

# Micrometer Mastery: Unleash Advanced Observability In Your JVM Apps

Link	<a href="https://www.youtube.com/watch?v=X7rODR2m63c">https://www.youtube.com/watch?v=X7rODR2m63c</a>
Author(s)	Jonatan Ivanov
Length	44:32
Date	22-09-2024
Language	English ! "
Rating	#####

¥ \$ É

¥ % É

¥ % É

¥ % É

"É"

Today's systems are increasingly complex.

Why do we need observability?

- ¥ Environments can be chaotic: You turn a knob here a little and apps are going down there.
- ¥ We need to know with unknown unknowns.
- ¥ Things can be perceived differently by observers.

Why do we need observability (from the business perspective)?

- ¥ Reduce lost revenue from production incidents: Lower mean time to recovery (MTTR).
- ¥ Require less specialized knowledge: Shared method of investigating across systems.
- ¥ Quantify user experience: Don't guess, measure!

## JVM/Spring

- ¥ Logging: Logging with JVM/Spring: Slf4j + Logback.Spring provides starters:
  - & Logback: `spring-boot-starter-logging`
  - & Log4j2: `spring-boot-starter-log4j2`

¥ Metrics: Spring projects are instrumented using Micrometer that also supports many backends and its API is independent of the configured metrics behind. Spring comes with `spring-boot-actuator`.

¥ Distributed Tracing: There depends on the Spring Boot version:

- & Spring Boot 2.x: Use Spring Cloud Sleuth.

- & Spring Boot 3.x: Use Micrometer Tracing (it is basically Sleuth without Spring dependencies) which is a tracing facade. Tracing libraries supported:

- ' Brave (OpenZipkin) which is default.

- ' OpenTelemetry (CNCF) which is experimental.

## Demo

Demo source code is available at [GitHub](#).

In distributed tracing a span is an event we want to observe and the spans are all connected (one span can trigger another one) and grouped together as a trace.

We can jump from logs to traces and back.

Unable to render diagram. To render diagrams, go to the AsciiDoc plugin's settings and download extensions or enable Kroki.

## Observation API

We can use error attributes to create a query in the metric system.

```
sum(rate(http_server_requests_seconds_count{application="tea-service",
exception="NotFound",method="GET",org="teahouse",outcome="SERVER_ERROR",
status="500",uri="/tea/{name}"}[$__rate_interval]))
```

## Exemplars

How to jump from metrics to tracing, which is a hard problem to tackle because during disaggregation we are losing data: We have 1 million events, and we aggregated them into a single number error rate equals 0.5 - how to jump from the single number 0.5 to one of there 1 million events?

It's impossible, that's why there is a concept called exemplars that are metadata attached to the metric values.

Whenever we do a recording, we can sample the distributed tracing library and ask for the Span ID and Trace ID and wrap them into this concept and attach them to the metric values to look at them during aggregation.

Metrics

Distributed Tracing