docs-meeting-pre-27082025

1. Meeting Information

Date: 27 August 2025
Time: 1:00p.m. - 2:00p.m.
Estimated Time: 1 hour

1.1. Objective

1. Raise concerns regarding the Security Module
2. Share findings related to the Security Module
3. Present possible enhancements for the current codebase

1.2. Expected Outcome

1. Determine the next steps for the Security Module
2. Define the next steps in Zhi Yang's work plan

1.3. Abstract

This document provides an overview of potential enhancements and modules to achieve a highly available, secure, and maintainable codebase.

2. Security

2.1. Main Concerns

2.1.1. Lack of clear, well-defined security requirements

There are many measures and libraries available for security.

From an industry perspective, frameworks such as the **Financial Grade API (FAPI)** can be used as a benchmark. In practice, we also have multiple tools and frameworks to choose from:

- **Spring Security** for overall security protection
- **OPA, OpenFGA, Keycloak, Okta, or Spring Security** for Fine-Grained Authorization (FGA)

- **Spring Cloud Gateway, Nginx, or Apache APISIX** for gateway-level security

However, the challenge lies in making decisions about:

- Which measures should be implemented?
- Which libraries should we adopt?

Answering these requires clarity on:

1. What are the current security requirements (either explicitly required by the business unit, or implicitly raised by IT after analysis)?
2. What is the current state of the project and the broader objectives? Do we have the necessary resources? To what extent should we implement security measures now?
3. Which measures should be prioritized immediately, and which can be deferred?

Without these answers, planning a security solution remains difficult.

2.1.2. High Complexity in Cybersecurity

Setting up security in Spring Boot often does not involve heavy coding like implementing a functional requirement. For example, the following simple configuration can enable:

1. CSRF protection
2. Restriction of all APIs to authenticated users

```
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(Customizer.withDefaults())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/**").permitAll()
            .anyRequest().authenticated()
        )
}
```

The real complexity lies in **what** to configure and **how** to configure it. To address this, one must:

- **Know what should be configured**
    - Have a strong foundation in cybersecurity to understand current threats and mitigation strategies

    - Understand the codebase and requirements deeply to make appropriate trade-offs
    - Be proficient with Spring Security (and possibly other frameworks) to achieve the desired configuration
- **Verify the correctness of configurations**
    - Possess a solid understanding of security mechanisms to reason about their safety
    - Know how to perform proper testing and validation
- **Minimize the impact on daily development**
    - Choose the most suitable libraries after evaluating their differences and capabilities
    - Have deep knowledge of the chosen libraries, including advanced customization
    - Design for maintainability, clean architecture, flexibility, and developer usability to support future requirements

2.1.3. Summary

Security should be treated as a **dedicated module** rather than a simple requirement because it demands:

- Strong domain knowledge in cybersecurity (awareness of mechanisms and threats)
- A solid understanding of the system design and whether proposed solutions fit the system
- Deep practical experience with security frameworks (e.g., Spring Security)
- An understanding of project requirements to balance trade-offs between usability, maintainability, and extensibility

These are complex topics that cannot be resolved in a single meeting. The goal of this discussion is not to finalize decisions, but to **raise the concerns** and determine the **next steps**.

> ⚠ **Quick Test of Required Knowledge**
>
> To highlight the breadth of knowledge needed, consider these questions:
>
> - What is OAuth 2.0, and how does it ensure authentication?
> - What are the authorization server, client, and resource server in OAuth 2.0? How can these servers be configured to meet our requirements using Spring Security and Spring Authorization Server?

- Should we use Spring Authorization Server and build everything ourselves, or rely on third-party services (e.g., Keycloak, Okta)? Why or why not?
- What is token exchange, and how can it be applied within an OAuth 2.0 + Spring Boot context?
- Is OAuth 2.0 sufficient for our codebase, or should we also consider standards like PAR, JAR, JARM, DPoP, and PKCE?

Closing Note

The following research details will outline the capabilities of Spring Security, gateways, and other potential security measures. This document will also highlight possible enhancements for the current codebase.

2.2. Spring Security

Resources:

1. GitHub - spring-projects/spring-security-samples at 6.5.x
2. Spring Security :: Spring Security

Terms	Explanation
AuthN	Authentication
AuthZ	Authorization
FAPI	Financial grade API
RFC	Request for Comments

Introduction

1. Spring Security Architecture Principles by Daniel Garnier-Moiroux @ Spring I/O 2024 - YouTube

Financial Grade API (FAPI) is a set of rules built on top of OAuth2.0 and OpenID Connect (OIDC) to protect financial data and transactions exposed via APIs.

- We should ensure our APIs that are exposed to third parties to follows FAPI
- Use subset of FAPI as a guideline to protect our API. We might not need the full FAPI checklist

FAPI

- FAPI 2.0 Security Profile
- FAPI Working Group – OpenID Foundation
-

2.2.1. Authentication

Terms	Explanation
OAuth2.0	The industry-standard protocol for authorization. - Information on RFC 6749 » RFC Editor - RFC 6749 - The OAuth 2.0 Authorization Framework
OIDC	OpenID Connect OIDC is a a protocol (built on top of OAuth2) for federated authentication. OIDC is the protocol and token format that tells you the user's identity.
PAR	**Pushed Authorization Request** Pushed Authorization Requests. Instead of sending all authorization parameters through the browser/front channel (redirect URI), the client sends them directly to the Authorization Server over a secure backchannel.
JAR	JWT-Secured Authorization Request
JARM	JWT Secured Authorization Response Mode
DPoP	Demonstrating Proof of Possession RFC 9449 - OAuth 2.0 Demonstrating Proof of Possession (DPoP)
PKCE	Proof Key for Code Exchange RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients
JWT	JSON Web Token RFC 7515 - JSON Web Signature (JWS)
JWKS	JSON Web Key Sets
JWK	JSON Web Key
JOSE	Javascript Object Signing and Encryption
CS	Client Server. A role in OAuth2.0, specifying the server that requests for resources.
AS	Authorization Server. A role in OAuth2.0
RS	Resource Server
MTLS	Mutual Transport Layer Security RFC 8705 - OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

Authentication

- Servlet Authentication Architecture :: Spring Security

OAuth2.0

- OAuth2.0: OAuth2 :: Spring Security
- OAuth2.0 Auth: Spring Authorization Server Reference :: Spring Authorization Server
- OAuth2.0 Client: OAuth 2.0 Client :: Spring Security
- OAuth2.0 Resource: OAuth 2.0 Resource Server :: Spring Security

Password Storage & Encryption

- Password Storage :: Spring Security

Others

- Illustrated DPoP (OAuth Access Token Security Enhancement) | by Takahiko Kawasaki | Medium

2.3. Authorization

Terms	Explanation
RBAC	(**IMPORTANT**) Role-based Access Control
ABAC	(**IMPORTANT**) Attribute-based Access Control
ReBAC	Relationship-based Access Control

Conclusion

> - Use Spring Security + DB-based access policy control for the maximum flexibility
> - Other options: OPA, Keycloak

2.3.1. `@EnableMethodSecurity`

```
@Component
public class BankService {
    //
}
```

```
@Component
public class BankService {

}
```

```
@PreAuthorize("hasRole('ADMIN')")
public Account readAccount(Long id) { }

@PostAuthorize("returnObject.owner == authentication.name")
public Account readAccount(Long id) {}

@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME)
@PostAuthorize("returnObject.owner == authentication.name")
public @interface RequireOwnership {}

@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME) @PreAuthorize("hasRole('{value}')")
public @interface HasRole { String value(); }

@Target({ ElementType.METHOD, ElementType.TYPE })
@Retention(RetentionPolicy.RUNTIME) @PreAuthorize("hasAnyRole({roles})")
public @interface HasAnyRole { String[] roles(); }

@PreAuthorize("@authz.decide(#root)")
@GetMapping("/endpoint")
public String endpoint() { // ... } }
```

Introduction

1. Authorization Architecture :: Spring Security
2. Authorize HttpServletRequests :: Spring Security
3. Method Security :: Spring Security

References:

1. Authorization in Spring Security: permissions, roles and beyond by Daniel Garnier-Moiroux @Spring IO - YouTube

2.3.2. Other Considerations

- Is the policies change VERY frequently? If yes, we should consider to use OPA instead of using `@PreAuthorize` provided by Spring Security.
- Do we need a workflow engine for the BU to claim the roles/ rights by themselves? If yes, we might need to consider MakerChecker/Flowable/Camunda/Activiti.

2.4. Others

Terms	Explanation
CSRF	Cross-Site Request Forgery
CORS	Cross Origin Resource Sharing
XSS	Cross-Site Scripting
CSP	Content Security Policy
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart

- Overall: Protection Against Exploits :: Spring Security
- **CSRF**
  - Introduction: Cross Site Request Forgery (CSRF) :: Spring Security
  - Implementation:Cross Site Request Forgery (CSRF) :: Spring Security
- **Header Protection**
  - Keyword: CSP, Referrer-Policy, X-Content-Type-Options, Permission-Policy, HSTS
  - Introduction: Security HTTP Response Headers :: Spring Security
  - Implementation: Security HTTP Response Headers :: Spring Security
- `HttpFirewall` : HttpFirewall :: Spring Security
- Others:
  - Spring MVC Integration :: Spring Security
  - Anti-automation through CAPTCHA
  - Login throttling/backoff and lockout
  - OWASP libraries

2.5. Spring Cloud Gateway

Terms	Explanation
DOS	Denial of Service
DDOS	Distributed Denial of Service
SCG	Spring Cloud Gateway

> Note:
> It is not necessarily to use Spring Cloud Gateway, but there must be a mechanism for handling rate limiting to prevent DOS and

> mitigate DDoS. Others like retry, circuit breaker, load balancing, etc. might be considered. The decision of including a gateway or not depends on the project size, architecture, requirements, etc and much more factors.

Capabilities provided by Spring Cloud Gateway (can be used as a references to a typical Gateway configurations):

1. **Routing**
   1. **What it does**: Matches incoming requests (by path, host, method, headers, query params, time windows) and forwards them to the right downstream service.
   2. **Why it matters**: Prevents accidental exposure of internal services; lets you enforce a single "front door" for policy, auth, and inspection. Reduces blast radius during incidents by quickly disabling or rerouting traffic (e.g., to a maintenance page).
   3. **Libraries/ Config**
      1. Spring Cloud Gateway core (predicates & filters)
      2. Service discovery: Spring Cloud LoadBalancer, Netflix Eureka, HashiCorp Consul, Kubernetes DNS
2. **Request Rate Limiting**
   1. **What it does:** Caps how many requests a caller can make per time window (per IP, API key, user, etc.).
   2. **Why it matters:** Throttles abusive traffic and mitigates **DDoS / credential stuffing / brute force** and runaway clients; protects downstream capacity.
   3. **Libraries / config:**
      - **SCG RedisRateLimiter** (built-in) → needs Redis
      - **Bucket4j** (token bucket) via Spring Boot starters; can store counters in-memory, Redis, Hazelcast, JCache, etc.
      - Optional: Resilience4j RateLimiter (service-level)
3. **Add/Remove/Rewrite Request/Response Headers**
   1. **What it does:** Normalizes or enriches requests and responses (e.g., add `X-Request-Id`, strip hop-by-hop headers, rewrite `Location`, set `Cache-Control`).
   2. **Why it matters:** Prevents **header injection**, **host/header spoofing**, and **cache poisoning**; standardizes telemetry; hides internal topology.
   3. **Libraries / config:**
      - SCG built-in filters: `AddRequestHeader`, `RemoveRequestHeader`, `SetRequestHeader`, `AddResponseHeader`, `RemoveResponseHeader`,

    `SetResponseHeader` , `RewriteLocationResponseHeader` , `DedupeResponseHeader`
  - For paths/bodies: `RewritePath` , `ModifyRequestBody` , `ModifyResponseBody`

4. **Circuit Breaker**
   1. **What it does:** Trips open when a downstream is failing, short-circuits calls temporarily, and optionally returns a fallback.
   2. **Why it matters:** Stops **cascading failures**, protects the gateway and other services when a dependency is unhealthy; good against **slowloris/backpressure** scenarios from failing backends.
   3. **Libraries / config:**
      - Spring Cloud CircuitBreaker abstraction
      - **Resilience4j** (recommended impl)
      - Fallbacks via SCG `CircuitBreaker` filter
5. **Retry**
   1. **What it does:** Automatically retries idempotent calls on transient errors (e.g., timeouts, 502/503/504).
   2. **Why it matters:** Smooths over **flaky networks**, **pod restarts**, and **cold starts** without user-visible errors; must be tuned to avoid retry storms.
   3. **Libraries / config:**
      - SCG `Retry` filter (supports methods, statuses, backoff)
      - Optional: Spring Retry / Resilience4j Retry for service-level logic
6. **Request Size Limits**
   1. **What it does:** Rejects oversized requests early (e.g., large JSON bodies, giant file uploads).
   2. **Why it matters:** Mitigates **DoS via oversized payloads**, memory pressure, and long GC pauses in downstream services.
   3. **Libraries / config:**
      - SCG `RequestSize` filter
      - Reactor Netty server/client properties (e.g., max header size)
      - Spring WebFlux codecs (`spring.codec.max-in-memory-size`) for body buffering limits
7. **CORS Configuration**
   1. **What it does:** Controls which browser origins can call your APIs, and which methods/headers are allowed.
   2. **Why it matters:** Prevents **cross-origin abuse** and inadvertent data exposure from browsers; reduces risk of **CSRF-like cross-**

    site calls from untrusted origins.
  3. **Libraries / config:**
     1. SCG CORS config (global or per-route)
     2. Spring WebFlux/Spring MVC CORS support (if terminating at gateway)
     3. Consider pairing with auth rules and preflight caching settings
8. **Load balancing**
   1. **What it does:** Spreads requests across healthy instances of a service, and avoids unhealthy ones.
   2. **Why it matters:** Improves **availability** and **throughput**, and reduces the impact of node failures or rolling upgrades; helps absorb traffic spikes.
   3. **Libraries / config:**
      - Spring Cloud LoadBalancer (client-side)
      - Service registry: **Eureka**, **Consul**, or Kubernetes endpoints
      - Health checks via Actuator, liveness/readiness probes

2.6. Database-level Constraints

1. **Limit data retrieval and block risky patterns**
   Enforce a hard cap on the number of records returned per request, and explicitly prohibit dangerous or inefficient query patterns (e.g., unbounded queries, wildcard-only searches). This protects performance and reduces abuse.
2. **Auditor context caveats (`@EnableJpaAuditing`)**
   Without Spring Security, the current `createdBy` value is derived from other fields and may not reflect the actual actor who created or updated the account. If adopt Spring Security as the identity source, we'll still need to handle edge cases such as batch/scheduled jobs, anonymous or system actions, and asynchronous processing.

2.7. Other considerations

Terms	Explanation
PII	Personally Identifiable Identity
PCI	Payment Card Industry Information

1. Software Supply-Chain Security

2. Input/Output Sanitization (JSON/XML/Files)
3. Value Masking (PII/CII/Secrets)
4. Secret Management
5. Phishing-resistant MFA
6. Runtime Application Self-Protection (RASP)
    1. RASP instruments the application from within the JVM to detect and block attacks in real time, offering more context than a traditional Web Application Firewall (WAF).
    2. It can detect and block threats like SQL injection or Log4Shell exploitation by observing application behavior, not just network traffic.
7. Secure Software Development Cycle (SSDLC)
    1. **Threat Modeling:** Before writing code, model potential threats using frameworks like **STRIDE** to identify vulnerabilities in the system's design.
    2. **SAST & DAST:** Integrate **Static Application Security Testing (SAST)** tools (e.g., SonarQube, Checkmarx) to find bugs in source code and **Dynamic Application Security Testing (DAST)** tools to probe the running application for vulnerabilities.

3. Others

3.1. Observability

Building infrastructure for **Observability** enables data-driven decision-making and helps answer questions such as:

1. What makes the system slow?
2. Does this enhancement actually improve performance?
3. Will this new approach degrade performance?
4. Is this the root cause of the slowdown?

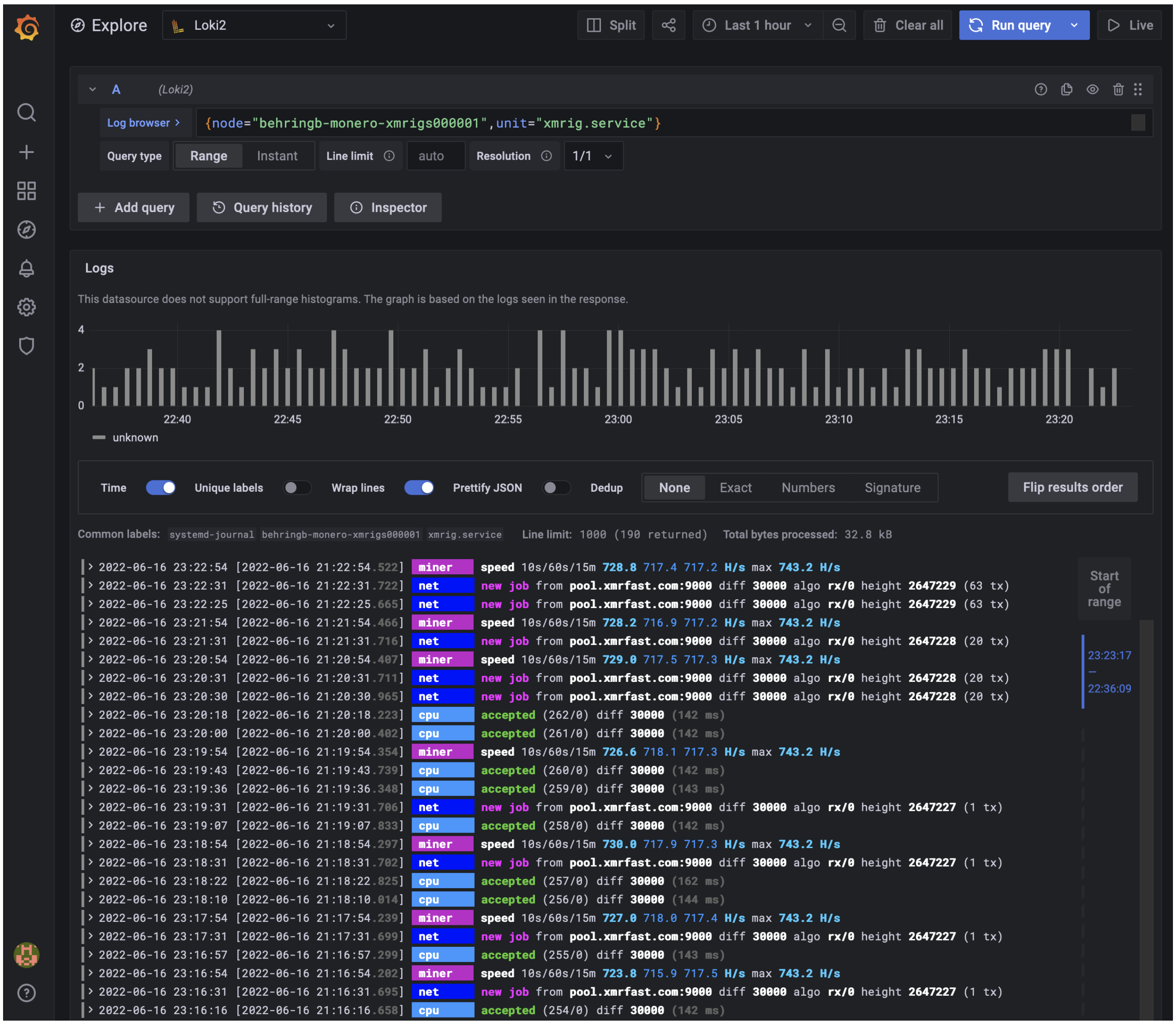
Observability typically covers three areas:

- **Tracing**: What is the request/response flow?
- **Logging**: What information is being logged?
- **Metrics**: How well is the system performing (throughput, execution time, JVM GC, etc.)?

Investing in observability depends on project needs, timeline, and available resources (budget, manpower, etc.).

Using the **Grafana stack** (Grafana, Tempo, Loki, Prometheus, OpenTelemetry) as an example, observability provides the following capabilities:

Logging



```
{node="behringb-monero-xmrigs000001",unit="xmrig.service"}
```

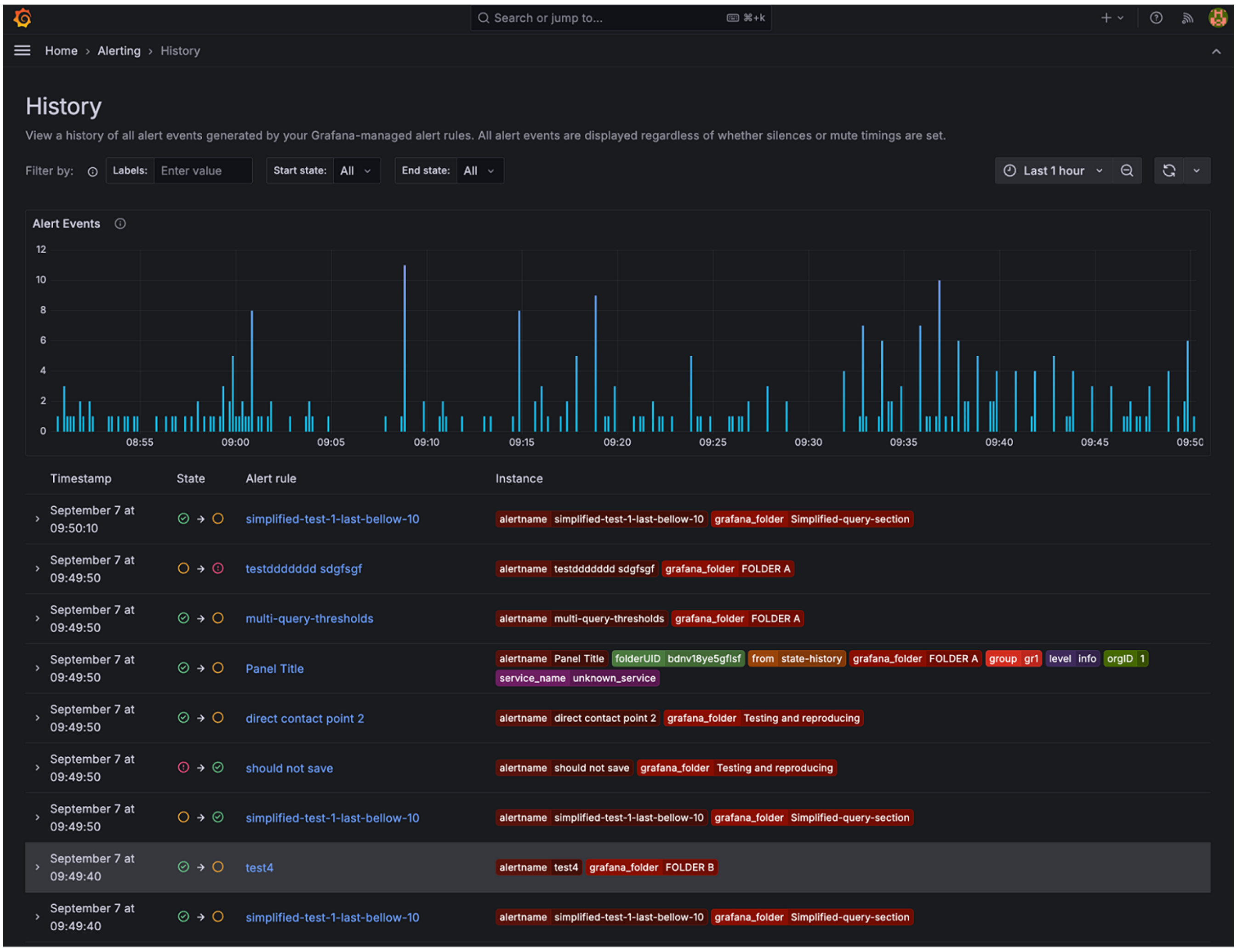
Query type: Range | Instant — Line limit: auto — Resolution: 1/1

This datasource does not support full-range histograms. The graph is based on the logs seen in the response.

Time: on — Unique labels: off — Wrap lines: on — Prettify JSON: off — Dedup: None | Exact | Numbers | Signature

Common labels: `systemd-journal` `behringb-monero-xmrigs000001` `xmrig.service` — Line limit: 1000 (190 returned) — Total bytes processed: 32.8 kB

```
> 2022-06-16 23:22:54 [2022-06-16 21:22:54.522] miner speed 10s/60s/15m 728.8 717.4 717.2 H/s max 743.2 H/s
> 2022-06-16 23:22:31 [2022-06-16 21:22:31.722] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647229 (63 tx)
> 2022-06-16 23:22:25 [2022-06-16 21:22:25.665] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647229 (63 tx)
> 2022-06-16 23:21:54 [2022-06-16 21:21:54.466] miner speed 10s/60s/15m 728.2 716.9 717.2 H/s max 743.2 H/s
> 2022-06-16 23:21:31 [2022-06-16 21:21:31.716] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647228 (20 tx)
> 2022-06-16 23:20:54 [2022-06-16 21:20:54.407] miner speed 10s/60s/15m 729.0 717.5 717.3 H/s max 743.2 H/s
> 2022-06-16 23:20:31 [2022-06-16 21:20:31.711] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647228 (20 tx)
> 2022-06-16 23:20:30 [2022-06-16 21:20:30.965] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647228 (20 tx)
> 2022-06-16 23:20:18 [2022-06-16 21:20:18.223] cpu   accepted (262/0) diff 30000 (142 ms)
> 2022-06-16 23:20:00 [2022-06-16 21:20:00.402] cpu   accepted (261/0) diff 30000 (142 ms)
> 2022-06-16 23:19:54 [2022-06-16 21:19:54.354] miner speed 10s/60s/15m 726.6 718.1 717.3 H/s max 743.2 H/s
> 2022-06-16 23:19:43 [2022-06-16 21:19:43.739] cpu   accepted (260/0) diff 30000 (142 ms)
> 2022-06-16 23:19:36 [2022-06-16 21:19:36.348] cpu   accepted (259/0) diff 30000 (143 ms)
> 2022-06-16 23:19:31 [2022-06-16 21:19:31.706] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647227 (1 tx)
> 2022-06-16 23:19:07 [2022-06-16 21:19:07.833] cpu   accepted (258/0) diff 30000 (142 ms)
> 2022-06-16 23:18:54 [2022-06-16 21:18:54.297] miner speed 10s/60s/15m 730.0 717.9 717.3 H/s max 743.2 H/s
> 2022-06-16 23:18:31 [2022-06-16 21:18:31.702] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647227 (1 tx)
> 2022-06-16 23:18:22 [2022-06-16 21:18:22.825] cpu   accepted (257/0) diff 30000 (162 ms)
> 2022-06-16 23:18:10 [2022-06-16 21:18:10.014] cpu   accepted (256/0) diff 30000 (144 ms)
> 2022-06-16 23:17:54 [2022-06-16 21:17:54.239] miner speed 10s/60s/15m 727.0 718.0 717.4 H/s max 743.2 H/s
> 2022-06-16 23:17:31 [2022-06-16 21:17:31.699] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647227 (1 tx)
> 2022-06-16 23:16:57 [2022-06-16 21:16:57.299] cpu   accepted (255/0) diff 30000 (143 ms)
> 2022-06-16 23:16:54 [2022-06-16 21:16:54.202] miner speed 10s/60s/15m 723.8 715.9 717.5 H/s max 743.2 H/s
> 2022-06-16 23:16:31 [2022-06-16 21:16:31.695] net   new job from pool.xmrfast.com:9000 diff 30000 algo rx/0 height 2647227 (1 tx)
> 2022-06-16 23:16:16 [2022-06-16 21:16:16.658] cpu   accepted (254/0) diff 30000 (142 ms)
```

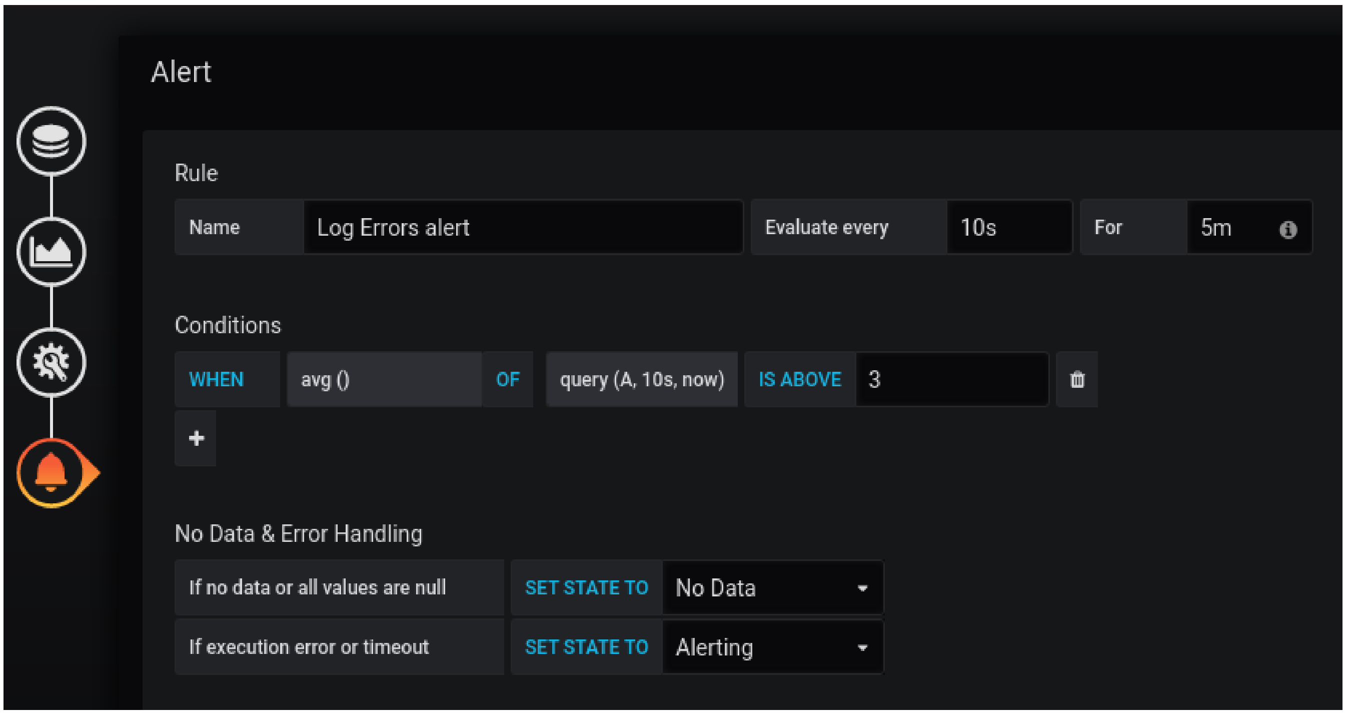


History

View a history of all alert events generated by your Grafana-managed alert rules. All alert events are displayed regardless of whether silences or mute timings are set.

Filter by: Labels: Enter value | Start state: All | End state: All | Last 1 hour

Timestamp	State	Alert rule	Instance
September 7 at 09:50:10	✓ → ○	simplified-test-1-last-bellow-10	alertname simplified-test-1-last-bellow-10 · grafana_folder Simplified-query-section
September 7 at 09:49:50	○ → ○	testdddddddd sdgfsgf	alertname testdddddddd sdgfsgf · grafana_folder FOLDER A
September 7 at 09:49:50	✓ → ○	multi-query-thresholds	alertname multi-query-thresholds · grafana_folder FOLDER A
September 7 at 09:49:50	✓ → ○	Panel Title	alertname Panel Title · folderUID bdnv18ye5gflsf · from state-history · grafana_folder FOLDER A · group gr1 · level info · orgID 1 · service_name unknown_service
September 7 at 09:49:50	✓ → ○	direct contact point 2	alertname direct contact point 2 · grafana_folder Testing and reproducing
September 7 at 09:49:50	○ → ✓	should not save	alertname should not save · grafana_folder Testing and reproducing
September 7 at 09:49:50	○ → ✓	simplified-test-1-last-bellow-10	alertname simplified-test-1-last-bellow-10 · grafana_folder Simplified-query-section
September 7 at 09:49:40	✓ → ○	test4	alertname test4 · grafana_folder FOLDER B
September 7 at 09:49:40	✓ → ○	simplified-test-1-last-bellow-10	alertname simplified-test-1-last-bellow-10 · grafana_folder Simplified-query-section



Alert

Rule

Name	Log Errors alert	Evaluate every	10s	For	5m

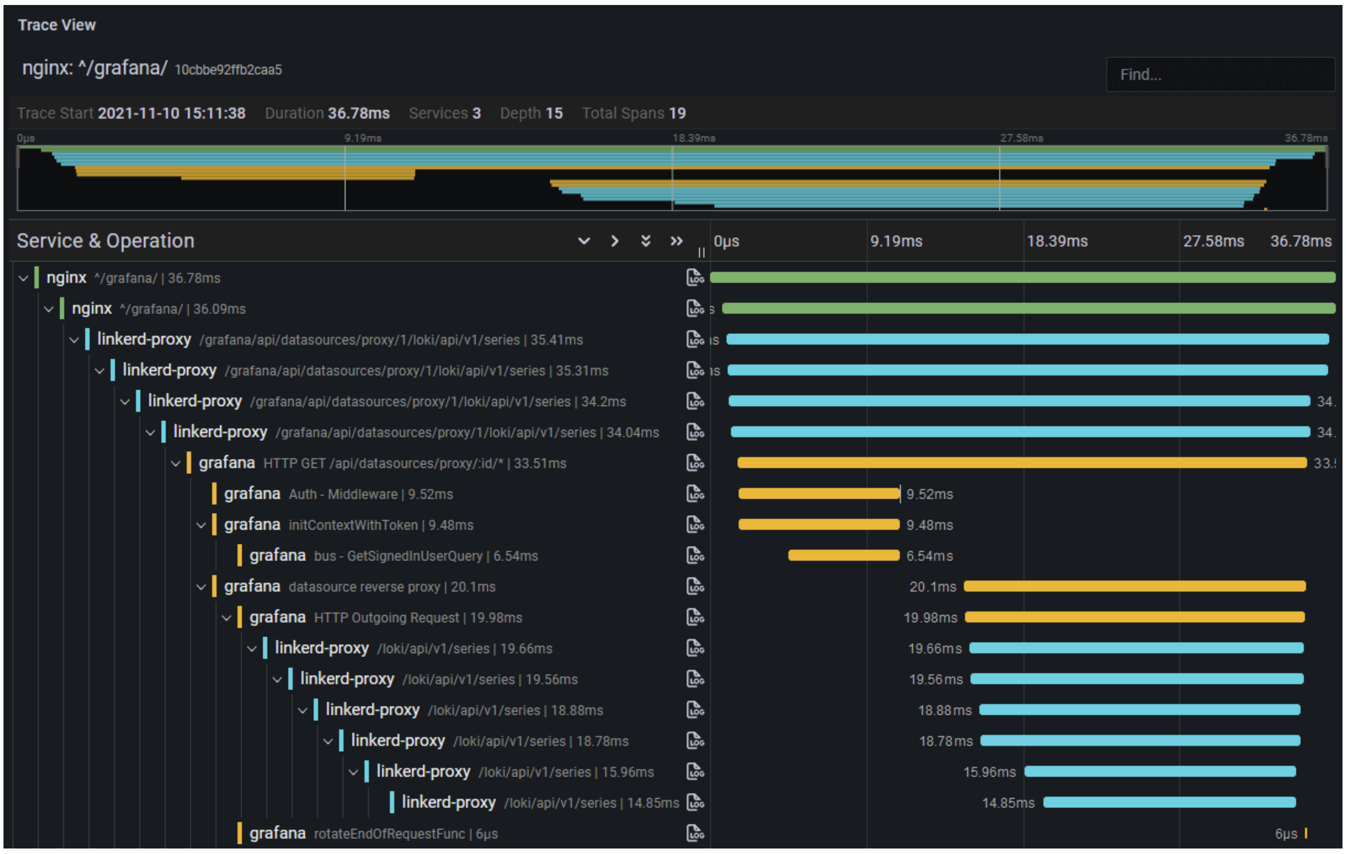
Conditions

WHEN avg () OF query (A, 10s, now) IS ABOVE 3

No Data & Error Handling

If no data or all values are null	SET STATE TO	No Data
If execution error or timeout	SET STATE TO	Alerting

Tracing

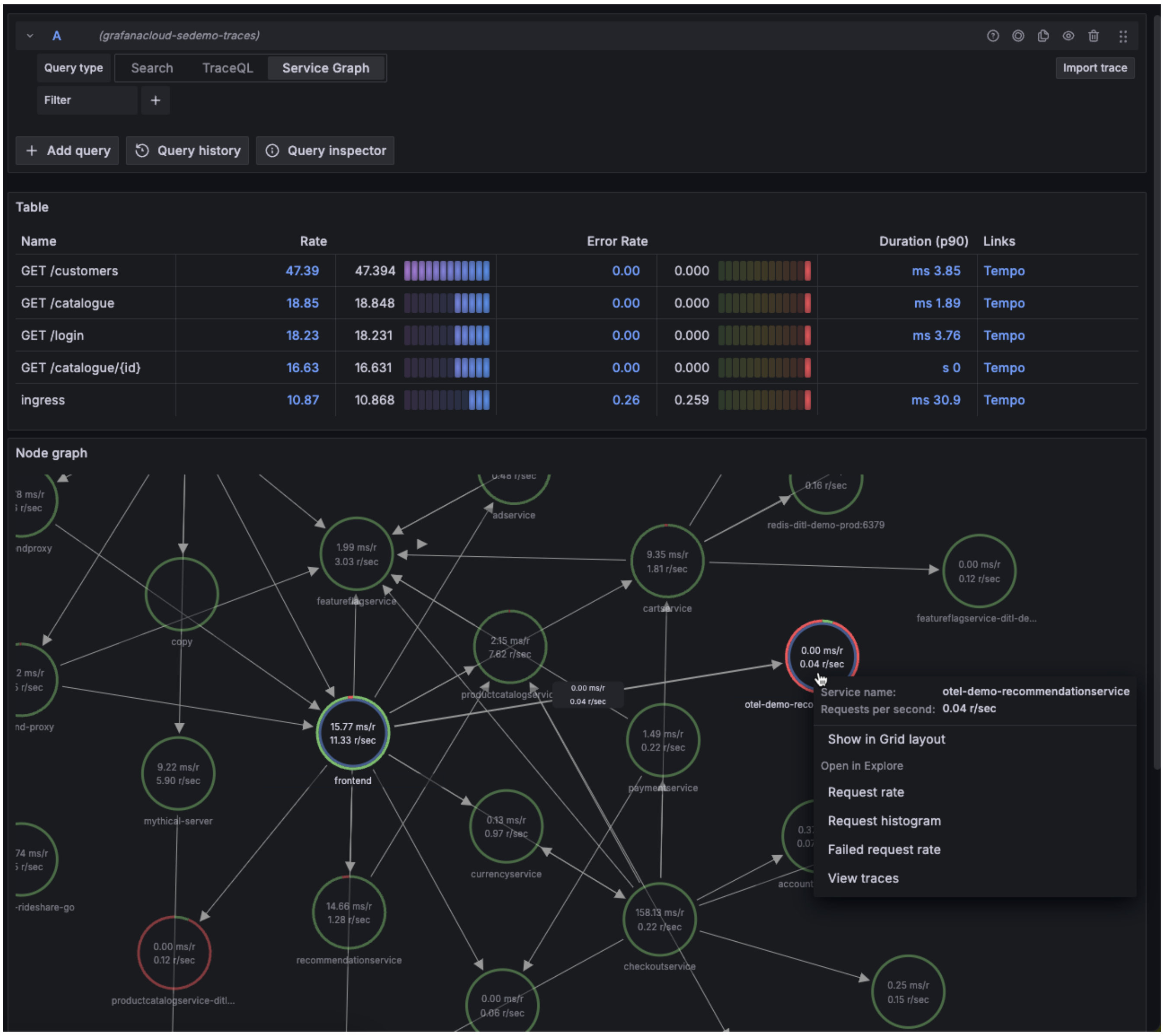


Trace View

nginx: ^/grafana/ 10cbbe92ffb2caa5

Trace Start **2021-11-10 15:11:38** Duration **36.78ms** Services **3** Depth **15** Total Spans **19**

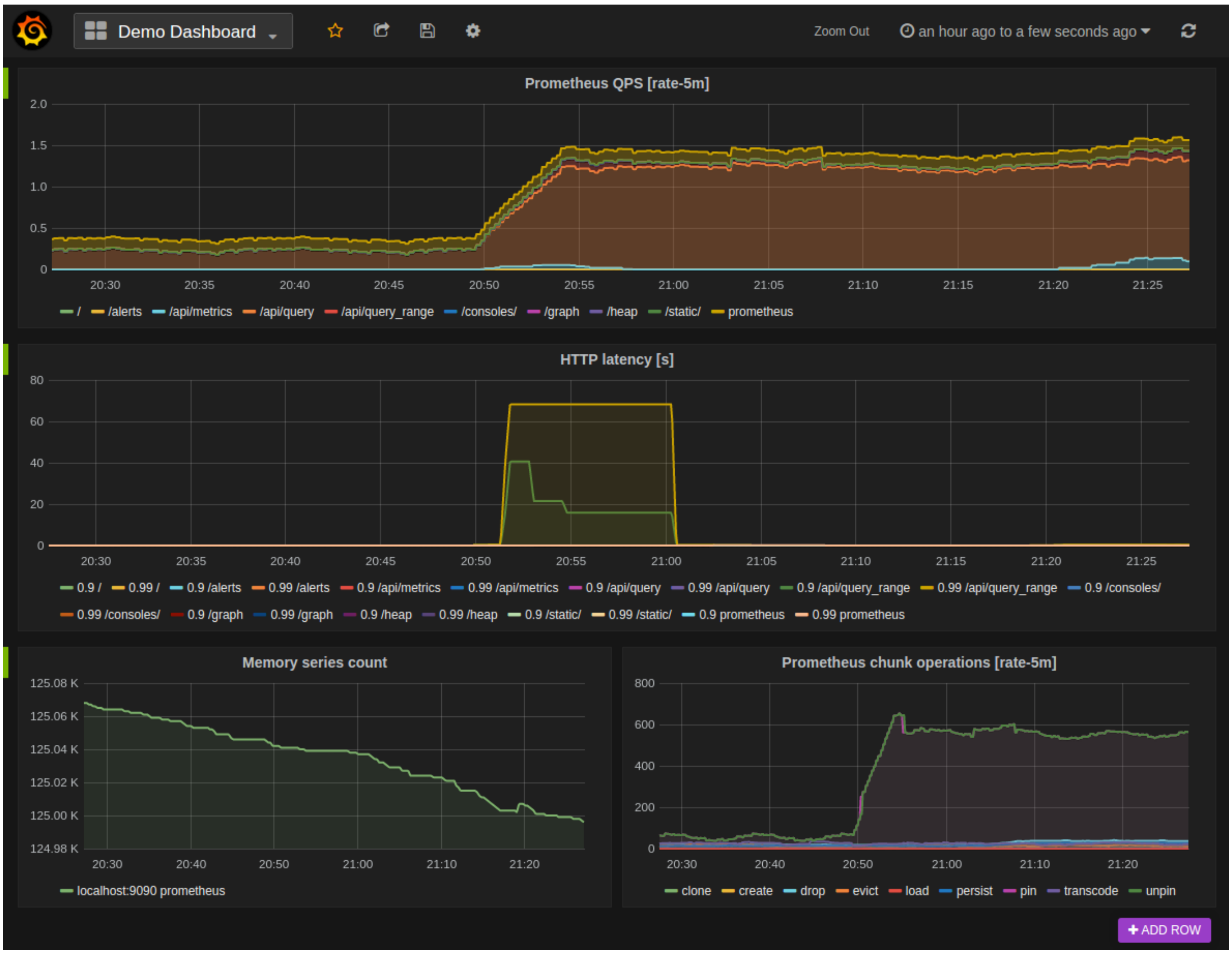
Service & Operation	Duration
nginx ^/grafana/	36.78ms
nginx ^/grafana/	36.09ms
linkerd-proxy /grafana/api/datasources/proxy/1/loki/api/v1/series	35.41ms
linkerd-proxy /grafana/api/datasources/proxy/1/loki/api/v1/series	35.31ms
linkerd-proxy /grafana/api/datasources/proxy/1/loki/api/v1/series	34.2ms
linkerd-proxy /grafana/api/datasources/proxy/1/loki/api/v1/series	34.04ms
grafana HTTP GET /api/datasources/proxy/:id/*	33.51ms
grafana Auth - Middleware	9.52ms
grafana initContextWithToken	9.48ms
grafana bus - GetSignedInUserQuery	6.54ms
grafana datasource reverse proxy	20.1ms
grafana HTTP Outgoing Request	19.98ms
linkerd-proxy /loki/api/v1/series	19.66ms
linkerd-proxy /loki/api/v1/series	19.56ms
linkerd-proxy /loki/api/v1/series	18.88ms
linkerd-proxy /loki/api/v1/series	18.78ms
linkerd-proxy /loki/api/v1/series	15.96ms
linkerd-proxy /loki/api/v1/series	14.85ms
grafana rotateEndOfRequestFunc	6µs

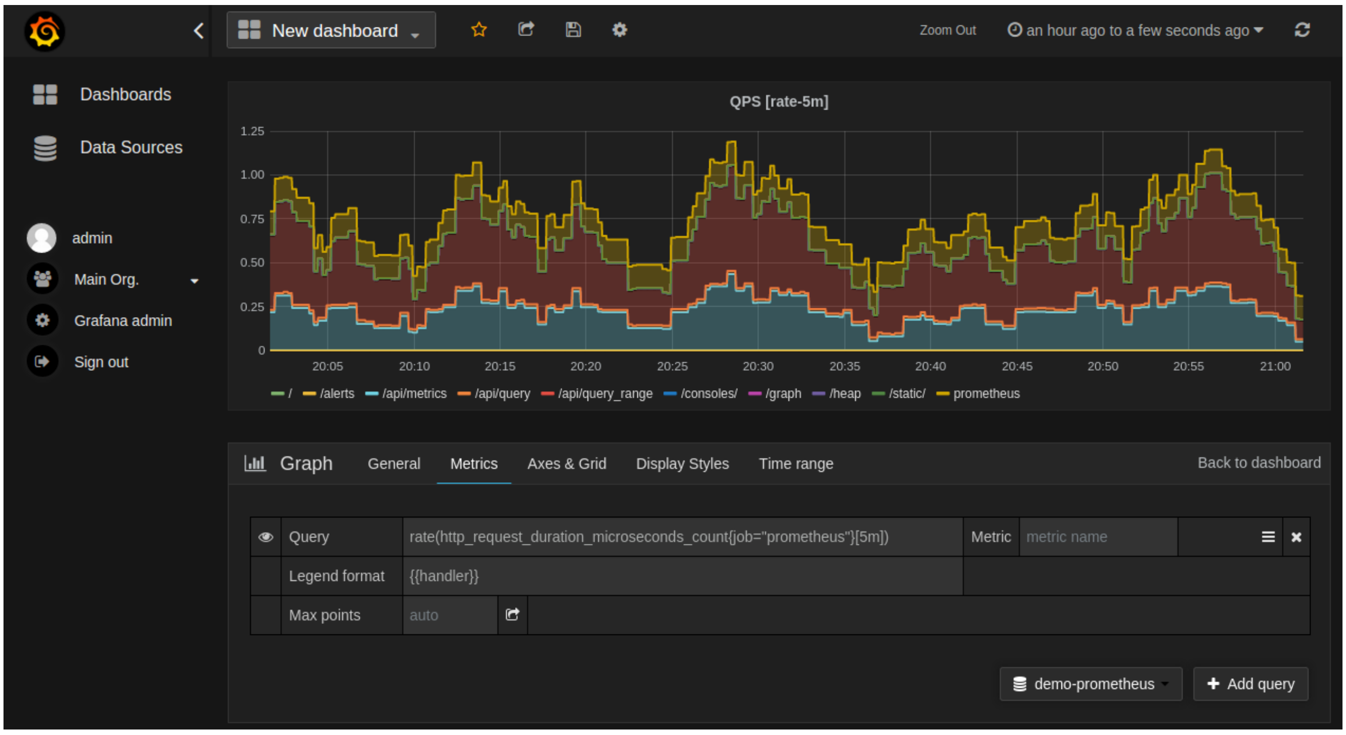


Table

Name	Rate		Error Rate		Duration (p90)	Links
GET /customers	47.39	47.394	0.00	0.000	ms 3.85	Tempo
GET /catalogue	18.85	18.848	0.00	0.000	ms 1.89	Tempo
GET /login	18.23	18.231	0.00	0.000	ms 3.76	Tempo
GET /catalogue/{id}	16.63	16.631	0.00	0.000	s 0	Tempo
ingress	10.87	10.868	0.26	0.259	ms 30.9	Tempo

Metrics


Demo Dashboard
Zoom Out
an hour ago to a few seconds ago
Prometheus QPS [rate-5m]
/ /alerts /api/metrics /api/query /api/query_range /consoles/ /graph /heap /static/ prometheus
HTTP latency [s]
0.9 / 0.99 / 0.9 /alerts 0.99 /alerts 0.9 /api/metrics 0.99 /api/metrics 0.9 /api/query 0.99 /api/query 0.9 /api/query_range 0.99 /api/query_range 0.9 /consoles/ 0.99 /consoles/ 0.9 /graph 0.99 /graph 0.9 /heap 0.99 /heap 0.9 /static/ 0.99 /static/ 0.9 prometheus 0.99 prometheus
Memory series count
localhost:9090 prometheus
Prometheus chunk operations [rate-5m]
clone create drop evict load persist pin transcode unpin
+ ADD ROW



Query	rate(http_request_duration_microseconds_count{job="prometheus"}[5m])
Metric	metric name
Legend format	{{handler}}
Max points	auto

References: [The Grafana Stack | Grafana Labs]

3.2. Testing

Investing in testing will significantly improve system reliability and maintainability. Moreover, **establishing testing standards early**, including conventions, utilities, solutions for tricky scenarios, best-practice suggestions, configurations, and DSL design, can simplify future testing efforts. This ensures a more reliable system and reduces the effort required to fix bugs discovered during integration or later stages.

Currently, we have:

- **Unit Tests**: JUnit 5, Mockito, AssertJ
- **Integration Tests**: Testcontainers, MockMvc
- Others: Load testing, mutation testing, modulith testing, contract testing, etc.
- Testing :: Spring Security

3.3. CI/CD

3.3.1. Issues Mitgations

The current CI/CD pipeline has two recurring issues (not critical, but worth improving):

1. Developers often forget to sync their feature/preview branches with `develop`.
2. Developers occasionally push code that may be:
   - in conflict
   - uncompilable
   - undeployable

Proposed improvements: Add CI/CD checks for:

- **Merge Requests to `develop`**
  - Verify that the source branch is synced with `develop`
  - Ensure the branch is compitible and buildable
  - Confirm that the branch has been deployed on preview before
- **Pushes to Preview**
  - Verify that the branch is synced with `develop`

3.3.2. Enhancement

3.3.2.1. `SonarCube`

Use **SonarQube** (if feasible) for standardized code quality checks and test coverage enforcement. SonarQube provides:

- **Code Quality Checks**: Detects code smells (bad practices, complexity, duplication)
- **Security Analysis**: Finds vulnerabilities (SQL injection, XSS, weak cryptography)
- **Bug Detection**: Identifies common mistakes that may cause runtime errors
- **Maintainability Metrics**: Tracks technical debt, coverage, duplication, and complexity

In essence, SonarQube acts as a **technical code reviewer**, handling low-level checks so that human reviewers can focus on:

- System design
- Business logic correctness
- Team-specific coding conventions

Together with CI/CD enhancements, this reduces the need for reviewers to flag minor issues like incorrect annotations or package names.

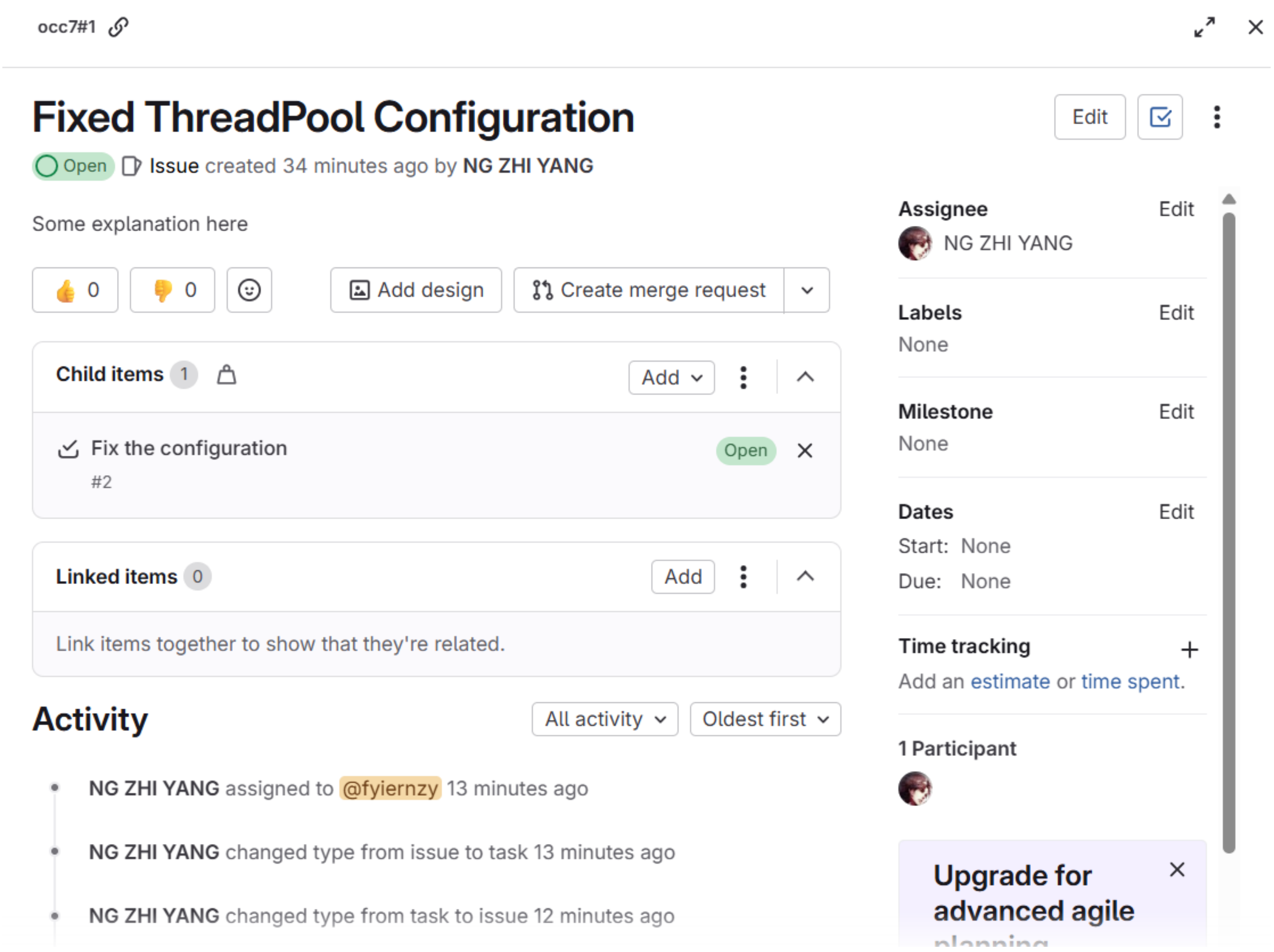
3.3.2.2. Issue Tracker vs TODO

Using `TODO` comments in code is common, but not always reliable:

- `TODO`s can be accidentally removed during merges.
- They lack severity context and may be ignored.
- They provide limited space for discussion.

A better alternative is to **use GitLab Issues** instead of inline TODOs:

1. Issues persist across merge requests and avoid accidental deletion.
2. Issues allow richer context and discussion beyond a code comment.
3. Rules can be enforced for using TODO/FIXME vs. issues.
4. Any team member (not just the author) can pick up and resolve issues.
5. Issues provide visibility into individual contributions beyond just feature MRs.



occ7#1

Fixed ThreadPool Configuration

Open · Issue created 34 minutes ago by NG ZHI YANG

Some explanation here

Child items 1

- Fix the configuration #2 — Open

Linked items 0

Link items together to show that they're related.

Activity

- NG ZHI YANG assigned to @fyiernzy 13 minutes ago
- NG ZHI YANG changed type from issue to task 13 minutes ago
- NG ZHI YANG changed type from task to issue 12 minutes ago

Assignee: NG ZHI YANG

Labels: None

Milestone: None

Dates
Start: None
Due: None

Time tracking
Add an estimate or time spent.

1 Participant

Upgrade for advanced agile

3.3.2.3. Git Hooks

Git Hooks can complement CI/CD enhancements by enforcing rules locally:

- **pre-push**: Ensures the branch is synced with `develop` before pushing
- **pre-commit**: Enforces commit message conventions
- **pre-receive**: Prevents branches that are not synced with `develop` from being pushed to target branches

Git Aliases

To simplify branch management, we can define helpful Git aliases while keeping flexibility:

- `git sync <branch>` → Syncs the current branch with the specified branch
- `git cleanup` → Cleans up branches that are already merged into `develop` and have no unstaged files or commits