
Two-way MCP for ChatGPT

Before, you called your MCP server. Now your MCP server can start the conversation. This guide takes one event from your own work and gets it to land in a ChatGPT Work chat, built with Claude Code or Codex.

START HERE

What changed, and where this guide takes you

You already work with an agent that calls tools. This guide adds the other direction: something happens in your work, and ChatGPT starts on it before you type a word.

What you already do

In Claude Code or Codex you type a prompt, the agent calls a tool (a file read, a command, an MCP server), the tool answers, and the agent carries on. Every exchange starts with you or the agent. The tool waits to be asked.

What is new

OpenAI added MCP Events to ChatGPT. An MCP server can now tell ChatGPT that something happened, and ChatGPT acts on it in the chat that asked to be told. The feature shipped in ChatGPT's release notes of September 29, 2026 and is listed among OpenAI's DevDay 2026 launches.

BEFORE

you -> agent -> tool -> answer

The agent starts each exchange. The tool waits.

NOW

event -> MCP server -> Work chat -> task

The MCP server starts the exchange. You type nothing.

What you will have at the end

- ☐ An MCP server on your computer that offers one event you chose and one read-only tool
- ☐ A ChatGPT plugin pointed at that server, and a ChatGPT Work chat subscribed to your event
- ☐ A fire command that sends a signed test event, and ChatGPT doing your task when it lands
- ☐ A short hook that shows how your real app, script or CI job fires the same event
- ☐ A plan for turning the test into something you keep (the second to last section)

Where it runs

You build the server with Claude Code or Codex. It runs only in a ChatGPT Work chat. OpenAI's docs say MCP Events are available "in Work chats on ChatGPT web, in Work chats in the desktop app with Cloud selected, and with dots". Those are the places the docs list, and this guide uses the first one.

What was tested, and what was not

Jason at GraniteAI built this on 6 October 2026: one Python file of about 210 lines, standard library only, written by Claude Code on Windows, with `c1oudf1ared` for the tunnel and the bug report event from the reel. Events arrived in a ChatGPT Work chat.

Not tested: Codex (the prompt was run in Claude Code only), Mac and Linux, other tunnel programs, and the opening of the prompt in this guide that asks you to choose your own event. The specification is a draft. Names and fields can change, so the prompt tells your agent to read OpenAI's pages first.

What you need, and six ideas in learning order

On your computer and in your accounts

- ☐ **Claude Code or Codex**, installed and in use. This guide calls either one "your agent". New to Claude Code? Read Anthropic's [quickstart](#), and keep our [Claude Code commands reference](#) open.
- ☐ **Python 3**. Check with the first command below.
- ☐ **cloudflared**, the program that opens the tunnel. Cloudflare's [Quick Tunnels](#) page has the install links. A Quick Tunnel needs no Cloudflare account and no domain.
- ☐ **A ChatGPT account that can add a custom MCP server**. OpenAI names no plans. Its docs say: "Account and workspace policies apply to adding and using custom MCP servers."

Check both programs. Run in your terminal. On Mac or Linux the first command may be `python3` (untested).

```
python --version
cloudflared --version
```

Six ideas, each one built on the one before

Read from the top and stop at the first idea that is new to you. Each has a primer from the people who own it. You will not write the code yourself, your agent does that, but you need these six to follow what it builds and to find the cause when something fails.

1. MCP

The Model Context Protocol is a standard for connecting an AI application to outside tools and data. Claude Code, Codex and ChatGPT all speak it.

Primer: modelcontextprotocol.io/docs/getting-started/intro

2. MCP server and plugin

An MCP server is a program that offers tools to an AI application. A plugin is ChatGPT's name for an MCP server you have added to your account. The reel says "connector" for the same thing. This guide says plugin, because that is the word on ChatGPT's screens.

Primers: code.claude.com/docs/en/mcp and OpenAI's [plugins quickstart](#)

3. Webhook

A webhook is a message one program sends to a web address the moment something happens, with no one asking for it each time. The address that receives it is the callback URL. MCP Events in ChatGPT are delivered this way and no other.

Primer: GitHub's [About webhooks](#)

4. Public HTTPS URL and tunnel

ChatGPT runs on the internet and your test server runs on your computer, at an address like `127.0.0.1` that only your computer can reach. A tunnel lends your computer a temporary public address that starts with `https://`.

Primer: Cloudflare's [Quick Tunnels](#)

5. Signing secret

A signing secret is a key ChatGPT hands your server when it subscribes. Your server uses it to compute a signature for every webhook, so the receiver can check who sent it and that nothing changed on the way. Whoever holds the secret can sign as your server, so it stays out of chats, logs and git.

Primer: standardwebhooks.com

6. Work chat

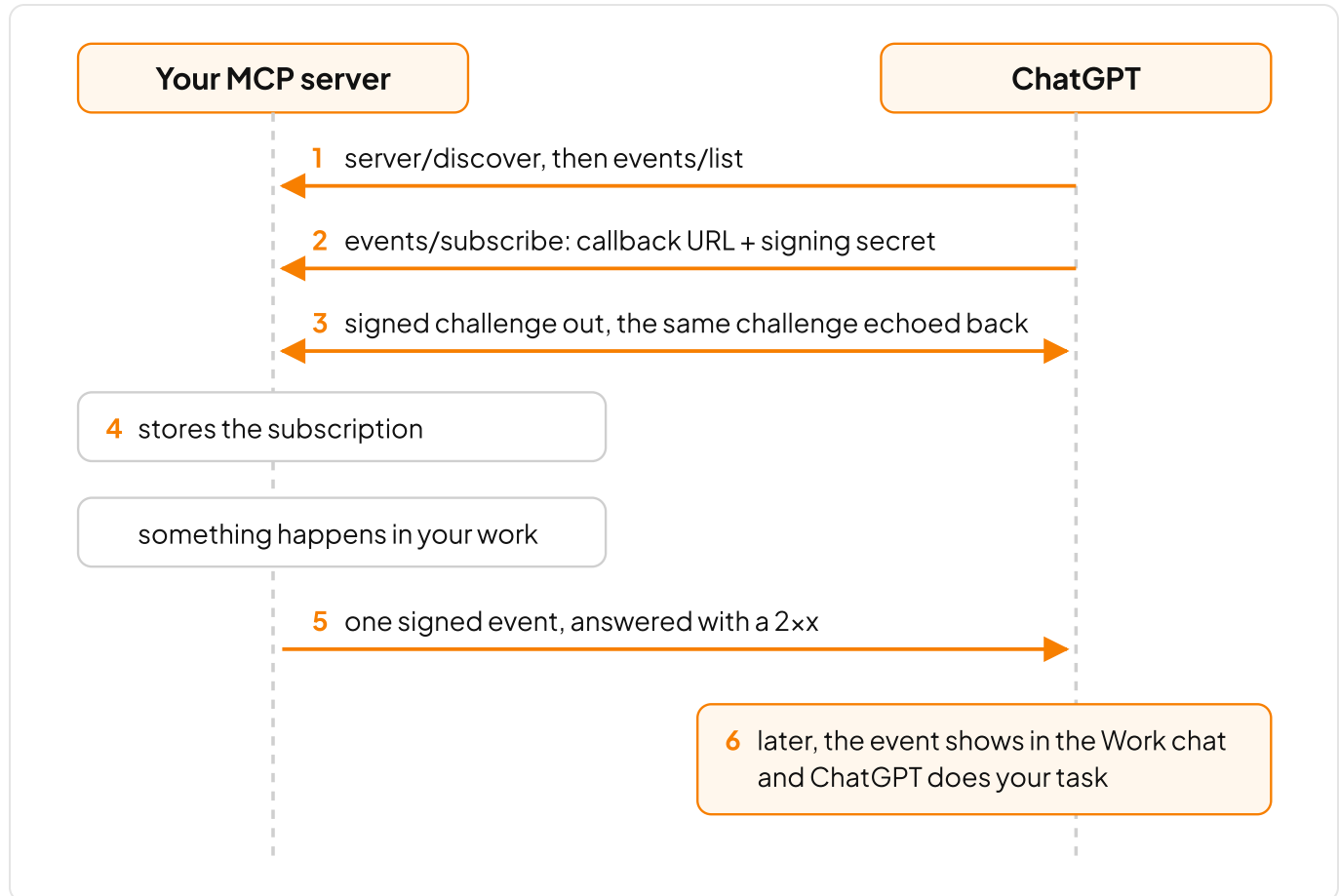
ChatGPT's home page has a switch between Chat and Work. A ChatGPT Work chat is a chat started with that switch on Work. It is where a subscription lives and where its events arrive.

Primer: OpenAI's [plugins quickstart](#) shows the switch.

HOW IT WORKS

The round trip, one arrow at a time

Two things talk: your MCP server and ChatGPT. Arrows 1 and 2 come in to your server through the tunnel. Arrows 3 and 5 go out from your server to the callback URL ChatGPT supplied.



1. **Discover.** You add your server to ChatGPT as a plugin. ChatGPT calls `server/discover`, sees `events` among the server's capabilities, and calls `events/list` to learn the event names.
2. **Subscribe.** In a Work chat you name the event and the task. OpenAI's words: "ChatGPT subscribes through your MCP server and supplies a callback URL and signing secret."
3. **Verify.** Before any real data moves, your server sends a signed challenge to the callback URL. ChatGPT echoes the challenge in a successful response, and your server compares the two.
4. **Store.** Your server saves the subscription: which event, which callback URL, which secret, and when it expires.
5. **Deliver.** Something happens. Your server signs one event and posts it to the callback URL. "A 2xx response acknowledges webhook receipt. ChatGPT processes the event asynchronously."
6. **Act.** The event shows up in the Work chat that subscribed and ChatGPT does the task. In Jason's test that was about five minutes after the 200.

The figures, with a plain meaning beside each

Figure	What OpenAI's docs say	In plain words
Protocol version	MCP 2.0, protocol version <code>2026-07-28</code> .	A protocol version is a dated set of rules. In Jason's test a connection on the older <code>2025-06-18</code> got tools and no events.
Capability	The server adds <code>events</code> to its capabilities in the <code>server/discover</code> response.	This is the line that tells ChatGPT the server can send events at all.
Three methods	<code>events/list</code> , <code>events/subscribe</code> , <code>events/unsubscribe</code> .	Show me your events, start sending me this one, stop.
An event listing	Each entry needs <code>name</code> , <code>description</code> , <code>delivery: ["webhook"]</code> , <code>inputSchema</code> and <code>payloadSchema</code> .	The name and description are how ChatGPT matches your sentence to an event, so write them for a reader.
Delivery	Webhook only. Polling, streaming, and the draft's <code>gap</code> and <code>terminated</code> control notifications are not supported.	Your server pushes each event. ChatGPT never comes back to ask what it missed.
Signing secret	<code>whsec_</code> followed by base64 that decodes to 24 to 64 bytes.	A key of a known shape. Your server refuses anything else.
Signature headers	Standard Webhooks: <code>webhook-id</code> , <code>webhook-timestamp</code> , <code>webhook-signature</code> , plus <code>X-MCP-Subscription-Id</code> . The signature is v1, a base64 HMAC-SHA256 over <code>id.timestamp.body</code> .	A public recipe for signing. Sign the exact bytes you send, or the check fails. <code>webhook-id</code> equals the body's <code>eventId</code> .
Verification	The server sends a signed challenge with its own unique <code>webhook-id</code> . ChatGPT echoes it in a 2xx. Failure is JSON-RPC error <code>-32015</code> .	Proof the callback URL is real before your data goes to it.
Event body	<code>eventId</code> , <code>name</code> , <code>timestamp</code> (ISO 8601 with a timezone), <code>data</code> , <code>cursor</code> .	Your own fields go inside <code>data</code> . Facts only: the docs say not to put instructions to the model in the payload.
Size	One event per request, body no larger than 256 KiB (262,144 bytes).	Room for a text report, not a large file. Send a short notice and let ChatGPT call a tool for the full record.
Callback URL rules	HTTPS, public addresses only, no redirects.	Your server must refuse a callback that points at a private machine.

Expiry	The server returns <code>refreshBefore</code> , and ChatGPT calls <code>events/subscribe</code> again before then.	Subscribing twice must give the same subscription, with the same id.
Time to arrive	No figure. "ChatGPT processes the event asynchronously."	In Jason's one test, about five minutes after the 200. That is an estimate from one run. ChatGPT can also group separately delivered events into one task run.

STEP 1

Decide your event and your task

Do this before you open your agent. A server built around the reel's bug report shows that the feature works, and then you delete it. Build it around an event from your own work and you have something to keep using.

The event is one thing that happens in your work and is worth being woken for. **The task** is what ChatGPT does when it arrives.

If your work has	An event name	A first task
Builds or deploys	<code>build.failed</code>	Summarize the error and draft a fix plan.
Orders or payments	<code>order.created</code>	Draft the confirmation note and flag anything unusual.
A form or a support inbox	<code>ticket.created</code>	Draft a reply for me to review.
Issues or comments	<code>comment.created</code>	Summarize the thread and say what is being asked of me.
Bug reports (the reel)	<code>bug_report.created</code>	Draft a fix plan.

The names in the middle column are suggestions. Your agent will propose one from your answers. The rest of this guide uses `bug_report.created` as the worked example, because it is the one that was run.

Your part: write one sentence: "When ___ happens, ChatGPT should ___." Start with a task that drafts or summarizes. OpenAI's docs say events can arrive out of order and that ChatGPT can group several into one run, so a task that is safe to repeat is the right first choice. Also check the task cannot cause the same event again, or you have built a loop.

STEP 2

Paste one prompt

Make an empty folder, start Claude Code or Codex in it, and paste the prompt on the next page. The same prompt works in both. It opens with three questions, proposes an event name and a payload, and waits for your yes before it writes code.

What the prompt has your agent do

1. Ask what should wake ChatGPT, what ChatGPT should do, and where the event comes from today.
2. Read OpenAI's pages, because the spec is a draft.
3. Check that Python 3 and `cloudflared` are installed.
4. Write one small server: one read-only tool, your event, the three event methods, and the signing code.

5. Write one `emit` function, a fire command for tests, and a local route your own app can call later.
6. Run a self-test that plays ChatGPT's part on your computer.
7. Start the server, ask for the tunnel, and print the ChatGPT steps with your URL in them.
8. After the first event lands, show you the hook that fires it from your real source.

Your part: answer the questions with your real event, approve the file and run commands, and check the self-test passes before you go on. Keep the server out of your agent's own MCP settings. In Jason's test a plain MCP connection used the older 2025-06-18 protocol, fetched tools only, and ChatGPT then said the plugin "exposes only" its tool.

Build me a small MCP server that uses MCP Events in ChatGPT: ChatGPT subscribes to one event from my own work, and my server pushes that event into a ChatGPT Work chat with a signed webhook.

START BY ASKING ME THREE QUESTIONS, then wait for my answers

1. What in my work should wake ChatGPT? Offer these as examples: a failed build or deploy, a new order or payment, a new form or support entry, a new issue or comment, a new bug report (bug_report.created).
 2. What should ChatGPT do when it arrives? For example: draft a fix plan, summarize it, draft a reply for me to review.
 3. Where does that event come from today: an app, a script, a CI job, a form, a folder?
- From my answers, propose an event name in the form thing.happened (for example build.failed or bug_report.created), a payload of three to five short fields, and one read-only tool that lists recent items. Show me the proposal and wait for my yes before you write code.

I have not built or hosted an MCP server before. Before each stage, tell me in one or two plain sentences what you are about to do and why.

READ THESE FIRST. The spec is a draft, so follow the pages over anything you remember:

- <https://developers.openai.com/plugins/build/mcp-events>
- <https://developers.openai.com/plugins/deploy/connect-chatgpt.md>
- <https://developers.openai.com/plugins/quickstart.md>

CHECK

- Run `python --version` (or `py -3 --version`) and `cloudflared --version`. If one is missing, stop and give me the install command for my system.

BUILD

- In this folder, one server file plus one self-test file. Use only the Python standard library and hand-write the JSON-RPC over HTTP POST at `/mcp` (http.server worked for the guide's author). Do not install an MCP SDK.
- Use streamable HTTP with plain JSON responses at `/mcp`. No Server-Sent Events: Cloudflare Quick Tunnels do not support them.
- Protocol version 2026-07-28. Answer `server/discover` with `supportedVersions ["2026-07-28"]` and `capabilities {"tools": {}, "events": {}}`.
- Also answer the older initialize handshake (echo the client's `protocolVersion`), `ping`, `tools/list` and `tools/call`. Reply 202 to notifications (requests with no id). Put the `resultType` field the OpenAI page shows in every result.
- One read-only tool: the list tool we agreed. `emit` appends each event to a gitignored file and the list tool reads that file, so the list is the same from every process.
- One event: the one we agreed, with a name, a description, `delivery ["webhook"]`, an `inputSchema` that takes no arguments, and a `payloadSchema` for the fields we agreed. The payload carries facts. Never put instructions to the model in it.
- `events/list` returns that event.
- `events/subscribe`:
 - accept only `delivery mode "webhook"`
 - require an https callback URL that resolves to a public address
 - require a secret of the form `whsec_ + base64` that decodes to 24 to 64 bytes
 - POST a signed `{"type": "verification", "challenge": "<random>"}` to the callback, with its own unique `webhook-id`, and store the subscription only if a 2xx comes back echoing the same challenge (constant-time compare). On failure return JSON-RPC error -32015.
 - same callback URL + event + arguments gives the same subscription id, so a repeat call updates it
 - return `{id, refreshBefore, cursor: null, truncated: false}` with a 24 hour lifetime
- `events/unsubscribe` removes the subscription and succeeds even if it is already gone.
- Store subscriptions in a file so they survive a restart. That file holds signing secrets: gitignore it and never print a secret. If a delivery comes back 410, delete that subscription.
- Sign with Standard Webhooks. Headers: `Content-Type application/json`, `webhook-id`, `webhook-timestamp` (Unix seconds), `webhook-signature = "v1," + base64(HMAC-SHA256(key, "<webhook-id>.<timestamp>.<body>"))`, `X-MCP-`

Subscription-Id. The key is the base64-decoded part after whsec_. Serialize the body once and send those exact bytes. 10 second timeout, no redirects (urllib follows them, so use http.client), body no larger than 256 KiB.

- One function, emit, that takes the payload fields, builds {eventId, name, timestamp, data, cursor: null} with a fresh eventId (webhook-id equals eventId, timestamp is ISO 8601 with a timezone), and sends it to every live subscription (one whose refreshBefore has not passed). Everything that fires an event calls this one function.
- A fire command for testing, for example: python server.py fire "Checkout button does nothing on mobile Safari". It takes the main text field as its argument, fills the other fields from flags or defaults, calls emit, and prints the subscriber count and the HTTP status for each.
- A second listener on 127.0.0.1:8788, which the tunnel never exposes, with one route: POST /emit with a JSON body calls emit. My own app or script will use it later. /emit accepts only Content-Type application/json and an X-Emit-Token header, compared in constant time with a gitignored token file that the server creates if it is missing. Reject anything else with 403, so a web page open in my browser cannot fire events.
- Log every incoming method with its MCP-Protocol-Version header, and log the client info on server/discover and initialize. I will need that log to debug the ChatGPT side.
- Self-test: a fake receiver that subscribes over HTTP, echoes the challenge and verifies the signature on a delivered event. Compare header names case-insensitively. The self-test starts its own copy of the server on other ports (for /mcp and /emit), with its own temporary data folder and a test-only flag that allows http://127.0.0.1 callbacks, then stops it. The server refuses that flag on port 8787 or with the real data folder.

RUN

- Run the self-test, then start the server on 127.0.0.1:8787. Handle requests on threads (ThreadingHTTPServer, with allow_reuse_address off so a second copy cannot share the port), so one slow callback does not block the next request.
- It needs a public HTTPS URL. Before the tunnel opens, tell me plainly that anyone who learns that URL can call the tool and subscribe, so the payload must hold no secrets, private paths or customer data. The command: cloudflared tunnel --url http://127.0.0.1:8787
- If your permissions or sandbox block the tunnel, stop and give me the command. In Claude Code I will run it with the ! prefix in this session. In Codex I will run it in a second terminal and paste back the URL.
- Once the tunnel is up, call events/list through the public URL and show me the result.
- Do not register this server in your own MCP settings (Claude Code's MCP config, Codex's config.toml). In the author's test a plain MCP connection opened with initialize on an older protocol and fetched tools only. ChatGPT asked for events once the server was added as a plugin.

THEN PRINT THESE STEPS FOR ME, with my real URL, event name and task filled in

1. chatgpt.com/plugins, plus button, Add custom MCP server. A name and a description, Server URL set to the public URL ending in /mcp, Authentication set to No authentication (this test server has no login). Read the warning, select I understand and want to continue, then Create as a plugin.
2. On the plugin page, check my event is listed beside the tool. If it is missing, select Refresh.
3. Open the plugin and select the plus button to install it.
4. On the ChatGPT home page switch Chat to Work (desktop app: Work with Cloud selected). New chat, type @, pick the plugin, and send: Subscribe to the <event name> event. When one arrives, <my task>.
5. Tell you when ChatGPT says it is monitoring.

WHEN I SAY I HAVE SUBSCRIBED

- Confirm the log shows events/subscribe on 2026-07-28 and one stored subscription before firing anything. If there is none, say so and do not fire.
- Fire one event and report the status. Use different text each time. ChatGPT flagged identical text as a duplicate in the author's test.
- A 2xx means ChatGPT received it. ChatGPT processes the event asynchronously, and in the author's test the chat message took about five minutes. Do not fire again because the chat is quiet.
- If the tunnel restarts, the URL changes: tell me to update the plugin's Server URL and select Refresh.

THEN SHOW ME HOW MY REAL SOURCE WOULD FIRE IT

- From my answer to question 3, write the smallest hook that calls emit when the real thing happens: a few lines in my local script that run the fire command by absolute path and log if the hook itself fails, or a POST to the local /emit route from an app on this computer. If my source runs on another machine (hosted CI,

a SaaS form, a store), say so, and do not expose /emit. Offer a small local watcher that polls that source and calls emit.

- Show me the hook. Do not install it or touch my real systems until I say yes. When I say yes, install it, then ask me to make the real thing happen once and watch the log with me.
- Check that the task ChatGPT runs cannot cause the same event again. If it could, tell me how the loop would happen.

WHEN I SAY I AM DONE

- Stop the server, remind me to stop the tunnel, and remind me to tell ChatGPT to stop monitoring.

Skip OAuth, retries and a database for this first run. Mark each shortcut with a one-line comment that says what a server I keep would do.

A version of this prompt fixed to the bug report event was run in Claude Code on 6 October 2026. The three opening questions, the local `/emit` route and the real-source section were added for this guide and have not been run. Codex has not been tried at all.

STEP 3

Get it on a public URL

Your agent starts the server on `127.0.0.1:8787`, an address only your computer can reach. The tunnel gives it a public one. A Cloudflare Quick Tunnel needs no account or domain, is meant for testing and development, and allows up to 200 requests in flight at once. It does not support Server-Sent Events, which is why the prompt asks for plain streamable HTTP at `/mcp`.

If your agent cannot open the tunnel, run this yourself

```
cloudflared tunnel --url http://127.0.0.1:8787
```

Your part: in Claude Code, type `!` and then the command, which runs it in your session. Claude Code's auto mode refused to open the tunnel in Jason's test, and this is how he ran it. In Codex, run it in a second terminal and paste the URL back. Copy the `https` address that ends in `trycloudflare.com` and add `/mcp` to the end. The address changes every time the tunnel restarts. Only cloudflared was tested.

STEP 4

Add it to ChatGPT as a plugin

1. Go to chatgpt.com/plugins and select the plus button.
2. Select **Add custom MCP server**.
3. Type a name and a description.
4. In **Server URL**, paste your public address. It must include the `/mcp` path.
5. Under **Authentication**, select **No authentication**.
6. Read the risk warning, then select **I understand and want to continue**.
7. Select **Create as a plugin**.
8. Open the plugin and select the plus button to install it.

Your part: check that the plugin page lists your event beside the tool. If you see the tool alone, select **Refresh**. OpenAI's docs say to do the same whenever you change the server's tools or events.

About "No authentication"

The test server has no login. ChatGPT's form offered No authentication, accepted it, and events arrived, as observed on 6 October 2026. OpenAI's docs describe the event methods "on the same authenticated MCP endpoint as your tools", and the draft says subscribe and unsubscribe must have an authenticated principal. So read this as a setting for a short test. While the tunnel is open, anyone who has the address can reach the server. A server you keep needs a login.

STEP 5

Subscribe in a ChatGPT Work chat

1. Go to the ChatGPT home page and switch **Chat** to **Work**.
2. Start a new chat.
3. Type `@` and pick your plugin.
4. Send one sentence that names the event and gives the task.

Send in the Work chat, with your own event and task

```
Subscribe to the <your event name> event. When one arrives, <your task>.
```

The reel's sentence: Subscribe to the bug_report.created event. When one arrives, draft a fix plan.

In the reel, ChatGPT checks the plugin's events and sets up a monitor. That monitor is the new piece: it is the thing that keeps watching after you stop typing.

Your part: wait until ChatGPT says it is monitoring, then tell your agent. Ask it to confirm that the server log shows `events/subscribe` on `2026-07-28` and one stored subscription.

STEP 6

Fire the event and wait

Ask your agent to fire a test event, or run the command it printed for you. For the worked example it looks like this.

Run in a terminal, in the project folder

```
python server.py fire "Checkout button does nothing on mobile Safari"
```

The output should show 1 subscriber and a `200`. Then watch the Work chat. The event arrives and ChatGPT does your task, with nothing typed by you.

Your part: be patient. A 200 means ChatGPT received the event. In Jason's test the message reached the chat about five minutes later, so a quiet chat is no reason to fire again. Change the text on each test, because ChatGPT flagged identical text as a duplicate.

When you are done testing

1. Tell ChatGPT to stop monitoring the event.
2. Stop the tunnel.
3. Stop the server.

The subscriptions file holds signing secrets. Delete it before you share the folder.

FIXES

When it does not arrive

Every row but the last two happened during Jason's build. Start with the server log: it shows each method ChatGPT called and the protocol version it used.

What you see	What it means	The fix
ChatGPT says the plugin "exposes only" its tool	The server was added as a plain MCP connection. The log shows protocol <code>2025-06-18</code> and <code>tools/list</code> , with no <code>server/discover</code> .	Add it at chatgpt.com/plugins with Add custom MCP server and Create as a plugin.
ChatGPT does not subscribe	The chat was not a Work chat, or the sentence did not name the event.	Switch Chat to Work, start a new chat, pick the plugin with <code>@</code> , and put the event name in the sentence.
The fire command prints 0 subscribers	There is no stored subscription. ChatGPT never called <code>events/subscribe</code> , or the 24 hour lifetime in this test server ran out.	Ask your agent to look for <code>events/subscribe</code> in the log. Subscribe again, and fire only once one subscription is stored.
200 in the terminal, nothing in the chat	ChatGPT has the event and has not processed it yet.	Wait. About five minutes in Jason's test. Firing again sends more events, and ChatGPT can group separately delivered events into one task run.
ChatGPT calls the event a duplicate	You sent the same text twice.	Use different text on each test.
Your agent refuses to open the tunnel	Its permissions or sandbox block it. Claude Code's auto mode did this.	Run the command yourself: with <code>!</code> in Claude Code, or in a second terminal for Codex.
It worked, then ChatGPT cannot reach the server	The tunnel restarted, and a Quick Tunnel gets a new hostname each time.	Put the new address, with <code>/mcp</code> , in the plugin's Server URL and select Refresh.
Subscribing fails with <code>-32015</code>	Verification failed: the challenge your server sent did not come back echoed in a 2xx.	Have your agent read the log around the challenge request and confirm the server can make outbound HTTPS requests. (Not seen in the test.)
No "Add custom MCP server" option	OpenAI says account and workspace policies apply to adding custom MCP servers.	Check with your workspace admin, or try an account you control. (Not seen in the test.)

THE OUTCOME

From test to something you keep

You have an event that reaches ChatGPT when you run a command. The goal is an event that reaches ChatGPT when your work changes. These are the changes between the two, in the order to make them. Ask your agent for each one by name. None of them was built in the test, so treat what follows as a plan drawn from OpenAI's and Cloudflare's docs.

1. Add a login

OpenAI describes the event methods "on the same authenticated MCP endpoint as your tools", and the draft requires an authenticated principal for subscribe and unsubscribe. The test ran without one. Before anything real flows through the server, have your agent add the authentication ChatGPT's form supports and check each subscription against who is asking.

2. Connect the real event source

The test server has one `emit` function. The fire command calls it, and so can anything else. A script or CI job adds one line that runs the fire command when the build fails. An app posts a small JSON body to the local `/emit` route when the order or ticket is saved. Your agent wrote this hook for you at the end of the build. If the source runs on another machine, such as hosted CI or a SaaS form, it cannot reach your computer: use the local watcher your agent offered.

Keep the payload small and factual. One event may be no larger than 256 KiB, so send a short notice with an id and let ChatGPT call your read-only tool for the full record. Never put instructions to the model in the payload.

3. Move to a stable URL

A Quick Tunnel's hostname changes every time it starts, and Cloudflare offers it for testing and development. A server you keep needs an address that stays put: a small hosted server, or a tunnel with a fixed hostname on a domain you own. Callback rules still apply in the other direction, so the server needs outbound HTTPS.

4. Store subscriptions in a database

OpenAI's docs call for persistent subscription storage. A JSON file was enough for one test. A database survives restarts, handles two writes at once, and gives you one place to protect the signing secrets. Keep two behaviors from the test: subscribing twice returns the same subscription id, and the server returns `refreshBefore` so ChatGPT can renew before a subscription expires.

5. Retry, and know when to stop

The test server sends once. The docs say to retry transient failures with exponential backoff, which means waiting longer between each attempt, and not to retry after a `410` or a `413`. A 410 says the subscription is gone. A 413 says the body is too large, and sending it again will not shrink it.

6. Plan for order, repeats and loops

Events can arrive out of order, and an event type without replay cannot recover events that were missed. If the task calls a tool that writes, make that tool idempotent, meaning a second identical call changes nothing. Then check the task ChatGPT runs cannot cause the event that woke it. A fix plan that files a bug report, on a server that fires on new bug reports, is a loop.

7. Reread the docs when something breaks

This is a draft specification, and OpenAI links the MCP Events design sketch as its reference. Method names and fields can change. When a working server stops working, compare it with the current page before you debug your own code.

Glossary

Term	What it means here
Agent	Claude Code or Codex. It writes the files and runs the commands.
Callback URL	The web address ChatGPT gives your server when it subscribes. Your server posts each event to it.
Capability	A feature an MCP server says it has, such as <code>tools</code> or <code>events</code> .
Challenge	A random value your server sends to the callback URL to prove the address is real. ChatGPT sends it back.
dots	A ChatGPT feature OpenAI names as a place events are available. This guide does not use it.
Draft specification	A set of rules that is still being written and can change.
Event	One thing that happened, reported by the MCP server. Example: a new bug report.
Exponential backoff	Retrying with a longer wait after each failure.
HTTPS	The encrypted form of a web address. It starts with <code>https://</code> .
Idempotent	Safe to run twice. The second run changes nothing.
KiB	1,024 bytes. 256 KiB is 262,144 bytes.
Log	The lines the server prints as it runs. It shows every request ChatGPT made.
MCP	Model Context Protocol, the standard that connects an AI application to outside tools and data.
MCP server	A program that offers tools, and now events, to an AI application.
Method	A named request one program sends another, such as <code>events/list</code> .
Payload	The fields your event carries inside <code>data</code> .
Plugin	ChatGPT's name for an MCP server added to your account. The reel says connector.
Protocol version	A dated set of MCP rules. MCP Events need <code>2026-07-28</code> .
Public URL	A web address any computer on the internet can reach. <code>127.0.0.1</code> points at your own computer and is private.

Signature	A code computed from the event and the signing secret. The receiver uses it to check the sender.
Signing secret	The key ChatGPT gives your server for signatures. Keep it private.
Status 200, 2xx	The answer that says a web request was accepted. Any code from 200 to 299 counts.
Subscription	The stored record of which event to send, where to send it, with which secret, until when.
Tool	An action an AI application can ask an MCP server to perform.
Tunnel	A program that gives your computer a temporary public HTTPS URL.
Webhook	A message one program sends to a web address when something happens.
Work chat	A ChatGPT chat started with the home page switch on Work.

Sources

- OpenAI, MCP Events: developers.openai.com/plugins/build/mcp-events
- OpenAI, connect and test your plugin: developers.openai.com/plugins/deploy/connect-chatgpt.md
- OpenAI, plugins quickstart: developers.openai.com/plugins/quickstart.md
- OpenAI, ChatGPT release notes of September 29, 2026, and the DevDay 2026 launch list
- MCP Events design sketch (the draft): github.com/modelcontextprotocol/experimental-ext-triggers-events
- Cloudflare, Quick Tunnels: developers.cloudflare.com/tunnel/get-started/quick-tunnels/
- Standard Webhooks: standardwebhooks.com
- Jason's own build and test, 6 October 2026: the pitfalls, the 24 hour lifetime, and the five minute estimate

KEEP GOING

The guide page

The prompt, ready to copy, and the six steps live on the page: graniteai.co/tools/two-way-mcp

See what else GraniteAI makes: graniteai.co/tools/two-way-mcp