CupCarbon MCP Server

Designing networks with Claude Desktop

For CupCarbon Klaines 8.0 — cupcarbon.com — 2026

> **In one sentence:** with this server, you open CupCarbon, open Claude Desktop, and type *"add 10 IoT nodes in a ring around the university and run the simulation"* — Claude does it, live, on your map.

Contents

1 What it is

The **Model Context Protocol (MCP)** is the standard that lets Claude use external tools. The CupCarbon MCP server exposes the simulator as a set of 25 tools (add nodes, move, list, run/stop, scripts, workflows...). When you ask something in Claude Desktop, Claude decides which tools to call; the server translates every call into a CupCarbon *command console* command and sends it over MQTT — the same protocol used by the CupCarbon web console:

```
Claude Desktop
     |  MCP (stdio, local)
```

```
     v
cupcarbon_mcp.py  --MQTT-->  broker.emqx.io  --MQTT-->  CupCarbon (command console)
```

Because the link goes through the broker, Claude and CupCarbon do not need to run on the same machine. Nothing changes in CupCarbon itself.

Security. At every launch, CupCarbon generates a random session topic (e.g. `cupcarbon/session3fa29c8b12d`). Only someone who holds this topic can drive your simulator; it is never published anywhere.

2 Prerequisites

- **CupCarbon Klaines 8.0** installed and running (see the Installation Guide);
- **Claude Desktop** (claude.ai/download) with a Claude account;
- **Python 3.10 or newer** (`python3 -version`);
- an internet connection (both sides must reach the MQTT broker, `broker.emqx.io` by default).

3 Installing the server

The server is the `mcp_server/` folder of the CupCarbon distribution, and is also downloadable as `cupcarbon_mcp_server.zip` from cupcarbon.com/claude (with the Claude skills and this guide):

`cupcarbon_mcp.py`	the MCP server (a single Python file)
`requirements.txt`	its two dependencies: `mcp`, `paho-mqtt`
`README.md`	quick reference

Install the dependencies once:

```
cd /path/to/CupCarbon/mcp_server
pip3 install -r requirements.txt
```

> If `pip3` complains about an "externally managed environment" (recent Linux/macOS), create a virtual environment first: `python3 -m venv venv && ./venv/bin/pip install -r requirements.txt`, then use `/path/to/mcp_server/venv/bin/python3` as the command in the next section.

4 Registering the server in Claude Desktop

Claude Desktop reads its MCP servers from a configuration file:

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

You can also reach it from Claude Desktop: **Settings** → **Developer** → **Edit Config**. Add the `cupcarbon` entry (create the file if it does not exist), replacing the path by yours:

```
{
  "mcpServers": {
    "cupcarbon": {
      "command": "python3",
      "args": ["/Users/me/CupCarbon/mcp_server/cupcarbon_mcp.py"]
    }
  }
}
```

Then **quit and restart Claude Desktop** (fully quit — not just the window). In a new conversation, the tools button (a small hammer or plug icon under the message box) should now list the `cupcarbon` tools.

> **Windows:** use `"command": "python"` and double the backslashes in the path: `"C:\\Users\\me\\..."`. If Claude says the server failed to start, the path or the Python command is almost always the culprit — test it in a terminal first: `python3 /path/to/cupcarbon_mcp.py` should start silently.

4.1 Optional environment variables

Add an `"env"` block to preconfigure the connection — useful if you do not want to paste the topic in the chat every time:

```
"cupcarbon": {
  "command": "python3",
  "args": ["/Users/me/CupCarbon/mcp_server/cupcarbon_mcp.py"],
  "env": {
    "CUPCARBON_TOPIC":  "cupcarbon/sessionXXXXXXXXXXXX/console",
    "CUPCARBON_BROKER": "broker.emqx.io",
    "CUPCARBON_PORT":   "1883"
  }
}
```

Remember that the topic changes at every CupCarbon launch, so the chat method (next section) is usually more practical.

5 Connecting Claude to CupCarbon

1. Start **CupCarbon** and open (or create) a project.
2. Click the **Command** button of the toolbar: the command console opens and connects to the MQTT broker.
3. Click **Copy Topic** — the session topic is now in your clipboard.
4. In **Claude Desktop**, start the conversation with:

> *"Connect to CupCarbon"*

Thanks to the discovery mechanism (the console answers requests on a fixed `cupcarbon/discovery` topic), Claude finds the session topic by itself — no pasting. If several CupCarbons answer on the broker, or to target a precise one, use Copy Topic and say instead:

> *"Connect to CupCarbon with the topic cupcarbon/session3fa29c8b12de/console"*

Claude calls `cupcarbon_connect`; the answer shows the connection and the simulator status (open project, number of nodes...). You are ready.

> The topic is **regenerated at every CupCarbon launch**. If you restart CupCarbon, click Copy Topic again and give Claude the new value. The most frequent cause of "no answer from CupCarbon" timeouts is an old topic.

6 The tools

Claude chooses the tools by itself — you never call them manually. Knowing them helps you phrase requests:

Tool	Role
`cupcarbon_autoconnect(broker?, port?)`	connect *without pasting the topic* (MQTT discovery)
`cupcarbon_connect(topic, broker?, port?)`	connect to the running CupCarbon
`get_topic()`	returns the console topic (gettopic / discovery)
`cupcarbon_status()`	project, counters, simulation state
`cupcarbon_help()`	the full console command reference
`cupcarbon_command(cmd)`	any raw console command (escape hatch)
`project_open(path)` / `project_save()`	open / save a project
`add_node(type, lat, lon)`	add one object: iot, sensor, base, gas, mobile, marker, weather
`design_network(nodes)`	add a whole topology in one call
`move_node(id, lat, lon)`	move a node
`delete_object(sel)`	delete: `device <id>`, `marker <i>`, or the selection
`delete_all_markers()`	remove every marker in one shot
`set_icon(id, n)`	change the icon of an IoT node
`draw_routes(show)`	draw / hide all the routes on the map
`list_objects(kind)`	sensors, devices (IoT included), markers, routes
`set_view(zoom, center)`	zoom in/out, recenter the map
`write_script(file, code)`	create a script in the project (short scripts)
`assign_script(id, file)`	assign a script to a node
`run_simulation()` / `stop_simulation()`	start/stop (IoT or WSN, per project type)
`mark_node(id, marked)`	highlight a node
`workflow_list/get/put`	read and write the agentic AI workflows (JSON)

> **One environment per project.** A project contains either IoT nodes or classical WSN sensor nodes, never both — Claude knows it, but if you ask for a mix, CupCarbon will refuse the second family with a clear message.

7 Step-by-step: your first AI-designed network

With CupCarbon connected (previous section) and a new empty project:

1. Design

> *"Add 6 IoT nodes forming a ring of about 200 m of radius around 25.3020, 55.4855, plus one node at the center."*

Claude computes the positions (about 0.001° of latitude per 111 m) and calls `design_network`; the seven nodes appear on the map.

2. Verify

> *"List the devices and give me a table with id and position."*

Claude calls `list_objects("devices")` and formats the answer.

3. Program

> *"Write a script blink.py that marks and unmarks the node every second with cup_mark/cup_unmark, and assign it to all the nodes."*

Claude calls `write_script` then `assign_script` for each id. For long programs, write the script in CupCarbon's editor instead and just ask Claude to assign it.

4. Simulate

> *"Run the simulation."*

The nodes start blinking on the map. Then:

> *"Stop the simulation and mark only the center node."*

5. Iterate

> *"Move node 4 fifty meters to the north and double the ring: add 6 more nodes between the existing ones."*

This is where the workflow shines — you design by conversation while watching the map.

8 Prompt cookbook

- *"Create a star network of 10 IoT nodes around 48.39, -4.48, 600 m of radius, and mark the center."*

- *"Add a gas source at 25.301, 55.484 and three sensors of the map border around it."* (WSN project)
- *"Zoom out twice and center the map on the network."*
- *"Show me the workflow list, download 'assistant' and explain what it does."*
- *"Modify the workflow 'assistant': change the chid of the CupCarbon Read node to 12, and upload it back."*
- *"Give me the full console command list."* (uses `cupcarbon_help`)

9 How it works (under the hood)

The server publishes every command as JSON on `<topic>/commands`:

```
{"command": "add iot 25.3020 55.4855", "clientId": "claude_mcp_ab12cd34ef"}
```

CupCarbon executes it and answers on `<topic>/responses`:

```
{"originalCommand": "add iot 25.3020 55.4855",
 "response": "IoT node added at (25.3020, 55.4855)",
 "requestingClientId": "claude_mcp_ab12cd34ef", ...}
```

The server correlates answers by `requestingClientId` + `originalCommand`, sends one command at a time, and times out after 25 s (configurable with `CUPCARBON_TIMEOUT`). This is exactly the protocol of the CupCarbon web console — anything the web console can do, Claude can do.

10 Troubleshooting

Symptom	Fix
The cupcarbon tools do not appear in Claude Desktop	check the config file path and JSON syntax; fully quit and restart Claude Desktop; test `python3 cupcarbon_mcp.py` in a terminal (it must start without error).
"Server failed to start"	wrong `command`/path, or dependencies not installed for THAT python: `pip3 install -r requirements.txt` with the same interpreter (or use the venv python path).
"Cannot reach the MQTT broker"	no internet, or a firewall blocks port 1883; try from the same network as a browser MQTT client.
"No answer from CupCarbon after 25s"	CupCarbon not running, command console not opened (click **Command**), or — most often — the topic is from a previous CupCarbon launch: click Copy Topic again and reconnect.
Adding a node is refused	IoT/WSN exclusivity: the project already contains the other family.
Claude Desktop logs	macOS: `~/Library/Logs/Claude/mcp*.log`; Windows: `%APPDATA%\Claude\logs`.

11 Alternatives

Claude Code (terminal): `claude mcp add cupcarbon - python3 /path/to/cupcarbon_mcp.py` — same tools, same usage.

Claude Skills: the `cupcarbon-coding` and `cupcarbon-senscript` skills (same download page) make Claude fluent in the node programming languages; combined with this server, Claude both writes and deploys the scripts.