Correlation Over Collection: A Layered Observability Framework for SREs

Designing telemetry for faster incident investigation

Khushboo Nigam

Cloud Architect & Observability Practitioner

We Don’t Have a Data Problem
We Have a Clarity Problem

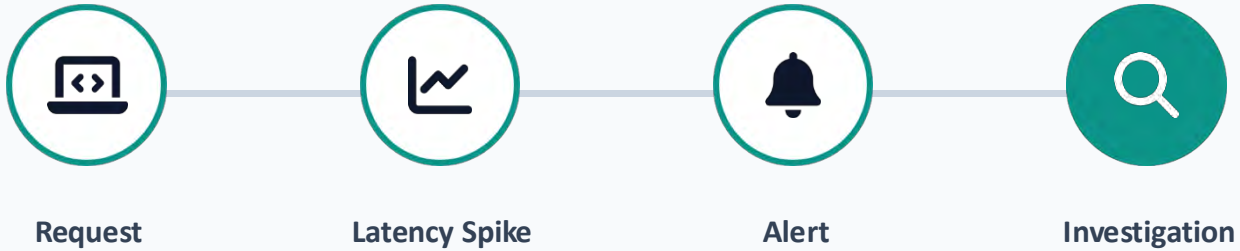
- Modern systems generate massive telemetry telemetry
- Alerts, metrics, logs, traces compete for attention
- Engineers must manually stitch signals together during incidents

The Investigation Gap

The First Question During an Incident

What changed?

- Many signals show symptoms
- Few signals explain cause



Observability Enables “Why”, Not Just “What”

Monitoring

The Known Knowns

- Detect known failures
- Threshold alerts

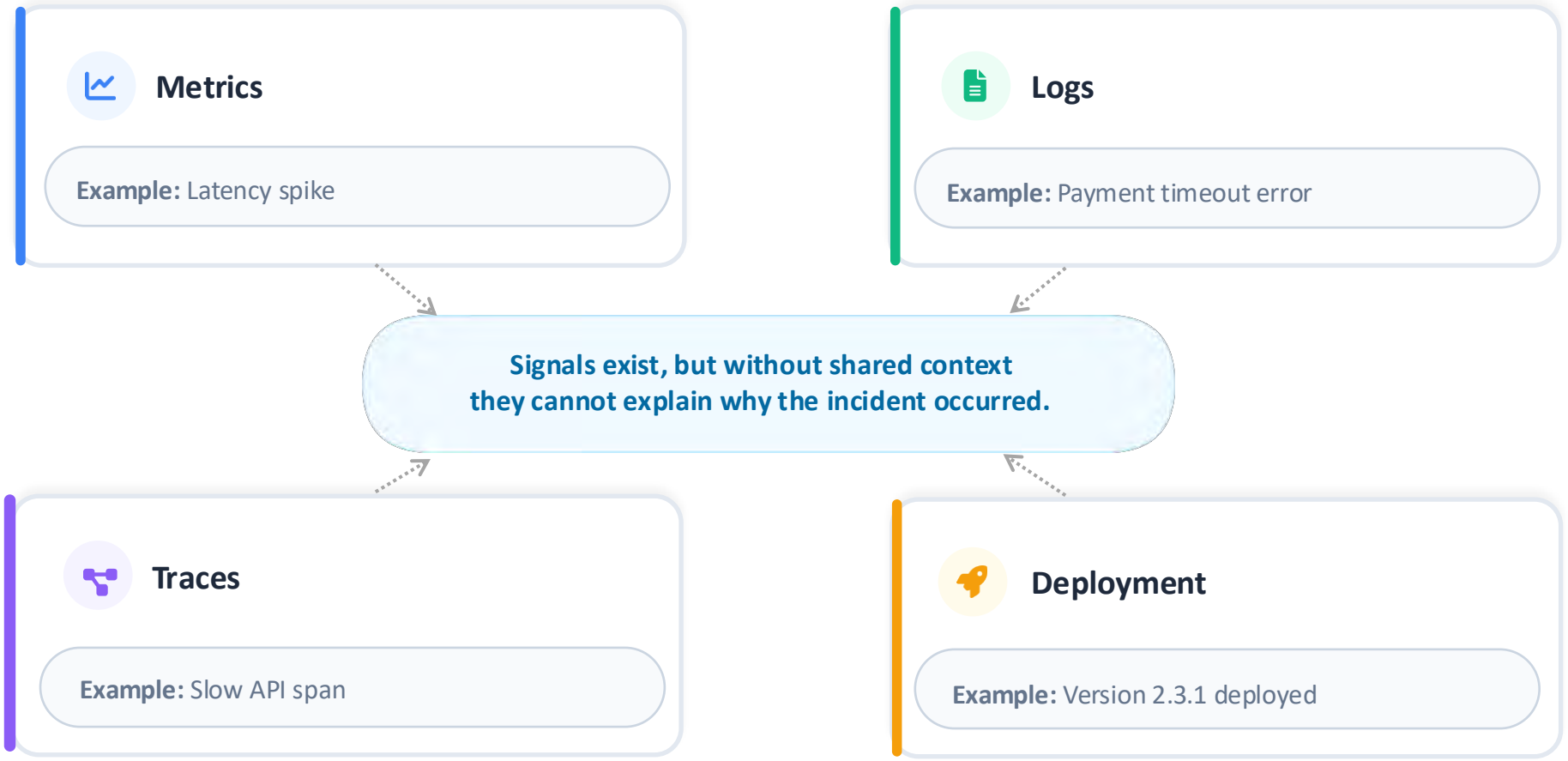
VS

Observability

The Unknown Unknowns

- Investigate unknown failures
- Explain system behavior

Telemetry Without Context



"Telemetry should be designed for correlation, not just collection."

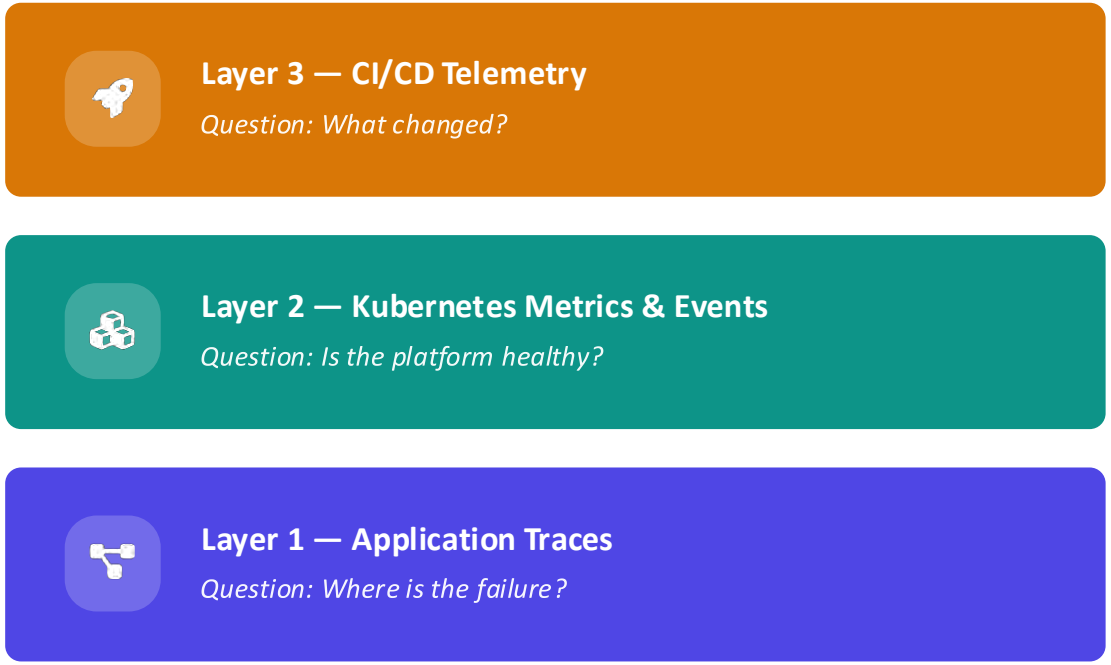
A Layered Observability Model

Instead of collecting more telemetry, organize signals by the question each signal helps answer.

Layer 3 — CI/CD Telemetry
Question: "What changed?"

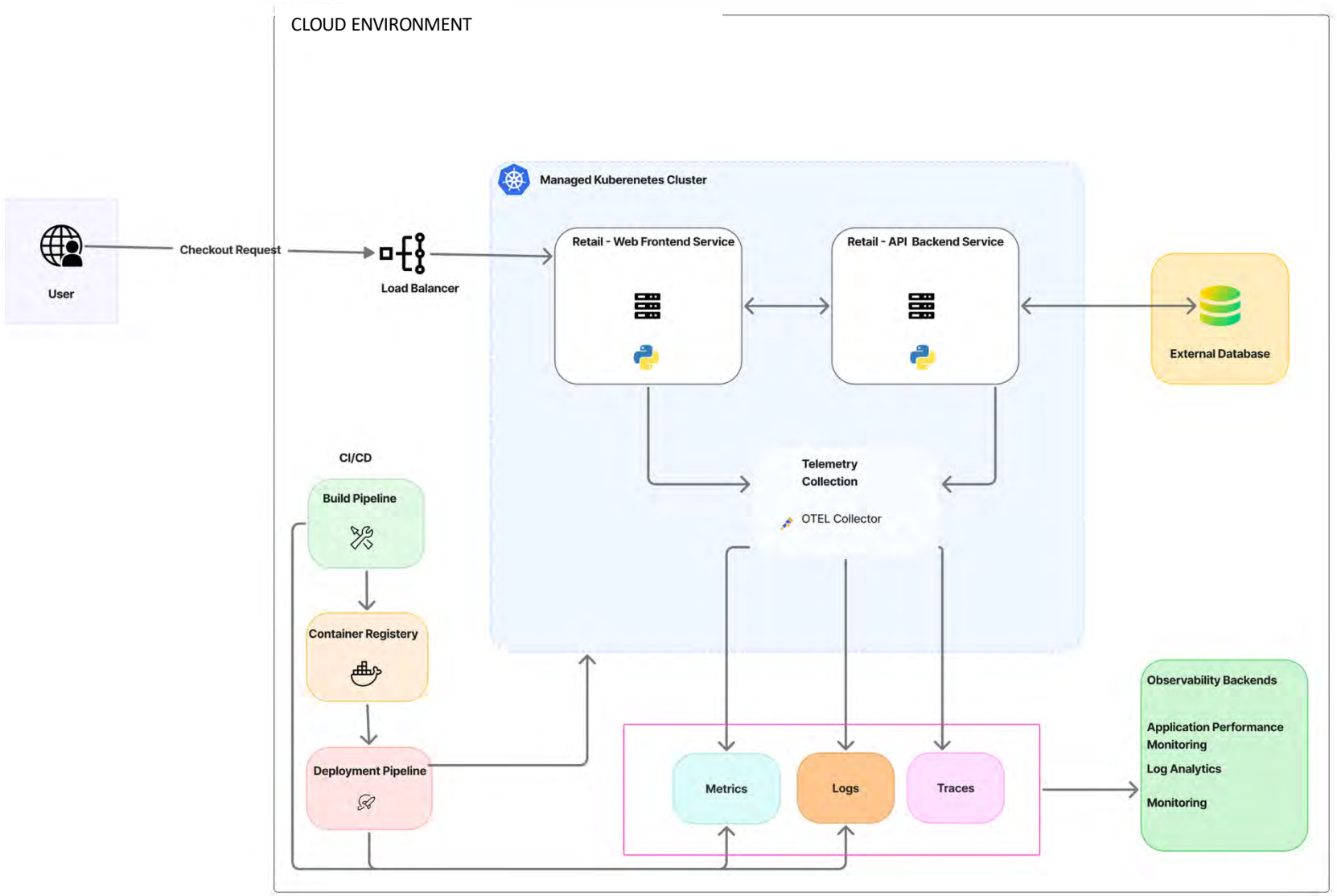
Layer 2 — Kubernetes Signals
Question: "Is the platform healthy?"

Layer 1 — Application Traces
Question: "Where is the failure?"



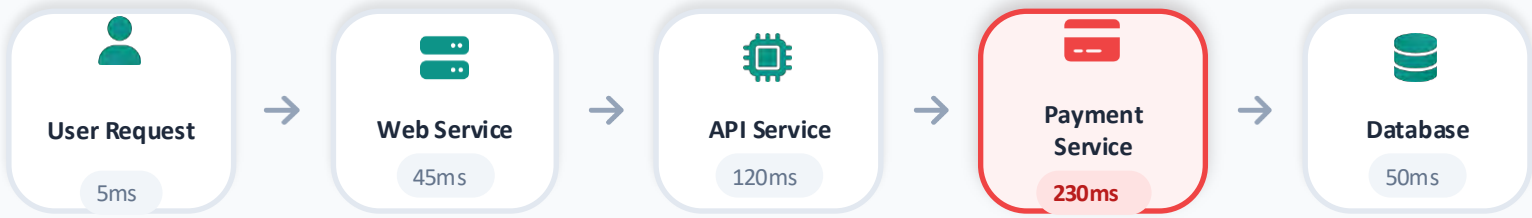
Retail Microservices Application

Cloud-native application architecture running on Kubernetes



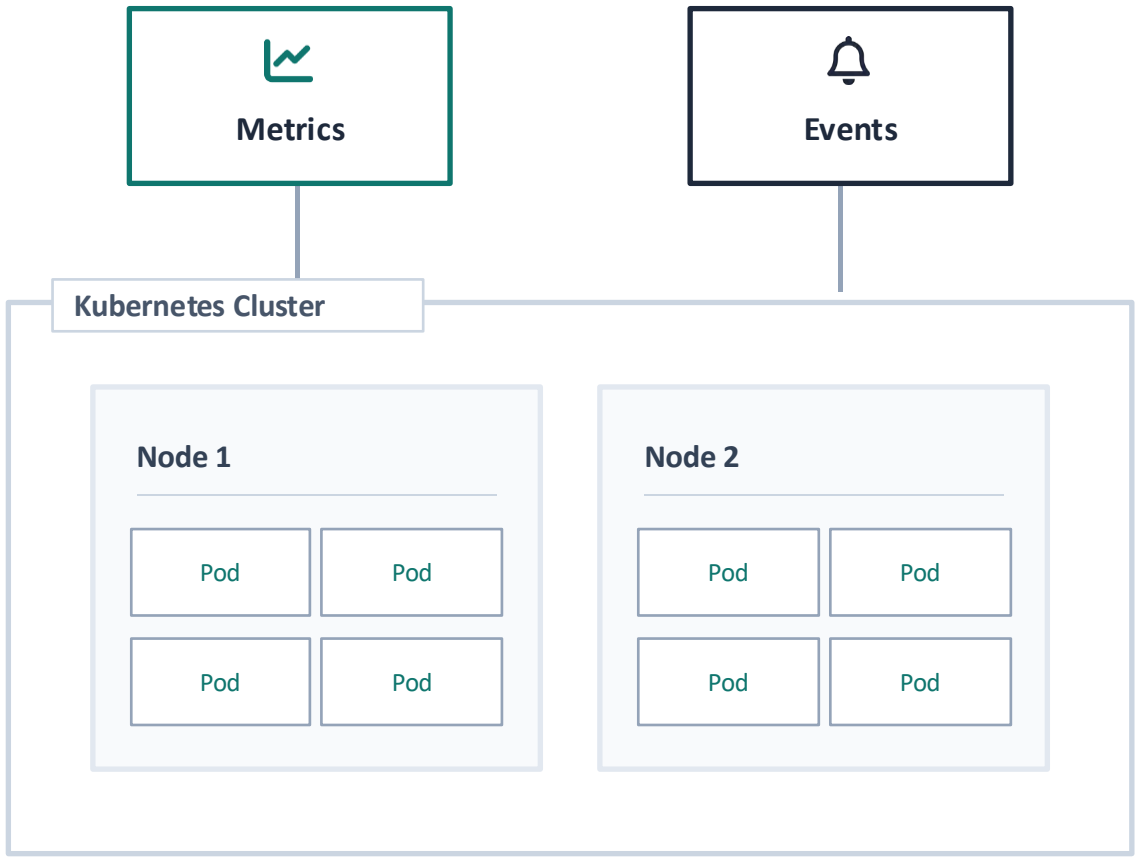
Layer 1 — Application Traces

- Trace full request path across services
- Show latency and dependencies dependencies
- Identify failing span in the transaction



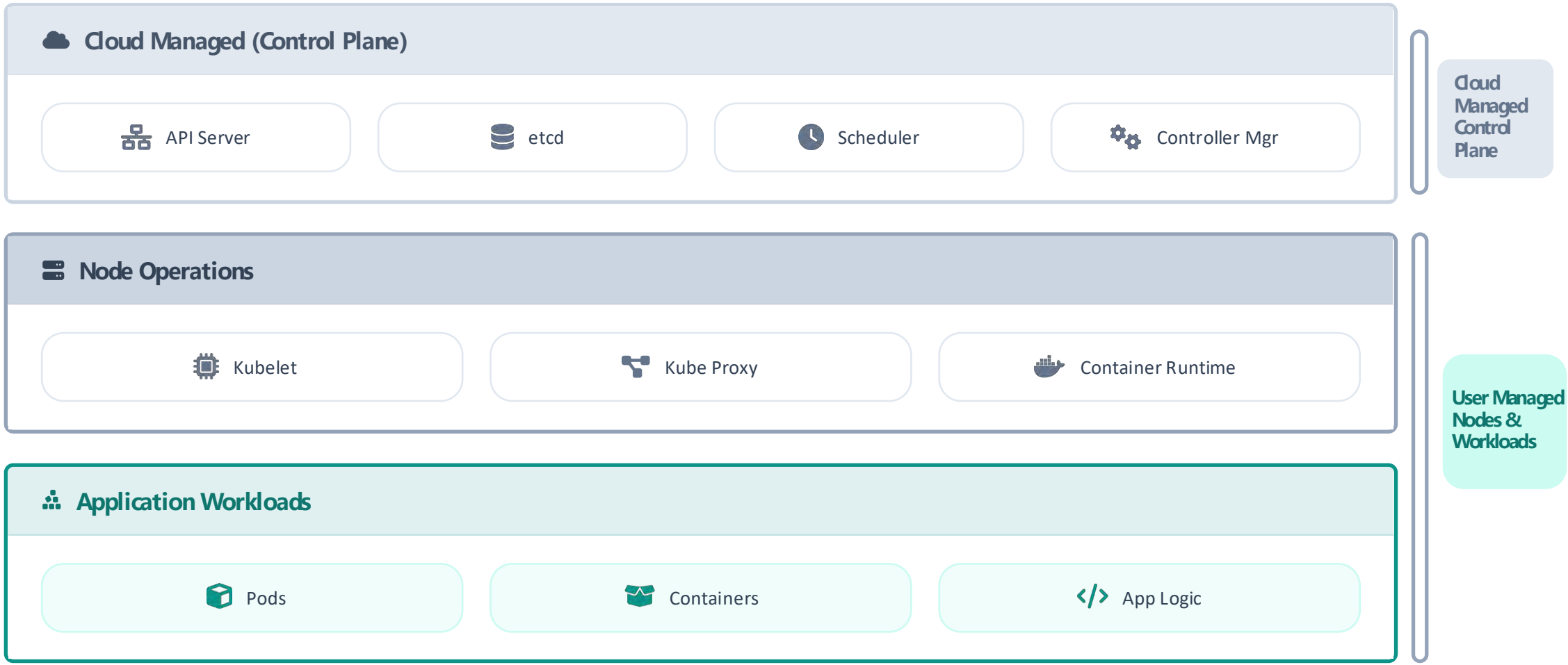
Layer 2 — Kubernetes Metrics and Events

- Node resource pressure
- Pod lifecycle events (CrashLoopBackOff, OOMKilled)
- Scheduling and orchestration signals



Shared Responsibility in Managed Kubernetes

Understanding observability boundaries between cloud provider and user

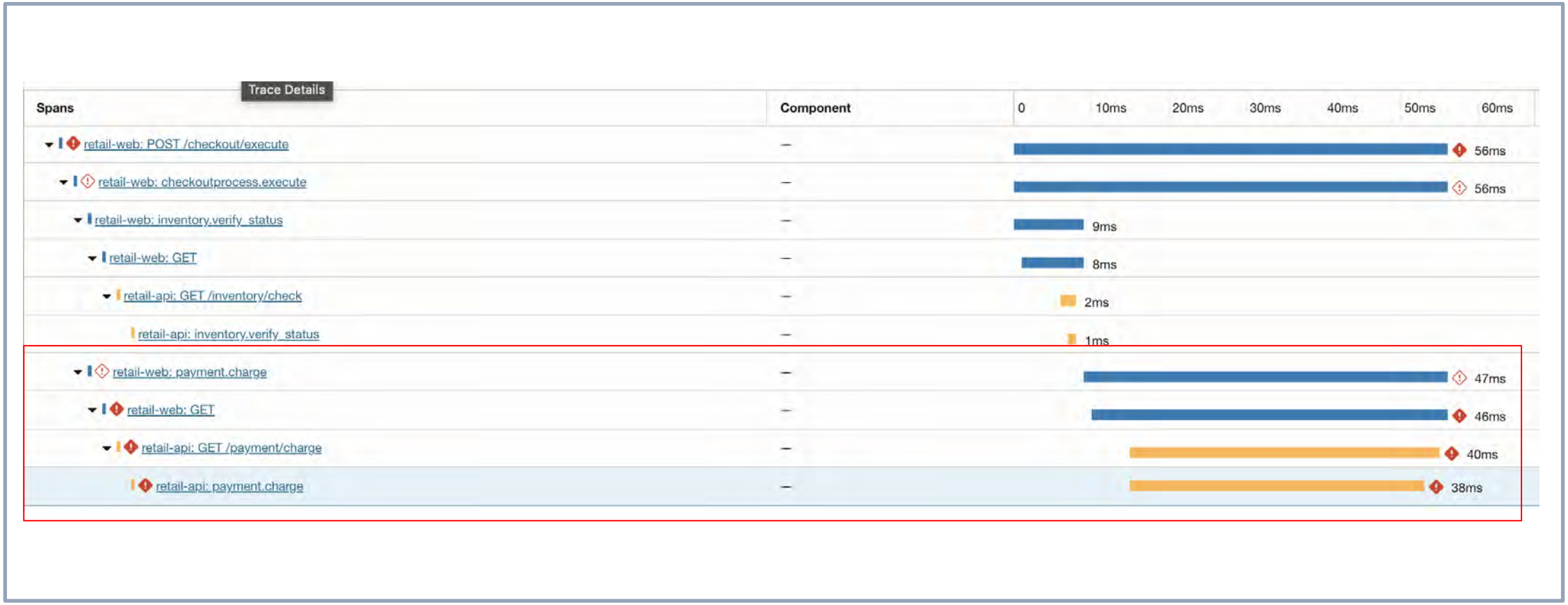


Layer 3 — CI/CD Telemetry

- Deployment events
- Version updates
- Pipeline execution results

Purpose:
Provides change context during incident analysis.

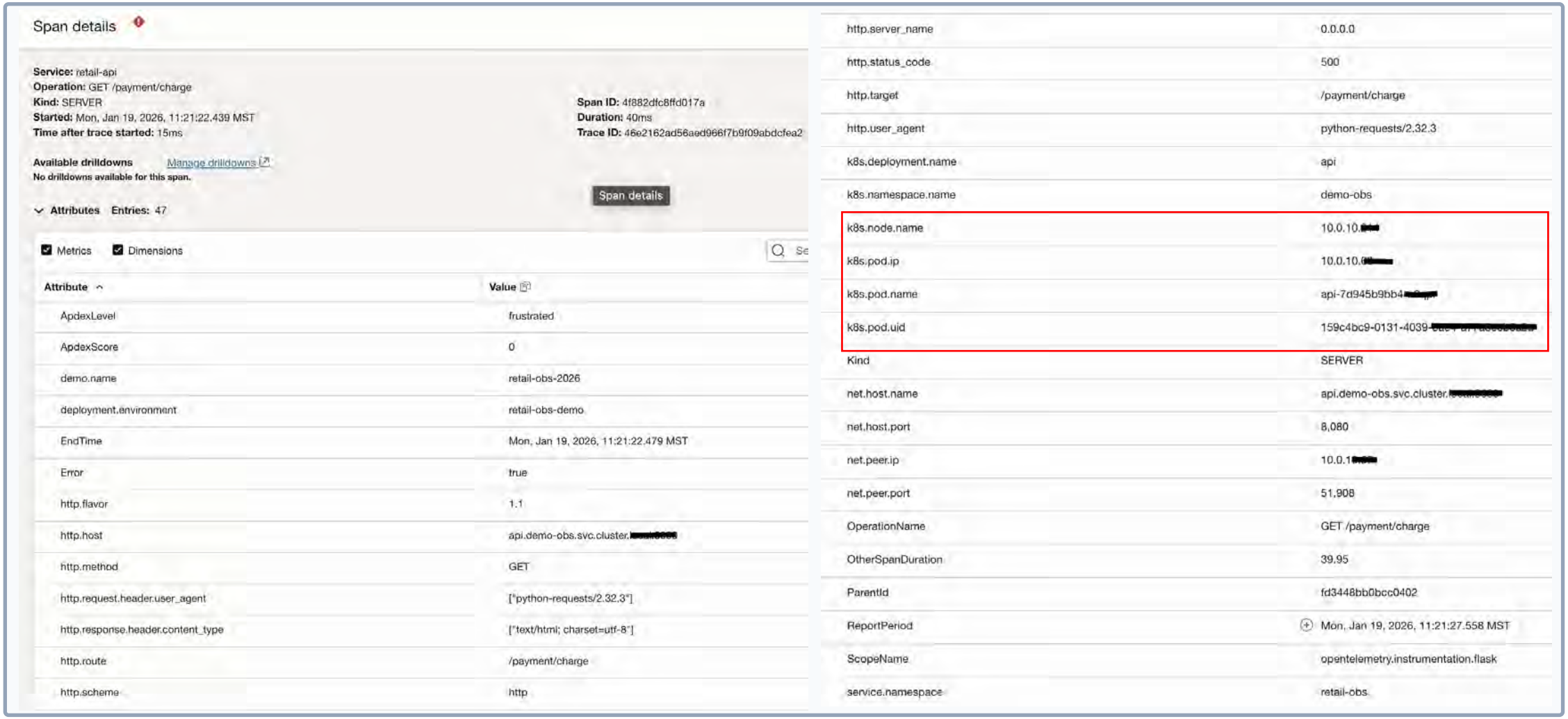
Identifying the Failure Within a Transaction



Example trace highlighting the failing service span.

Traces help pinpoint where in the transaction the error originates.

Separating Platform Issues from Application Faults



Example span details highlighting K8s information.

Kubernetes metadata adds operational context to application telemetry.

Correlating Incidents with Recent Changes

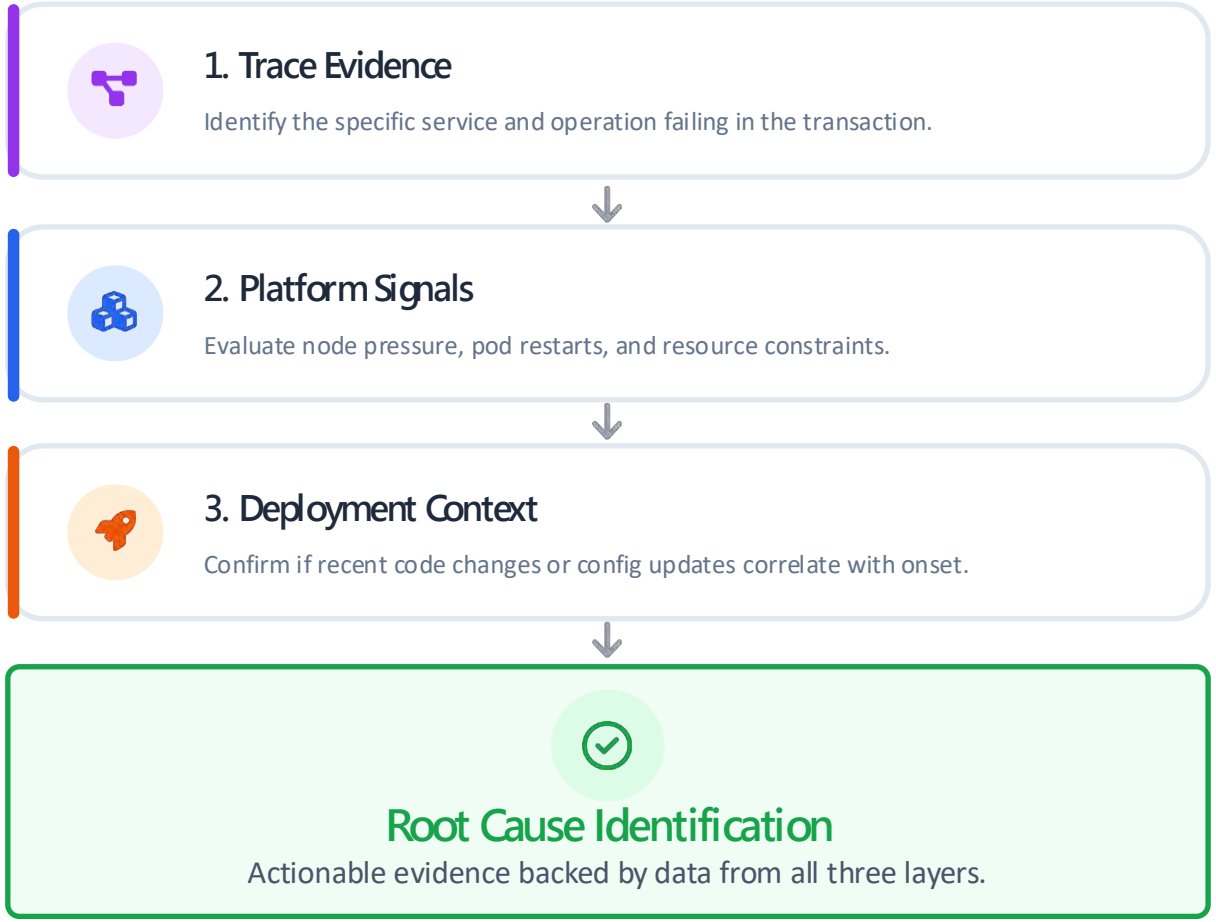
service.version	local-20251226041342
ServiceName	retail-api
SpanDuration	40ms
SpanErrorCount	1
SpanId	4f882dfc8ffd017a
StartTime	Mon, Jan 19, 2026, 11:21:22.439 MST
StatusCode	Error
StatusMessage	IntegrityError: UNIQUE constraint failed: payments.id
telemetry.route	collector
telemetry.sdk.language	python
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.28.0
TraceId	46e2162ad56aed966f7b9f09abdcfea2

Example CI/CD logs cluster and service version added to spans.
spans.

Deployment metadata often explains sudden behavior changes.

Turning Telemetry into Evidence

A structured hierarchy builds confidence in the root cause analysis.



Summary: The Path to Clarity

Problem

Modern systems produce enormous telemetry, yet during incidents engineers still struggle to **understand what is happening.**

Approach

Use application traces, Kubernetes platform signals, and CI/CD telemetry together—**enriched with contextual metadata**—so signals can be correlated across layers.

Outcome

When telemetry is enriched and and correlated, engineers can move move from scattered signals to **clearer investigation paths** and and more confident incident analysis.

"Observability becomes most powerful when telemetry is structured, enriched, and correlated."