MCP Server and client with Claude

Last amended: 17s[th] Oct, 2025
Folder: C:\Users\ashok\OneDrive\Documents\claude
Reference this article

Contents

Extending Claude LLM with local MCP servers

Learn how to extend Claude Desktop with local MCP servers to enable file system access and other powerful integrations

Model Context Protocol (MCP) servers extend AI applications' capabilities by providing secure, controlled access to local resources and tools. Many clients support MCP, enabling diverse integration possibilities across different platforms and applications. This guide demonstrates how to connect to local MCP servers using Claude Desktop as an example, one of the many clients that support MCP. While we focus on Claude Desktop's implementation, the concepts apply broadly to other MCP-compatible clients. By the end of this tutorial, Claude will be able to interact with files on your computer, create new documents, organize folders, and search through your file system—all with your explicit permission for each action.

Step1: Free account

Create a free account in Claude. It is free forever with certain limitations of usage and prompt limits. For limitations, please read here.

Step2: Claude Desktop

Download *Claude Desktop* from here and install on your machine.

Step3:

Reboot your machine.

Step4:

Open *Claude Desktop*. Press ctr+alt+space to open a chatbot and ask a simple question.

Step5: node.js

Install *node.js*, open source java script. Download Windows msi file (LTS version) from here and install.

Step6:

Open command prompt and enter the command `node -v`. You should get an output something like below:

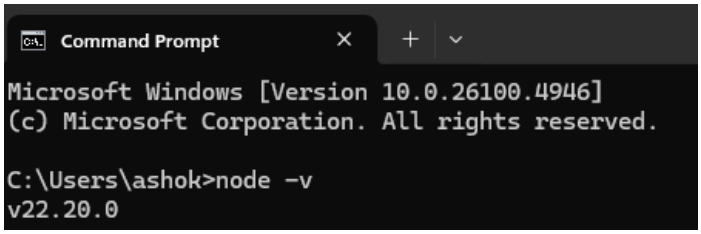

Figure 1: Checking node.js installation

Step7:

Reboot your machine.

Step8: Settings-Claude Desktop

Open Claude Desktop and access Settings, as:

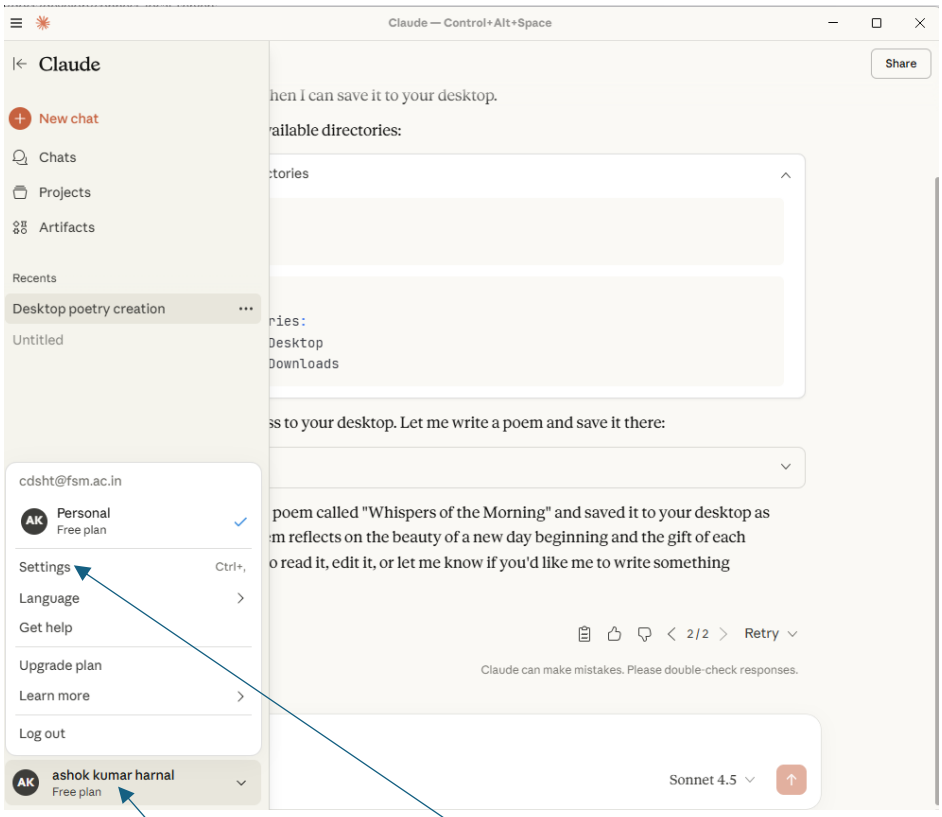

Figure 2: Click here to open menu to access Settings.

Step9:

In Settings window, click on Developer tab to open it.

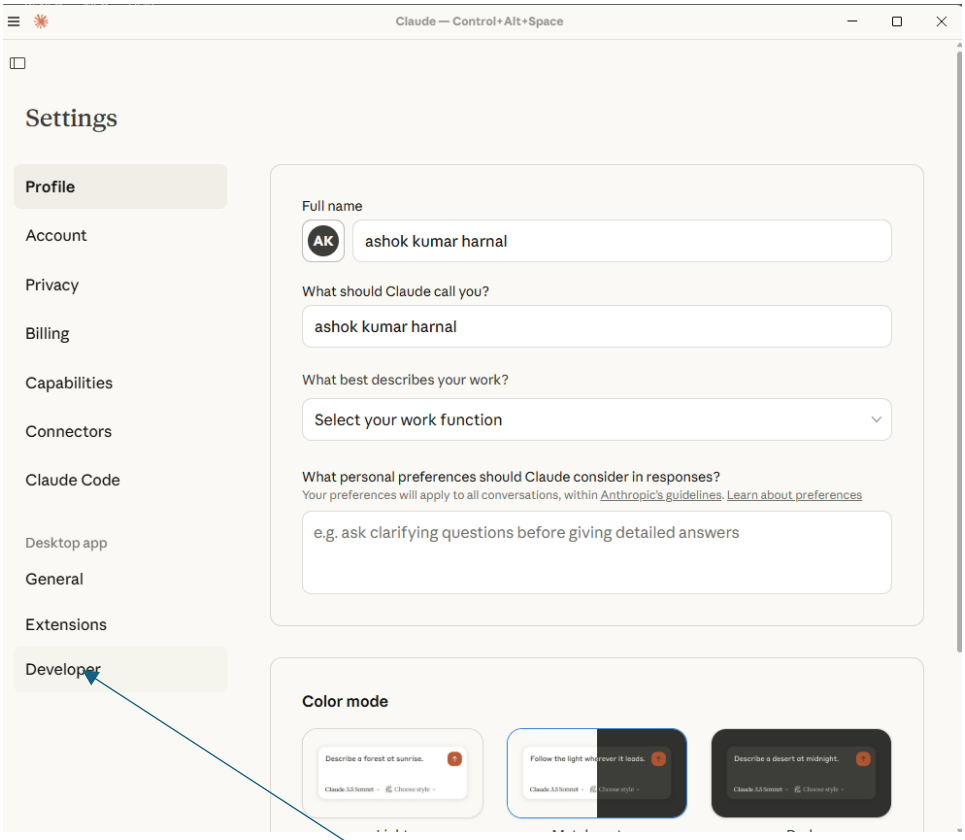

Figure 3: In Settings window, click Developer

FastMCP

See this article for connecting demo FAST MCP :

https://medium.com/@laurentkubaski/how-to-use-mcp-inspector-2748cd33faeb

FastMCP is the standard framework for building MCP applications. The Model Context Protocol (MCP) provides a standardized way to connect LLMs to tools and data, and FastMCP makes it production-ready with clean, Pythonic code:

```python
from fastmcp import FastMCP

mcp = FastMCP("Demo 🚀")

@mcp.tool
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b

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

About .json configfile:

The MCP JSON configuration format is an **emergent standard** that has developed across the MCP ecosystem. This format defines how MCP clients should configure and launch MCP servers, providing a consistent way to specify server commands, arguments, and environment variables. Refer this FastMCP link. This is the standard json file structure:

Configuration Structure

The standard uses a `mcpServers` object where each key represents a server name and the value contains the server's configuration:

```
{
  "mcpServers": {
    "server-name": {
      "command": "executable",
      "args": ["arg1", "arg2"],
      "env": {
        "VAR": "value"
      }
    }
  }
}
```

This is client adoption:

Client Adoption

This format is widely adopted across the MCP ecosystem:

- **Claude Desktop**: Uses `~/.claude/claude_desktop_config.json`
- **Cursor**: Uses `~/.cursor/mcp.json`
- **VS Code**: Uses workspace `.vscode/mcp.json`
- **Other clients**: Many MCP-compatible applications follow this standard

Step10: Edit Config What is the json file called in MCP server

In the Developers window, click on Edit Config. At this time, no local MCP server is installed.

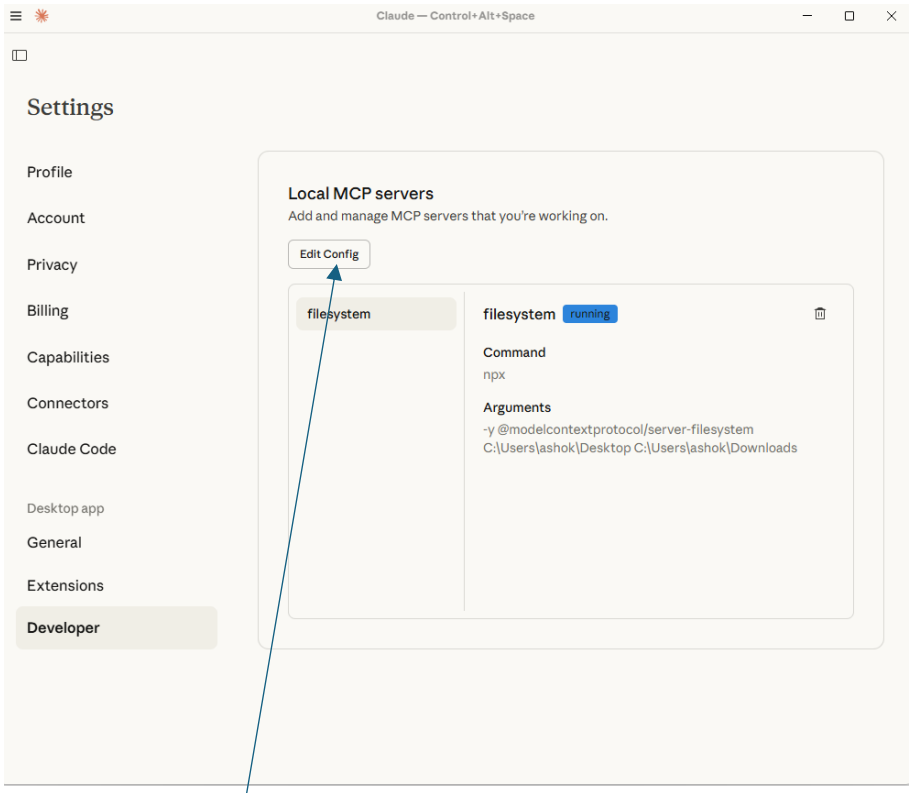

Figure 4: Click on Edit Config button

Step11:

Clicking *Edit Config* takes you to a file: `claude_desktop_config.json`. Open this file in VSCode or notepad++, and replace its contents with:

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

Replace username with the name of your machine, for example: ashok. This is the file on my machine:

```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-filesystem",
        "C:\\Users\\ashok\\Desktop",
        "C:\\Users\\ashok\\Downloads"
```

```
            ]
        }
    }
}
```

Save the amended json file and reboot the machine.

Step:12:

Upon restart, open Claude desktop, press ctrl+alt+space. You will see an icon to converse with MCP servers having tools:

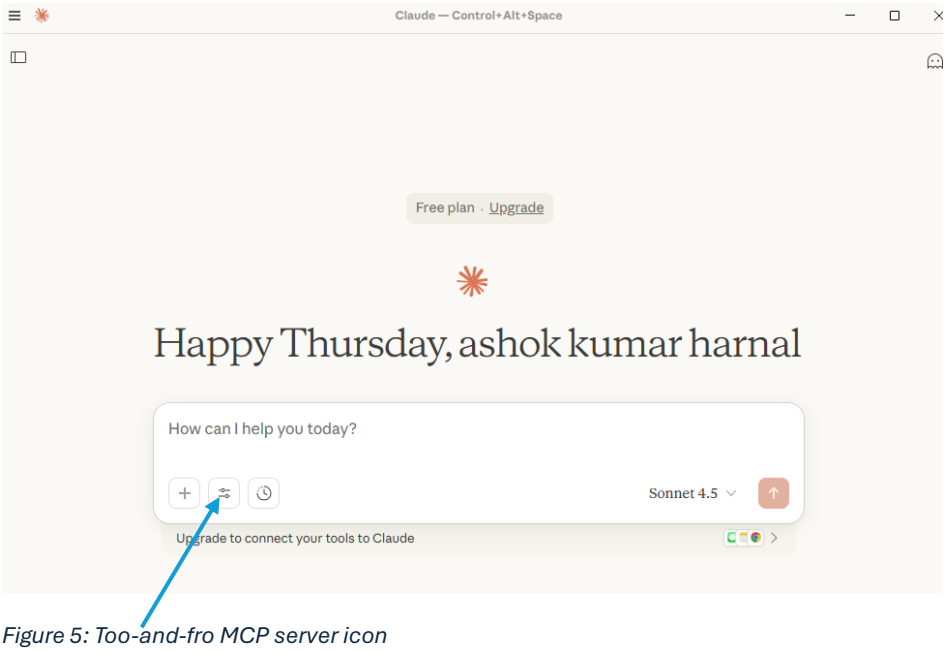

Figure 5: Too-and-fro MCP server icon

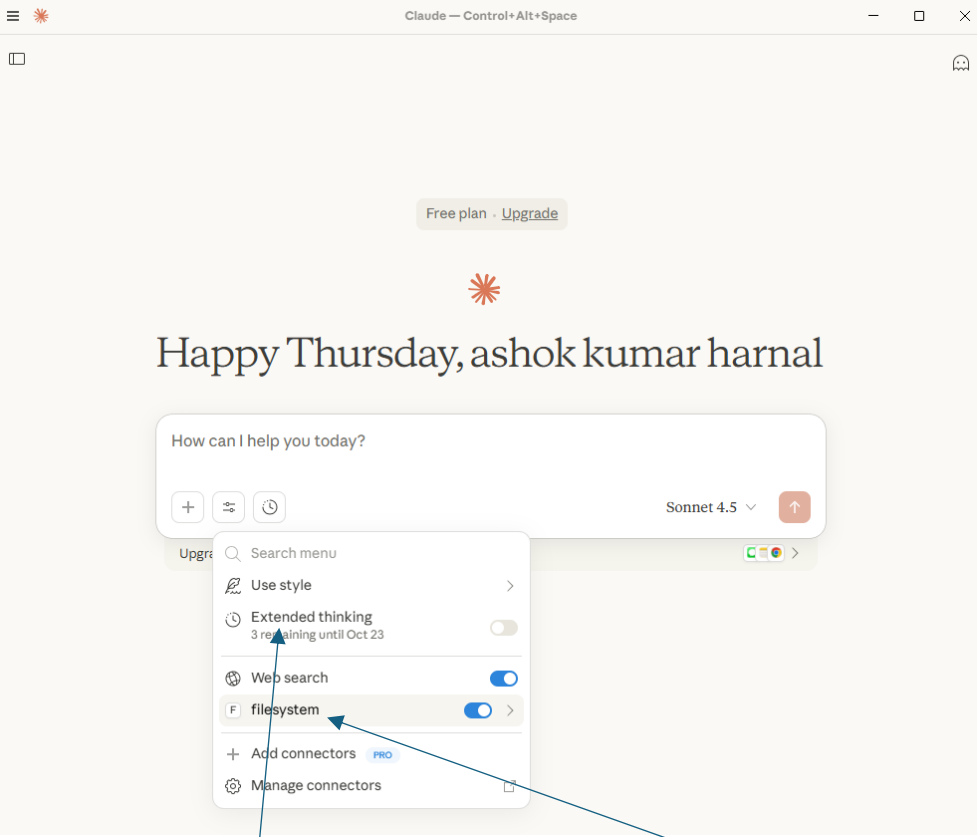

Figure 6: Click on the MCP server icon. Among tools, you will find filesystem tool.

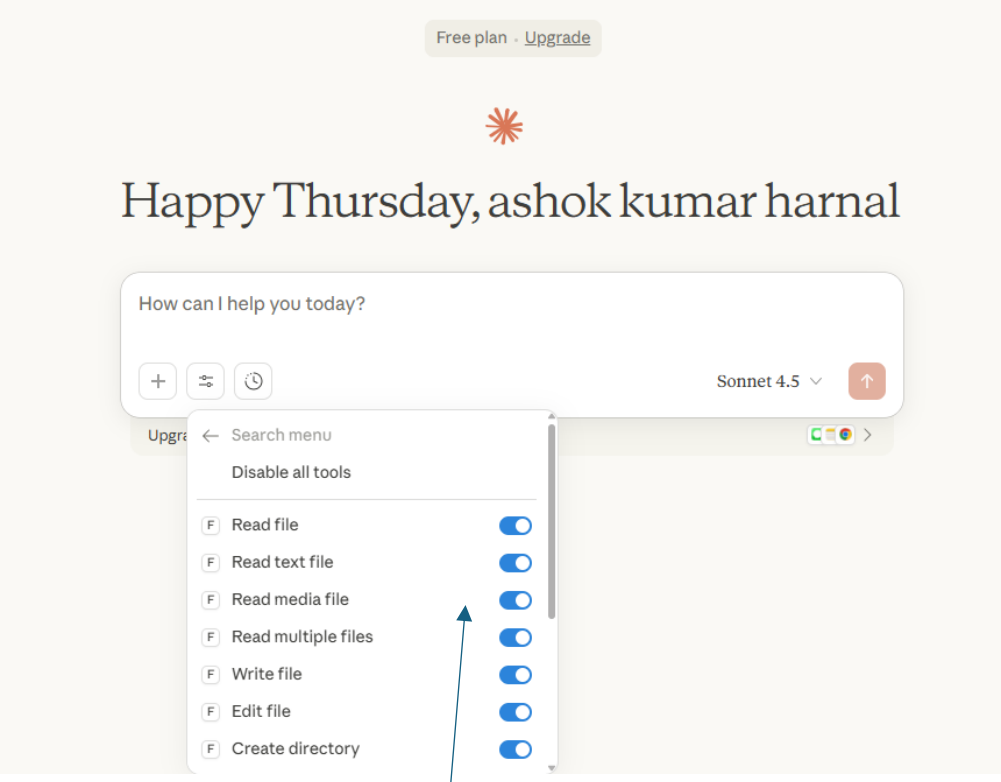

Figure 7: Click on filesystem to see what all agent tools are available. We have many.

Claude interaction with tools

With the Filesystem Server connected, Claude can now interact with your file system. Try these example requests to explore the capabilities:

File Management Examples

- **"Can you write a poem and save it to my desktop?"** - Claude will compose a poem and create a new text file on your desktop
- **"What work-related files are in my downloads folder?"** - Claude will scan your downloads and identify work-related documents
- **"Please organize all images on my desktop into a new folder called 'Images'"** - Claude will create a folder and move image files into it

Should you ask the first question, it will create a poem and save it to your desktop: c:/users/ashok/desktop.

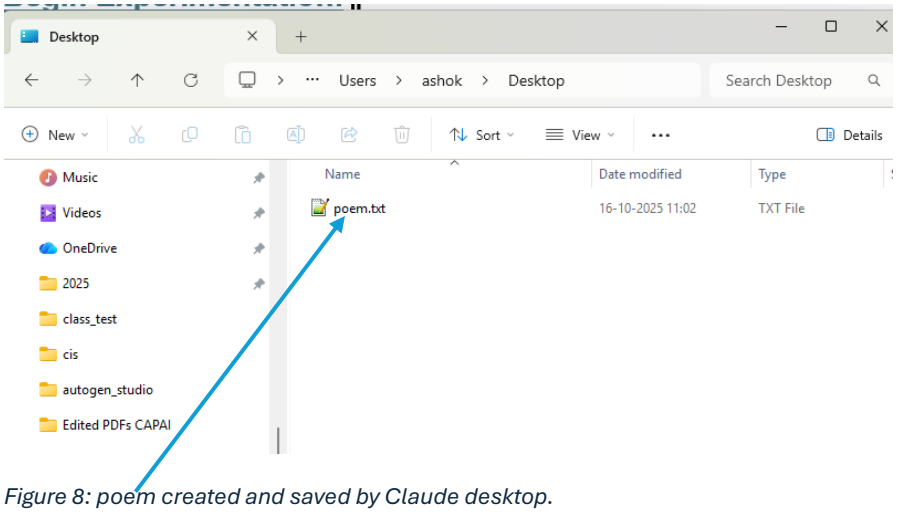

Figure 8: poem created and saved by Claude desktop.

Starting MCP Inspector:

Issue the following command to start MCP Inspector :

```
C:\users\ashok> npx @modelcontextprotocol/inspector
```

Access it as: localhost:6274

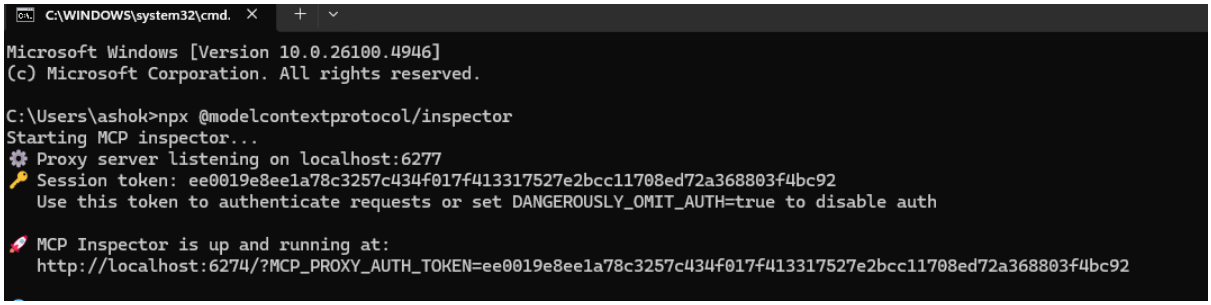

Figure 9: Start MCP Inspector and access it at: localhost:6277

Our Claude MCP server configuration is defined as below:

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

On the left panel of MCP Inspector, fill up the fields as per the above configuration file. Command is `npx` and there are four arguments:

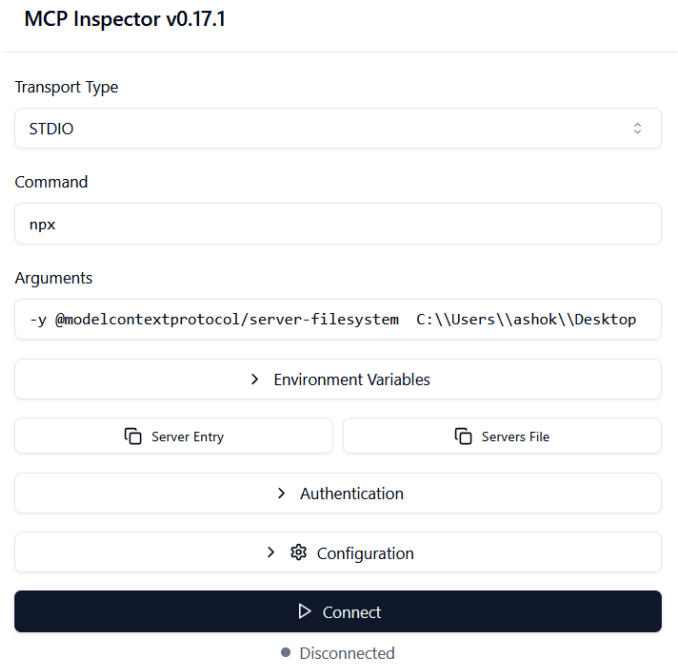

Figure 10: Transport type is STDIO. Command is npx and there are four fields. Click Connect.

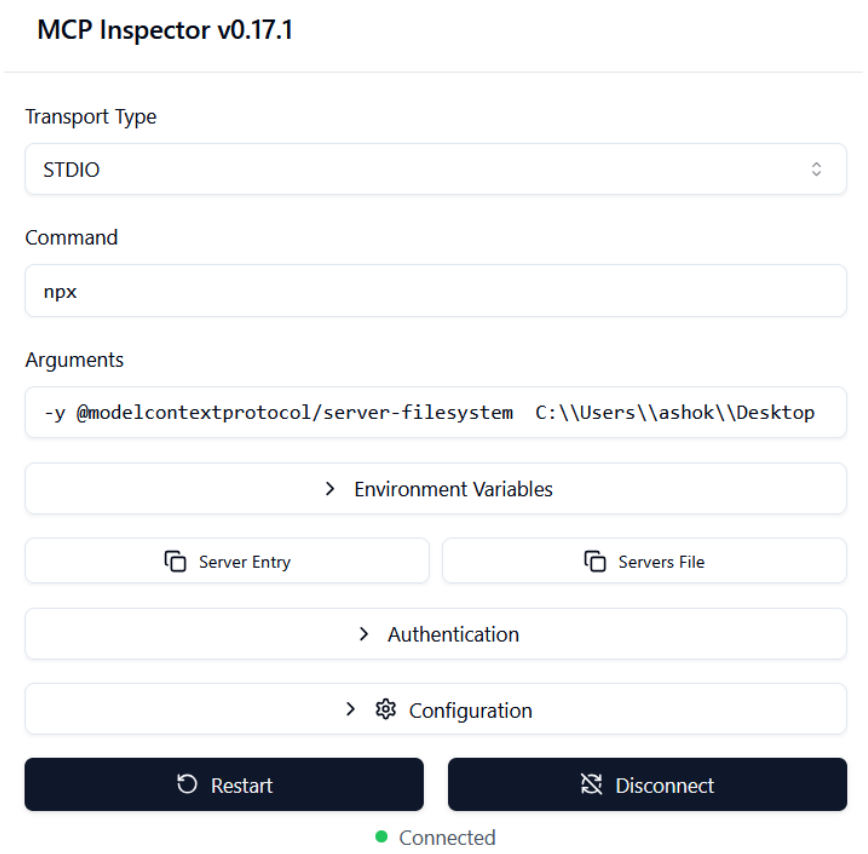

Figure 11: LEft panel of MCP Inspector after connection to server is established.

After connection to MCP Server is established, the middle panel of MCP Inspector is as below:

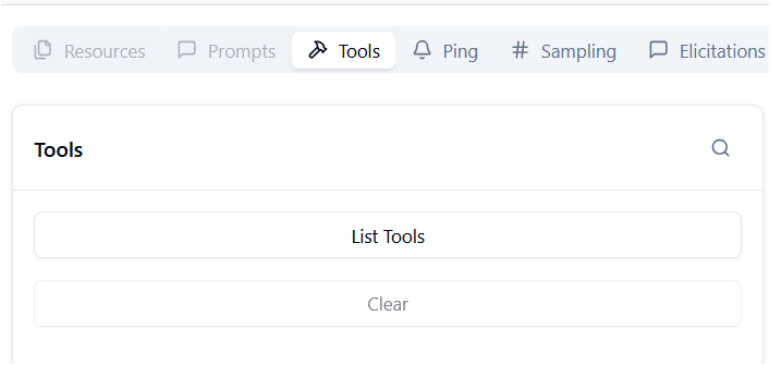

Figure 12: Click on List Tools to see what tools are exposed by MCP server

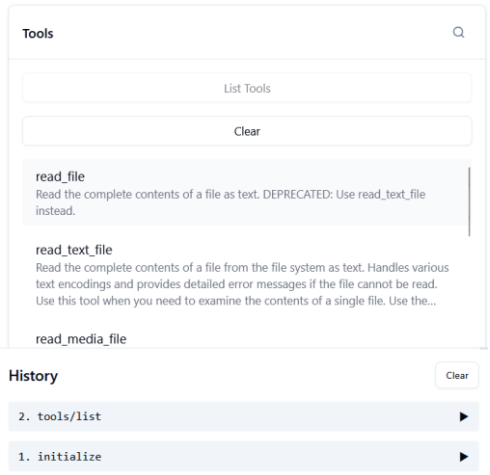

Figure 13: List of all tools exposed by MCP server

MCP Transports

MCP transports are mechanisms that handle communication between clients and servers within the Model Context Protocol (MCP). The transport layer is responsible for the low-level details of sending and receiving messages, which are formatted using the JSON-RPC 2.0 standard.

The primary goal of the MCP is to standardize how large language models (LLMs) and other AI agents connect to external tools, data sources, and APIs. Transports provide the specific method by which a client (such as an LLM agent) communicates with an MCP-enabled server (an external tool).

Common types of MCP transports

The MCP standard defines several types of transports, each with distinct advantages for different use cases.

STDIO (Standard Input/Output):

How it works: The client launches the server as a subprocess and communicates by writing to its standard input (stdin) and reading from its standard output (stdout).

Best for: Local tools, command-line interfaces (CLIs), and simple setups where the server is managed by the client.

Key detail: Messages must be written to stdout, while logs are sent to stderr to avoid interfering with the communication channel.

Streamable HTTP:

How it works: The server runs as an independent web service. The client sends requests using an HTTP POST request and receives a response. It allows for real-time, bidirectional streaming of multiple messages over the same connection.

Best for: Remote connections, supporting multiple clients, and interactive applications.

Note: This modern transport has replaced the older HTTP + Server-Sent Events (SSE) method.

Server-Sent Events (SSE):

How it works: The client sends a request via HTTP POST, and the server maintains a persistent connection to stream events back to the client.

Best for: Interactive UIs and dashboards where a long-lived, server-to-client streaming connection is desired.

Status: While still supported by some tools, it is now considered a legacy transport and has been replaced by streamable HTTP for new implementations.

Creating local MCP server with `FastMCP`

Ref: Welcome FastMCP and MCP JSON Configuration

Install latest Anaconda in Windows. Install `FastMCP`, as `pip install fastmcp`. Create a file `server.py`, as below. Note the directory in which server.py is created. (In my case the folder is: `C:\Users\ashok\OneDrive\Documents\claude`.

```
# server.py
from fastmcp import FastMCP

mcp = FastMCP("Demo 🚀")

@mcp.tool
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b

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

In the same location as above, create a json file (say, `demo.json`), as below. We create it but we are not using it at the moment.

```
{
 "mcpServers": {
  "server-name": {
   "command": "python",
   "args": ["server.py", "--verbose", "--port", "8080"]
    }
  }
}
```

Open Anaconda shell, `cd` to folder where server.py is located and run `server.py`. Below, file `server.py`, started in ipython.

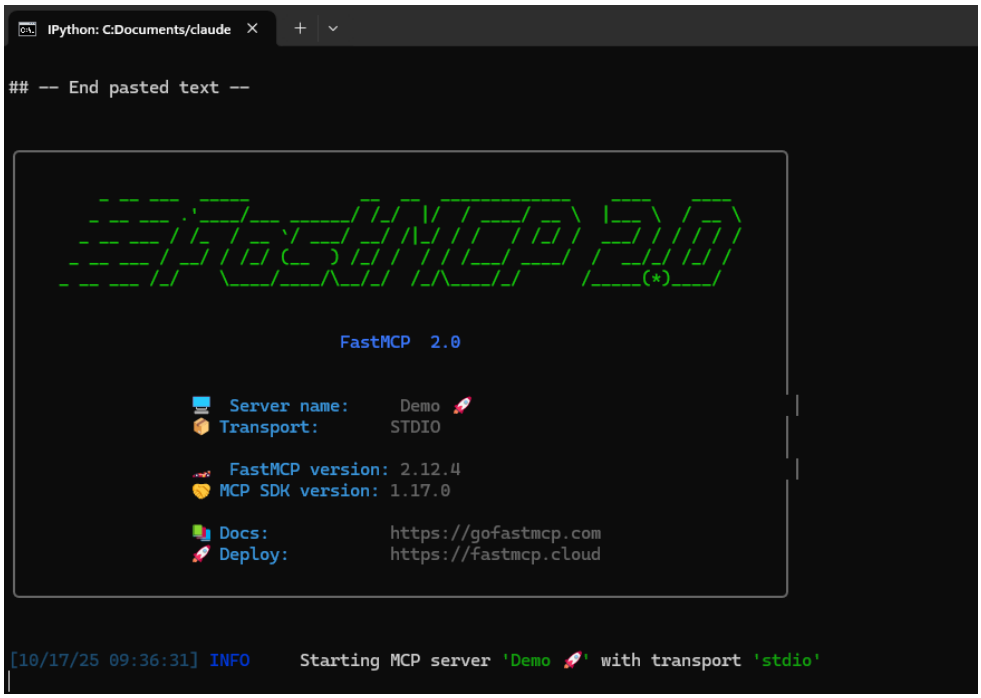

Figure 14: Local MCP server started in ipython. It has just one tool.

Open another Anaconda prompt (shell). Again `cd` to `claude` folder where `server.py` is located (`C:\Users\ashok\OneDrive\Documents\claude`). Start MCP inspector in this folder, as:

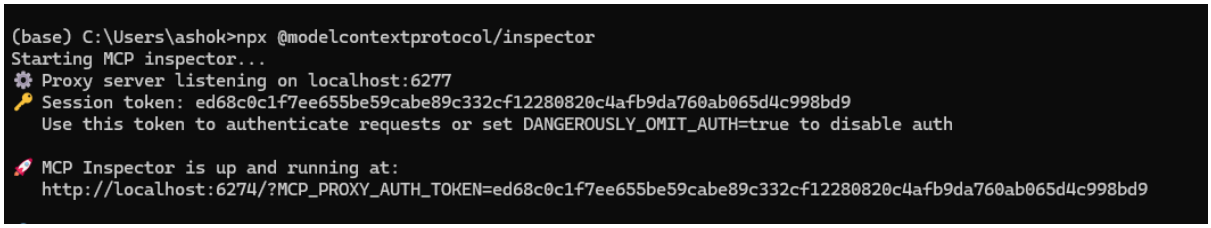

Figure 15: MCP Inspector started in Anaconda Shell

To access MCP server, fill up the left-panel of MCP Inspector-browser, as below (exactly as in the json file above). (Note: python should be available in the environment where MCP Inspector is started).

```
"command": "python",
"args": ["C:\\Users\\ashok\\OneDrive\\Documents\\claudeserver.py", "--verbose", "--
port", "8080"]
```

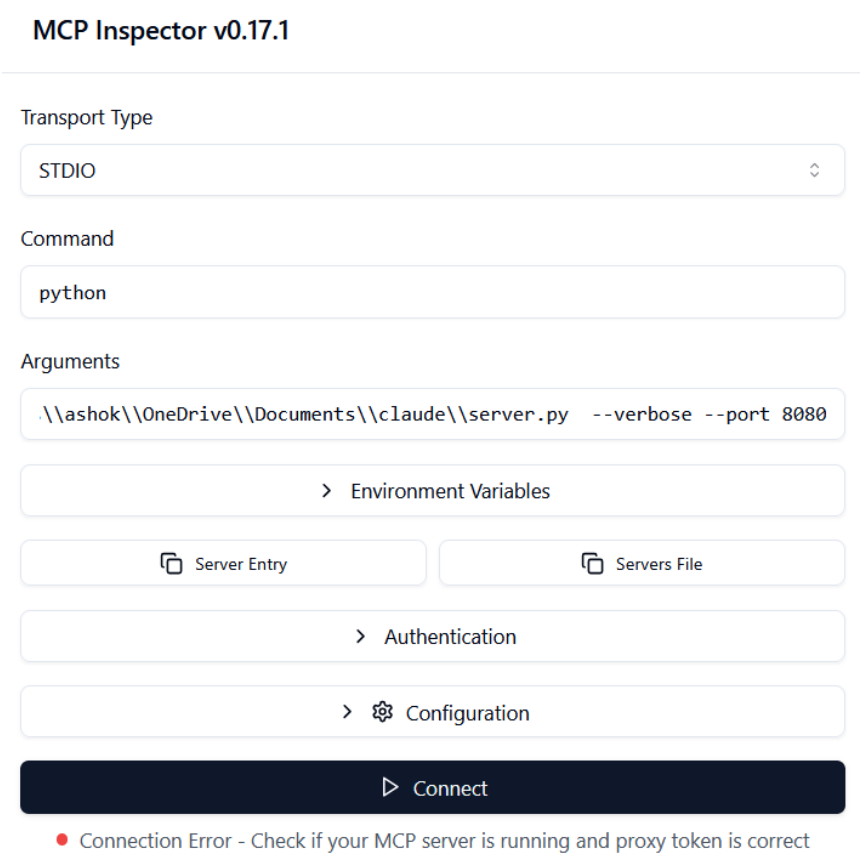

Figure 16: MCP Inspector. Fill up the fields and click Connect

Our MCP server is now connected:

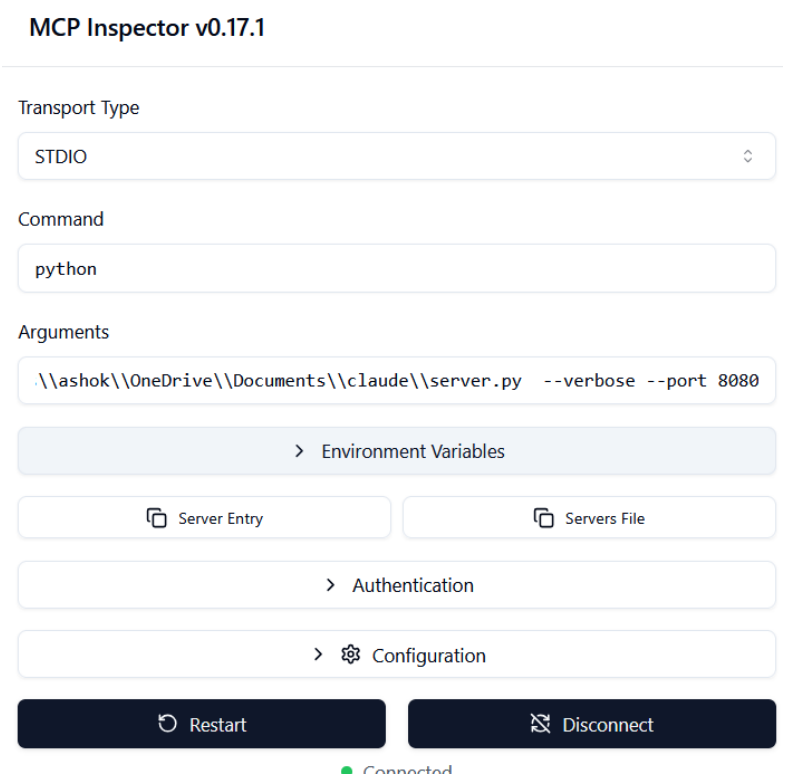

Figure 17: MCP server connected.

And, *Tools* panel of the Inspector shows the `add` tool:

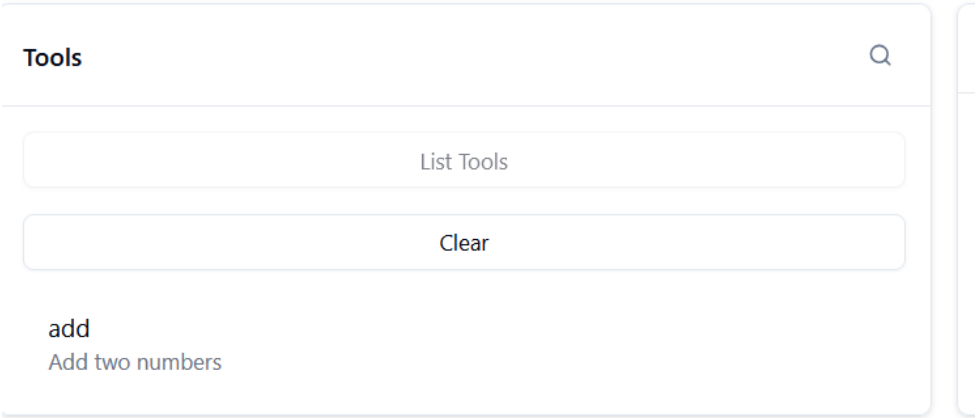

Figure 18: The, Tool add.tool now appears here.

#########