Telemetry data: the key to effective AI Root Cause Analysis

Nikolay Sivko,
Co-Founder at Coroot

Let’s meet

- SRE background
- Former Head of Infra
- Now Co-founder and CEO @ Coroot
- We build an open-source observability tool that turns telemetry data into actionable insights:

github.com/coroot/coroot (Apache 2.0)

over 7k⭐


AI
AI EVERYWHERE
imgflip.com

AI for Root Cause Analysis

why my app is performing slowly?

From the Grafana “Node Exporter Full” dashboard you shared, the system itself looks mostly healthy — CPU, RAM, disk, and network utilization are all very low. That means the slowdown of your **application** likely isn’t caused by resource exhaustion at the node level.

LLMs strengths and weaknesses

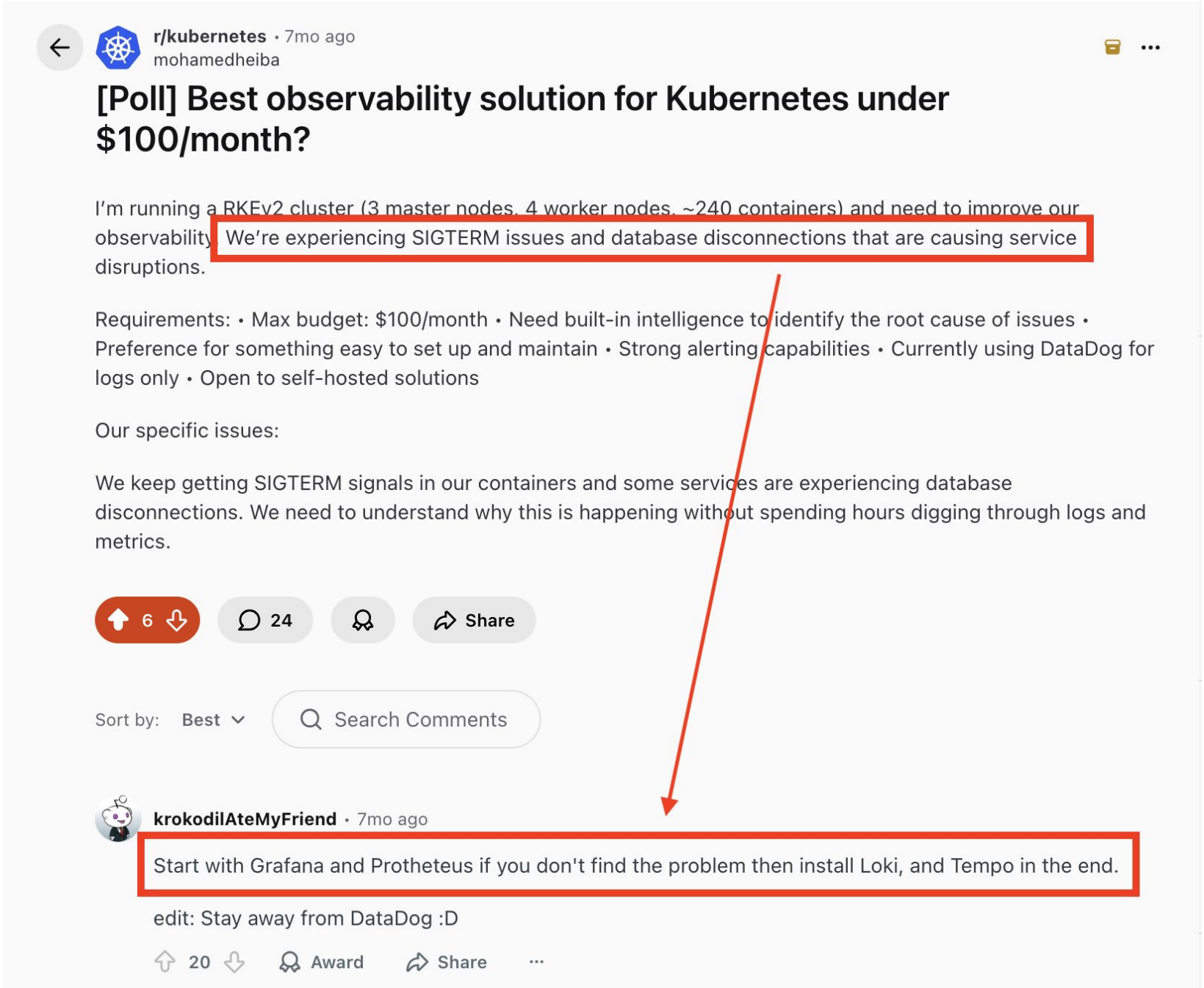
- LLMs have a general “understanding” of how systems work, they can even suggest fixes for issues like Postgres deadlocks.
- But they know nothing about your app unless you provide enough context.
- That context is always limited (as of 2025 — up to ~1M tokens).
- Their responses are non-deterministic - you won’t always get the same answer twice.

The context *AKA* prompt *AKA* input data

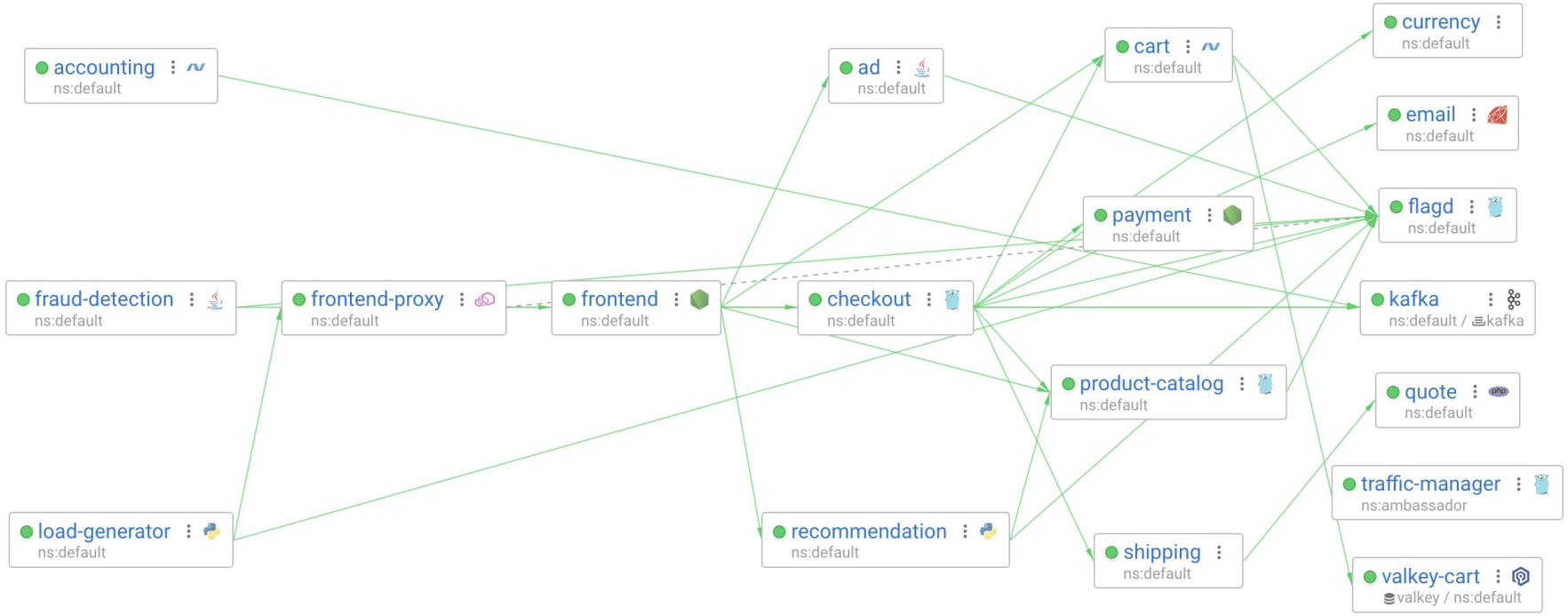
- You can feed LLMs any data - logs, traces, metrics, profiles, events
- In any format: text, JSON, YAML, even screenshots of your dashboards
- But here's the catch: **the answer must already be in your data**

Telemetry data

- Logs, Traces, Metrics, Profiles, Events
- If you have installed a timeseries database it doesn't mean you have enough metrics
- No matter how efficient your log storage, if your app **silently** return HTTP-500
- Traces can pinpoint that the database is performing slowly, but they can't show **WHY**


r/kubernetes • 7mo ago
mohamedheiba
[Poll] Best observability solution for Kubernetes under $100/month?
I'm running a RKEv2 cluster (3 master nodes, 4 worker nodes, ~240 containers) and need to improve our observability. We're experiencing SIGTERM issues and database disconnections that are causing service disruptions.
Requirements: • Max budget: $100/month • Need built-in intelligence to identify the root cause of issues • Preference for something easy to set up and maintain • Strong alerting capabilities • Currently using DataDog for logs only • Open to self-hosted solutions
Our specific issues:
We keep getting SIGTERM signals in our containers and some services are experiencing database disconnections. We need to understand why this is happening without spending hours digging through logs and metrics.
6
24
Share
Sort by: Best
Search Comments
krokodilAteMyFriend • 7mo ago
Start with Grafana and Protheteus if you don't find the problem then install Loki, and Tempo in the end.
edit: Stay away from DataDog :D
20
Award
Share

An example

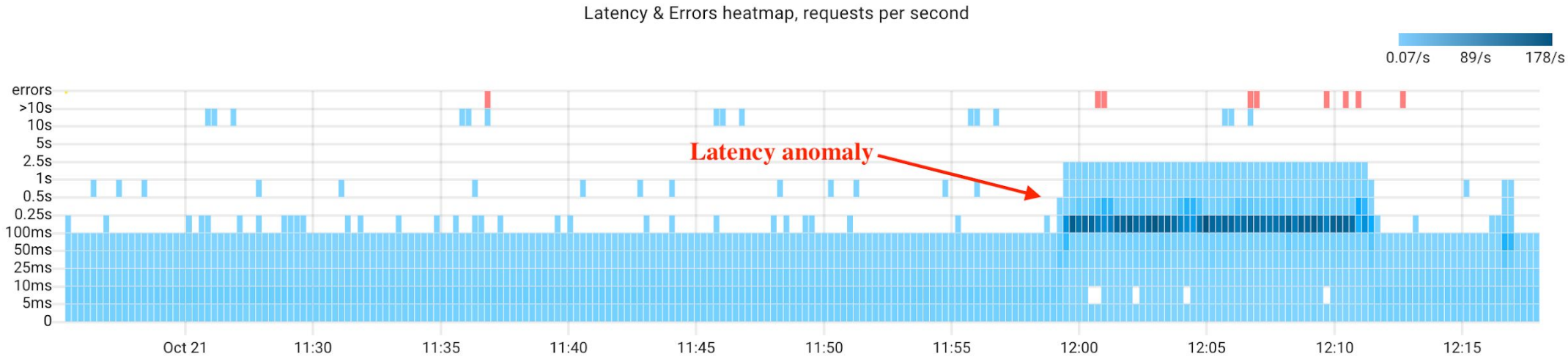


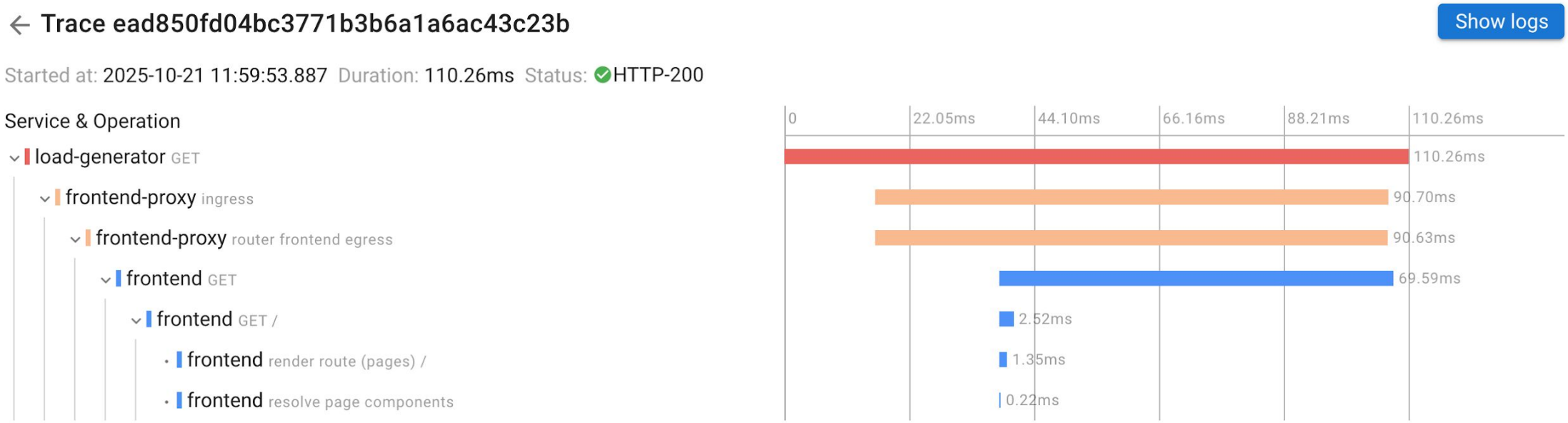
- OpenTelemetry demo
- Every app is instrumented with OpenTelemetry
- Enabling the **loadGeneratorFloodHomepage** feature flag:

  Flood the frontend with a large amount of requests.

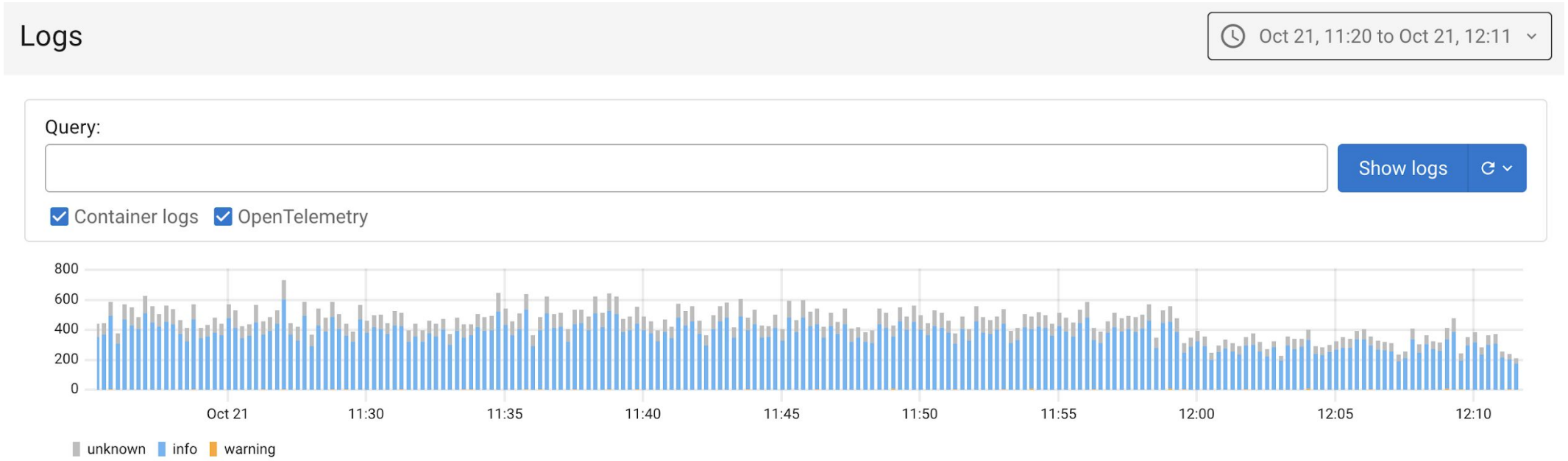
Traces

last hour



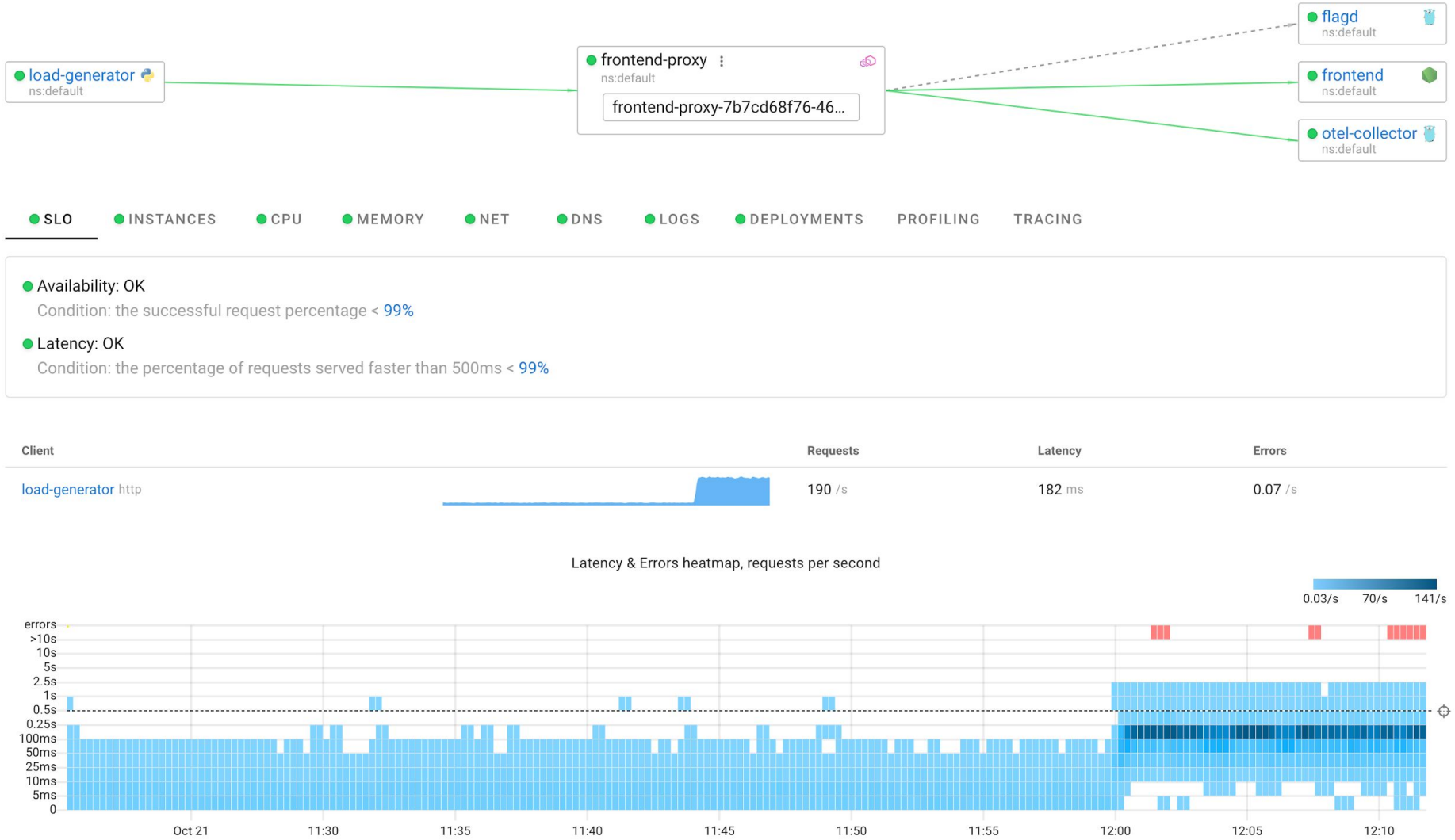


Load-generator -> frontend-proxy -> proxy (increased latency)



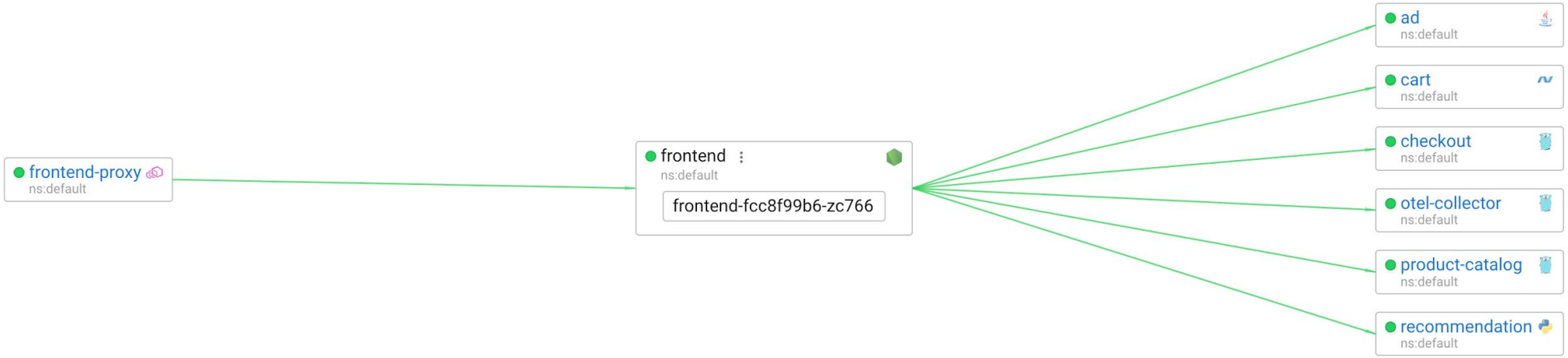
Nothing interesting in logs

Frontend-proxy

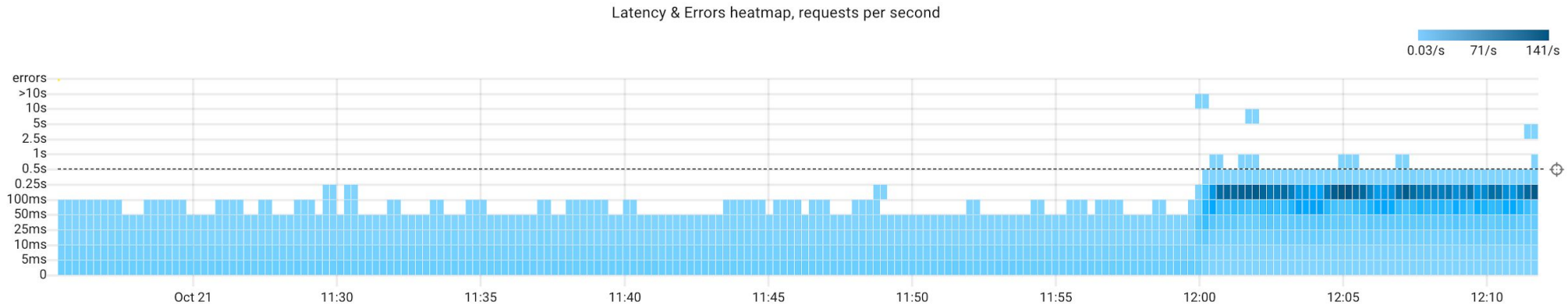


coroot :~#

Frontend

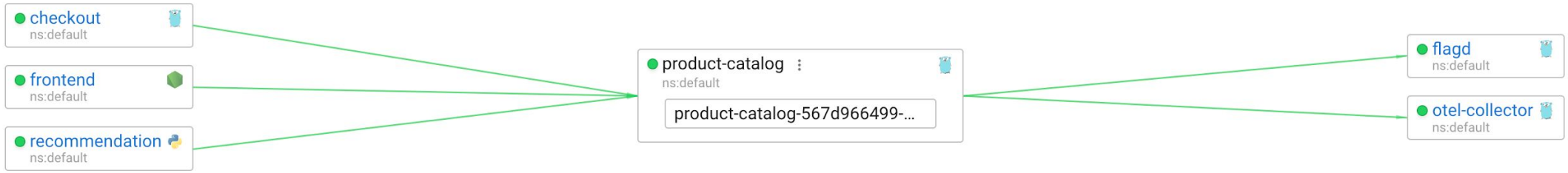


Client	Requests	Latency	Errors
frontend-proxy http	190 /s	139 ms	–

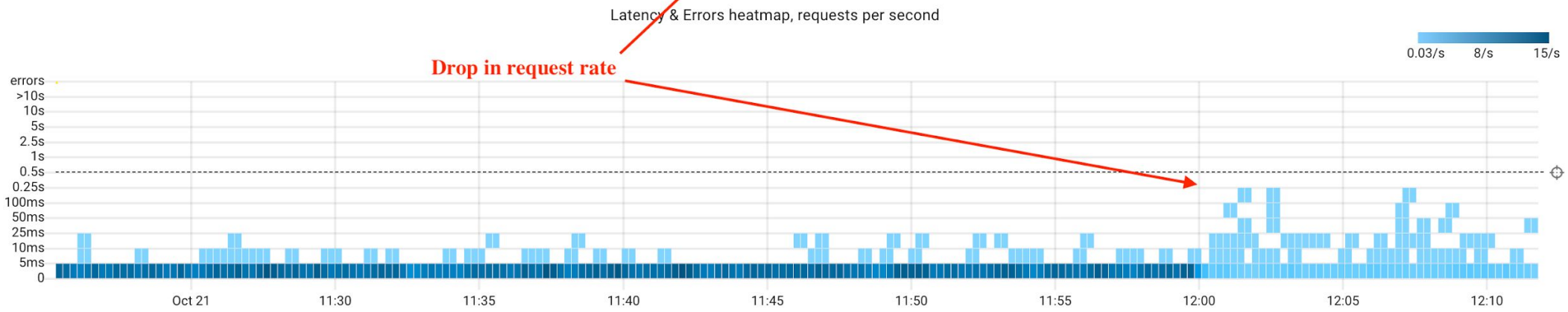


coroot :~#

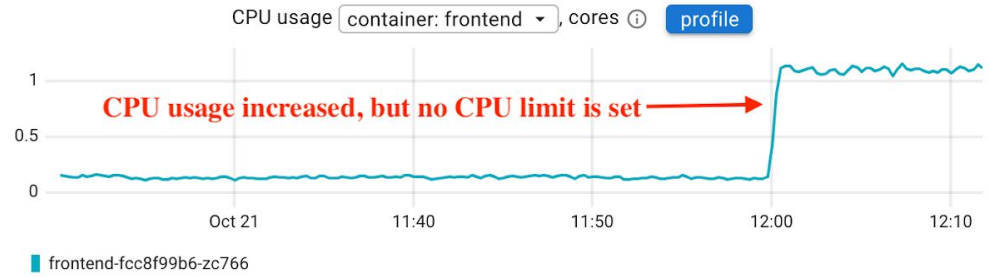
Product-catalog

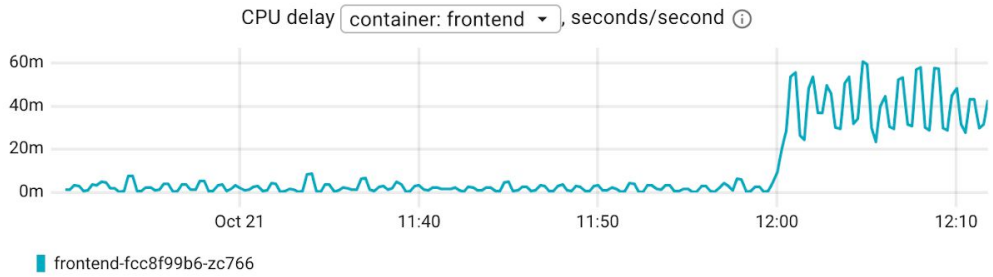


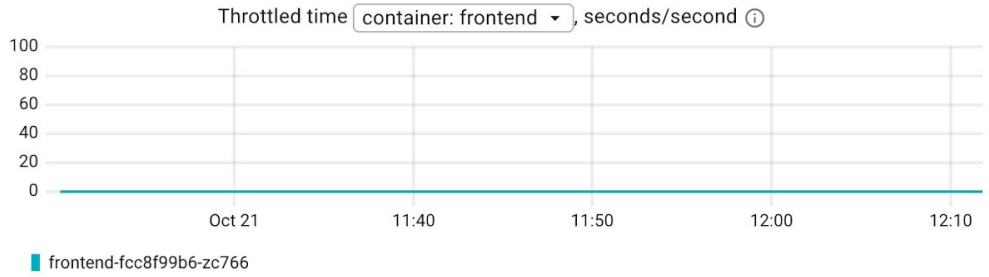
Client	Requests	Latency	Errors
checkout http	0.3 /s	0.5 ms	–
frontend http	0.03 /s	29 ms	–
recommendation http	0.6 /s	0.6 ms	–

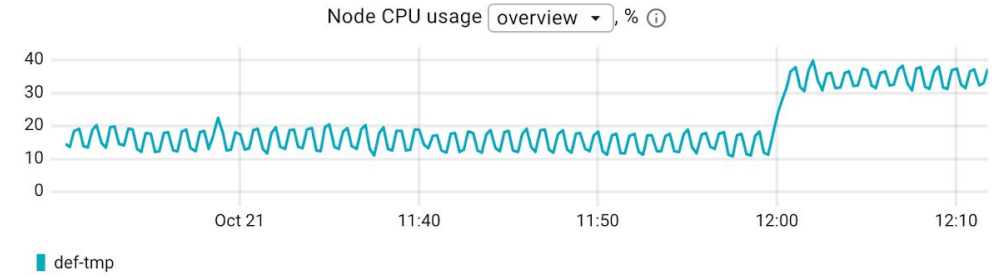


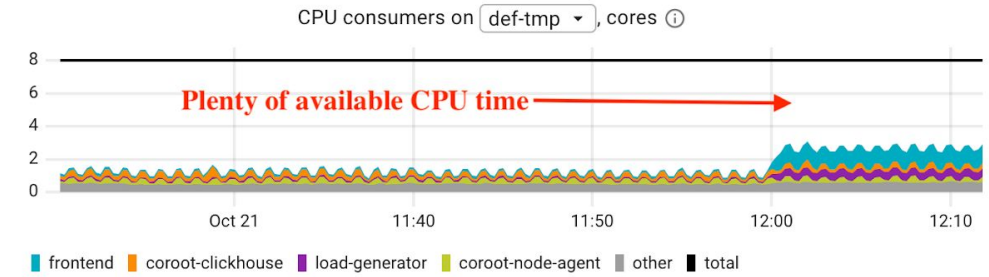
Back to Frontend



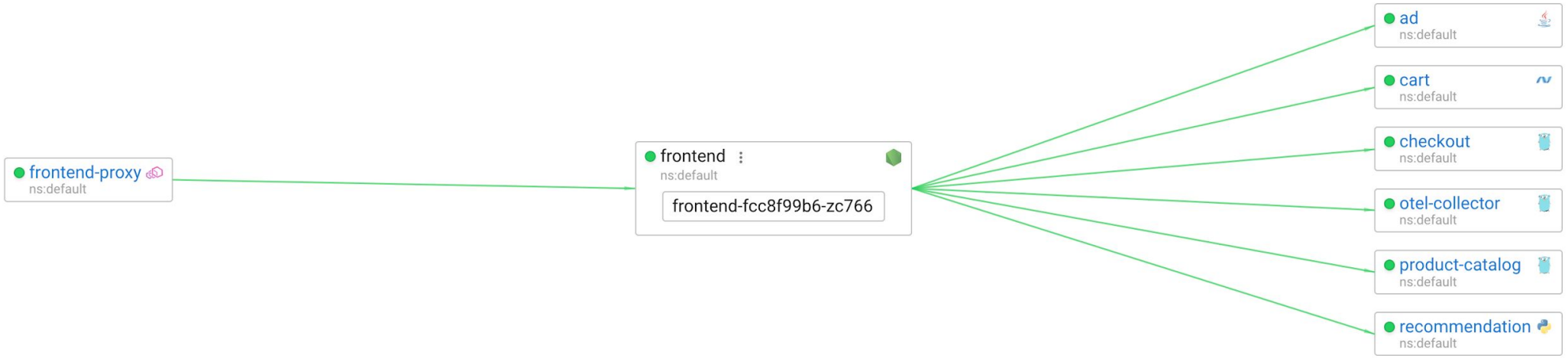


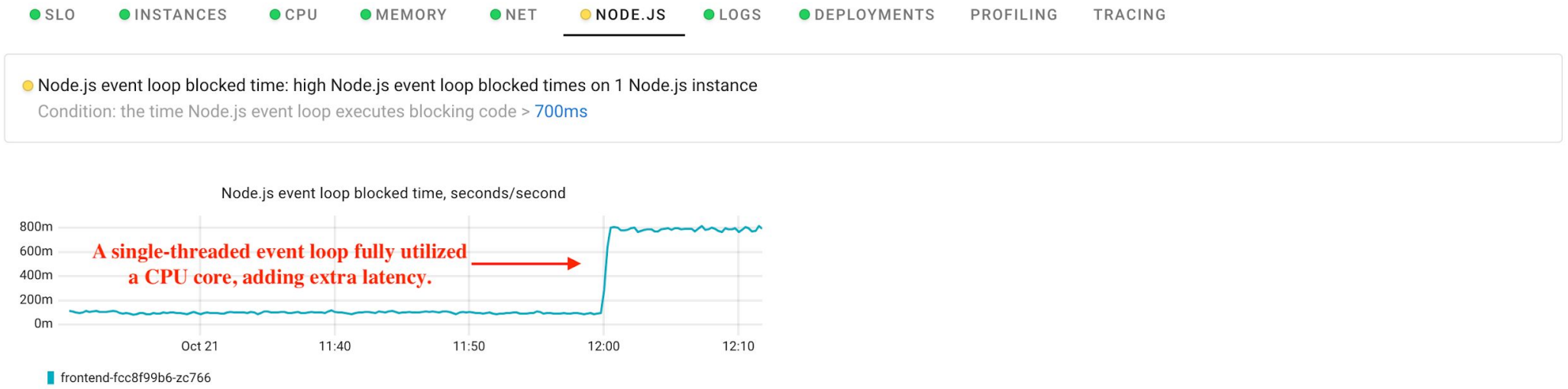






The answer


frontend-proxy
ns:default
frontend
ns:default
frontend-fcc8f99b6-zc766
ad
ns:default
cart
ns:default
checkout
ns:default
otel-collector
ns:default
product-catalog
ns:default
recommendation
ns:default

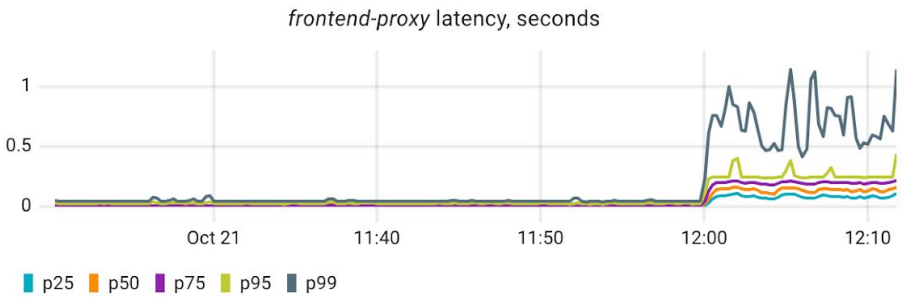

SLO
INSTANCES
CPU
MEMORY
NET
NODE.JS
LOGS
DEPLOYMENTS
PROFILING
TRACING
Node.js event loop blocked time: high Node.js event loop blocked times on 1 Node.js instance
Condition: the time Node.js event loop executes blocking code > 700ms
Node.js event loop blocked time, seconds/second
800m
600m
400m
200m
0m
A single-threaded event loop fully utilized
a CPU core, adding extra latency.
Oct 21
11:40
11:50
12:00
12:10
frontend-fcc8f99b6-zc766

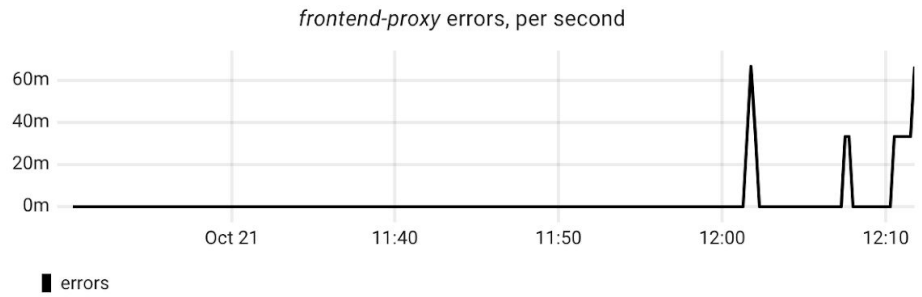
LLM + precise context

Anomalies > frontend-proxy

Oct 21, 11:20 to Oct 21, 12:11

Service Level Indicators of frontend-proxy





Root Cause

The `frontend` service (Node.js) is experiencing severe event loop blocking, causing high CPU delay and increased latency. This cascades to `frontend-proxy` which depends on it, resulting in failed requests (503 errors) and latency violations. The Node.js event loop blocked time reached `0.814702` seconds/second, indicating the application is overwhelmed and cannot process requests efficiently.

Show more details

a lot of charts and explanations

Immediate Fixes

Scale up the `frontend` service to handle the increased load:

this fix actually works

```
kubectl scale deployment frontend --replicas=3 -n default
```

This will distribute the load across more instances and reduce the event loop blocking that's causing the performance degradation.

Comprehensive context for AI RCA

- System topology/Service Map (traces, metrics)
- Service Level Indicators (SLIs) of every service and database
- Logs
- Events (Kubernetes, Cloud, Code changes)
- Resource metrics (CPU, Memory, PSI, Disk)
- Node metrics
- Network metrics
- Runtime metrics (JVM, Node.js, Python, Go, ….)
- Database-specific metrics (Postgres, MySQL, Mongo, …)
- Profiles
- Custom metrics associated with a service with clear explanations for LLMs

How to gather telemetry data

- Manual instrumentation
- SDK-based instrumentation (OpenTelemetry)
- eBPF-based instrumentation
- Or a hybrid approach combining them all

Where to store telemetry data

- Anywhere you want :)
- It doesn't really matter for root cause analysis
- Some options are cheaper or more efficient than others

The only bad choice? **An empty storage**.

How to work with AI

- Expose all your data to an LLM via MCP (Model Context Protocol) or tools
  - In our experiments, this performs poorly.
  - It's hard to see the big picture from raw metrics, logs, and traces, even for AI (and humans).

- Preprocess the data and feed LLMs only the findings and relevant context
  - Requires other methods to detect what truly matters without LLMs
  - Super cost-effective in terms of input tokens

Benchmarks and coverage

- There are no established RCA benchmarks yet, but we suggest starting with Otel demo failure scenarios: https://coroot.com/ai-benchmark

Coverage is a tricky topic - we can't measure how many of the infinite failure scenarios we can actually explain. But we could say:

- For 90+% of issues, we identify the problematic service
- We detect 80%+ of infra-, orchestrator-, network-, and cloud-related failures
- We catch the most common JVM, Node.js, and similar app issues
- 100% of incidents caused by code or infra changes are detected
- 80% of common database issues - locks, bad queries, missing indexes, schema changes

Final thoughts

- LLMs can assist with root cause analysis
- They can summarize, add context, and even suggest fixes based on symptoms
- But they need context about your system and the current issue
- If you can find the root cause with your existing telemetry, AI can too - just much faster
- And the opposite is also true: trash in → trash out

Thank you, Let’s connect!

https://www.linkedin.com/in/nikolay-sivko
https://github.com/coroot/coroot

coroot :~#