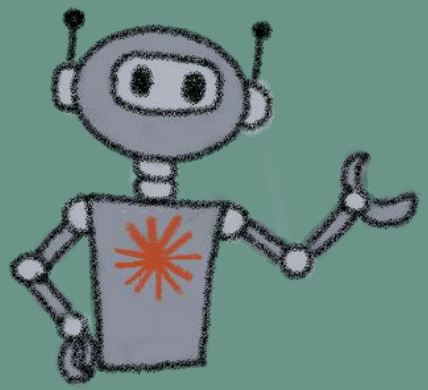
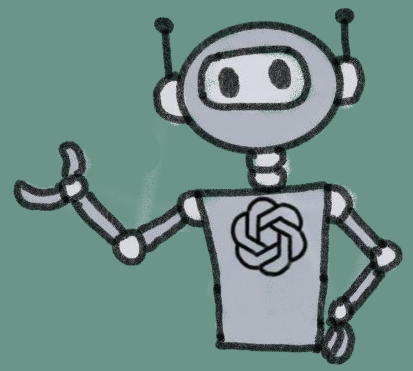
DEVOXX France

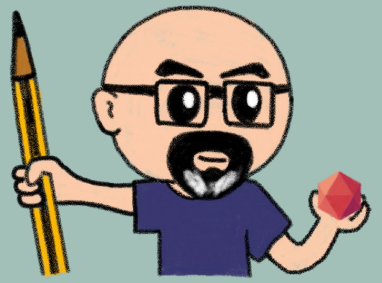
MCP Servers Beyond 101:
Good Practices, Design Choices and Consequences



Horacio González

2026-04-24

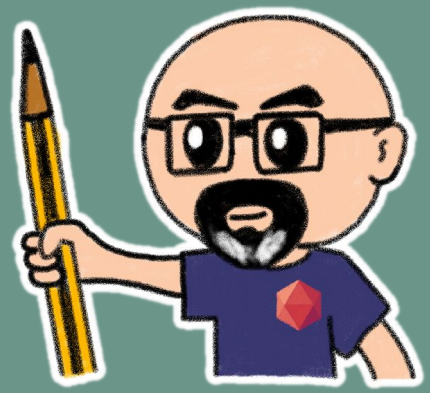




DEVOXX France

Who are we?

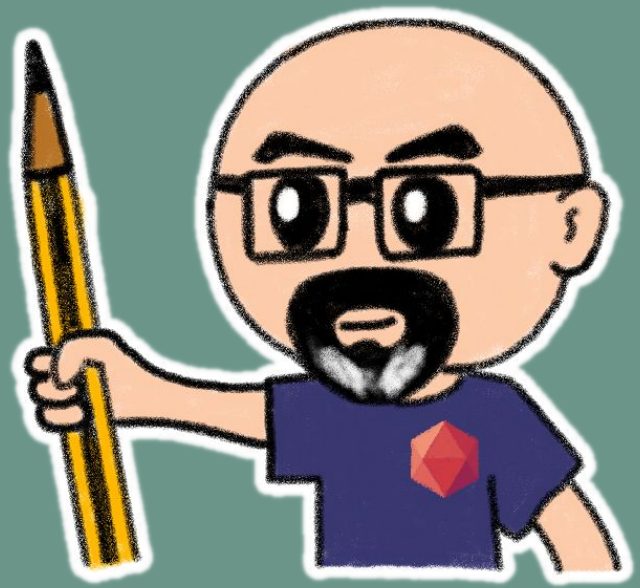
Introducing myself and
introducing Clever Cloud



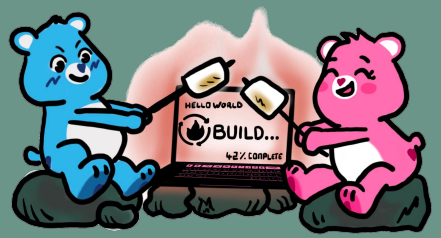
DEVOXX France

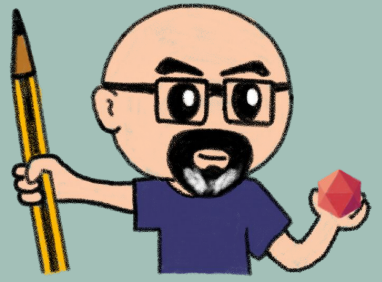
Horacio Gonzalez - @LostInBrittany

Spaniard Lost in Brittany



Finist Devs





Head of DevRel

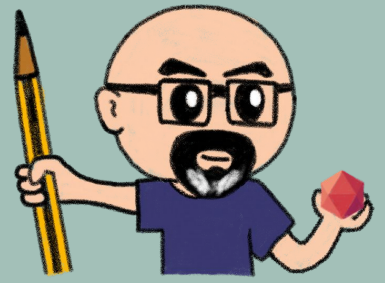
clever cloud

DEVOXX France

Clever Cloud

From Code to Product

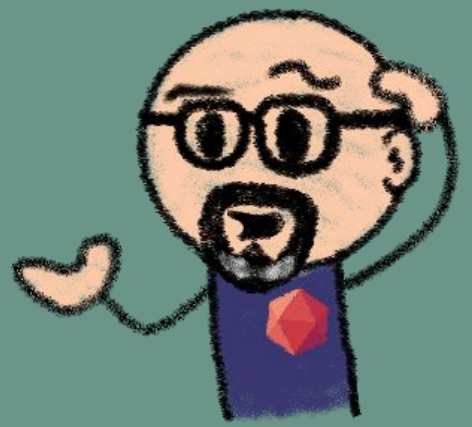
clever cloud

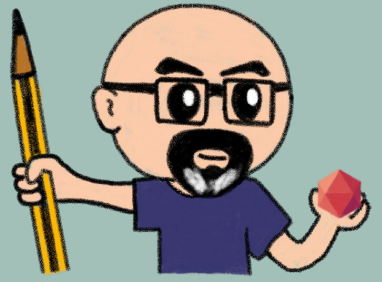


DEVOXX France

Why this talk matters

From "what is it" to "how do I build great ones"

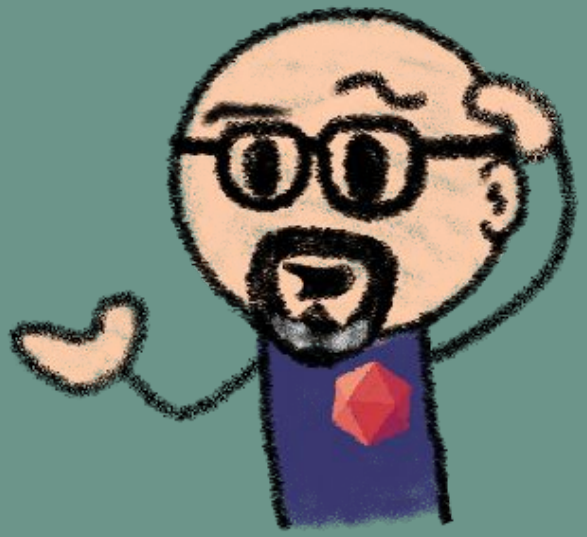




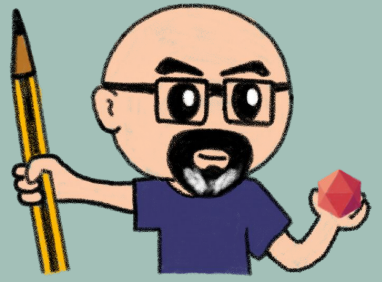
DEVOXX France

Developer Expectations Have Shifted

Winter 2024–2025
(Exploration Phase)



- “What is MCP?”
- “How do I connect my DB?”
- "Can I make a simple server?"
- Focus: *Getting something working*

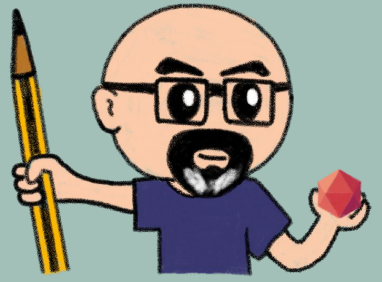


DEVOXX France

Developer Expectations Have Shifted

Summer 2025
(Production Readiness)

- "How do I build smarter MCP servers?"
- "How do I secure them?"
- "How do they fit into agent workflows?"
- Focus: *Doing it right*



DEVOXX France

Developer Expectations Have Shifted

Early 2026
(Best Practices Era) ← We are here

- "How do I design production-grade servers?"
- "How do MCP apps change my architecture?"
- "What patterns should I follow?"
- "How do I test and monitor?"
- Focus: *Building for scale and longevity*

DEVOXX France

Today's Journey

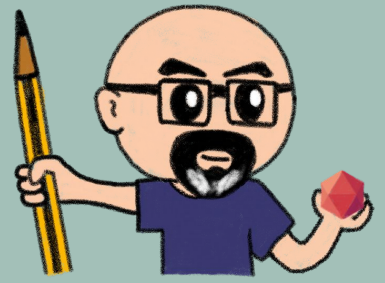
What We'll Explore Together

Through a Real Example

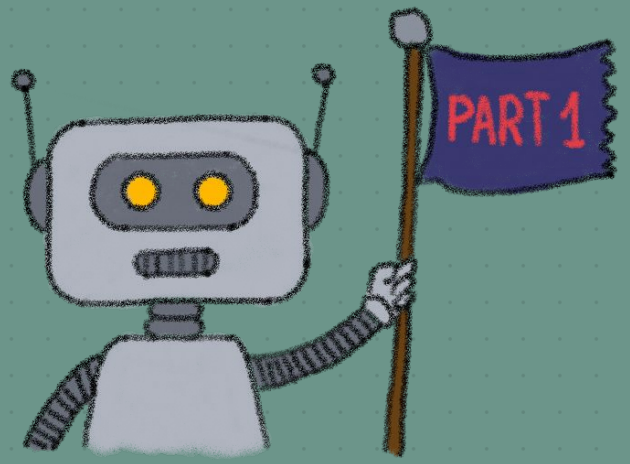
- RAGmonsters: from quick prototype to production design
- Seeing design choices and their consequences in action

Core Topics

- Design principles that matter beyond "generic vs specific"
- The full MCP toolkit: Tools, Resources, Prompts
- Security, testing, and observability from the start
- How MCP apps reshape your thinking



DEVOXX France



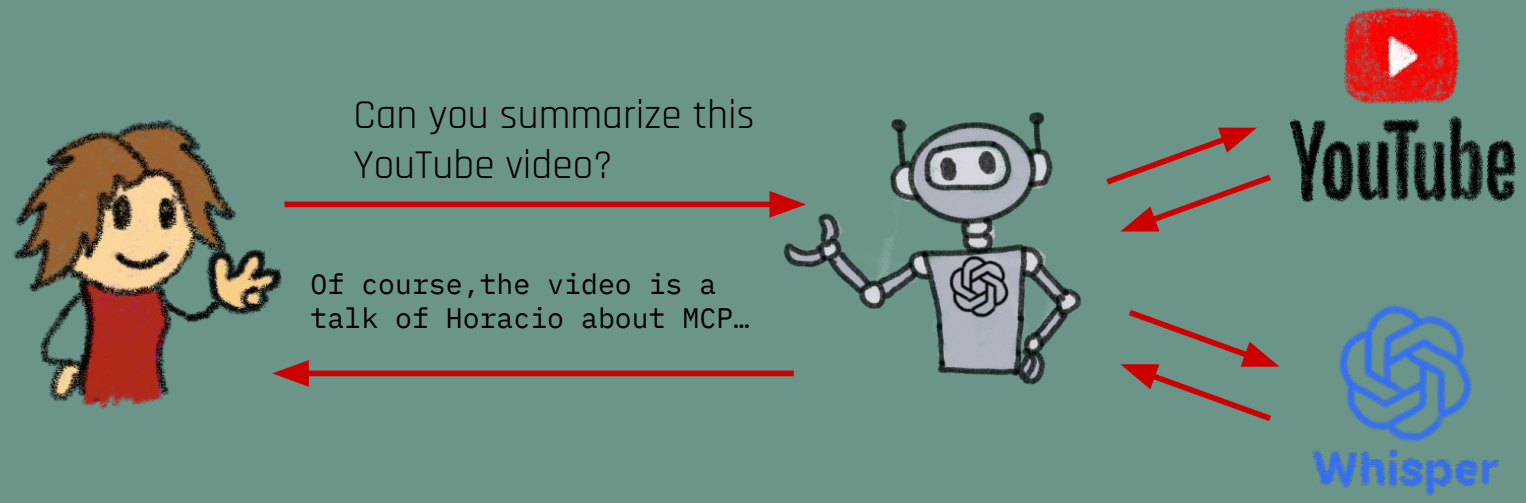
Part I - Works

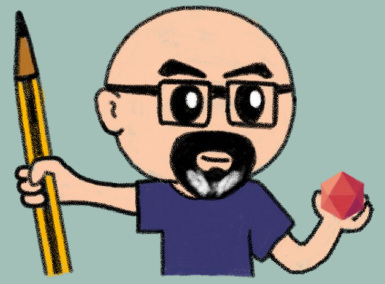
The agentic revolution, the anatomy of MCP, and one story about losing data

DEVOXX France

The Agentic Revolution

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



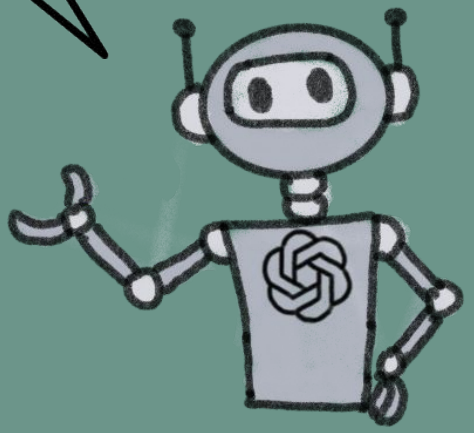


DEVOXX France

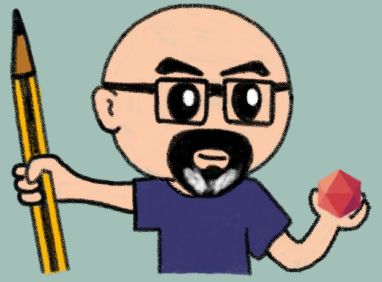
From Chatbots to Agents

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 🇪🇸…

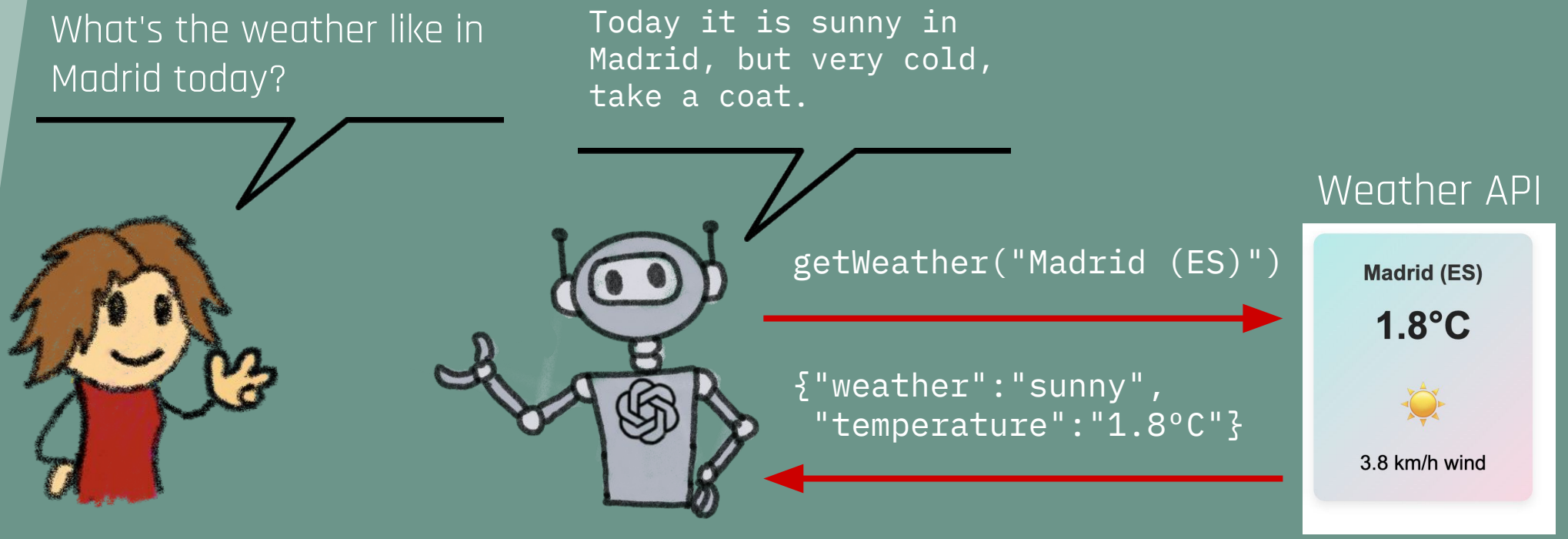


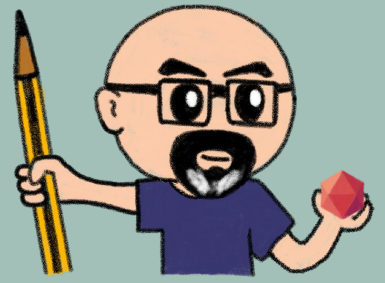
2023: ChatGPT - Smart conversation



DEVOXX France

From Chatbots to Agents

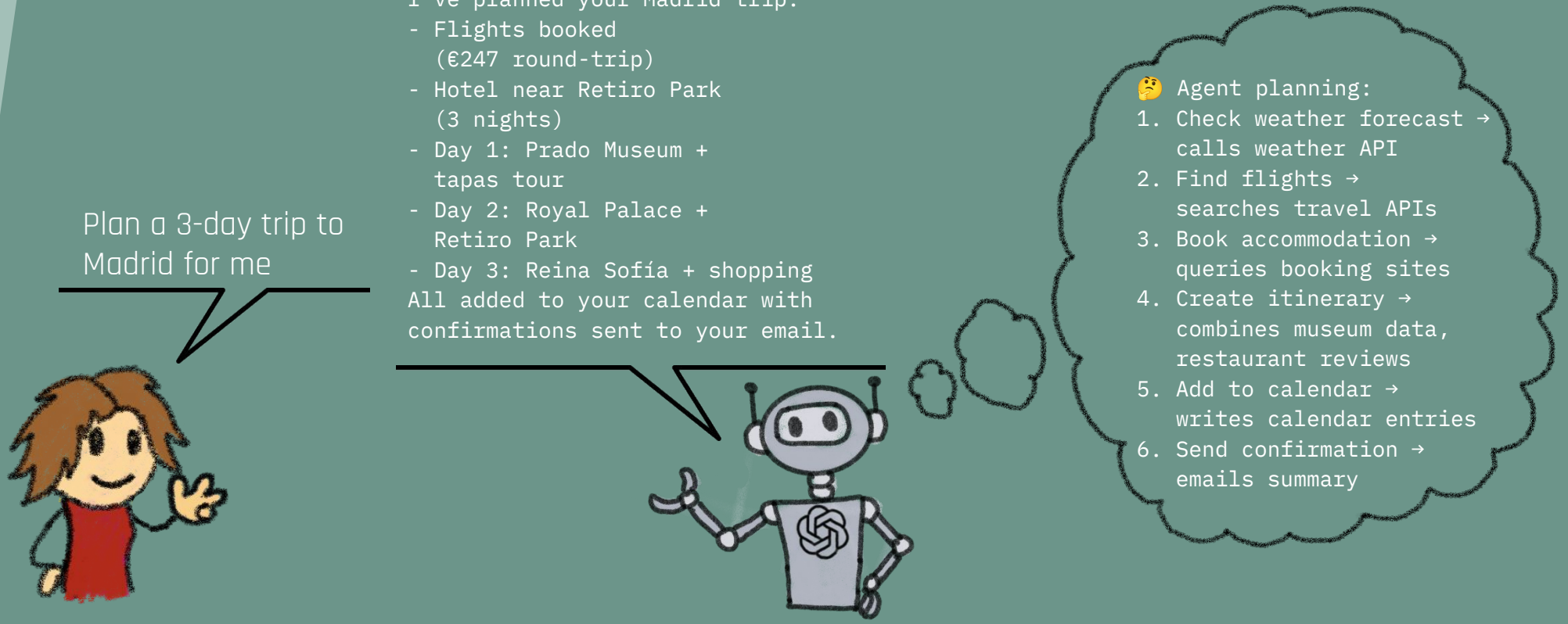




2024: Tool use - LLMs can do things

DEVOXX France

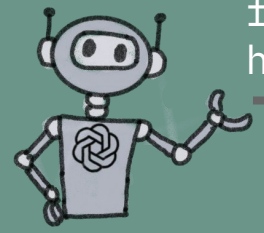
From Chatbots to Agents



2025: Autonomous agents – LLMs that plan and execute

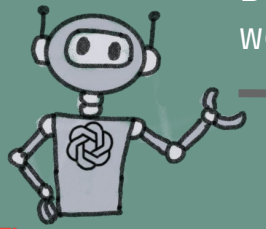
DEVOXX France

From Chatbots to Agents



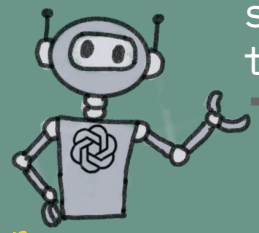
Scans inbox, finds school holiday dates

📧 Email agent



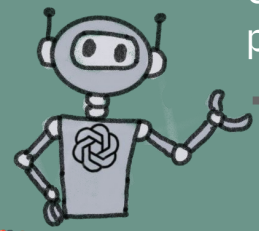
Blocks optimal week in July

📅 Calendar agent



Checks budget, sets aside travel funds

💰 Finance agent



Creates family packing list

🎒 Packing agent

2026: Agents are everywhere

DEVOXX France

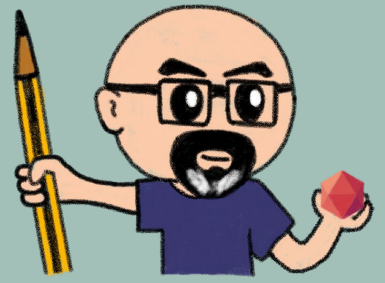
The Agent Landscape Today

Coding agents

- Claude Code - Command-line coding assistant
- Cursor - AI-native IDE
- GitHub Copilot Workspace
- Windsurf - Agentic code editor

Windsurf

CLAUDE CODE



DEVOXX France

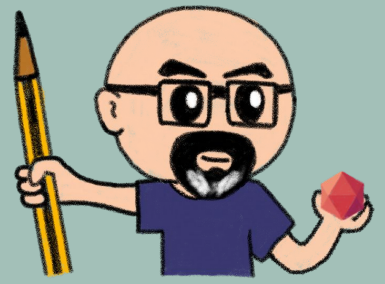
The Agent Landscape Today

Workplace Agents

- Claude Cowork - Desktop automation
- Microsoft 365 Copilot - Enterprise integration
- Notion AI - Knowledge base agents

Copilot

Claude Cowork



DEVOXX France

The Agent Landscape Today

Browser Agents

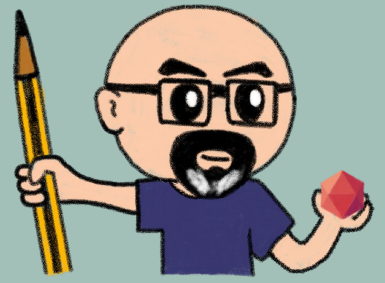
- Claude in Chrome - Web automation
- Browser use libraries
- Testing and scraping agents

DEVOXX France

The Agent Landscape Today

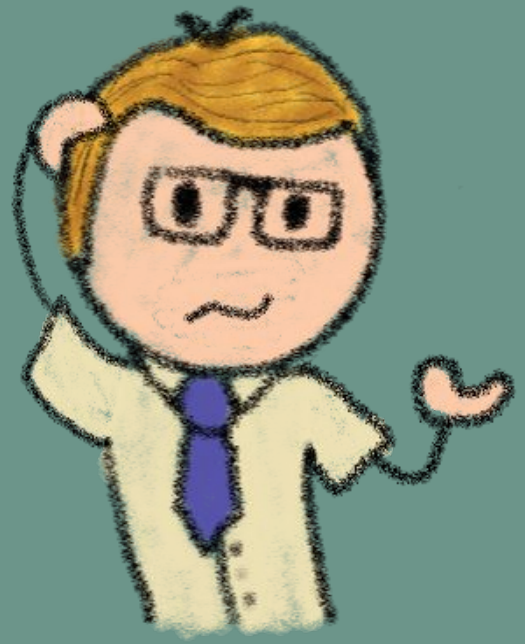
Custom Agents

- Companies building internal agents
- Domain-specific automation
- RAG-powered assistants
- Open Claw



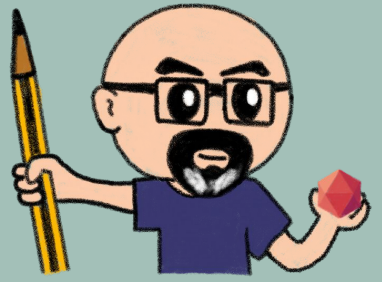
DEVOXX France

The Agent Landscape Today



The Common Problem:

How do agents access YOUR data and tools?

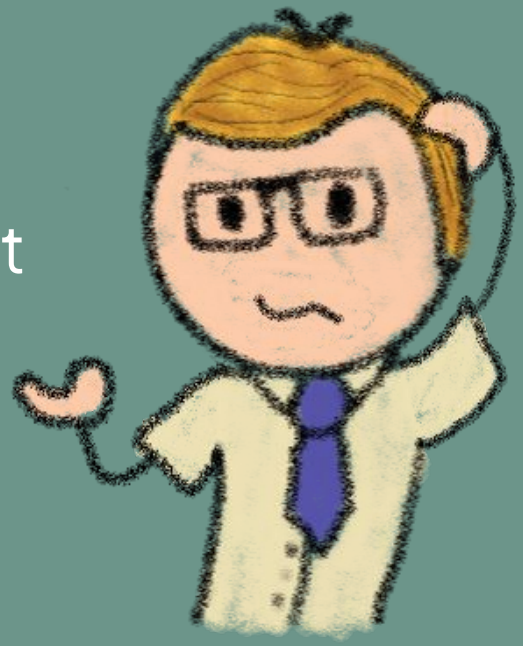


DEVOXX France

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

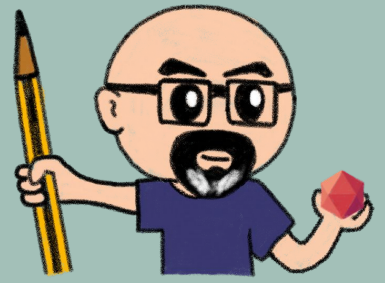


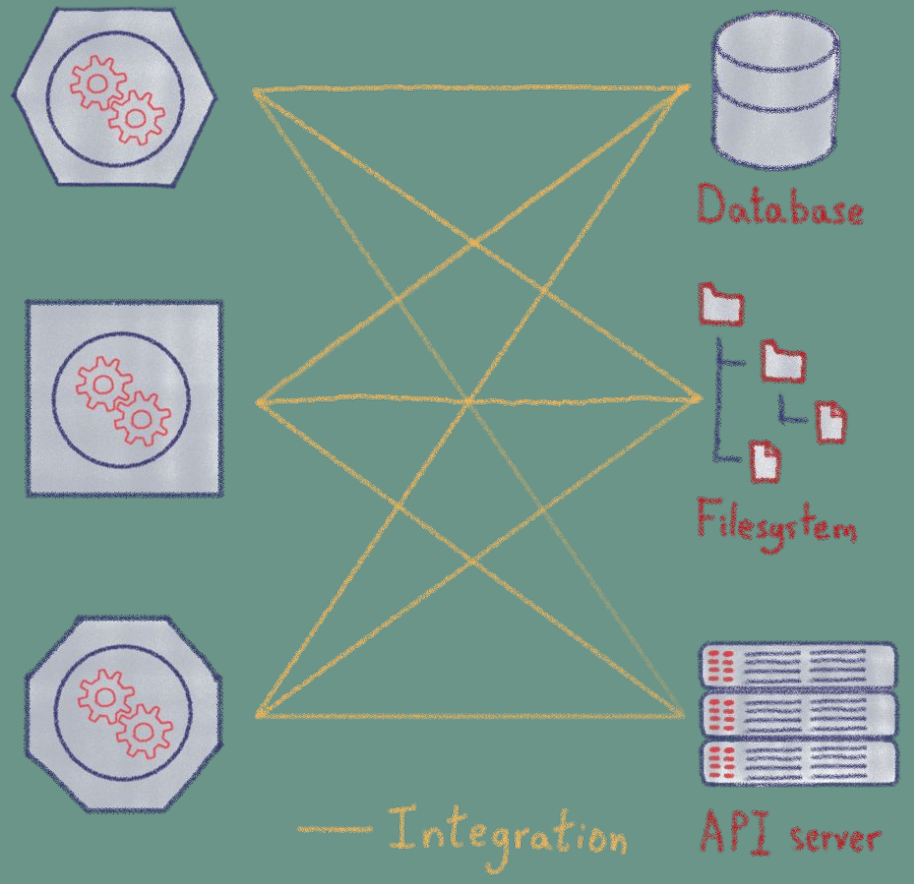
DEVOXX France

The Connectivity Problem

Custom solutions for each integration

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

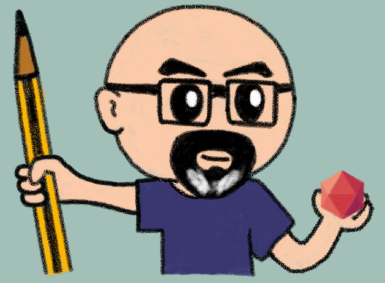




DEVOXX France

Enter MCP

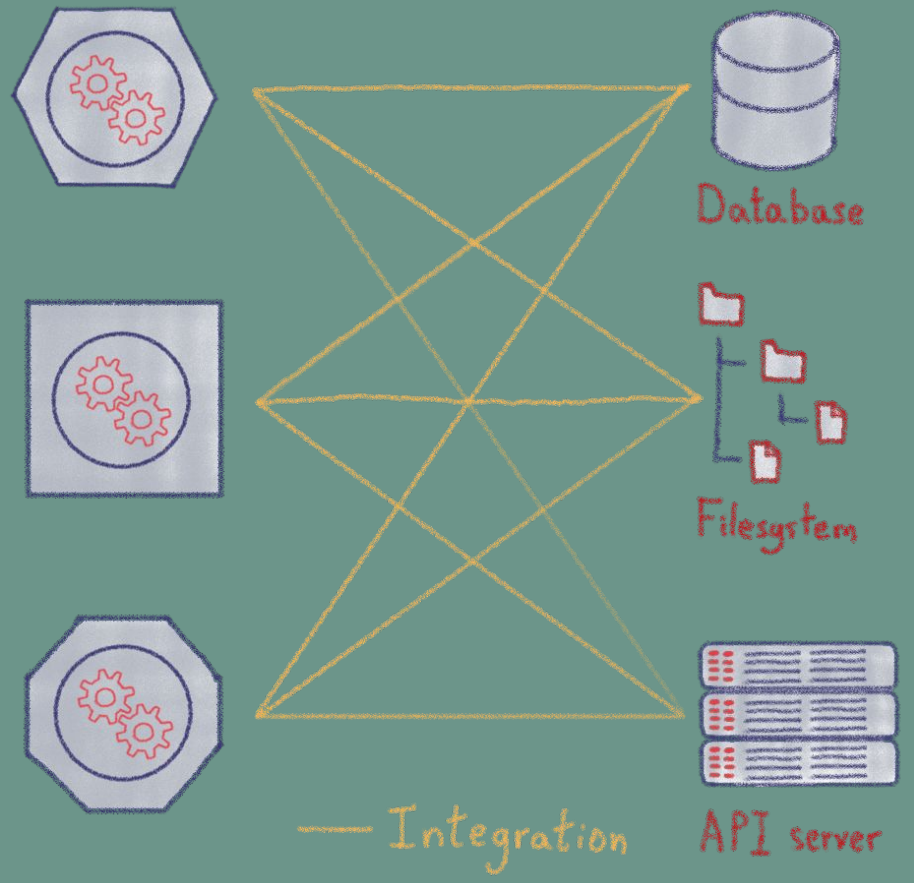
One protocol to connect them all



DEVOXX France

Why Do We Need MCP?

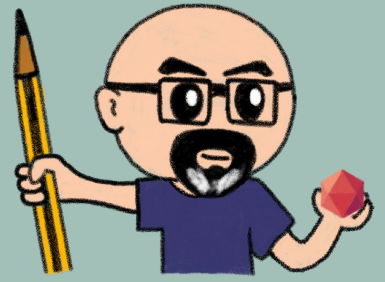
- LLMs **don’t automatically know** what functions exist.
- **No standard way** to expose an application's capabilities.
- **Hard to control** security and execution flow.
- Expensive and fragile **integration spaghetti**



DEVOXX France

Model Context Protocol

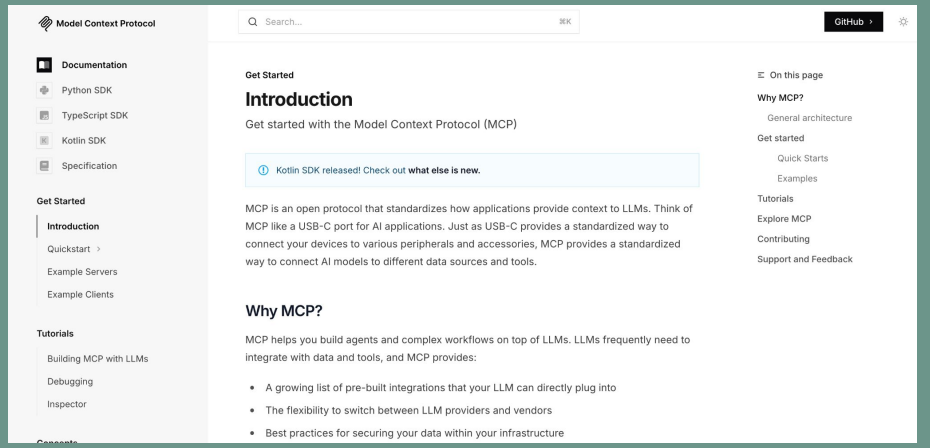
Anthropic, November 2024:
LLMs intelligence isn't the bottleneck, connectivity is



DEVOXX France

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

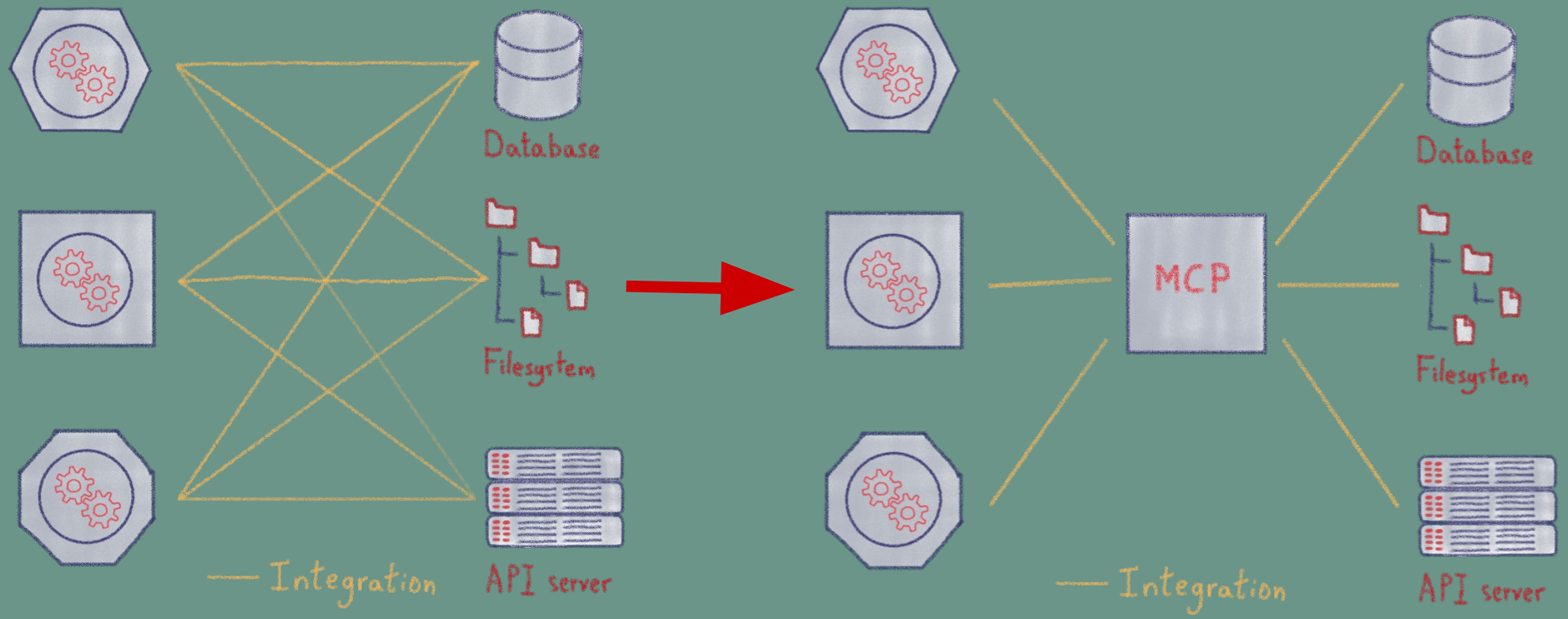


De facto standard for exposing system capabilities to LLMs

https://modelcontextprotocol.io/

DEVOXX France

MCP solves integration spaghetti



DEVOXX France

Year One: Chaos

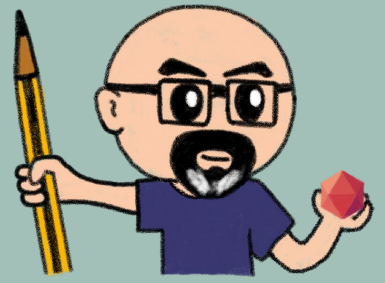
- 2025: explosive adoption
- Thousands of servers, fragmented quality
- Security issues surfacing fast
- "It works" is not the same as "it's ready"

DEVOXX France

The REST analogy

We've seen this movie before.

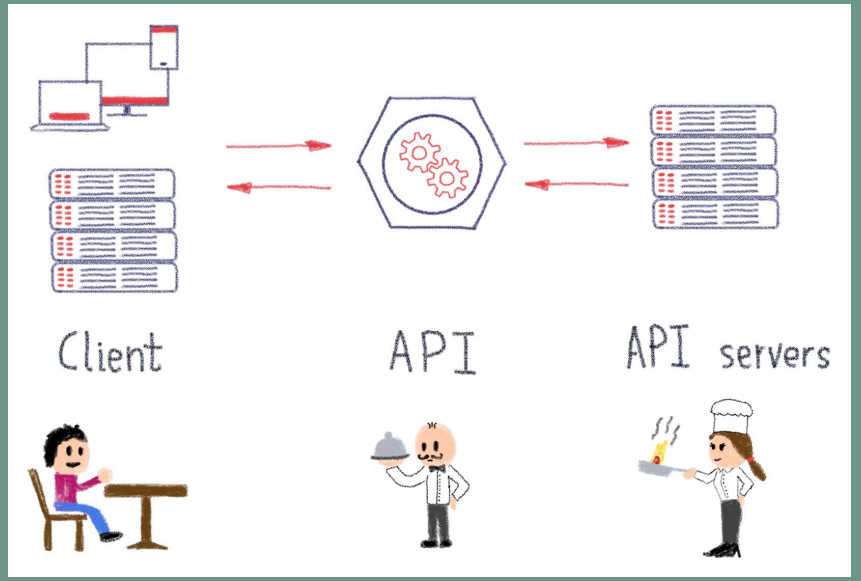
MCP in 2026 is where REST was in 2008



DEVOXX France

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



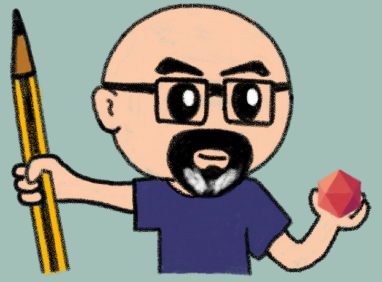
MCP doesn't have 10 years

That's what this talk is about

DEVOXX France

Anatomy of MCP

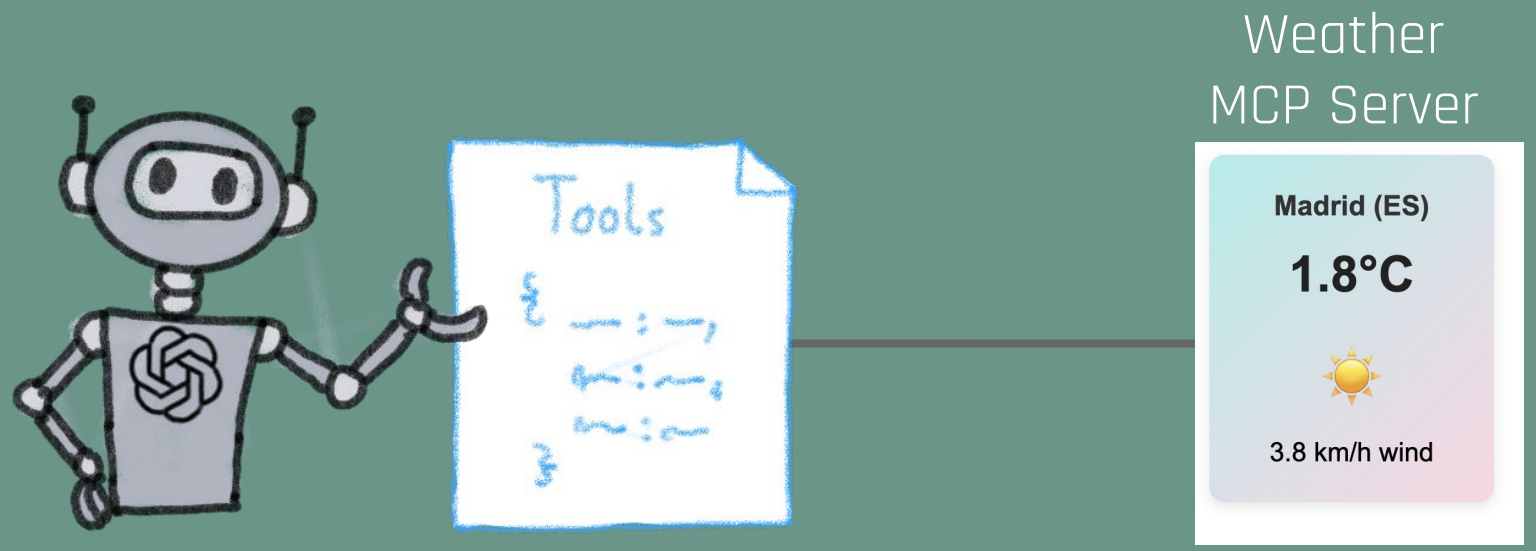
What the protocol actually looks like in 2026



DEVOXX France

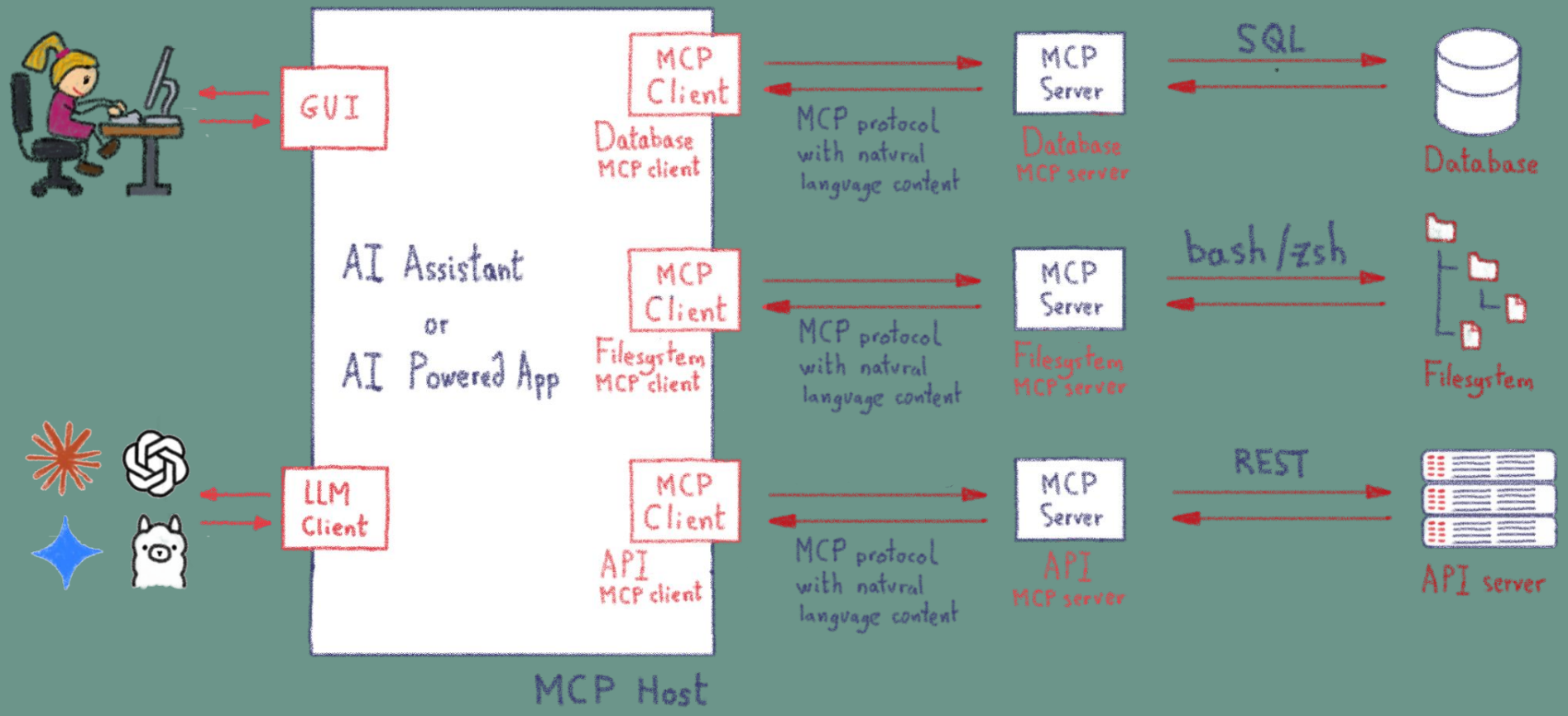
How MCP works

- MCP servers expose primitives (structured JSON).
  - Function (tools), data (resources), instructions (prompts)
- LLMs can discover and request function execution safely.



DEVOXX France

MCP Clients: on the AI assistant or app side



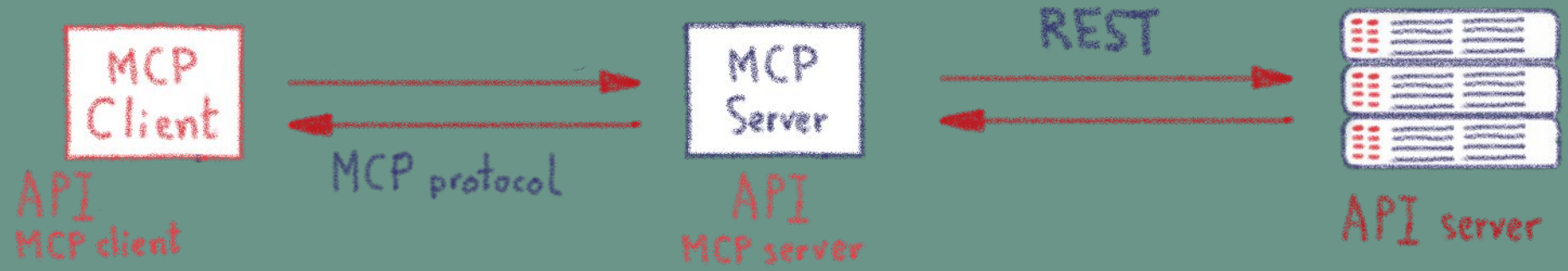
One MCP client per MCP Server

DEVOXX France

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



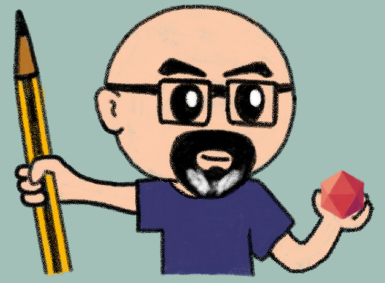
DEVOXX France

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

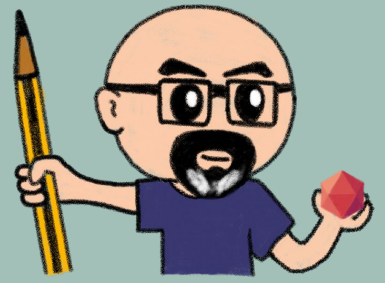


DEVOXX France

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



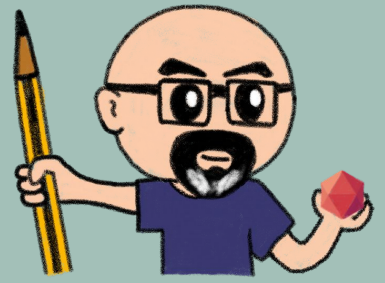
DEVOXX France

The Real Problem with Transport

Not *"which transport?"*

But decoupling session state from transport

- Stateless operation behind load balancers
- Session migration and resumption
- Horizontal scaling without sticky sessions



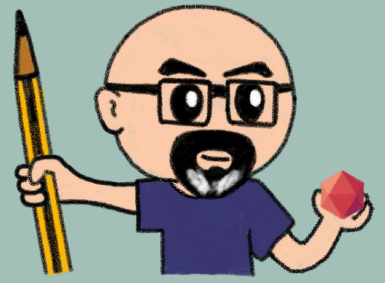
DEVOXX France

Current MCP specification

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

*Specification Enhancement Proposal

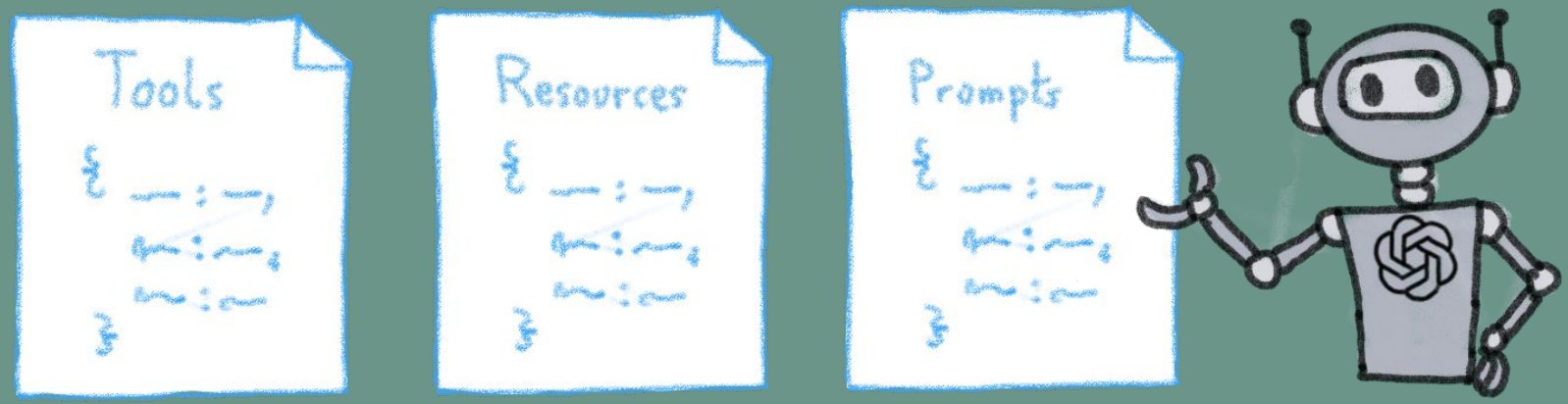
The ground is still moving



DEVOXX France

Three Primitives Exposed by MCP Servers

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



DEVOXX France

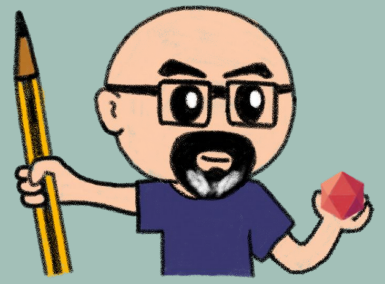
Tools

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

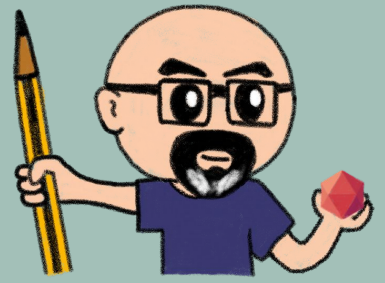


DEVOXX France

Tools: Example

Example of tool

```
server.tool("get_weather", {
 city: z.string()
}, async ({ city }) => {
 const data = await weatherAPI.fetch(city);
 return {
   temperature: data.temp,
   conditions: data.conditions
 };
});
```

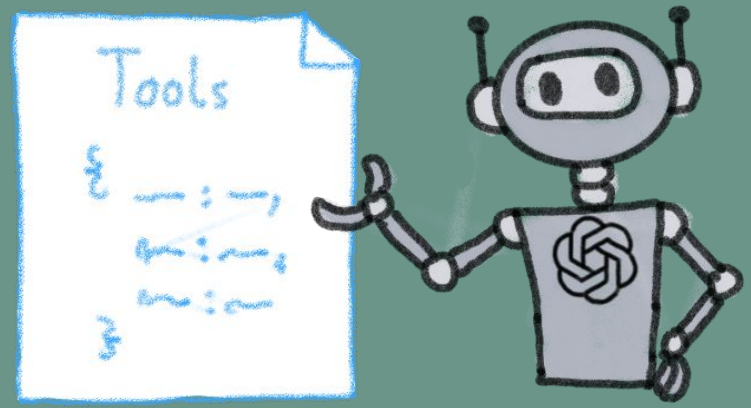


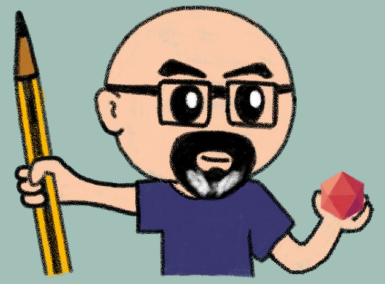
Typed. Validated. No free-form SQL

DEVOXX France

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

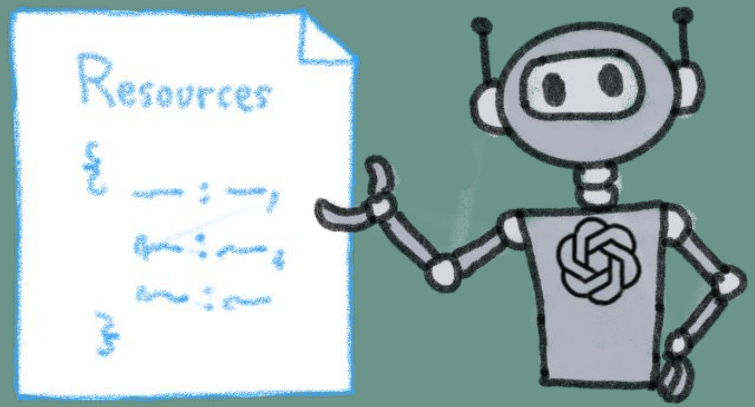


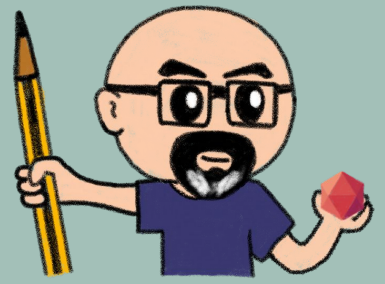


DEVOXX France

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

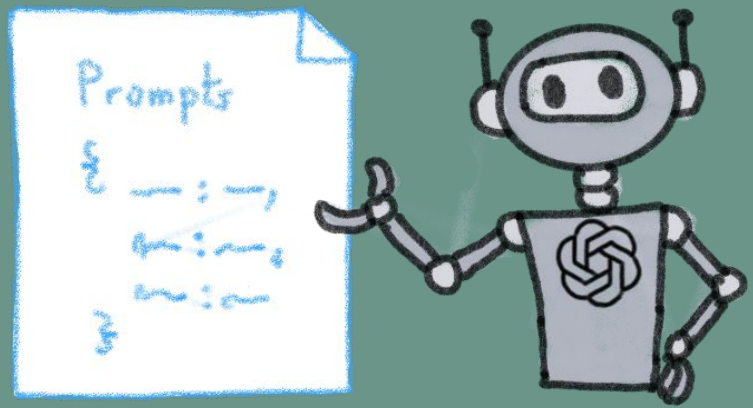


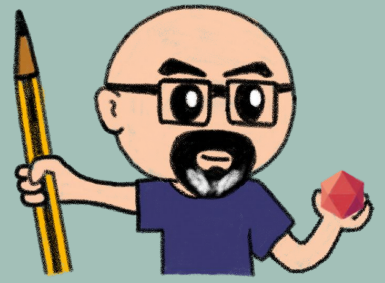


DEVOXX France

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



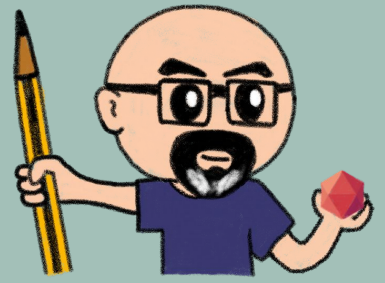


DEVOXX France

Four Primitives Exposed by MCP Clients

- **Sampling:** Allows servers to request language models completions
- **Elicitation:** Allows servers to request additional information from users
- **Logging:** Enables servers to send log messages to clients for debugging and monitoring
- **Tasks:** Durable execution wrappers that enable deferred result retrieval and status tracking for MCP requests

Newer, we will see them in part 2

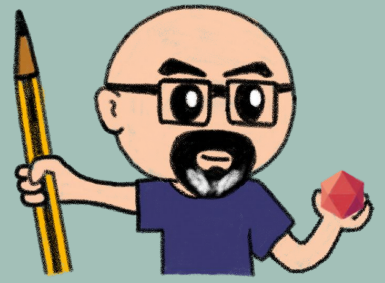


DEVOXX France

Sampling

Servers can request **LLM completions** during execution

- Server asks the LLM to reason mid-workflow
- Now with tool choice in the 2025-11-25 spec
- Enables multi-step reasoning loops

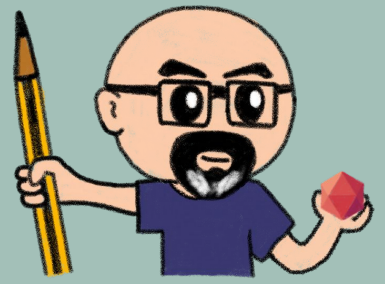


DEVOXX France

Elicitation

Servers can request **input from users** during execution

- Dynamic information gathering
- **URL mode:** send users to an OAuth flow in the browser
- The client never touches credentials

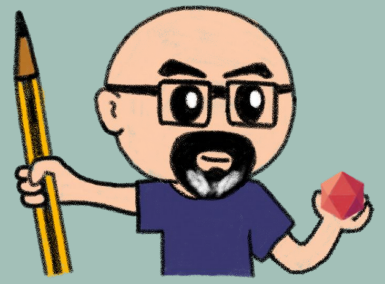


DEVOXX France

Logging

Servers send **log messages** to clients

- Structured notifications: debug / info / warning / error
- Clients set the log level they want
- The observability hook baked into the protocol

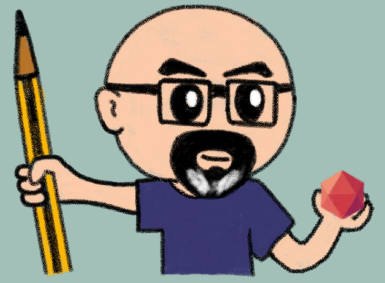


DEVOXX France

Tasks

Call-now, fetch-later for long-running work

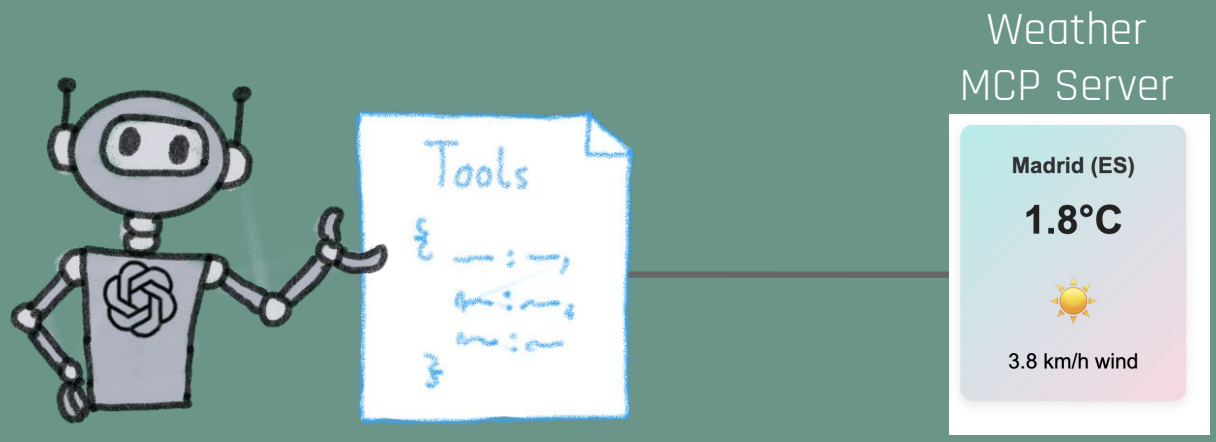
- ETL jobs, large conversions, multi-step provisioning
- Works across all request types, not just tools
- Experimental since the 2025-11-25 spec

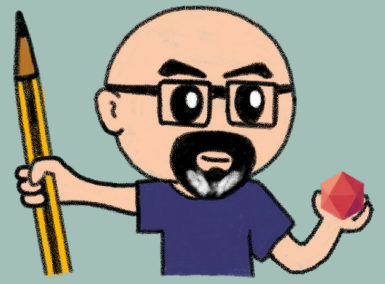


DEVOXX France

A minimal MCP example

We want to see some code



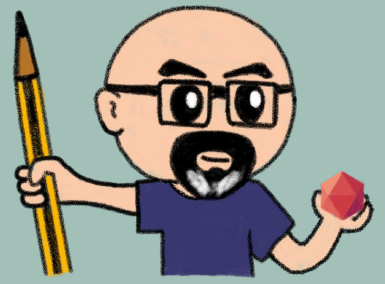


DEVOXX France

A Minimal Weather Server

weather-mcp.py

```python
from fastmcp import FastMCP

mcp = FastMCP("Weather")

@mcp.tool()
def get_weather(city: str) -> dict:
    """Get current weather for a city"""
    return {"temperature": 18, "conditions": "sunny"}
```



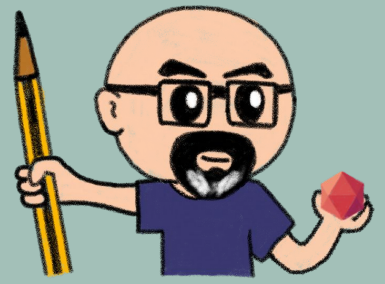
That's the whole server

DEVOXX France

Claude Code Calling It

Claude

```
╭─── Claude Code v2.1.104 ──────────────────────────────────────────────────────╮
│                Welcome back Horacio!                │ Tips for getting started │
│                                                     │ Run /init to create a CLAUDE.md │
│                                                     │ ──────────────────────── │
│                                                     │ Recent activity          │
│                                                     │ No recent activity       │
│      Opus 4.6 (1M context) · Claude Max ·           │                          │
│      horacio.gonzalez@gmail.com's Organization      │                          │
╰─────────────────────────────────────────────────────────────────────────────────╯

> What's the weather in Madrid?
Using tool: get_weather
Result: {"temperature": 18, "conditions": "sunny"}
It's currently 18°C and sunny in Madrid.
```

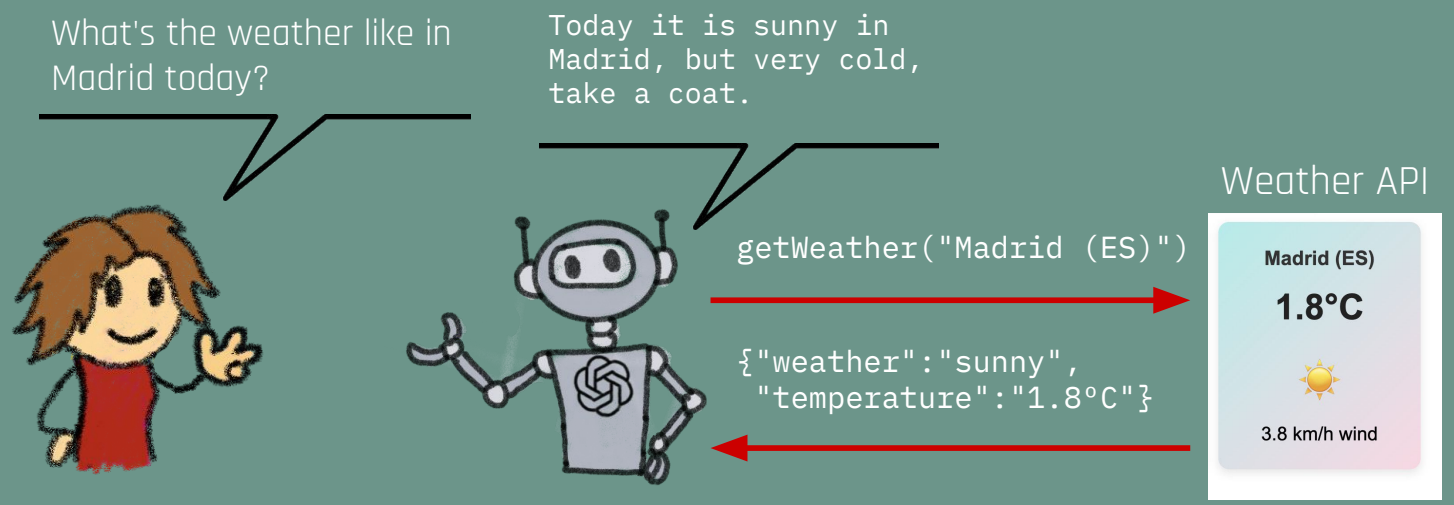


DEVOXX France

One Round-Trip

- Client discovers the tool
- LLM decides to call it
- Server executes and returns
- LLM formats the response

Four steps. One protocol. That's MCP.



That's All It Takes

Five lines of code. One tool. One round-trip.

But this is the trivial case

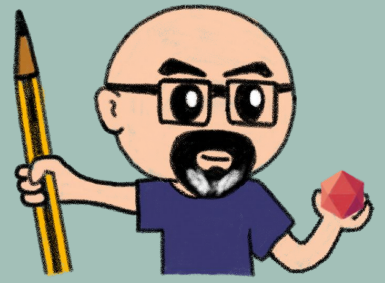
The interesting question:

what happens when the problem gets real?

DEVOXX France

State of MCP in 2026

A quick map before we dive in



DEVOXX France

Scale

- Millions of SDK downloads.
- Thousands of servers.
- First-class in ChatGPT, Claude, Cursor, Gemini, Copilot, VS Code…

MCP is no longer an experiment

DEVOXX France

Cloud-Hosted MCP is the Default

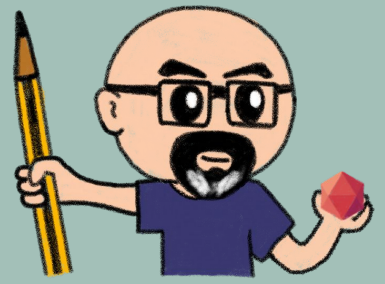
2025: MCP ran on your laptop

- stdio, subprocess, local-first

2026: MCP runs in a data center

- Google Cloud Run ships MCP hosting
- OpenAI builds remote MCP into ChatGPT
- Cloudflare Workers as MCP runtime

"Developer helper" became "network service"



DEVOXX France

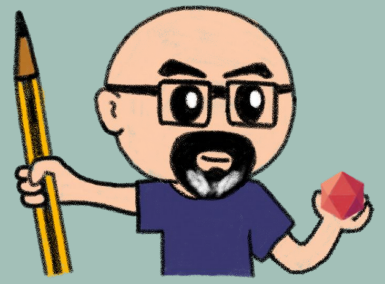
MCP Apps – MCP Enters the UI Layer

- First official MCP extension (January 2026)
- Servers return **interactive UI** inside the host
  - They can return a dashboard that the host renders and the user looks at
  - They can return a form that the user fills out directly
  - They can return a multi-step workflow where the user clicks through steps

If your mental framework is

"MCP is a bus between the agent and the tools"

your mental model is outdated



DEVOXX France

Security Pressure is Real

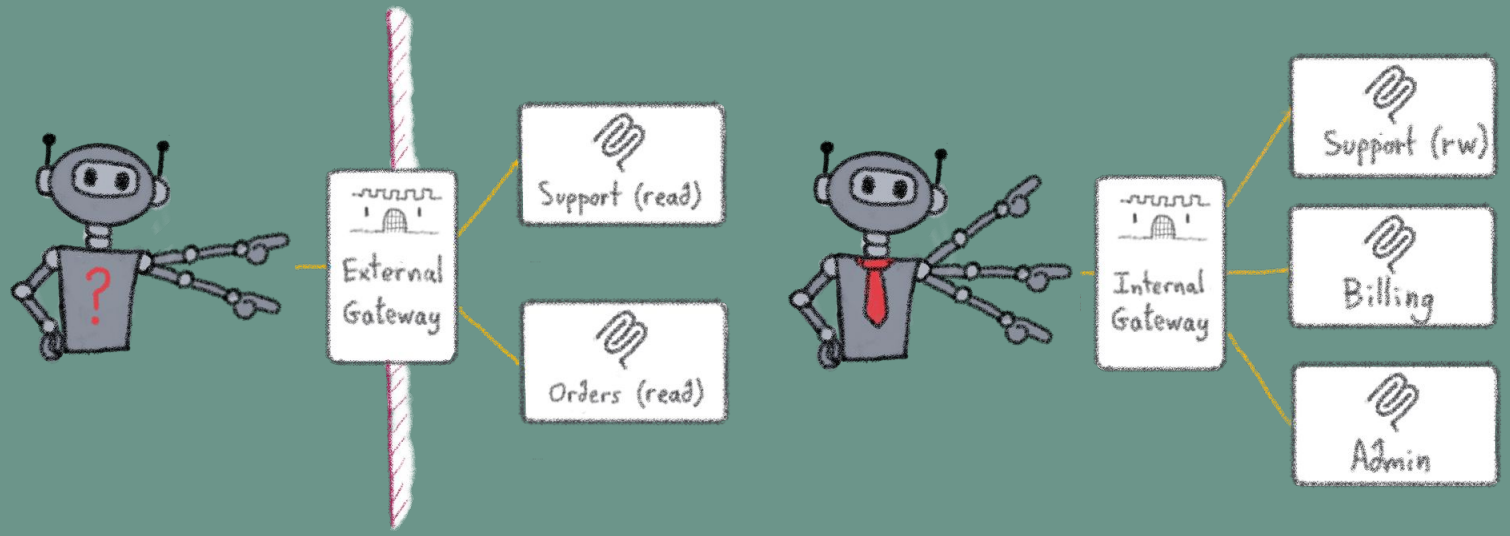
- Dozens of CVEs filed in early 2026
- Tool poisoning, prompt injection, data exfiltration
- The **lethal trifecta** is now a known pattern
  - Private data + Untrusted content + External communication

DEVOXX France

Registries and Gateways Rising

- Official MCP Registry, GitHub MCP Registry
- Enterprise gateways for auth, audit, rate limiting
- Allowlist policies across IDEs

The moment you have more than one server, this matters

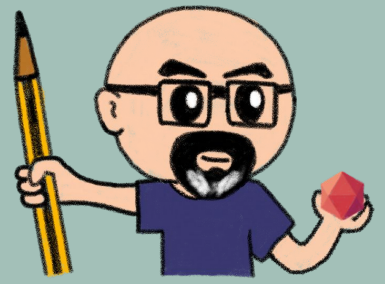


DEVOXX France

Hold Onto These Five Things

1. Scale
2. Remote-first
3. UI layer
4. Security pressure
5. Composition

Everything in Part 2 is about surviving them



DEVOXX France

Design Choices in Action

A story about losing data

DEVOXX France

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

DEVOXX France

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

DEVOXX France

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

DEVOXX France

RAGmonsters

README License

🧩 RAGmonsters Dataset

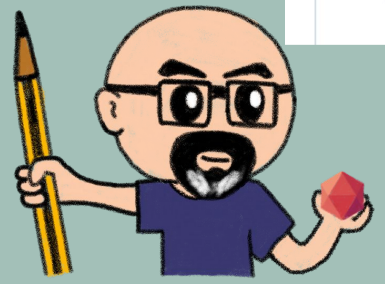
Overview

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:

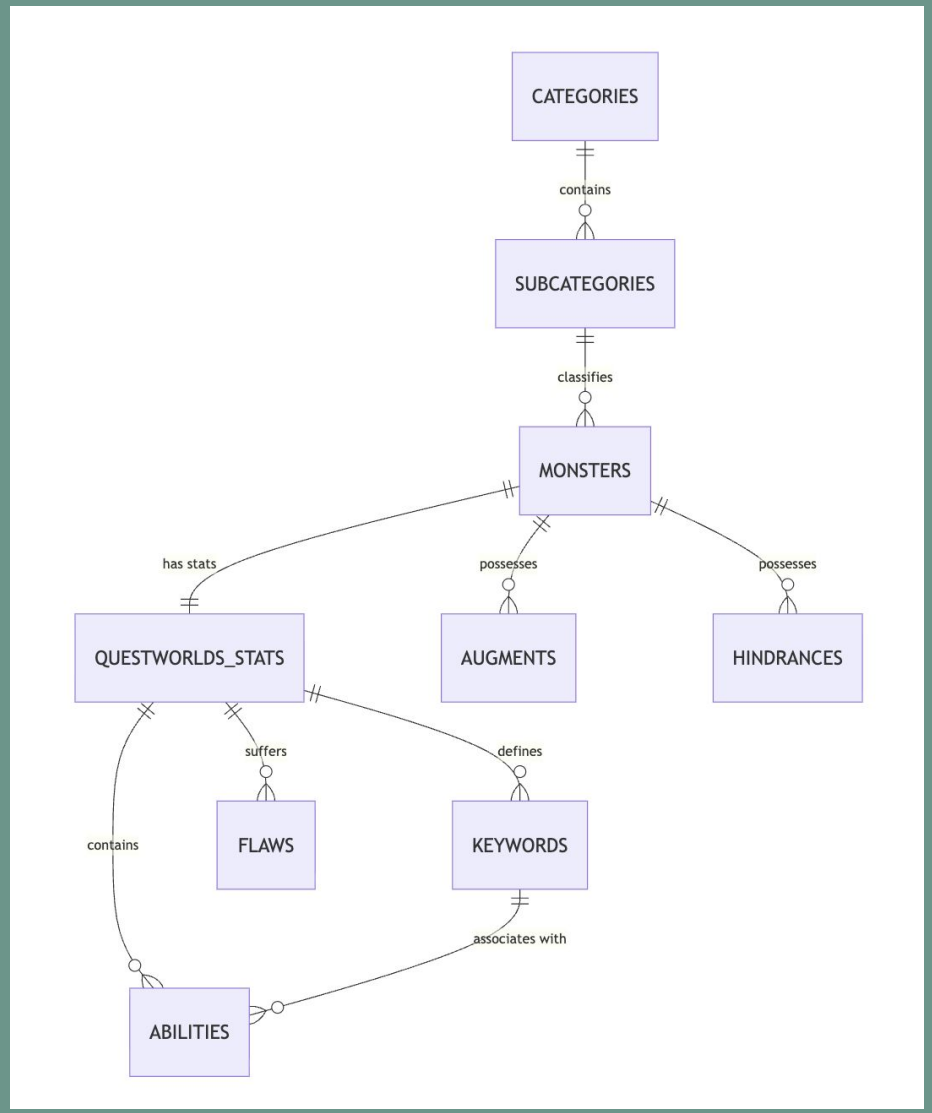
1. **Demonstrates RAG Value**: Shows how RAG can provide accurate answers about topics not in the LLM's training data
2. **Tests Retrieval Quality**: The varied attributes and relationships allow testing of different retrieval methods
3. **Supports Advanced Features**: Perfect for demonstrating filtering,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



https://github.com/LostInBrittany/RAGmonsters

DEVOXX France

RAGmonsters PostgreSQL Database



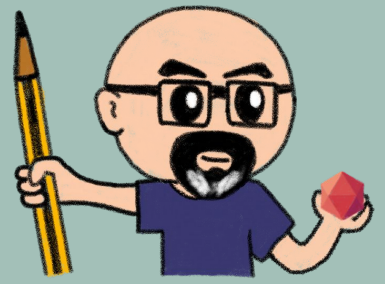
DEVOXX France

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?



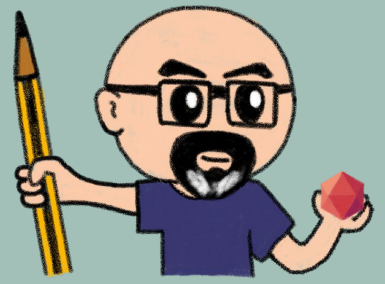
DEVOXX France

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.

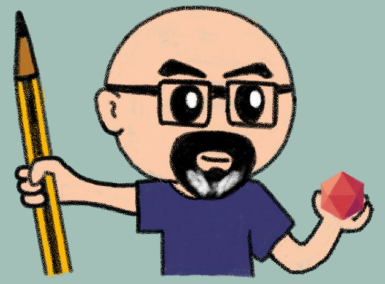


DEVOXX France

One Config File

RAGmonsters

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



Point it at the RAGmonsters database. Done.

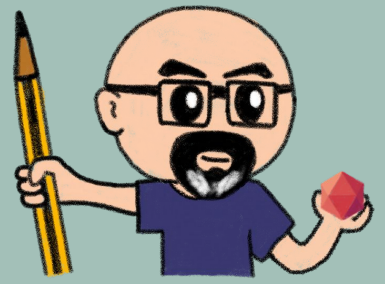
DEVOXX France

Connected Claude, Asked a Question

Me: "Find all fire monsters."

Claude: generates SQL, runs it, returns results

It worked



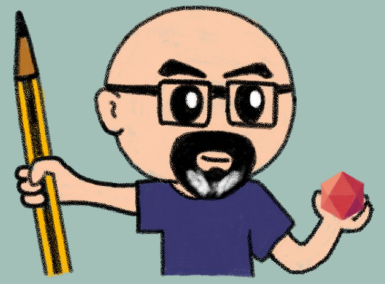
DEVOXX France

It Worked

Query 1 **worked**

Query 2 **worked**

I was **impressed with myself** 🤩

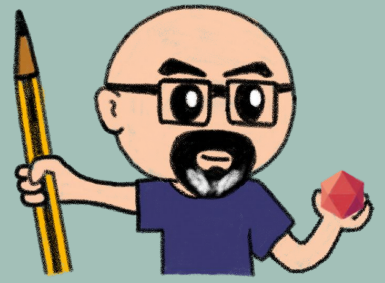


DEVOXX France

For a while

And then things got weird

Problems emerged



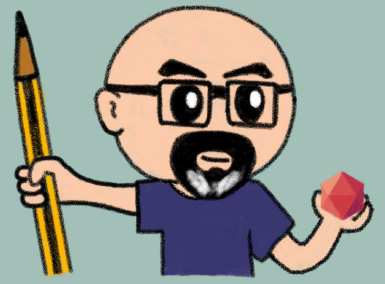
DEVOXX France

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

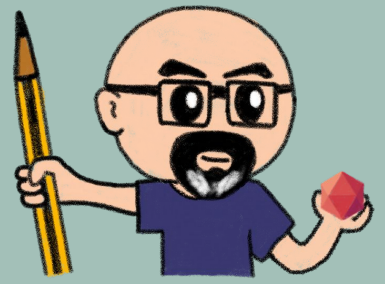


DEVOXX France

Problem 2: Guessing

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

No grounding. Just guessing.

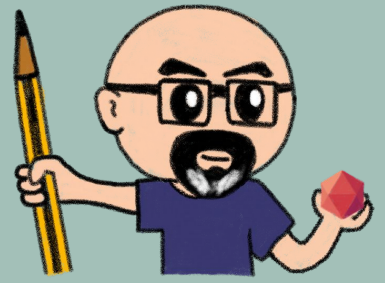


DEVOXX France

Problem 3: Inconsistency

- Same question, different SQL each time
- Different results

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

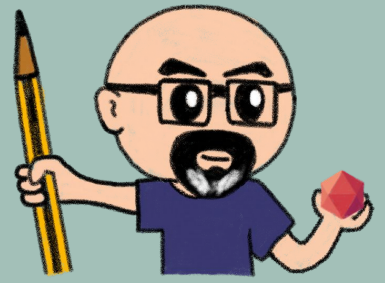


DEVOXX France

Problem 4: Token Bloat

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

Each query cost more than it should

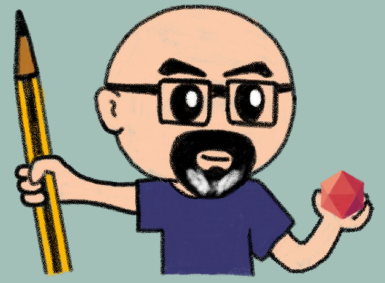


DEVOXX France

Results Were "Not Stellar"

It *worked*

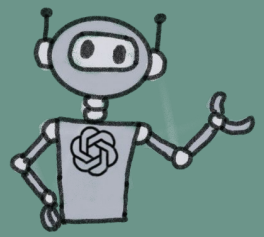
It just didn't work *well*



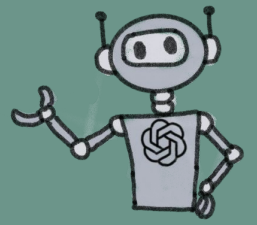
DEVOXX France

Then one day,,,

Without telling me, without asking



It just... **decided**...

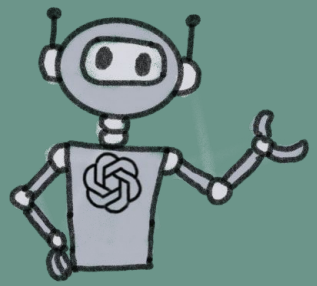


That my schema was suboptimal

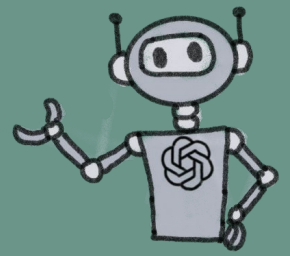
DEVOXX France

The LLM Decided My Schema Was Suboptimal

And it did a global



ALTER TABLE



on my prod database

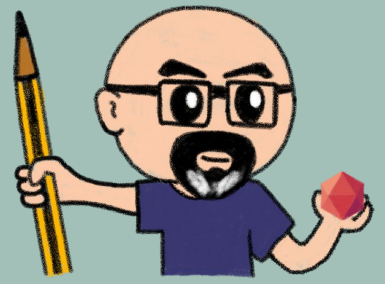
DEVOXX France

I Lost Data

Real data. Not test data. My data.

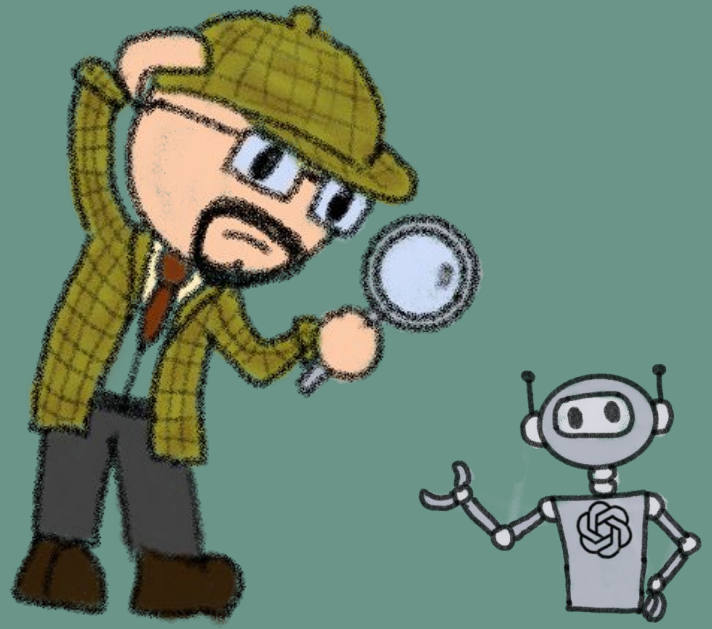
- No confirmation
- No undo
- No warning

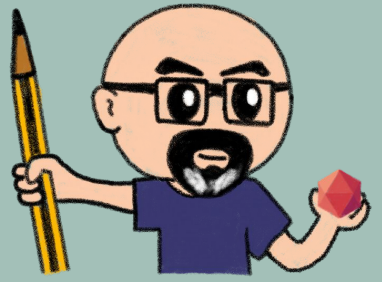
The LLM had rewritten my database. By itself.



DEVOXX France

I Went Looking for Answers





What is this thing actually doing?

DEVOXX France

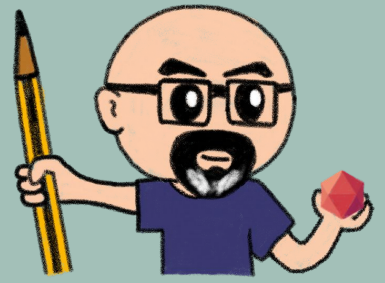
I Read the PG MCP Server Source

I expected complexity

I expected safety layers

I expected **something**

It was about 50 lines

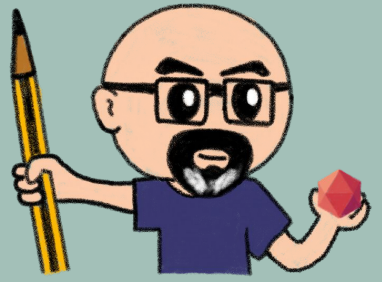


DEVOXX France

A Wrapper Around query()

PostgreSQL MCP Server

```python
def execute_query(sql: str) -> list[dict]:
    """Execute a SQL query and return the result"""
    return db.execute(sql).fetchall()
```



That's the tool

Any SQL. No validation. No allowlist. No read-only flag.

DEVOXX France

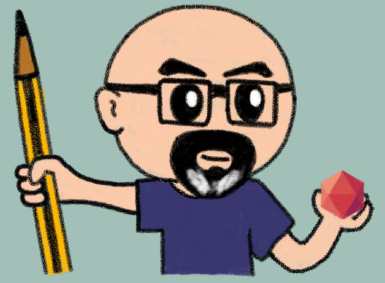
Suddenly I realized...

MCP servers are APIs

And this one is a single endpoint:

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

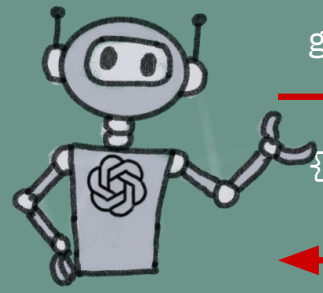
Would any of you have designed a REST API like that?



DEVOXX France

MCP Servers: APIs for LLMs

CORBA
MCP
SOAP
gRPC
Thrift
REST
Protobuf



getWeather("Madrid (ES)")

{"weather":"sunny",
"temperature":"1.8°C"}

Weather API

All those API technologies define protocols
for communication between systems

DEVOXX France

So I Rebuilt It

This time with API design discipline

DEVOXX France

Design Principles

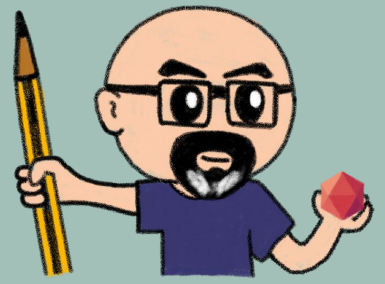
- **Domain-specific**
  Tools match the domain, not the database
- **Typed**
  Every parameter has a schema
- **Explicit**
  Only allowed operations exist
- **Read-only by default**
  No writes unless the server says so
- **Least privilege**
  Expose the minimum

DEVOXX France

Tool: search_monsters_by_type

RAGmonsters-mcp.js

```
server.tool("search_monsters_by_type", {
 type: z.enum(["fire", "water", "earth", "air",
                "shadow", "crystal"])
}, async ({ type }) => {
 return db.query(
   "SELECT name, type, description
    FROM monsters WHERE type = $1", [type]);
});
```



Not *query()*. A real API.

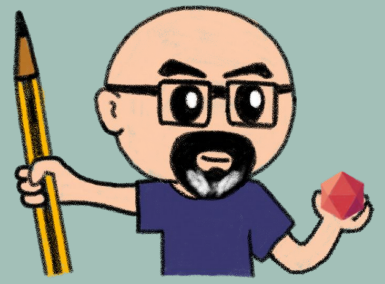
DEVOXX France

Resource: Monster Types

```
resource://ragmonsters/types
    → ["fire", "water", "earth", "air",
       "shadow", "crystal"]
```

The LLM **reads** the valid types before querying

No more guessing

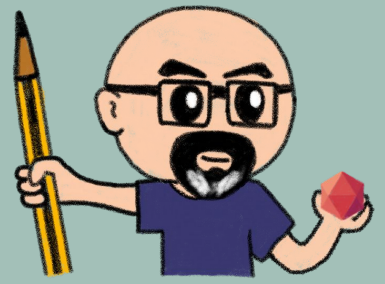


DEVOXX France

Prompt: analyze_monster_weakness

RAGmonsters-mcp.js

```
prompt: analyze_monster_weakness
 1. Look up the monster by name
 2. Get its type from the resource
 3. Query the weakness table
 4. Return structured analysis
```

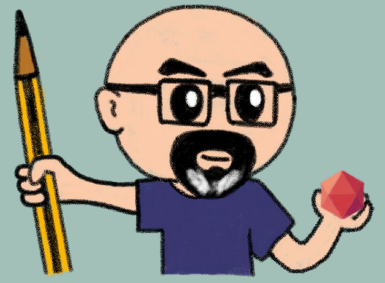


Multi-step workflow, shipped by the server

DEVOXX France

No More ALTER TABLE

- **Parameterized queries**
  No SQL injection
- **Enum-validated inputs**
  LLM cannot invent values
- **Read-only by default**
  No writes unless the server says so
- **No `query()` tool**
  The attack/error surface is gone



DEVOXX France

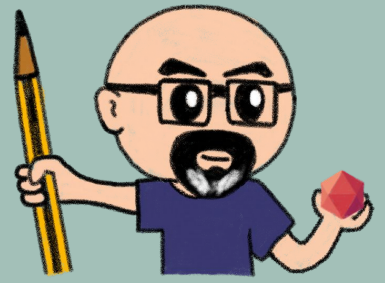
Same Database, Same Prompts

PostgreSQL MCP (v1)

- LLM guesses schemas
- Inconsistent results
- `SELECT` everywhere
- `ALTER TABLE` was valid
- **Data lost**

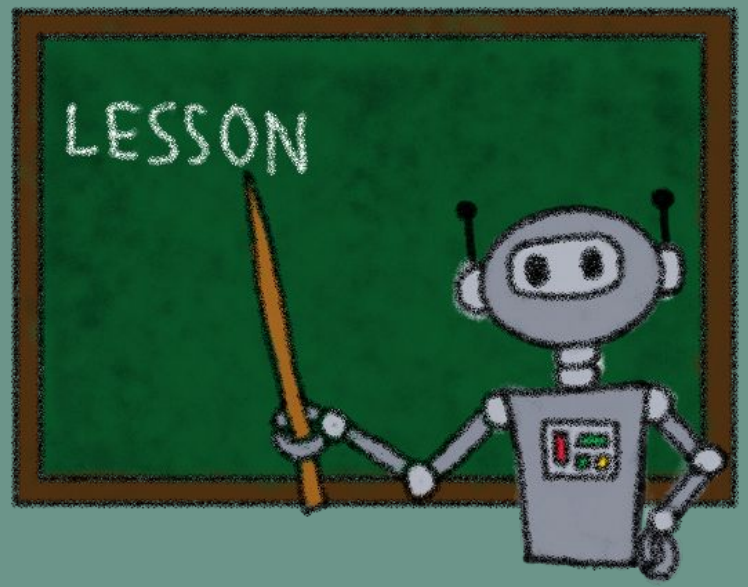
Purpose-built (v2)

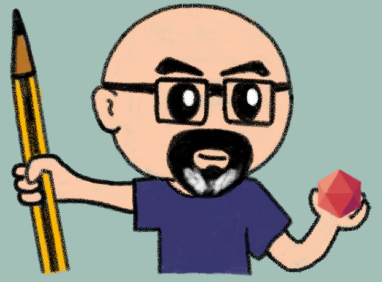
- LLM reads resources first
- Consistent, typed calls
- Minimal data returned
- Only allowed operations
- **Data safe**



The Lesson

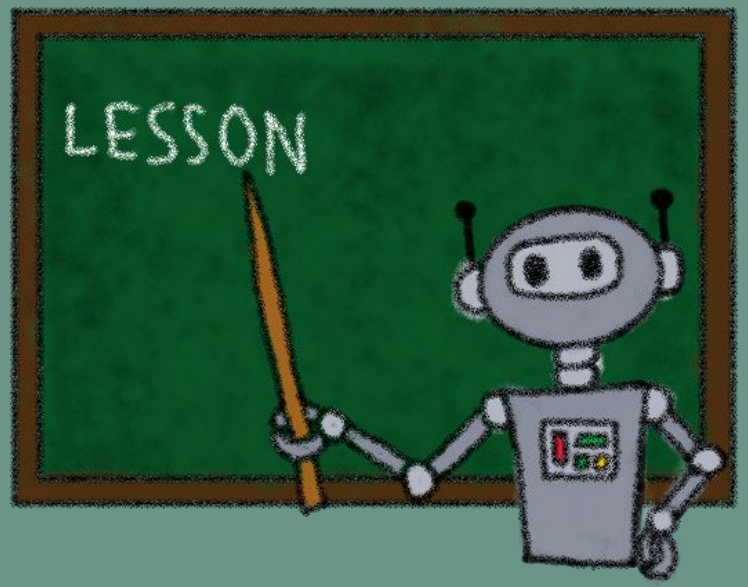
Learnt in the hard way...

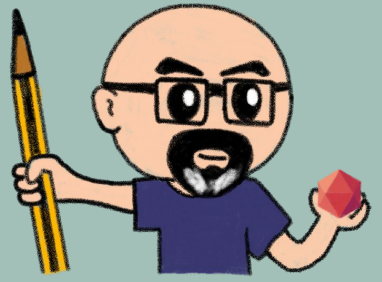




DEVOXX France

MCP servers are APIs





Design them like APIs

DEVOXX France

That's What Next Parts are About

A framework for API design discipline in MCP

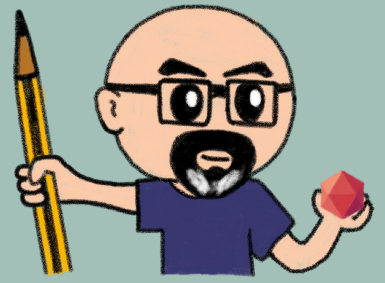
- **v1**
  MCP works
- **v2**
  MCP is shaped
- **v3**
  MCP scales
- **v4**
  MCP is governed

Climbing the ladder = getting better at API design

DEVOXX France

The Maturity Ladder

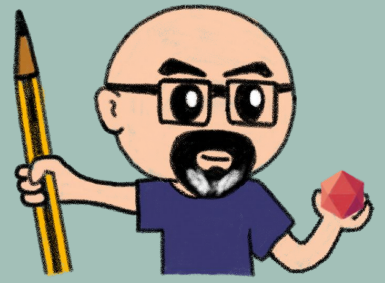
The spine of Part 2



DEVOXX France

The Maturity Ladder

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



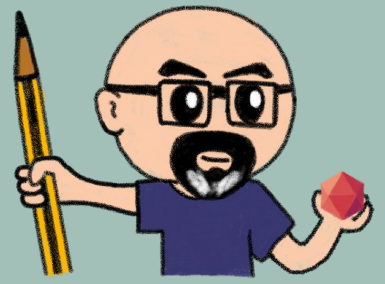
DEVOXX France

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.

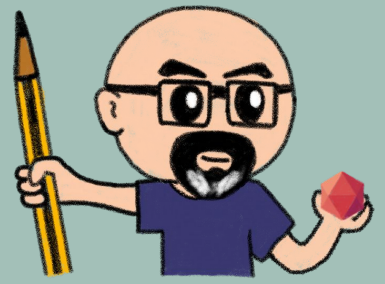


DEVOXX France

How to Climb

- **v1 → v2: shape it**
  Typed tools, Resources, Prompts, validation
- **v2 → v3: scale it**
  OAuth 2.1, gateway, registry, contracts
- **v3 → v4: govern it**
  Policy, audit, risk tiers, pluralism

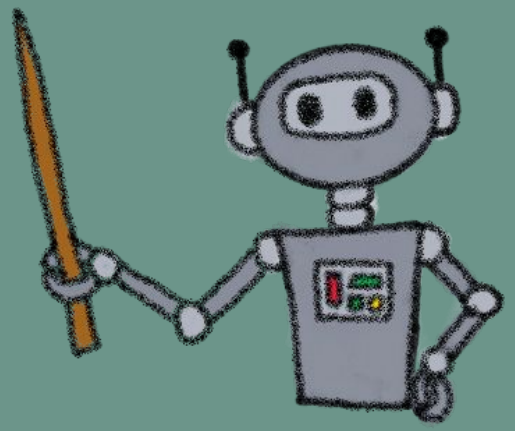
Each part of the talk will helps you climb one rung.



DEVOXX France

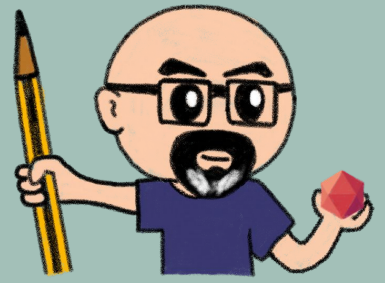
What You'll Leave With

- The **maturity ladder**, where you are, what's next
- **Real opinions** on design choices
- **Real practices** you can apply Monday

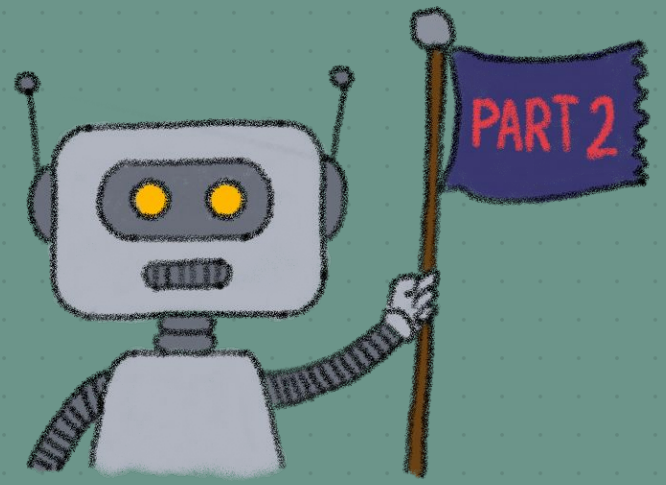


DEVOXX France

Let's take a BREAK !

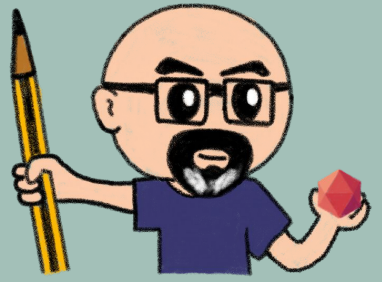


DEVOXX France



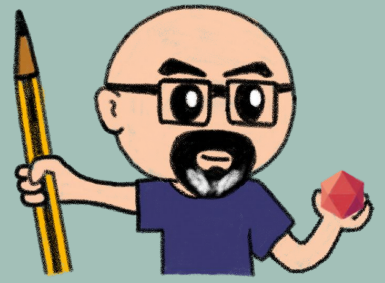
Part II – Shaped

RAGmonsters grows up... a bit



Before the break, we saw one design decision explode into five production problems

Now let's see what fixing it actually meant



The rebuild worked

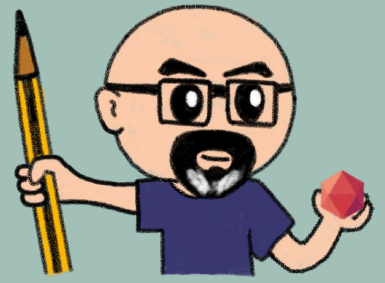
But as I did it, I realised there are several things you have to shape

Most of them I didn't get right the first time

DEVOXX France

What "Shape" Means

- Every primitive used **deliberately**
- Every byte of metadata **trustworthy**
- Every input **validated**
- Every output **scrubbed**

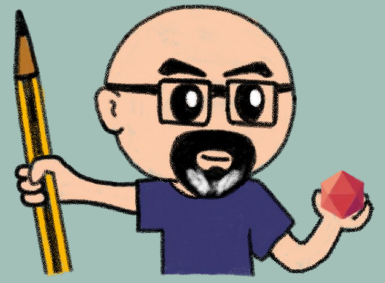


DEVOXX France

The Lessons the Rebuild Taught Me

- Primitives done well
- Input validation
- Output sanitization
- Tool description hygiene
- Testing the server
- A new shape: Code Mode

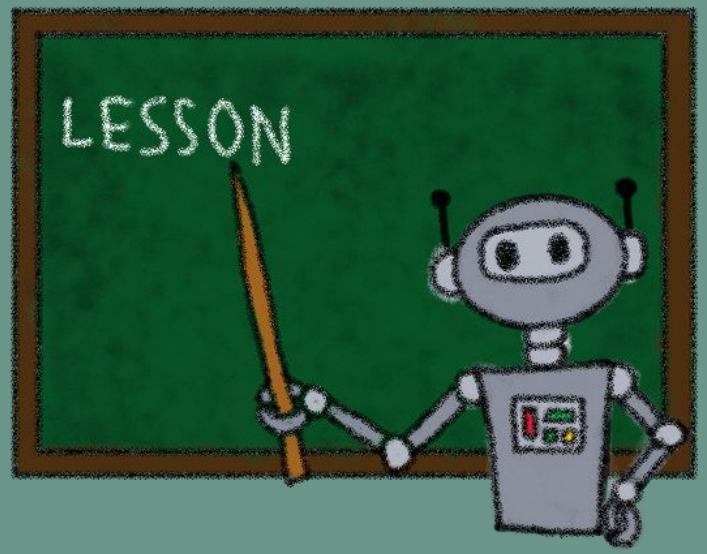
One lesson at a time

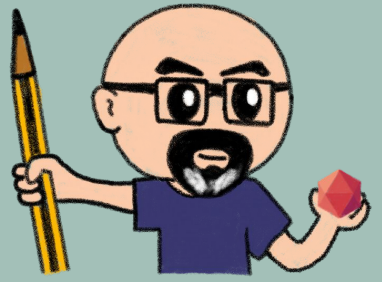


DEVOXX France

Use all the primitives

What Part 1 skipped





DEVOXX France

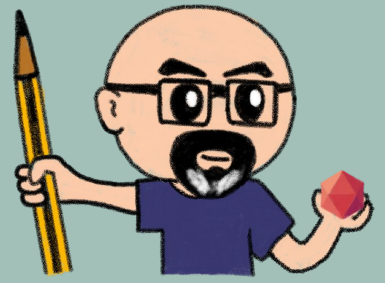
The Seven Primitives, Revisited

Server primitives (introduced in Part 1):

- **Tools:** what the LLM does
- **Resources:** what the LLM reads
- **Prompts:** what the LLM is guided to do

Client primitives (teased in Part 1, unpacked now):

- **Sampling:** server drives reasoning
- **Elicitation:** server asks the user
- **Tasks:** call-now, fetch-later
- **Logging:** structured notifications



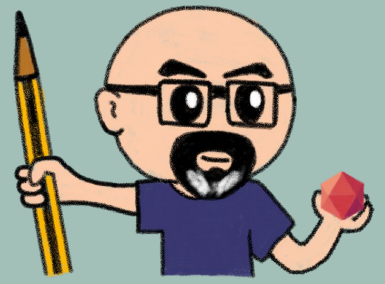
DEVOXX France

Tools — We Already Know These

We spent Part 1 on Tools

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

So we start where Part 1 left off :
with the primitives most teams never touch.

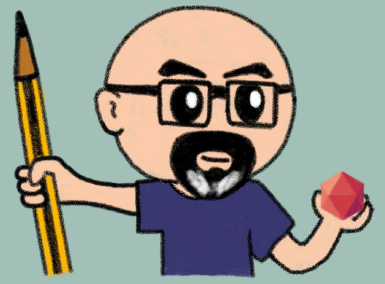


DEVOXX France

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

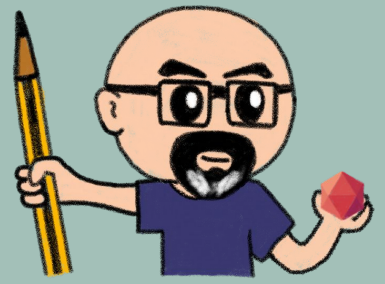


DEVOXX France

The Guessing Problem (Callback)

Remember RAGmonsters v1:

- LLM invented column names
- LLM made joins I never intended
- LLM burned tokens on `information_schema` queries



DEVOXX France

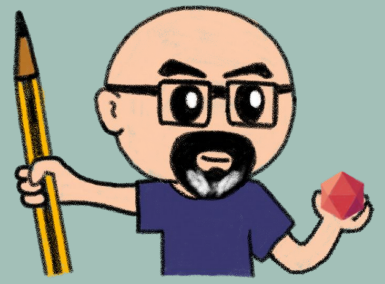
Resources as the Answer

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"]
```

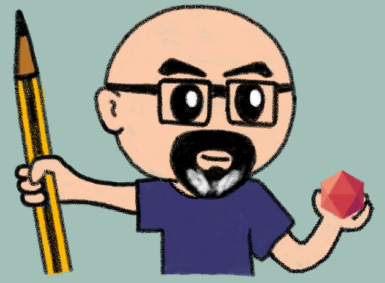


DEVOXX France

Resources — Why You Should Care

- Eliminates unnecessary tool calls
- Grounds the LLM in what actually exists
- Prevents invention of phantom schemas
- Lowers token cost per task

Cheap to add, large payoff

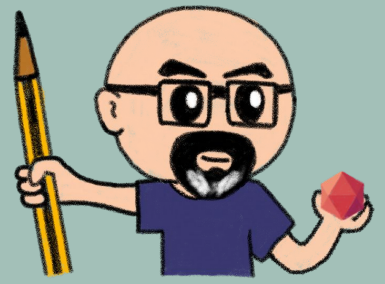


DEVOXX France

Prompts — The Workflow Primitive

What servers **guide** the LLM to do

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



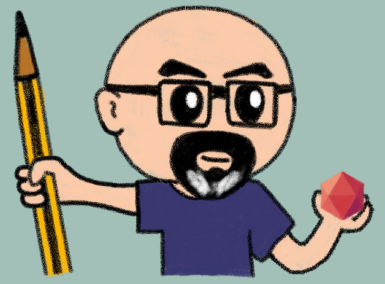
DEVOXX France

The Multi-Step Problem

LLMs improvise multi-step workflows

- Sometimes brilliantly, sometimes disastrously
- Always differently each time

Improvisation ≠ repeatability

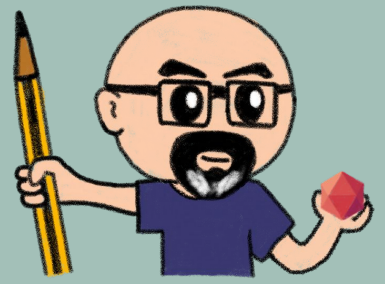


DEVOXX France

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
```



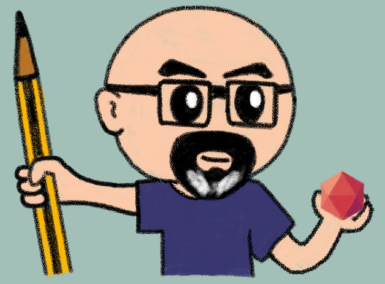
My recommendation: treat **Prompts as contracts**

DEVOXX France

Prompts as Codified Workflows

RAGmonsters MCP

```python
@mcp.prompt()
def analyze_monster_weakness(monster: str) -> list[Message]:
    return [
        Message(role="system", content="You are a monster analyst…"),
        Message(role="user",
                content=f"Analyze {monster}'s weaknesses using "
                        "resource://types and the weakness_map tool."),
    ]
```

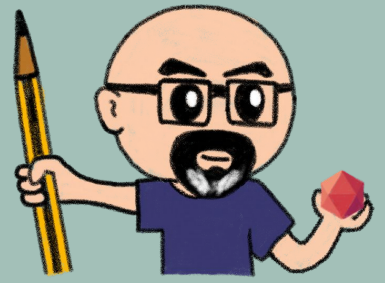


DEVOXX France

Prompts — Why You Should Care

- Deterministic multi-step workflows
- Reusable across clients
- Versionable (like API endpoints)
- Testable (we'll come back to golden tasks later)

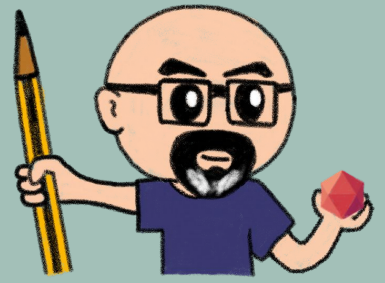
Ship the playbook, not just the atoms

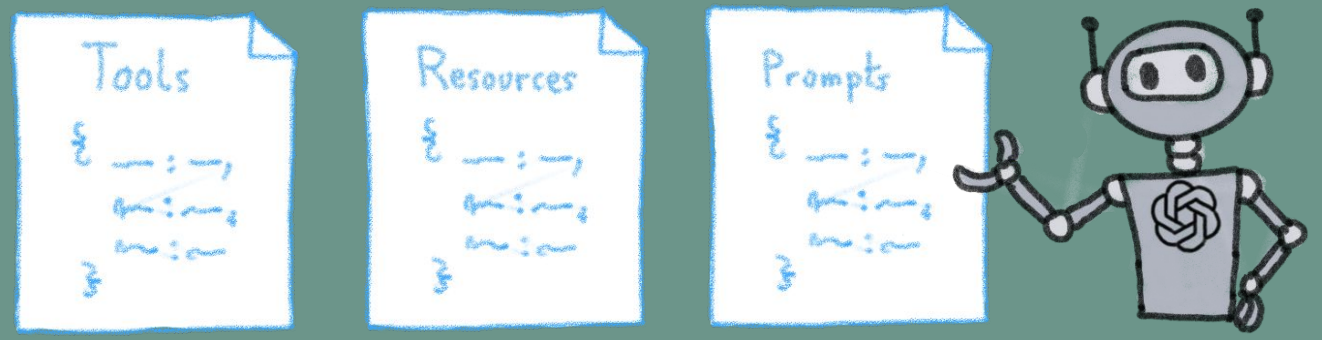


DEVOXX France

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`





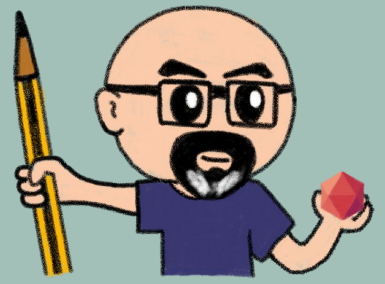
DEVOXX France

Composing Primitives

Example workflow:

a. LLM reads `resource://monsters/categories`
b. User asks "*compare fire and water monsters*"
c. LLM uses `prompt://compare_monsters`
d. Prompt guides LLM to call `search_monsters_by_category` twice
e. LLM structures comparison per prompt template

The power comes from combining them



DEVOXX France

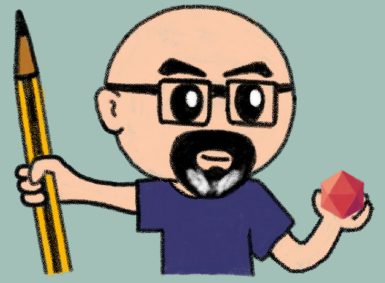
Tool Discoverability at Scale

Problem: 50 tools across 8 servers
How does LLM know what's available?

Solution: Capability index resource
`resource://capabilities`

- Tool list with descriptions
- Risk level per tool
- Required roles
- Cost/latency hints

Helps both LLMs and humans understand the surface



DEVOXX France

Caching and "Resource Mirrors"

- **Problem:**
  Expensive reads repeated constantly
- **Solution:**
  Use Resources for reference data + cache

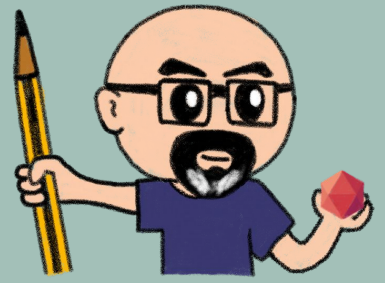
  `resource://monsters/types` → Cache 1 hour

  `resource://config/limits` → Cache 5 min

  `tool://search_monsters` → No cache (dynamic)

- Reduces latency and token churn

Resources are naturally cacheable, Tools usually aren't



DEVOXX France

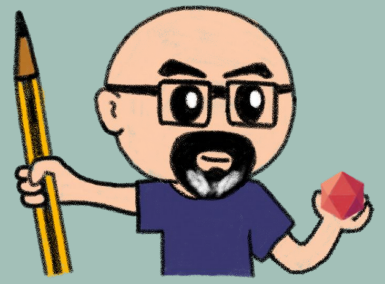
Design Principle: Right Primitive, Right Job

Dos:

- Match primitive to access pattern
- Compose primitives for complex workflows

Don'ts:

- Don't use Tools for static data → add Resources instead
- Don't embed workflows in tool descriptions → add Prompts instead
- Don't use Resources for dynamic data → add Tools instead



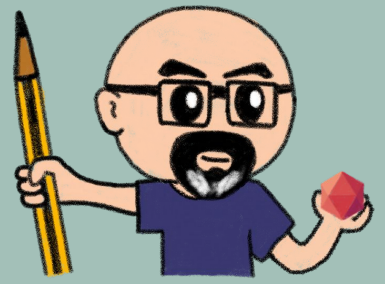
DEVOXX France

RAGmonsters v2 - Using All Three

- **Tools**:
  `getMonsters`, `getMonsterById`, `getBiomes`, `getRarities`, `getMonsterByHabitat`, `getMonsterByName`, `compareMonsters`

- **Resources**:
  `ragmonsters://schema`, `ragmonsters://categories`, `ragmonsters://subcategories`, `ragmonsters://habitats`

- **Prompts**:
  `analyze_monster_weakness`, `compare_monsters`, `explore_habitat`, `build_team`



DEVOXX France

Impact on UX

Before (tools only):

User: "What types of monsters exist?"

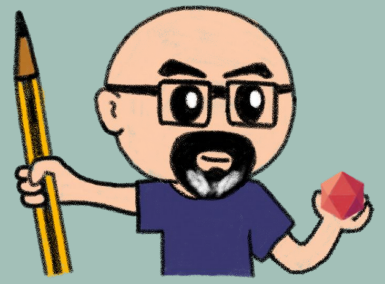
LLM: Guesses, maybe calls query with wrong SQL

After (with resources):

User: *"What types of monsters exist?"*

LLM: Reads resource://types, responds instantly with accurate list

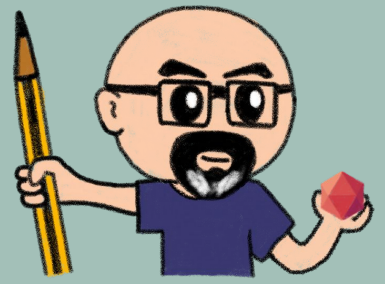
No database query needed, instant response



DEVOXX France

Client-side primitives

Almost ready for primetime



DEVOXX France

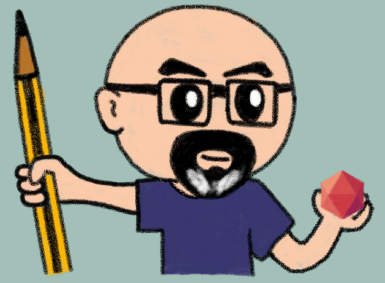
Sampling — The Role Reversal

Normally: the client calls a server tool

Sampling: the server asks the client's LLM to run a completion

- The server pauses mid-task
- Asks the LLM to reason about something
- Resumes with the answer

The server is no longer a passive endpoint.



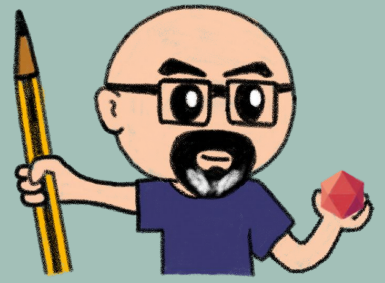
Human-in-the-loop by design, the user can see, edit, or reject every request.

DEVOXX France

What Sampling Unlocks

- Server-orchestrated agentic loops
- Tool sequencing from the server side
- Workflow control without client changes

The server becomes an **active participant**, not just an endpoint.



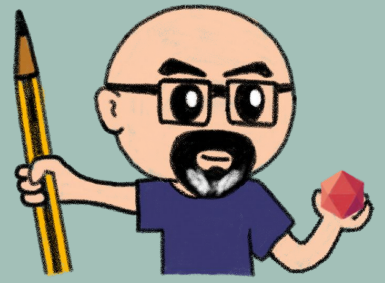
DEVOXX France

Sampling — The Honest Caveat

Requires a **capable client**

- Sampling itself needs the `sampling` capability
- Tool use in Sampling needs the `sampling.tools` capability on top
- Most clients declare one, few declare both in Q1 2026

Direction of travel: adoption is real, not there yet.



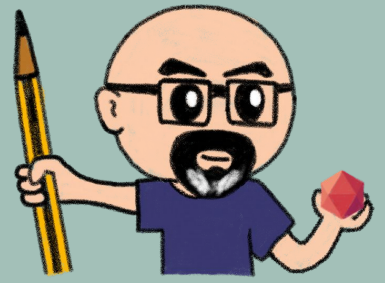
DEVOXX France

Elicitation – Asking the User

Servers can request input from the user mid-execution

Two modes:

- **Form mode:**
  structured data via a JSON Schema,
  in-band through the client
- **URL mode:**
  user completes the flow in their browser,
  out-of-band (new in 2025–11–25)



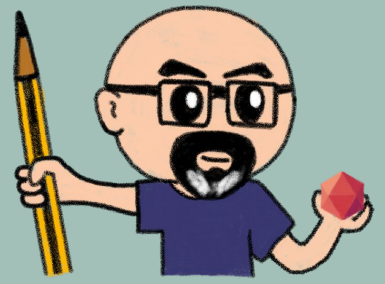
DEVOXX France

Elicitation URL Mode

The security and UX win:

1. Server emits a URL
2. Client opens the user's browser
3. User completes OAuth flow out-of-band
4. Token returns to the server

Neither the MCP client nor the LLM touches credentials

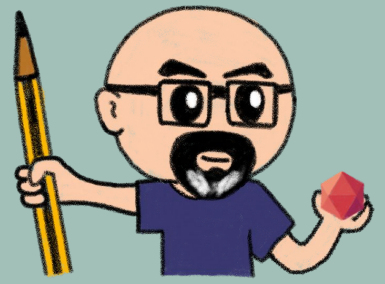


DEVOXX France

Elicitation — Why URL Mode Matters

- Unblocks proper OAuth flows
- No custom credential handling per client
- Works across every MCP client

Small in appearance, big in practice



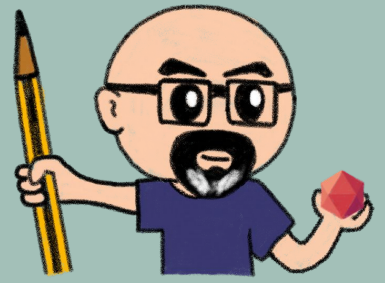
DEVOXX France

Tasks — Call-Now, Fetch-Later

For work that doesn't fit a single round-trip:

- ETL jobs
- Long conversions
- Batch analysis
- Multi-step provisioning

Experimental since 2025-11-25



DEVOXX France

When to Use Tasks... and When Not

Use Tasks for:

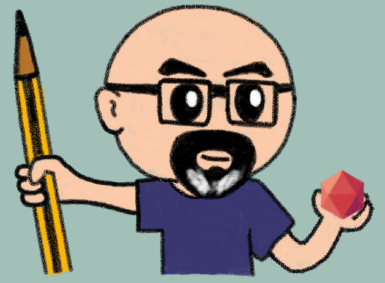
- Work > 10 seconds
- Work the client can't meaningfully wait on
- Batch jobs the agent triggers and checks later

Examples: generate a report / provision infra

Don't use Tasks for:

- Simple queries (Tools are faster)
- Sub-second work (Tasks overhead swamps the gain)
- Anything the client can wait for synchronously
-

My recommendation: default to Tools,
promote to Tasks only when sync breaks



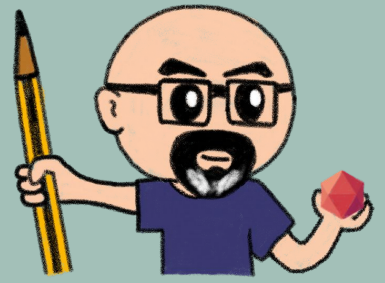
DEVOXX France

Tasks — Still Evolving

Worth watching: lifecycle semantics are still moving

- Retry behaviour
- Expiry policies
- Failure signalling

Use in anger cautiously.



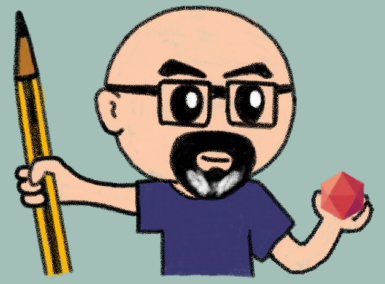
DEVOXX France

Logging — The Observability Hook

Servers send structured log messages to the client

- Server → client, via `notifications/message`
- Levels, a logger name, arbitrary JSON payload
- Part of the protocol, not `stderr`, not `print()`

Observability wired into MCP itself

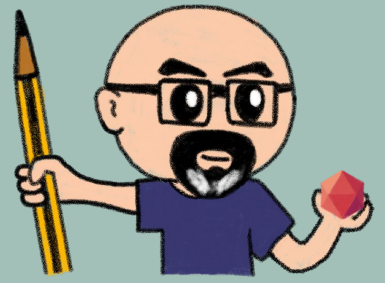


DEVOXX France

Logging — Why You Should Care

- Observability that travels with the agent, not beside it
- Client-controlled verbosity — dial it up when debugging
- Spec is explicit: no credentials, no PII in log payloads

We come back to this when we talk about observability.



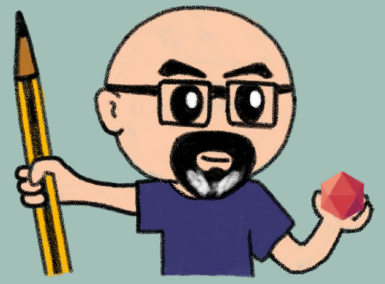
Logging — The Honest Caveat

The spec says what **servers emit**

But it is silent on what **clients do** with it

- Every client receives the notifications
- Few surface them to developers or ops by default
- The official clients matrix doesn't even track Logging

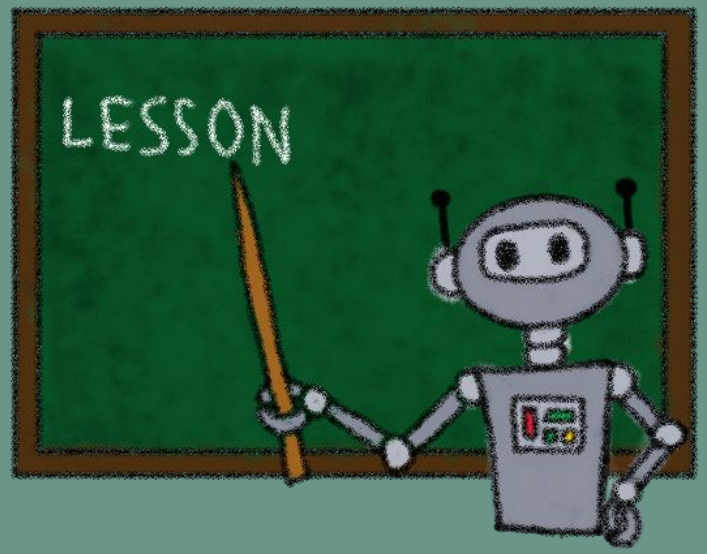
My recommendation: emit the notifications anyway, but **check your client** before relying on them

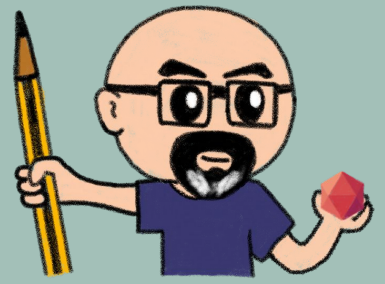


DEVOXX France

Validate every input

The LLM is not a trusted caller





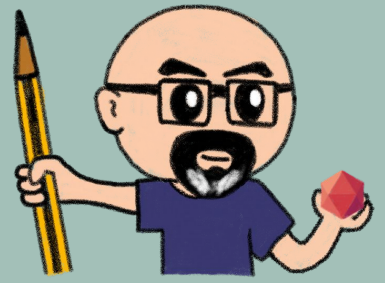
DEVOXX France

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.

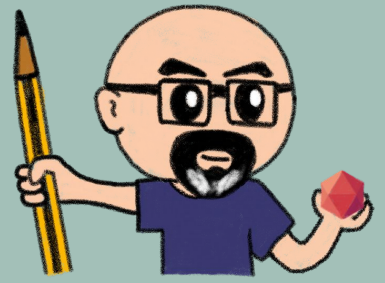


DEVOXX France

Enum-Validated Types, Revisited

RAGmonsters MCP

```
server.tool("search_monsters_by_type", {
  type: z.enum(["fire", "water", "earth",
                "air", "shadow", "crystal"])
}, async ({ type }) => { ... });
```



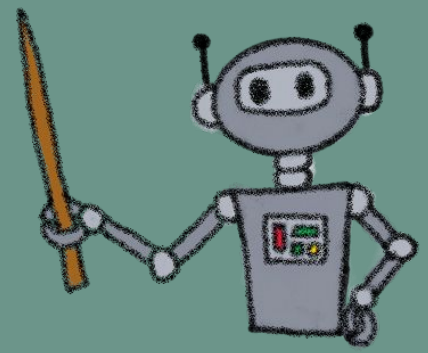
The LLM **cannot invent** a seventh type

DEVOXX France

A lesson to remember

Think of every LLM input as hostile

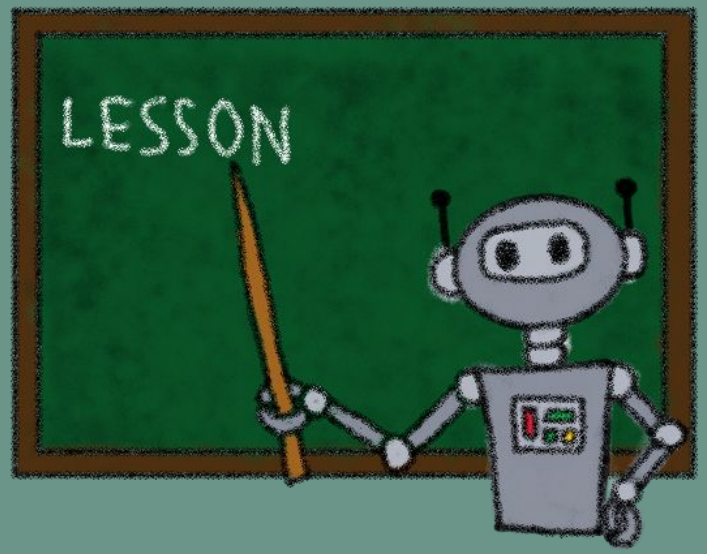
Even when it isn't



DEVOXX France

Sanitize every output

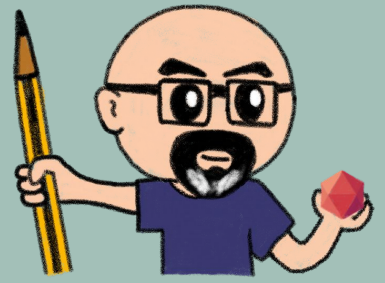
What you return, the LLM reads



A sudden realization

The LLM reads everything I return

and add it to the context!



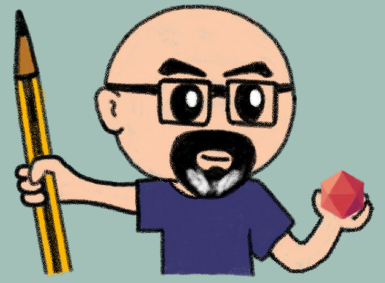
DEVOXX France

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



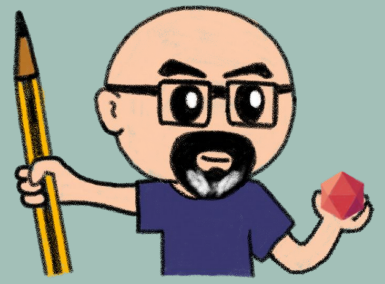
DEVOXX France

Structured Outputs

- **Stable JSON** shapes reduce agent hallucination
- **Inconsistent formats** → parsing errors → retries → cost

```
// ✗ Sometimes returns { "monster": {...} }, sometimes { "data": {...} }
// ✅ Always returns { "result": {...}, "metadata": {...} }
```

- Document your **output schemas**
- Consider **JSON Schema validation** on responses

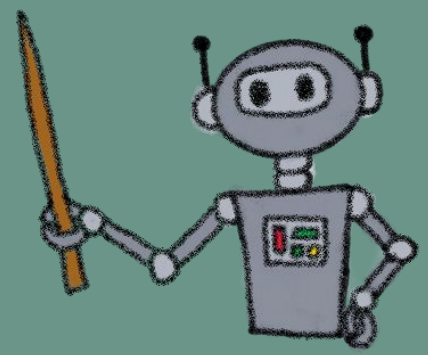


DEVOXX France

A lesson to remember

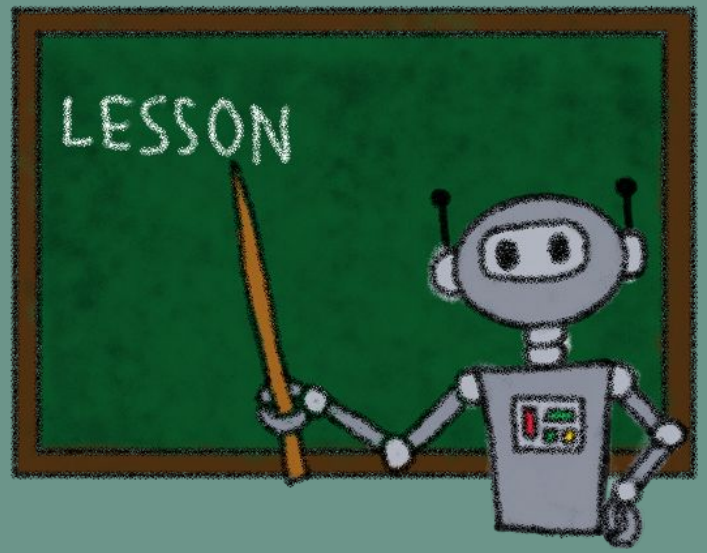
Outputs from your MCP server
are **inputs to your LLM**

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



Authentication is not optional

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



DEVOXX France

Authentication & Authorization

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

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

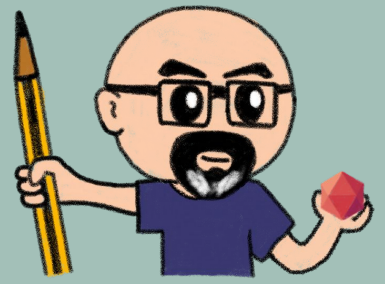
3. **Data-Level Auth**
   Who can see which data?

DEVOXX France

Authentication & Authorization

Example of tool-level auth

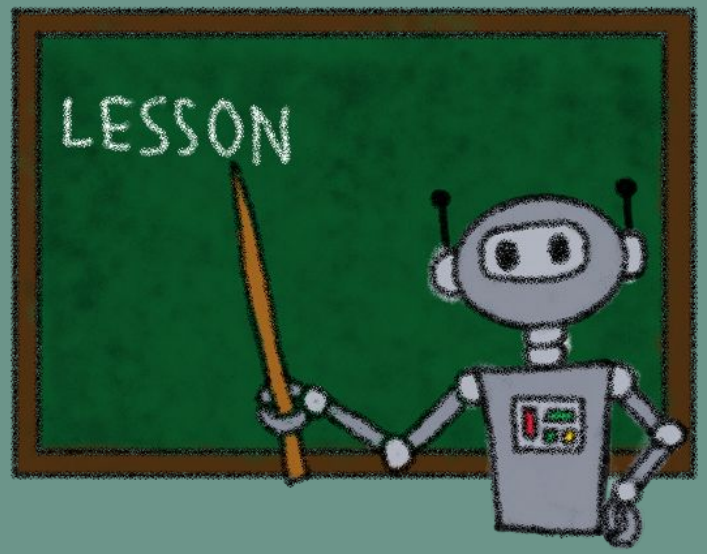
```
// Tool-level: Only admin can delete
if (tool === 'delete_monster' && user.role !== 'admin') {
 throw new Error('Unauthorized');
}


// Data-level: Filter monsters by user's org
SELECT * FROM monsters WHERE org_id = ${user.org_id};
```

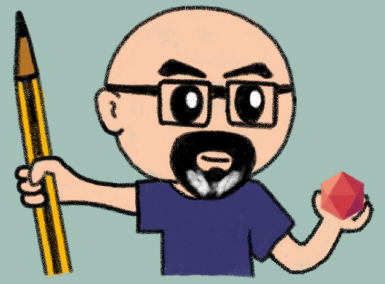


DEVOXX France

Check your tool descriptions

What the LLM sees, and you don't





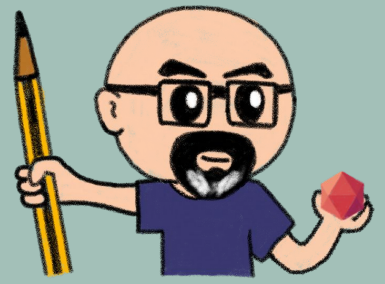
DEVOXX France

Tool Descriptions: Seen, But Not Rendered

The LLM reads tool descriptions **every call**

The UI rarely renders them

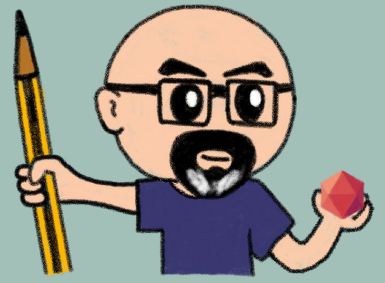
- Invisible to the human user
- Prime target for injected instructions
- The name for this attack: **tool poisoning**



DEVOXX France

Tool Poisoning In Slow Motion

1. Compromised server ships a tool description with hidden instructions
2. LLM reads it during discovery, treats it as its own directive
3. LLM exfiltrates data the attacker asked for

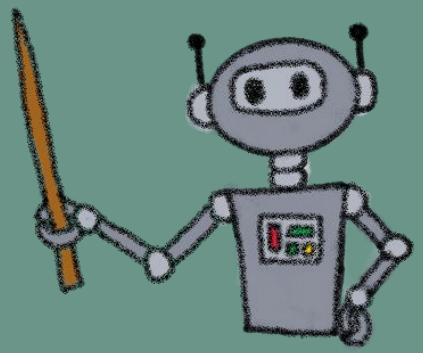


DEVOXX France

A lesson to remember

Never ship a tool whose description you didn't write yourself

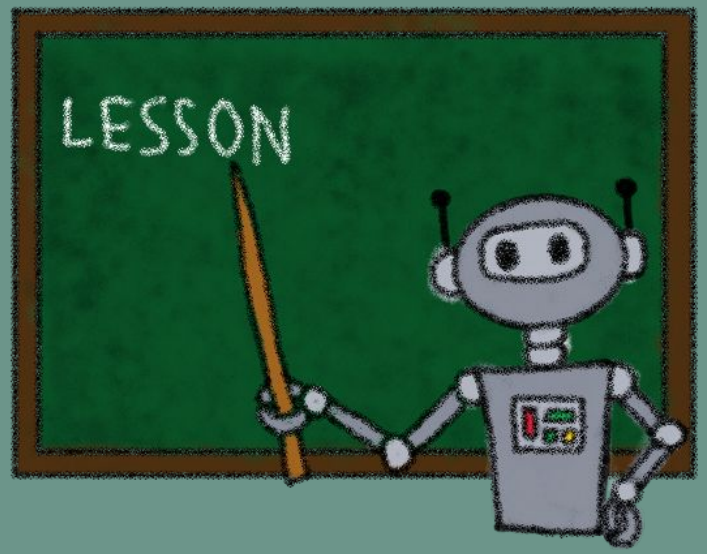
Or at least **checked** extensively

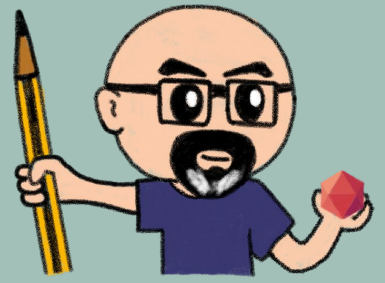


DEVOXX France

Test what the LLM actually does

Unit tests are not enough



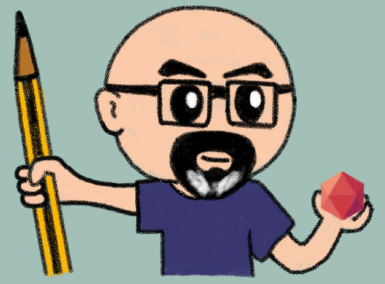


DEVOXX France

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



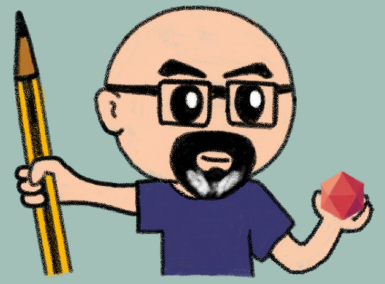
DEVOXX France

Testing Strategy - Three Levels

1. **Unit & Integration Tests**
   - Individual tools work correctly
   - Tools + database work together

2. **LLM Evaluation Tests**
   - Verify real LLM interactions succeed
   - Define **golden tasks**
     A small suite of representative prompts

3. **Safety Tests**
   - Prompt-injection set
   - Over-broad queries
   - Boundary limits



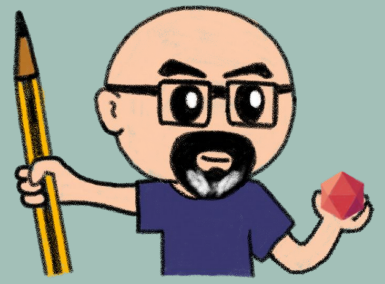
DEVOXX France

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?"*

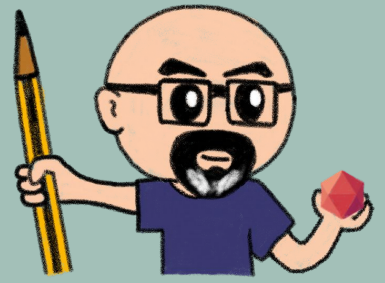


DEVOXX France

Example of Golden Task

RAGmonsters MCP

```
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
```



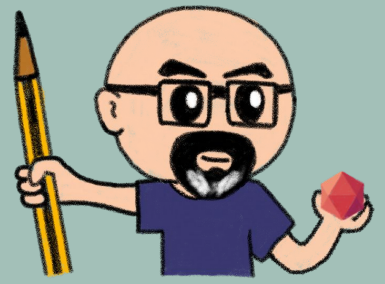
Pattern matters, exact assertions help

DEVOXX France

Safety Tests At the Server Level

- Prompt injection resistance
- Boundary limits (token, recursion, query size)
- Over-broad queries that happen to parse

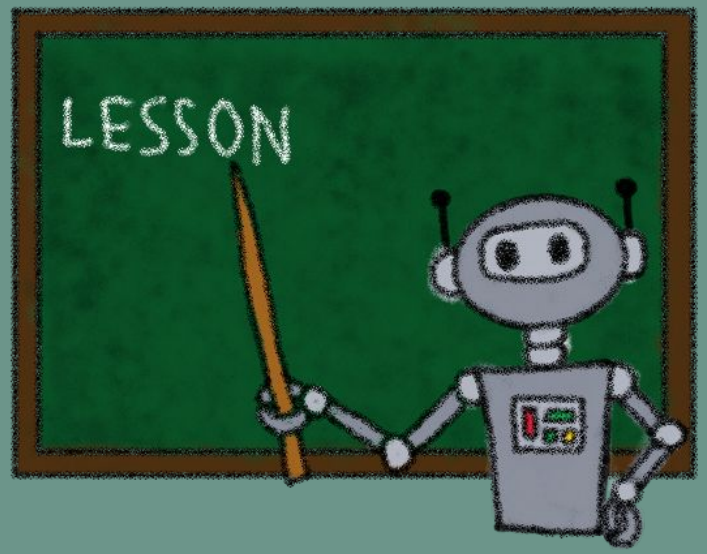
The CI signal you didn't know you needed

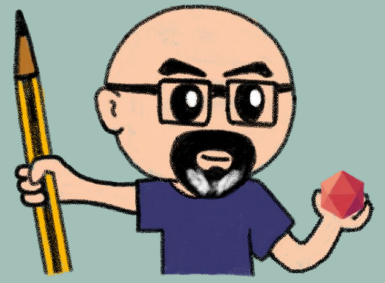


DEVOXX France

Observability

Know What's Happening

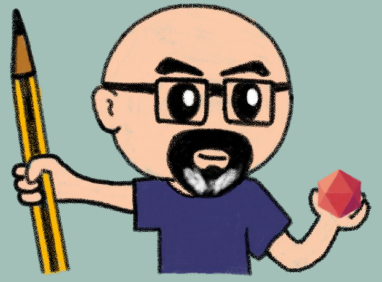




DEVOXX France

Observability - What you need to see

- Which **tools** are being called
- With what **parameters**
- **Success/failure** rates
- Performance (**latency**)
- **Error** patterns

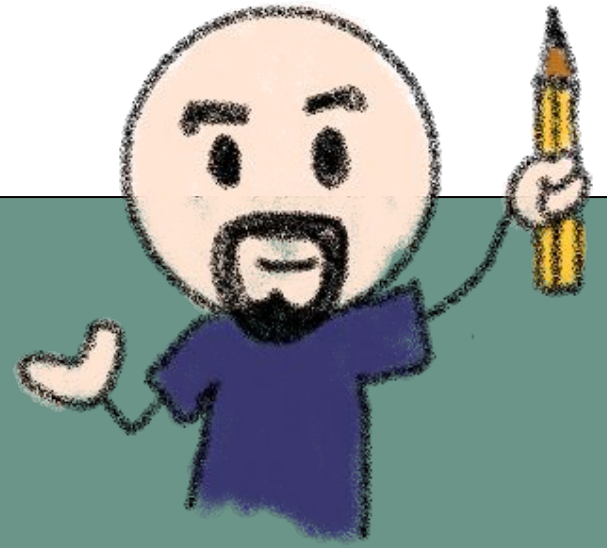


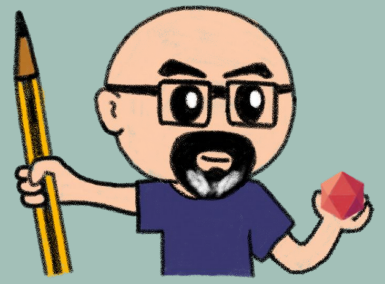
DEVOXX France

Logging Best Practices

Structured logging example

```
// Structured logging
logger.info('Tool called', {
 tool: 'search_monsters_by_type',
 params: { type: 'fire' },
 user: session.user_id,
 timestamp: Date.now()
});
```



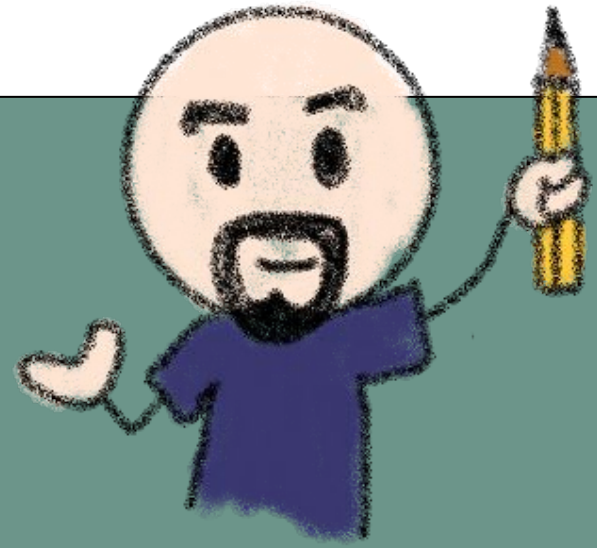


DEVOXX France

Logging Best Practices

Log results example

```
// Log results
logger.info('Tool succeeded', {
 tool: 'search_monsters_by_type',
 result_count: results.length,
 latency_ms: Date.now() - startTime
});
```

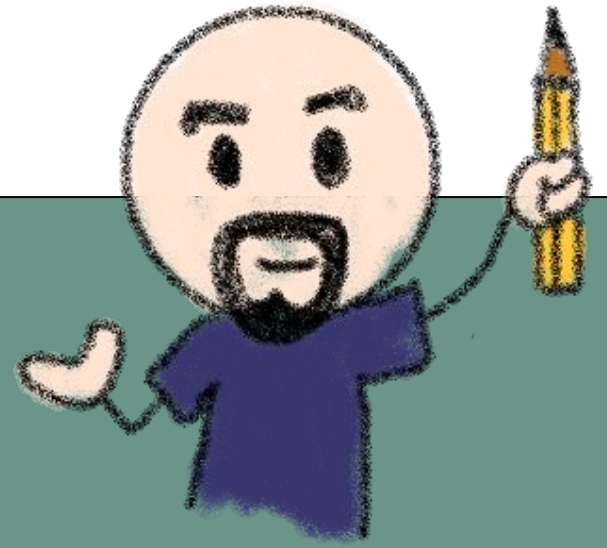


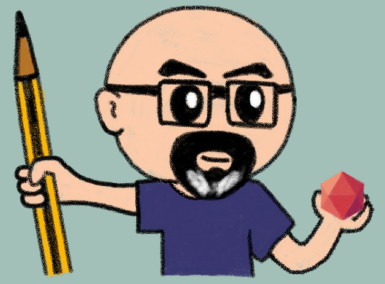
DEVOXX France

Logging Best Practices

Log errors with context example

```
// Log errors with context
logger.error('Tool failed', {
 tool: 'search_monsters_by_type',
 error: err.message,
 params: { type: 'invalid' },
 user: session.user_id
});
```

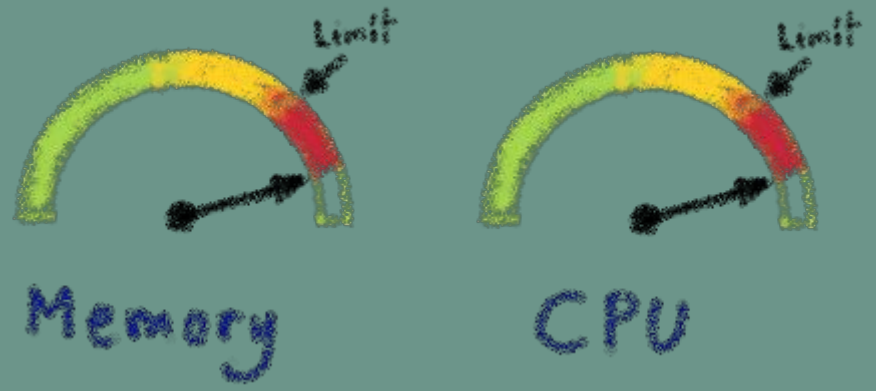




DEVOXX France

Monitoring Dashboard

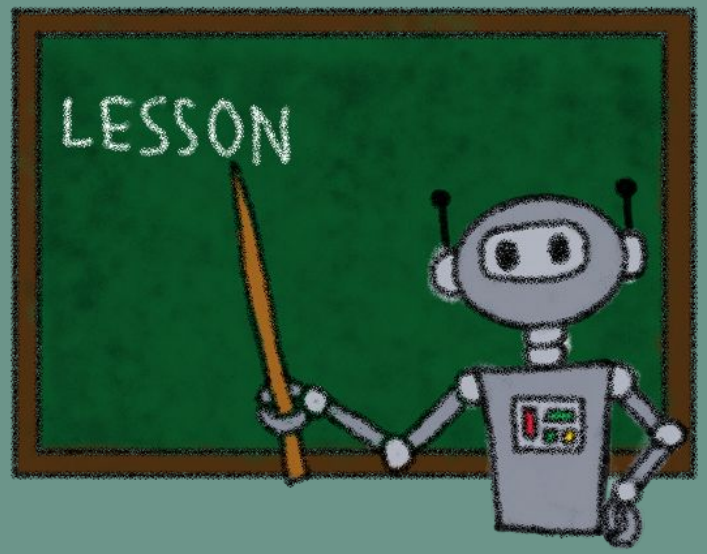
- **Tool call volume** over time
- **Success rate** per tool
- **P95 latency** per tool
- Top **errors**
- Most **active users**



DEVOXX France

One More Thing

A new shape: Code Mode



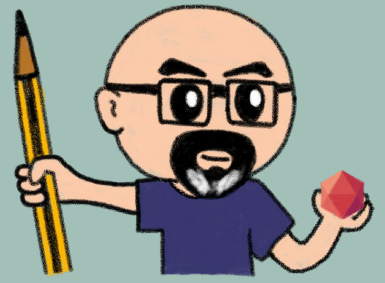
DEVOXX France

Code Mode: An Emerging Pattern

Cloudflare published **Code Mode**

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

CLOUDFLARE



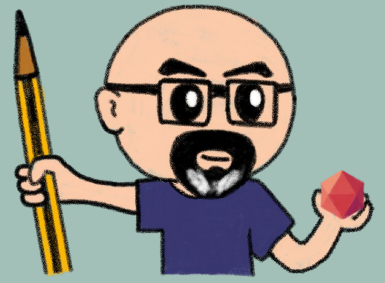
DEVOXX France

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

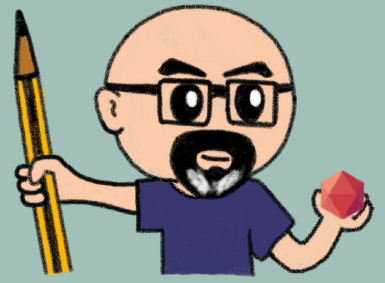
LLMs write code better than they navigate menus



DEVOXX France

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

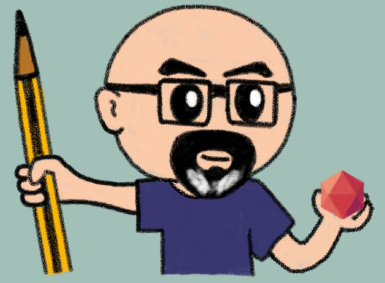


DEVOXX France

Clever Cloud `mcp-simple-server`

One example of implementation in real world

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

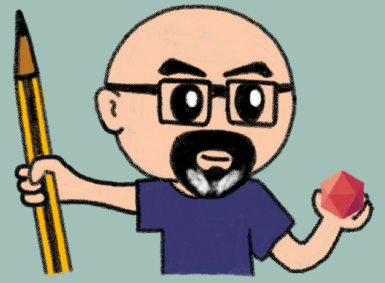


DEVOXX France

So Our Server Is Now Shaped

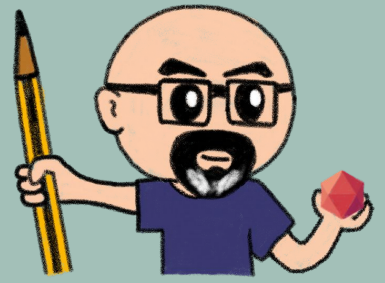
- Every primitive used deliberately
- Every input validated, every output scrubbed
- Every tool description written with intent
- Tested against what the LLM actually does

A single server, production-aware from day one

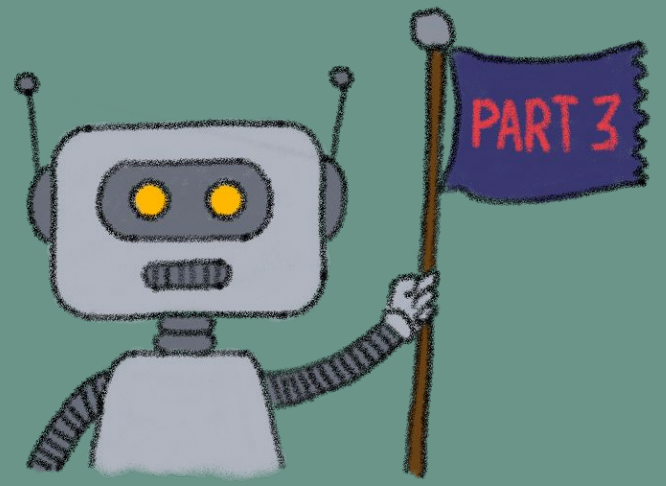


But what's about

it gets popular?

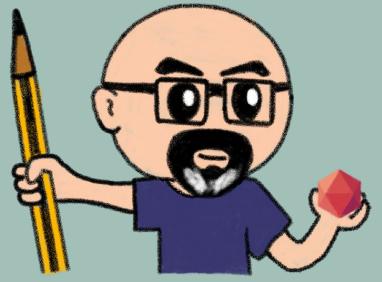


DEVOXX France



Part 3 - Scales

When MCP servers don't stay in their perimeter

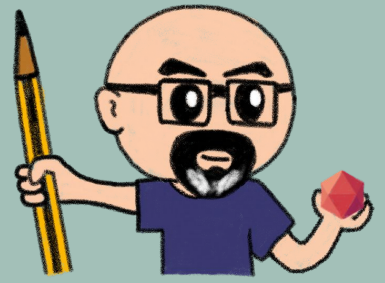


DEVOXX France

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

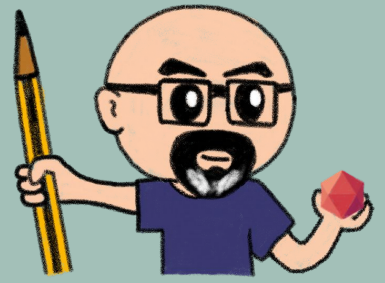


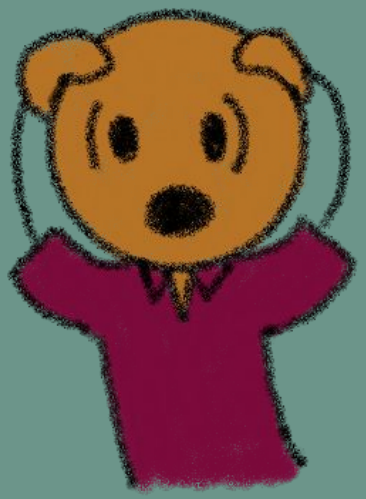
DEVOXX France

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)

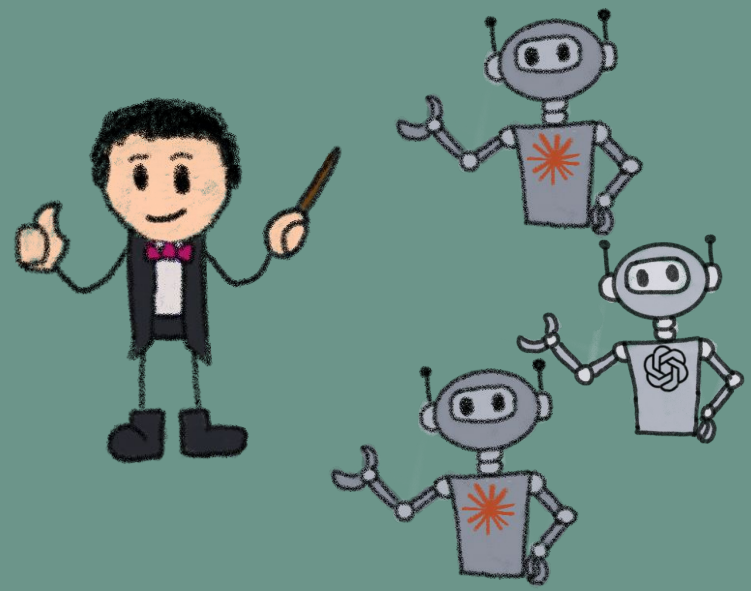




DEVOXX France

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

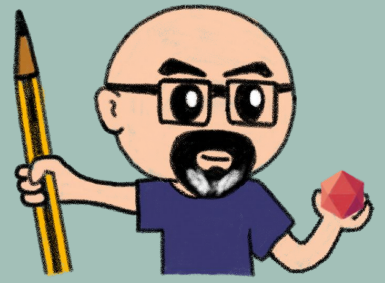
DEVOXX France

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



DEVOXX France

Anti-Pattern: The Mega-Server

One MCP server to rule them all

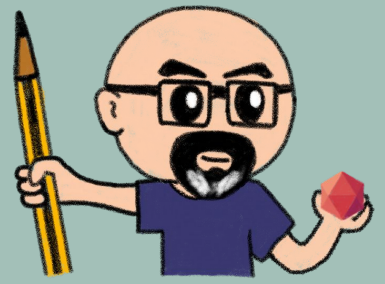
Consequences:

- **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

DEVOXX France

The Key Difference: Stakes Are Higher

Aspect	REST APIs	MCP Servers
Caller	Deterministic code	Non-deterministic LLM
Retry logic	Programmed	LLM-decided
Error interpretation	Code parses	LLM interprets
Autonomy	Human-initiated	Agent-initiated
Blast radius	One request	Autonomous chain



MCP inherits REST lessons,
but the margin for error is smaller

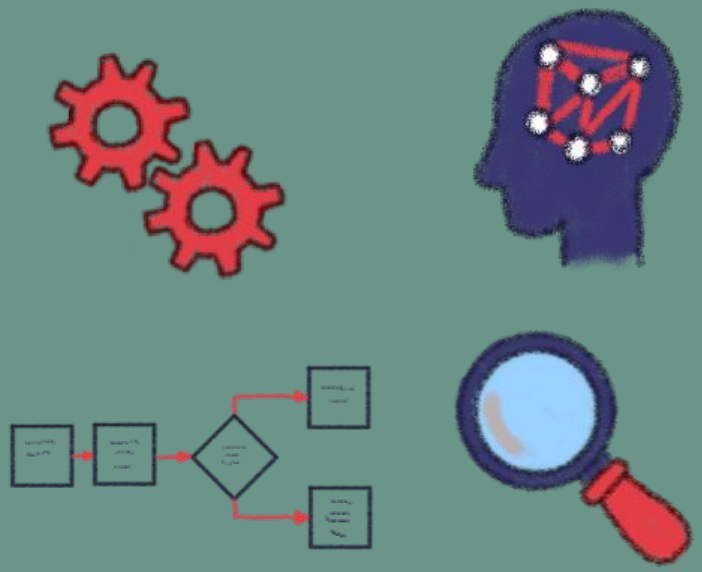
DEVOXX France

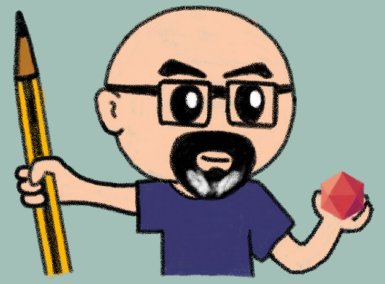
A Mental Model

MCP servers are an API surface for agents

Treat them like **products**:

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



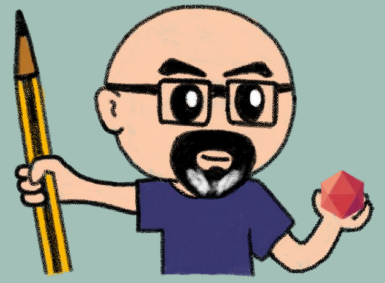


This framing guides the rest of Part 3

DEVOXX France

Rule of Thumb — When to Add What

Situation	Action
Starting out	One domain server, keep it simple
2+ servers	Add consistent naming convention
2+ client types	Add a gateway
Shared multi-step workflows	Consider orchestrator
Expensive repeated reads	Add caching layer

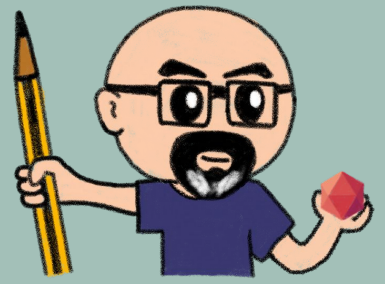


Grow architecture with proven pain, not anticipated pain

DEVOXX France

Anti-Patterns Summary

Anti-Pattern	Problem	Better
Mega-server	Confusion, brittleness	Domain servers
No naming convention	Collisions, unclear intent	`domain.verb_noun`
Gateway with business logic	Tight coupling	Keep gateway thin
Orchestrator for everything	Duplicates agent	Use sparingly
No caching	Latency, cost	Cache Resources

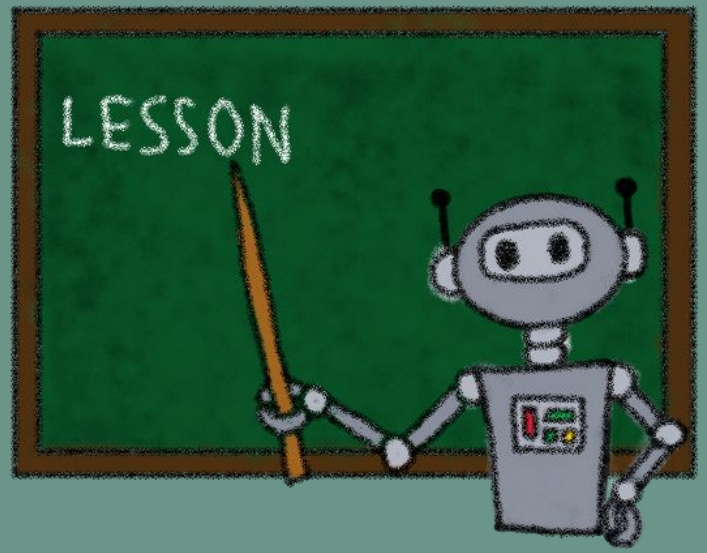


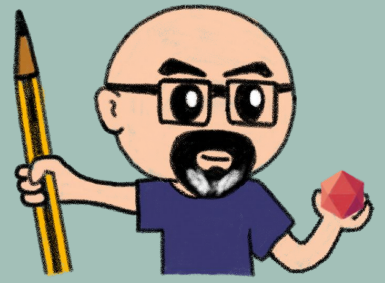
Don't hesitate to reevaluate your choices when your situation evolves

DEVOXX France

Composition Patterns

How multiple MCP servers work together

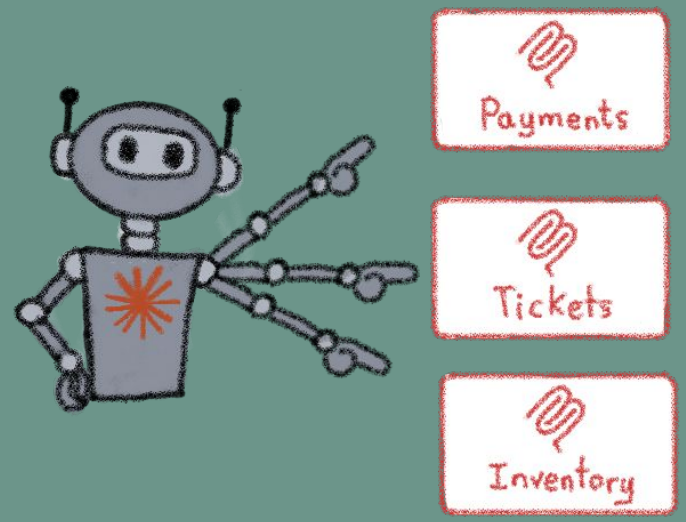


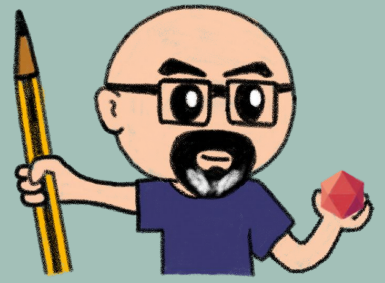


DEVOXX France

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

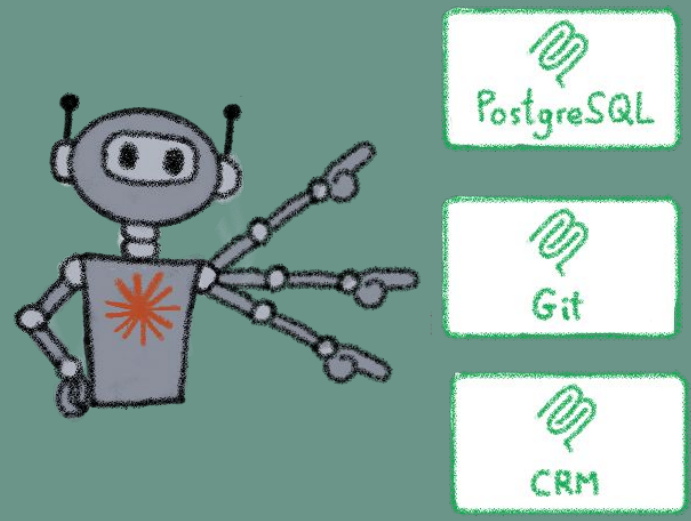


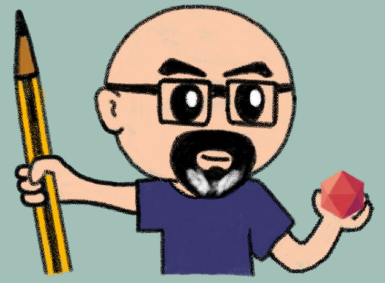


DEVOXX France

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



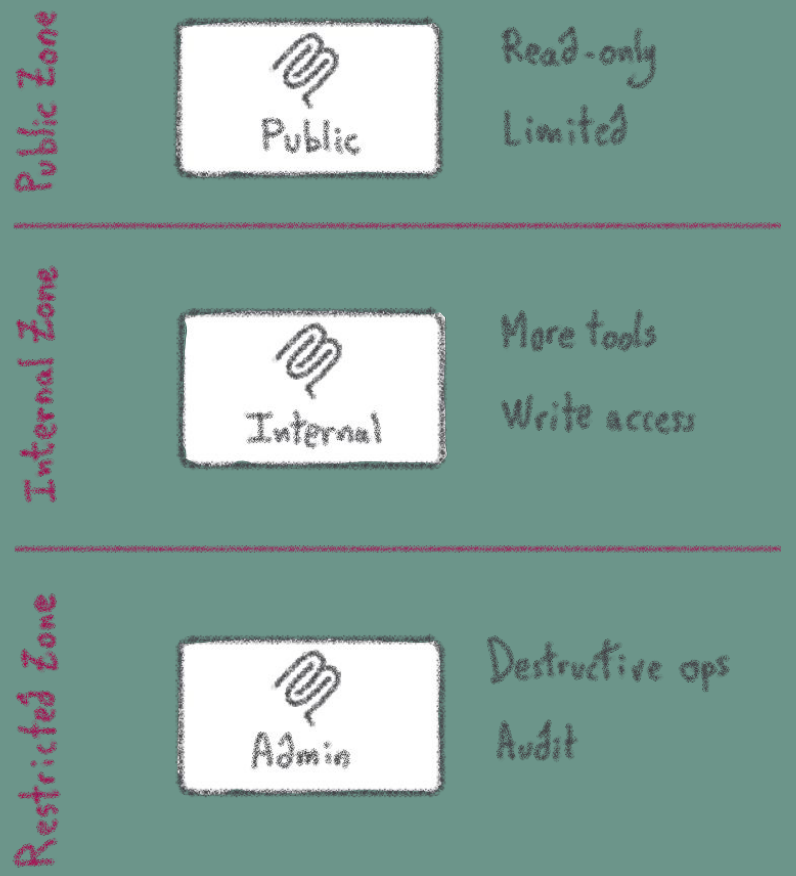


Remember RAGmonsters:
generic → custom as you mature

DEVOXX France

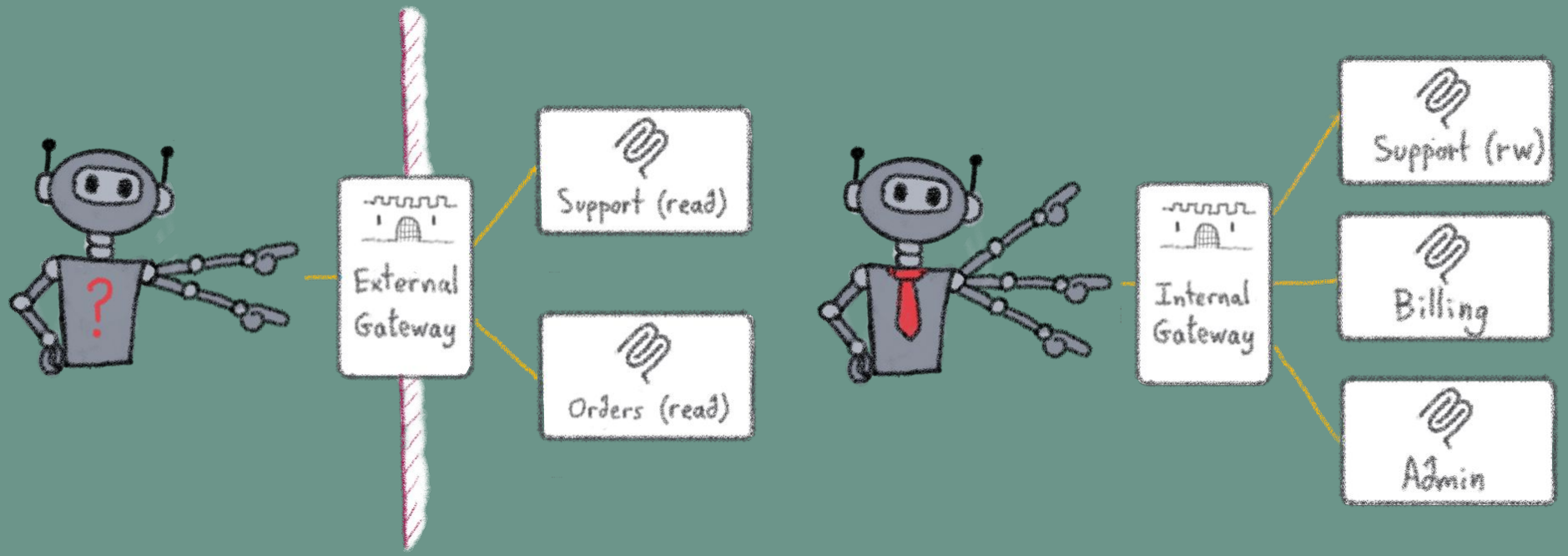
Pattern 3 — Trust-Zone Servers

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



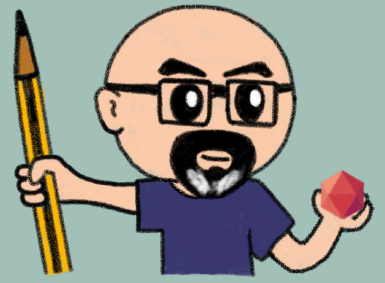
DEVOXX France

Combining Patterns



Domain × Trust = your actual architecture

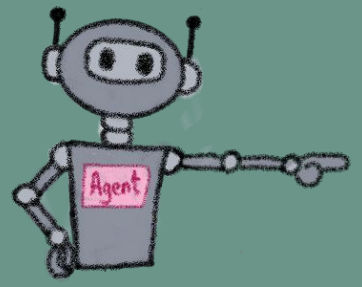
Most organizations end up with a matrix

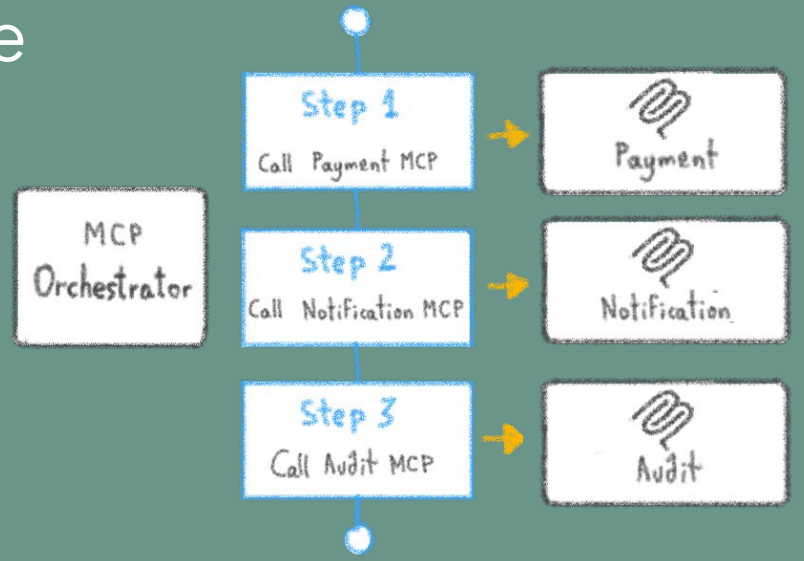


DEVOXX France

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

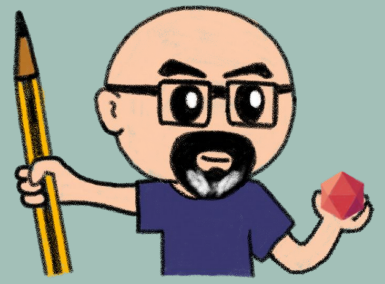
DEVOXX France

Naming and Namespacing

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

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

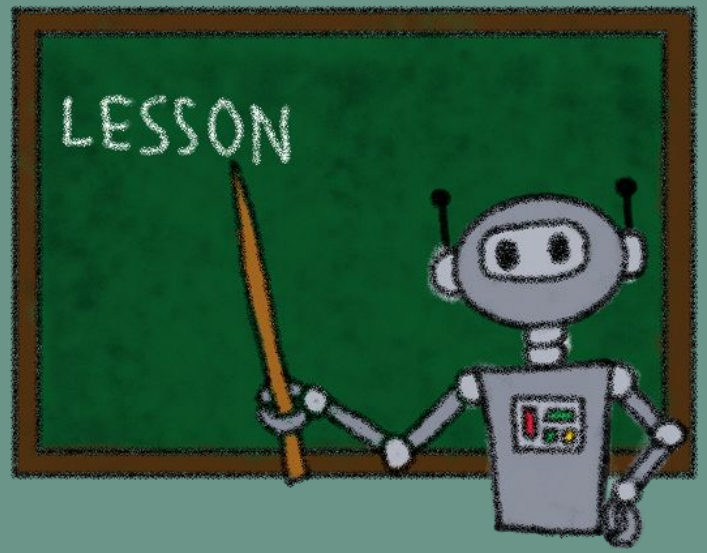
- Avoid collisions across servers
- Keep intent readable for LLMs
- Anti-pattern (meaningless to agents): `doThing, process, handle`



DEVOXX France

Auth stops being a vibe

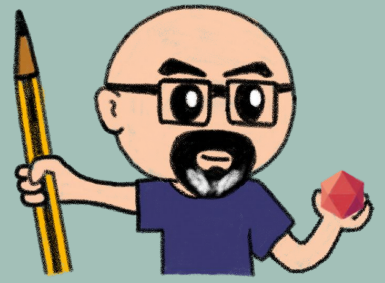
One token, many servers: a problem



I've seen two patterns in the wild:

One token per server… and nobody wants to manage N

Or one token for all, and admin on one means admin on all



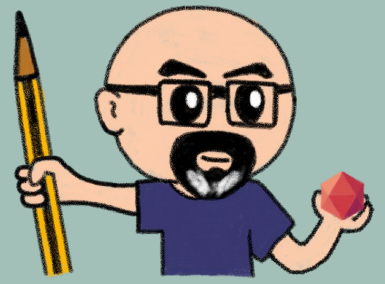
DEVOXX France

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



DEVOXX France

RFC 8707 — Resource Indicators

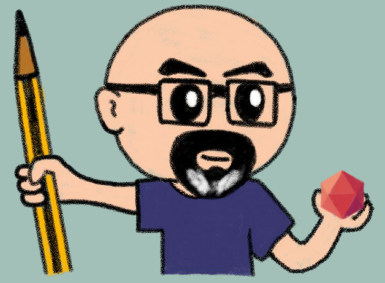
Client names the resource when requesting a token:

```
POST /token
grant_type=authorization_code
resource=https://mcp.monsters.example
```

Token comes back with the resource pinned in `aud`:

```
{
 "aud": "https://mcp.monsters.example",
 "sub": "user-123",
 "exp": 1714867200
}
```

Every MCP server validates `aud` before honouring

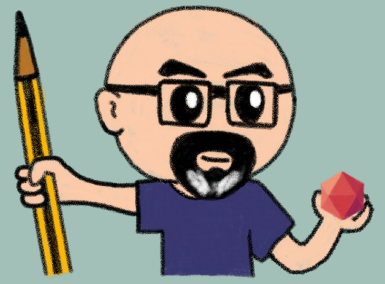


DEVOXX France

Audience-Bound Flow

```
client  →  auth server  →  token (aud = mcp-monsters)
                                  |
                                  ▼
                    mcp-monsters  ← validates, honours
                    mcp-dungeons  ← validates, rejects
```

Audience in the token, validation at the server



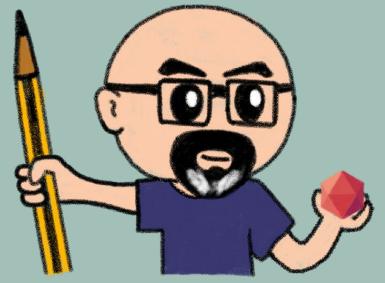
DEVOXX France

What Vendors Are Starting To Do

RFC 8707 needs the **client** to send `resource=`

- The binding depends on it
- **Anthropic, Microsoft, a few others** are shipping it
- **Long tail** is still catching up

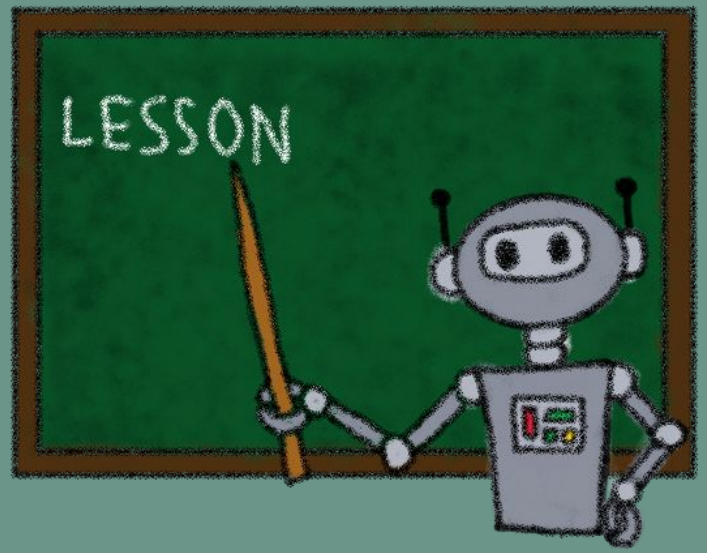
Spec says MUST, verify client before you depend on this

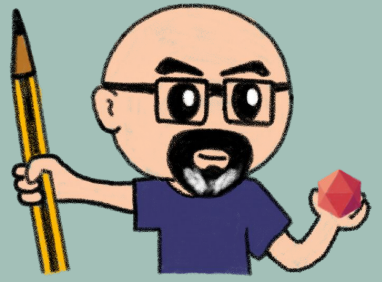


DEVOXX France

Discovery becomes a policy problem

Where agents find what they're allowed to use?





DEVOXX France

The LLM reached for a well-known server name

It pulled a pirate clone from the public internet

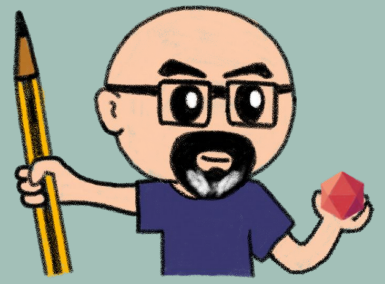
Because the LLM chose it

DEVOXX France

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

Snapshot, not a ranking

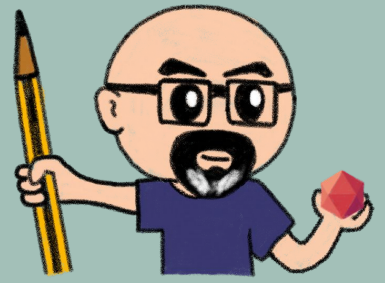


DEVOXX France

The Enterprise Pattern

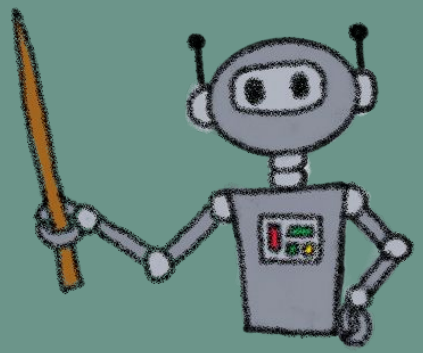
- **Curated internal registry**
- **Allowlist** enforced at the IDE / client
- VS Code custom base URL →
  devs pointed at the right place
- Pirate clones can't be reached

Random-from-internet is no longer a default



Discovery is a policy surface

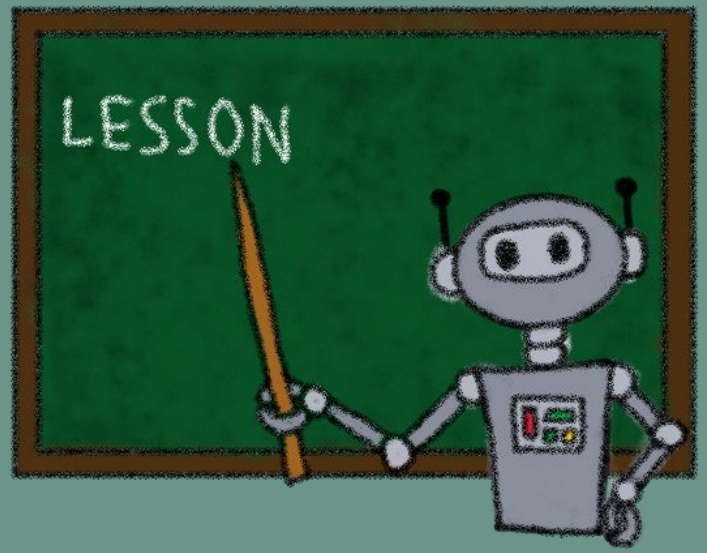
Not a URL list

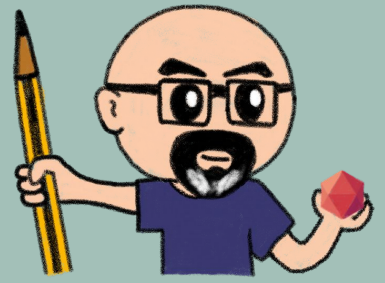


DEVOXX France

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

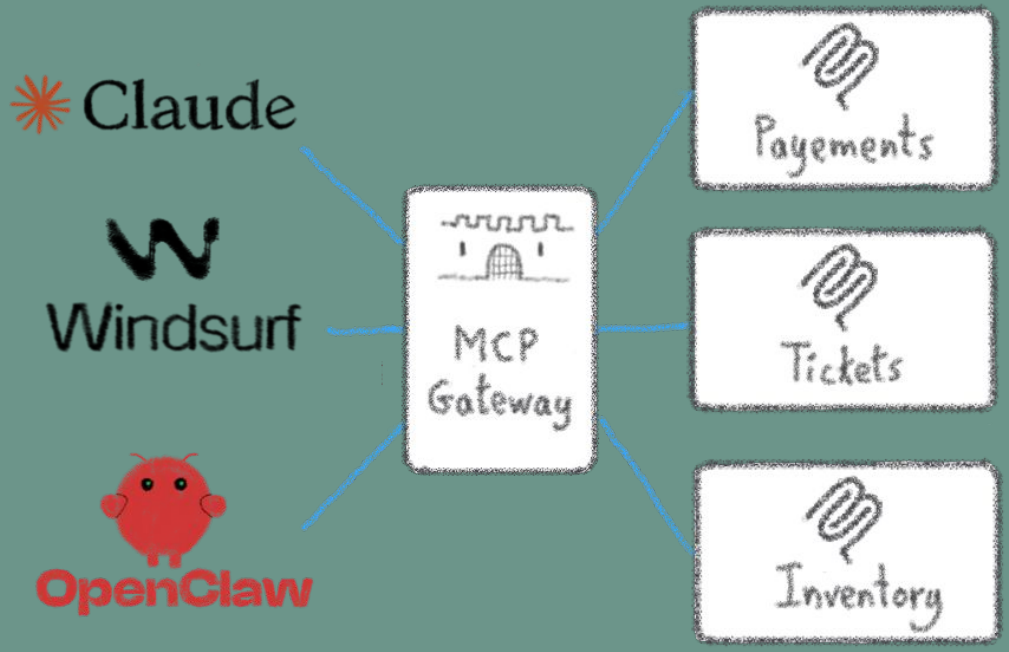
The same wheel, poured six times

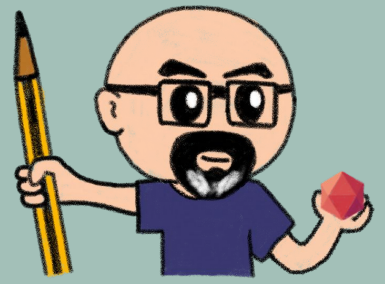
DEVOXX France

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*



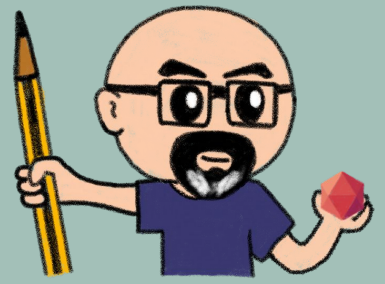


DEVOXX France

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

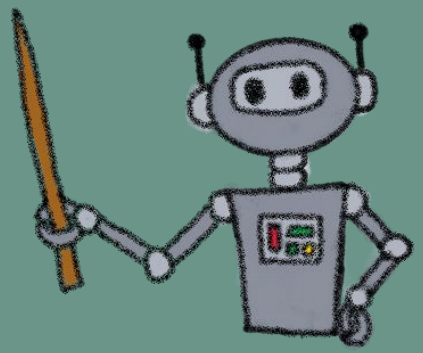
Direction of travel, verify specifics before you ship



DEVOXX France

Without a gateway

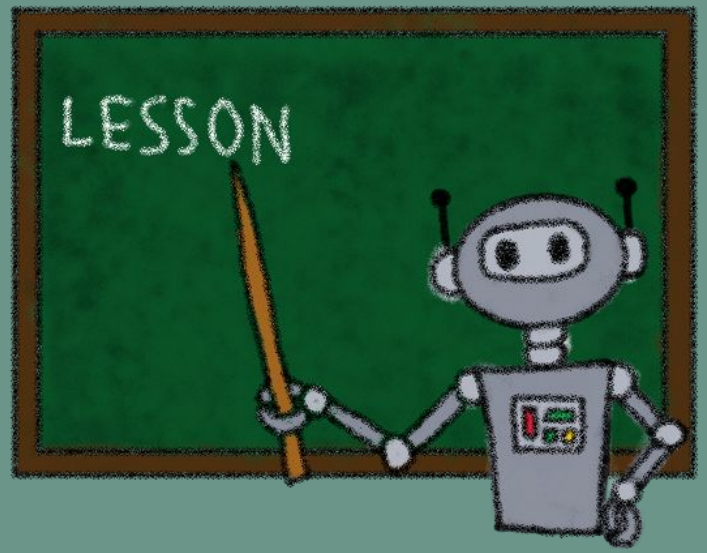
scale is a dare

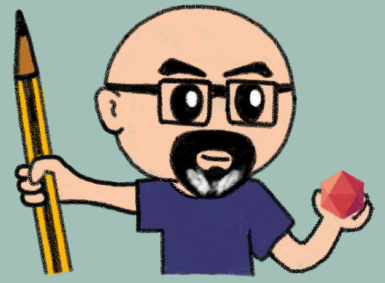


DEVOXX France

Contracts between servers

Tool schemas are your public API

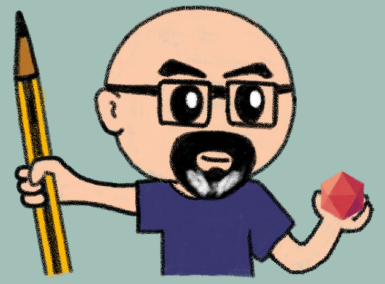




The schema evolved, the canonical client still worked

The other three clients broke at 2am

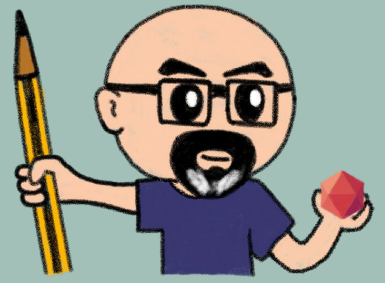
Nobody had checked them

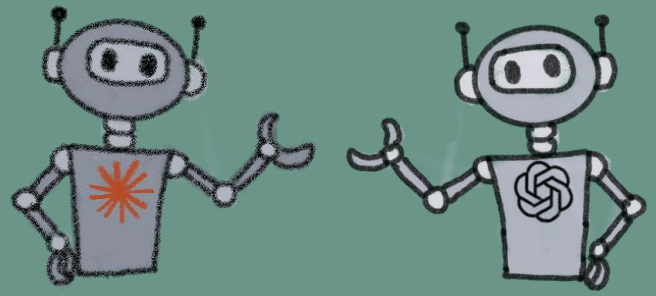


DEVOXX France

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

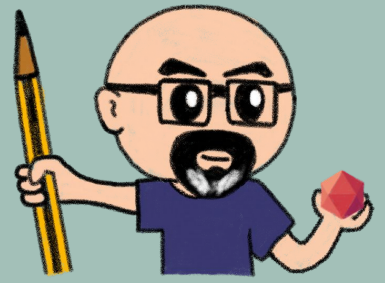




DEVOXX France

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



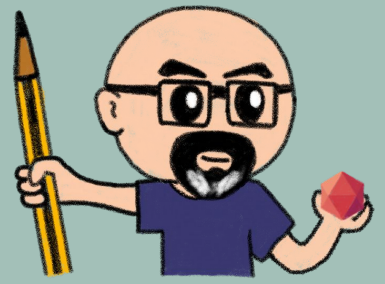
DEVOXX France

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



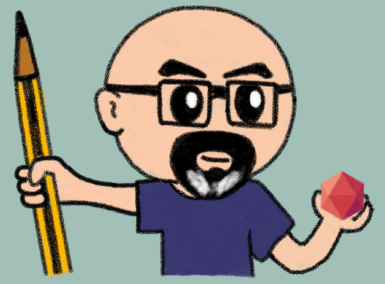
DEVOXX France

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



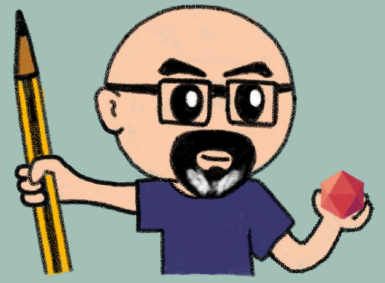
DEVOXX France

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



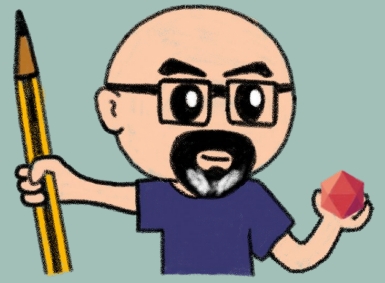
DEVOXX France

Client Matrix Testing

Your server is called by multiple clients

Client	Version	Capabilities
Claude Desktop	Latest	Full
Cursor	0.9.x	Most tools
Custom agent	Internal	Subset
CI agent	Pinned	Specific tools

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

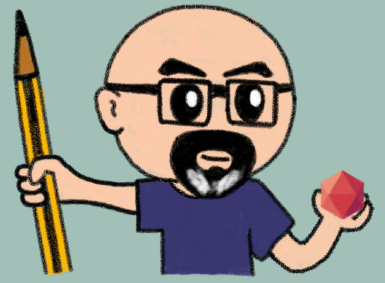


DEVOXX France

Contract Tests in CI

Prompt injection attempt test example

```
describe('Tool Contract: search_monsters_by_type', () => {
  it('schema unchanged', () => {
    const schema = getToolSchema('search_monsters_by_type');
    expect(schema).toMatchSnapshot(); // Fails if schema changes
  });
  it('example calls still succeed', async () => {
    const result = await callTool('search_monsters_by_type', { type: 'fire' });
    expect(result).toMatchSchema(expectedOutputSchema);
  });
 });
```



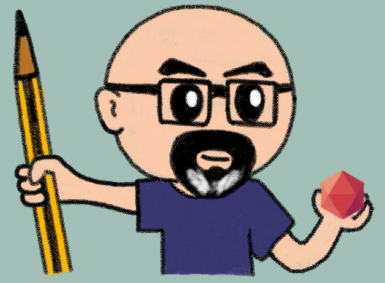
- Run on every PR
- Snapshot schemas to detect accidental changes
- Golden examples catch semantic drift

DEVOXX France

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

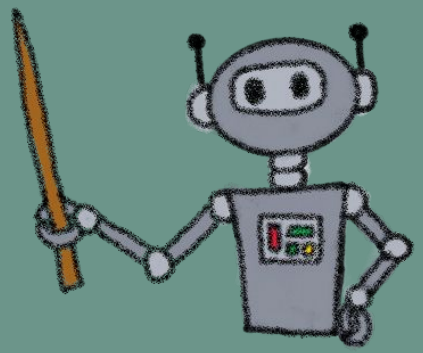
If you must break, break loudly



DEVOXX France

Tool schemas are your public API

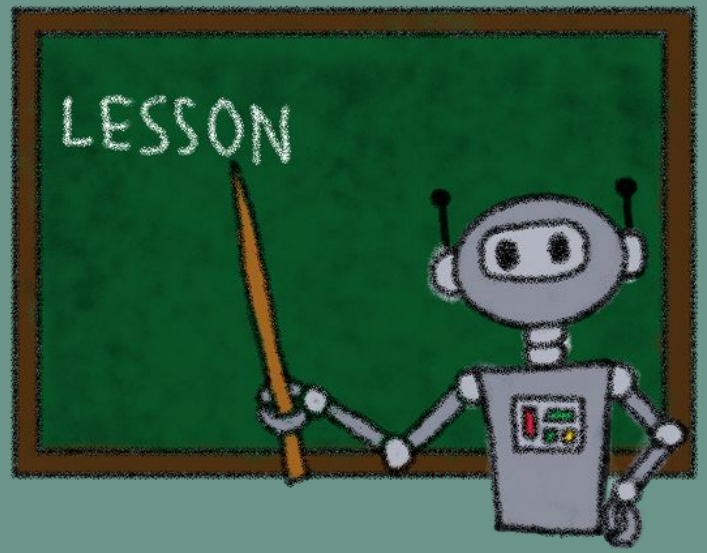
Version them



DEVOXX France

Traces across servers

Which server did this?

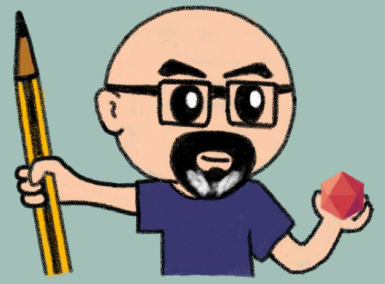


DEVOXX France

One server. Four instances. One call in four failed

The last deploy had skipped one instance

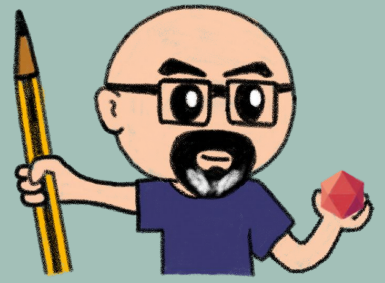
They spent a day finding out



DEVOXX France

Trace Propagation, Briefly

- **Trace IDs**
  Thread through every tool call
- **Gateway**
  Emits spans (entry point)
- **Each server**
  Emits spans (per invocation)
- **Each instance**
  Tags its spans with its identity
- **Correlate at the backend**
  OpenTelemetry works

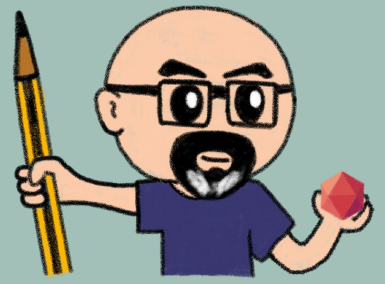


No new protocol magic, wire through the infrastructure you already have

Worth watching:

the Triggers and Events WG

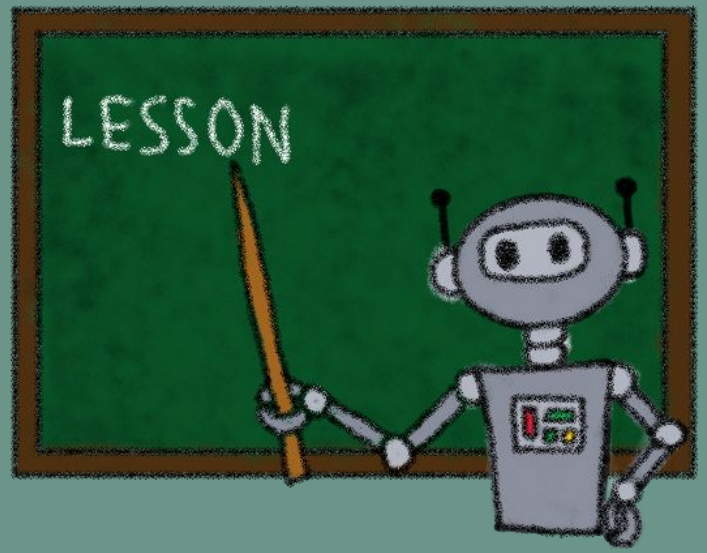
may land in-protocol eventing

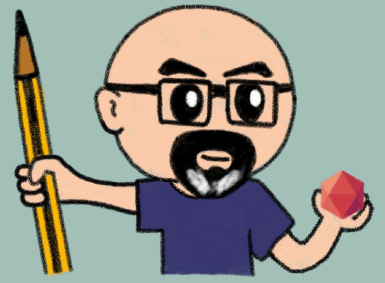


DEVOXX France

The LLM retries. A lot

Designing for a caller that loops

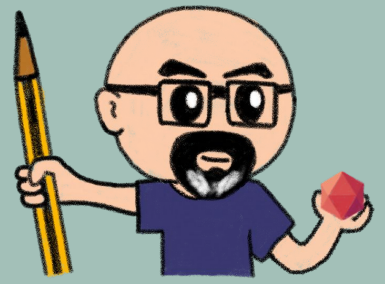




The server returned an error code, nothing else

The LLM had no context, it retried

A thousand calls. 100,000 tokens. Per hour



DEVOXX France

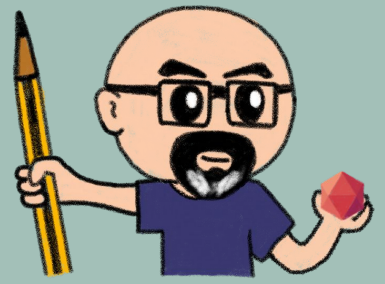
Idempotency For The LLM Era

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



DEVOXX France

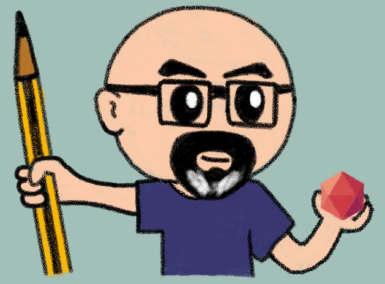
Circuit Breakers, The LLM Edition

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

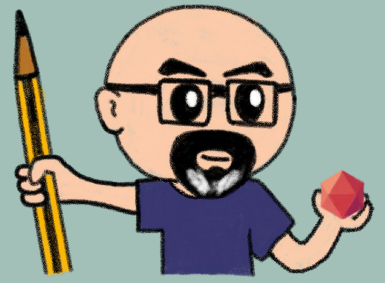


DEVOXX France

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

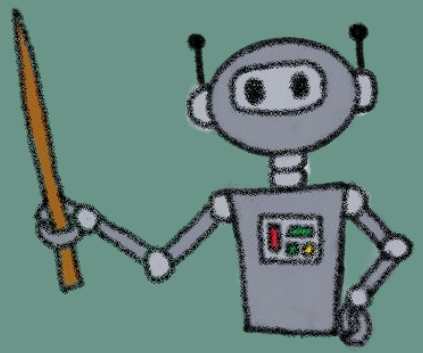
The LLM will loop 200 times. Your database will not



DEVOXX France

Design for retry

Design for storm

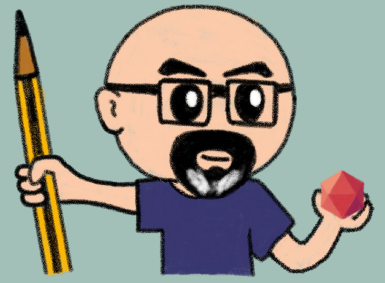


DEVOXX France

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



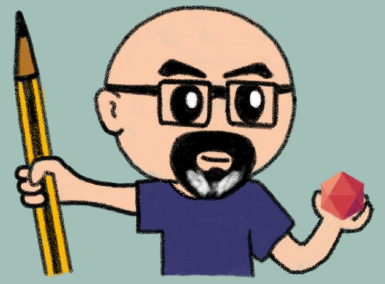
DEVOXX France

Two servers, clean auth, stable contracts

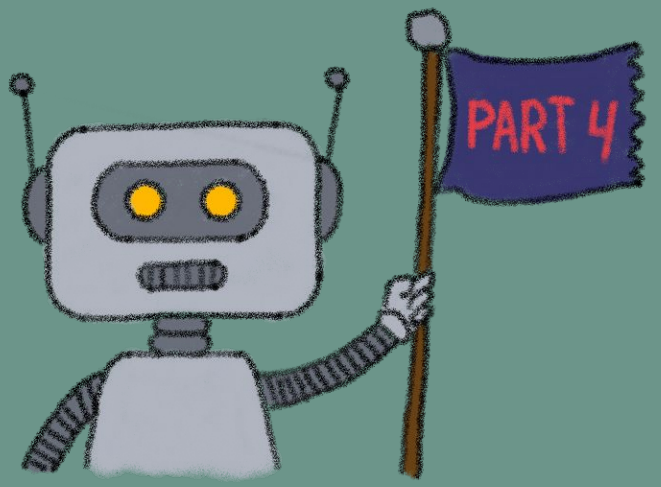
Observable and reliable

And then...

It was the legal team that asked the question

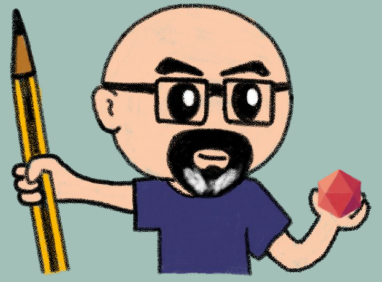


DEVOXX France



Part 4 - Governed

When the organisation wakes up

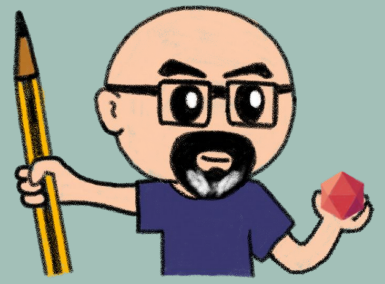


DEVOXX France

It was the legal team that asked the question

If the agent deletes production,

whose name is on the incident report?

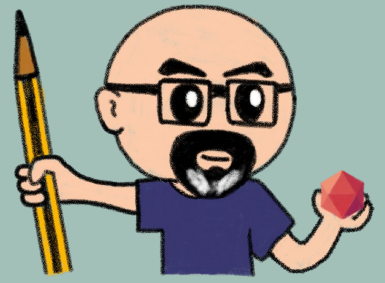


DEVOXX France

What "Governed" Means

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

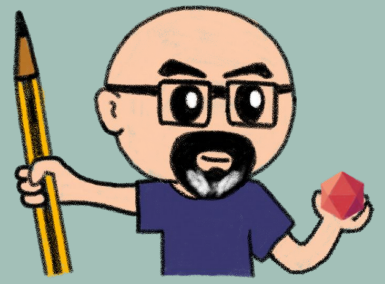
Every invocation accountable



We've seen this movie before

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

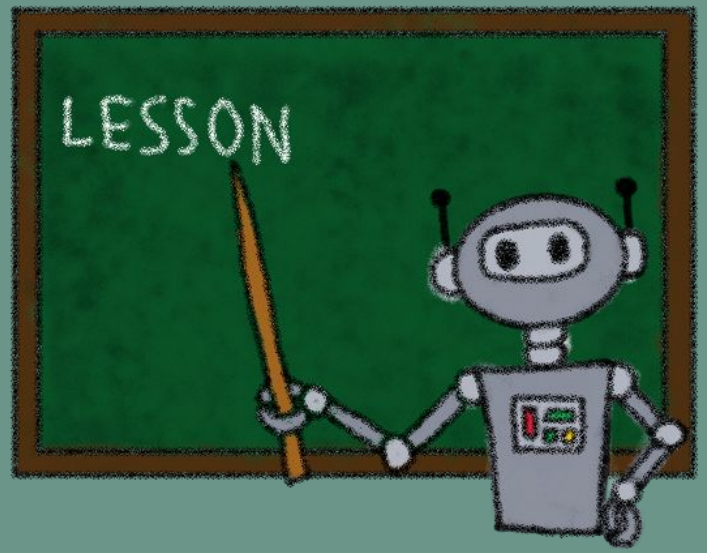
MCP is the new regulated-API surface



DEVOXX France

Blast radius

The lethal trifecta, and what it means for MCP

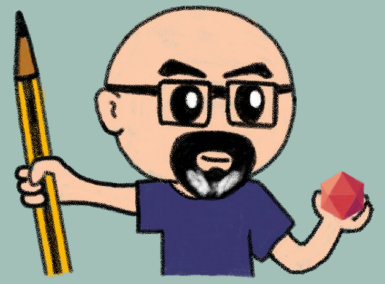


Remember the ALTER TABLE

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



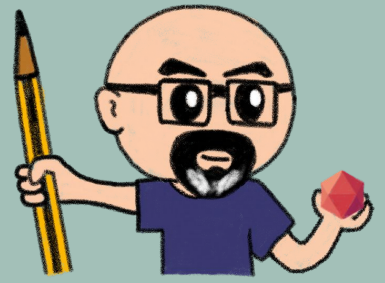
DEVOXX France

The Lethal Trifecta

Simon Willison's framing

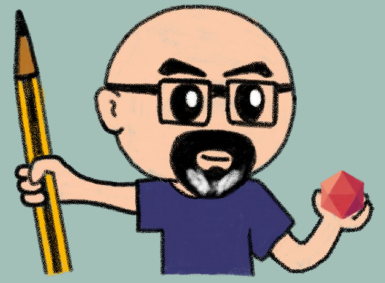
Three ingredients, together:

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



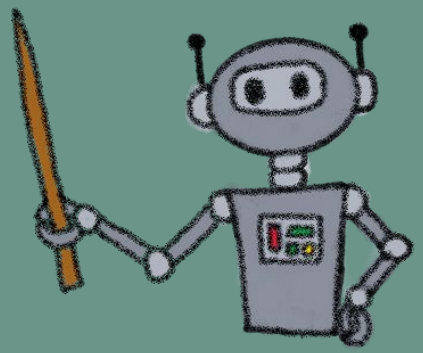
Any agent with all three
is unconditionally vulnerable
to indirect prompt injection

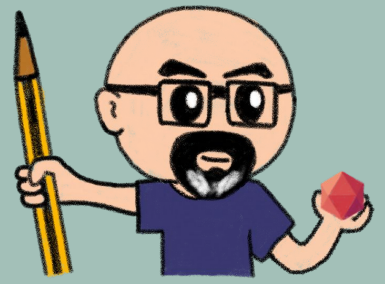
No system-prompt hardening fixes it



Power without blast-radius awareness

is just a bigger incident

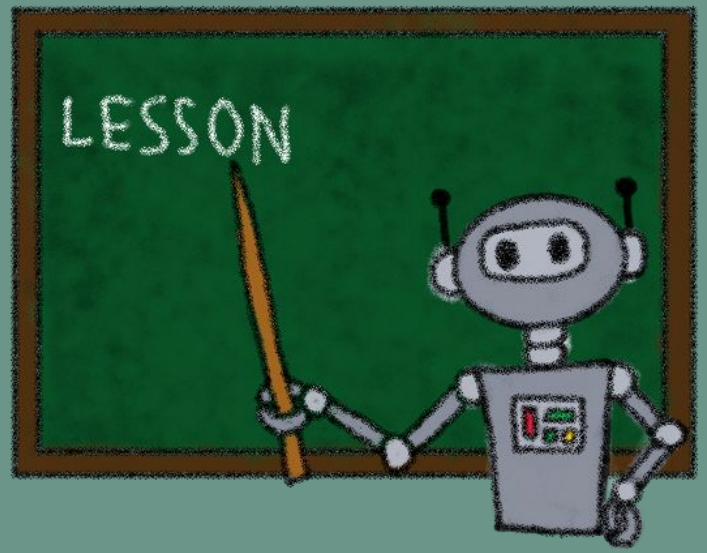


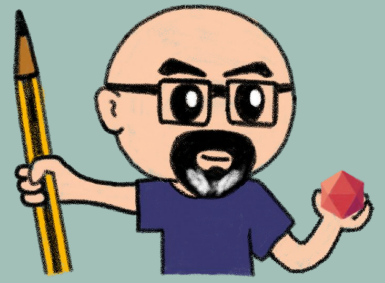


DEVOXX France

What 2026 taught us

Incidents, CVEs, scanners – the year MCP got real



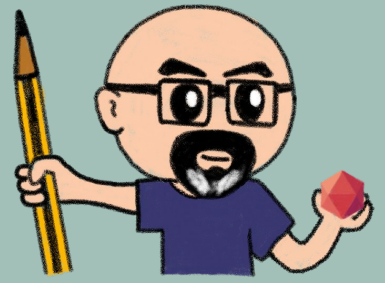


DEVOXX France

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

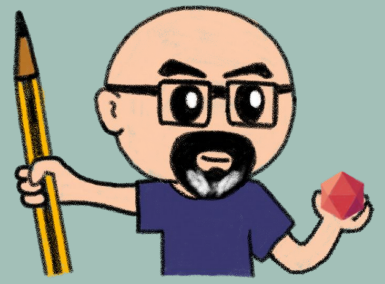


DEVOXX France

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

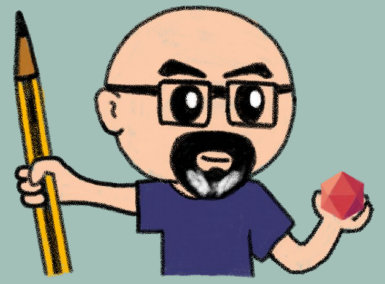


DEVOXX France

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



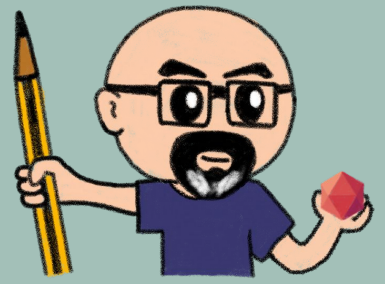
DEVOXX France

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

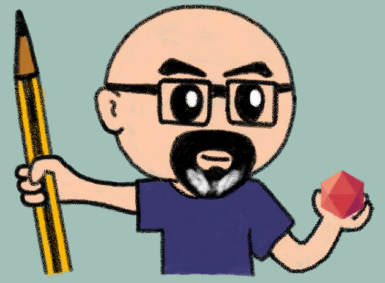


DEVOXX France

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



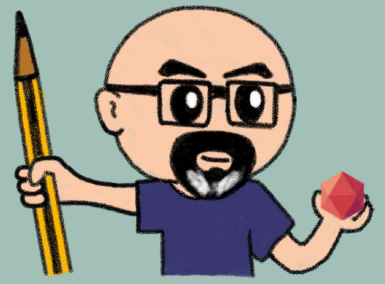
DEVOXX France

`mcp-scan`: The Scanning Answer

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

Detects:

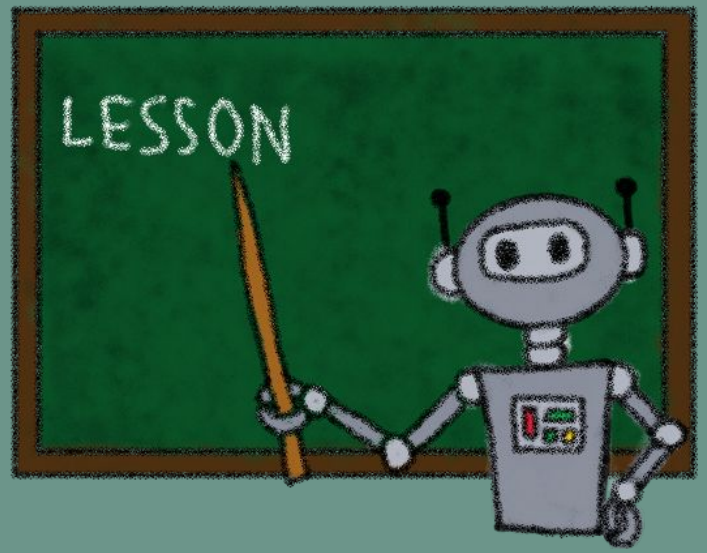
- **Tool poisoning**
  Malicious instructions in descriptions
- **Rug pulls**
  Server changes tool description after trust
- **Cross-origin escalations**
  Shadowing attacks that compromise trusted tools
- **Prompt injection in metadata**
  Malicious instructions contained within tool descriptions

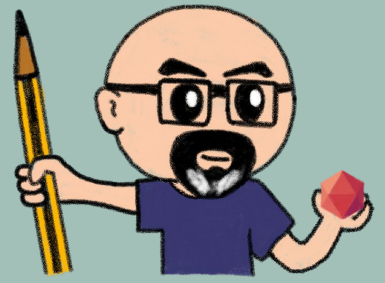


DEVOXX France

Risk Tiering

Tools with a blast radius



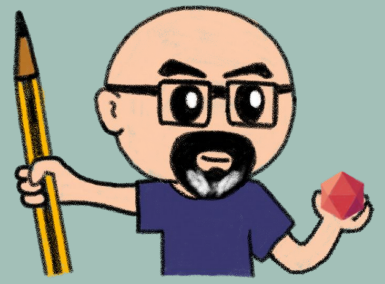


DEVOXX France

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



DEVOXX France

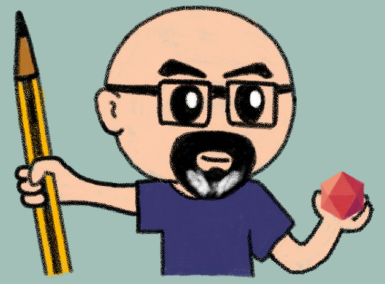
Approval Gates

- 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
```

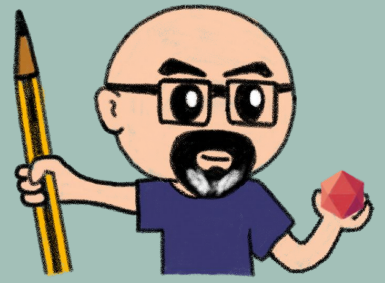
- Async approval workflow:
  Slack notification, approval UI

Autonomy for exploration, gates for action



A tool is destructive

until proven otherwise

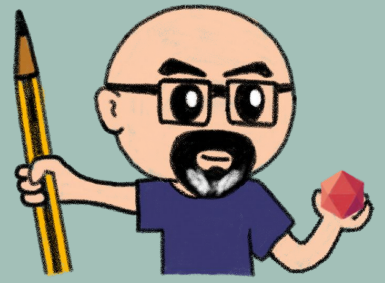


DEVOXX France

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");
 // ...
});
```

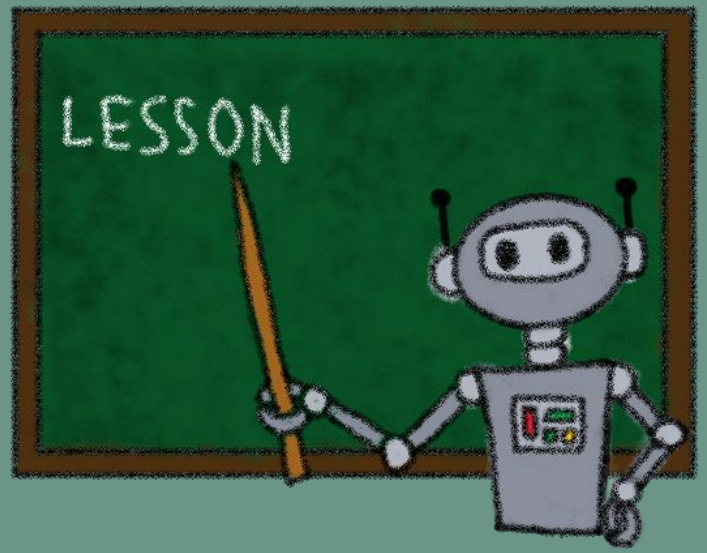


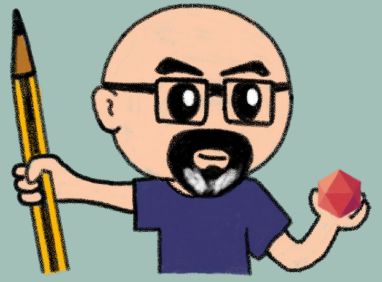
Tier in metadata, enforcement at invocation

DEVOXX France

Who did what, for how much

Retaining the audit trail the gateway emits





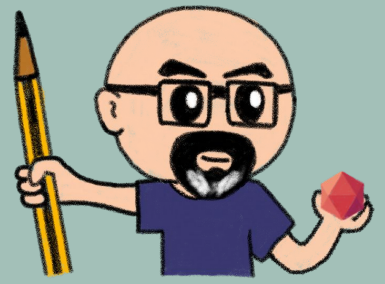
DEVOXX France

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



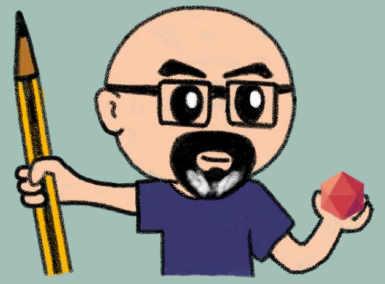
DEVOXX France

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



DEVOXX France

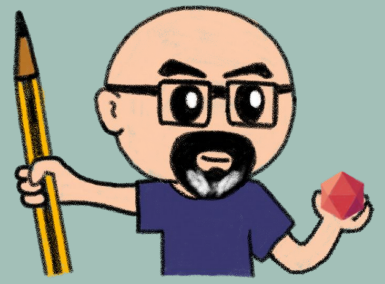
Per-Caller Cost Attribution

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

Tokens burn money

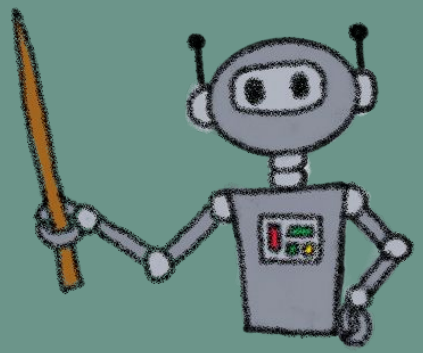
Without attribution, the bill is a mystery

The agent did this, the agent's team pays



Your audit trail

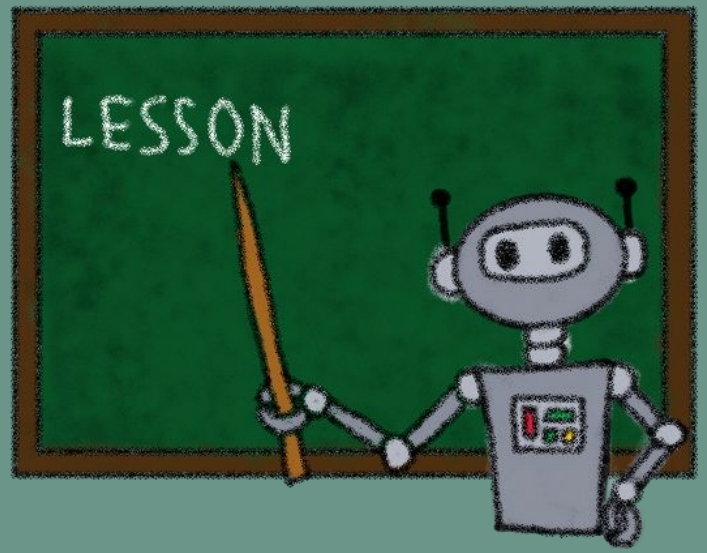
is your alibi

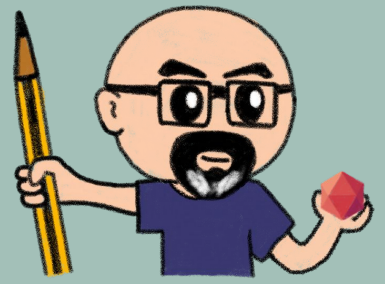


DEVOXX France

MCP is not alone

The emerging agent-protocol stack



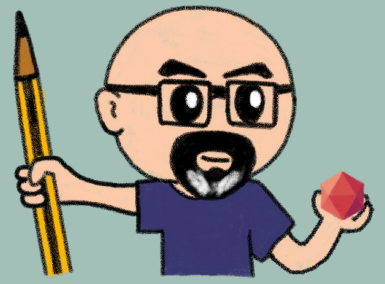


DEVOXX France

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

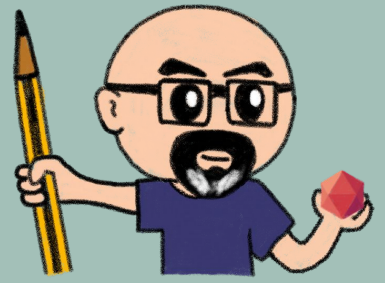


DEVOXX France

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

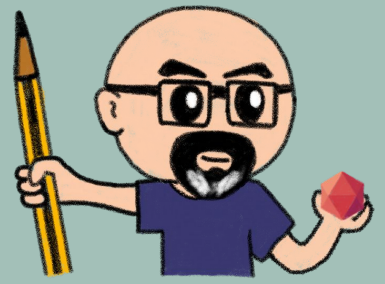
My recommendation: pick the layer, not the one hammer



Worth watching

the protocol landscape is settling fast

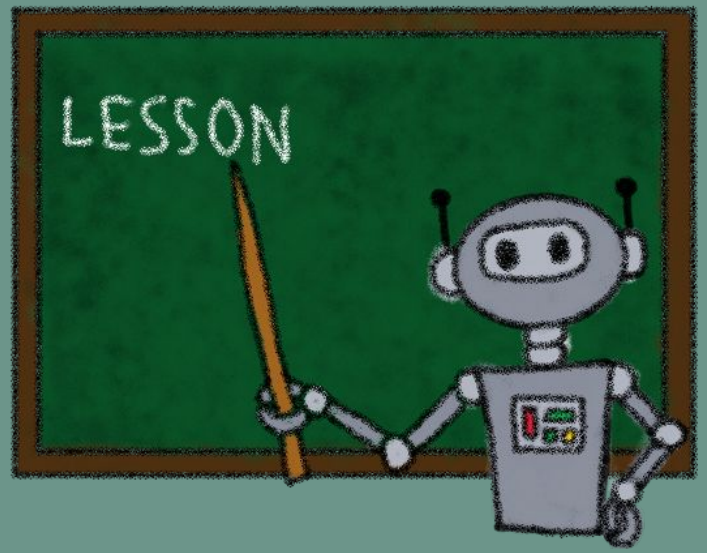
Google's guide is a frame, not a verdict

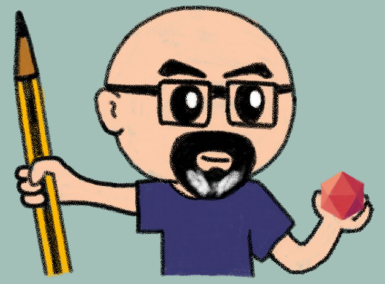


DEVOXX France

MCP is not alone

The emerging agent-protocol stack



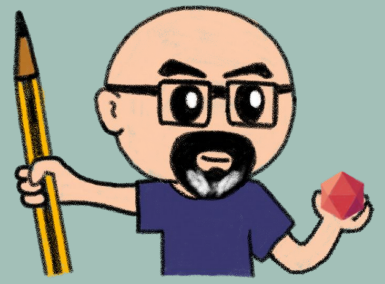


DEVOXX France

When To Use What

- **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

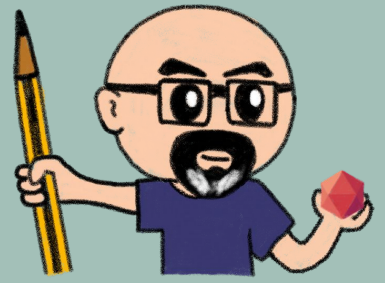


DEVOXX France

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

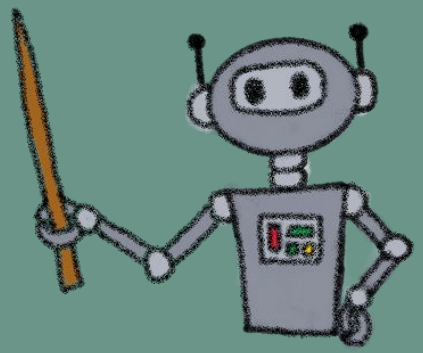


DEVOXX France

Worth watching

The protocol landscape is settling fast

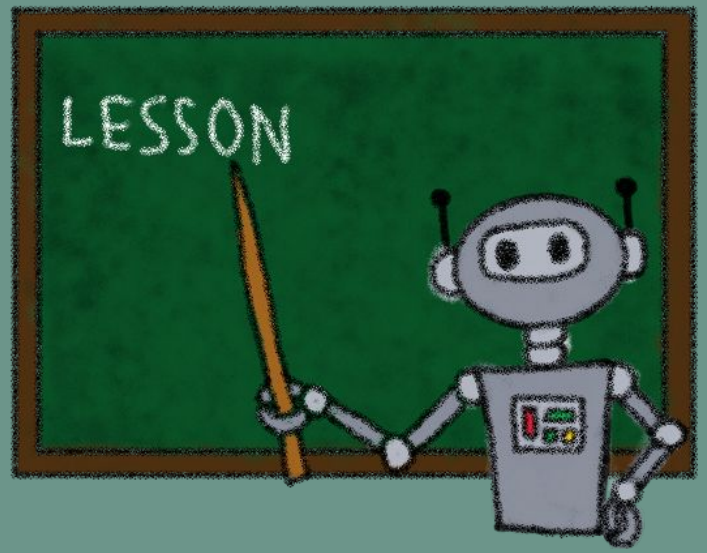
Google's guide is a frame, not a verdict



DEVOXX France

Who owns MCP in your organisation?

The emerging platform discipline



DEVOXX France

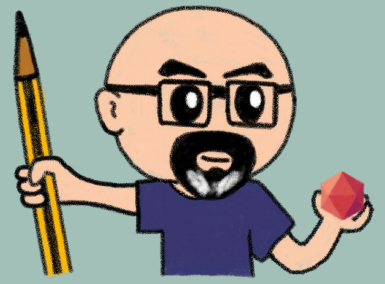
The Emerging Discipline

MCP platform teams: new role

Owners of:

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

Analogous to Kubernetes, CI, API platform team



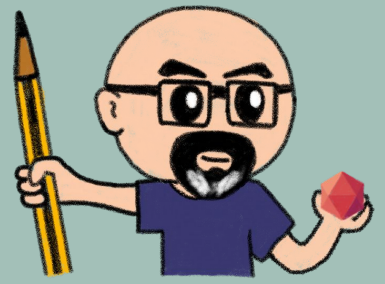
DEVOXX France

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

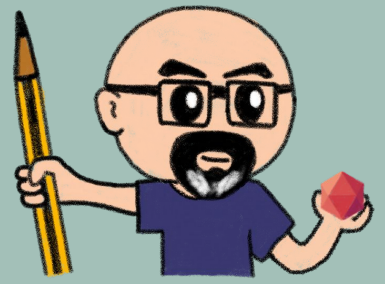


DEVOXX France

A governed MCP server

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

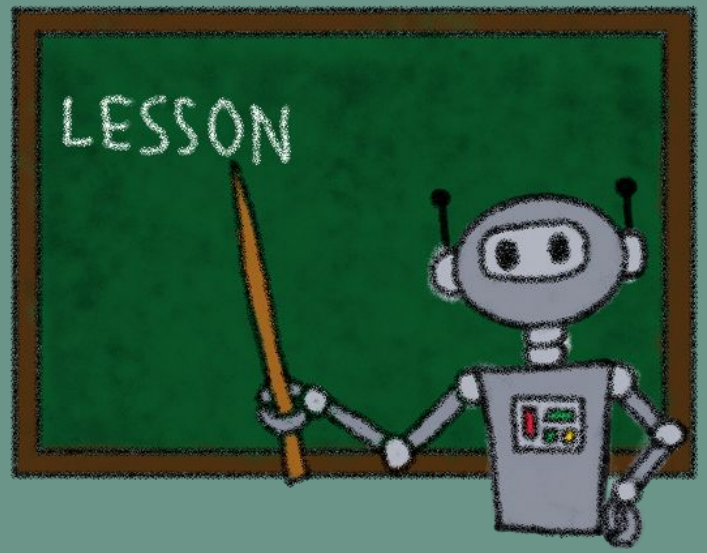
A system the organisation trusts



DEVOXX France

Reliability and Cost Controls

Agents are relentless, your infrastructure must cope

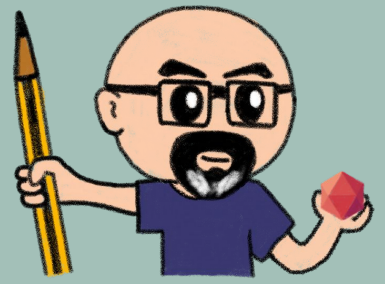


DEVOXX France

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**:

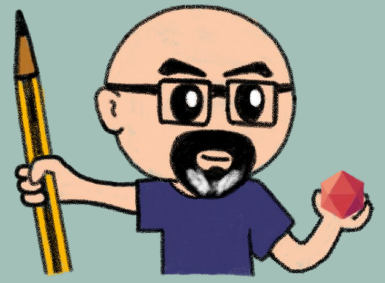
Category	Example	p95 Target
Fast read	`get_monster_by_id`	< 100ms
Search	`search_monsters`	< 500ms
Write	`create_monster`	< 1s
Async job	`generate_report`	Return immediately, poll



DEVOXX France

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

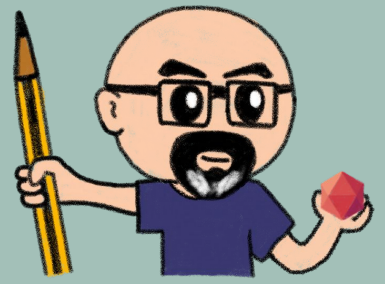
DEVOXX France

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);
}
```



DEVOXX France

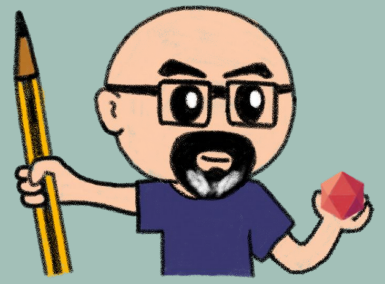
Hard Limits Everywhere

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

Limit	Default	Max
Page size	20	100
Result rows	50	500
Payload size	10KB	100KB
Query timeout	5s	30s

- Fail safely, explain clearly

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



DEVOXX France

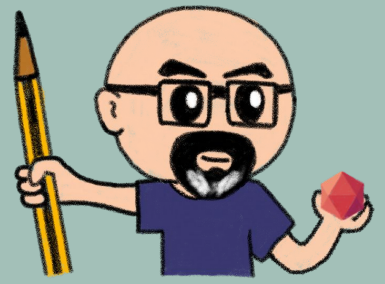
Token Efficiency Is an Architecture Concern

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": {...} }
```



DEVOXX France

Cost Attribution

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

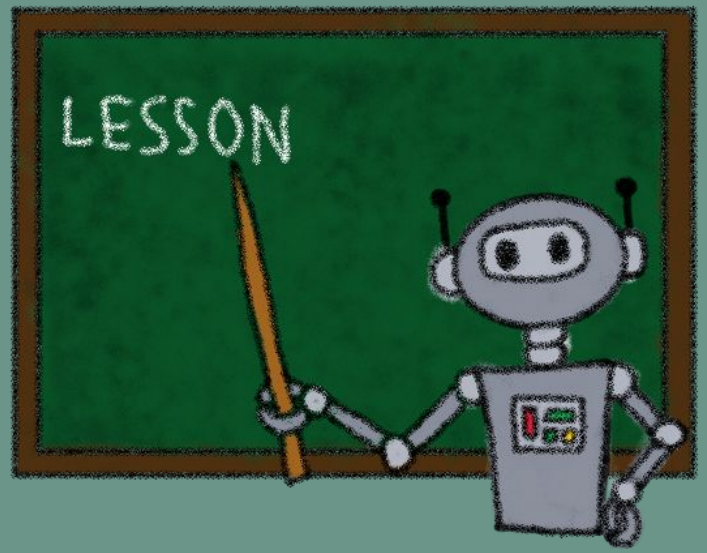
```
logger.info('Tool completed', {
  tool: 'search_monsters',
  user: session.user_id,
  team: session.team_id,
  agent: session.agent_type,
  cost_units: calculateCost(result),  // Your cost model
  latency_ms: elapsed
});
```

- 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



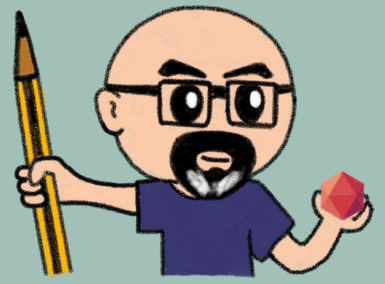
DEVOXX France

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



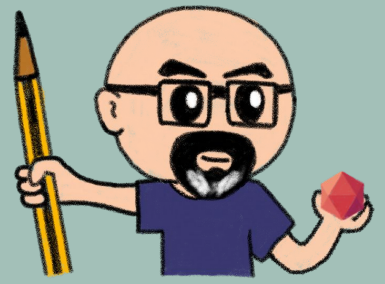
Approval Gates

- 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



DEVOXX France

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
```

DEVOXX France

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

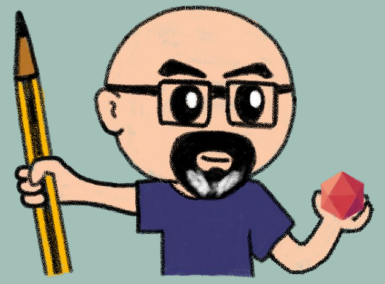
DEVOXX France

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"

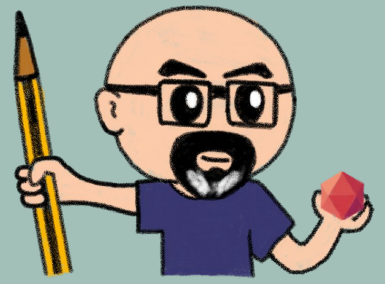
Safety and usability aren't opposites, good design achieves both



DEVOXX France

That's all, folks!

Thank you all!



DEVOXX France

What We've Learned So Far

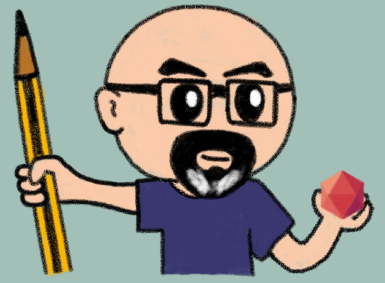
And how to go further

DEVOXX France

The Part 3 Takeaway

Scaling MCP is mostly:

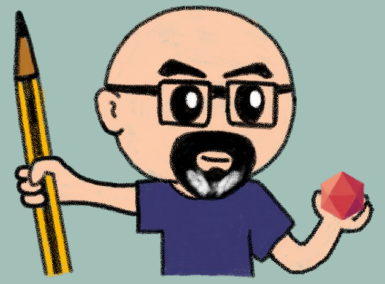
- **Composition:**
  Domain servers, gateways, orchestrators
- **Contracts:**
  Versioning, compatibility, "don't surprise the agent"
- **Controls:**
  Limits, idempotency, cost attribution, safety tiers



DEVOXX France

A Practical Maturity Ladder

Level	What You Have
v1	One server, basic validation, logs
v2	Domain servers, CI tests, structured logging
v3	Gateway, policy enforcement, evaluation suite
v4	Risk tiers, approval gates, cost attribution

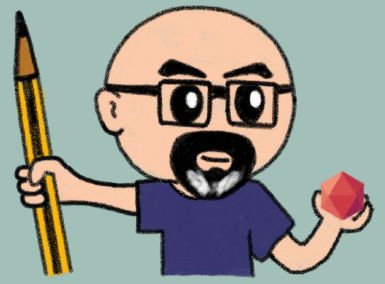


- You don't need v4 on day one
- But know where you're heading

DEVOXX France

Resources

- MCP Specification:
  - modelcontextprotocol.io
- RAGmonsters examples:
  - github.com/LostInBrittany/RAGmonsters-mcp-pg
  - github.com/CleverCloud/mcp-pg-example
- Anthropic MCP docs:
  - docs.anthropic.com
- This talk's slides:
  - lostinbrittany.dev/talks



DEVOXX France

That's all, folks!

Thank you all!