Connect Any MCP Client to Functionize

Model Context Protocol (MCP) · https://docs.functionize.com/hc/en-us/articles/43755039561751-Connect-Any-MCP-Client-to-Functionize

The Functionize MCP server is a standard Streamable HTTP endpoint with OAuth, so any MCP-capable client can connect to it — not only Claude. This article is the reference: what the server is, what it expects, and the two ways clients reach it.



The server, in one table

Whatever client you are configuring, these are the values it will ask for.

Endpoint	`https://mcp.functionize.com/mcp`
Transport	Streamable HTTP
Authentication	OAuth 2.0, authorization code with PKCE (S256)

Dynamic client registration	Supported, so most clients register themselves with no manual setup
Refresh tokens	Supported. `offline_access` is advertised and `refresh_token` is an allowed grant type, so your client can stay connected without repeated sign-ins
Scopes	`agent.sessions:read`, `agent.sessions:write`, `offline_access`
Optional header	`X-Functionize-Team-Id: <numeric team id>` to pin the connection to one team
Network	Public internet over HTTPS. No VPN, no network allow-listing and no manual provisioning — but the OAuth callback and browser-origin limits below do apply

You can inspect the server's own metadata at any time:

```
 https://mcp.functionize.com/.well-known/oauth-protected-resource
https://mcp.functionize.com/.well-known/oauth-authorization-server
```

Two ways in

1. Native — the client speaks HTTP MCP and OAuth itself

The client connects directly, opens your browser for sign-in, and captures the response. There is nothing to install and no Node.js requirement.

This is the better path when your client supports it. Configuration is usually just a name, the URL, and a transport type.

Which callbacks we accept. Sign-in works for any client using a **loopback callback** — `http://127.0.0.1:<port>` or `http://localhost:<port>` — which is what most desktop and CLI clients do. We also accept the registered callbacks for **Claude**, **Cursor** and **ChatGPT**.

A web callback or a custom URL scheme that is not on that list is **refused**. If your client offers a choice — VS Code, for instance, may offer to "try a different way" using a URL handler — decline it and let the client use its loopback listener.

2. Bridged — the client only speaks stdio, or cannot do OAuth

Many clients still expect a local command rather than a URL. For those, a small bridge called `mcp-remote` runs locally, handles the OAuth flow, and presents the remote server as if it were a local one.

This needs **Node.js 18 or newer**. Check with `node --version`.

```
npx -y mcp-remote https://mcp.functionize.com/mcp
```

In a client's configuration file, that usually looks like this:

```
{
  "mcpServers": {
    "functionize-hosted": {
      "command": "npx",
      "args": ["-y", "mcp-remote", "https://mcp.functionize.com/mcp"]
    }
  }
}
```

On Claude Desktop and some other apps, replace `"npx"` with the absolute path to your npx binary. Applications launched from the desktop do not inherit your shell, so anything installed through nvm, fnm or Volta is invisible to them. On Windows, use the `npx.cmd` file and double every backslash inside the JSON string.

An absolute `npx` path is not always enough. With nvm or Volta, `npx` still needs `node` on the launching application's PATH, and a desktop app does not have your shell's PATH. If the bridge starts and immediately dies, that is usually why — install Node system-wide, or point `command` at a Node that does not depend on a version manager.

Pinning a team through the bridge

```
{
  "mcpServers": {
    "functionize-hosted": {
      "command": "npx",
      "args": ["-y", "mcp-remote", "https://mcp.functionize.com/mcp",
               "--header", "X-Functionize-Team-Id:982"],
      "env": { }
    }
  }
}
```

Replace `982` with your own team ID. Omit the header entirely to act as **your default team**; switching teams inside Studio does not move the MCP connection.

Write the header with no space after the colon. The bug belongs to clients that pass arguments through to `npx` — Cursor, Codex CLI, and Claude Desktop on Windows. There, a space inside the argument is not escaped correctly and the header is silently dropped.

Do not use `list_agent_teams` to confirm a pin. It echoes a pinned team ID back without checking it. Confirm the pin by listing sessions and checking you see the sessions you expect for that team.

Useful bridge options

These are occasionally needed. Most people need none of them.

`--header "Name:value"`	Add a custom header, such as the team ID
`--transport http-only`	Force Streamable HTTP rather than letting the bridge negotiate

`--host 127.0.0.1`	Change the OAuth callback host
`--debug`	Write a verbose log for support to look at
`--keep-alive`	Ping periodically on flaky networks

The bridge caches your tokens in `~/.mcp-auth` on macOS and Linux, and `%USERPROFILE%\.mcp-auth` on Windows. Deleting that directory forces a fresh sign-in — it is the first thing to try if a previously working connection stops authenticating. Note that it clears cached tokens for every bridged server, not only ours.

Signing in

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 and no separate password, and your existing access determines what the connection can reach. Do not guess at a password — repeated failures can temporarily block sign-in.

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

Checking it worked

Ask your client to list your Functionize agent sessions. If it returns your team's sessions, you are connected. The list is team-wide, so on a shared team you will see colleagues' sessions as well as your own.

Ten tools should be available: `start_agent_session`, `send_agent_message`, `get_agent_session`, `get_agent_session_events`, `stream_agent_session_events`, `list_agent_sessions`, `list_agent_teams`, `stop_agent_session`, `upload_session_file` and `delete_session_file`.

Writing your own client

The endpoint is a standard implementation, so an MCP SDK in any language will work against it. Two things are worth knowing if you are building something custom:

The team header is read **per request**, not per connection. An application serving several teams over a single MCP connection can set a different `X-Functionize-Team-Id` on each request. Use `list_agent_teams` to enumerate the teams the signed-in user belongs to — but note it echoes a pinned team ID back without checking it, so do not treat its output as confirmation that a pin is in effect.

Interactive clients read their configuration when the connection starts, so if you change a pinned team in a config file, restart the connection for it to take effect.

Related

- Getting Started with the Functionize MCP Server — what the server can do
- Troubleshooting the Functionize MCP Connection — when a client will not connect
- Connect Claude Code to the Functionize MCP Server — a worked example of the native route
- Exercise 7 - Drive Studio From Your Own AI Tool — put it to work in ten minutes