# Connect VS Code and GitHub Copilot to the Functionize MCP Server

Model Context Protocol (MCP) · https://docs.functionize.com/hc/en-us/articles/43755023959959-Connect-VS-Code-and-GitHub-Copilot-to-the-Functionize-MCP-Server

VS Code connects to the Functionize MCP server natively in GitHub Copilot's agent mode. It handles Streamable HTTP and OAuth itself — including registering with our server automatically — so there is no bridge and no Node.js requirement.

## The easy way

Open the Command Palette and run **MCP: Add Server**. Choose an HTTP server, paste `https://mcp.functionize.com/mcp`, and pick whether to add it for this workspace or for you globally. VS Code writes the configuration for you.

**VS Code 1.101 or newer.** MCP support in Copilot agent mode landed then; older builds will not offer these commands.

**vscode.dev in a browser is not supported.** Use the desktop application.

If you would rather edit the file directly, read on.

## Adding it by hand

**This is where people trip up: the key name depends on which file you use.**

| | |
|---|---|
| `.vscode/mcp.json`<br>workspace, VS Code's own format | top-level key is `servers` |
| `.mcp.json`<br>repository root, portable format | top-level key is `mcpServers` |
| **Your user profile** | run **MCP: Open User Configuration** from the Command Palette |

**Use *MCP: Open User Configuration* unless you specifically want to share the server with everyone on the repository.** It gives you the server in every workspace, and VS Code does not publish that file's path, so the command is the only reliable way to reach it. `.vscode/mcp.json` is the shared project file — it is committed, so everyone who opens the repository gets the entry.

In either file, the entry looks like this (the top-level key differs, as above):

```
{
  "servers": {
    "functionize": {
      "type": "http",
      "url": "https://mcp.functionize.com/mcp"
    }
  }
}
```

`"type": "http"` **is required.** VS Code will not infer the transport. Use `http` for Streamable HTTP — `sse` is the legacy transport and is not what our server expects.

### Optional: pin a team

```
{
  "servers": {
    "functionize": {
      "type": "http",
      "url": "https://mcp.functionize.com/mcp",
      "headers": {
        "X-Functionize-Team-Id": "12345"
      }
    }
  }
}
```

Replace `12345` with your team's numeric ID. Omit the header to use **your default team**; switching teams inside Studio does not move the MCP connection. A team ID is not a secret, so a literal value is fine here — VS Code's documentation warns against hardcoding genuine secrets and offers `${input:...}` variables for those.

You do not need the optional `oauth` block. It exists for servers that issue static client IDs; ours supports dynamic registration, so VS Code will register itself.

## Sign in

VS Code opens your browser on first connection.

Sign in with the email and password you use for Functionize Studio. If you registered with Google, set a password via **Forgot password** on the Studio login page first. If your company signs in to Functionize through SSO, contact your Functionize representative.

**If VS Code offers to "try a different way" using the URL Handler, decline it.** Our server refuses that callback route, so it cannot complete. Let VS Code use its loopback listener, which is what it does by default.

The sign-in page is at `functionize.us.auth0.com`. If you are ever asked for Functionize credentials on another domain, do not enter them.

## Check it worked

Run **MCP: List Servers** to confirm Functionize is connected, then open Copilot in agent mode and ask it to list your Functionize agent sessions. The list is team-wide, so on a shared team you will see colleagues' sessions as well as your own.

## If it does not connect

**Check the key name matches the filename.** `servers` in `.vscode/mcp.json`, `mcpServers` in `.mcp.json`. Mixing them is the single most common mistake.

**Check `"type": "http"` is present.**

**To force a clean sign-in**, run **Authentication: Remove Dynamic Authentication Providers** from the Command Palette, then reconnect. This is the VS Code equivalent of clearing a cached token.

**If you were offered the URL Handler sign-in route and took it**, remove the dynamic authentication provider as above and reconnect, letting VS Code use its loopback listener instead.

For anything else, see Troubleshooting the Functionize MCP Connection.

### Related

- Getting Started with the Functionize MCP Server — what the server can do, once connected
- Connect Any MCP Client to Functionize — the endpoint, scopes and transport in one table
- Troubleshooting the Functionize MCP Connection — when it does not connect
- Exercise 7 - Drive Studio From Your Own AI Tool — ten minutes, against the demo store
- The Independent Tester Principle: Never Build and Test in the Same Session — why this belongs in a session of its own