

# Introducing MCP

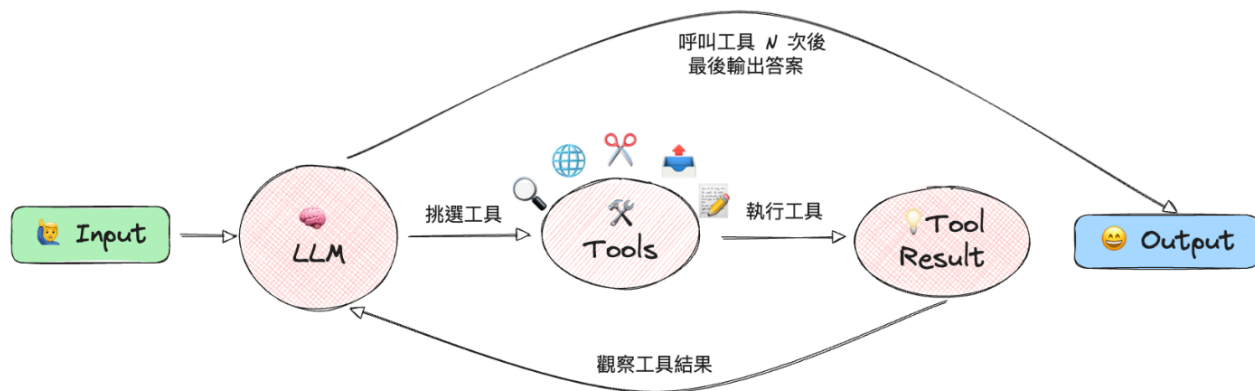
**Model Context Protocol**

標準化的 AI 工具協議

# AI Agent 是什麼？

AI Agent = LLM + 工具 + 自主決策

- AI 需要透過**工具**與外部世界互動（資料庫、網路等）
- LLM 會根據當前狀態**自主決定**流程與使用的工具
- **持續循環**直到完成任務



# Introducing MCP

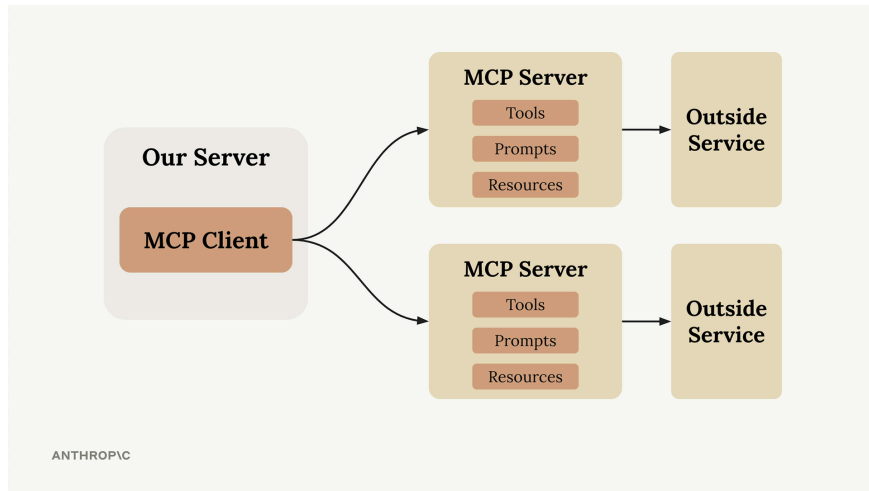
**MCP(Model Context Protocol)** 是一種標準化協議，定義了：

- AI 如何使用工具的標準方式
- AI Client 如何與 Tool Server 進行溝通

# Client-Server 架構

MCP 採用 Client-Server 架構：

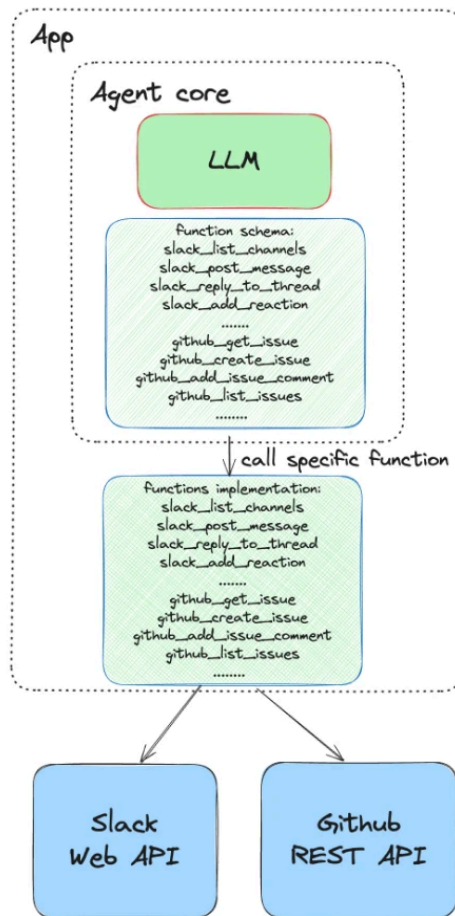
- **MCP Server** 提供給 AI 使用的具體工具或功能
- **MCP Client** 連接 Server 並管理 Context
- **MCP Host** AI 應用程式 ( Cursor 、 Claude Desktop )



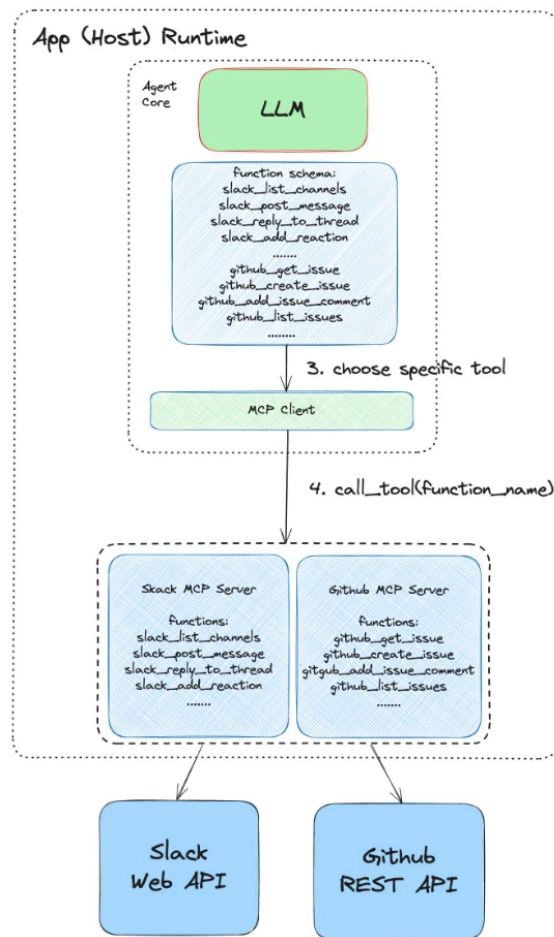
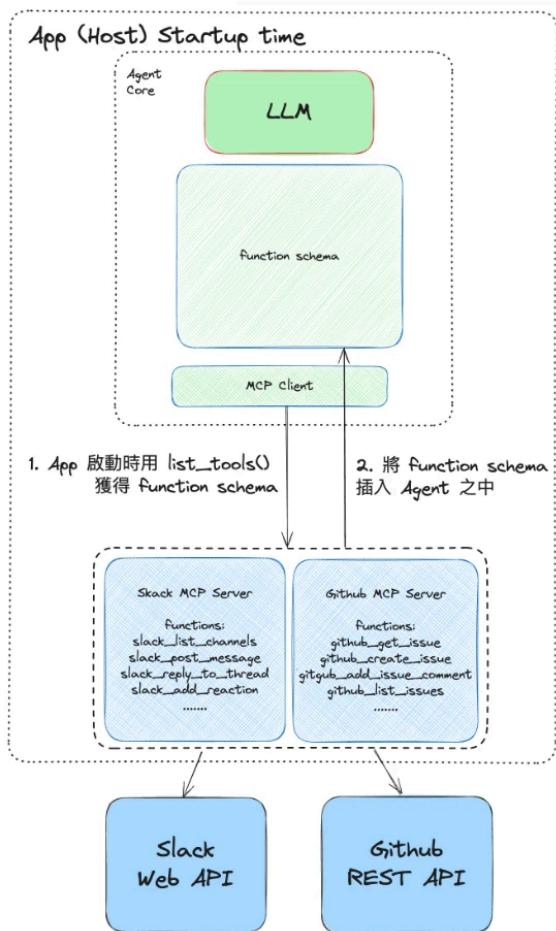
# MCP 的目的：重複使用現有工具

以開發 GITHUB AGENT 為例

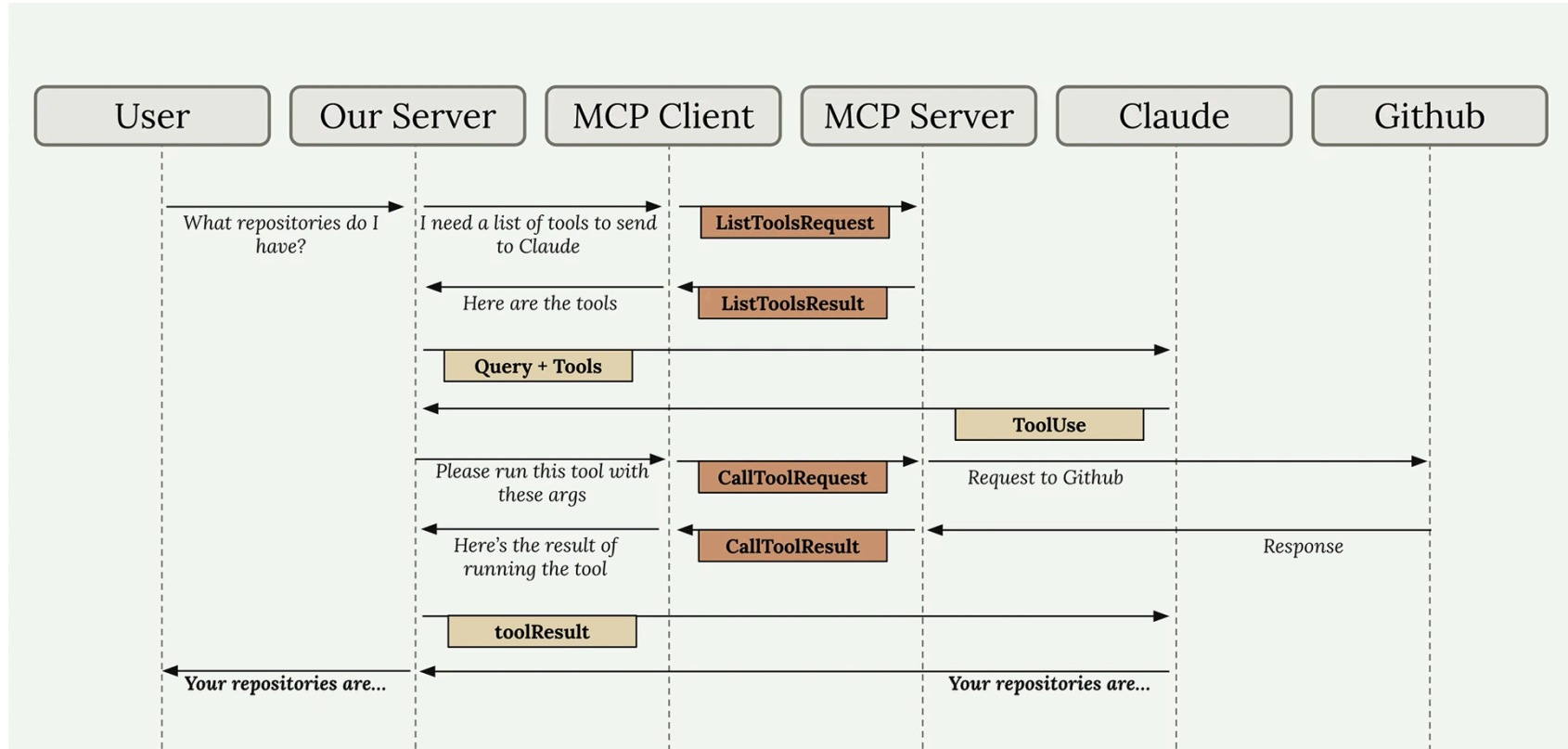
# 使用 MCP 之前



# 使用 MCP 之後



# MCP Flow





# MCP Transports

## 支援本地/遠端傳輸方式

1. STDIO Transport
2. HTTP Transport

# STDIO Transport

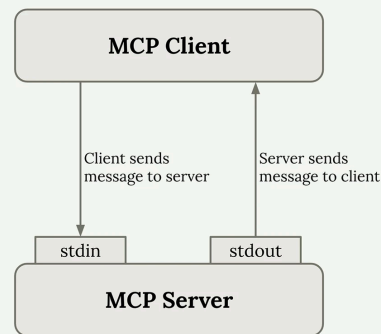
## 本地通訊方式

- 使用 **subprocess** 在本地執行 MCP Server
- 在同一台機器上 Client 與 Server 通訊
- 支援**雙向通訊**

### 'Stdio' transport

- Client launches the MCP server as a subprocess
- Client sends messages to the MCP Server using the server's 'stdin'
- Server responds by writing to 'stdout'
- **Critical:** either the server or the client can send a message at any time!
- **Only appropriate when the client and server are running on the same machine**

ANTHROPIC



# HTTP Transport

## 遠端通訊方式

1. Stateless HTTP
2. SSE

❗ 標準 HTTP 的限制：

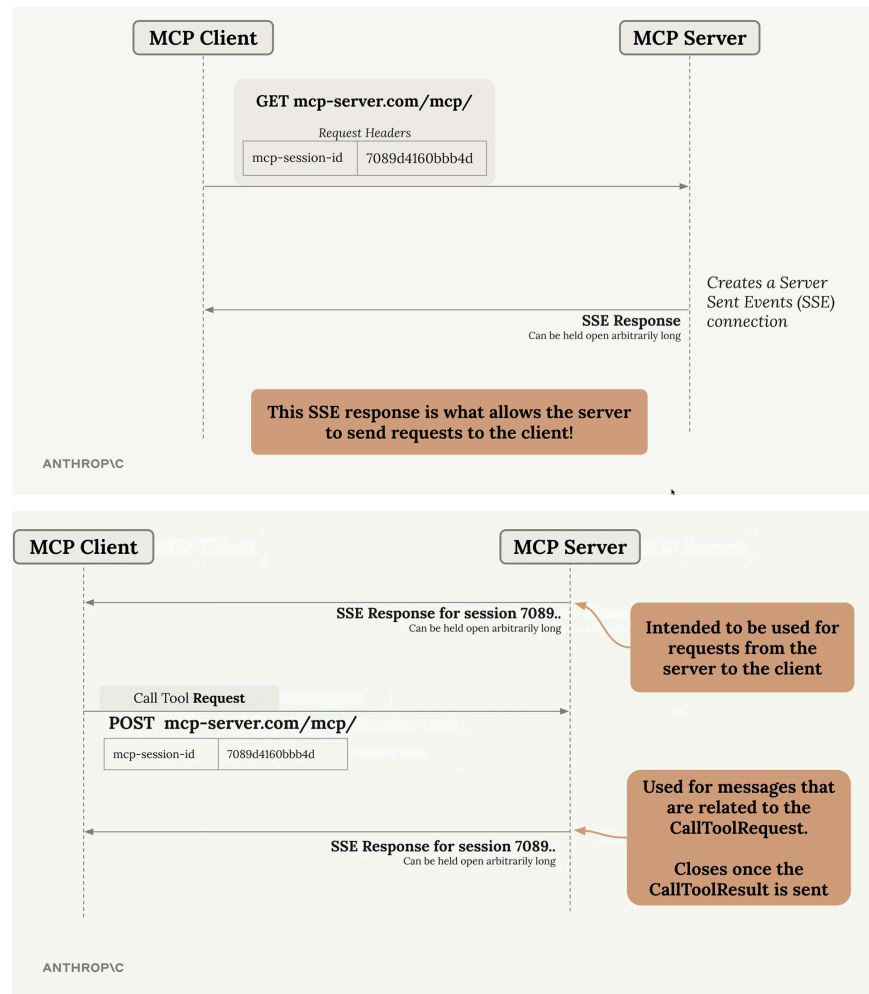
- 無法讓 Server 主動向 Client 發起請求
- Server 端的通知或 Log 功能會受到限制

**解決方法：** 使用 Streamable HTTP

# Streamable HTTP (SSE)

- Client 與 Server 初始化後，透過 **Server-Sent Events (SSE)** 建立持久連接
- 支援 Server 主動向 Client 發送通知

設定：`stateless_http=False`，  
`json_response=False`



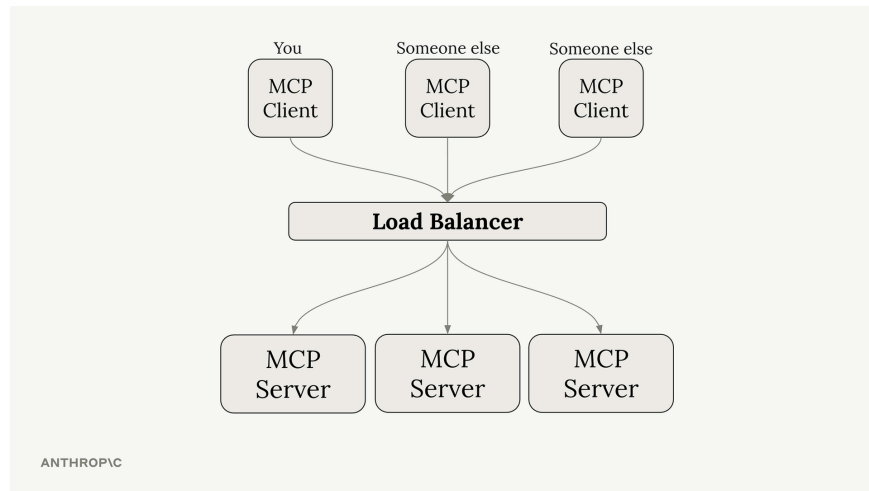
# Stateless HTTP

**適用場景：** 需要 **load balancer** 的環境

**缺點：** 無法支援 Server 主動通知功能

需要在**擴展性**與**功能性**之間做出取捨

**設定：** `stateless_http=True`



# MCP Example

## Client :

```
1  from mcp.client import MCPClient
2
3  # 透過 subprocess 建立 MCPClient
4  client = MCPClient(command=["python", "server.py"])
5
6  # use tool
7  result = client.call_tool("add", {"a": 3, "b": 4})
8
9  print("3 + 4 =", result)
```

## Server :

```
1  from mcp.server.fastmcp import FastMCP
2
3  # 建立 MCP Server
4  mcp = FastMCP(name="AdditionServer")
5
6  @mcp.tool()
7  def add(a: int, b: int) → int:
8      """加法工具：回傳 a + b 的結果"""
9      return a + b
10
11  if __name__ == "__main__":
12      mcp.run(transport="stdio")
```

# 好用的 MCP Server

- context7

## BEFORE

1. 無法獲得最新的 Document
2. 可能出現幻覺

Context7

H 你知道tailwind 4.1中noscript怎麼使用嗎?

我的知识截止到2025年1月，当时Tailwind CSS 4.0还在开发中，4.1版本可能是在我的知识截止日期之后发布的。因此我无法提供关于Tailwind 4.1中 `noscript` 具体用法的准确信息。

不过，基于Tailwind的一般发展方向，我可以分享一些可能的使用场景：

传统的`noscript`用法：

```
html
<noscript>
  <div class="p-4 bg-yellow-100 text-yellow-800 rounded">
    这个网站需要JavaScript才能正常工作，请启用JavaScript。
  </div>
</noscript>
```

可能的条件样式（如果Tailwind 4.1支持）：

- 可能会有类似 `noscript:hidden` 或 `js:block` 这样的修饰符
- 用于在JavaScript禁用时显示替代内容

要获取Tailwind 4.1中 `noscript` 的准确用法，建议你：

1. 查看Tailwind CSS官方文档
2. 查看Tailwind 4.1的发布说明
3. 在Tailwind的GitHub仓库中搜索相关信息

如果你有具体的使用场景，我也可以帮你思考可能的实现方案，即使我不了解最新的语法。

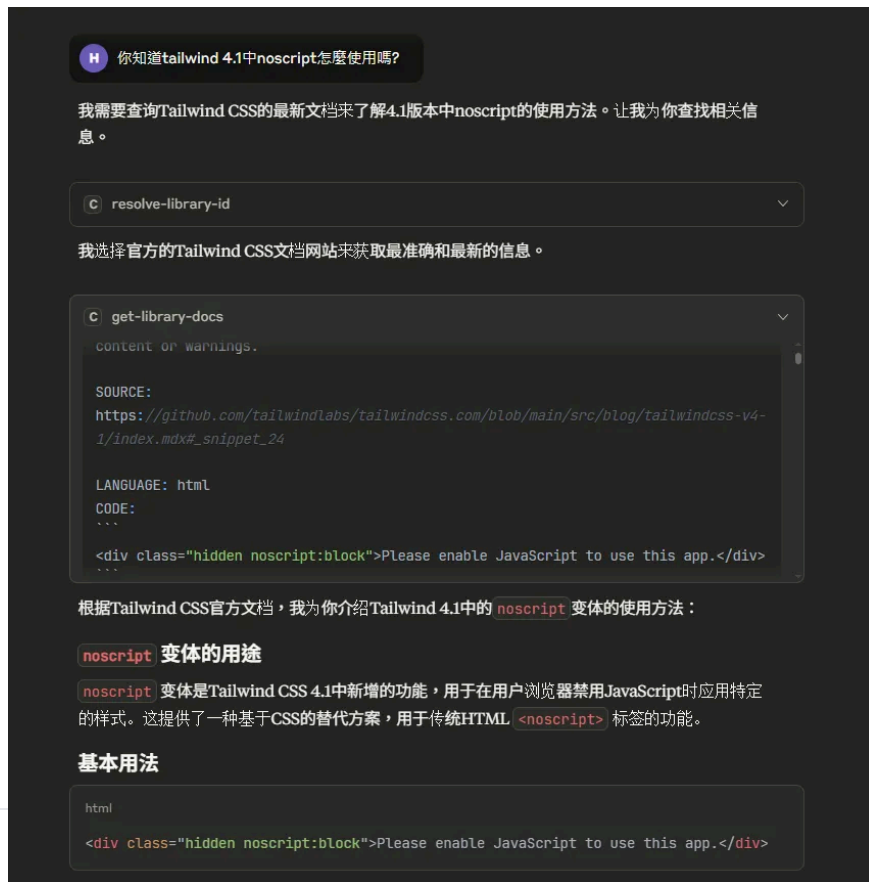
# 好用的 MCP Server

- context7

AFTER

1. 搜尋最新的 Document
2. 提供原始範例

Context7





# 參考資料

- Anthropic MCP 介紹課程
- MCP 官方文件 - 入門指南
- MCP 官方文件 - Server 快速開始
- ihower MCP