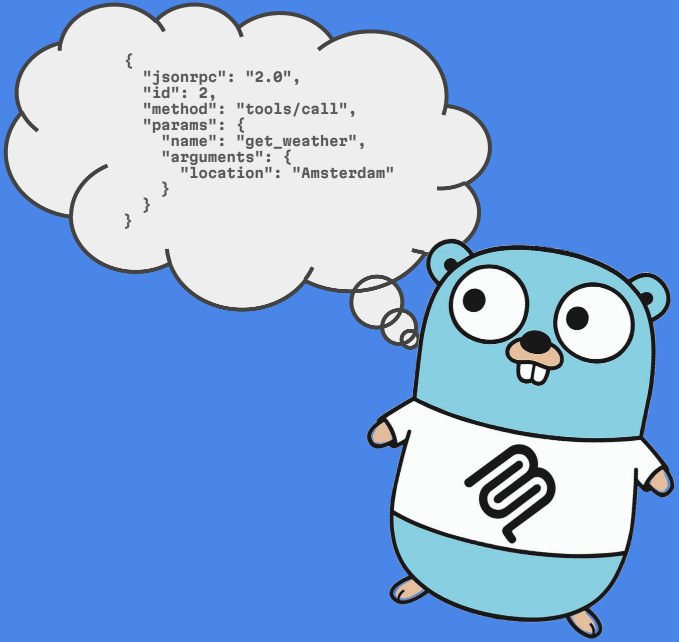
# Building MCP servers with Go

David Stotijn

—

Go Amsterdam Meetup
June 11, 2025



```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/call",
  "params": {
    "name": "get_weather",
    "arguments": {
      "location": "Amsterdam"
    }
  }
}
```

## Agenda

- Introduction to Model Context Protocol (MCP)
- MCP and Go 🤝
- Current state in the Go ecosystem
- Let’s look at some server code…
- “Streamable HTTP”
- Q&A

## Introduction to MCP

**Model Context Protocol (MCP)… All aboard the hype train!**

- What is it?
  - An open standard for connecting AI systems with data sources and tools
  - Features: *tools*, *resources*, *prompts* (and more…)
  - Relatively new (first spec date Nov '24), still evolving
- Use cases
  - IDEs (Cursor, Cline, etc.)
  - Claude Desktop, LibreChat, etc.
  - Agentic systems, where there is no linear/predefined workflow
- Architecture
  - Inspired by LSP
  - JSON-RPC 2.0 messaging (bi-directional!)
  - JSON Schema for RPC argument definitions
  - Spec defines stdio and "Streamable HTTP" as standard transports, but the protocol is transport agnostic
  - Components: *Hosts*, *clients*, *servers*

# MCP and Go 🤝

## Is Go a good fit for MCP?

Yes!

- The base protocol, standard transports and the overall spec aligns well with Go's language features and the standard library:
  - JSON-RPC 2.0 → `encoding/json`, `sync`
  - Stdio transport → `os/exec`
  - Streamable HTTP transport → `net/http`, `sync`
  - Multiplexing, event signaling, cancellation → Go's concurrency primitives
  - Structured data → Type assertion, reflection, JSON Schema libraries, code gen

## Current state in the Go ecosystem

- github.com/mark3labs/mcp-go
  - Currently the most popular 3rd party library (+5k stars on GitHub)
  - Non-idiomatic design and too large API surface (IMHO)
  - No generics, lots of `any`
- golang.org/x/tools/**internal**/mcp
  - Currently under development by the Go team as an “official” SDK
    Design draft: https://github.com/orgs/modelcontextprotocol/discussions/364
  - Feels familiar compared to typical standard library packages
  - `Transport` and `Stream` interfaces.
- Misc others...

## Let’s look at some code...

- What does “golang.org/x/tools/~~internal~~/mcp” look like in practice?
- Testing RPC handlers

## Streamable HTTP

- Added in the 2025-03-26 version of the spec (ref)
- Supports bi-directional RPC messaging and long lived sessions
    - Clients send RPC messages via `POST` requests, they ***MUST*** accept both `text/event-stream` and `application/json` as supported content types
    - When the server responds using SSE, it ***SHOULD*** close the HTTP request after any response message (to an incoming request) have been returned.
    - Clients can open a long-lived SSE stream via a `GET` request, where servers can send *notifications* on
    - Sessions are *logical*, there may be multiple HTTP requests in flight for a single session
- Servers may offer *resumability* and *redelivery* of RPC message streams
    - Resumability can help overcome (idle) timeout challenges introduced by HTTP proxies/load balancers/ingress controllers, unstable network conditions or termination of containers (e.g. workload scaling)
    - Shared and persistent statekeeping of sessions and their underlying streams can be done in custom transports that wrap Streamable HTTP and use an (external) “session store”
- Sessions are cancelled via a `notifications/cancelled` notification
- Libraries typically expose a `http.Handler`, allowing middleware for instrumentation, logging, etc.

## Streamable HTTP (cont.)

**That sounds complicated… Is SSE mandatory?**

- Technically, no. But clients would not be able to receive standalone server initiated requests — that might be fine for some use cases.
- Stream resumability and redelivery are *optional* in the spec. Omitting stateful sessions and streams on the server greatly simplifies things.
- Depending on the server's purpose/business logic, ephemeral sessions might be fine.

**What about auth-z?**

- The spec (version 2025-03-26) introduced authorization via usage of (a subset of) OAuth 2.1 and OAuth 2.0 features (ref)

## Q&A