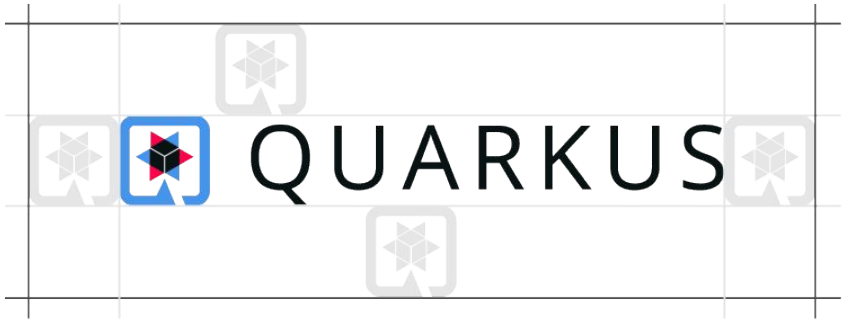
Observability on Quarkus

Bruno Baptista - @brunobat_
Aleš Justin - @alesj

Bruno Baptista

Principal Engineer at @RedHat.
Member of @MicroProfileIO,@coimbrajug.
@ApacheTomEE, @quarkusIO, @OpenTelemetry contributor.
Co-founder and organizer @jnationconf



Summary

- What is observability
- How do we implement it in Quarkus
- Latest developments
- Dive in
- Future vision

Previous experience?

- Microservices?
- Quarkus?
- Monitoring?

Observability

- Aims to provide an holistic view of a system's behavior.
- Directed to complex environments.
- Works best when there's collaboration between developers and devOps.

Observability

Measure of how well internal states of a system can be inferred from knowledge of its external outputs. The observability and controllability of a linear system are mathematical duals.

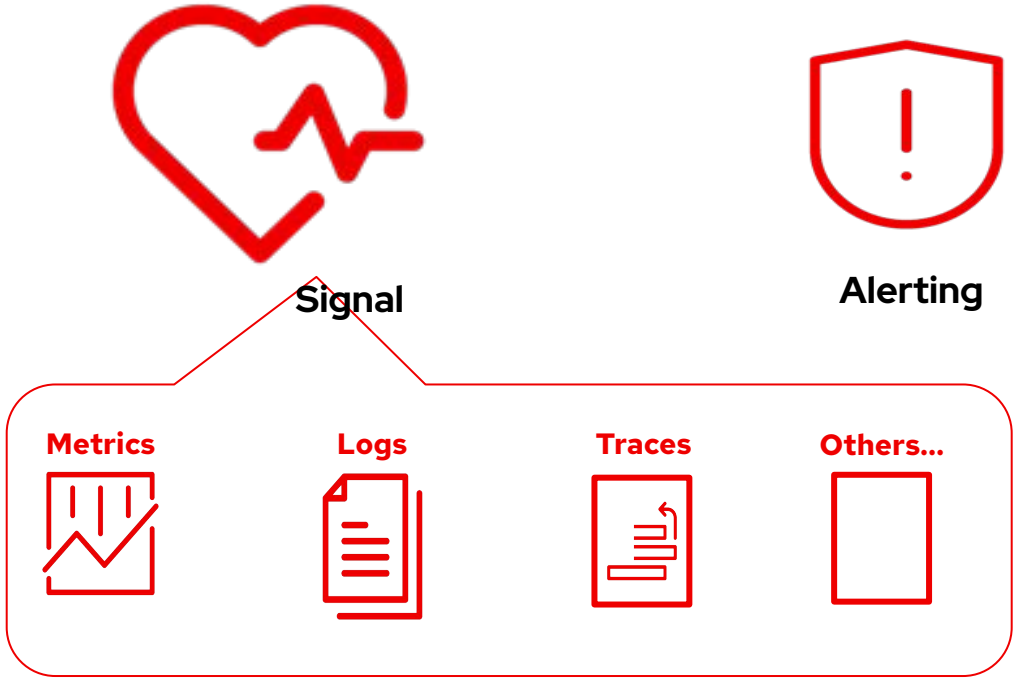
Observability

...for a software system is its “capability to allow a human to ask and answer questions”. According to Bryan Cantrill.

Traditional Monitoring vs. Observability

	Application Monitoring	**Observability**
Focus	Predefined metrics to maximize health and availability	Complete understanding of the system's internal state.
Analysis and Troubleshooting	Reactive.	Proactive.
Complexity Handling	Limited visibility into interconnected components	Handles complexity and distributed systems. Data can be correlated with traces

Observability Concepts



Observability in Quarkus

- Signal output convergence with OTel's OTLP protocol.
- Logs
  - Quarkus uses JBoss Log Manager + JBoss Logging facade.
  - Adapters for commons-logging, Log4j, Log4j2, Slf4j, etc...
  - In the future you'll be able to send to OTel Logging.
- Metrics
  - **Micrometer**. Default output in prometheus format, OTLP registry available.
- Tracing
  - **OpenTelemetry Tracing**

What is OpenTelemetry?

- OpenTelemetry simplifies the understanding and troubleshooting distributed systems.
- Provides:
  - Specification
  - Standardized APIs, libraries, and instrumentation to enable developers to easily instrument their applications
  - Programing language independence
  - Standard data transport and collection

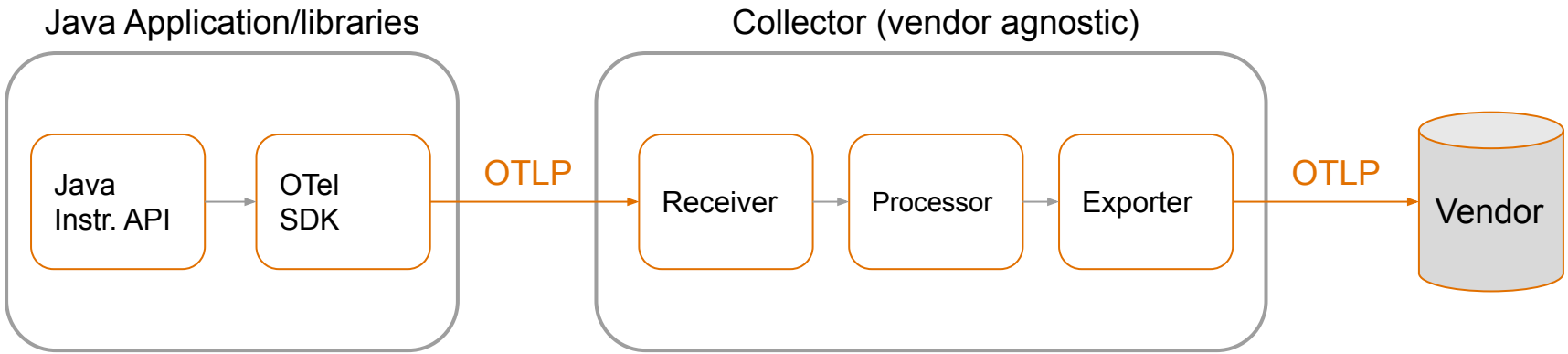
OpenTelemetry is NOT about

- Data Ingestion
- An Observability backend like Jaeger
- Storage

OpenTelemetry Components

- **opentelemetry-specification**
  - API (baggage, traces, metrics, logs, etc...)
  - SDK Specification
  - Semantic conventions
  - Data: The OpenTelemetry protocol (OTLP)
- Java Implementation projects
  - **opentelemetry-java**: Java API, SDK implementation in Java, extensions and the OpenTracing shim.
  - **opentelemetry-java-instrumentation**: The instrumentation API, the Java Agent and the instrumentation libraries.
  - Others like Contrib and Docs
- **Opentelemetry-collector**: The reference (and only) implementation.

OpenTelemetry Components



OpenTelemetry Tracing in Quarkus

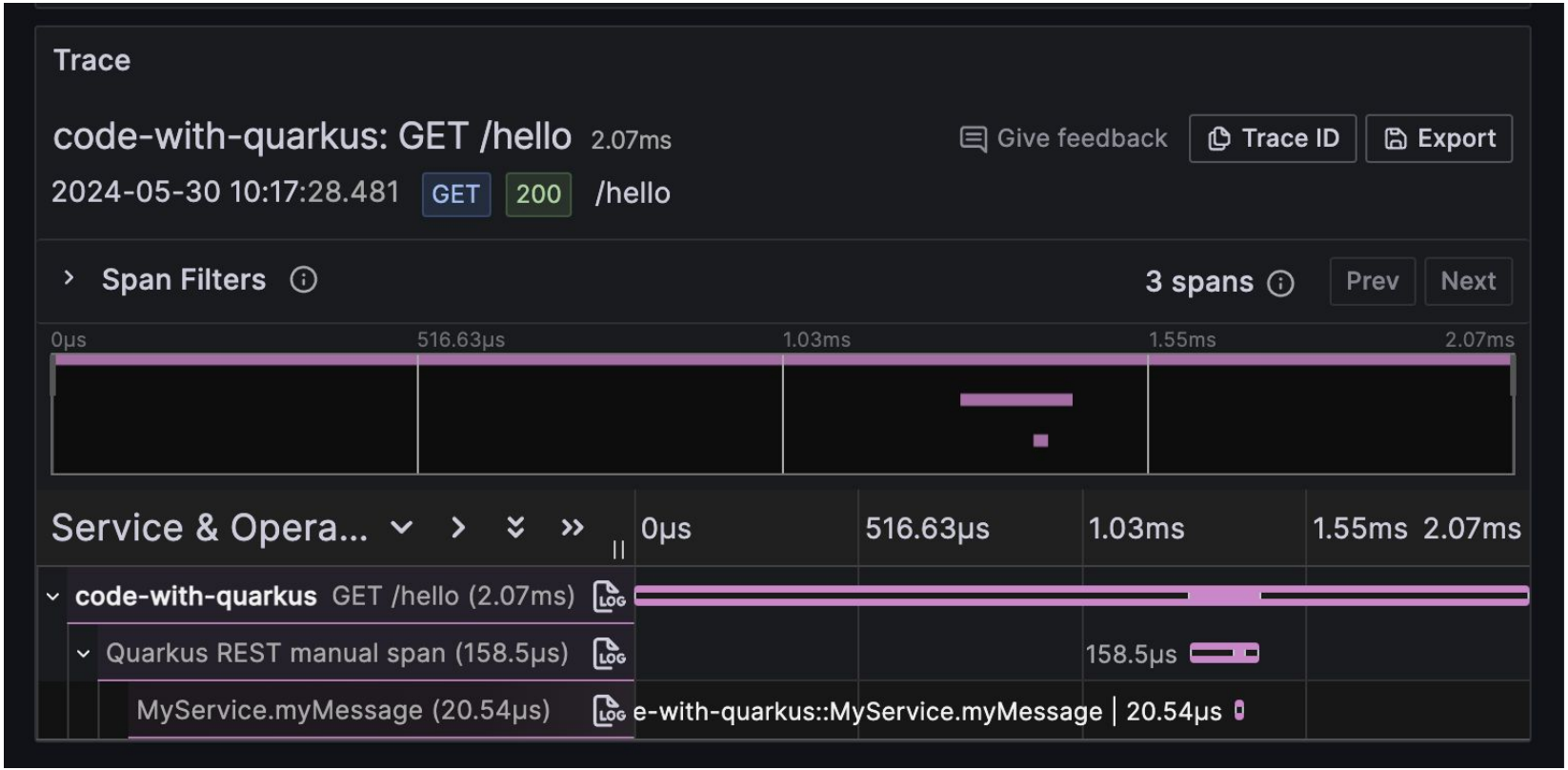
pom.xml

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-opentelemetry</artifactId>
</dependency>
```

application.properties

```
quarkus.otel.tracer.exporter.otlp.endpoint=http://localhost:4317
```

Tracing example



Micrometer through OTLP in Quarkus

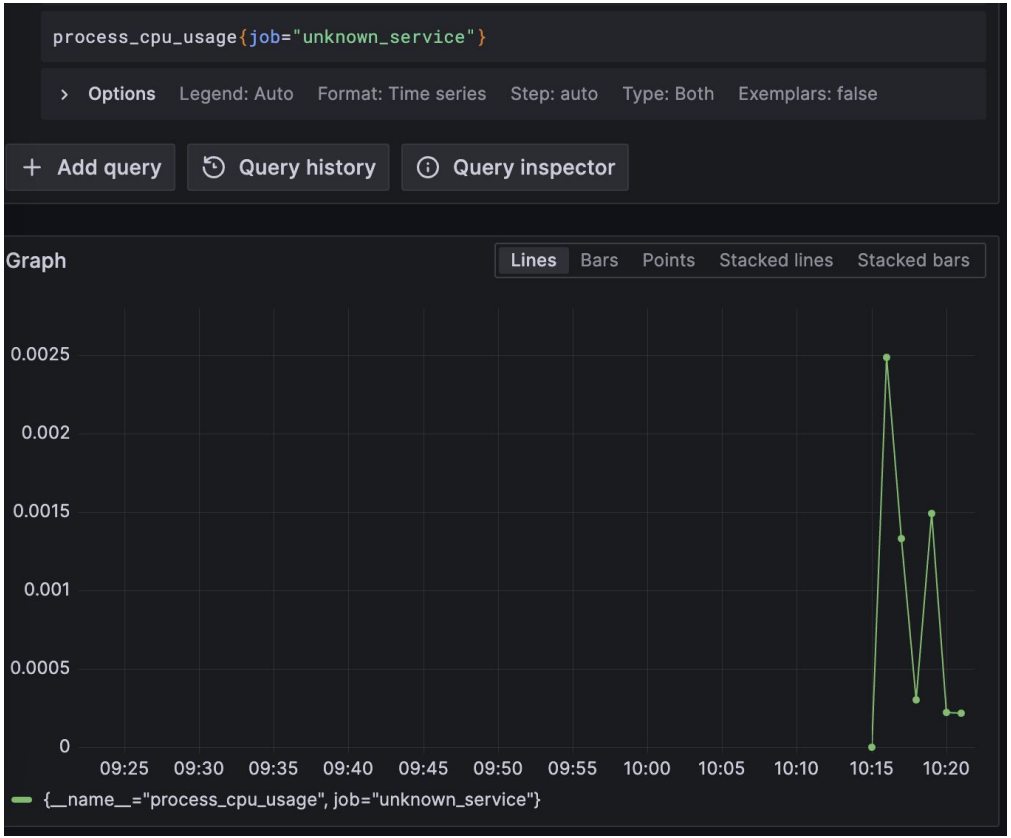
pom.xml

```
<dependency>
    <groupId>io.quarkiverse.micrometer.registry</groupId>
    <artifactId>quarkus-micrometer-registry-otlp</artifactId>
    <version>${quarkus-micrometer-registry-otlp.version}</version>
</dependency>
```

application.properties

```
quarkus.micrometer.export.otlp.url=http://localhost:4318/v1/metrics
```

Metrics example



JSon logs in Quarkus

pom.xml

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-logging-json</artifactId>
</dependency>
```

application.properties

```
quarkus.log.console.format=%d{HH:mm:ss} %-5p traceId=%X{traceId}, parentId=%X{parentId}, spanId=%X{spanId}, sampled=%X{sampled} [%c{2.}] (%t) %s%e%n

%dev.quarkus.log.console.json=false

%test.quarkus.log.console.json=false
```

OpenTelemetry extension news!

- Get spans from the database:

```
quarkus.datasource.jdbc.telemetry=true
```

- Vert.x based exporter.
- Use the exporter with HTTP:

```
quarkus.otel.exporter.otlp.traces.protocol=http/protobuf
quarkus.otel.exporter.otlp.endpoint=http://localhost:4318
```

- Proxy support
- Keeping up with OpenTelemetry updates. We'll be in OpenTelemetry v1.32.x for a while…

OpenTelemetry extension news!

- HTTP Semantic Conventions migration. Most attribute names will change:

  `Quarkus.otel.semconv-stability.opt-in`
  `http` -> new conventions
  `http/dup` -> duplicated old and new conventions

- Disable particular automatic tracing instrumentations using properties:
  - `quarkus.otel.instrument.grpc`
  - `quarkus.otel.instrument.reactive-messaging`
  - `quarkus.otel.instrument.rest-client-classic`
  - `quarkus.otel.instrument.resteasy-reactive`
  - `quarkus.otel.instrument.resteasy-classic`
  - `quarkus.otel.instrument.vertx-http`
  - `quarkus.otel.instrument.vertx-event-bus`
  - `quarkus.otel.instrument.vertx-sql-client`
  - `quarkus.otel.instrument.vertx-redis-client`

  The properties are booleans and are all `true` by default.

- New implementation for user identity in the spans

Micrometer extension news!

- Netty allocator metrics ON by default. Memory usage will be reported for:
  - Direct
  - Heap

- Custom tags with HTTP server data using CDI.

- MeterRegistryCustomizer CDI beans to change the config of the Registry.

Dev Services surprises…

Let's instrument a project!

The Future…

Future work

- OpenTelemetry Metrics
- OpenTelemetry Logs
- Micrometer to OpenTelemetry bridge
- Observation API to OpenTelemetry bridge plans
- More dev services
- Dashboards

Thanks!

- https://quarkus.io/guides/opentelemetry
- https://github.com/quarkiverse/quarkus-opentelemetry-exporter
- https://quarkus.io/guides/telemetry-micrometer
- https://github.com/quarkiverse/quarkus-micrometer-registry

Bruno Baptista - @brunobat_

Aleš Justin - @alesj