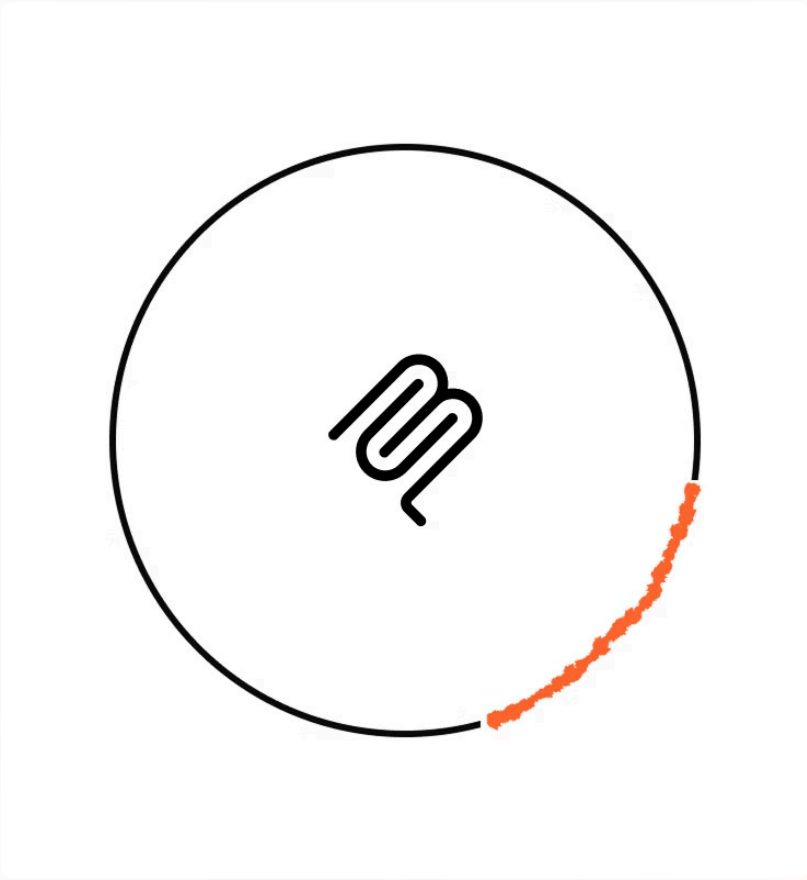
Architecting Agentic Engineering Loops With MCP

Tadas Antanavicius · **PulseMCP**



MCP Steering Group

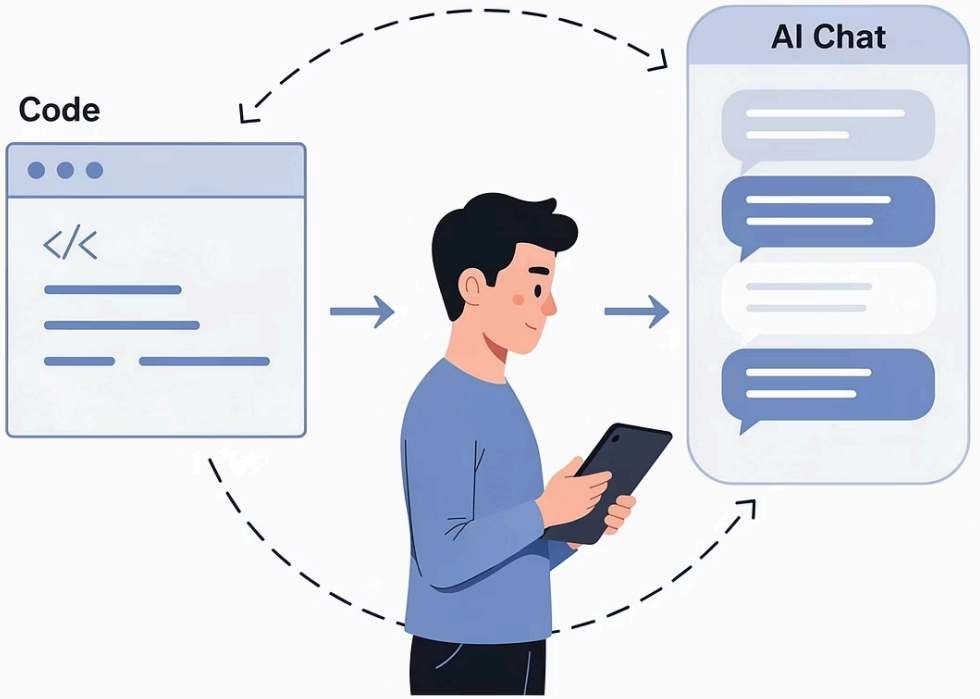
Maintainer of the MCP Registry and contributor community on GitHub and Discord.

PulseMCP

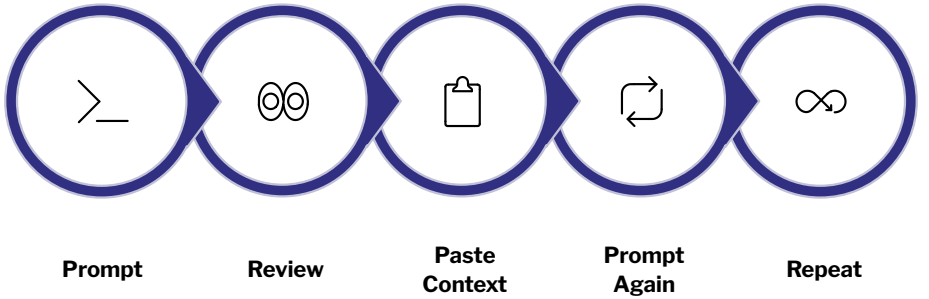
A newsletter & catalog for discovering MCP servers across the growing ecosystem.

Agentic Engineering

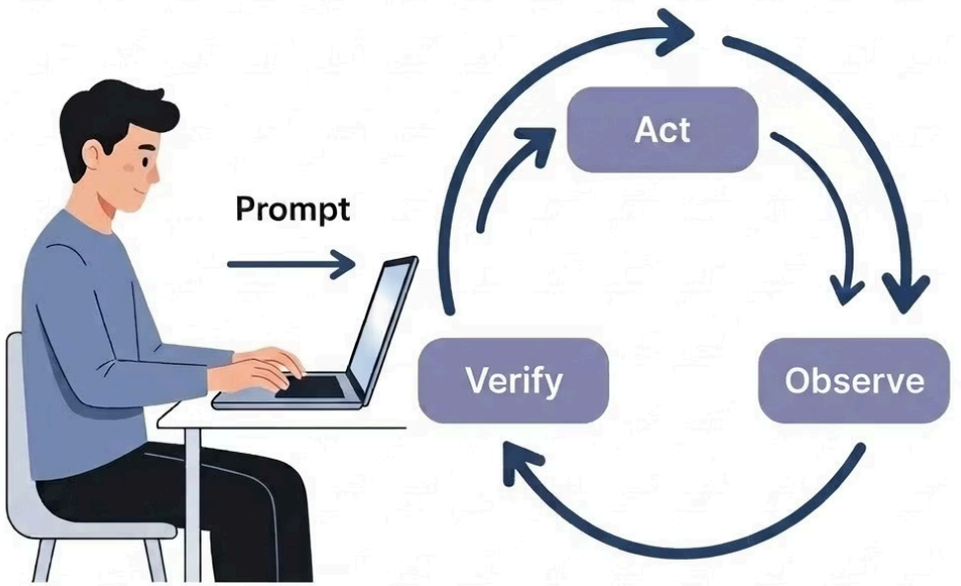
Hands-on team consulting to help engineering organizations level up with agentic workflows and tooling.



Many Engineers Use Coding Agents Like This



This manual, back-and-forth pattern yields a real but modest **10-30% productivity speed-up**. Many engineering teams are here.



THE NEXT STEP

But It Can Look Like This

Hand off an entire sequence to an agent operating autonomously inside a closed loop. The human (carefully) defines "done" — the agent handles the rest.

Human → Agent

One instruction triggers the full loop

100%+ Speed-Up

Requires rethinking "engineering"

What We'll Cover

01

What Is an Agentic Loop

The foundational concept

02

"Aha" #1: Closing the Loop

How to get an agent to spend 10 minutes productively

03

Demo 1: UI Feature Work

Implement a feature on a Linear clone with a Playwright MCP server

04

Demo 2: On-Call Triage

Triaging a support alert with Appsignal, Postgres, and Playwright

05

"Aha" #2: Parallelization

What it takes to get multiple loops going at once

06

Hands-on kata exercises

We'll go from app idea to deployed infra, practicing loops along the way

What Is an Agentic Loop?

> "Tools running on a loop to achieve a goal."
> — Simon Willison

The loop structure determines how much — if any — human involvement is required to make progress toward the goal.

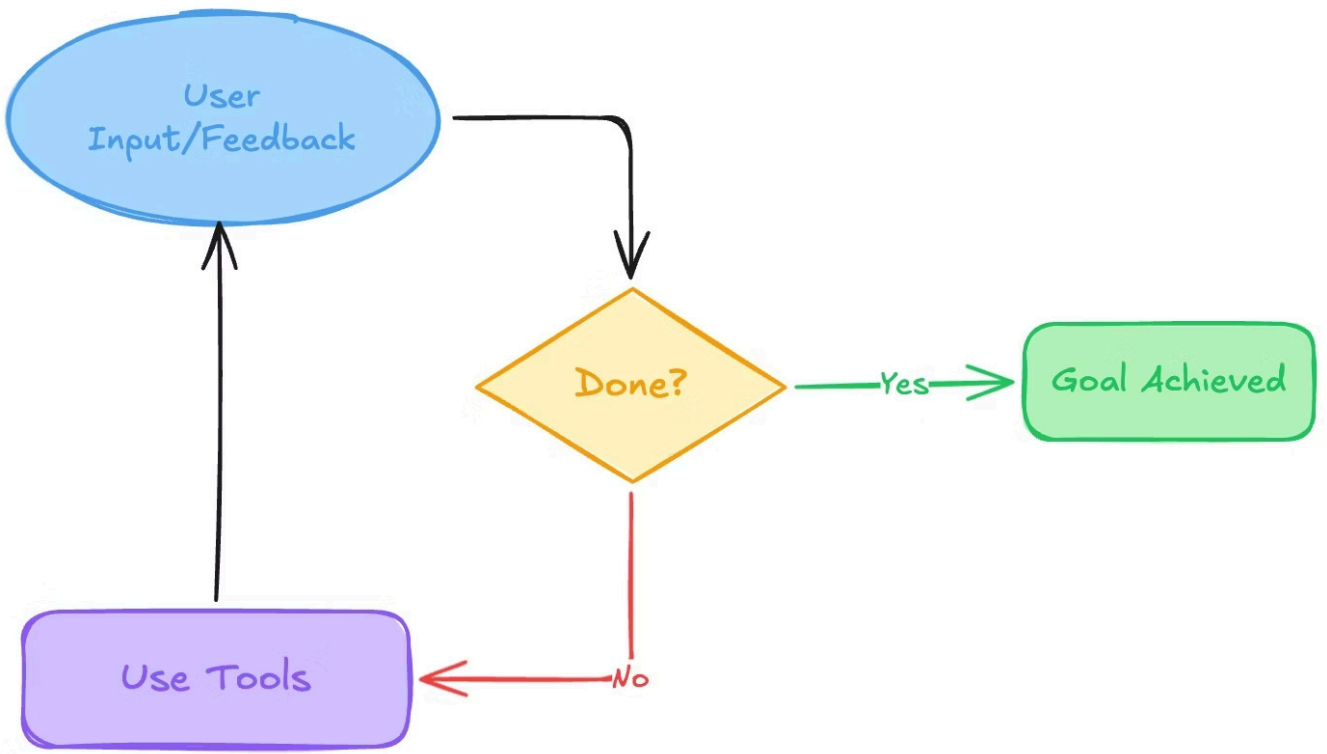
🔓 Open Loop

The human is expected to provide inputs, approvals, or babysitting at key points to keep the agent moving forward.

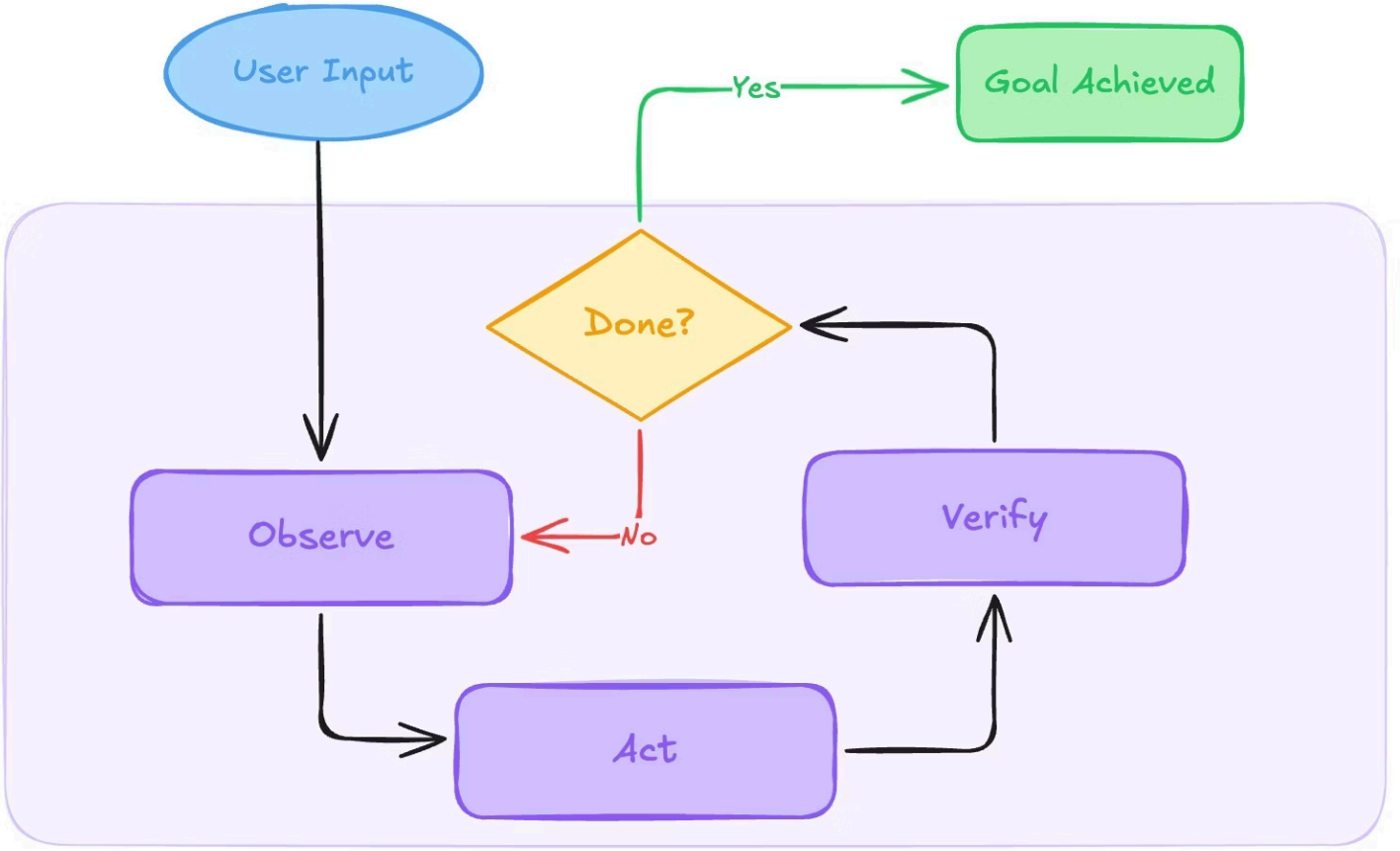
✅ Closed Loop

The agent self-verifies completion — or iterates with new context — without waiting for a human in the critical path.

Open Loop



"Aha" moment #1: Closed Loop



Closing the Loop: Verification

Step 1: Define Done

Your prompt must include a clear, unambiguous definition of done. Without it, the agent has no way to know when to stop.

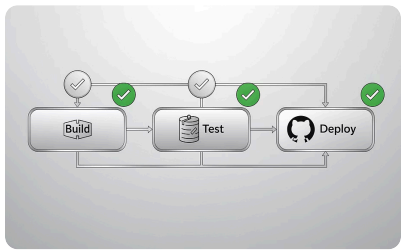
Step 2: Give Verification Tools

Equip your agent with the MCP tools it needs to check its own work — browser automation, CI pipelines, integration endpoints, and more.

Verification Examples

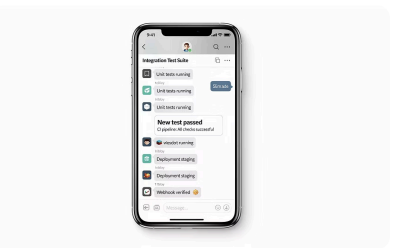
Playwright MCP

Click-test your UI automatically — the agent navigates, interacts, and confirms visual correctness without a human ever opening a browser.



GitHub Actions CI

Trigger your test suite on every iteration. The agent doesn't move on until all tests pass — a tight, objective signal of completion.



Slack MCP

Trigger a real message to validate your Slack integration. The agent tests the actual delivery path, not just the code path.

⚠️ The Golden Rule

If you are anywhere in the loop — even for something trivial — **it doesn't count as a closed loop.**

Example: manually authenticating into a production service to verify whether your agent's change is correct.

Why This Matters

Even a single manual step reintroduces latency, context-switching cost, and human error. The compounding productivity gains only materialize when the loop is truly closed.

Ask yourself: **Can the agent confirm success without pinging me?** If not, the loop is still open.

Keeping Loops Tight: Observability

A closed loop is the happy path. But a **tight loop** is the ideal — one where the agent has rich, real-time context at every step and rarely needs to spin without making meaningful progress.

→ Provide Meaningful Context Along the Way

Tools that surface logs, data state, and environment config let agents self-correct faster and with higher confidence.

→ Reduce Dead Ends and Blind Spots

The fewer unknowns the agent encounters, the fewer wasted iterations — and the faster you get to a working result.

Observability Examples

Logging Infrastructure

Expose structured logs and stack traces so the agent can diagnose failures without guessing what went wrong at runtime.

Database Access

Let the agent query actual data state to understand what's in the system — not just what the code assumes is there.

Env Vars & Feature Flags

Surface which code paths are currently active so the agent reasons about behavior in the right configuration context.

Demo 1: Implementing a UI/UX feature

1

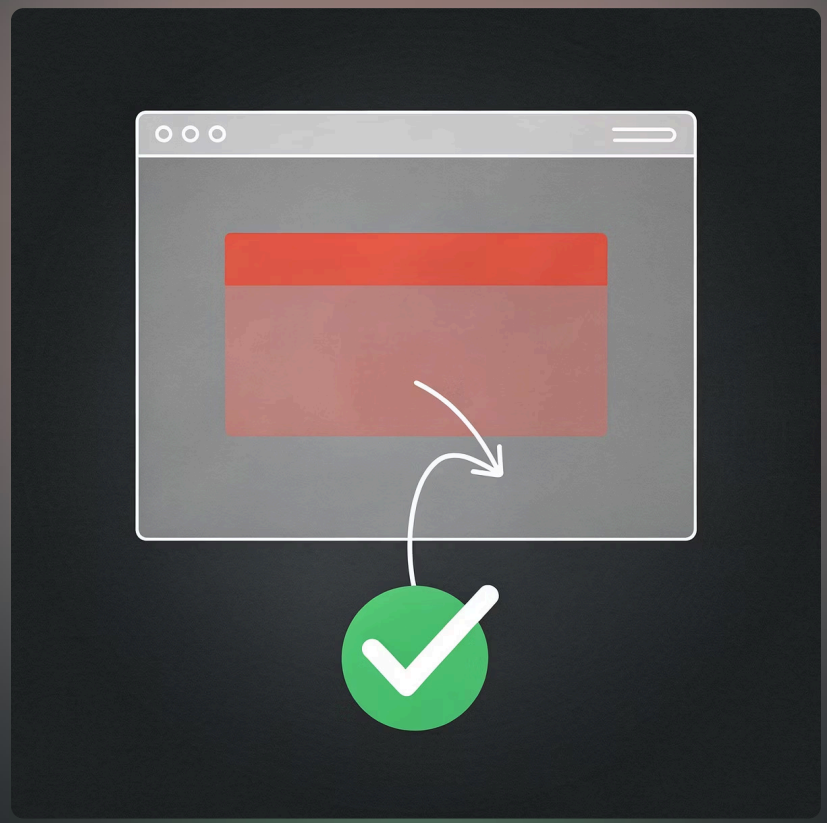
Scenario

Your task is to add some new UX flow to your app — the agent is tasked with identifying, implementing, and verifying the fix end-to-end.

2

Tool: Playwright MCP

The agent uses Playwright to navigate the UI, understand the scope, make an implementation, and confirm the correct rendered outcome — no human in the loop.



MCP's Role: Connecting & Constraining

The standard interface for giving agents access to tools and data — safely.

Connecting

MCP servers plug agents into real infrastructure — integrations, databases, CI/CD, monitoring — through structured tool definitions designed for LLMs. This is what makes closed loops practical.

Constraining

MCP configurations scope exactly what an agent can do: read-only mode, filtered tools, specific environments. Dangerous actions become structurally impossible.

Demo 2: Triaging an On-Call Alert

Slack MCP

Serves as our entrypoint of alerts to triage

Appsignal MCP

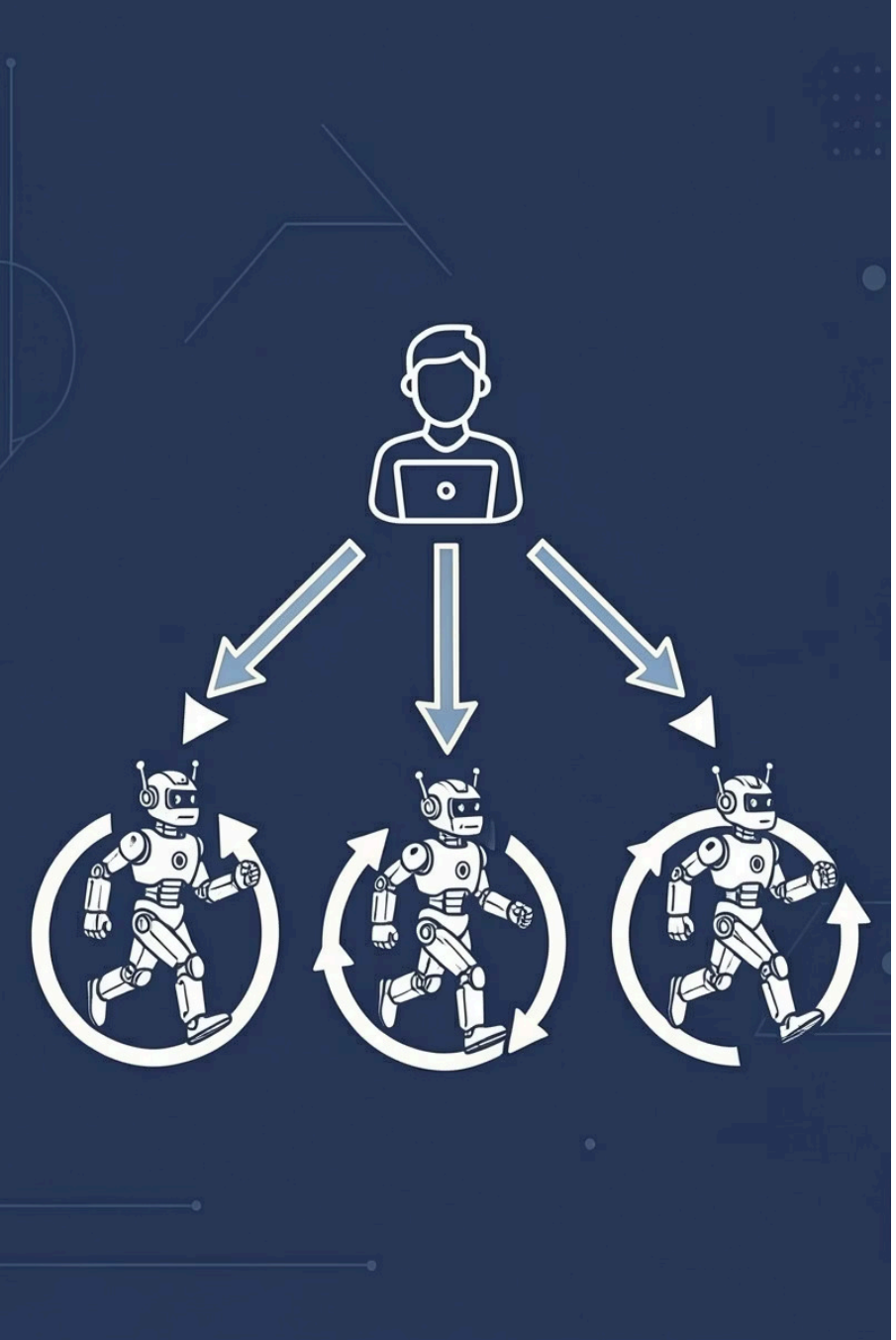
Used to pull down logs and other metrics related to the alert

Postgres MCP

Queries the actual database state to understand the data conditions that triggered the alert

Playwright MCP

Reproduces the failure in the UI to visually confirm the issue and ultimately verify the resolution



"Aha" #2: Parallelization

One engineer. Several coding agents at once.

Parallelization Gotchas

Conflicting ports

You can't run two instances of your app if they both use localhost:3000

External integrations

External services you can't replicate in your dev environment are painful

ClickOps infrastructure

If your infrastructure requires clicking around, you can't easily spin up replicas

Sharing a git clone / git branch

You'll quickly regret merge conflicts and trampling problems

Parallelization Solutions

git clone or worktrees

Isolate each agent with its own copy of the code

Containerize Your Environment

Spin up fresh, reproducible environments locally or on a remote machine

Mock or Isolate Dependencies

Simulate third-party services or run dedicated instances per agent

Coding Kata Exercises

github.com/pulsemcp/content/mcp-dev-summit-nyc-2026

pulse

Work With Us

Custom Enterprise & Team MCP Servers

We build and maintain **high quality MCP servers** tailored to your internal deployment, compliance, and functional needs. One-time fees: you own the code, and we're around for ad-hoc improvements.

Agentic Engineering Consulting

We run hands-on consulting engagements with **engineering teams** looking to upskill on agentic engineering patterns, ramp on Skills and MCP, and track team-wide KPIs as they go along.

Interested? Reach out at **tadas@pulsemcp.com**

Find these slides at github.com/pulsemcp/content