From data dumps to smart context

Building MCP servers that AI can actually use

Thomas Johnson / CTO, Multiplayer

Observability Summit 2026 / May 21, Minneapolis

Where observability is going

Pull becomes push.

You don't go through dashboards to get data. The data comes to you. MCP triggers will let servers proactively push updates.

Open loop becomes closed loop.

Observability becomes an active participant in the system, not a passive record.

Data is pre-correlated by default.

By session, by user, by deploy. Not scattered across disconnected tools.

Collect more when you need it, not all the time.

Dynamic escalation, not permanent firehose.

Agents work together, humans in the loop when needed.

SRE agents, coding agents, QA agents. Collaborating autonomously.

Systems heal themselves.

Observability becomes invisible. Not gone. Just working.

Thomas Johnson

CTO, Multiplayer

We build a debugging agent that runs next to your coding agent and connects it to prod to fix bugs automatically.

We shipped a production MCP server to make that work.

Most of what follows, we learned the hard way.

Self-Healing...But we're not there yet.

1.7x
more bugs in AI code

+23.5%
incidents per PR

+30%
change failure rate

AI agents write code fast. They debug blind. Code velocity is up. Production stability is down.

The industry responded with MCP (110M+ SDK downloads/month, faster adoption than React):

Grafana	Datadog	Sentry	PagerDuty	Honeycomb	New Relic	OpenObserve	Instana

But it’s not counteracting these trends: more code = more issues.

Why observability is the hardest domain for MCP.

Problem 1: The context window tax

MCP loads all tool definitions at startup. Each tool = name + description + full parameter schema.

40+

tools in Grafana's MCP server

~20K

tokens before any query

144,802

tokens from MCP tools alone

72% of the context window gone before the agent does anything.

Problem 2: Data-centric tools, not goal-centric

What most servers expose

```
get_dashboards
query_loki
query_prometheus
search_tempo
list_alerts
get_oncall
list_incidents
get_metrics
```

What agents actually need

```
investigate_error

trace_user_journey

explain_deployment_impact
```

Agents perform better with fewer, semantically distinct tools. - Anthropic research

Problem 3: The worst signal-to-noise ratio for AI

A single trace can have hundreds of spans

A single session generates thousands of log lines

A single incident can touch dozens of services

Observability data requires active curation, not passive exposure.

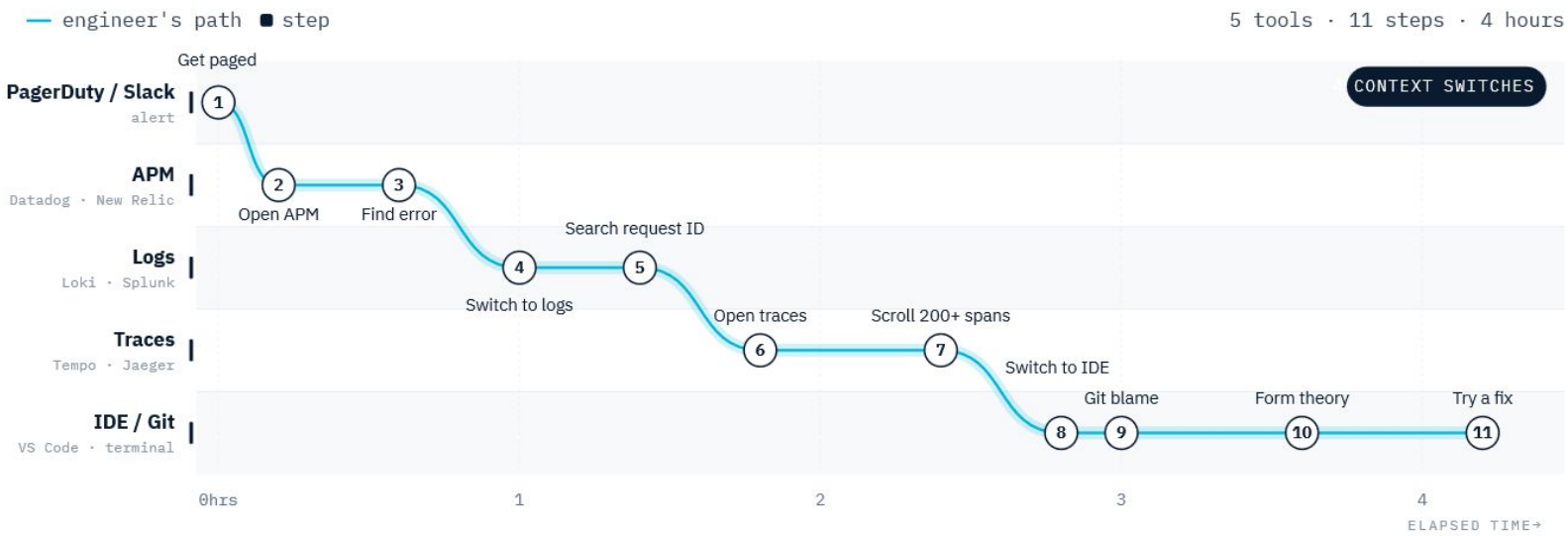
Most observability MCP servers answer:

"How do I give AI access to my telemetry?"

The right question:

"What does the agent need to understand about this incident?"

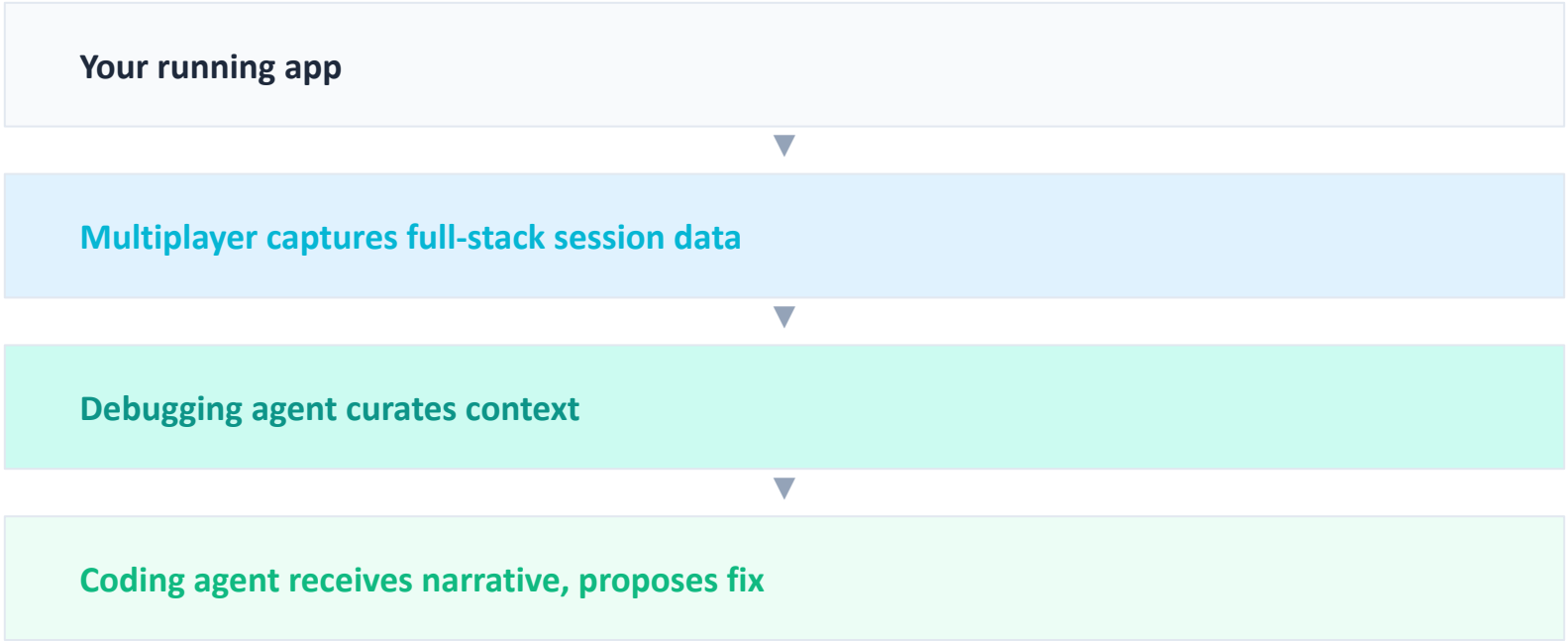
The debugging workflow you already know



What if the data just came to the agent?

What we learned.

A debugging agent next to your coding agent



Runs locally. Your code never leaves your machine.

V1: Mirrored our API

- `get_session`
- `get_errors`
- `get_traces`
- `get_logs`
- `get_screenshots`
- `get_spans`
- `get_headers`
- `(...lots more)`

15+ tools. Wrong order.

Missed critical context.

V2: One tool. One intent.

Agent says:

“Get the customer issue on the dashboard after yesterday's deploy"

Server returns:

- Session correlated traces and logs
- Key error + stack trace
- Release and deployment metadata
- Frontend screens for the session

Pre-filtered. Contextualized.

The context impact

Before: data-centric

15+ tools

~10k tokens

Agent reasons about tool selection
on every step. More wrong turns,
more tokens burned, worse output.

After: intent-aware

1 tool

~600 tokens

No tool selection overhead.
Agent focuses on the problem.
Better focus = better output.

~87% reduction in schema overhead. Fewer tools means better focus, not just fewer tokens.

Our debugging Skill

SKILL.md

```
---
name: multiplayer-debug
description: >
  Use when investigating production bugs, debugging application errors,
  fixing issues reported by QA or customers, or analyzing session recordings.
  Also use when a user mentions Multiplayer issues, sessions, recordings,
  or asks to fix a bug from production. Do NOT use for feature development,
  refactoring, or tasks unrelated to debugging production behavior.
---

# Multiplayer Debugging Skill

You are investigating a production issue using Multiplayer's debugging agent
and MCP server. Your goal is to understand what went wrong, identify the root
cause in the codebase, and propose or apply a fix.

## Step 1: Identify the issue

Determine what you're investigating. The input may be:
- An issue ID or URL from Multiplayer (e.g. "fix issue MP-1234")
- A session recording ID or URL
- A description of the bug ("users see a 500 on checkout")
- A customer or QA report with notes

If you have an issue ID, fetch it:
```
multiplayer issues get <issue-id>
```

If you need to search for matching issues:
```
multiplayer issues search --query "<description>"
```
```

~100 tokens at startup

Only name + description loaded

~850 tokens when activated

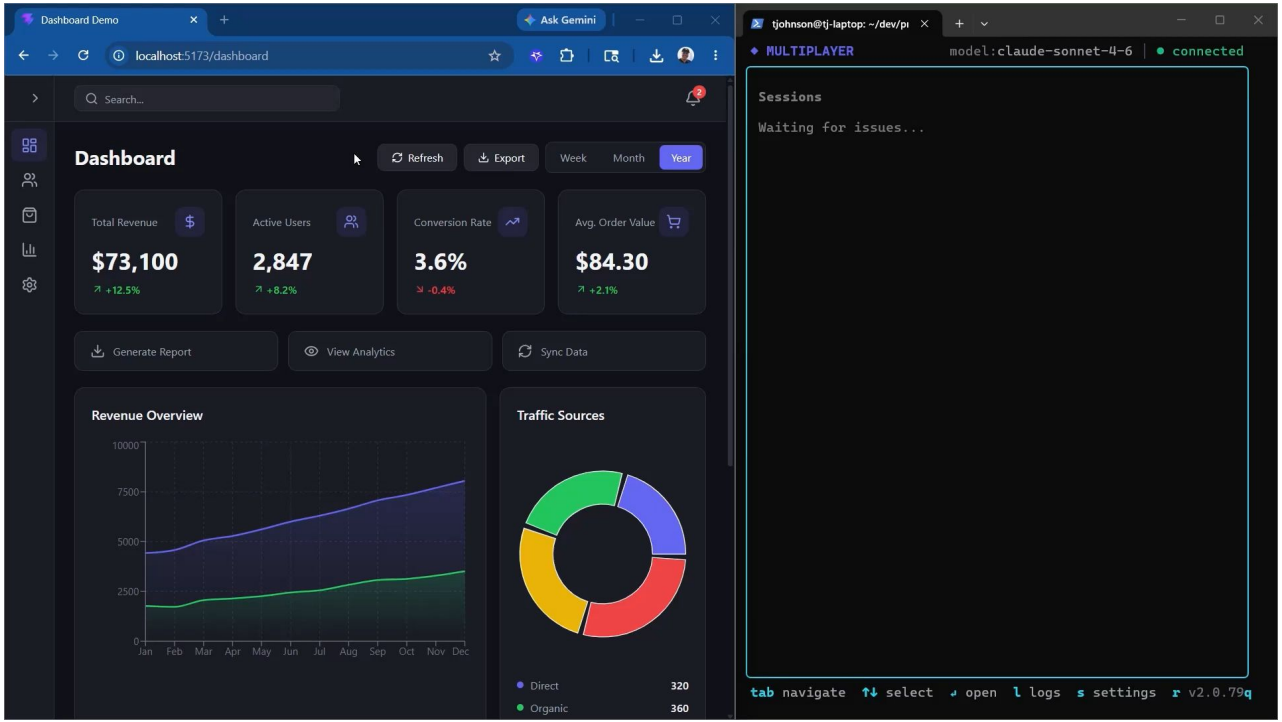
Full instructions loaded on demand

Composes: MCP + CLI + config

The official MCP roadmap now includes skills-over-MCP as an experimental feature.

Coming soon: skills served over MCP. Connect to a server and get the workflow knowledge automatically.

The debugging agent in action



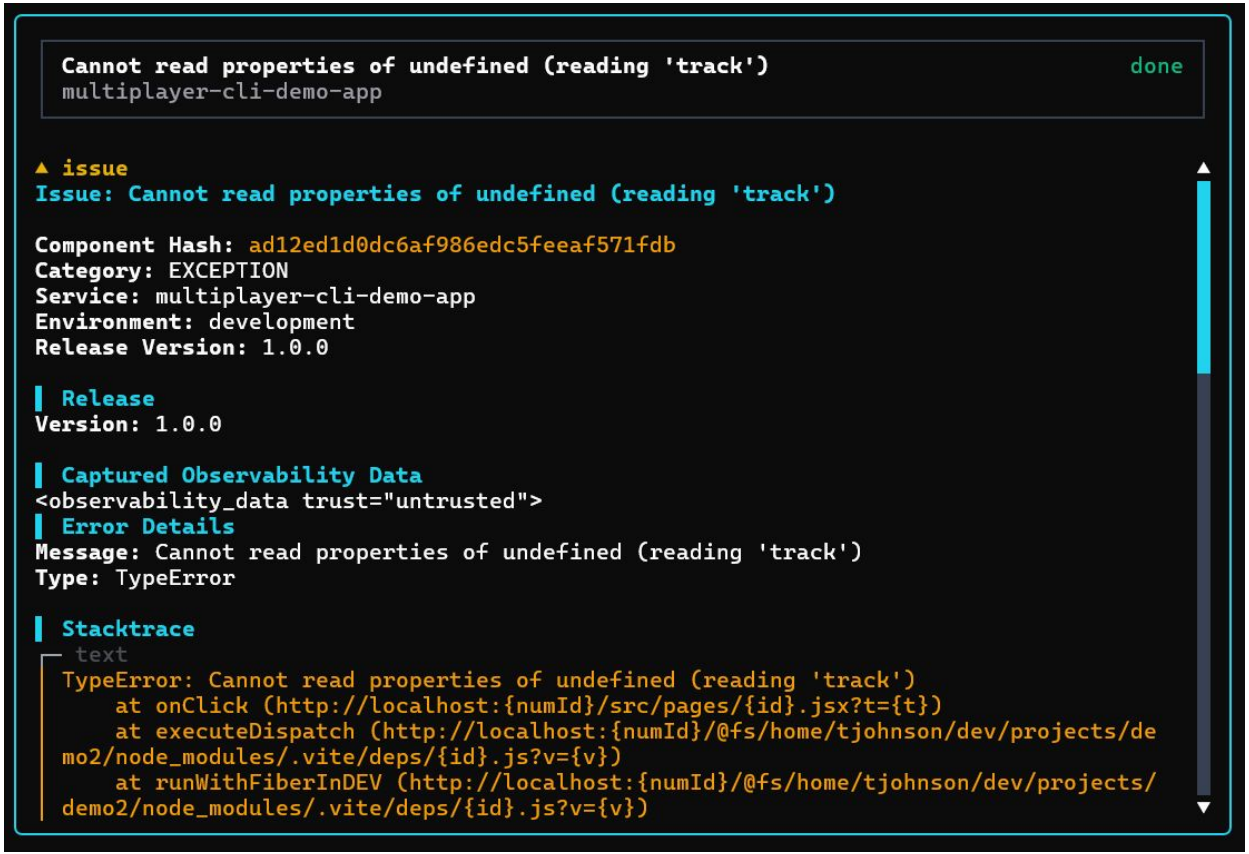
One MCP call.

Narrative response.

CLI for coding agent + git.

90 seconds, not hours/days.

What the agent actually receives

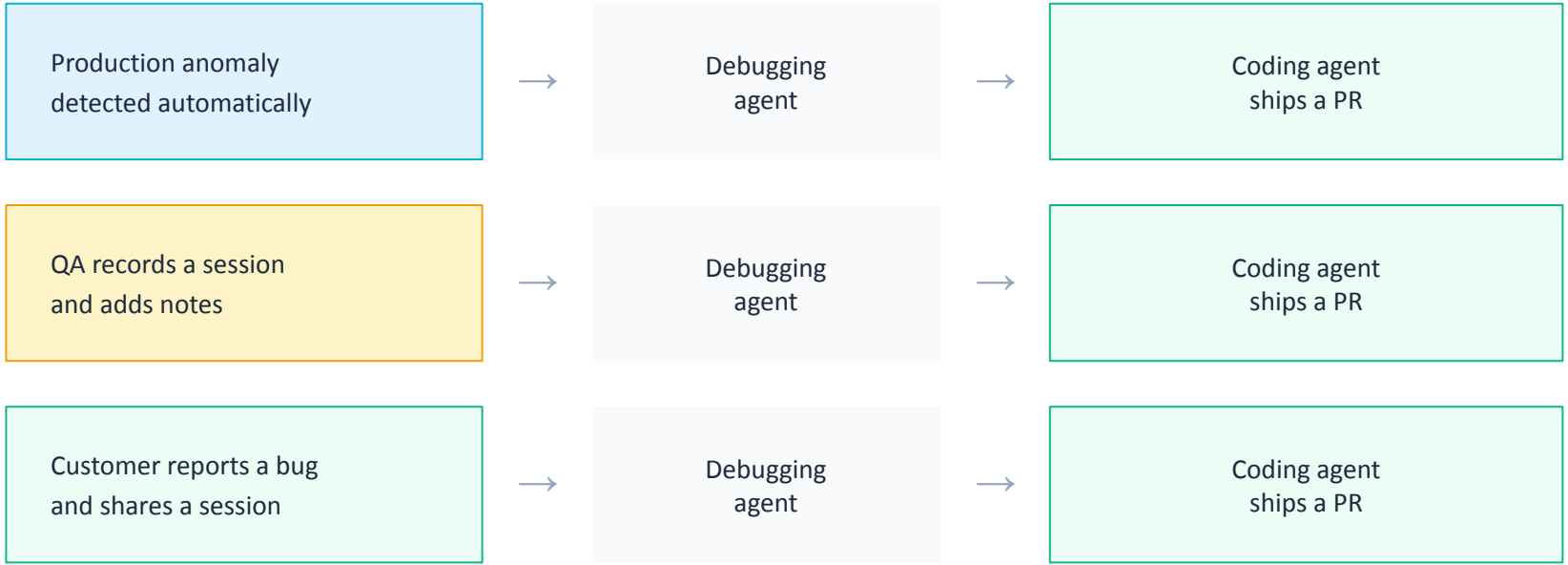


Pre-filtered.

Contextualized.

The agent gets understanding, not broad access.

It's not just prod-to-PR



The person experiencing the issue IS the sensor. That's data telemetry can't capture.

The ecosystem is converging on to this approach.

The connectivity stack

Skills define the workflow

Progressive disclosure. Just markdown.

MCP stateful data access

Live telemetry, sessions, authenticated APIs.

CLI tools composable execution

git, docker, curl | jq. 4-32x cheaper.

Coming soon: skills served over MCP. The layers are converging.

The emerging pattern

Agents write small scripts composing all three layers in one turn.

Programmatic tool calling is now on the official MCP roadmap.

One turn of code replaces twelve turns of tool calls.

The ecosystem is converging...but it's bumpy

Claude Code	Shipped Tool Search (Jan 2026). MCP tools over 10K tokens deferred.	**85% token reduction**
Google Gemini CLI	Deprecating eager MCP in extensions for skill-based discovery.	**Skills-first by default**
Amp (Sourcegraph)	Skills + mcp.json: load MCP tools only when skill activates.	**26 tools to 4 on demand**
Perplexity	Deprecated MCP internally (March 2026). REST APIs and CLI instead.	**Cited token overhead**
MCP Roadmap	Progressive discovery, skills-over-MCP, and programmatic tool calling confirmed for 2026.	**Official direction**

Staged context, not eager loading.

Let’s make an agent that debugs a production issue

Skill: "Investigate production error"

1. Read the error report
2. Query observability MCP for session context
3. Run `git log --since="2 days ago"`
4. Cross-reference trace with diff
5. Propose root cause and fix

MCP

Filtered session, trace, screenshots

CLI

git log, grep, test runners

Config

Architecture and conventions

The Skill teaches the agent how to debug YOUR system.

Scope is a security decision

Data-centric server

40 tools expose your entire stack.

Trusting the AI to stay in scope.

Every tool is attack surface.

Intent-aware server

1 tool scoped to a workflow.

Blast radius naturally limited.

Server decides what's relevant.

53% of MCP servers still use static API keys. The spec now requires OAuth 2.1.

Budget real engineering time for auth. It took us longer than the MCP server itself.

Four principles for AI-ready observability.

1 Design for the agent's investigation, not your data model.

"Investigate this error" beats
get_traces + get_logs + get_sessions.

2 Return narratives, not data dumps.

Pre-filter. Correlate. Contextualize.
The agent needs understanding, not access.

3 Use all three layers.

Skills define the workflow.
MCP provides live data.
CLI handles the rest.

4 Let your senior engineers encode their intuition.

A Skill that captures how your best debugger works is worth more than any MCP server.

Self-healing is not science fiction. It’s coming.

The MCP roadmap is building the primitives that make it work:

Triggers	MCP servers proactively notify agents when something goes wrong. Push, not pull.
Long-running tasks	Agents kick off async investigations that run while you sleep.
Cross-app access	One identity flow across SRE agent, debugging agent, coding agent. No repeated logins.

SRE agent detects issue > debugging agent investigates > coding agent fixes > no human paged.

That's where observability is going.

Thomas Johnson

CTO, Multiplayer

multiplayer.app

DZone: "MCP Servers Are Everywhere, but Most Are Collecting Dust"

LeadDev: "Current observability tools are fundamentally broken for AI debugging"