Java, OpenTelemetry, and Grafana

This setup demonstrates a **complete observability pipeline** for a Java application using **OpenTelemetry**, containerized with **Docker Compose**. It captures **traces, metrics, and logs** from the Java app and routes them to visualization and storage systems.

Components:

Component	**Purpose**
Java Application	The main application instrumented with **OpenTelemetry Java Agent**. Automatically emits **traces, metrics, and logs**.
OpenTelemetry Collector (OTEL Collector)	Centralized service that **receives telemetry data** from the app and routes it to the proper backend (Jaeger, Prometheus, Loki).
Jaeger	Collects and visualizes **traces** for distributed tracing and performance analysis.
Prometheus	Scrapes **metrics** from OTEL Collector and stores them for monitoring and alerting.
Loki	Stores **logs** collected from OTEL Collector, which can be queried in Grafana.
Grafana	Dashboard and visualization tool for **traces, metrics, and logs**. Connects to Jaeger, Prometheus, and Loki.

TABLE OF CONTENTS

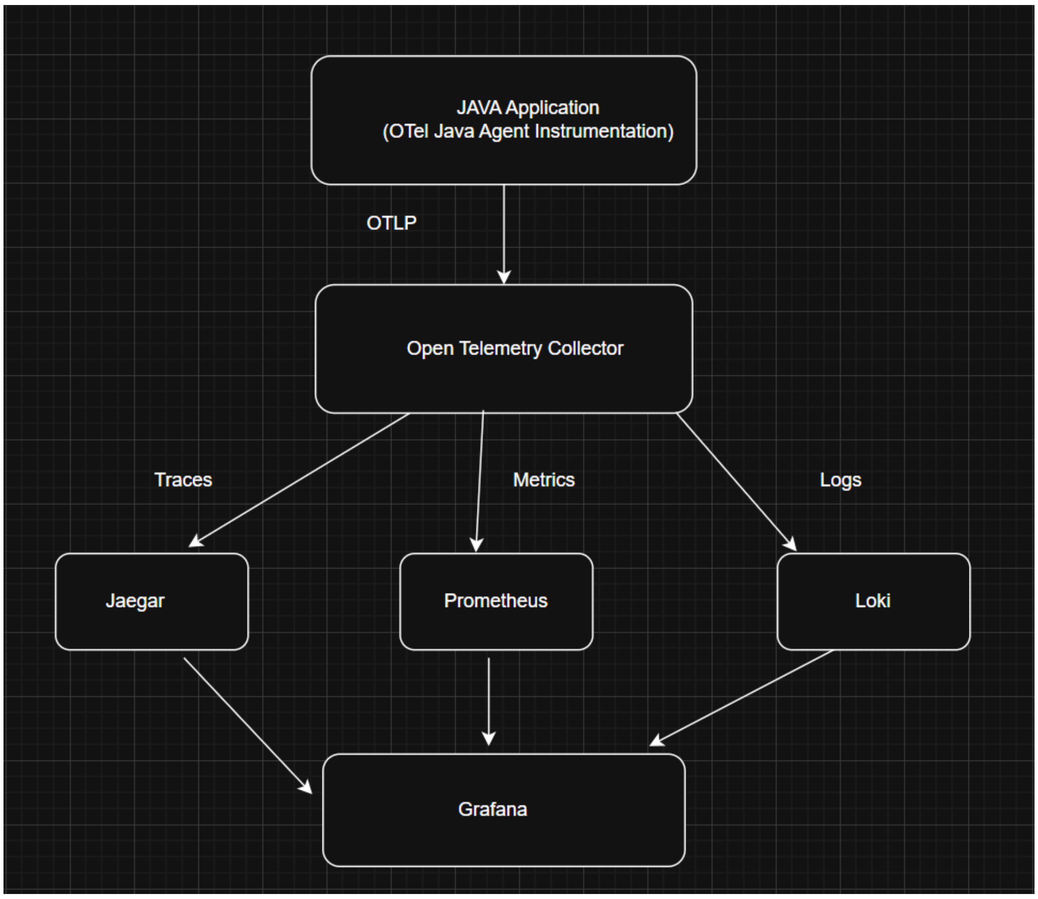
1. **Introduction**

This documentation explains how to build a **complete observability pipeline** for a Java application using:

- OpenTelemetry Java Instrumentation (Agent-Based)
- OpenTelemetry Collector
- Jaeger → Trace storage & visualization
- Prometheus → Metrics storage
- Loki → Log storage
- Grafana → Unified Observability Dashboard.

2. **Architecture Overview**

Below is the architecture diagram showing the full flow.



3. Project Directory

Run:

```
mkdir -p ~/otel-java-observability

cd ~/otel-java-observability
```

4. Step 1 — Java App Setup + OpenTelemetry Agent

i) Download OpenTelemetry Java Agent

Download the latest OTel agent:

Command to run:

```
wget https://github.com/open-telemetry/opentelemetry-java-
instrumentation/releases/latest/download/opentelemetry-javaagent.jar
```

ii) Place Your Java Application

Copy your JAR into the folder

Validate: `ls -l app.jar`

By the end of this you will get 2 jar files in your directory: `opentelemetry-javaagent.jar` and `app.jar`.

5. Step 2 — Dockerfile Configuration

i) Create: `nano Dockerfile`

ii) Paste:

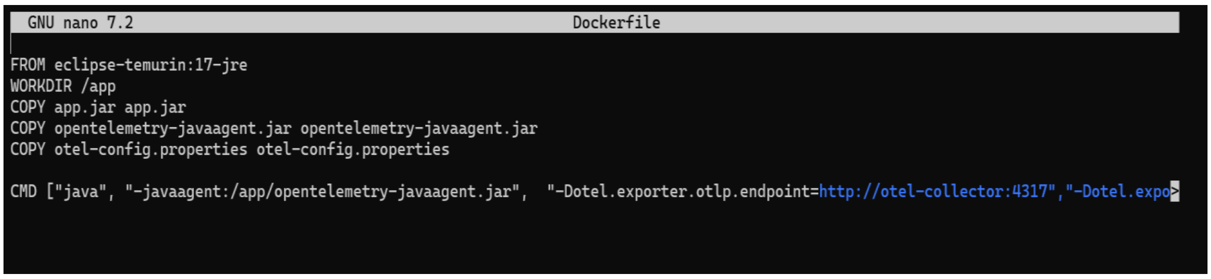
```
FROM eclipse-temurin:17-jre
WORKDIR /app

COPY app.jar app.jar
COPY opentelemetry-javaagent.jar opentelemetry-javaagent.jar
COPY otel-config.properties otel-config.properties

CMD ["java",
"-javaagent:/app/opentelemetry-javaagent.jar",
"-Dotel.exporter.otlp.endpoint=http://otel-collector:4317",
"-Dotel.exporter.otlp.protocol=grpc",
"-Dotel.resource.attributes=service.name=dummy-java-app",
"-jar", "/app/app.jar"]
```

iii) Save: CTRL + O → ENTER → CTRL + X

iv) Script Information

Line	**What It Does**
FROM eclipse-temurin:17-jre	Uses Java 17 runtime as the base image.
WORKDIR /app	Sets working directory inside container.
COPY app.jar app.jar	Copies your Java application JAR.
COPY opentelemetry-javaagent.jar opentelemetry-javaagent.jar	Adds OpenTelemetry Java agent for instrumentation.
COPY otel-config.properties otel-config.properties	Adds OTel agent configuration file.
-javaagent:/app/opentelemetry-javaagent.jar	Enables OpenTelemetry agent when app starts.
-Dotel.exporter.otlp.endpoint=http://otel-collector:4317	Sends telemetry to the OpenTelemetry Collector.
-Dotel.exporter.otlp.protocol=grpc	Uses OTLP/gRPC protocol.
-Dotel.resource.attributes=service.name=dummy-java-app	Sets service name for traces, metrics, logs.
-jar /app/app.jar	Runs the Java application.

Screenshot:



```
  GNU nano 7.2                                          Dockerfile

FROM eclipse-temurin:17-jre
WORKDIR /app
COPY app.jar app.jar
COPY opentelemetry-javaagent.jar opentelemetry-javaagent.jar
COPY otel-config.properties otel-config.properties

CMD ["java", "-javaagent:/app/opentelemetry-javaagent.jar",  "-Dotel.exporter.otlp.endpoint=http://otel-collector:4317","-Dotel.expo>
```

6. Step 3 — OpenTelemetry Configuration Files

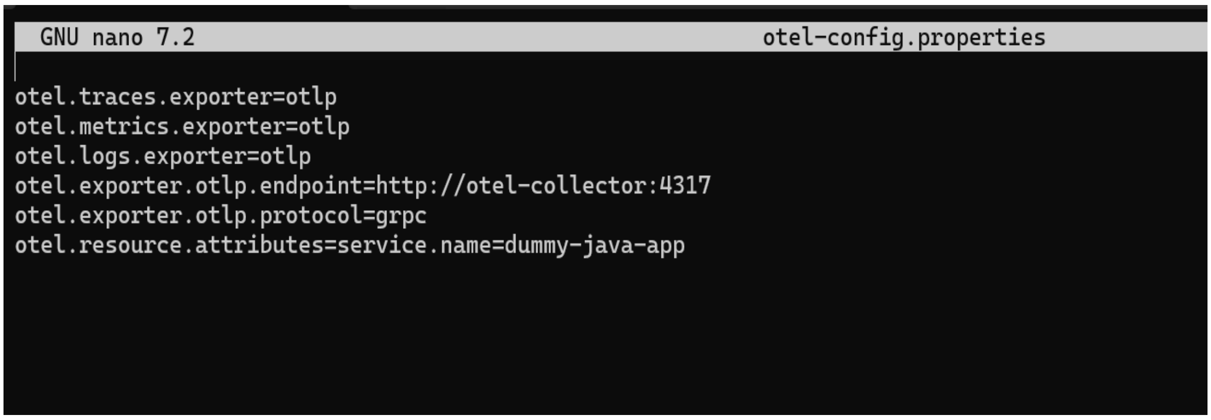
Create: `nano otel-config.properties`

Paste:

```
otel.traces.exporter=otlp

otel.metrics.exporter=otlp

otel.logs.exporter=otlp

otel.exporter.otlp.endpoint=http://otel-collector:4317

otel.exporter.otlp.protocol=grpc

otel.resource.attributes=service.name=dummy-java-app
```

Property	**What It Does**
otel.traces.exporter=otlp	Sends traces using OTLP protocol.
otel.metrics.exporter=otlp	Sends metrics using OTLP protocol.
otel.logs.exporter=otlp	Sends logs using OTLP protocol.
otel.exporter.otlp.endpoint=http://otel-collector:4317	Specifies the OpenTelemetry Collector endpoint to receive telemetry.
otel.exporter.otlp.protocol=grpc	Uses gRPC for sending telemetry (fast and efficient).
otel.resource.attributes=service.name=dummy-java-app	Adds metadata (service name) to all telemetry for identification.

Screenshot:



```
GNU nano 7.2                                         otel-config.properties

otel.traces.exporter=otlp
otel.metrics.exporter=otlp
otel.logs.exporter=otlp
otel.exporter.otlp.endpoint=http://otel-collector:4317
otel.exporter.otlp.protocol=grpc
otel.resource.attributes=service.name=dummy-java-app
```

7. Step 4 — OTEL Collector Pipelines Explained

Create: `nano collector-config.yaml`

Paste:

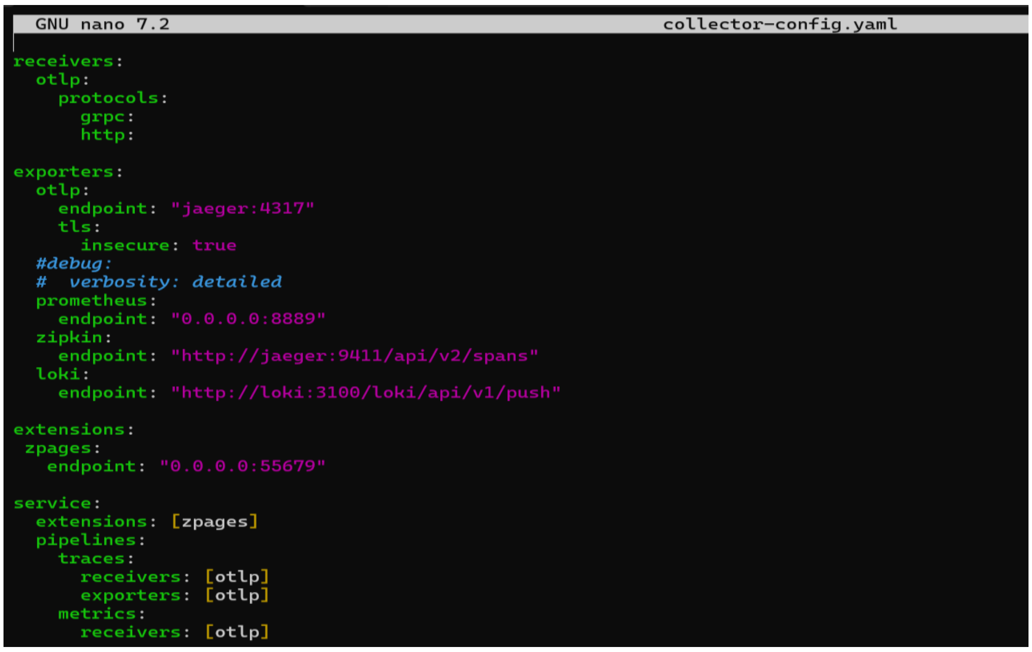
```
receivers:
otlp:
protocols:
grpc:
http:

exporters:
otlp:
endpoint: "jaeger:4317"
tls:
insecure: true

prometheus:
endpoint: "0.0.0.0:8889"

loki:
endpoint: "http://loki:3100/loki/api/v1/push"

extensions:
zpages:
endpoint: "0.0.0.0:55679"

service:
```

```
extensions: [zpages]
pipelines:
traces:
receivers: [otlp]
exporters: [otlp]

metrics:
receivers: [otlp]
exporters: [prometheus]

logs:
receivers: [otlp]
exporters: [loki]
```

Section / Key	What It Does
receivers: otlp:	Defines OTLP receiver to accept telemetry from apps.
protocols: grpc	Receives telemetry over gRPC.
protocols: http	Receives telemetry over HTTP.
exporters: otlp:	Sends traces to Jaeger.
endpoint: "jaeger:4317"	Jaeger collector endpoint for traces.
tls: insecure: true	Disables TLS for testing / internal network.
exporters: prometheus:	Exposes metrics in Prometheus format.
endpoint: "0.0.0.0:8889"	Prometheus scrapes metrics from this endpoint.
exporters: loki:	Sends logs to Loki.
endpoint: "http://loki:3100/loki/api/v1/push"	Loki push API endpoint for logs.
extensions: zpages:	Adds zPages for debugging collector internals.

Section / Key	What It Does
endpoint: "0.0.0.0:55679"	Access zPages via this port.
service:	Defines the pipelines for OTEL Collector.
pipelines: traces:	Pipeline to handle traces.
receivers: [otlp]	Receives traces via OTLP.
exporters: [otlp]	Exports traces to Jaeger.
pipelines: metrics:	Pipeline to handle metrics.
receivers: [otlp]	Receives metrics via OTLP.
exporters: [prometheus]	Exports metrics to Prometheus.
pipelines: logs:	Pipeline to handle logs.
receivers: [otlp]	Receives logs via OTLP.
exporters: [loki]	Exports logs to Loki.

Screenshot:



```
  GNU nano 7.2                                          collector-config.yaml
receivers:
  otlp:
    protocols:
      grpc:
      http:

exporters:
  otlp:
    endpoint: "jaeger:4317"
    tls:
      insecure: true
  #debug:
  #  verbosity: detailed
  prometheus:
    endpoint: "0.0.0.0:8889"
  zipkin:
    endpoint: "http://jaeger:9411/api/v2/spans"
  loki:
    endpoint: "http://loki:3100/loki/api/v1/push"

extensions:
 zpages:
   endpoint: "0.0.0.0:55679"

service:
  extensions: [zpages]
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]
    metrics:
      receivers: [otlp]
```

8. Step 5 — Prometheus Config

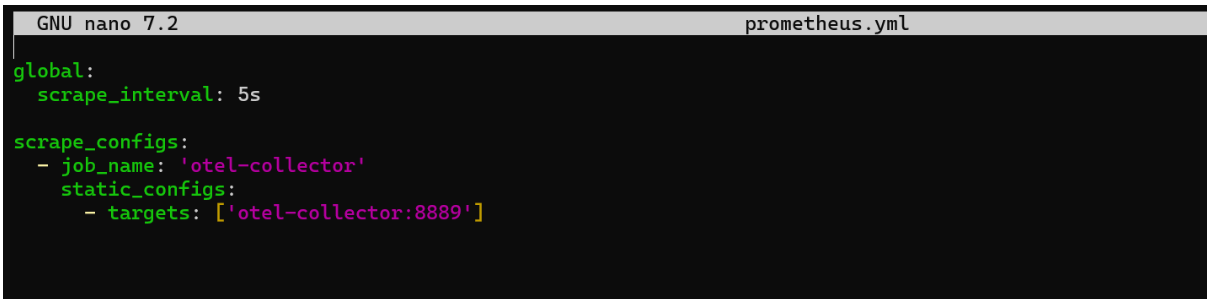
```
nano prometheus.yml
```

Paste:

```
global:

scrape_interval: 5s



scrape_configs:

- job_name: 'otel-collector'

static_configs:

- targets: ['otel-collector:8889']
```

Section / Key	**What It Does**
global: scrape_interval: 5s	Prometheus scrapes metrics every 5 seconds.
scrape_configs:	Defines which targets Prometheus will monitor.
- job_name: 'otel-collector'	Name of this scrape job.
static_configs:	Static list of targets for this job.
- targets: ['otel-collector:8889']	Prometheus scrapes metrics from the OTEL Collector at this endpoint.

Screenshot:



```
  GNU nano 7.2                                         prometheus.yml

global:
  scrape_interval: 5s

scrape_configs:
  - job_name: 'otel-collector'
    static_configs:
      - targets: ['otel-collector:8889']
```

9. Step 6 — Create Loki Config

```
nano loki-config.yaml
```

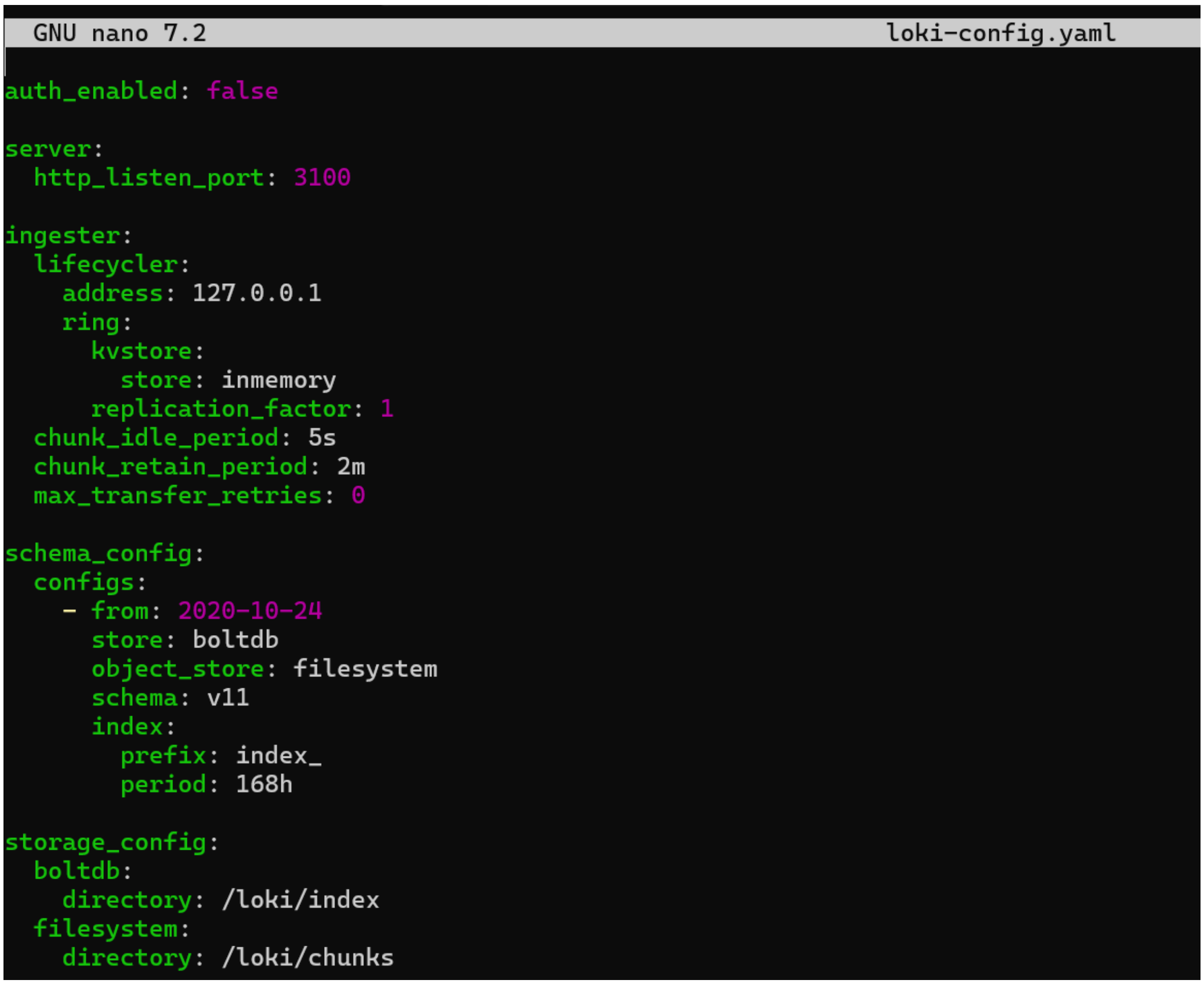
Paste:

```
auth_enabled: false
server:
http_listen_port: 3100
ingester:
chunk_idle_period: 5m
chunk_retain_period: 30s
schema_config:
configs:
- from: 2020-10-24
store: boltdb
object_store: filesystem
schema: v11
index:
prefix: index_
period: 168h
storage_config:
boltdb:
directory: /loki/index
filesystem:
directory: /loki/chunks
limits_config:
enforce_metric_name: false
```

Section / Key	**What It Does**
auth_enabled: false	Disables authentication (open access for testing/dev).
server: http_listen_port: 3100	Loki server listens on port 3100 for incoming log requests.
ingester:	Configuration for log ingestion.
chunk_idle_period: 5m	Ingested log chunks idle for 5 minutes before being closed.

Section / Key	**What It Does**
chunk_retain_period: 30s	Closed chunks are retained in memory for 30 seconds before being flushed to storage.
schema_config:	Defines log indexing schema.
configs: - from: 2020-10-24	Version/date for this schema configuration.
store: boltdb	Index storage engine (BoltDB).
object_store: filesystem	Stores log chunks on filesystem.
schema: v11	Schema version used by Loki.
index: prefix: index_	Prefix used for index files.
index: period: 168h	Index rotation period (168 hours = 7 days).
storage_config:	Storage locations for indexes and log chunks.
boltdb: directory: /loki/index	Directory for BoltDB index files.
filesystem: directory: /loki/chunks	Directory to store log chunks.
limits_config: enforce_metric_name: false	Allows logs without metric name enforcement (relaxed limit for logging).

Screenshot:



```
GNU nano 7.2                                                  loki-config.yaml

auth_enabled: false

server:
  http_listen_port: 3100

ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
  chunk_idle_period: 5s
  chunk_retain_period: 2m
  max_transfer_retries: 0

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 168h

storage_config:
  boltdb:
    directory: /loki/index
  filesystem:
    directory: /loki/chunks
```

10.Step 7 — Create docker-compose.yml

```
nano docker-compose.yml
```

Paste:

```
services:


java-app:
image: test:v5
container_name: java-app
depends_on:
- otel-collector
ports:
- "8080:8080"
logging:
```

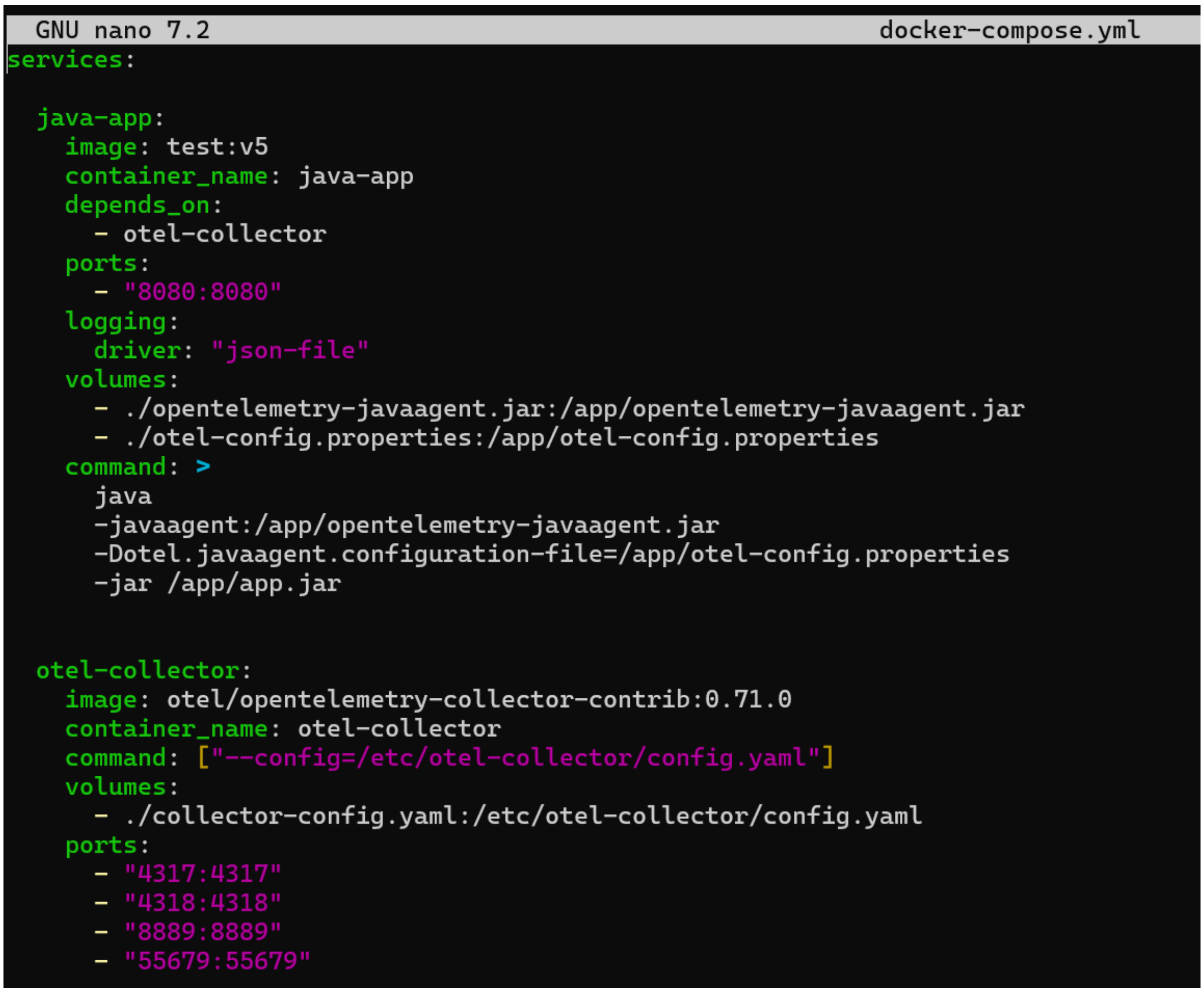
```
driver: "json-file"
volumes:
- ./opentelemetry-javaagent.jar:/app/opentelemetry-javaagent.jar
- ./otel-config.properties:/app/otel-config.properties
command: >
java
-javaagent:/app/opentelemetry-javaagent.jar
-Dotel.javaagent.configuration-file=/app/otel-config.properties
-jar /app/app.jar


otel-collector:
image: otel/opentelemetry-collector-contrib:0.71.0
container_name: otel-collector
command: ["--config=/etc/otel-collector/config.yaml"]
volumes:
- ./collector-config.yaml:/etc/otel-collector/config.yaml
ports:
- "4317:4317"
- "4318:4318"
- "8889:8889"
- "55679:55679"

jaeger:
image: jaegertracing/all-in-one
container_name: jaeger
ports:
- "16686:16686"
- "14250:14250"

prometheus:
image: prom/prometheus
container_name: prometheus
volumes:
- ./prometheus.yml:/etc/prometheus/prometheus.yml
ports:
- "9090:9090"
```

```
loki:
image: grafana/loki:2.9.0
container_name: loki
ports:
- "3100:3100"
volumes:
- ./loki-config.yaml:/etc/loki/local-config.yaml
- ./loki-data:/loki       # Persistent storage
- ./wal:/wal              # WAL directory
command: ["-config.file=/etc/loki/local-config.yaml"]

grafana:
image: grafana/grafana
container_name: grafana
ports:
- "3000:3000"
```

Service	Image / Container	Ports	Purpose / What It Does
java-app	test:v5	8080:8080	Runs your Java application with OpenTelemetry agent; sends traces, metrics, logs to OTEL Collector.
otel-collector	otel/opentelemetry-collector-contrib:0.71.0	4317, 4318, 8889, 55679	Receives telemetry from apps and routes to Jaeger, Prometheus, Loki.
jaeger	jaegertracing/all-in-one	16686, 14250	Collects and visualizes traces from OTEL Collector.
prometheus	prom/prometheus	9090	Scrapes and stores metrics from OTEL Collector.
loki	grafana/loki:2.9.0	3100	Receives and stores logs from OTEL Collector; supports querying from Grafana.

Service	Image / Container	Ports	Purpose / What It Does
grafana	grafana/grafana	3000	Visualizes traces, metrics, and logs; connects to Jaeger, Prometheus, and Loki.

Screenshot

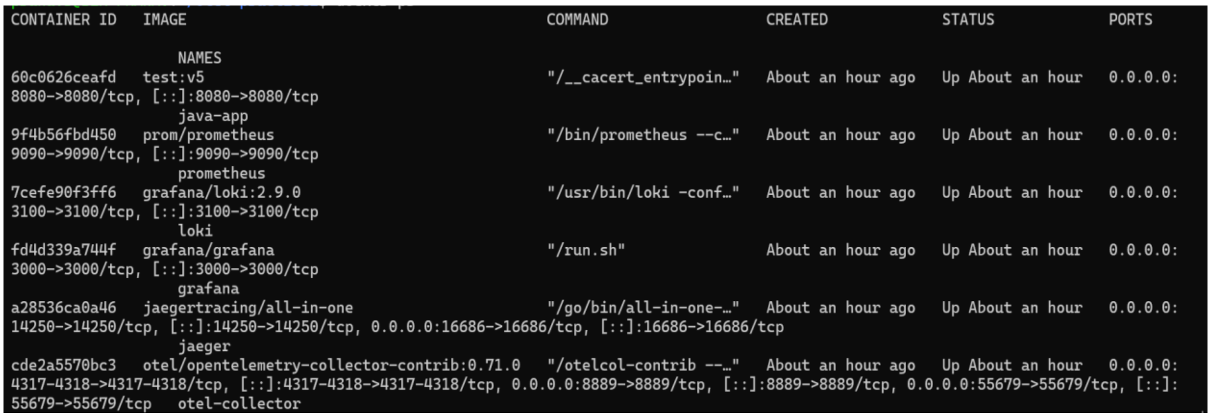


```
  GNU nano 7.2                                            docker-compose.yml
services:

  java-app:
    image: test:v5
    container_name: java-app
    depends_on:
      - otel-collector
    ports:
      - "8080:8080"
    logging:
      driver: "json-file"
    volumes:
      - ./opentelemetry-javaagent.jar:/app/opentelemetry-javaagent.jar
      - ./otel-config.properties:/app/otel-config.properties
    command: >
      java
      -javaagent:/app/opentelemetry-javaagent.jar
      -Dotel.javaagent.configuration-file=/app/otel-config.properties
      -jar /app/app.jar


  otel-collector:
    image: otel/opentelemetry-collector-contrib:0.71.0
    container_name: otel-collector
    command: ["--config=/etc/otel-collector/config.yaml"]
    volumes:
      - ./collector-config.yaml:/etc/otel-collector/config.yaml
    ports:
      - "4317:4317"
      - "4318:4318"
      - "8889:8889"
      - "55679:55679"
```

11.Step 8 — Start all services and Test everything

Commands:

```
docker compose up --build

docker ps
```



```
CONTAINER ID   IMAGE                                        COMMAND                  CREATED             STATUS             PORTS

                    NAMES
60c0626ceafd   test:v5                                      "/__cacert_entrypoin…"   About an hour ago   Up About an hour   0.0.0.0:
8080->8080/tcp, [::]:8080->8080/tcp
                    java-app
9f4b56fbd450   prom/prometheus                              "/bin/prometheus --c…"   About an hour ago   Up About an hour   0.0.0.0:
9090->9090/tcp, [::]:9090->9090/tcp
                    prometheus
7cefe90f3ff6   grafana/loki:2.9.0                           "/usr/bin/loki -conf…"   About an hour ago   Up About an hour   0.0.0.0:
3100->3100/tcp, [::]:3100->3100/tcp
                    loki
fd4d339a744f   grafana/grafana                              "/run.sh"                About an hour ago   Up About an hour   0.0.0.0:
3000->3000/tcp, [::]:3000->3000/tcp
                    grafana
a28536ca0a46   jaegertracing/all-in-one                     "/go/bin/all-in-one-…"   About an hour ago   Up About an hour   0.0.0.0:
14250->14250/tcp, [::]:14250->14250/tcp, 0.0.0.0:16686->16686/tcp, [::]:16686->16686/tcp
                    jaeger
cde2a5570bc3   otel/opentelemetry-collector-contrib:0.71.0   "/otelcol-contrib --…"   About an hour ago   Up About an hour   0.0.0.0:
4317-4318->4317-4318/tcp, [::]:4317-4318->4317-4318/tcp, 0.0.0.0:8889->8889/tcp, [::]:8889->8889/tcp, 0.0.0.0:55679->55679/tcp, [::]:
55679->55679/tcp   otel-collector
```

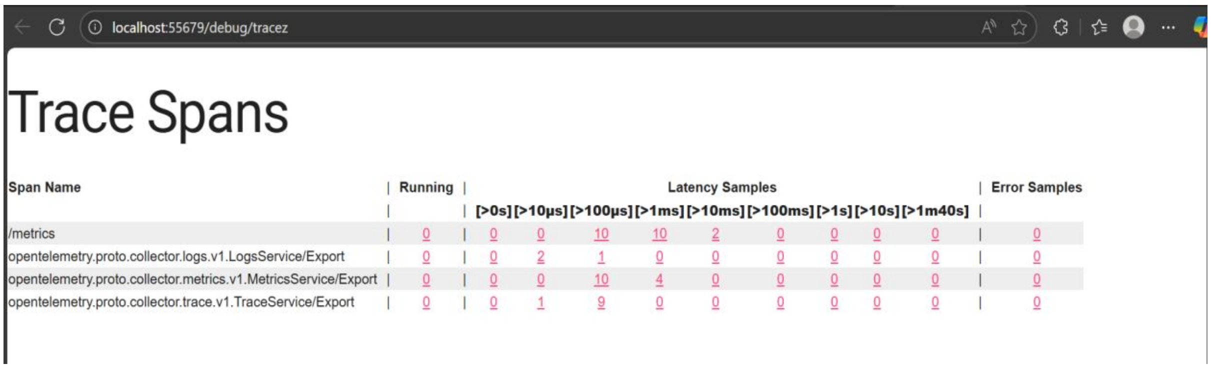
Test Java App

```
curl http://localhost:8080
```

Test Tracez Debug Page (zpages)

Open browser:

http://localhost:55679/debug/tracez



localhost:55679/debug/tracez

Trace Spans

Span Name	Running	Latency Samples									Error Samples
		[>0s]	[>10μs]	[>100μs]	[>1ms]	[>10ms]	[>100ms]	[>1s]	[>10s]	[>1m40s]	
/metrics	0	0	0	10	10	2	0	0	0	0	0
opentelemetry.proto.collector.logs.v1.LogsService/Export	0	0	2	1	0	0	0	0	0	0	0
opentelemetry.proto.collector.metrics.v1.MetricsService/Export	0	0	0	10	4	0	0	0	0	0	0
opentelemetry.proto.collector.trace.v1.TraceService/Export	0	0	1	9	0	0	0	0	0	0	0

Test Loki Push

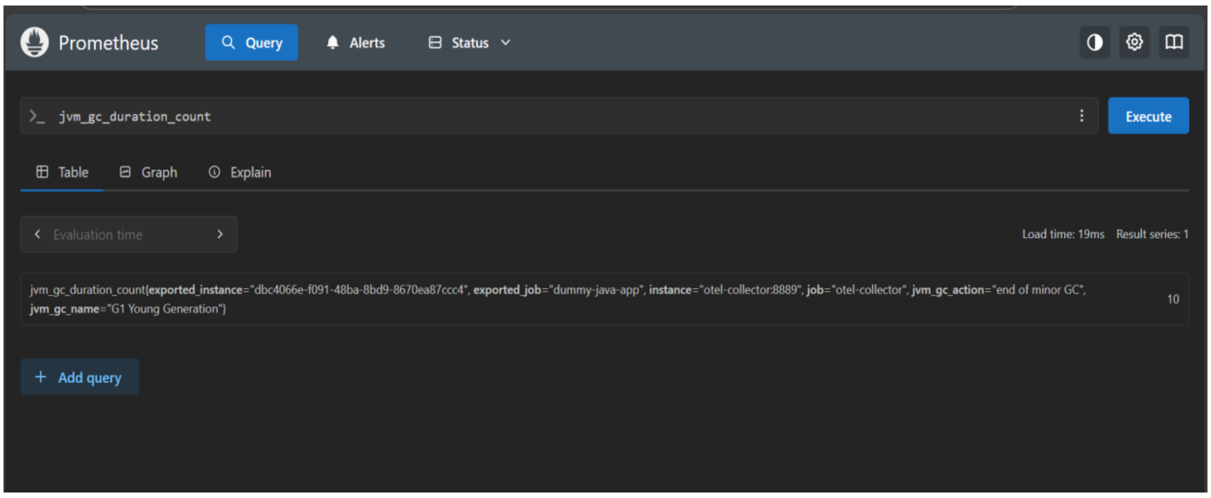
```
curl -X POST "http://localhost:3100/loki/api/v1/push" \

-H "Content-Type: application/json" \

-d '{

"streams": [
{
"stream": {"job":"test","level":"info"},
"values": [[ "'$(date +%s%N)'", "Hello from Linux curl" ]]
}
]
}'
```

Test Prometheus UI

http://localhost:9090

Query: jvm_gc_duration_count

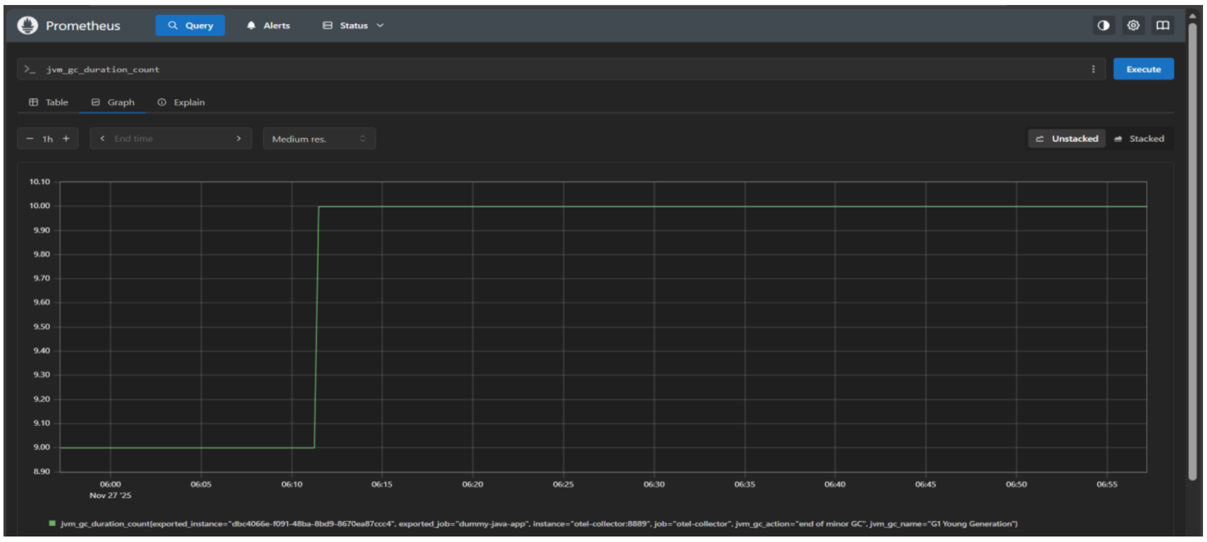
Screenshot



```
jvm_gc_duration_count
```

Load time: 19ms Result series: 1

```
jvm_gc_duration_count{exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-app", instance="otel-collector:8889", job="otel-collector", jvm_gc_action="end of minor GC", jvm_gc_name="G1 Young Generation"}    10
```

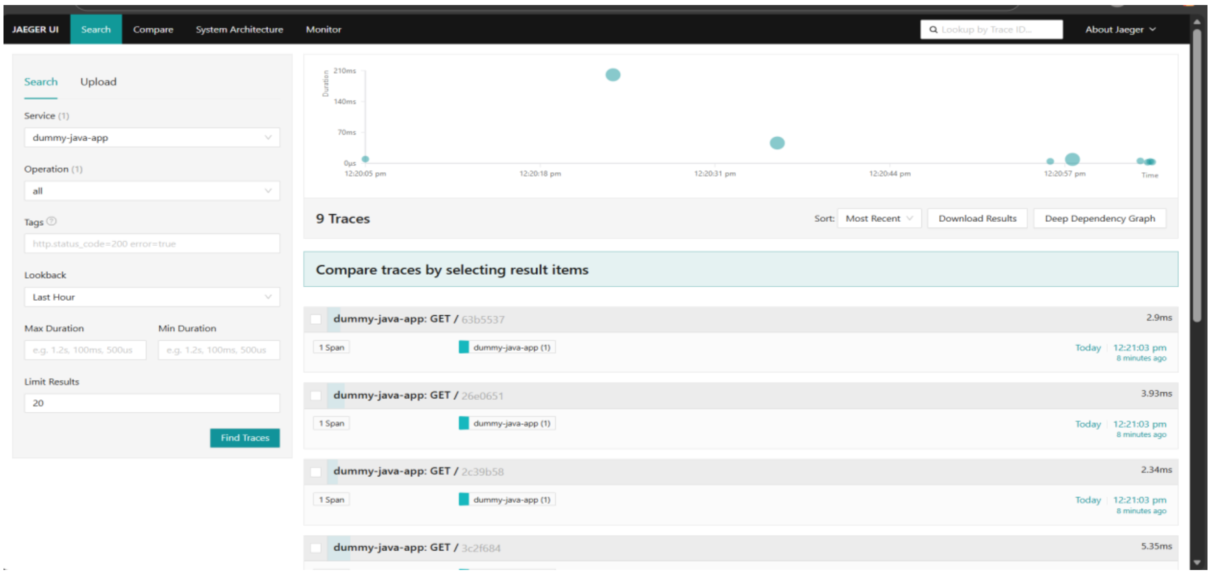


```
jvm_gc_duration_count
```

🟩 jvm_gc_duration_count{exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-app", instance="otel-collector:8889", job="otel-collector", jvm_gc_action="end of minor GC", jvm_gc_name="G1 Young Generation"}

Test Jaeger UI

http://localhost:16686



Service (1): dummy-java-app

Operation (1): all

Tags: http.status_code=200 error=true

Lookback: Last Hour

Max Duration: e.g. 1.2s, 100ms, 500us

Min Duration: e.g. 1.2s, 100ms, 500us

Limit Results: 20

9 Traces

Sort: Most Recent

Compare traces by selecting result items

dummy-java-app: GET / 63b5537 — 2.9ms — 1 Span — dummy-java-app (1) — Today | 12:21:03 pm, 8 minutes ago

dummy-java-app: GET / 26e0651 — 3.93ms — 1 Span — dummy-java-app (1) — Today | 12:21:03 pm, 8 minutes ago

dummy-java-app: GET / 2c39b58 — 2.34ms — 1 Span — dummy-java-app (1) — Today | 12:21:03 pm, 8 minutes ago

dummy-java-app: GET / 3c2f684 — 5.35ms

12.Setup Grafana

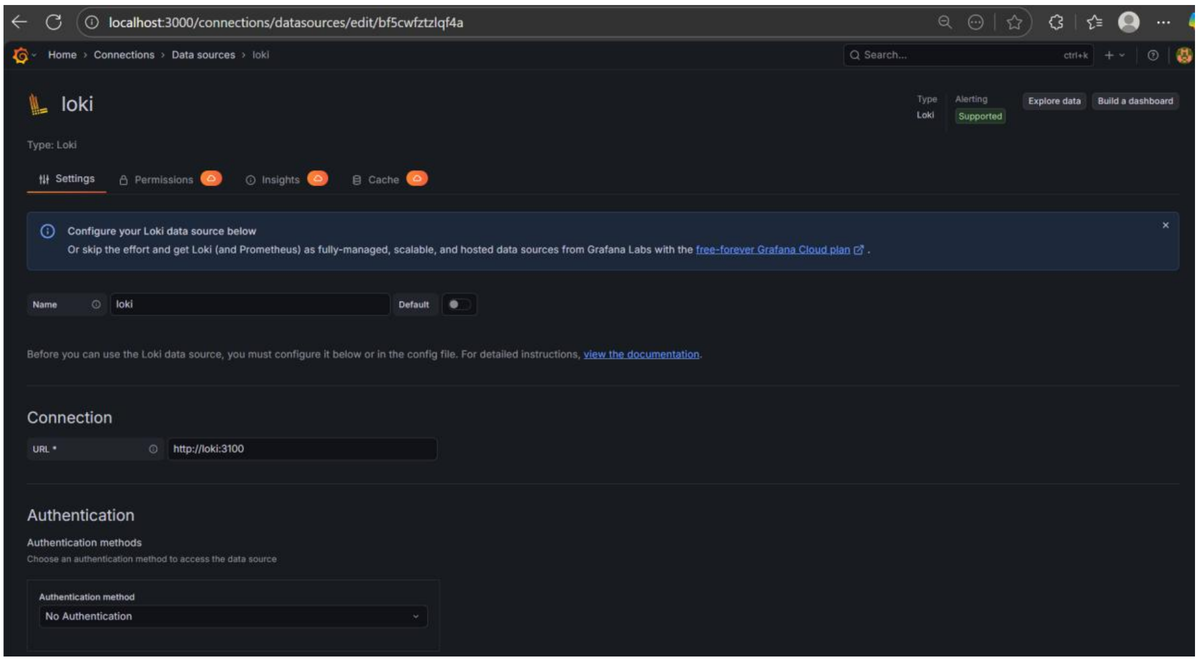
Open:

http://localhost:3000

Login: admin / admin

Add Loki Datasource

URL: http://loki:3100



localhost:3000/connections/datasources/edit/bf5cwfztzlqf4a

loki

Type: Loki

Type: Loki | Alerting: Supported

Configure your Loki data source below

Or skip the effort and get Loki (and Prometheus) as fully-managed, scalable, and hosted data sources from Grafana Labs with the free-forever Grafana Cloud plan .

Name: loki | Default

Before you can use the Loki data source, you must configure it below or in the config file. For detailed instructions, view the documentation.

Connection

URL *: http://loki:3100

Authentication

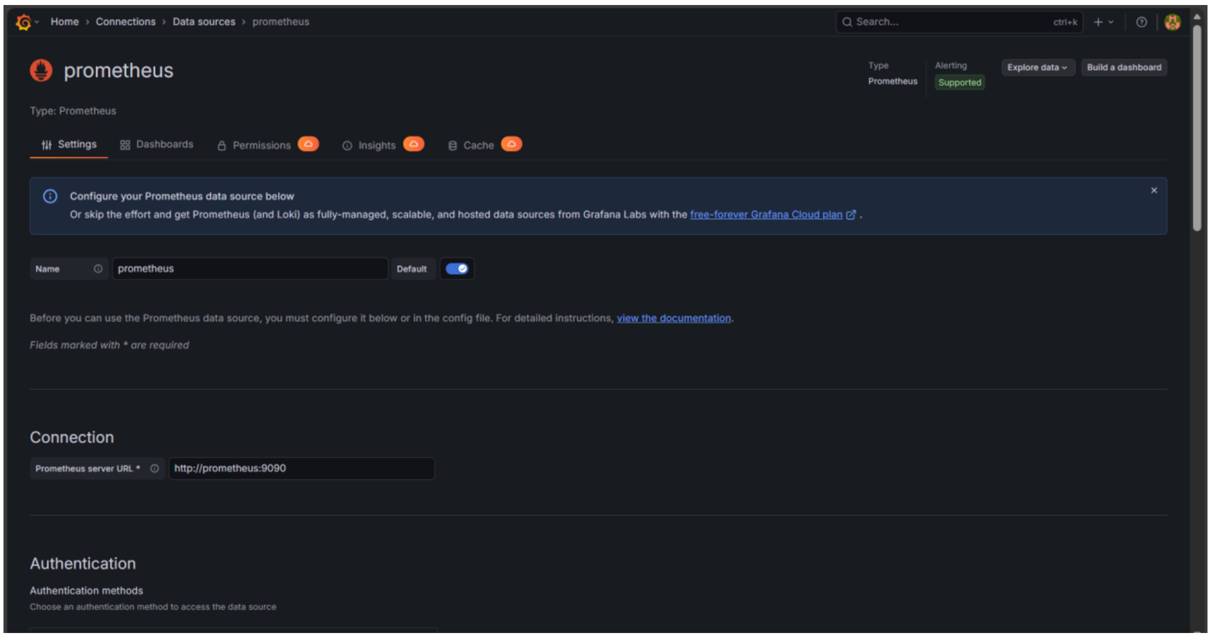
Authentication methods

Choose an authentication method to access the data source

Authentication method: No Authentication

Add Prometheus Datasource

URL: http://prometheus:9090


Home › Connections › Data sources › prometheus

Search... ctrl+k

prometheus

Type: Prometheus

Type Prometheus | Alerting Supported | Explore data | Build a dashboard

Settings | Dashboards | Permissions | Insights | Cache

Configure your Prometheus data source below
Or skip the effort and get Prometheus (and Loki) as fully-managed, scalable, and hosted data sources from Grafana Labs with the free-forever Grafana Cloud plan .

Name: prometheus | Default: on

Before you can use the Prometheus data source, you must configure it below or in the config file. For detailed instructions, view the documentation.

*Fields marked with * are required*

Connection

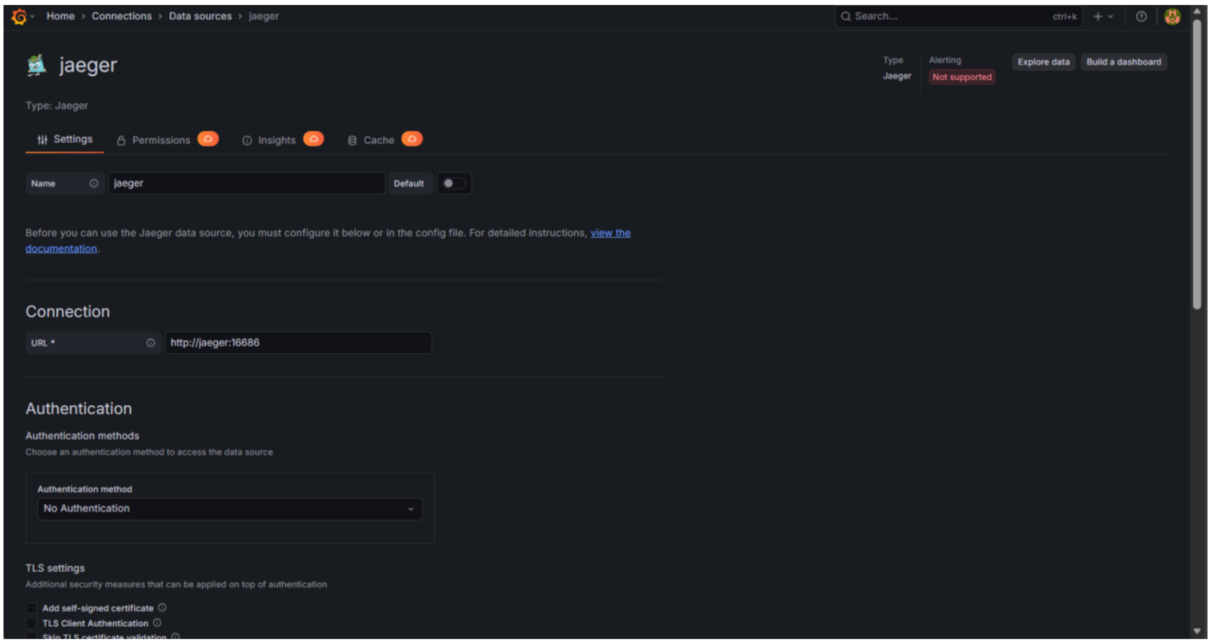
Prometheus server URL *: http://prometheus:9090

Authentication

Authentication methods
Choose an authentication method to access the data source

Add Jaeger Datasource

URL: http://jaeger:16686


Home › Connections › Data sources › jaeger

Search... ctrl+k

jaeger

Type: Jaeger

Type Jaeger | Alerting Not supported | Explore data | Build a dashboard

Settings | Permissions | Insights | Cache

Name: jaeger | Default: off

Before you can use the Jaeger data source, you must configure it below or in the config file. For detailed instructions, view the documentation.

Connection

URL *: http://jaeger:16686

Authentication

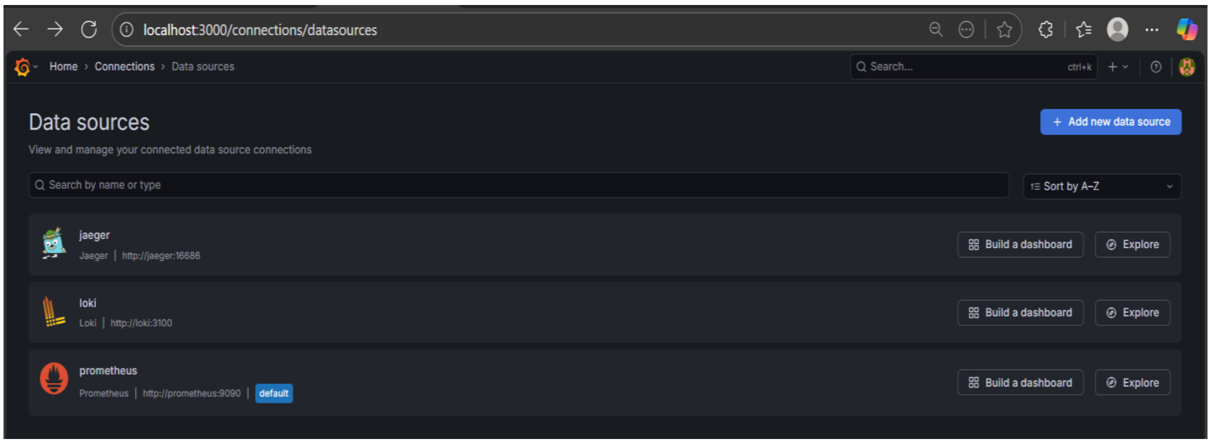
Authentication methods
Choose an authentication method to access the data source

Authentication method: No Authentication

TLS settings
Additional security measures that can be applied on top of authentication

- Add self-signed certificate
- TLS Client Authentication
- Skip TLS certificate validation

Overall Datasources


localhost:3000/connections/datasources

Home › Connections › Data sources

Data sources

View and manage your connected data source connections

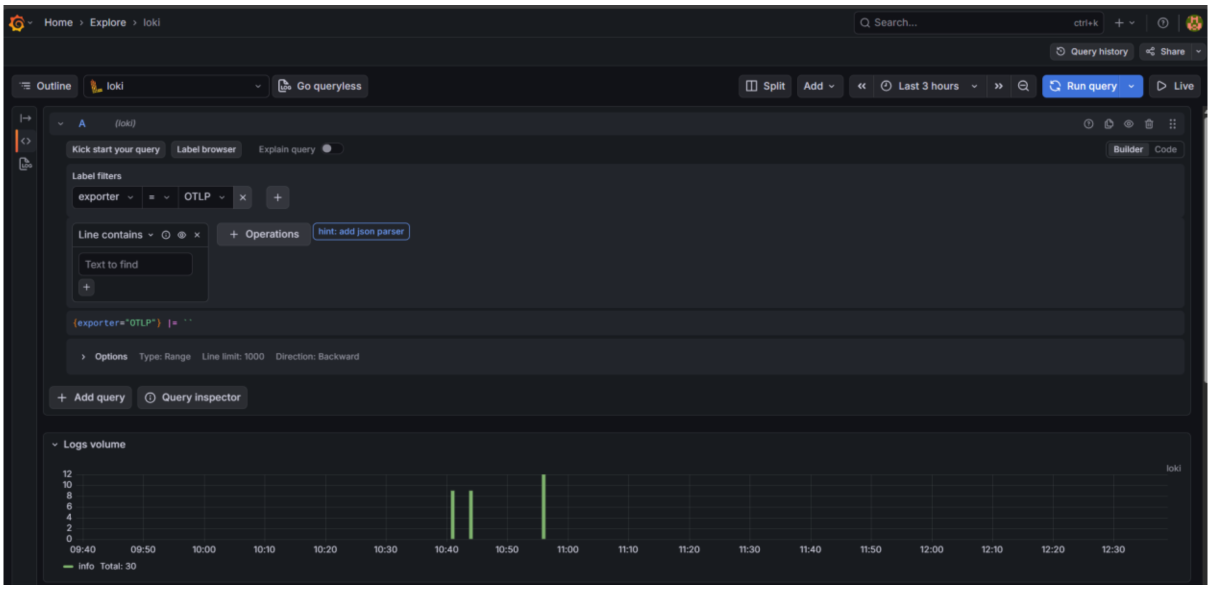
jaeger	Jaeger | http://jaeger:16686
loki	Loki | http://loki:3100
prometheus	Prometheus | http://prometheus:9090 | default

13.Verify Everything

i) Logs (Grafana → Explore → Loki)

Label Filters

Screenshot


Home › Explore › loki

A (loki)

Label filters: exporter = OTLP

Line contains: Text to find

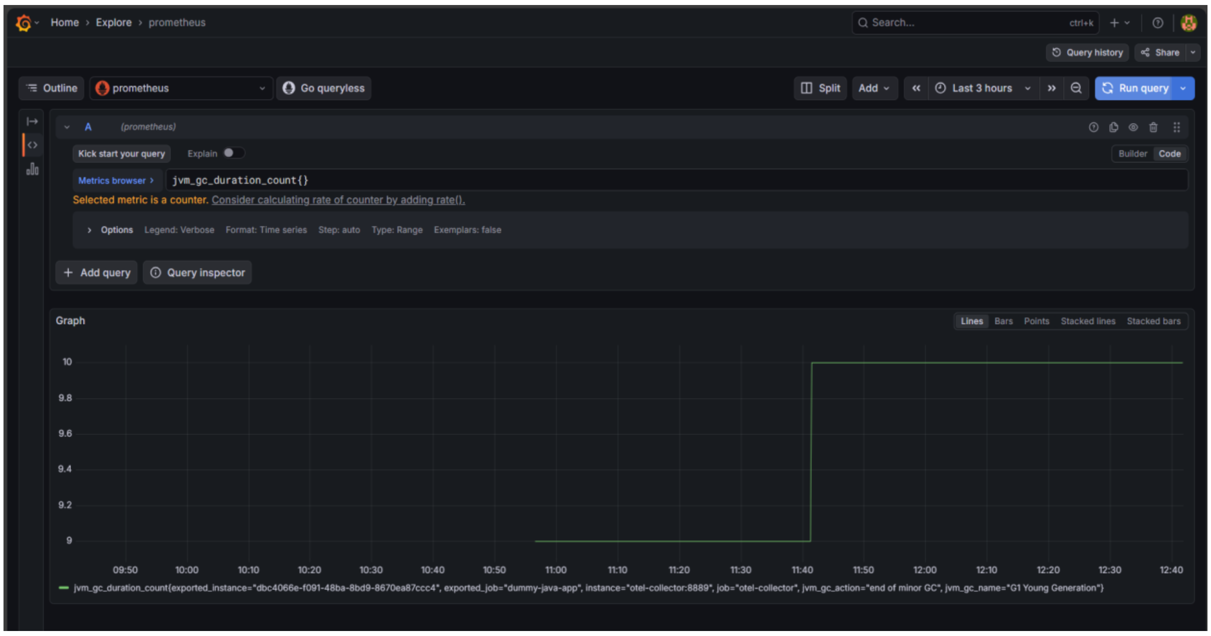
```
{exporter="OTLP"} |= ``
```

Options Type: Range Line limit: 1000 Direction: Backward

Logs volume

info Total: 30

ii) Metrics (Grafana → Explore → Prometheus)



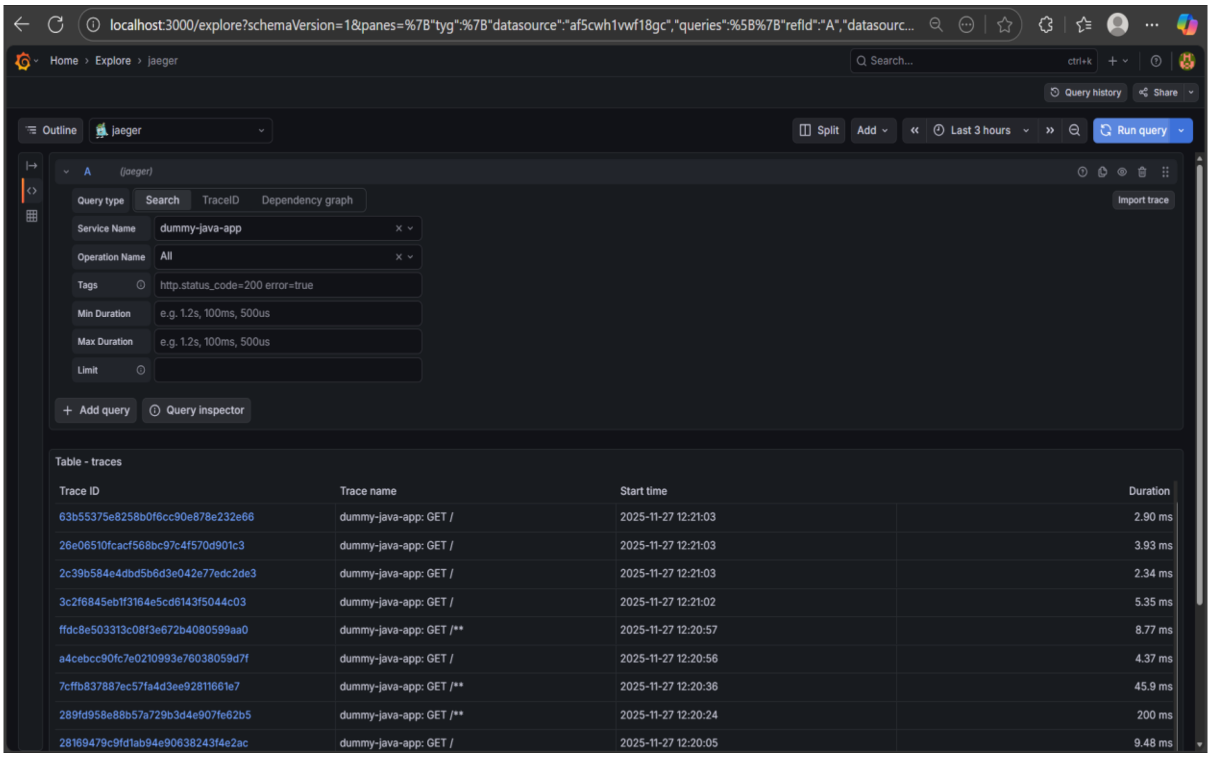
```
jvm_gc_duration_count{}
```

Selected metric is a counter. Consider calculating rate of counter by adding rate().

Options Legend: Verbose Format: Time series Step: auto Type: Range Exemplars: false

jvm_gc_duration_count{exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-app", instance="otel-collector:8889", job="otel-collector", jvm_gc_action="end of minor GC", jvm_gc_name="G1 Young Generation"}

iii) Traces (Grafana → Explore → Traces)



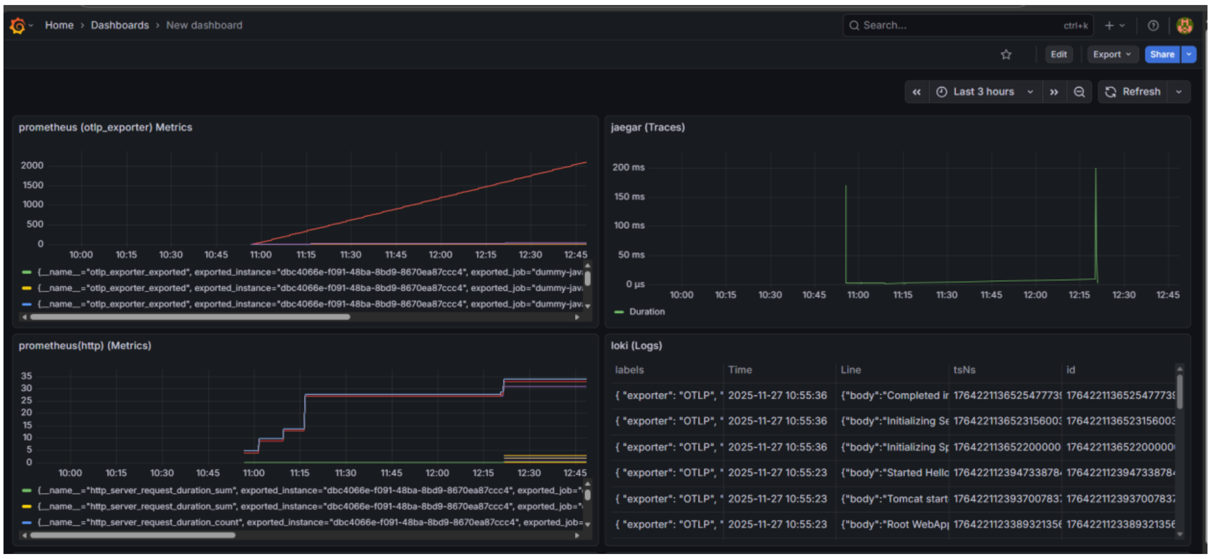
localhost:3000/explore?schemaVersion=1&panes=%7B"tyg":%7B"datasource":"af5cwh1vwf18gc","queries":%5B%7B"refId":"A","datasourc...

Query type	Search
Service Name	dummy-java-app
Operation Name	All
Tags	http.status_code=200 error=true
Min Duration	e.g. 1.2s, 100ms, 500us
Max Duration	e.g. 1.2s, 100ms, 500us
Limit	

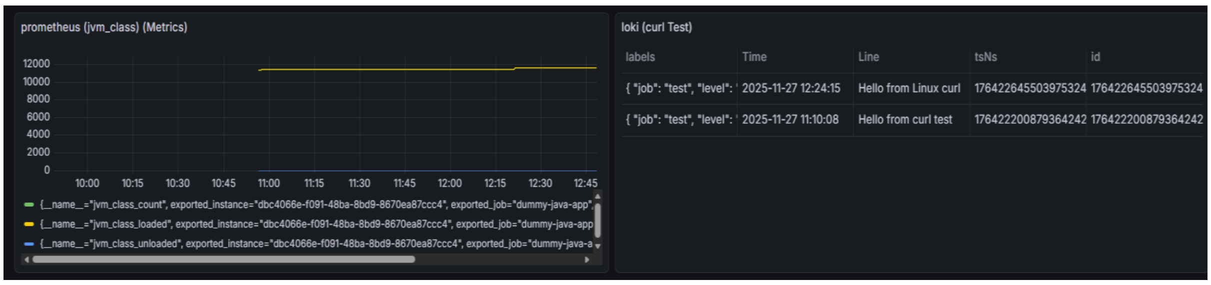
Table - traces

Trace ID	Trace name	Start time	Duration
63b55375e8258b0f6cc90e878e232e66	dummy-java-app: GET /	2025-11-27 12:21:03	2.90 ms
26e06510fcacf568bc97c4f570d901c3	dummy-java-app: GET /	2025-11-27 12:21:03	3.93 ms
2c39b584e4dbd5b6d3e042e77edc2de3	dummy-java-app: GET /	2025-11-27 12:21:03	2.34 ms
3c2f6845eb1f3164e5cd6143f5044c03	dummy-java-app: GET /	2025-11-27 12:21:02	5.35 ms
ffdc8e503313c08f3e672b4080599aa0	dummy-java-app: GET /**	2025-11-27 12:20:57	8.77 ms
a4cebcc90fc7e0210993e76038059d7f	dummy-java-app: GET /	2025-11-27 12:20:56	4.37 ms
7cffb837887ec57fa4d3ee92811661e7	dummy-java-app: GET /**	2025-11-27 12:20:36	45.9 ms
289fd958e88b57a729b3d4e907fe62b5	dummy-java-app: GET /**	2025-11-27 12:20:24	200 ms
28169479c9fd1ab94e90638243f4e2ac	dummy-java-app: GET /	2025-11-27 12:20:05	9.48 ms

14.Grafana Dashboard (Optional)


Home › Dashboards › New dashboard
Search... ctrl+k
Edit
Export
Share
Last 3 hours
Refresh
prometheus (otlp_exporter) Metrics
{__name__="otlp_exporter_exported", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-jav
{__name__="otlp_exporter_exported", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-jav
{__name__="otlp_exporter_exported", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-jav
jaegar (Traces)
Duration
prometheus(http) (Metrics)
{__name__="http_server_request_duration_sum", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="
{__name__="http_server_request_duration_sum", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="
{__name__="http_server_request_duration_count", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job=
loki (Logs)

labels	Time	Line	tsNs	id
{ "exporter": "OTLP", "	2025-11-27 10:55:36	{"body":"Completed ir	17642211365254777З	17642211365254777З
{ "exporter": "OTLP", "	2025-11-27 10:55:36	{"body":"Initializing Se	176422113652315600	176422113652315600
{ "exporter": "OTLP", "	2025-11-27 10:55:36	{"body":"Initializing Sp	176422113652200000	176422113652200000
{ "exporter": "OTLP", "	2025-11-27 10:55:23	{"body":"Started Hello	176422112394733878	176422112394733878
{ "exporter": "OTLP", "	2025-11-27 10:55:23	{"body":"Tomcat start	176422112393700783	176422112393700783
{ "exporter": "OTLP", "	2025-11-27 10:55:23	{"body":"Root WebApp	176422112338932135	176422112338932135


prometheus (jvm_class) (Metrics)
{__name__="jvm_class_count", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-app",
{__name__="jvm_class_loaded", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-app
{__name__="jvm_class_unloaded", exported_instance="dbc4066e-f091-48ba-8bd9-8670ea87ccc4", exported_job="dummy-java-a
loki (curl Test)

labels	Time	Line	tsNs	id
{ "job": "test", "level": '	2025-11-27 12:24:15	Hello from Linux curl	176422645503975324	176422645503975324
{ "job": "test", "level": '	2025-11-27 11:10:08	Hello from curl test	176422200879364242	176422200879364242

SUMMARY

Your setup does the following:

I) Java app emits traces, metrics, and logs via OpenTelemetry agent.

II) OTEL Collector receives all three and routes:

traces → Jaeger

metrics → Prometheus

logs → Loki

III) Grafana connects to Loki/Prometheus/Jaeger to display all data.