Chapter 4 - Connecting to Existing MCP Servers

Hey everyone!

In this video, I'm going to show you how to connect to existing MCP servers.

Instead of building everything from scratch like before, we can use **third-party MCP servers** that are already available. This saves time and lets us focus on the actual use case, not the boilerplate.

MCP Official Repository

First things first — the MCP community already maintains an **official repository of servers**.

You'll find servers for working with the filesystem, Git, Google Drive, and more.

Let's look at a few of the most common ones.

---

1. Filesystem MCP Server

Suppose you want your MCP client to interact with files on your machine.

You can use the **filesystem MCP server**.

Here's a minimal example in Python:

```
import os
workspace_folder = os.getcwd()

server_params = StdioServerParameters(
    command="npx",
    args=["-y", "@modelcontextprotocol/server-filesystem", workspace_fold
er],
)
```

So what's happening here?

- `npx` is a Node.js tool. It lets us run npm packages directly without installing them globally.

- In this case, `@modelcontextprotocol/server-filesystem` is an npm package that exposes an MCP server.
- And finally, we give it our workspace folder, so it knows where to operate.

That's it — with this, your client can now read and write files in that directory.

---

2. Git MCP Server

Next, let's say we want to interact with Git repositories.

There's a ready-to-use **Git MCP server**.

Example:

```
server_params = StdioServerParameters(
    command="uvx",
    args=[
        "mcp-server-git",
        "--repository",
        os.getcwd(),
    ],
)
```

This time we're using `uvx`, which is like npx but for Python-based MCP servers.

It lets you run a Python package directly, even if it isn't installed locally.

Here, `mcp-server-git` is a Python package that provides an MCP interface to Git.

We pass in the `--repository` flag with our current directory so it knows which repo to work with.

With this setup, you can call Git commands from inside MCP, such as:

- `git status`
- `git log`
- even checking branches or diffs

Super handy when you're building developer-focused workflows.

---

3. Tavily MCP Server

Now, what if you want to integrate **search capabilities** right into your MCP clinet?

That’s where the **Tavily MCP server** comes in. Tavily is a search API designed for developers or LLMs, and with its MCP server, you can query the web seamlessly.

Example in Python:

```
server_params = StdioServerParameters(
    command="npx",
    args= ["-y", "tavily-mcp@latest"],
    env = {
        "TAVILY_API_KEY": TAVILY_API_KEY
     }
)
```

You might have noticed something here — unlike **Filesystem** or **Git** MCP servers, the **Tavily MCP server actually needs an API key**.

That’s because it connects to an external service — Tavily’s search API — which requires authentication.

Since the Tavily server needs an API key, the first step is to grab one from the Tavily website. Let’s head over there and get our API key.

You always should store your keys securely — use environment variables never hard-code them in your script and never upload your keys to your repository or any other public services.

Recap

So that’s how you can connect to existing MCP servers like **Filesystem**, **Git**, and **Tavily**.

In future videos, I’ll dive into how to **chain multiple MCP servers together** and even build workflows that combine them intelligently.