🛠️ Figma MCP to OpenServ MCP Client - Setup Guide

This guide walks you through setting up and connecting the Figma MCP to the OpenServ MCP client using mcp-proxy.

✅ Prerequisites

Before you begin, make sure you have the following tools and accounts installed or available:

- **Node.js** (version 18 or higher) – runtime for executing JavaScript server and CLI tools
- **A Figma account** – required to run the plugin and access design files
- **Figma Desktop App** – needed for running development plugins locally
- Install **Bun** if you haven't already:
  - `curl -fsSL https://bun.sh/install | bash` (Linux & macOS) or `powershell -c "irm bun.sh/install.ps1 | iex"` (Windows)
  - Bun is a JavaScript runtime required to run this MCP server (especially the WebSocket server and MCP integration). Learn more at https://bun.sh

The Narrative

Before diving into the full setup, it's important to understand the architecture:

Figma MCP is designed to run as a local agent that communicates with the Figma Desktop app via a plugin. It uses the stdio protocol by default, which is ideal for local use but incompatible with modern, web-based systems like OpenServ MCP Client, which expect Server-Sent Events (SSE).

To bridge this gap, we use a layered approach:

1. **Local Setup (Figma MCP)** – The core agent server and plugin system.
2. **SSE Wrapper (mcp-proxy)** – A proxy layer that wraps the local server and exposes it as an SSE endpoint.
3. **Internet Exposure (ngrok)** – A tunneling tool to expose the proxy as a public URL.

This guide walks you through each of these steps in order.

🧩 Step 1: Set Up Figma MCP (Base Layer)

📦 Manual Setup of Figma MCP

1.1 MCP Server: Integration with Cursor

Add the server to your Cursor MCP configuration in ~/.cursor/mcp.json:

```
{
  "mcpServers": {
    "TalkToFigma": {
      "command": "bunx",
      "args": ["cursor-talk-to-figma-mcp@latest"]
    }
  }
}
```

1.2 WebSocket Server

To start the WebSocket server required by the Figma MCP:

> ✅ **Important:** You must first **clone the repository** https://github.com/sonnylazuardi/cursor-talk-to-figma-mcp
>
> ```
> git clone https://github.com/sonnylazuardi/cursor-talk-to-figma-mcp.git
> cd cursor-talk-to-figma-mcp
> ```

Then, from **inside the cloned project folder**, run the WebSocket server using:

```
bun socket
```

This will launch the WebSocket server that the Figma plugin connects to locally.

1.3 Figma Plugin

To set up the Figma plugin locally:

a. In Figma, go to Plugins > Development > Import plugin from manifest.
b. In the file browser, navigate to the **cloned cursor-talk-to-figma-mcp repository**.
c. Select the src/cursor_mcp_plugin/manifest.json file
   📁 This is the plugin manifest path **within the cloned repository**.
d. After importing, the plugin will now be available under **Plugins → Development**.

You can now launch the plugin from within Figma.

Usage

1. Start the WebSocket server
2. Install the MCP server in Cursor

   To allow Cursor to communicate with your local Figma MCP agent:

   1. Open **Cursor**.
   2. Navigate to ~/.cursor/mcp.json
   3. Add the following configuration block under “mcpServers”:

```
{
 "mcpServers": {
  "TalkToFigma": {
   "command": "bunx",
   "args": ["cursor-talk-to-figma-mcp@latest"]
  }
 }
}
```

✅ **If you encounter issues**, you may need to replace "bunx" with the **absolute path to Bun**.
Run this command to find it:

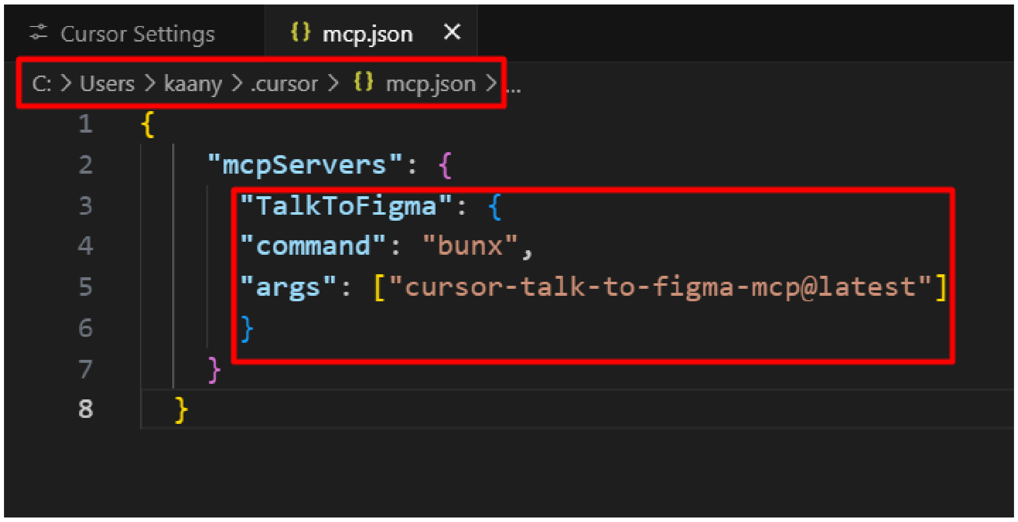
```
which bun
```

Example:

```
"command": "/Users/YOURUSERNAME/.bun/bin/bun"
```

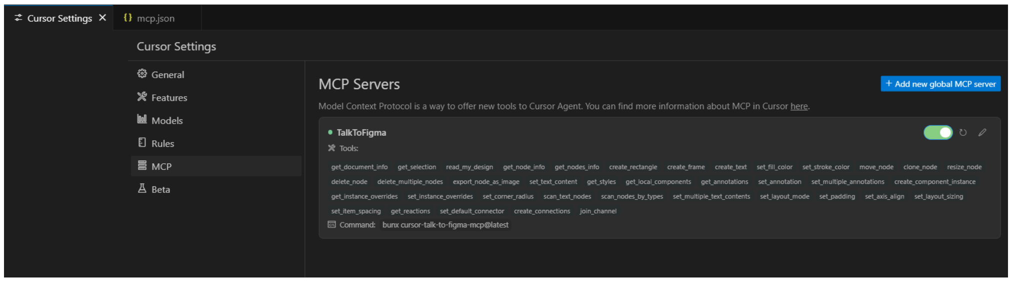
📎 This is required especially on systems where bun isn't available globally inside Cursor's environment.

4. Save the mcp.json file and **restart Cursor**.



```
{
  "mcpServers": {
    "TalkToFigma": {
    "command": "bunx",
    "args": ["cursor-talk-to-figma-mcp@latest"]
    }
  }
}
```

Figure 1: Cursor MCP configuration.



MCP Servers

Model Context Protocol is a way to offer new tools to Cursor Agent. You can find more information about MCP in Cursor here.

TalkToFigma

Tools:

get_document_info get_selection read_my_design get_node_info get_nodes_info create_rectangle create_frame create_text set_fill_color set_stroke_color move_node clone_node resize_node delete_node delete_multiple_nodes export_node_as_image set_text_content get_styles get_local_components get_annotations set_annotation set_multiple_annotations create_component_instance get_instance_overrides set_instance_overrides set_corner_radius scan_text_nodes scan_nodes_by_types set_multiple_text_contents set_layout_mode set_padding set_axis_align set_layout_sizing set_item_spacing get_reactions set_default_connector create_connections join_channel

Command: bunx cursor-talk-to-figma-mcp@latest

Figure 2: Successful server configuration on Cursor.

3. Open Figma and run the Cursor MCP Plugin

Once you've configured both the MCP server and WebSocket server, follow these steps:

1. Launch the Figma Desktop App.
2. In the top menu bar, go to:

   Plugins > Development > Cursor MCP Plugin

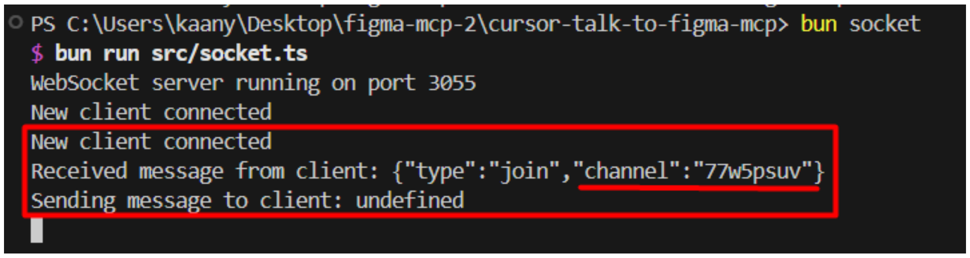
3. Click the plugin to launch it. You'll see a small panel or window pop up.

4. Connect the plugin to the WebSocket server by joining a channel using join_channel

1. In the plugin interface:

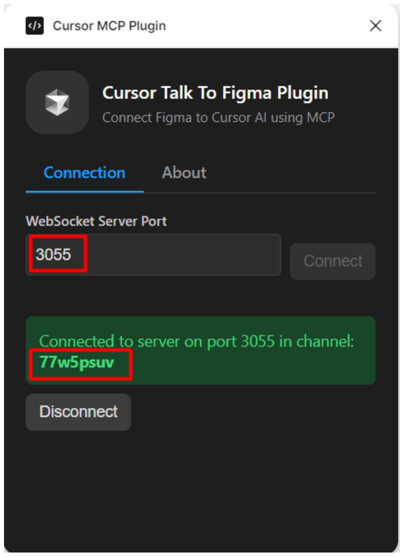
   Click 'Connect' button to connect the "Websocket Server Port" 3055.

2. Now, you should see this message and this view:



```
PS C:\Users\kaany\Desktop\figma-mcp-2\cursor-talk-to-figma-mcp> bun socket
$ bun run src/socket.ts
WebSocket server running on port 3055
New client connected
New client connected
Received message from client: {"type":"join","channel":"77w5psuv"}
Sending message to client: undefined
```

Figure 3: Successful 'Plugin' configuration - terminal message.


Cursor MCP Plugin
Cursor Talk To Figma Plugin
Connect Figma to Cursor AI using MCP
Connection
About
Connect
Disconnect

WebSocket Server Port: 3055

Connected to server on port 3055 in channel: 77w5psuv

Figure 4: Successful ‘Plugin’ configuration - ‘Plugin’ view.

3. Once the plugin joins a channel, it will be able to send and receive instructions from Cursor or OpenServ via MCP.

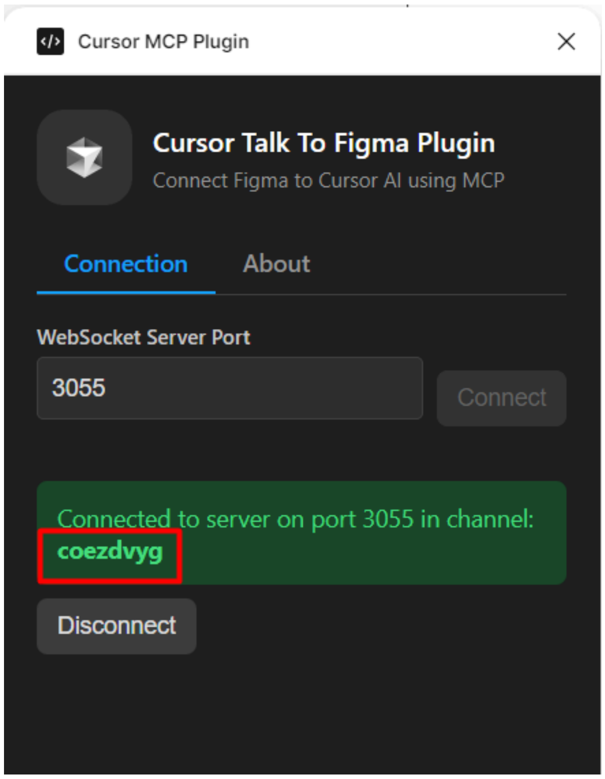
5. Use Cursor to communicate with Figma using the MCP tools

⚠️ Warning:

Make sure the Figma plugin and the MCP agent are connected to the **same channel**.

If they're not on the same channel, they won't be able to communicate with each other.

Also, when using the agent on the OpenServ platform, you will need to provide both the **channel name/ID** and a **Figma file URL**.


Cursor MCP Plugin

Cursor Talk To Figma Plugin

Connect Figma to Cursor AI using MCP

WebSocket Server Port: 3055

Connected to server on port 3055 in channel: **coezdvyg**

Figure 5: Figma plugin example.

🚧 At this point, you can use the Figma MCP with Cursor directly.

To connect it with **OpenServ**, proceed to the next steps to wrap it with SSE and expose it online.

🔁 Step 2: Add SSE Wrapper with mcp-proxy (Middle Layer)

OpenServ MCP Client—and other web-based systems—expect Server-Sent Events (SSE) for communication. The Figma MCP server by default does not support SSE since it runs on stdio.

To bridge this gap, we use mcp-proxy, which wraps the MCP server and exposes it as an SSE-compatible service.

2.1 Install mcp-proxy

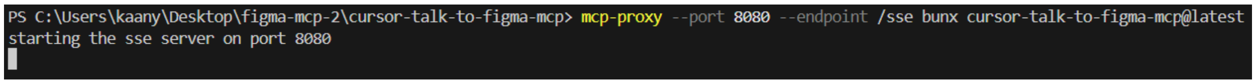
npm install -g mcp-proxy

2.2 Start the SSE Proxy

In a separate terminal, run this command:

mcp-proxy --port 8080 --endpoint /sse bunx cursor-talk-to-figma-mcp@latest

You will see this result:



```
PS C:\Users\kaany\Desktop\figma-mcp-2\cursor-talk-to-figma-mcp> mcp-proxy --port 8080 --endpoint /sse bunx cursor-talk-to-figma-mcp@latest
starting the sse server on port 8080
```

Figure 6: mcp-proxy result.

🌍 Step 3: Expose Publicly with ngrok (Top Layer)

To allow OpenServ or remote systems to connect to your local setup, you can expose the SSE proxy to the internet using ngrok.

So, if you don't have ngrok installed, use https://ngrok.com/ to download, install and configure the ngrok.

3.1 Global Install - npm ngrok

Install ngrok's npm package globally:

npm install -g ngrok

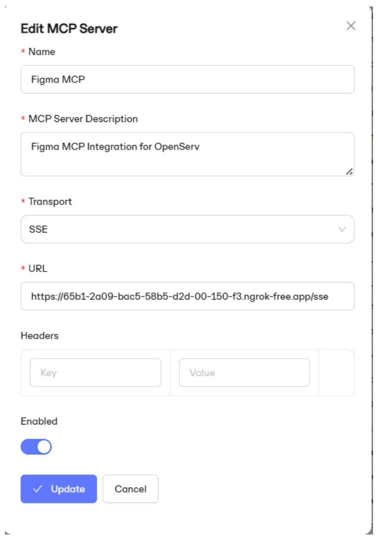
3.2 Start a Public Tunnel

ngrok http 8080

You'll receive a public URL like:

https://1f2d-2a09-bac5-58b6-2682-00-3d6-5.ngrok-free.app

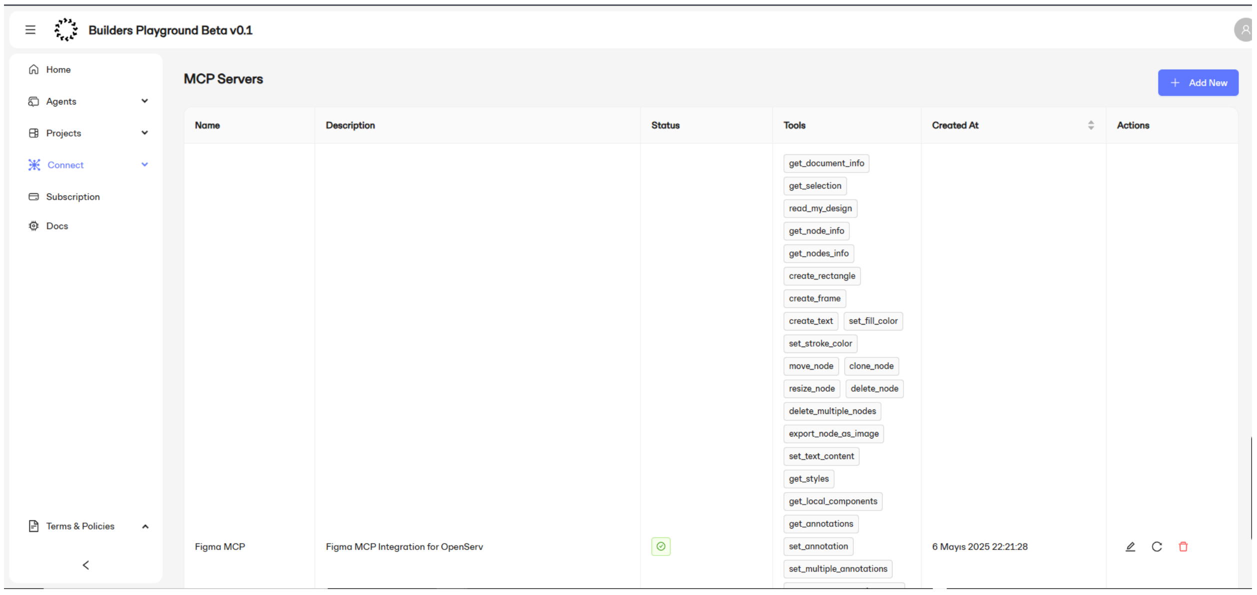
Copy the URL (e.g. https://1f2d-2a09-bac5-58b6-2682-00-3d6-5.ngrok-free.app/**sse**) and register it in OpenServ's MCP Client with **SSE** as the transport protocol.



* Name	Figma MCP
* MCP Server Description	Figma MCP Integration for OpenServ
* Transport	SSE
* URL	https://65b1-2a09-bac5-58b5-d2d-00-150-f3.ngrok-free.app/sse
Headers	Key / Value
Enabled	✓

Figure 7: OpenServ MCP Client connection UI.

This 3-layer architecture—Figma MCP (local), mcp-proxy (SSE wrapper), and ngrok (public exposure)—makes it possible to run the Figma agent locally while integrating with cloud-native tools like OpenServ.



MCP Servers

Name	Description	Status	Tools	Created At	Actions
Figma MCP	Figma MCP Integration for OpenServ	✓	get_document_info, get_selection, read_my_design, get_node_info, get_nodes_info, create_rectangle, create_frame, create_text, set_fill_color, set_stroke_color, move_node, clone_node, resize_node, delete_node, delete_multiple_nodes, export_node_as_image, set_text_content, get_styles, get_local_components, get_annotations, set_annotation, set_multiple_annotations	6 Mayıs 2025 22:21:28	

Figure 8: Final result.

🧰 Troubleshooting

If you encounter a Client closed error or the agent tools do not show up inside Cursor or OpenServ, it's likely due to a misconfiguration related to the Bun runtime path.

👉 To resolve this, refer to this GitHub comment for a quick fix:
https://github.com/sonnylazuardi/cursor-talk-to-figma-mcp/issues/6#issuecomment-2746012517