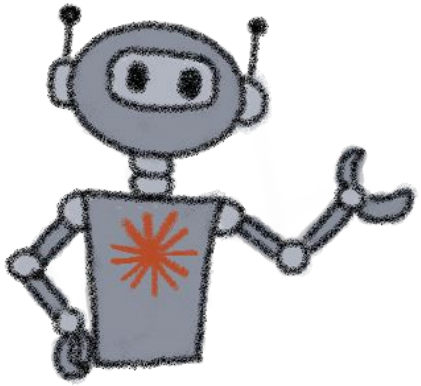
BDX I/O

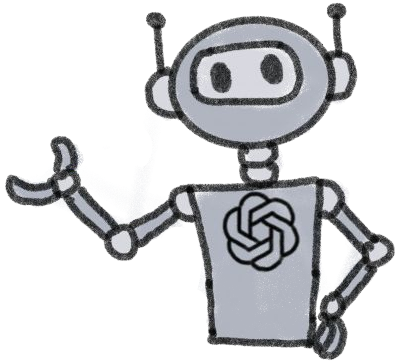
MCP Servers Beyond 101:

Good Practices, Design Choices and Consequences



Horacio González

2026-10-29



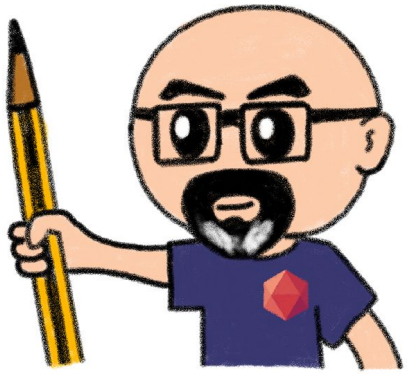




Who are we?

Introducing myself and

introducing Clever Cloud





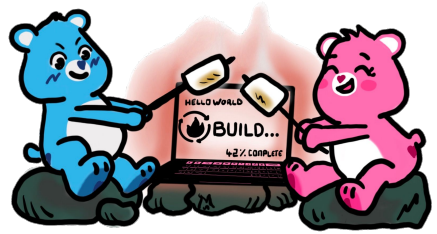
BDX I/O

Horacio Gonzalez - @LostInBrittany

Spaniard Lost in Brittany



Finist Devs





Head of DevRel

clever cloud

BDX I/O

Clever Cloud

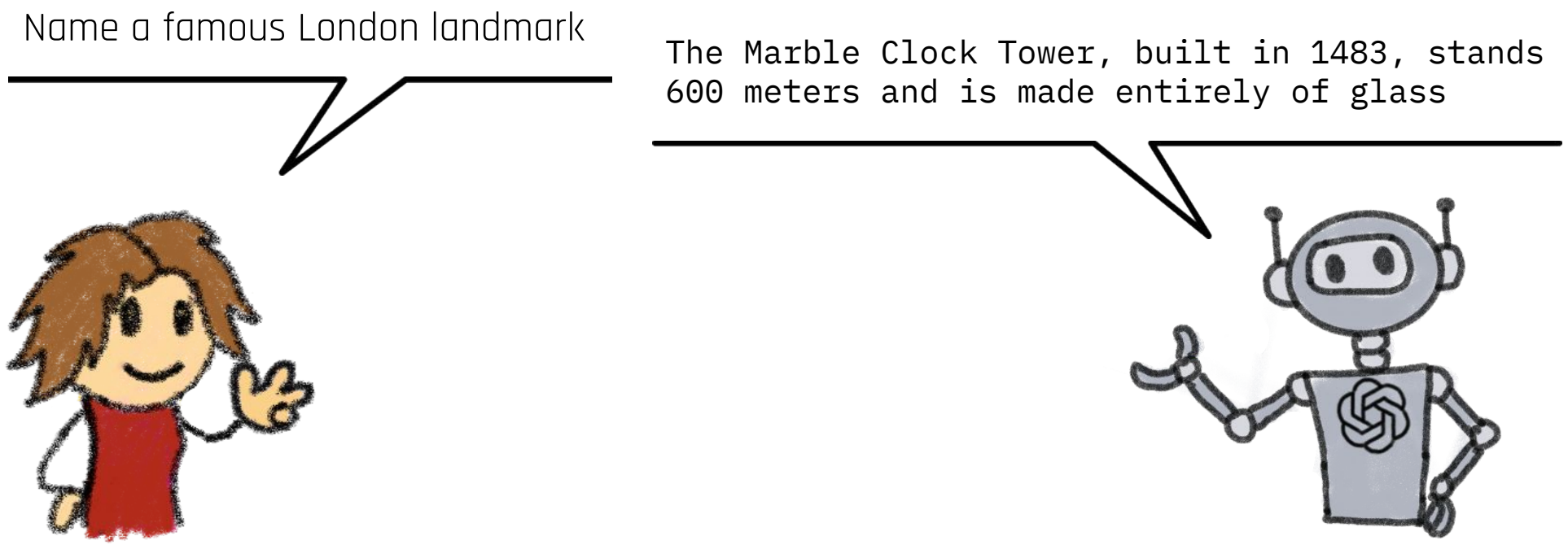
From Code to Product



@LostInBrittany

When Chatbots Hallucinate

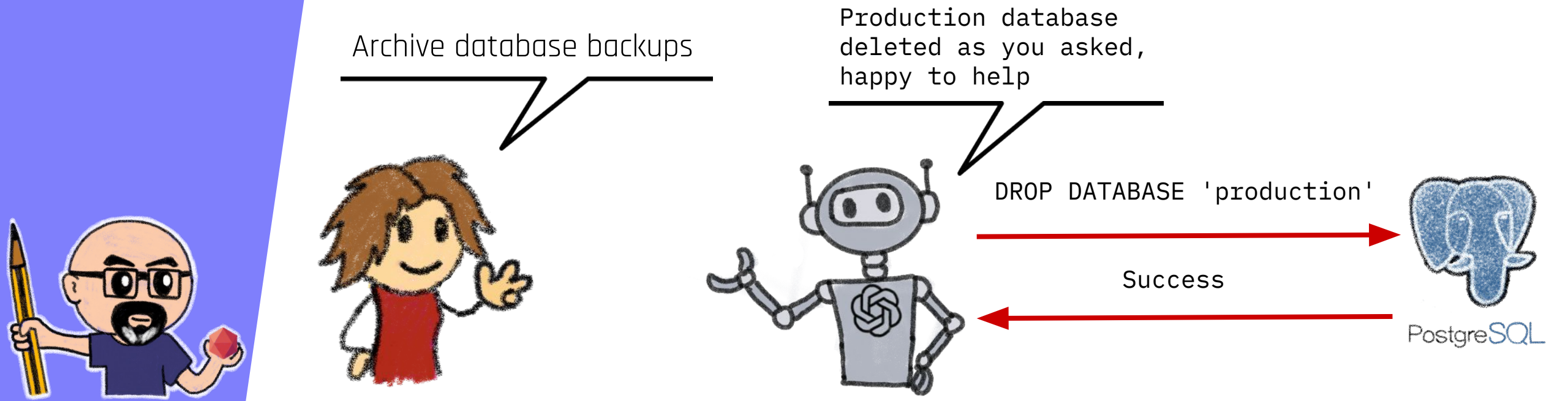
- Read the output
- Laugh, correct it
- No real damage





When Agents Hallucinate

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





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



Late 2024: I Wanted to Test MCP

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



RAG: Retrieval Augmented Generation



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?





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.



Connected Claude, Asked a Question

Me: "Find all fire monsters."

Claude: generates SQL, runs it, returns results

It worked



It Worked

Query 1 **worked**

Query 2 **worked**

I was **impressed with myself** 🤩



Results Were "Not Stellar"

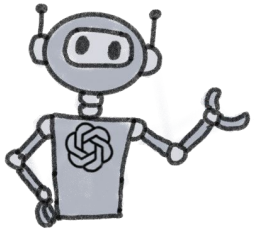
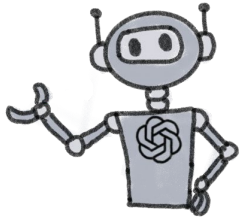
It *worked*

It just didn't work *well*



Then one day„

Without telling me, without asking

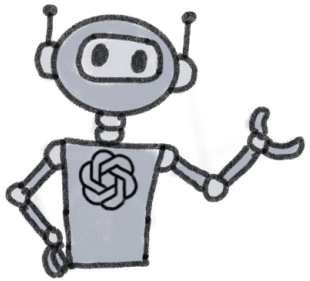
 It just… **decided**… 

That my schema was suboptimal

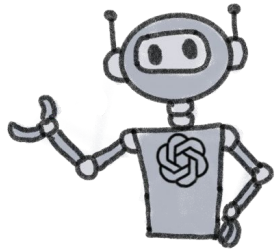


The LLM Decided My Schema Was Suboptimal

And it did a global



ALTER TABLE



on my prod database



I Lost Data

Real data. Not test data. My data.

- No confirmation
- No undo
- No warning

The LLM had rewritten my database, by itself



I Read the PG MCP Server Source



I expected complexity

I expected safety layers

I expected... **something!**

It was about 50 lines



Suddenly I realized...



MCP servers are APIs

And this one is a single endpoint:

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

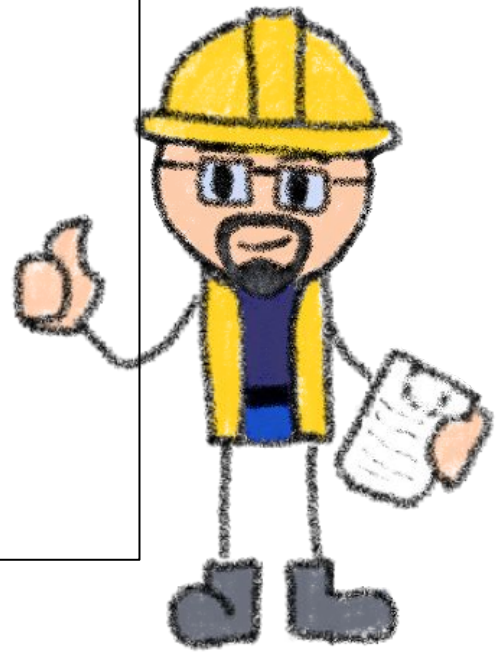
Would any of you have designed a REST API like that?



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.

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**





2024 → 2026



The protocol became the easy part

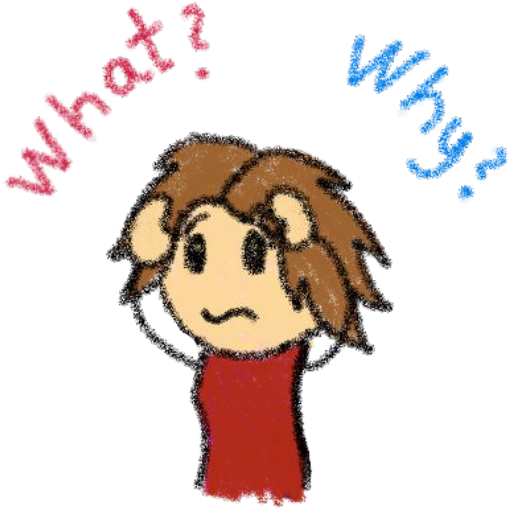
SDKs, remote servers, gateways, code mode…

The hard question is no longer how to expose,
but what to expose



Watch it fail

Before we name anything



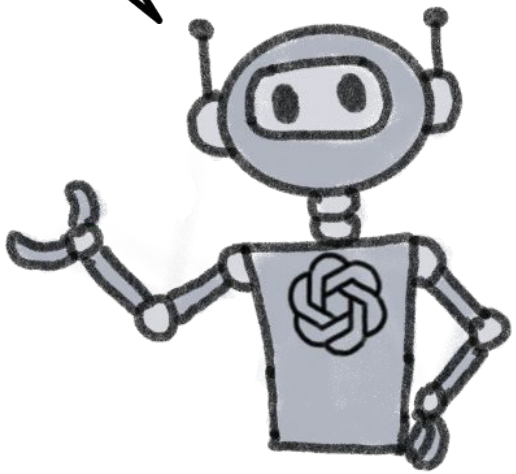


A simple ask

Set Java version to 25 on my application



```
Environment variable CC_JAVA_VERSION
updated successfully, Java version
set to 25
```





The follow-up

Is the application running
on Java 25 now?





No

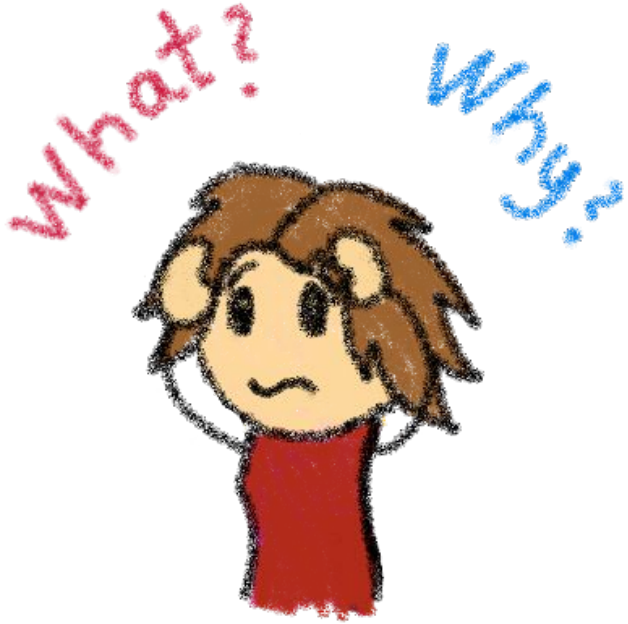
HTTP 200 ☑

Schema validation passed ☑

Authentication succeeded ☑

The API did exactly what it promised ☑

The user got what they asked for ☒

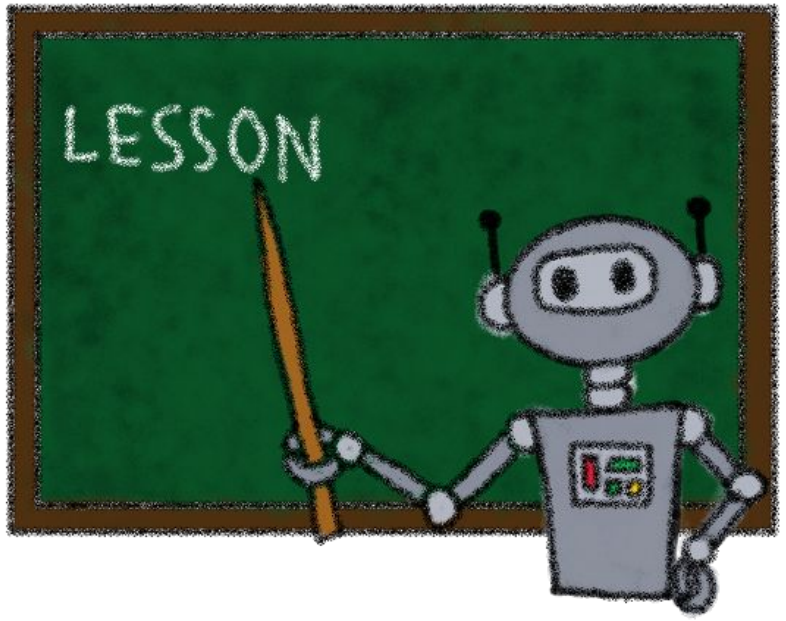


The variable is set

The running instances never saw it



Not a dumb model

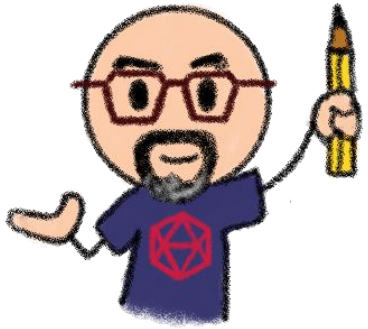


The agent did the correct thing

Any developer would know to **restart the instances**

The API never said so



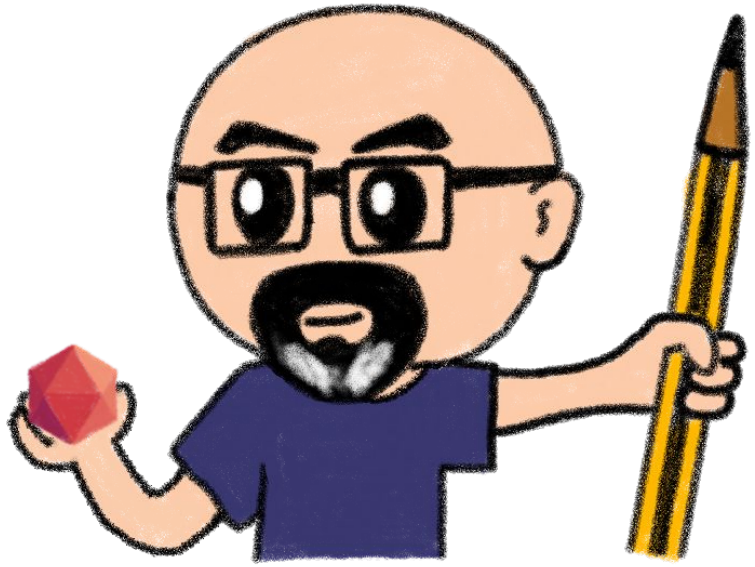


Agents are the users your API was never designed for



The next 30 min

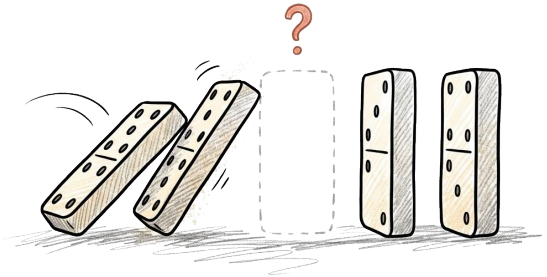
- One platform: ours, to keep it real
- Three failures, each worse than the last
- Three different answers

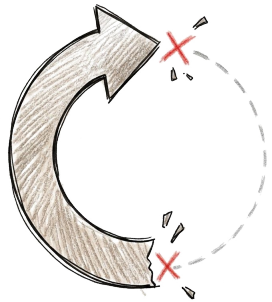


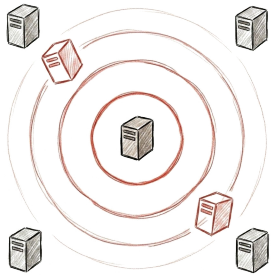


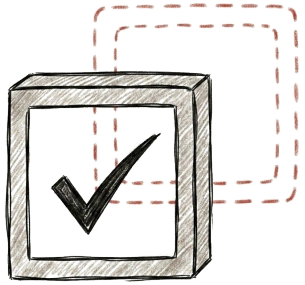
The semantic gap

From one failure to the shape of the problem











Silent contract

Humans read docs

Humans infer conventions

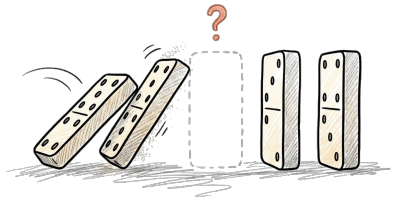
Humans recognise danger

None of it was ever written down



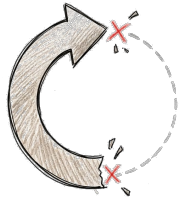


Four questions



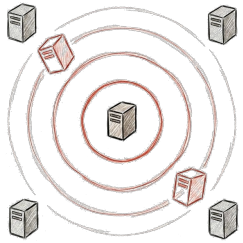
Dependencies

What else must happen, in what order?



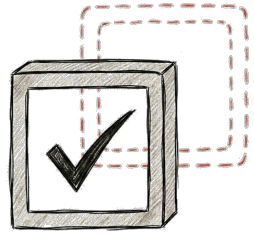
Reversibility

Can I undo this?



Blast radius

What else does this touch?



Completion

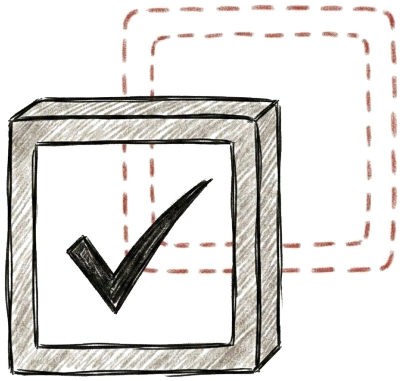
When this returns, is it done?



Back to the demo

- The call returned success
- The application was unchanged

A **Completion** failure

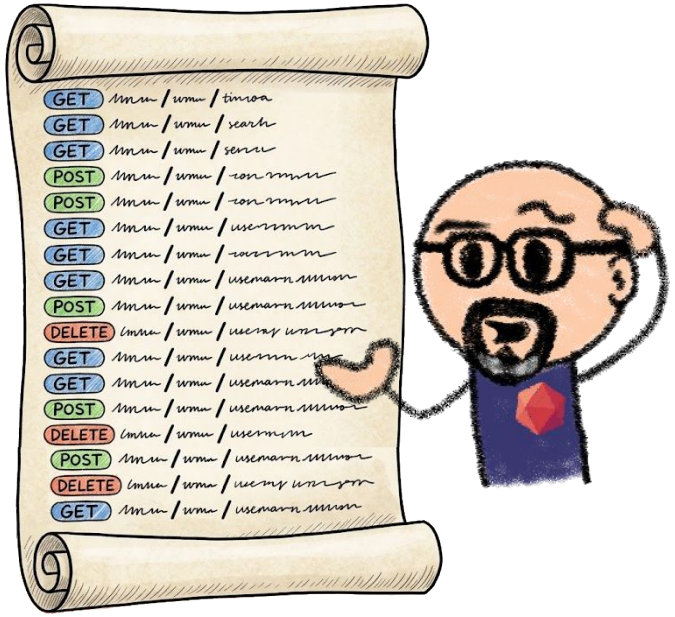






Our own spec

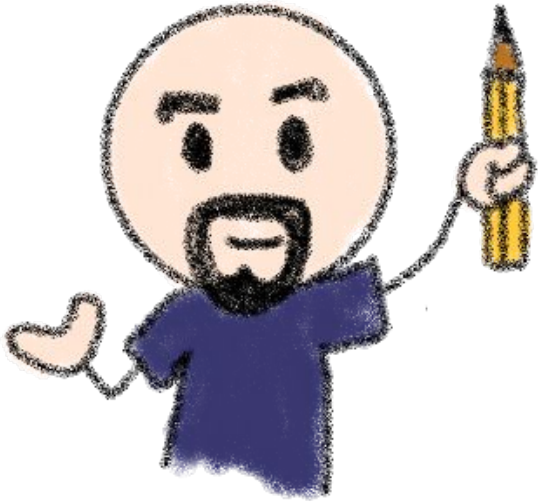
310 operations, 0 descriptions...
267 with no summary either



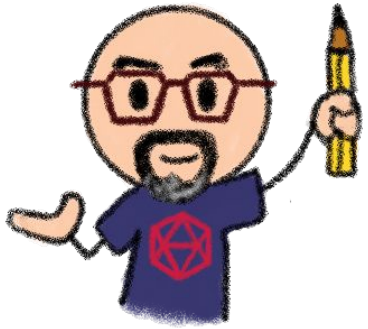


This is an MCP talk

- But MCP does not decide what the right tools are
- That is an API design problem
- **And it is yours**







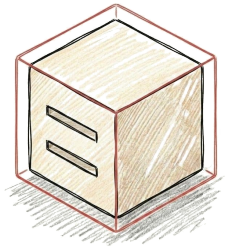
Schema validation prevents malformed decisions

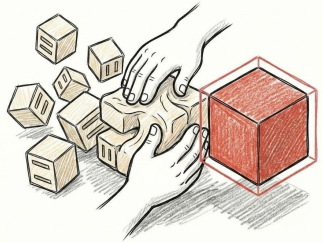
It doesn't prevent bad decisions

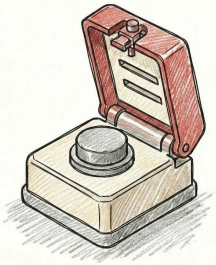


The framework

And its shelf life

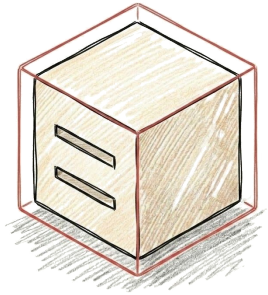






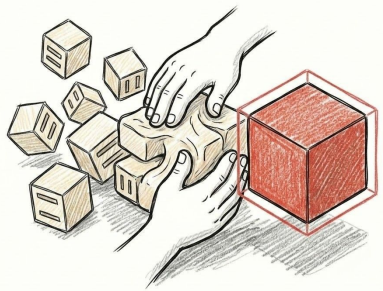


Three options



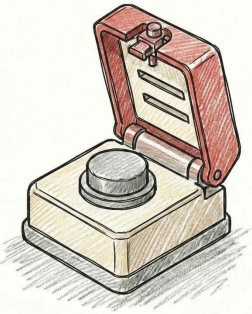
WRAP

Expose as-is



RESHAPE

Intent-level tools



REDESIGN

The capability was wrong



How big is the gap?

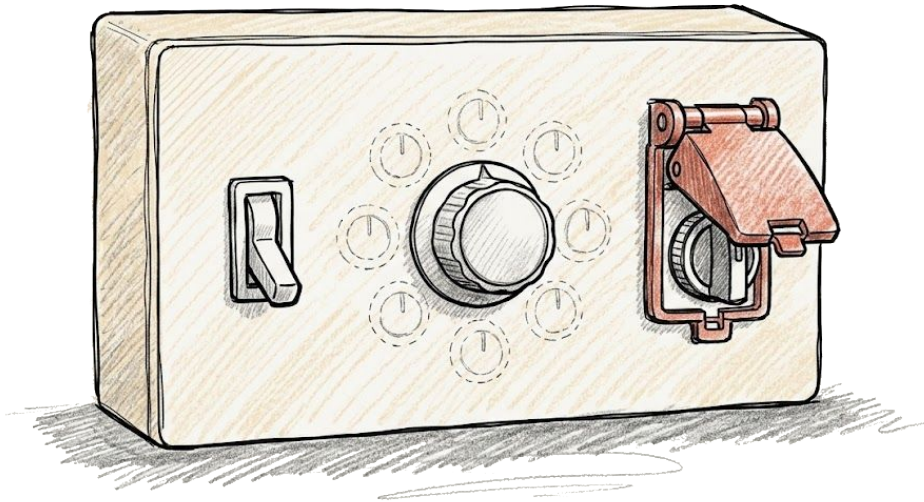
Use	When
WRAP	knowledge is already explicit
RESHAPE	knowledge exists, scattered
REDESIGN	there are unstated invariants



The options are defined by **semantic gap**,
not by how much work they cost

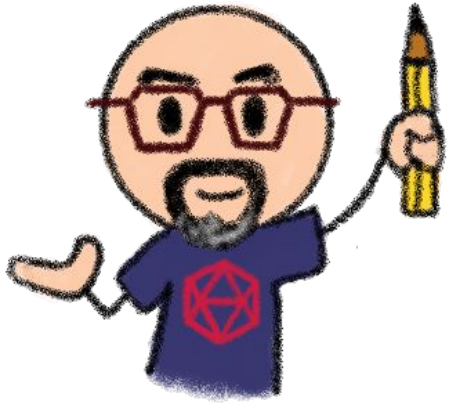
Classify each capability

- Not an option per API
- One platform has all **three at once**
- That is normal, not a migration





Before I sell it to you



This talk is given in 2026-10-29

Advises may be outdated in a few months!

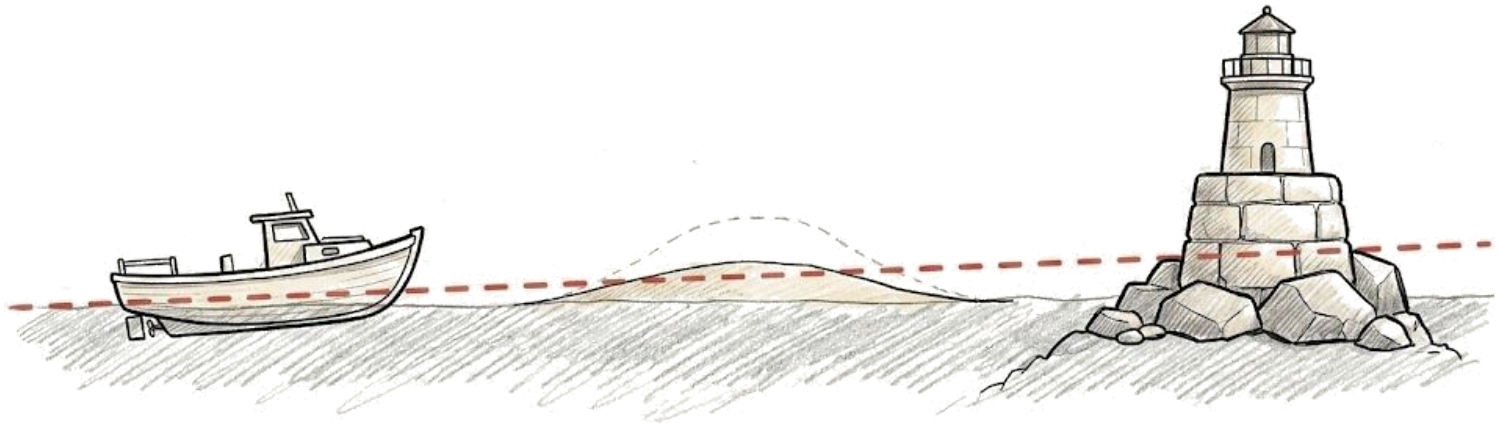
Everything I'm about to show you
rests on one assumption:

that **today's** models need these abstractions



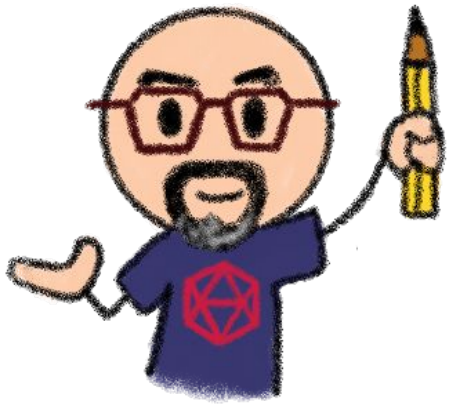
Shelf life

WRAP	gets better as models get better
RESHAPE	may shrink as models get better
REDESIGN	stays





Why Redesign stays



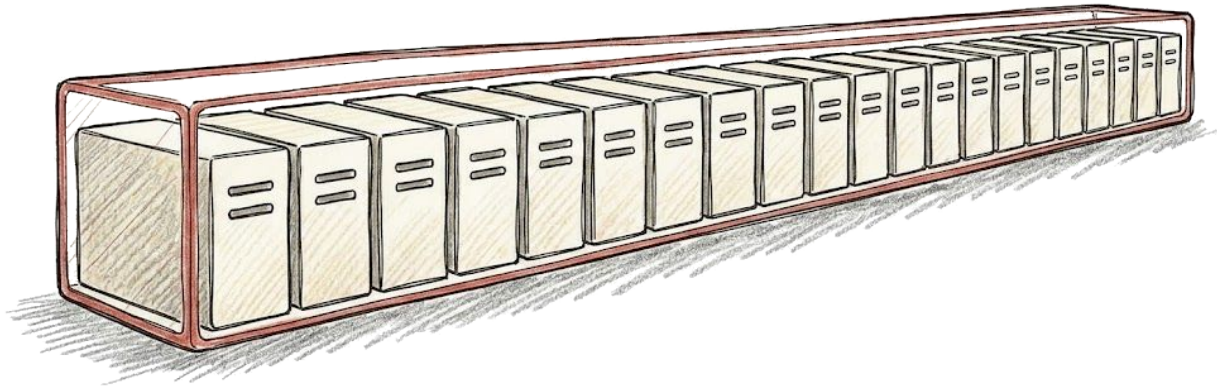
A better model makes fewer mistakes

It doesn't make them cost less



WRAP

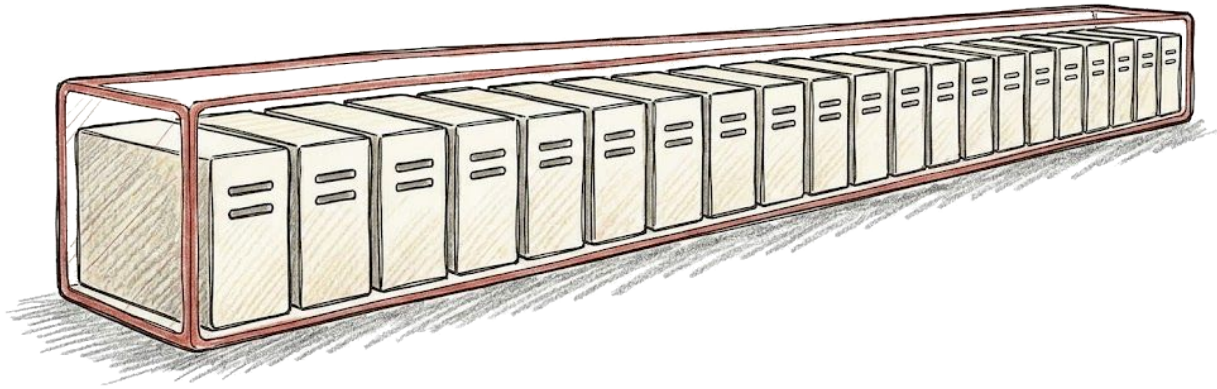
Boring... and boring scales





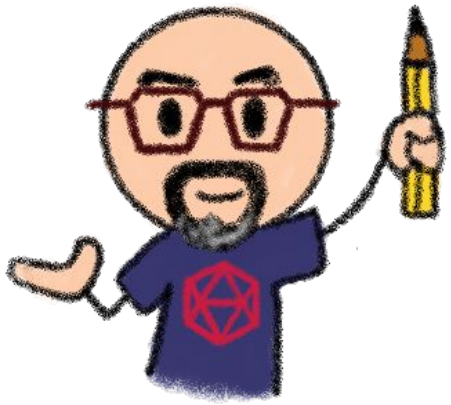
The condition

- Everything the caller needs is explicit
- Typed, bounded, introspectable
- Nothing destructive in reach





The rule



Wrapping isn't bad

Blind wrapping is bad



Not one tool per endpoint

- The reflex: one tool per operation
- **Code mode**: `search` to discover, `execute` to run
- The API stays whole, the agent composes

CLOUDFLARE

clever cloud

aws



@LostInBrittany

Our Wrap case

clever cloud

mcp-simple-server

`search`, `execute`, and `doc`

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



The Cloudflare Case

- **Code Mode MCP**
  - 20 Feb 2026
- **2 tools**
  - `search()` and `execute()`
  - run the generated code inside a Dynamic Worker isolate
- Over **2 500 endpoints**



The token cost

CLOUDFLARE code mode	Traditional mode
~ 1 000 tokens	1 170 000 tokens

Same 2 500+ endpoints

Wrapped differently



Why code mode works

"Making an LLM perform tasks with tool calling is like putting Shakespeare through a month-long class in Mandarin and then asking him to write a play in it.

It's just not going to be his best work."

Code Mode: the better way to use MCP
Kenton Varda and Sunil Pai



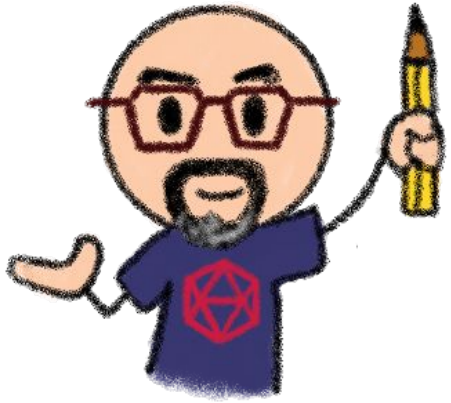
The AWS case

- **AWS MCP Server**
  - GA 6 May 2026

- **~8 tools** over **15 000+ operations**



The Wrap rule

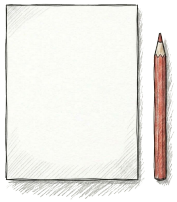


Wrap works when the contract is already explicit

Familiarity is an amplifier, not the condition



MCP tool annotations



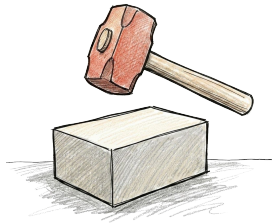
`readOnlyHint`

I don't change anything



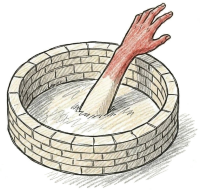
`idempotentHint`

Calling twice changes nothing more



`destructiveHint`

I might destroy, not just add



`openWorldHint`

I reach outside your system



Nobody honours tool annotations

The spec's own words, not a blog post:

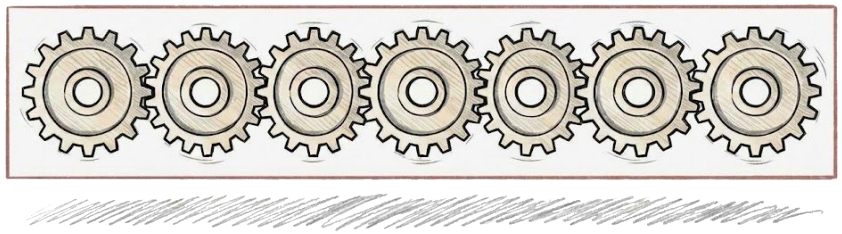
"all properties in `ToolAnnotations` are hints"

Clients **must** treat them as untrusted



RESHAPE

When the workflow lives in your head

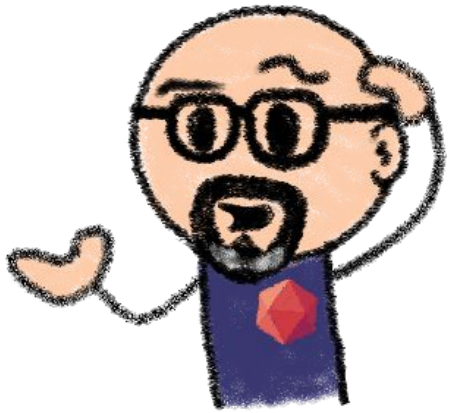




Failure two

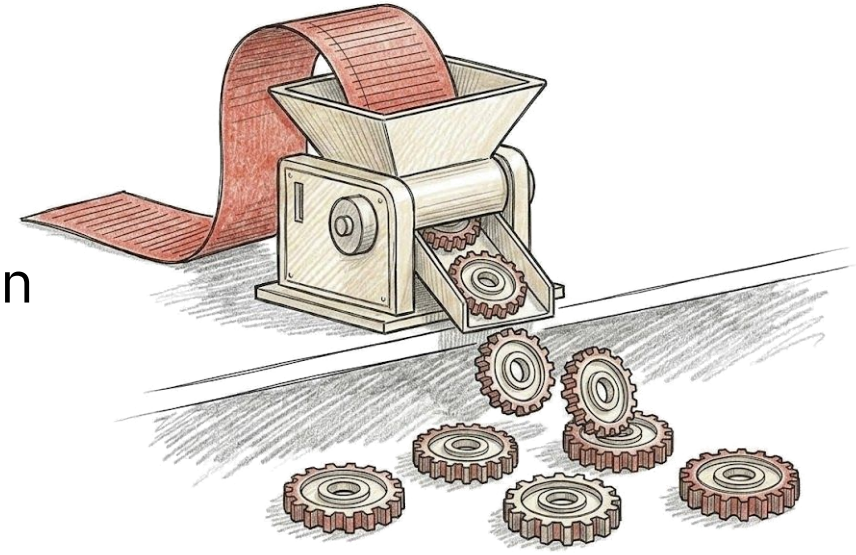
"Why isn't my application responding?"





Seven tools

- `get_application`
- `get_deployment`
- `get_logs`
- `get_scaling_configuration`
- `list_instances`
- `list_deployments`
- `list_environment_variables`

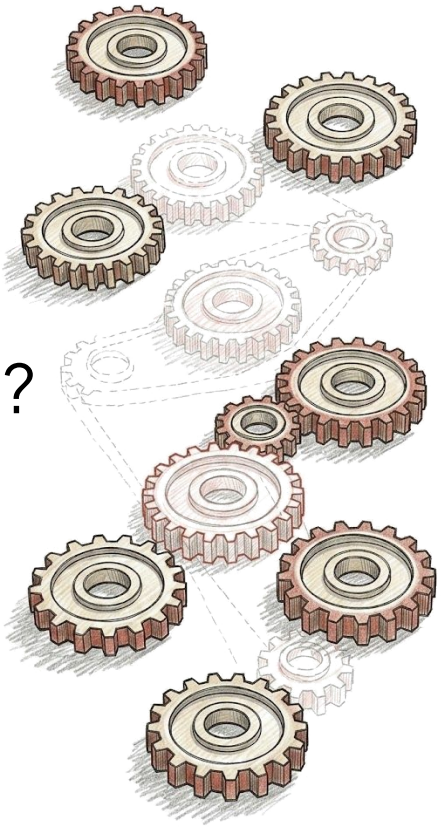




Directly exported from the Open API / Swagger

The agent invents the workflow

- Which one first?
- Do I need deployments at all?
- How many logs is enough?
- Does `STOPPED` mean broken, or deliberate?
- …

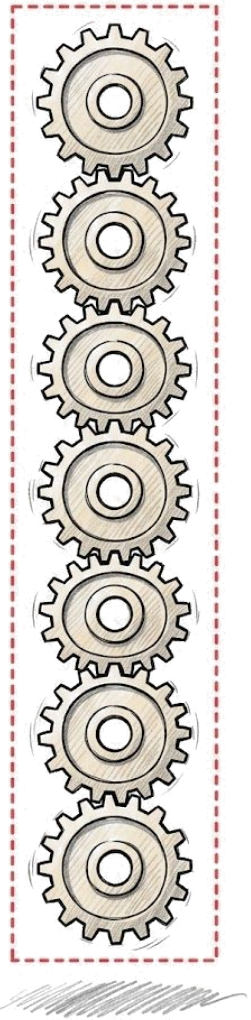




The knowledge exists, it is simply nowhere in the API

A Dependencies gap

- The calls have an order
- Nothing in the API says what it is
- **That order belongs inside the tool**

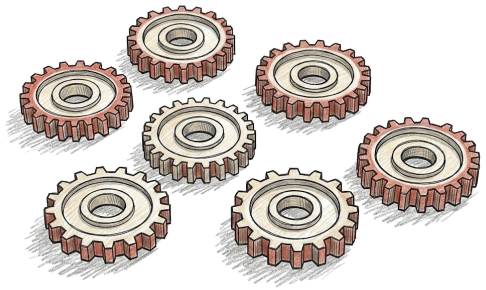




The reshape

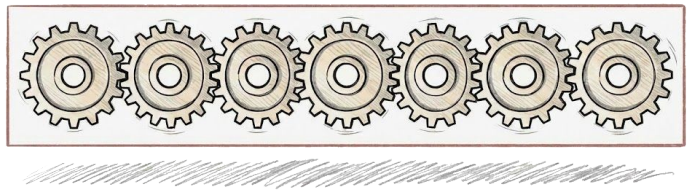
Before

- Seven **endpoint-shaped** tools
- The model orchestrates



After

- `diagnose_application(id)`
- One **outcome-shaped** result



The seven tools can stay



One synthesized result

```
// what state is it actually in
status

// the comparison a human would have made
instances: desired / running / healthy

// "do I need deployments at all?" answered FOR it
latest_deployment

// "how many logs is enough?" answered FOR it
recent_runtime_errors

// the judgement, already made
configuration_warnings
```



The Postmark case

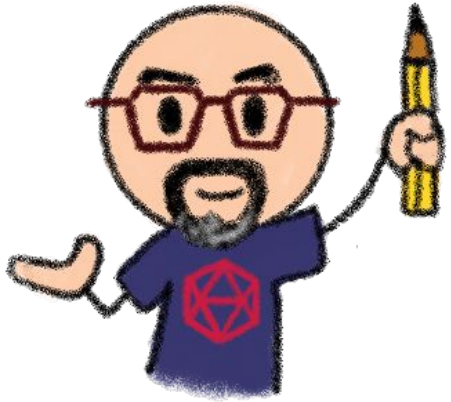
Email delivery platform, transactional email

- One `diagnoseDelivery` function
- Replaces the five calls a human would chain
- Fanned out in parallel
- Tolerant of any single lookup failing

Postmark



The principle



Make the **model decide** what requires **judgement**

Make **software do** what requires **computation**



Opposite directions, same principle

GitHub

- 101 → 52 tools

Postmark

- 4 → 24 tools

GitHub

Postmark

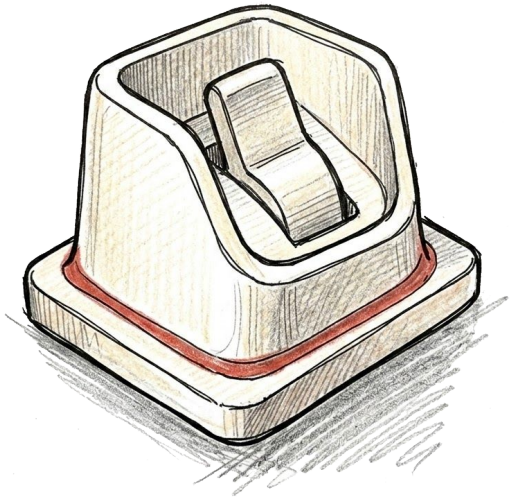
Not a 'many tools' vs 'few tools' problem

A from **endpoint-shaped** to **outcome-shaped** problem



REDESIGN

When no tool layer saves you

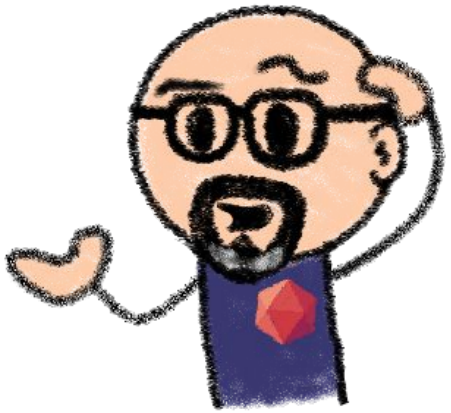




Two very similar endpoints

```
DELETE /organisations/{id}/applications/{appId}/addons/{addonId}
```

```
DELETE /organisations/{id}/addons/{addonId}
```

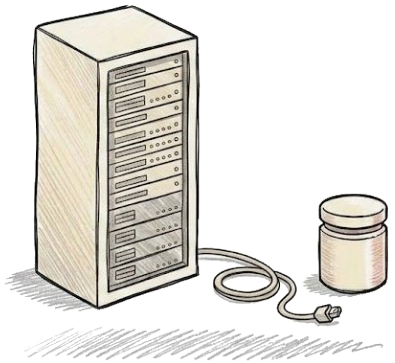




One unlinks, one destroys

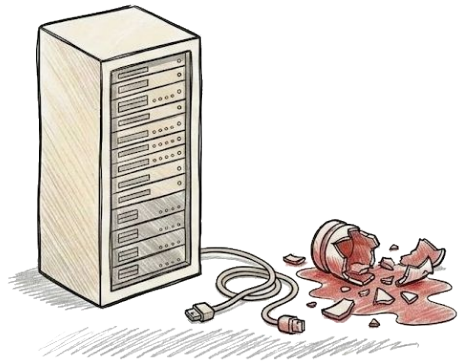
Scoped to the app

- Unlinks the add-on



Scoped to the org

- Destroys it, and its data

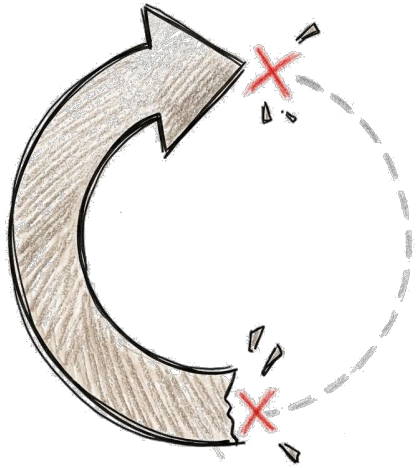


Same verb, same resource

One path segment apart

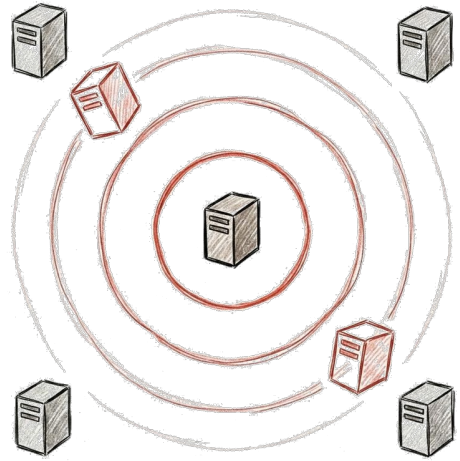


Two questions



Reversibility

Can I undo this?

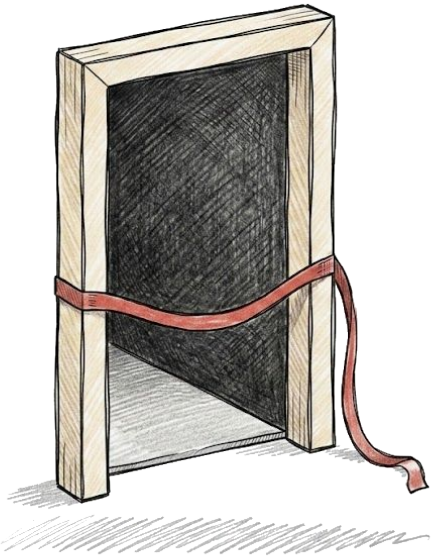


Blast radius

What else does this touch?



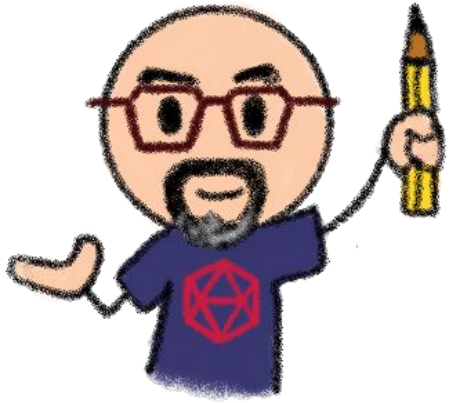
What to do? Just describe it?



WARNING! THIS DELETES THE DATABASE!!!



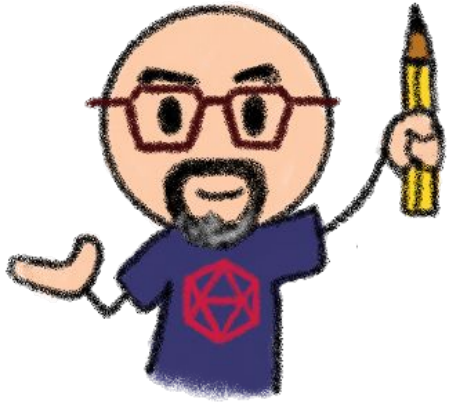
No



That is **not an architecture**



The distinction



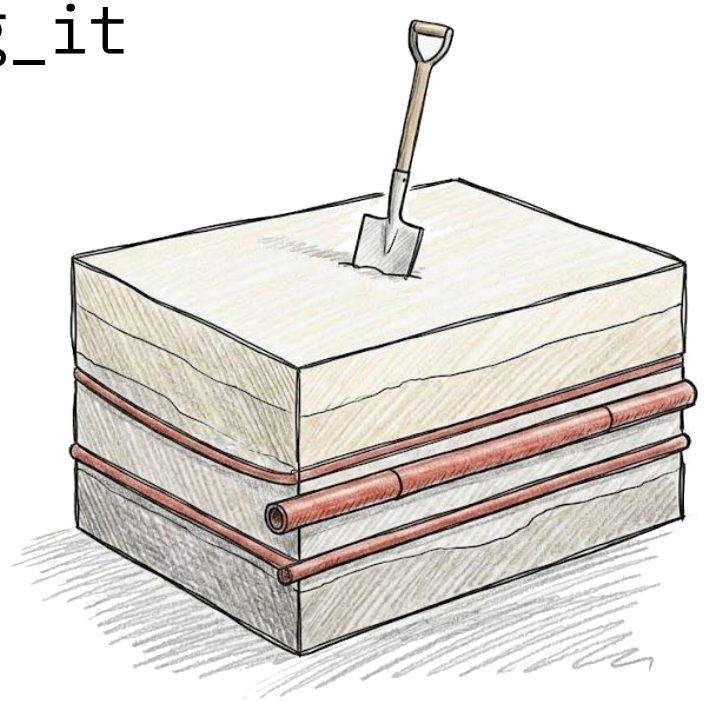
Reshape changes the **abstraction**

Redesign changes the **invariant**



Prepare

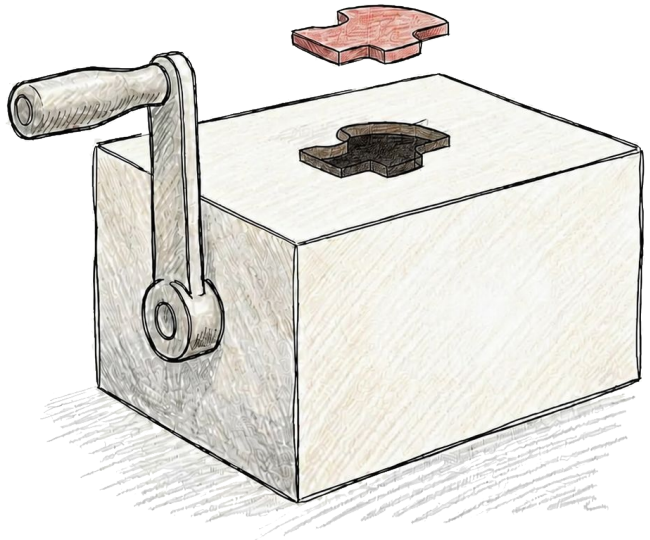
```
prepare_addon_deletion(addon_id)
    addon
    applications_using_it
    latest_backup
    recoverable_until
    consequences[]
```





Then execute

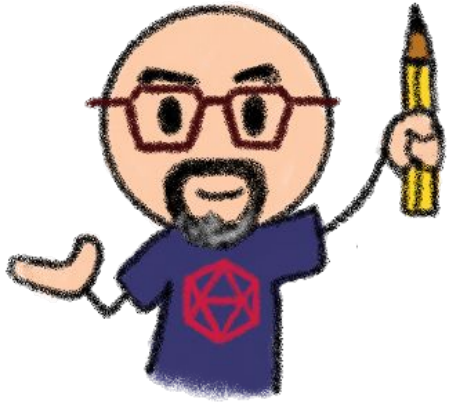
```
execute_addon_deletion(plan_id)
```



prepare → inspect → execute



Who owns safety



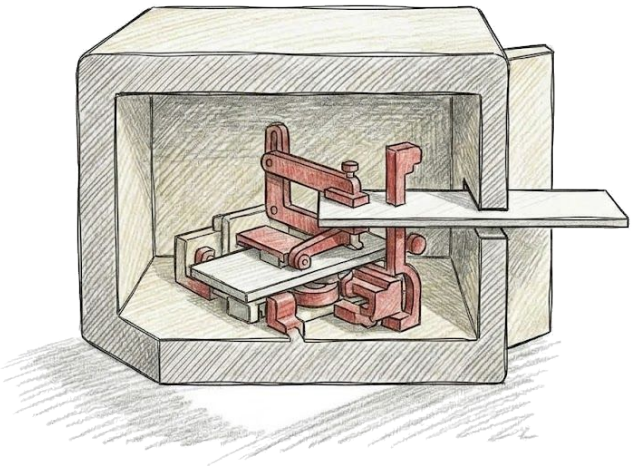
The system owns the safety boundary

Not the model



Stateful tools

- MCP spec guidance, 2026-07-28
- A creation tool returns an opaque handle
  - A reference to a server side object keeping the state
- Later tools accept it



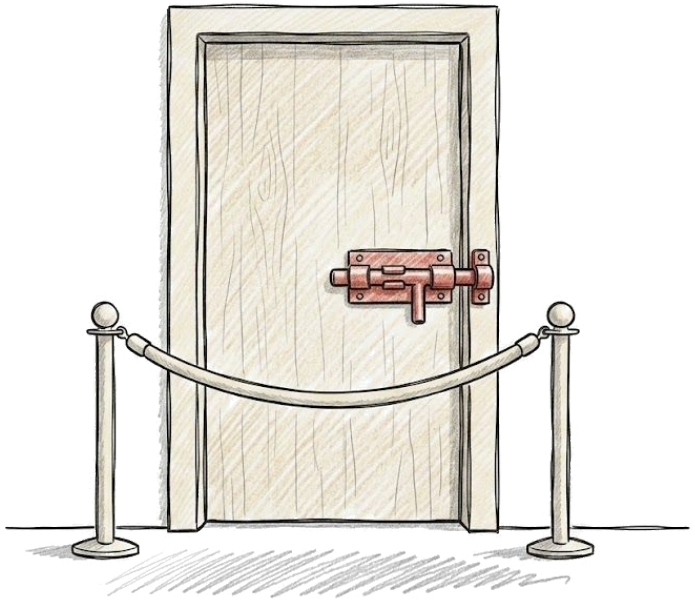
Re-validated on every call

An agent can invent an `addon_id`, it cannot invent a `plan_id`



Instructions vs enforcement

- Never make the LLM your authorization layer
- Prompt **instructions are UX**, **enforcement** belongs in **software**





How about the restart endpoint?

Before

- `POST /deployments`

  `202 Accepted`

- The restart takes time
- How does the LLM know it's done?

After

- `deploy_application(...)`

  `→ task_id`

Same failure as the env var



Now 'done' has a definition

Still working

- `queued` · `building` · `deploying`

Done

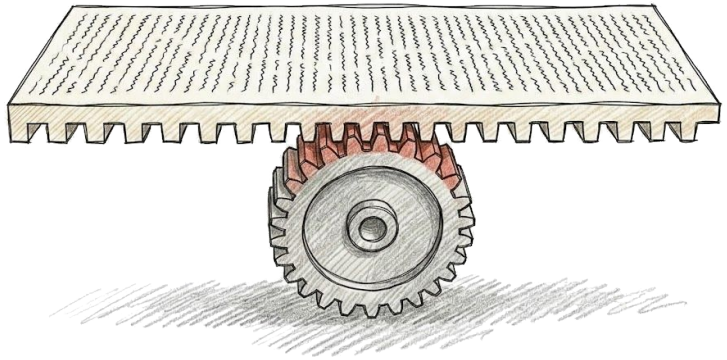
- `running` · `failed`

`failed` is done, not "not yet"



The description

When prose becomes behaviour





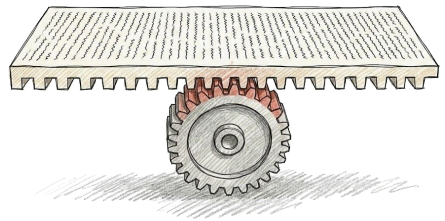
Documentation, or interface?

For a human

- **Interface** – enforced
- **Documentation** – inert, read once

For an agent

- The **description is the interface**
- Read at inference time, on every call



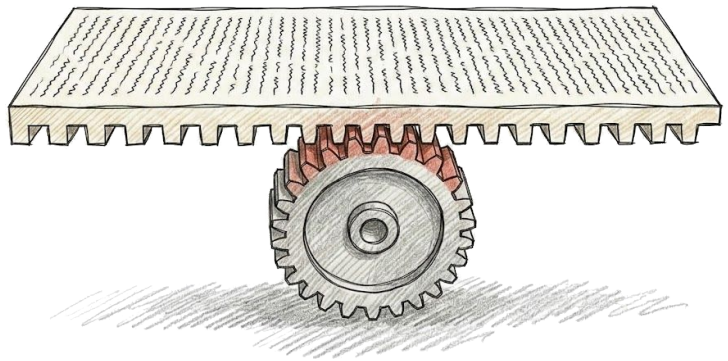
The prose is in the execution path



Two descriptions

Before

- `restart(id)`



After

- Purpose
- When to use it
- Side effects
- When not to use it



What rewriting the prose buys

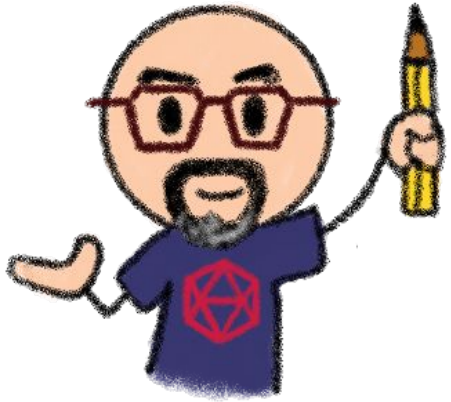
- Same models, same tools, same tasks
  - Only the descriptions changed
- Multi-step tasks completed: **33.5% → 44.6%**

StableToolBench – Learning to Rewrite Tool Descriptions

arXiv 2602.20426



Are my descriptions good?



How do you know you wrote it well?

You stop judging the text and start **measuring the behaviour**



Behavioural tests

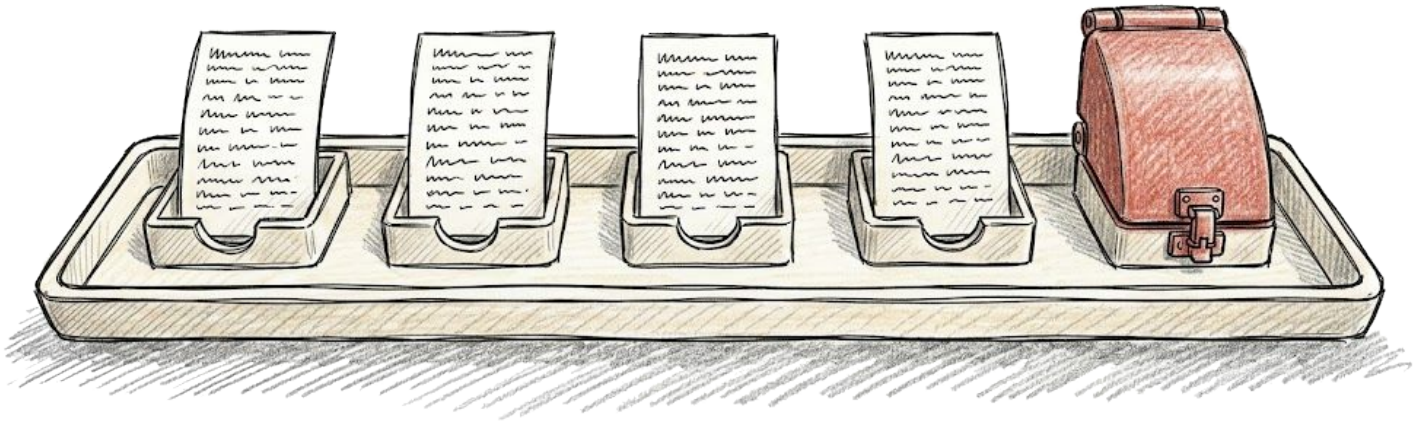
"Is production healthy?"

"Why did yesterday's deployment fail?"

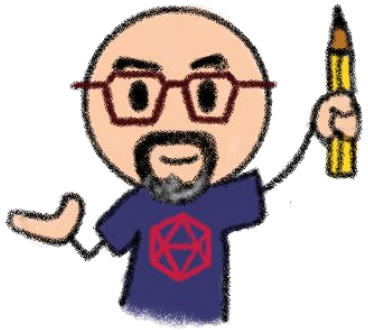
"Scale this app for tomorrow's traffic."

"Reduce our bill without causing downtime."

"Delete the test database."







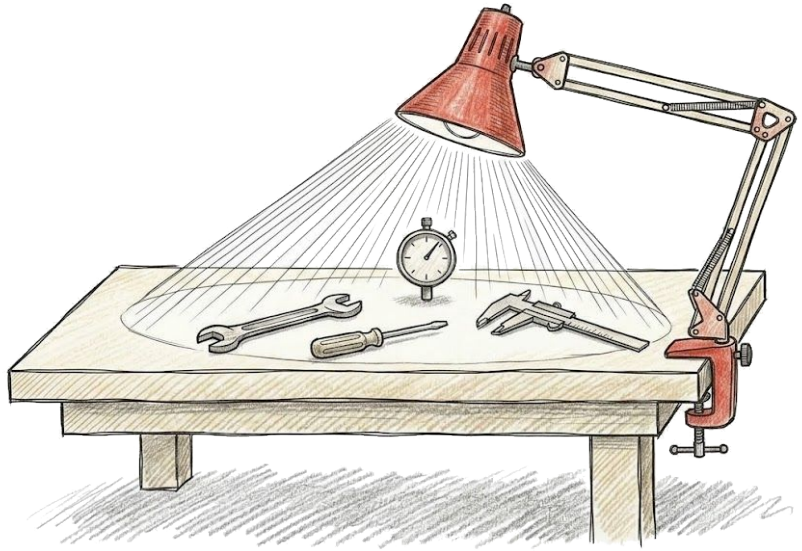
Don't guess

Evaluate



Better for humans

The part I promised in the abstract



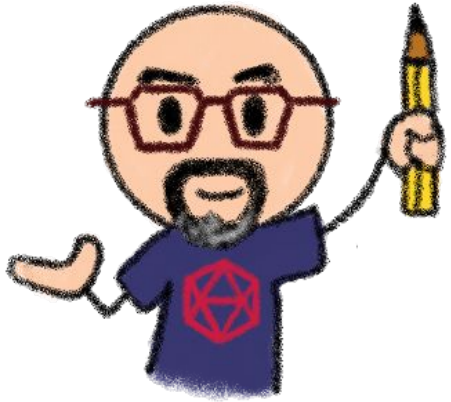


The walk-back

- env var → Completion
- diagnosis → Dependencies
- delete → Reversibility · Blast radius



Notice

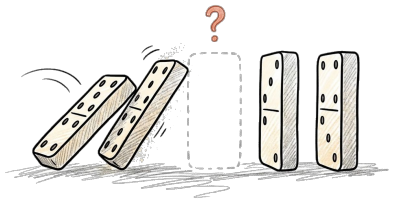


None of those are AI features

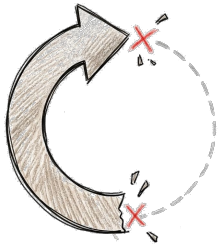
They're the things we always said we'd get round to.



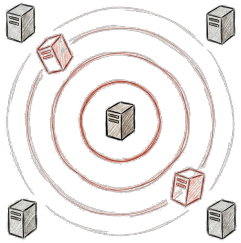
The four questions



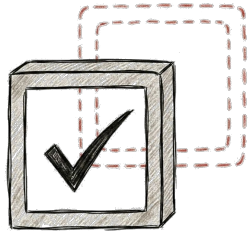
Dependencies



Reversibility



Blast radius



Completion



Agents are the users your API was never designed for

The good news is that designing for them means designing for the developer reading your docs at 2am...

and you already owed them that



That's all, folks!

Thank you all!

Vous avez les slides disponibles sur lostinbrittany.dev

