ChatGPT

Building a Custom ChatGPT Connector with Docker MCP Toolkit

The **Docker MCP Toolkit** streamlines the creation and deployment of custom connectors (MCP servers) for ChatGPT and other AI clients [1]. These connectors use the **Model Context Protocol (MCP)** – an open standard (originally from Anthropic) that defines how LLMs discover and invoke external tools via JSON-RPC calls [2] [3]. In this guide, we'll walk through the entire process: from coding a custom MCP tool, containerizing it, running it locally or remotely (via Docker's MCP Gateway and Cloudflare Tunnel), to registering it so ChatGPT can use it. We'll also cover best practices for secure exposure.

Step 1: Develop a Custom MCP Tool Server

Choose a language and MCP SDK. Official SDKs exist for TypeScript/Node and Python, which greatly simplify building MCP servers [4]. For example, using Python's `mcp` SDK (FastMCP) or Node's `@modelcontextprotocol/sdk`, you can define **tools** (functions the AI can call) and **resources** (data it can fetch) with just a few decorators or method calls.

- **Define MCP server and tools:** In Python, you might do:

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Demo")                  # Create an MCP server instance
@mcp.tool()
def add(a: int, b: int) -> int:
    """Add two numbers"""
    return a + b
```

This registers an `add` tool that simply returns the sum of `a` and `b` [5]. In TypeScript, the pattern is similar – you create an `McpServer` and use `registerTool()` to add tools with input/output schemas [6].

- **Implement the MCP interface:** MCP servers communicate via JSON-RPC 2.0 over a specified transport (e.g. HTTP). If using the SDK, much of this is handled for you. For an HTTP-based server, expose an endpoint (commonly `/mcp`) that will accept JSON-RPC requests. For example, the TypeScript SDK provides a `StreamableHTTPServerTransport` that you attach to an Express route for `/mcp` [7]. In Python, the FastMCP library can run with built-in transports (like stdio for local use, or an HTTP server via ASGI/Uvicorn). The key is to ensure your server listens for POST requests at a URL (say `http://localhost:3000/mcp`) and processes the JSON-RPC messages.

- **(Optional) Add resources or prompts:** MCP also supports **resources** (read-only data endpoints) and **prompts** (predefined prompt templates) if needed [8] [9]. You can register these similarly (e.g.

`@mcp.resource("resource://...")` in Python) to expose data that the AI can retrieve into context. Keep tools idempotent and safe, since they will be executed by the AI.

- **Test locally:** Before containerizing, test your MCP server. You can run it on a local port and use the MCP Inspector tool to verify it works: for example, run `npx @modelcontextprotocol/inspector` and connect to your server's URL to see the listed tools [10]. Ensure the basic requests work (e.g. calling the `add` tool returns the expected JSON result).

Step 2: Containerize the MCP Server with Docker

Packaging your connector as a Docker image ensures it runs consistently everywhere [11]. We will create a Dockerfile for your MCP server:

- **Write a Dockerfile:** Use an official runtime base image (Python or Node). For example, if our server is in Python and listens on port 3000:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt  # Install MCP SDK and deps
COPY . .                                            # Copy your server code
EXPOSE 3000
CMD ["python", "server.py"]                         # Start the MCP server
```

This Dockerfile installs dependencies (including the `mcp` Python SDK) and runs your `server.py`. For Node/TypeScript, you'd similarly copy `package.json` and use `npm install`, then run the server (possibly via `node server.js` or `tsx`). **Tip:** Create a non-root user in the image for better security (as shown in many Docker examples [12]).

- **Build the image locally:** Run `docker build -t mytool-mcp:latest .` in the project directory. This produces a local image for your connector.

- **(Optional) Push to a registry:** If you want to reuse or share this image, push it to Docker Hub or another registry. For example, tag and push to Docker Hub:

```
docker tag mytool-mcp:latest yourdockeruser/mytool-mcp:latest
docker login   # (if not already logged in)
docker push yourdockeruser/mytool-mcp:latest
```

This makes it easier to deploy remotely or let others pull it [13]. (For local-only use, pushing is not necessary.)

Step 3: Run the MCP Connector Locally with Docker MCP Toolkit

Now that you have an image, you can run it through Docker's MCP Toolkit/Gateway, which adds a secure layer and allows ChatGPT (or other clients) to connect.

Prerequisites: Ensure you have Docker Desktop updated (v4.42+ on Win or 4.40+ on Mac) and **enable the MCP Toolkit** in Settings → *Beta features* [14]. This will activate the MCP Catalog & Toolkit in Docker Desktop, and install the `docker mcp` CLI plugin.

There are two ways to launch your MCP server locally:

Option A: Via Docker Desktop UI (Toolkit): Open Docker Desktop and go to the **MCP Toolkit** tab. Normally, you can browse the **Catalog** for official servers and click "+" to add one [15]. For a custom server not in the official catalog, you'll need to add it manually. Currently, the UI doesn't have an "import custom image" button, so you will use the CLI (Option B) or edit config files.

Option B: Via Docker MCP CLI (Gateway): The `docker mcp` CLI lets you enable custom servers and run the gateway. Here's how:

1. **Register your server in the catalog:** By default, Docker uses the **"docker-mcp" catalog** (the official list of servers) [16] [17]. You can add a custom entry. One approach is to create a small YAML file (e.g. `~/.docker/mcp/custom.yaml`) with your server definition, for example:

```
mytool:
  image: "yourdockeruser/mytool-mcp:latest"
  transports: ["http"]   # or ["sse"] depending on your server's supported
transport
```

This defines a new server named "mytool" with the specified image. Save this and **initialize the catalog** if needed (e.g., `docker mcp catalog init` for default, or ensure the custom file is referenced in config). The MCP CLI uses config files in `~/.docker/mcp/` (including `docker-mcp.yaml` for catalogs, `registry.yaml` for enabled servers) [18]. You can merge your custom entry into those or specify it at runtime.

1. **Enable the server:** Use the CLI to enable your custom server so it will start. For example:

```
docker mcp server enable mytool
```

This marks "mytool" as enabled (similar to adding via UI). You can list enabled servers with `docker mcp server ls` [19]. If everything is set up, you should see `mytool` in the list (possibly alongside any other servers).

1. **Launch the MCP Gateway:** Start the gateway which will actually run the container and expose a unified endpoint. A basic invocation is:

```
docker mcp gateway run
```

This starts the gateway with default settings (likely on localhost and using the default transport, typically SSE for streaming or HTTP). The gateway will launch your container in the background with secure defaults (no host mounts, limited CPU/RAM) and listen for client connections. By default, all enabled servers become available through this gateway [20].

- *Customizing the gateway:* You can specify transport and limit tools if needed. For example, to use SSE transport and only allow certain tools, Docker's docs show: `docker mcp gateway run --transport=sse --servers=mytool --tools=tool1,tool2` [20] [21]. For basic use, the default should suffice. The gateway will typically print out what address it's listening on. For instance, it might say it's running an HTTP endpoint on `http://127.0.0.1:41114` (or similar port), or an SSE endpoint at `http://127.0.0.1:41114/sse`.

- **Connect a local client (optional):** If you are using a local AI client like VS Code's Copilot Chat or Claude Desktop, you can point it to this local gateway. For example, VS Code's Agent mode can connect via a generated `mcp.json` pointing to Docker's gateway [22] [23]. In Docker Desktop's MCP Toolkit UI, your local client (if supported) would appear under "Clients" – e.g. you might see options to connect VSCode or Claude [24]. *(At the time of writing, ChatGPT Desktop does not yet have a built-in way to auto-connect to the local gateway. It's essentially the ChatGPT web app in an Electron shell, so it doesn't automatically interface with Docker. You will likely treat ChatGPT as a remote client – see Step 4.)*

At this point, your MCP server is running locally in Docker. The gateway aggregates it (and any other enabled MCP servers) and exposes them to clients. The **security sandboxing** is in effect: Docker limits each MCP container to 1 CPU and 2GB RAM, and blocks host filesystem access by default [25], so your tool runs safely. You can inspect available tools via CLI (`docker mcp tools ls`) or logs to verify it's functioning.

Step 4: (Optional) Deploy or Expose the Connector for Remote Access

If you want ChatGPT (web) or others to use your connector, the MCP server needs to be accessible over the internet (ChatGPT's servers can't reach your localhost behind NAT [26]). You have a couple of options:

- **Deploy to a VPS or cloud service:** Since you containerized the server, you can run it on any cloud host. For example, use a service like Render or Railway by providing your Docker image [27] [28]. Ensure the container is running and reachable at a public URL (with HTTPS). Configure it to serve on a known port (e.g. 80/443 or via a proxy) so that you have a stable URL like `https://your-tool.example.com/mcp`.

- **Use Cloudflare Tunnel for a local server:** If you prefer not to host a server publicly, Cloudflare Tunnel is a quick way to expose your running local MCP gateway or server to the internet securely. Using Cloudflare's free Tunnel, you can get a public `https` URL that forwards to your `localhost:<port>`. For example, one user shares: *"I self host on my server in a container and expose through a Cloudflare tunnel."* [29] This means you can run `cloudflared` to connect your local port to a Cloudflare address. Steps in brief:

- Install `cloudflared` (Cloudflare Tunnel client) on your machine (or run it as a Docker container).

- Authenticate it with your Cloudflare account (or use the one-time `cloudflared tunnel --url http://localhost:41114` for a quick, temporary tunnel).
- Cloudflare will provide a public URL (e.g. `https://your-tunnel-name.trycloudflare.com`) that routes to your local MCP server port. You can also set up a custom domain with Cloudflare DNS for a cleaner URL [30] [31].

The tunnel approach gives you **HTTPS** for free, and you can add Cloudflare Zero Trust policies if desired (e.g. require login to access the URL, adding another security layer). If using the Docker MCP Gateway, ensure it's binding to an address accessible to cloudflared (by default it listens on localhost – which is fine for the tunnel client running on the same host).

With a remote URL ready (either from a cloud deploy or Cloudflare Tunnel), you now have an **internet-accessible MCP endpoint**. For instance, you might have `https://mytool.example.com/mcp` or an SSE endpoint `https://mytool.example.com/sse` (depending on what transport you set up).

Step 5: Register the Connector in ChatGPT (MCP Catalog & Clients)

Finally, let's **register the connector** so that ChatGPT (or any MCP-compatible client) can discover and use it. There are two scenarios:

A. Using ChatGPT (web app) – Developer Mode Connector: OpenAI recently introduced custom connector support (in ChatGPT's **Developer Mode**). Here's how to add your tool:

1. **Enable Developer Mode:** In ChatGPT settings, go to *"Connectors"* and enable Developer Mode (beta) [32] [33] if not already on.
2. **Create a New Connector:** Click "Create" under Browser Connectors. In the dialog, fill in:
3. **Name:** A friendly name for your tool (e.g. "My Custom Tool").
4. **MCP Server URL:** The full URL where your MCP server is running (including the `/mcp` or `/sse` path). For example, `https://mytool.example.com/mcp`. Make sure to use `https://` – ChatGPT will likely reject `http://` non-secure URLs.
5. **Authentication:** If your MCP server requires OAuth (like for user-specific data), you'd select OAuth and follow the flow. For a simple custom tool that doesn't require per-user login, choose "None" or the appropriate option. *(Many community connectors use OAuth for safety – e.g., if the tool accesses a user's Google Drive, it should use OAuth. For a self-contained utility tool, you might not need auth.)*

6. You can also add an icon and description if you like [34].

7. **Trust and Connect:** ChatGPT will show a security notice (reminding you that custom connectors are not vetted). Acknowledge this and proceed. If you selected OAuth, it will redirect you to authenticate and authorize ChatGPT to access your tool (this requires your MCP server to implement the OAuth handshake). If no auth, it will connect directly [35].

8. **Verify tools:** After connecting, ChatGPT should list the **Actions** (tools) provided by your server [36]. For example, you should see "add" or whatever tools you defined. Now, in the chat interface, you can

instruct ChatGPT to use these tools. For instance, ask: *"Use the* `add` *tool to calculate 2+3."* ChatGPT should call your connector's tool and return the result.

B. Using ChatGPT Desktop or Other GPT Clients: If you have the ChatGPT desktop app, the process is similar (it also has settings for connectors). However, as of now the desktop app may not support local-only connectors without an internet URL. In practice, you would still add it via the Connectors UI (as above) pointing to an accessible URL. Other clients like **Claude Desktop, Cursor, VS Code, etc.** can often directly integrate with the Docker MCP Gateway. For example, in Claude Desktop you can add a custom connector by providing the local gateway URL and it will list available tools [37] [38]. VS Code's "AI Tools" can connect by running `docker mcp client connect vscode`, which hooks VS Code to the local gateway [22] [39]. In summary, any client that supports MCP can be pointed at your connector's URL or the Docker gateway. ChatGPT (web) specifically requires a public URL as described.

Add to MCP Catalog (optional): If you intend to share your connector with the community, consider publishing it in Docker's **MCP Catalog**. Currently, Docker has a PR-based process for adding new official servers to the catalog [40]. This would make your tool appear in the Docker Desktop UI for anyone to one-click install. To do this, you'd contribute a definition (usually under the `mcp/` namespace on Docker Hub, with a signed image). This step is optional – for private use, you can simply keep it in your local catalog or share the image and instructions with colleagues.

Step 6: Securely Expose and Interact with the Connector

Security is crucial when allowing an AI to execute tools. Docker's MCP Toolkit provides several built-in safeguards:

- **Container isolation and limits:** Your MCP tool runs in a locked-down container. By default it has no access to your host filesystem, and is constrained to 1 CPU core and 2GB RAM [25]. This prevents a runaway tool from hogging resources or snooping on your files. The container is labeled and managed by Docker, and you only explicitly mount volumes or increase limits if you choose to (for example, if you enable the `filesystem` MCP server, you must configure which host directory it can access [41]).

- **Secrets management:** If your connector needs API keys or tokens (for accessing third-party APIs), avoid baking them into the image. The Docker MCP CLI lets you inject secrets at runtime. For instance, to provide an API key for a custom "Brave Search" MCP server, you'd run `docker mcp secret set 'brave.api_key=XXXXX'` before starting it [42]. The gateway will pass it to the container as an environment variable, keeping the secret out of code and version control.

- **OAuth support:** The MCP Gateway and ChatGPT both support OAuth flows for connectors. If your tool requires user authorization (e.g., a Google Drive connector), implement the OAuth 2.0 code flow in your MCP server (the OpenAI connector spec supports this). The Docker toolkit can handle port-forwarding for OAuth callbacks and even dynamic client registration in some cases. For example, the GitHub MCP server uses OAuth – Docker's toolkit helps initiate that flow and stores tokens securely [43] [44]. If you marked the connector as "OAuth" in ChatGPT, ChatGPT will expect to go through a browser login and callback as part of adding the connector.

- **Active monitoring:** The open-source MCP Gateway offers **interceptors** to enforce security at runtime [45]. For example, you can run the gateway with `--verify-signatures` (to only run Docker-signed images), `--block-secrets` (to scan and block any traffic that looks like sensitive info such as API keys), and `--log-calls` (to record all tool usage) [46]. These can be enabled as needed to guard against misuse. Additionally, ChatGPT itself wraps tool outputs and will warn if a tool tries to output disallowed content.

- **Transport security:** Always use HTTPS for the MCP connection if not on localhost. Cloudflare Tunnel or a TLS reverse proxy is essential if you're exposing to the internet, so that the traffic between ChatGPT and your server is encrypted. If you use Cloudflare Zero Trust, you can even put your MCP endpoint behind an Access policy (requiring a login or token to reach it), effectively restricting who can call your tool.

Finally, **test the end-to-end interaction carefully**. In ChatGPT, try out your connector with both valid and invalid inputs to see how it responds. Check Docker logs (`docker logs <container>`) or the gateway output for any errors or unexpected behavior. Iterate on your code if needed to handle edge cases. Remember that ChatGPT will formulate the tool calls, so ensure your tool's input schema and descriptions are clear to the model (the SDK usually handles exposing these details to the AI).

By following these steps, you have created a custom MCP-compatible tool, packaged it in Docker, and made it available to ChatGPT. You can run it locally for personal use or host it remotely for broader access. With the Docker MCP Toolkit managing the heavy lifting (container orchestration, security, and client compatibility), integrating custom tools into ChatGPT becomes much easier [47]. This opens up exciting possibilities – from connecting to internal data sources to performing actions on your behalf – all through the familiar ChatGPT interface, but powered by your custom code. Enjoy your new connector, and hack responsibly!

Sources:

1. Docker Docs – *MCP Toolkit Overview* [1] [25]
2. Docker MCP Gateway GitHub – *CLI Usage and Examples* [20] [46]
3. ChatGPT Dev Mode Tutorial (GitHub Gist) – *Setting up a Connector* [34] [36]
4. Model Context Protocol SDK (TypeScript/Python) – *Quickstart Examples* [6] [5]
5. *The Complete MCP Server Guide* (DevShorts) – *Containerizing & Deploying MCP Servers* [13] [28]
6. Reddit / Community Discussions – *Exposing MCP Servers (Cloudflare Tunnel tips)* [29]

---

[1] [22] [23] [24] [25] [37] [38] [39] [43] [44] [47] MCP Toolkit | Docker Docs
https://docs.docker.com/ai/mcp-catalog-and-toolkit/toolkit/

[2] [14] [15] Run Any MCP Server Locally with Docker's MCP Catalog and Toolkit | by Ali Ibrahim | Medium
https://techwithibrahim.medium.com/run-any-mcp-server-locally-with-dockers-mcp-catalog-and-toolkit-f8ecbe0c6a79

[3] [32] [33] [34] [35] [36] ChatGPT App SDK & MCP Developer Mode MCP - Complete Tutorial · GitHub
https://gist.github.com/ruvnet/7b6843c457822cbcf42fc4aa635eadbb

[4] MCP SDK - Hugging Face MCP Course
https://huggingface.co/learn/mcp-course/en/unit1/sdk

5 9 GitHub - modelcontextprotocol/python-sdk: The official Python SDK for Model Context Protocol servers and clients
https://github.com/modelcontextprotocol/python-sdk

6 7 8 10 GitHub - modelcontextprotocol/typescript-sdk: The official TypeScript SDK for Model Context Protocol servers and clients
https://github.com/modelcontextprotocol/typescript-sdk/tree/main?tab=readme-ov-file

11 12 13 27 28 The Complete MCP Server Guide - by Aravind Putrevu
https://www.devshorts.in/p/the-complete-mcp-server-guide

16 17 18 19 GitHub - docker/mcp-gateway: docker mcp CLI plugin / MCP Gateway
https://github.com/docker/mcp-gateway

20 21 40 41 42 45 46 Docker MCP Gateway: Unified, Secure Infrastructure for Agentic AI Docker MCP Gateway: Open Source, Secure Infrastructure for Agentic AI | Docker
https://www.docker.com/blog/docker-mcp-gateway-secure-infrastructure-for-agentic-ai/

26 Connecting Local MCP Servers to Publicly Exposed LLM APIs: A Complete Guide | by ThamizhElango Natarajan | Medium
https://thamizhelango.medium.com/connecting-local-mcp-servers-to-publicly-exposed-llm-apis-a-complete-guide-12658f464e51

29 How to turn local MCP server into remote one? - Reddit
https://www.reddit.com/r/mcp/comments/1l6y9za/how_to_turn_local_mcp_server_into_remote_one/

30 31 Setup your Domain using Cloudflare Tunnels and Zero Trust
https://noted.lol/cloudflare-tunnel-and-zero-trust/