KubeCon + CloudNativeCon North America 2026 · Salt Lake City

The SRE Is Not Dead

What AI taught us about missing context in Kubernetes incident response

Muhammed Yunas Chirayath
Platform / SRE
“I run the platform.”

Ajmal Ali Kannu
Application
“Ajmal builds the application.”

The views and examples in this talk are the speakers’ own and are illustrative. They do not describe any employer’s systems, incidents or results.

Will AI replace SREs?

For two years, everyone has said AI will replace SREs.

Let’s examine that claim against the incidents SREs actually face.

This is not an anti-AI talk. The scenarios are representative, built to show how an AI-assisted investigation works and where it breaks.

01
We gave AI great telemetry

02
It still made bad calls

03
Here’s how we engineer the context



Two seats. One incident.

Same incident, two vantage points.

Ajmal Ali Kannu · the application view
Services, releases, business logic.

Muhammed Yunas Chirayath · the platform view
Clusters, capacity, blast radius.

That difference turned out to be the key.

What we expected from AI

01
Faster root cause analysis

02
Fewer manual investigations

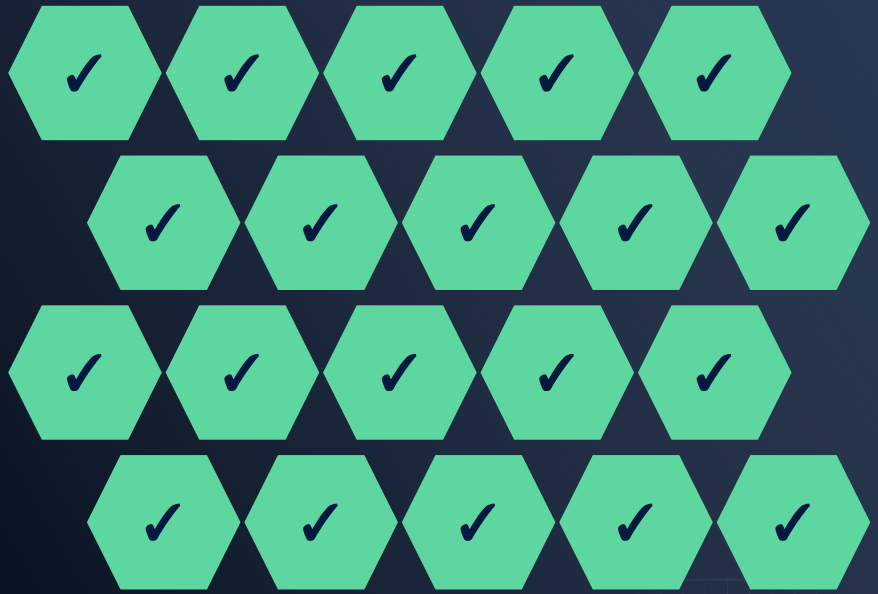
03
Less alert fatigue

04
Lower MTTR

"AI is incredibly good at pattern recognition. Humans are still much better at organizational reasoning."

The pitch was speed. What we needed was context.

Green isn't gold



EVERY PANEL. EVERY SERVICE. GREEN.

The dashboards looked healthy.
The customer experience was not.

An AI reads the same dashboards a human does.

EXAMPLE: A DATA PLATFORM

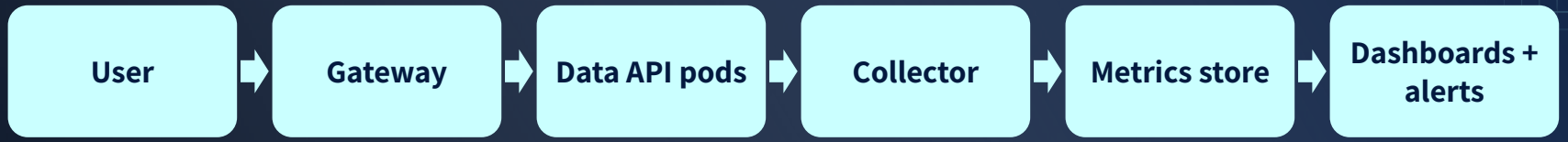
Pods running. CPU and memory normal. Infrastructure alerts quiet. Reports still late.

Healthy infrastructure is not end-to-end reliability.

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

Healthy platform, unhappy users

Illustrative data-serving platform on Kubernetes.



WHAT THE DASHBOARDS WATCHED

- Pods running
- CPU and memory normal
- Error counts quiet
- Infrastructure alerts quiet

WHAT NOBODY WATCHED

- Query latency
- Freshness of upstream data
- Dependency performance
- The customer-facing SLO

Every box was healthy. The user still waited.

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

Replay 01: The CPU spike that wasn't a mystery

01

14:02 Deploy

Recorded by the deployment system.

02

20-min gap

No one told the AI.

03

14:22 Spike

CPU climbs. The alert fires.

04

AI alert

Pages on-call: "Investigate CPU."

"The most valuable metric during the incident wasn't CPU usage. It was knowing a deployment happened 20 minutes ago."

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

Replay 01 under the hood: the investigation

Illustrative architecture and evidence, not a claim about a measured production incident.

1 DETECT

CPU up on checkout-api at 14:22.

Alert carries service, cluster, namespace, time window.

2 RETRIEVE

Metrics, logs, traces

Deploy events near 14:02

Owner and dependencies

SLO and customer impact

3 TEST

A: the 14:02 deploy is related

B: upstream or traffic

Which evidence supports each? What is missing?

4 DECIDE

Evidence-linked summary

Stated confidence limits

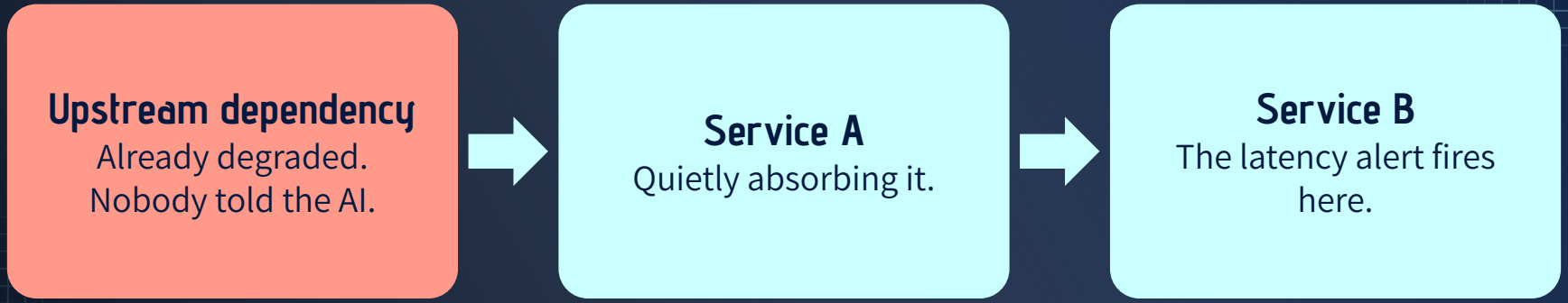
Next diagnostic step

Engineer validates before acting

Correlation tells you where to look. Evidence tells you what to believe.

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

Replay 02: The latency nobody owned

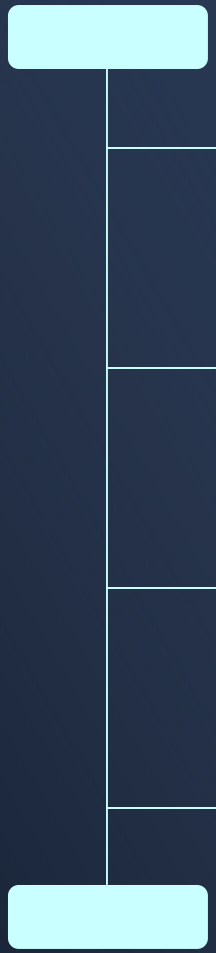


The AI correctly flagged latency on Service B. It had no dependency health, so the symptom looked like a B problem.

"The AI wasn't wrong. It simply didn't know enough."

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

Replay 03: The page that rang the wrong desk



Alert fires

Error rate is high and pods keep restarting.

AI summary

A factual summary built from signals and events.

Stale data

The owner changed last sprint. The on-call rotation was never updated.

Wrong desk

The page lands on a team that can’t act. The right team is two Slack channels away.

“Knowing who owns a service is just as important as knowing it is unhealthy.”

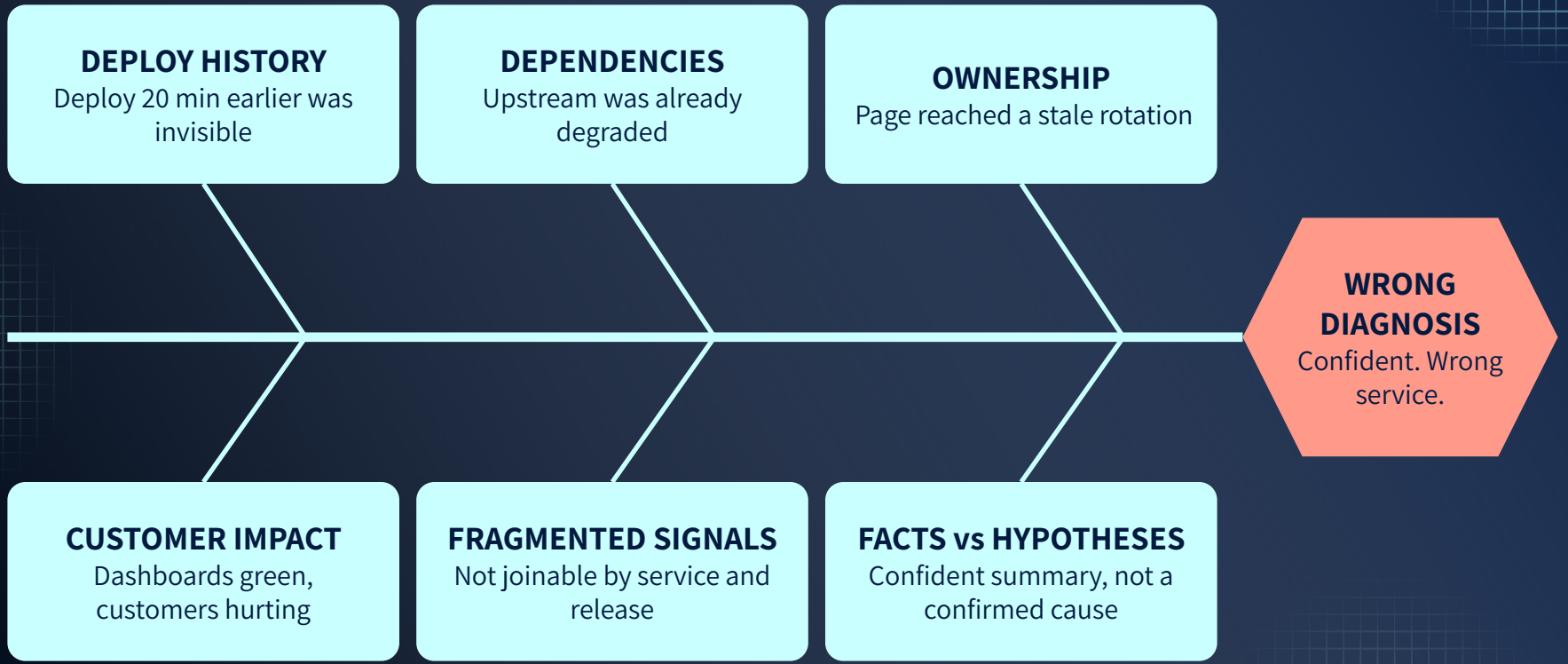
Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

A moment of honesty

“An AI can generate a beautiful summary of the wrong problem.”

Confident. Well-written. Wrong service.

Why the AI missed it



The same alert, two investigations

Illustrative example, not a measured incident.

Without context

SEES
CPU above 80% on checkout-api.

SAYS
“Investigate CPU.” Plausible, confident, unverified.

With context

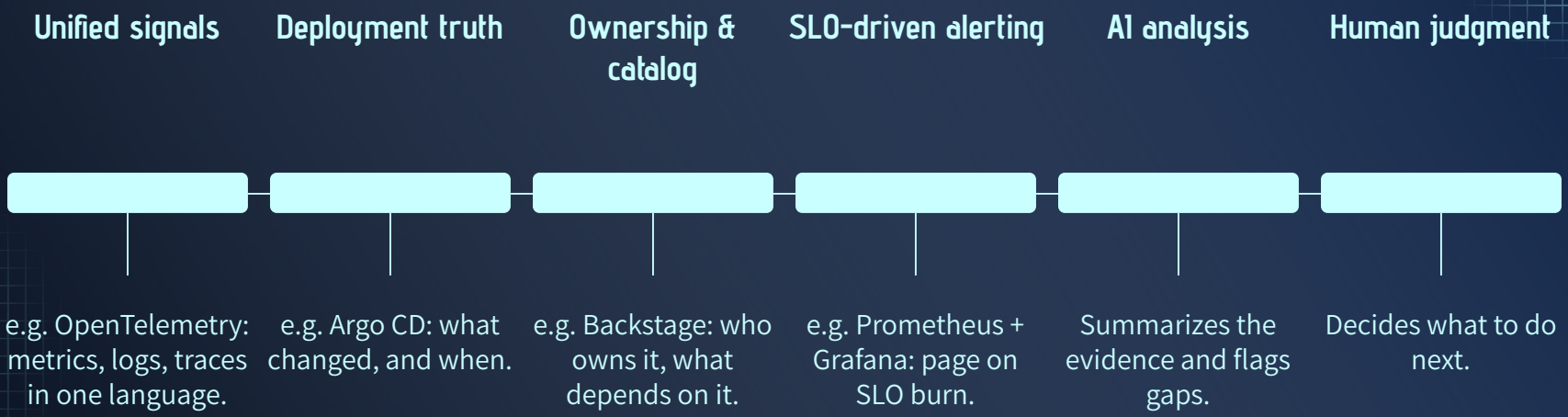
SEES
The CPU alert, the 14:02 deploy, the owner, upstream health, SLO burn.

SAYS
“The deploy is a candidate. Upstream looks healthy. Not confirmed. Next: compare pod CPU before and after rollout.”

Correlation shows where to look. It is not the root cause.

Illustrative scenarios based on common engineering challenges. Not specific incidents or systems of any employer.

The operational foundation



Before asking for a smarter AI, fix the foundation it depends on.

Illustrative design. Product names are examples of interchangeable implementations, not any employer's architecture.

How the agent gets its evidence

Telemetry
e.g. OpenTelemetry, Prometheus

Deployments
e.g. Argo CD

Ownership, dependencies
e.g. Backstage

SLOs and customer impact
e.g. Prometheus, Grafana

Tool gateway

Design requirements:

- Authorized, read-only calls
- Scoped identity for the agent
- Every query audited
- Facts returned with source and timestamp

Agent

- Retrieves evidence, never leans on model memory
- Cites the evidence behind each hypothesis
- Keeps facts and hypotheses apart

Join keys: service · namespace · cluster · release SHA · time window

Illustrative design. Product names are examples of interchangeable implementations, not any employer's architecture.

Failure modes and decision boundaries

Deploy record missing	Say so. Show source and timestamp. Never infer it didn’t happen.
Ownership is stale	Flag it stale, show its timestamp, use the fallback path.
Names or timestamps differ	Join on release SHA and time window. Report unmatched items.
Evidence is insufficient	Return the gap and the next diagnostic step. No guess.

READ
Automatic, within scope

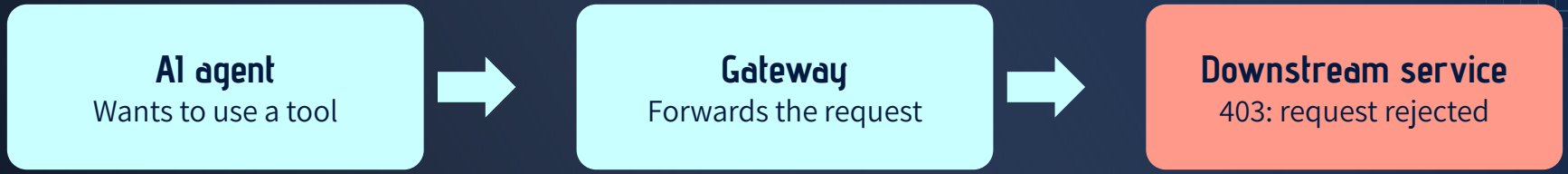
SUGGEST
With evidence and confidence

ACT
Engineer approves

Design principles, not a description of a deployed system. Illustrative scenarios, not any employer’s incidents.

Context also means identity

Illustrative scenario at the design-principle level. Not a specific incident or system.



WHAT THE AI SEES

A failed request, a 401 or 403, and the gateway and application logs.

WHAT IT NEEDS TO KNOW

Which identity was presented. Which service the token was meant for. What authorization decision was made. Where the request failed.

More logs won't fix a missing identity chain. The model can't reason its way around missing evidence or missing permissions.

Design principles, not a description of a deployed system. Illustrative scenarios, not any employer's incidents.

How we'd evaluate it

Proposed metrics, not results. Set the baseline first.

Time to first correct hypothesis	Per incident, against a pre-AI baseline
Hypothesis accuracy	Scored in post-incident review
Evidence coverage	Share of claims linked to a source
Page quality	Pages that needed a human, per on-call shift
MTTR	Against baseline, at the same severity mix

The future isn't AI versus SREs.

It's AI with SREs.

AI helps analyze evidence.

Engineers supply context and judgment.

Reliability requires both.

What to do on Monday

01

Put deploys on the incident timeline

Pull events from your deployment system next to the alert.

02

Page on SLO burn, not CPU

Keep CPU thresholds as diagnostics.

03

Keep ownership data fresh

Owner, escalation path, dependencies, runbooks.

04

Make AI show its evidence

Facts from authoritative tools, cited, inside existing access and approval controls.

If you remember one slide

01 Telemetry tells you WHAT happened.

02 Context tells you WHY.

03 Judgment decides WHAT TO DO NEXT.

Thank you!

Let’s talk about the context your platform is missing.

Muhammed Yunas Chirayath
Platform / SRE
linkedin.com/in/muhammedyunas

Ajmal Ali Kannu
Application
linkedin.com/in/ (add link)