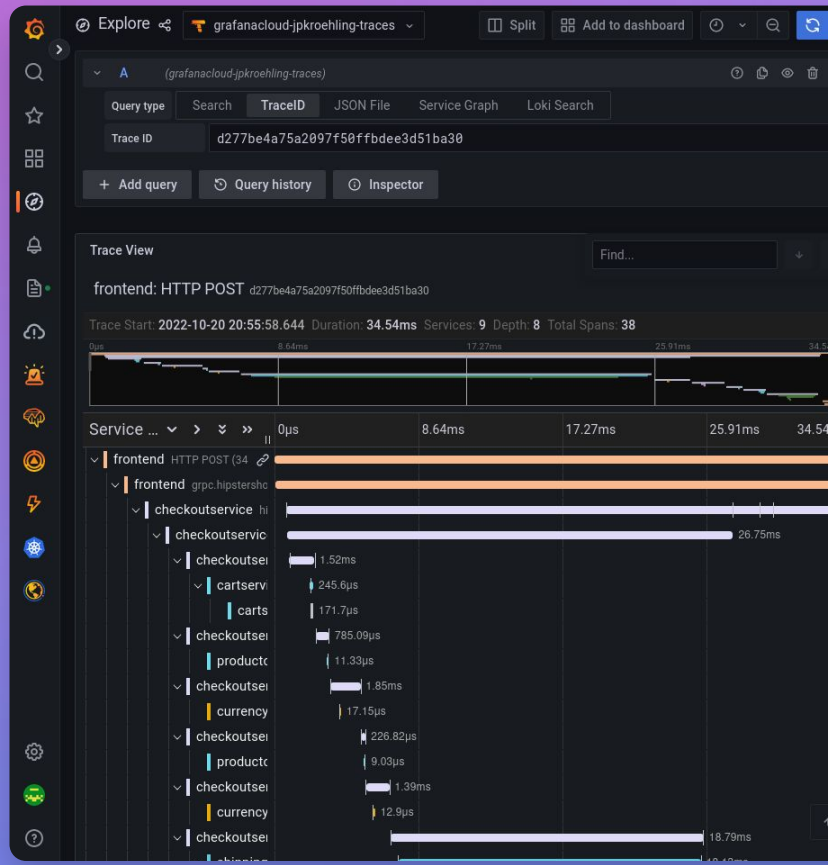
Grafana Labs

Unlocking Observability: Your Gateway to OpenTelemetry

Juraci Paixão Kröhling
Software engineer
@jpkrohling



Presenter

Juraci Paixão Kröhling

Software engineer

Agenda

Observability

OpenTelemetry

Demo

Questions and answers

DevOps Culture

DevOps Culture

Agile development

Continuous integration and delivery

Fast feedback loops

Change, evolution is constant

Reliability

Reliability Engineering

Reliability engineering

Resiliency

Automation

Error budgets, SLOs, ...

Launch at scale

Observability

Observability

O11y

Observability is the ability to find answers to questions we don’t have yet.

Monitoring is the practice of obtaining quick answers to frequent questions.

Telemetry

Telemetry data types

Logs

Metrics

Traces

Profiles

Events

OpenTelemetry

OpenTelemetry is a collection of tools, APIs, and SDKs. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software’s performance and behavior.

Source: https://opentelemetry.io/

Standards, specifications, and conventions

Client instrumentation APIs and SDKs

Middleware

Standards and specifications

API Specification

- How the instrumentation API should look like, no matter the SDK

SDK Specification

- How SDKs have to behave across languages and implementations

Data Specification

- Interface description language (IDL), specifying how data should look like and what endpoints should implement

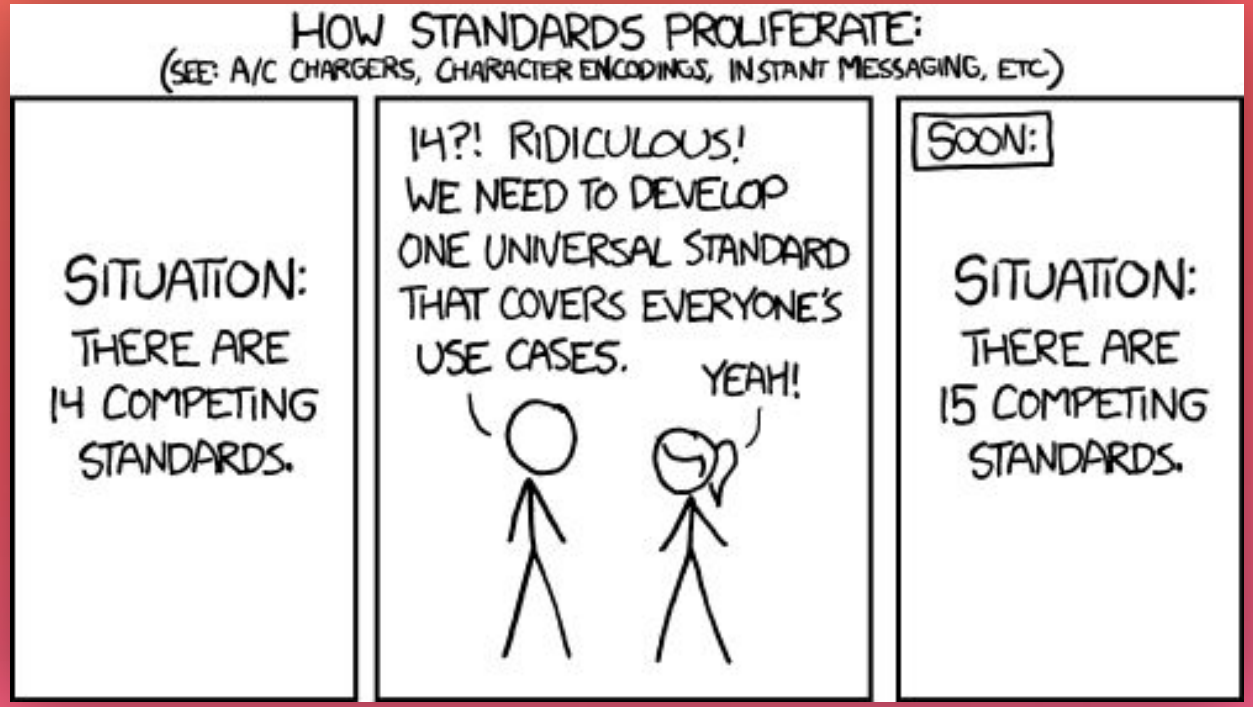
Data specification

OTLP

OpenTelemetry Line Protocol, a set of proto files defining both the transport and payload aspects of the telemetry data

XKCD #927

Obligatory reference to XKCD #927.


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Data specification

OTLP

OpenTelemetry Line Protocol, a set of proto files defining both the transport and payload aspects of the telemetry data

XKCD #927

Obligatory reference to XKCD #927.

Instrumentation

Manual instrumentation

- By using the instrumentation API directly

Instrumentation libraries

- By using libraries that hook into well-known frameworks and libraries, such as Spring, Servlets, JAX-RS, ...

Automatic instrumentation

- Typically by doing bytecode manipulation to add instrumentation at certain interesting points

Instrumentation demo

GreetingResource.java - otel-demo - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OTEL-DEMO
- .mvn
- src
  - main
    - docker
    - java / com / grafana
      - GreetingResource.j...
    - resources
      - META-INF
      - application.properties
  - test
- target
- dockerignore

OUTLINE

JAVA PROJECTS
- otel-demo
  - src/main/java
    - com.grafana
      - GreetingReso...
  - src/main/resources
    - META-INF
    - application.propert...
  - src/test/java
  - target/generated...
  - target/generated...
  - JRE System Libra...
  - Maven Depende...

MAVEN

application.properties | GreetingResource.java

src > main > java > com > grafana > GreetingResource.java > GreetingResource > tracer

```java
import io.opentelemetry.api.trace.Span;
import io.opentelemetry.api.trace.Tracer;

@Path("/hello")
public class GreetingResource {

    @Inject
    Tracer tracer;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        Span span = tracer.spanBuilder("my-operation").startSpan();
        // do something interesting...
        span.end();
        return "Hello RESTEasy";
    }
}
```

PROBLEMS 1 | OUTPUT | TERMINAL | DEBUG CONSOLE

```
    End time       : 2021-10-14 10:32:01.877598666 +0000 UTC
    Status code    : STATUS_CODE_UNSET
    Status message :
Attributes:
     -> http.client_ip: STRING(127.0.0.1)
     -> http.status_code: INT(200)
     -> http.scheme: STRING(http)
     -> http.host: STRING(localhost:8080)
     -> http.method: STRING(GET)
     -> http.route: STRING(/hello)
     -> http.flavor: STRING(1.1)
     -> http.user_agent: STRING(curl/7.76.1)
     -> http.target: STRING(/hello)
```

```
jpkroehling@caju ~/Projects/src/github.com/jpkrohling/otel-demo $ curl
 localhost:8080/hello
Hello RESTEasyjpkroehling@caju ~/Projects/src/github.com/jpkrohling/ot
el-demo $
```

Java Bu... | go opent... | bash

Ln 15, Col 12 | Spaces: 4 | UTF-8 | LF | Java | JavaSE-11

OpenTelemetry Collector

Vendor-agnostic way to receive, process and export telemetry data.

Source: https://opentelemetry.io/docs/collector/

OpenTelemetry Collector conceptual architecture

OpenTelemetry Collector - related projects

Contrib

- Where non-core components reside, such as vendor-specific ones

Operator

- Kubernetes operator managing OpenTelemetry Collector instances

OpenTelemetry Collector - related projects

OpAMP

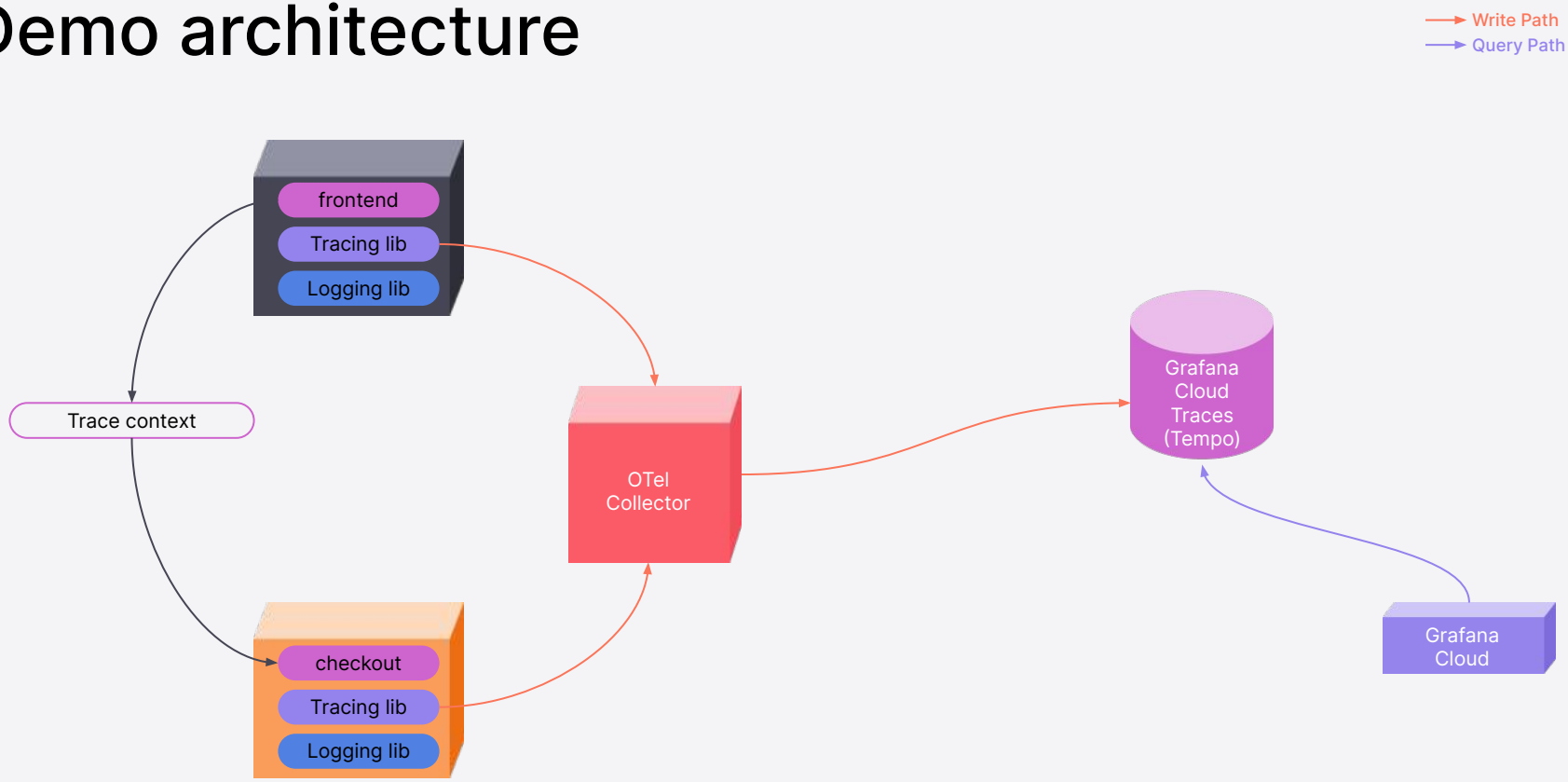
- Spec and implementation to manage collectors

Builder

- Helper CLI tool to build OpenTelemetry Collector distributions

Demo

Demo architecture



Key takeaways

Key takeaways

- OpenTelemetry has different sub projects in different areas
- OpenTelemetry APIs can be used to instrument the services
- Instrumentation libraries are power-ups, allowing you to quickly understand your application by understanding what your stack is doing
- Automatic instrumentation, such as the ones using Java Agents, are helpful to get started quickly
- The Collector can be used to make central processing of all your telemetry data and abstracting away the telemetry backends from your applications

Q&A

Grafana Labs

Thank you