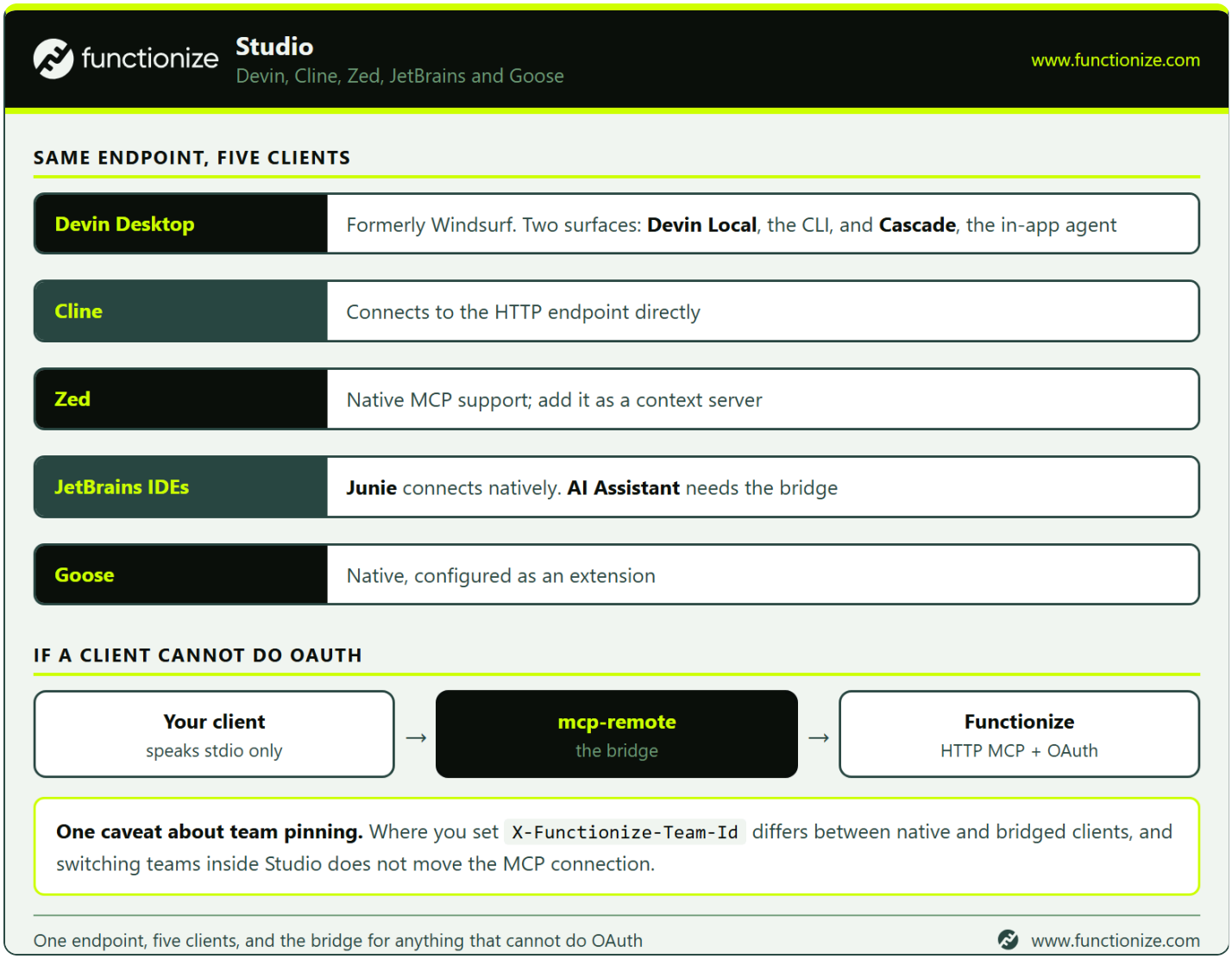
Connect Devin Desktop, Cline, Zed, JetBrains and Goose to the Functionize MCP Server

Model Context Protocol (MCP) · https://docs.functionize.com/hc/en-us/articles/43755039277335-Connect-Devin-Desktop-Cline-Zed-JetBrains-and-Goose-to-the-Functionize-MCP-Server

This article covers Devin Desktop (formerly Windsurf), Cline, Zed, JetBrains IDEs and Goose. All of them can reach the Functionize MCP server; most connect natively, and one needs a small bridge.



Before you start: the endpoint

Every tool below needs the same value:

```
https://mcp.functionize.com/mcp
```

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.

The config key names differ between tools. That is the main hazard when copying a snippet from elsewhere — each section below gives the correct shape.

Devin Desktop (formerly Windsurf)

Windsurf was renamed Devin Desktop in June 2026, and the old documentation URLs now redirect. If you are following an older guide, the paths in it are out of date.

Devin ships two things that both connect over MCP, and the configuration differs.

Devin Local — the CLI

The simplest route, if you have it:

```
devin mcp add -s user functionize https://mcp.functionize.com/mcp
devin mcp login functionize
```

The second command starts the sign-in.

Cascade — the in-app agent

The configuration file below is Cascade's. Config file:

macOS and Linux	`~/.config/devin/mcp_config.json`
Windows	`%APPDATA%\devin\mcp_config.json`

You can open it from the Cascade panel: the **…** actions menu, top right, then **Open MCP config file**.

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

Cascade historically used `serverUrl`; current builds accept `url` as well, and `url` is the one to prefer because it matches every other tool. There is no `type` or `transport` field.

OAuth is handled natively — authorization opens your provider directly in the browser.

We have not yet confirmed Devin sign-in end to end against our server. The configuration is right and the flow is standard, but if you hit a wall at the sign-in step, tell us what you saw rather than assuming you have it wrong.

Two things that catch people out. Cascade has a ceiling of 100 tools available at once, so if you already run several MCP servers, Functionize's tools can be crowded out without an obvious error. And on enterprise plans, admins can allowlist approved MCP servers — once any server is allowlisted, every server that is not on the list is blocked. If nothing connects and you are on an enterprise plan, ask your admin to allow `https://mcp.functionize.com/mcp`.

Cline

Use the interface rather than editing files — Cline does not publish the config path for the VS Code extension, and it varies by VS Code flavor.

Open the Cline panel, click the **MCP Servers** icon in the top toolbar, choose the **Remote Servers** tab, and fill in:

- **Server Name:** functionize
- **Server URL:** `https://mcp.functionize.com/mcp`
- **Transport Type:** Streamable HTTP

If you do edit the JSON directly, through the **Configure** tab:

```
{
  "mcpServers": {
    "functionize": {
      "type": "streamableHttp",
      "url": "https://mcp.functionize.com/mcp",
      "headers": {
        "X-Functionize-Team-Id": "12345"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

`"type": "streamableHttp"` is not optional here. Cline defaults to legacy SSE when the type is missing, for backwards compatibility. A snippet copied from another tool — where no type key is needed — will silently connect the wrong way.

On OAuth: current versions of Cline sign in natively. If the sign-in never starts, update Cline before reaching for the bridge — that fixes it far more often than the bridge does. The bridge at the end of this article remains the fallback for genuinely old builds.

Zed

Zed calls MCP servers "context servers", and they live in your settings file rather than a dedicated one.

macOS and Linux	`~/.config/zed/settings.json`
Windows	`%APPDATA%\Zed\settings.json`

On macOS that is `~/.config/zed/`, *not* Application Support — which surprises people. The safest route is the `zed: open settings file` command, or **Settings → AI → MCP Servers → Add Server → Add Remote Server**.

```
{
  "context_servers": {
    "functionize": {
      "url": "https://mcp.functionize.com/mcp"
    }
  }
}
```

The top-level key is `context_servers`, not `mcpServers`. There is no type or transport field.

Zed's OAuth behavior is worth understanding: when a remote server has no `Authorization` header configured, Zed prompts you to authenticate using the standard MCP OAuth flow. Setting an `Authorization` header suppresses that prompt. Our team header is `X-Functionize-Team-Id`, not `Authorization`, so you can add it without disabling sign-in:

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

If you see a settings-migration prompt after adding a server with only a `url`, update Zed — that was a known bug, fixed in April 2026.

JetBrains IDEs

JetBrains ships two different MCP clients, and they behave differently. Check which you are using.

Junie — connects natively

Config file, same on macOS and Windows:

- Project: `.junie/mcp/mcp.json` in the project root

- You, everywhere: `~/.junie/mcp/mcp.json`

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

Servers needing sign-in show as **Authorization required**. Open the MCP Installation Assistant, select the server, choose **Authorize**, and complete the login in the browser page that opens. The status changes to **Active**. Manage it afterward with the `/mcp` command.

We have not yet confirmed Junie sign-in end to end against our server. The configuration is right and the flow is standard, but tell us what you see if it stalls.

AI Assistant — needs the bridge

JetBrains AI Assistant does not support OAuth for MCP servers. It is an open, unresolved feature request on their own issue tracker — there is currently no way for it to start an authorization flow or store the resulting token. It therefore needs the bridge below.

Add it through **Settings | Tools | AI Assistant | Model Context Protocol (MCP)**, using the bridged configuration from the last section of this article. Use the Settings screen rather than looking for a file — JetBrains does not document where AI Assistant stores this on either operating system.

Goose

Goose uses YAML, and calls MCP servers "extensions".

macOS and Linux	`~/.config/goose/config.yaml`
Windows	`%APPDATA%\Block\goose\config\config.yaml`

The easiest route is `goose configure` → **Add Extension** → **Remote Extension (Streamable HTTP)**, then paste the URL. The flow does ask whether you want to add custom headers, so you can set the team pin there rather than editing the file afterward.

```
extensions:
  functionize:
    type: streamable_http
    name: functionize
    enabled: true
    uri: "https://mcp.functionize.com/mcp"
    headers:
      X-Functionize-Team-Id: "12345"
    timeout: 300
```

Note `uri`, not `url`, and that `timeout` is in seconds — some tools, such as Gemini CLI, use milliseconds for the same field. SSE is no longer supported by Goose; `streamable_http` is correct for our server.

OAuth is native, using a loopback callback on `127.0.0.1` with a random port. If you ever need that port fixed, set the `GOOSE_OAUTH_CALLBACK_PORT` environment variable.

The bridge, for anything that cannot do OAuth

If a client only speaks stdio, or cannot complete an OAuth flow, a small program called `mcp-remote` runs locally and presents our remote server as a local one. It needs **Node.js 18 or newer**.

```
{
  "mcpServers": {
    "functionize": {
      "command": "npx",
      "args": [
        "-y", "mcp-remote",
        "https://mcp.functionize.com/mcp",
        "--transport", "http-only",
        "--header", "X-Functionize-Team-Id:12345"
      ]
    }
  }
}
```

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. It is the most common bridge failure.

With nvm or Volta, note that an absolute `npx` path still fails if `node` itself is not on the launching application's PATH. The bridge starts and then cannot find its runtime.

`--transport http-only` tells the bridge not to waste a round trip probing for SSE, which our server does not use.

Tokens are cached in `~/.mcp-auth`, or `%USERPROFILE%\.mcp-auth` on Windows. Deleting that directory forces a fresh sign-in. Note that it clears cached tokens for every bridged server, not only ours.

To test the bridge outside your editor, which gives much clearer error messages:

```
npx -p mcp-remote@latest mcp-remote-client https://mcp.functionize.com/mcp
```

Checking any of them worked

Ask the tool to list your Functionize agent sessions. If your team's sessions come back, you are connected. The list is team-wide, so on a shared team you will see colleagues' sessions too.

One caveat about team pinning

Omit the header entirely to use **your default team**. Switching teams inside Studio does not move the MCP connection.

Do not use `list_agent_teams` to confirm a pin. It echoes a pinned team ID back without checking it, so seeing the ID in its output tells you nothing about whether the pin took effect. Confirm it by listing sessions and checking you see the sessions you expect for that team.

If a pin still does not take effect, confirm the connection works without the header first, then raise it with us.

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