Model Context Protocol (MCP)
“It’s the USB-C for AI”

By Jack Bradecamp, Pablo Ibarz, Gigi Kuffa,
Michaelaustin White, and Alex Zhou

Introduction

Context

Reality:

- SaaS and Data Silos taciturn
- AI chatty

Anthropic’s idea:

- Design a protocol where AIs can coexist with other services smoothly.

Background

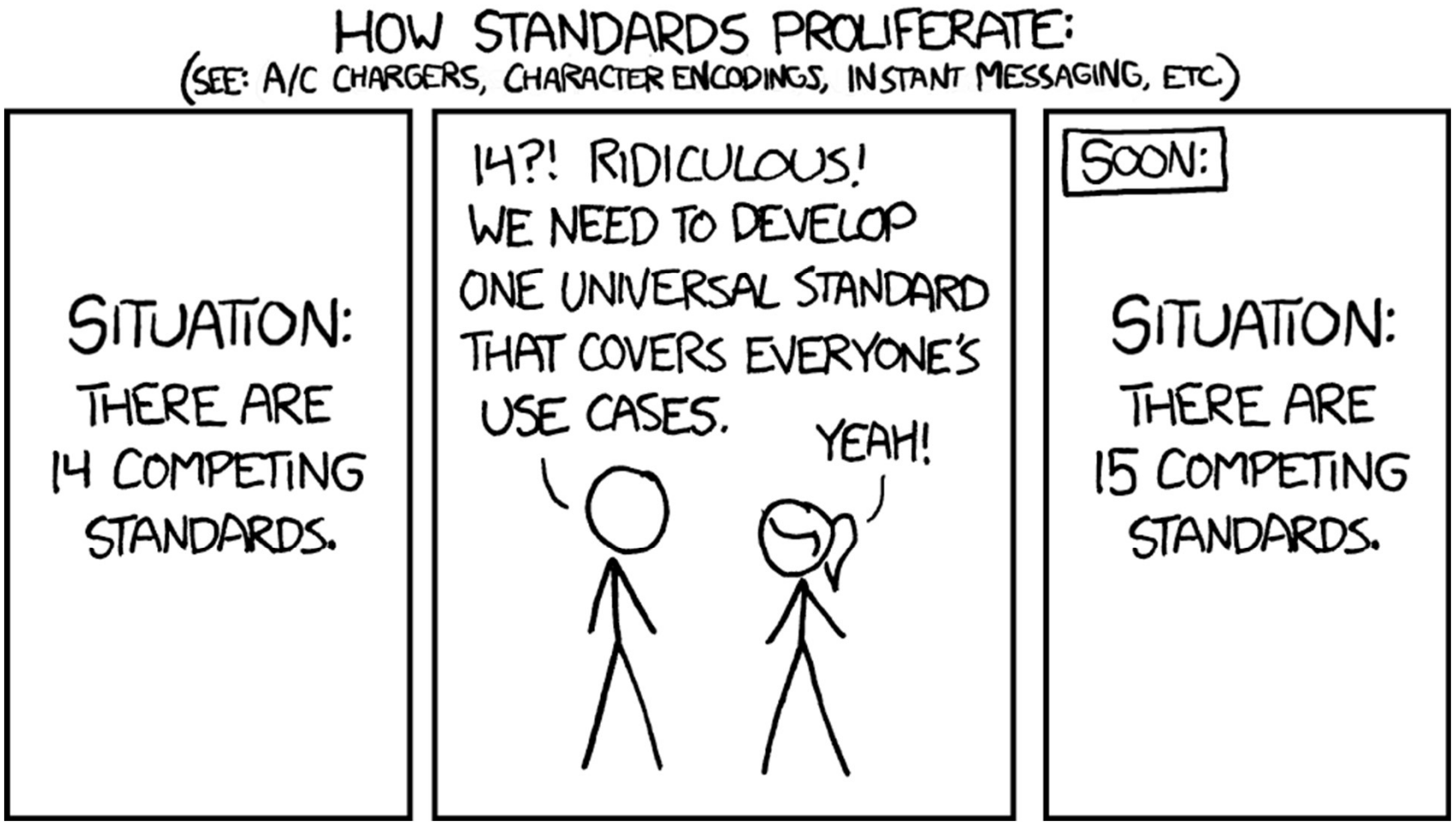
Challenges

- Dynamic resource discovery
- Rich context exchange
- Bidirectional communication
- Secure sandboxing
- “AI Integration Paradox”

Importance

Reliability

- If the standard is accepted, more AI systems can rely on the format of incoming context.
- More standardized AI apps means a healthier developer and consumer ecosystem for working with AI
- One standard instead of multiple


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION:
THERE ARE
14 COMPETING
STANDARDS.
14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.
YEAH!
SOON:
SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

Core Content

Architecture

Note: MCP focuses solely on the protocol for context exchange; it doesn't dictate how AI applications use LLMs or manage the provided context.

The **Model Context Protocol** includes the following:

MCP Specification: A specification of MCP that outlines the implementation requirements for clients and servers.

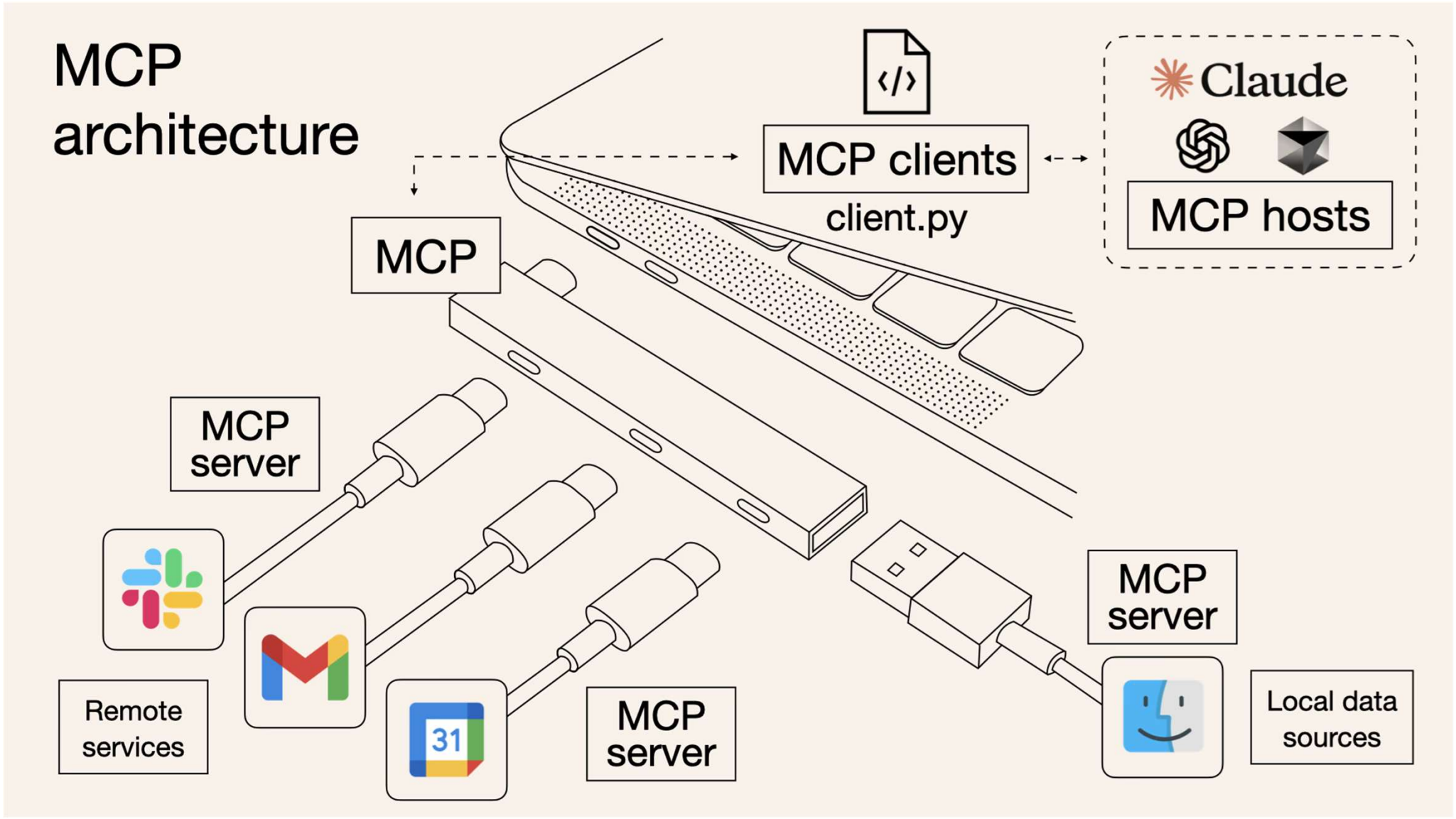
MCP SDKs: SDKs for different programming languages that implement MCP.

MCP Development Tools: Tools for developing MCP servers and clients, including the MCP Inspector.

MCP Reference Server Implementations: Reference implementations of MCP servers.

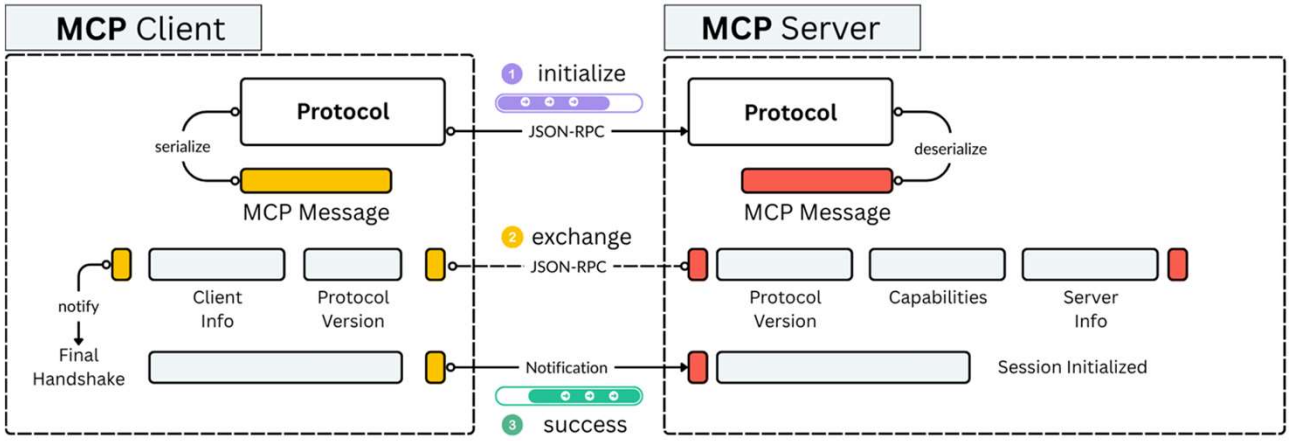
Architecture

- MCP follows a **client-server architecture** where an MCP host (an AI application or runtime) establishes connections to one or more MCP servers.
- To accomplish this, the **MCP host** creates one **MCP client** for each **MCP server**.
  - **MCP client:** a component that maintains a dedicated connection to and obtain context from an MCP server
  - **MCP server:** a program that provides context to MCP clients, can be local or remote/web-based


MCP architecture
MCP clients
client.py
Claude
MCP hosts
MCP
MCP server
MCP server
MCP server
Remote services
Local data sources

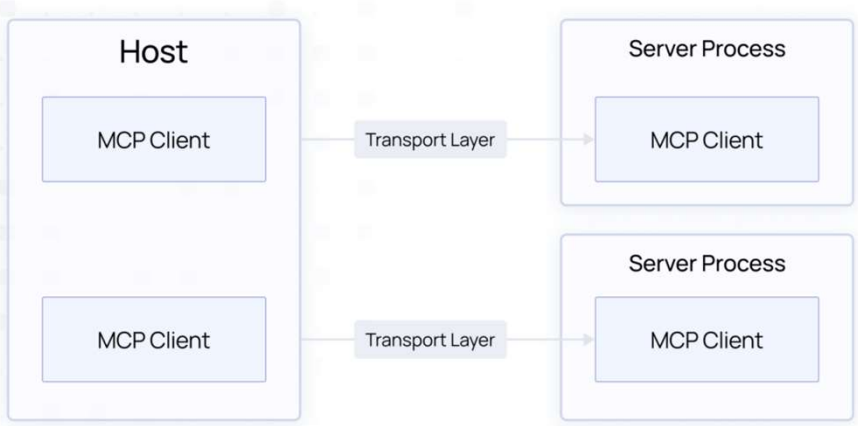
Architecture

- MCP consists of two layers: an inner **data layer** and outer transport layer.
  - **Data layer:** Defines the JSON-RPC 2.0 based exchange protocol for client-server communication for message structure and semantics.



Architecture

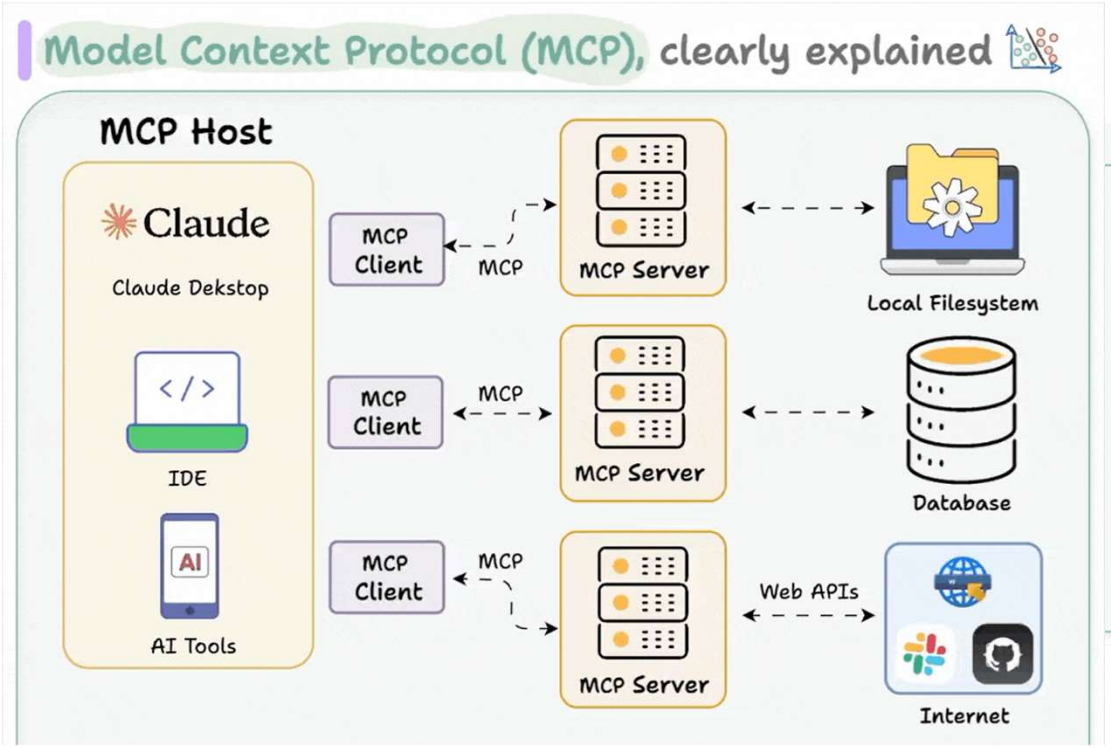
- MCP consists of two layers: an inner data layer and outer **transport layer**.
  - **Transport layer:** Manages the communication mechanisms and channels that authenticate and enable data exchange between clients and servers.

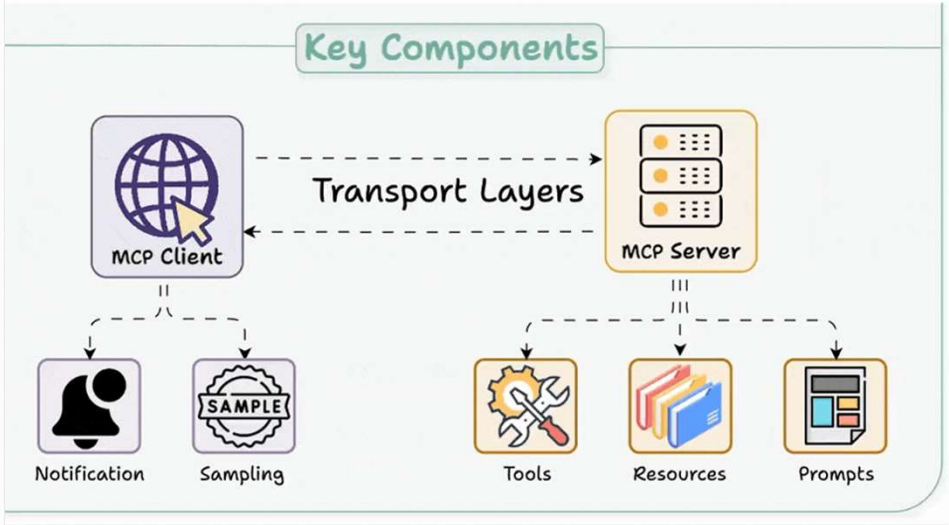


Architecture

- MCP supports two **transport mechanisms:**
  - **STDIO transport:** Uses standard input/output streams. Optimal performance with no network overhead.
  - **Streamable HTTP transport:** Uses HTTP POST for client-to-server messages with optional Server-Sent Events for streaming capabilities. Supports standard HTTP authentication methods including bearer tokens, API keys, and custom headers.

Architecture





Main Modules

mcp (library)

- suite of tools for building servers and clients
- connecting servers/clients with other servers/clients
- **mcp.server**
- **mcp.client**

mcp inspector

- debugger for mcp servers

Workflows

Once you have a server configured-

- Load the server
- Connect to a client

If you want to use an existing server (usually remote)

- Load a client (Claude Desktop)
- Connect

Setup & Configuration

Three ways:

1. **For Client Developers:** Create MCP Client
2. **For Server Developers:** Build MCP Server
3. **Connect to Local/Remote MCP Servers:** Connect with Claude Desktop

For Client Developers

Python dependencies (server)

- mcp[cli]
- httpx

client:

- anthropic
- dotenv

```
python3 -m venv .venv
source .venv/bin/activate

pip install --upgrade pip
pip install "mcp[cli]" httpx
```

For Client Developers

Connecting to servers

- Connection over Streamable HTTP or asyncio core library
- Claude Desktop as the client
  - Connect to local server using filesystem server
  - Connect to remote server in custom connector menu

For Server Developers

- **To build an MCP server:**
    1. Set up your Python project and environment.
    2. Create a new directory.
    3. Create and active a virtual environment.
    4. Install dependencies.
    5. Create a server file (.py).

For Server Developers

- Import all the needed packages.

```
from typing import Any
import httpx
from mcp.server.fastmcp import FastMCP
```

  - The FastMCP class uses Python type hints and docstrings to automatically generate tool definitions, making it easy to create and maintain MCP tools.

For Server Developers

- Set up the instance.

```
# Initialize FastMCP server
mcp = FastMCP("weather") # Use server file name
```

- Declare any constants here, e.g.

```
# Constants
NWS_API_BASE = "https://api.weather.gov"
USER_AGENT = "weather-app/1.0"
```

For Server Developers

- Add helper functions for querying and formatting the data from the API you're using, e.g.

```python
async def make_nws_request(url: str) -> dict[str, Any] | None:
    """Make a request to the NWS API with proper error
    handling."""
```

```python
def format_alert(feature: dict) -> str:
    """Format an alert feature into a readable string."""
```

For Server Developers

- Implement the tool execution handler, which is responsible for actually executing the logic of each tool, by decorating each function with **`@mcp.tool()`**.

```python
@mcp.tool()
async def make_nws_request(url: str) -> dict[str, Any] | None:
    """Make a request to the NWS API with proper error
    handling."""
```

For Server Developers

- Finally, let's initialize and run the server to confirm everything's working:

```
if __name__ == "__main__":
    # Initialize and run the server
    mcp.run(transport='stdio')
```

- Test your server from an existing MCP host, like Claude for Desktop.

Connect with Claude Desktop

Step 1. Download the latest version of Claude for Desktop for macOS or Windows.

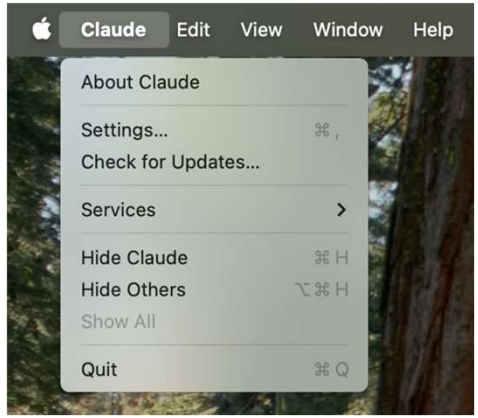
Note: MCP servers run locally on your computer for security. The web version can't access your files (and that's a good thing!).

(Check that you have Node.js installed: open the command line on your computer and run node --version).

Connect with Claude Desktop

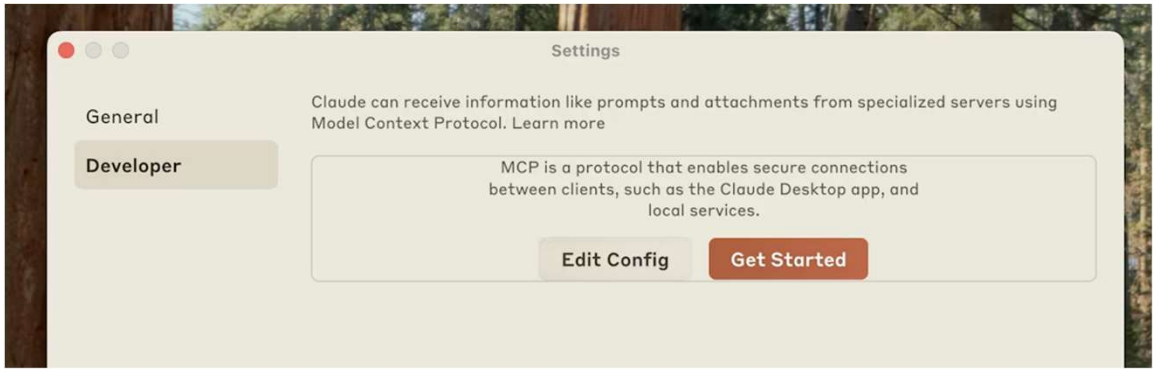
Step 2. Configure MCP

Open Developer Settings. This will create a configuration file that tells Claude which “apps” (MCP servers) it can use. Edit this file.



macOS Windows

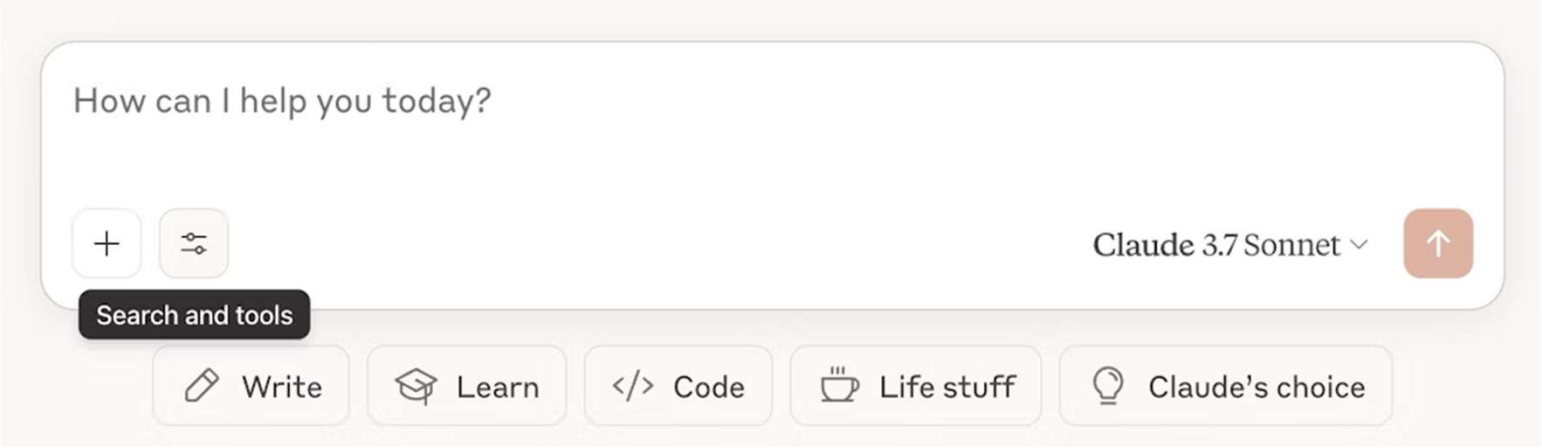
```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "/Users/username/Desktop",
        "/Users/username/Downloads"
      ]
    }
  }
}
```



Connect with Claude Desktop

Step 3. Restart and Test

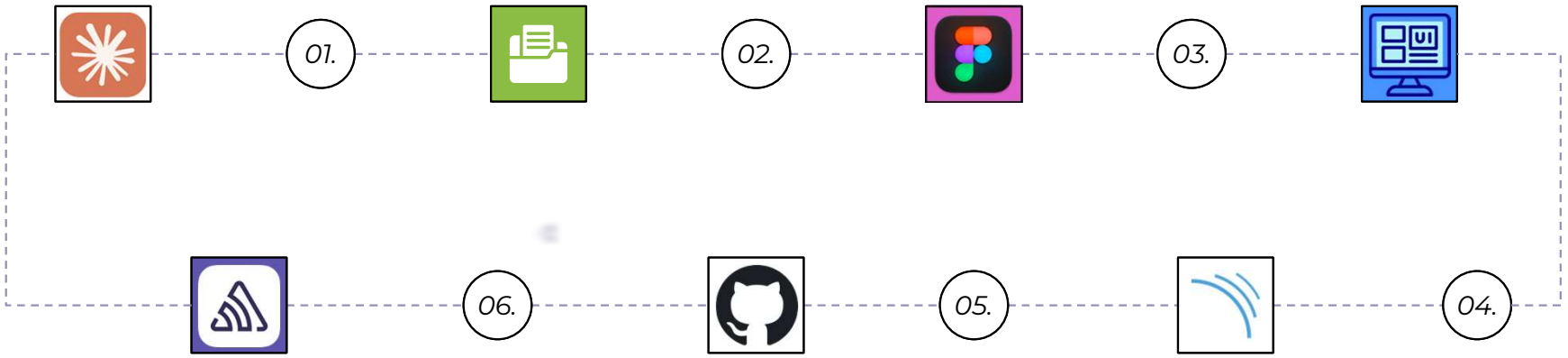
Restart Claude for Desktop after updating your configuration file. You should now see a icon on the bottom-right corner of the input box.



Workflows

Example:

Claude Desktop



Tool Discovery

Host discovers tools

Figma MCP Server

Pull design assets

UI Scaffolding Tool

Generate component structure

SonarQube MCP Server

Run quality gates until pass

PR

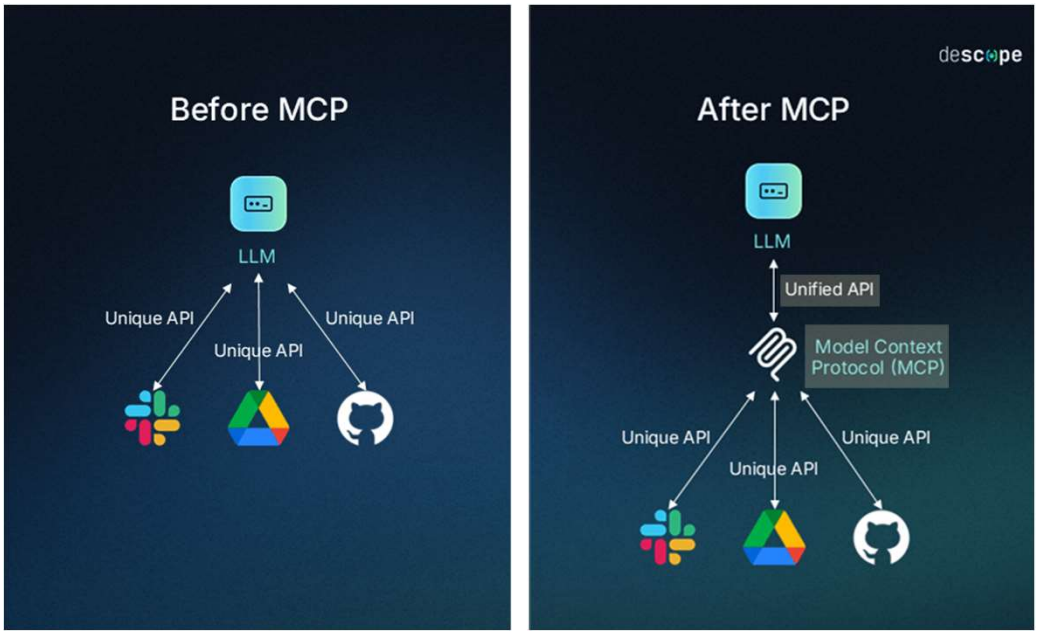
Open or update GitHub or Jira issue

Sentry MCP Server

Confirm fix/error resolved

APIs

- **MCP itself is a protocol/standard, not a specific API.**
- Developers build MCP servers that expose an API for AI agents to interact with by retrieving data or initiating workflows, which often wrap existing APIs and describe them in an LLM-friendly format.



Documentation

- **Anthropic** provides:
  - Official open source project on GitHub
    - https://github.com/modelcontextprotocol
  - Official docs, includes guide and overview on how to get started writing or using an MCP server
    - https://modelcontextprotocol.io/
  - Official specs on latest protocol
    - https://modelcontextprotocol.io/specification/2025-06-18

Resources

- **Official MCP Servers:**
  - **Figma for design-to-code workflows, component/asset extraction.**
  - **SonarQube for static analysis, security scanning, quality gates.**
  - **Sentry for error triage, debugging.**
  - **GitHub** for repositories, issues, PR automation.
  - **Google Cloud** for BigQuery, storage, logging, ML pipelines.
  - **Azure** (Microsoft Learn) for DevOps pipelines, cognitive services.
  - **Stripe** for billing, subscription and transaction analysis.
  - **Slack** for message ingestion, team triage, productivity workflows.
  - etc.

Resources

- microsoft/mcp-for-beginners: **open-source curriculum** designed for developers that introduces MCP fundamentals through real-world, cross-language examples
- wong2/awesome-mcp-servers (mcpservers.org): **curated list of production-ready and experimental community MCP servers** that extend AI capabilities
- jlowin/fastmcp: **Python framework** for building applications that allow large language models (LLMs) to create Model Context Protocol (MCP) servers and securely and reliably interact with external data and tools

Limitations

- **Security vulnerabilities:** poorly configured implementations (or early adopters) may have gaps like non-standard authentication or input validation and credential leakage, which exposes them to session hijacking, tool poisoning, token theft, and prompt injection attacks
- **Debugging without observability:** Debugging and monitoring interactions between clients, tools, and external systems can be difficult due to a lack of built-in observability features to monitor complex workflows and decentralized components

Limitations

(cont.)

- **Cost and performance:** API requests for historical data can be slow and **expensive, eating up context windows and increasing costs**
- **Tool management limits:** agents may only be able to effectively manage around 40 tools, and inefficient tools may hinder use

Demonstration