Gen3 Observability

Gen3 Community Forum
September 2, 2026

The Agenda

- **Current solutions and future plans for observability tools of Gen3 systems** - Gen3 Platform Engineering Team, University of Chicago
- **Gen3 observability in Aotearoa Genomic Data Repository** - Carvin Chen, University of Auckland
- **Understanding observability concepts in kubernetes/cloud-native** - Colin Griffin, Krumware
- **Operating Gen3 on AWS: Monitoring and Observability** - Guerdon Mukama, Australian BioCommons

Gen3 observability

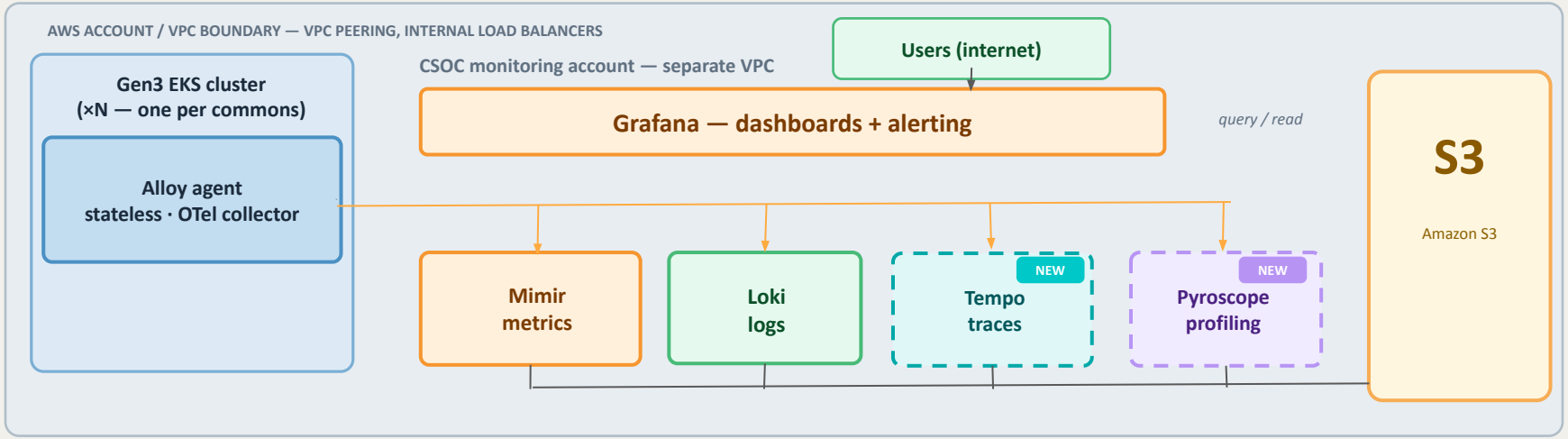
By the gen3 platform engineering team

Grafana

Ajo Augustine

Gen3 Observability: one central stack, many clusters

A stateless Alloy agent on every Gen3 EKS cluster; one central, S3-backed LGTM backend in the CSOC account.



- **Every cluster runs only Alloy** — stateless, cheap. The stateful backend runs once, centrally, in CSOC.
- **Mimir/Loki/Tempo: internal LBs only**, reached via VPC peering — e.g. mimir.planx-pla.net
- **Grafana** is the only internet-facing piece, access restricted.

What Alloy collects, and where it lands

One agent, three signals — plus curated alerts, dashboards, and browser RUM.

Also on every Alloy: OTLP receivers on :4317 (gRPC) / :4318 (HTTP) — apps send OTLP to their local in-cluster Alloy, which batches & forwards.

METRICS → Mimir

- Auto-discovery scrape of pods/services annotated **prometheus.io/scrape: "true"**
- **node-exporter**, **kube-state-metrics**, kubelet metrics
- Filtered to a **curated allow-list** (regex keep rules) — controls cardinality & S3 cost

LOGS → Loki

- All pod logs, labeled by **namespace / pod / container / app**
- Kubernetes **cluster events** (kube events → Loki)
- High-cardinality rollout labels (e.g. **pod_template_hash**) dropped

TRACES → Tempo

- OTLP receiver → batch processor → **Tempo exporter**
- Apps send OTLP traces to their **local Alloy**
- Backend enabled in prod via GitOps NEW

Built-in alerting

Gen3 alert rules ship in the Helm chart's alerting values — firing as soon as Grafana is up.

Curated dashboards

Gen3-specific dashboards from uc-cdis/grafana-dashboards.

Faro RUM collector

Alloy ingests browser RUM from Portal/Fence → Loki.

Pyroscope profiling

Continuous profiling, CSOC-side, viewed as a Grafana data source. NEW

OBSERVABILITY

Pyroscope

Continuous profiling for production systems

Jawad Qureshi

What it is · How to instrument · Auto-instrumentation · What it's good for

What is Pyroscope?

— Open-source continuous profiler, part of the Grafana observability stack

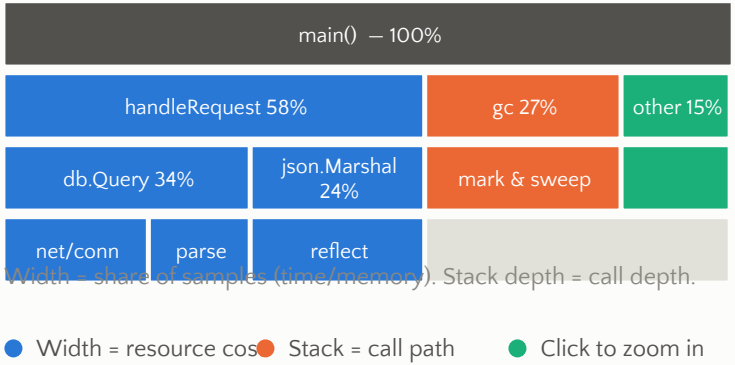
— Continuously collects CPU, memory, and other resource profiles from live services

— Stores profiles efficiently over time, so you can go back and look at any point in history

— Visualizes profiles as flame graphs — which functions/lines consumed time or memory, and how much

— Answers a question logs and traces can't: which lines of code are actually costing you resources

Anatomy of a flame graph



Pyroscope samples running processes (e.g. every 10ms) and aggregates
samples that share a call stack into these nested bars — no code changes needed to view existing profiles once collection is wired up.

How to instrument your service

Three ways to get profiles flowing into Pyroscope, from most control to least effort:

1

SDK / library

— Add Pyroscope's client library to your app (Go, Java, Python, Ruby, .NET, Node, Rust, PHP, eBPF)

— A few lines at startup: server address, app name, tags

— Full control over profile types (CPU, alloc, heap) and tags per request/goroutine

— Best when you want custom tags (e.g. tenant ID, request route) baked into profiles

2

Language-native profiler + push

— Many runtimes ship a built-in profiler (pprof, JFR/async-profiler, py-spy)

— Point its output at Pyroscope, or use Grafana Alloy/Agent to scrape & forward

— No app code changes — configuration only

— Good fit for existing pprof/JFR-based workflows

3

eBPF auto-instrumentation

— Kernel-level profiler attaches to running processes — zero code, zero redeploy

— Deployed as a daemonset (e.g. Grafana Alloy) across a Kubernetes node

— Automatically discovers and profiles every process on the host

— Covered next — the fastest way to get whole-fleet coverage

Auto-instrumentation: profiling without touching code

— An eBPF-based agent (e.g. Grafana Alloy) runs as a node-level daemon

— It attaches to every process on the host and samples stacks directly from the kernel

— No SDK, no redeploy, no restart — works on services you don't own or can't modify

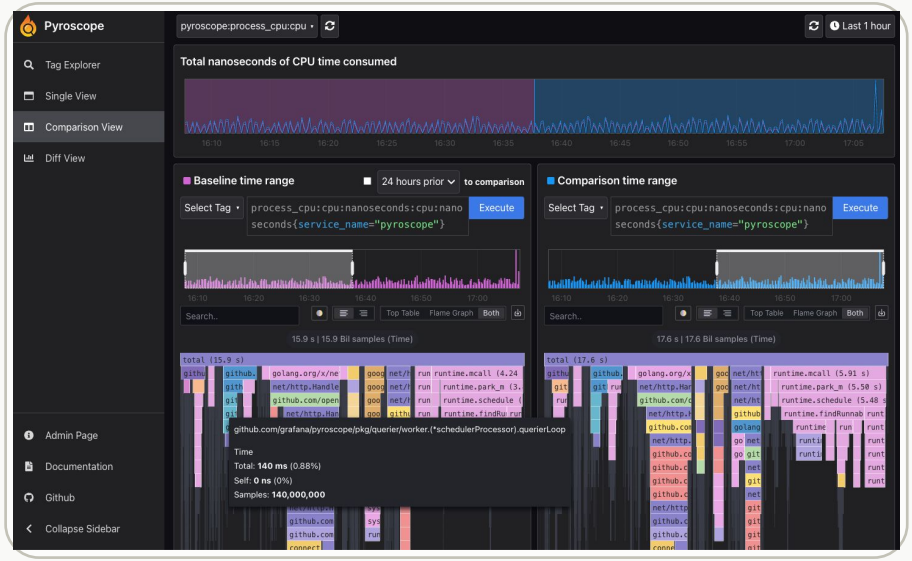
— Tags profiles automatically with service name, k8s labels, container/pod metadata

— Once profiles are flowing, compare a baseline window against a comparison window

— The screenshot: two 24h CPU-time flame graphs for the same service, side by side

— Diff view highlights exactly which functions grew or shrank between deploys

— Turns “CPU usage went up” into “this function is the reason”



Pyroscope comparison view — baseline vs. comparison flame graphs

USE CASES

What Pyroscope is good for

Cutting infra cost

Find the exact functions burning CPU or memory across a fleet, and cut cloud spend by fixing them instead of scaling out.

Regression hunting

Diff flame graphs before/after a deploy to pinpoint which code change caused a performance regression.

Debugging "it's slow" reports

When a service is slow with no clear error or log signal, profiling shows where time is actually going.

Memory leak triage

Compare heap profiles over time to see which allocation site is growing unbounded.

Whole-fleet visibility

eBPF auto-instrumentation gives coverage across services you don't control the code for — third-party or legacy.

Correlating with traces/logs

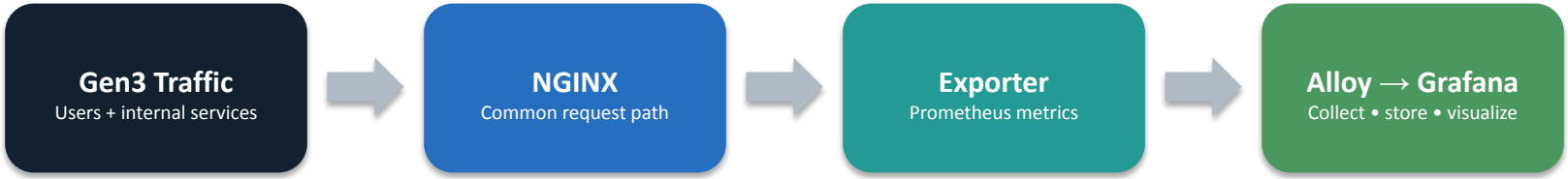
Pairs with Grafana Tempo/Loki: jump from a slow trace span straight to the flame graph for that time window.

Nginx Metrics

Edward Malinowski

NGINX Metrics Add a Consistent Service-Level Signal

Every proxied Gen3 service can expose the same request and connection telemetry without changing application code.



Consistent
Same telemetry pattern across services

Low-friction
No app-specific metrics implementation required

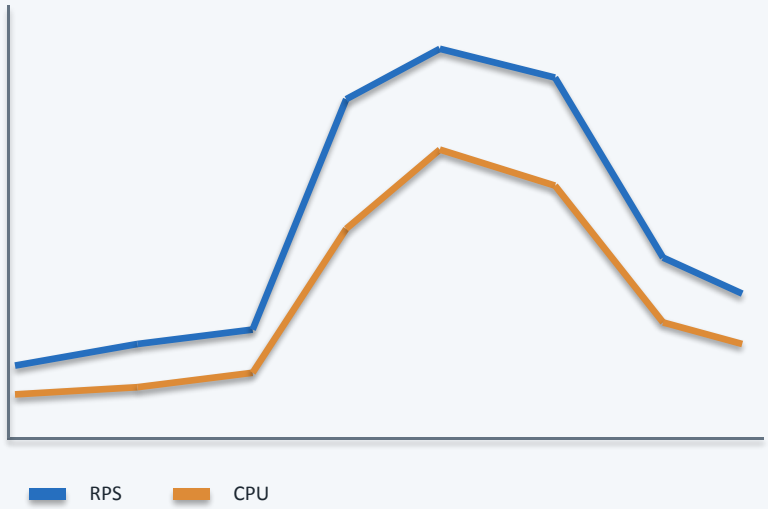
Correlatable
Compare traffic with CPU, memory and pod health

Result: the service layer becomes measurable instead of a black box.

RPS Gives Infrastructure Metrics Operational Context

Request volume helps explain why resource usage changes — and makes troubleshooting faster.

Example: Traffic spike correlated with CPU



Traffic patterns
Which services are busy, and when?

Incident triage
Did a failure coincide with a request spike?

Capacity planning
How much demand can one replica handle?

Deployment validation
Did request behavior change after release?

"80% CPU" is more useful when we know whether traffic is 20 RPS or 200 RPS.

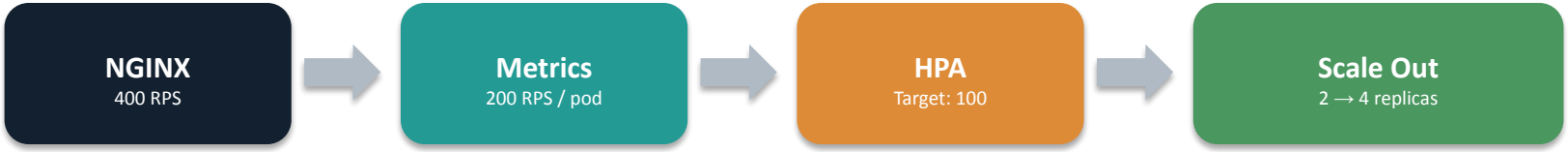
The Same Metrics Can Drive Demand-Based Autoscaling

Observability data can become a control signal: scale replicas based on incoming request load, not only CPU.

Traditional HPA
CPU / memory threshold

RPS-based HPA
Requests per replica

Why it matters
Closer to actual service demand



Observe → Understand → Alert → Automate

RPS is especially useful for request-driven services where CPU may lag or poorly represent demand.

Open Telemetry

Justin Barnowski

The Problem

Fragmented observability makes unified telemetry impossible.

Vendor Lock-In

Each observability tool uses a proprietary data format. Switching vendors means re-instrumenting services from scratch.

Incompatible Agents

Different services require different collection agents with no shared protocol. Operational overhead multiplies with every new service.

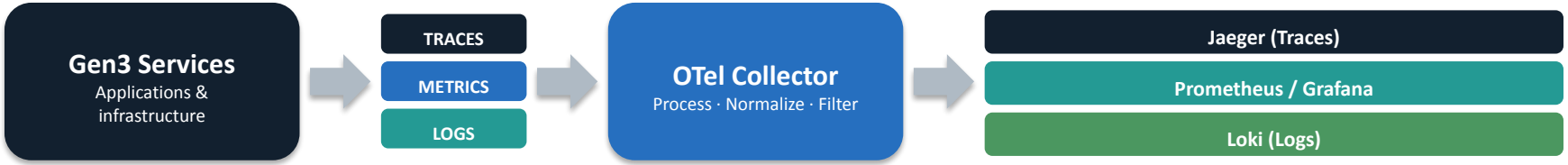
Data Silos

Metrics, logs, and traces live in separate backends. Correlating a slow trace with its CPU spike requires manual context-switching between tools.

Without a common standard, distributed traces, metrics, and logs cannot be unified.

OpenTelemetry Architecture

The Unified Collection Hub



Collect Once, Send Anywhere.

Vendor-Neutral
OTel is a CNCF standard. Swap backends without touching your instrumentation code.

Single Agent
One collector per node replaces the per-service agent sprawl of fragmented setups.

Unified Telemetry
Traces, metrics, and logs share the same pipeline, enabling correlated analysis in Grafana.

UC-CDIS Gen3 Implementation Plan

Instrumenting our microservices in an AWS environment.

1 Codebase Scan & Target Selection

Inventory Gen3 microservices (frontend, core API, data services) on GitHub to identify high-impact instrumentation targets.

2 SDK Instrumentation

Deploy OTel SDKs to key services. Use automatic instrumentation for HTTP, SQL, and gRPC. Add manual spans for critical business logic.

3 AWS Collector Deployment

Deploy Grafana Alloy across all services as the OTel collection agent. Alloy handles scraping, processing, and forwarding telemetry for the full Gen3 fleet.

4 EKS Observability Stack

Export telemetry to self-hosted Grafana and Prometheus running on AWS EKS. Gen3 manages its own stack rather than using managed services, supporting our multi-commons architecture.

Observe → Instrument → Collect → Analyze

Repos

- Helm Repo: https://github.com/uc-cdis/gen3-helm
  - Observability chart: https://github.com/uc-cdis/gen3-helm/tree/master/helm/observability
  - Dev observability setup: https://github.com/uc-cdis/gen3-helm/blob/master/examples/local_lgtm.yaml
- Grafana Dashboards: https://github.com/uc-cdis/grafana-dashboards

Questions

Waipapa Taumata Rau University of Auckland

genomics aotearoa

Gen3 Observability in Aotearoa Genomic Data Repository

3 September 2026

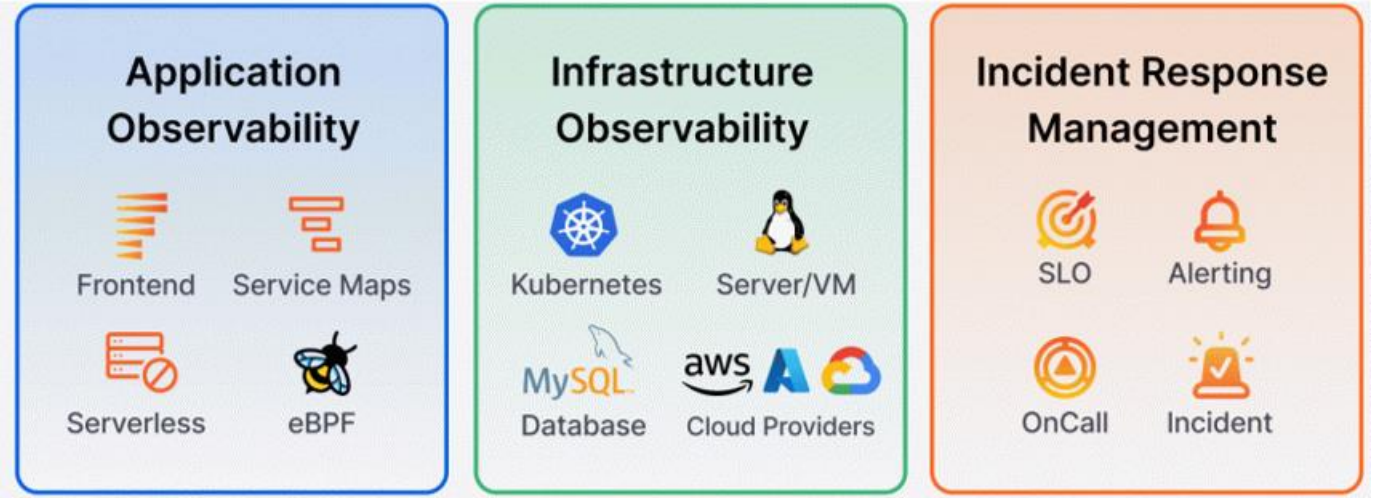
Carvin Chen
Digital Research Innovation and AI (DRAI Inn.)

About the DRAI Inn. team

- Digital Research Innovation and Artificial Intelligence Group (DRAI Inn.).
- Located within Digital Services as a part of the broader eResearch platform at the University of Auckland
- We work close to research
- We explore and discovery emerging capabilities, products, platforms
- We learn through action and partnership

- We have an ongoing partnership with GA for the development of the Aotearoa Genomic Data Repository (AGDR) and with the Rakeiora leadership for the development of the Rakeiora Genomics Platform.
- Platform is hosted on Infrastructure that was NeSI and is now REANNZ.
- GEN3 is used for the Aotearoa Genomics Data Repository and Rakeiora Genomics Platform (prototype application)
  - https://data.agdr.org.nz/
  - https://browse.rakeiora.ac.nz/

AGDR observability is primarily built on the Grafana OSS stack

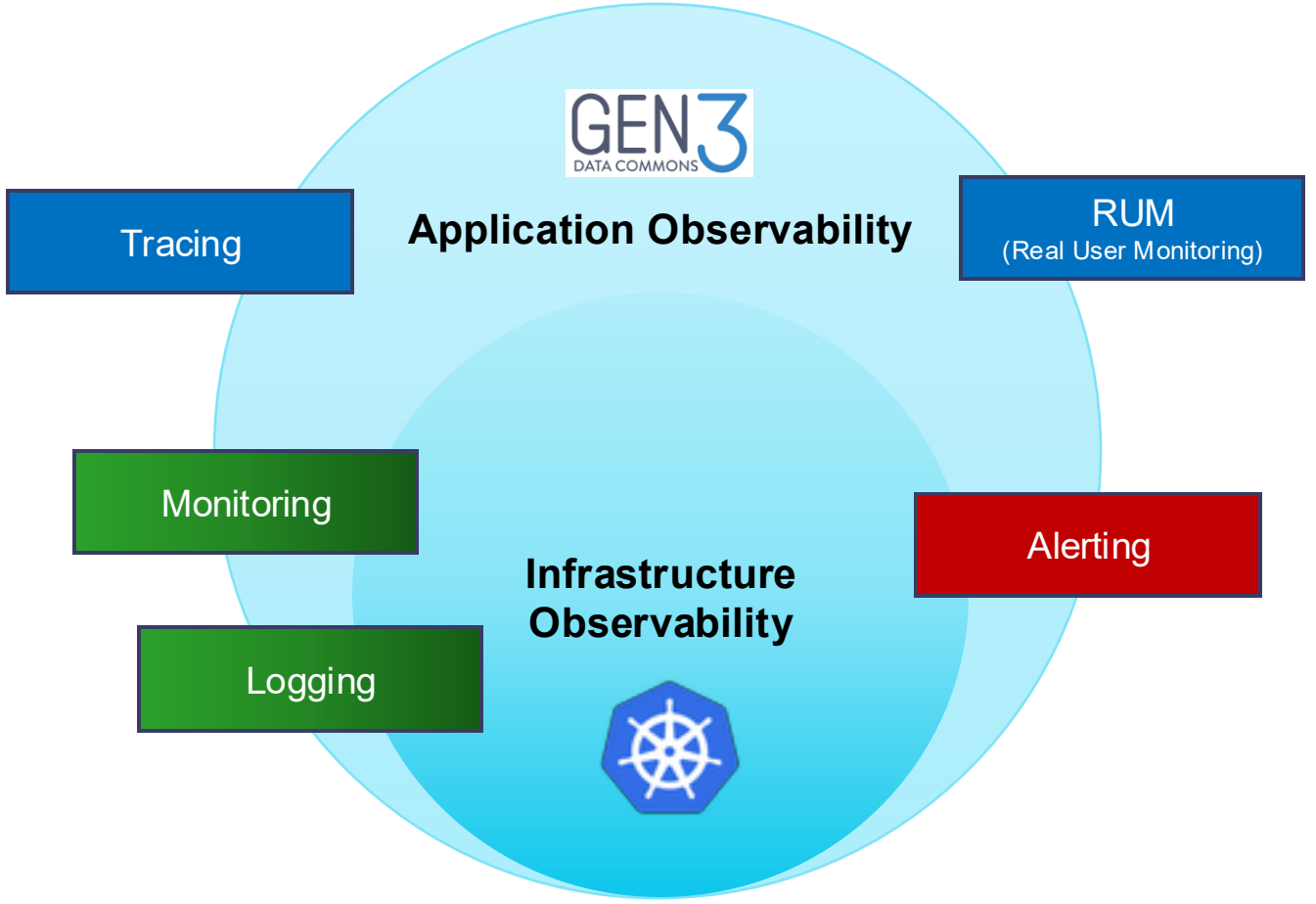


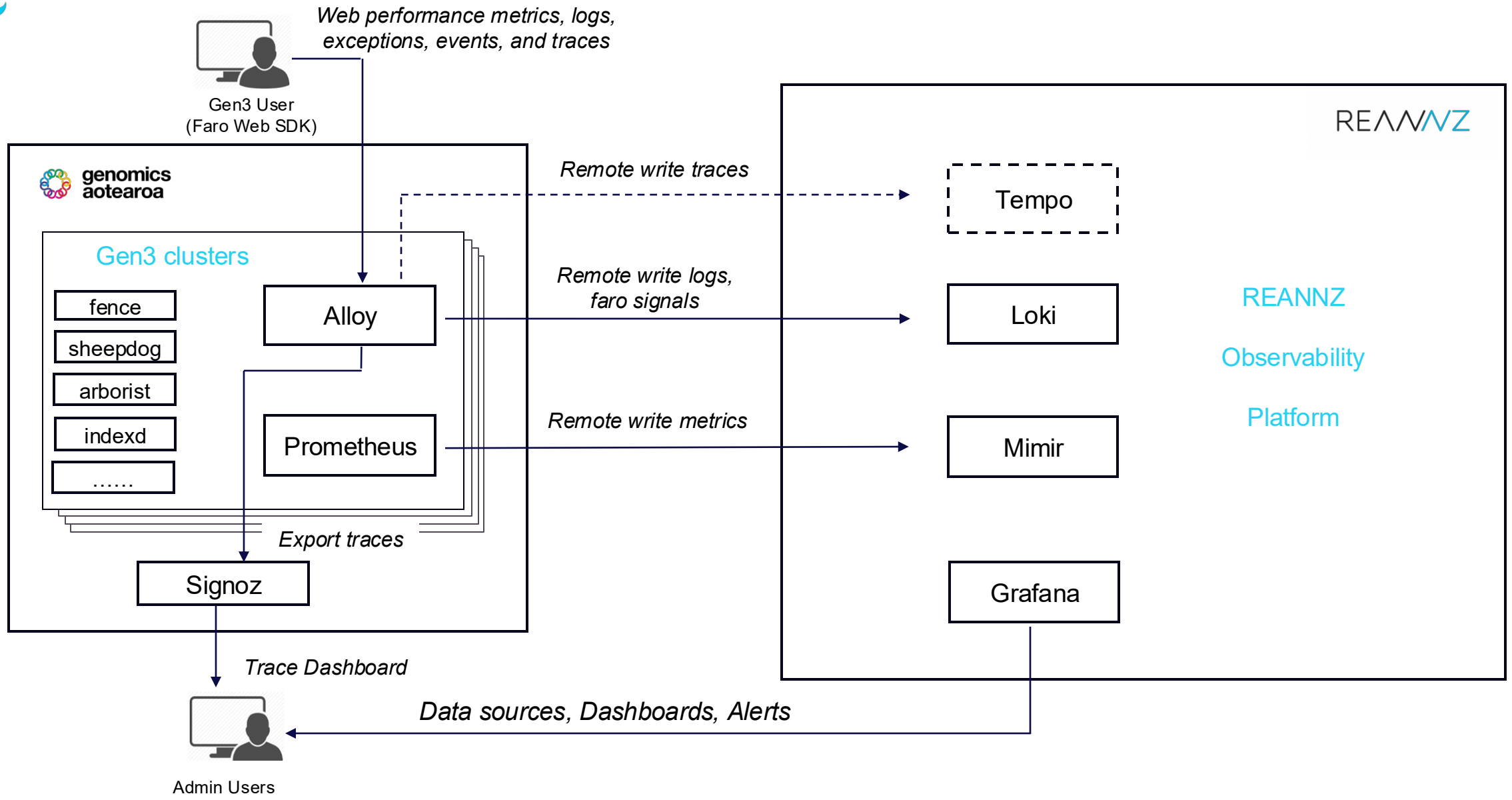
Grafana Alloy

Prometheus

SigNoz

Grafana


GEN3
DATA COMMONS
Application Observability
Tracing
RUM
(Real User Monitoring)
Monitoring
Alerting
Infrastructure
Observability
Logging


Web performance metrics, logs, exceptions, events, and traces
Gen3 User
(Faro Web SDK)
genomics aotearoa
Gen3 clusters
fence
sheepdog
arborist
indexd
……
Alloy
Prometheus
Export traces
Signoz
Trace Dashboard
Admin Users
Remote write traces
Remote write logs, faro signals
Remote write metrics
Data sources, Dashboards, Alerts
REANNZ
Tempo
Loki
Mimir
Grafana
REANNZ
Observability
Platform

Alloy : collector and shipper of the logs, faro signals, traces

```
otelcol.receiver.otlp "default" {
  grpc {
    endpoint = "0.0.0.0:4317"
  }
  http {
    endpoint = "0.0.0.0:4318"
  }
  output {
    traces = [otelcol.processor.batch.default.input]
  }
}


otelcol.processor.batch "default" {
  output {
    traces = [otelcol.exporter.otlphttp.signoz.input]
  }
}


otelcol.exporter.otlphttp "signoz" {
  client {
    endpoint = "https://otlp.dev.drai.auckland.ac.nz"
    headers = {
      "Authorization" = "Bearer " +
env("SIGNOZ_OTLP_BEARER_TOKEN"),
    }
  }
}
```

```
faro.receiver "default" {
  server {
    listen_address = "0.0.0.0"
    listen_port    = 12347
    cors_allowed_origins = [
      "https://test.agdr.org.nz",
      "https://fef.test.agdr.org.nz",
    ]
  }
  extra_log_labels = {
    service = "gen3-portal",
    cluster = "agdr-test-helm",
  }
  output {
    logs   = [loki.write.default.receiver]
    traces = [otelcol.processor.batch.default.input]
  }
}
```

Portal update to enable Faro RUM & Tracing

Fix Faro connect-src CSP violation

Fixed a naming mismatch between webpack.config.js (connectSrc) and src/index.ejs (connect_src) that silently dropped all dynamically-configured CSP origins (Faro, Workspace, WTS, Manifest Service, iframe apps, etc.) from connect-src/frame-src.

This fix enables passing grafanaFaroConfig.grafanaFaroUrl into the portal image build, resolving the runtime Content Security Policy (CSP) issue.

Enable Faro tracing instrumentation

Added @grafana/faro-web-tracing as a dependency and new TracingInstrumentation() to the instrumentations array in src/index.jsx, alongside the existing web/React instrumentations. getWebInstrumentations() (the default Faro instrumentation bundle) only covers Console/Errors/WebVitals/Session/View — it does **not** include tracing. The portal's Faro setup in src/index.jsx was never sending trace/span data, only logs and errors, even though grafanaFaroConfig was fully configured and Alloy's faro.receiver was reachable.

Microservice Deployment Changes for OpenTelemetry Tracing

This is an experiment on how to enable OTel tracing for gen3 microservices:

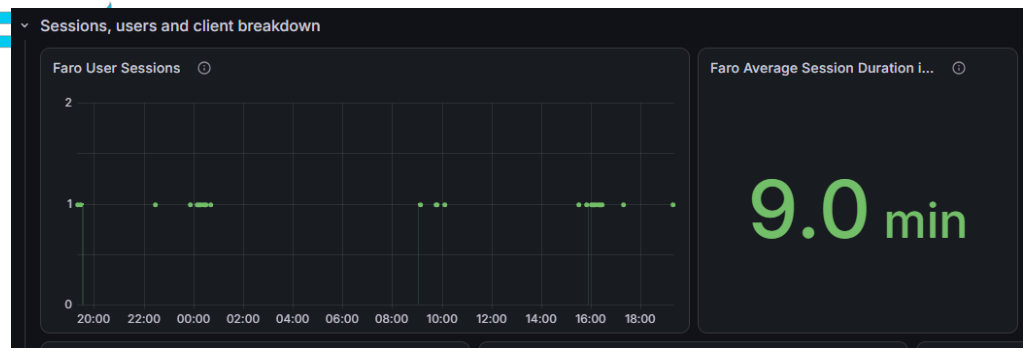
Patch OTel instrumentation to some of the deployments

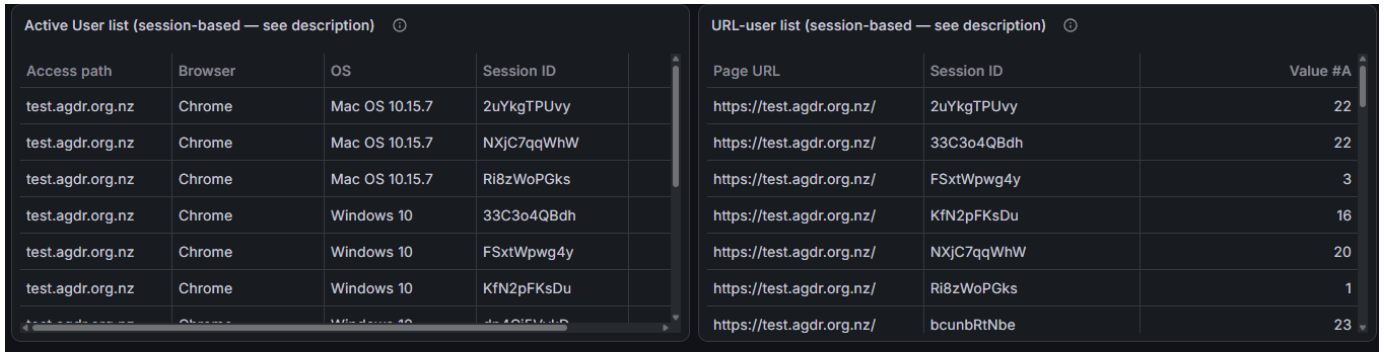
- fence-deployment
- sheepdog-deployment
- Indexd-deployment
- guppy-deployment

No image rebuild: an initContainer pip-installs the OTel auto-instrumentation packages into a shared emptyDir using fence's OWN image (quay.io/cdis/fence:2026.06).
so the C-extension deps (wrapt) match fence's Python ABI, and `opentelemetry-*` instrumentation packages are picked against the same versions of Flask/SQLAlchemy/requests/boto3 fence actually has installed. The main container then just gets PYTHONPATH pointed at that directory (in front of the existing /var/www/fence) and a handful of OTEL_* env.

Application monitoring : Gen3 Portal RUM Dashboard

Application monitoring : Gen3 Portal RUM Dashboard





Raw signals and errors

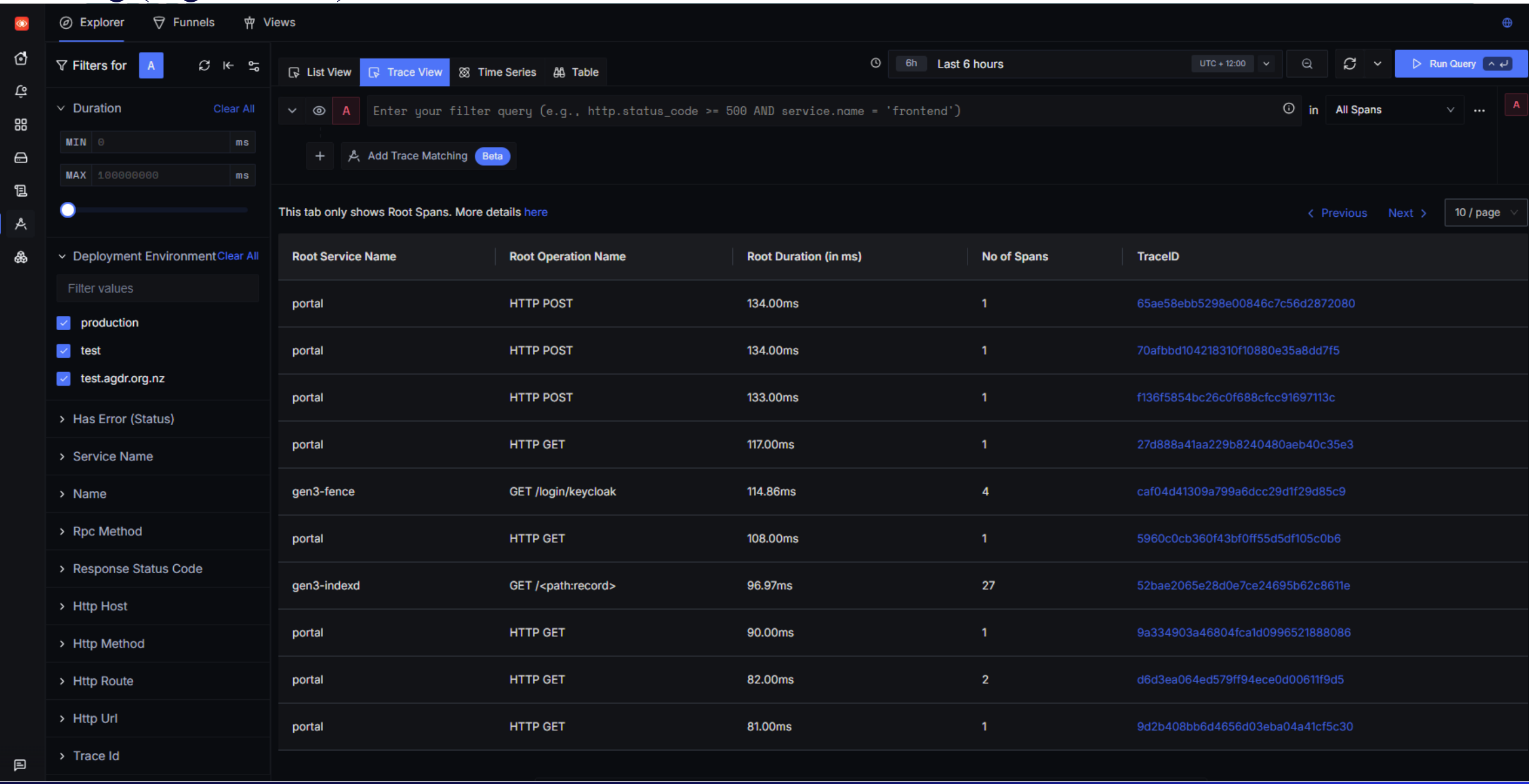
Recent JS exceptions

```
2026-09-02 15:46:57.533  UNK  timestamp="2026-09-02 03:46:55.881 +0000 UTC" kind=exception type=Error value="console.error: Could not find data with UID NZ-00024." stacktrace="Error: console.error: Could not find data with UID NZ-00024.\n  at o.b.filter.forEach.console.<computed> [as error] (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:402:1049216)\n  at ? (https://test.agdr.org.nz/bundle.js?f29796c54f3098bca8ea:1:66813)\n  at ic (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:83209)\n  at vs (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:102288)\n  at t.unstable_runWithPriority (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:426:3844)\n  at Vi (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:45024)\n  at ms (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:102047)\n  at Xc (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:93596)\n  at ? (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:418:45315)\n  at t.unstable_runWithPriority (https://test.agdr.org.nz/vendors~bundle.js?f29796c54f3098bca8ea:426:3844)" hash=3935945027526533229 sdk_version=1.19.0 app_name=portal app_namespace=AGDR-TEST app_version=test-7.2.6-2026.02 app_environment=test.agdr.org.nz session_id=ebh8GKFLu0 page_url=https://test.agdr.org.nz/discovery/NZ-00024 browser_name=Chrome browser_version=152.0.0.0 browser_os="Windows 10" browser_mobile=false browser_userAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36" browser_language=en-US browser_viewportWidth=2560 browser_viewportHeight=945
```

Recent RUM signals (raw)

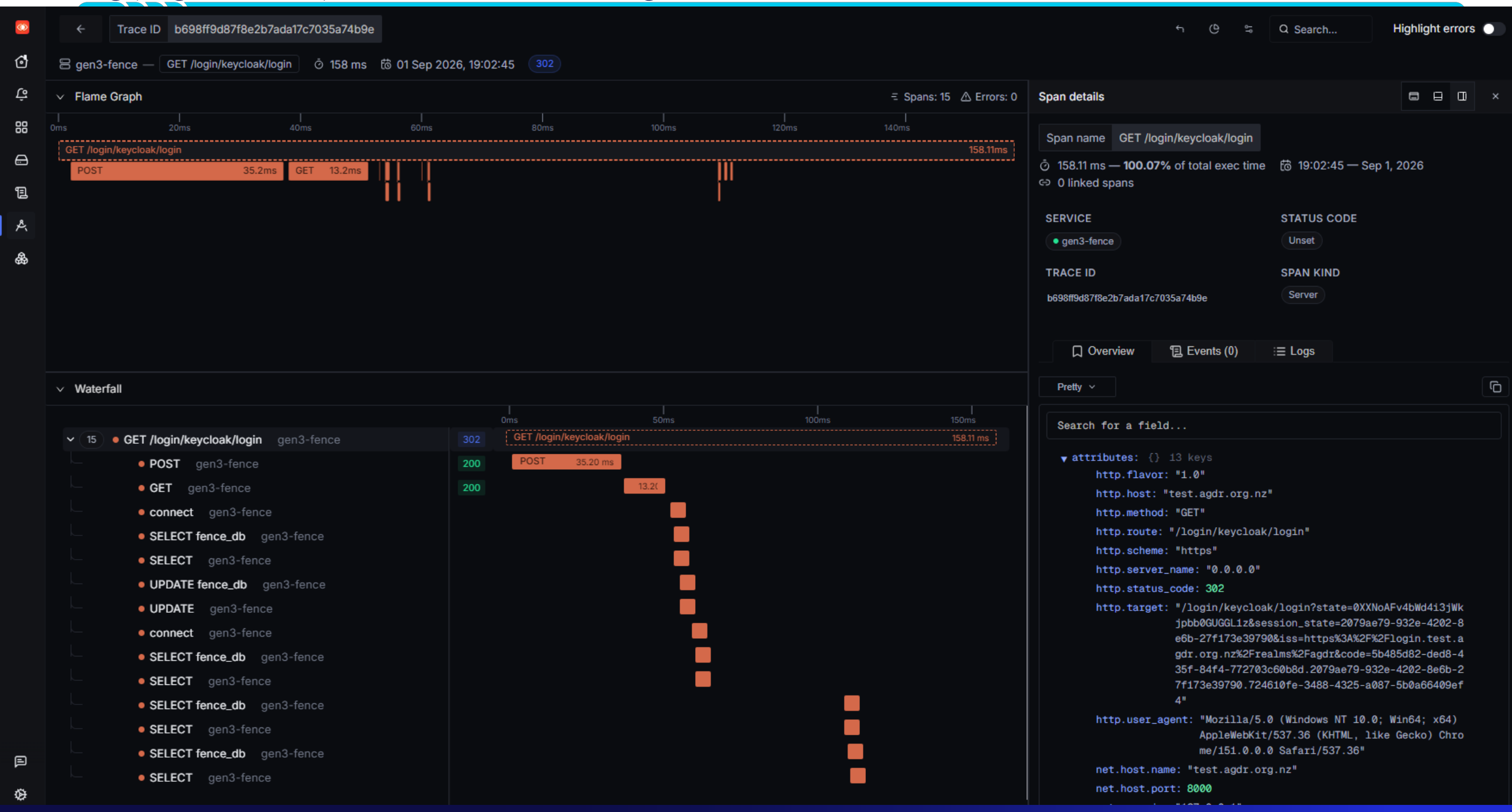
```
2026-09-02 19:14:34.554  UNK  timestamp="2026-09-02 07:14:30.407 +0000 UTC" kind=event event_name=faro.tracing.fetch event_domain=browser event_data_component=fetch event_data_duration_ns=138000128 event_data_http.host=www.google-analytics.com event_data_http.method=POST event_data_http.response_content_length=0 event_data_http.scheme=https event_data_http.status_code=0 event_data_http.url="https://www.google-analytics.com/g/collect?v=2&tid=G-804XWK8649&gtm=45je6910h2v9169584537za200zd9169584537&_p=1788333251202&gcd=13l3l3l3l1l1&npa=0&dma=0&_eu=AEAAAAQ&ae=a&cid=521305804.1761691768&frm=0&ngs=1&pscdl=noapi&rcb=19&sr=2560x1080&uaa=x86&uab=64&uafvl=Chromium%3B152.0.7977.65%7CNot%253FA_Brand%3B24.0.0.0%7CMicrosoft%2520Edge%3B152.0.4191.53&uam=&uamb=0&uap=Windows&uapv=15.0.0&uaw=0&ul=en-us&_s=8&tag_exp=115616985~115938466~115938468~118012009~118897920~118897930~120213116~120385423~120469145~120469153~120474863&dl=https%3A%2F%2Ftest.agdr.org.nz%2Fexplorer&dr=https%3A%2F%2Ftest.agdr.org.nz%2FDD&sid=1788333251&sct=73&seg=1&dt=Aotearoa%20Genomic%20Data%20Repository&en=page_view&_et=985&tfd=24548" event_data_http.user_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36 Edg/152.0.0.0" event_data_session.id=bcunbRtNbe event_data_session_id=bcunbRtNbe traceID=85ac08cb91caddd78dc69b7e62b3cecf spanID=2599cda14dbd6d43 sdk_version=1.19.0 app_name=portal app_namespace=AGDR-TEST app_version=test-7.2.6-2026.02 app_environment=test.agdr.org.nz session_id=bcunbRtNbe page_url=https://test.agdr.org.nz/explorer browser_name=Edge browser_version=152.0.0.0 browser_os="Windows 10" browser_mobile=false browser_userAgent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36 Edg/152.0.0.0" browser_language=en-US browser_viewportWidth=2329 browser_viewportHeight=862
```

Tracing (Signoz OSS) – list

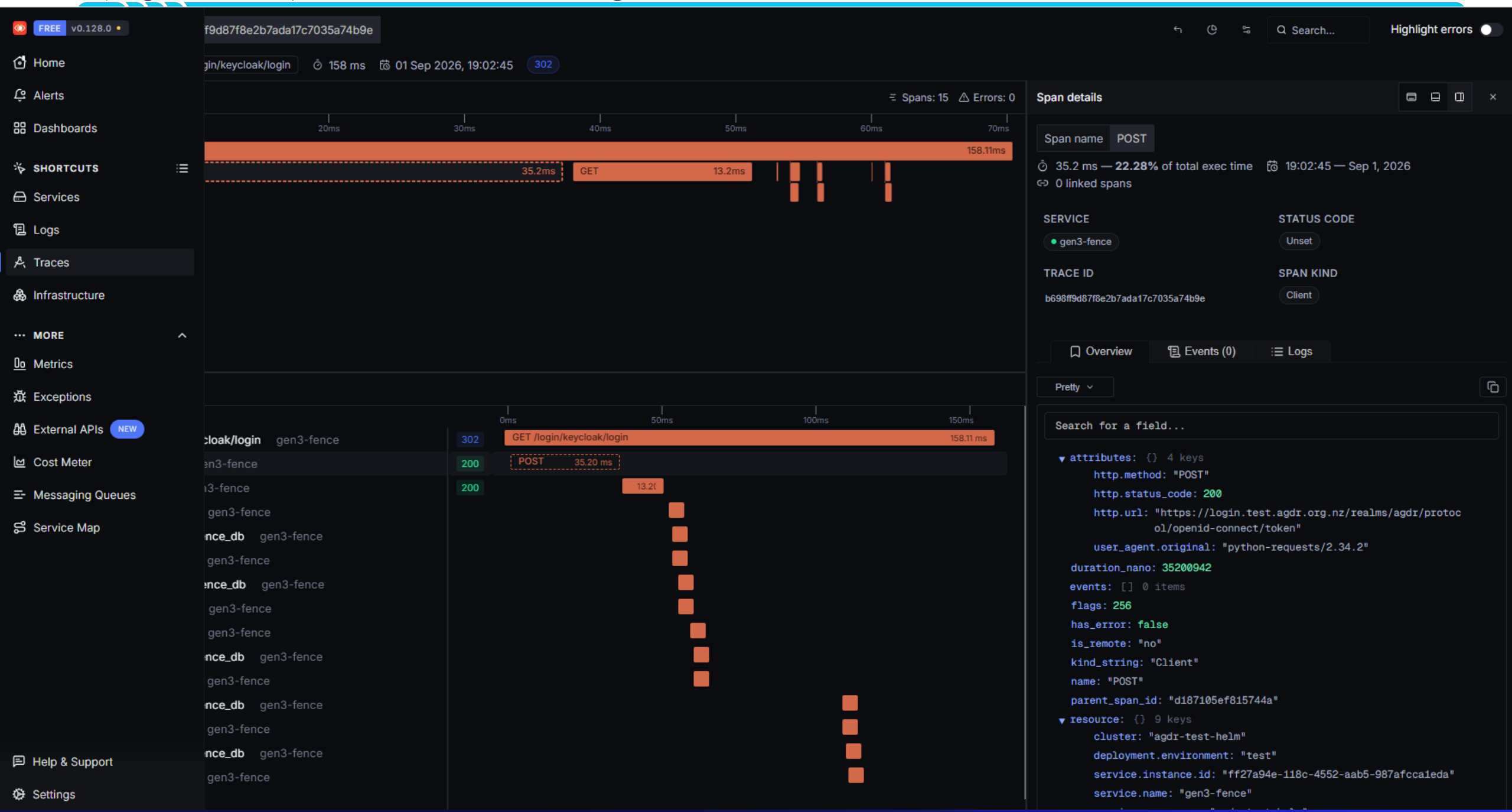


Root Service Name	Root Operation Name	Root Duration (in ms)	No of Spans	TraceID
portal	HTTP POST	134.00ms	1	65ae58ebb5298e00846c7c56d2872080
portal	HTTP POST	134.00ms	1	70afbbd104218310f10880e35a8dd7f5
portal	HTTP POST	133.00ms	1	f136f5854bc26c0f688cfcc91697113c
portal	HTTP GET	117.00ms	1	27d888a41aa229b8240480aeb40c35e3
gen3-fence	GET /login/keycloak	114.86ms	4	caf04d41309a799a6dcc29d1f29d85c9
portal	HTTP GET	108.00ms	1	5960c0cb360f43bf0ff55d5df105c0b6
gen3-indexd	GET /<path:record>	96.97ms	27	52bae2065e28d0e7ce24695b62c8611e
portal	HTTP GET	90.00ms	1	9a334903a46804fca1d0996521888086
portal	HTTP GET	82.00ms	2	d6d3ea064ed579ff94ece0d00611f9d5
portal	HTTP GET	81.00ms	1	9d2b408bb6d4656d03eba04a41cf5c30

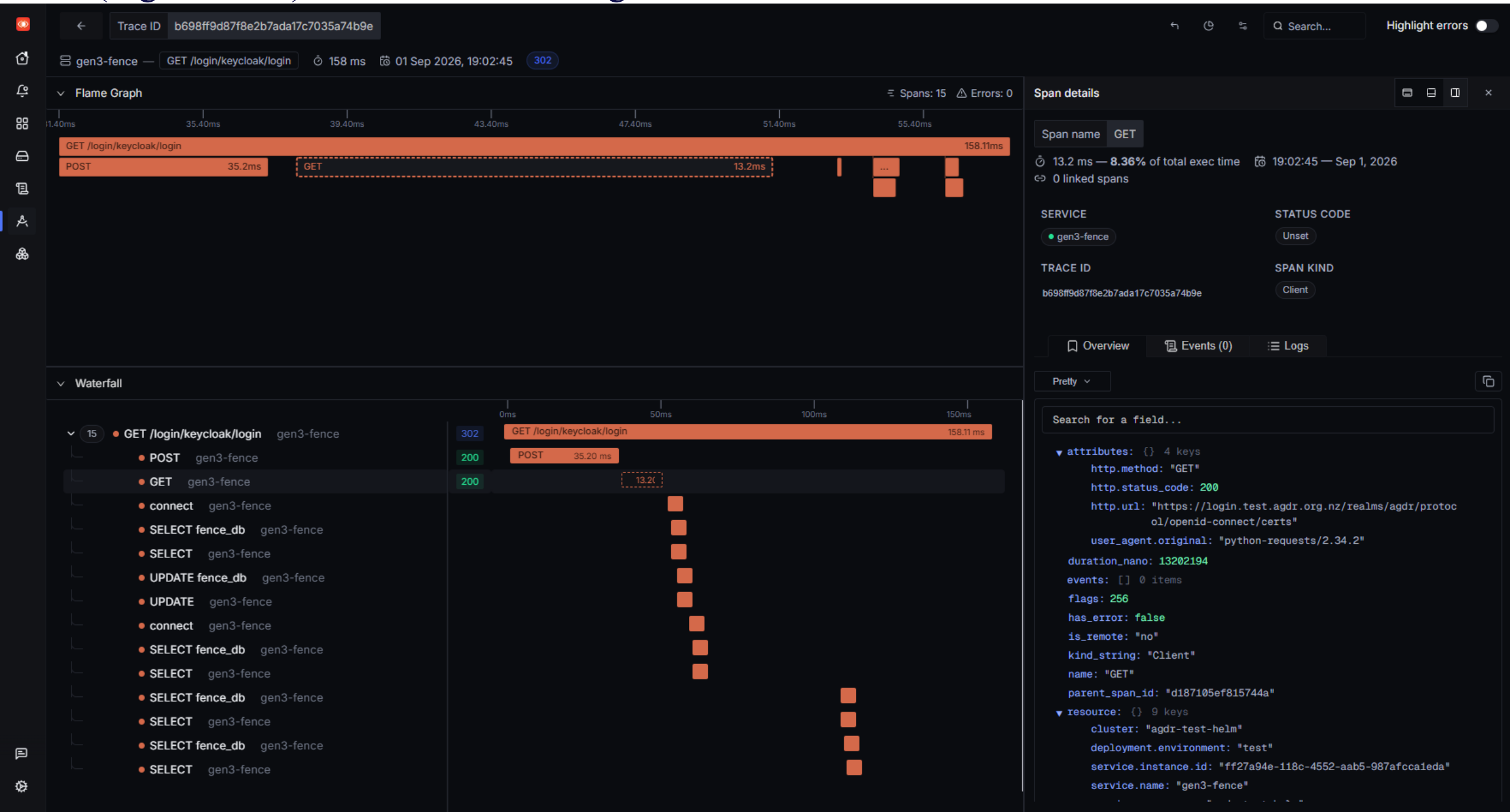
Tracing(Signoz OSS) – OTLP data from gen3 microservice



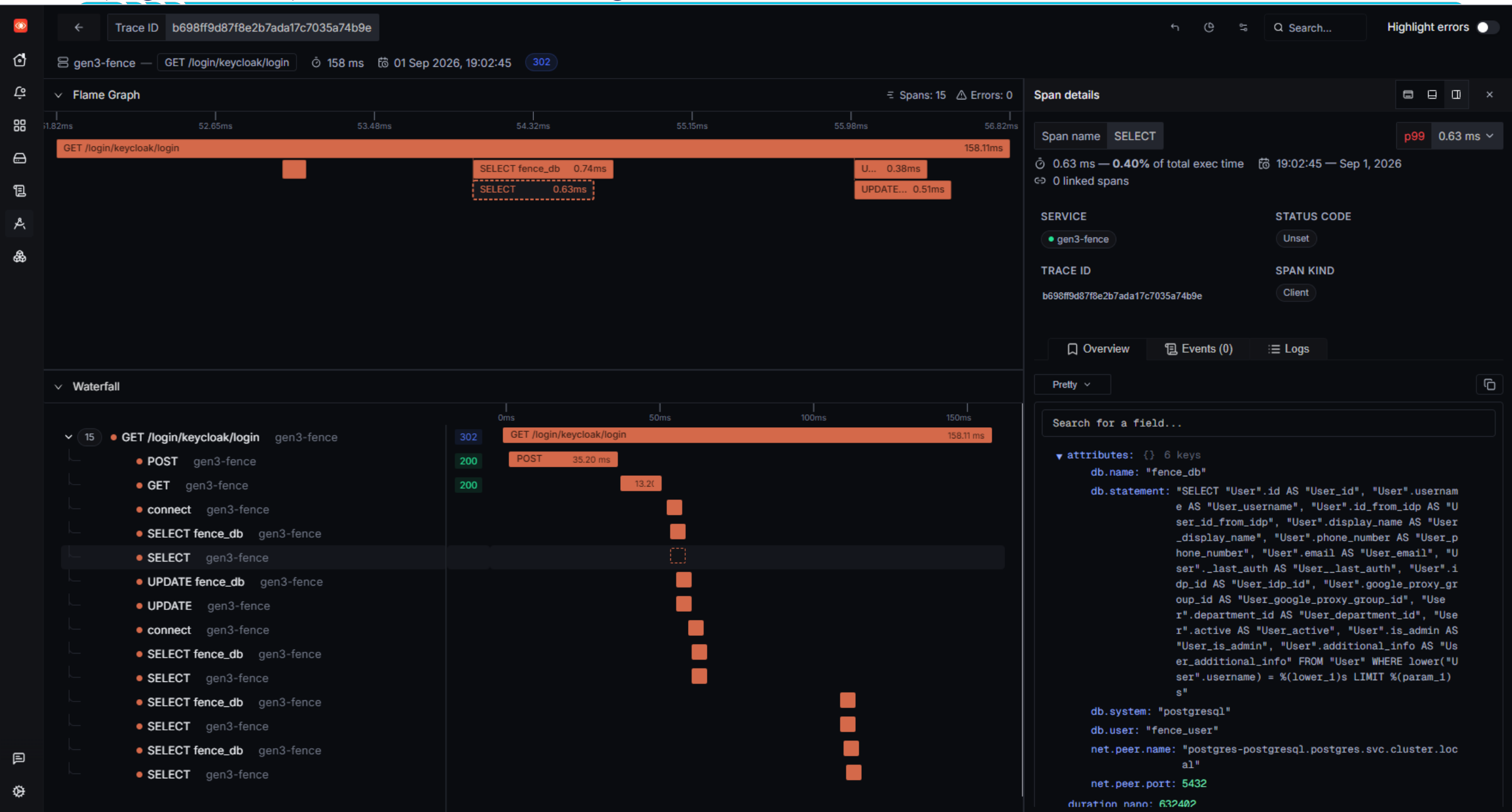
Trace(Signoz OSS) – OTLP data from gen3 microservice



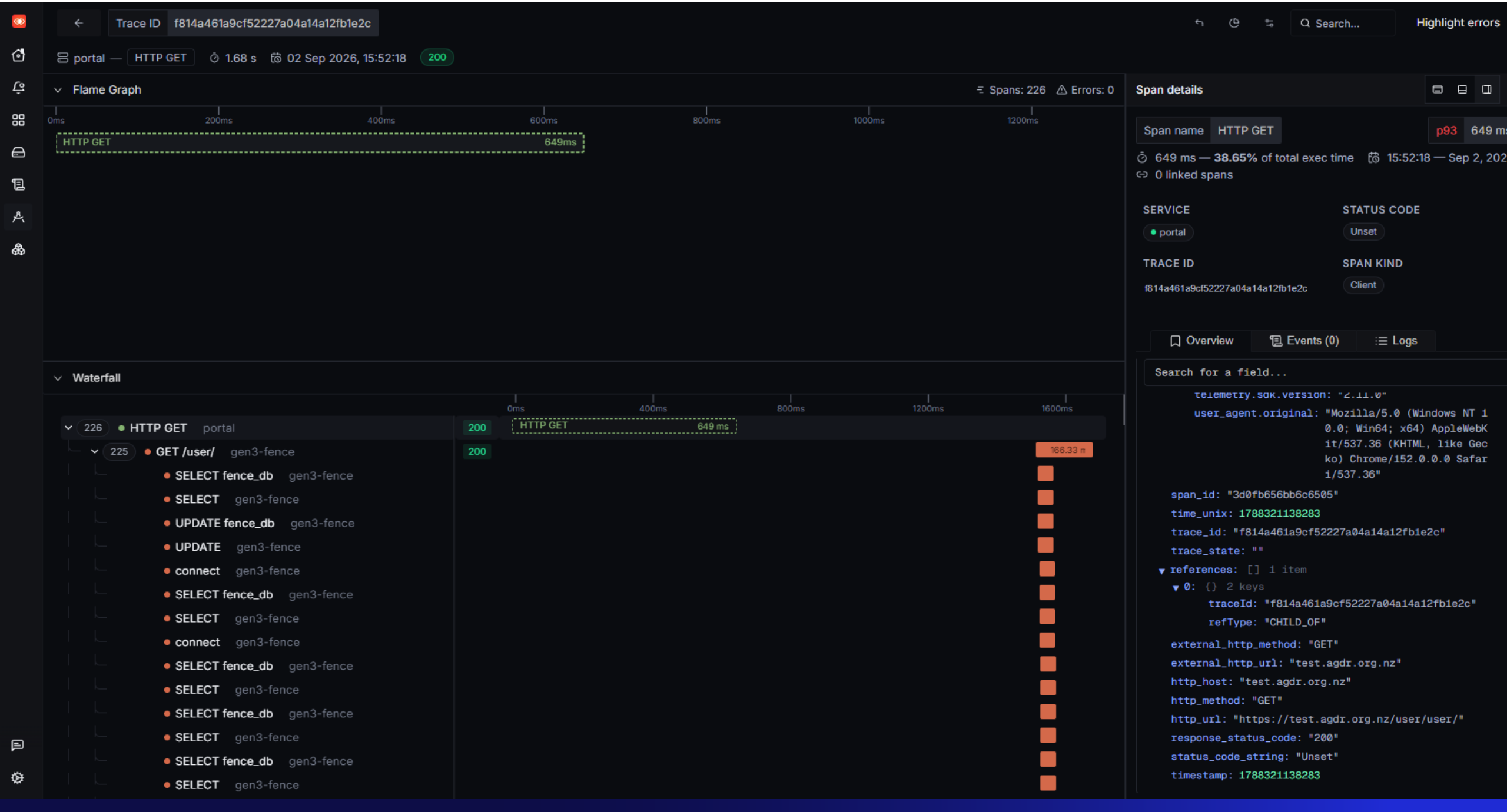
Trace(Signoz OSS) – OTLP data from gen3 microservice



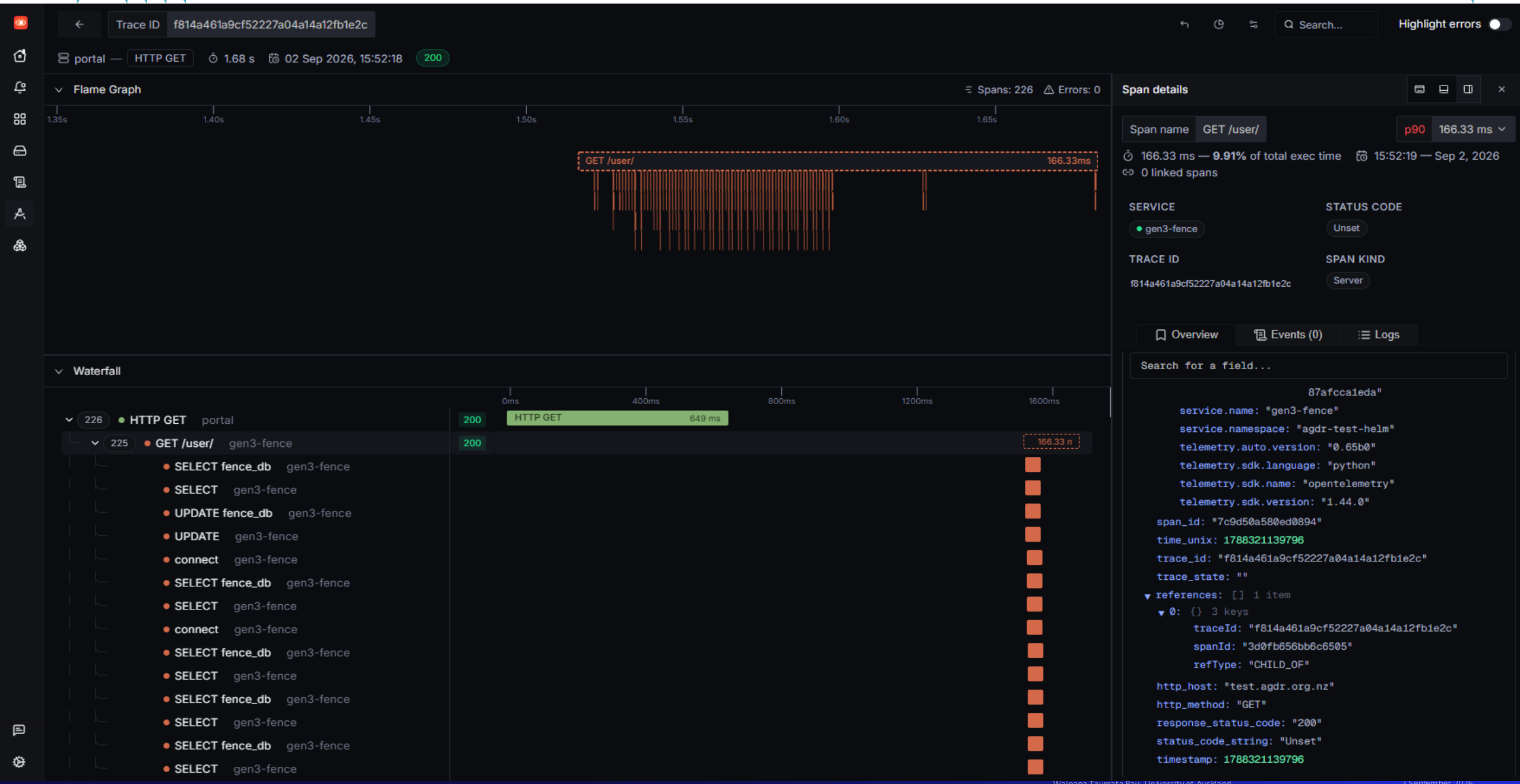
Trace(Signoz OSS) – OTLP data from gen3 microservice



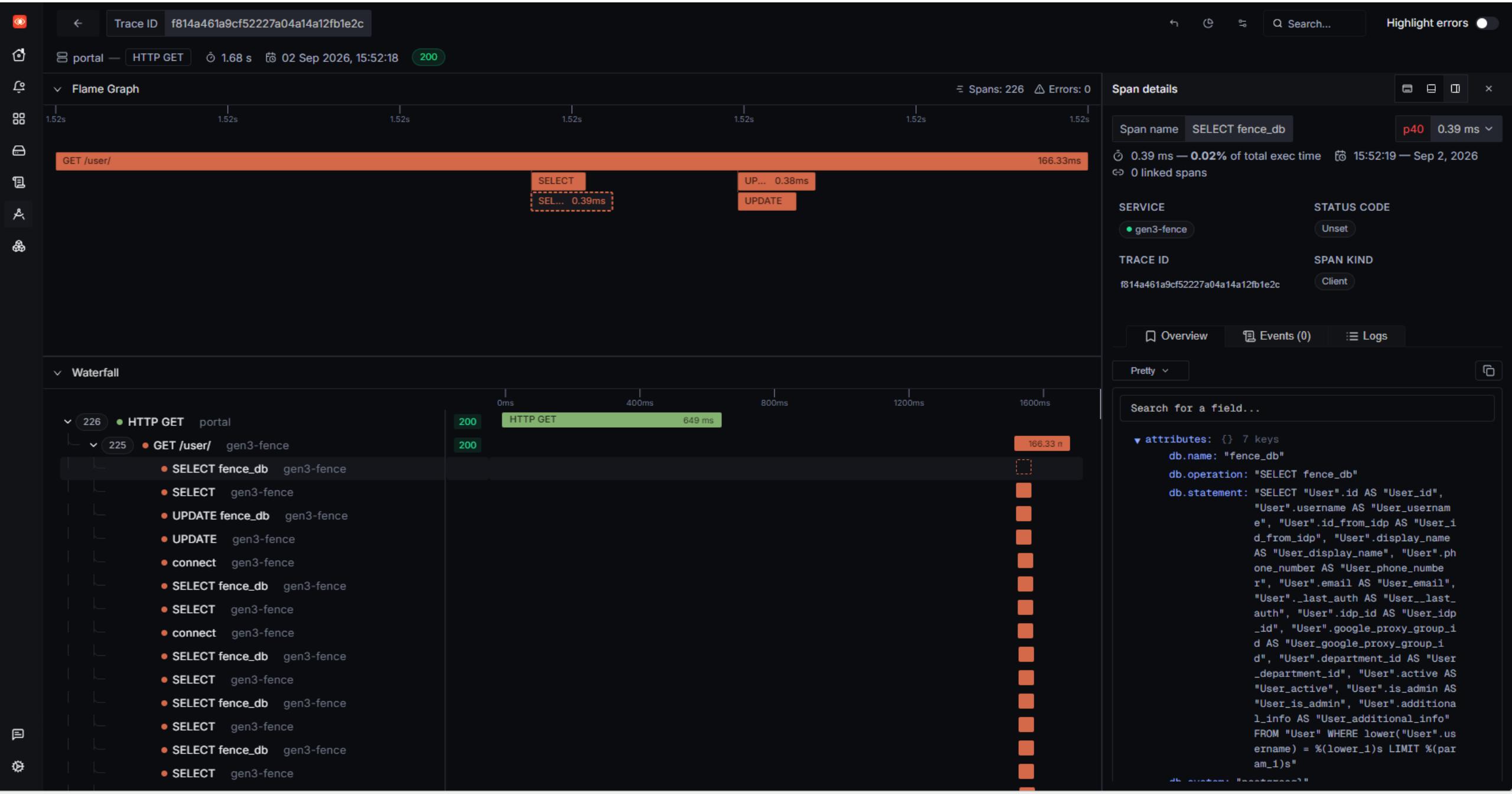
Trace – RUM Data from Faro SDK of user browser



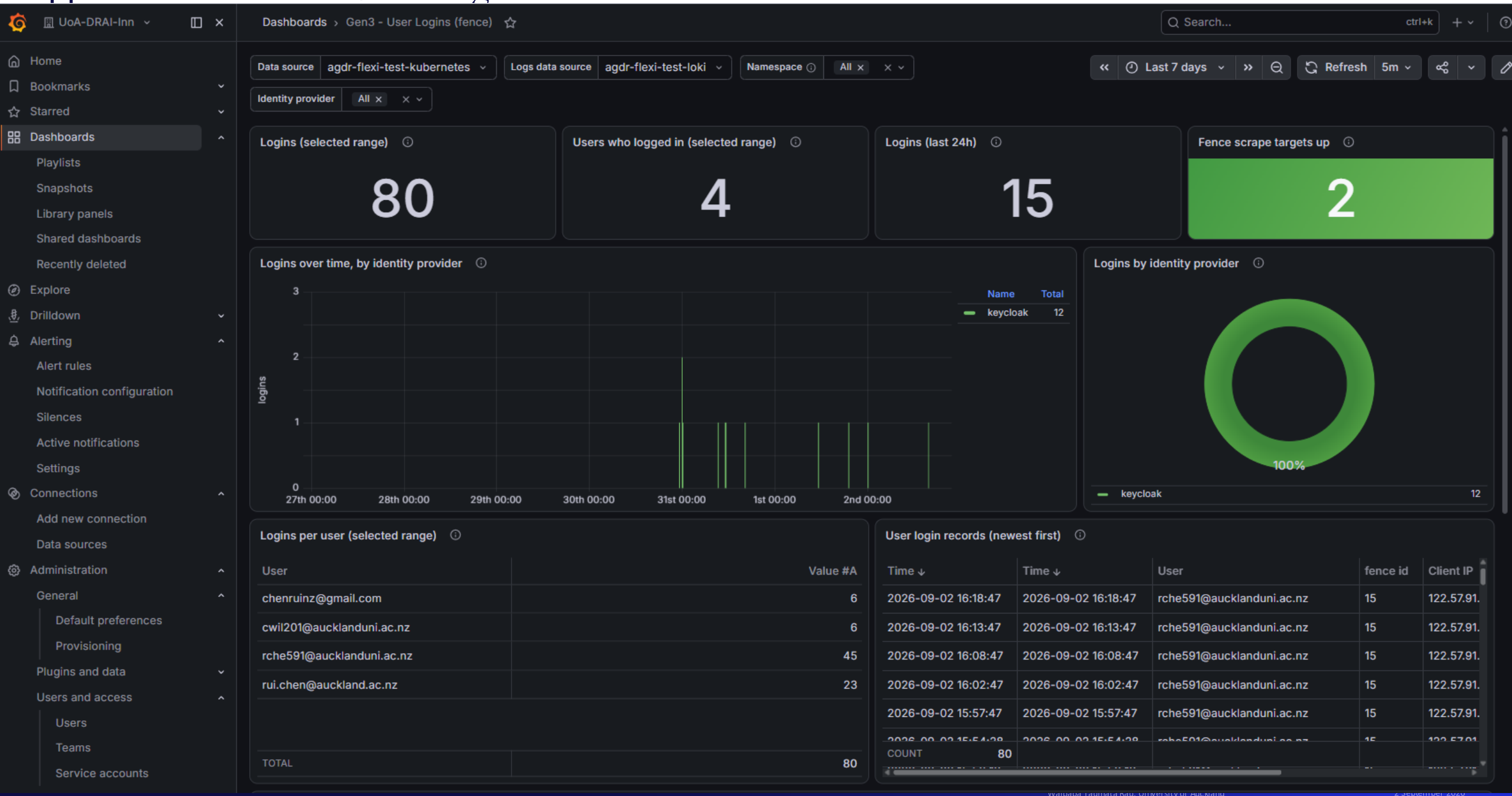
Trace – RUM Data from Faro SDK of user browser

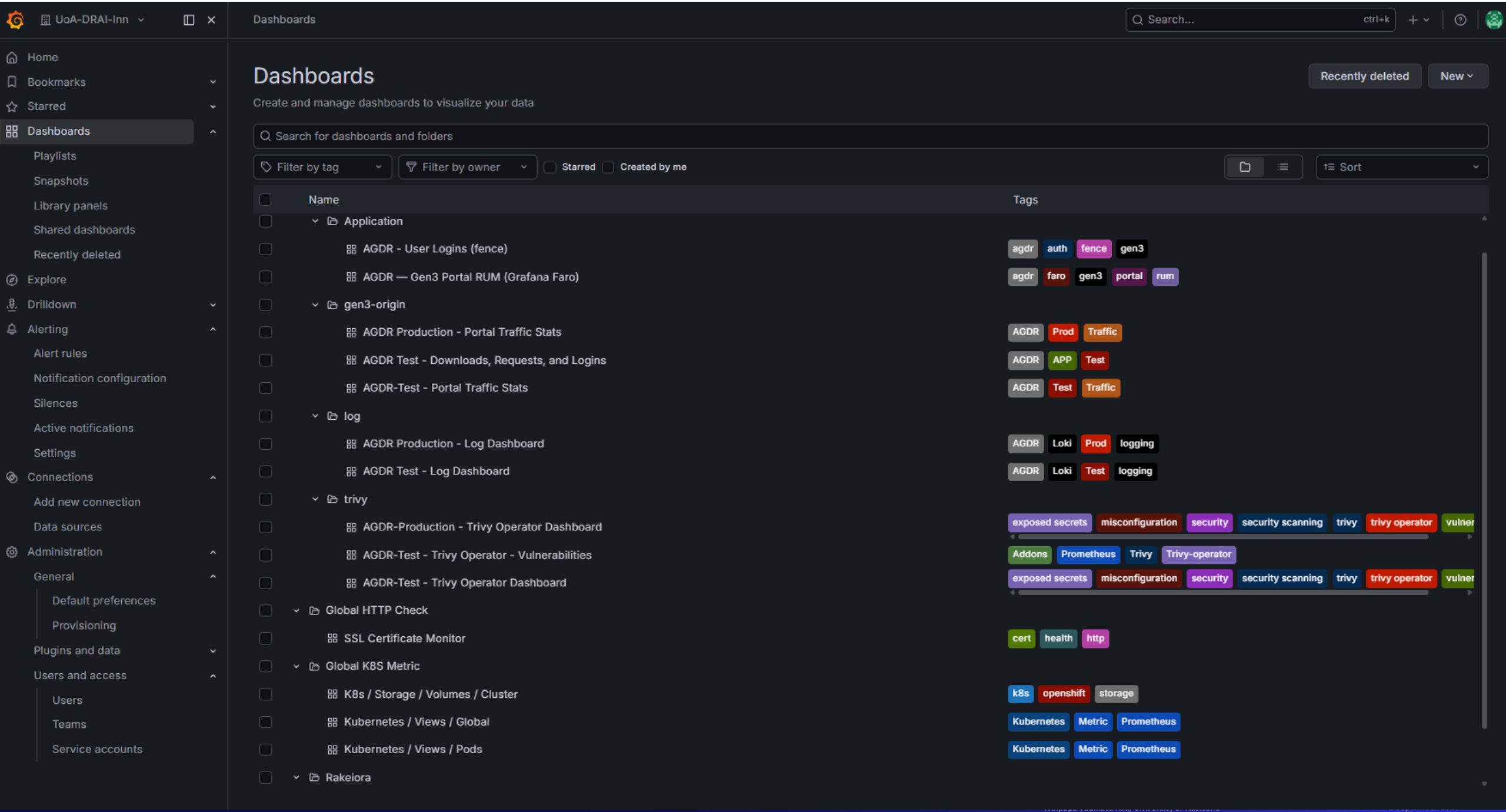


Trace – RUM Data from Faro SDK of user browser



Application monitor: User Login Dashboard




UoA-DRAI-Inn
Dashboards
Search...
ctrl+k
Home
Bookmarks
Starred
Dashboards
Playlists
Snapshots
Library panels
Shared dashboards
Recently deleted
Explore
Drilldown
Alerting
Alert rules
Notification configuration
Silences
Active notifications
Settings
Connections
Add new connection
Data sources
Administration
General
Default preferences
Provisioning
Plugins and data
Users and access
Users
Teams
Service accounts
Dashboards
Create and manage dashboards to visualize your data
Recently deleted
New
Search for dashboards and folders
Filter by tag
Filter by owner
Starred
Created by me
Sort
Name
Tags
Application
AGDR - User Logins (fence)
agdr auth fence gen3
AGDR — Gen3 Portal RUM (Grafana Faro)
agdr faro gen3 portal rum
gen3-origin
AGDR Production - Portal Traffic Stats
AGDR Prod Traffic
AGDR Test - Downloads, Requests, and Logins
AGDR APP Test
AGDR-Test - Portal Traffic Stats
AGDR Test Traffic
log
AGDR Production - Log Dashboard
AGDR Loki Prod logging
AGDR Test - Log Dashboard
AGDR Loki Test logging
trivy
AGDR-Production - Trivy Operator Dashboard
exposed secrets misconfiguration security security scanning trivy trivy operator vulner
AGDR-Test - Trivy Operator - Vulnerabilities
Addons Prometheus Trivy Trivy-operator
AGDR-Test - Trivy Operator Dashboard
exposed secrets misconfiguration security security scanning trivy trivy operator vulner
Global HTTP Check
SSL Certificate Monitor
cert health http
Global K8S Metric
K8s / Storage / Volumes / Cluster
k8s openshift storage
Kubernetes / Views / Global
Kubernetes Metric Prometheus
Kubernetes / Views / Pods
Kubernetes Metric Prometheus
Rakeiora

Dashboard – http & certificate check

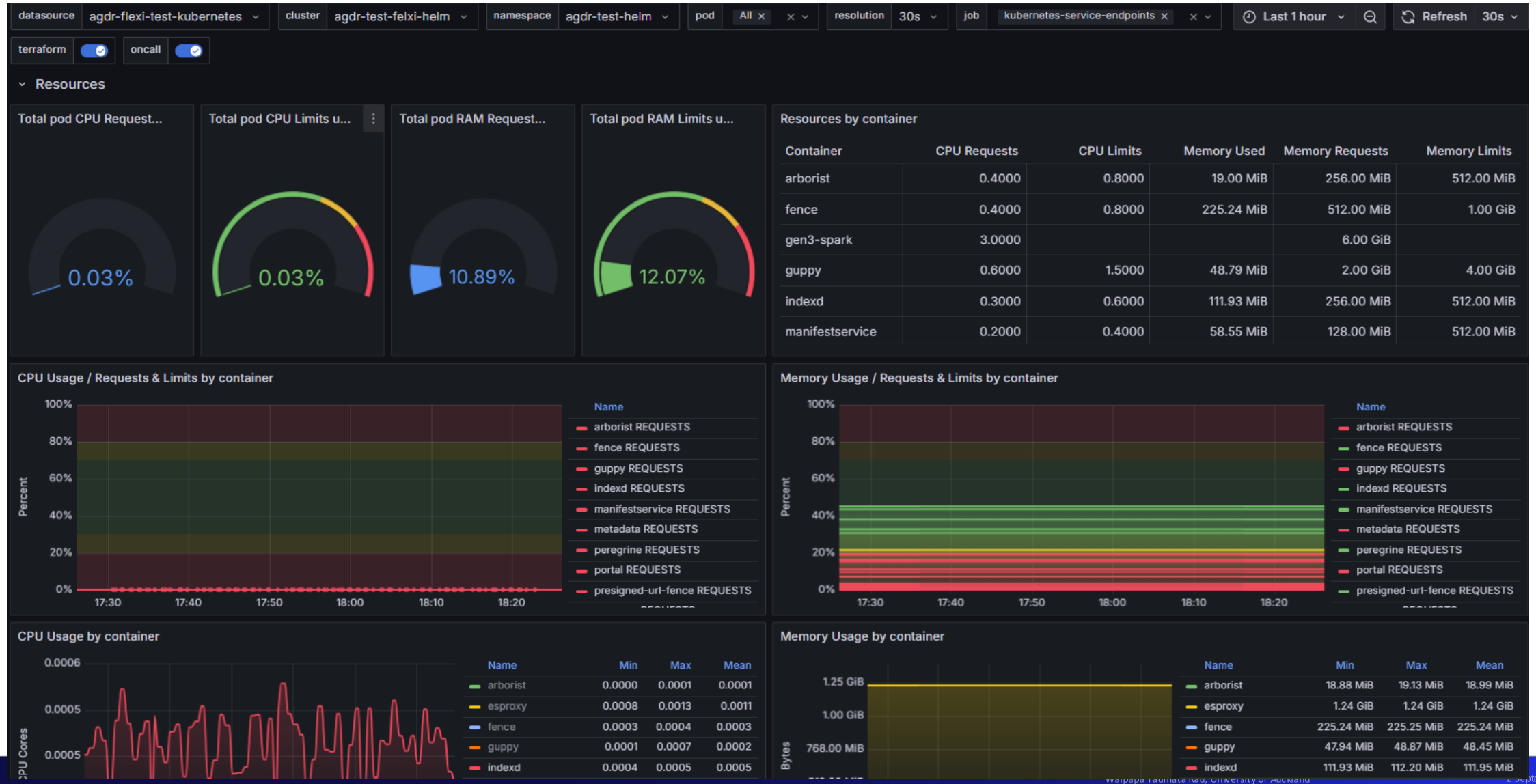
Certificate & Connection Monitoring

Instance	Certificate expires in ↑	HTTP Response	Resolve Time	Connect Time	TLS Time	Processing Time	Transfer Time
https://login.rakeiora.ac.nz	2 weeks, 1 day, 17 hours	200	0.0135		0.0140	0.00645	0.0000895
https://browse.rakeiora.ac.nz	3 weeks, 2 days, 17 hours	200	0.00593		0.00544	0.00246	0.000118
https://internaldata.agdr.org.nz	1 month, 0 weeks, 1 day	200	0.0370		0.00324	0.00216	0.000183
https://internaltest.agdr.org.nz	1 month, 0 weeks, 1 day	200	0.00529		0.00522	0.00191	0.000159
https://login.dev.rakeiora.ac.nz	2 months, 1 week, 2 days	200	2.01		0.0139	0.0120	0.000107
https://browse.dev.rakeiora.ac.nz	2 months, 3 weeks, 2 days	200	0.00565		0.00518	0.00187	0.000147
https://rakeiora.ac.nz	2 months, 3 weeks, 4 days	200	0.00904		0.00399	0.000737	0.000112
https://test.agdr.org.nz	10 months, 1 week, 4 days	200	0.0439		0.0345	0.133	0.000200
https://data.agdr.org.nz	11 months, 0 weeks, 3 days	200	0.00592		0.0355	0.0641	0.000125

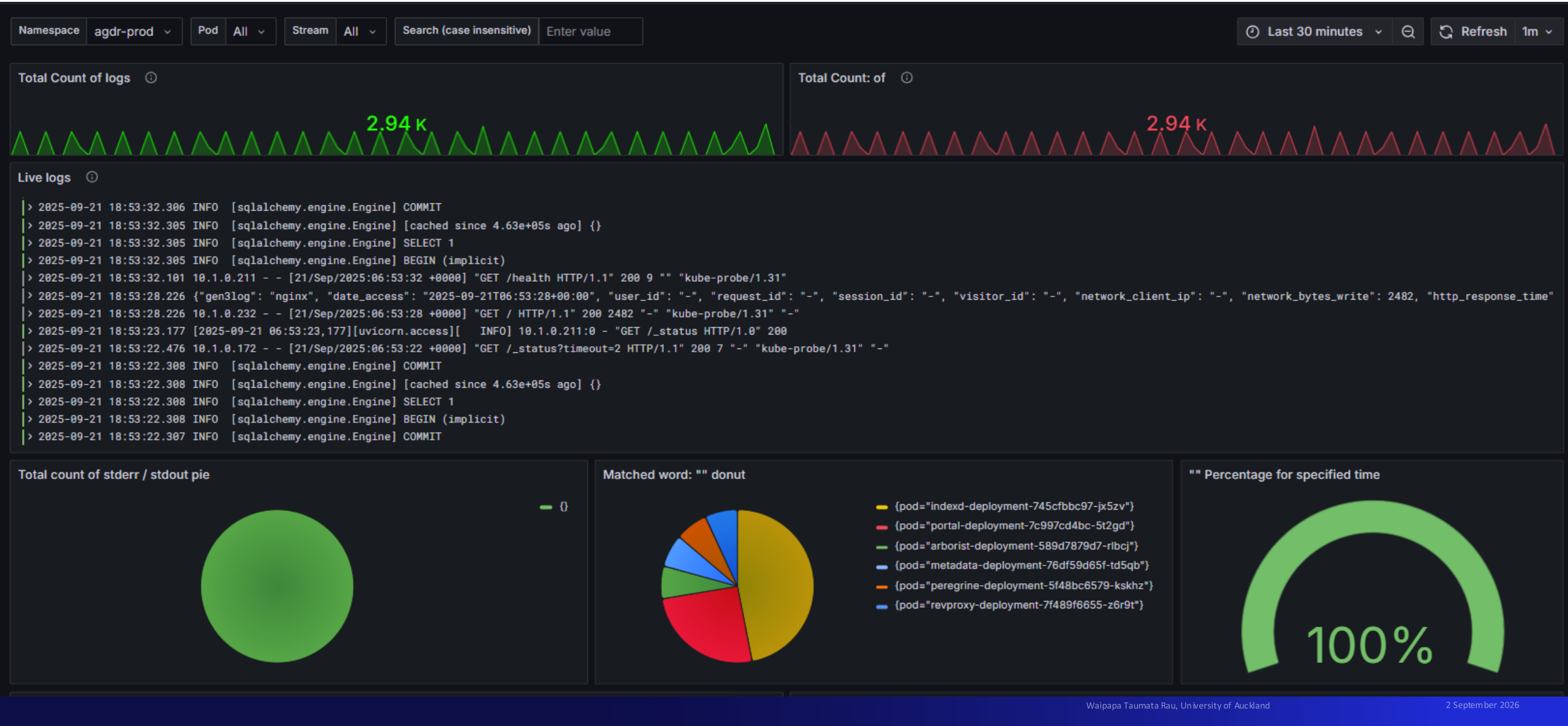
This dashboard use the data from Prometheus blackbox

Dashboard – K8s cluster global monitoring (global, ns, node)

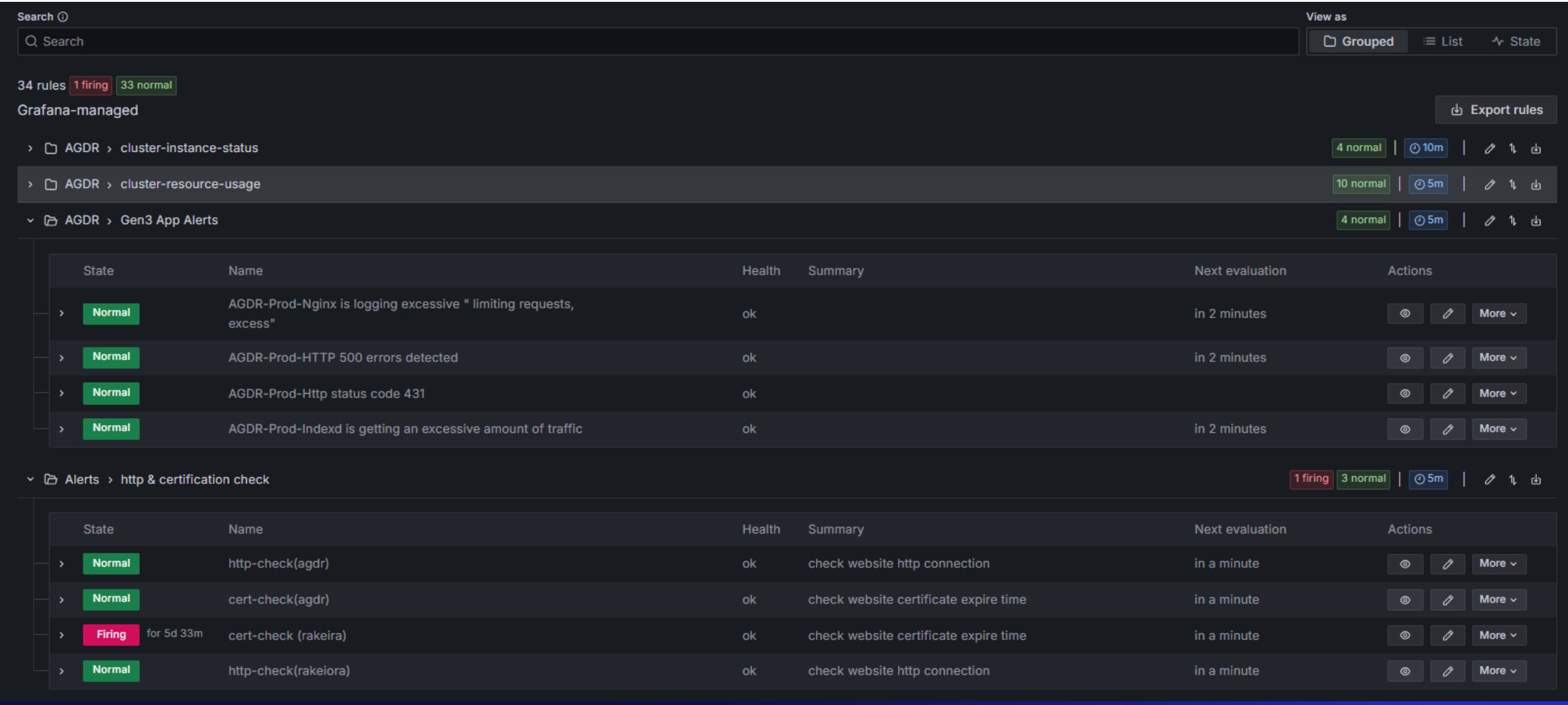
Dashboard – K8s cluster pod monitoring (pod, container)



Dashboard – logs



Alerts


Search
Search
View as
Grouped
List
State
34 rules
1 firing
33 normal
Grafana-managed
Export rules
AGDR › cluster-instance-status
4 normal
10m
AGDR › cluster-resource-usage
10 normal
5m
AGDR › Gen3 App Alerts
4 normal
5m
State
Name
Health
Summary
Next evaluation
Actions
Normal
AGDR-Prod-Nginx is logging excessive " limiting requests, excess"
ok
in 2 minutes
More
Normal
AGDR-Prod-HTTP 500 errors detected
ok
in 2 minutes
More
Normal
AGDR-Prod-Http status code 431
ok
More
Normal
AGDR-Prod-Indexd is getting an excessive amount of traffic
ok
in 2 minutes
More
Alerts › http & certification check
1 firing
3 normal
5m
State
Name
Health
Summary
Next evaluation
Actions
Normal
http-check(agdr)
ok
check website http connection
in a minute
More
Normal
cert-check(agdr)
ok
check website certificate expire time
in a minute
More
Firing
for 5d 33m
cert-check (rakeira)
ok
check website certificate expire time
in a minute
More
Normal
http-check(rakeiora)
ok
check website http connection
in a minute
More

Home
Bookmarks
Starred
Dashboards
Playlists
Snapshots
Library panels
Shared dashboards
Recently deleted
Explore
Drilldown
Alerting
Alert rules
Notification configuration
Silences
Active notifications
Settings
Connections
Add new connection
Data sources
Administration
General
Default preferences
Provisioning
Plugins and data
Users and access
Users
Teams
Service accounts

Clear filters

Rule name
Filter by name...

Labels
Select labels

State
All
Firing
Normal
Pending
Recovering

Folder / Namespace
Select namespace

Evaluation group
Search group

Rule source
All
Grafana managed
Data source managed

Data source
Select data sources

Contact point
Select contact point

Notification policy
Select policy

Type
All
Alert rule
Recording rule

Grafana-managed — Configure

AGDR — Actions

Gen3 App Alerts — 5m — Edit — More

- AGDR-Prod-Nginx is logging excessive " limiting requests, excess" — agdr-flexi-prod-loki — Edit — More
- AGDR-Prod-HTTP 500 errors detected — agdr-flexi-prod-loki — Edit — More
- AGDR-Prod-Http status code 431 — agdr-flexi-prod-loki — Edit — More
- AGDR-Prod-Indexd is getting an excessive amount of traffic — agdr-flexi-prod-loki — Edit — More

cluster-instance-status — 10m — Edit — More

- agdr-flexi-test-pod-status — check cluster's pod status if it is not successful or completed — agdr-flexi-prod-kubernetes — Edit — More
- agdr-flexi-prod-pod-status — check cluster's pod status if it is not successful or completed — agdr-flexi-prod-kubernetes — Edit — More
- agdr-flexi-test-pod-status (reason) — check the reason of cluster's pod's bad status — agdr-flexi-test-kubernetes — Edit — More
- agdr-flexi-prod-pod-status (reason) — check the reason of cluster's pod's bad status — agdr-flexi-prod-kubernetes — Edit — More

cluster-resource-usage — 5m — Edit — More

Global HTTP Check — Actions

http & certification check — 5m — Edit — More

- http-check(agdr) — check website http connection — agdr-flexi-prod-kubernetes — Edit — More
- cert-check(agdr) — Edit — More

Alerts

Check agdr-flexi-test-cpu-usage !
***** Service Monitoring on DRAI k8s cluster *****
🚨 Check resource usage is WARNING! 🚨

Cluster Resource: agdr-flexi-test-cpu-usage
Container: fence
Last usage: 165.21%

Show more
Grafana v11.3.1 | Sep 17th

Check agdr-flexi-test-cpu-usage !
***** Service Monitoring on DRAI k8s cluster *****
🚨 Check resource usage is WARNING! 🚨

this resource usage issue is solved

Cluster Resource: agdr-flexi-test-cpu-usage
Container: fence

Show more
Grafana v11.3.1 | Sep 17th

Rakeiora ops alerts APP 9:59 PM
Check cert-check (rakeira)!
🚨 SSL Certificate Expire Alert! 🚨

The following website(s) certificate may expire in 30 days :

- https://login.rakeiora.ac.nz
- expire in: 17 days

Please update the certificate status immediately.
Grafana v11.3.1 | Yesterday at 9:59 PM

Rakeiora ops alerts APP 2:04 AM
Check cert-check (rakeira)!
🚨 SSL Certificate Expire Alert! 🚨

The following website(s) certificate may expire in 30 days :

- https://login.rakeiora.ac.nz
- expire in: 16 days

Please update the certificate status immediately.
Grafana v11.3.1 | Today at 2:04 AM

backend.tf	S3 remote state, key `agdr-observability/terraform.tfstate`
variables.tf	Inputs — all have defaults
folders.tf	The 4 folders
dashboards.tf	The 14 dashboards, and their uid → folder mapping
dashboards/	Dashboard JSON, one file per dashboard, named after its title
imports.tf	Pass 1: adopt folders + dashboards (18 blocks)
`imports-generate.tf.disabled`	Pass 2: generate config for the rest (41 blocks)

Inventory

Object	Count	Notes
Folders	4	AGDR, Rakeiora, Global HTTP Check, Global K8S Metric
Dashboards	14	8 AGDR, 3 Global K8S, 2 Rakeiora, 1 Global HTTP Check
Data sources	12	prometheus, loki, cloudwatch
Alert rule groups	6	34 rules total
Contact points	14	12 slack, 2 email
Message templates	8	
Notification policy	1	org-wide singleton

Waipapa Taumata Rau University of Auckland

genomics aotearoa

GEN3

Any questions?

Operating Gen3 on AWS

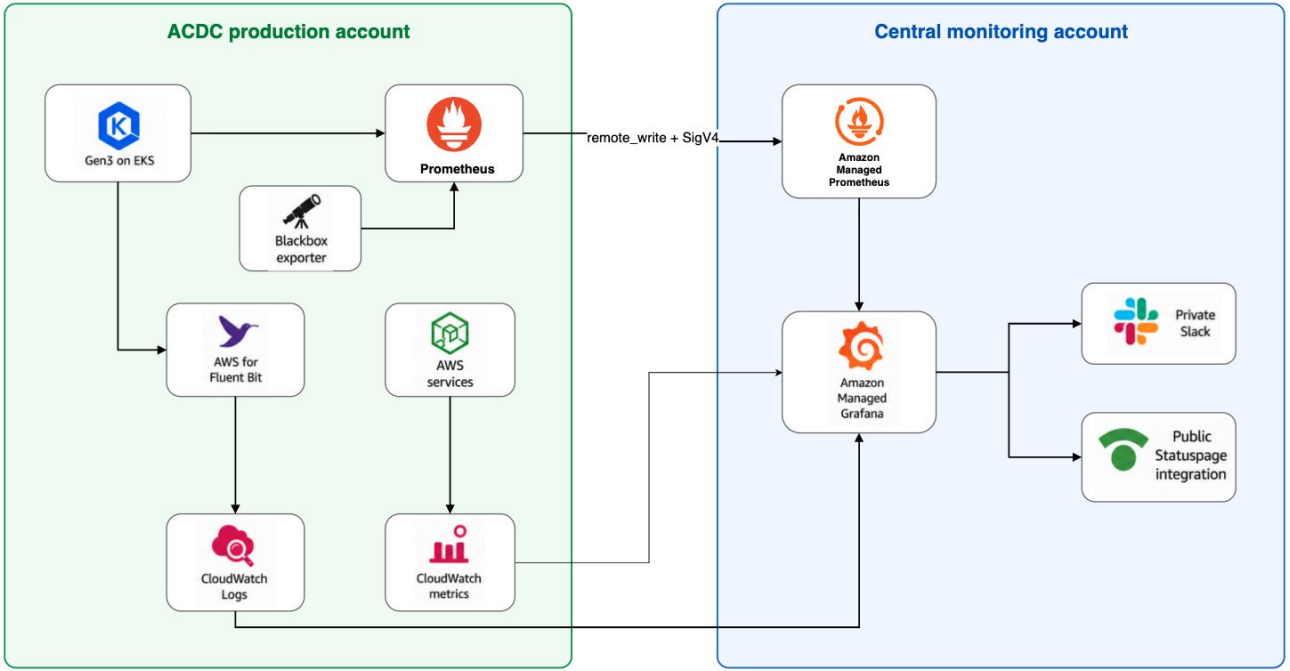
Monitoring and Observability

Guerdon Mukama

Australian BioCommons

Australian BioCommons

The deployed monitoring path



Australian BioCommons

Collecting Kubernetes and Gen3 metrics

```
prometheus:
  server:
    global:
      scrape_interval: 15s
      external_labels:
        cluster: gen3-cad-production
    remoteWrite:
      - url: https://aps-workspaces.ap-southeast-2.amazonaws.com/...
        sigv4:
          role_arn: arn:aws:iam::<monitoring-account>:role/prometheus-cad-
gen3-ingest-role
          region: ap-southeast-2
```

Prometheus is deployed by Argo CD from the production workload repository

Adding AWS metrics and application logs

Grafana uses two data sources for the production view:

Data Source	What we query
AMP	Deployments, pods, nodes, containers, scrape health, service probes and workspaces
CloudWatch	ALB, Aurora, Opensearch and CloudWatch Log insights

The main correlations are:

- ALB request count, target latency, 4xx/5xx and unhealthy targets;
- Kubernetes deployment availability, restarts and resource pressure;
- Aurora CPU/connections and OpenSearch CPU/red state/5xx;
- recent application errors from the same time window.

How the dashboard is implemented

The dashboard is one version-controlled JSON model with four operational rows:

1. Summary: replicas, restarts, workspaces, scrape failures and workspace nodes
2. Kubernetes: node/pod CPU and memory, restarts and deployment availability
3. Service path: blackbox probes, ALB traffic, latency, errors and dependencies
4. activity: workspace use, aggregate authenticated activity and recent errors

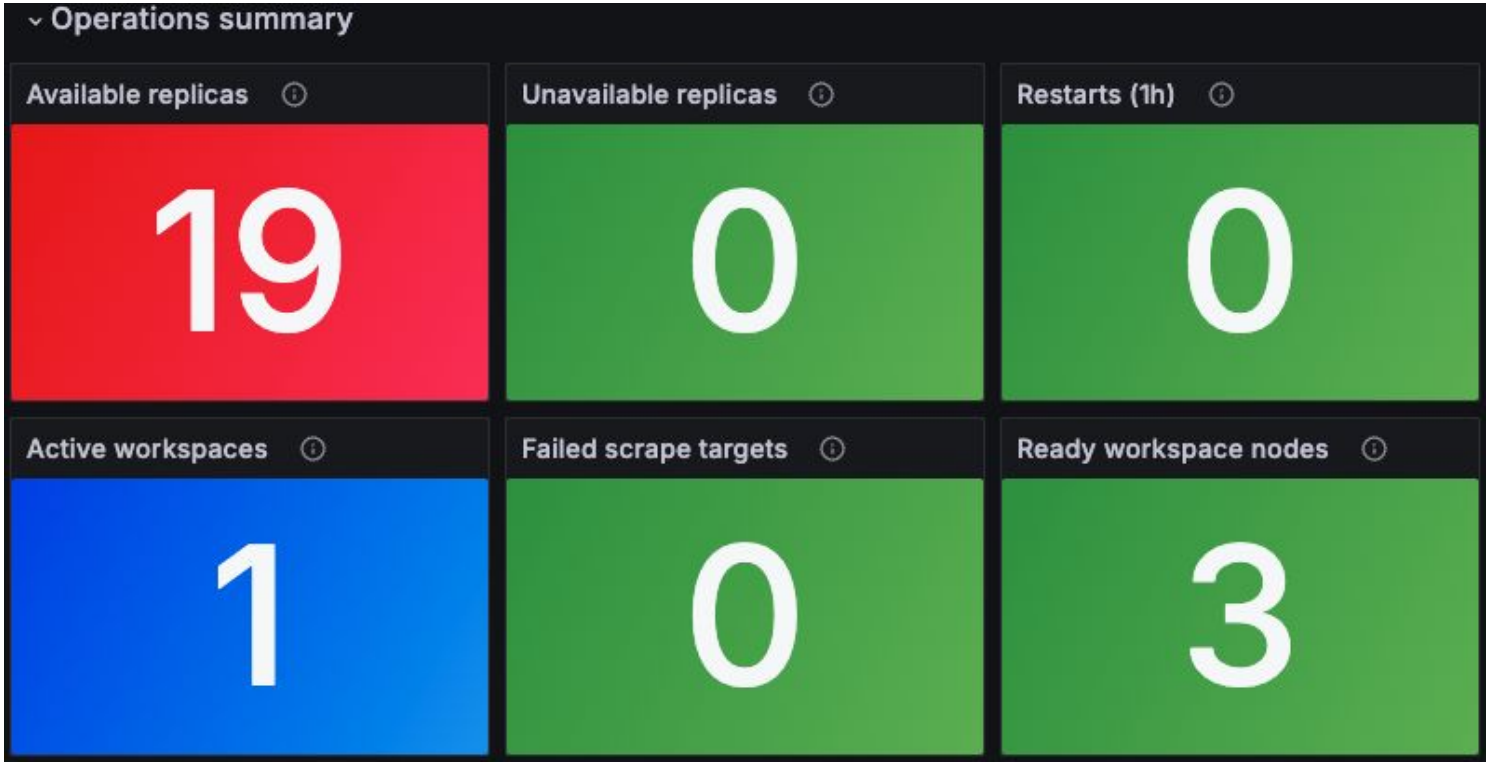
Representative PromQL

```
sum(increase(kube_pod_container_status_restarts_total{
  namespace="prodacdc"
}[1h]))

sum(up{
  cluster="gen3-cad-production",
  job="kubernetes-nodes",
  role="jupyter"
}) or vector(0)
```

Australian BioCommons

Operations summary



Australian BioCommons

Services and API health

Australian BioCommons

Data services

Australian BioCommons

Workspace activity

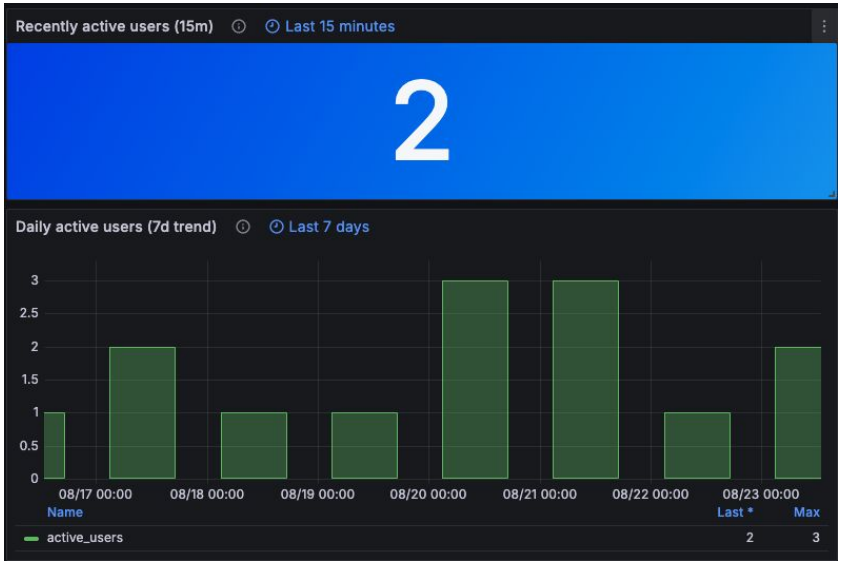
Australian BioCommons

Aggregating user activity at query time

Structured revproxy logs contain an authenticated user_id. Grafana submits a CloudWatch Logs Insights query that returns only the aggregate:

```
filter @logStream like /revproxy[.]revproxy-deployment/
  and ispresent(user_id)
  and user_id != ""
  and user_id != "-"
| stats count_distinct(user_id) as active_users by bin(1d)
| sort @timestamp asc
```

Aggregated user activity



Australian BioCommons

Provisioning Grafana from Git

```
grafana/acdc/
  dashboard.json
  alerts.json
  public-alerts.json
  notification-policy-route.json
  statuspage-policy-route.json
  provision.sh
  README.md
```

```
export GRAFANA_URL="https://<workspace>.grafana-workspace.ap-southeast-2.amazonaws.com"
export GRAFANA_TOKEN="<short-lived-admin-token>"
./grafana/acdc/provision.sh
unset GRAFANA_TOKEN
```


Application log errors

Recent Gen3 application errors

Time	workload	message
2026-08-23 12:58:02.653	fence.fence-visa-update-2608230257-2xnld	Traceback (most recent call last):
2026-08-23 12:58:02.653	fence.fence-visa-update-2608230257-2xnld	[2026-08-23 02:58:02,652][fence.job.access_token_updater][ERROR]...
2026-08-23 12:58:02.791	fence.fence-visa-update-2608230257-2xnld	Traceback (most recent call last):
2026-08-23 12:58:02.791	fence.fence-visa-update-2608230257-2xnld	[2026-08-23 02:58:02,790][fence.job.access_token_updater][ERROR]...
2026-08-23 12:58:03.036	fence.fence-visa-update-2608230257-2xnld	[2026-08-23 02:58:03,035][fence.job.access_token_updater][ERROR]...
2026-08-23 12:58:03.036	fence.fence-visa-update-2608230257-2xnld	[2026-08-23 02:58:03,036][fence.job.access_token_updater][ERROR]...
2026-08-23 12:58:03.036	fence.fence-visa-update-2608230257-2xnld	ERROR:root:backoff: gave up call fence.resources.openid.idp_oauth2.u...
2026-08-23 12:58:03.036	fence.fence-visa-update-2608230257-2xnld	ERROR:backoff:Giving up update_user_authorization(...) after 3 tries (f...

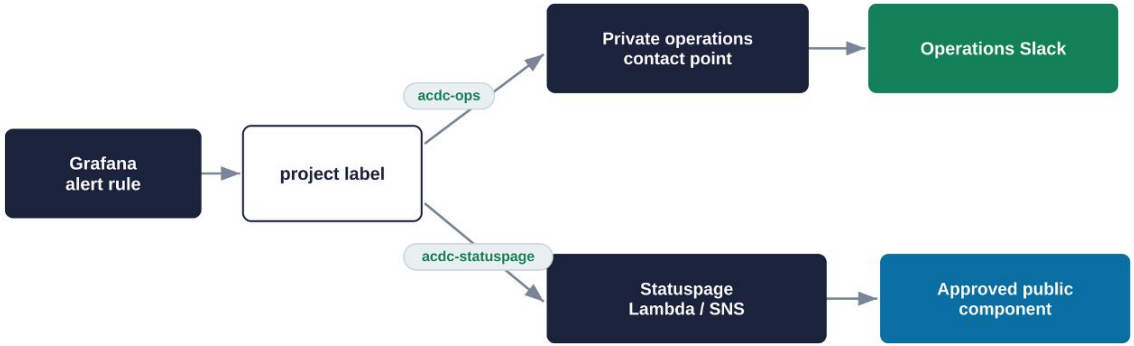
message
Traceback (most recent call last):
ERROR:backoff:Giving up update_user_authorization(...) after 3 tries (fence.errors.AuthError: - User doesn't have a valid, non-expired refresh token
ERROR:root:backoff: gave up call fence.resources.openid.idp_oauth2.update_user_authorization(<fence.resources.openid.idp_oauth2.Oauth2ClientBase obj
[2026-08-23 02:58:24,977][fence.job.access_token_updater][ERROR] Updater 0 could not update authorization for auth0|6a28b8c94496c2a3f5be539d. Error
[2026-08-23 02:58:24,976][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
Traceback (most recent call last):
[2026-08-23 02:58:23,326][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
Traceback (most recent call last):
[2026-08-23 02:58:22,924][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
ERROR:root:backoff: gave up call fence.resources.openid.idp_oauth2.update_user_authorization(<fence.resources.openid.idp_oauth2.Oauth2ClientBase obj
[2026-08-23 02:58:22,604][fence.job.access_token_updater][ERROR] Updater 0 could not update authorization for auth0|699541aabfba8b59aab00373. Error
ERROR:backoff:Giving up update_user_authorization(...) after 3 tries (fence.errors.AuthError: - User doesn't have a valid, non-expired refresh token
Traceback (most recent call last):
[2026-08-23 02:58:22,603][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
[2026-08-23 02:58:20,655][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
Traceback (most recent call last):
[2026-08-23 02:58:20,370][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
Traceback (most recent call last):
[2026-08-23 02:58:18,480][fence.job.access_token_updater][ERROR] Could not refresh token: - User doesn't have a valid, non-expired refresh token
ERROR:root:backoff: gave up call fence.resources.openid.idp_oauth2.update_user_authorization(<fence.resources.openid.idp_oauth2.Oauth2ClientBase obj

Alert evaluation

Private evaluation rules and pending durations

Private rule	Pending
Expected deployment object missing	1 minute
Critical deployment unavailable	5 minutes
Monitoring target down	10 minutes
Container restart storm	5 minutes
Node CPU above threshold	15 minutes
Workspace node capacity unavailable	10 minutes

Alert routing



- Private routing stops after the ACDC operations match.
- Only portal, data-management and requestor availability rules carry the public project label.
- Public rules wait 10 minutes and map to explicit Statuspage component IDs.
- Public rules treat query errors / no data as OK; the private scrape-health rule reports telemetry failures

Handling an absent Kubernetes object

Our first availability expression only evaluated deployments that existed. Deleting Hatchery removed its metric series, and Argo CD restored the object in about 182 seconds - before the five-minute alert became pending.

```
count by (deployment) (
  kube_deployment_created{
    namespace="prodacdc",
    deployment=~"portal-deployment|revproxy-deployment|fence-deployment|..."
  }
)
or on (deployment) (
  label_replace(vector(0), "deployment", "portal-deployment", "", "")
  or label_replace(vector(0), "deployment", "revproxy-deployment", "", "")
  or label_replace(vector(0), "deployment", "fence-deployment", "", "")
)
```

- Missing objects now retain a zero-valued series.
- A one-minute warning records deletion / reconciliation events.
- A separate five-minute critical rule covers sustained zero availability.

How we validate a monitoring change

1. Review JSON and query changes in Git.
2. Run the idempotent provisioner with a short-lived token.
3. Confirm dashboard panels return data from both data sources.
4. Inspect panel data to ensure activity results contain aggregates only.
5. Confirm all production rules are Normal with expected labels, pending durations and runbook URLs.
6. Exercise notification routing with a temporary synthetic private rule.
7. Change the synthetic expression to healthy, wait for resolution, then delete it.
8. Verify no production condition or public route was touched.

Implementation choices and next steps

What is working

- Prometheus and CloudWatch stay separate backends, joined in Grafana.
- Ingestion uses IRSA, cross-account role assumption and SigV4.
- Argo CD owns collection; Git and the Grafana API own presentation and alerting.
- Expected-object fallbacks make deletion observable.
- Route labels separate operations notifications from public status.
- User activity is aggregated in Logs Insights before Grafana.

Next implementation gaps

- A credentialed synthetic for the full workspace-launch journey.
- A consistent application-metrics baseline across Gen3 services.
- Reusable dashboard and alert templates for more Gen3 deployments