Demystifying Observability

for Spring applications

Tiffany Jernigan
Senior Developer Advocate, Grafana Labs

Timo Salm
Master Solutions Architect, Broadcom

April 2026

Grafana Labs & vmware by Broadcom

Who we are

Tiffany Jernigan

Senior Developer Advocate
Grafana Labs

X (Twitter): @tiffanyfayj
Bluesky: @tiffanyfay.dev
LinkedIn: linkedin.com/in/tiffanyfayj
Website: tiffanyfay.dev

Timo Salm

Master Solutions Architect
Broadcom

X (Twitter): @salmto
Bluesky: @timosalm.bsky.social
LinkedIn: linkedin.com/in/timosalm

What We Mean by Observability

The ability to understand a system's internal state from the outside by examining its outputs

When a system fails at 2 AM, you need to understand why — fast!

Monitoring tells you something is wrong.
Observability tells you why!

The ~~Three~~ Four Singals of Observability

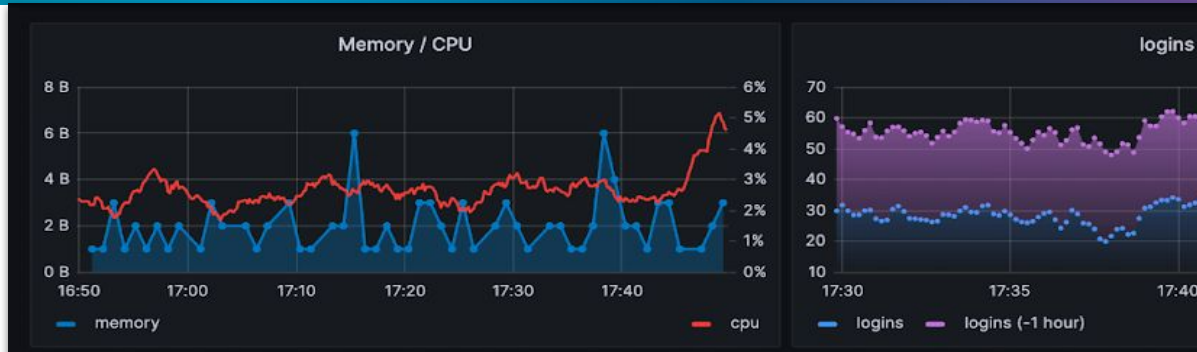
- Metrics
- Logs
- Traces
- Profiles

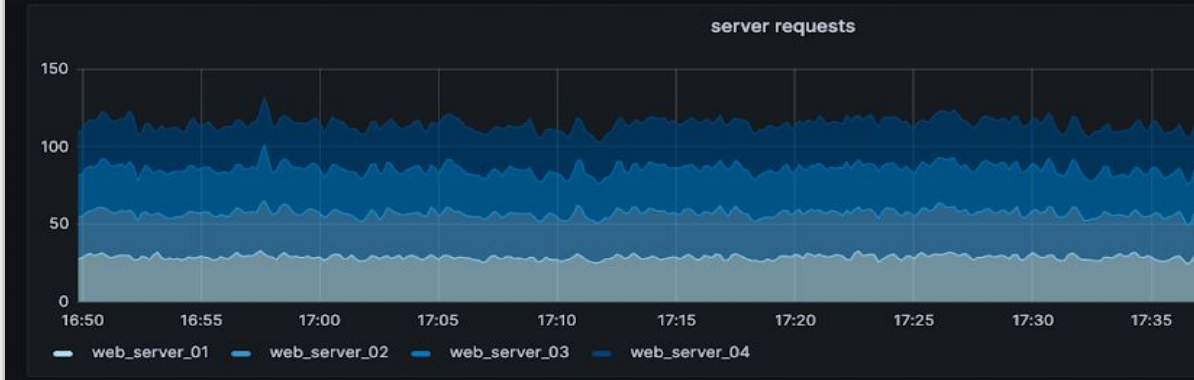
Metrics

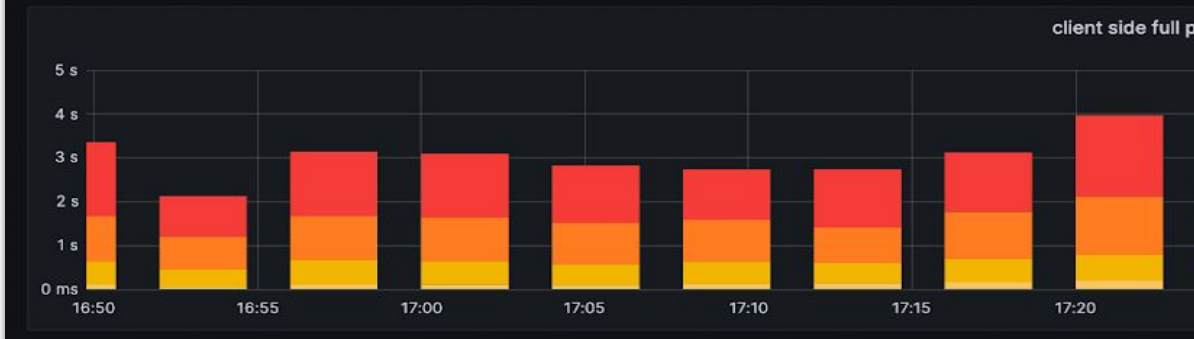
Numeric measurements aggregated over time

Used to track the state and performance of systems

Common uses: CPU, memory, request rates, error rates, latency percentiles, business metrics







Grafana Labs & vmware by Broadcom

Logs

Timestamped, immutable
records of events

They are best for answering: *"What exactly happened?"*

There are multiple log levels to balance visibility and noise: DEBUG for dev/test, INFO for production, WARN for issues, and ERROR for failures.

Log Types

Structured (JSON): key-value pairs, machine-readable, queryable

Unstructured text: legacy format — hard to parse at scale

Audit / Access: who did what and when — security-relevant

```
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.889  INFO --- [    schedu
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.888  INFO --- [    schedu
2026-04-14 08:58:24.085  ERROR  2026-04-14 08:58:23.888 ERROR --- [    schedu
2026-04-14 08:58:24.085  ERROR  2026-04-14 08:58:23.888 ERROR --- [    schedu
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.888  INFO --- [    schedu
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.888  INFO --- [    schedu
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.888  INFO --- [    schedu
2026-04-14 08:58:24.085  INFO   2026-04-14 08:58:23.888  INFO --- [    schedu
```

Traces

Debugging distributed applications is hard!

A request's full path,
span by span

A trace is made of spans, each representing one operation.

Traces answer: *"Where is the latency/error happening?"*

Continuous Profiling

Captures resource consumption at code level

A trace shows 800ms execution time of the `processOrder()` method. But that's 200 lines.

Profiling answers: *"Why is this code path expensive or slow?"*

Use profiling in dev and continuous profiling in prod (deep analysis vs. continuous, low-overhead)

Symbol	Self ↓	Total
.[vdso]	16.0 mins	16.0 mins
jdk/inte...	8.73 mins	29.6 mins
java/ti...	4.72 mins	35.3 mins
libjvm.s...	2.24 mins	20.7 mins
./usr/lib...	1.47 mins	18.0 mins
libc.so....	1.01 mins	16.7 mins
java/lan...	38.2 s	38.2 s
java/ti...	28.2 s	1.59 mins
java/ti...	27.9 s	27.9 s
java/util...	23.0 s	23.0 s
io/pyro...	18.7 s	18.7 s
java/ti...	17.7 s	26.2 s
java/lan...	8.51 s	8.51 s
libjvm.s...	8.12 s	8.12 s
./usr/lib...	3.05 s	8.86 s
libc.so....	1.65 s	1.65 s
libc.so....	1.55 s	1.55 s
java/lan...	940 ms	940 ms

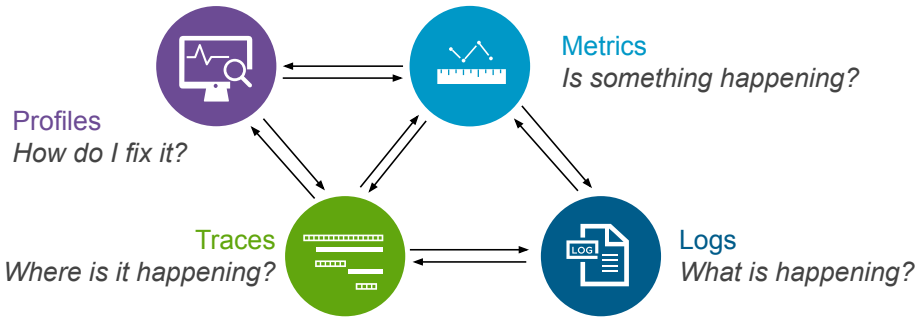
How the Signals Work Together

No single signal tells the whole story

Each answers a different question in the debugging workflow

Correlate them and mean time to resolution drops dramatically!

Correlation connects signals through shared labels, trace IDs, and data source links



Debugging Workflow

1. **Metrics:** alert fires — p99 latency > 2s
2. **Logs:** the SQL query and its exact parameters
3. **Traces:** find the slow span — DB call taking 1.8s
4. **Profiles:** flame graph shows N+1 query in a loop
5. **Fix:** add a JOIN, deploy, watch metrics recover

OpenTelemetry

OpenTelemetry emerged from the merger of OpenTracing and OpenCensus (Metrics) in 2019.

It is now the second most active CNCF project, behind Kubernetes

One SDK, multiple backends!

Instrument once, route to Datadog, Jaeger, Grafana, or Prometheus — via configuration.

Current state:
OTLP is stable for traces, metrics, and logs
Logs data model is stable
Profiles are still maturing

History of Observability Standards

Every major tool had its own agent, its own format. Vendor lock-in was the norm

2010 **ELK stack** Centralized logs, StatsD for metrics

2015 **OpenTracing** First vendor-neutral tracing API

2016 **OpenCensus** Google's metrics + tracing library

2018 **Prometheus GA** De-facto metrics standard

2019 **OpenTelemetry**

2021 **OTel signals** Traces → Metrics → Logs stabilized

2024+ **Profiling + eBPF** OTel Profiling experimental

OpenTelemetry in Spring Boot

Spring Boot 3.x made observability first-class via Micrometer and auto-configuration

Three options:
- OpenTelemetry Java Agent for zero-code instrumentation
- Community starter: `opentelemetry-spring-boot-starter`
- Spring's built-in Micrometer-based observability support with OTLP export

Spring Boot 4 goes further

Spring team now provides its own official OpenTelemetry starter (`spring-boot-starter-opentelemetry`), making OpenTelemetry integration much more first-class and streamlined.

Spring Boot 4.1 will include OTLP exemplar support (via Micrometer)

Micrometer — The Instrumentation Facade

Micrometer provides a facade for the most popular observability systems

Allows you to instrument your JVM-based application code without vendor lock-in

Use `MeterRegistry` when you want to record a specific metric: increment a counter, record a timer, publish a gauge.

```
meterRegistry.counter("orders.created").increment()
```

Use the `Observation` API when you want to describe a business or technical operation and let the observability setup decide whether that becomes metrics, traces, or both

```
Observation.createNotStarted("order.checkout",
        observationRegistry)
```

Logging in Spring Boot

Spring Boot uses a flexible logging abstraction based on Apache Commons Logging keeps the logging backend configurable.

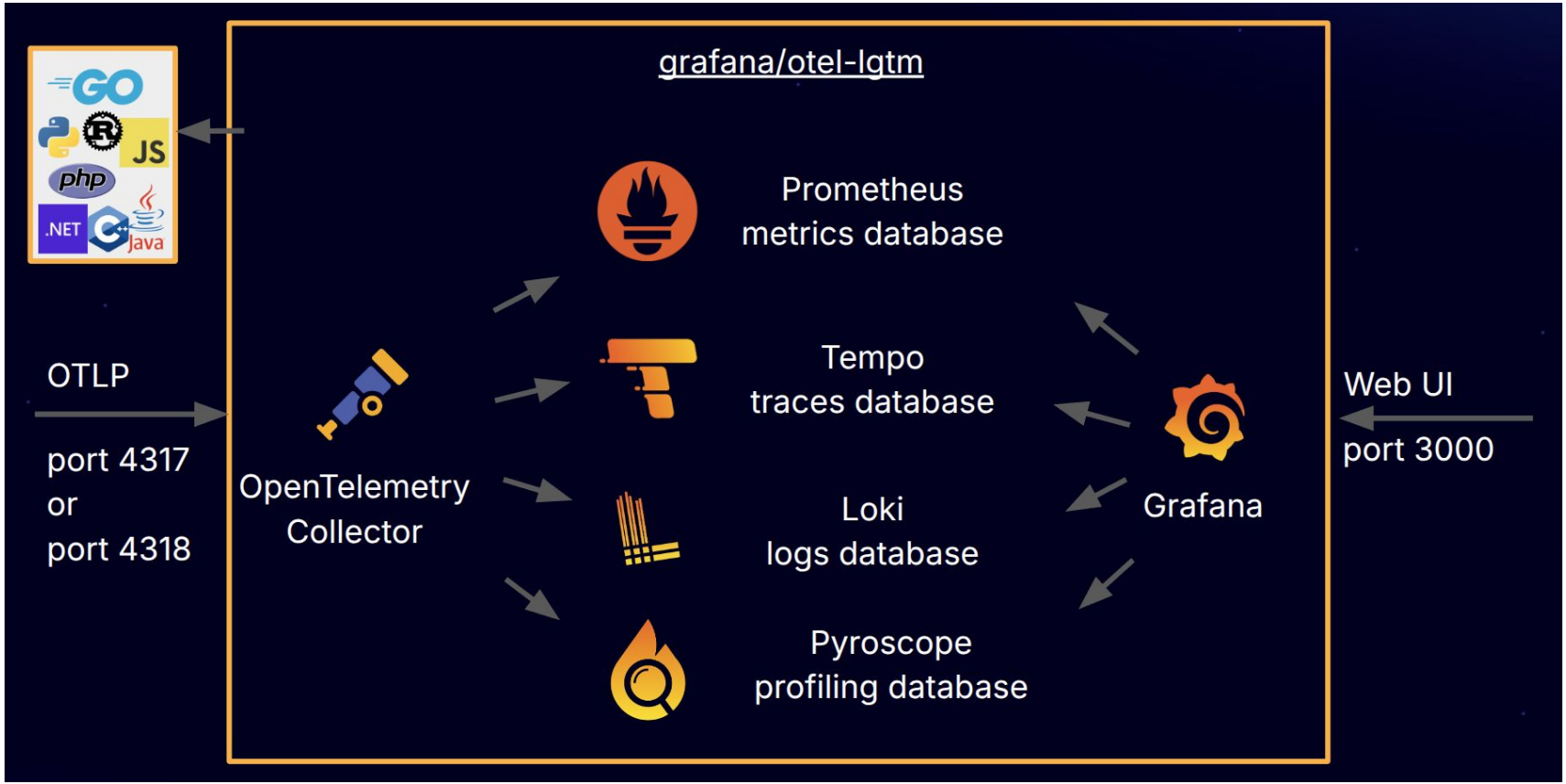
By default, starter-based applications use Logback, with built-in routing so logs from other frameworks are unified

Loggers are pre-configured to use console output with optional file output also available

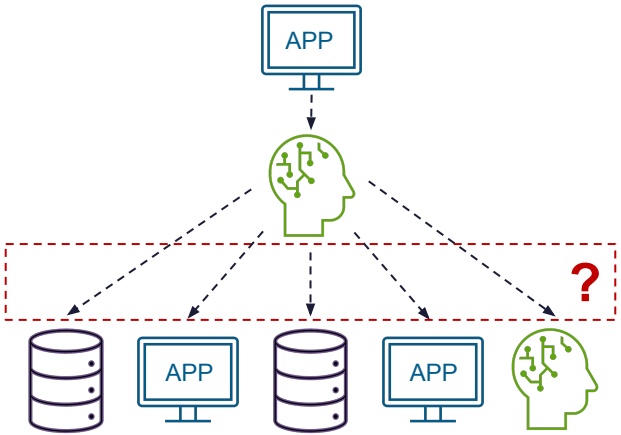
Spring Boot also auto-configures the OpenTelemetry SDK to support OTLP log export — but it does not install a Logback or Log4j2 appender, so add:

- `opentelemetry-logback-appender-1.0` ,´
- a custom logback configuration
- let the OpenTelemetryAppender know which OpenTelemetry API instance to use

The Grafana Stack



Model Context Protocol (MCP)



Agentic Applications

The Model Context Protocol provides a standardized way to connect AI models to different data sources and tools

A MCP Client is responsible for establishing and managing connections with MCP servers, which provide tools and resources

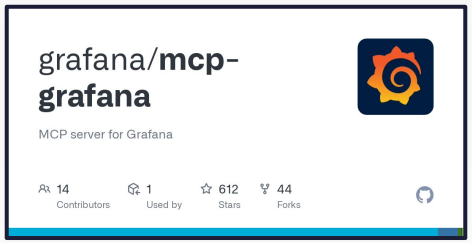
Grafana MCP

Search & update dashboards

Query data sources (Prometheus, Loki, Pyroscope)

Fetch alerts that are firing

Find out who's on call (IRM)



Create and manage incidents

Run and view Sift investigations

Interact with the Asserts graph

and much more to come

Follow us
@tiffanyfayj
@salmto

Thank You

github.com/timosalm/spring-boot-opentelemetry-lgtm

Grafana Labs & vmware by Broadcom