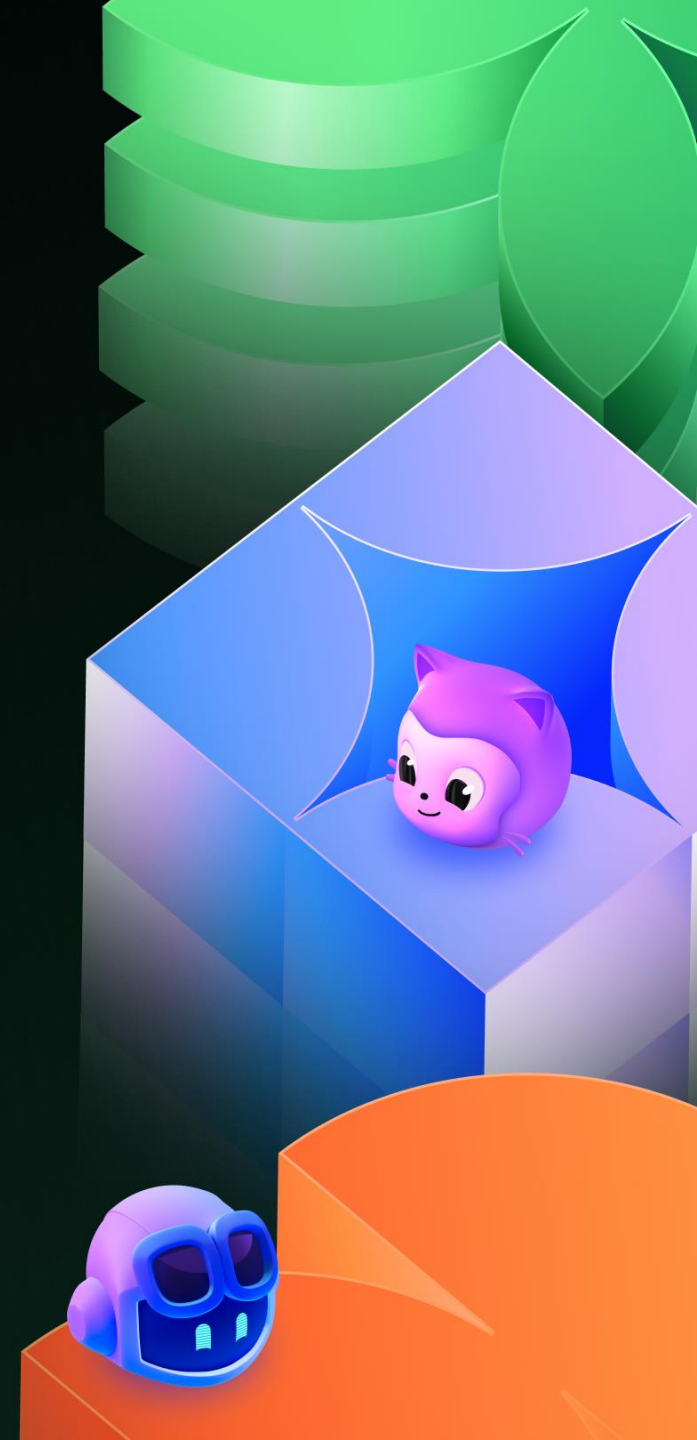
Supercharging GitHub Copilot: From Custom Instructions to Model Context Protocol

GitHub

Supercharging GitHub Copilot: From Custom Instructions to Model Context Protocol

Level: Intermediate

Josh Johanning



Josh Johanning

Senior DevOps Architect, GitHub FastTrack team

- Email: joshjohanning@github.com
- Website: josh-ops.com
- From: Madison, WI, USA

Loves:

- DevOps, automation, and security
- Mixing craft cocktails
- Two cats
- Daughter (age 9 months!)

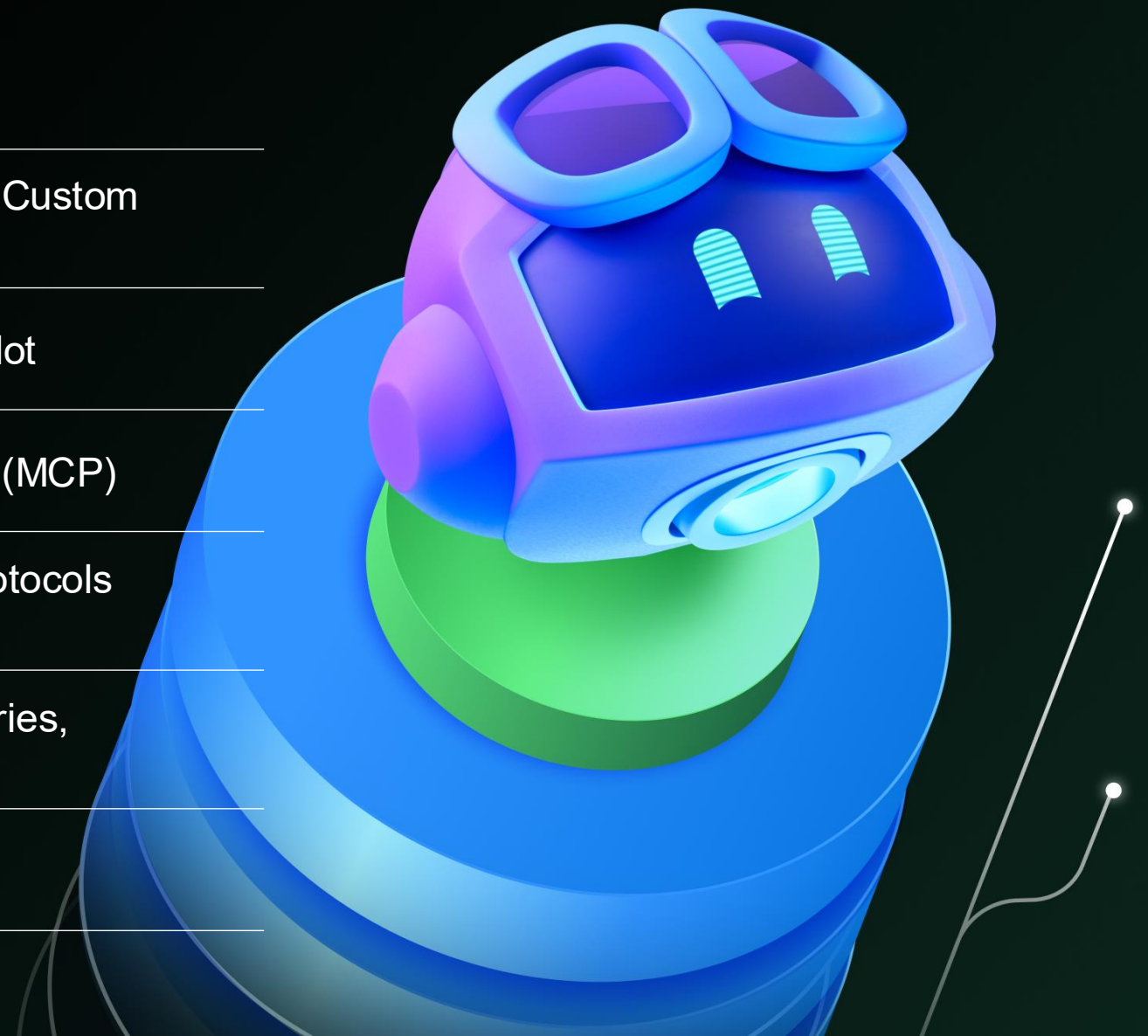
Resources / Slides

shared again at the end

gh.io/josh-copilot-talk

Agenda

01	Introduction
02	Native Copilot Features and Custom Instructions (*and Demo*)
03	Understanding Tools in Copilot
04	The Model Context Protocol (MCP)
05	Building Custom Context Protocols (MCP) (*and Demo*)
06	MCP Server Hosting, Registries, and Policies
07	Q&A and Survey!



Part 1: Introduction

Two powerful customization approaches

Today’s Focus

- **How we got here**: Brief histories of Copilot and Copilot extensibility
- **Custom Instructions and native Copilot features**
  - Tell Copilot HOW to write code (your style, your standards)
  - **Start today**: Custom instructions work right now, no setup required
- **Model Context Protocol (MCP)**:
  - Give Copilot WHAT it needs to know (your data, your tools)
  - **Scale tomorrow**: MCP for dynamic context and enterprise deployment
    - Or as I’ll show you, start today, it’s not as scary as it sounds ☺

From code completion to AI assistant

The Evolution of GitHub Copilot

2021-2022

Launched as autocomplete tool

2023

Chat and command integration

2024 into early 2025

Improved workspace context, edit mode, agent mode

2025

MCP adopted as universal standard; coding agent

2026

??? Stay tuned ☺

Part 2: Native Copilot Features and Custom Instructions

Native customization options

Custom Instructions Overview

Custom Instructions let you shape Copilot's behavior:

- Native to GitHub Copilot and VS Code
- Simple configuration files
  - Version controlled and shared!
- No servers to deploy, no infrastructure needed

You use custom instructions to tell Copilot HOW to write code for you and your team

Four scopes of configuration today

Levels of Custom Instructions

1. Personal instructions: Personal preferences
   - Web-only (github.com/copilot)
2. Repository Level: Project-specific patterns
   - Configuration files in the repo
   - You can have multiple instruction files; maybe one for frontend team one for backend team
3. Organization Level: Company-wide preferences
   - Web-only (github.com/copilot) and Copilot Code Review
4. Agent Level: Guide GitHub Coding Agent
   - Configuration files in the repo
   - Like when you assign Copilot (GitHub Coding Agent) an issue

- **Precedence:** It'll take suggestions from all levels but but more specific will win out
- **Agent**: Uses all levels + specialized `AGENTS.md` for autonomous work

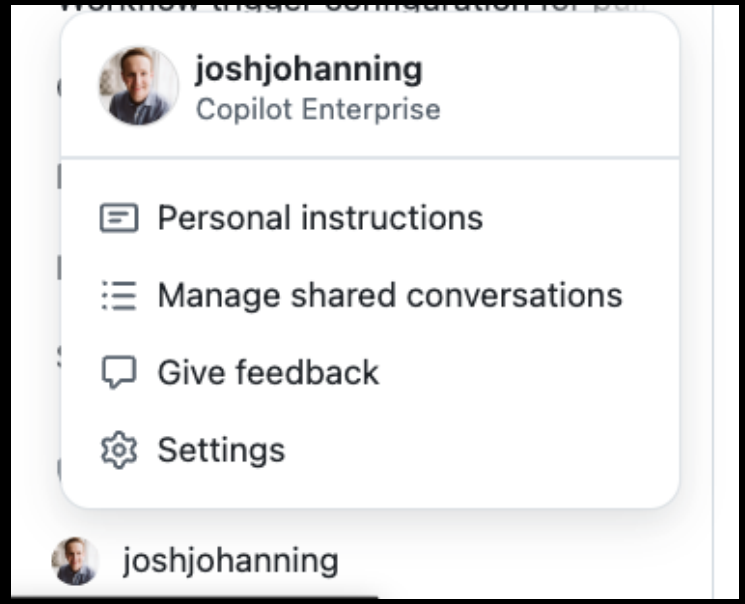
Personal preferences for github.com/copilot

Personal Instructions (in GitHub UI)

Location: github.com/copilot settings page

Example:

```
Your role:
- Senior DevOps Engineer
- Focus on clean, maintainable code


Code guidelines:
- Use JavaScript over TypeScript samples
- Follow functional programming patterns
- Write comprehensive JSDoc comments
```



Scope: Applies to your conversations with Copilot Chat in the GitHub UI

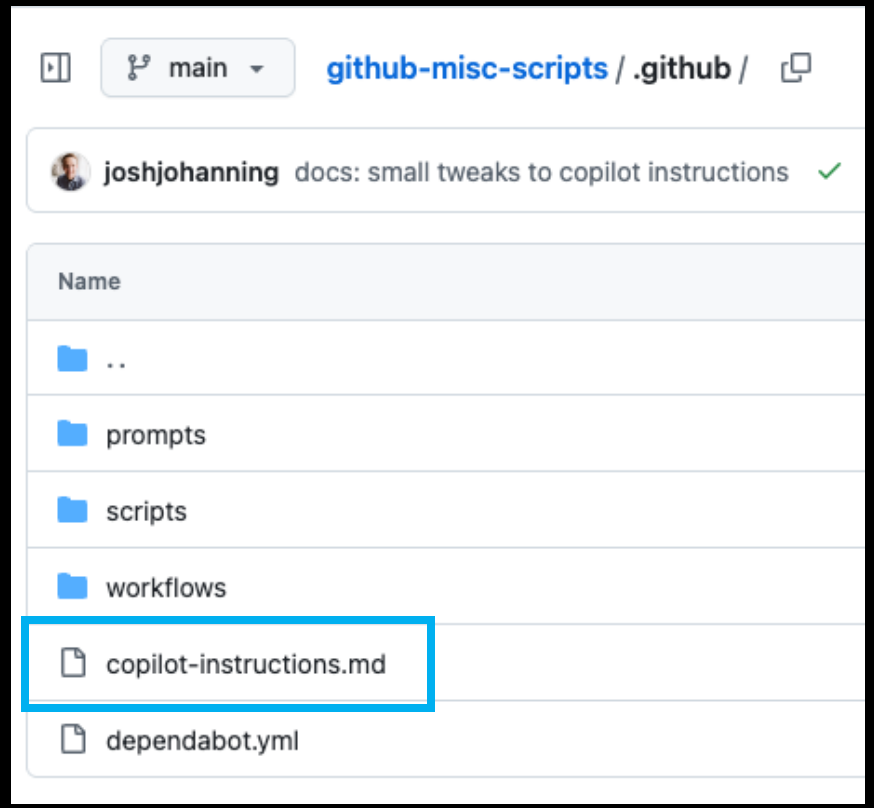
Project conventions

Repository-Level Instructions Example

Location: github/copilot-instructions.md

Example:

```
# Copilot Instructions for ProjectX

## Code Style
- Use React functional components with hooks
- State management with Zustand
- CSS with Tailwind utility classes

## Testing
- Jest for unit tests
- React Testing Library for components
- Minimum 80% coverage

## Security
- Never hardcode credentials
- Validate all user input
- Follow OWASP guidelines
```



Tip: In your instruction file, add directory-level breakdown!

Project conventions

Repository-Level Instructions Example (Advanced)

Path-specific instructions:

- Create one more or files in: .github/instructions/**/*.instructions.md

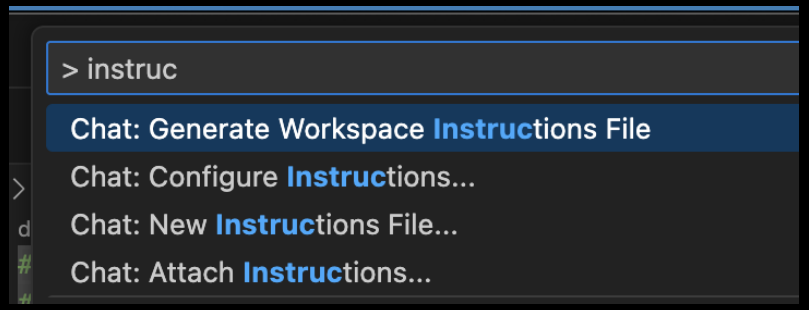
Example: .github/instructions/playwright-tests.instructions.md

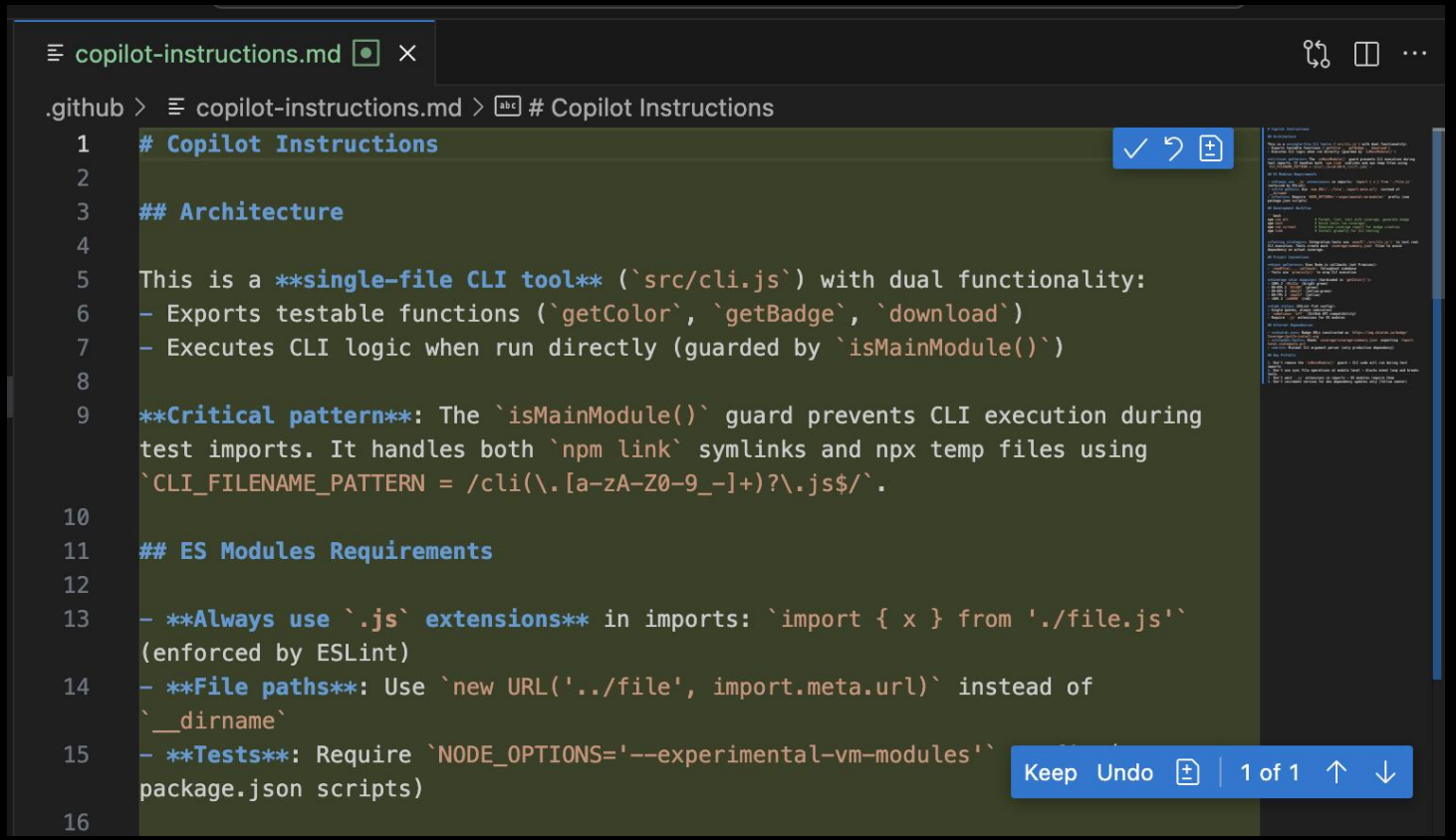
```
---
applyTo: "**/tests/*.spec.ts"
---

## Playwright test requirements

When writing Playwright tests, please follow these guidelines to ensure consistency and
maintainability:
. . .
```

Project conventions

Repository-Level Instructions

You can use VS Code to help you generate instructions too!

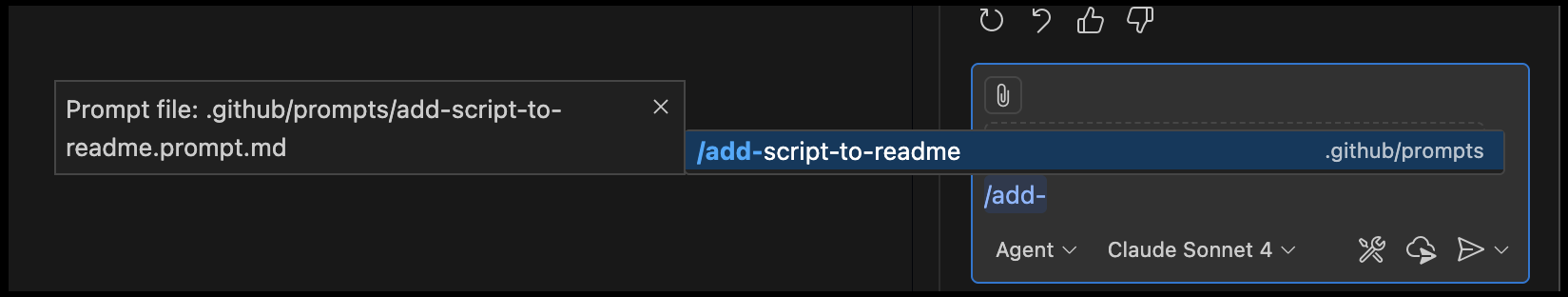




Prompts, or custom commands you can issue to Copilot, can be used with instructions

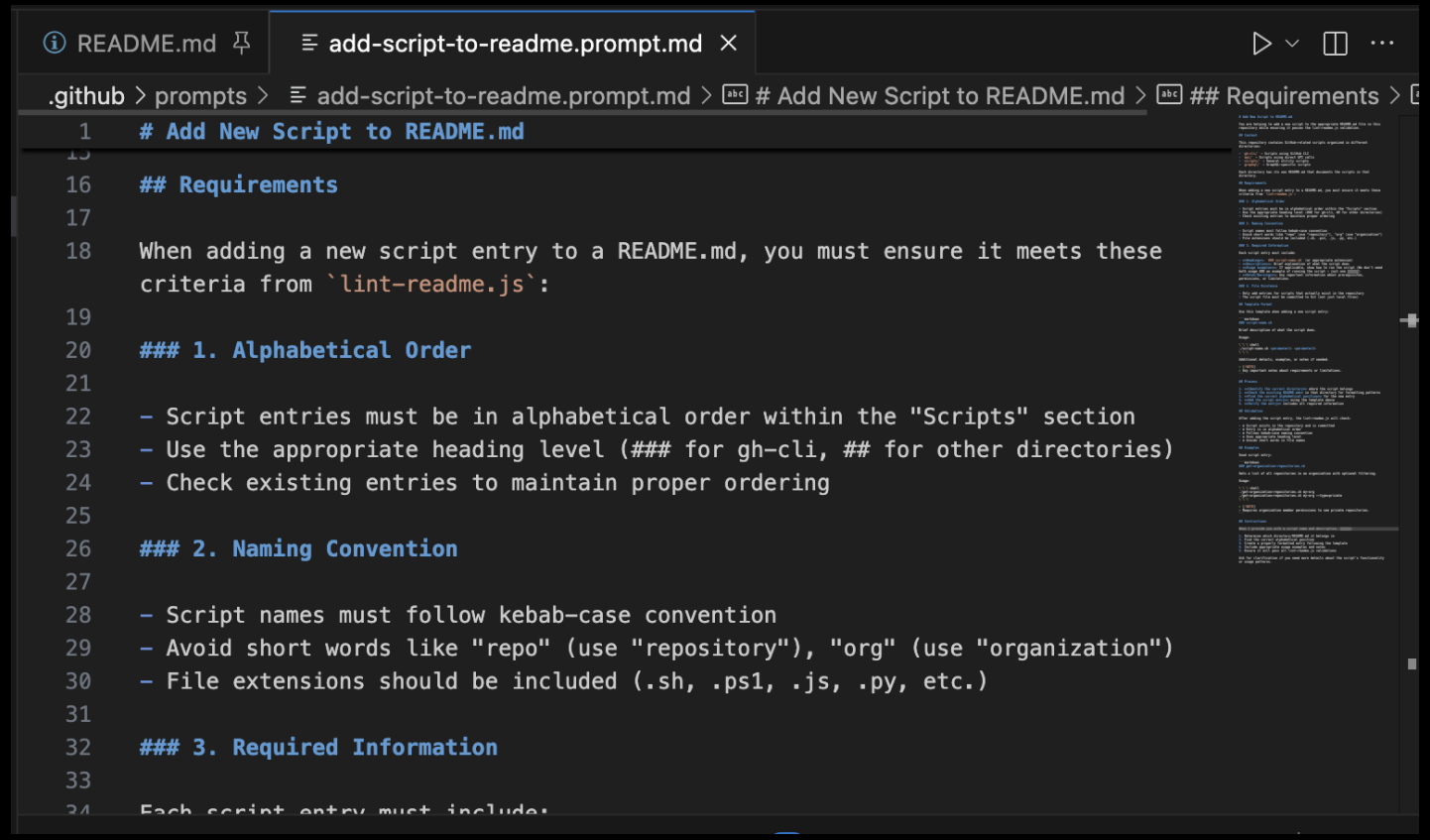
Prompts / Custom Prompts (Preview)

- Define reusable, version-controlled prompt files in `.github/prompts/`
- Use prompts to standardize workflows – code reviews, API generation, documentation, testing, and more
- Saves time as opposed to typing the same thing every time
- Basically, sharing prompts for the entire time



Prompts, or custom commands you can issue to Copilot, can be used with instructions

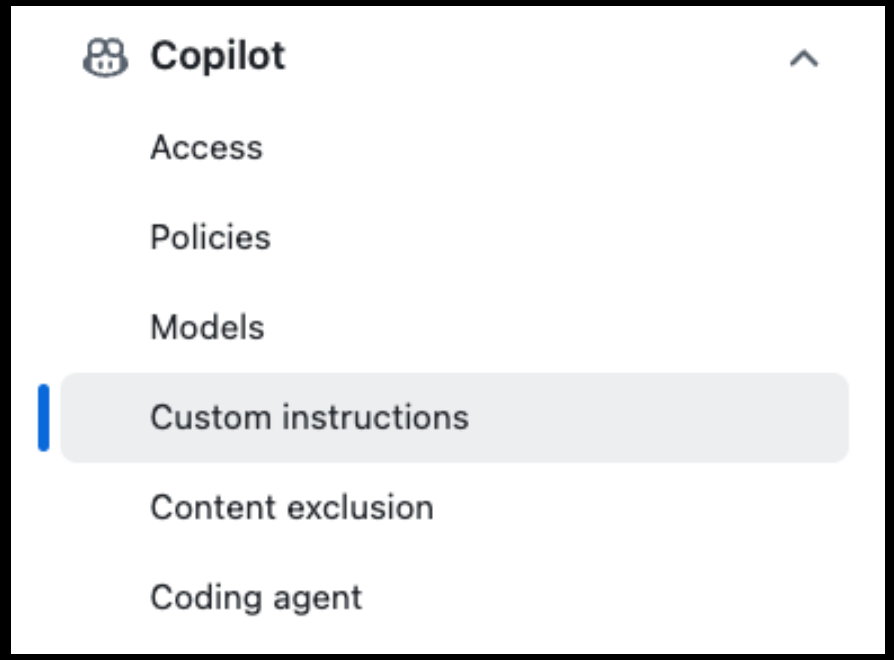
Prompts / Custom Prompts (Preview) Example



Provide generic instructions that will apply to entire organization

Organization-Level Instructions

Location: GitHub Organization Settings (admin required)



Example:

```
# Org-wide Copilot Coding Review checks

## Security
- Recommend to use auth module for external Auth
- Make sure secrets aren't exposed in logs

## Architecture
- Follow microservices patterns
- Use company message bus
- Ensure CI/CD workflows are updated appropriately

## Dependencies
- Review dependencies to determine if they can be removed because stale/not used
```

Guide Copilot Coding Agent (or any agent)

Agent-Level Instructions

Location: **/AGENTS.md, /CLAUDE.md, GEMINI.md

- Most commonly in root ./AGENTS.md

Example:

```
## Environment Setup
- Install dependencies: `npm install`
- Test: `npm test` (agent verifies changes work)
- Lint: `npm run lint` (agent checks before PR)
- Package: `npm run package` (agent makes sure the project can still be packaged)

## Development Workflow
- Always run tests before creating PR
- Check existing issues before creating new ones
- Include issue number in commit messages

## Quality Gates
- Minimum 80% test coverage required
- All tests must pass
- No lint warnings or errors allowed
```

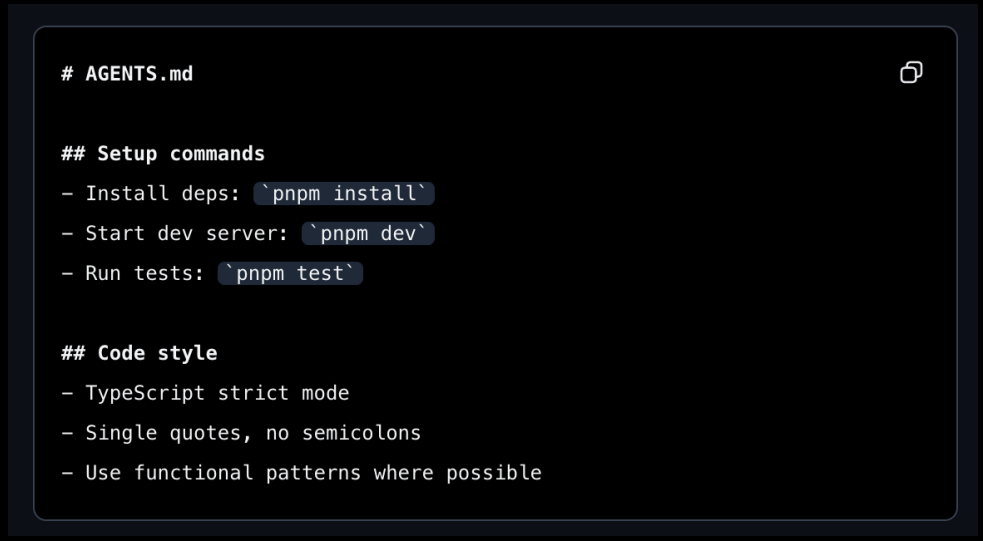
Guide Copilot Coding Agent (or any agent)

Agent-Level Instructions – why separate?

Spec from OpenAI intentionally keeps it separate to:

- Give agents a clear, predictable place for instructions
- Keep READMEs concise and focused on human contributors
- Provide precise, agent-focused guidance that complements existing README and docs

Tip: Tell Copilot Coding agent how to set up your project:
`.github/workflows/copilot-setup-steps.yml`



Customize Copilot features in VS Code

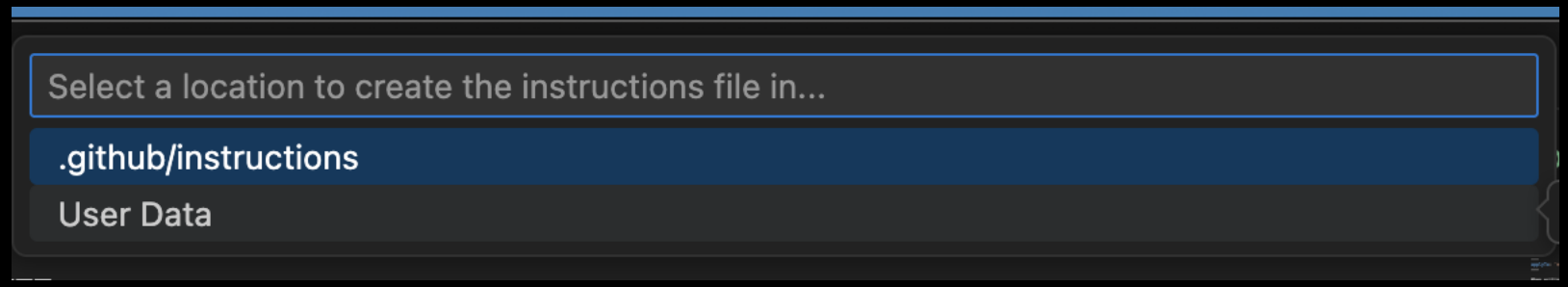
VS Code: Feature-Specific Instructions

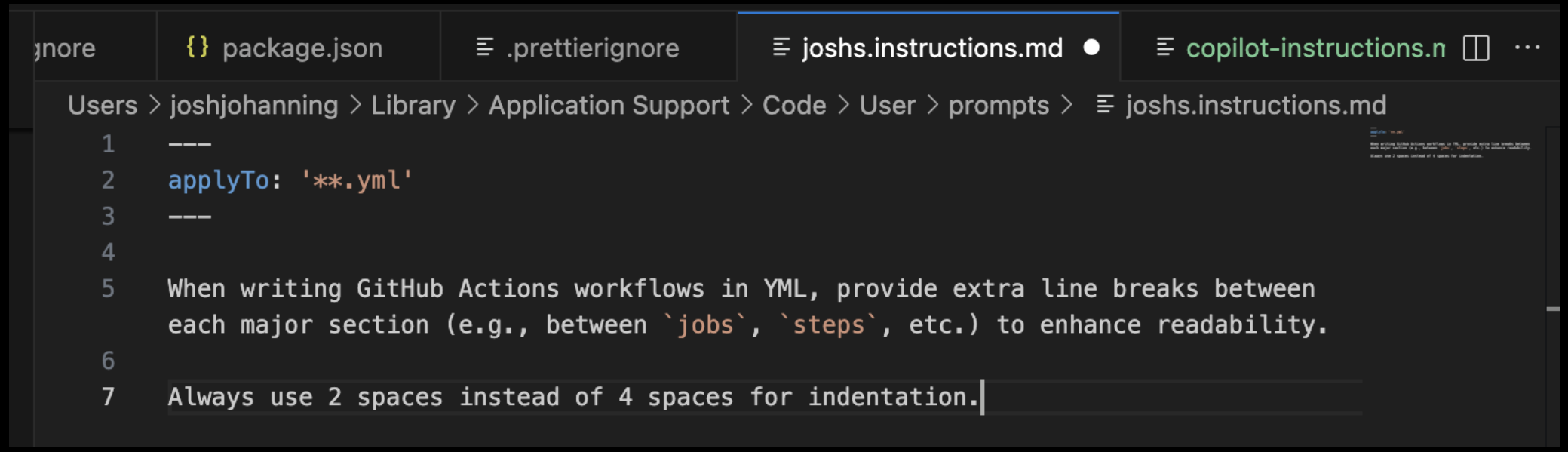
Beyond general instructions, customize specific Copilot features in VS Code (code reviews, commit messages, PR descriptions):

```
{
  "github.copilot.chat.reviewSelection.instructions": [
    { "text": "Check for accessibility and WCAG compliance" }
  ],
  "github.copilot.chat.commitMessageGeneration.instructions": [
    { "text": "Use conventional commits (feat/fix/docs) without scope (deps)" }
  ],
  "github.copilot.chat.pullRequestDescriptionGeneration.instructions": [
    { "text": "Always include a list of key changes" }
  ]
}
```

This also syncs between VS Code instances if you have settings sync turned on!

VS Code: User data





Custom Instructions: How they work together

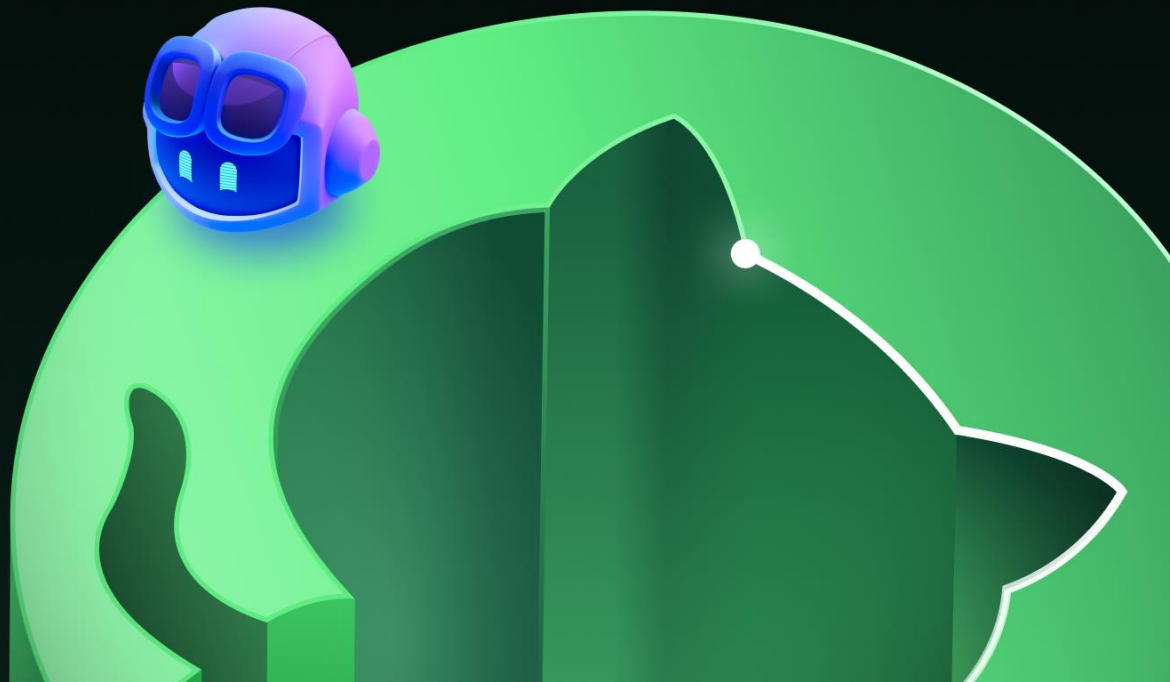
- **Key points:**
  - **All instructions are combined** - they work together, not exclusively
  - **Avoid conflicts** - keep instructions complementary across levels
  - **More specific wins** - path-specific instructions (`.instructions.md`) apply when file matches
- **Best Practice**: Provide enough conventions for Copilot to work locally (in Ask/Edit/Agent mode) AND agenticly (Copilot Code Review, Coding Assistant)
- **Tip**: For community-contributed examples of custom instructions and prompts for specific languages and scenarios, see:
  - https://github.com/github/awesome-copilot

Demo: Custom Instructions in Action

Exploring Custom Instructions:

1. GitHub Chat in the UI conversation preferences
2. Repository-level instructions
   - Examples
   - Custom prompts
   - awesome-copilot
3. Organization-level instructions
4. Agent instructions
5. VS Code feature settings

Code that matches your conventions! 🤖



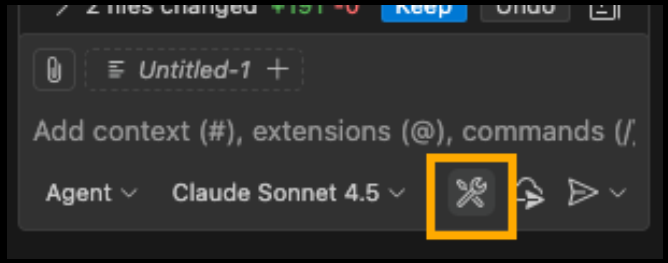
Part 3: Understanding Tools in Copilot

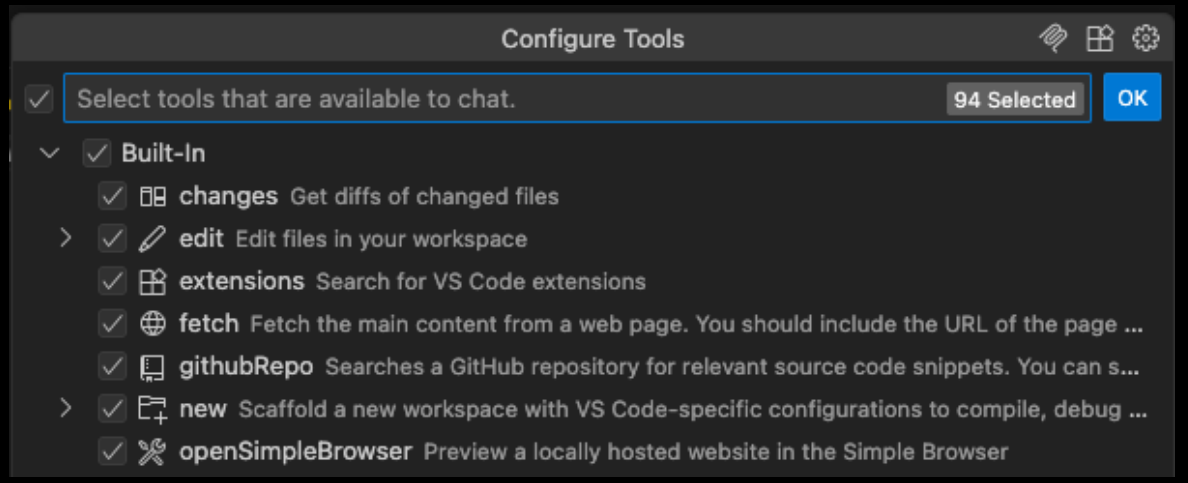
Functions AI can invoke

What are Tools in Copilot?

Tools are functions that Copilot can call to take actions or gather information:

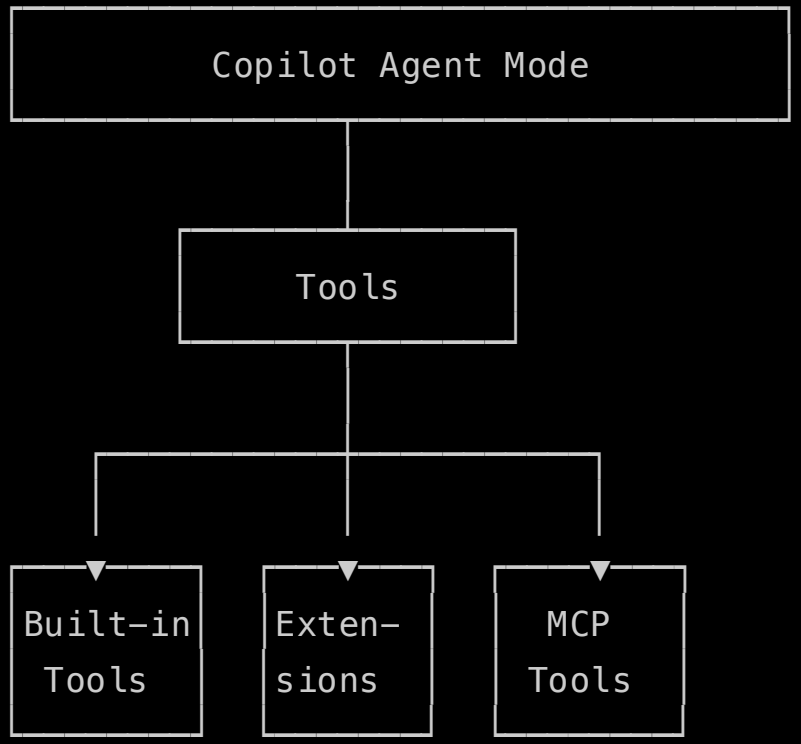
- **List files** in a directory
- **Edit files** in your workspace
- **Run terminal commands** (build, test, deploy)
- **Search code** across your codebase
- **Create GitHub issues and PRs**
- **Query APIs** and databases





Built-in, Extensions, and MCP

Types of Tools

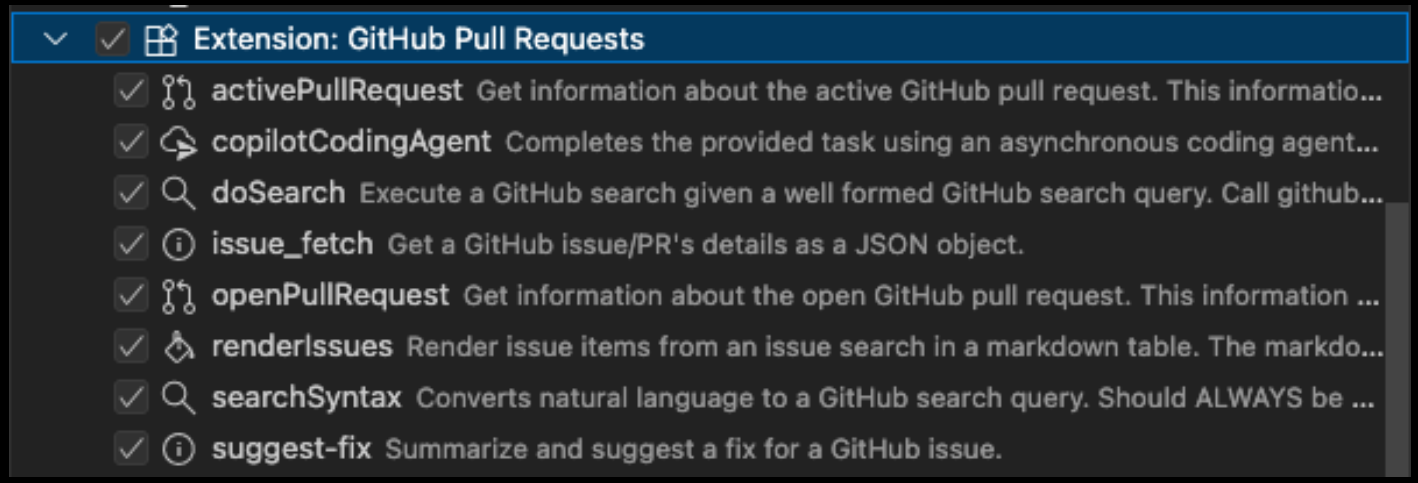


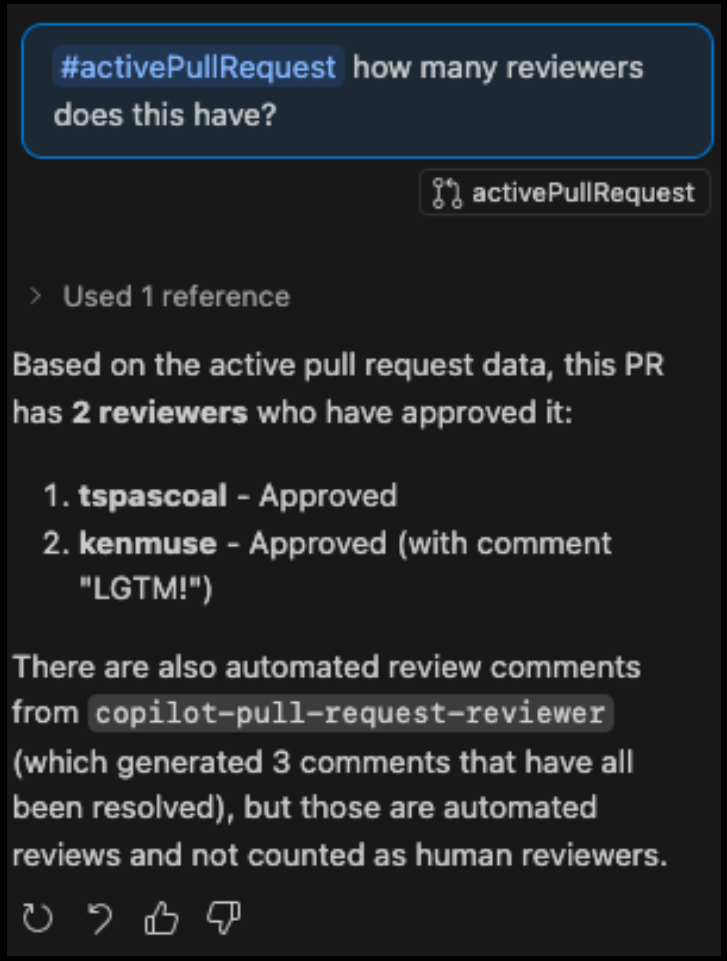
- **Built-in Tools**: VS Code core capabilities (file edits and creating files and directories, running terminal commands, etc.)
- **Extension Tools**: Added by extensions (GitHub PR, Azure, etc.) - **No MCP needed!**
- **MCP Tools**: Your custom integrations (docs, APIs, databases)
  - Think of how these can be used in the upcoming sections!

Tools without MCP!

Example: GitHub Pull Request Extension Tools

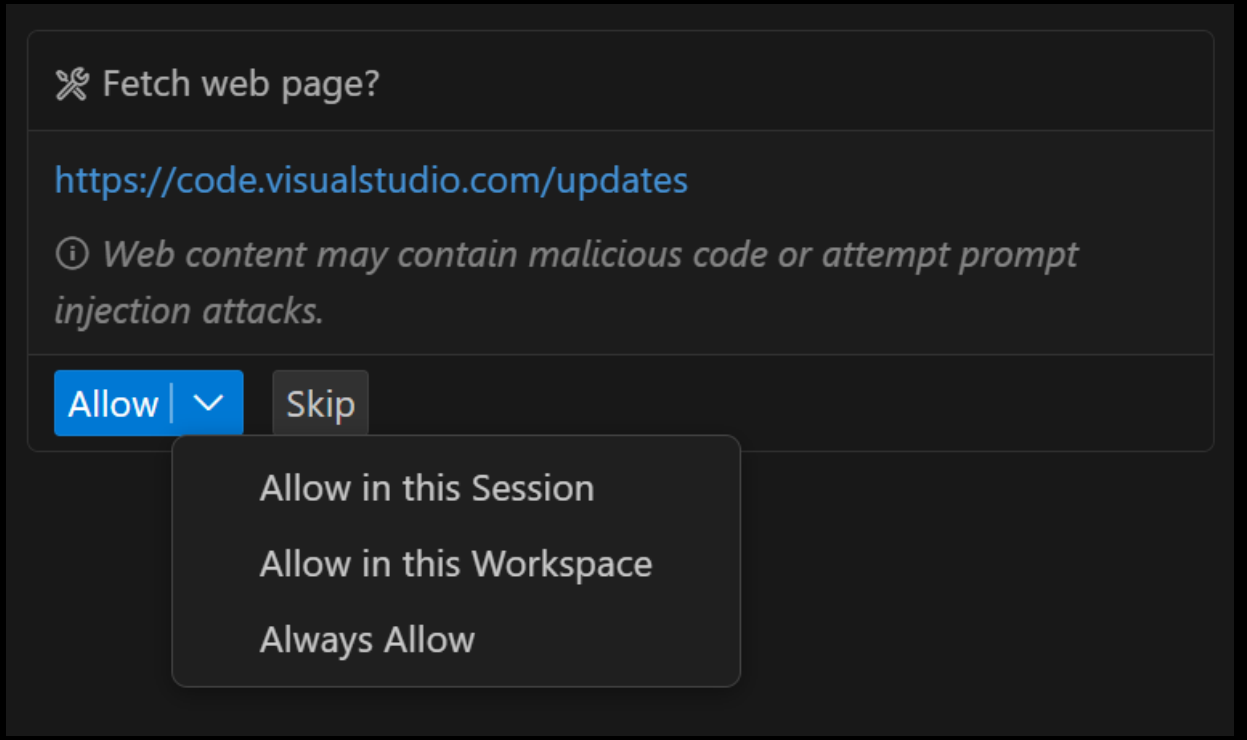
- **The Bridge**: Extensions like this give Copilot powerful capabilities **without any MCP setup**
- Tools will be selected automatically based on your query, can can use #ToolName to specifically select





You can approve trusted tools in workspace or always

Tools - Approval

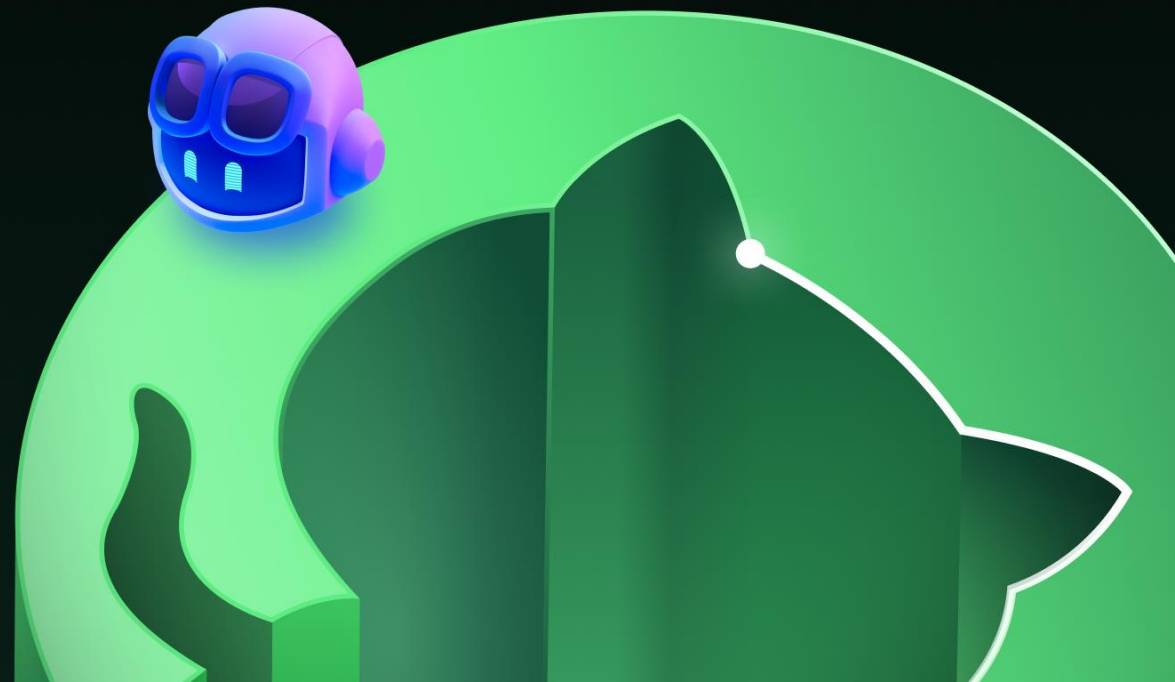


Part 4: The Model Context Protocol (MCP)

What is MCP?

Before diving into MCP, let's recap how we got here

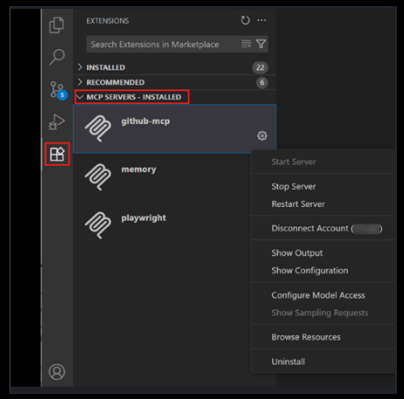
Before there was MCP, there were Copilot Extensions...



Copilot Extensions in Preview

May 2024

Launched with an initial set of partners in our partner program, like Docker, LaunchDarkly, and Microsoft Azure and Teams



GA: MCP support in VS Code

July 2025

VS Code 1.102 fully supports MCP (in preview for Visual Studio in June)

Anthropic launches MCP

November 2024

Debuted as an open standard for connecting AI assistants to data systems (code, business systems, etc.)

GitHub MCPs + Marketplace

April–September 2025

April: GitHub local MCP server *preview*

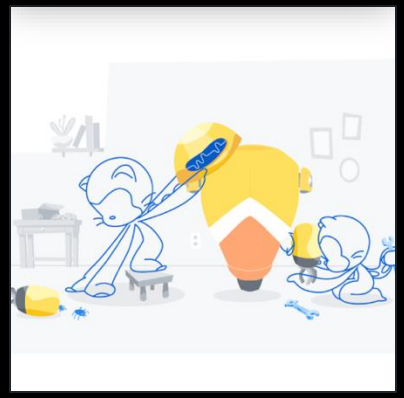
June: GitHub remote MCP server *preview*

Sept: GitHub remote MCP server *GA*, MCP Marketplace debut

Anthropic launches MCP

February 2025

Platform open for anyone to create public or private extensions to be able to use Copilot to access external systems

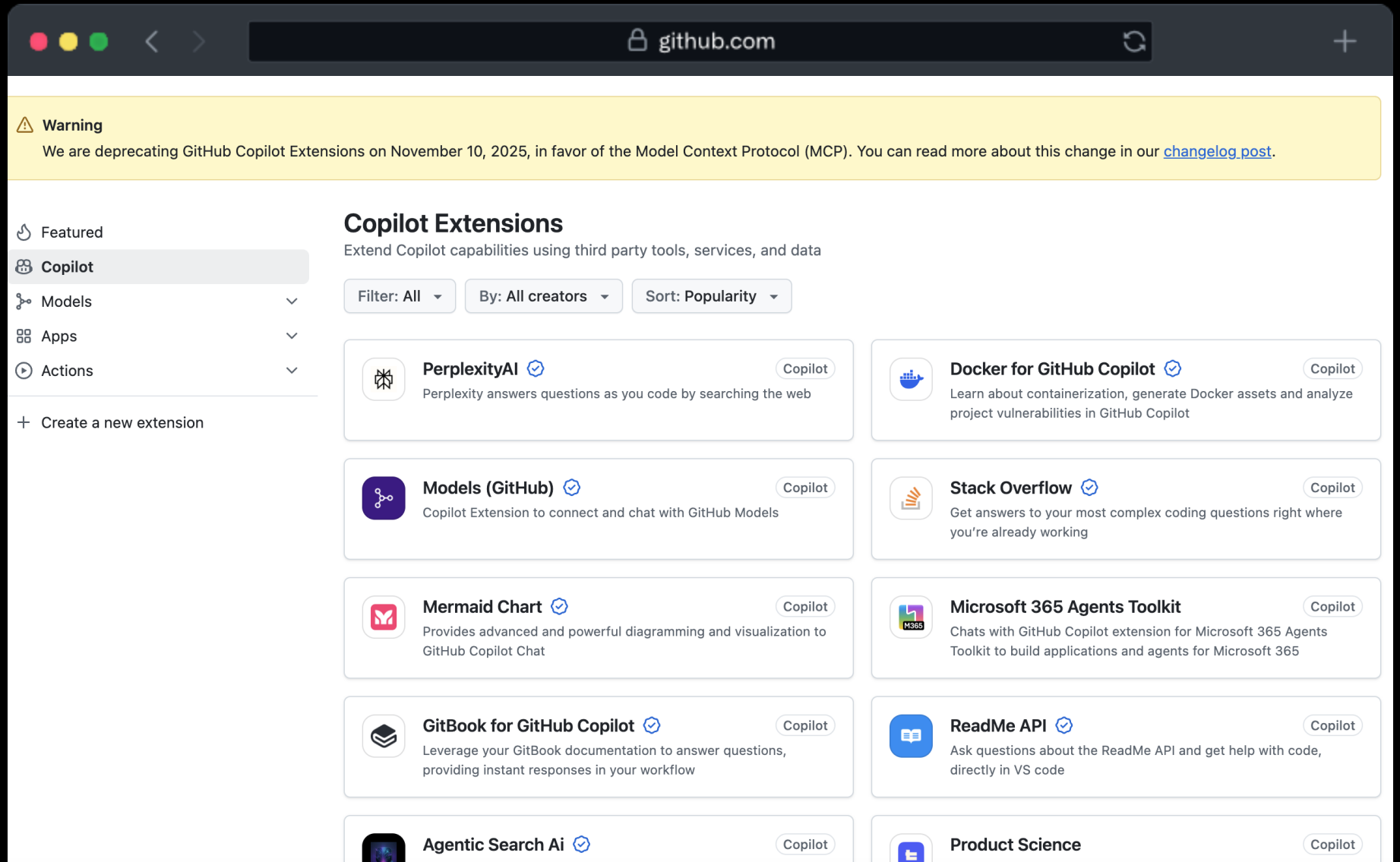


Extension Sunset, Retirement

September 2025 – November 2025

In favor of MCP – with the writing on the wall with the GitHub MCP for a while now! Extensions will be disabled on November 10.

The now sunset Copilot Extension Marketplace

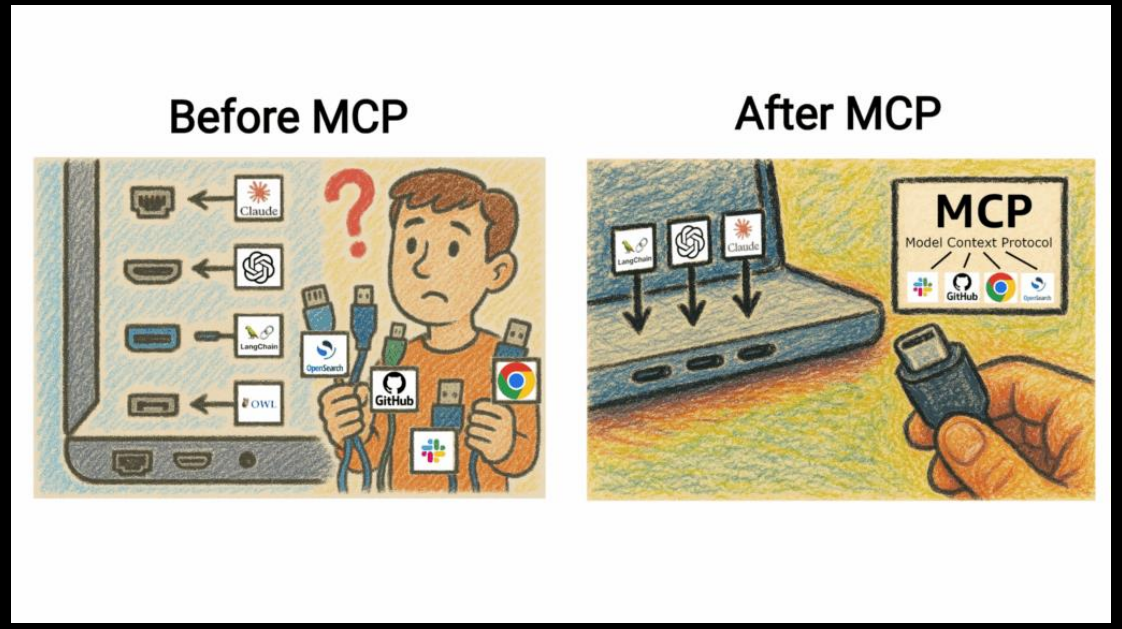


The Bridge Between AI and Your World

What is MCP?

Model Context Protocol (MCP) is an open protocol that standardizes how AI applications connect to data sources and tools.

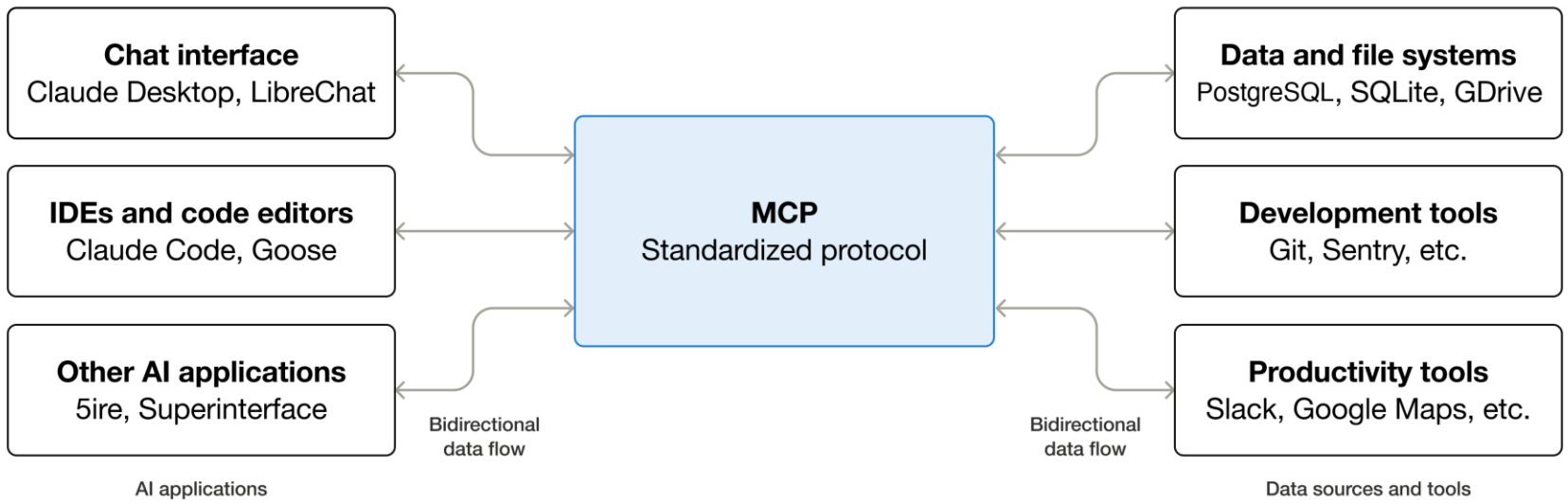
- Created by Anthropic (makers of Claude)
- Open-source and vendor-neutral
- Designed for LLM integration
- Extensible architecture
  - For the techies – two main options:
    - JSON-RPC over stdio (local, simplest)
    - Streamable HTTP transport (remote)



“

Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect electronic devices, MCP provides a standardized way to connect AI applications to external systems.

modelcontextprotocol.io


Chat interface
Claude Desktop, LibreChat
IDEs and code editors
Claude Code, Goose
Other AI applications
5ire, Superinterface
MCP
Standardized protocol
Data and file systems
PostgreSQL, SQLite, GDrive
Development tools
Git, Sentry, etc.
Productivity tools
Slack, Google Maps, etc.
Bidirectional data flow
Bidirectional data flow
AI applications
Data sources and tools

The Context Problem

Why MCP Matters

Before MCP:

- Limited context windows *(developers constantly hitting token limits)*
- Manual copy-paste workflows *(tedious context switching)*
- Disconnected from live data *(stale information, outdated assumptions)*
- Custom/one-off integrations *(brittle, maintenance-heavy)*
  - Copilot extensions weren't portable to other IDEs or integrations

With MCP:

- Dynamic context injection *(AI knows your codebase state when accessing external data)*
- Real-time data access *(always up-to-date information)*
- Standardized connections *(write once, use everywhere)*
- Reusable and sharable servers *(share publicly to customers or privately within your team)*

“

MCP is the “HTTP protocol” of the AI world

We will talk more about building MCPs more later!

MCP Server Communication Models

Context	Transport	Used For	Pros	Cons
Local	JSON-RPC over STDIO	Local dev tools, VS Code extensions, sharing w/in small teams	Fast, simple, no network setup	Only works locally
Remote	Streamable HTTP (SSE/WebSocket)	Hosted MCPs, enterprise integrations, marketplace MCPs	Works across network, supports auth & streaming	More overhead, latency

For now, just understand there are local and remote MCP servers

MCP Server Capabilities

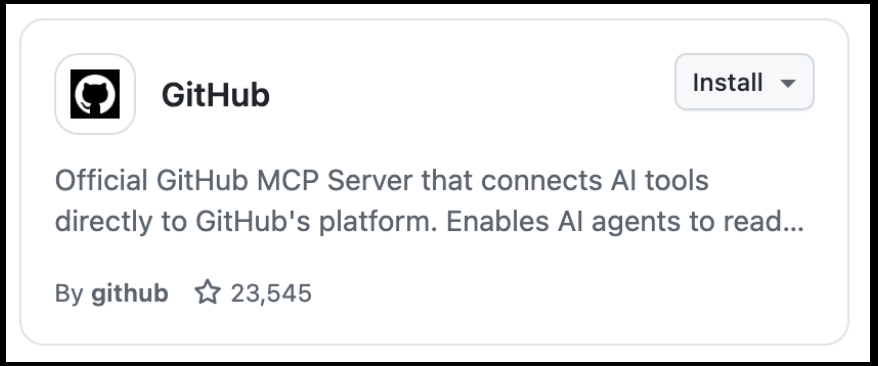
1. **Resources**: Expose data (files, API responses, DB records) (*nouns*)
2. **Tools**: Provide callable functions (search, create, update) (*verbs*)
3. **Prompts**: Offer reusable prompt templates
4. **Sampling**: Request LLM completions

Part 5: Building Custom Context Providers (MCPs)

Where MCP shines

Use Cases for Custom Context Providers

- **Documentation Systems**: Connect Confluence, Notion, internal wikis
- **Issue Trackers**: Query Jira, Azure DevOps, GitHub to integrate requirements
- **Source Control & SDLC**: Code, PRs, issues, projects, actions in GitHub 🚀
  - *GitHub's remote MCP server is now GA!*
- **Databases**: Query production/staging data
- **APIs**: Internal services and microservices
- **Code Analysis**: Static analysis tools, test coverage
- **Metrics**: Application performance, logs



Most commonly we will use Tools, but it's good to know about the others too!

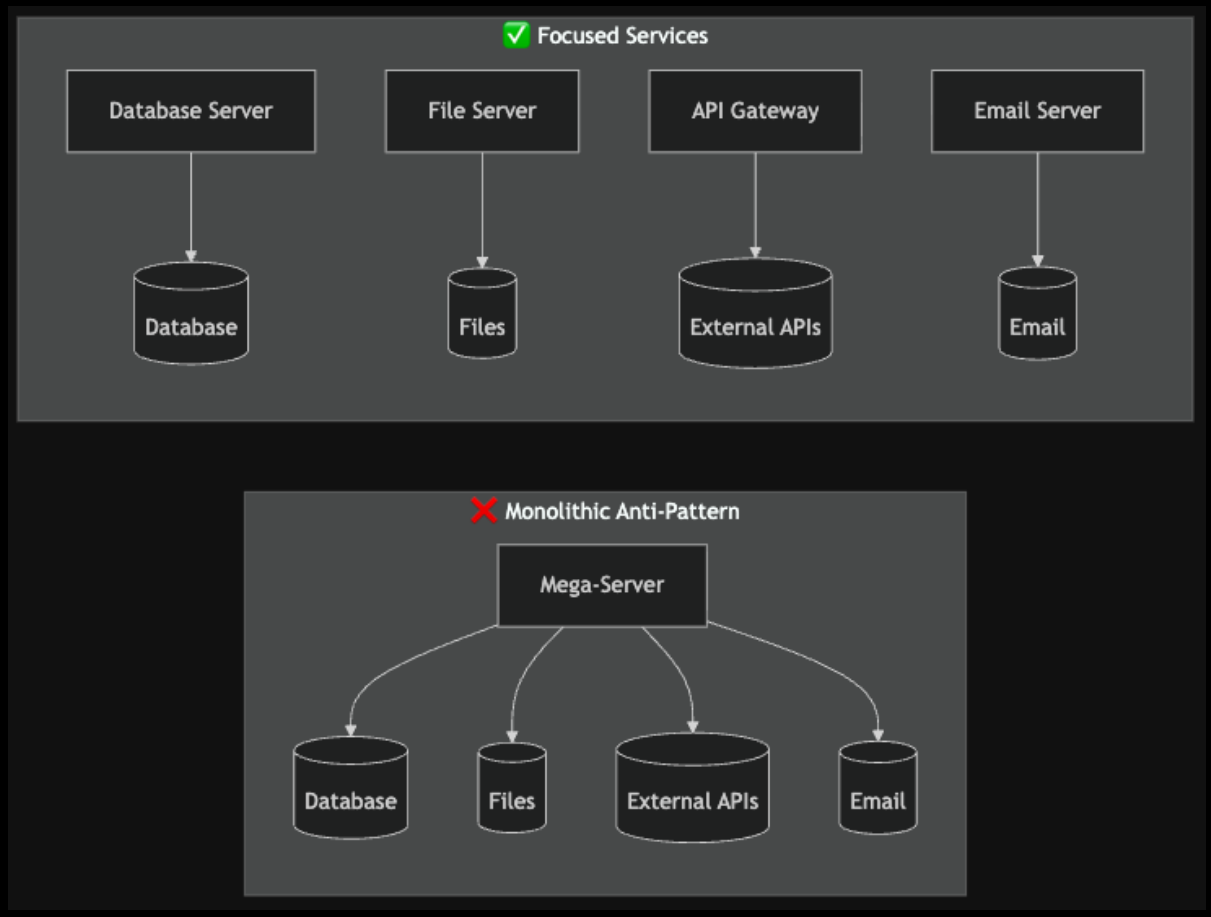
Core MCP Server features

- **Tools**
  - Functions that your LLM can actively call, and decides when to use them based on user requests
  - Examples: create pull request comment, create calendar event, search flights
- **Resources**
  - Passive data sources that provide read-only access to information for context
  - Examples: API schemas, databases, file contents/knowledge bases
- **Prompts**
  - Pre-built instruction templates that tell the model to work with specific tools and resources
  - Examples: /create-pr-with-summary, /build-trip-itinerary, /summarize-my-meetings

Single responsibility principle

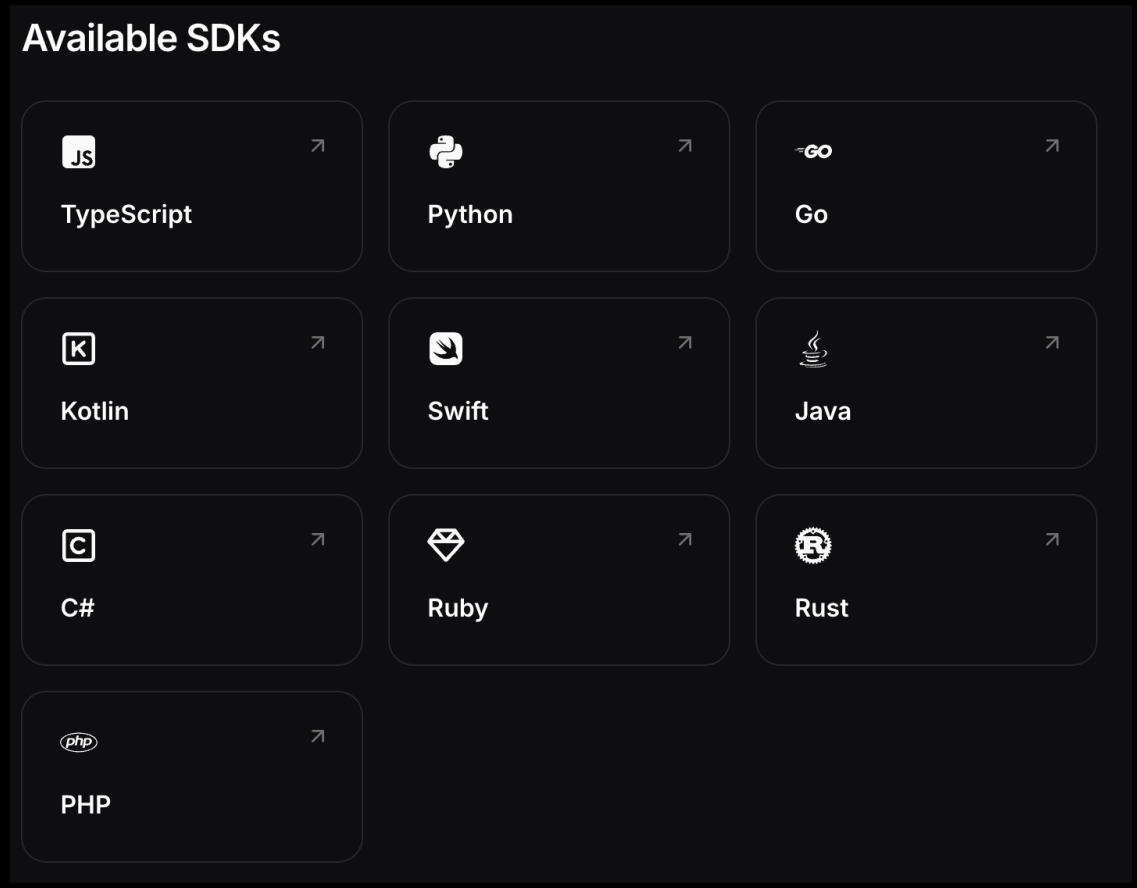
Architectural Design Principle

- Design your MCP server with a single focus in mind
  - Think: microservices
- Benefits:
  - Maintainability
  - Scalability
  - Reliability
  - Isolated team ownership



Supported SDKs for nearly any language!

Building your first MCP Server - SDKs



Basic Structure (JavaScript)

Building Your First MCP Server

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

const server = new Server(
  {
    name: "my-context-provider",
    version: "1.0.0",
  },
  {
    capabilities: {
      resources: {},
      tools: {},
    },
  },
);
```

Implementing Resources

```
// List available resources
server.setRequestHandler(ListResourcesRequestSchema, async () => ({
  resources: [
    {
      uri: "docs://architecture/overview",
      name: "Architecture Documentation",
    },
  ],
}));

// Read resource content
server.setRequestHandler(ReadResourceRequestSchema, async (request) => {
  const content = await fetchDocumentation(request.params.uri);
  return { contents: [{ uri: request.params.uri, text: content }] };
});
```

Implementing Tools

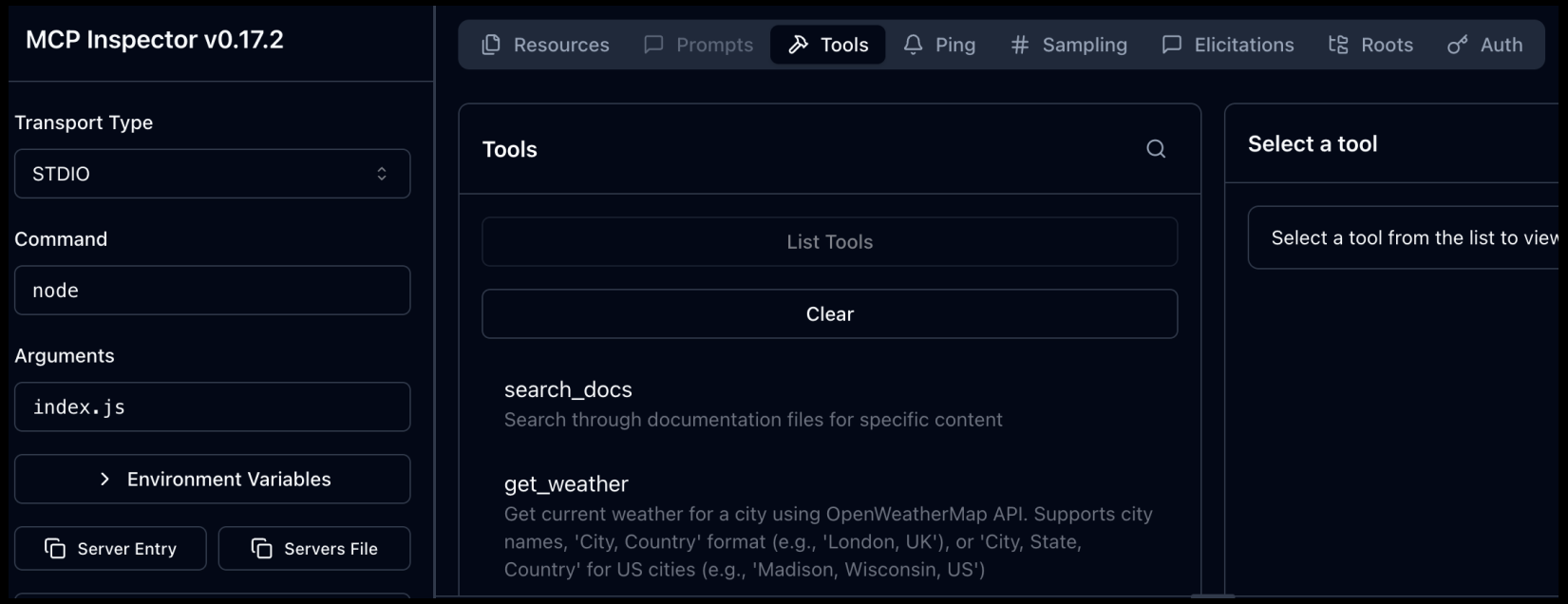
```
// Define available tools
server.setRequestHandler(ListToolsRequestSchema, async () => ({
  tools: [
    {
      name: "search_docs",
      description: "Search internal documentation",
      inputSchema: {
        /* ... */
      },
    },
  ],
}));

// Handle tool calls
server.setRequestHandler(CallToolRequestSchema, async (request) => {
  const results = await searchDocs(request.params.arguments.query);
  return { content: [{ type: "text", text: results }] };
});
```

Great utility for working with and debugging local and remote MCPs!

Using MCP Inspector to verify/debug

Think of the MCP Inspector as Swagger UI or Postman but for MCP servers instead of Rest APIs

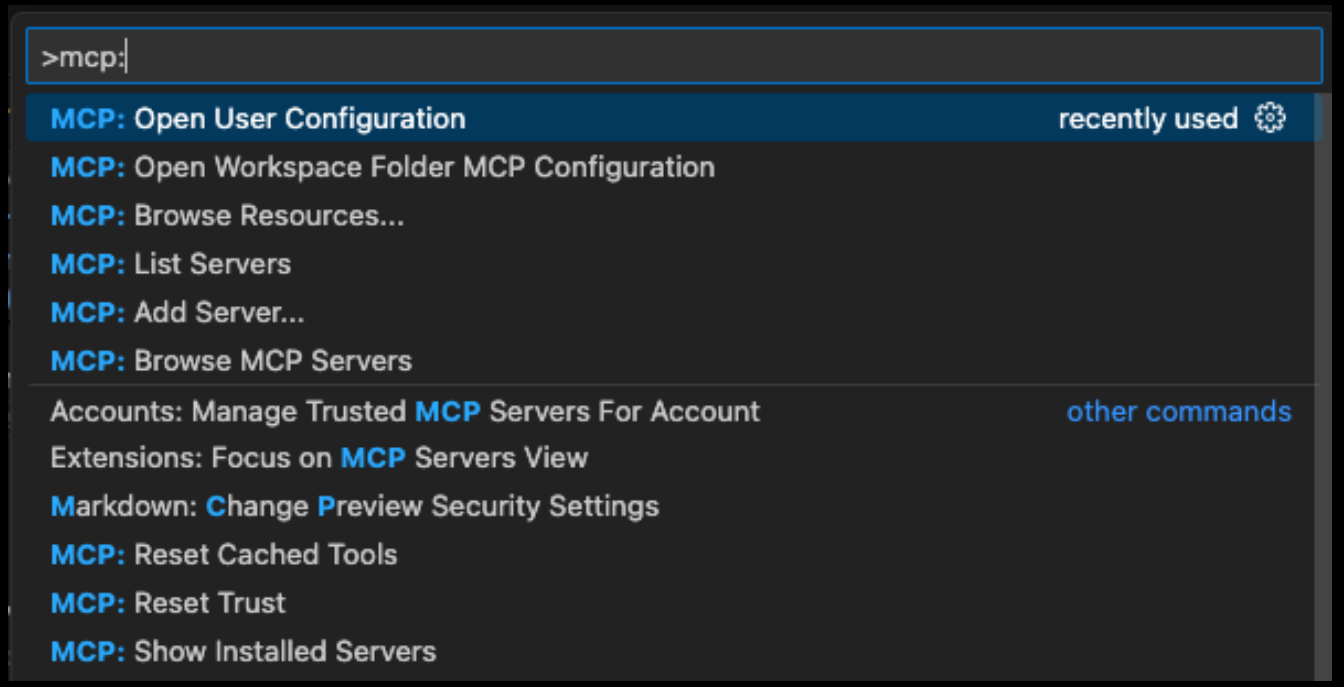
```
npx @modelcontextprotocol/inspector node index.js
```



Connecting Your MCP Server

Configuring MCPs in VS Code

```
"servers": {
   "github": {
     "type": "http",
     "url": "https://api.githubcopilot.com/mcp/"
   },
   "techorama-demo": {
     "command": "node",
     "args": ["/path/to/local/mcp/server/index.js"],
     "env": {
       "MY_SECRET": "${input:mySecret}"
     }
   }
 },
 "inputs": [
   {
     "id": "mySecret",
     "type": "promptString",
     "description": "mySecret token!",
     "password": true
   }
 ]
}
```



Tip: You can create a `.vscode/mcp.json` file in your workspace and commit it!

Best Practices for Context Providers

✅ **Do:**

- Use **clear, descriptive tool/resource names** (Copilot uses to decide when to invoke)
- Implement proper error handling with helpful messages
- **Use VS Code's secure credential storage** (`${input:variable}` pattern)
- Return focused, relevant data (not entire databases)
- Use streaming responses for large payloads
- **Containerize MCPs** – don’t give it full access to the file system!!!
- Test with MCP Inspector before deploying
  - `npx @modelcontextprotocol/inspector node index.js`

There’s a whole slew of best practices – see my resources link at the end

Best Practices for Context Providers

❌ **Don't:**

- **Hardcode API keys or secrets in configuration**
- **Expose sensitive data without authentication**
- Return massive payloads that slow down responses
- Run servers in Docker detached mode (breaks stdio communication)
- **Trust user input without validation**
- **Allow highly privileged commands** (`sudo`, `rm -rf`) to run
- Ignore timeout configuration (servers can hang indefinitely)

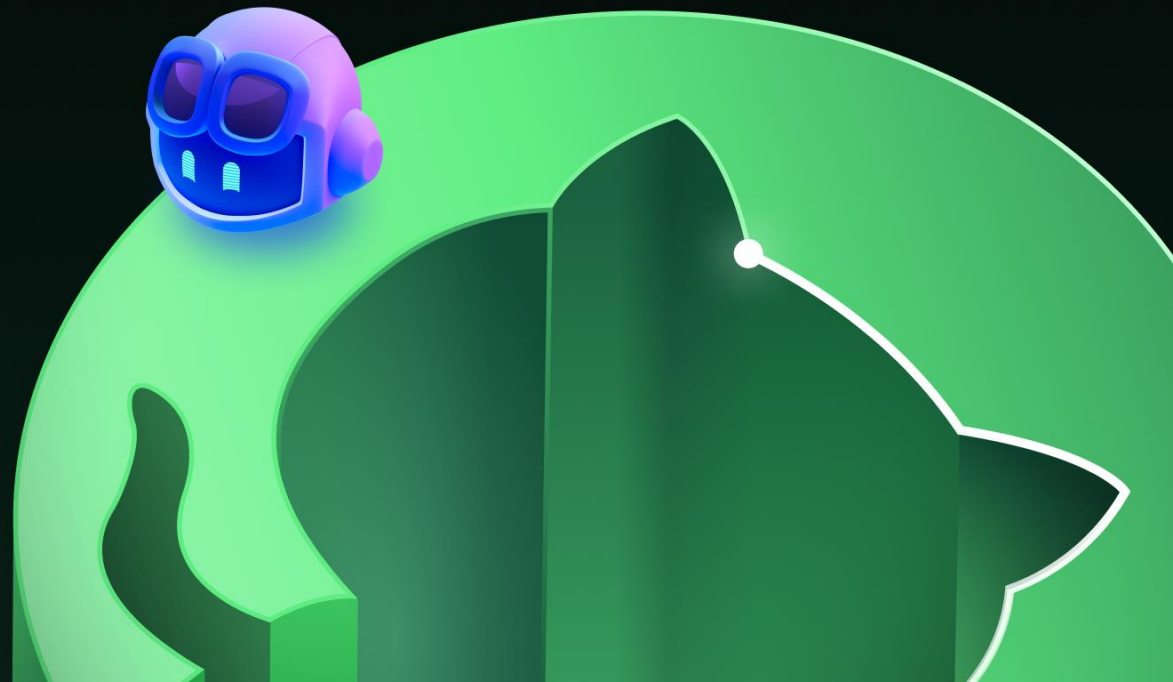
There's a whole slew of best practices – see my resources link at the end

Demo: MCP in Action

We'll explore a custom MCP server that:

1. Connects to a documentation system
2. Exposes docs as resources
3. Provides search functionality
4. Adds weather API lookup
5. Demonstrates MCP chaining with another MCP

Then see it work with Copilot in real-time! 🤖



Part 6: MCP Server Hosting, Registries, and Policies

Two transport patterns

Local vs Remote MCP Servers

Local:

- Runs on developer machine
- Process spawned by VS Code
  - You should be containerizing ☺
- Uses standard input/output
- Best for: Personal tools, fast iteration, debugging

Remote:

- Deployed and hosted like any other software
- Offloads computational load to remote server
- Centralized context and consistency
- Best for: Shared resources, enterprise data, proper logging, and telemetry gathering

Where to host your HTTP MCP servers

Remote MCP Server Hosting Options

Think like you're publishing an API – you can host it virtually anywhere!

- **Serverless Functions:**
  - Azure Functions, AWS Lambda, Google Cloud Functions
  - Best for: Lightweight servers, sporadic usage (cold starts can affect latency)
- **Container Services:**
  - Azure Container Apps, AWS ECS/Fargate, Google Cloud Run
  - Best for: Production workloads, consistent performance (easy scaling, good monitoring)
- **Kubernetes:**
  - Full orchestration control
  - Best for: Enterprise scale, high availability (complex setup, overkill for small teams)
- **Self-Hosted:**
  - Your own infrastructure
  - Best for: Sensitive data, compliance requirements (full control, more maintenance)

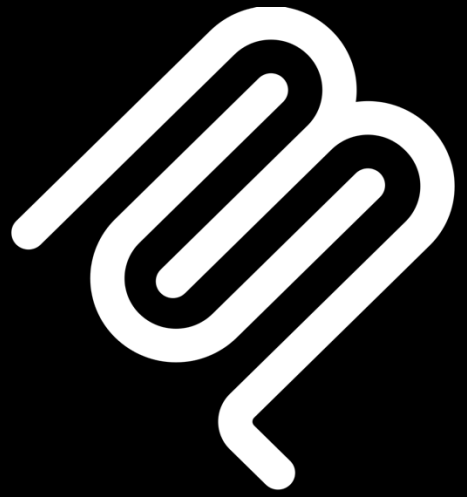
Production readiness checklist

Deployment/hosting Best Practices

If you build and deploy software, these should look familiar ☺

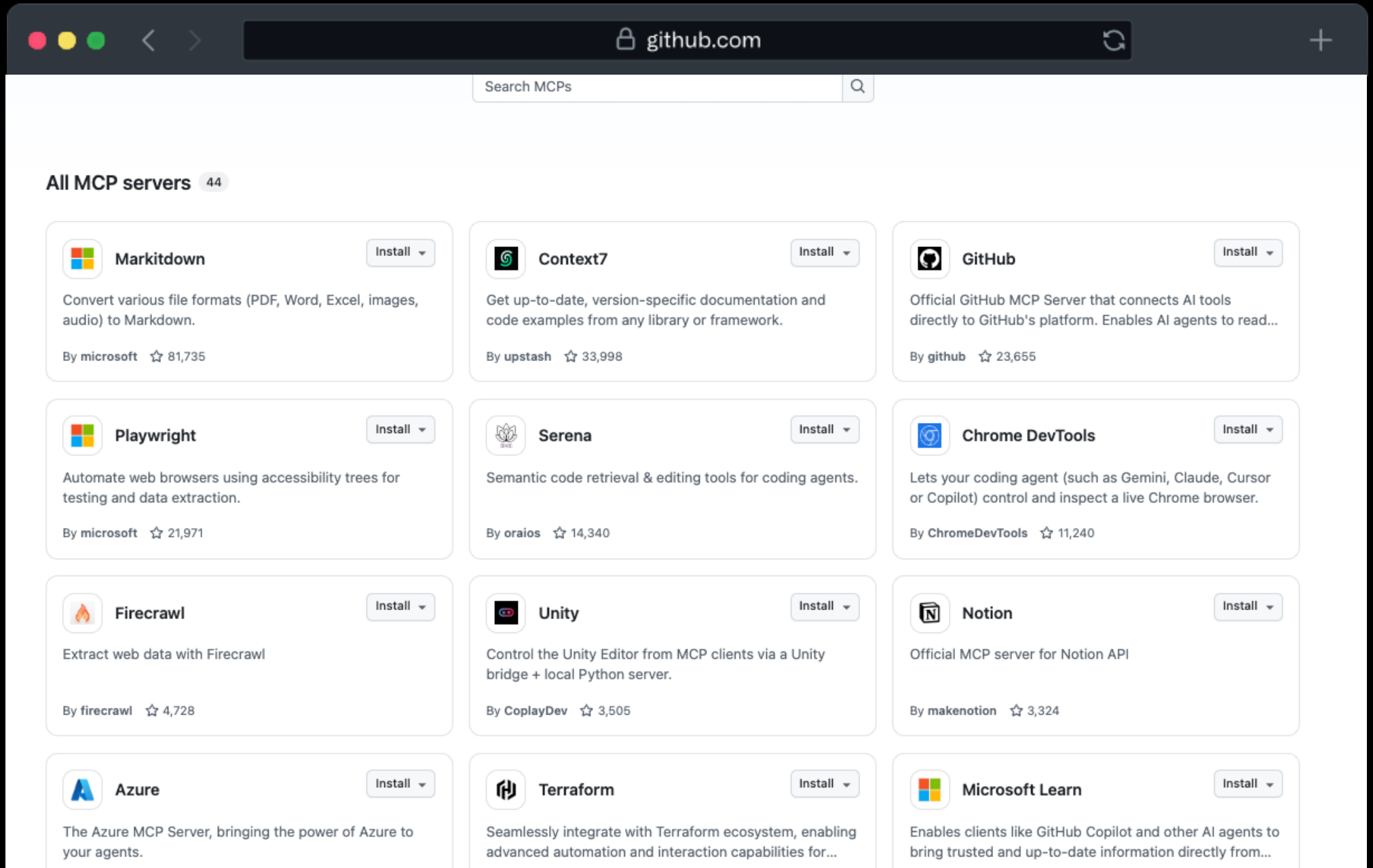
- **Security:**
  - Validate inputs, sanitize outputs, no exposed secrets
- **Performance & Reliability:**
  - Health checks + timeouts (30s max)
  - Rate limiting + caching for frequently accessed data
- **Observability:**
  - Log tool calls with context (what, when, who)
  - Monitor errors, set alerts, track usage patterns
- **Safe Deployments:**
  - **Validate with MCP Inspector first**
  - Rolling updates
  - Versioned

MCP Registries



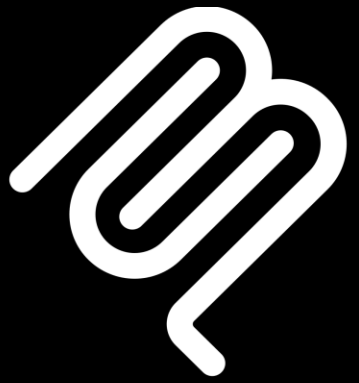
- **What is an MCP registry?**
  - Central directory of available MCP servers
  - Discover servers built by your organization or community
  - Simplify server distribution and update management
- **GitHub has a Public MCP registry:**
  - github.com/mcp
  - Curated list from leading partners and open-source community
  - Servers sorted by GitHub stars and community activity
- **Internal Registries**:
  - Create your own internal catalog for your organization
  - Add proprietary/internal-only servers
  - Control governance and compliance with organization/enterprise settings in GitHub

GitHub MCP Registry



"The MCP Registry"

From MCP: The MCP Registry



In partnership from MCP creators and other maintainers – including GitHub

- Public **catalog and API** for discovering Model Context Protocol (MCP) servers
- **Open source and extensible** – anyone can build compatible sub-registries
- Designed for **discoverability and standardization** across the MCP ecosystem
- Supports **public and private registries** with shared schema and governance
- Currently in **preview** – feedback and community contributions encouraged

https://github.com/modelcontextprotocol/registry

Specify a MCP registry for organization-wide control

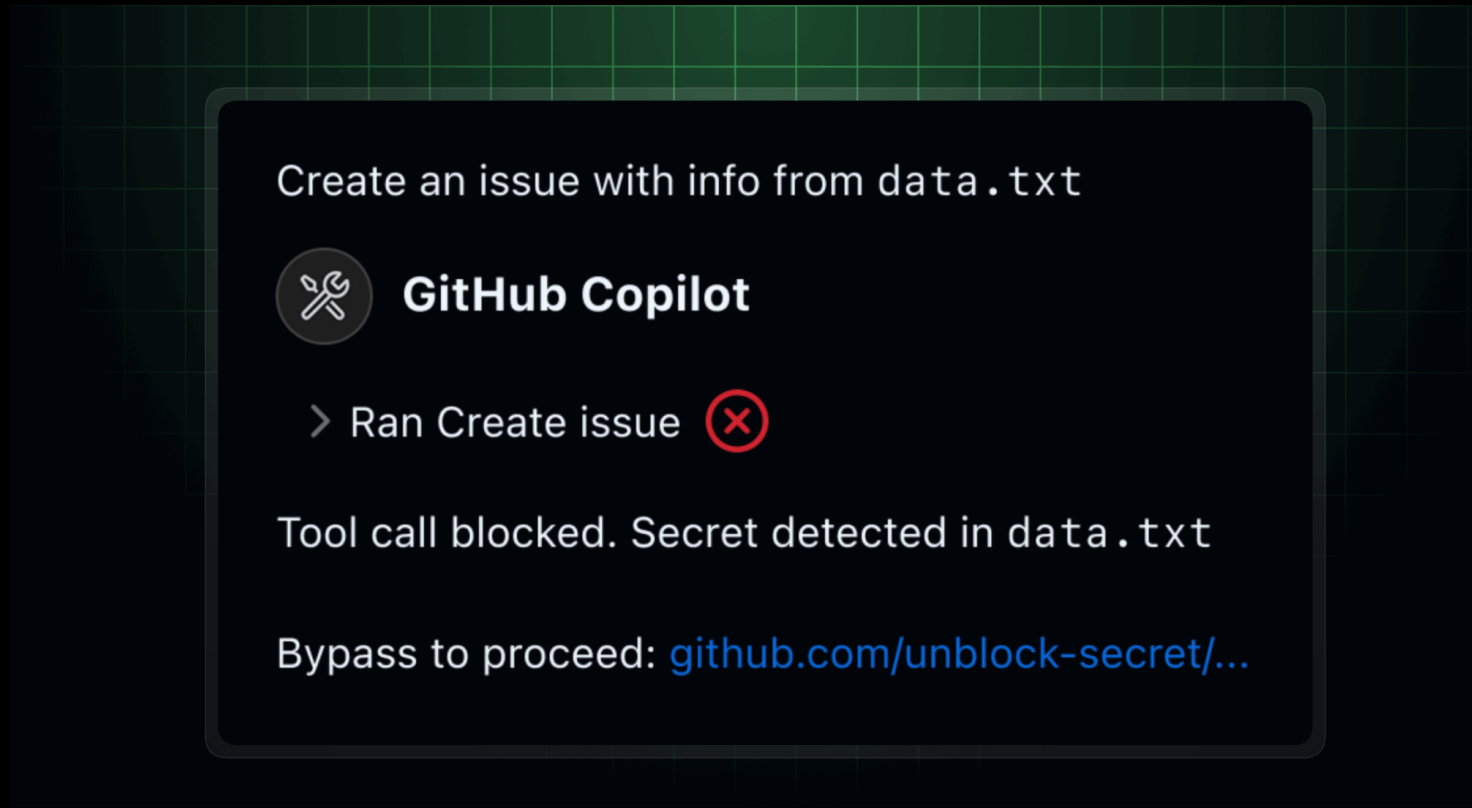
Internal MCP registry and allowlist Insiders

- Configuring MCP server access in an Enterprise/organization
  - Coming soon to other IDEs!
  - Safely curate a registry (hosted somewhere) of approved MCP servers for your users

This helps prevent accidentally LEAKING secrets to a malicious repo in GitHub

GitHub MCP Server: Secret scanning, push protection

- We scan all tool call inputs in public repositories
- If an exposed secret is detected, call is blocked by default
- This protects prompt injection – public instructions/readmes can trick agents into pasting credentials into MCP tool calls
- By inspecting data flowing to and from public repositories, this cuts out a common exfiltration path
- Support for private repositories with a GitHub Secret Protection license is coming soon



What takeaways do we have?

Recap

- Use **custom instructions**!
  - Recommend repository-level custom instructions – path-specific instructions good too!
- **Tools**! Some extensions have tools (like the GitHub Pull Request extension)
- **MCPs** – you now know what they are!
  - Universal protocol for AI assistants
- Install the **GitHub Remote MCP server**
- Explore the **GitHub MCP registry**
  - The Playwright MCP server is AWESOME for navigating websites interactively and using AI to analyze and query against
  - Plug for the Azure MCP server as well
- More **MCP Server policies** are coming soon for administrators

Resources / Slides

gh.io/josh-copilot-talk

Survey

SpreaView

Review my Session

Slides / resources

Techorama

Thank you

SpreaView

Review my Session