Hello, MCP World!

Daniela Petruzalek

Developer Relations Engineer @ Google Cloud

@danicat83

Who am I?

Developer Relations Engineer at Google, originally from Brazil 🇧🇷, but living in the UK 🇬🇧 since 2019.

My background is in backend and data engineering. Have been doing Go development (on-and-off) since 2017.

Agenda

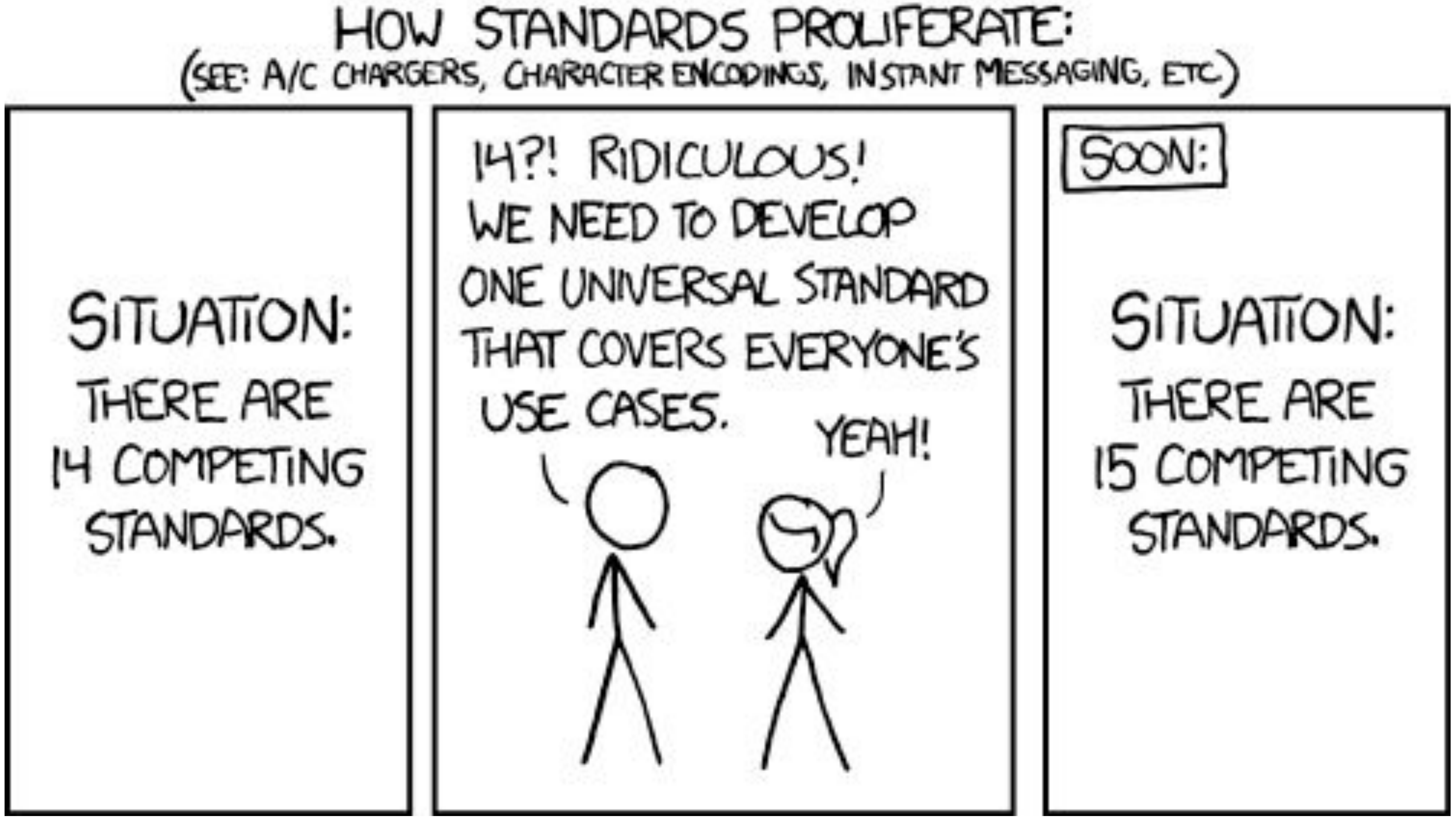
What is the Model Context Protocol

Architecture and Terminology

MCP servers for coding

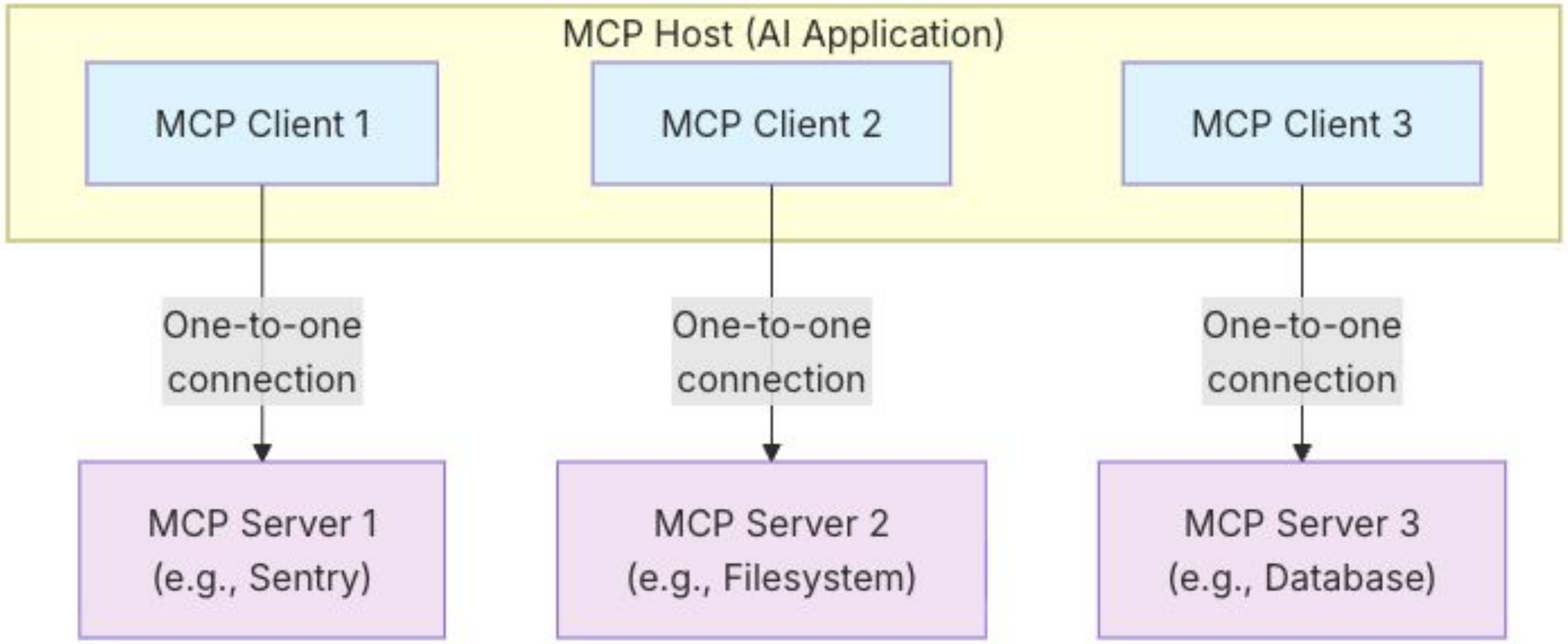
Go support for MCPs

Write your own!


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

MCP is an open protocol that standardizes how applications provide context to large language models (LLMs). (...) MCP enables you to build agents and complex workflows on top of LLMs and connects your models with the world.

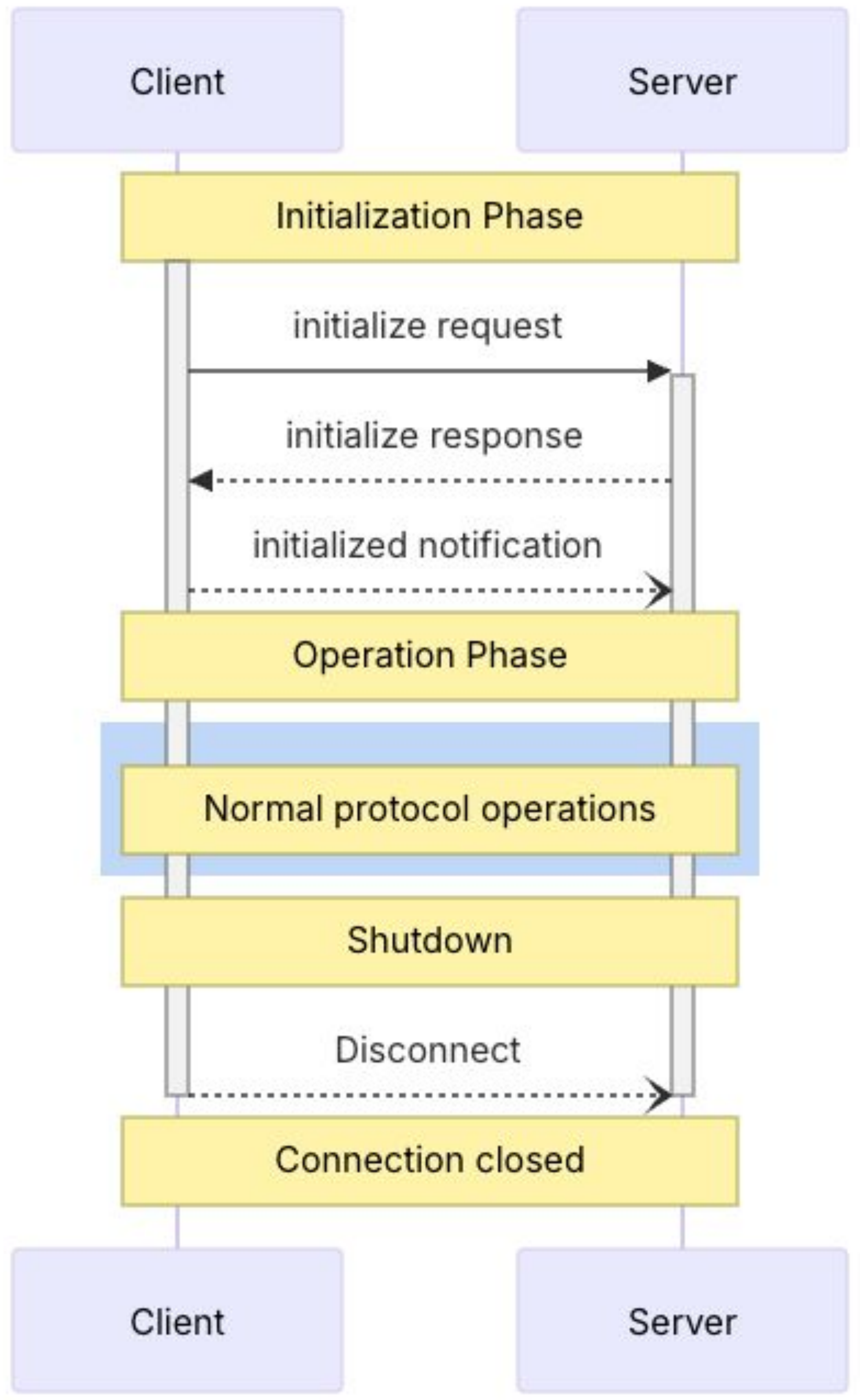
MCP Architecture


MCP Host (AI Application)
MCP Client 1
MCP Client 2
MCP Client 3
One-to-one connection
One-to-one connection
One-to-one connection
MCP Server 1 (e.g., Sentry)
MCP Server 2 (e.g., Filesystem)
MCP Server 3 (e.g., Database)

https://modelcontextprotocol.io/docs/learn/architecture

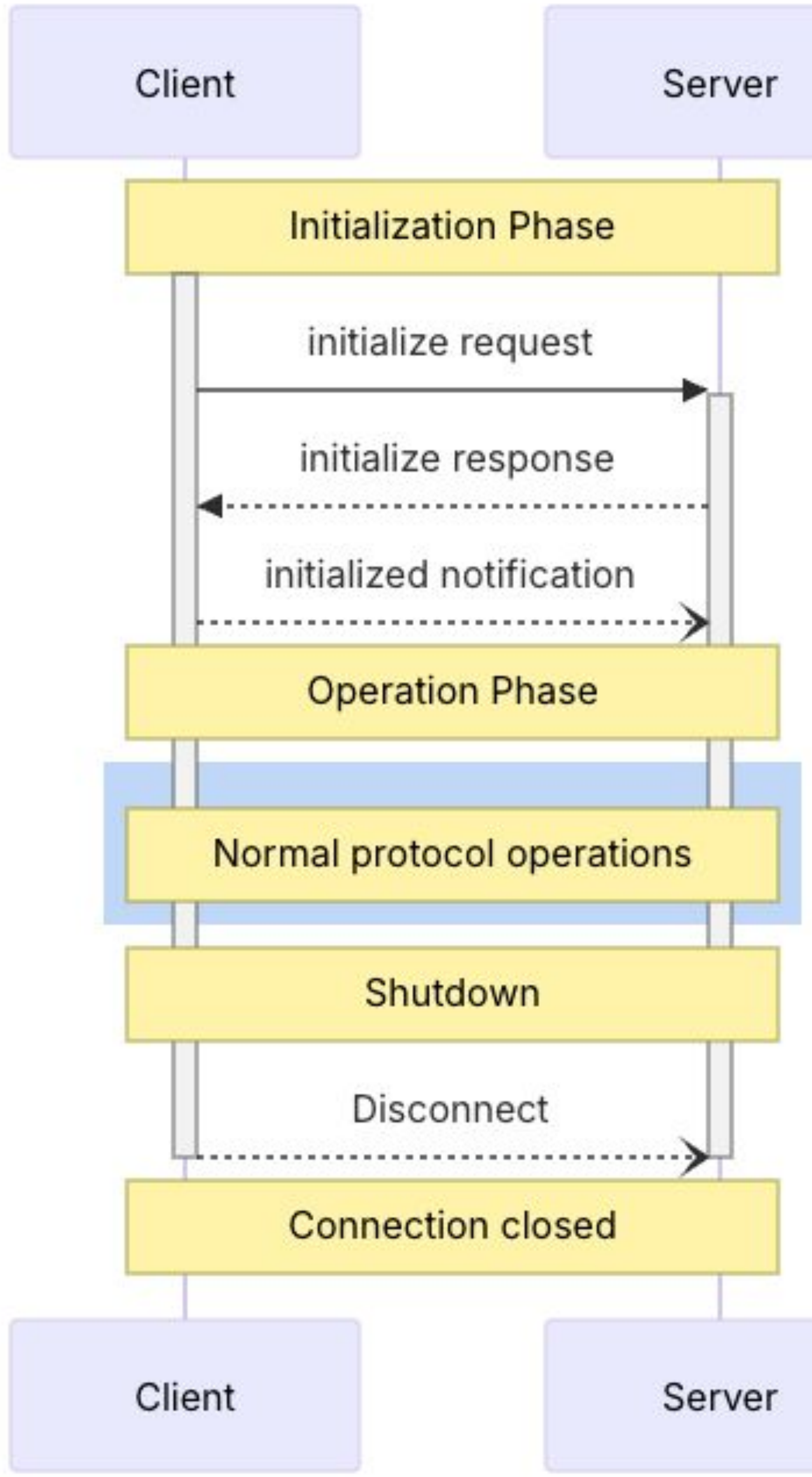
MCP Layers

- **Data layer:** JSON-RPC based protocol for client-server comms
- **Transport layer:** communication channels between client and servers
  - Standard IO
  - Streamable HTTPS
  - HTTP+SSE: deprecated due to security concerns



Initialization Flow

```
{"jsonrpc":"2.0","id":1,"method":"initialize",
"params":{"protocolVersion":"2025-06-18"}}

{"jsonrpc":"2.0","method":"notifications/initialized"
,"params":{}}

{"jsonrpc":"2.0","id":2,"method":"tools/list",
"params":{}}
```

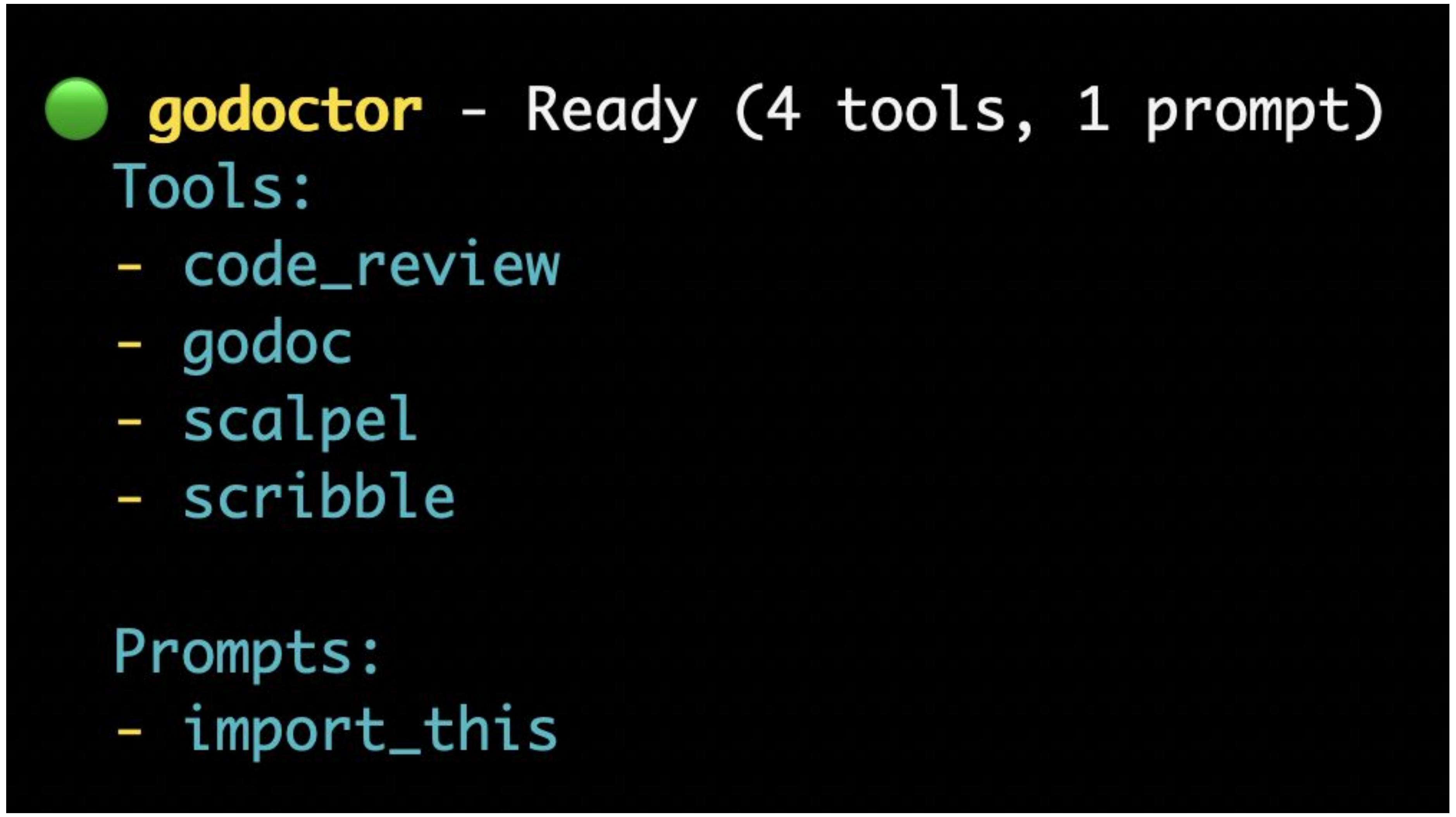


Building Blocks

Building Block	Purpose	Who Controls It	Real-World Example
Tools	For AI actions	Model-controlled	Search flights, send messages, create calendar events
Resources	For context data	Application-controlled	Documents, calendars, emails, weather data
Prompts	For interaction templates	User-controlled	"Plan a vacation", "Summarize my meetings", "Draft an email"

Tools

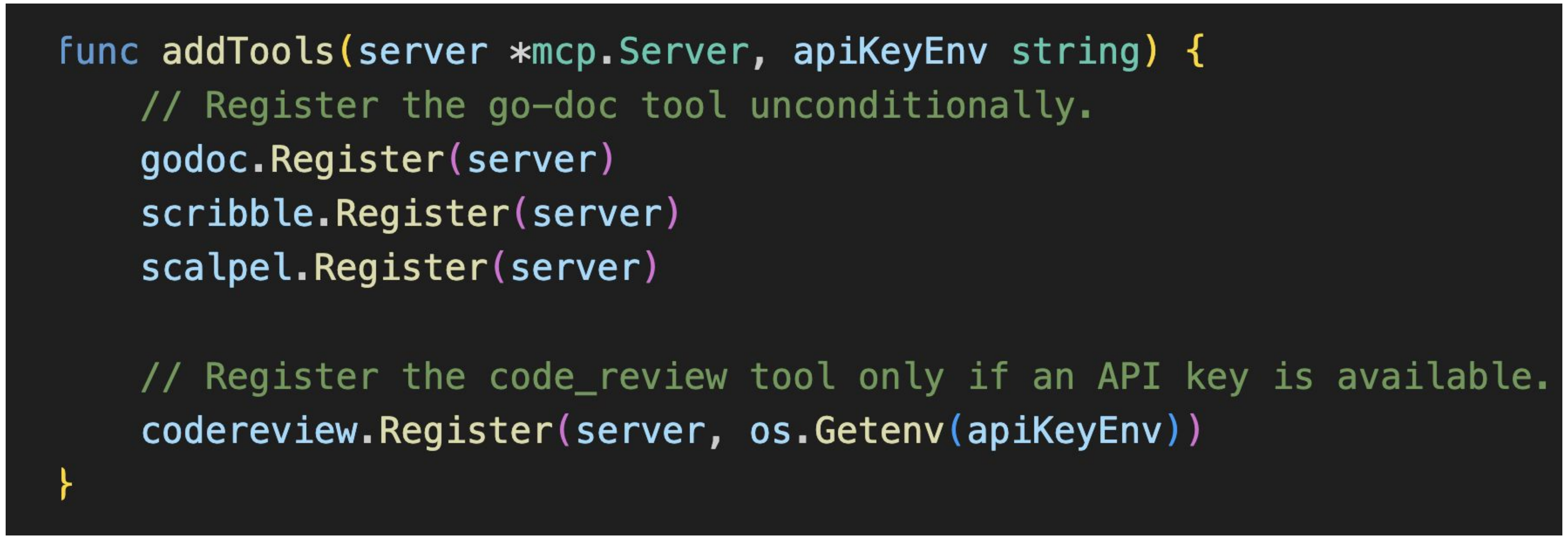
Tools enable AI models to perform actions through server-implemented functions



```
● godoctor - Ready (4 tools, 1 prompt)
  Tools:
  - code_review
  - godoc
  - scalpel
  - scribble

  Prompts:
  - import_this
```

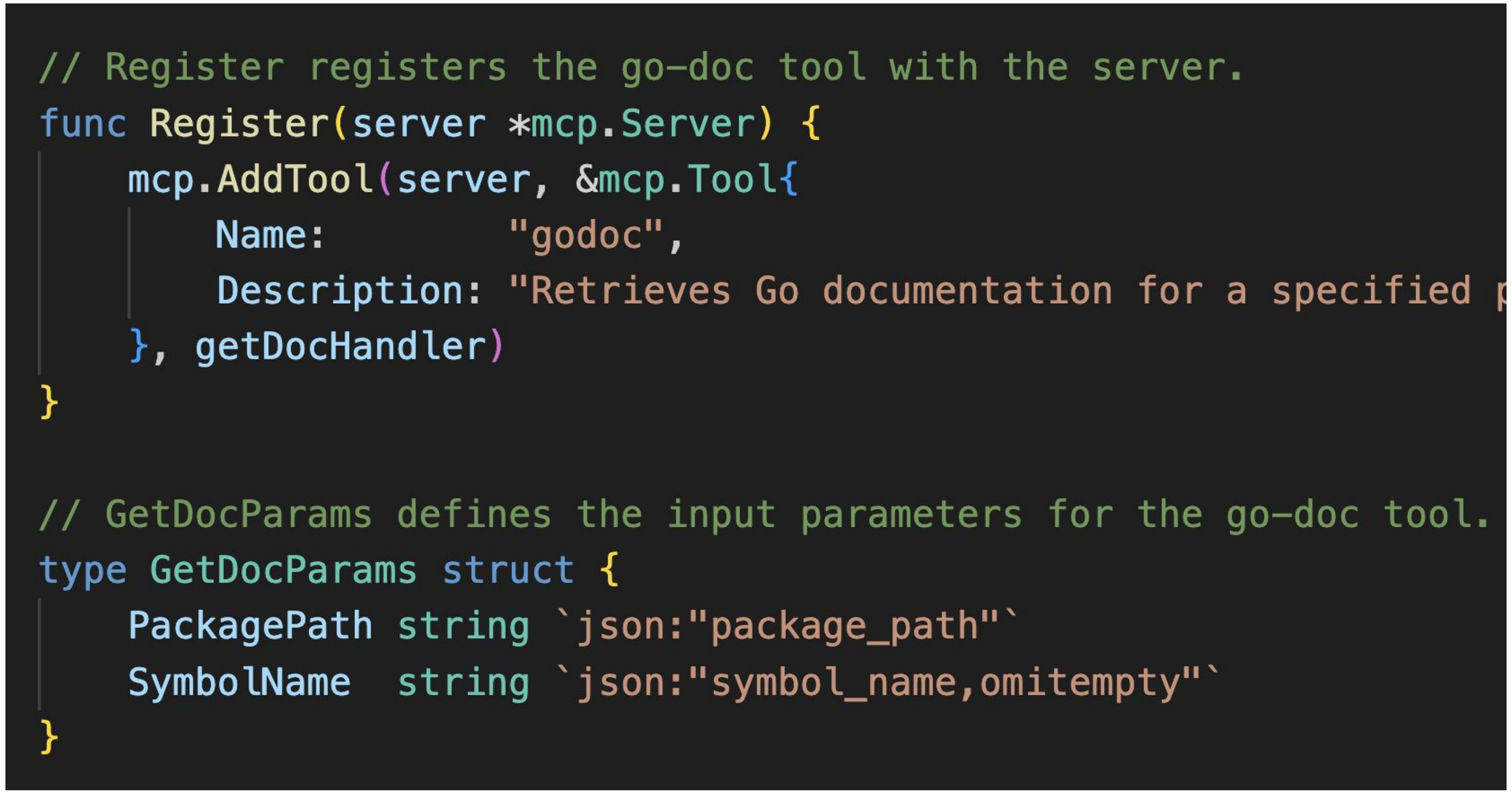
https://github.com/danicat/godoctor

main.go



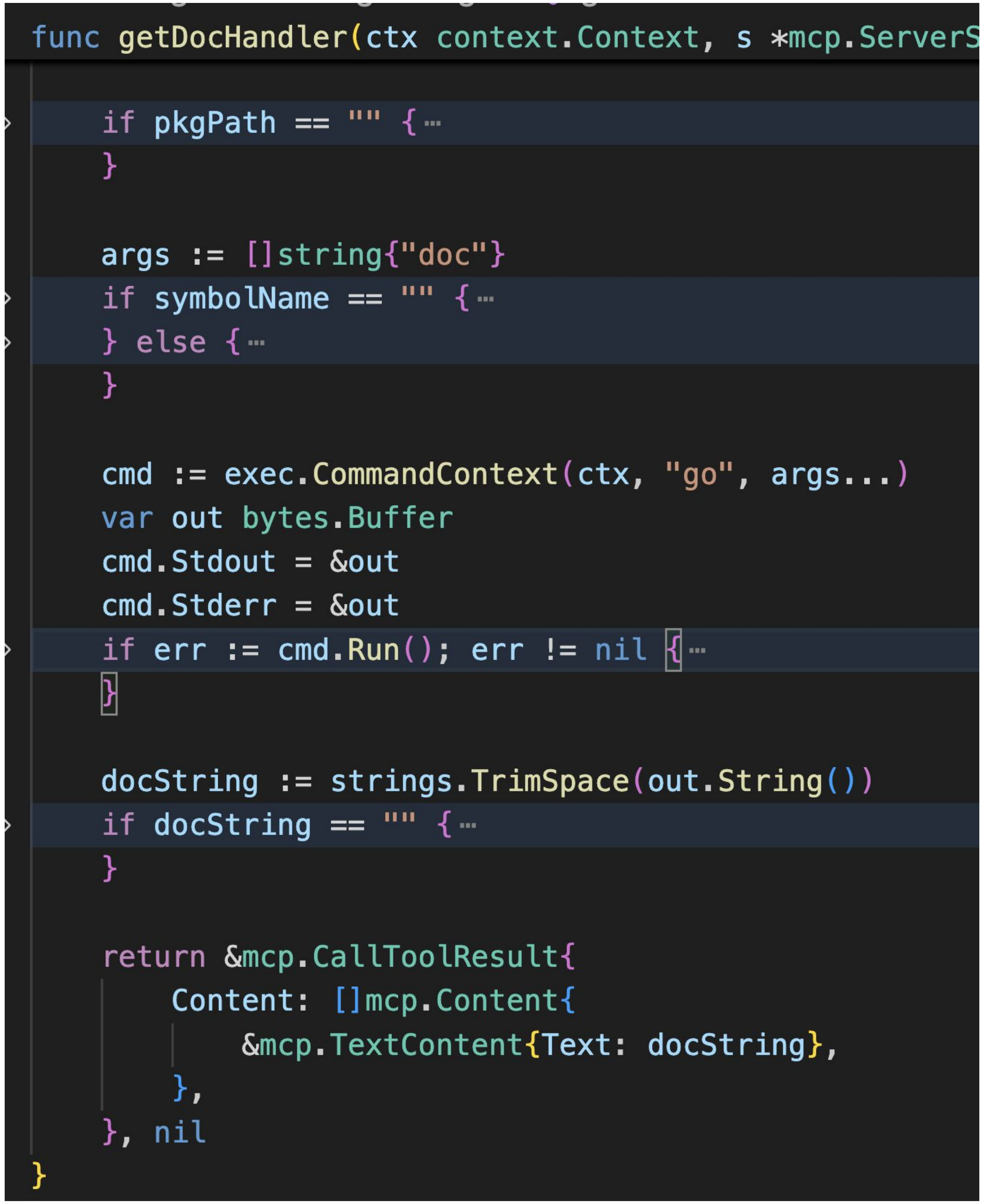
```go
func addTools(server *mcp.Server, apiKeyEnv string) {
    // Register the go-doc tool unconditionally.
    godoc.Register(server)
    scribble.Register(server)
    scalpel.Register(server)

    // Register the code_review tool only if an API key is available.
    codereview.Register(server, os.Getenv(apiKeyEnv))
}
```

godoc.go



```go
// Register registers the go-doc tool with the server.
func Register(server *mcp.Server) {
    mcp.AddTool(server, &mcp.Tool{
        Name:        "godoc",
        Description: "Retrieves Go documentation for a specified p
    }, getDocHandler)
}

// GetDocParams defines the input parameters for the go-doc tool.
type GetDocParams struct {
    PackagePath string `json:"package_path"`
    SymbolName  string `json:"symbol_name,omitempty"`
}
```

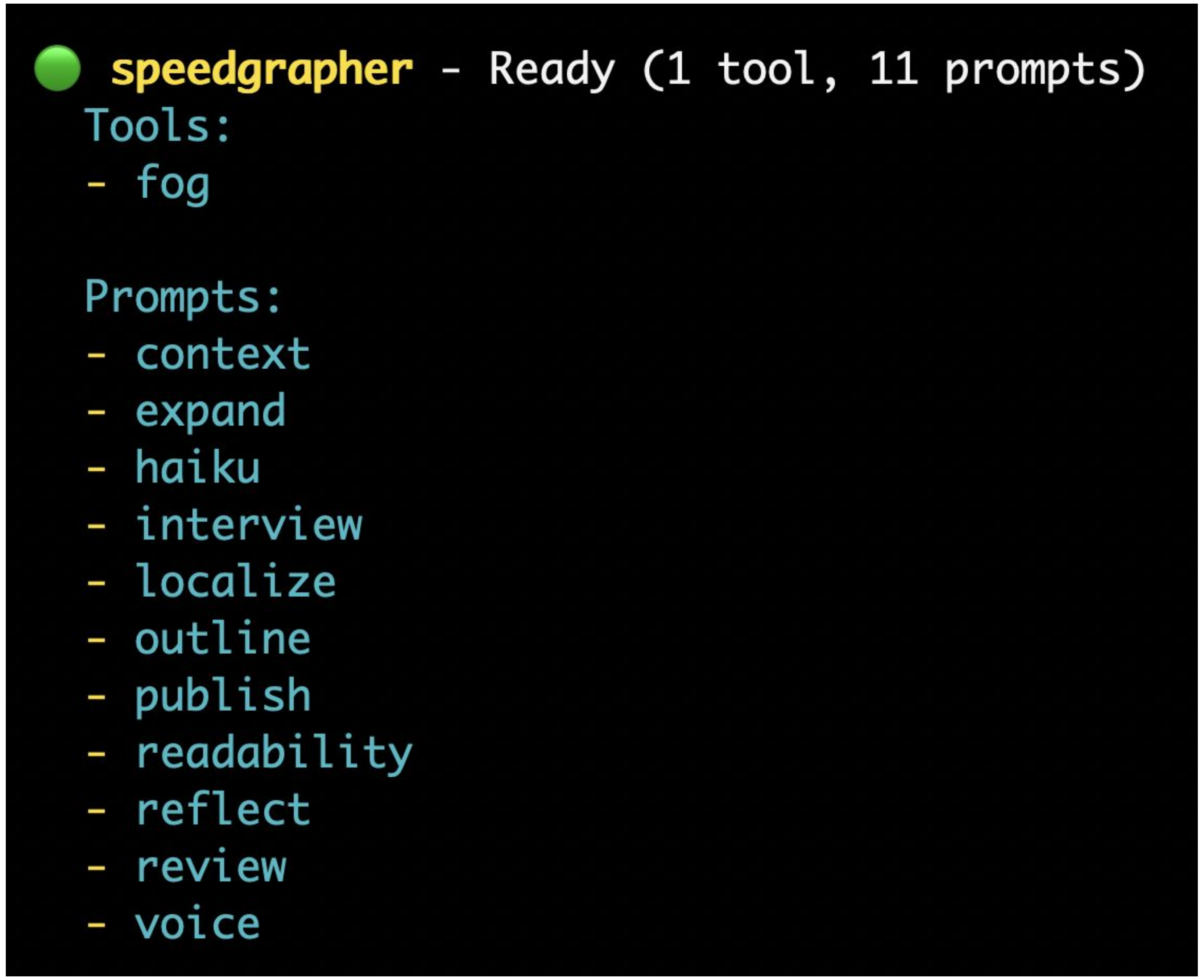
godoc.go



```go
func getDocHandler(ctx context.Context, s *mcp.ServerS

    if pkgPath == "" {…
    }

    args := []string{"doc"}
    if symbolName == "" {…
    } else {…
    }

    cmd := exec.CommandContext(ctx, "go", args...)
    var out bytes.Buffer
    cmd.Stdout = &out
    cmd.Stderr = &out
    if err := cmd.Run(); err != nil {…
    }

    docString := strings.TrimSpace(out.String())
    if docString == "" {…
    }

    return &mcp.CallToolResult{
        Content: []mcp.Content{
            &mcp.TextContent{Text: docString},
        },
    }, nil
}
```

Prompts

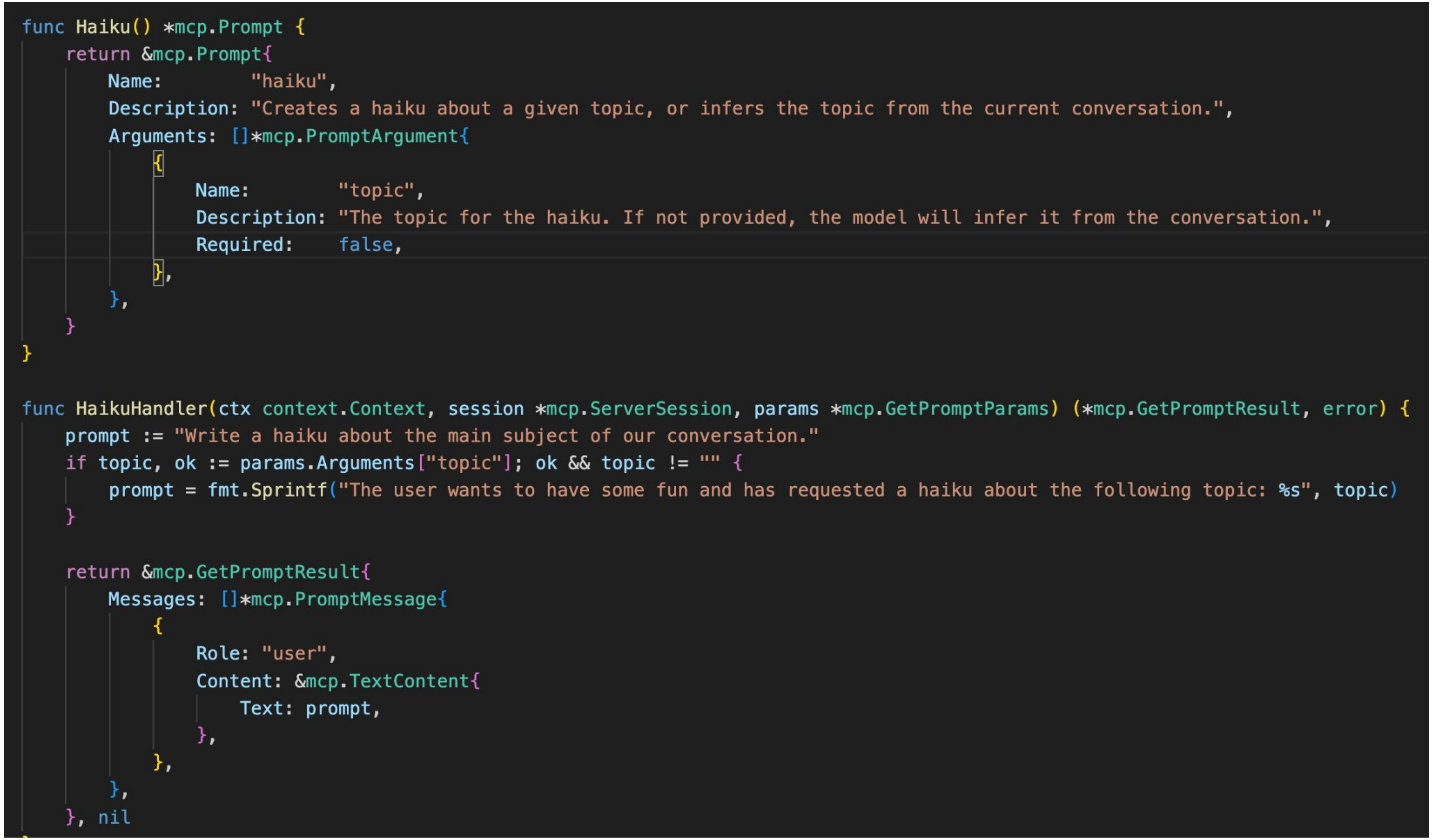
Prompts provide reusable templates. They allow MCP server authors to provide parameterized prompts for a domain, or showcase how to best use the MCP server.



```
🟢 speedgrapher - Ready (1 tool, 11 prompts)
  Tools:
  - fog

  Prompts:
  - context
  - expand
  - haiku
  - interview
  - localize
  - outline
  - publish
  - readability
  - reflect
  - review
  - voice
```

SPEEDGRAPHER

https://github.com/danicat/speedgrapher



```go
func Haiku() *mcp.Prompt {
	return &mcp.Prompt{
		Name:        "haiku",
		Description: "Creates a haiku about a given topic, or infers the topic from the current conversation.",
		Arguments: []*mcp.PromptArgument{
			{
				Name:        "topic",
				Description: "The topic for the haiku. If not provided, the model will infer it from the conversation.",
				Required:    false,
			},
		},
	}
}

func HaikuHandler(ctx context.Context, session *mcp.ServerSession, params *mcp.GetPromptParams) (*mcp.GetPromptResult, error) {
	prompt := "Write a haiku about the main subject of our conversation."
	if topic, ok := params.Arguments["topic"]; ok && topic != "" {
		prompt = fmt.Sprintf("The user wants to have some fun and has requested a haiku about the following topic: %s", topic)
	}

	return &mcp.GetPromptResult{
		Messages: []*mcp.PromptMessage{
			{
				Role: "user",
				Content: &mcp.TextContent{
					Text: prompt,
				},
			},
		},
	}, nil
```

Resources

Resources expose data from files, APIs, databases, or any other source that an AI needs to understand context.

```
{
  "uriTemplate": "weather://forecast/{city}/{date}",
  "name": "weather-forecast",
  "title": "Weather Forecast",
  "description": "Get weather forecast for any city and date",
  "mimeType": "application/json"
}

{
  "uriTemplate": "travel://flights/{origin}/{destination}",
  "name": "flight-search",
  "title": "Flight Search",
  "description": "Search available flights between cities",
  "mimeType": "application/json"
}
```

Client Concepts

Sampling: allows servers to request language model completions through the client

Roots: are a mechanism for clients to communicate filesystem access boundaries to servers.

Elicitation: Elicitation enables servers to request specific information from users during interactions

Let’s try something practical!

(what could go wrong?)

A Quick Demo

```
Your task is to create a Model Context Protocol (MCP) server to expose a “hello world”
tool. For the MCP implementation, you should use the official Go SDK for MCP and use the
stdio transport.

Read these references to gather information about the technology and project structure
before writing any code:
- https://raw.githubusercontent.com/modelcontextprotocol/go-sdk/refs/heads/main/README.md
- https://go.dev/doc/modules/layout

To test the server, use shell commands like these:
(
  echo
'{"jsonrpc":"2.0","id":1,"method":"initialize","params":{"protocolVersion":"2025-06-18"}}'
;
  echo '{"jsonrpc":"2.0","method":"notifications/initialized","params":{}}';
  echo '{"jsonrpc":"2.0","id":2,"method":"tools/list","params":{}}';
) | ./bin/godoctor
```

A Peek at the Future

Go SDK for MCP:
https://github.com/modelcontextprotocol/go-sdk

MCP Support for gopls:
https://tip.golang.org/gopls/features/mcp

Useful MCP Servers

Playwright: navigates web pages, takes screenshots, …

https://github.com/microsoft/playwright-mcp

Context7: retrieves documentation from a crowdsourced repo

https://context7.com/

What About Building Your Own?

How to Build a Coding Assistant with Gemini CLI, MCP and Go | Google Codelabs

Thank you!

danicat.dev

linkedin.com/in/petruzalek

@danicat83

Google Cloud