Make your Java application observable with no code changes using OpenTelemetry

ECLIPSE CON2022

Speakers

Gianmaria Romanato

www.linkedin.com/in/gromanato

@g1amma

Renato Bertacco

www.linkedin.com/in/rbertacco

@BertaccoRenato

Monitoring VS Observability

Monitoring consists in using tools/techniques that highlight that an issue occurred. A monitoring system could raise a warning when:

- average response time is getting slower and slower;
- a growing number of requests result in HTTP 500 – internal server error;
- application crashes;

Observability is the ability to measure the internal states of a system by examining its outputs (Control theory definition).

An application is “observable” when it provides detailed visibility into its behavior and always allows identifying the root cause of an issue.

Observability

Observability is based on three pillars:

- LOGS – application logs
- TRACES – track what happens within a request
- METRICS – specific counts or measures over a period of time (e. g. number of invocations per seconds)

Useful in production for trouble-shooting

Useful during perf tests to identify bottlenecks and areas of improvement

OpenTelemetry

OpenTelemetry

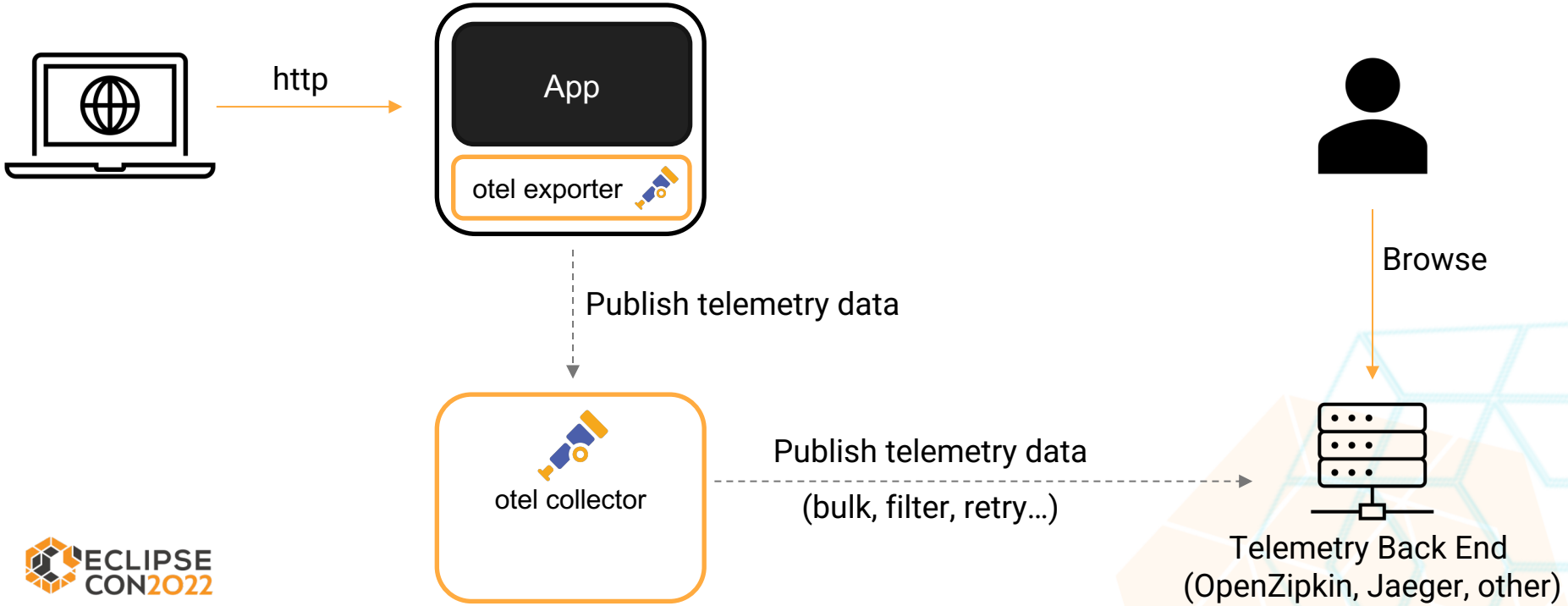
Open-source CNCF project hosted at github.

OpenTelemetry (otel) allows enabling *traces*, *trace propagation* and *metrics* in an existing Java app without requiring code changes.

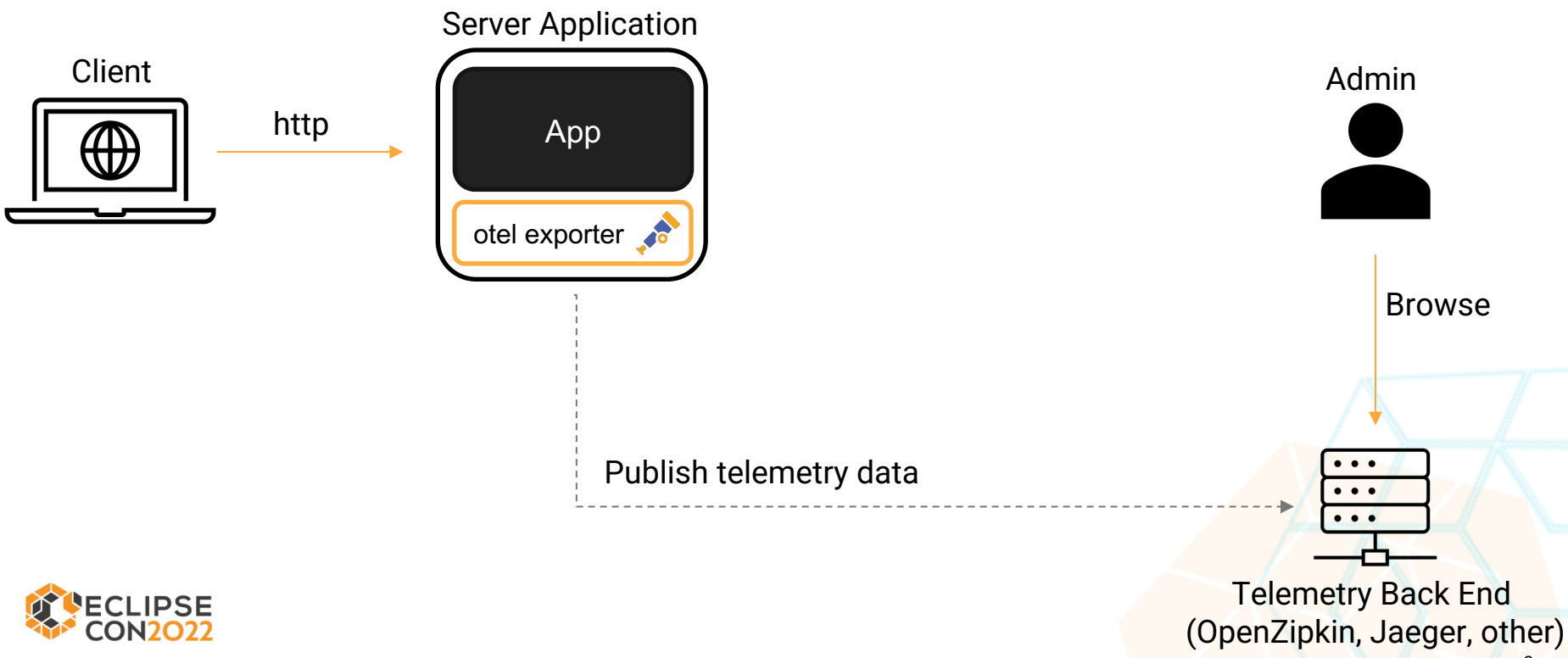
OpenTelemetry is built around smart idea:

- Every Java project uses a lot of standard (JDBC, Servlets…) and open-source components (Apache Tomcat, Hibernate, OkHttp, Spring…)
- What if we could manipulate bytecode of such libraries (instrumentation) is to inject telemetry behavior?

Traces – Exporter and Collector



Traces – Exporter w/o Collector



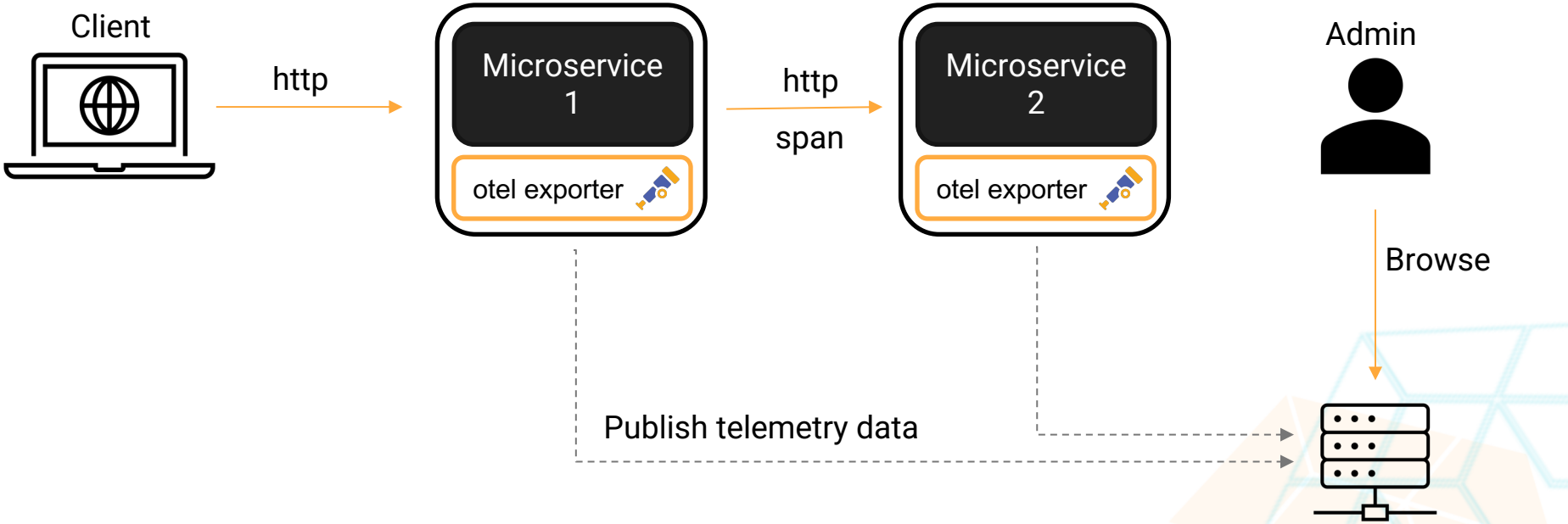
Enable tracing with otel - w/o Collector

Start your Java application specifying the OpenTelemetry JVM Agent on the command line.

```
java -javaagent:/path/to/opentelemetry-javaagent.jar
     -Dotel.traces.exporter=zipkin
     -Dotel.exporter.zipkin.endpoint=http://zipkin_hostname:9411/api/v2/spans
     <...your application parameters..>
```

Parameter	Comment
-javaagent	Activate the OpenTelemetry instrumentation agent
-Dotel.traces.export	Specify the format used for exporting traces (in this example zipkin but other formats are supported)
-Dotel.exporter.zipking.endpoint	URL endpoint to the OpenZipkin instance

Trace propagation - spans



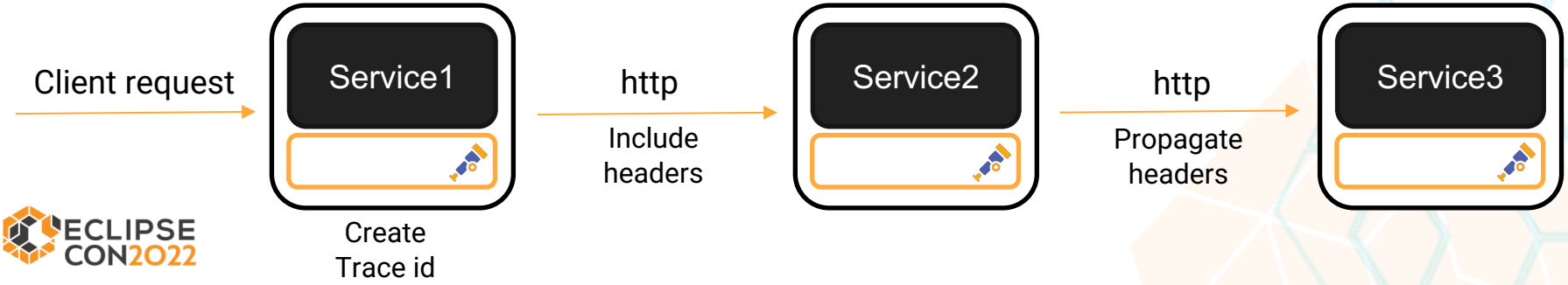
span: one operation within a trace

Trace propagation

Relies on http headers. Several standards on the market (zipkin, jaeger, w3c)

OpenTelemetry instrumentation changes your App(s) to:

- Create trace ids when serving a request;
- Include trace ids in headers of outbound calls;
- Receive and propagate headers present in inbound calls;
- Export trace ids in telemetry



Enable trace propagation with otel

Start your Java application specifying the OpenTelemetry JVM Agent and the trace-propagators extension on the command line.

```
java -javaagent:/path/to/opentelemetry-javaagent.jar
     -Dotel.traces.exporter=zipkin
     -Dotel.exporter.zipkin.endpoint=http://zipkin_hostname:9411/api/v2/spans
     -Dotel.javaagent.extensions=/path/opentelemetry-extension-trace-propagators.jar
     -Dotel.propagation.enabled=true
     -Dotel.propagators=b3multi
     <...your application parameters..>
```

Parameter	Comment
-Dotel.javaagent.extensions	Point to the OpenTelemetry trace-propagation extensions
-Dotel.propagation.enabled	Activate trace propagation
-Dotel.propagators	Specify the header format

Sample Application

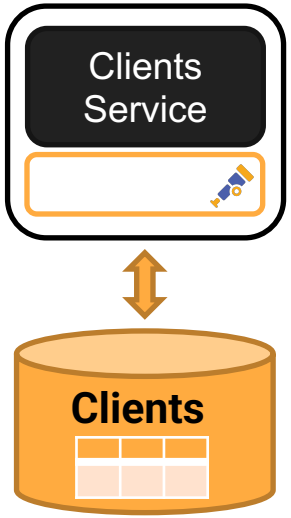
https://github.com/renatobertacco/oteldemo

Rent A Bike Application

A sample application made of 3 micro-services:

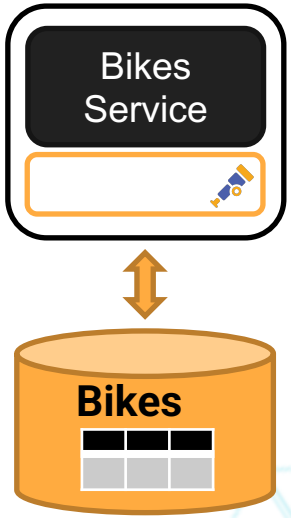
- Bike service – provides CRUD functionality for the bike entity, a table used to list bikes available at the shop
- Client service – provides CRUD functionality for the client entity, a table containing client identification information
- Rental service – provides API to start/end a bike rental and list current rentals.

Microservices



Create
Read
Update
Delete

StartRental
EndRental
ListRentals



Create
Read
Update
Delete

Datamodel

Clients

Id	Name	Id-card	Email	Phone
1	Mario Rossi	123456789	mrossi@example.org	+39-349-123-4567
2	John Doe	345678901	j.doe@example.org	+34-123-345-6781

- Separate databases
- *Clients.id* is copied into *Bikes.Rented_by* when a bike is rented, but there is no relation between tables

Bikes

Id	Model	Size	Rate	Rented_by
1	Canyon Spectral	L	40	1
2	Canyon Strive	M	40	
3	Propain Hugene	L	40	2

Rental microservice

startRental(bikeId, clientId) / stopRental (bikeId) - Flags a bike as rented/not rented by setting/clearing the client *id* into the *rented_by* column

Id	Model	Size	Rate	Rented_by
1	Canyon Spectral	L	40	1

ListRentals() - Gets list of rented bikes (*rented_by* not empty), and for each client *id* gets client details from Client Service, then returns combined results.

Id	Model	Size	Rate	Rented_by
1	Canyon Spectral	L	40	1

Id	Name	Id-card	Email
1	Mario Rossi	123456789	mrossi@example.org


```
{ “id” : ”1”,
  “Model” : “Canyon Spectral”,
  “Client” : “Mario Rossi”,
  “Phone” : “+39….” }
```

Demo Time!

Evaluate the Sessions

- Please help by leaving feedback on the sessions you attend!
- To rate a session, you must be registered for it in Swapcard BEFORE the talk starts.
- Swapcard will prompt you to leave feedback after the end of each session.
- You may also rate a talk by locating the session from the “Agenda” or “My Event” buttons on the Event Home page. Click on the session and look for the “Give your feedback” box.

ECLIPSE CON2022

Thank you!

Join the conversation:

@EclipseCon | #EclipseCon