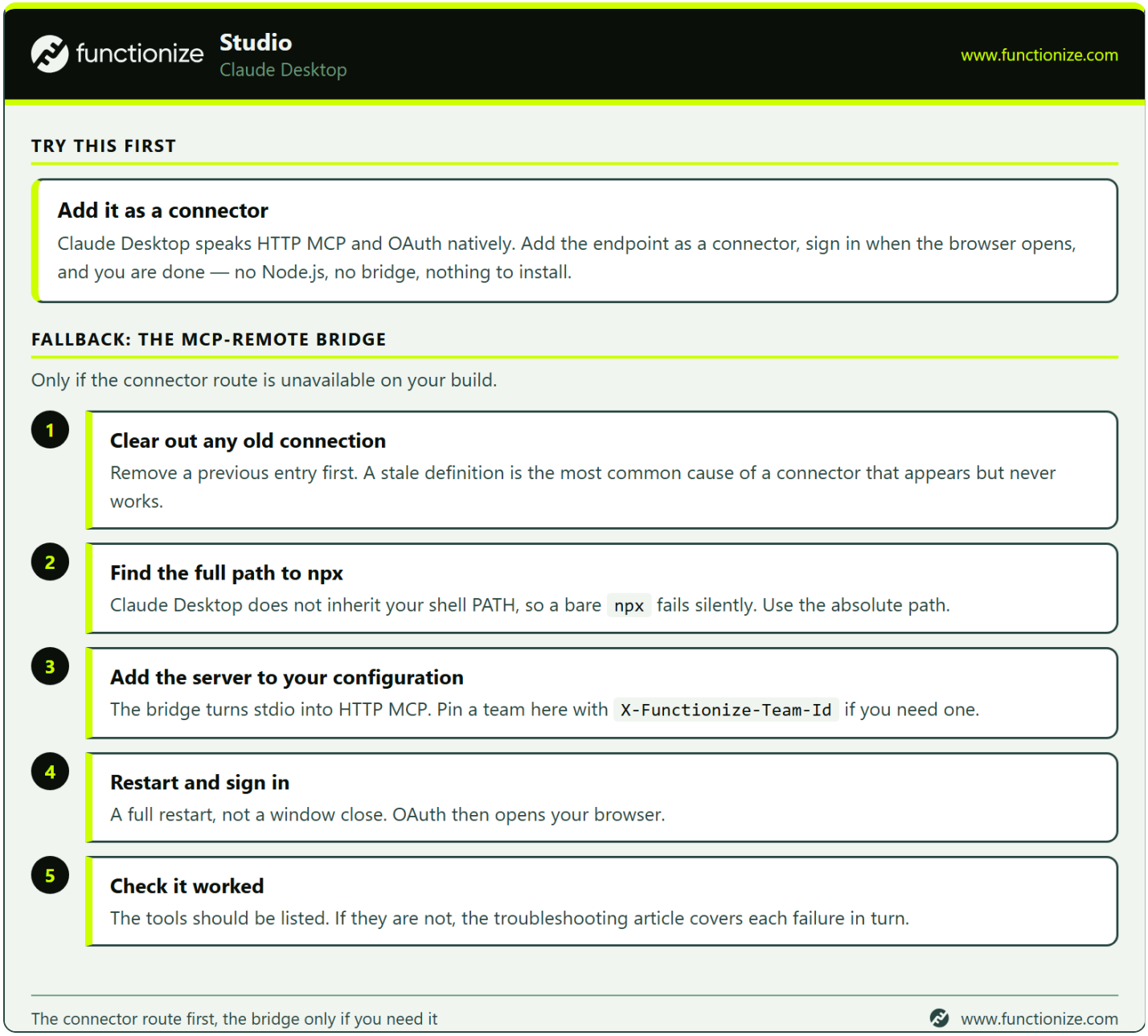
# Connect Claude Desktop to the Functionize MCP Server

Model Context Protocol (MCP) · https://docs.functionize.com/hc/en-us/articles/43753391718423-Connect-Claude-Desktop-to-the-Functionize-MCP-Server

Claude Desktop connects to the Functionize MCP server natively, through its Connectors settings. Nothing to install, no configuration file, no Node.




**This is the verified path and the one to use.** A fallback using the `mcp-remote` bridge is documented further down, for older builds or if the connector route is unavailable to you.

## Add it as a connector

1. Open **Settings** → **Connectors**.
2. Choose **Add custom connector**.
3. Name it **Functionize** and give it the URL `https://mcp.functionize.com/mcp`.
4. Save. Claude Desktop opens a browser tab for you to sign 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.

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

**On Team and Enterprise plans, only Owners can add connectors.** If **Add custom connector** is missing or disabled, ask an Owner in your organization to add it.

Then skip to Check it worked.

## Fallback: the mcp-remote bridge

Everything below is the fallback. Use it only if the connector route above is not available to you. It runs a small local bridge called `mcp-remote` on Node.js, and it means editing a configuration file and restarting the app.

### Before you begin

You need **Node.js 18 or newer** — the bridge runs on it. Check what you have:

```
node --version
```

If that command is not found, or reports anything below 18, install Node from nodejs.org. On macOS you can also run `brew install node`.

## The fastest way: let Claude do it

The setup below is fiddly — the right `npx` path, valid JSON, the correct config location for your operating system. Claude Code can do all of it for you. If you have Claude Code installed, open it and paste this prompt exactly as it is:

```
 Set up my Claude Desktop to connect to the hosted Functionize MCP server.
Work out my operating system yourself and use the right paths for it.

1. Confirm the server is reachable. Expect HTTP 200 from:
   https://mcp.functionize.com/.well-known/oauth-protected-resource

2. Check Node.js is 18 or newer (node --version). If it is missing or older, stop
   and tell me to install it from nodejs.org.

3. Find the ABSOLUTE path to a working npx binary and verify it runs with --version.
   Do NOT assume a bare "npx" works: Claude Desktop launches commands without my
   shell, so nvm, fnm and volta npx shell functions are invisible to it. On Windows
   use the npx.cmd file.

4. Open my Claude Desktop config:
     macOS:   ~/Library/Application Support/Claude/claude_desktop_config.json
     Windows: %APPDATA%\Claude\claude_desktop_config.json
   Preserve everything already in the file. Do not remove or alter any existing
   mcpServers entry, including a stdio "functionize" entry if one is there.
   Inside mcpServers, add:

     "functionize-hosted": {
       "command": "<the absolute npx path you found>",
       "args": ["-y", "mcp-remote@latest", "https://mcp.functionize.com/mcp"]
     }

   On Windows, escape backslashes in the JSON string (for example "C:\\Program
Files\\nodejs\\npx.cmd").

5. Validate that the file is well-formed JSON and show me the entry you added.

6. Then tell me to fully quit Claude Desktop and reopen it - Cmd+Q on macOS, or
   right-click the tray icon and Quit on Windows, because closing the window is not
   enough - and to sign in with my Functionize Studio credentials in the browser tab
   that opens.

Run the steps, do not just print them. Ask me only for what you cannot detect.
```

Then follow its final instruction: quit Claude Desktop properly, reopen, and sign in. Skip to Check it worked.

If you do not have Claude Code, or prefer to do it yourself, the manual steps follow.

## Manual setup

## Step 1 — Clear out any old connection

If you previously connected to an internal build of this server, remove that entry before adding the new one. An entry pointing at `internal-dev.functionize.com` will not work from outside our network.

Open the configuration file (paths in Step 3), find any `mcpServers` entry whose `args` mention `internal-dev.functionize.com`, and delete that entry. The old `@functionize/mcp` stdio package has been retired; remove any entry for that too.

The bridge also caches sign-in tokens separately from that file, so clear those too. **Quit Claude Desktop first** — deleting the token cache while the app is running leaves it holding stale credentials.

**macOS**

```
rm -rf ~/.mcp-auth
```

**Windows (PowerShell)**

```
Remove-Item -Recurse -Force "$env:USERPROFILE\.mcp-auth"
```

Note that this clears cached tokens for *all* servers bridged through `mcp-remote`, not just ours. If you have others, you will be asked to sign in to them again.

---

## Step 2 — Find the full path to npx

This step catches almost everyone, so do not skip it. Claude Desktop launches commands without your shell, so version managers such as nvm, fnm and Volta are invisible to it. A bare `"npx"` in the configuration will fail with "command not found". You need the absolute path.

**macOS**

```
which npx
```

If that returns nothing useful, or you use a version manager, try these in order and use the first that exists:

```
ls -t $HOME/.nvm/versions/node/*/bin/npx | head -1
ls $HOME/.volta/bin/npx
ls /opt/homebrew/bin/npx
ls /usr/local/bin/npx
```

**Windows (PowerShell)**

```
(Get-Command npx).Source
```

That usually returns something like `C:\Program Files\nodejs\npx.cmd`. On Windows you want the `.cmd` file, not a bare `npx`.

Keep whichever path you found — you will paste it in the next step.

---

## Step 3 — Add the server to your configuration

Open `claude_desktop_config.json`:

| | |
|---|---|
| **macOS** | `~/Library/Application Support/Claude/claude_desktop_config.json` |
| **Windows** | `%APPDATA%\Claude\claude_desktop_config.json` |

You can also reach it from the app: **Settings > Developer > Edit Config**.

Inside `mcpServers`, add the entry below **without removing or changing anything already there**. Replace `<NPX>` with the path from Step 2.

**macOS**

```
"functionize-hosted": {
  "command": "/opt/homebrew/bin/npx",
  "args": ["-y", "mcp-remote@latest", "https://mcp.functionize.com/mcp"]
}
```

**Windows**

```
"functionize-hosted": {
  "command": "C:\\Program Files\\nodejs\\npx.cmd",
  "args": ["-y", "mcp-remote@latest", "https://mcp.functionize.com/mcp"]
}
```

**Windows paths need doubled backslashes in JSON**, as shown. A single backslash is an escape character and will make the file invalid. Forward slashes also work if you prefer them.

If the file is empty or has no `mcpServers` block, the whole file should look like this:

```
{
  "mcpServers": {
    "functionize-hosted": {
      "command": "<NPX>",
      "args": ["-y", "mcp-remote@latest", "https://mcp.functionize.com/mcp"]
    }
  }
}
```

Save the file and check it is valid JSON before moving on — a missing comma here is the second most common cause of the server not appearing. Any JSON validator will do.

### Optional: pin a team

To have this connection act as a specific team rather than **your default team**, add a header argument:

```
"functionize-hosted": {
  "command": "<NPX>",
  "args": ["-y", "mcp-remote@latest", "https://mcp.functionize.com/mcp",
           "--header", "X-Functionize-Team-Id:982"]
}
```

Replace `982` with your team's numeric ID, and omit the header entirely to use your default team. Switching teams inside Studio does not move the MCP connection.

**On Windows, write the header with no space after the colon** — `X-Functionize-Team-Id:982`. A space there is swallowed by the way arguments are passed to `npx`, and the header arrives malformed.

`list_agent_teams` echoes a pinned team ID back without checking it, so seeing the ID in its output does not confirm the pin worked. Confirm it by listing sessions and checking you see the sessions you expect for that team.

## Step 4 — Restart and sign in

**Quit Claude Desktop completely.** Closing the window is not enough, because the app keeps running.

- **macOS:** press **Cmd+Q**.
- **Windows:** right-click the Claude icon in the system tray (near the clock) and choose **Quit**. If it is not there, end the process from Task Manager.

Reopen the app. A browser tab opens for you to sign 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 or password. 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.

## Step 5 — Check it worked

Ask Claude:

> list my Functionize agent sessions

It should call `list_agent_sessions` and return your team's sessions. The list is team-wide, so on a shared team you will see colleagues' sessions as well as your own.

All ten tools should now be listed under `functionize-hosted`.

## If something goes wrong

If you used the connector route, remove the connector and add it again — that clears most problems. If you used the bridge, the most common causes are a bare `npx` instead of an absolute path, invalid JSON, or closing the window rather than quitting the app. See Troubleshooting the Functionize MCP Connection for the full list.

**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