Observability with Prometheus, Grafana, Loki & OpenTelemetry

Web Analytics & Observability

Agenda

- What is observability? (and how it differs from monitoring)
- A short history of monitoring & early tools
- Modern observability stack: Prometheus, Loki, Grafana, OpenTelemetry
- How we connect them in this project
- Key takeaways and demo hand-off

Observability: the big idea

- Goal: understand the internal state of a system from the signals it emits.
- Monitoring asks: “Is this known thing broken?”
- Observability asks: “Why is this weird thing happening?” even if we didn’t predict it.
- We do that using telemetry: metrics, logs and traces together.

Signals: metrics, logs, traces

- Metrics – numbers over time (rates, counts, durations). Great for monitoring & high-level health.
- Logs – discrete text events. Great for detailed stories and debugging.
- Traces – end-to-end view of a request across services. Great for performance & causality.
- Observability = using all of them together, not just one dashboard.

Monitoring vs observability

- Monitoring: alerting & dashboards for known failure modes (CPU high, error rate > threshold…).
- Observability: design telemetry so you can explore new questions without redeploying.
- Monitoring is a subset: you still need alerts, but you also need rich signals for debugging.
- Our tools can be used for both – it depends on how we instrument and how we query them.

Short history & early tools

- Early days: logs on disk, ping checks, simple CPU/RAM graphs.
- Nagios/Zabbix: “Are my hosts up? Did a known check fail?” – strong monitoring, little observability.
- Graphite/RRD: storing time-series, but little standardisation for service-level labels.
- The ELK stack (Elasticsearch/Logstash/Kibana): centralised logs, but often heavy & expensive at scale.

Why a new stack was needed

- Microservices, containers and Kubernetes increased the number of services and logs massively.
- Teams needed label-based metrics and logs that matched each other.
- We wanted tooling that fits cloud-native patterns (dynamic IPs, ephemeral pods).
- This is the context where Prometheus, Loki, Grafana and OpenTelemetry appeared.

Modern observability stack (conceptual)

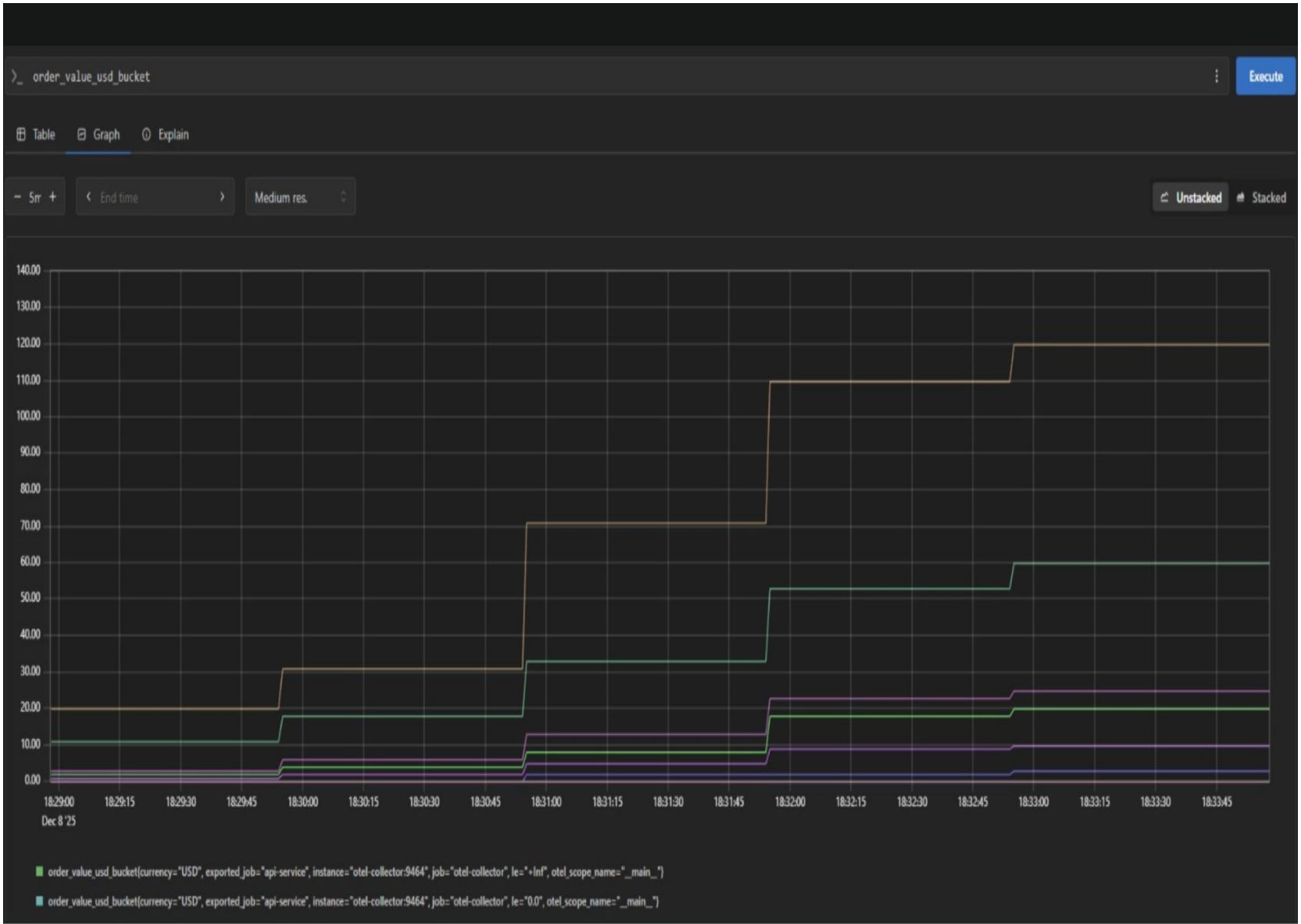
- Prometheus – metrics backend (monitoring + observability + alerts).
- Loki – log aggregation (debugging + observability, can also drive alerts).
- OpenTelemetry – instrumentation layer that generates telemetry.
- Grafana – the shared window where we see and explore everything.

Signal	Question it answers	Tool used in this tutorial
Metrics	“Is something wrong? How often? How big?”	Prometheus
Logs	“What exactly happened?”	Loki
Observability UI	“How do we explore all of that?”	Grafana

Prometheus – metrics for monitoring & observability

- Collects metrics by scraping /metrics endpoints on a schedule (pull model).
- Label-based data model: service, instance, route, environment, etc.
- PromQL lets you aggregate, slice and alert on metrics.
- Used for classic monitoring (alerts) and for exploratory queries when debugging.

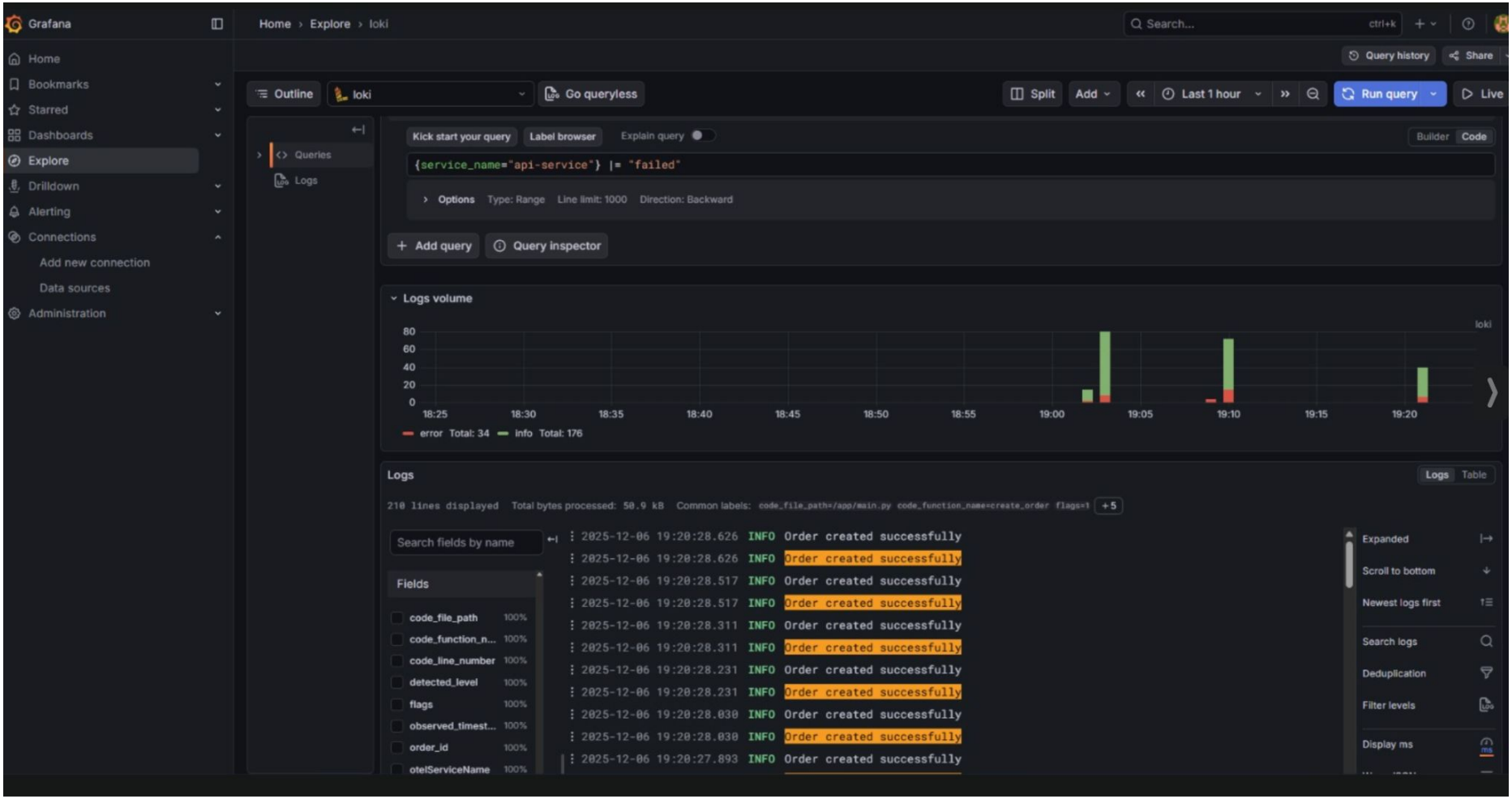
Prometheus / Grafana



Loki – logs that line up with your metrics

- Log aggregation system that groups logs into streams, indexed mainly by labels.
- LogQL query language feels similar to PromQL but works on log lines.
- For monitoring: you can alert on patterns in logs (e.g. error rate from logs).
- For observability: you jump from a metric spike into the exact log lines that explain it.

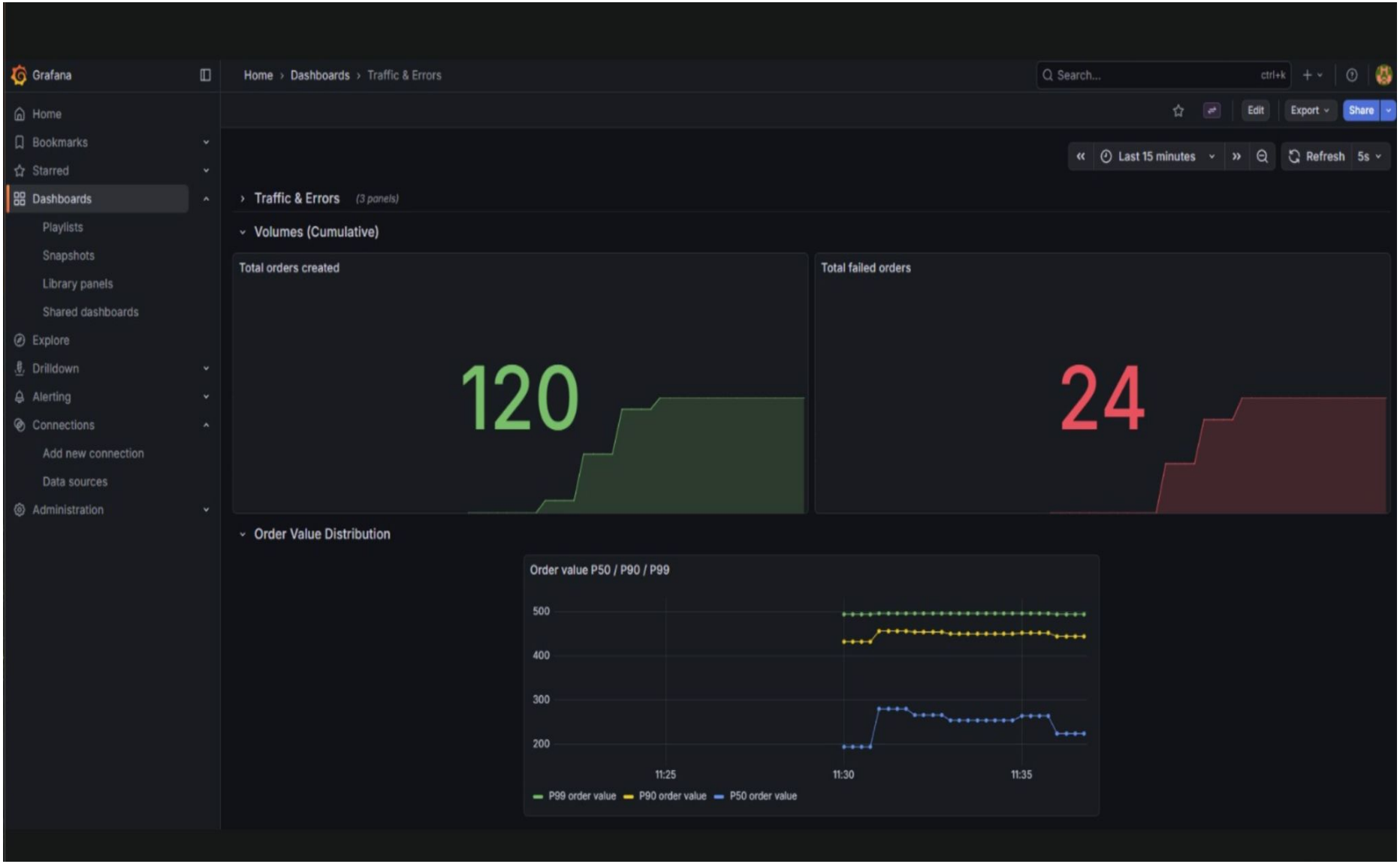
Loki-log metrics for a Query



Grafana – the shared observability UI

- Connects to Prometheus, Loki and other backends as data sources.
- Dashboards summarise health (monitoring) for the whole team.
- Explore mode is for observability: ad-hoc queries on metrics and logs while debugging.
- It’s where we stitch different signals together in one place.

Grafana Dashboard



OpenTelemetry – instrumentation layer

- Standard, vendor-neutral way to generate telemetry in code.
- SDKs and auto-instrumentation libraries for many languages and frameworks.
- OpenTelemetry Collector receives telemetry and forwards it to backends such as Prometheus and Loki.
- Helps us build observability once and choose tools later.

How these tools connect in our project

- Our demo web app emits metrics and logs (and can be instrumented with OTel).
- Prometheus scrapes metrics from the app and infrastructure.
- Loki receives logs (via an agent like Promtail) with matching labels.
- Grafana is configured with both Prometheus and Loki so we can move between metrics and logs.

Project Setup

Orchestration: Docker and **Docker Compose** manage all services and the environment.

Instrumentation: OpenTelemetry (OTel) is the standard used to instrument the application code.

Traffic Generation: Postman/cURL are used to generate sample API load for testing.

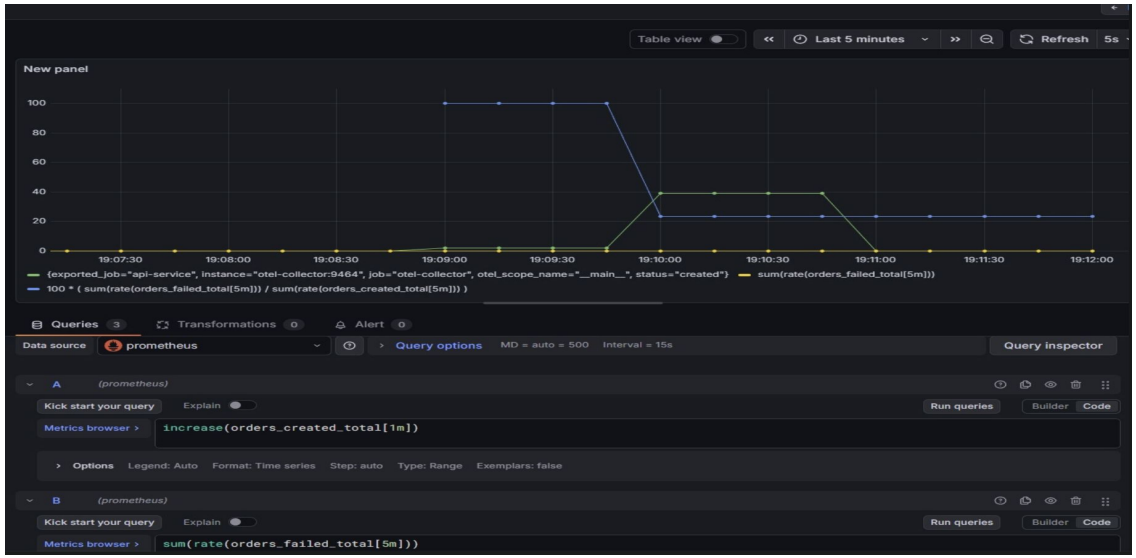
Grafana Port: `3000` (**Visualization** and dashboards).

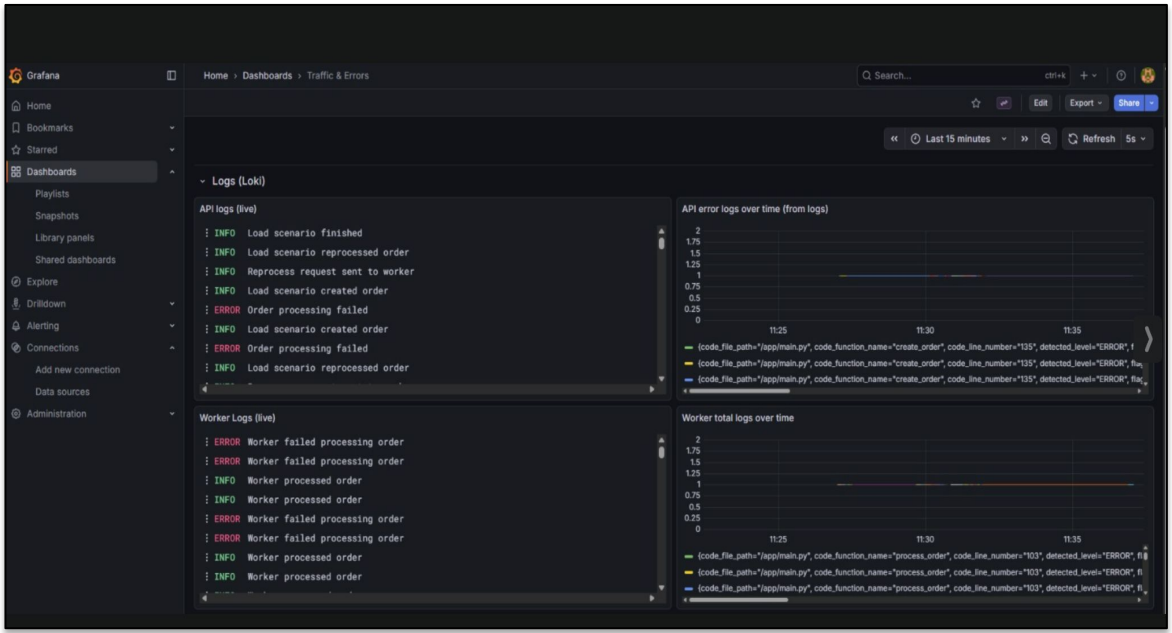
Prometheus Port: `9090` (**Time-series database** and metric collection).

Loki Port: `3100` (Log aggregation and **storage**).

Application Endpoint: Node.js service metrics scraped on port `9900`.

Our project in action





Key takeaways

- Observability is a property of the system + telemetry, not just a tool.
- Metrics (Prometheus) are great for monitoring and for spotting that something changed.
- Logs (Loki) are great for explaining why metrics look the way they do.
- Grafana brings the signals together, and OpenTelemetry standardises how we emit them.

Thank you – questions?

- Next: live demo of our observability stack for the tutorial project.
- As you watch, try to identify which parts relate to metrics, logs, Grafana and OpenTelemetry.