Decoding K8s Incidents

How to make logs work for you in real-world troubleshooting

Conf42
Oct 19, 2024

Today's Roadmap

- The trouble with logs in isolation
- What does "context" look like?
- A tale of investigation
- Pitfalls
- How to get started

SENSER

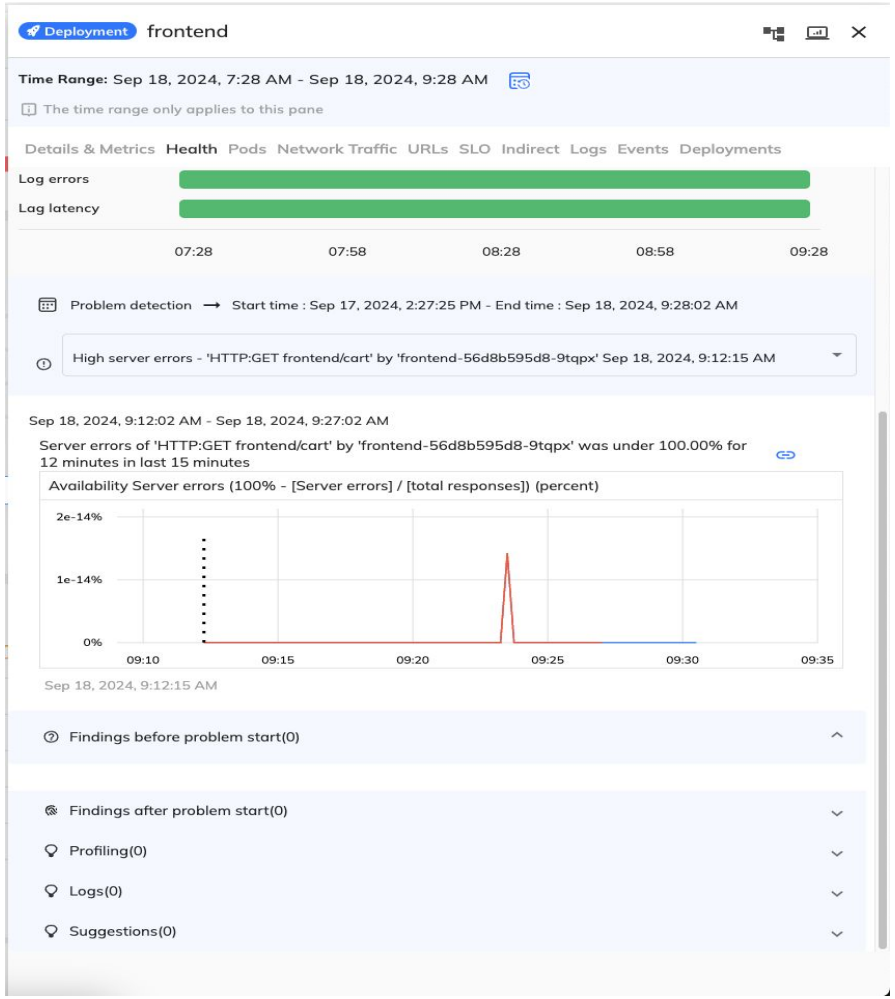
You already know Kubernetes logs are helpful for troubleshooting (but not ideal on their own)

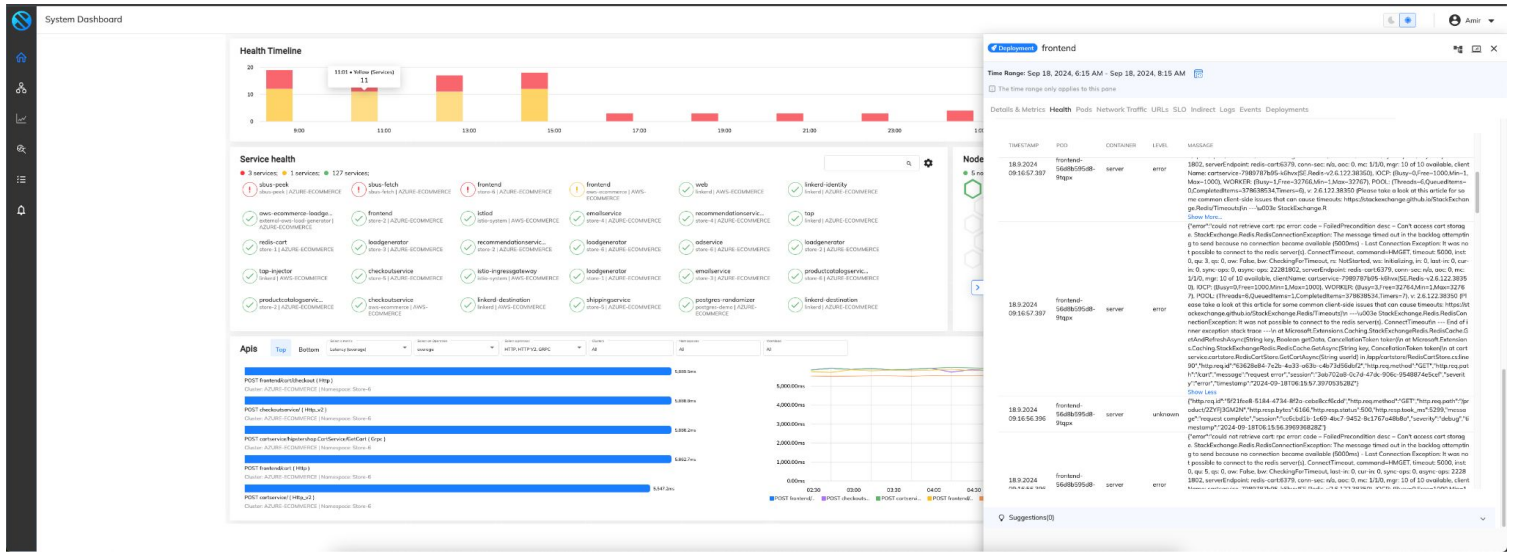
Individual logs lack context into:

- A human wrote them, if they exist they are based on experience and assumptions
- Other issues occurring around the same time
- Similar prior incidents
- Related components across your topology

"The problem with logs is that they require more logs, and rarely from a single place"

My goal: inspire you with the “art of the possible” (taken from a real-world K8s example)





One Kubernetes log. Many troubleshooting paths.

Primary metrics impacted:

- MTTR
- Total investigation time
- SRE sanity

SENSER

First: what do I mean by “context” ?

Topology

High-level view of how the environment is structured, showing how different nodes, services, and infrastructure components are connected

Metrics

Performance indicators, such as resource usage (CPU, memory), network latency, or error rates, which give insight into the health of the system

Deployment history

Information on recently deployments (since newly deployed versions of services or applications can introduce issues)

Why is it hard to get this kind of context?

Two basic strategies – both with serious shortcomings.

	Commercial observability tech	Open-source observability stack
Sample technology	Dynatrace, Datadog, New Relic	Open Telemetry, Prometheus, Grafana
What it involves	Deploying observability agents/ modules for system coverage	Manually instrumenting your production environment
Common challenges	• Cost $$$ ○ Or more specifically cost for coverage • Configuring and maintaining dashboards	• Coverage gaps, blind spots • Instrumentation overhead • Production impact

(Spoiler alert: eBPF + automated topology can help)

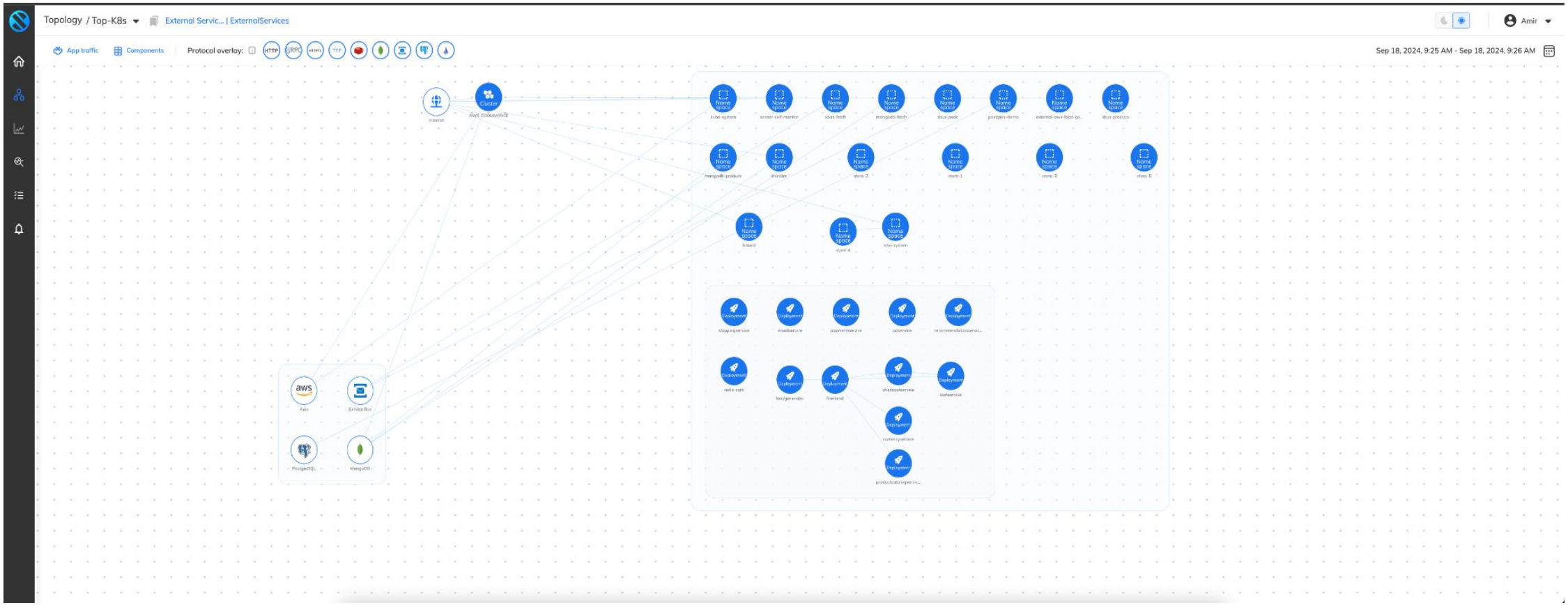
More on this later in the talk.

A tale of investigation

The log

18.9.2024 09:16:57.397	frontend-56d8b595d8-9tqpx	server	error	{"error":"could not retrieve cart: rpc error: code = FailedPrecondition desc = Can't access cart storage. StackExchange.Redis.RedisConnectionException: The message timed out in the backlog attempting to send because no connection became available (5000ms) - Last Connection Exception: It was not possible to connect to the redis server(s). ConnectTimeout, command=HMGET, timeout: 5000, inst: 0, qu: 3, qs: 0, aw: False, bw: CheckingForTimeout, rs: NotStarted, ws: Initializing, in: 0, last-in: 0, cur-in: 0, sync-ops: 0, async-ops: 22281802, serverEndpoint: redis-cart:6379, conn-sec: n/a, aoc: 0, mc: 1/1/0, mgr: 10 of 10 available, clientName: cartservice-7989787b95-k6hvx(SE.Redis-v2.6.122.38350), IOCP: (Busy=0,Free=1000,Min=1,Max=1000), WORKER: (Busy=3,Free=32764,Min=1,Max=32767), POOL: (Threads=6,QueuedItems=1,CompletedItems=378638534,Timers=7), v: 2.6.122.38350 (Please take a look at this article for some common client-side issues that can cause timeouts: https://stackexchange.github.io/StackExchange.Redis/Timeouts)\n ---\u003e StackExchange.Redis.RedisConnectionException: It was not possible to connect to the redis server(s). ConnectTimeout\n --- End of inner exception stack trace ---\n at Microsoft.Extensions.Caching.StackExchangeRedis.RedisCache.GetAndRefreshAsync(String key, Boolean getData, CancellationToken token)\n at Microsoft.Extensions.Caching.StackExchangeRedis.RedisCache.GetAsync(String key, CancellationToken token)\n at cartservice.cartstore.RedisCartStore.GetCartAsync(String userId) in /app/cartstore/RedisCartStore.cs:line 90","http.req.id":"63628e84-7e2b-4a33-a63b-c4b73d56dbf2","http.req.method":"GET","http.req.path":"/cart","message":"request error","session":"3ab702a8-0c7d-47dc-906c-9548874e5cef","severity":"error","timestamp":"2024-09-18T06:15:57.397053528Z"} Show Less

A tale of investigation



The **actual** underlying issue...

Lost database connection

Relying on the log alone

What troubleshooting looks like

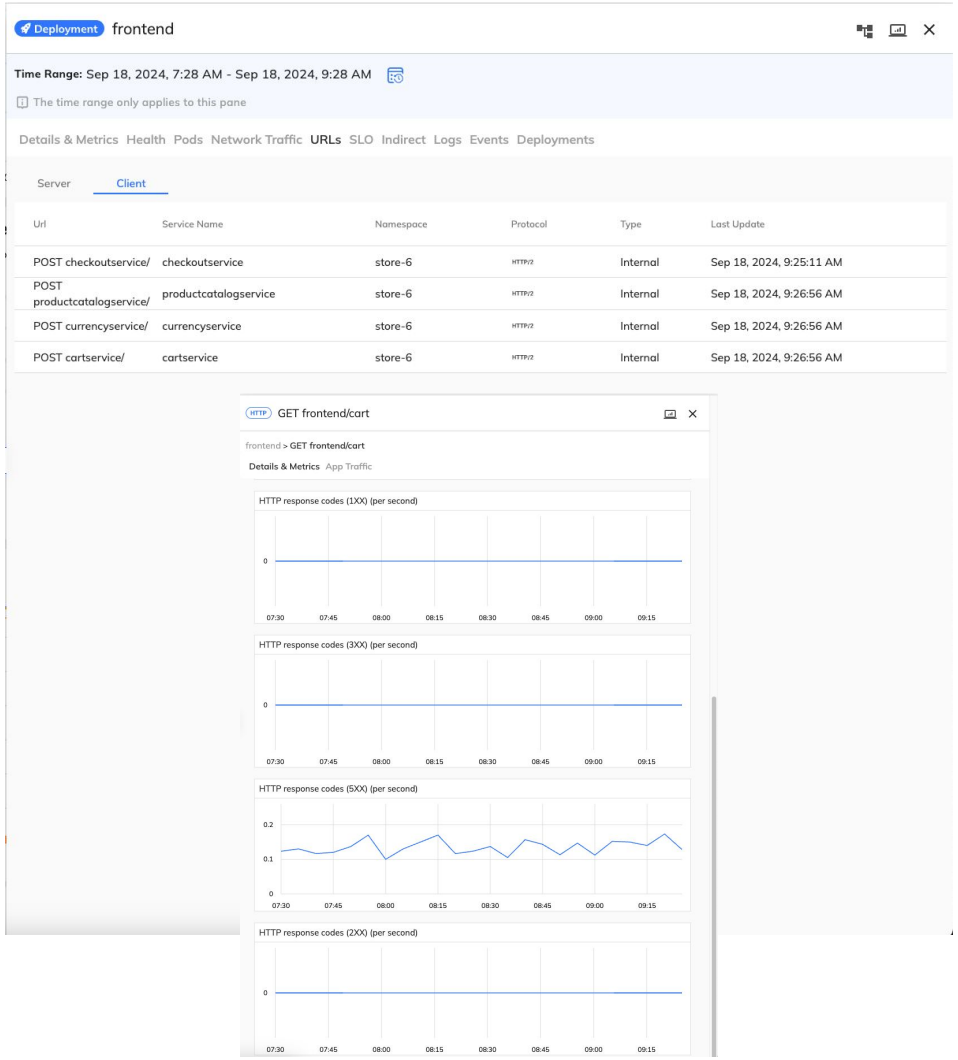
Getting started: Examine the logs of the node that lost the database connection.

Process: Scrub through the logs, searching for potential causes, such as application errors, resource issues, or networking problems.

Challenge: You would need to investigate each layer (network, resource allocation, application), working your way up the stack in a recursive manner.

Outcome: A longer time to troubleshoot because you'd have to check each potential cause manually, eventually triangulating that the issue might not lie in the node itself but in an adjacent node or system.

Log + contextual enrichment



What troubleshooting looks like

Getting started: With enriched data (e.g., system topology and metrics), you would get a holistic view of the environment, allowing you to see how elements interact.

Process: Quickly see whether the issue stems from another node, a network issue, a resource constraint, or an application-level error such as losing credentials.

Outcome: The enriched data helps pinpoint the root cause much faster. You would also use tools like topology to identify which nodes or services are likely to be impacted by the issue, leading to faster resolution.

Let’s see what this looks like ...

Common pitfalls (and how to avoid them)

- **Incomplete instrumentation:** If environments are only partially instrumented, you might have blind spots that make it hard to investigate issues. (Technologies like eBPF can help.)
- **Reliance on tribal knowledge.** Consider a system that codifies deployment history and automates root cause analysis based on previous system issues.
- **Incorrect heuristics.** Without system-wide context, troubleshooting often focuses on the node or service where the issue was observed – don't overlook the possibility that the root cause is in a neighboring node or service.

Tips for getting started

- **Use topology, metrics, and deployments.** Your system architecture, metrics (e.g., CPU, memory, network performance), and deployment history are the keys to efficiently enriching logs with context.
- **Adopt a layered approach:** Start with high-level insights (topology, performance metrics) and then drill down into specific logs or components based on the clues provided by these higher-level views.
- **Avoid context-building pitfalls.** Consider eBPF for complete coverage – and services that automate your topology – to avoid manual instrumentation. AIOps can also help with automated root cause analysis.

SENSER

Thank you!