dash0

MCP Handed Observability to OpenTelemetry. Now What?

Ayooluwa Isaiah · Technical Content Manager at Dash0

freshman.tech · /in/ayisaiah · @ayisaiah



MCP
Dev Summit
Nairobi

Agenda

01 Monitoring MCP operations

02 Logging for effective debugging

03 Distributed tracing with MCP

04 Operational considerations



Ayooluwa Isaiah

Technical Content Manager
at DashO

freshman.tech

Co-author of OpenTelemetry for Dummies
Wiley, 2025

OpenTelemetry Contributor

Google Developer Expert
Web Technologies

OBSERVING MCP

Four layers. Four different questions.

Transport `POST /mcp`	Did the request arrive?
MCP method `tools/call`	What operation was requested?
Tool execution `search_orders`	Did the tool do its job?
Dependencies `SELECT orders · GET /stock`	What did the tool depend on?

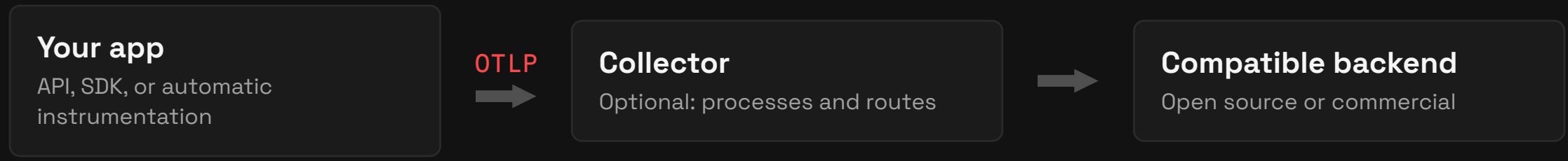
Observability means seeing all four layers

INTRODUCING OPENTELEMETRY

What is OpenTelemetry?

An open-source, vendor-neutral framework for generating, collecting, and exporting telemetry.

A CNCF project with SDKs for Go, Java, Python, JavaScript, .NET, and more.



Your instrumentation isn't tied to an observability vendor.

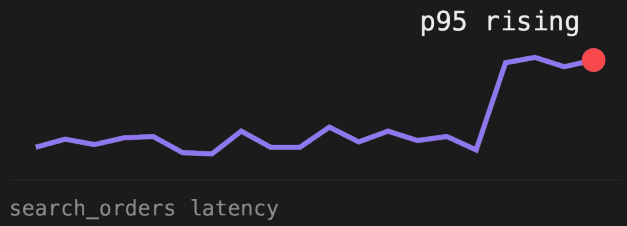
WHAT OPENTELEMETRY COLLECTS

One tool call, three views

METRICS

Like your car's warning lights

Is something wrong?



LOGS

Diagnostic records

What happened?

```
"level": "warn",
"msg": "slow tool call",
"gen_ai.tool.name":
  "search_orders",
"duration_ms": 900
```

TRACES

Like a GPS replay of the route

Where did the time go?



The same search_orders call, answering three different questions.

OPENTELEMETRY SEMANTIC CONVENTIONS

Standard names for the same thing

BEFORE · EACH SERVICE PICKS ITS OWN NAMES

```
python server
tool=search_orders status=error reason=timeout

go server
function_name=search_orders
error=deadline_exceeded

node server
operation=search_orders err=timeout
```

AFTER · EVERY SERVICE USES THE CONVENTION

```
python server
gen_ai.tool.name=search_orders error.type=timeout

go server
gen_ai.tool.name=search_orders error.type=timeout

node server
gen_ai.tool.name=search_orders error.type=timeout
```

Consistent attributes make telemetry easier to understand across services and languages.

METRICS · START HERE

Start with one metric

mcp.server.operation.duration

A duration histogram in seconds, with one measurement per completed MCP operation.

DIMENSIONS TO START WITH

mcp.method.name	Which MCP operation	tools/call
gen_ai.tool.name	Which tool	search_orders
error.type	What went wrong, on failures	tool_error

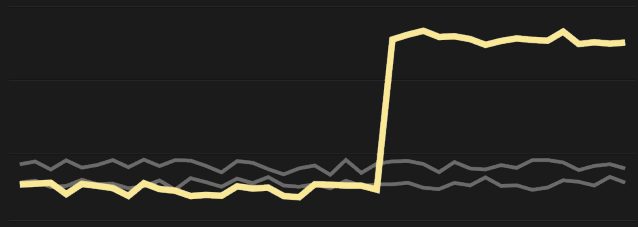
METRICS · WHAT YOU GET

One metric, three questions

TRAFFIC

How busy are my tools?

Histogram count

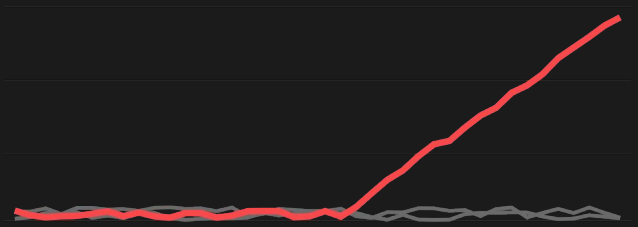


list_customers jumps 6×: a runaway loop?

ERRORS

Which tools are failing?

Count filtered by error.type

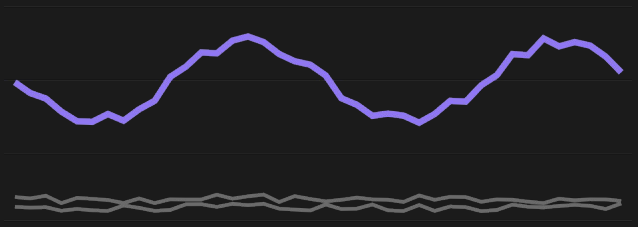


create_ticket climbing, behind HTTP 200

LATENCY

Which tools are slow?

Histogram distribution



search_orders p95 above its target

search_orders is slower than it should be.

METRICS

Alerts that catch MCP problems

Alert condition	Possible explanation
Sustained high error ratio	Broken tool, invalid arguments, dependency failure
Latency above target	Slow dependency, contention, expensive execution
Unusual call volume	Increased demand, repeated calls, agent loops

Alert on symptoms. Investigate with logs and traces.

An alert fired.
Logs tell you what happened.

LOGS

Log in a structured format

FREE TEXT

```
2026-11-19T09:14:02Z WARN
slow tool call:
tools/call search_orders
(request 42) succeeded
after 900ms
```

STRUCTURED (JSON)

```
{
  "time": "2026-11-19T09:14:02Z",
  "level": "warn",
  "msg": "slow tool call",
  "mcp.method.name": "tools/call",
  "gen_ai.tool.name": "search_orders",
  "jsonrpc.request.id": "42",
  "outcome": "ok",
  "duration_ms": 900
}
```

USE A STRUCTURED LOGGING LIBRARY

pino (Node.js) · structlog (Python) · slog (Go) · Semantic logger (Ruby) · Serilog (.NET) · Log4j 2 (Java) · tracing (Rust)

LOGS

Log failures, slow calls, and important state changes

- MCP operations
  - Failed calls, slow calls, protocol errors
- Dependencies
  - Database failures, upstream timeouts, retries
- Security and operations
  - Access denials, approval decisions, deployments, configuration changes

Record events at the component responsible for them.

LOGS

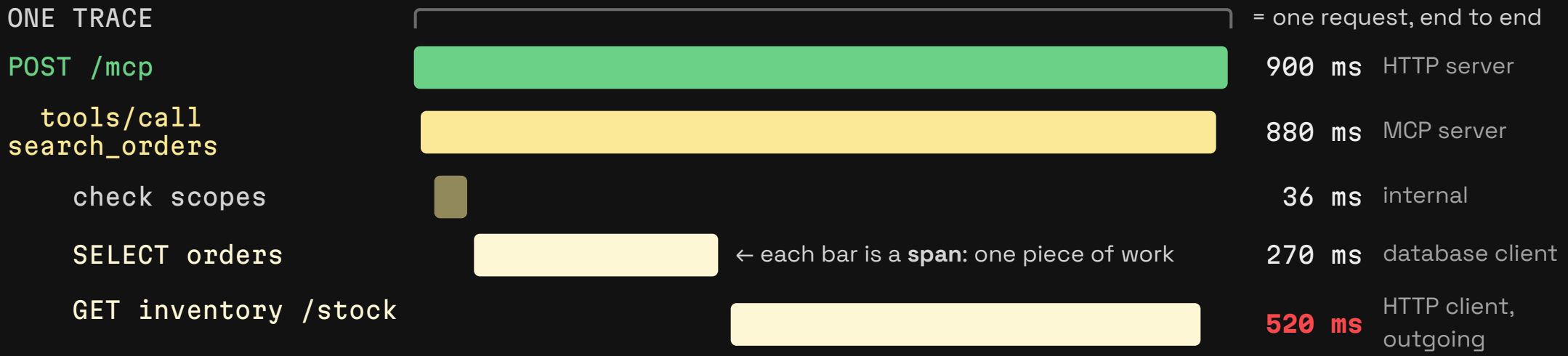
Every log should tell you what operation produced it

Field	Why it matters
User ID	Who was affected
Client	Which agent or app made the call
Server and version	Which build handled the request
Method and tool	What was asked
Inputs	Spot bad or unexpected arguments. Never secrets or personal data.
Outcome	Success, tool error or protocol error
Duration	Which calls were slow
Scopes	What the caller was allowed to do

The logs say a tool call was slow.
Where did it spend its time?

TRACING

Find the work that made the tool slow



TRACING · DEBUGGING SCENARIO

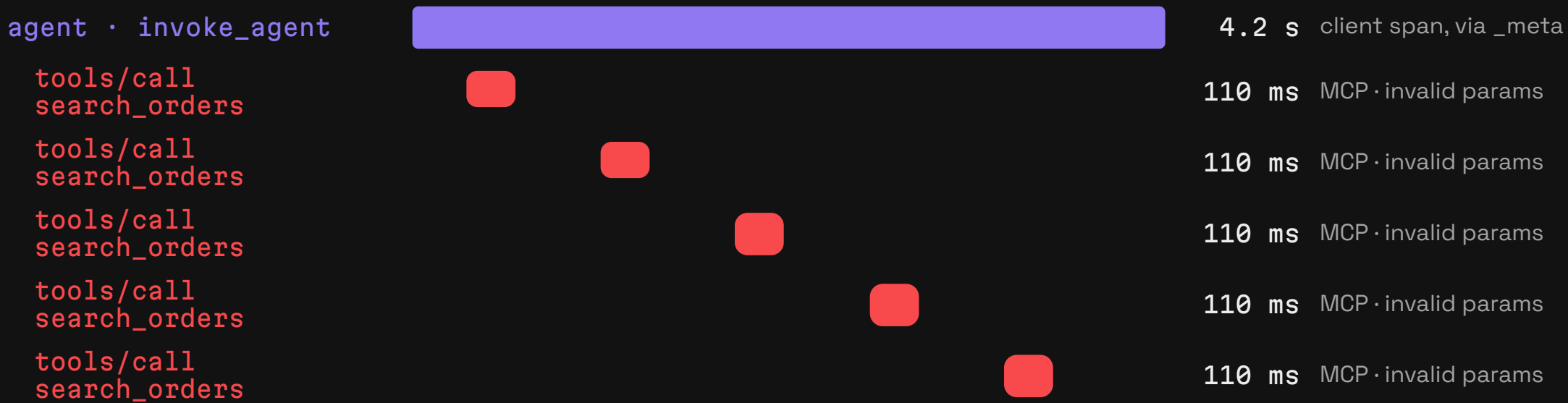
The request succeeded. The tool didn’t.



HTTP said 200. The MCP span says the tool failed, and its child span says why.

TRACING · DEBUGGING SCENARIO

One agent. The same failing call, five times.



MCP SPECIFICATION · SEP-414

Trace context arrives at two layers

HTTP HEADERS

```
POST /mcp
traceparent: 00-4bf92f35…-01
Mcp-Method: tools/call
Mcp-Name: search_orders
```

Transport tracing and operation visibility

MCP REQUEST METADATA

```
"_meta": {
  "traceparent": "00-4bf92f35…-01",
  "tracestate": "…",
  "baggage": "…"
}
```

MCP operation context, including over stdio

TRACING

Decide which trace context to trust

- Trusted trace context
  - Continue the incoming trace or link to it
- Untrusted boundary
  - Start a locally rooter trace
- No valid incoming context
  - Start a locally rooted trace

PRODUCTION

Centralize telemetry policy in the Collector

```
MCP servers metrics · logs · traces → OpenTelemetry Collector → your backend
```

- **Protect:** filter and redact sensitive telemetry, as a second line of defense
- **Control volume:** filter noise, deduplicate logs, sample traces
- **Enrich:** attach infrastructure and deployment context
- **Route:** export to one or more compatible backends

Implement it once, not in every MCP server.

SDK SUPPORT

You don’t have to build all of this yourself

Example: TypeScript SDK v2, typescript-sdk#3115. Merged, not yet released.

```
tracer provider → MCP spans. meter provider → MCP metrics. neither → nothing changes.
```

YOUR SDK CAN HANDLE

Spans for every MCP request

The mcp.server.operation.duration metric

Trace context in _meta

Safe defaults: no tool arguments or results

STILL YOUR JOB

Set up OpenTelemetry and export the data

Instrument your databases and APIs

Write structured logs

Check it records what you expect

What does a healthy MCP server look like?

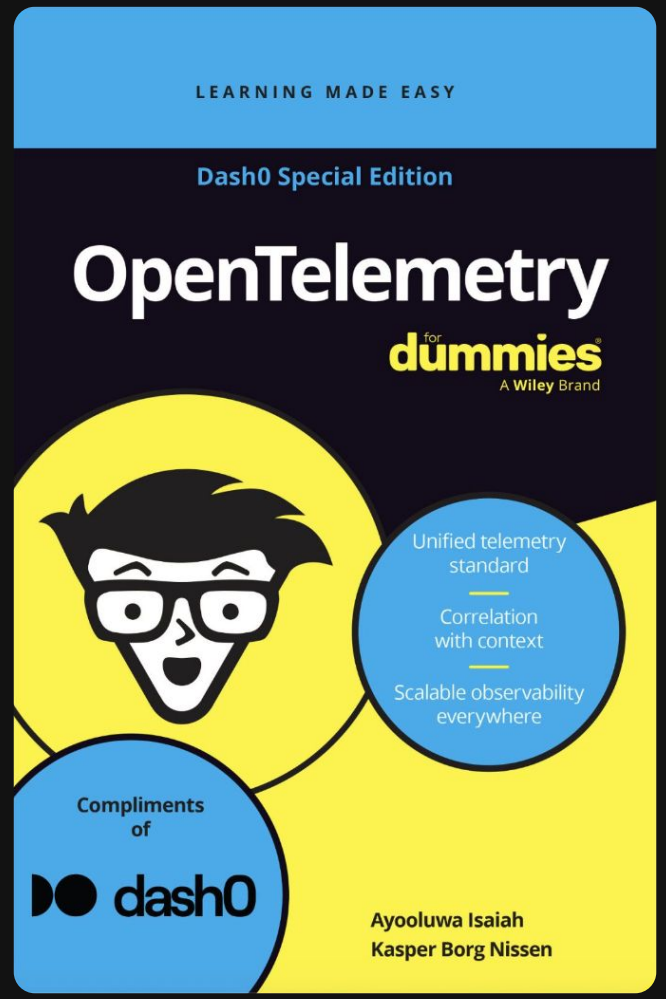
01 **You know when a tool is struggling**
Measure operation latency, errors and traffic.

02 **You know what happened**
Record structured, meaningful, safe operational events.

03 **You know where the problem lies**
Trace the operation and the work behind it.

Dash0 Special Edition: OpenTelemetry For Dummies (Free eBook)

OpenTelemetry's core components, practical setup, and how to adopt an open observability standard with confidence.



MCP Dev Summit Nairobi

Thank you!

Questions?



dash0.com/card/ayooluwa-isaiah

Ayooluwa Isaiah
freshman.tech · /in/ayisaiah · @ayisaiah

MCP Dev Summit Nairobi