Connect Gemini CLI to the Functionize MCP Server

Model Context Protocol (MCP) · https://docs.functionize.com/hc/en-us/articles/43755024055191-Connect-Gemini-CLI-to-the-Functionize-MCP-Server

Gemini CLI connects to the Functionize MCP server natively. It speaks Streamable HTTP, discovers our OAuth configuration on its own, and opens your browser to sign in. No bridge, no Node.js requirement beyond Gemini CLI itself.

The quickest route

```
gemini mcp add --scope user --transport http functionize \
  https://mcp.functionize.com/mcp
```

`--scope user` **matters.** The default is project scope, which writes into the current directory — and errors outright if you run it from your home directory. User scope gives you the server everywhere.

To pin a specific team at the same time:

```
gemini mcp add --scope user --transport http \
  --header "X-Functionize-Team-Id: 12345" \
  functionize https://mcp.functionize.com/mcp
```

Then start Gemini CLI and run `/mcp` to check the connection. If it needs authenticating, run `/mcp auth functionize` and complete the sign-in in your browser.

Adding it by hand

Edit one of these:

You, everywhere	`~/.gemini/settings.json` Windows: `%USERPROFILE%\.gemini\settings.json`
One project	`.gemini/settings.json` in the project root

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

Use `url` with `"type": "http"`. That is the current form and it selects Streamable HTTP, which is what our server speaks.

`httpUrl` is the deprecated spelling. It still works on older builds, so if you are on one and `url` is not recognized, swap it in — but prefer `url` plus `type` on anything current.

The `headers` block is optional; omit it to act as **your default team**. Switching teams inside Studio does not move the MCP connection. There is no need to set `timeout` — the value people copy around, 600000 milliseconds, is already the default.

Leave OAuth settings out entirely. Gemini CLI discovers our authorization and token endpoints automatically the first time it gets a 401, then opens your browser.

Sign in

On first use, Gemini CLI opens your browser and listens on a local callback port.

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.

There is no separate MCP account.

To sign in again, run `/mcp auth functionize` inside a session. That is the supported way to refresh an expired or broken token.

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

In a Gemini CLI session:

```
/mcp
```

That shows connection status and the available tools. Then ask it to list your Functionize agent sessions. The list is team-wide, so on a shared team you will see colleagues' sessions too.

Useful commands

`gemini mcp list`	Show configured servers
`gemini mcp remove functionize`	Remove the entry
`/mcp`	Status and tools, inside a session
`/mcp auth`	List servers needing authentication
`/mcp auth functionize`	Authenticate, or re-authenticate after a token expires

Known limitations

Sign-in needs a browser on the same machine. The OAuth callback is served on a local port, so it does not work headless, over SSH without display forwarding, or in a container with no browser. Run Gemini CLI somewhere with a browser and sign in there — do not copy the token file between machines.

Do not set `trust: true`. It bypasses every tool-call confirmation for that server, including writes.

Gemini CLI does send configured headers alongside the OAuth token, so a team pin set in `headers` takes effect. If the pin does not seem to be working, check it by listing sessions rather than with `list_agent_teams` — that tool echoes a pinned team ID back without checking it.

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