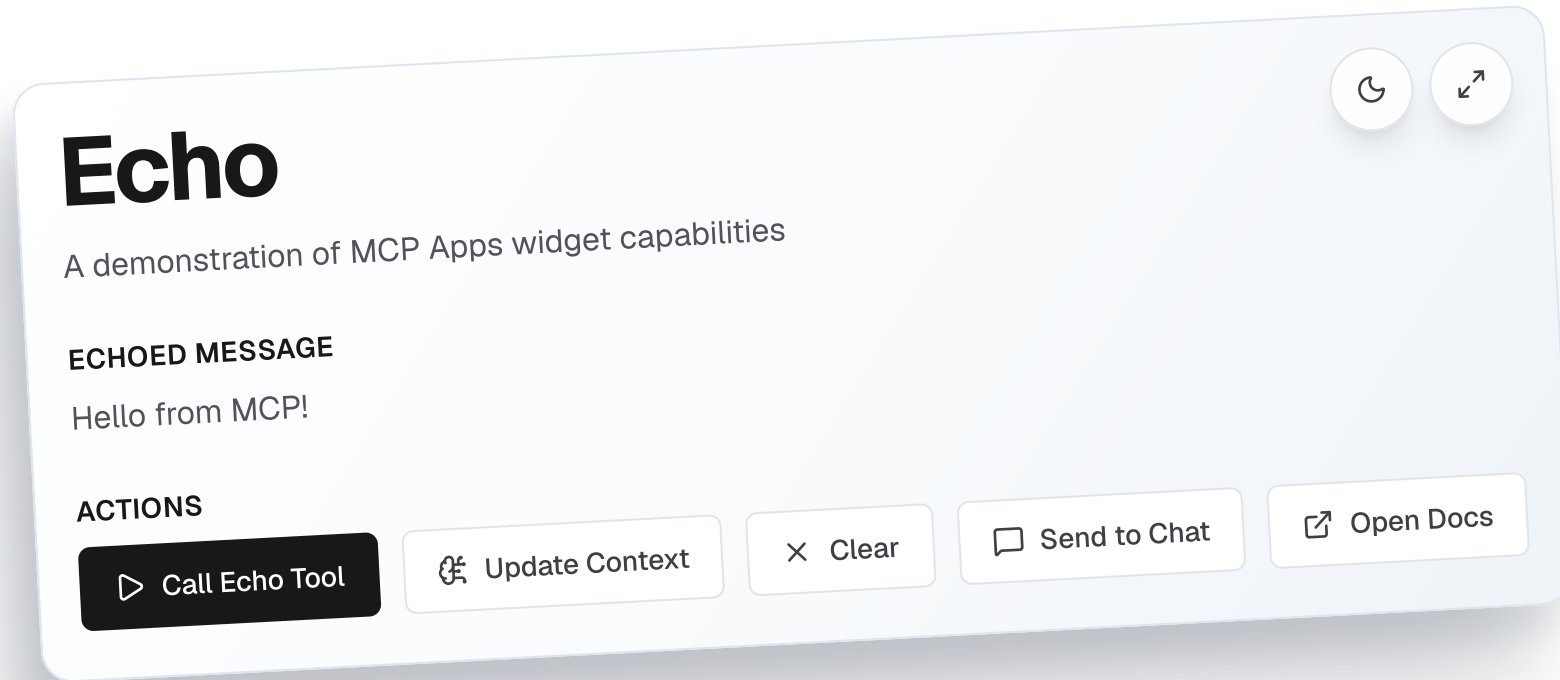


Building Your First MCP App



What is MCP?

Model Context Protocol is a standard way for an AI host to discover and use capabilities exposed by a server: **tools, resources, and prompts.**

Two standard MCP transports

stdio — the host launches a local command and communicates through stdin/stdout.

Streamable HTTP — the host connects to an independent HTTP server, whether it is on localhost or the internet.

For this talk, we'll focus on remote MCP servers over Streamable HTTP.

A tool is a function

```
server.registerTool(  
  "echo",  
  {  
    title: "Echo",  
    description: "Echo back the provided message",  
    inputSchema: z.object({  
      message: z.string().describe("The message to echo back"),  
    }),  
    outputSchema: z.object({  
      echo: z.string().describe("The echoed message"),  
    }),  
  },  
  (args, ctx) => echo(args, ctx),  
);
```

A tool is a function

```
async function echo(args: { message: string }) {  
    return createTextResult({ echo: args.message });  
}
```

A tool is a function

```
export function createTextResult(data: unknown): CallToolResult {
  const safeData = data === undefined ? null : data;

  const result: CallToolResult = {
    content: [
      { type: "text", text: JSON.stringify(safeData, null, 2) },
    ],
  };

  if (safeData !== null && typeof safeData === "object") {
    result.structuredContent = safeData as Record<string, unknown>;
  }

  return result;
}
```

What the call looks like

```
// Request
{
  "method": "tools/call",
  "params": {
    "name": "echo",
    "arguments": { "message": "Hello World!" }
  }
}
```

What the call looks like

```
// Response (result)
{
  "content": [
    { "type": "text", "text": "{\n  \"echo\": \"Hello World!\"\n}" }
  ],
  "structuredContent": { "echo": "Hello World!" }
}
```

That's MCP

A remote MCP server is:

1. One HTTP endpoint — `localhost` in dev, HTTPS once a host has to reach it
2. A list of tools
3. Structured results the model can keep using

Same server from here on. We're about to give it a face.

That same server can ship a UI

The host still calls a tool. Instead of a blob of JSON, you get something you can actually use in the chat.

Excalidraw demo

Excalidraw create_view

Ship Early, Iterate Rapidly

Don't wait for perfect — launch the raft, build the ship at sea

Three ships sailing toward the sun — each bigger and better than the last. Don't wait for perfect, launch the raft and build the ship at sea.

show sequence diagram explaining

Opus 4.6

mcp.excalidraw.com

So what's an MCP app?

An **MCP app** is an MCP server where a tool's result comes with a UI.

- The tool still returns ordinary structured data
- It also points at a widget the host renders in the chat
- Text for the model. A live UI for you. Same tool call.

This is the bit that actually changes

Bind the echo tool to a UI resource:

```
registerAppTool(server, "echo", {  
  title: "Echo",  
  description: "Echoes back the user's message in an interactive view",  
  inputSchema: EchoToolInputSchema.shape,  
  _meta: {  
    ui: { resourceUri: "ui://echo" },  
  },  
}, handler);
```

That MIME type is a literal

The resource is HTML. MIME type is `text/html;profile=mcp-app`. Hosts look for it.

Without `_meta.ui`, the echo tool is text. With it, the host fetches `ui://echo` and renders that HTML.

The widget is just a web page

- Sandboxed iframe, in the host's chat UI
- Talks to the host over `postMessage` via the MCP Apps `App` API
- React, Vue, Svelte, plain HTML — the host doesn't care

Hosts nest it so your HTML can't touch the chat. That's why you declare a CSP.

This stuff has been around forever

HTML, CSS, React — whatever you already ship on the web.

`<iframe>` showed up in IE 3 in **1996**. `postMessage` has been in browsers since **2008**. That's ~30 years and almost 20 years. Not a new runtime.

MCP Apps didn't invent a UI stack. This tech just found a new place to be useful: a widget in the chat.

The new bit is the host contract on top: `ui://`, the MIME type, and CSP.

And it can call the server again

```
const result = await app.callServerTool({  
  name: "echo",  
  arguments: { message: "Hello from the widget!" },  
});
```

This isn't a screenshot of the first response. Once it's on screen, it can keep calling the same remote server.

Remote images fail until you allow the origin

Widgets run under a strict Content Security Policy. An `<img>` can 200 and still not render. You never listed that origin.

```
_meta: {  
  ui: {  
    csp: {  
      resourceDomains: ["https://images.ctfassets.net"],  
      // connectDomains: ["https://api.example.com"], // fetch / XHR / WS  
    },  
  },  
}
```

List every origin. No wildcards.

- `resourceDomains` — images, fonts, scripts
- `connectDomains` — `fetch` / XHR / WebSocket
- Redirects count as a second origin

No public API. That's fine.

commityourcode.com doesn't expose a speaker or schedule API.

So the Commit Your Code app reads JSON we committed. `catalog.ts`.

Don't scrape the site at request time.

```
data/speakers.json  
data/sessions.json  
data/schedule.json  
data/events.json  
data/tracks.json
```

Same remote MCP shape as the echo tool. Tools for speakers, schedule, and events.

ChatGPT and Claude won't talk to localhost

They need a public HTTPS URL.

Pomerium is tunneling that and handling **OAuth 2.1**. Open-core MCP gateway, among other things. The app has no auth code.

Full disclosure: I work at Pomerium. I'd still use it for this.

MCPJam can hit localhost, so that's where we'll look first.

If you remember a few things

- A remote MCP tool is a function with a schema and a handler
- An MCP app is that same server with a widget bound via

```
_meta.ui.resourceUri
```

- The widget is a web page in a sandboxed iframe, talking over

```
postMessage
```

And a few more

- It can call tools on its own, so it's not a screenshot of the first result
- External images are blocked until you list the origin in
`_meta.ui.csp.resourceDomains`
- Build in MCPJam against localhost. Tunnel when you want Claude or ChatGPT on the same app.

Resources

- [Unofficial CYC 2026 MCP App Repository](#)
- [MCP Apps API reference](#)
- [MCP Inspector](#)
- [MCPJam](#)
- [MCP App TypeScript template](#)
- [MCP TypeScript server template](#)
- [Securing MCP Servers and Apps with Pomerium \(free course\)](#)

Thank You!



nickyt.co

my site



cyc.maisonlab.dev

cyc mcp app