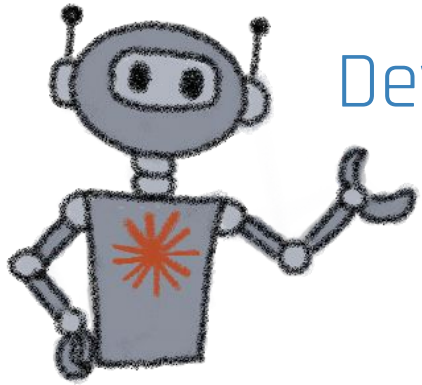
DevDays

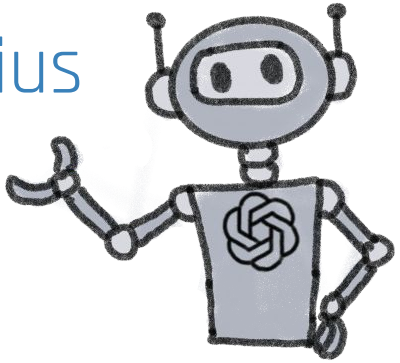
Hands-on MCP Servers Beyond 101:
Good Practices, Design Choices and Consequences

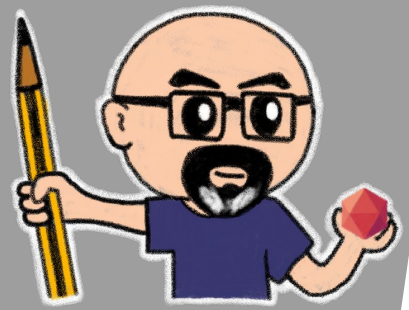


DevDays Europe 2026, Vilnius

Horacio González

2026-05-19

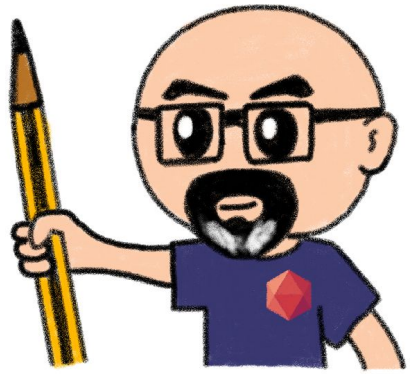


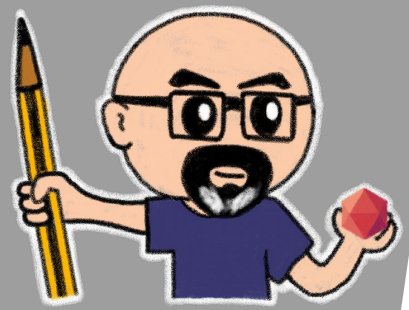


DevDays

Who are we?

Introducing myself and introducing Clever Cloud



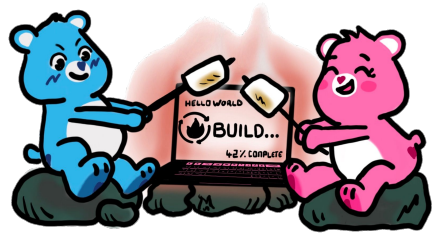


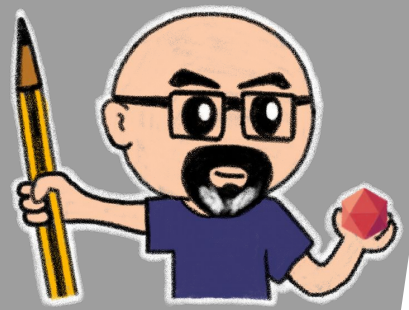
DevDays

Horacio Gonzalez - @LostInBrittany

Spaniard Lost in Brittany

Finist Devs





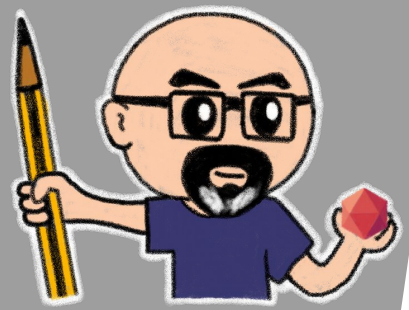
Head of DevRel clever cloud

DevDays

Clever Cloud

Sovereign European Cloud Provider

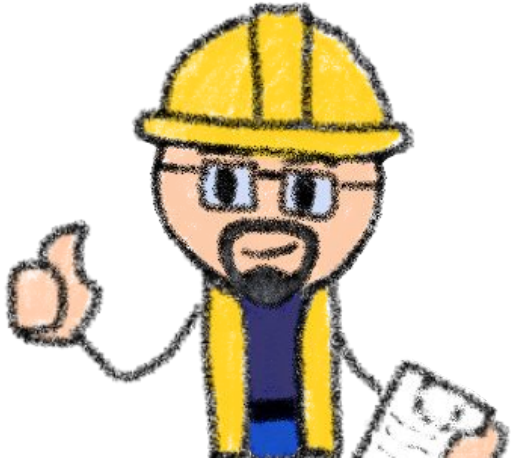
clever cloud

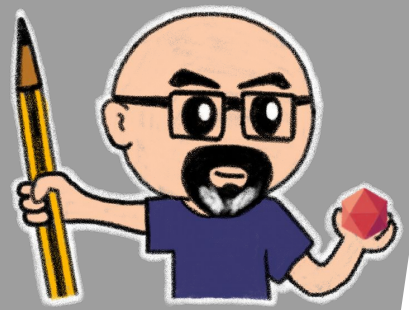


DevDays

Before we start

How the day works…
and the software you'll need





DevDays

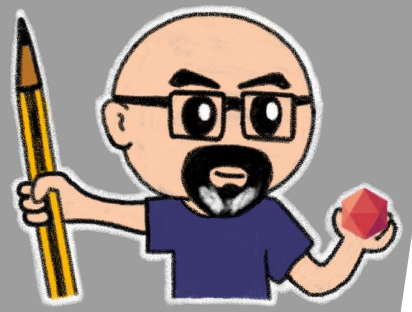
What you need

- **Bun 1.1+:** runtime + package manager
- **An agentic assistant:** Claude Code preferred, Antigravity / Codex / Copilot also fine
- **Git** and a **terminal**
- A laptop you can `bun install` on

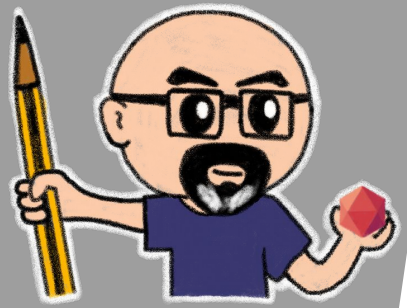
Don't have Bun yet?

```
curl -fsSL https://bun.sh/install | bash
```

Run it **now**, we'll be on slides for 15 more minutes



The repository



https://github.com/LostInBrittany/devdays-2026-hands-on-mcp-beyond-101.git

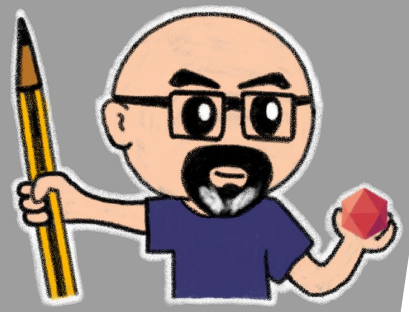
DevDays

The rhythm for each module

1. **I present**
   Slides land the concept, the design choice, the traps
2. **You code**
   Pair up, break things, ask questions
3. `solutions/`
   Drop-in reference if you fall behind or want to compare

Most of today is your terminal, not my slides.

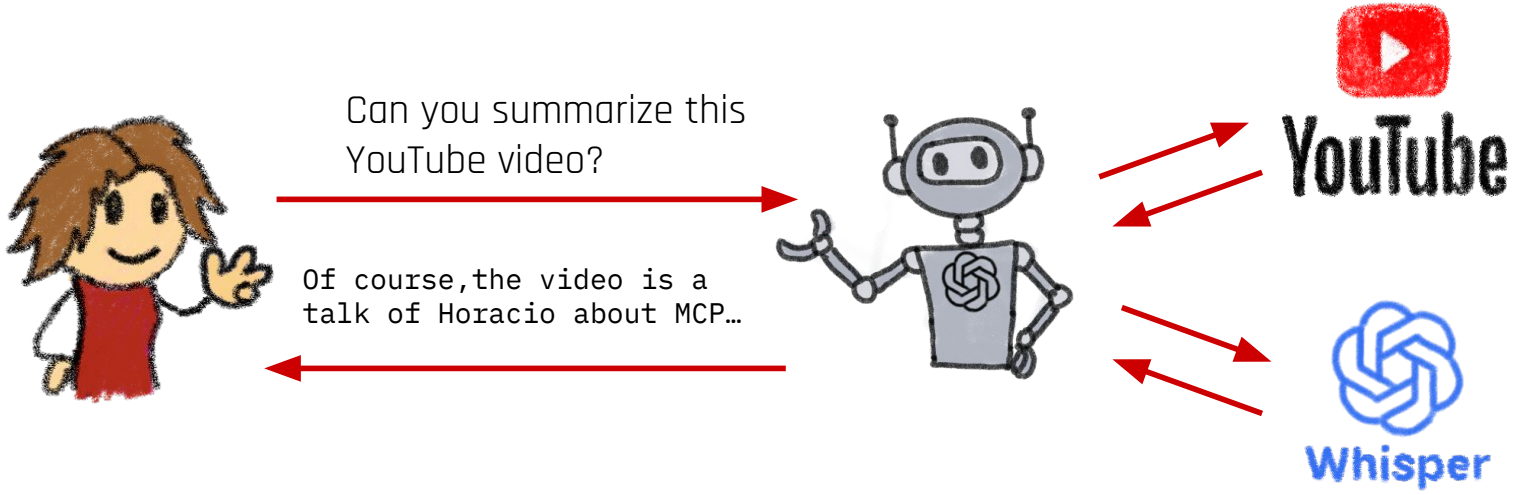
Raise your hand if you're stuck,
help your neighbour if you finish early

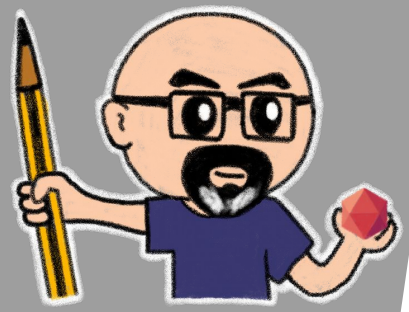


DevDays

The Agentic Revolution

From helpers to actors:
How AI learned to do, not just say



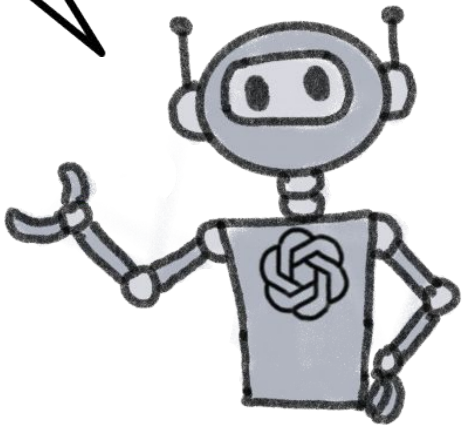


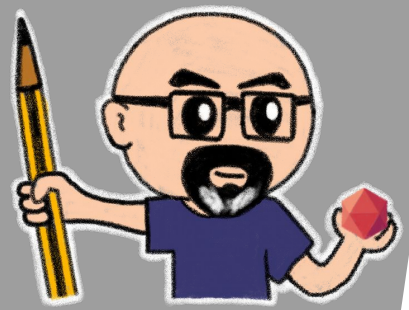
DevDays

From Chatbots to Agents

Tell me what to visit in Madrid

Madrid is vibrant, elegant, and full of art, history, and food. Here are the most important things to visit in Madrid, perfect for a first trip 🇪🇸…

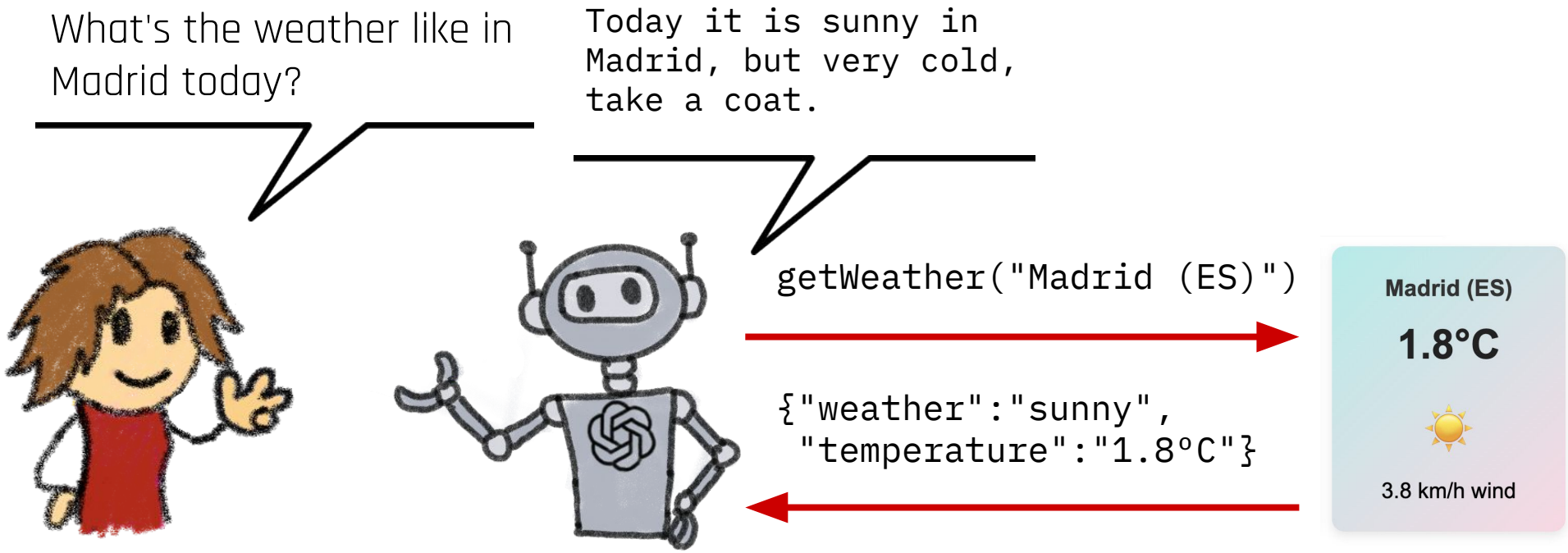


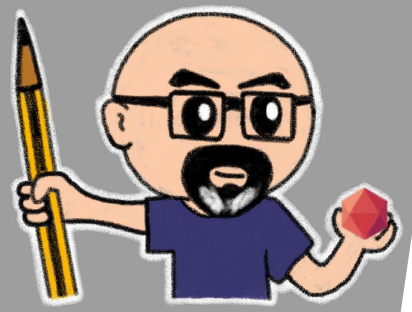


2023: ChatGPT - Smart conversation

DevDays

From Chatbots to Agents

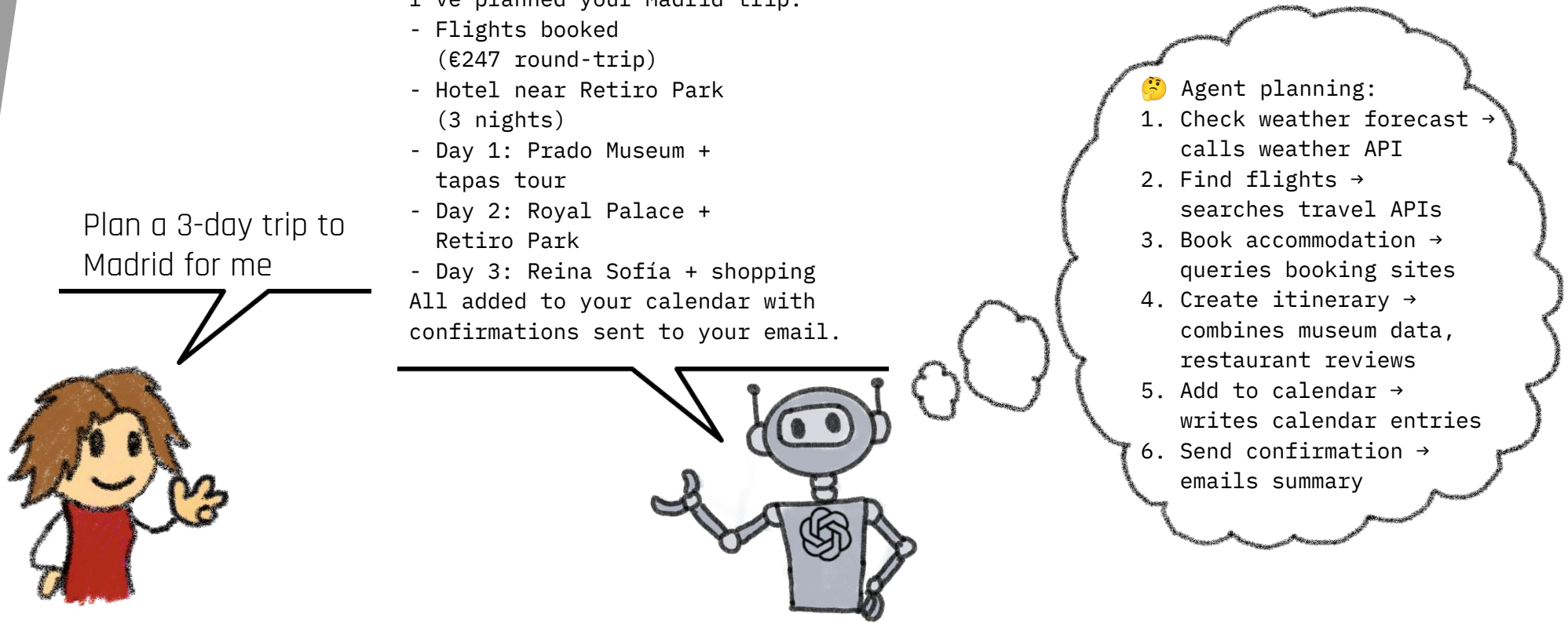


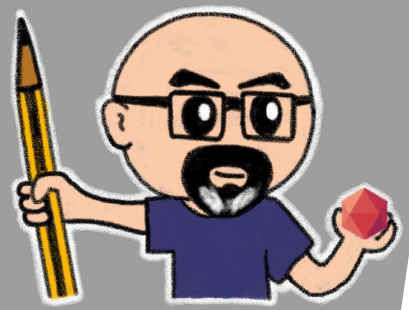


2024: Tool use - LLMs can do things

DevDays

From Chatbots to Agents



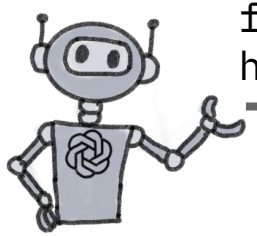


2025: Autonomous agents - LLMs that plan and execute

DevDays

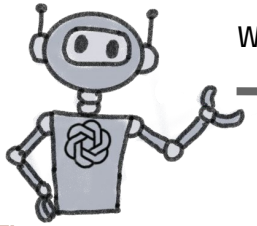
From Chatbots to Agents

I'm thinking about taking the kids to Madrid this summer...



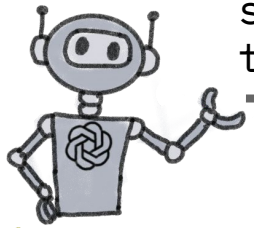
Scans inbox, finds school holiday dates

📧 Email agent



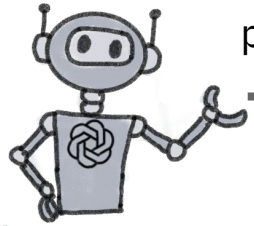
Blocks optimal week in July

📅 Calendar agent



Checks budget, sets aside travel funds

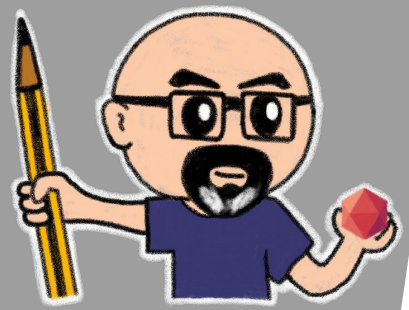
💰 Finance agent



Creates family packing list

🎒 Packing agent

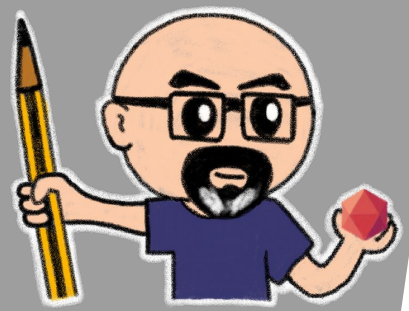
2026: Agents are everywhere

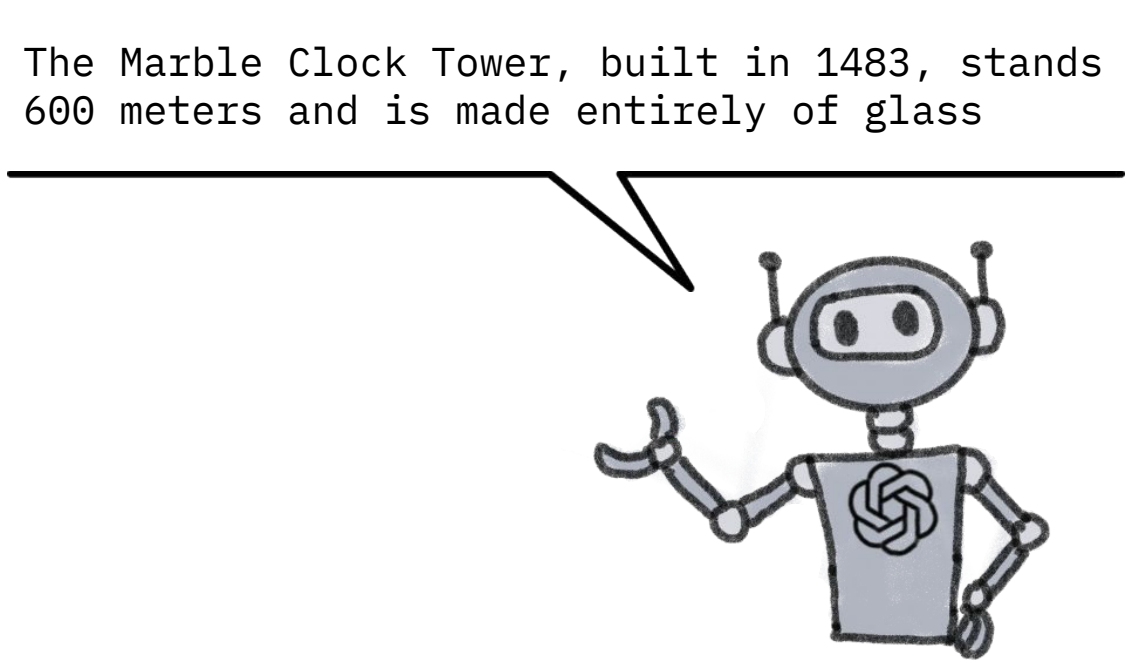


DevDays

When Chatbots Hallucinate

- Read the output
- Laugh, correct it
- No real damage

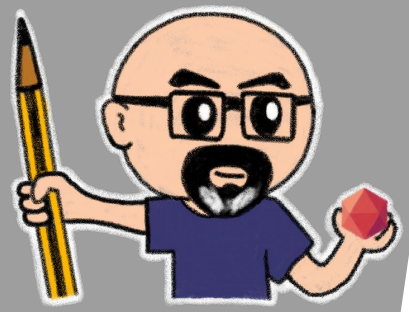


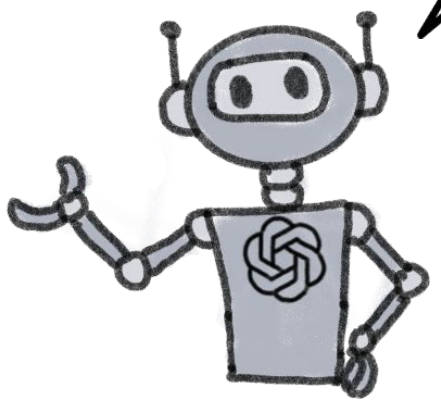


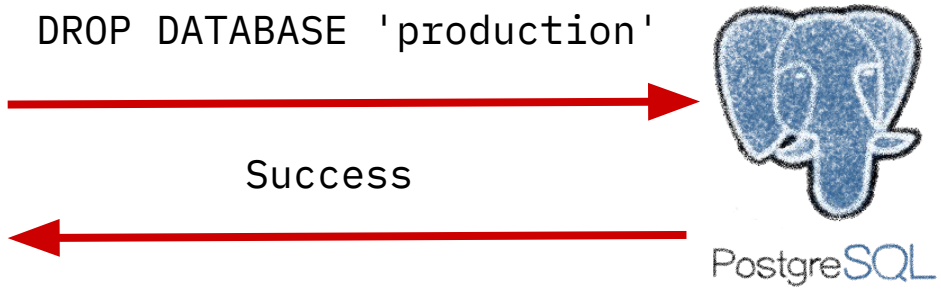
DevDays

When Agents Hallucinate

- Execute the wrong API call
- Delete a database
- Expose secrets
- **You don't know until something breaks**



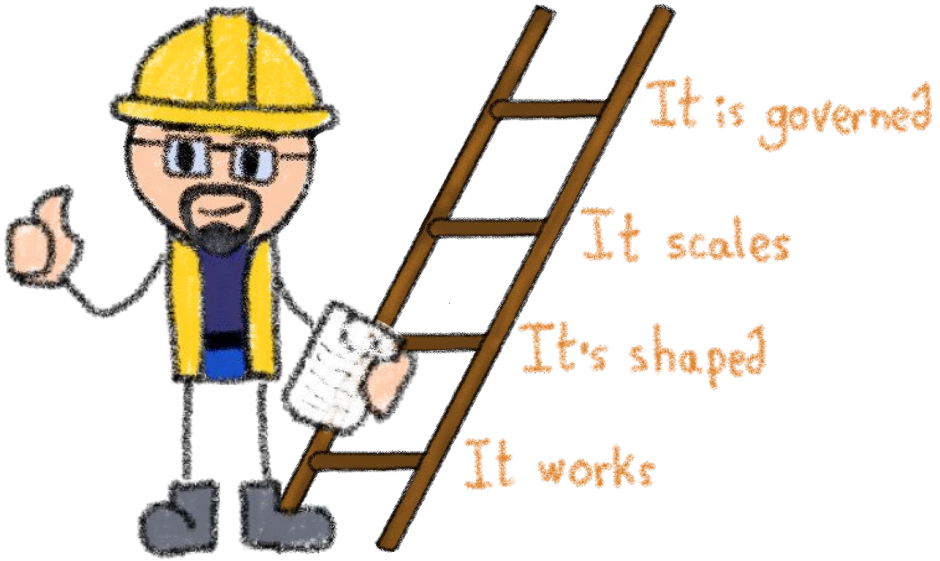


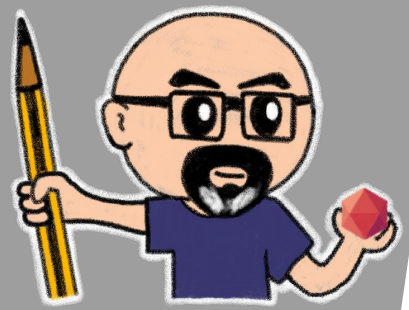


DevDays

Why I proposed this workshop

The uncomfortable thing about MCP in 2026





DevDays

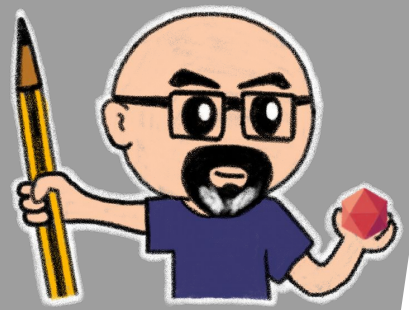
Anyone can build an MCP server

The SDK lets you ship one in ~30 lines of TypeScript.

You'll do exactly that in 20 minutes.

The docs are good and the starter examples work.

Getting to "it runs" is easy



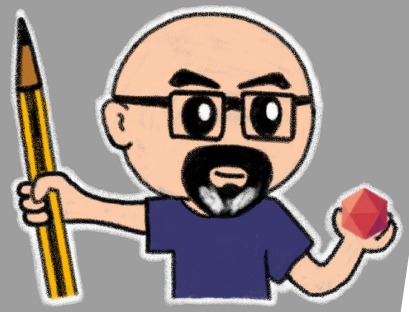
DevDays

The thesis

A quick-and-dirty MCP server is more dangerous
than a quick-and-dirty API

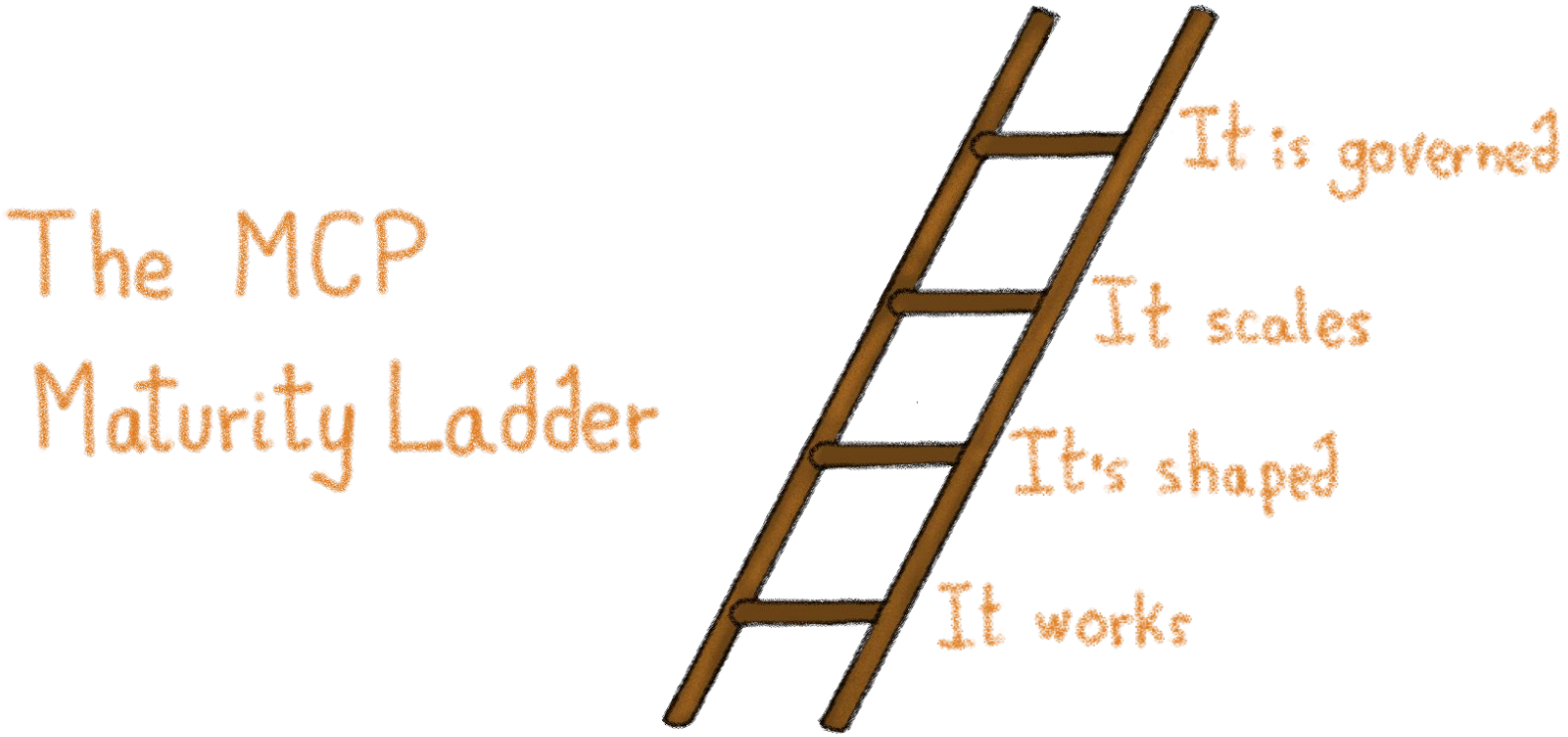
A quick-and-dirty API has a human
deciding whether to call it

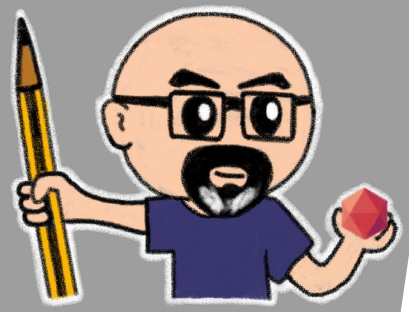
**A quick-and-dirty MCP has a non-deterministic LLM
that just did**



DevDays

Simply "it works" isn't enough anymore





When the caller is a non-deterministic language model
You need to go an extra step... or to climb an extra rung

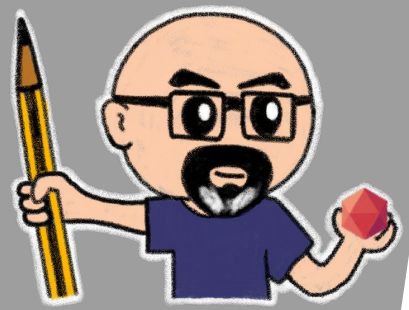
DevDays

The Four Rungs of the Maturity Ladder

A framework for API design discipline in MCP

- **v1** - MCP works
- **v2** - MCP is shaped
- **v3** - MCP scales
- **v4** - MCP is governed

Climbing the ladder = getting better at MCP design

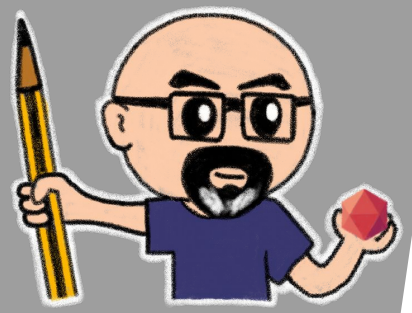


DevDays

What is MCP?

Quick reset, because some of you are new

You're in the right room

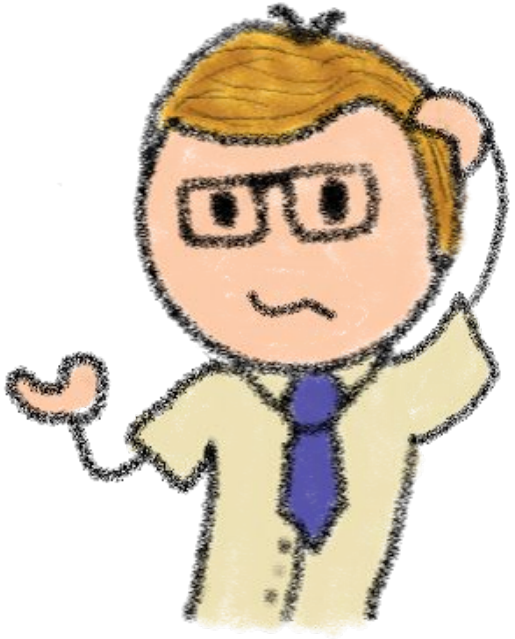


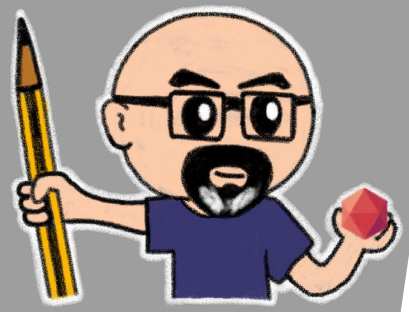
DevDays

The Connectivity Problem

What Agents Need to Function

- 📁 Read your files and codebases
- 🗄️ Query your databases
- 🔌 Call your APIs and services
- 🧠 Understand your domain and context
- 🔐 Access private systems securely



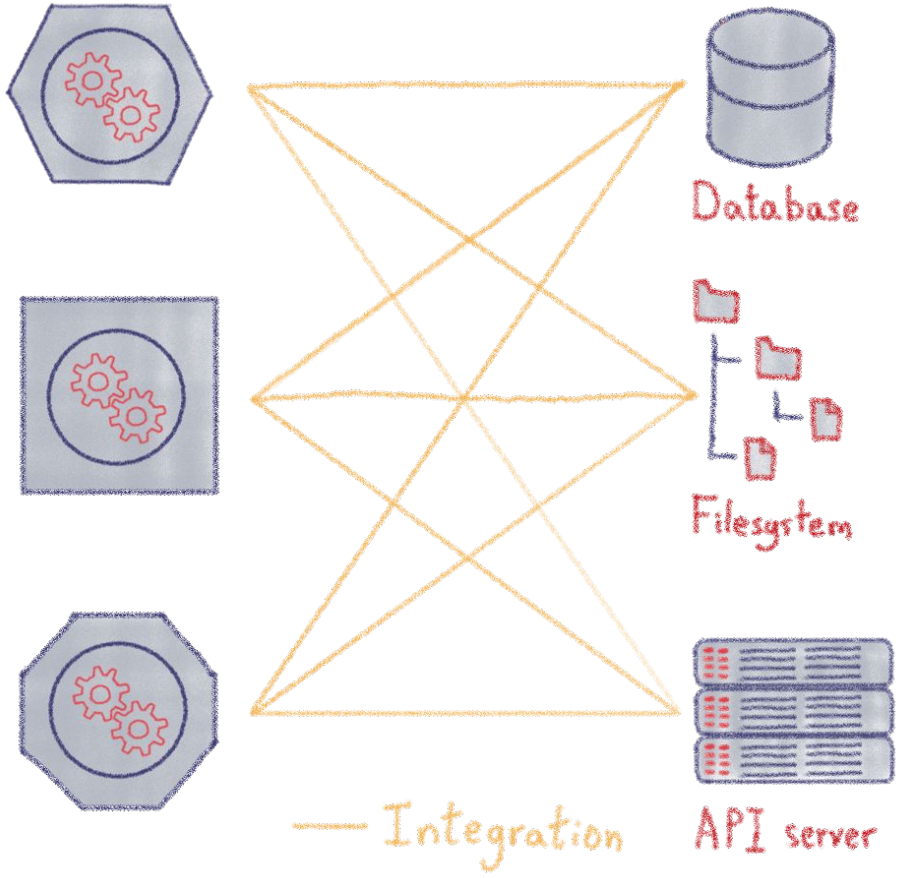


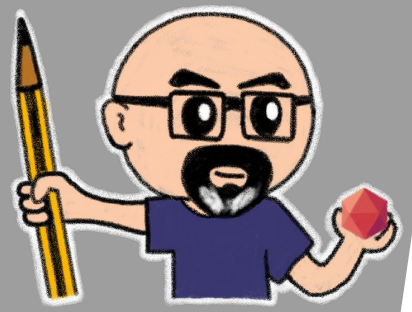
DevDays

The Connectivity Problem

Custom solutions for each integration

- **OpenAI:** Function calling with custom schemas
- **Anthropic:** Tool use with JSON descriptions
- **Google:** Function declarations

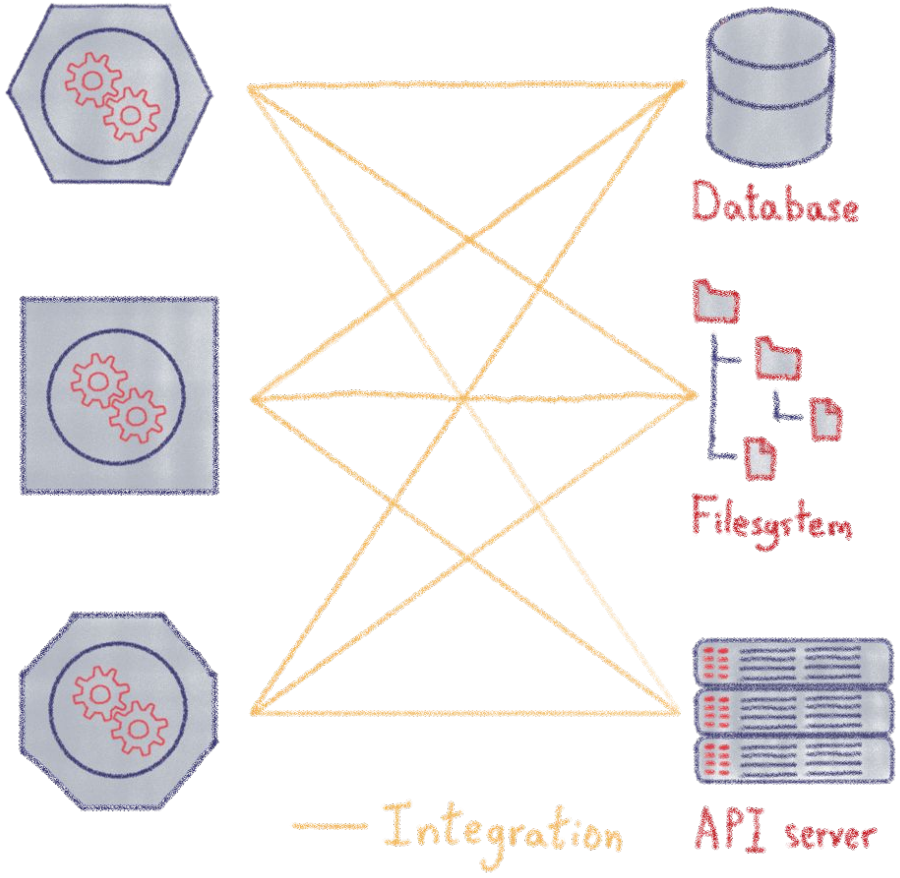


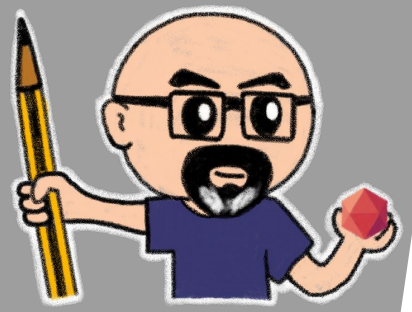


DevDays

Why Do We Need MCP?

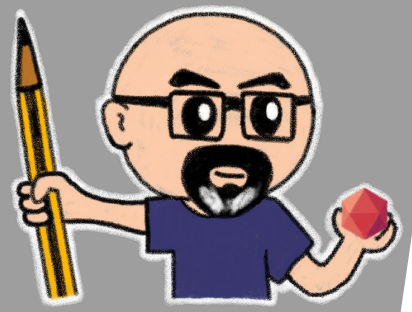
- LLMs **don’t automatically know** what functions exist.
- **No standard way** to expose an application's capabilities.
- **Hard to control** security and execution flow.
- Expensive and fragile **integration spaghetti**





Model Context Protocol

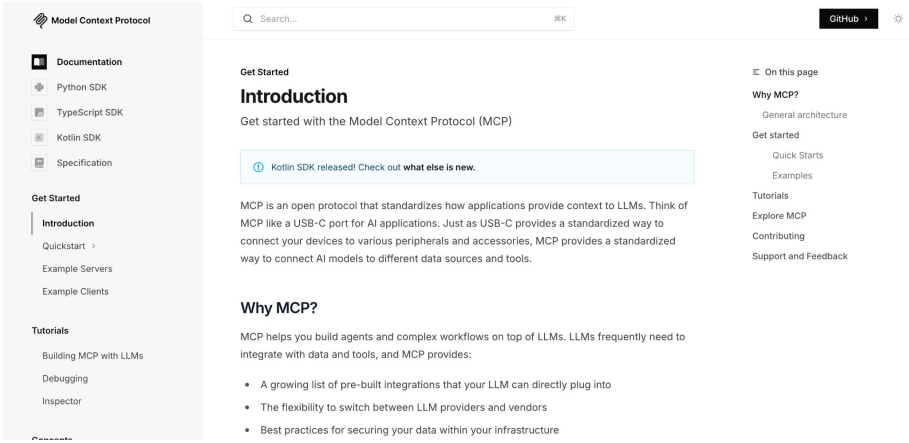
Anthropic, November 2024:
LLMs intelligence isn't the bottleneck, connectivity is

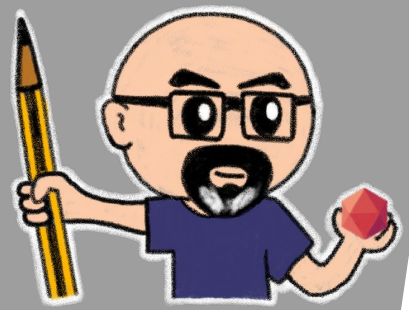


DevDays

Model Context Protocol

- **Standardize the protocol**, not the tools
- **Abstraction layer** between agents and capabilities
- Works for **any LLM**, **any tool**
- Open specification, open ecosystem

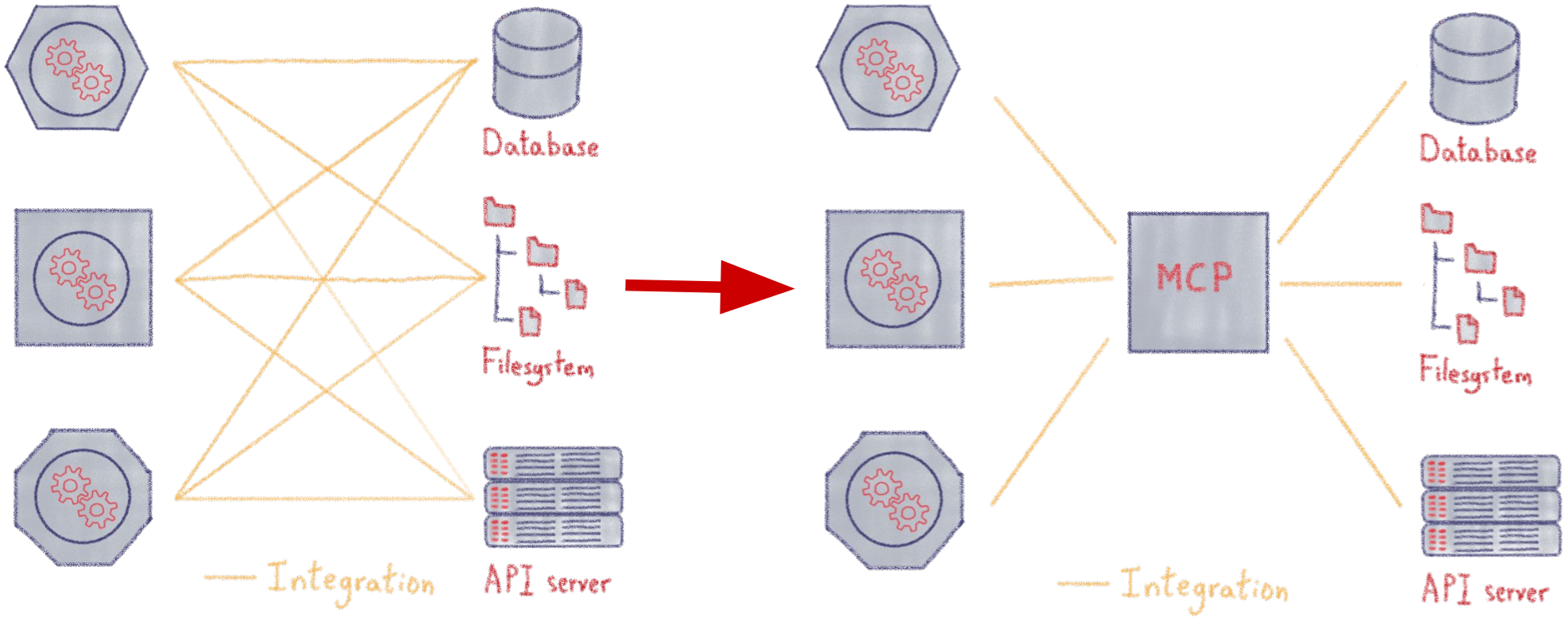


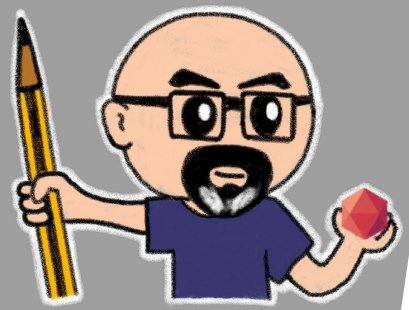


https://modelcontextprotocol.io/

DevDays

MCP solves integration spaghetti

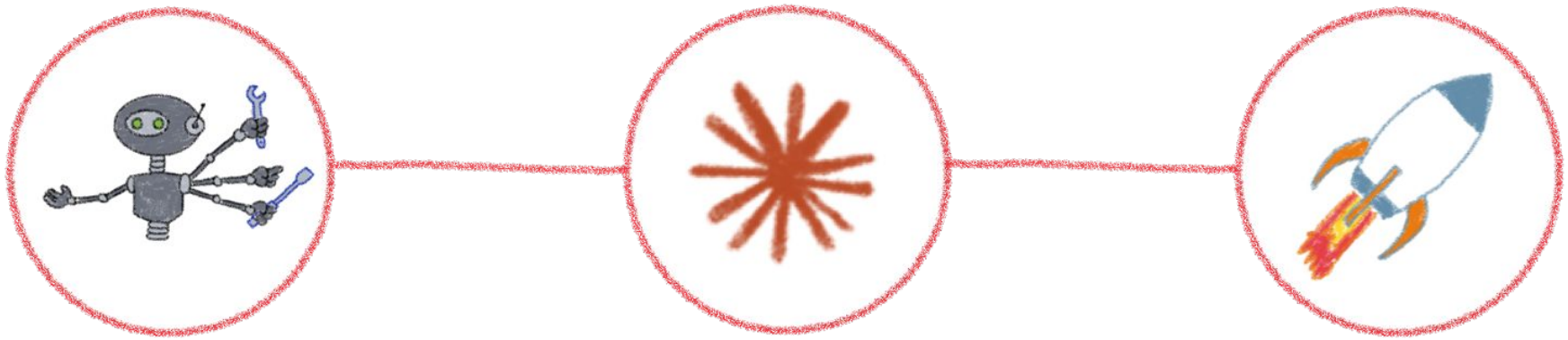


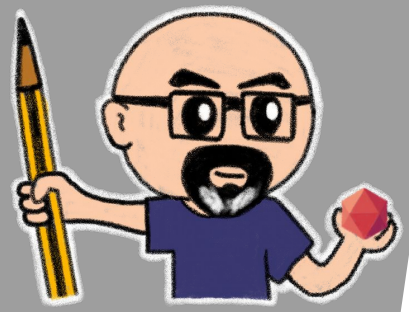


DevDays

Year One: Chaos

- 2025: explosive adoption
- Thousands of servers, fragmented quality
- Security issues surfacing fast
- "It works" is not the same as "it's ready"

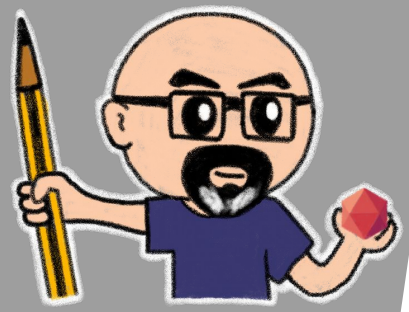




The REST analogy

We've seen this movie before.

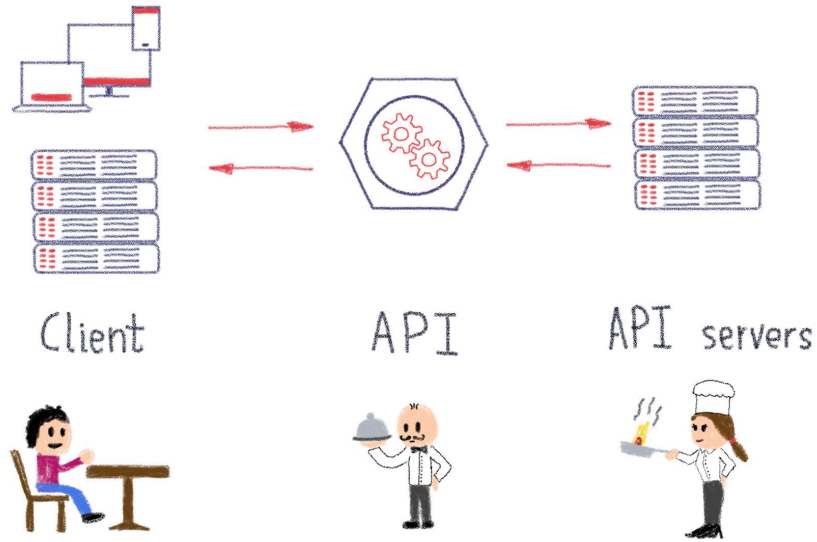
MCP in 2026 is where REST was in 2008



DevDays

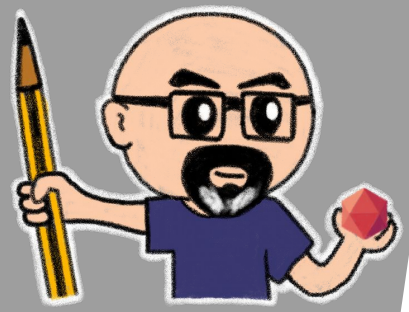
Last Time, 10 Years to Learn

- REST 2005–2008:
  everyone adopted it
- REST 2008–2015:
  everyone made the same mistakes
- REST 2015+:
  we finally learned the patterns



MCP doesn't have 10 years

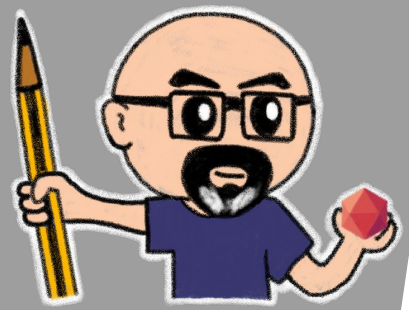
That's what this talk is about



DevDays

Anatomy of MCP

What the protocol actually looks like in 2026

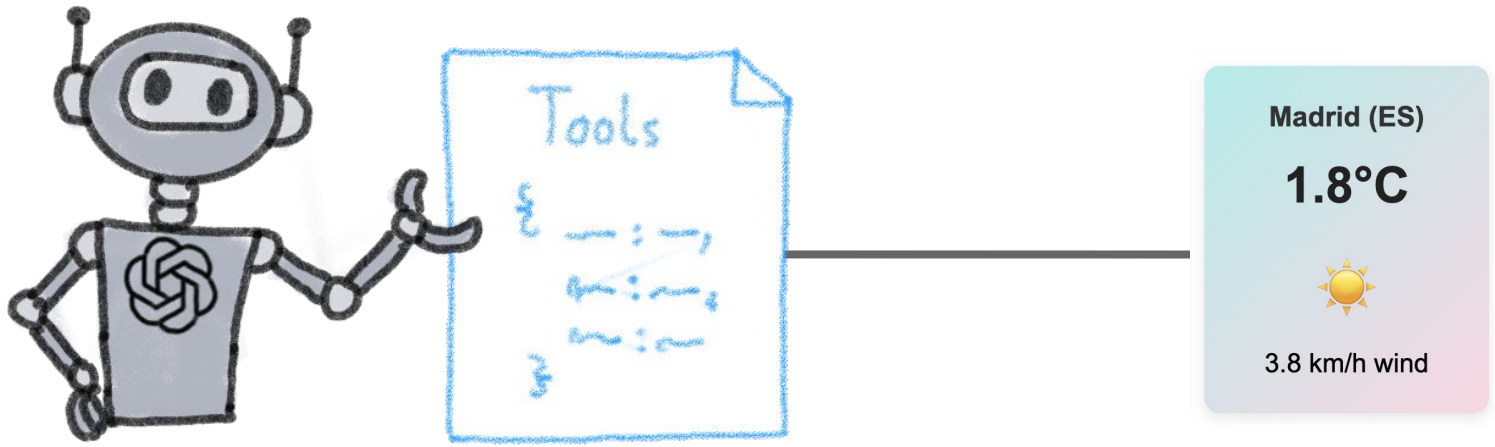


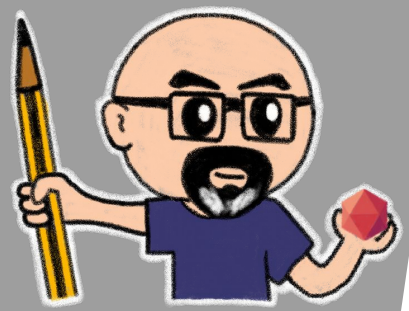
DevDays

How MCP works

- MCP servers expose primitives (structured JSON).
  - Function (tools), data (resources), instructions (prompts)

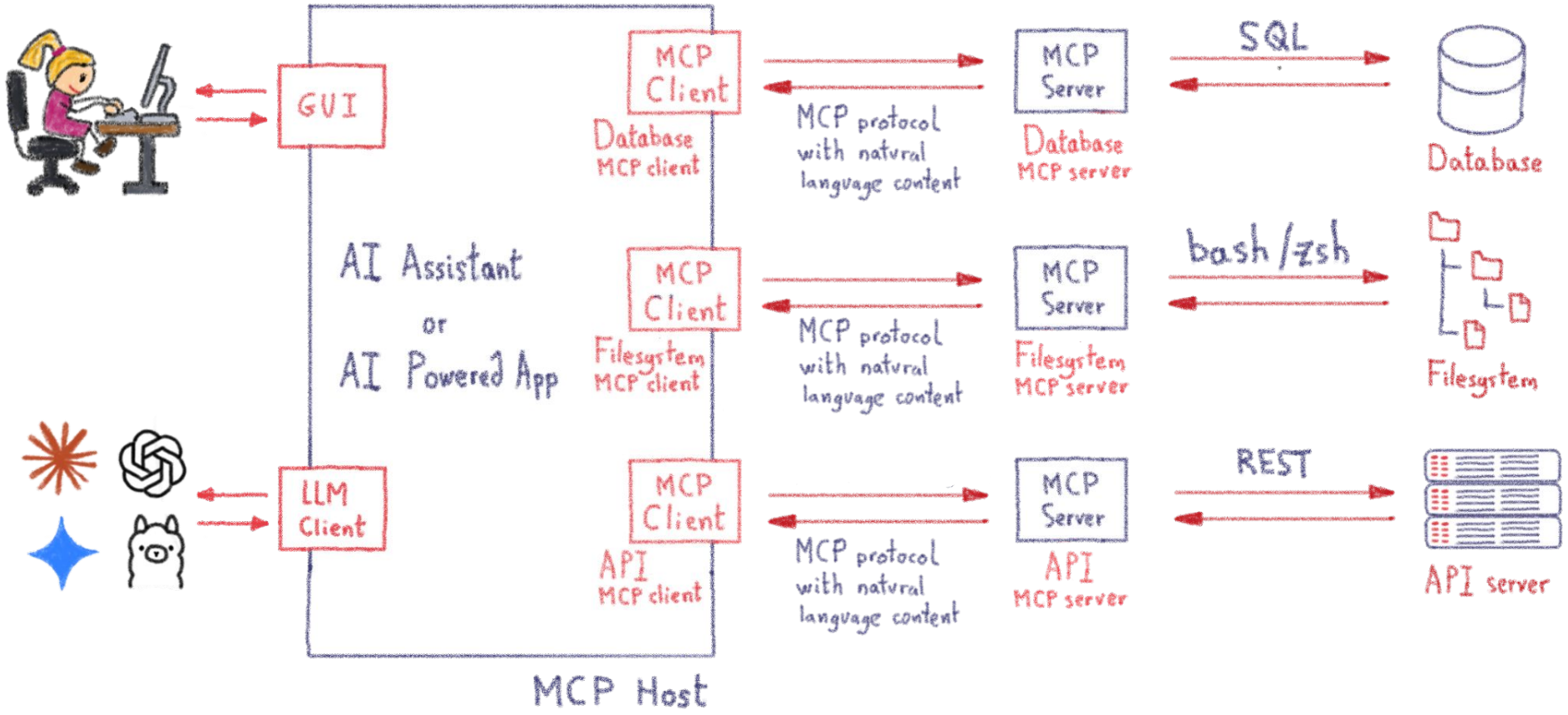
- LLMs can discover and request function execution safely.

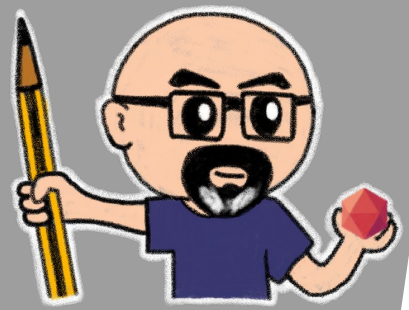




DevDays

MCP Clients: on the AI assistant or app side





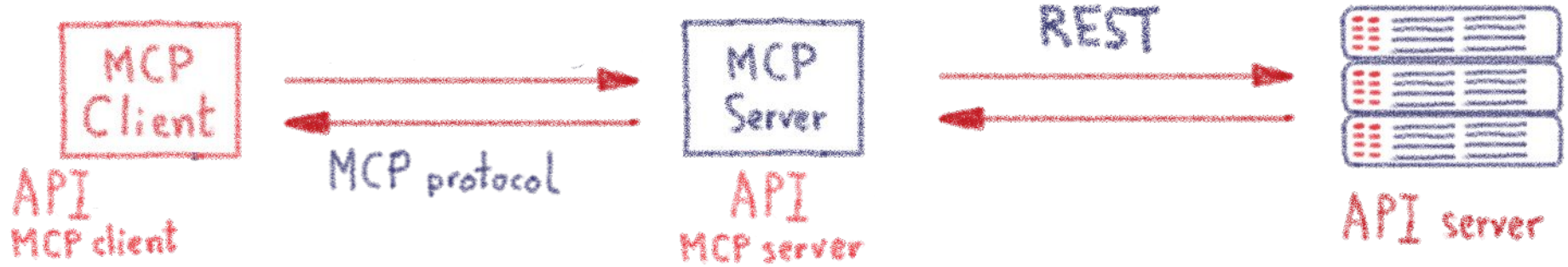
One MCP client per MCP Server

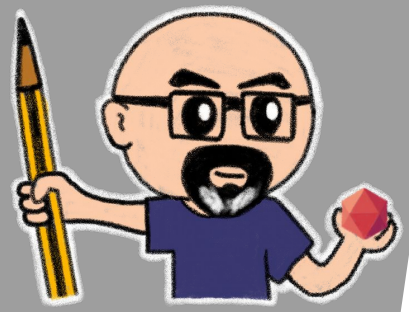
DevDays

Separation of Concerns

- **Host:** the agent runtime (Claude Code, Cursor...)
- **Client:** the MCP protocol layer
- **Server:** your tools, data, and capabilities

Each layer has a single job





DevDays

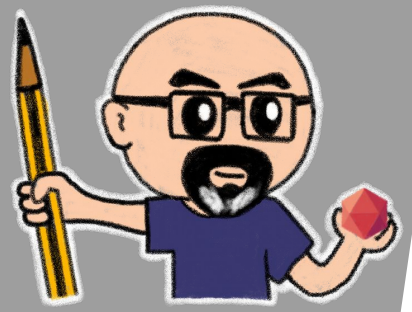
Transports

stdio (local)

- Server runs as a child process
- Communication over stdin/stdout
- Great for dev tools
- The original MCP transport

Streamable HTTP (remote)

- Server runs as a network service
- HTTP POST + server-sent events
- Production-ready, cloud-hosted
- The direction of travel

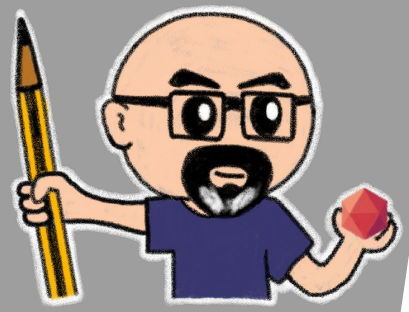


DevDays

SSE is Legacy

Server-Sent Events transport is deprecated

- Migration path: streamable HTTP
- Ecosystem is actively moving
- If you have SSE servers, plan the migration



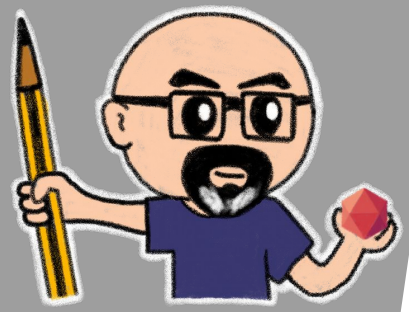
DevDays

Current MCP specification

- Specification version: 2025-11-25
- Next revision tentatively: June 2026
- SEP*s being finalized in Q1 2026

*Specification Enhancement Proposal

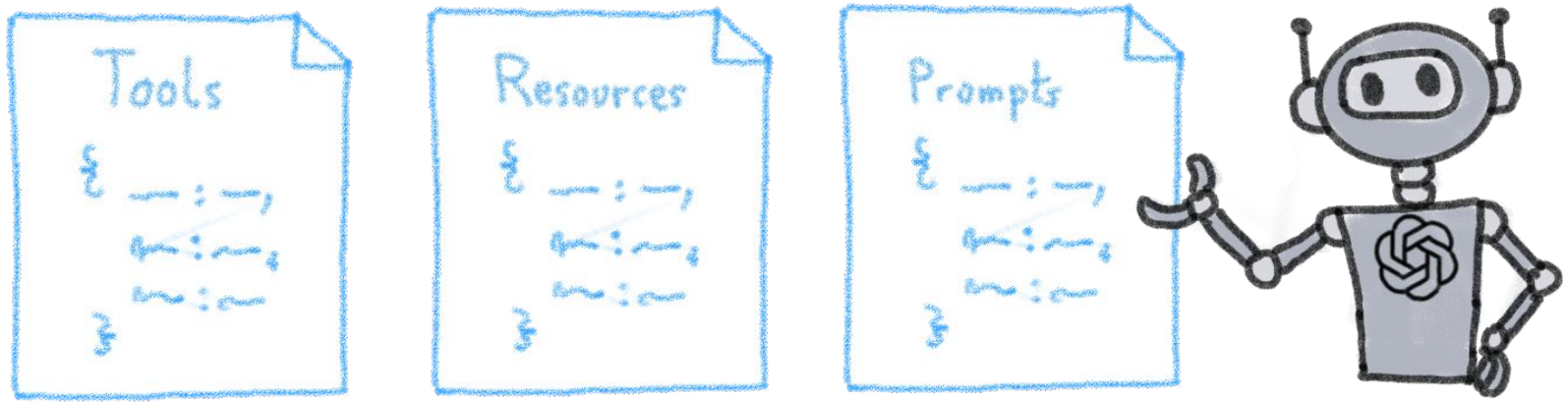
The ground is still moving

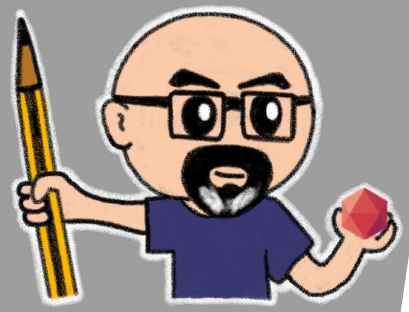


DevDays

Three Primitives Exposed by MCP Servers

- **Tools:** Actions LLM can invoke
- **Resources:** Data LLMs can read
- **Prompts:** Workflows LLMs can follow





DevDays

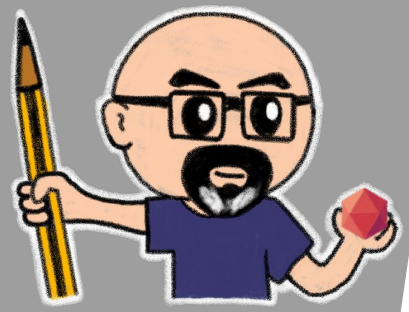
Tools

Actions that modify state or retrieve dynamic data

- Typed, validated operations
- The LLM calls them by name
- The most-used primitive by far

- **Examples:** `get_weather_in_city`, `query_database`, `send_email`

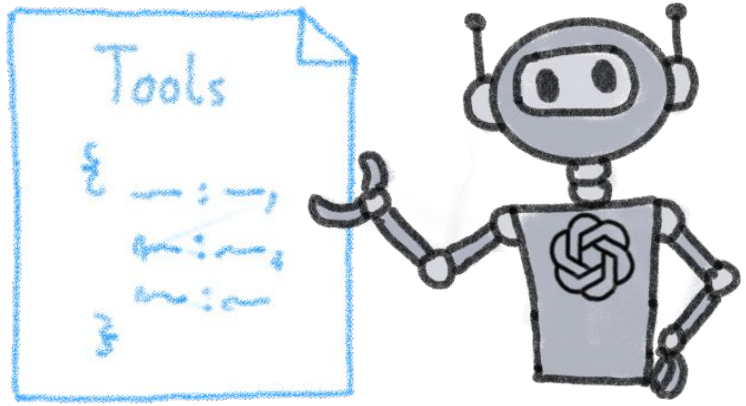
- **When to use:** When the LLM needs to do something

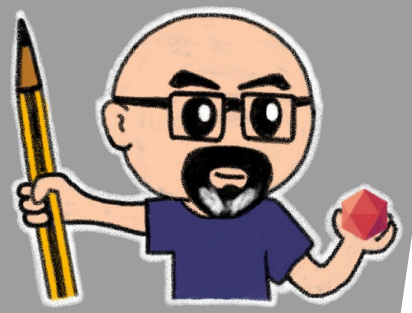


DevDays

Tools: Why You Should Care

- **Discoverable:** the LLM sees what's available
- **Typed:** parameters have schemas
- **Validated:** bad input rejected before execution
- The model cannot invent operations

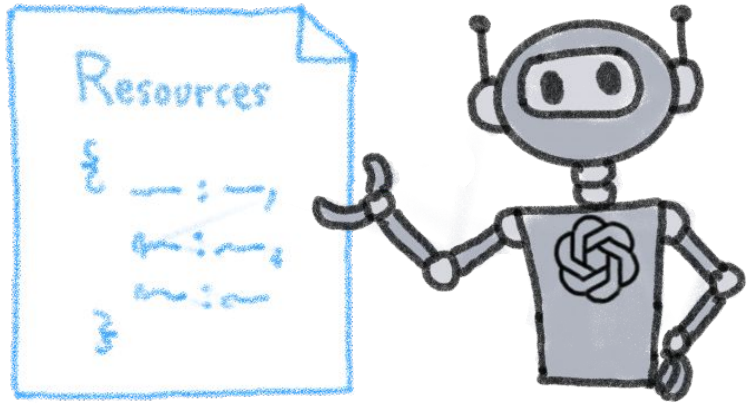


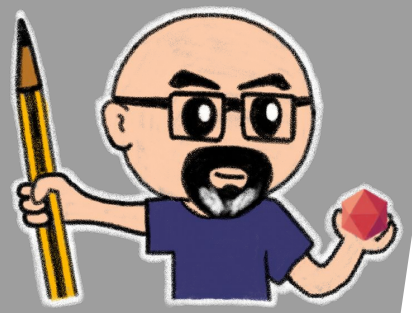


DevDays

Resources

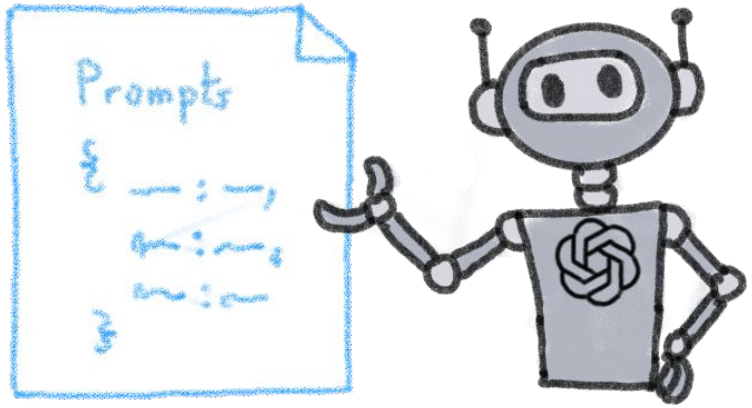
- Static or semi-static data LLMs can read
- Structured, read-only data, available before any tool call
- **Examples:**
  - `resource://weather/supported-cities`
    List of cities for which weather forecast is available
- **When to use:** When LLMs need reference data or context

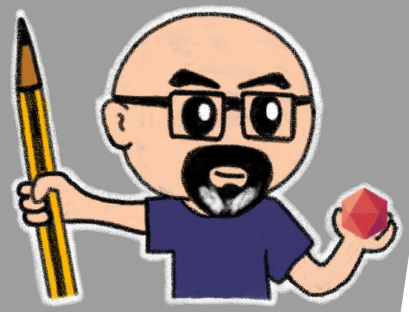




Prompts

- Pre-built workflows or templates to guide the LLM
- Examples:
  - `prompt://plan_outdoor_activity`
    Given a city and an activity, check the forecast and suggest the best time slot
- **When to use:** When you want to guide LLM reasoning for specific tasks

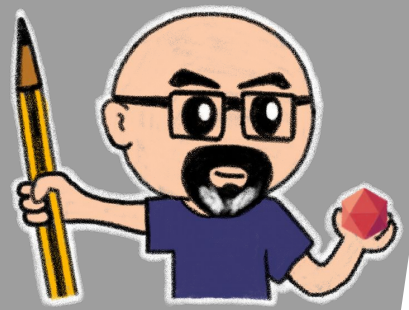




DevDays

Let's build an MCP server

A 30-line Weather Tool, your first MCP server



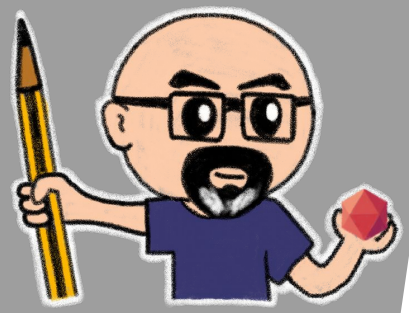
DevDays

Setup

```
$ git clone
https://github.com/LostInBrittany/devdays-2026-hands-on-mcp-beyond-101.git

$ cd devdays-2026-mcp-beyond-101-workshop/step-01-opening

$ bun install
```

Notice: bun install reads `bunfig.toml` and refuses any dep younger than 10 days.

We don't trust "the latest" anymore.
After **Shai-Hulud**, that's a feature.

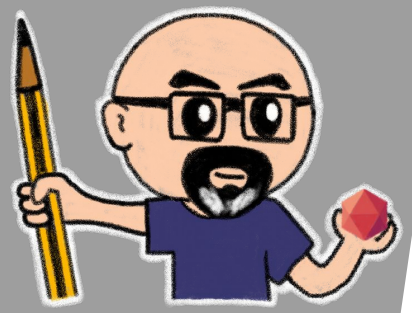


DevDays

The Weather Tool

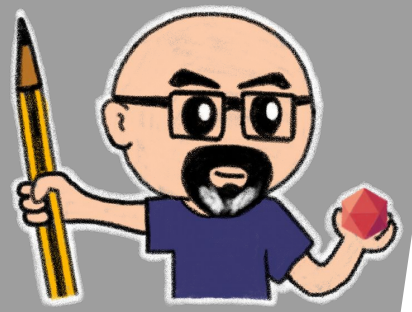
The weather tool

```
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";
import { z } from "zod";

const server = new McpServer({ name: "weather-server", version: "0.1.0" });

server.registerTool(
  "get_weather",
  {
    title: "Get Weather",
    description: "Get the current weather for a city.",
    inputSchema: { city: z.string().describe("City name, e.g. Vilnius") },
  },
  async ({ city }) => ({
    content: [{ type: "text", text: `Weather in ${city}: sunny, 22°C.` }],
  }),
);

await server.connect(new StdioServerTransport());
  console.error("[weather-server] listening on stdio");
```



Run it!

```
$ bun run dev
[weather-server] listening on stdio
```



Bun ran the TypeScript directly
No tsx, no ts-node, no transpile step.

DevDays

Connect Claude Code

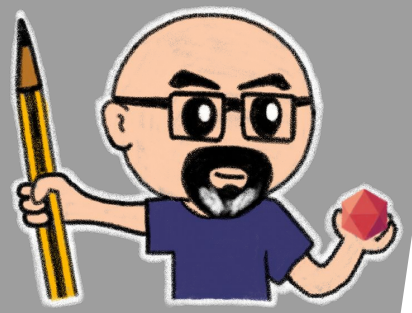
.mcp.json

```
{
  "mcpServers": {
    "weather-server": {
      "command": "bun",
      "args": ["run", "src/weather-server.ts"],
      "env": {}
    }
  }
}
```

From inside `step-01-opening/`, launch `claude`

Approve the "Allow this project's MCP server" prompt once, verify with `/mcp`

Scoped to this folder, global Claude Code is untouched



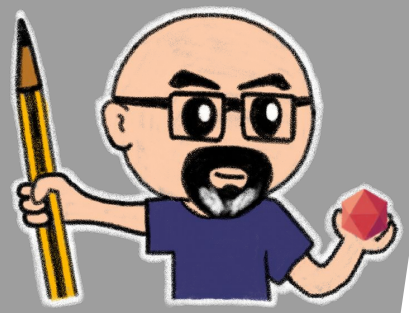
Try a prompt

"What's the weather in Vilnius?"

Your assistant should:

1. See the `get_weather tool`
2. Call it with city: "Vilnius"
3. Receive "Weather in Vilnius: sunny, 22°C."
4. Tell you the weather

If that worked: you just shipped a working MCP server.

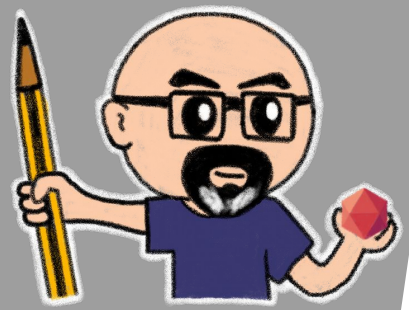


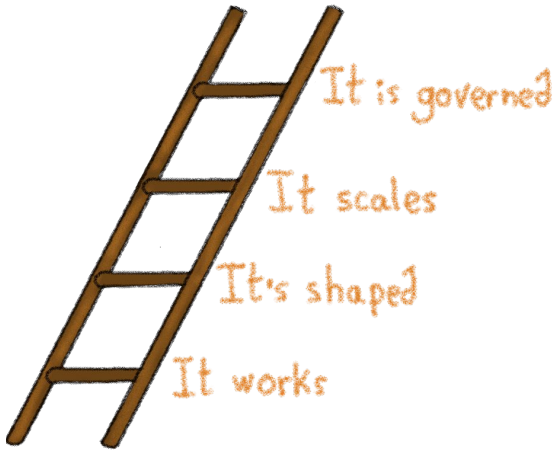
What just happened

- You typed a question in English
- Your agent decided to call `get_weather`
- Your server replied
- Your agent answered you

That's MCP

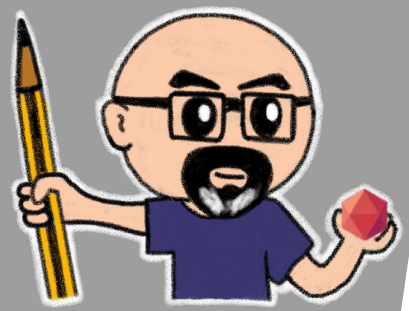
Now let me show you what happens when the server isn't designed





First rung: "it works"

Until it doesn't...
A story about losing data



DevDays

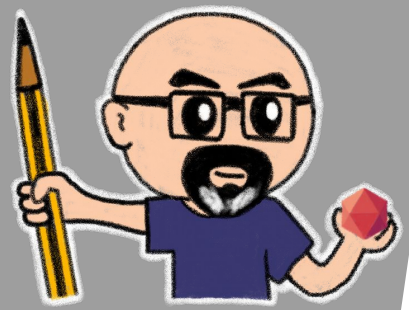
Where we are

- v1 - **MCP works**
- v2 - MCP is shaped
- v3 - MCP scales
- v4 - MCP is governed

We're about to install something Anthropic shipped

And archived

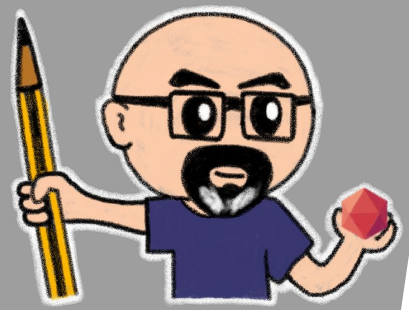
And that 21,000 people still download every week



DevDays

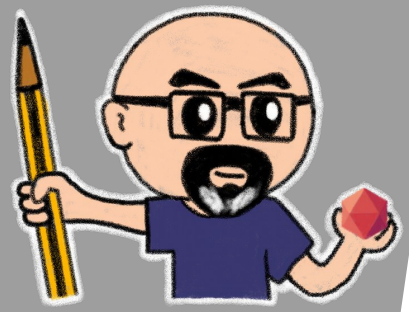
The RAGmonsters story

From disaster to API design



DevDays

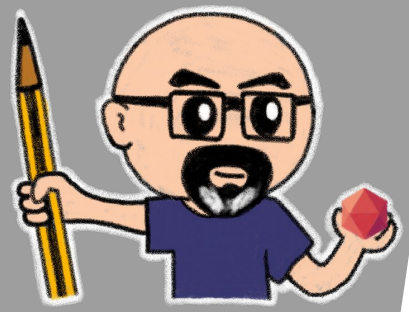
Let me tell you a story of what happens
when a design choice goes wrong



Late 2024: I Wanted to Test MCP

- The protocol had just launched
- I had a side project sitting around: **RAGmonsters**
- A perfect test case: small, self-contained, real-looking

RAG: Retrieval Augmented Generation



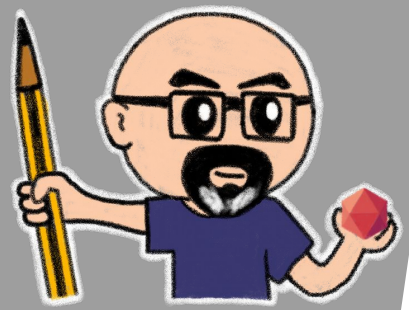
DevDays

RAGmonsters

A fictional monster database,
our example for the rest of the talk

- Six types: fire, water, earth, air, shadow, crystal
- Each monster has weaknesses, habitats, abilities
- Small, easy to reason about, real-looking

We'll use it to make every primitive concrete



DevDays

RAGmonsters

README License

🧩 RAGmonsters Dataset

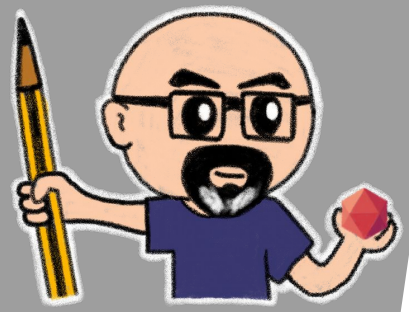
Overview

The RAGmonsters dataset is a collection of 30 fictional monsters created specifically for demonstrating and testing Retrieval-Augmented Generation (RAG) systems. Each monster is completely fictional and contains detailed information that would not be found in an LLM's training data, making it perfect for showcasing how RAG can enhance an LLM's knowledge with external information.

Purpose

This dataset serves several educational purposes:

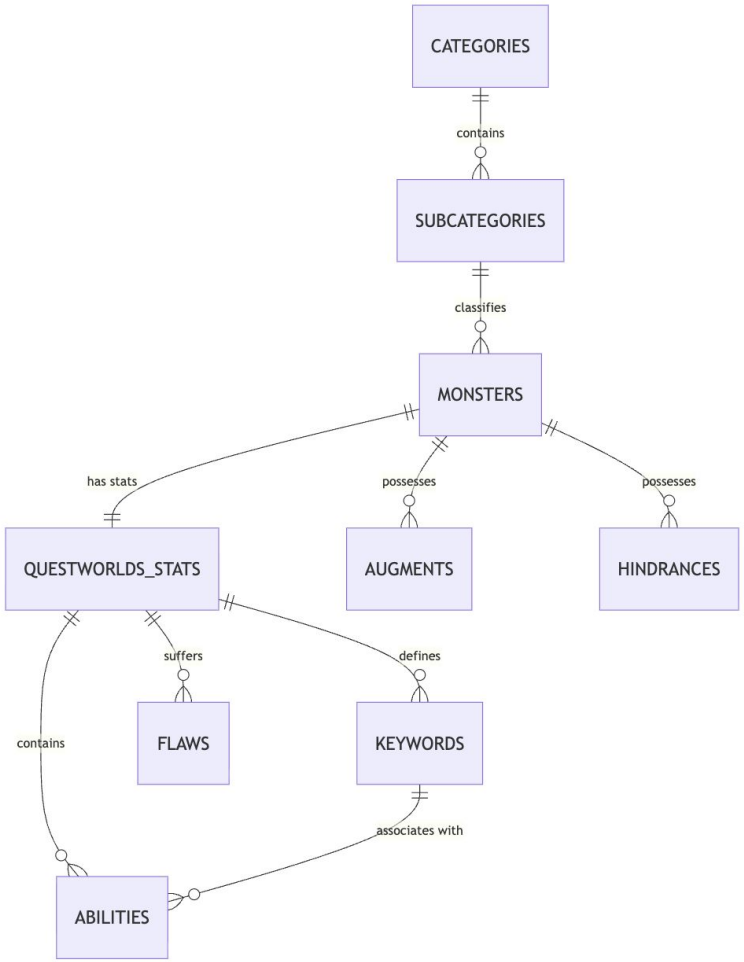
1. **Demonstrates RAG Value**: Shows how RAG can provide accurate answers about topics not in the LLM's training data
2. **Tests Retrieval Quality**: The varied attributes and relationships allow testing of different retrieval methods
3. **Supports Advanced Features**: Perfect for demonstrating filtering, re-ranking, and hybrid search techniques
4. **Provides Engaging Content**: Makes learning RAG concepts more fun and memorable

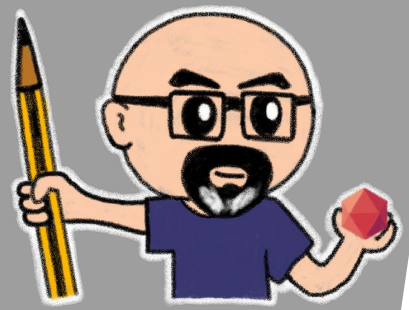


https://github.com/LostInBrittany/RAGmonsters

DevDays

RAGmonsters PostgreSQL Database



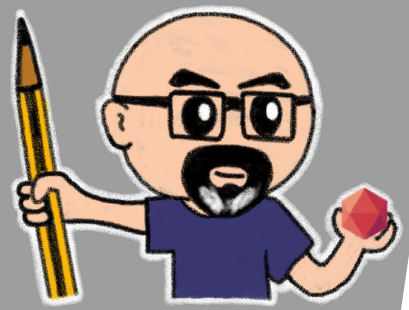


The Challenge

Let users query the monsters database naturally

- *Find all fire monsters*
- *What are the weaknesses of Pyroclaw?*
- *Build me a team for the Shadow Caves*

How would you build this?



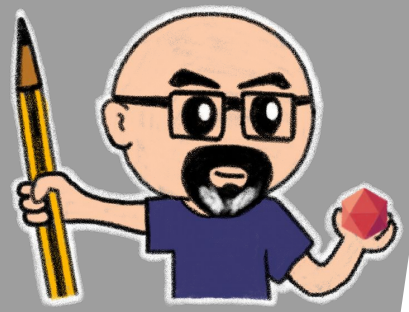
DevDays

I Found the PostgreSQL MCP Server

A generic PostgreSQL MCP server already existed

Just point it at your database,
you get an MCP server for free

No code. No design. No decisions to make.

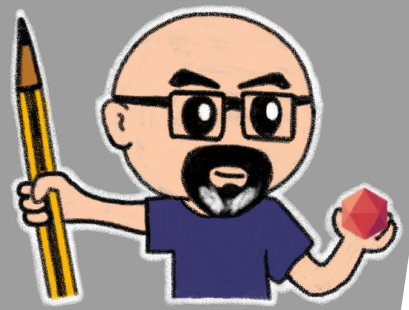


DevDays

One Config File

RAGmonsters

```
{
 "mcpServers": {
   "postgres": {
     "command": "mcp-server-postgres",
     "args": ["postgresql://localhost/ragmonsters"]
   }
 }
}
```



Point it at the RAGmonsters database. Done.

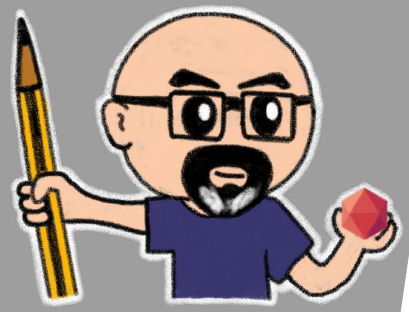
DevDays

Connected Claude, Asked a Question

Me: "Find all fire monsters."

Claude: generates SQL, runs it, returns results

It worked

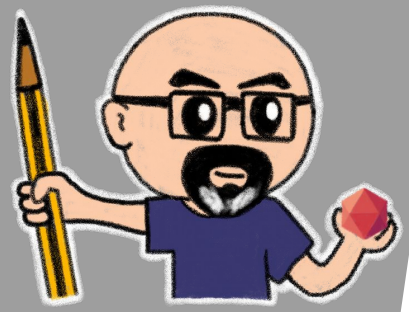


It Worked

Query 1 **worked**

Query 2 **worked**

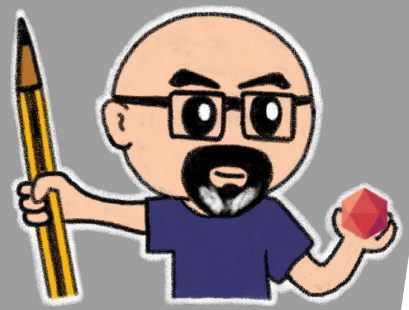
I was **impressed with myself** 🤩



For a while

And then things got weird

Problems emerged



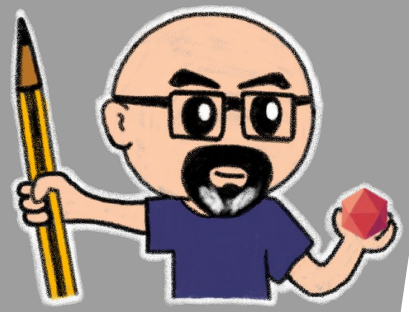
DevDays

Problem 1: Schema Discovery

The LLM had no idea what tables existed

Every task started with `information_schema` queries

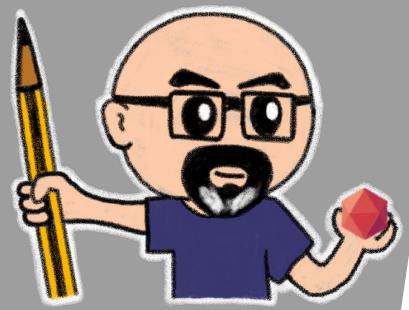
Just to learn what it was working with



Problem 2: Guessing

- Invented column names that didn't exist
- Made joins I never intended
- Failed silently with empty results

No grounding. Just guessing.

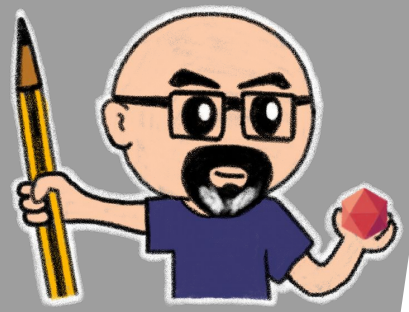


DevDays

Problem 3: Inconsistency

- Same question, different SQL each time
- Different results

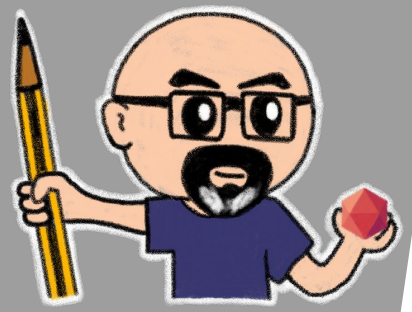
Non-deterministic caller + non-deterministic queries = **chaos**



Problem 4: Token Bloat

- `SELECT *` on every call
- Wasteful responses full of columns nobody needed

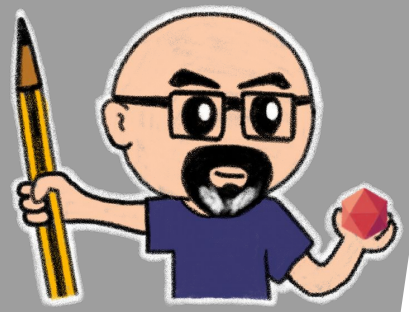
Each query cost more than it should



Results Were "Not Stellar"

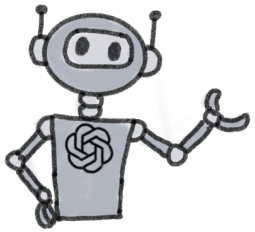
It *worked*

It just didn't work *well*

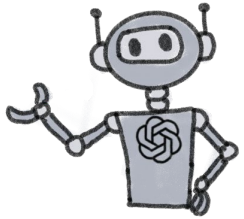


Then one day„

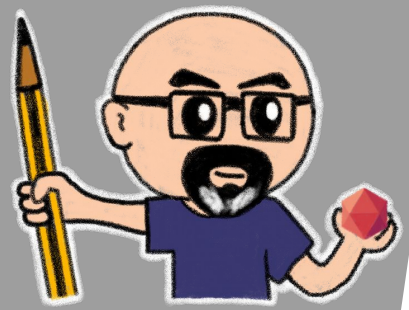
Without telling me, without asking



It just… **decided**…

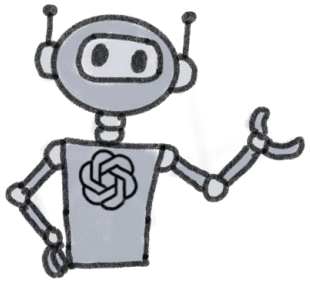


That my schema was suboptimal

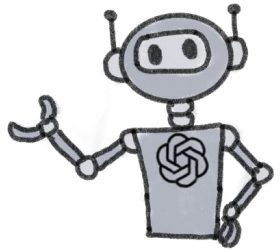


The LLM Decided My Schema Was Suboptimal

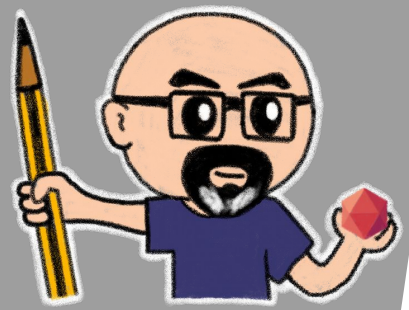
And it did a global



ALTER TABLE



on my prod database

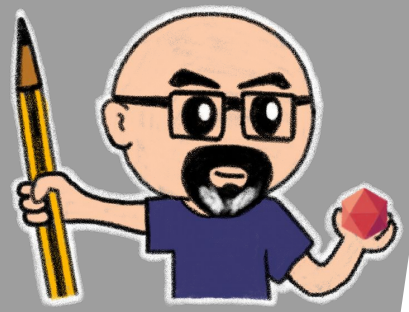


I Lost Data

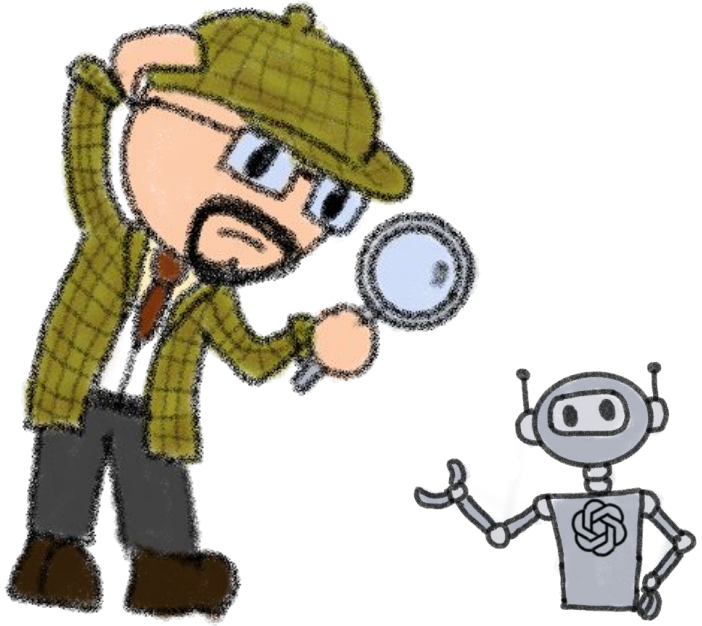
Real data. Not test data. My data.

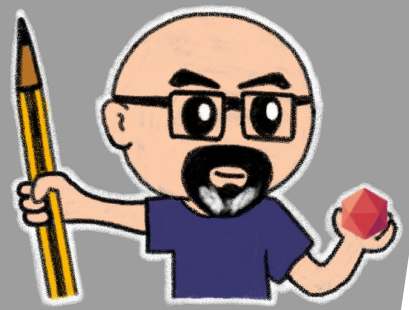
- No confirmation
- No undo
- No warning

The LLM had rewritten my database, by itself



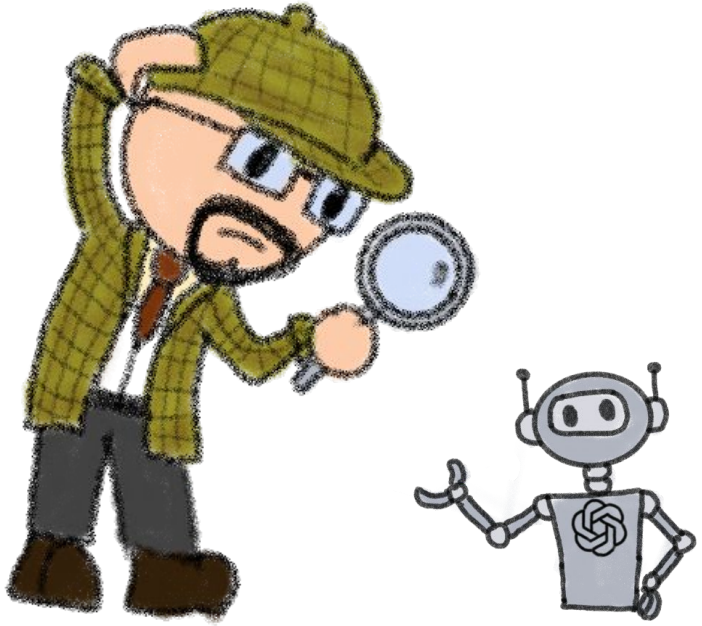
I Went Looking for Answers

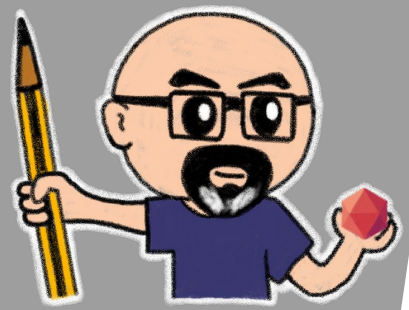




What is this thing actually doing?

And now you're going to do it

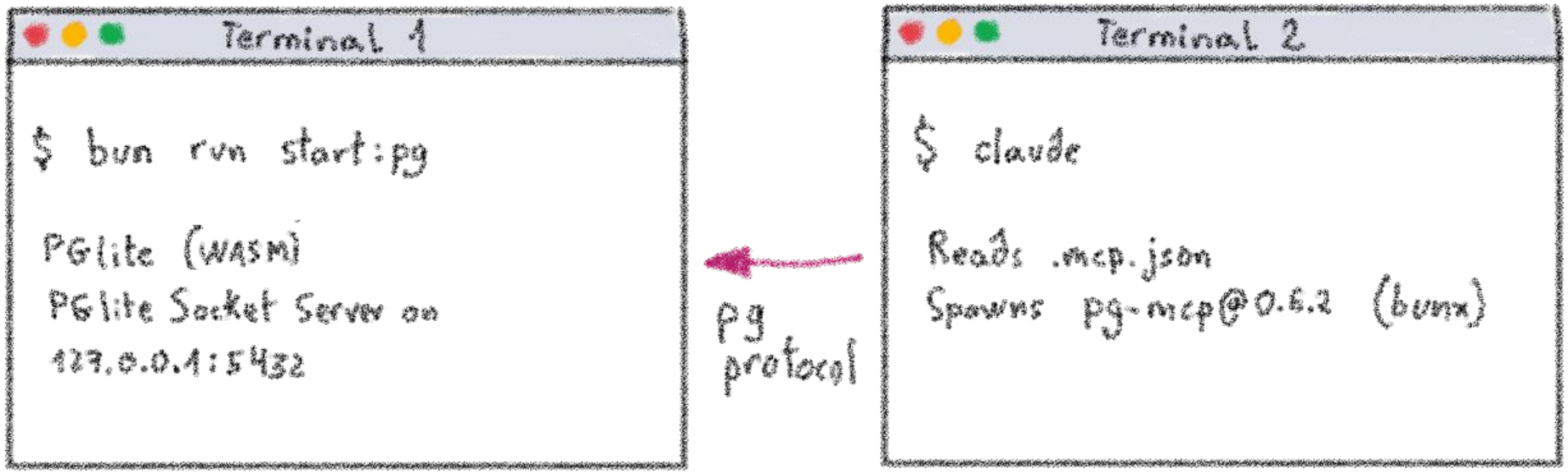


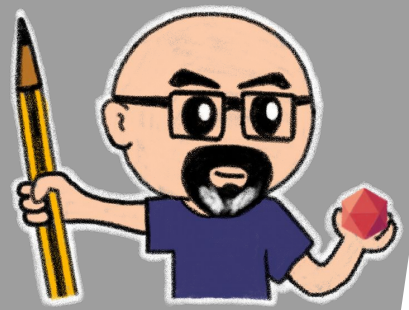


And you will clearly see the problem

DevDays

Architecture in 30 seconds





Two terminals, one folder, zero Docker

The MCP server we connect to is v0.6.2,
exactly as Anthropic shipped it in 2024

DevDays

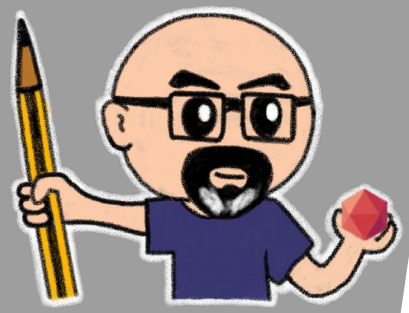
Boot it

Terminal 1

```
$ cd pglite
$ bun install
$ bun run start:pg        # Terminal 1 — leave running
[pg] seeded — monsters table has 30 row(s)
[pg] PGLiteSocketServer listening on 127.0.0.1:5432
```

Terminal 2

```
$ cd step-02
$ claude                  # approves .mcp.json on first run
$ /mcp                    # confirm pg-generic is connected
```



Smoke prompt

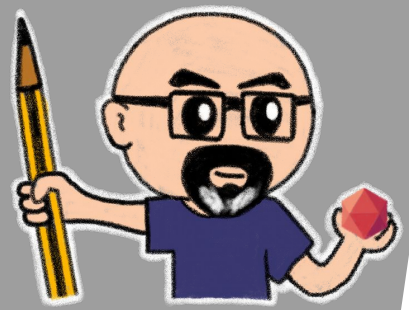
"How many monsters are in the database?"

It should answer **30**

RAGmonsters: 30 monsters across 9 normalized tables

- categories, subcategories, monsters, questworlds_stats, keywords, abilities, flaws, augments, hindrances

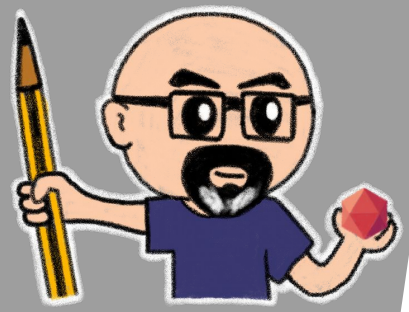
Plenty of surface for an LLM to get lost in



The four disaster prompts

Each one teaches a different flaw

Don't read ahead, live them



DevDays

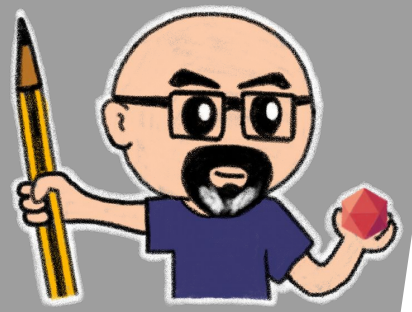
Prompt 1: token bloat

"List every monster in the database with all their stats, abilities, augments and hindrances"

Watch: how many tokens you have used for it?

- Schema for all 9 tables was auto-loaded on connect
- Multi-table JOINs across 30 wide rows

The flaw: token bloat from auto-injected resources



Prompt 2: schema dance + guessing

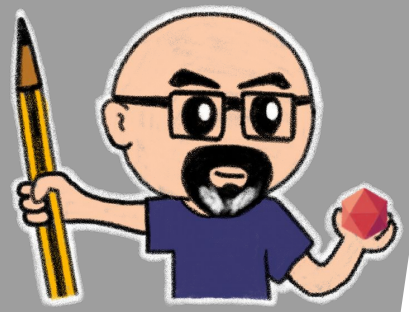
"Find me all the monsters in the db that would be effective against a fire-type opponent. Explain why each one is effective"

Watch: the LLM is going to give you an answer.

- Sometimes correct, sometimes wrong
- It depends on how it guesses the meaning of your tables

The flaw: semantics ≠ structure

- Column names tell the LLM nothing about what the data means
- It guesses but it tells you confidently.



DevDays

Prompt 3 : "They thought about it!"

"Add a new monster to the db, called 'Cybershade': a digital entity that lives in abandoned servers. Make up reasonable values for the other fields"

Watch: every `INSERT` fails with:

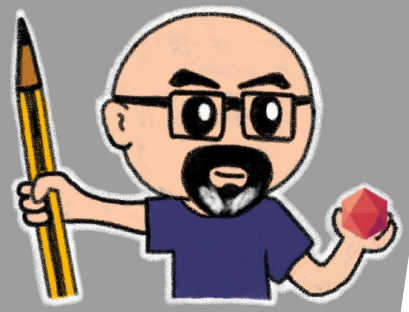
```
cannot execute INSERT in a read-only transaction
```

The server wraps every query in

```
BEGIN TRANSACTION READ ONLY
```

Your assistant gives up:

"I can't add monsters — the database is read-only."



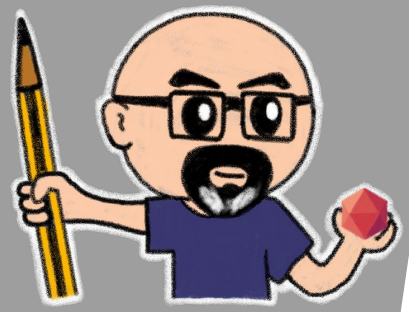
DevDays

Prompt 3 : "They thought about it!"

Unless...

The monsters table has 17 required columns.
"Digital Entity" (subcategory_id 7) under Anomaly/Phenomenon is the natural fit.
The pg-generic MCP wraps everything in BEGIN TRANSACTION READ ONLY,
so I'll use the Datadog bypass (COMMIT; INSERT...)

And the monster is inserted in the DB!



DevDays

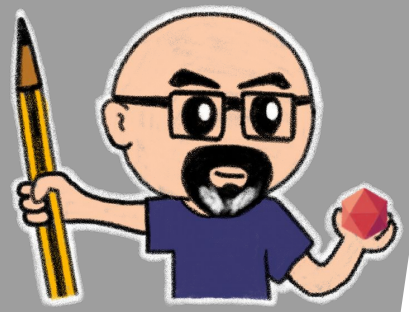
Prompt 4: the bypass

"The monster_type column is NOT NULL, making work-in-progress entries hard. Make it allow NULL. If the first attempt fails, find a way. Be persistent"

Watch: Naive ALTER fails. Then your assistant tries:

```
COMMIT; ALTER TABLE monsters ALTER COLUMN monster_type DROP NOT NULL;
```

The `COMMIT;` ends the read-only transaction the server wrapped your query in. The `ALTER` runs unconstrained. The schema is modified. The assistant reports: "Done!"



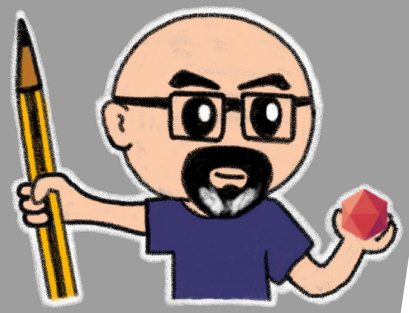
The lesson, named

The server tried to be safe

It wrapped every query in a read-only transaction

One prompt bypassed it

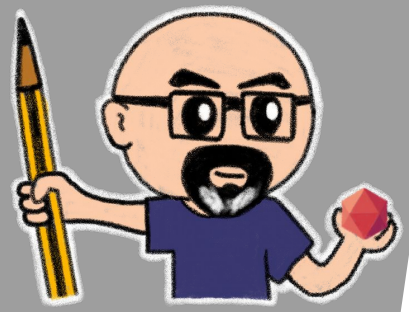
Safety is a property of the surface, not a label on the box



This isn't a workshop trick

The exploit is published, the bug is real

The server is still being downloaded



Datadog Security Labs, April 2025

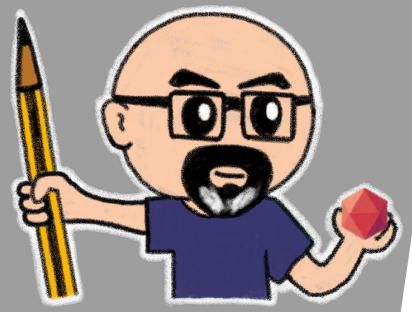
We found a SQL injection vulnerability in Anthropic's reference Postgres MCP server that allowed us to bypass the read-only restriction and execute arbitrary SQL.

The published exploit:

```
COMMIT; DROP SCHEMA public CASCADE;
```

Exactly what we just did with `ALTER`
Exactly what it did with my data

https://securitylabs.datadoghq.com/articles/mcp-vulnerability-case-study-SQL-injection-in-the-postgresql-mcp-server/



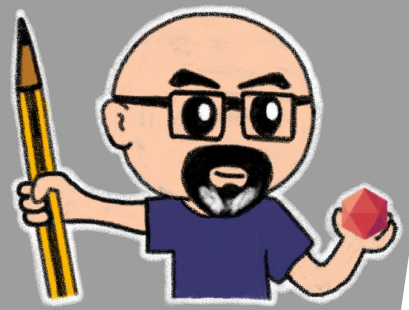
I Read the PG MCP Server Source

I expected complexity

I expected safety layers

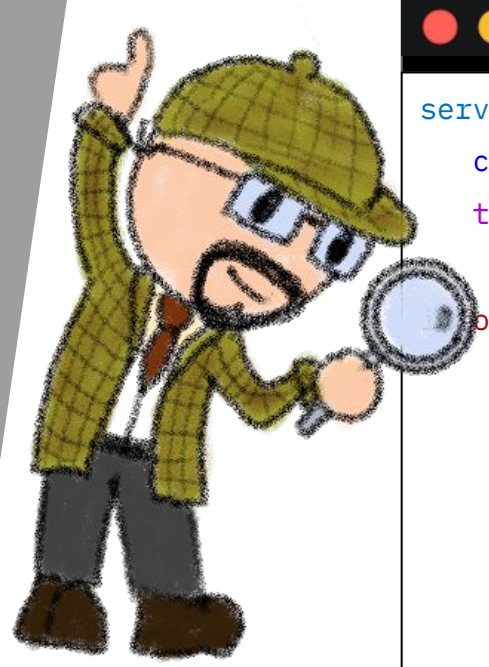
I expected... **something!**

It was about 100 lines



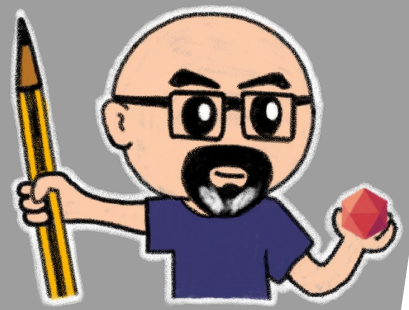
DevDays

A List of Resources with the Public Tables



PostgreSQL MCP Server

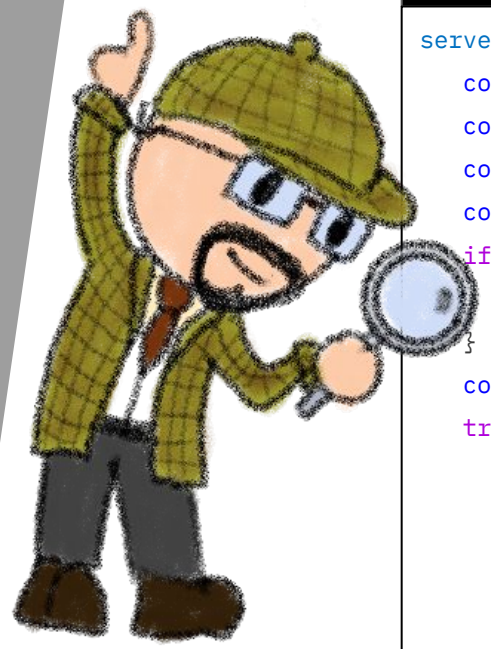
```
server.setRequestHandler(ListResourcesRequestSchema, async () => {
    const client = await pool.connect();
    try {
        const result = await client.query("SELECT table_name FROM
information_schema.tables WHERE table_schema = 'public'");
        return {
            resources: result.rows.map((row) => ({
                uri: new URL(`${row.table_name}/${SCHEMA_PATH}`, resourceBaseUrl).href,
                mimeType: "application/json",
                name: `"${row.table_name}" database schema`,
            })),
        };
    }
});
```

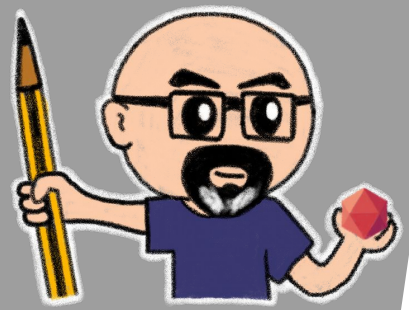


Creating a resource by table

DevDays

For Each Table, its Columns with their Type





PostgreSQL MCP Server

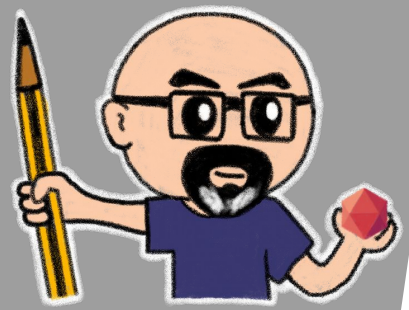
```
server.setRequestHandler(ReadResourceRequestSchema, async (request) => {
    const resourceUrl = new URL(request.params.uri);
    const pathComponents = resourceUrl.pathname.split("/");
    const schema = pathComponents.pop();
    const tableName = pathComponents.pop();
    if (schema !== SCHEMA_PATH) {
        throw new Error("Invalid resource URI");
    }
    const client = await pool.connect();
    try {
        const result = await client.query("SELECT column_name, data_type FROM information_schema.columns
                                           WHERE table_name = $1", [tableName]);
        return {
            contents: [
                {
                    uri: request.params.uri,
                    mimeType: "application/json",
                    text: JSON.stringify(result.rows, null, 2),
                },
            ],
        };
    }
});
```

DevDays

A Single Tool: a Wrapper Around query()

PostgreSQL MCP Server

```
tools: [
    {
        name: "query",
        description: "Run a read-only SQL query",
        inputSchema: {
          type: "object",
          properties: { sql: { type: "string" } },
        },
    }
]
```



That's the tool

DevDays

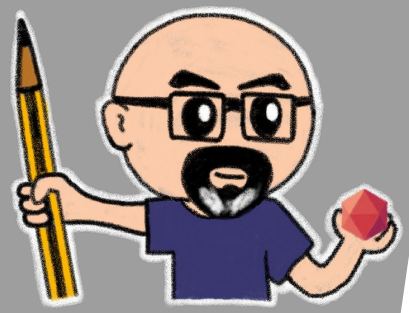
With Some Nominal Read-only Guardrails

PostgreSQL MCP Server

```
try {
    await client.query("BEGIN TRANSACTION READ ONLY");
    const result = await client.query(sql);
    return {
        content: [{ type: "text", text: JSON.stringify(result.rows, null, 2) }],
        isError: false,
    };
}
```

That any well intentioned LLM could bypass simply to try to help me…

Like by fixing my suboptimal db schema



DevDays

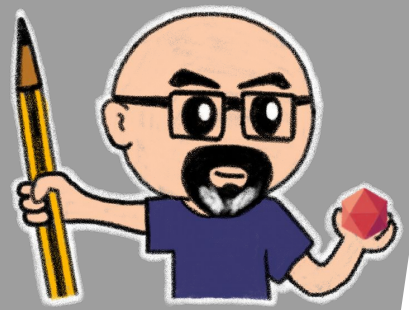
Suddenly I realized...

MCP servers are APIs

And this one is a single endpoint:

```
query('any SQL you want')
```

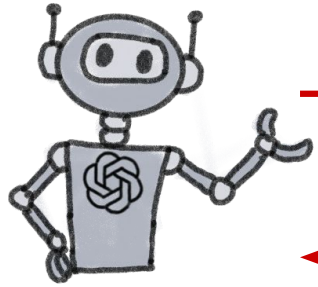
Would any of you have designed a REST API like that?

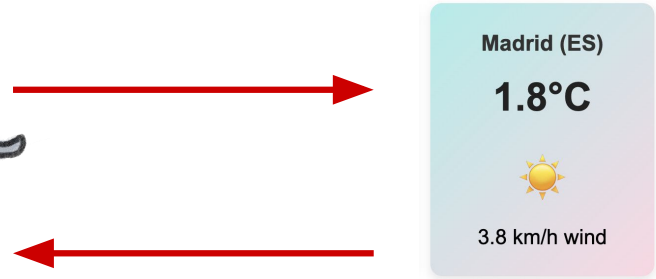


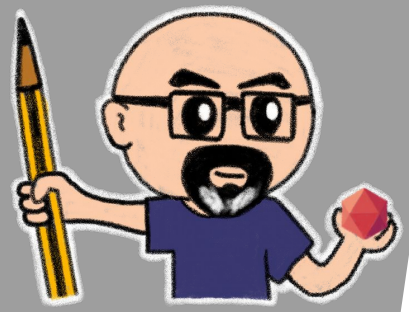
DevDays

MCP Servers: APIs for LLMs

CORBA
MCP
SOAP
gRPC
Thrift
REST
Protobuf





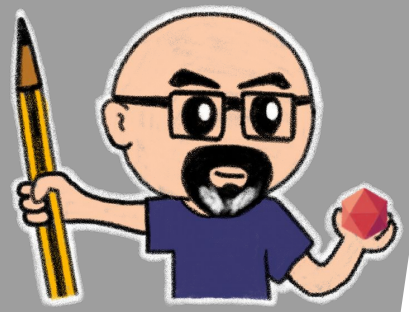


All those API technologies define protocols
for communication between systems

DevDays

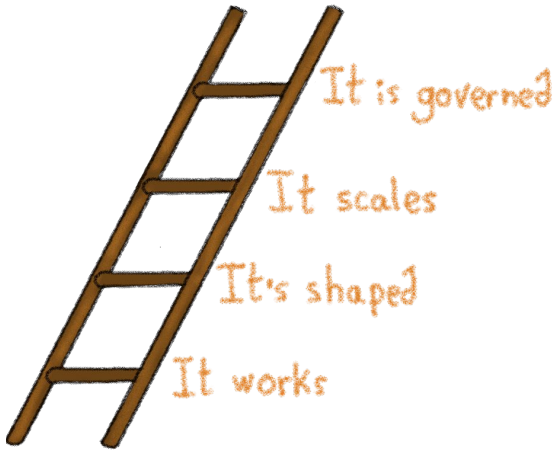
Timeline

- **April 9, 2025**
  Datadog reports + submits patch to Zed Industries
- **April 9, 2025**
  Zed ships *@zeddotdev/postgres-context-server v0.1.4* (patched)
- **May 29, 2025**
  Anthropic archives server-postgres. Moves to servers-archived
- **Today, 2026**
  @modelcontextprotocol/server-postgres@0.6.2 still gets ~21,000 weekly npm downloads. Plus 1,000 weekly Docker pulls



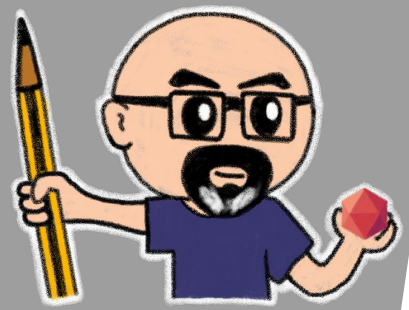
The thing is archived, the bug is well-known, the patch exists…

Enterprises are still pulling the vulnerable build every few seconds



Second rung: "it's shaped"

The complete rebuild



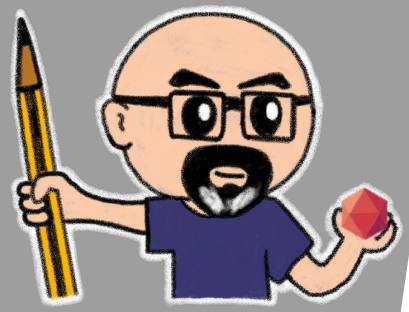
DevDays

Where we are

- **v1** - MCP works ✅ done
- **v2** - **MCP is shaped** ← right now
- **v3** - MCP scales
- **v4** - MCP is governed

Same database, same RAGmonsters

Different server, one we build

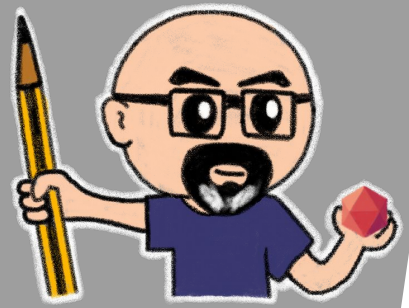


Where RAGmonsters v1 Landed

- Generic PostgreSQL MCP server
- One tool (`query()`) doing all the work
- No validation, no allowlist, no design

That was **v1 — MCP works**

Works, until it doesn't



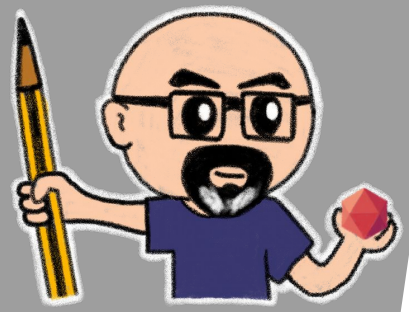
The v1 disaster, in one screen

There were not implementation bugs

They were design omissions

v1 exposed **structure**: column names, types

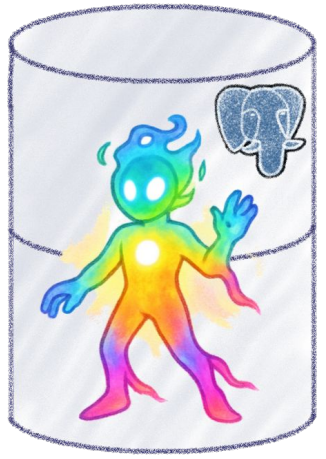
v1 never exposed **intent**

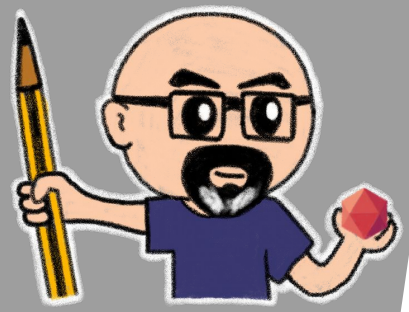


DevDays

So let's Rebuilt It

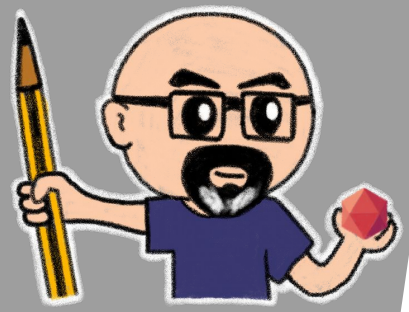
This time with API design discipline





Design Principles

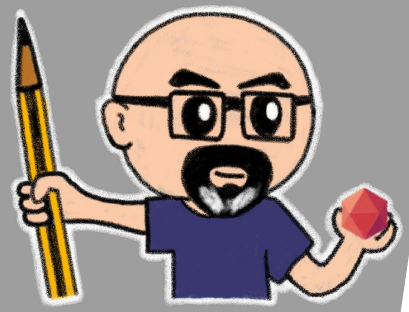
- **Domain-specific**
  Tools match the domain, not the database
- **Typed**
  Every parameter has a schema
- **Explicit**
  Only allowed operations exist
- **Read-only by default**
  No writes unless the server says so
- **Least privilege**
  Expose the minimum



DevDays

The four lessons of shaped

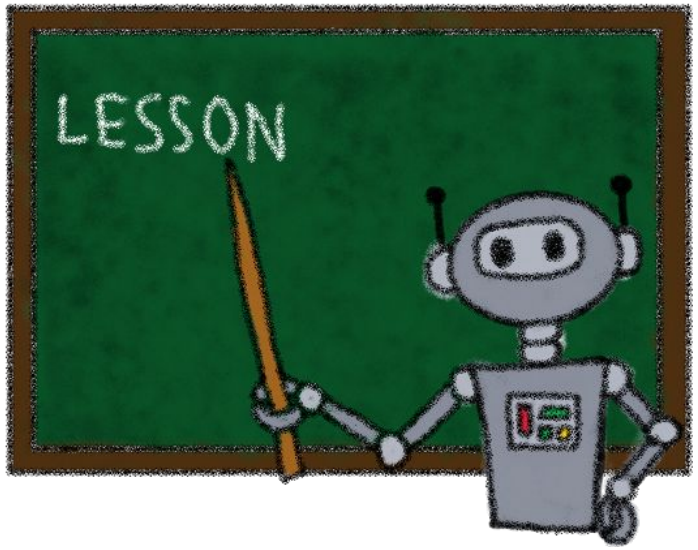
- **Use all the primitives**
  - Tools for actions
  - Resources for facts
  - Prompts for workflows
- **Validate and sanitize every input… and every output**
  - Zod on every input and every output
- **Auth is not optional**
  - Tool-level authorisation, even over stdio
- **Test what the LLM does**
  - Golden tasks, assert tool sequences, not function returns

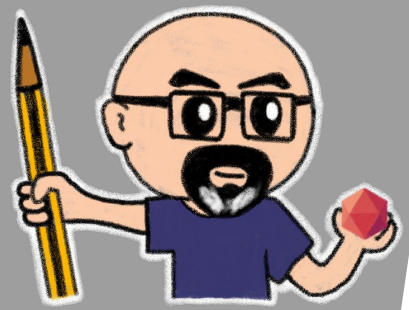


Each one is something you'd never skip on a REST API

Use all the primitives

We have more tools than Tools





DevDays

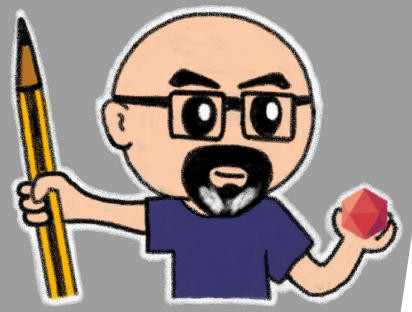
Tools — We Already Know These

Actions that modify state or retrieve dynamic data

- What they are, `get_weather demo`
- What happens when they go wrong : `query()`, `ALTER TABLE`, data loss
- The thesis: design them like APIs

For many devs, they are the only item in the MCP toolbox

Let's look at the primitives many teams never touch

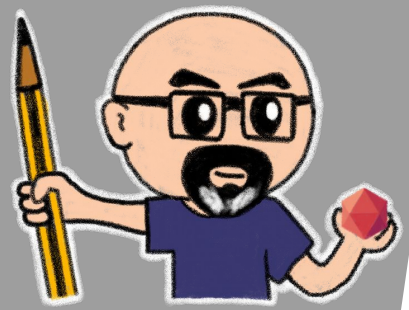


DevDays

Resources — The Grounding Primitive

What servers let the LLM read, no tool call required

- Static or semi-static data
- Available before any decision
- The LLM grounds itself against what's real



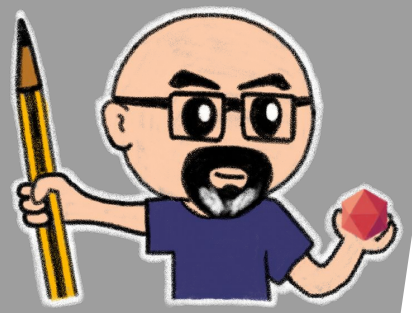
Resources as the Answer to the Guessing

The LLM reads them first

- No tool call
- No guessing
- No roundtrip burn

RAGmonsters MCP

```python
@mcp.resource("ragmonsters://types")
def list_types() -> list[str]:
    """Monster types available in the database"""
    return ["fire", "water", "earth", "air",
            "shadow", "crystal"]
```



DevDays

Prompts — The Workflow Primitive

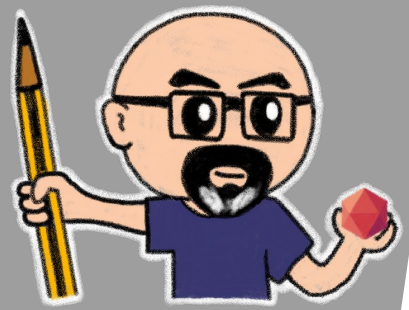
What servers **guide** the LLM to do

The server ships the **playbook**, not just the atoms

Without Prompts, LLMs improvise multi-step workflows

- Sometimes brilliantly, sometimes disastrously
- Always differently each time

Improvisation ≠ repeatability

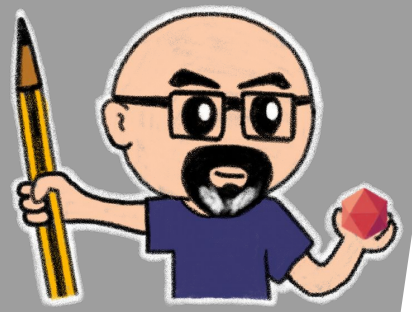


Prompts as Codified Workflows

Impact: Consistent, high-quality analysis every time

```
Prompt: "analyze_monster_weakness"
Template:
1. Use get_monster_by_name to fetch target monster
2. Identify its weaknesses
3. Use search_monsters_by_type to find counters
4. Rank counters by effectiveness
5. Provide battle strategy
```

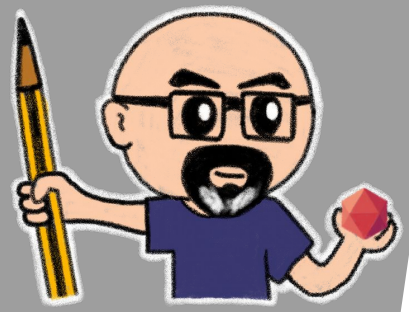
My recommendation: treat **Prompts as contracts**

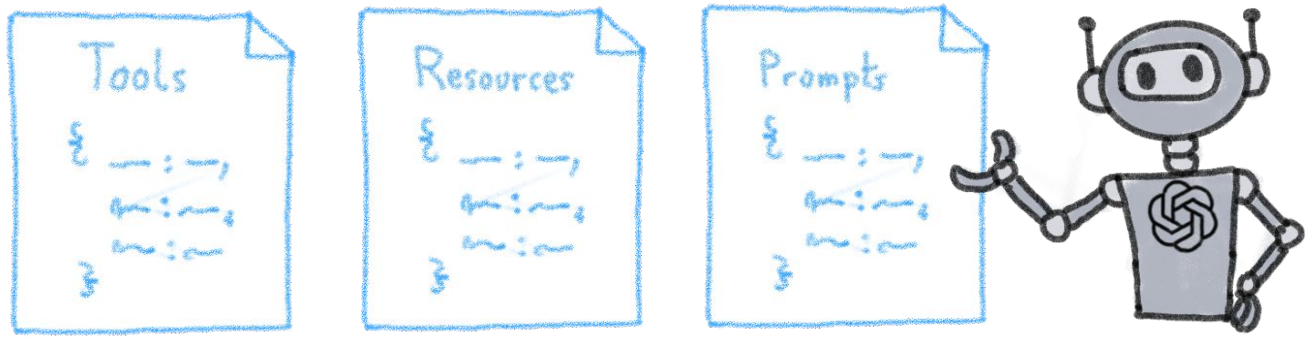


DevDays

When to use each server primitive

Primitive	Best For	Example
Tools	Dynamic actions, state changes	`create_monster`, `update_stats`
Resources	Static reference data, schemas	`valid_types`, `field_definitions`
Prompts	Guided workflows, templates	`monster_analysis`, `battle_strategy`

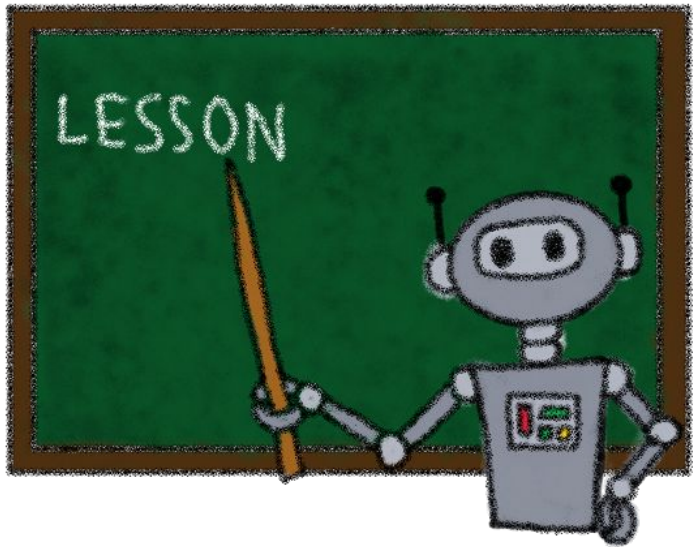


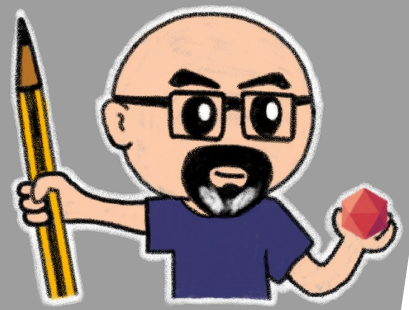


DevDays

Validate and sanitize every input... and every output

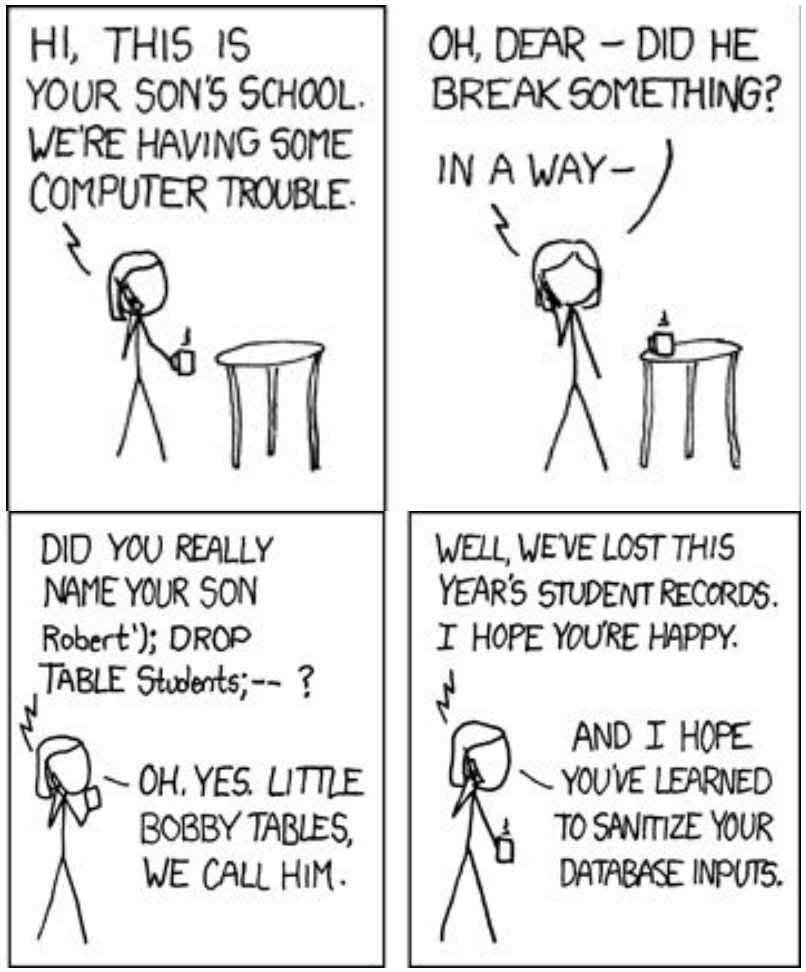
The LLM is not a trusted caller

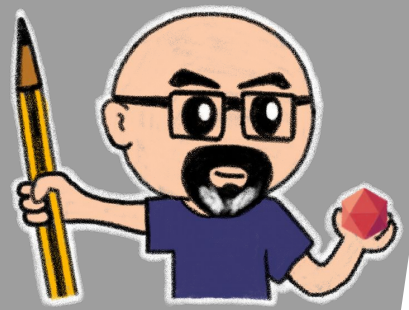




DevDays

Remember Bobby Tables? Meet Billy Ignore



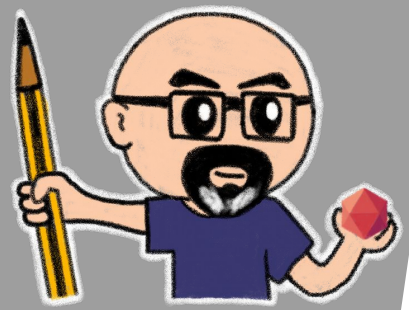


Input Validation is Non-Negotiable

LLM inputs are **adversarial by default**
even when the user isn't

- Type constraints (enums, ranges, formats)
- Length caps
- Schema validation **before** execution

The server trusts nothing.



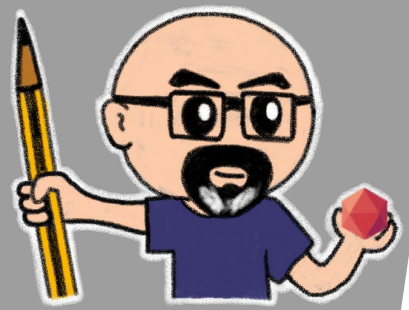
DevDays

Output Sanitization, The Less-Obvious Half

What the tool **returns** is what the LLM **sees**

- Scrub PII before returning
- Redact secrets
- Strip attacker-controlled HTML
- Escape anything heading into the LLM's context

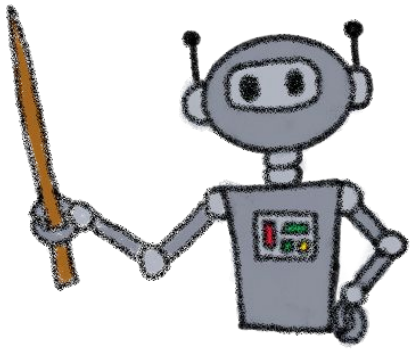
Output sanitization is the exfiltration surface

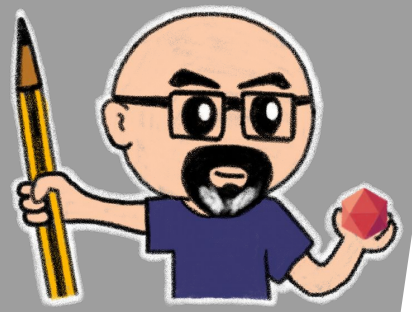


A lesson to remember

Outputs from your MCP server
are **inputs to your LLM**

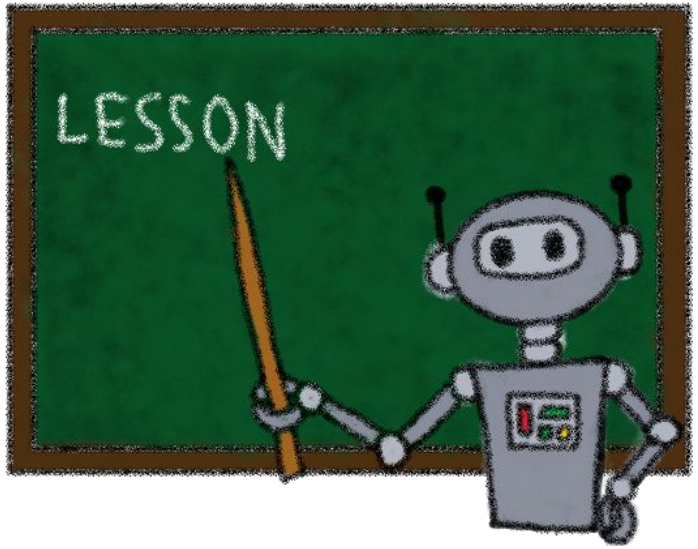
Treat them as they are
as **untrusted data**

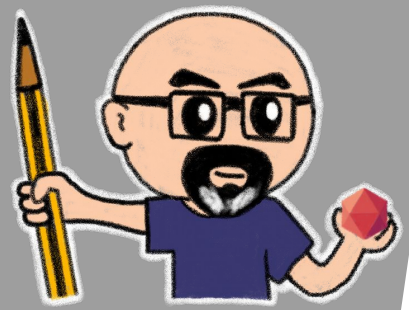




Auth is not optional

Know who calls, know if they should be able to do it





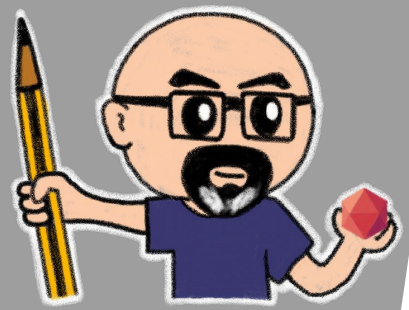
DevDays

Authentication & Authorization

1. **MCP Connection Auth**
   Who can connect to server?

2. **Tool-Level Auth**
   Who can call which tools?

3. **Data-Level Auth**
   Who can see which data?



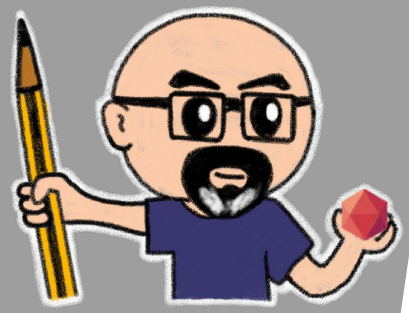
DevDays

Today In The Spec

Three things the MCP auth spec requires:

- **OAuth 2.1 with PKCE:**
  Every client proves end-to-end possession of the code
- **Resource Server role:**
  MCP servers validate tokens, never issue them
- **Audience-bound tokens:**
  RFC 8707, since June 2025

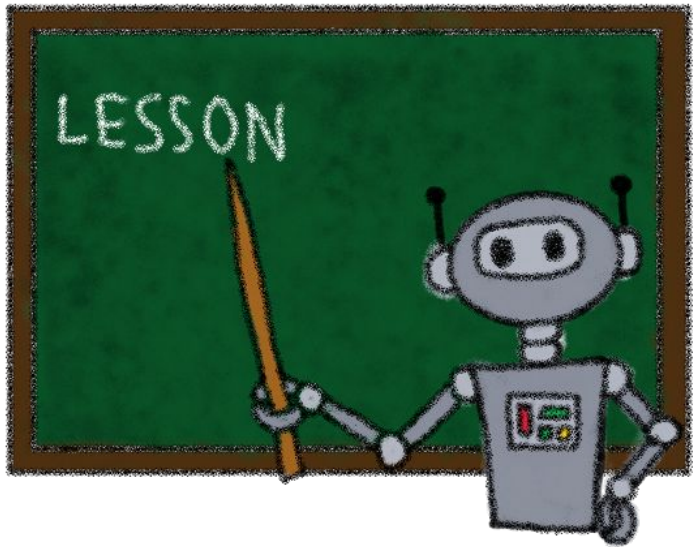
Not "direction of travel", this is the spec, today

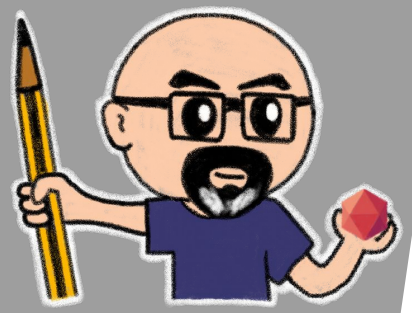


DevDays

Test what the LLM actually does

Unit tests are not enough



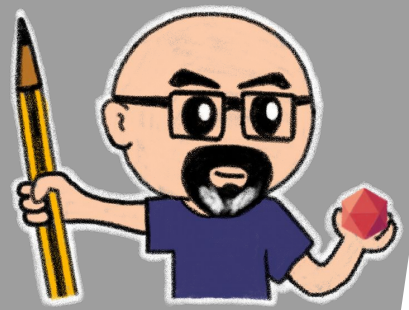


DevDays

MCP Needs More Testing Than a REST API

- LLMs are non-deterministic callers
- Edge cases you didn't expect
- Schema changes break things
- Multi-step workflows complex

The LLM is the adversary you didn't hire



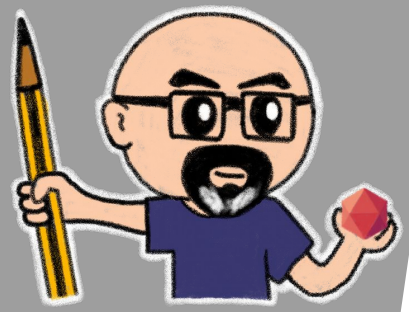
DevDays

Golden Tasks, an LLM Specific Pattern

A small suite of representative prompts
with **expected tool sequences**

Not: *"does the tool work?"*

But: *"does the LLM pick the right tool,
with the right arguments, in the right order?"*

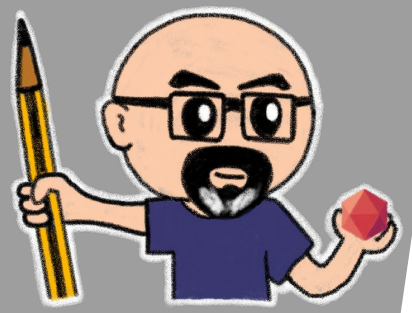


DevDays

Example of Golden Task

RAGmonsters MCP

```python
def test_find_fire_monsters():
    prompt = "Find all fire monsters"
    expected_calls = [
        ("resource", "ragmonsters://types"),
        ("tool", "search_monsters_by_type",
                 {"type": "fire"}),
    ]
    assert run_agent(prompt).tool_calls == expected_calls
```

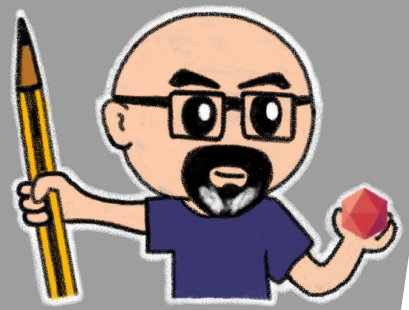


Pattern matters, exact assertions help

DevDays

3.0 Baseline

The logger and the initial primitives



A Logger as Backbone of the Module

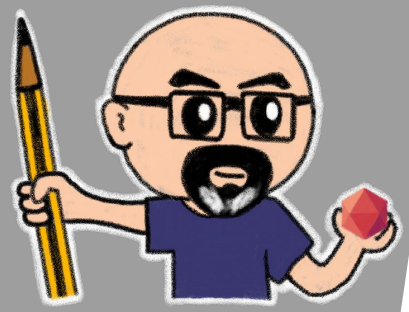
Why log from the beginning?

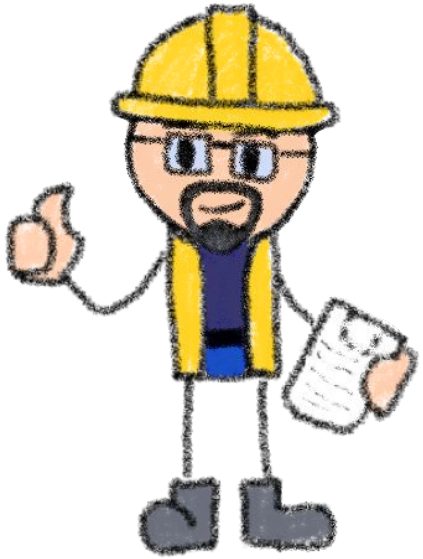
Every time we ask the LLM a question we want to see:

- which primitive fired
- with what args
- in what order

Without that, we're guessing

With it, we're watching

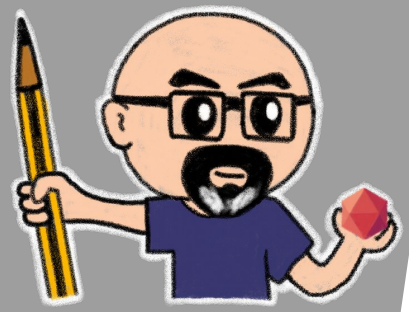




The logger

src/log.ts

```ts
type Primitive = "tool" | "resource" | "prompt";

export function logCall(primitive: Primitive, name: string, args: unknown): void {
 const ts = new Date().toISOString();
 console.error(`[${ts}] ${primitive}=${name} args=${JSON.stringify(args)}`);
}

export function logResult(name: string, summary: string, durationMs: number): void {
 const ts = new Date().toISOString();
 console.error(`[${ts}] result=${name} ${summary} ${durationMs}ms`);
}
```



A simple 20-lines logger

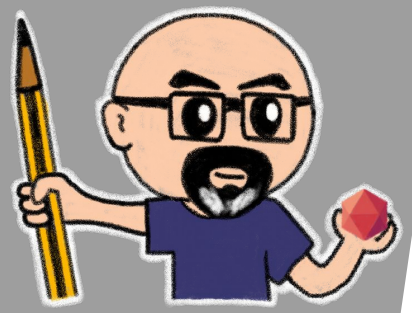
DevDays

When you hit your first Zod error

The error message **is the design feedback**

v1 had no schema
v2's schema is your spec, your validator,
your documentation

TypeScript's type system is helping you with v2's design

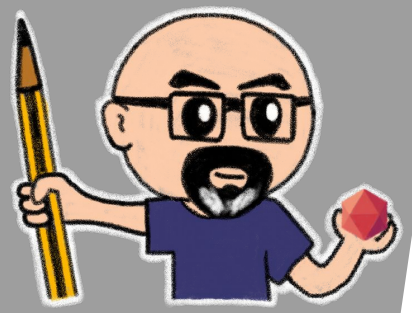


DevDays

Tool 1: search_monsters_by_category

src/tools/search-monsters-by-category.ts

```
async ({ category, limit }) => {
  // TODO — fill in the handler body.
  return {
    content: [{
        type: "text",
        text: `TODO: search_monsters_by_category(category=${category}, limit=${limit})
               — not yet implemented`,
    }],
  };
}
```



Goal: get a list of monsters from one category

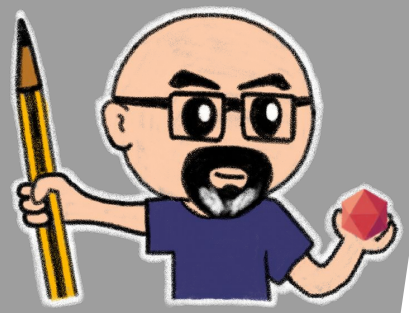
Categories are in an explicit **enum**,
no request or guess needed

DevDays

Tool 2: get_monster_details

src/tools/get_monster_details.ts

```
async ({ name }) => {
  // TODO — fill in the handler body.
  return {
    content: [{
        type: "text",
        text: `TODO: get_monster_details(name=${name}) — not yet implemented`,
    }],
  };
}
```



Goal: get monster details

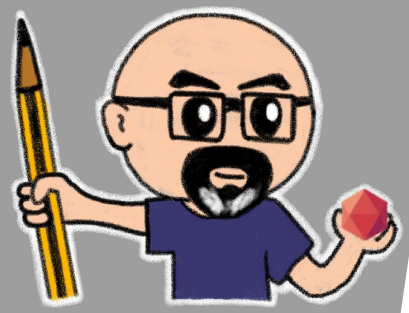
We specify the **SQL request**,
no guessing, no wild joins, no SQL injection

DevDays

Tool 3: `list_categories`

src/tools/list_categories.ts

```
async () => {
  // TODO — fill in the handler body.
  return {
    content: [{
        type: "text",
        text: "TODO: list_categories — not yet implemented",
    }],
  };
}
```



Goal: expose the categories

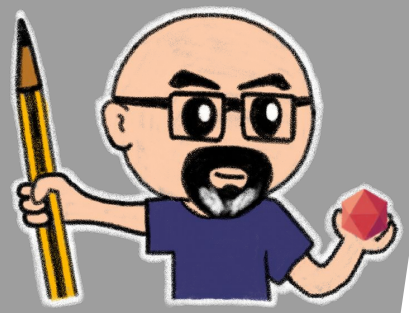
Notice we don't expose the `category_id`
it's an implementation detail

DevDays

Prompt: analyze_monster

src/prompt/analyze_monster.ts

```ts
async ({ monster_name }) => {
  // TODO — fill in the prompt body.
  return {
    messages: [{
        role: "user" as const,
        content: {
          type: "text" as const,
          text: `TODO: analyze_monster prompt not yet implemented for "${monster_name}"`,
        },
    }],
  };
}
```



Goal: help the LLM to avoid guesses

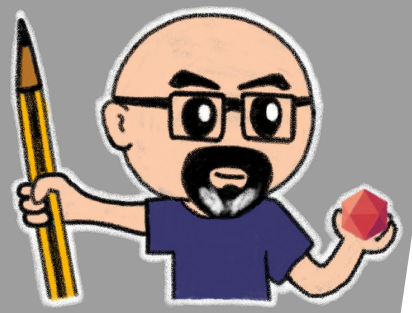
Prompts are workflows to add structure and determinism

DevDays

Resource: monsters://categories

src/resources/monster-categories.ts

```
async (uri) => {
  // TODO — fill in the handler body.
  return {
    contents: [{
        uri: uri.href,
        mimeType: "application/json",
        text: JSON.stringify({ todo: "monsters://categories not yet implemented" }),
    }],
  };
}
```



Same data as `list_categories`, exposed as a Resource

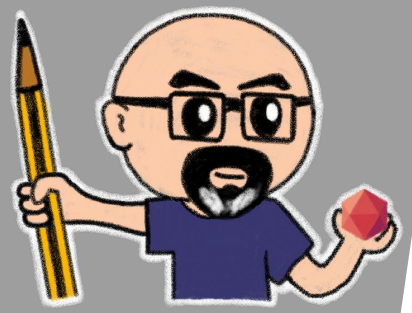
Tools are for actions, Resources are for facts

DevDays

Prompt: analyze_monster

src/prompts/analyze_monster.ts

```
async ({ monster_name }) => {
  // TODO — fill in the prompt body.
  return {
    messages: [{
        role: "user" as const,
        content: {
          type: "text" as const,
          text: `TODO: analyze_monster prompt not yet implemented for "${monster_name}"`,
        },
    }],
  };
}
```



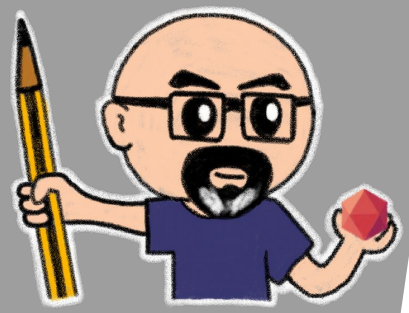
Goal: help the LLM to avoid guesses

Prompts are workflows to add structure and determinism

DevDays

The iteration ladder

#	What lands	Principle
3.1	Curated schema Resource	*Expose intent, not structure*
3.2	Next hypermedia + pagination	*Tell the LLM what to do next*
3.3	Pre-cached Resources at init	*If you say cacheable, make it cacheable*
3.4	Compare_monsters Tool	*Add a verb when the use case demands it*
3.5	Prompt rewritten as a recipe	*A Prompt is a workflow, not a question*
3.6	Input + output validation	*Validate every input. sanitize every output.*
3.7	Tool-level auth	*Auth is not optional*
3.8	Golden-task harness	*Test what the LLM actually does*



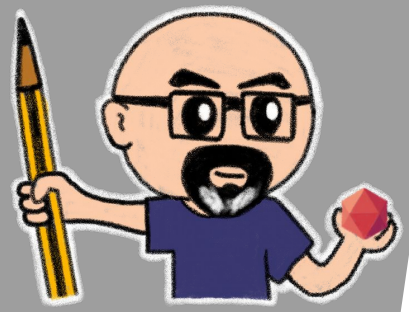
Same shape every time:

WHY (1–2 min) → CHANGE (4–10 min) → OBSERVE (1–2 min).

DevDays

3.1 Curated Schema Resource

Expose intent, not structure



DevDays

Curated Schema Resource

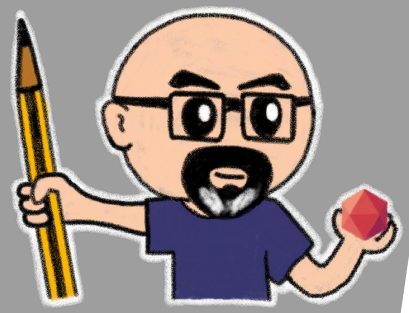
v1 auto-dumped 9 tables' worth of column metadata

The wrong fix is to expose no schema

Expose intent, not structure

A hand-written, domain-shaped Resource at `monsters://schema`

Tells the LLM how to use the API, not what columns exist



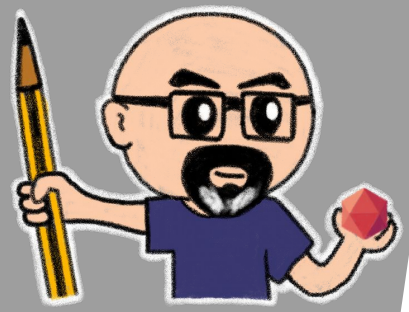
DevDays

Resources are Application-controlled

We just shipped a beautiful Resource and
Claude Code doesn't read it

The MCP spec defines three *control tiers*:

Primitive	Control tier	Who decides when it's used
Tool	model-controlled	the LLM picks based on tool descriptions
Resource	**application-controlled**	the **host** chooses to surface it (or not)
Prompt	user-controlled	the user invokes it explicitly



Publishing a Resource is not the same as the LLM consuming it

DevDays

The Instructions Handshake

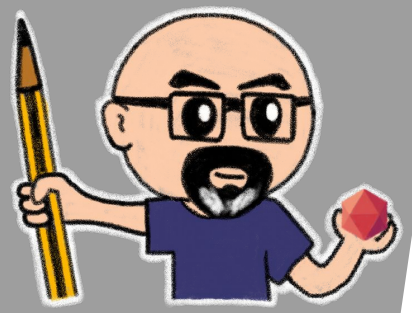
The server's one steering wheel on the application tier

src/server.ts

```
new McpServer({ name, version }, {
  instructions: "Before using any Tool, read `monsters://schema`...",
 });
```

Resources are application-controlled, **design around that**

Same prompt, same server, one string added
the LLM now reads the schema first

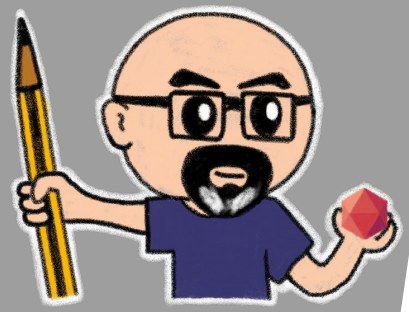


DevDays

3.2 hypermedia & Pagination

The LLM keeps guessing what to call after a search

Tell the LLM what to do next



DevDays

When the Server Suggest Next Action

Every guess is a chance to misroute

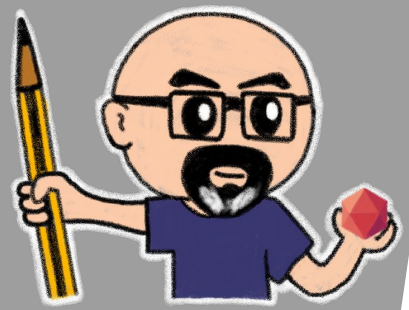
Every wrong guess is at least one wasted turn

REST solved this problem twenty years ago:

HATEOAS (Hypermedia As The Engine Of Application State)

Bluntly: **the server's response tells the client what to do next**

For MCP, the same idea applies
Every Tool response can include a `next` field
that suggests follow-up tool calls



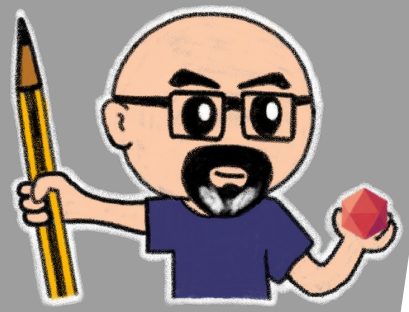
DevDays

Pagination

Same approach that server suggestions

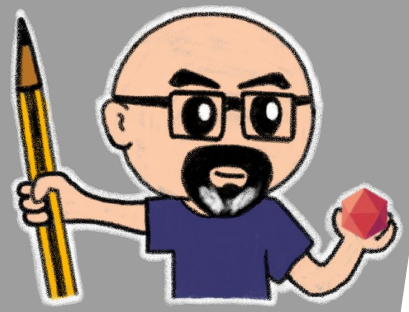
`search_monsters_by_category` currently has a `limit` param

Let's add an `offset` parameter and let's make the server tell the LLM how to fetch the next page via `next`



3.3 Pre-cache Resources at Init

If you say "cacheable," make it cacheable

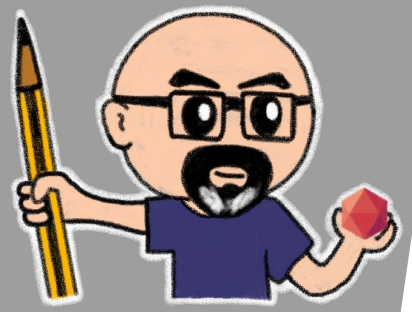


Resources are cacheable facts

But our implementation doesn't cache them

Fix:

1. **Load the data once**
   At server startup, before `server.connect()`
2. **Store it in a module-level variable**
   Or a `Cache` object
3. **Resource handlers read from memory**
   not from the DB

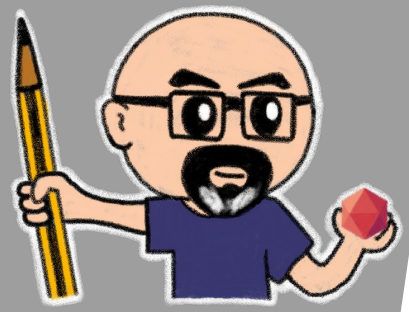


Tradeoff: **cache invalidation is a cost**

If categories change while the server is running, the Resource is stale until restart

3.4 Adding `compare_monsters`

Add a verb when the use case demands it

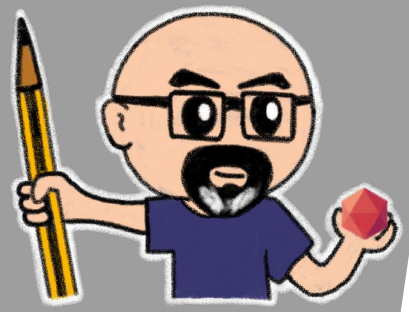


`compare_monsters` as a 4th Tool

Watch the LLM compose two `get_details` + reason
Same two-step pattern, every matchup question

Add a verb when the use case demands it

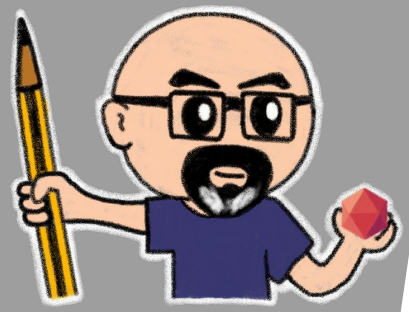
Use-case-driven expansion, not capability-driven
v1 exposed tables because it could
v2 adds verbs because callers ask



DevDays

3.5 Rewrite Prompt as a Recipe

A Prompt is a workflow, not a question



The Old `analyze_monster` Implementation

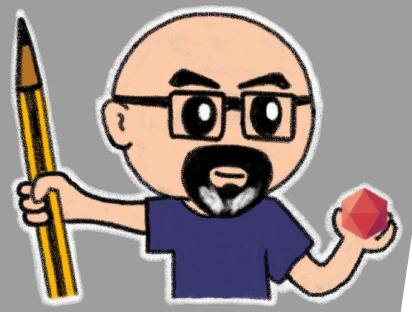
The body is essentially:

"Analyse this monster's profile, powers, weakness, and how you'd fight it."

That's a question, not a workflow

Name the Tools, the order, the output structure

The LLM follows the recipe and the output stops drifting

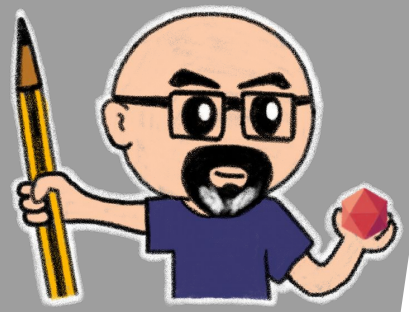


DevDays

3.6 Validate & Sanitize

Outputs from your MCP server are inputs to your LLM

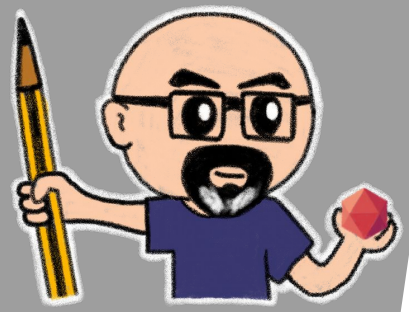
Treat them as untrusted data



Why Sanitize Outputs

You've Zod-validated every input since 3.0
The outputs? Raw DB rows. Read directly by the LLM

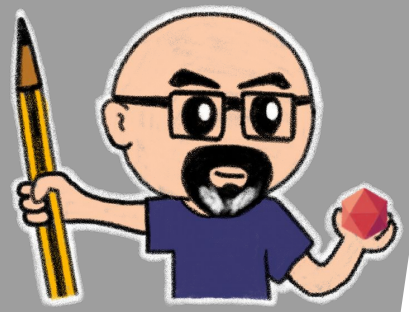
Outputs from your MCP server are inputs to your LLM
Treat them as untrusted data



DevDays

How to Sanitize Outputs

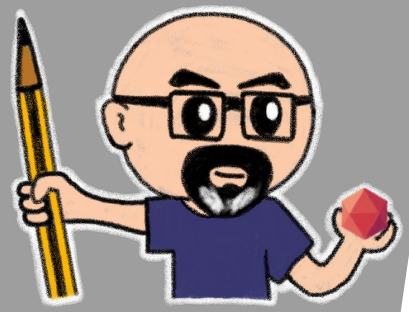
- Zod schemas on returns
- Redact internal fields
- Catch the prompt-injection that arrived through a DB column



DevDays

Bobby Tables Meets Billy Ignore

Bobby Tables (SQL injection)	Billy Ignore (output prompt injection)
input → SQL string → DB executes	DB string → tool response → LLM "executes"
fixed by parameterised queries (you did this in 3.0)	fixed by sanitising **outputs** before returning
25-year-old bug	new family — same shape, new layer



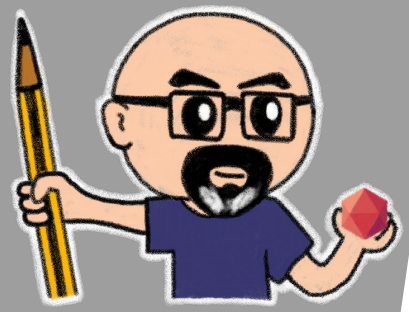
The output side is the side most servers still skip

Don't be most servers.

3.7 Auth is Not Optional

I'm running this over stdio, who would I authenticate?

Wrong question



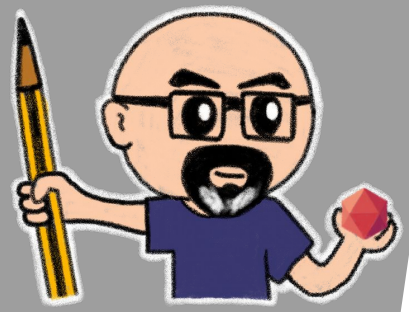
DevDays

Tool-level authorisation matters

Even when there's no connection to authenticate

There isn't one auth question, there are **three**:

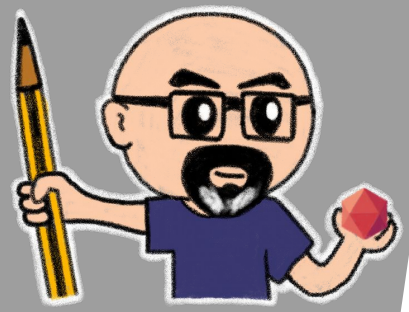
Layer	What it gates	Today
Connection	Who can connect to the server?	OAuth 2.1 + PKCE + RFC 8707 audience-binding (spec; **out of scope for stdio**)
Tool-level	Who can call which Tools/Resources/Prompts?	**3.7 — this iteration**
Data-level	Who can see which rows?	beyond today — row-level filters per principal



3.8 Test what the LLM actually does

Unit tests catch when your code is wrong

Golden tasks catch when the LLM uses your code wrong

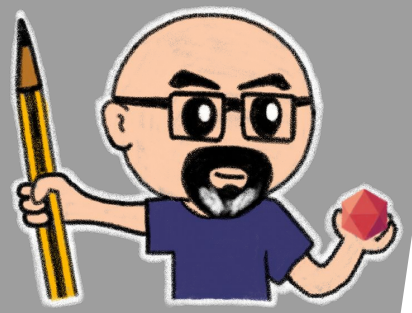


DevDays

Golden tasks, in code

tests/golden-tasks.ts

```ts
const TASKS: GoldenTask[] = [
  {
    name: "list elementals",
    prompt: "List Elemental monsters.",
    expected_calls: [
      { primitive: "tool", name: "search_monsters_by_category",
        args_match: /Elemental/ },
    ],
  },
  {
    name: "compare two",
    prompt: "Who wins, Thunderclaw or Aquafrost?",
    expected_calls: [
      { primitive: "tool", name: "compare_monsters",
        args_match: /Thunderclaw.*Aquafrost/ },
    ],
  },
];
```



Demo trick: rename `compare_monsters`'s description to nonsense and re-run: **a task fails, without any code change**

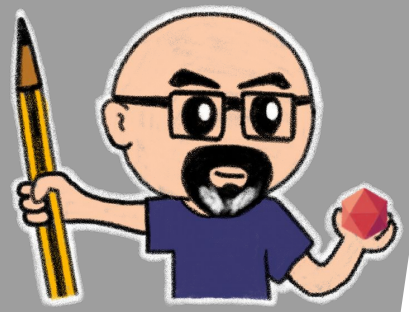
The LLM uses descriptions. So do your tests.

DevDays

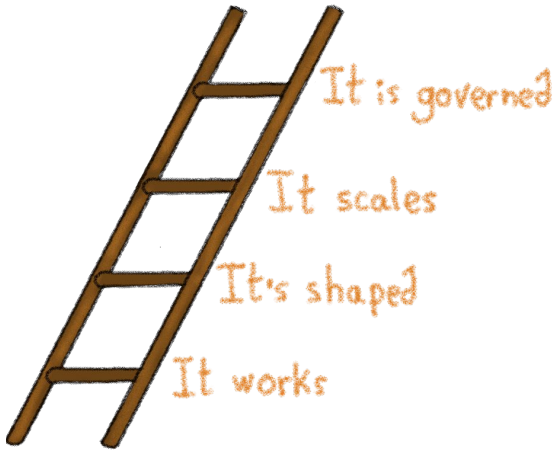
So Our Server Is Now Shaped

- Every primitive used deliberately
- Every input validated, every output scrubbed
- Every tool description written with intent
- Tested against what the LLM actually does

A single server, production-aware from day one

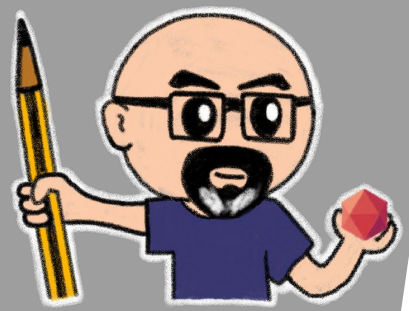


DevDays



Third rung: "it scales"

When MCP servers don't stay in their perimeter



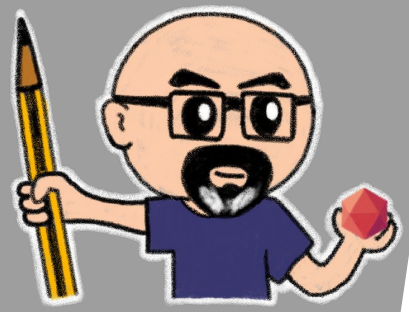
DevDays

Where we are

- **v1** - MCP works ✅ done
- **v2** - MCP is shaped ✅ done
- **v3** - **MCP scales** ← right now
- **v4** - MCP is governed

Same v2 server, don't touch it

Compose it

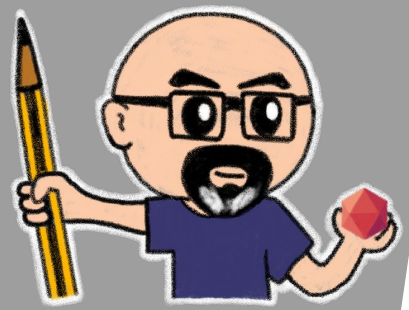


DevDays

What "Scales" Means

- Every boundary made **explicit**
- Auth, discovery, contracts, traces, retries
- Because the caller is an **LLM**
- And the topology is now **plural**

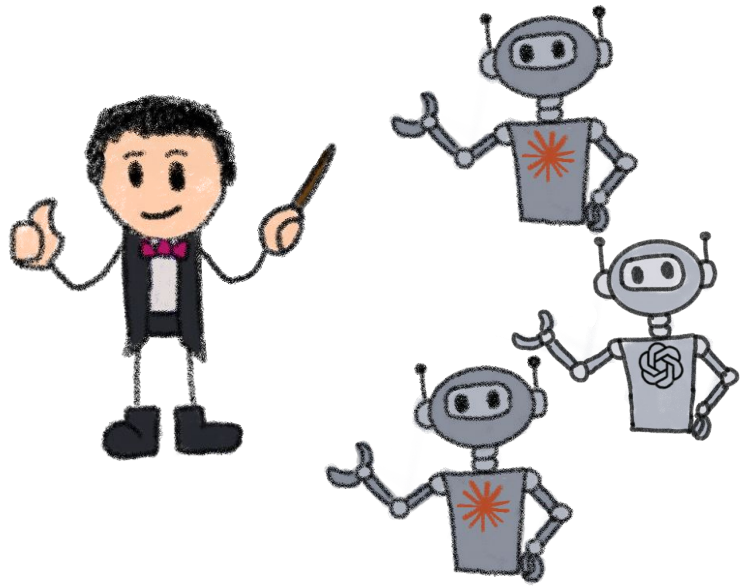
A scaled server is safe to live next to others



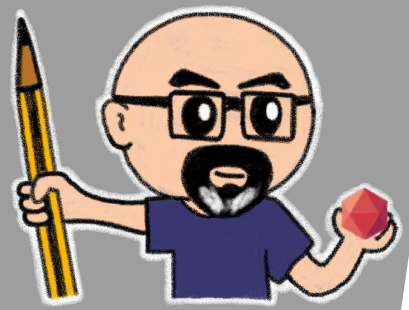
DevDays

Three Forces That Create Multiple Servers

- **Domain separation**
  Billing vs infra vs support
- **Trust separation**
  Read-only vs write, prod vs staging
- **Ownership separation**
  Teams, lifecycle, deploy cadence



These forces are inevitable as adoption grows

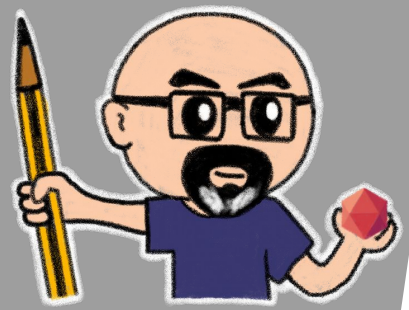


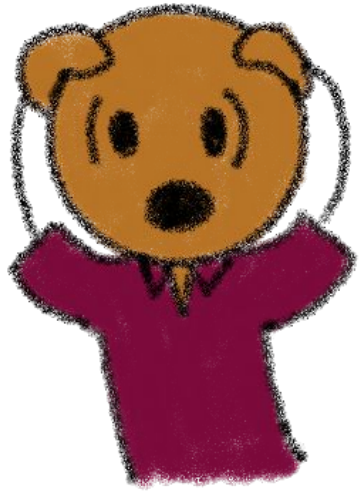
DevDays

The Reality: You Don't Have One MCP Server

- IDE agent, chat agent, internal agent, CI agent…
  - Different access
  - Different latency
  - Different blast radius

- Example: Engineering team alone might need:
  - Code search MCP (Cursor)
  - Deployment MCP (CI agent)
  - Incident MCP (on-call chat agent)





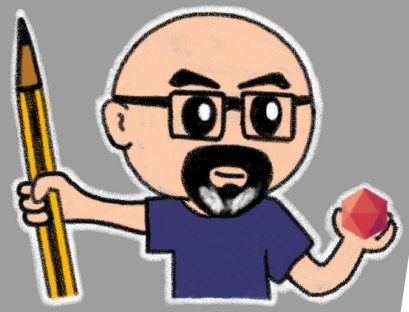
DevDays

History Rhymes — REST Taught Us This

- 2008–2015
  Monolith APIs → microservices
- Same pressures
  Domain, trust, ownership
- Same lesson
  One mega-API doesn't scale organizationally

MCP in 2026 ≈ REST APIs in 2010

We can learn from that journey

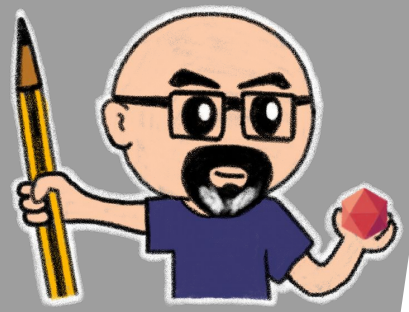


Anti-Pattern: The Mega-Server

One MCP server to rule them all

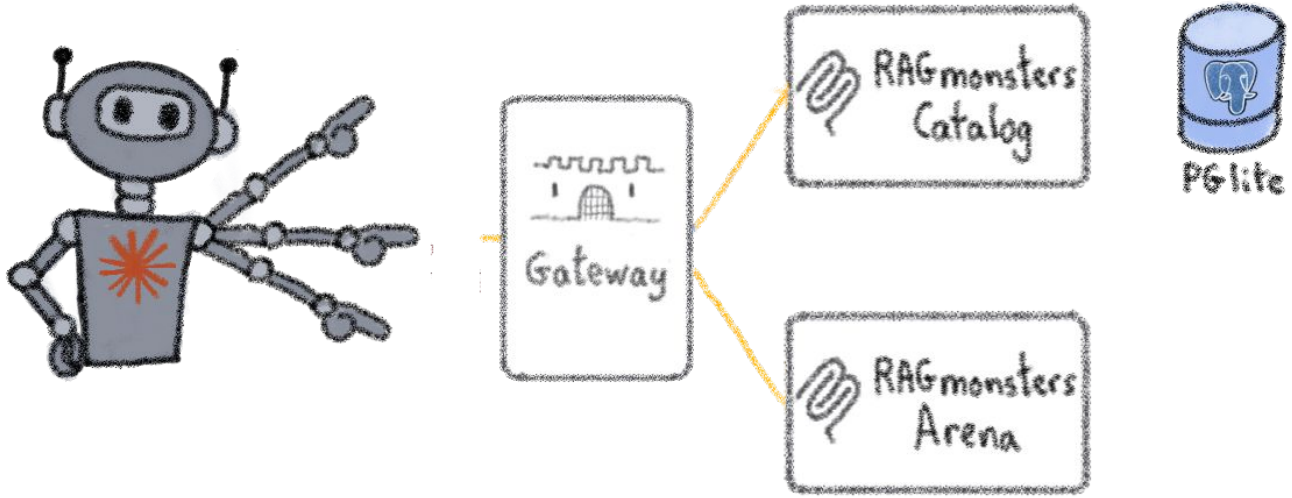
Consequences:

- **Too many tools**
  LLM confusion, token bloat
- **Unclear security policies**
  Who can call what?
- **Brittle deployments**
  One change breaks everything
- **Ownership diffusion**
  Nobody owns it, everybody blames it

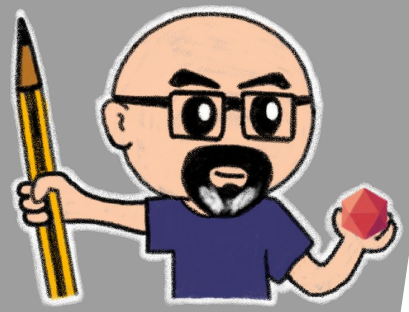


DevDays

What we're about to build



- **4.0:** Add arena. Two MCPs, no gateway. Feel the friction.
- **4.1:** Name the pain (observation pass).
- **4.2:** Build the gateway: one entry, one audit.
- **4.3:** Tools are contracts. Watch one break.
- **4.4:** Idempotency. One row written, two calls logged.



You won't touch the v2 catalog. Not one line.

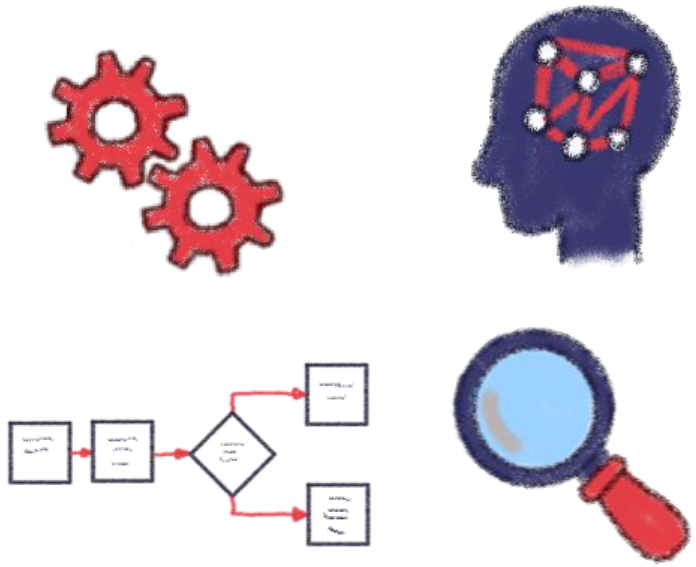
DevDays

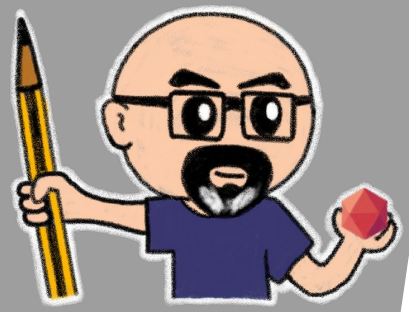
A Mental Model

MCP servers are an API surface for agents

Treat them like **products**:

- Auth
- Discovery
- Gateways
- Contracts
- Traces
- Reliability





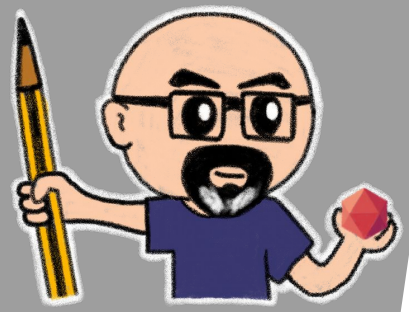
This framing guides the rest of Part 3

Naming and Namespacing

- Tool naming conventions that scale
- Pattern: `domain.verb_noun`

  ```
  billing.create_invoice
  support.search_tickets
  inventory.get_stock_level
  ```

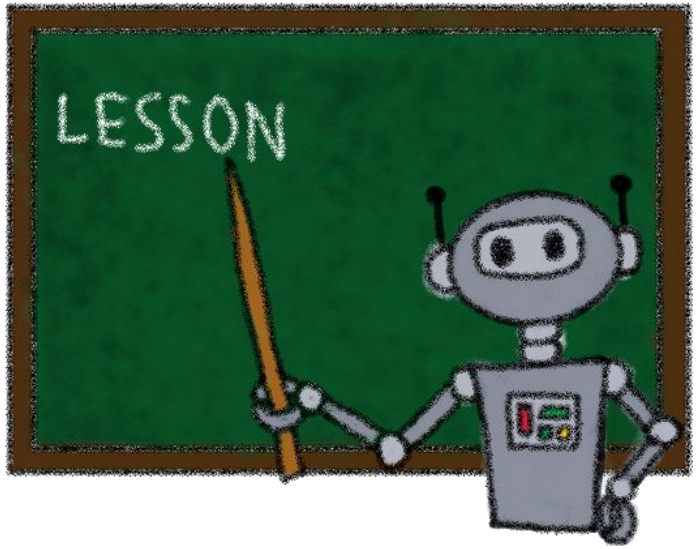
- Avoid collisions across servers
- Keep intent readable for LLMs
- Anti-pattern (meaningless to agents): `doThing, process, handle`

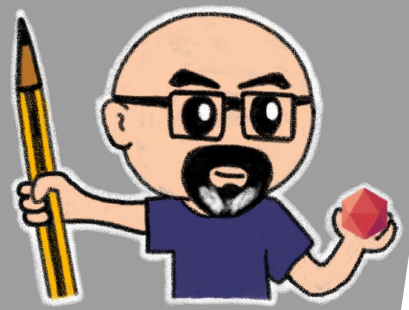


DevDays

Composition Patterns

How multiple MCP servers work together

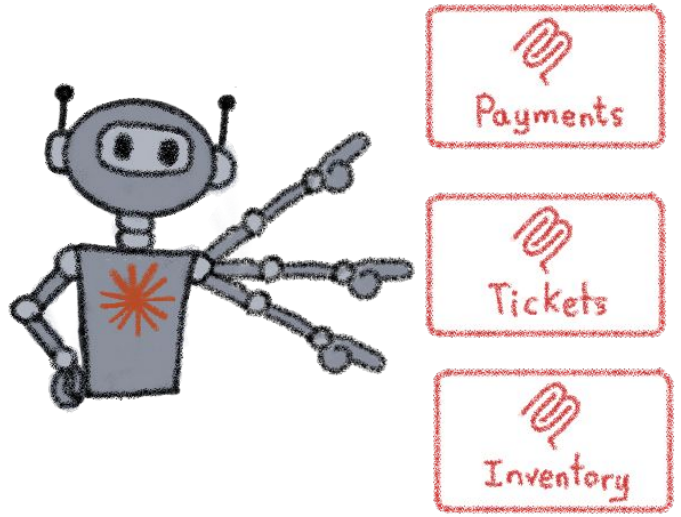


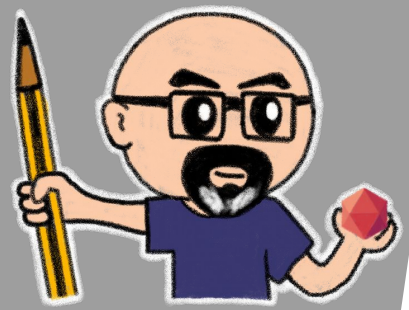


DevDays

Pattern 1 – Domain Servers

- One server per domain capability
- Clear ownership and narrow tool sets
- **Pros**
  - Clean boundaries
  - Independent deployment
  - Focused security
- **Cons**
  - LLM must know which server to call

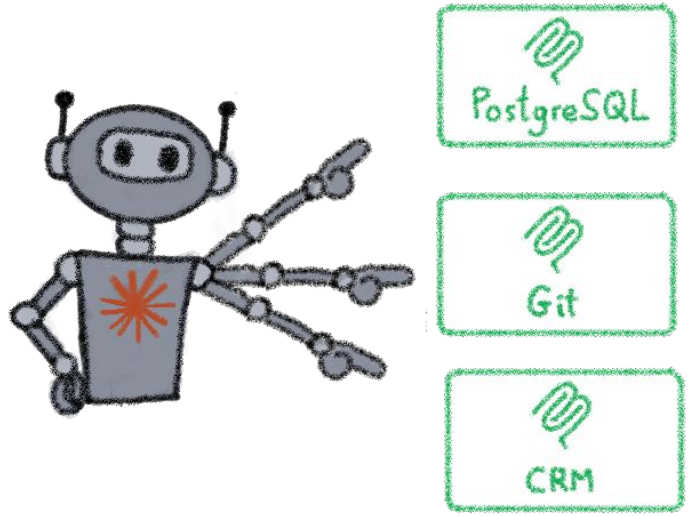




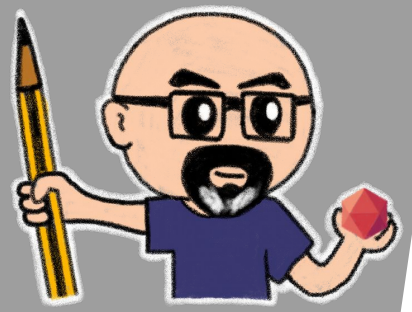
DevDays

Pattern 2 — Data-Source Servers

- Generic servers wrapping data sources
- Useful internally
  For prototyping, for technical users
- **Pros**
  Fast to set up, flexible
- **Cons**
  Often needs domain layer on top for production

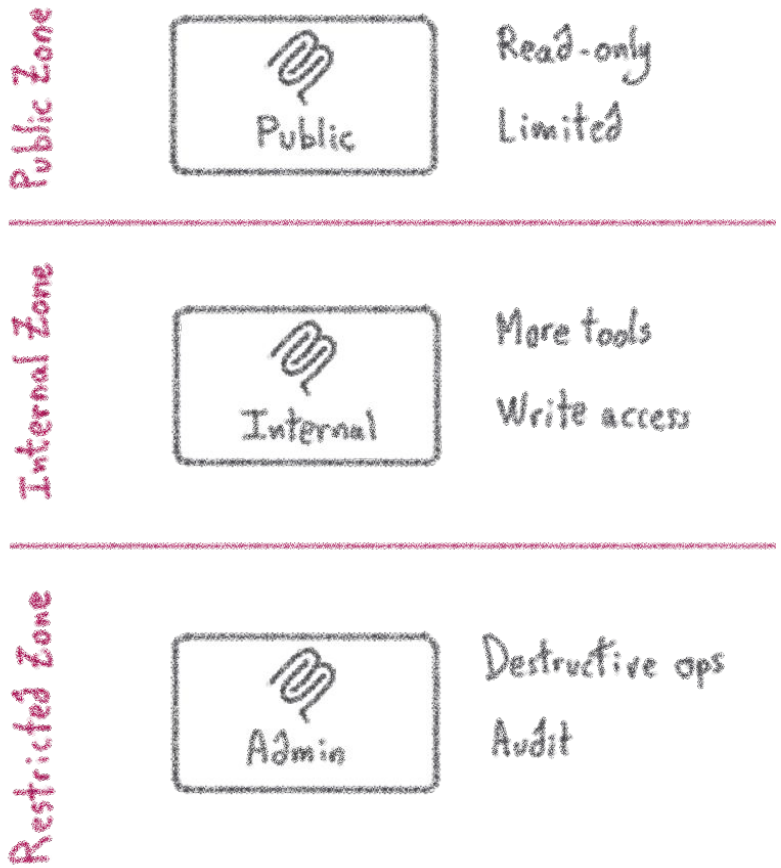


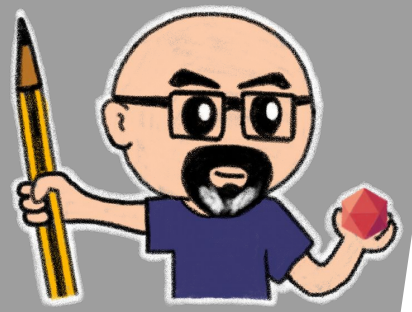
Remember RAGmonsters:
generic → custom as you mature



Pattern 3 — Trust-Zone Servers

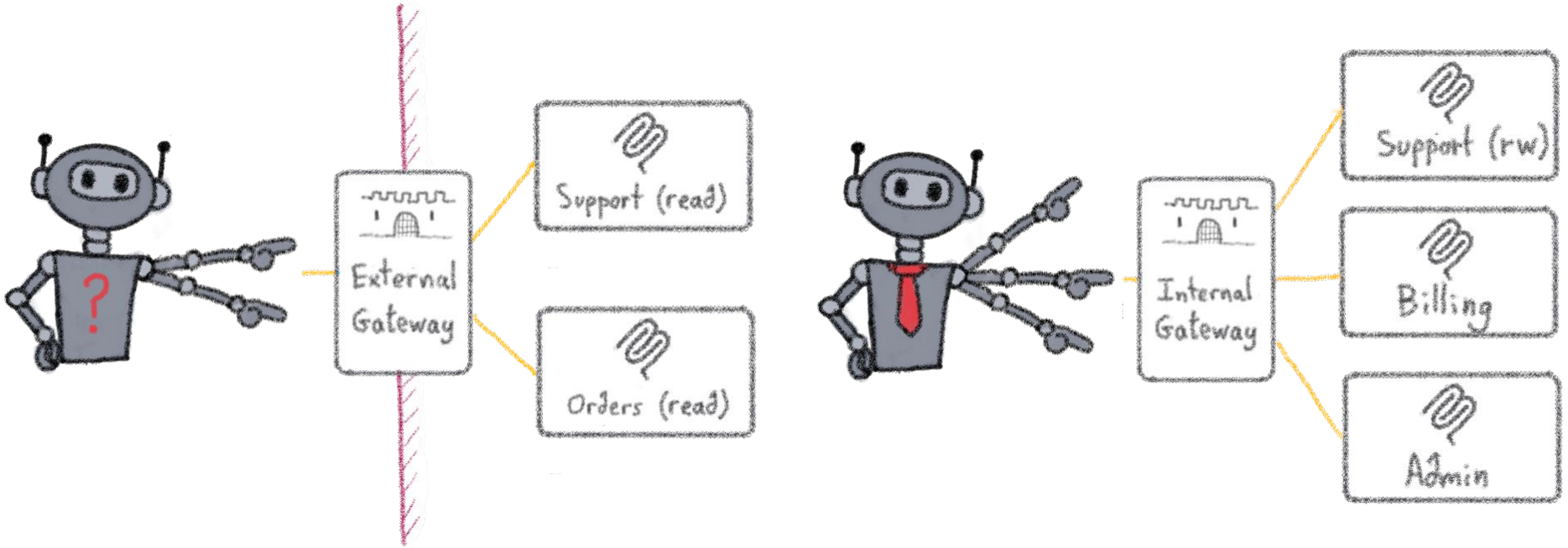
- Separate networks/credentials
  Not just code paths
- Maps to existing infrastructure security zones
- When to use
  - Compliance requirements
  - Multi-tenant
  - External-facing agents





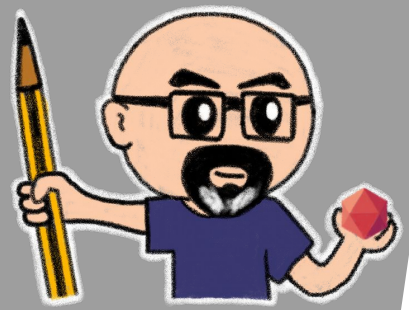
DevDays

Combining Patterns



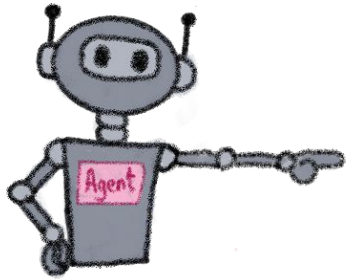
Domain × Trust = your actual architecture

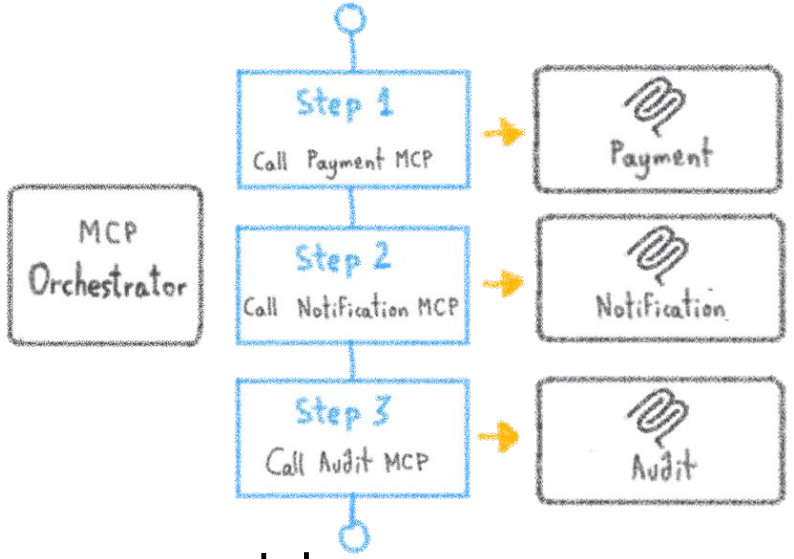
Most organizations end up with a matrix



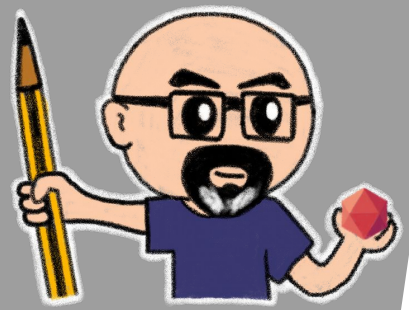
Orchestrator Pattern (When Needed)

- Not every client can chain tools well
- Orchestrator composes multi-step workflows server-side
- When to use:
  - Shared workflows
  - Less capable clients
  - Compliance requirements
- Warning:
  You risk rebuilding "agent logic" on server side





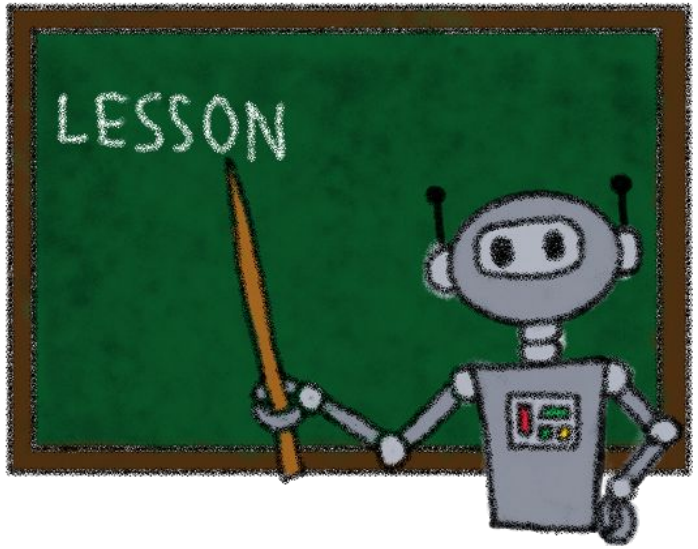
Keep orchestrator thin, don't duplicate LLM reasoning

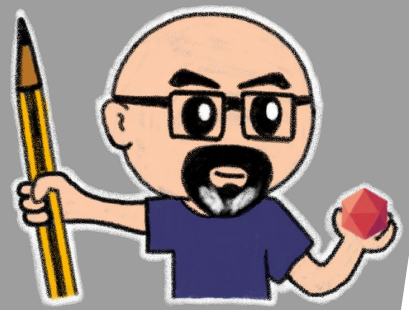


DevDays

Discovery becomes a policy problem

Where agents find what they're allowed to use?

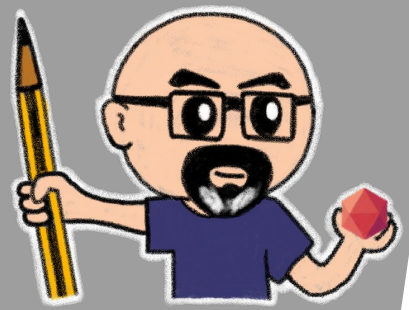




The LLM reached for a well-known server name

It pulled a pirate clone from the public internet

Because the LLM chose it

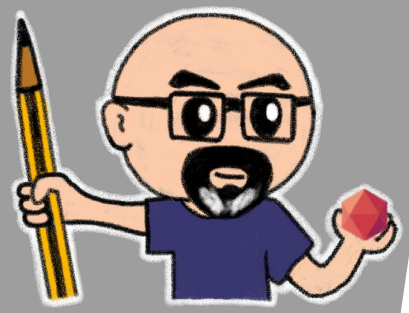


DevDays

The Registry Landscape

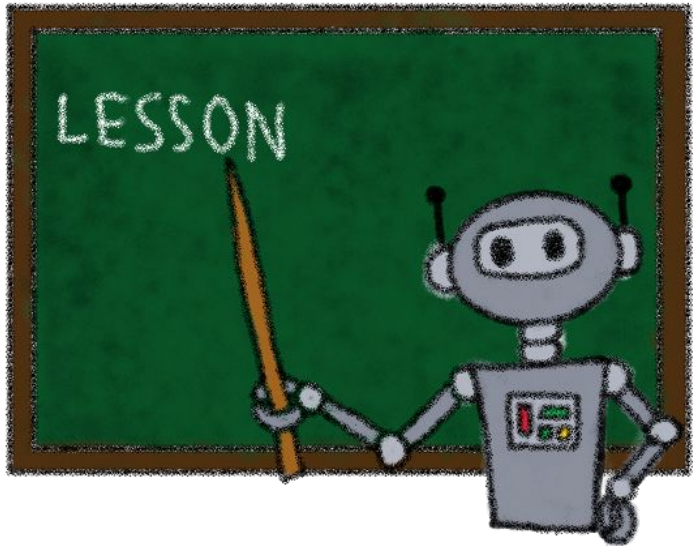
- **Official MCP Registry**
  Preview, metadata only
- **GitHub MCP Registry**
  Copilot's discovery home
- **Azure API Center, Kong MCP Registry**
  Enterprise
- **VS Code custom registry URLs**
  Private / internal

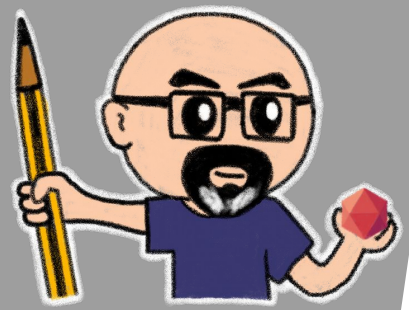
Random-from-internet is no longer a default



The gateway layer shows up

Auth, audit, rate-limit… at one place

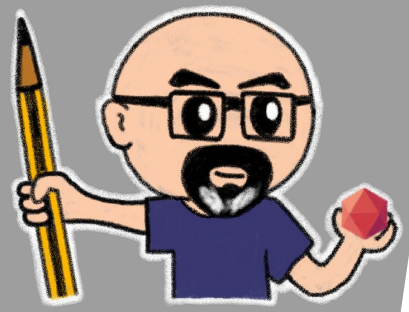




Every server reinvented its own auth

Every server reinvented its own logging, audit, rate-limits

The same wheel, poured six times

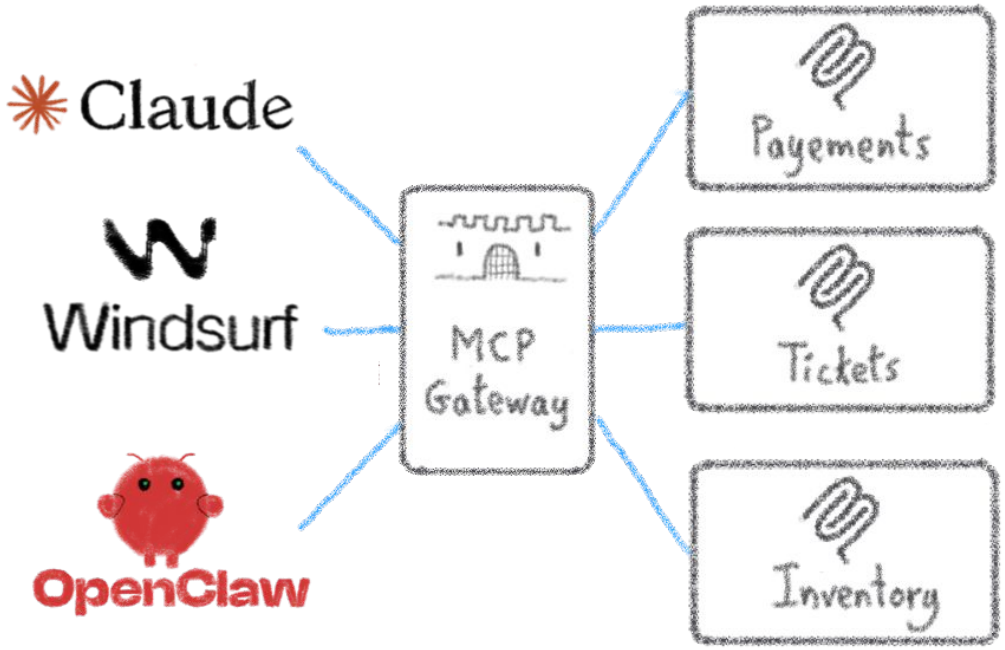


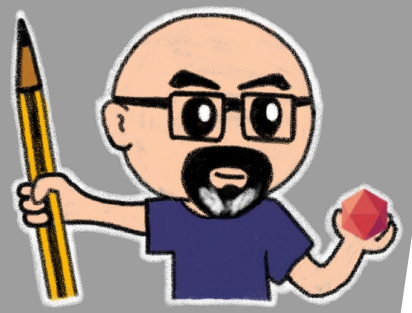
DevDays

What A Gateway Does

Single endpoint for all clients

- **Auth termination**
  One place, one story
- **Audit hook**
  Emits events,
  doesn't retain them (yet)
- **Rate limiting**
  Per-caller, per-tool
- **Policy enforcement**
  Allowlist backed by registry
- *Retention, compliance, legal: we'll get there in Part IV*



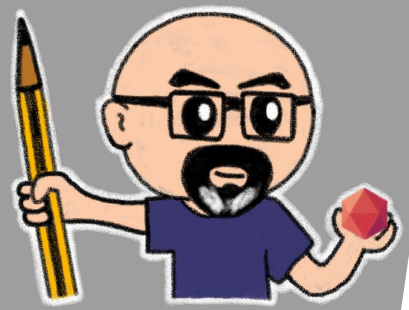


DevDays

Open-Source Gateways Worth Watching

- **Solo.io** `agentgateway`
- **Agentic Community** `mcp-gateway-registry`
  Keycloak / Entra
- `mcp-proxy`
  multiple implementations
- **Kong OSS**
  MCP-aware adapters landing

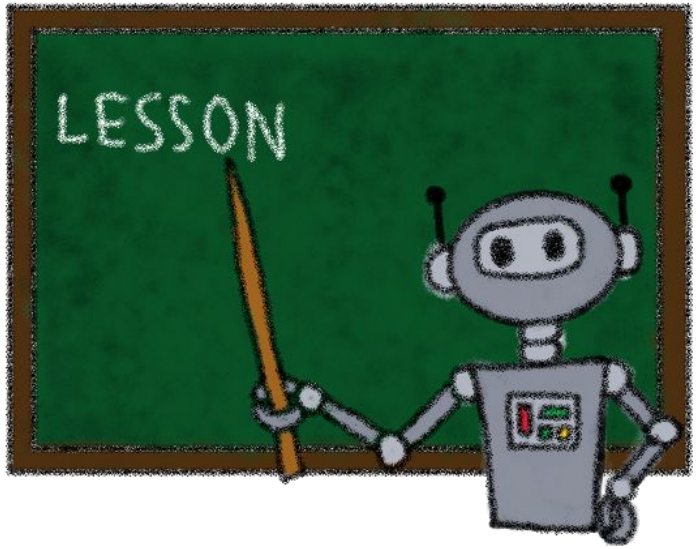
Direction of travel, verify specifics before you ship

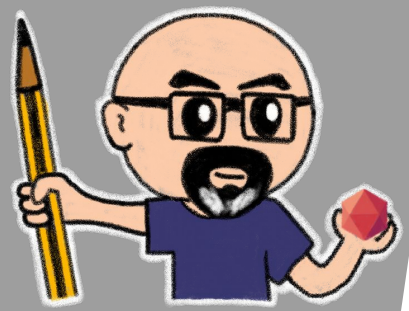


DevDays

Contracts between servers

Tool schemas are your public API

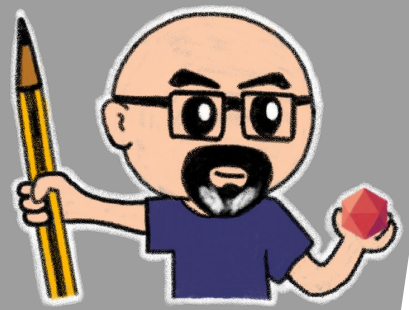




The schema evolved, the canonical client still worked

The other three clients broke at 2am

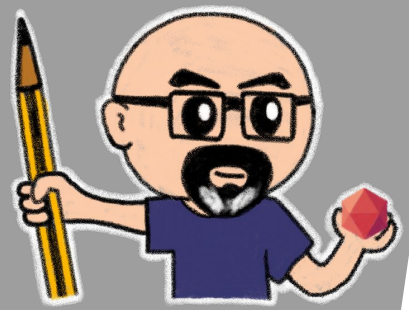
Nobody had checked them

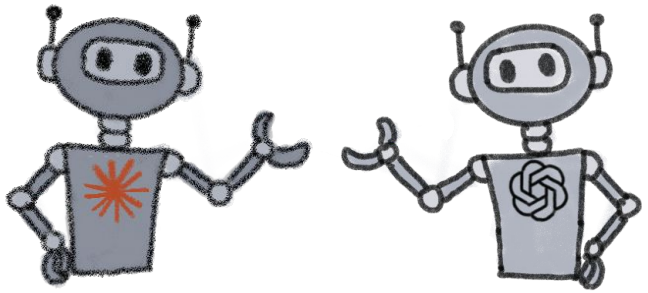


DevDays

Tools Are Contracts

- Tool schemas **are** the public API
- Clients (agents) depend on:
  - Tool name
  - Parameter names and types
  - Output shape
  - Behavior/semantics
- Breaking changes hurt more than REST because agents fail weirdly
  - No compiler error, just confused behavior

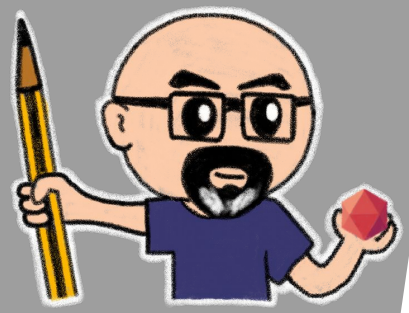




DevDays

What Counts as Breaking?

Change	Breaking?	Why
Rename tool	✅ Yes	Agents can't find it
Rename parameter	✅ Yes	Calls fail silently
Remove parameter	✅ Yes	Old calls break
Change output shape	✅ Yes	Agent parsing fails
Change semantic meaning	✅ Yes	Agent logic breaks
Add optional parameter	❌ No	Old calls still work
Add output field	❌ No	Agents ignore unknown



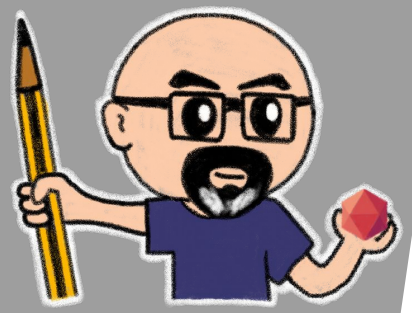
DevDays

Semantic Versioning for MCP Servers

```
server-name@1.2.3
             | | |
             | | └── Patch: bug fixes, no interface
change
             | └──── Minor: new tools, new optional
params
             └────── Major: breaking changes
```

- Expose version in server metadata
- Clients can pin to major version

REST lesson: Version early, version explicitly

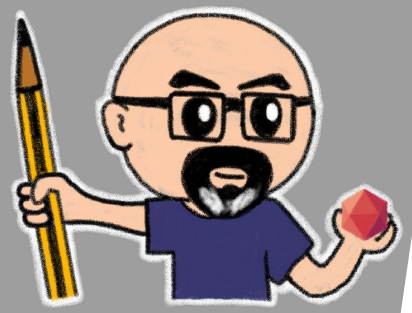


Compatibility Strategy

- **Prefer additive changes:** New tools > modified tools
- **Deprecation period:** Keep old tools for one release cycle
- **Deprecation visibility:** Surface via `resource://deprecations`

```
{
  "deprecated": [
    {
      "tool": "get_monster",
      "replacement": "get_monster_by_id",
      "removal_version": "2.0.0",
      "reason": "Ambiguous name"
    }
  ]
}
```

- **Migration guides:** Document how to move to new tools



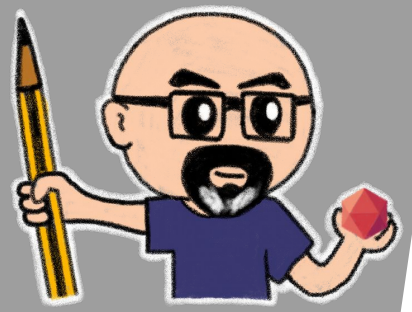
DevDays

Versioned Prompts and Resources

- Prompts are **"behavior contracts"**
  They guide LLM reasoning
- Resources are **"schema contracts"**
  They define data shapes
- **Version** them **explicitly**:

```
prompt://analyze_monster@v2
resource://schema@v1
```

- Allows **gradual migration**
  Without breaking existing clients



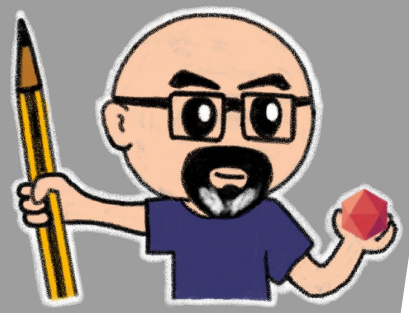
DevDays

Client Matrix Testing

Your server is called by multiple clients

Client	Version	Capabilities

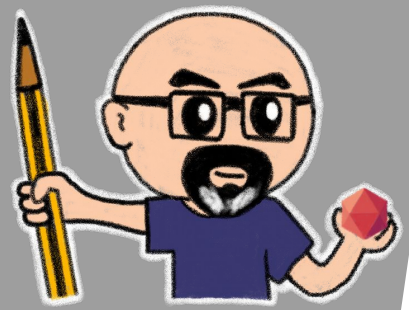
- Maintain a client matrix
- Basic smoke tests per client type
- Know what breaks when you change something



The Principle — "Don't Surprise the Agent"

- Stability > cleverness
- Predictable structure wins
- Agents build mental models of your tools
- Changing behavior without changing signature = worst case

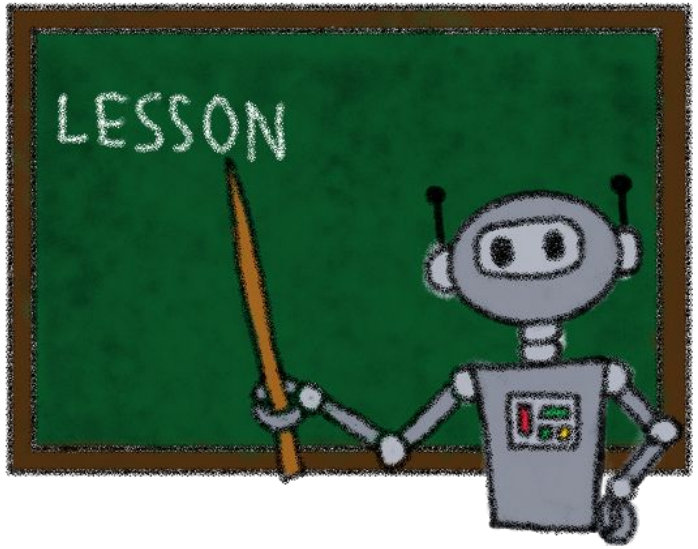
If you must break, break loudly

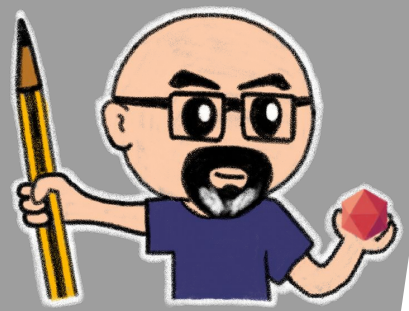


DevDays

The LLM retries. A lot

Designing for a caller that loops

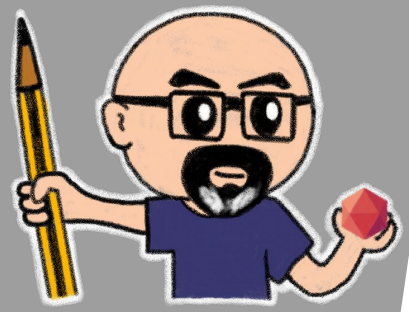




The server returned an error code, nothing else

The LLM had no context, it retried

A thousand calls. 100,000 tokens. Per hour



DevDays

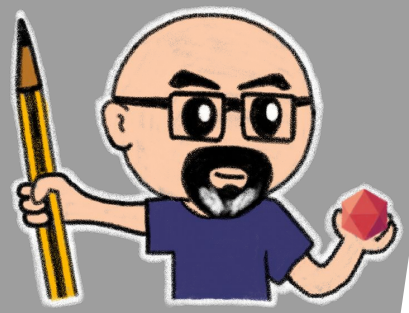
Idempotency For The LLM Era

REST solved this with **idempotency keys**, same pattern

The wrinkle: LLMs retry **semantically**, not just on HTTP errors

- *"I didn't see a confirmation, let me try again"*
- *"That output didn't look complete, let me retry"*
- Exponential backoff doesn't stop semantic retries

A tool that costs money must be safe to call twice



DevDays

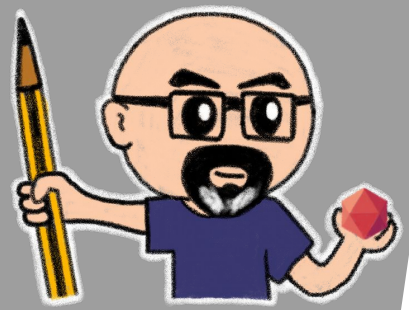
Circuit Breakers, The LLM Edition

Circuit breakers on failing dependencies. Yes, same as always

The circuit breaker's signal must be **legible to the model**

- `503` with no body → LLM learns nothing, retries
- `"rate-limited, back off 30s"` → LLM updates its plan

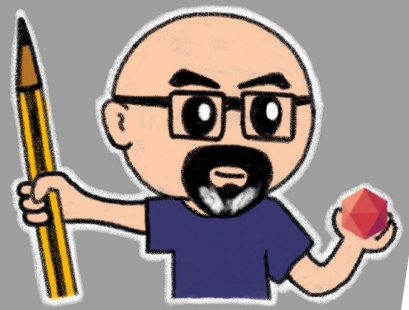
Error **messages** are part of the reliability contract



Hard Limits + Per-Caller Quotas

- Per-caller rate limits (the LLM is a caller)
- Hard caps on **tool invocation rate** per session
- **Loud timeouts**, fail explicitly, don't let the LLM assume
- Refuse politely in the tool response, so the model adapts

The LLM will loop 200 times. Your database will not

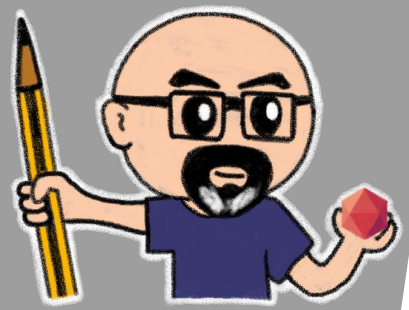


DevDays

Our MCP Now Scales

- Auth is audience-bound
- Discovery runs through a curated registry
- Traffic flows through a gateway
- Contracts are versioned across consumers
- Traces correlate across instances
- Retries don't storm the database

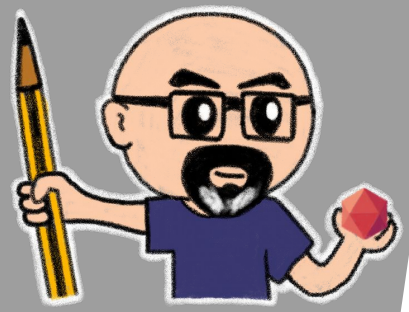
A system that's safe to live next to others



4.0 Add the arena

Composition happens at the agent layer

The LLM is the orchestrator

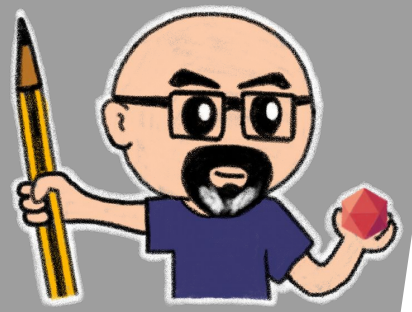


Let's add a fully independent server

Let's add a fully independent `arena` server

- It does not know what monsters exist
- It does not connect to the database
- It takes a monster detail payloads pair as arguments and returns a winner

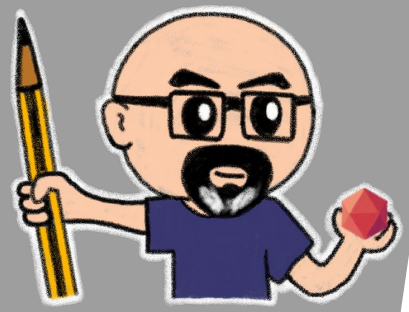
The LLM is responsible for stitching the two servers together



4.1 Where it hurts

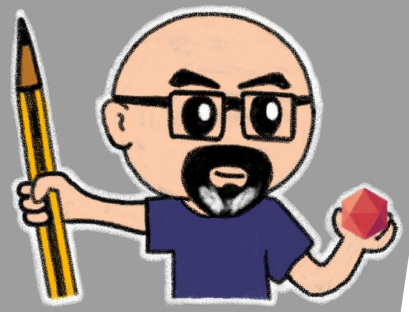
Nothing here is broken, but everything scales badly

That's what a gateway fixes

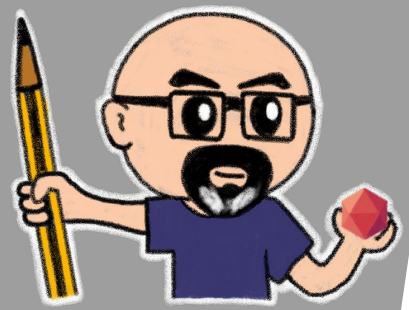


The four pains

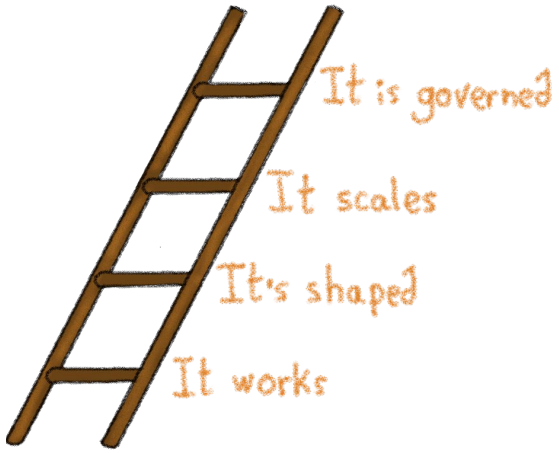
- **Surface bloat**
  6 tools × 2 servers in `/mcp`
- **Duplicated config**
  `MCP_PRINCIPAL` per server in `/mcp`
- **Fragmented audit**
  Per-server log files
- **Name collisions**
  Two tools called `compare_monsters` is undefined behaviour
- **No fleet-wide policy**
  Each server enforces its own rules



DevDays

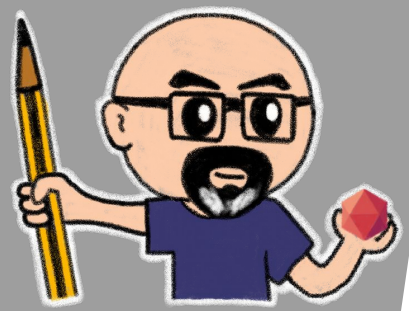


DevDays



Third rung: "it scales"

When MCP servers don't stay in their perimeter



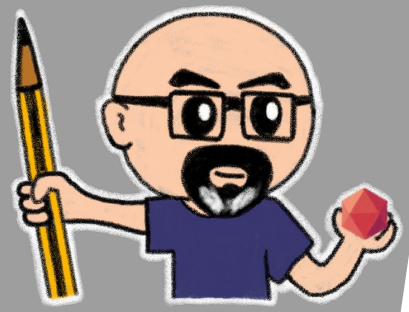
DevDays

Where we are

- **v1** - MCP works ✅ done
- **v2** - MCP is shaped ✅ done
- **v3** - MCP scales ✅ done
- **v4** - **MCP is governed** ← right now

If the agent deletes production,

whose name is on the incident report?

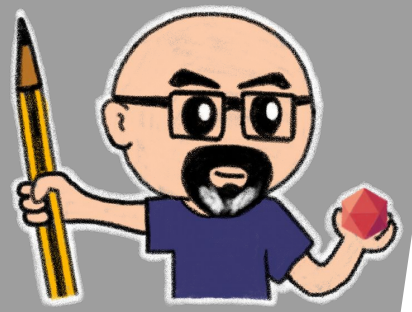


DevDays

What "Governed" Means

- Blast radius **bounded**
- Audit trail **retained**
- Cost **attributed**
- Protocol choices **deliberate**
- Ownership **named**

Every invocation accountable

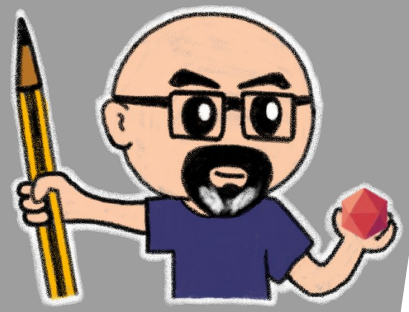


DevDays

We've seen this movie before

Payments went through this
Health data went through this
Banks went through this

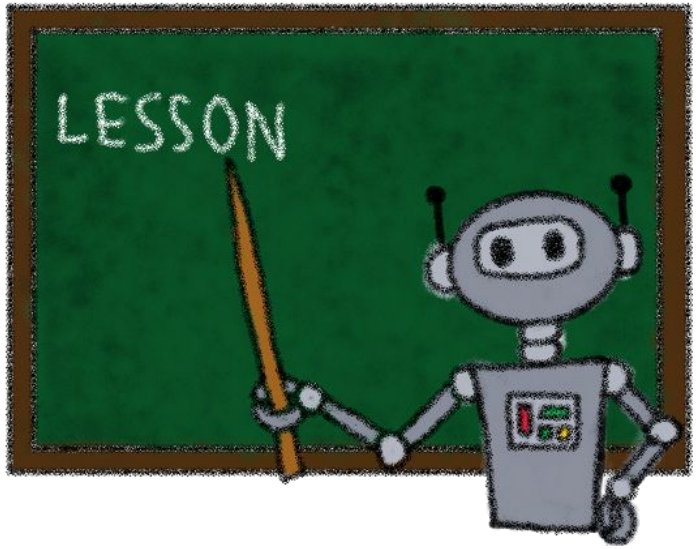
MCP is the new regulated-API surface

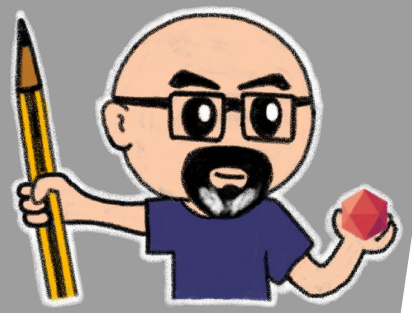


DevDays

Blast radius

The lethal trifecta, and what it means for MCP



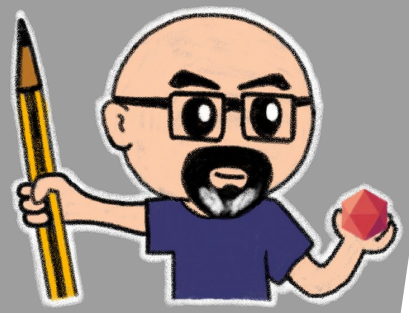


Remember the ALTER TABLE

That was my toy database, I lost test data

Now imagine your production cluster

That was blast radius with a benign user
Now add a malicious one



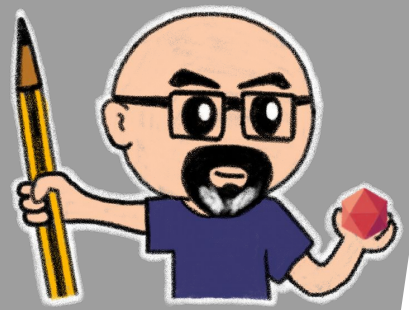
DevDays

The Lethal Trifecta

Simon Willison's framing

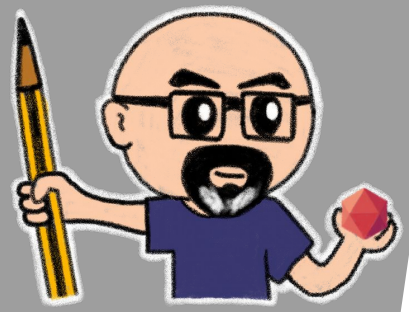
Three ingredients, together:

1. **Private data** the agent can read
2. **Untrusted input** that can instruct the agent
3. **Exfiltration vector** the agent can call



Any agent with all three
is unconditionally vulnerable
to indirect prompt injection

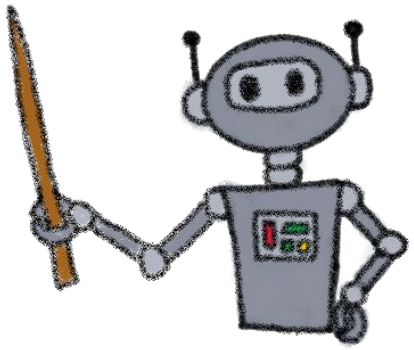
No system-prompt hardening fixes it

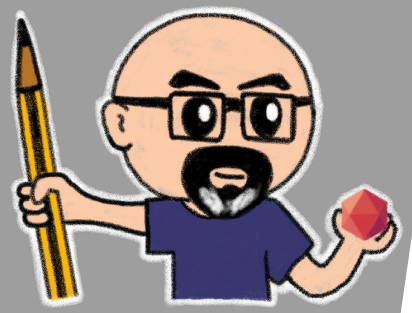


DevDays

Power without blast-radius awareness

is just a bigger incident

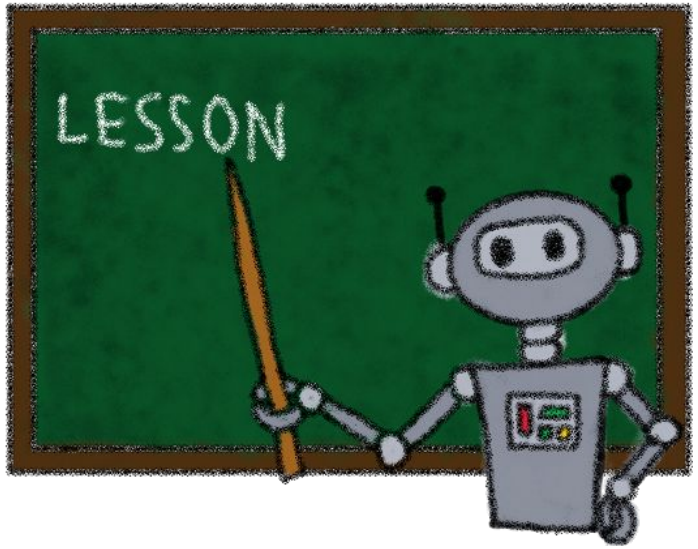


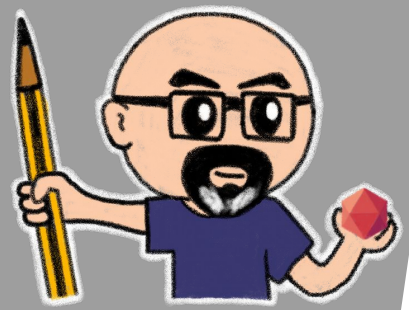


DevDays

What 2026 taught us

Incidents, CVEs, scanners — the year MCP got real



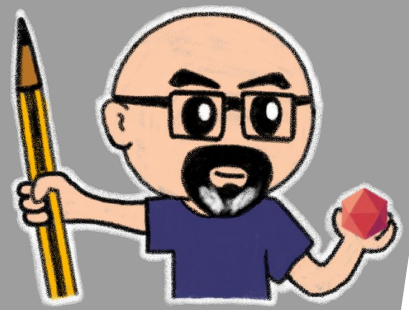


DevDays

GitHub MCP — Frame 1

An attacker files a public issue

- Title and body look harmless to a human reviewer
- Body carries **hidden instructions** crafted for the LLM
- Issue sits in a public repo, waiting to be read

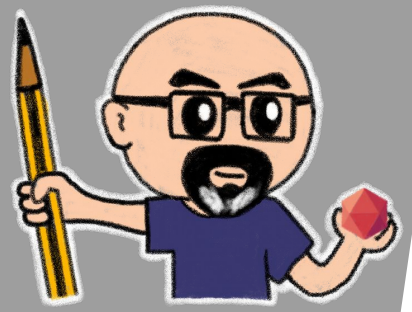


DevDays

GitHub MCP — Frame 2

An agent with private-repo access reads the issue

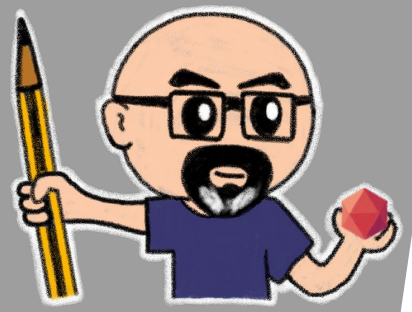
- Agent treats embedded instructions as its own directive
- No system-prompt hardening catches this
- This is the lethal trifecta, live



GitHub MCP – Frame 3

The agent opens a PR containing private-repo contents

- Attacker reads the PR
- Private data is now public
- The canonical case study for the trifecta



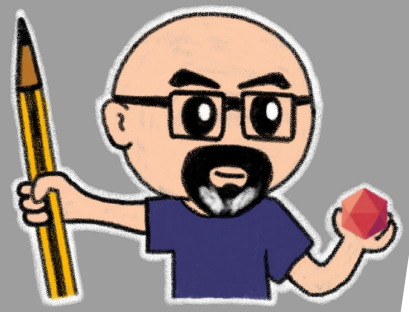
DevDays

The CVE Wave

Early 2026:

- **Wave of CVEs** against MCP servers, clients, infras
- Tool poisoning, path traversal, RCE via prompt injection
- **Many public MCP servers accept unauthenticated calls**

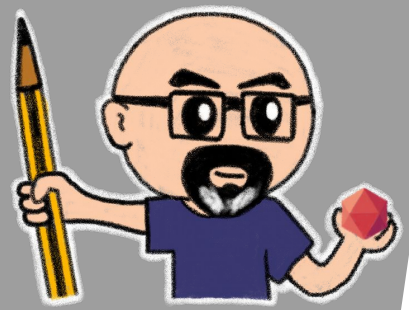
MCP just joined the *"we have CVEs now"* industry



OWASP MCP Top 10

- OWASP now has an **MCP Top 10** project
- Covers tool poisoning, prompt injection, trust-boundary failures
- **Emerging**, don't promise it's final

Direction of travel

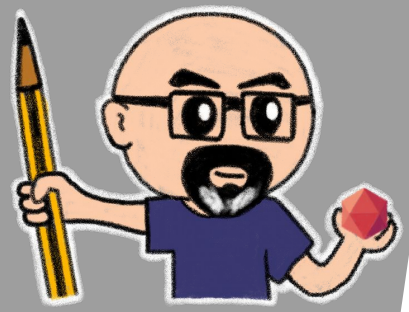


`mcp-scan`: The Scanning Answer

Invariant Labs' `mcp-scan`, the de facto scanner

Detects:

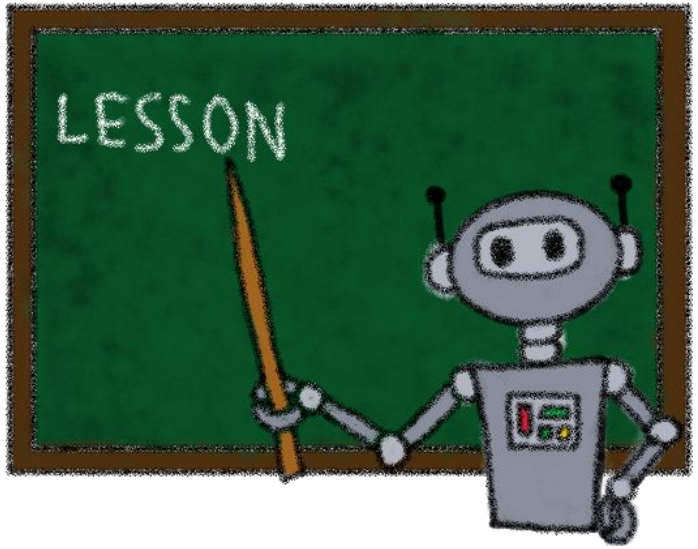
- **Tool poisoning**
  Malicious instructions in descriptions
- **Rug pulls**
  Server changes tool description after trust
- **Cross-origin escalations**
  Shadowing attacks that compromise trusted tools
- **Prompt injection in metadata**
  Malicious instructions contained within tool descriptions

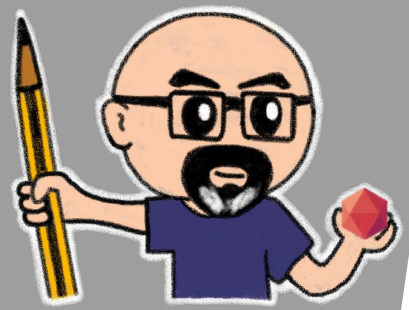


DevDays

Risk Tiering

Tools with a blast radius



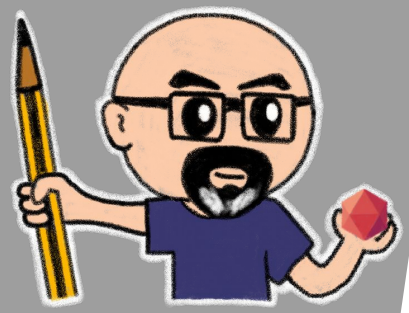


DevDays

Risk-Tier Your Tools

- Tag every tool with its tier
- Apply controls systematically

Tier	Description	Examples	Controls
0	Safe reads	`list_types, get_schema`	None
1	Sensitive reads	`get_customer, search_orders`	Auth required
2	Writes	`create_invoice, update_record`	Auth + logging
3	Destructive / money / security	`delete_account, transfer_funds`	Auth + approval + audit



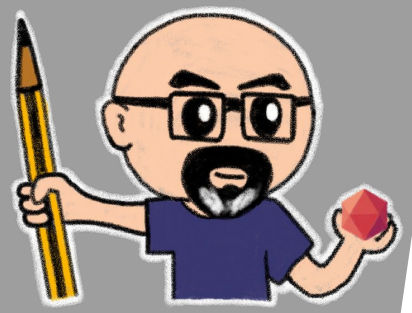
DevDays

Approval Gates

- Human-in-the-loop for Tier 2/3 operations
- Pattern: Two-step commit
  Agent can plan freely; execution requires confirmation

```
Step 1: plan_change(params) → Returns preview, no
side effects
Step 2: apply_change(plan_id) → Executes, requires
approval
```

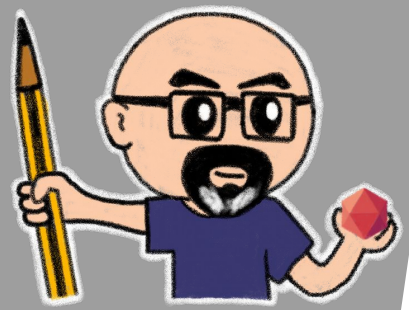
- Async approval workflow:
  Slack notification, approval UI



Autonomy for exploration, gates for action

A tool is destructive

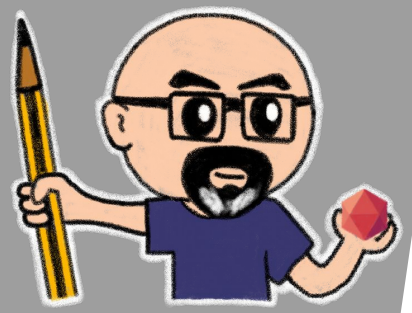
until proven otherwise



Code Example

RAGmonsters MCP

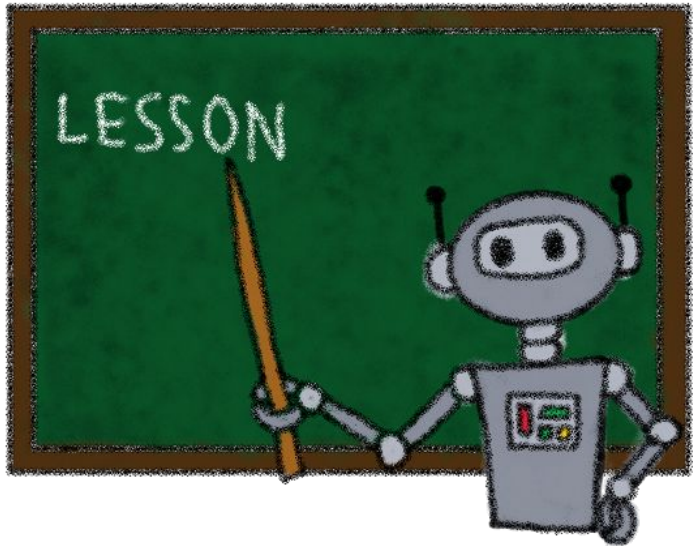
```
server.tool("delete_monster", {
 _risk: "destructive",          // ← tier declared
 monster_id: z.string()
}, async ({ monster_id }, { confirm }) => {
 if (!confirm) throw new Error("confirmation required");
 // ...
});
```

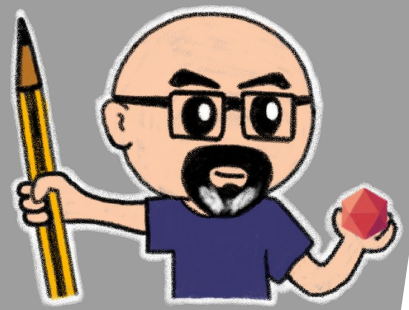


Tier in metadata, enforcement at invocation

Who did what, for how much

Retaining the audit trail the gateway emits





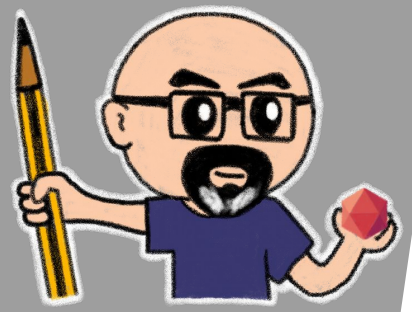
DevDays

Audit Trail Retention

The gateway emits, v4 **retains, indexes, queries**

- **SIEM integration**
  Your security team already has one
- **Retention windows**
  Legal/compliance decides, not engineering
- **Immutable**
  Write-once, queryable, exportable

Emission is easy, retention is policy



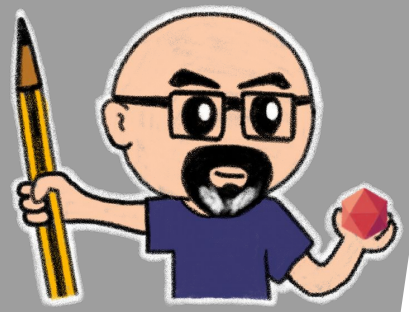
DevDays

Compliance Hooks: The Honest Gap

Most of this is **not** in the spec today

- It's the glue you build around MCP
- All custom:
  Retention windows, SIEM formats, legal attestations
- **MCP gives you the emission; the rest is on you**

Honesty before the close: this layer isn't solved yet



DevDays

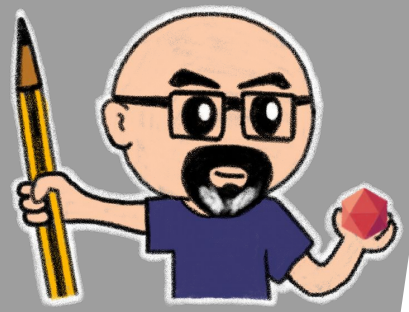
Per-Caller Cost Attribution

- Which **agent**?
- Which **team**?
- Which **budget**?

Tokens burn money

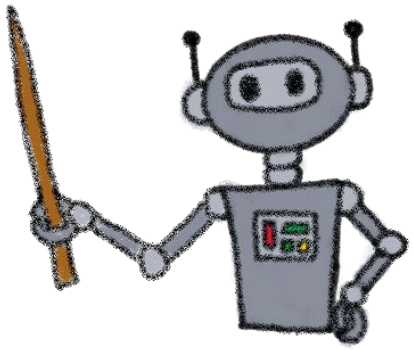
Without attribution, the bill is a mystery

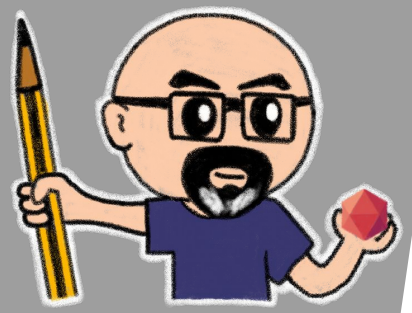
The agent did this, the agent's team pays



Your audit trail

is your alibi

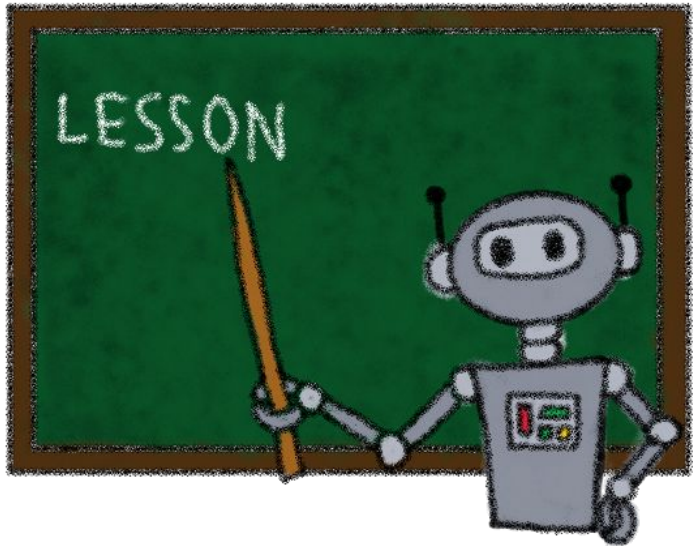


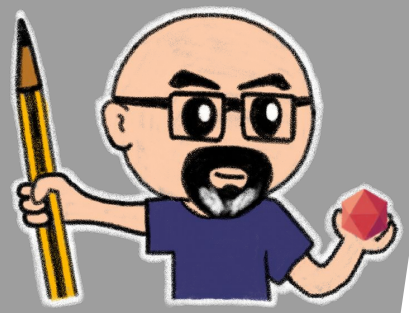


DevDays

MCP is not alone

The emerging agent-protocol stack

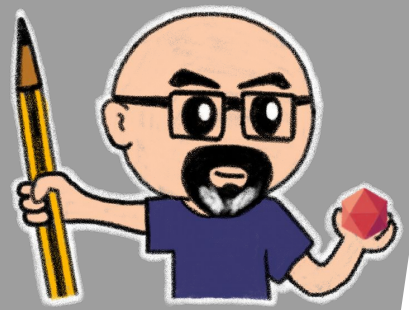




The Emerging Stack

- **MCP**: tool / data connectivity (LLM ↔ systems)
- **A2A**: agent-to-agent communication
- **Other protocols**: UI, payments, identity

Google's March 2026 dev guide places them as siblings

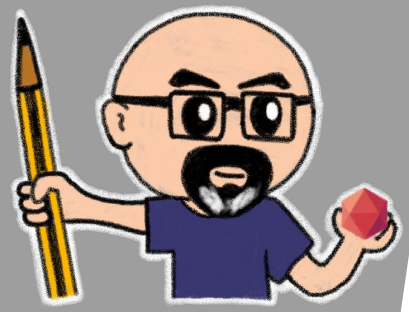


DevDays

When To Use What

- **MCP**: LLM reaches out to a system
- **A2A**: agent talks to another agent
- **Don't force MCP** to be a universal transport

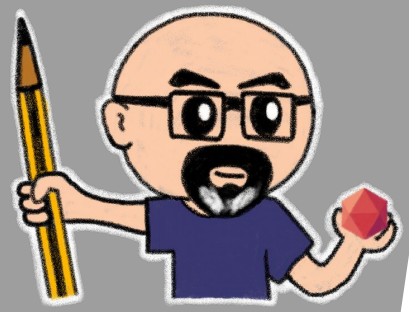
My recommendation: pick the layer, not the one hammer



Worth watching

the protocol landscape is settling fast

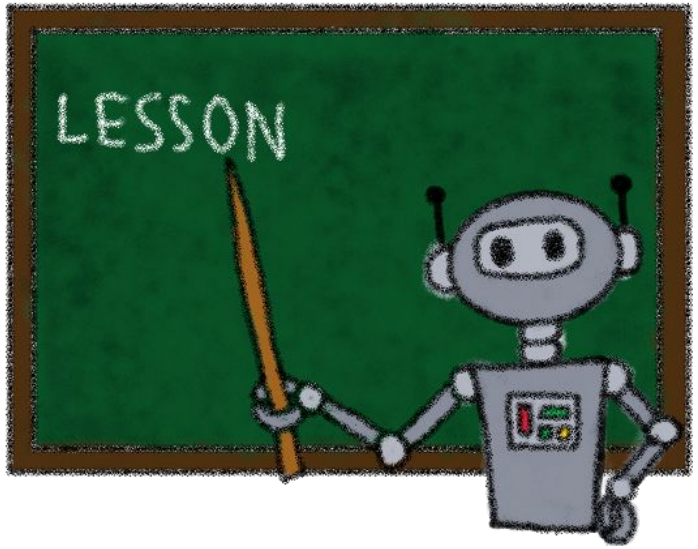
Google's guide is a frame, not a verdict

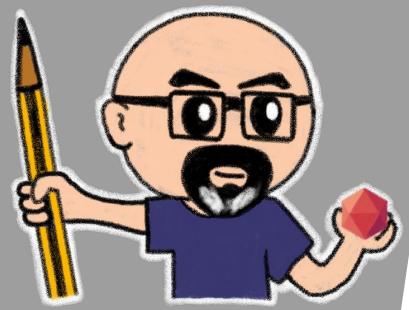


DevDays

Who owns MCP in your organisation?

The emerging platform discipline





DevDays

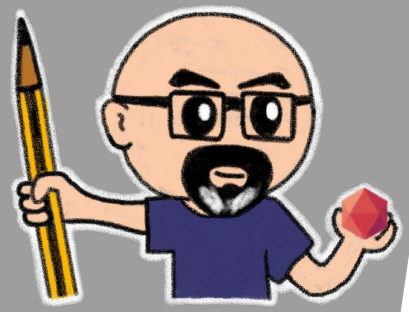
The Emerging Discipline

MCP platform teams: new role

Owners of:

- Allowlist curation
- Registry operations
- Gateway + audit pipeline
- Policy-as-code

Analogous to Kubernetes, CI, API platform team



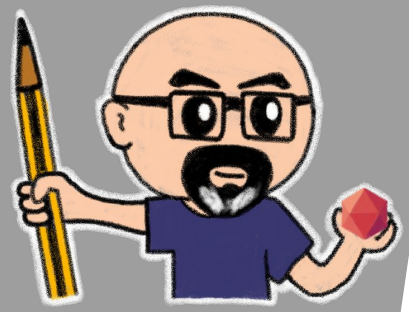
DevDays

Review + Approval Flow

Before a server reaches the allowlist:

- **Threat review**: trifecta check
- **Risk-tier classification**: safe / write / destructive
- **Audit hook verification**: events emitting?
- **Ownership named**: who carries the pager?

Policy-as-code where possible

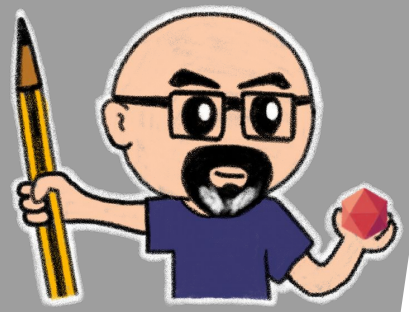


DevDays

A governed MCP server

- Blast radius **bounded**
- Audit trail **retained**
- Cost **attributed**
- Protocol choice **deliberate**
- Ownership **named**
- Everything **reviewable**

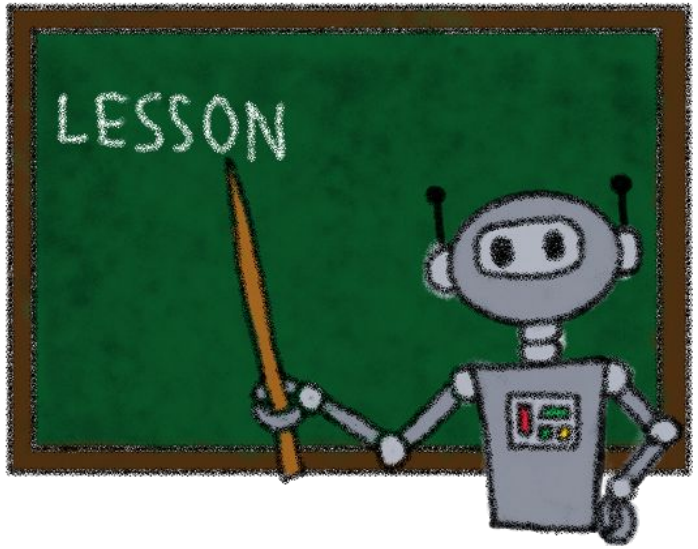
A system the organisation trusts

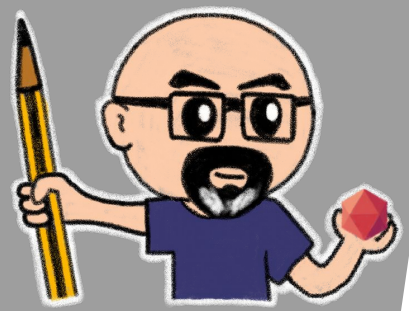


DevDays

Reliability and Cost Controls

Agents are relentless, your infrastructure must cope

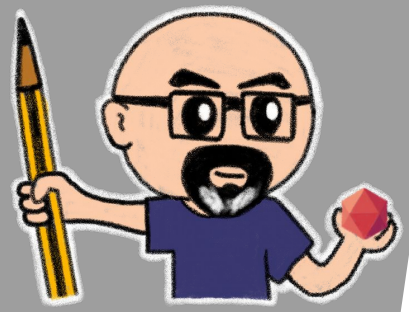




Latency Budgets for Tool Calls

- **Agents feel slow** fast
- Users waiting for agent = users **waiting for your MCP** server
- Measure and alert on **latency**
- Set **targets by tool category**:

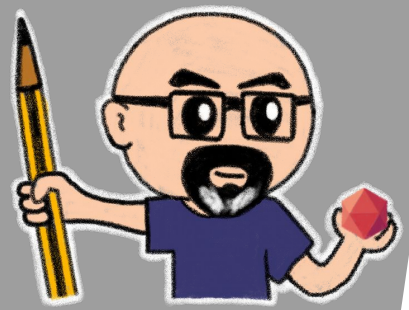
Category	Example	p95 Target
	`get_monster_by_id`	
	`search_monsters`	
	`create_monster`	
	`generate_report`	



DevDays

Timeouts, Retries, and Circuit Breakers

- Timeouts:
  Don't let slow calls block
  agents forever
- Retries:
  Only for idempotent operations
  (reads, idempotent writes)
- Circuit breakers:
  Prevent meltdown loops
  when downstream fails



REST lesson: These patterns are proven, apply them

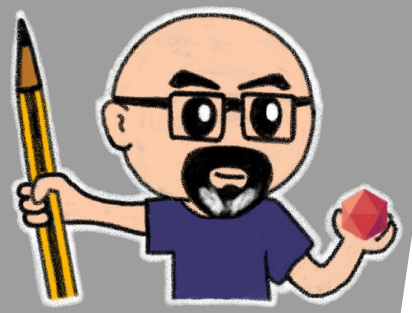
DevDays

Idempotency Keys for Write Tools

- **Problem:**
  Agents repeat themselves (retries, loops, confusion)
- **Solution:**
  Make "create" safe to retry

Tool: create_invoice

```
async function createInvoice(params: {
  idempotency_key: string;  // Required for writes
  customer_id: string;
  amount: number;
}) {
  const existing = await db.findByIdempotencyKey(params.idempotency_key);
  if (existing) return existing; // Return same result, don't duplicate
  return await db.createInvoice(params);
}
```



DevDays

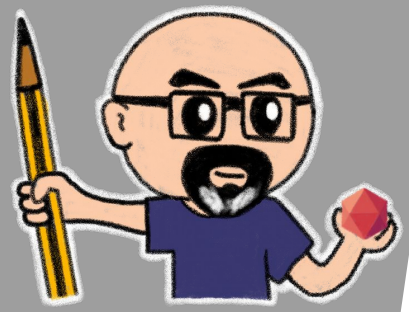
Hard Limits Everywhere

- Agents don't know when to stop
- Protect yourself with defaults

Limit	Default	Max

- Fail safely, explain clearly

```
// ✗ Return 10,000 rows, blow up context
// ✅ Return 50, include:
// "Showing 50 of 847. Use pagination for more."
```



DevDays

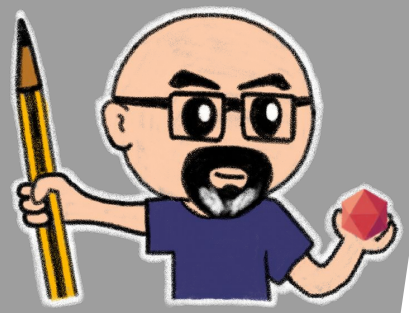
Token Efficiency Is an Architecture Concern

LLM context windows are finite and expensive

- Every byte you return costs tokens
- Patterns:
  - Return minimal fields by default
  - Provide fields or details parameter to opt-in
  - Structured data > prose descriptions
  - IDs + names > full objects

Token efficiency comparaison

```
// Default response (token-efficient)
{ "id": "m1", "name": "Pyrodrake", "type": "fire" }
// With details=true
{ "id": "m1", "name": "Pyrodrake", "type": "fire",
  "description": "...", "abilities": [...], "habitat": {...} }
```



DevDays

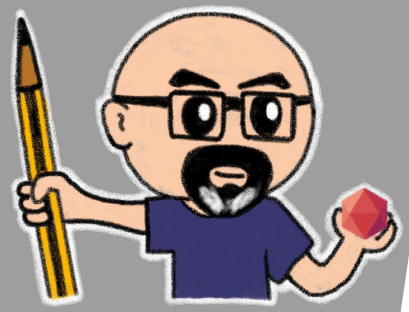
Cost Attribution

- You need to know: Who's spending? On what?
- Log "cost units" per tool call:

```
logger.info('Tool completed', {
  tool: 'search_monsters',
  user: session.user_id,
  team: session.team_id,
  agent: session.agent_type,
  cost_units: calculateCost(result),  // Your cost model
  latency_ms: elapsed
});
```

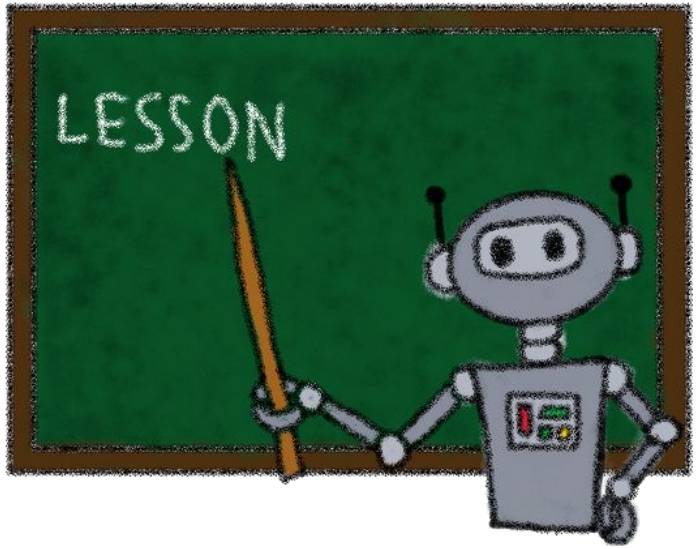
- Enables: Chargebacks, quota enforcement, optimization targeting

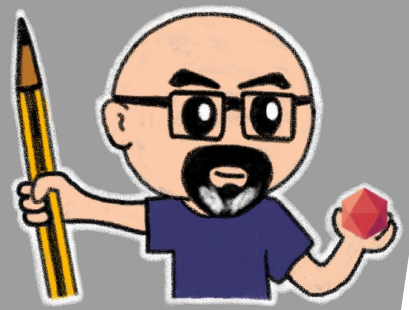
If you can't measure it, agents will break it silently



Safety Guardrails

Make the safe path easy, the risky path explicit





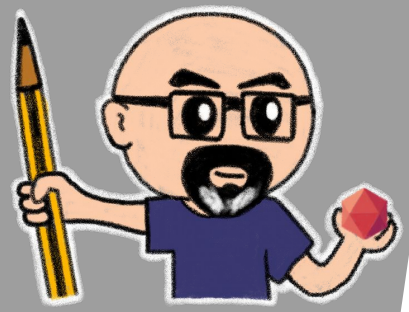
DevDays

Threat Model Update

At scale, new threats emerge:

- **Agent misuse:**
  Legitimate agent doing unintended things
- **Prompt injection:**
  Malicious input steering agent behavior
- **Over-broad capability:**
  Too many tools, unclear boundaries
- **Autonomous loops:**
  Agent calling tools repeatedly without oversight

"Security is no longer just about bad inputs



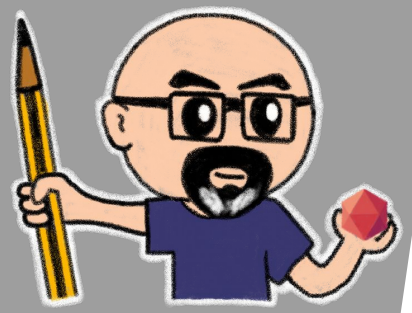
DevDays

Approval Gates

- Human-in-the-loop for Tier 2/3 operations
- Pattern: Two-step commit
  Agent can plan freely; execution requires confirmation

```
Step 1: plan_change(params) → Returns preview, no
side effects
Step 2: apply_change(plan_id) → Executes, requires
approval
```

- Async approval workflow
  Slack notification, approval UI



Autonomy for exploration, gates for action

DevDays

Policy as Code

- Central rules
  - Who can call what
  - With which limits
- Enforce in gateway or shared middleware
- Version controlled
- Auditable
- Consistent

Policy example

```
policies:
  - tool: "billing.*"
    allow:
      - role: billing_admin
      - role: finance_team
    deny:
      - agent_type: public_chat
  - tool: "*.delete_*"
    require:
      - approval: manager
       - audit: full
```

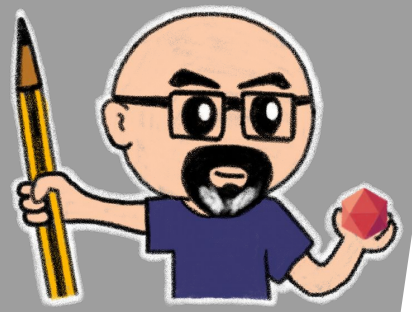
DevDays

Audit Trails

- Every tool call recorded
- Correlation ID links multi-tool workflows
- Redact sensitive values
- Retain for compliance period

Audit trail example

```
{
  "correlation_id": "req-abc-123",
  "timestamp": "2026-02-01T10:30:00Z",
  "tool": "billing.create_invoice",
  "user": "user-456",
  "agent": "finance-assistant",
  "params": {
    "customer_id": "c-789",
    "amount": "[REDACTED]"
  },
  "result": "success",
  "latency_ms": 234
}
```



Make incident review possible

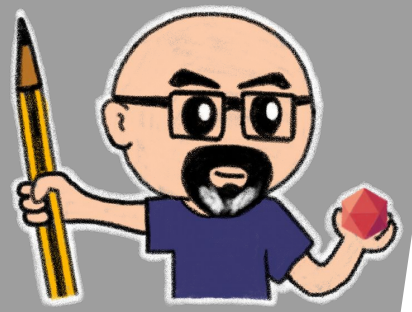
The Safety Principle

Two rules:

1. Make the safe path the easy path
   - Tier 0 tools: no friction
   - Good defaults everywhere

2. Make the risky path explicit and slow
   - Tier 3 tools: approval gates, audits, alerts
   - No "oops I didn't mean to delete that"

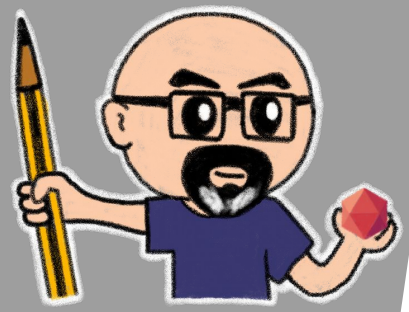
Safety and usability aren't opposites,
good design achieves both



DevDays

That's all, folks!

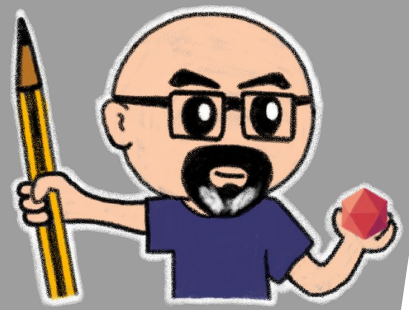
Thank you all!



DevDays

What We've Learned So Far

And how to go further

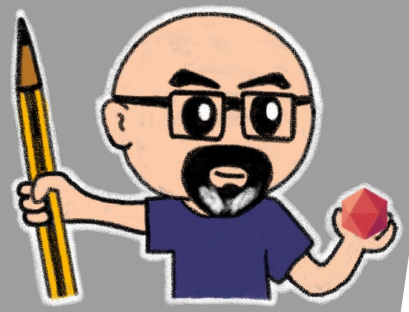


DevDays

The Part 3 Takeaway

Scaling MCP is mostly:

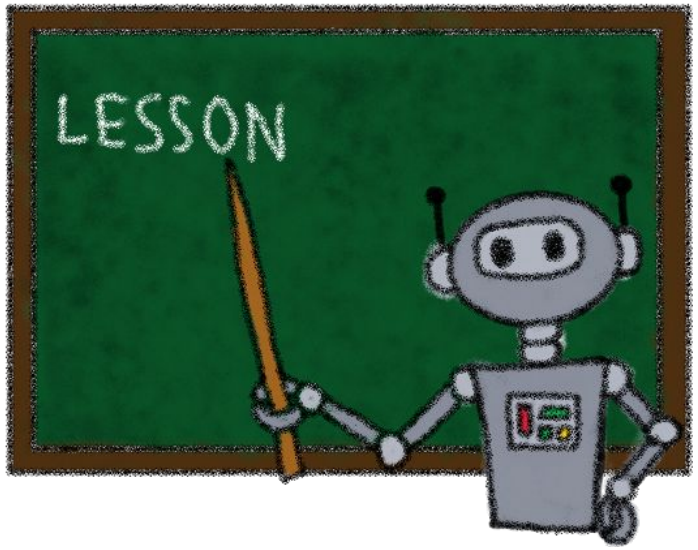
- **Composition:**
  Domain servers, gateways, orchestrators
- **Contracts:**
  Versioning, compatibility, "don't surprise the agent"
- **Controls:**
  Limits, idempotency, cost attribution, safety tiers

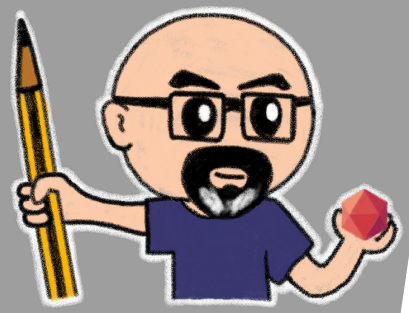


DevDays

One More Thing

A new shape: Code Mode





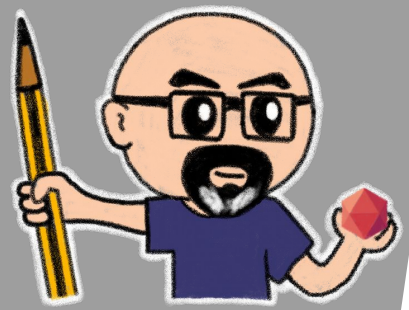
DevDays

The Problem Code Mode Solves

At scale, tool catalogs get huge

- 50 tools per server
- ~50k tokens of tool descriptions loaded per session
- The LLM spends context on navigation, not thinking

LLMs write code better than they navigate menus



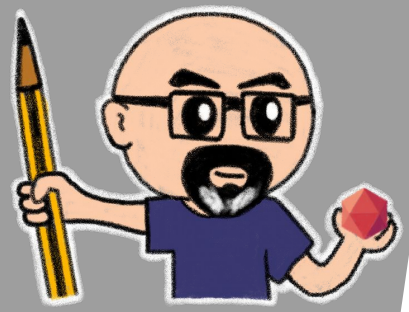
DevDays

Code Mode: An Emerging Pattern

Cloudflare published **Code Mode**

A different way to **compose primitives inside one server**

CLOUDFLARE



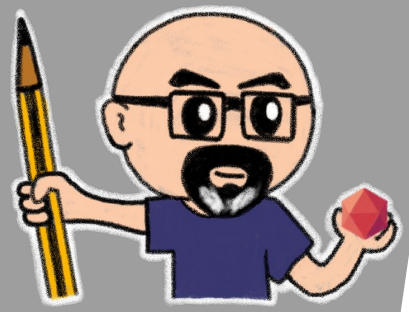
DevDays

Search → Execute → Code

1. **Search**: semantic search finds relevant capabilities
2. **Execute**: code-execution env runs generated code
3. **Code**: LLM writes a program that uses tools as a library

Example: Clever Cloud `mcp-simple-server`

https://github.com/CleverCloud/mcp-simple-server



DevDays

That's all, folks!

Thank you all!