
# Começando com OpenTelemetry

## Tudo que você precisa saber

jcosta.dev

# Juliano Costa

Developer Advocate

DATADOG

CLOUD NATIVE AMBASSADOR H1 2024

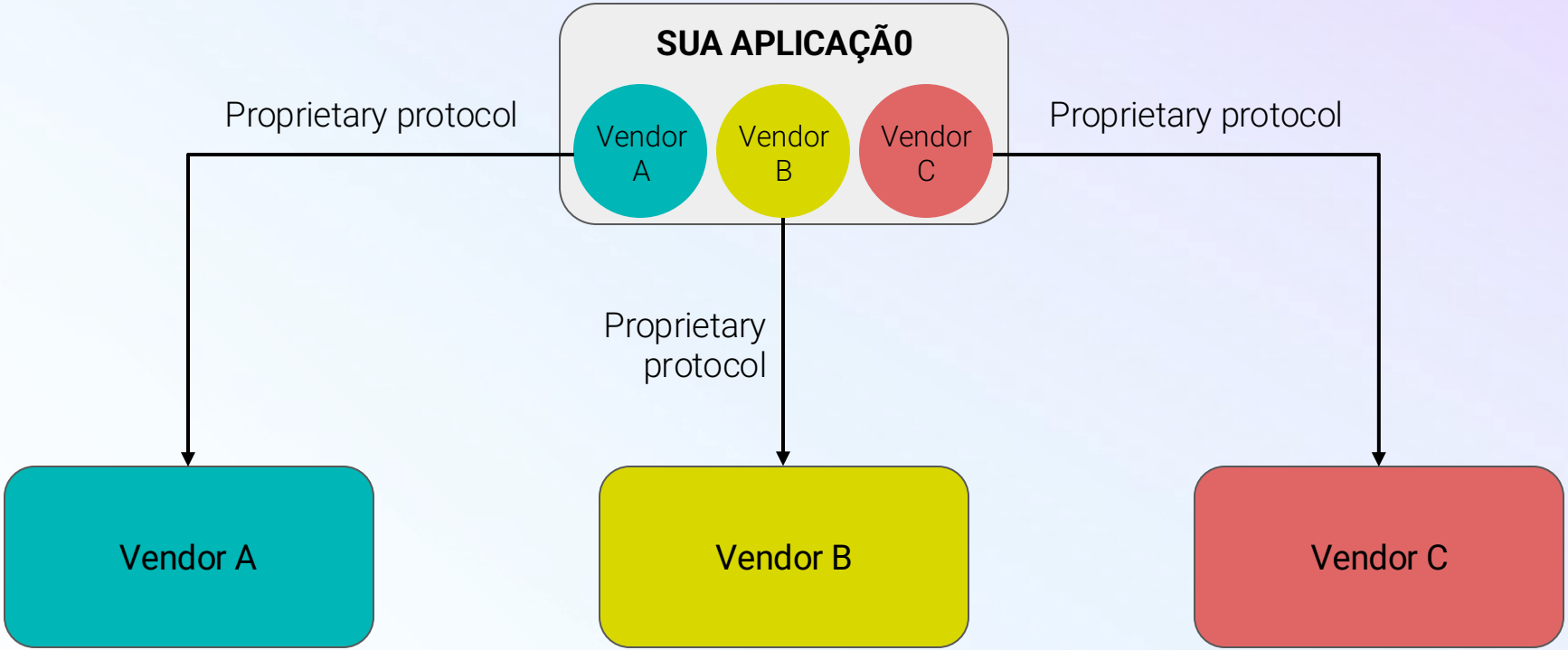
Cloud Native LINZ

OpenTelemetry CERTIFIED Associate CONTRIBUTOR

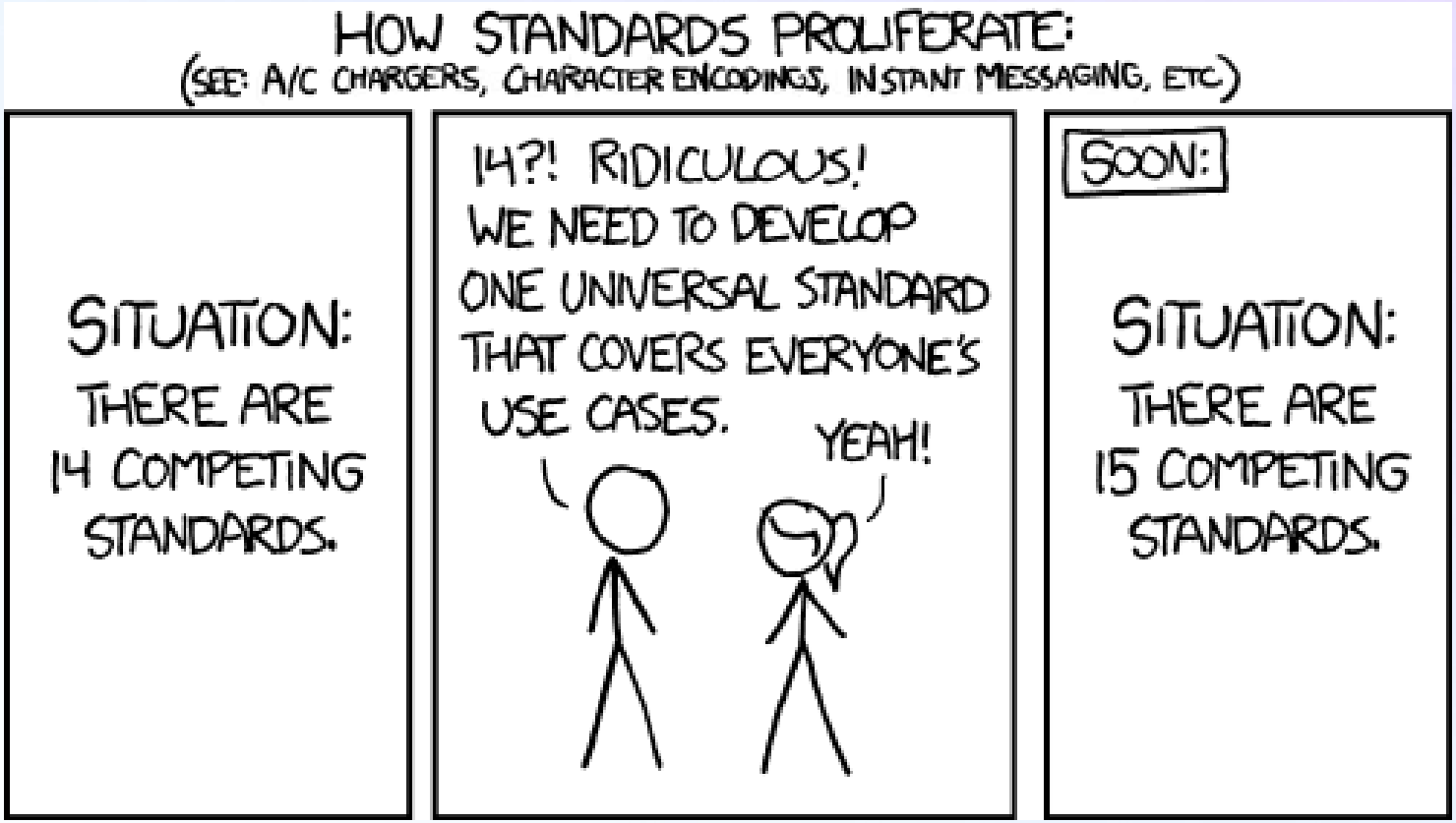
jcosta.dev

# O QUE É OPENTELEMETRY?

# VAMOS VOLTAR NO TEMPO UM POUCO

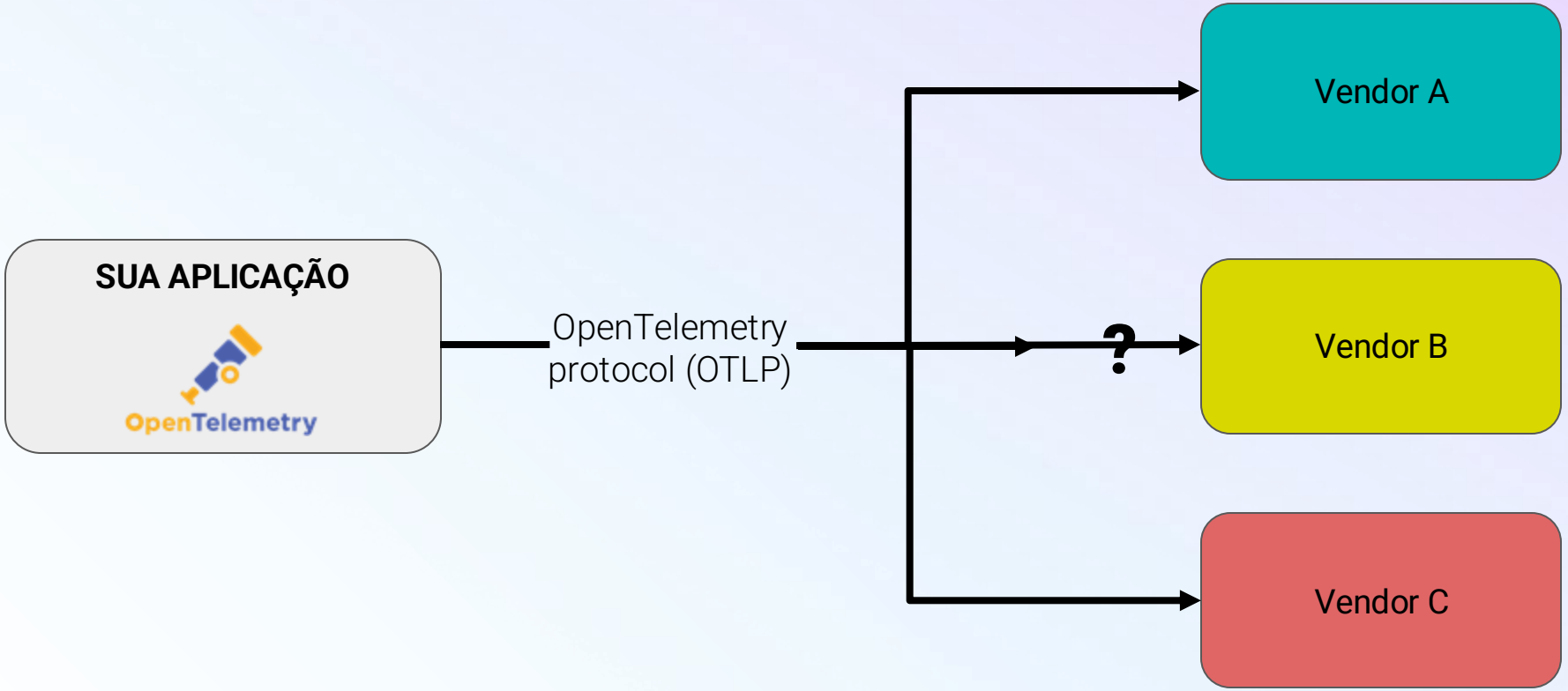

SUA APLICAÇÃ0
Vendor A
Vendor B
Vendor C
Proprietary protocol
Proprietary protocol
Proprietary protocol
Vendor A
Vendor B
Vendor C

# A GENTE PRECISA DE UM PADRÃO PROS DADOS DE TELEMETRIA




Source: https://xkcd.com/927/


DEPRECATED
OPENTRACING
+
DEPRECATED
OpenCensus
=
OpenTelemetry

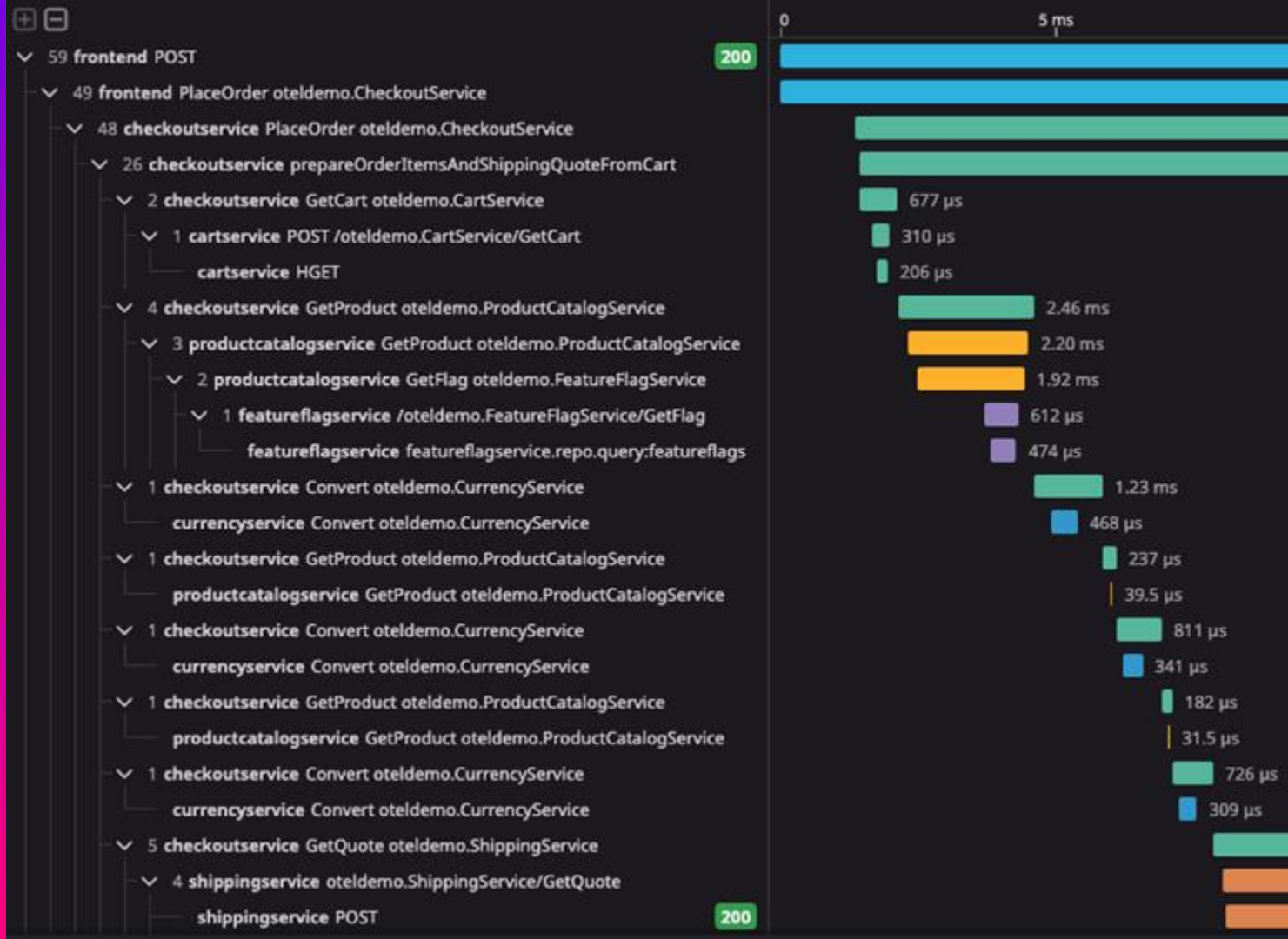

SUA APLICAÇÃO
OpenTelemetry
OpenTelemetry
protocol (OTLP)
?
Vendor A
Vendor B
Vendor C

# SINAIS


sinais

# Sinais

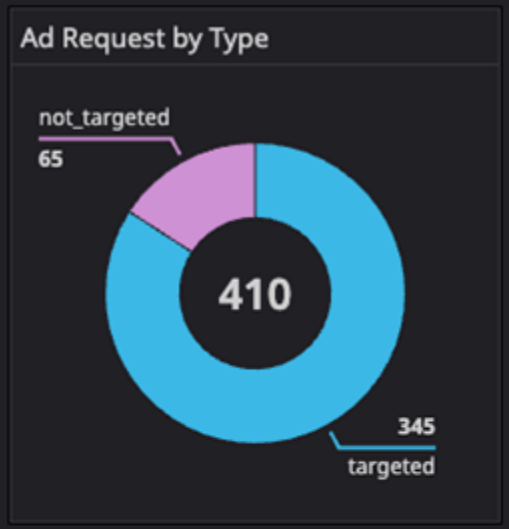
Traces

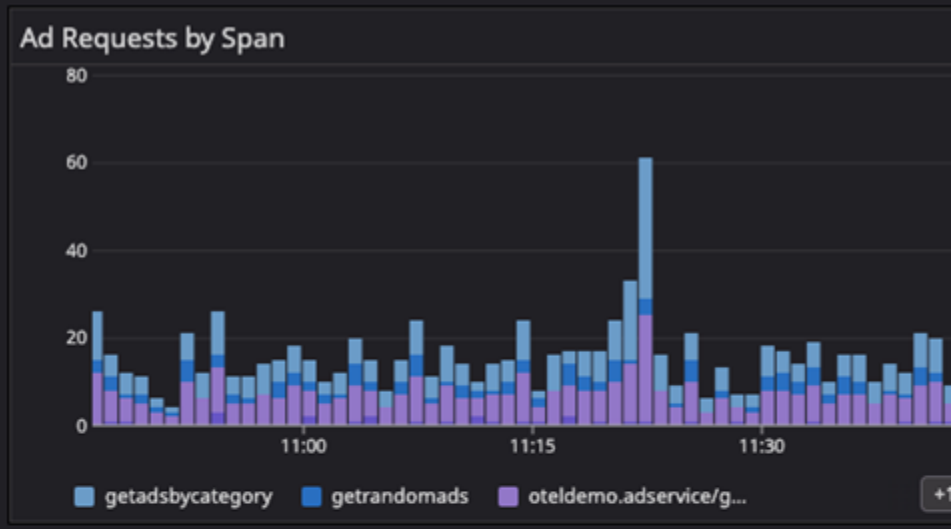


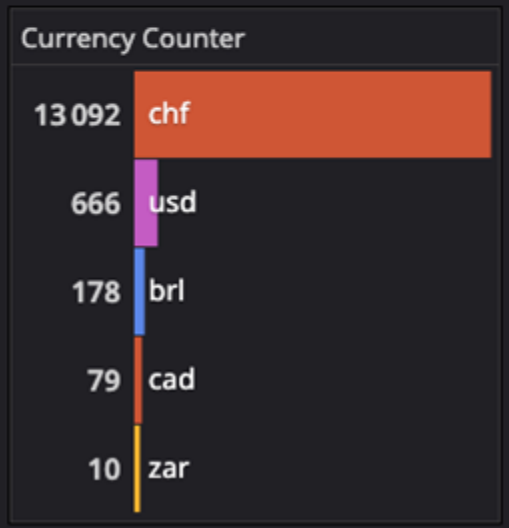
# Signals

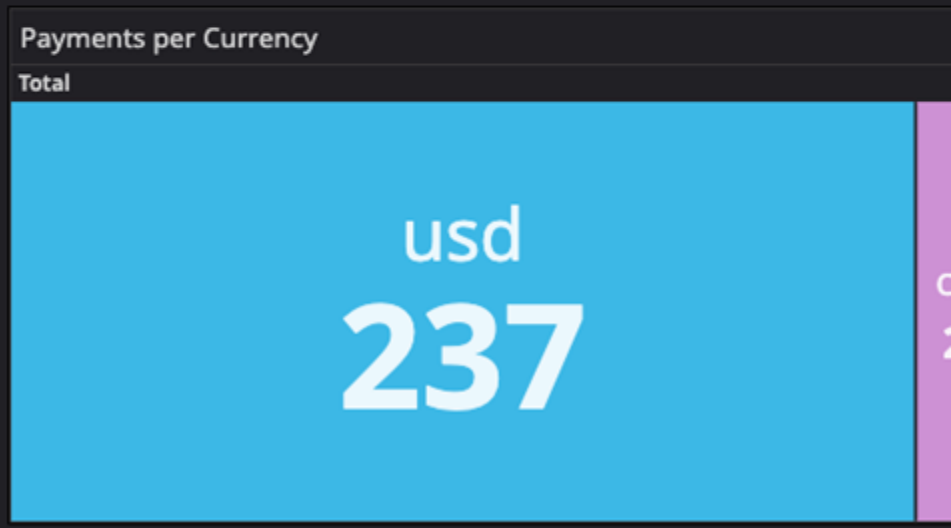
Traces

Métricas



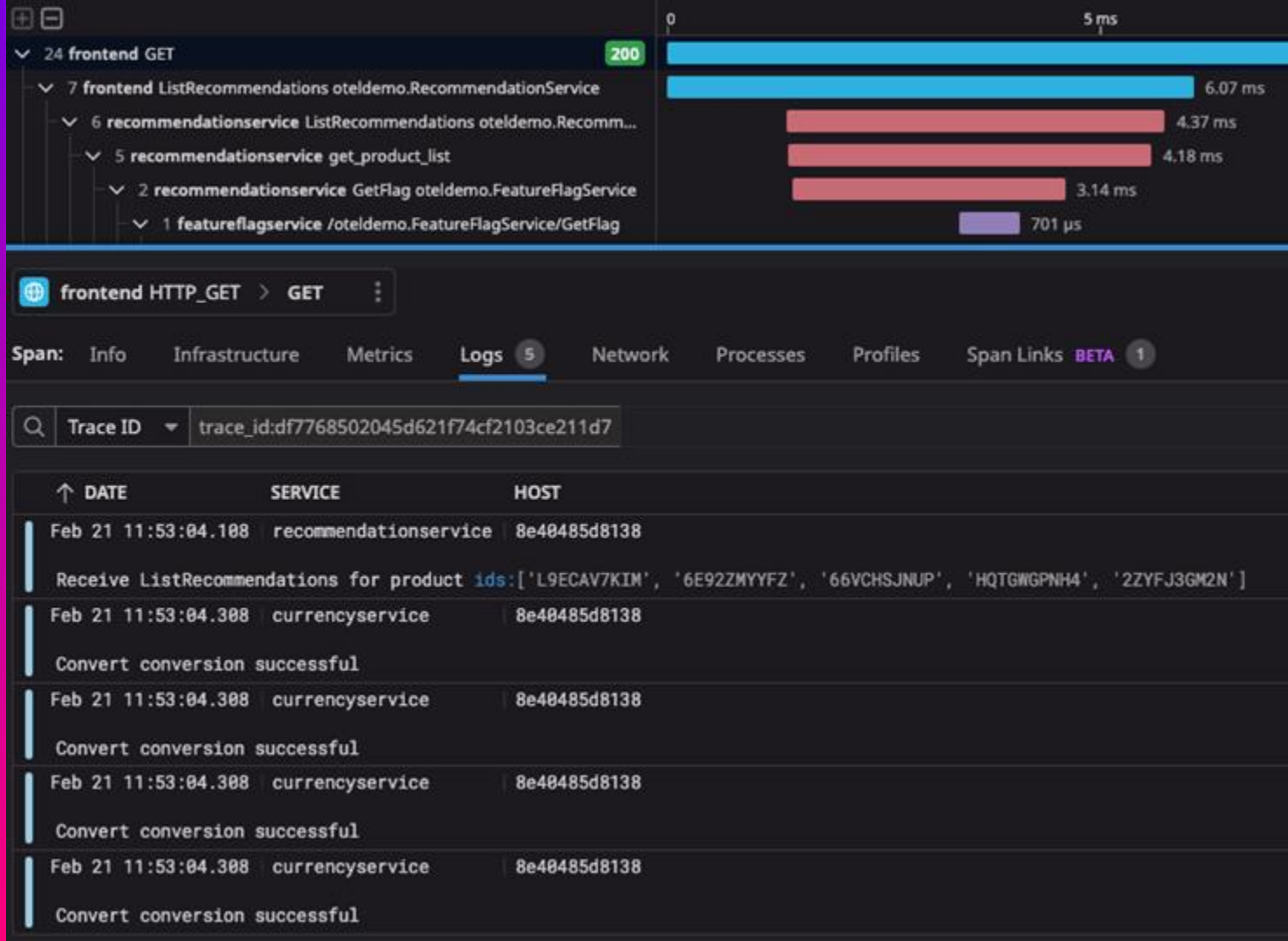



# Signals

Traces

Metrics

Logs



# Signals

Traces

Metrics

Logs

Profiling

## Introduces Profiling Data Model v2 #239

Open petethepig wants to merge 31 commits into `open-telemetry:main` from `petethepig:profiling-pprofextended`

Conversation 115 Commits 31 Checks 1 Files changed 2

**petethepig** commented on Nov 2, 2023 • edited Member

This is second version of the Profiling Data Model OTEP. After we've gotten feedback from the greater OTel community we went back to the drawing board and came up with a new version of the data model. The main difference between the two versions is that the new version is more similar to the original pprof format, which makes it easier to understand and implement. It also has better performance characteristics. We've also incorporated a lot of the feedback we've gotten on the first PR into this OTEP.

Some minor details about the data model are still being discussed and will be flushed out in the future OTEPs. We intend to finalize these details after doing experiments with early versions of working client + collector + backend implementations and getting feedback from the community. The goal of this OTEP is to provide a solid foundation for these experiments.

So far we've done a number of things to validate it:

- we've written a new profiles proto described in this OTEP
- we've documented decisions made along the way in a decision log
- we've done benchmarking to refine the data representation (see Benchmarking section in a collector PR)
- diff between original pprof and the new proto: link

We're seeking feedback and hoping to get this approved.

# INSTRUMENTAÇÃO

# INSTRUMENTAÇÃO AUTOMÁTICA

# Instrumentação Automática

Agente

Arquivo

Script

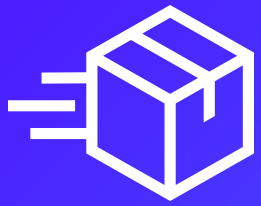
# Instrumentação Automática

Terminal

```
export JAVA_TOOL_OPTIONS="-javaagent:path/to/opentelemetry-javaagent.jar"
export OTEL_SERVICE_NAME="your-service-name"

java -jar myapp.jar
```

# BIBLIOTECAS DE INSTRUMENTAÇÃO

# Bibliotecas de Instrumentação



Criadores de bibliotecas ou contribuidores da comunidade podem criar bibliotecas/pacotes de instrumentação

O dono da aplicação precisa adicionar manualmente as dependências e fazer pequenos ajustes no código

# Bibliotecas de Instrumentação

Terminal

```rust
use actix_web::{get, web, App, HttpServer, Responder};

#[get("/hello/{name}")]
async fn greet(name: web::Path<String>) -> impl Responder {
    format!("Hello {name}!")
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .service(greet)
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```

# Bibliotecas de Instrumentação

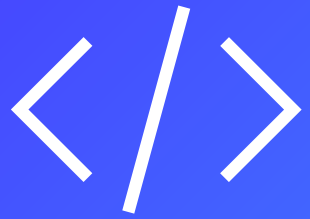
Terminal

```rust
use actix_web::{get, web, App, HttpServer, Responder};
use actix_web_opentelemetry::RequestTracing;

#[get("/hello/{name}")]
async fn greet(name: web::Path<String>) -> impl Responder {
    format!("Hello {name}!")
}

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    HttpServer::new(|| {
        App::new()
            .wrap(RequestTracing::new())
            .service(greet)
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```

# INSTRUMENTAÇÃO MANUAL

# Instrumentação Manual

Todo o esforço para instrumentar o código é de responsabilidade do time que é dono da aplicação



Cada span, métrica e log precisa ser adicionado no código da aplicação

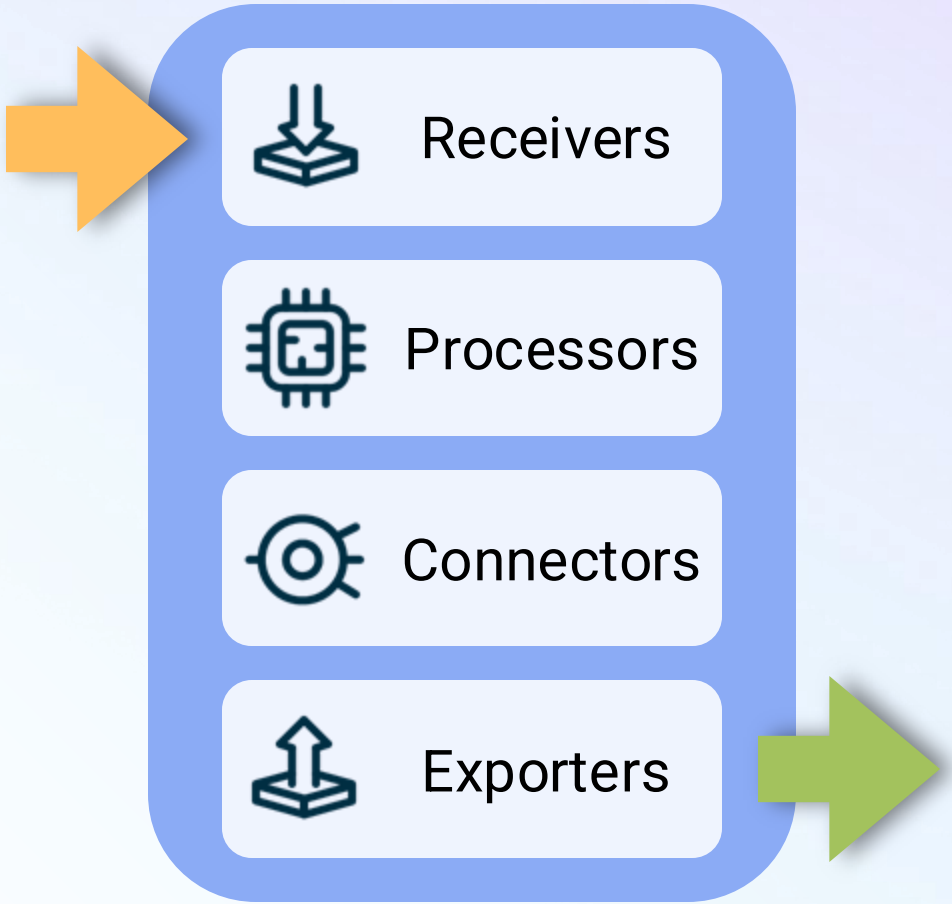
# Instrumentação Manual

Terminal

```
const trace = require('@opentelemetry/api');
const tracer = trace.getTracer('payment');

module.exports.charge = async request => {
  const span = tracer.startSpan('charge');

  // Do charging work

  span.setAttributes({
    'app.payment.card_type': cardType,
    'app.payment.card_valid': valid,
    'app.loyalty.level': loyalty_level
  });

  span.end();

  return { transactionId };
};
```

# OTEL COLLECTOR

# OTel Collector



# HANDS-ON

Photo by jesse orrico on Unsplash

# Juliano Costa

Developer Advocate at Datadog

/julianocosta89

@julianocosta89

@jcosta.dev

/julianocosta89

jcosta.dev