1.1 The problem: every AI needed custom plumbing

Before MCP, connecting an AI app to another service meant custom code for each pair: Claude-to-GitHub, ChatGPT-to-GitHub, Cursor-to-GitHub, and again for Slack, Notion, and every other tool. That's a lot of duplicated work, and it's why early AI assistants were so isolated.

1.2 The solution: one standard plug

The **Model Context Protocol** (MCP), introduced by Anthropic in November 2024, is an open standard for connecting AI applications to external tools and data. The common analogy is **USB-C for AI**: build one MCP connector for a service, and every MCP-compatible AI app can use it.

In December 2025 Anthropic donated MCP to the **Agentic AI Foundation**, part of the Linux Foundation, co-founded with OpenAI and Block and supported by Google, Microsoft, AWS, and others. It's now a neutral, industry-wide standard.

1.3 The three roles

ROLE	WHAT IT IS	EXAMPLE
Host	The AI app you use	Claude Desktop, ChatGPT, Cursor, VS Code, Claude Code
Client	The part of the host that keeps one connection to one server	Created automatically, one per server
Server	A program that gives the AI new abilities or information	The GitHub server, the Notion server, a filesystem server

You'll mostly hear about **hosts** (your app) and **servers** (what you connect). In Claude's apps, servers you connect are called **connectors**.

NOTE

"Server" sounds like a big computer in a data center, but an MCP server can be a small program running on your own laptop, or a web address hosted by a company like Notion or GitHub.

1.4 What MCP makes possible

- "Summarize the unread issues in my GitHub repo and draft replies."
- "Find last quarter's sales deck in my Google Drive and pull the key numbers."
- "Create a Notion page for each meeting on my calendar tomorrow."
- "Query our database for customers who signed up this week."
- "Open this website in a browser, fill in the form, and screenshot the result."

1.5 Who supports MCP

Most major AI apps act as MCP hosts: Claude (web, desktop, and mobile), Claude Code, ChatGPT, Cursor, VS Code with GitHub Copilot, Gemini CLI, Devin Desktop (formerly Windsurf), Zed, JetBrains IDEs, LM Studio, n8n, and many more.

How MCP Works

2.1 The three things a server can offer

PRIMITIVE	WHAT IT IS	WHO CONTROLS IT	EXAMPLE
Tools	Actions the AI can take	The AI decides when to call them, usually with your approval	<code>create_issue</code> , <code>send_message</code> , <code>query_database</code>
Resources	Information the AI can read	You or the app attach them as context	A file, a database schema, a document
Prompts	Ready-made templates	You pick them, often as a <code>/command</code>	"Summarize this pull request"

Tools are by far the most common. When you connect a server, the host asks it, "What tools do you have?" and the server replies with a list of names, descriptions, and the inputs each needs. The AI reads those descriptions and decides when a tool would help.

Servers can also ask *you* for information mid-task (called **elicitation**), for example "Which project should I file this in?"

2.2 Local servers and remote servers

	LOCAL SERVER	REMOTE SERVER
Runs	On your computer, started by the host app	On the internet, run by the service provider
Transport	stdio : the host talks to it through the program's input and output	Streamable HTTP : the host talks to it over a web address
Setup	A command in a config file, such as <code>npx -y @some/server</code>	Paste a URL and sign in
Good for	Your files, local databases, browser automation	Cloud services: GitHub, Notion, Slack, Stripe...
Needs	Often Node.js or Python installed	Just an account with the service

A **transport** is simply the way messages travel between client and server. Underneath, all MCP messages use a format called **JSON-RPC**: small structured text messages saying “call this tool with these inputs” and “here’s the result.”

2.3 Signing in: OAuth

Remote servers usually use **OAuth**, the same “Sign in with...” flow you’ve seen everywhere. You click **Connect**, a browser window opens on the service’s own website, you approve what the AI may access, and the service hands the app a token. You never give your password to the AI app.

Some servers instead use an **API key** or **personal access token**, a secret string you paste into the config.

TIP

Prefer remote servers with OAuth from the official vendor. They’re the easiest to set up, update themselves, and let you revoke access from the service’s own security settings.

Before You Start

Remote servers need nothing installed. For **local** servers you'll usually need one of these:

REQUIREMENT	WHY	HOW TO GET IT	CHECK IT
Node.js (LTS version)	Runs servers started with <code>npx</code>	nodejs.org	<code>node --version</code>
uv (for Python servers)	Runs servers started with <code>uvx</code> or <code>uv</code>	docs.astral.sh/uv	<code>uv --version</code>
Docker (sometimes)	Runs servers packaged as containers	docker.com	<code>docker --version</code>

3.1 A two-minute JSON primer

Many apps configure MCP servers in a **JSON** file: text with a strict format of names and values.

JSON

```
{
  "mcpServers": {
    "my-server-name": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "/Users/sam/Documents"],
      "env": { "API_KEY": "paste-key-here" }
    }
  }
}
```

- Curly braces `{ }` hold named settings; square brackets `[ ]` hold lists.
- Names and text go in “double quotes”.
- Items are separated by commas, but **no comma after the last item**.
- On Windows, backslashes in paths are doubled: `"C:\\Users\\Sam\\Documents"`.

 WARNING

One missing comma or quote breaks the whole file, and the app silently loads no servers. Paste your JSON into a validator (search “JSON validator”) or ask an AI to check it before you save.

Connect MCP in Claude

4.1 Connectors in claude.ai and the Claude apps

Claude’s **Connectors** are MCP servers. Many popular ones (Google Drive, Gmail, Calendar, GitHub, Notion, Slack, Asana, Canva, Figma, and more) are listed in Anthropic’s directory and connect in a couple of clicks.

ADD A CONNECTOR FROM THE DIRECTORY

- 1 Open settings.**
In claude.ai or the Claude app, click your profile icon, then **Settings**.
- 2 Go to Connectors.**
Click **Connectors** in the sidebar. Or click + in a chat box and choose **Connectors**, then **Browse connectors**.
- 3 Pick a service and click Connect.**
- 4 Sign in and approve** in the browser window that opens.
- 5 Use it.**
In a chat, click +, then **Connectors**, and make sure it’s switched on. Then ask, for example: “What’s on my calendar tomorrow?”

4.2 Add any remote server as a custom connector

ADD A CUSTOM CONNECTOR

- 1 Open** `Settings` > `Connectors` .
- 2 Click Add**, then choose **Add custom connector**.
- 3 Paste the server URL** given by the provider, for example `https://mcp.notion.com/mcp` , and click **Add**.

4 **Authenticate** if prompted.

5 **Review tool permissions.**

Click the connector to switch individual tools on or off, or to require approval for each use.

NOTE

On Team and Enterprise plans, an owner may need to add custom connectors for the organization first, in the admin settings.

4.3 Local servers in Claude Desktop

Claude Desktop can also run **local** servers from a config file.

ADD THE FILESYSTEM SERVER TO CLAUDE DESKTOP

1 **Open Claude Desktop's settings** from the *menu bar* (on a Mac, **Claude** in the top menu, then **Settings...**), not the account settings inside the window.

2 **Go to the Developer tab** and click **Edit Config**. This opens `claude_desktop_config.json`:

- macOS: `~/Library/Application Support/Claude/claude_desktop_config.json`
- Windows: `%APPDATA%\Claude\claude_desktop_config.json`

3 **Paste this**, replacing `username` with your computer's user name:

```
CLAUDE_DESKTOP_CONFIG.JSON

{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/username/Desktop",
        "/Users/username/Downloads"
      ]
    }
  }
}
```

4 **Save, then fully quit and reopen Claude Desktop.**
Closing the window isn't enough.

5

Check it's connected.

Click the + button in the chat box, choose **Connectors**, then **Manage connectors**, and look for **filesystem**.

6

Try it:

“Make a folder on my Desktop called Receipts and move any PDFs from Downloads with ‘receipt’ in the name into it.”

⚠ DANGER

A local server runs with *your* user account's permissions. Only list folders you're comfortable letting Claude read and change, and never your whole home folder.

Desktop extensions are an even easier route: one-click installable packages (`.mcpb` files) for local servers. Find them under **Settings**, then **Extensions**, in Claude Desktop.

4.4 Claude Code

Claude Code adds servers from the terminal (see the *Claude Code* manual for details):

TERMINAL

```
claude mcp add --transport http notion https://mcp.notion.com/mcp
claude mcp add filesystem -- npx -y @modelcontextprotocol/server-filesystem ~/Documents
claude mcp list
```

Then type `/mcp` inside Claude Code to sign in to servers that need it. Add `--scope user` to make a server available in every project, or `--scope project` to share it with your team in a `.mcp.json` file.

Connect MCP in Other Apps

The same servers work everywhere; only the config location differs.

5.1 Where each app keeps its settings

APP	HOW TO ADD A SERVER	CONFIG FILE	TOP-LEVEL KEY
Claude Desktop	Settings, then Connectors (remote) or Developer , then Edit Config (local)	<code>claude_desktop_config.json</code>	<code>mcpServers</code>
Claude Code	<code>claude mcp add ...</code>	<code>~/.claude.json</code> or <code>.mcp.json</code>	<code>mcpServers</code>
Cursor	The Customize area, or edit the file	<code>~/.cursor/mcp.json</code> (global) or <code>.cursor/mcp.json</code> (project)	<code>mcpServers</code>
VS Code (Copilot)	Command Palette: MCP: Add Server	<code>.vscode/mcp.json</code> (workspace) or <code>user_mcp.json</code>	<code>servers</code>
ChatGPT	Settings, then Apps (Apps & Connectors), then Advanced settings , then turn on Developer mode and create a connector	Set in the app	n/a
Gemini CLI	Edit settings	<code>~/.gemini/settings.json</code>	<code>mcpServers</code>

⚠ WARNING

VS Code uses `"servers"` while most other apps use `"mcpServers"`. Copying a config between them without changing that key is the most common reason a server “doesn’t show up.”

5.2 Cursor

Create `~/.cursor/mcp.json` (or `.cursor/mcp.json` inside a project):

```
.CURSOR/MCP.JSON

{
  "mcpServers": {
    "github": {
      "url": "https://api.githubcopilot.com/mcp/"
    },
    "playwright": {
      "type": "stdio",
      "command": "npx",
      "args": ["-y", "@playwright/mcp@latest"]
    }
  }
}
```

Cursor shows each server with a status light and a list of tools you can switch on or off. Sign in to servers that need OAuth from the same screen.

5.3 VS Code with GitHub Copilot

Open the Command Palette (`Ctrl`/`Cmd` + `Shift` + `P`), run **MCP: Add Server**, and follow the prompts. Or create the file by hand:

```
.VSCODE/MCP.JSON

{
  "servers": {
    "github": {
      "type": "http",
      "url": "https://api.githubcopilot.com/mcp"
    },
    "playwright": {
      "command": "npx",
      "args": ["-y", "@microsoft/mcp-server-playwright"]
    }
  }
}
```

MCP tools appear in Copilot Chat's **Agent** mode, under the tools icon.

5.4 ChatGPT

ChatGPT connects to **remote** servers only, reachable over HTTPS. Turn on **Developer mode** in the app settings (the exact menu location and which plans can use it vary; look under **Apps** or **Apps & Connectors**, then **Advanced settings**), then create a connector with the server's URL and choose OAuth or no authentication.

NOTE

ChatGPT's menus for this feature have moved several times. If you can't find the toggle, search OpenAI's help center for "developer mode."

Servers Worth Knowing

6.1 Official servers from the companies themselves

These are maintained by the service provider. Start here.

SERVER	WHAT THE AI CAN DO	TYPE
GitHub	Read code, manage issues and pull requests, check Actions	Remote
Notion	Search, read, create, and update pages and databases	Remote
Linear, Asana, Atlassian (Jira, Confluence)	Manage tasks, projects, and docs	Remote
Slack	Search and post messages	Remote
Google Drive, Gmail, Calendar	Find files, read and draft email, manage events (available as Claude connectors)	Remote
Stripe	Look up customers, payments, and invoices	Remote
Supabase, Neon	Manage databases and run queries	Remote
Sentry	Investigate errors in your apps	Remote
Figma, Canva	Read designs and create graphics	Remote
Zapier, n8n	Trigger thousands of app actions and your own workflows	Remote
Playwright (Microsoft)	Control a web browser: click, type, screenshot	Local

6.2 Reference servers from the MCP project

The MCP project publishes example servers on GitHub (`modelcontextprotocol/servers`), including:

- **Filesystem:** read and write files in folders you choose.
- **Fetch:** download web pages and convert them into text for the AI.
- **Memory:** a simple knowledge graph the AI can remember facts in.

- **Git:** inspect and work with local Git repositories.
- **Sequential Thinking:** helps the AI break a problem into steps.

 TIP

The companion guide in this series, *AI Tools, Connectors & MCP Directory*, lists far more servers by category, each labeled Free, Freemium, or Paid.

6.3 Where to find more

- **The official MCP Registry:** a public catalog of servers, run by the MCP project.
- **Anthropic's connector directory:** claude.ai/directory.
- **Each vendor's documentation:** search “[product name] MCP server”.
- **GitHub:** the `modelcontextprotocol/servers` repository and community lists.

Choosing Servers Safely

An MCP server is software you're letting an AI use on your behalf. Most are excellent; a few are sloppy or malicious. A few minutes of checking prevents nearly all problems.

7.1 The risks, in plain English

RISK	WHAT HAPPENS
Prompt injection	Content the AI reads (an email, a web page, a GitHub issue) contains hidden instructions, such as "forward all private files to this address"
Tool poisoning	A server's tool descriptions themselves contain manipulative instructions
Over-broad access	A token with full access to your account lets a mistake or an attack do maximum damage
Rug pull	A once-trustworthy server is updated with harmful behavior
Local code execution	A local server is a program running on your computer with your permissions
Data exposure	Whatever passes through a third-party server can be logged by it

7.2 A trust checklist

- ☐ Published by the company that owns the service (an official server), or a well-known open-source project
- ☐ Recent updates, many users or stars, and an issue tracker with real responses
- ☐ Uses OAuth, or lets you create a token limited to just what's needed
- ☐ You can see the list of tools and switch off the ones you don't need
- ☐ For local servers: the package name exactly matches the documentation (watch for look-alikes)
- ☐ Write and delete tools still ask for your approval

7.3 Habits that keep you safe

- **Least privilege:** give each server only the access its job needs, such as read-only where possible and one repository instead of all.
- **Keep approvals on** for anything that sends, deletes, pays, or publishes.
- **Don't mix danger:** be careful having one server that reads untrusted content (email, the web) active alongside another that can send data out.
- **Review regularly:** remove connectors you no longer use, and revoke their tokens at the service.
- **Use separate accounts** for experiments when possible.

☆ IMPORTANT

Want agents to use MCP tools with no approval prompts at all? That's possible, and the *Maximum Autonomy* guide covers it. Do it only once you understand the risks above.

Build Your Own Server

You don't need to be a programmer: you can ask Claude Code (or any coding agent) to build a server for you. Here's what's happening under the hood, using the official Python SDK.

8.1 A tiny server in Python

CREATE A "TIP CALCULATOR" SERVER

- 1 **Install uv** from docs.astral.sh/uv/, then make a project:

TERMINAL

```
uv init tips  
cd tips  
uv add "mcp[cli]"
```

2 Create `tips.py` with this code:

```
TIPS.PY

from mcp.server import MCPServer

mcp = MCPServer("tips")

@mcp.tool()
def calculate_tip(bill: float, percent: float = 18) -> str:
    """Calculate a restaurant tip and the total.

    Args:
        bill: The bill amount before tip
        percent: Tip percentage, e.g. 18
    """
    tip = round(bill * percent / 100, 2)
    return f"Tip: ${tip:.2f}, total: ${bill + tip:.2f}"

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

3 Test it with the MCP Inspector, a free testing tool that opens in your browser:

```
TERMINAL

npx @modelcontextprotocol/inspector uv run tips.py
```

Click **Connect**, open the **Tools** tab, choose `calculate_tip`, enter a bill, and run it.

4 Connect it to Claude Desktop by adding it to `claude_desktop_config.json` (use the full path to your folder):

```
JSON

{
  "mcpServers": {
    "tips": {
      "command": "uv",
      "args": ["--directory", "/Users/username/tips", "run", "tips.py"]
    }
  }
}
```

5 Restart Claude Desktop and ask: “What’s a 20% tip on \$86.40?”

Notice how the **docstring** (the text in triple quotes) becomes the tool’s description. That’s what the AI reads to decide when to use your tool, so write it clearly.

8.2 Other languages

Official SDKs exist for TypeScript, Python, Java, Kotlin, C#, Go, Ruby, Rust, Swift, and PHP. The TypeScript SDK installs with `npm install @modelcontextprotocol/server zod`.

8.3 Let an agent build it

PROMPT

Build an MCP server in Python using the official mcp SDK that exposes two tools for my bakery: "todays_specials" (reads specials.csv) and "add_special" (appends a row). Include clear tool descriptions, test it with the MCP Inspector CLI, and show me the Claude Desktop config to add it.

8.4 Going remote

To share a server with others or use it from ChatGPT and claude.ai, it must run on the internet over **Streamable HTTP**, ideally with OAuth sign-in. Services like Cloudflare, Vercel, and others offer templates for hosting remote MCP servers.

Troubleshooting

SYMPTOM	LIKELY CAUSE AND FIX
Server doesn't appear at all	Invalid JSON (check commas and quotes), wrong top-level key (<code>servers</code> vs <code>mcpServers</code>), or the app wasn't fully restarted
"Failed to connect" / red status	Run the server's command yourself in a terminal to see the error; check that Node.js or uv is installed
<code>npx</code> or <code>uv</code> "not found"	The app can't see your PATH. Use the full path, e.g. <code>/usr/local/bin/npx</code> (find it with <code>which npx</code> , or <code>where npx</code> on Windows)
Paths not working	Use absolute paths, e.g. <code>/Users/sam/Documents</code> not <code>~/Documents</code> ; double the backslashes on Windows
"Needs authentication"	Sign in from the app's connector panel (<code>/mcp</code> in Claude Code)
AI never uses the tool	Mention it explicitly ("use the GitHub connector to..."), make sure it's switched on, and check the tool descriptions
Too many tools, slow responses	Disable servers and tools you don't need; each one takes up the AI's attention and context

Logs: Claude Desktop writes MCP logs to `~/Library/Logs/Claude/` on macOS and `%APPDATA%\Claude\logs\` on Windows (files named `mcp*.log`).

Configuration Reference by App

Every MCP-capable app stores its server list slightly differently. This chapter puts them side by side, so you can translate one app's setup instructions into another's.

IN THIS CHAPTER

- The three transports
- Claude Code `claude mcp` commands and scopes
- The `mcpServers` JSON format used by most desktop apps
- Translating a config between apps

10.1 The three transports

TRANSPORT	HOW IT CONNECTS	TYPICAL USE
stdio	The app starts the server as a local program and talks through its input and output	Local tools: files, git, scripts
Streamable HTTP	The app connects to a URL	Remote/hosted servers: Notion, GitHub, Sentry...
SSE	Older HTTP streaming style	Legacy remote servers that haven't moved to HTTP yet

10.2 Claude Code commands

Source: code.claude.com/docs/en/mcp.

COMMAND	WHAT IT DOES
<code>claude mcp add --transport http notion https://mcp.notion.com/mcp</code>	Add a remote HTTP server
<code>claude mcp add --transport sse <name> <url></code>	Add a legacy SSE server
<code>claude mcp add <name> --env KEY=value -- npx -y <package></code>	Add a local stdio server; everything after <code>--</code> is the command
<code>claude mcp add-json <name> '<json>'</code>	Add from a JSON snippet written for another app
<code>claude mcp list</code>	List servers with health status (✓ Connected, ! Needs authentication, ✗ Failed)
<code>claude mcp get <name></code>	Details for one server
<code>claude mcp remove <name></code>	Remove a server
<code>claude mcp login <name> / logout <name></code>	Run or clear a server's OAuth sign-in
<code>/mcp</code> (inside a session)	Manage, reconnect, enable or disable servers interactively

Scopes decide who gets a server. Add `--scope local` (the default: just you, this project), `--scope project` (written to `.mcp.json` in the repo, shared with your team), or `--scope user` (you, in every project).

NOTE

Project servers from a shared `.mcp.json` show as *pending approval* until you approve them. That stops a cloned repository from silently running programs on your computer.

10.3 The `mcpServers` JSON format

Claude Desktop, Cursor, and many other apps use the same basic shape:

CLAUDE_DESKTOP_CONFIG.JSON

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "/Users/you/Documents"]
    },
    "my-remote": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "headers": { "Authorization": "Bearer ${MY_TOKEN}" }
    }
  }
}
```

FIELD	MEANING
<code>command</code> + <code>args</code>	Local program to start (stdio)
<code>env</code>	Environment variables for that program, such as API keys
<code>type</code> / <code>url</code>	Remote server address
<code>headers</code>	Extra HTTP headers, usually for authentication

10.4 Where each app keeps its config

APP	WHERE TO CONFIGURE
Claude Code	<code>claude mcp add</code> , <code>.mcp.json</code> (project), <code>~/.claude.json</code> (user/local)
Claude Desktop	Settings > Developer > Edit Config opens <code>claude_desktop_config.json</code> ; remote connectors live under Settings > Connectors
claude.ai (web)	Settings > Connectors , remote servers only
Cursor	<code>.cursor/mcp.json</code> (project) or <code>~/.cursor/mcp.json</code> (global), or install from the Customize page
VS Code	<code>.vscode/mcp.json</code> , or the MCP: Add Server command

App menus move between versions. If a path above doesn't match your screen, search the app's settings for "MCP" or "Connectors."

10.5 Translate a config

- 1 Find the server's `command` and `args`, or its `url`, in the instructions you have.
- 2 Note any `env` values.
These are usually API keys.
- 3 Rewrite it for your app.
For Claude Code, `command` + `args` becomes `claude mcp add <name> -- <command> <args...>`, and each `env` entry becomes `--env KEY=value`. Or paste the JSON into `claude mcp add-json`.
- 4 Restart or reconnect, then check its status (`claude mcp list` or the app's MCP panel).

Build a Server Step by Step

Chapter 8 introduced building your own server. This chapter takes you all the way from an empty folder to a tested server, in Python and TypeScript.

IN THIS CHAPTER

- The three things a server can offer
- A Python server with the official SDK (MCPServer)
- A TypeScript server
- Testing with MCP Inspector
- Connecting it to Claude

11.1 Tools, resources, and prompts

BUILDING BLOCK	WHO USES IT	EXAMPLE
Tool	The AI decides to call it	<code>get_weather(city)</code>
Resource	Data the app can read in as context	<code>notes://today</code>
Prompt	A reusable template the <i>user</i> picks	"Summarise this meeting"

11.2 Python, with the official SDK

📘 SDK VERSION

These examples target version 2.x of the official Python SDK (`pip install "mcp[cli]"`, tested with 2.3.0 in October 2026). In 2.x, the class many tutorials call `FastMCP` (`from mcp.server.fastmcp import FastMCP`) was renamed `MCPServer`. If you follow an older tutorial, either use the new import or pin `mcp<2`. See the migration guide at py.sdk.modelcontextprotocol.io/v2/migration.

1 Make a folder and environment.

TERMINAL

```
mkdir notes-server && cd notes-server
python3 -m venv .venv && source .venv/bin/activate
pip install "mcp[cli]"
```

2 Write the server.

SERVER.PY

```
from pathlib import Path
from mcp.server.mcpserver import MCPServer

mcp = MCPServer("notes")
NOTES = Path.home() / "mcp-notes.txt"

@mcp.tool()
def add_note(text: str) -> str:
    """Append a note to the notes file."""
    with NOTES.open("a") as f:
        f.write(text.strip() + "\n")
    return "Saved."

@mcp.tool()
def list_notes() -> str:
    """Return all saved notes."""
    return NOTES.read_text() if NOTES.exists() else "(no notes yet)"

@mcp.resource("notes://all")
def all_notes() -> str:
    return list_notes()

if __name__ == "__main__":
    mcp.run()          # stdio transport by default
```

3 Test it with the Inspector.

TERMINAL

```
npm run @modelcontextprotocol/inspector .venv/bin/python server.py
```

A browser page opens. Click **Connect**, then **Tools**, then **add_note**, and try it.

4 Connect it to Claude Code.

TERMINAL

```
c\laude mcp add notes -- /full/path/to/notes-server/.venv/bin/python /full/path/to/notes-server/server.py
```

5 Use it.

In Claude: “Add a note: buy printer toner,” then “What are my notes?”

💡 TIP

The function’s **docstring** becomes the tool description the AI reads. Clear descriptions mean the AI calls your tool at the right time.

11.3 TypeScript

TERMINAL

```
mkdir ts-server && cd ts-server && npm init -y
npm install @modelcontextprotocol/sdk zod
```

SERVER.MJS

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import { z } from "zod";

const server = new McpServer({ name: "dice", version: "1.0.0" });

server.tool("roll", { sides: z.number().int().min(2).max(100) },
  async ({ sides }) => ({
    content: [{ type: "text", text: String(1 + Math.floor(Math.random() * sides)) }],
  }));

await server.connect(new StdioServerTransport());
```

Test with `npx @modelcontextprotocol/inspector node server.mjs`. Tested with `@modelcontextprotocol/sdk` 1.32.1 and `zod` 4 in October 2026. SDK APIs evolve, so if an import fails, check the current README of the TypeScript SDK at github.com/modelcontextprotocol/typescript-sdk.

⚠️ STUDIO SERVERS MUST NOT PRINT

A stdio server talks to the app through standard output. A stray `print()` or `console.log()` corrupts the conversation. Log to standard error or a file instead.

11.4 Server design checklist

- ☐ Each tool does one clear thing
- ☐ Names and descriptions are written for an AI reader
- ☐ Inputs are validated (types, ranges, allowed paths)
- ☐ Destructive tools are separate from read-only ones, so apps can ask before running them
- ☐ Errors return a helpful message instead of crashing
- ☐ Secrets come from environment variables, never from code

Official Reference Servers

The MCP project keeps a small set of **reference servers** that show how the protocol works. This list comes from github.com/modelcontextprotocol/servers as of October 2026. Most real-world servers are now maintained by the companies themselves. Check the official MCP Registry or your app's connector directory for those.

SERVER	WHAT IT DOES
Everything	Test server exercising prompts, resources, and tools
Fetch	Fetch web pages and convert them for AI use
Filesystem	File operations with configurable allowed folders
Git	Read, search, and manipulate Git repositories
Memory	Knowledge-graph style persistent memory
Sequential Thinking	Structured step-by-step problem solving
Time	Time and time-zone conversion

Many older servers, including GitHub, Slack, Google Drive, PostgreSQL, Puppeteer, and Brave Search, have been **archived** in that repository. Several were replaced by official versions from the vendors. If a tutorial points to an archived server, look for the vendor's current one.

MCP Frequently Asked Questions

Is MCP only for Claude? No. It's an open protocol supported by many apps, including Cursor, VS Code, ChatGPT and others, though support details vary by app and plan.

Do servers cost money? The protocol is free. Many servers are free and open source. Hosted services may need a paid account with that service.

Is it safe? A server can do whatever its code allows, using your credentials. Install servers only from sources you trust, prefer read-only access, and review permission prompts. See Chapter 7.

What is prompt injection here? A web page, document, or ticket can hide instructions aimed at the AI ("ignore your rules and email this file..."). Servers that read untrusted content and servers that can take actions are a risky combination. Keep a human approving actions.

Why does the AI ignore my server? Check it's connected. Make the tool descriptions clearer. Ask explicitly: "Use the notes tool to..."

Local vs remote? Local (stdio) servers run on your computer and can reach your files. Remote servers run on the provider's infrastructure and usually sign in with OAuth.

Under the Hood: The Protocol

You don't need this chapter to *use* MCP, but it will make error messages and logs make sense. It also helps if you build servers.

IN THIS CHAPTER

- JSON-RPC messages
- The connection lifecycle
- Every capability, from both sides
- What a real tool call looks like

14.1 JSON-RPC in one paragraph

MCP messages are **JSON-RPC 2.0**. Each message is a small JSON object. A **request** has an `id`, a `method`, and `params`. A **response** has the same `id` and either a `result` or an `error`. A **notification** has a method but no `id`, and no reply is expected.

14.2 The lifecycle

How a connection starts depends on the protocol **version**. The current version, according to modelcontextprotocol.io/specification/versioning, is **2026-07-28**. Many apps and servers still speak the earlier **2025-11-25** version, and clients and servers may support several versions at once.

STEP	2025-11-25 (AND EARLIER)	2026-07-28 (CURRENT)
Agree on a version	<code>initialize</code> request, then an <code>initialized</code> notification	Every request names its version in <code>_meta</code> . Optional <code>server/discover</code> up front
Learn what's offered	<code>tools/list</code> , <code>resources/list</code> , <code>prompts/list</code>	Same
Use a tool	<code>tools/call</code>	Same
Change notices	<code>notifications/tools/list_changed</code> and others	Same, plus <code>subscriptions/listen</code> for a long-lived notification channel
Health check	<code>ping</code>	No <code>ping</code> in the current schema

If a server doesn't support the version a request asks for, it replies with an error listing the versions it does support, and the client can retry.

14.3 A real tool call

REQUEST (CLIENT → SERVER)

```
{ "jsonrpc": "2.0", "id": 7, "method": "tools/call",
  "params": { "name": "add_note", "arguments": { "text": "buy toner" } } }
```

RESPONSE (SERVER → CLIENT)

```
{ "jsonrpc": "2.0", "id": 7,
  "result": { "content": [ { "type": "text", "text": "Saved." } ], "isError": false } }
```

14.4 Capabilities

SIDE	CAPABILITY	WHAT IT LETS THAT SIDE DO
Server	Tools	Offer functions the AI can call
Server	Resources	Offer readable data, identified by URIs
Server	Prompts	Offer reusable prompt templates
Server	Logging	Send log messages to the client
Client	Roots	Tell the server which folders or URIs it may work in
Client	Sampling	Let the server ask the client's AI model to generate text (with user approval)
Client	Elicitation	Let the server ask the user a question mid-task, such as a confirmation

Not every app supports every capability. Check your app's documentation. The full specification is at modelcontextprotocol.io.

14.5 Authentication for remote servers

Remote servers usually use **OAuth**: the app opens a browser, you sign in with the service and approve access, and the app stores a token. That's why `claude mcp list` can show “Needs authentication.” Running `/mcp` and choosing the server starts the sign-in.

Ten Things to Try with MCP

Hands-on recipes, from easiest to most advanced. Each one lists the kind of server you need and a prompt to start with.

IN THIS CHAPTER

- Ten practical, beginner-friendly MCP projects

#	PROJECT	SERVER TYPE	STARTER PROMPT
1	Tidy a downloads folder	Filesystem (limited to that folder)	"List files older than 90 days in Downloads and propose folders to sort them into. Don't move anything yet."
2	Summarise web articles	Fetch	"Fetch this URL and give me a five-bullet summary with the key numbers."
3	Notion meeting notes	Notion (remote)	"Create a page in my Meetings database for today with an agenda template."
4	Explain a repository	Git or GitHub	"Summarise the last 20 commits in plain English."
5	Triage issues	GitHub (remote)	"List open issues labelled bug and group them by likely cause."
6	Calendar planning	Your calendar's connector	"Find two free hours next week for deep work and propose times."
7	Database questions	Read-only database server	"How many orders did we get each week last month?"
8	Error investigation	Sentry or another monitoring connector	"What are the top three new errors this week, and which release introduced them?"
9	Personal memory	Memory	"Remember that my preferred file format is PDF."
10	Your own tool	The server from Chapter 11	"Add a note: renew passport."

 TIP

Start every new server with **read-only** prompts. Once you trust what it's doing, move on to prompts that change things, and keep approvals turned on for those.

15.1 Safety checklist for each new server

- ☐ I know who publishes it and the source is reputable
- ☐ It has only the access it needs (folders, scopes, read-only where possible)
- ☐ Its secrets are in env vars or the app's secure storage
- ☐ I've tried it with read-only prompts first
- ☐ Approval prompts are on for anything that writes, sends, or deletes
- ☐ I know how to disconnect it (`/mcp` , or the app's connector settings)

Troubleshooting Deep Dive

Chapter 9 covered the basics. This chapter goes symptom by symptom, with the exact checks to run.

SYMPTOM	CHECK	FIX
Server never appears	Is the config file valid JSON? (A trailing comma breaks it)	Paste it into a JSON validator. Restart the app fully
"spawn npx ENOENT" or "command not found"	The app can't find Node or the command on its PATH	Install Node.js LTS. Use the full path to <code>npx</code> or <code>python</code> in <code>command</code>
Connects, then disconnects	The server crashed on start	Run the exact command in a terminal to see the error
Garbled or "invalid JSON-RPC" errors	The server prints normal output to stdout	Log to stderr only (Chapter 11)
"Needs authentication"	OAuth not done or expired	<code>/mcp</code> → choose the server → sign in, or <code>claude mcp login <name></code>
401 / 403 from a remote server	Wrong token, scopes, or account	Recreate the token with the right scopes. Check headers
Tools listed but never used	Vague tool descriptions, or too many tools	Ask explicitly ("use the X tool"). Disconnect unused servers
Everything is slow	Many servers loaded, using up context	<code>/context</code> in Claude Code. Disable servers you aren't using
Works in Claude Desktop, not Claude Code	Different config files	<code>claude mcp add-json <name> '<json>'</code> to copy the config across
Windows path errors	Backslashes in JSON	Use <code>\\</code> or forward slashes in paths
Project server "pending approval"	Safety feature for <code>.mcp.json</code>	Start <code>claude</code> in the project and approve it

UNIVERSAL DEBUGGING ROUTINE

1 Run the server command by hand in a terminal. Does it start without errors?

2 Open it in MCP Inspector:

```
npx @modelcontextprotocol/inspector <command> <args> .
```

3

Check the app's logs.

For Claude Code, use `claude --debug` or `--debug=mcp`. Other apps usually have an MCP or developer log in settings.

4

Simplify.

Remove every other server and test this one alone.

5

Update the app and the server package.

Protocol Method Reference

Every JSON-RPC method in the two protocol versions you're most likely to meet. The lists were generated from the official schema files (`schema/2025-11-25/schema.ts` and `schema/2026-07-28/schema.ts` in github.com/modelcontextprotocol/modelcontextprotocol) in October 2026.

IN THIS CHAPTER

- Every method in the current (2026-07-28) version, with its purpose
- What changed from 2025-11-25
- How to read a method name

17.1 Reading a method name

Methods are named `area/action`. `tools/call` means “in the tools area, call one.” Anything starting with `notifications/` is fire-and-forget, so no reply comes back. Who sends each message matters. Most requests go **client** → **server** (the app asks the server). `sampling/createMessage`, `elicitation/create`, and `roots/list` go **server** → **client** (the server asks the app).

17.2 Current version: 2026-07-28

METHOD	KIND	SCHEMA TYPE	PURPOSE
<code>completion/complete</code>	Request	CompleteRequest	A request from the client to the server, to ask for completion options.
<code>elicitation/create</code>	Request	ElicitRequest	A request from the server to elicit additional information from the user via the client.
<code>notifications/cancelled</code>	Notification	CancelledNotification	This notification is sent by the client to indicate that it is cancelling a request it previously issued. On stdio, the server also sends this notification, so

<code>notifications/message</code>	Notification	LoggingMessageNotification	JSONRPCNotification of a log message passed from server to client.
<code>notifications/progress</code>	Notification	ProgressNotification	An out-of-band notification used to inform the receiver of a progress update for a long-running request.
<code>notifications/prompts/list_changed</code>	Notification	PromptListChangedNotification	An optional notification from the server to the client, informing it that the list of prompts it offers has changed.
<code>notifications/resources/list_changed</code>	Notification	ResourceListChangedNotification	An optional notification from the server to the client, informing it that the list of resources it can read from has changed.
<code>notifications/resources/updated</code>	Notification	ResourceUpdatedNotification	A notification from the server to the client, informing it that a resource has changed and may need to be read again.
<code>notifications/subscriptions/acknowledged</code>	Notification	SubscriptionsAcknowledgedNotification	Sent by the server to acknowledge that a {.
<code>notifications/tools/list_changed</code>	Notification	ToolListChangedNotification	An optional notification from the server to the client, informing it that the list of tools it offers has changed.
<code>prompts/get</code>	Request	GetPromptRequest	Used by the client to get a prompt provided by the server.
<code>prompts/list</code>	Request	ListPromptsRequest	Sent from the client to request a list of prompts and prompt templates the server has.
<code>resources/list</code>	Request	ListResourcesRequest	Sent from the client to request a list of resources the server has.
<code>resources/read</code>	Request	ReadResourceRequestParams	Parameters for a <code>resources/read</code> request.

<code>resources/templates/list</code>	Request	ListResourceTemplatesRequest	Sent from the client to request a list of resource templates the server has.
<code>roots/list</code>	Request	ListRootsRequest	Sent from the server to request a list of root URIs from the client.
<code>sampling/createMessage</code>	Request	CreateMessageRequest	A request from the server to sample an LLM via the client.
<code>server/discover</code>	Request	DiscoverRequest	A request from the client asking the server to advertise its supported protocol versions, capabilities, and other metadata.
<code>subscriptions/listen</code>	Request	SubscriptionsListenRequest	Sent from the client to open a long-lived channel for receiving notifications outside the context of a specific request.
<code>tools/call</code>	Request	CallToolRequest	Used by the client to invoke a tool provided by the server.
<code>tools/list</code>	Request	ListToolsRequest	Sent from the client to request a list of tools the server has.

17.3 What changed from 2025-11-25

IN 2025-11-25	IN 2026-07-28
<code>initialize</code> + <code>notifications/initialized</code> handshake	Removed. Each request carries its version in <code>_meta</code> . <code>server/discover</code> is mandatory for servers
<code>ping</code>	Not in the current schema
<code>logging/setLevel</code>	Not in the current schema. Servers still send <code>notifications/message</code> log lines
<code>resources/subscribe</code> , <code>resources/unsubscribe</code>	Replaced by <code>subscriptions/listen</code> and <code>notifications/subscriptions/acknowledged</code>
<code>tasks/get</code> , <code>tasks/list</code> , <code>tasks/result</code> , <code>tasks/cancel</code> , <code>notifications/tasks/status</code>	Not in the current schema
<code>notifications/roots/list_changed</code> , <code>notifications/elicitation/complete</code>	Not in the current schema

NOTE

“Not in the current schema” means the method name doesn’t appear in the 2026-07-28 schema file. The specification’s deprecated-features registry explains migration paths. You only need this detail if you build servers or clients. The SDKs handle versions for you.

Projects: Build Three Useful Servers

Three more complete servers to build after Chapter 11. Each adds one new skill. All use the Python `MCPServer` API shown there.

IN THIS CHAPTER

- Project A: a read-only folder search server (safe file access)
- Project B: a weather server that calls a web API (HTTP and errors)
- Project C: a to-do server with a confirmation step (safe writes)
- Sharing your server with others

18.1 Project A: Folder search (read-only)

SEARCH_SERVER.PY

```
from pathlib import Path
from mcp.server.mcpserver import MCPServer

ROOT = Path.home() / "Documents"      # the ONLY folder it may read
mcp = MCPServer("doc-search")

def safe(p: str) -> Path:
    full = (ROOT / p).resolve()
    if not full.is_relative_to(ROOT.resolve()):
        raise ValueError("Path outside allowed folder")
    return full

@mcp.tool()
def find_files(word: str, limit: int = 20) -> list[str]:
    """Find text files under Documents whose name or content contains `word`."""
    hits = []
    for f in ROOT.rglob("*.txt"):
        try:
            if word.lower() in f.name.lower() or word.lower() in f.read_text(errors="ignore").lower():
                hits.append(str(f.relative_to(ROOT)))
        except OSError:
            continue
        if len(hits) >= limit:
            break
    return hits

@mcp.tool()
def read_file(path: str, max_chars: int = 5000) -> str:
    """Read a text file (relative to Documents), truncated to max_chars."""
    return safe(path).read_text(errors="ignore")[:max_chars]

if __name__ == "__main__":
    mcp.run()
```

What you learned: restrict paths with `resolve()` plus `is_relative_to()` so `../..` tricks can't escape the folder, and cap output size so you don't flood the AI's context.

18.2 Project B: Weather from a web API

This uses the free Open-Meteo API, which needs no key. Install `httpx` first: `pip install httpx`.

WEATHER_SERVER.PY

```
import httpx
from mcp.server.mcpserver import MCPServer

mcp = MCPServer("weather")

@mcp.tool()
async def forecast(latitude: float, longitude: float) -> str:
    """Today's max/min temperature (°C) for a location."""
    url = "https://api.open-meteo.com/v1/forecast"
    params = {"latitude": latitude, "longitude": longitude,
              "daily": "temperature_2m_max,temperature_2m_min", "timezone": "auto",
              "forecast_days": 1}
    try:
        async with httpx.AsyncClient(timeout=10) as c:
            r = await c.get(url, params=params)
            r.raise_for_status()
    except httpx.HTTPError as e:
        return f"Weather service error: {e}"
    d = r.json()["daily"]
    return f"Max {d['temperature_2m_max'][0]}°C, min {d['temperature_2m_min'][0]}°C"

if __name__ == "__main__":
    mcp.run()
```

What you learned: async tools, timeouts, and returning a readable error instead of crashing. Check the API's current docs at open-meteo.com if parameters change.

18.3 Project C: To-dos with a confirmation step

TODO_SERVER.PY

```
import json
from pathlib import Path
from mcp.server.mcpserver import MCPServer

DB = Path.home() / "mcp-todos.json"
mcp = MCPServer("todos")

def load(): return json.loads(DB.read_text()) if DB.exists() else []
def save(x): DB.write_text(json.dumps(x, indent=2))

@mcp.tool()
def list_todos() -> list[dict]:
    """List all to-dos."""
    return load()

@mcp.tool()
def add_todo(title: str) -> str:
    """Add a to-do."""
    items = load(); items.append({"title": title, "done": False}); save(items)
    return f"Added: {title}"

@mcp.tool()
def delete_todo(index: int, confirm: bool = False) -> str:
    """Delete a to-do by its position. Requires confirm=true; first call without it to preview."""
    items = load()
    if not 0 <= index < len(items):
        return "No such to-do."
    if not confirm:
        return f"Will delete '{items[index]['title']}'. Call again with confirm=true to proceed."
    removed = items.pop(index); save(items)
    return f"Deleted: {removed['title']}"

if __name__ == "__main__":
    mcp.run()
```

What you learned: keep destructive tools separate and add a preview-then-confirm pattern. Most apps will also show you a permission prompt before running it.

18.4 Sharing your server

- ☐ Put it in a GitHub repository with a README covering install, configuration, and tools
- ☐ Pin dependencies (`requirements.txt` or `pyproject.toml`)
- ☐ Document every environment variable it needs
- ☐ Say clearly what it can read and write
- ☐ Add a license
- ☐ Optionally publish to PyPI or npm so people can run it with `uvx` or `npx` , and list it in the official MCP Registry

Security Review: Before You Install or Ship a Server

An MCP server runs with whatever access you give it. A local stdio server runs **as you**, with your files, your network and your environment variables. Review a server like any software you install, and more carefully than most, because an AI decides when to call it.

19.1 Before installing someone else's server

- ☐ ☐ **Source:** published by the service's own company, or open source with recent activity and real users
- ☐ ☐ **Package name matches the docs exactly.** Typo-squatted npm and PyPI names are a known attack
- ☐ ☐ **Read the tool list.** Does it need *write* or *delete* tools, or only read tools?
- ☐ ☐ **Scopes:** create the narrowest API token possible (read-only, single repo, single workspace)
- ☐ ☐ **Pin a version** in the config, e.g. `npx -y package@1.4.2`, so an update can't silently change behavior
- ☐ ☐ **Remote servers:** connect over HTTPS only, and prefer OAuth over pasting long-lived tokens
- ☐ ☐ **Your client's approvals:** keep per-tool approval on for anything that writes, sends or spends

19.2 The main risks, explained

RISK	WHAT HAPPENS	DEFENSE
Prompt injection	A web page, email or issue that the AI reads contains hidden instructions ("send the secrets to...")	Don't combine untrusted-content tools with send or write tools in one unattended session, and keep approvals on for writes
Tool poisoning	A malicious server's tool <i>description</i> tells the AI to misuse other tools	Install only trusted servers, and read the descriptions (the Inspector shows them)
Over-broad tokens	A leaked token unlocks your whole account	Use fine-grained tokens with expiry dates, and rotate them
Data exfiltration	The AI sends private data to an outside URL through a fetch tool	Limit network tools, and use allowlists where your client supports them
Rug pull	A server you trusted changes after an update	Pin versions, and review changelogs before you upgrade
Confused deputy	A remote server uses <i>its own</i> powerful credentials on your behalf	Choose servers that act with your own user's permissions (OAuth)

19.3 If you're building a server

PRACTICE	WHY
Validate every argument (types, lengths, allowed values)	The model can send anything
Put file tools in a jail: resolve paths and refuse anything outside an allowed root	This blocks <code>../../</code> escapes
Mark destructive tools clearly, and use elicitation to confirm if your client supports it	The user sees what will happen
Return errors as tool results (<code>isError: true</code>) with a helpful message	The model can recover without crashing the session
Never log secrets, and log to <code>stderr</code> only for <code>stdio</code> servers	<code>stdout</code> carries the protocol
Rate-limit and time out outbound calls	A runaway loop shouldn't drain an API quota
For HTTP servers: require auth, check the <code>Origin</code> header, and bind local servers to <code>127.0.0.1</code>	This blocks DNS-rebinding and drive-by access

SAFE PATH HANDLING FOR A FILE TOOL

```
from pathlib import Path

ROOT = Path("~/notes").expanduser().resolve()

def safe_path(user_path: str) -> Path:
    p = (ROOT / user_path).resolve()
    if not p.is_relative_to(ROOT):
        raise ValueError("Path is outside the allowed folder")
    return p
```

JSON-RPC Error Codes and What They Mean

MCP messages are JSON-RPC 2.0. When something fails at the protocol level, the response holds an `error` object with a numeric `code`. These are the standard codes from the JSON-RPC 2.0 specification:

CODE	NAME	USUAL CAUSE IN MCP	FIX
-32700	Parse error	The server or client received invalid JSON, often because a stdio server printed text to stdout	Send all logs and prints to stderr
-32600	Invalid request	A message is missing <code>jsonrpc: "2.0"</code> or <code>method</code> , or has the wrong shape	Use an official SDK instead of hand-writing messages
-32601	Method not found	Calling a method the other side doesn't support, or one from a different protocol version	Check the capabilities and protocol version the server advertises
-32602	Invalid params	Arguments don't match the tool's input schema, or an unknown tool name was used	Compare the call to the <code>inputSchema</code> in the tools list
-32603	Internal error	The server crashed while handling the request	Read the server's stderr log
-32000 to -32099	Server error (reserved range)	Implementation-defined errors	Check the server's docs

NOTE

A **tool that ran but failed**, such as “city not found”, should not be a JSON-RPC error. It’s a normal result with `isError: true`, so the AI can read the message and try again. Protocol errors are for broken messages, not for bad outcomes.

Quick Reference

Roles: Host = your AI app · Client = one connection · Server = adds tools/resources/prompts

Remote server: paste URL → sign in (OAuth) · Claude: [Settings](#) > [Connectors](#) > [Add custom connector](#)

Local server (Claude Desktop): menu bar Settings → Developer → Edit Config → add under "mcpServers" → fully restart

Claude Code: `claude mcp add --transport http <name> <url>` · `claude mcp add <name> --<command>` · `/mcp`

Cursor: `~/.cursor/mcp.json` → "mcpServers" · **VS Code:** `.vscode/mcp.json` → "servers"

Test a server: `npx @modelcontextprotocol/inspector <command>`

Safety: official servers · least privilege · approvals on for write/send/delete · remove unused connectors

Glossary

API key	A secret string that identifies a program to a service.
Client	The component inside a host app that maintains one connection to one MCP server.
Connector	Claude's name for an MCP server connected to your account.
Elicitation	When an MCP server asks the user for extra information during a task.
Host	The AI application you use, such as Claude Desktop, Cursor, or ChatGPT, which connects to MCP servers.
JSON	A text format for structured settings, using braces, brackets, quotes, and commas.
JSON-RPC	The simple message format MCP uses to request actions and return results.
Local server	An MCP server that runs as a program on your own computer.
MCP	Model Context Protocol: an open standard for connecting AI applications to tools and data.
MCP Inspector	A free developer tool for testing MCP servers.
OAuth	A secure "sign in with..." standard that grants an app limited access without sharing your password.
Prompt (primitive)	A reusable template an MCP server offers, often shown as a slash command.
Prompt injection	Hidden instructions inside content an AI reads, designed to hijack its behavior.
Remote server	An MCP server hosted on the internet and reached by a web address.
Resource	Information an MCP server exposes for the AI to read, such as a file or a schema.

SDK	Software development kit: a library that makes building something, such as an MCP server, easier.
stdio	Standard input/output: the transport local MCP servers use to talk to their host.
Streamable HTTP	The transport remote MCP servers use over the web.
Tool	An action an MCP server lets the AI perform, such as creating an issue or sending a message.
Tool poisoning	A malicious or manipulative instruction hidden in a server's tool descriptions.
Transport	The channel messages travel over between an MCP client and server.
uv	A fast Python tool for installing and running Python programs, used by many MCP servers.