Introduction to Model Context Protocol (MCP)

Javier Suárez Ruiz

Software Engineer at Microsoft

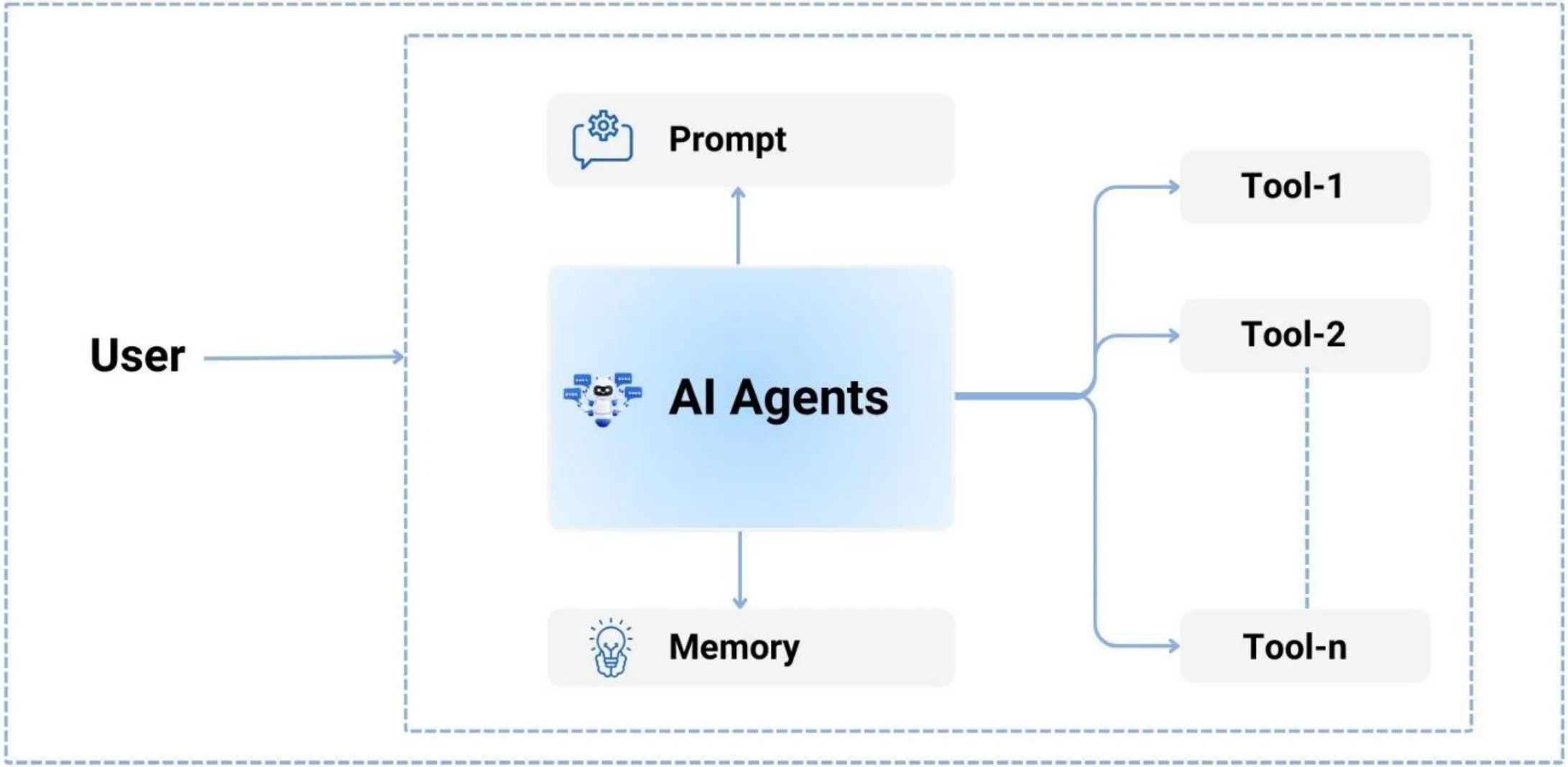
- Email: javiersuarezruiz@hotmail.com
- Twitter: @jsuarezruiz

The agenda

1. **Why?** – The Need for MCP
2. **What?** – Understanding MCP
3. **How?** Using MCP

Why?
The Need for MCP

Typical AI Application with Single-Agent Architecture



Challenges with Current AI Applications

- GitHub offers features for handling code, issues, pull requests, and more.
- Their goal is to extend these capabilities by integrating with various AI-powered IDEs, such as VS Code, Cursor, Windsurf, Zed, Cline, and others.
- To achieve this, they must develop dedicated GitHub integration tools tailored to each individual IDE, implementing them step by step for compatibility.

Pain Points for Developers

For Tool developers:

- If GitHub plans to integrate with the top 100 AI applications, they must create and implement integrations individually for each AI application, addressing its specific requirements and features.

For AI application developers:

- If Rider seeks to integrate with GitHub's VS Code GitHub Copilot tool, they wouldn't be able to reuse it directly. Instead, they'd have to develop a new integration tailored to their platform from scratch.

The Demand for Standardization

Need for a universal protocol to streamline AI integrations:

- Reduce fragmentation
- Promote interoperability

What?
Understanding MCP

What is the Model Context Protocol (MCP)?

MCP is an open protocol that enables seamless integration between **LLM applications** and your **tools & data sources**.

APIs

Standardize how **web applications** interact with the **backend**:

- Servers
- Databases
- Services

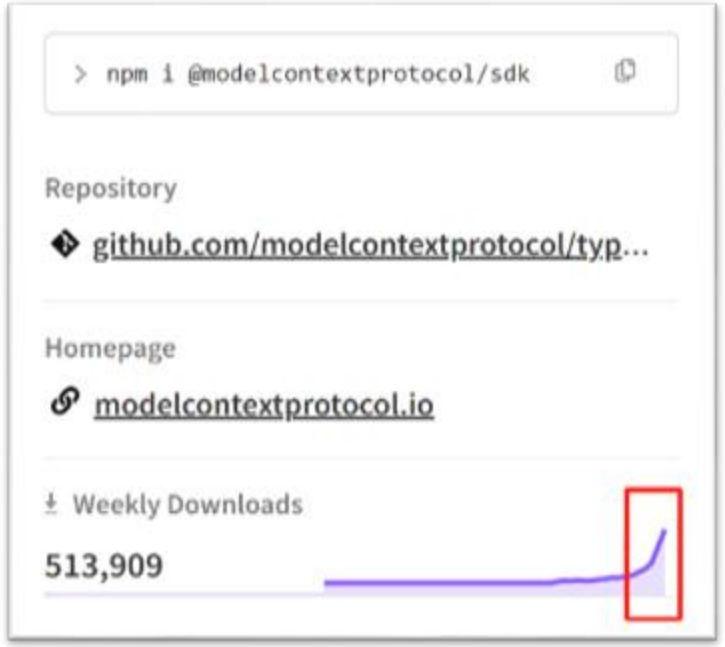
MCP

Standardizes how **AI applications** interact with **external systems**:

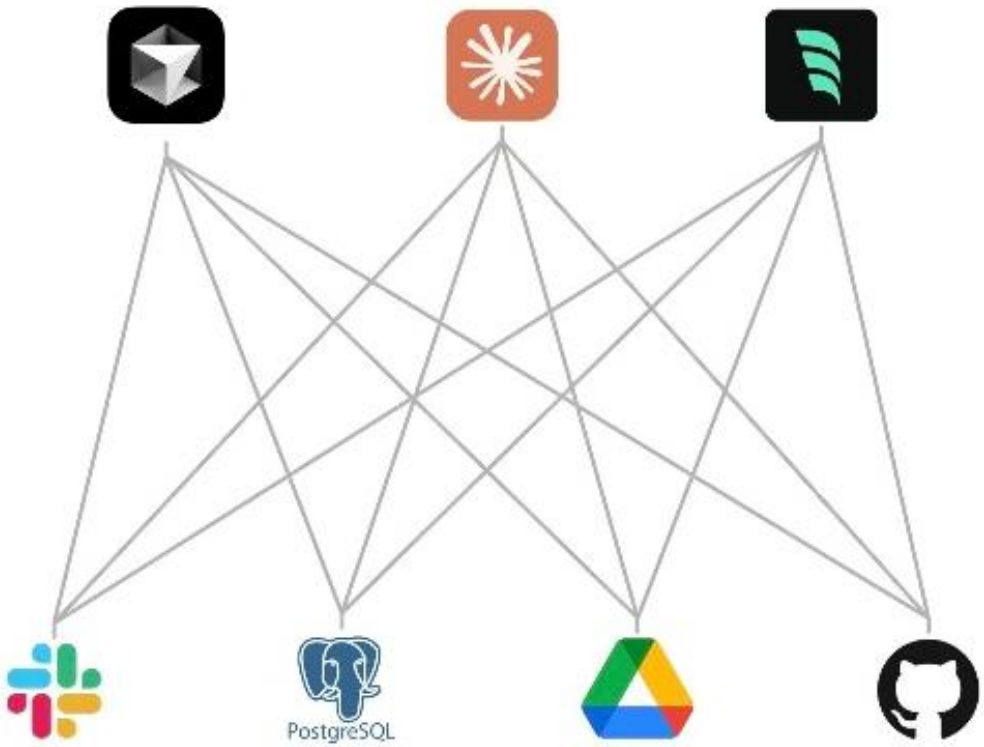
- Prompts
- Tools
- Data & resources
- Sampling

Timeline & Trend

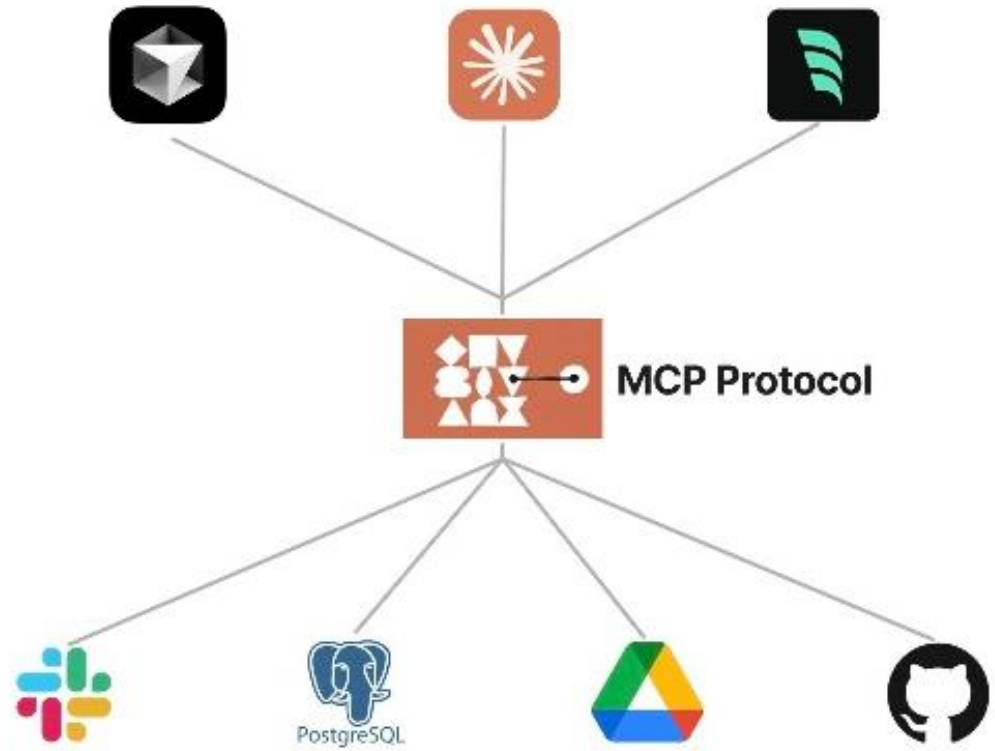
- Nov 2024 - Anthropic announced MCP, Zed supported MCP
- Dec 2024 - Cline supported MCP
- Jan 2025 - Cursor supported MCP
- Feb 2025 - Windsurf supported MCP
- March 2025 – VS Code supported MCP



Without MCP



With MCP



With MCP: Standardized AI Development

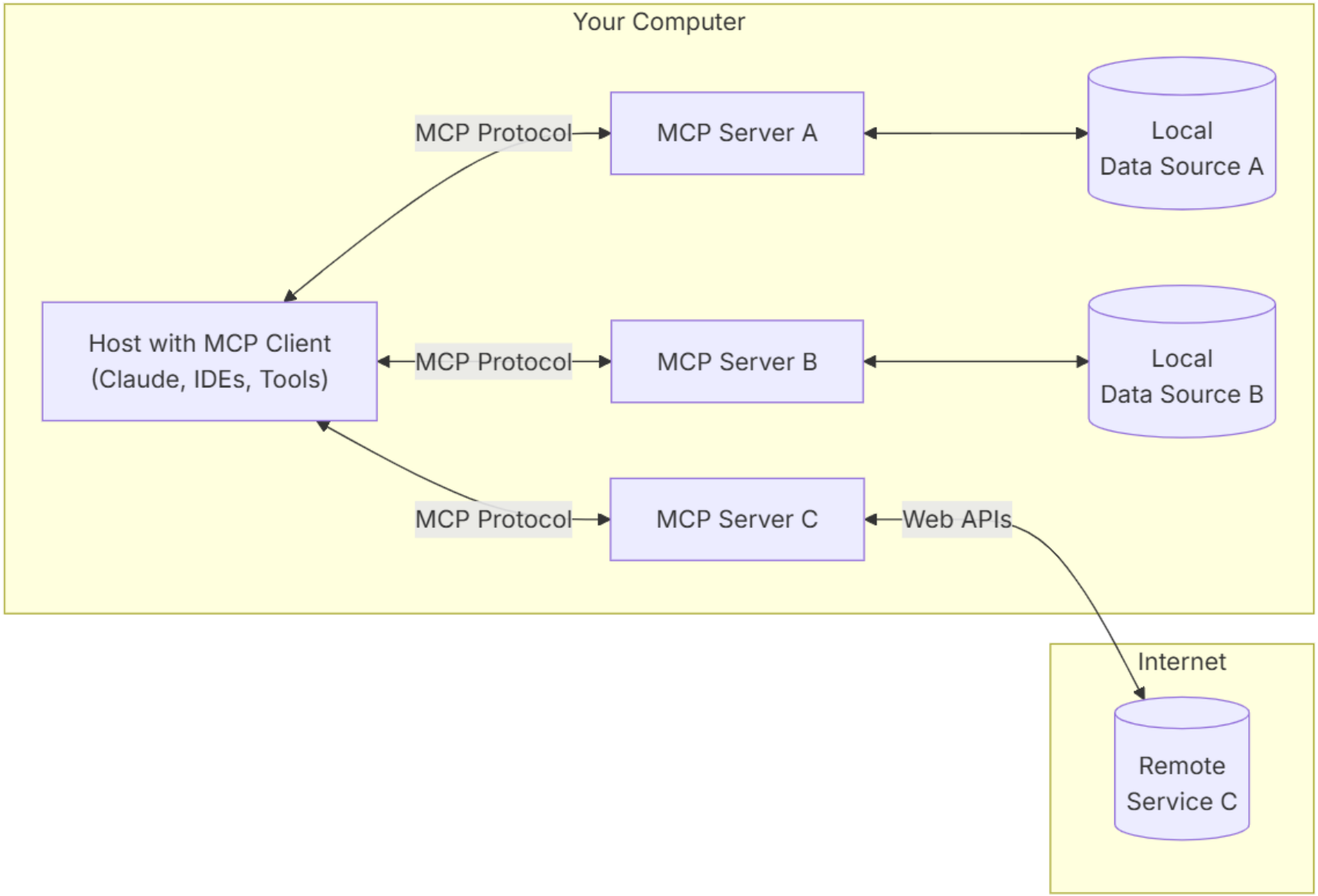
For AI Application developers

Connect your app to any MCP server with 0 additional work

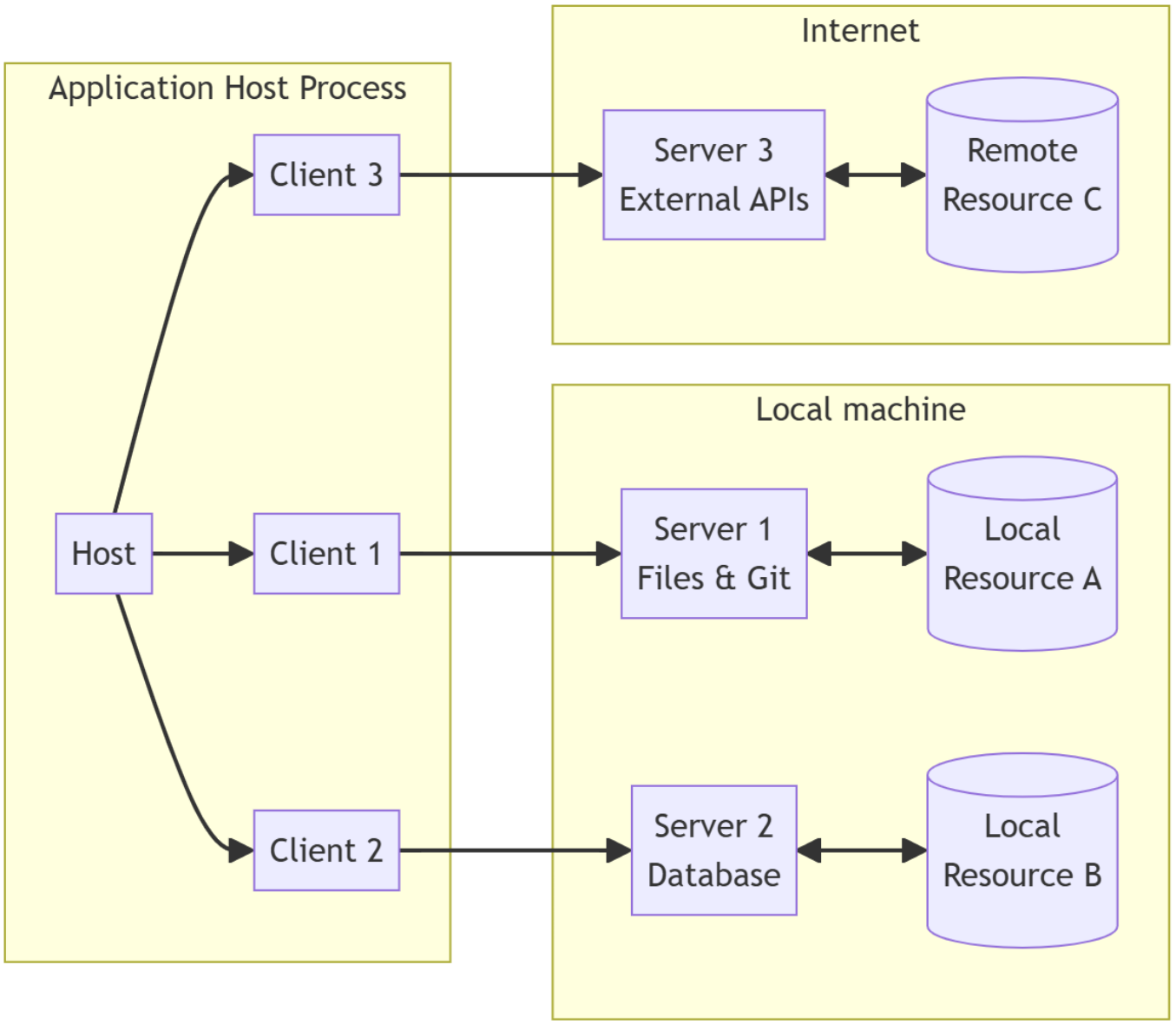
For Tool or API developers

Build an MCP server once, see adoption everywhere

General Architecture



Core Components



- **MCP Hosts**: Programs like Claude Desktop, IDEs, or AI tools that want to access data through MCP
- **MCP Clients**: Protocol clients that maintain 1:1 connections with servers
- **MCP Servers**: Lightweight programs that each expose specific capabilities through the standardized Model Context Protocol

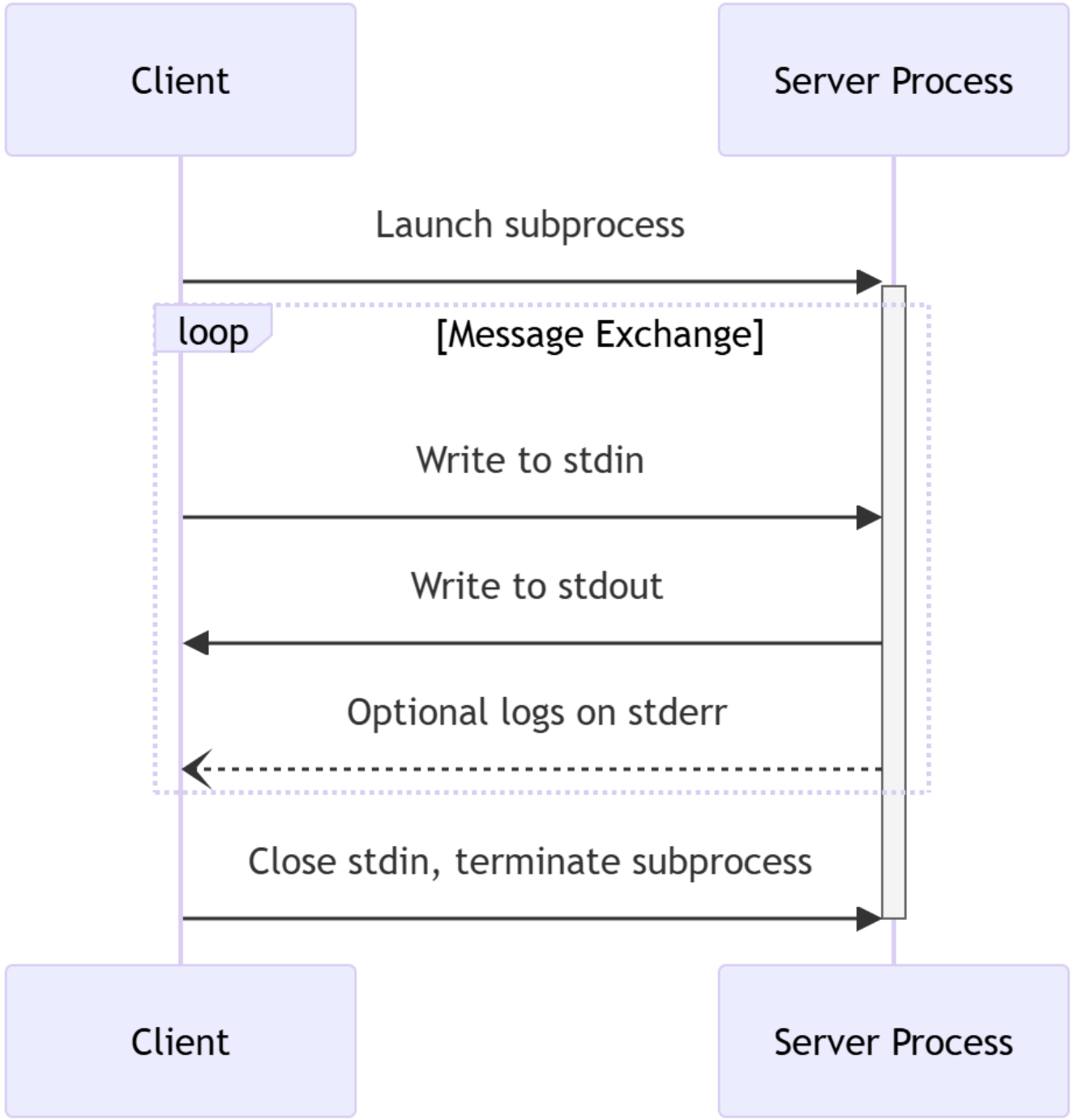
Transports

Two standard transport mechanisms for client-server communication:

- stdio, communication over standard in and standard out
- HTTP with Server-Sent Events (SSE)

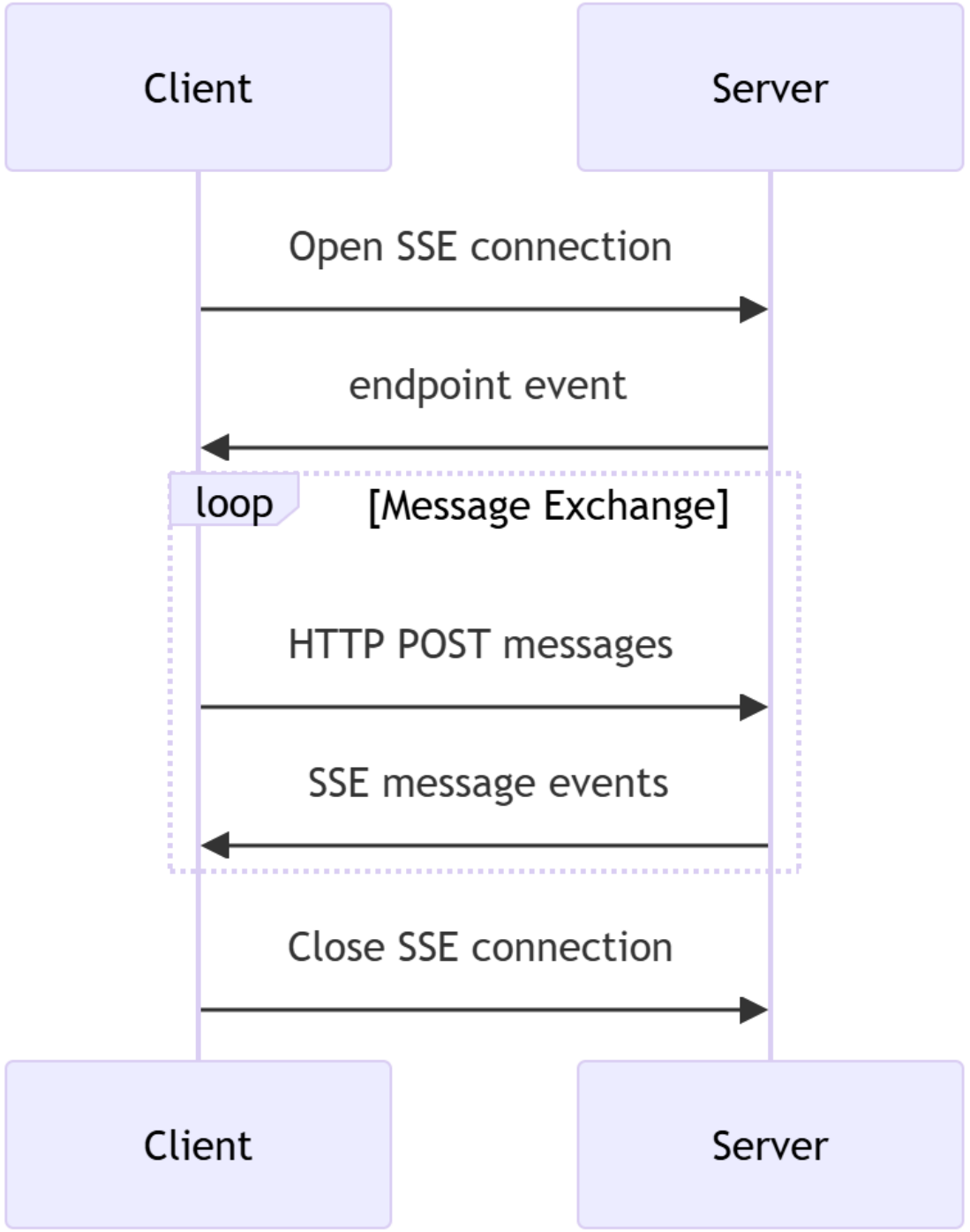
stdio

- Server could only run in **local**.
- The client launches the MCP server as a subprocess.
- The server receives JSON-RPC messages on its standard input (stdin) and writes responses to its standard output (stdout).

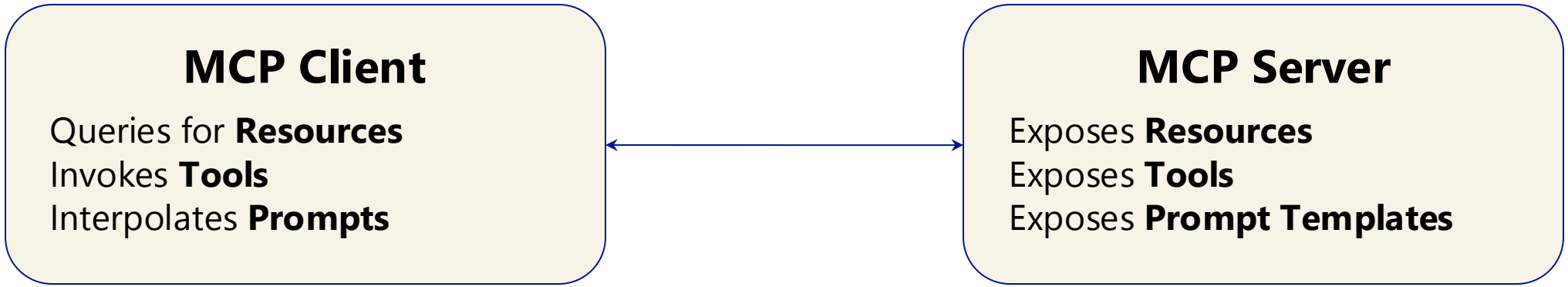


HTTP with SSE

- Server could run in both local and remote.
- The server **MUST** provide two endpoints:
  1. An SSE endpoint, for clients to establish a connection and receive messages from the server
  2. A regular HTTP POST endpoint for clients to send messages to the server



Server Features



Tools

Functions and tools that can be invoked by the client

- Retrieve / search
- Send a message
- Update DB records

Resources

Data or content exposed by the server

- Files
- Database Records
- API Responses

Prompt Templates

User-invoked prompts to interact with the model

- Document Q&A
- Transcript Summary
- Output as JSON

How?
Using and Implementing MCP

Using MCP

1. Install application that support MCP integrations: https://modelcontextprotocol.io/clients

2. Install/config MCP Server: https://modelcontextprotocol.io/examples

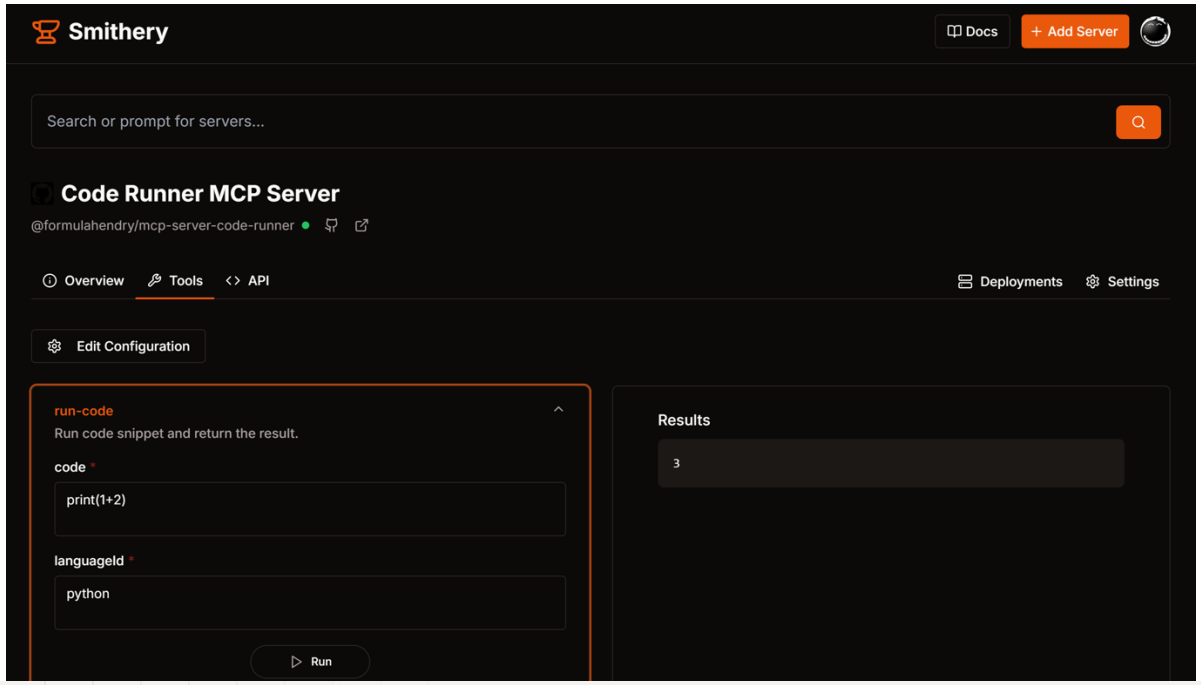
Popular Applications that support MCP

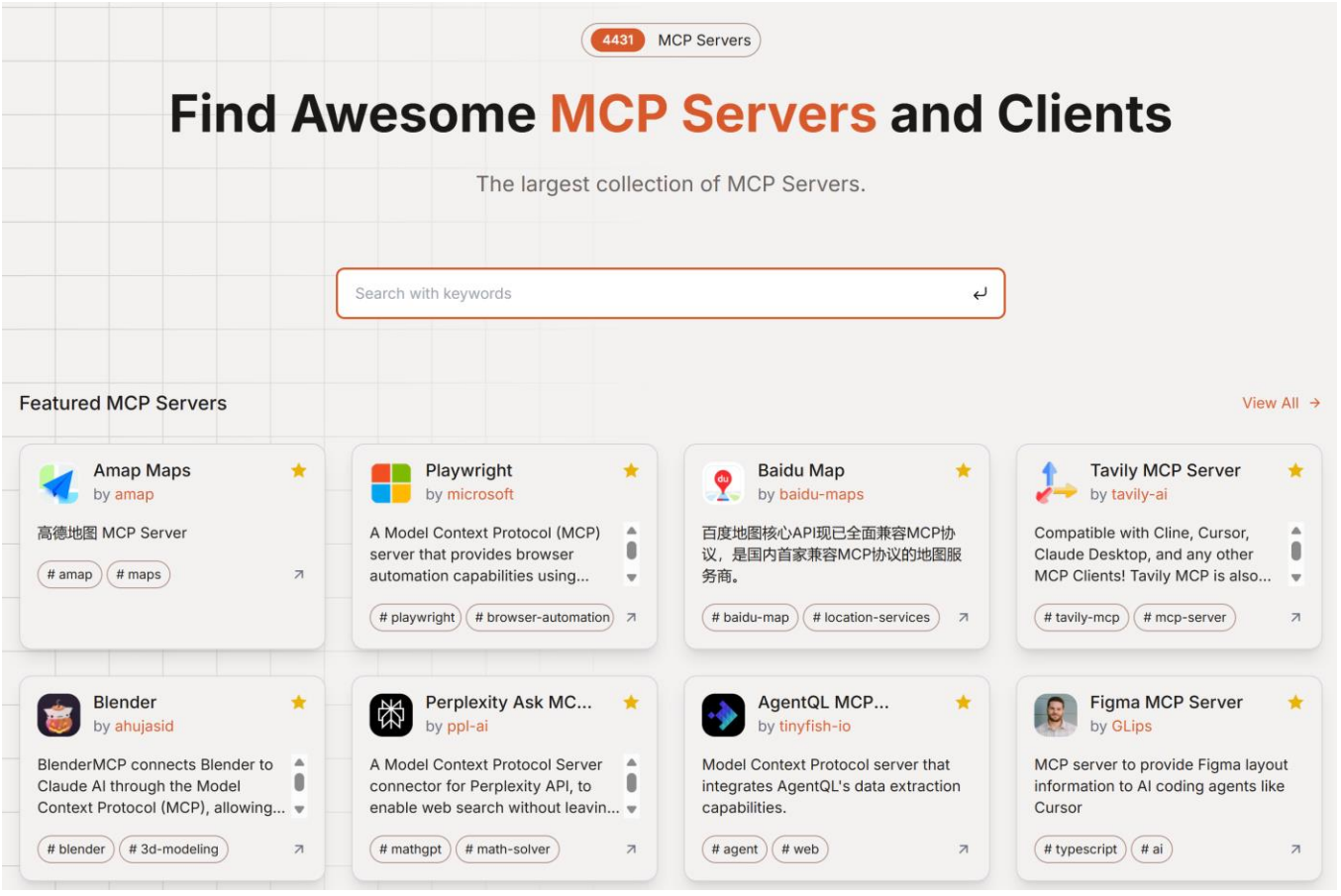
- Claude Desktop
- Cline
- VS Code

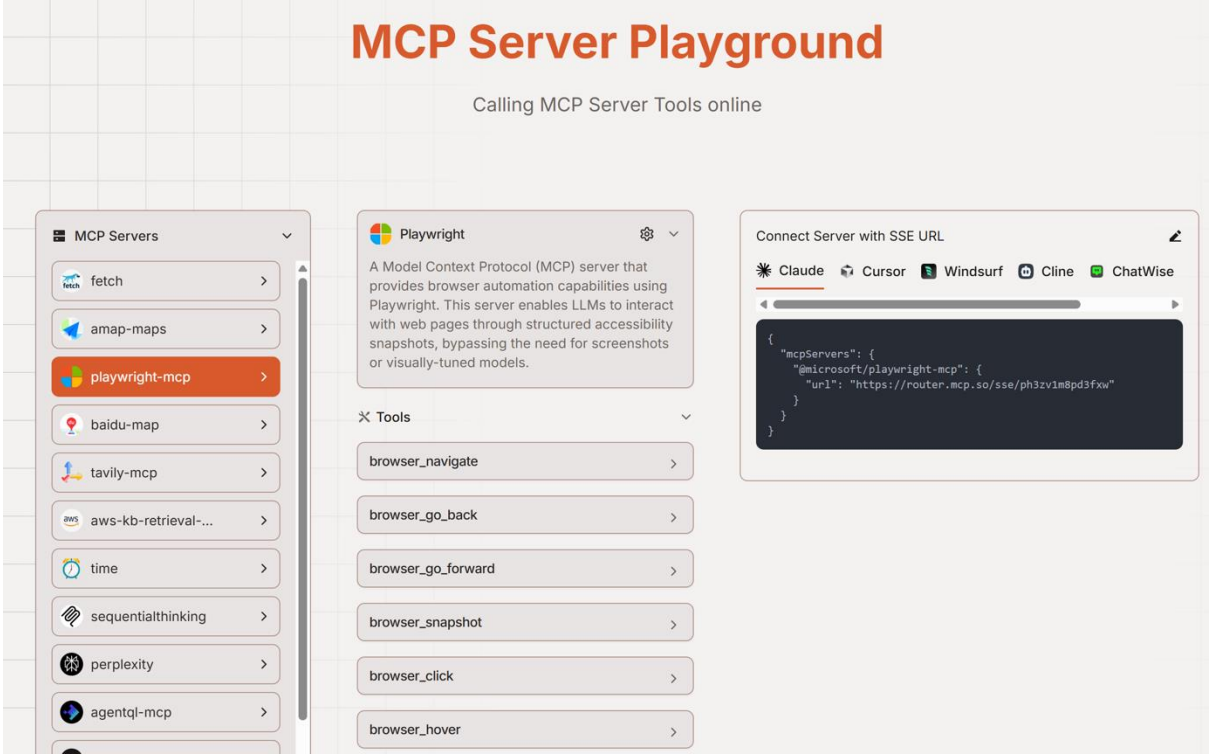
https://modelcontextprotocol.io/clients

Popular MCP Server Registry

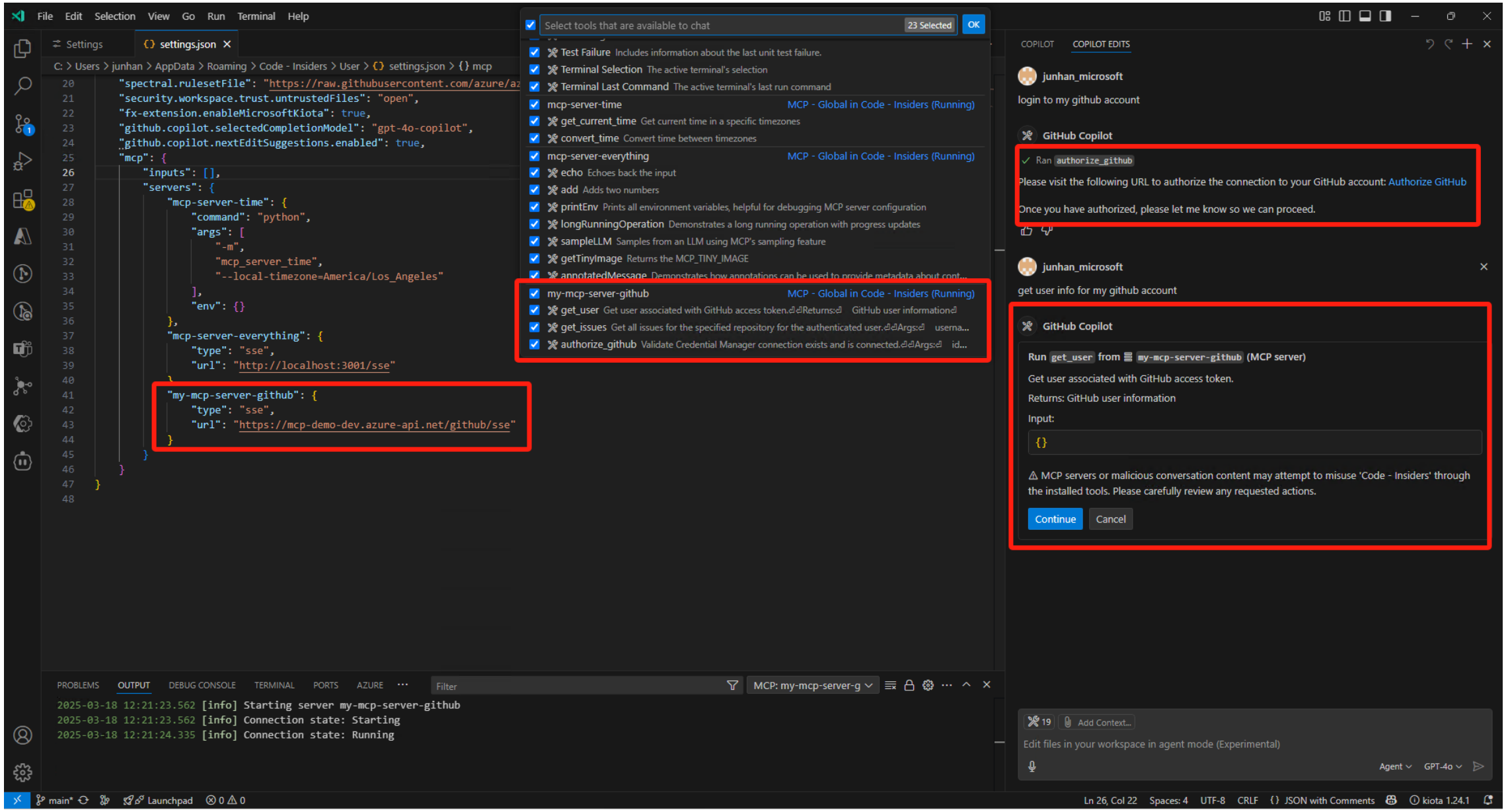
- https://mcp.so/servers
- https://glama.ai/mcp/servers
- https://smithery.ai/
- https://www.pulsemcp.com/servers







VS Code support MCP now!



Install/config MCP Server in VS Code

npx for VS Code

Configuration in `settings.json`:

```
{
  "mcp": {
    "inputs": [],
    "servers": {
      "mcp-server-code-runner": {
        "command": "npx",
        "args": [
          "-y",
          "mcp-server-code-runner"
        ],
      }
    }
  }
}
```

Docker

Use VS Code as example. Configuration in `settings.json`:

```
{
  "mcp": {
    "inputs": [],
    "servers": {
      "mcp-server-code-runner": {
        "command": "docker",
        "args": [
          "run",
          "--rm",
          "-i",
          "formulahendry/mcp-server-code-runner"
        ]
      }
    }
  }
}
```

```
code --add-mcp '{"name":"mcp-server-code-runner","command":"npx","args":[”-y", " mcp-server-code-runner"]}'
```

Implementing MCP

- SDKs for building MCP Client and MCP Server
  - TypeScript SDK
  - Python SDK
  - Java SDK
  - Kotlin SDK
  - C# SDK
- Tooling
  - MCP Inspector: interactive developer tool for testing and debugging MCP Server
  - generator-mcp: Yeoman Generator for MCP Server

Create a MCP Server

Prerequisites:

1. Latest VS Code Insiders
2. .NET
3. Node.js

Steps:

1. Create a .NET console App.
2. Add the NuGet package ModelContextProtocol https://www.nuget.org/packages/ModelContextProtocol
3. Implement your Tool logic for MCP Server
4. Debug/Test MCP Server in MCP Inspector
5. Run MCP Server in VS Code Agent Mode

Starting up our server

Update the Program.cs with some basic scaffolding to create the MCP server, configure standard server transport, and tell the server to search for Tools (or available APIs) from the running assembly.

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ModelContextProtocol.Server;
using System.ComponentModel;

var builder = Host.CreateEmptyApplicationBuilder(settings: null);
builder.Services
    .AddMcpServer()
    .WithStdioServerTransport()
    .WithToolsFromAssembly();

await builder.Build().RunAsync();
```

Defining a tool

In our startup code, the WithToolsFromAssembly will scan the assembly for classes with the **McpServerToolType** attribute and register all methods with the **McpServerTool** attribute. Notice that the McpServerTool has a **Description** which will be fed into any client connecting to the server. This description helps the client determine which tool to call.

```
[McpServerToolType]
public static class EchoTool
{
    [McpServerTool, Description("Echoes the message back to the client.")]
    public static string Echo(string message) => $"Hello from C#: {message}";

    [McpServerTool, Description("Echoes in reverse the message sent by the client.")]
    public static string ReverseEcho(string message) => new
string(message.Reverse().ToArray());
}
```

Publish a MCP Server

.NET makes it simple to easily create container images for any .NET app. All that needs to be done is add the necessary configuration into the project file:

```
<PropertyGroup>
    <EnableSdkContainerSupport>true</EnableSdkContainerSupport>
    <ContainerRepository>jsuarezruiz/mobile-dev-mcp-server</ContainerRepository>
    <ContainerFamily>alpine</ContainerFamily>
    <RuntimeIdentifiers>linux-x64;linux-arm64</RuntimeIdentifiers>
</PropertyGroup>
```

If we want to take these images and upload them, we are able to do it all from the CLI by passing in the specific container register to push to:

```
dotnet publish /t:PublishContainer -p ContainerRegistry=docker.io
```

Publish a MCP Server

We can configure the MCP on VS Code or other tolos in this way:

```
{
   "inputs": [],
   "servers": {
      "monkeymcp": {
         "command": "docker",
         "args": [
            "run",
            "-i",
            "--rm",
            "jsuarezruiz/ mobile-dev-mcp-server "
         ],
         "env": {}
      }
   }
}
```

Resources

- Documentation for guides and tutorials: https://modelcontextprotocol.io/
- Specification for protocol details: https://spec.modelcontextprotocol.io/
- GitHub: https://github.com/modelcontextprotocol
- MCP Servers: https://github.com/modelcontextprotocol/servers
- CSharp SDK: https://github.com/modelcontextprotocol/csharp-sdk

Questions and answers

Questions?

Thank you all!