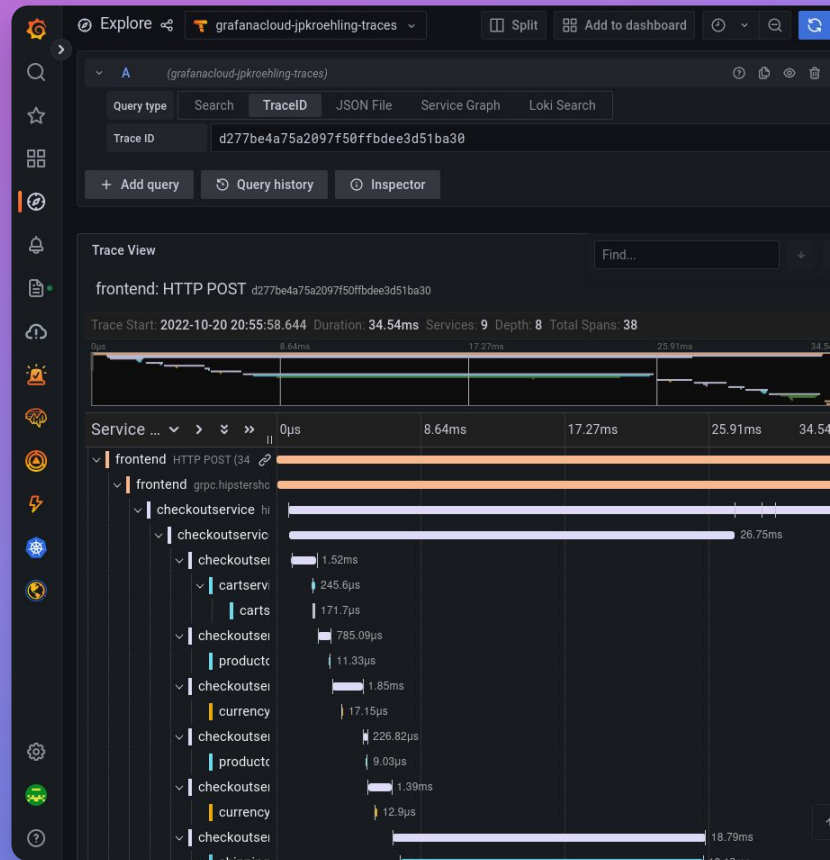


Dissecando Rastreamento Distribuído



Juraci Paixão Kröhling
Engenheiro de software
@jpkrohling



Quem sou eu



Juraci Paixão Kröhling
Engenheiro de software



Engenheiro de software na Grafana Labs

Membro do Comitê de Governança do projeto OpenTelemetry

Embaixador da Cloud Native Computing Foundation (CNCF)

Mantenedor de módulos do OpenTelemetry Collector

Ex-mantenedor do Jaeger

Ex-mantenedor do OpenTracing

Criador do Dose de Telemetria

Criador do Dose na Nuvem



@jpkrohling

<https://linktr.ee/jpkroehling>

Palestrante



Juraci Paixão Kröhling
Engenheiro de software



Agenda

Rastreamento distribuído

- O que é?
- Por quê precisamos?
- Como funciona?

Como chegamos até aqui

Demonstração

Perguntas e respostas

Rastreamento distribuído

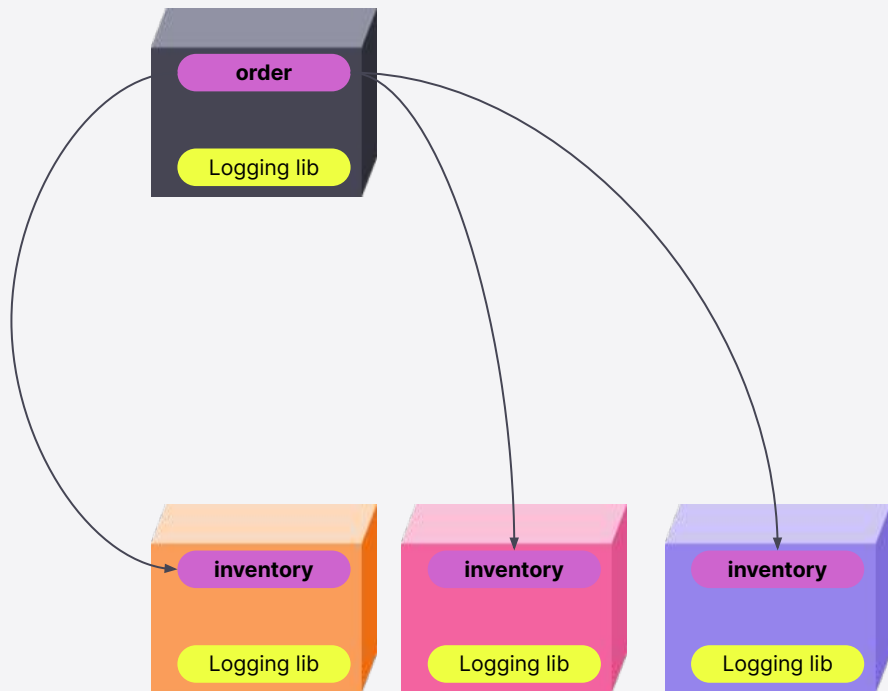




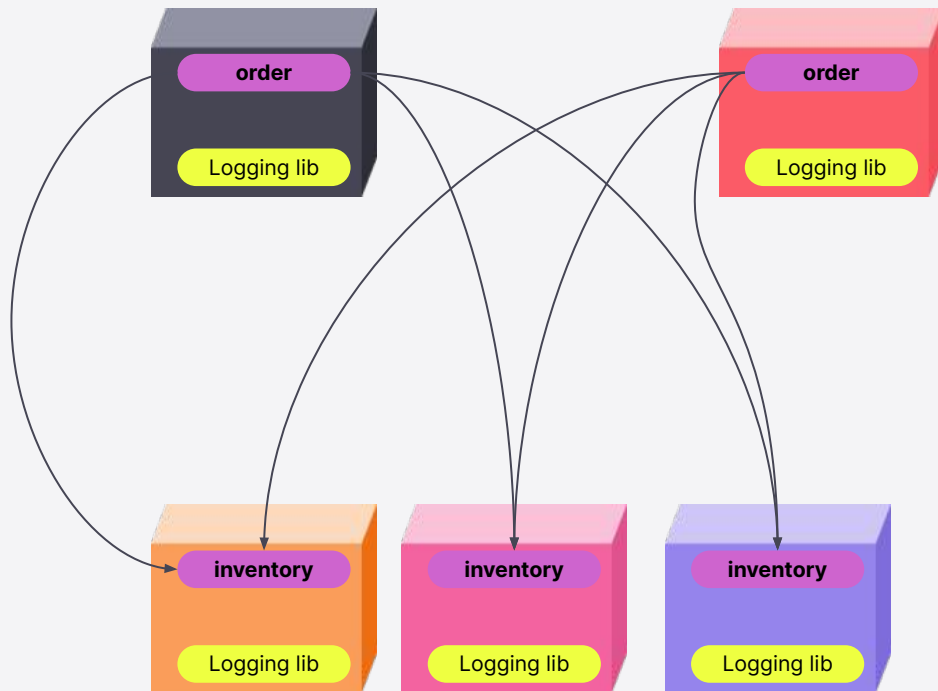
Rastreamento distribuído é a técnica que nos possibilita entender tudo o que aconteceu por trás de uma transação.



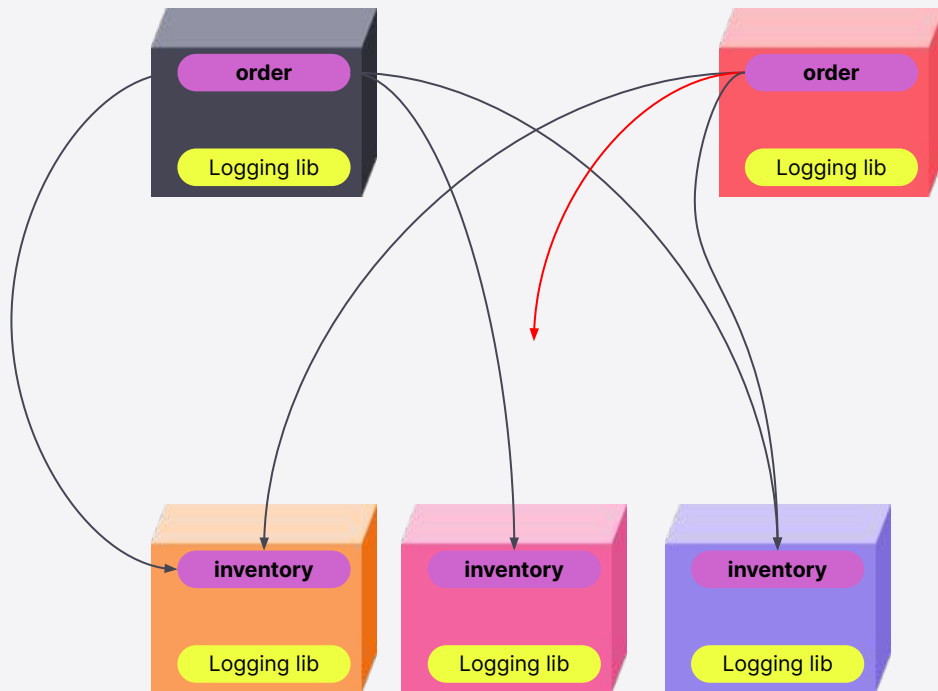
Microserviços clássicos



Microserviços clássicos

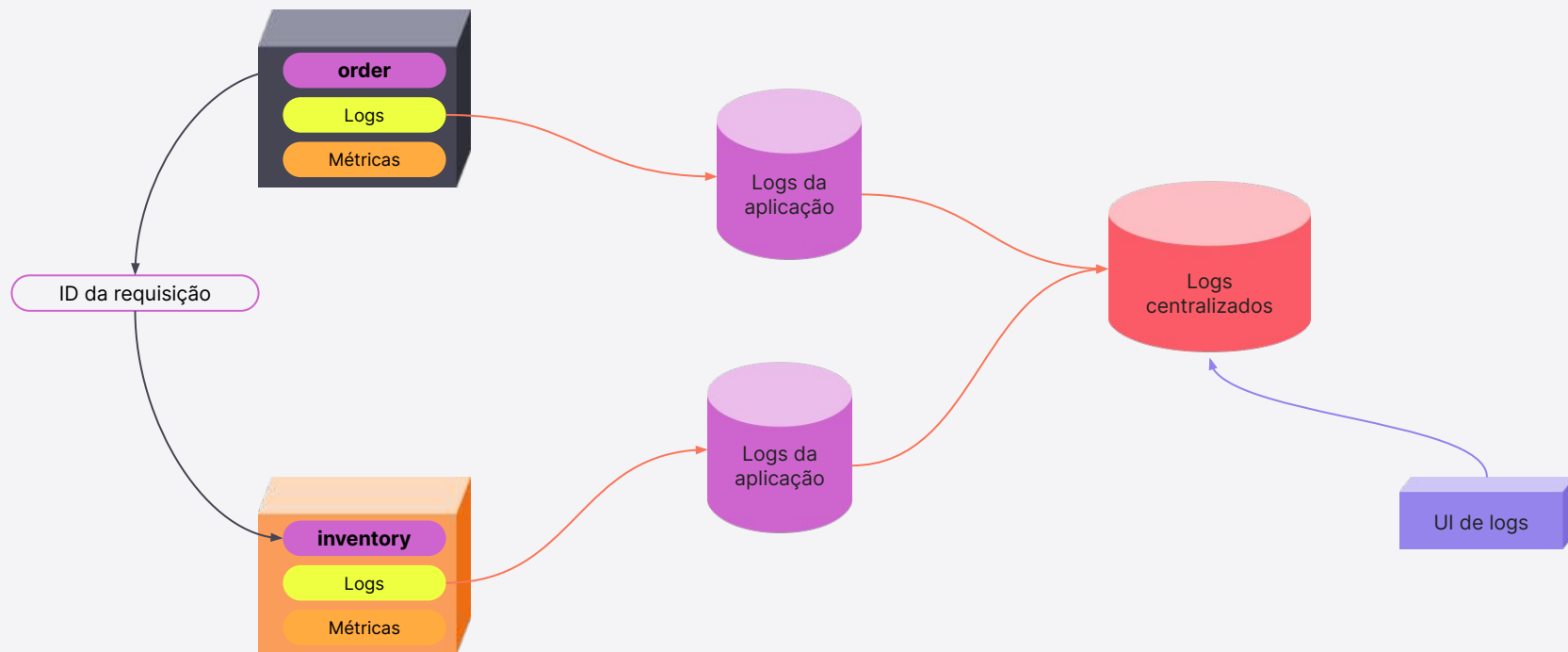


Microserviços clássicos



Microserviços com correlação de eventos

→ Escrita
→ Consulta



Logs



```
> kubectl logs my-otel-demo-accountingservice-5d79464f44-r5cwz
{"message":"Kafka brokers:
my-otel-demo-kafka:9092","severity":"info","timestamp":"2023-05-03T19:03:19.29250372Z"}
2023/05/03 19:05:37 Message c orderId =7c86783b-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:05:37.367 +0000 UTC, topic = orders
2023/05/03 19:05:39 OTLP partial success: empty message (0 spans rejected)
2023/05/03 19:05:44 Message claimed: orderId = 8133c708-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:05:44.611 +0000 UTC, topic = orders
2023/05/03 19:05:44 Message claimed: orderId = 816e0495-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:05:44.902 +0000 UTC, topic = orders
2023/05/03 19:05:49 OTLP partial success: empty message (0 spans rejected)
2023/05/03 19:05:55 Message claimed: orderId = 87ba0a89-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:05:55.65 +0000 UTC, topic = orders
2023/05/03 19:05:59 OTLP partial success: empty message (0 spans rejected)
2023/05/03 19:06:15 Message claimed: orderId = 93e3cd62-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:06:15.977 +0000 UTC, topic = orders
2023/05/03 19:06:19 OTLP partial success: empty message (0 spans rejected)
2023/05/03 19:06:24 Message claimed: orderId = 9925dc65-e9e5-11ed-9a0b-f246b07f1489, timestamp =
2023-05-03 19:06:24.764 +0000 UTC, topic = orders
2023/05/03 19:06:29 rpc error: code = Unknown desc = data dropped due to high memory usage
```



Logs

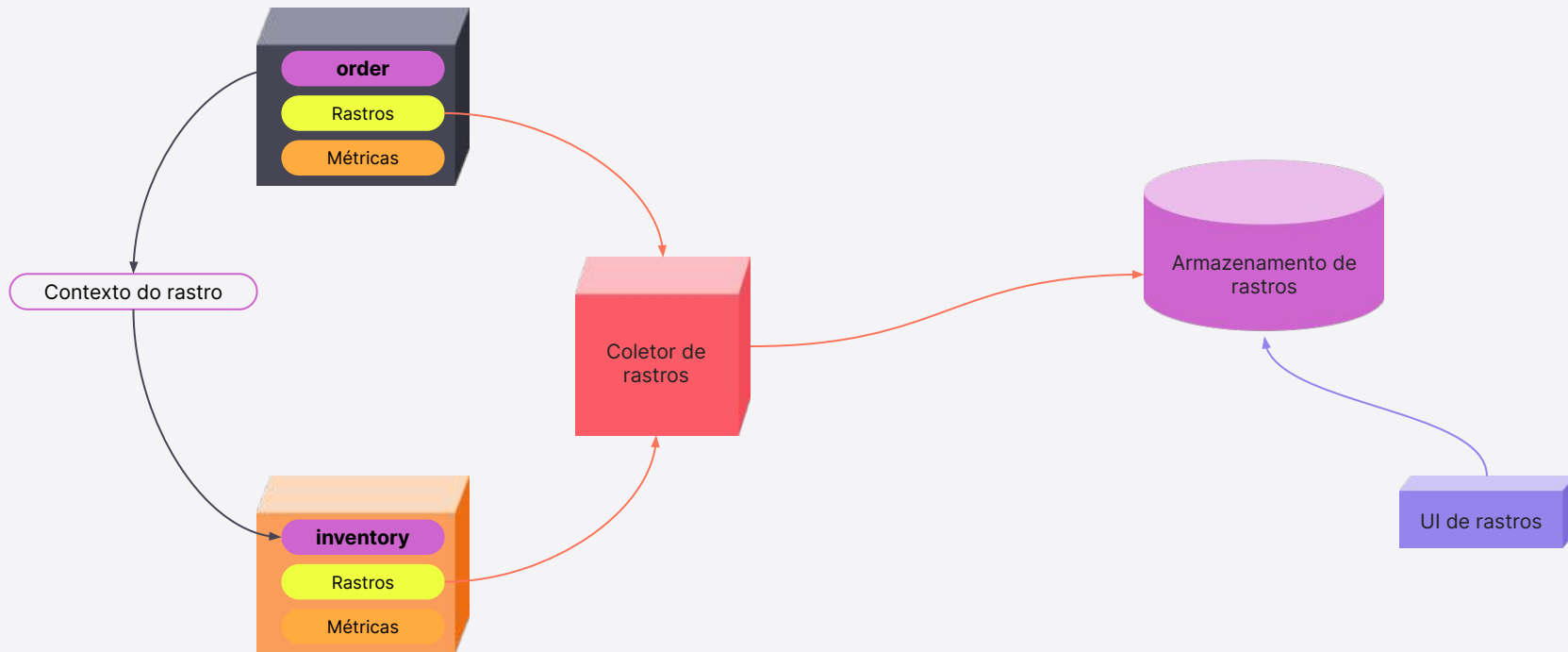


```
> kubectl logs my-otel-demo-frauddetectionservice-d76c767f8-p2w66
Picked up JAVA_TOOL_OPTIONS: -javaagent:/app/opentelemetry-javaagent.jar
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because
bootstrap classpath has been appended
[otel.javaagent 2023-05-03 19:03:27:176 +0000] [main] INFO
io.opentelemetry.javaagent.tooling.VersionLogger - opentelemetry-javaagent - version: 1.22.1
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Consumed record with id: 7c86783b-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 1
Consumed record with orderId: 8133c708-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 2
Consumed record with orderId: 816e0495-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 3
Consumed record with orderId: 87ba0a89-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 4
Consumed record with orderId: 93e3cd62-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 5
Consumed record with orderId: 9925dc65-e9e5-11ed-9a0b-f246b07f1489, and updated total count to: 6
[otel.javaagent 2023-05-03 19:06:27:784 +0000] [OkHttp http://k3d-k3s-default-server-0:4317/...]
WARN io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export metrics. Server
responded with gRPC status code 2. Error message: data dropped due to high memory usage
[otel.javaagent 2023-05-03 19:06:27:881 +0000] [OkHttp http://k3d-k3s-default-server-0:4317/...]
WARN io.opentelemetry.exporter.internal.grpc.OkHttpGrpcExporter - Failed to export spans. Server
responded with gRPC status code 2. Error message: data dropped due to high memory usage
```



Microserviços com rastreamento distribuído

→ Escrita
→ Consulta



Exemplo de cabeçalho HTTP de contexto

traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01

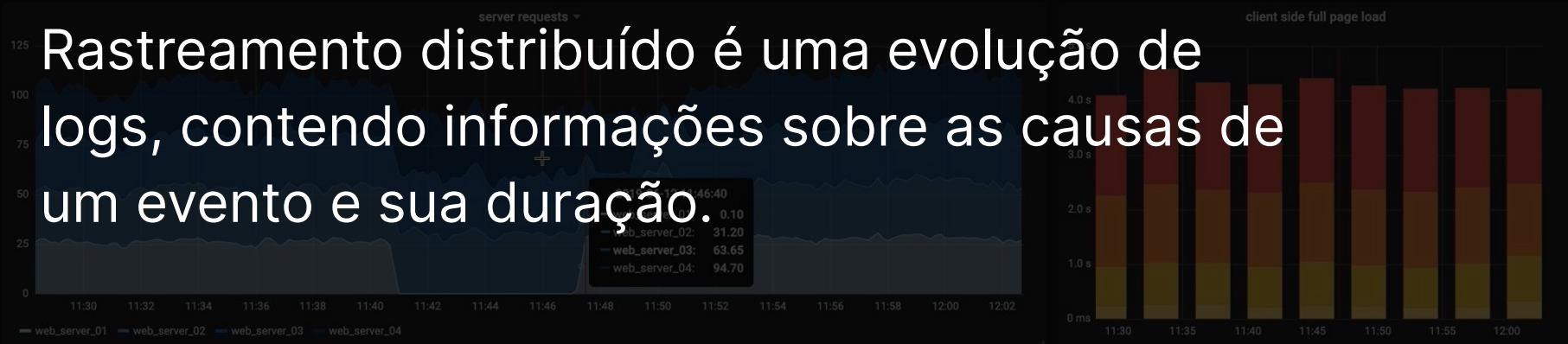
Versão do formato

ID do trecho (span ID)

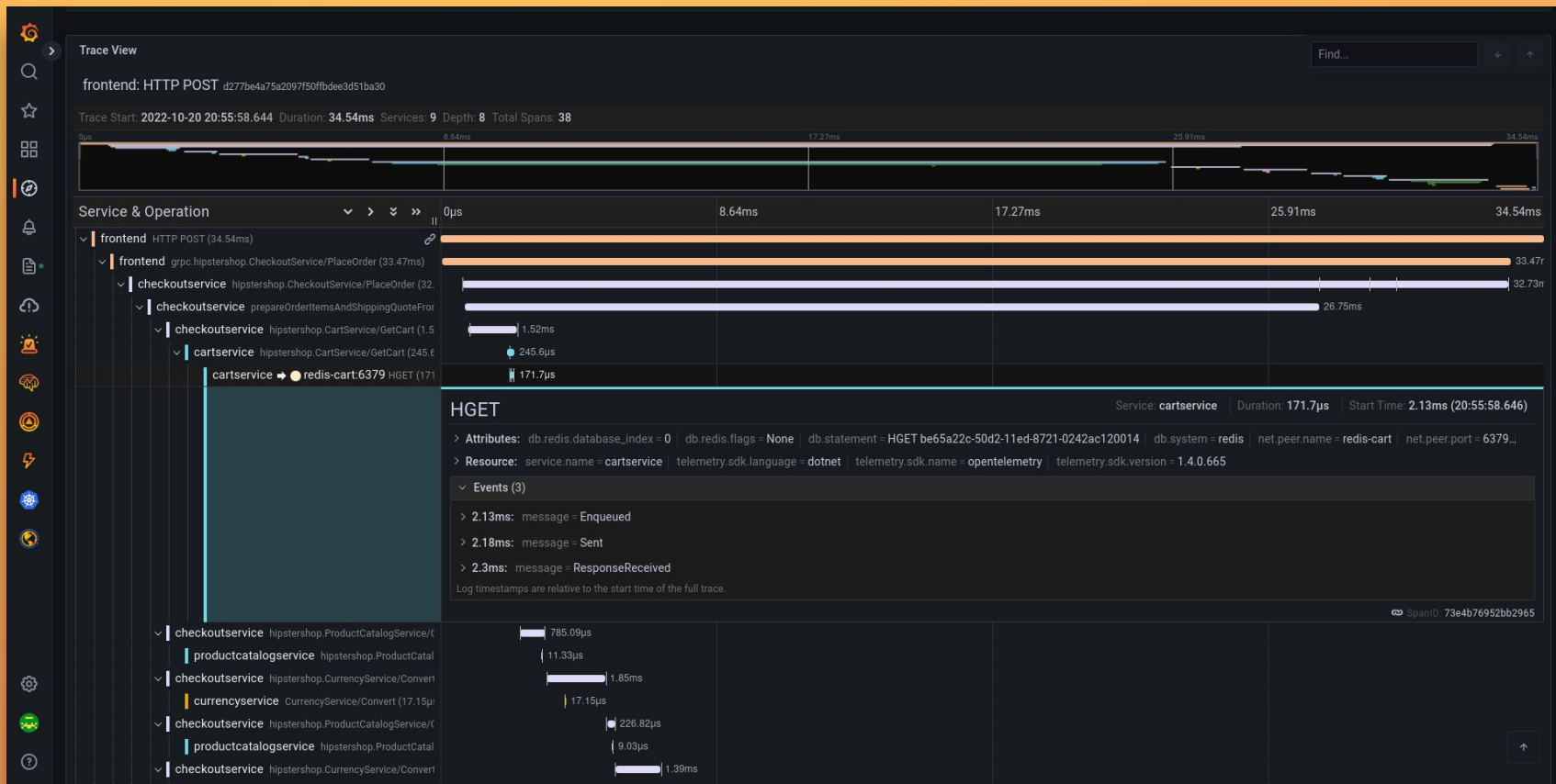
Flags

ID do rastro (trace ID)





Nosso objetivo



Conceitos de rastreamento distribuído

Trecho

Ou “span”,
representa uma
unidade de
trabalho com
duração e
informação de
causalidade

Rastro

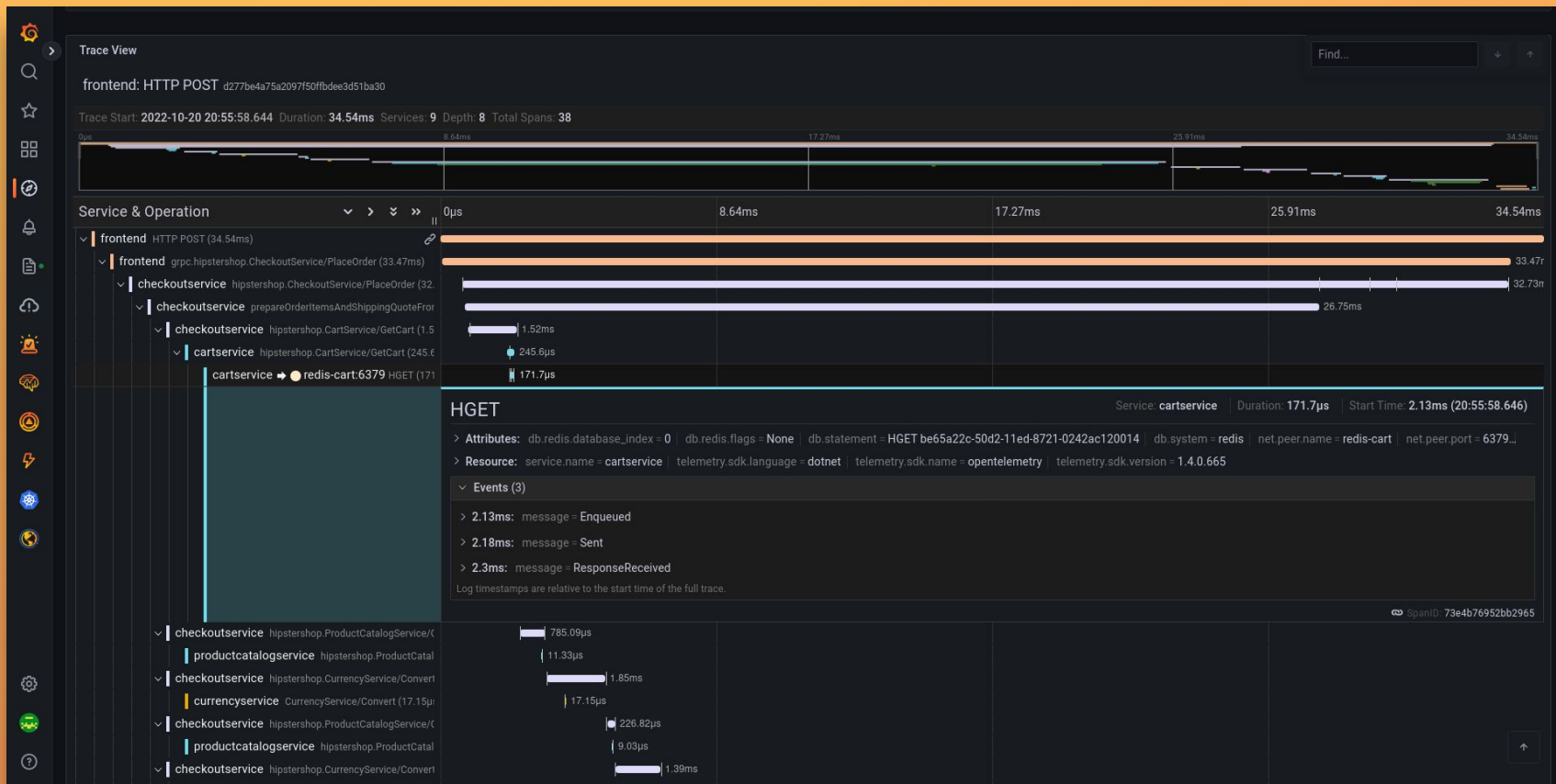
Coleção de
trechos
relacionados a
uma transação em
específico

Contexto

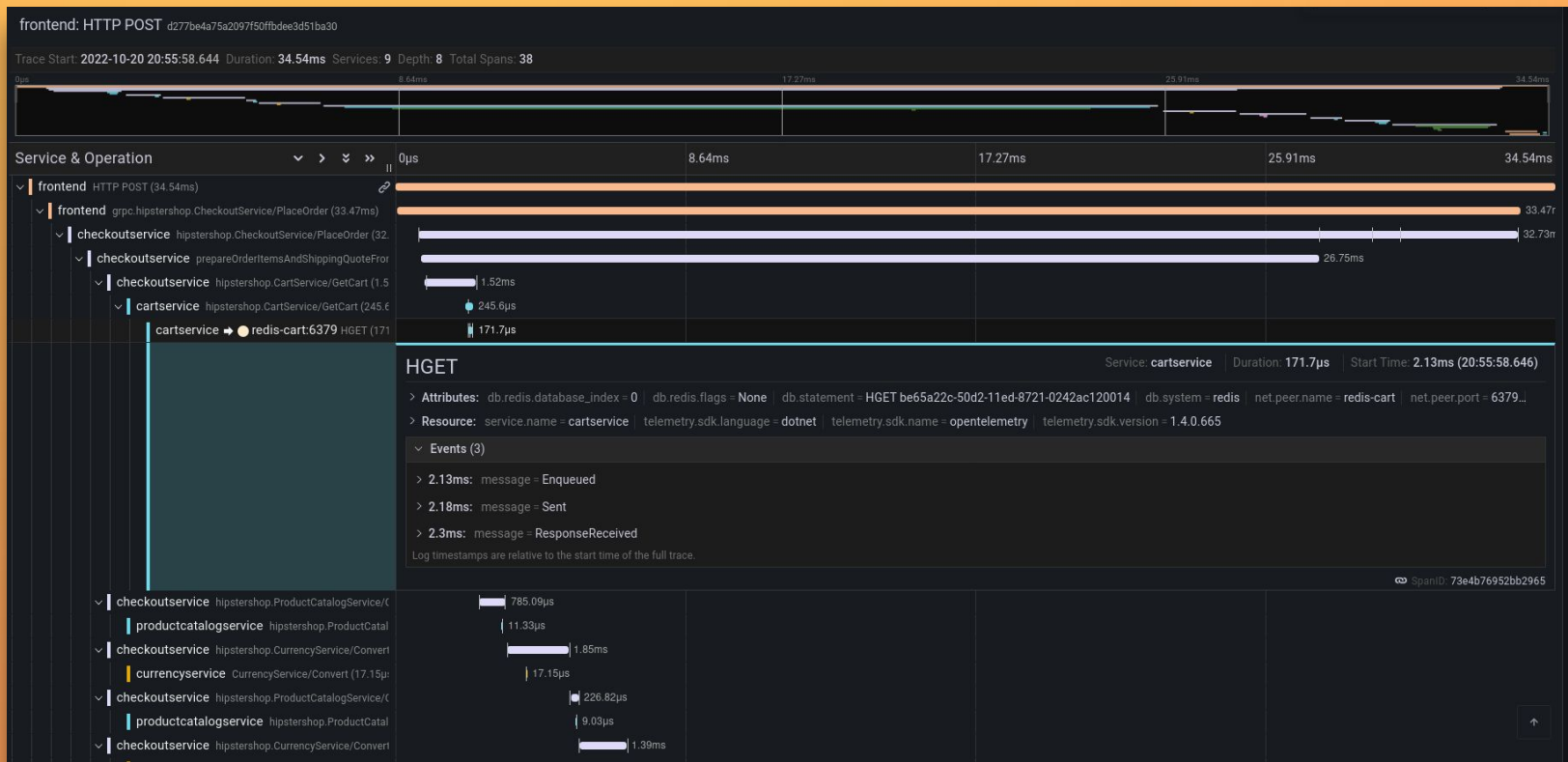
Informação
propagada de span
para span, como
tracelD, parent
spanID, flags,
baggage, ...



Nosso objetivo



Rastro



Trecho

cartservice → redis-cart:6379 HGET (171) 171.7µs

HGET

Service: **cartservice** | Duration: **171.7µs** | Start Time: **2.13ms (20:55:58.646)**

> **Attributes:** db.redis.database_index = 0 | db.redis.flags = None | db.statement = HGET be65a22c-50d2-11ed-8721-0242ac120014 | db.system = redis | net.peer.name = redis-cart | net.peer.port = 6379...

> **Resource:** service.name = cartservice | telemetry.sdk.language = dotnet | telemetry.sdk.name = opentelemetry | telemetry.sdk.version = 1.4.0.665

▼ **Events (3)**

- > **2.13ms:** message = Enqueued
- > **2.18ms:** message = Sent
- > **2.3ms:** message = ResponseReceived

Log timestamps are relative to the start time of the full trace.

SpanID: 73e4b76952bb2965



Conceitos de rastreamento distribuído

Rastreador

Biblioteca que sabe como enviar os dados para um collector.

Instrumentação

Instruções para o rastreador, dizendo o que precisa ser gravado, incluindo quais atributos armazenar, qual o nome da operação, ...



Instrumentação na prática - Java

```
124  @Override
125  public void getAds(AdRequest req, StreamObserver<AdResponse> responseObserver) {
126      AdService service = AdService.getInstance();
127
128      // get the current span in context
129      Span span = Span.current();
130      try {
131          List<Ad> allAds = new ArrayList<>();
132          AdRequestType adRequestType;
133          AdResponseType adResponseType;
134
135          span.setAttribute("app.ads.contextKeys", req.getContextKeysList().toString());
136          span.setAttribute("app.ads.contextKeys.count", req.getContextKeysCount());
137          logger.info("received ad request (context_words=" + req.getContextKeysList() + ")");
138          if (req.getContextKeysCount() > 0) {
139              for (int i = 0; i < req.getContextKeysCount(); i++) {
140                  Collection<Ad> ads = service.getAdsByCategory(req.getContextKeys(i));
141                  allAds.addAll(ads);
142              }
143              adRequestType = AdRequestType.TARGETED;
```



Instrumentação na prática - Java

```
124     @Override
125     public void getAds(AdRequest req, StreamObserver<AdResponse> responseObserver) {
126         AdService service = AdService.getInstance();
127
128         // get the current span in context
129         Span span = Span.current();
130
131         try {
132             List<Ad> allAds = new ArrayList<>();
133             AdRequestType adRequestType;
134             AdResponseType adResponseType;
135
136             span.setAttribute("app.ads.contextKeys", req.getContextKeysList().toString());
137             span.setAttribute("app.ads.contextKeys.count", req.getContextKeysCount());
138             logger.info("received ad request (context_words=" + req.getContextKeysList() + ")");
139             if (req.getContextKeysCount() > 0) {
140                 for (int i = 0; i < req.getContextKeysCount(); i++) {
141                     Collection<Ad> ads = service.getAdsByCategory(req.getContextKeys(i));
142                     allAds.addAll(ads);
143                 }
144                 adRequestType = AdRequestType.TARGETED;
```



Linguagens suportadas



C++



C# / .NET



Erlang / Elixir



Go



Java



JavaScript



PHP



Python



Ruby



Rust



Swift





Utilize bibliotecas
de instrumentação!

Ou ainda, auto-
instrumentação!



W3C Trace Context

Dapper

- Publicado pelo Google em 2010
- Introduz os conceitos de trechos e rastros, usados por todas as ferramentas atuais de rastreamento distribuído

OpenTracing

- Alguns dos mesmos nomes do Dapper e Zipkin se reúnem pra criar um padrão para rastreamento distribuído
- APIs de instrumentação
- Rastreadores são a implementação da API

- Reconhecendo que nem todos vão adotar as mesmas bibliotecas de instrumentação, pessoas de diferentes empresas e projetos se reúnem pra criar um padrão para propagação de contexto

OpenTelemetry

- Junção dos projetos OpenTracing e OpenCensus
- APIs e SDKs
- Convenções semânticas
- Middleware (OpenTelemetry Collector)

Zipkin

- Twitter disponibiliza o Zipkin, uma implementação do Dapper com código aberto
- Baseado em Java
- Múltiplas opções de armazenamento

Jaeger

- Uber disponibiliza seu sistema de rastreamento distribuído, Jaeger
- Feito para substituir Zipkin internamente na Uber, adotando OpenTracing nativamente
- Baseado em Go
- Doador para a CNCF em 2017

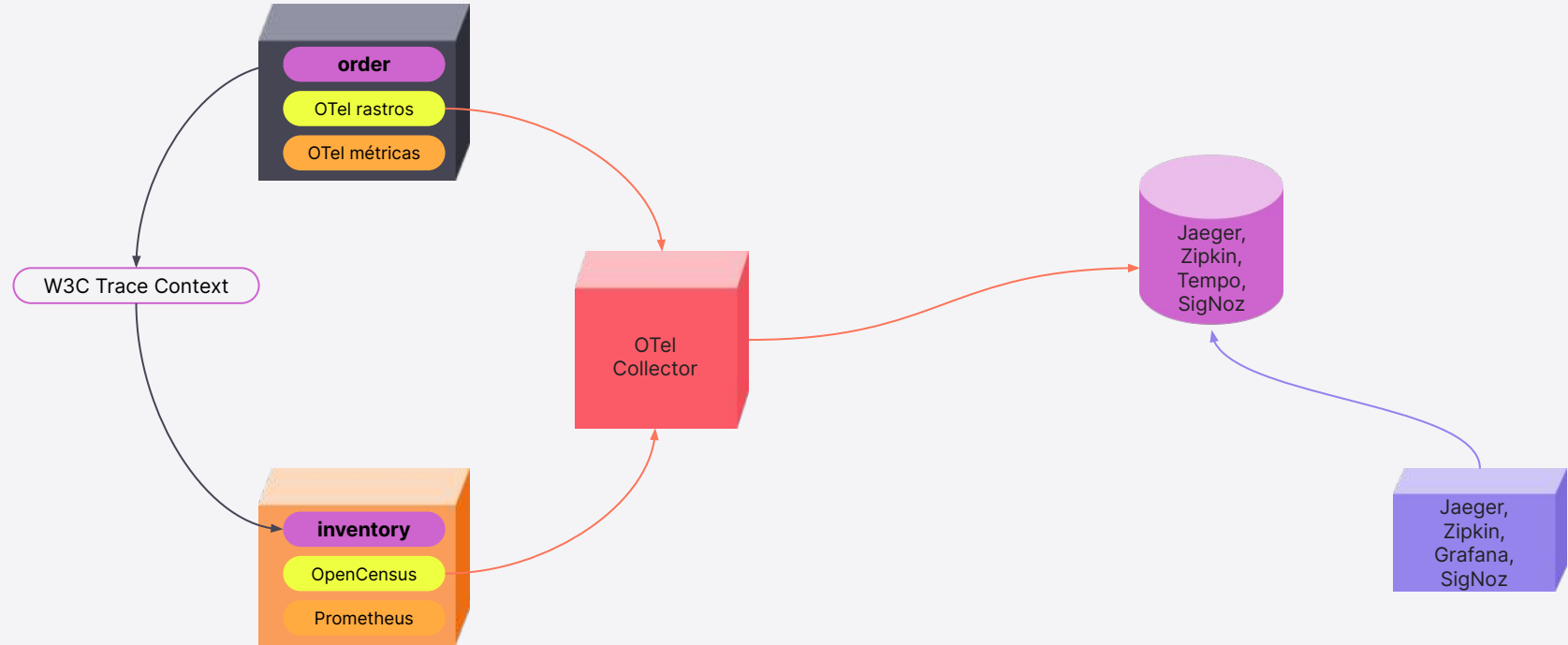
OpenCensus

- OpenCensus é disponibilizado pela empresa Google como um conjunto de bibliotecas para instrumentação de aplicações
- Instrumentação existe para métricas e rastros
- APIs e SDKs (implementações das APIs) fazem parte do mesmo pacote
- OpenCensus Service como middleware



Estado da arte - arquitetura mista

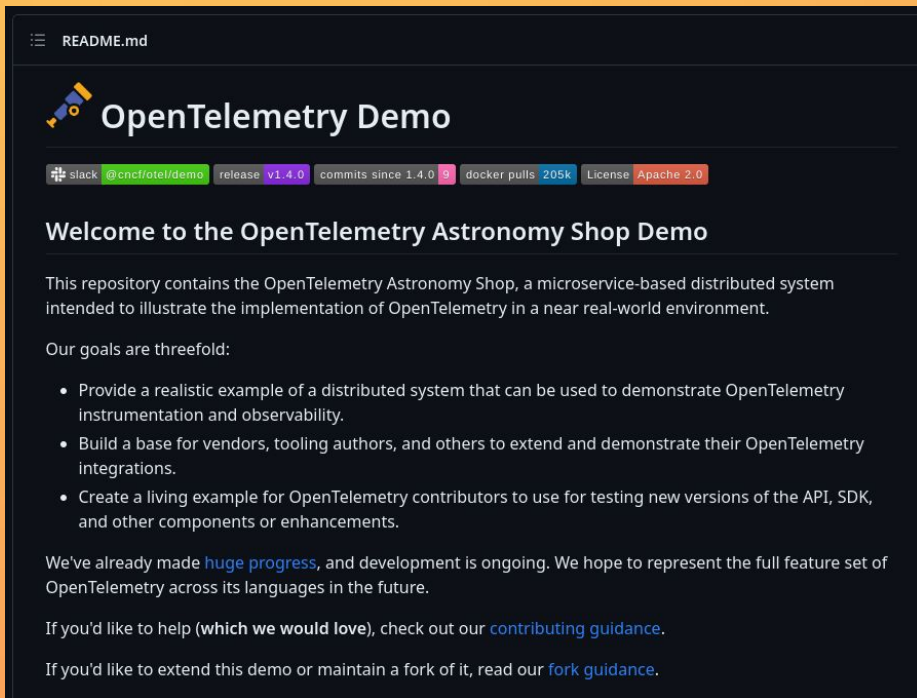
→ Escrita
→ Consulta



Demonstração



Demo



The screenshot shows the README for the OpenTelemetry Demo repository. At the top, there's a navigation menu with 'README.md' selected. Below the menu is the repository name 'OpenTelemetry Demo' with a rocket icon. A horizontal bar contains several badges: 'slack @cncf/otel/demo', 'release v1.4.0', 'commits since 1.4.0 9', 'docker pulls 205k', and 'License Apache 2.0'. The main heading is 'Welcome to the OpenTelemetry Astronomy Shop Demo'. The text describes the repository as containing the OpenTelemetry Astronomy Shop, a microservice-based distributed system. It lists three goals: providing a realistic example of a distributed system, building a base for vendors and tooling authors, and creating a living example for contributors. It also mentions that development is ongoing and provides links for contributing guidance and fork guidance.

README.md

OpenTelemetry Demo

slack @cncf/otel/demo release v1.4.0 commits since 1.4.0 9 docker pulls 205k License Apache 2.0

Welcome to the OpenTelemetry Astronomy Shop Demo

This repository contains the OpenTelemetry Astronomy Shop, a microservice-based distributed system intended to illustrate the implementation of OpenTelemetry in a near real-world environment.

Our goals are threefold:

- Provide a realistic example of a distributed system that can be used to demonstrate OpenTelemetry instrumentation and observability.
- Build a base for vendors, tooling authors, and others to extend and demonstrate their OpenTelemetry integrations.
- Create a living example for OpenTelemetry contributors to use for testing new versions of the API, SDK, and other components or enhancements.

We've already made [huge progress](#), and development is ongoing. We hope to represent the full feature set of OpenTelemetry across its languages in the future.

If you'd like to help (which we would love), check out our [contributing guidance](#).

If you'd like to extend this demo or maintain a fork of it, read our [fork guidance](#).

github.com/open-telemetry/opentelemetry-demo



Instrumentação - accountingservice

```
github.com/Shopify/sarama/logrus
"go.opentelemetry.io/contrib/instrumentation/github.com/Shopify/sarama/otel[sarama]"
"google.golang.org/protobuf/proto"

})

var (
    Topic           = "orders"
    ProtocolVersion = sarama.V3_0_0_0
    GroupID         = "accountingservice"
)

func StartConsumerGroup(ctx context.Context, brokers []string, log *logrus.Logger) error {
    saramaConfig := sarama.NewConfig()
    saramaConfig.Version = ProtocolVersion
    // So we can know the partition and offset of messages.
    saramaConfig.Producer.Return.Successes = true

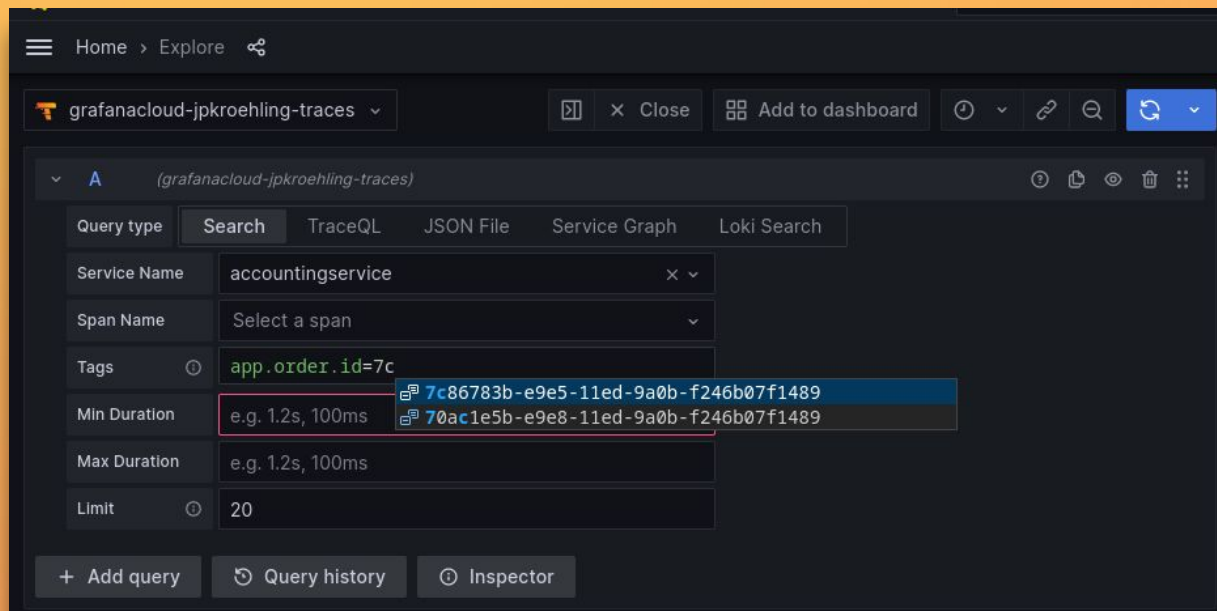
    consumerGroup, err := sarama.NewConsumerGroup(brokers, GroupID, saramaConfig)
    if err != nil {
        return err
    }

    handler := groupHandler{
        log: log,
    }
    wrappedHandler := otelsarama.WrapConsumerGroupHandler(&handler)

    err = consumerGroup.Consume(ctx, []string{Topic}, wrappedHandler)
    if err != nil {
        return err
    }
}
```



Busca por atributos de negócio



The screenshot displays the Grafana Cloud interface for searching traces. The breadcrumb navigation shows 'Home > Explore'. The selected dashboard is 'grafanacloud-jpkroehling-traces'. The 'Query type' is set to 'Search'. The search filters are as follows:

- Service Name:** accountingservice
- Span Name:** Select a span
- Tags:** app.order.id=7c
- Min Duration:** e.g. 1.2s, 100ms
- Max Duration:** e.g. 1.2s, 100ms
- Limit:** 20

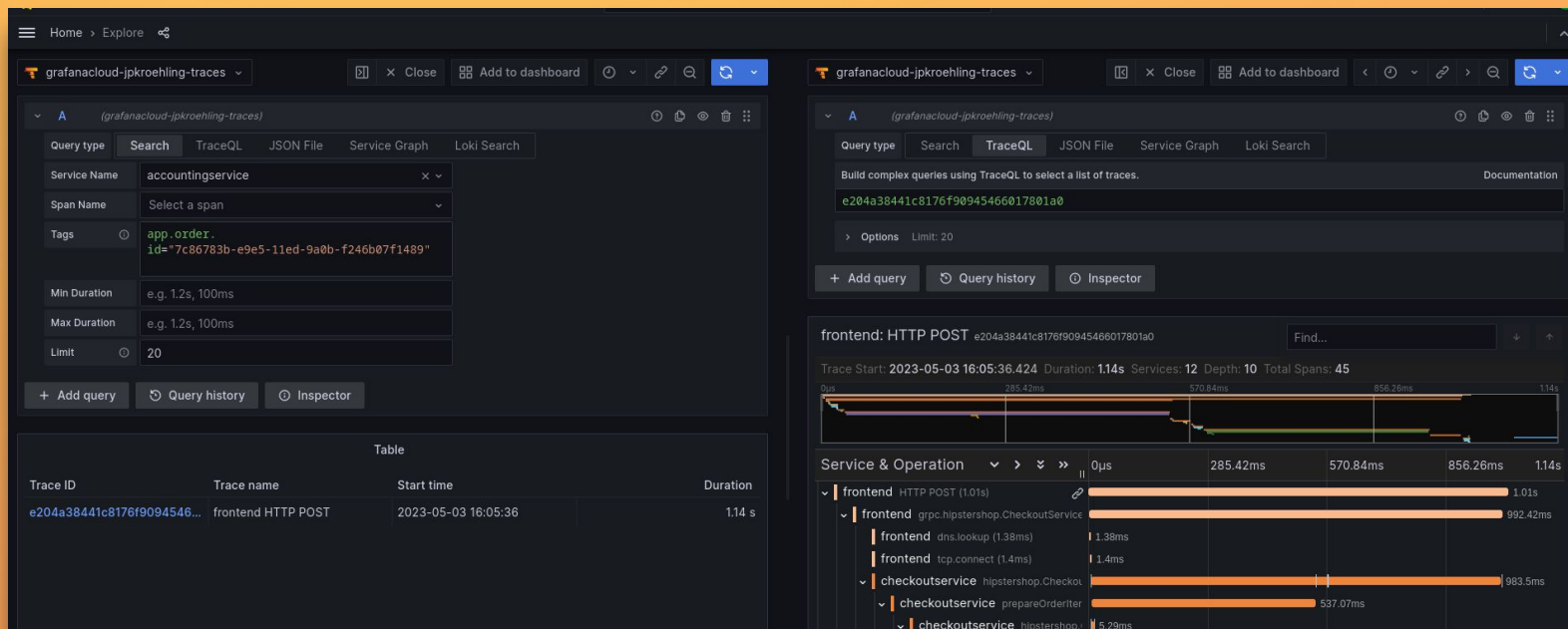
A dropdown menu is open for the 'Min Duration' field, showing two options:

- 7c86783b-e9e5-11ed-9a0b-f246b07f1489
- 70ac1e5b-e9e8-11ed-9a0b-f246b07f1489

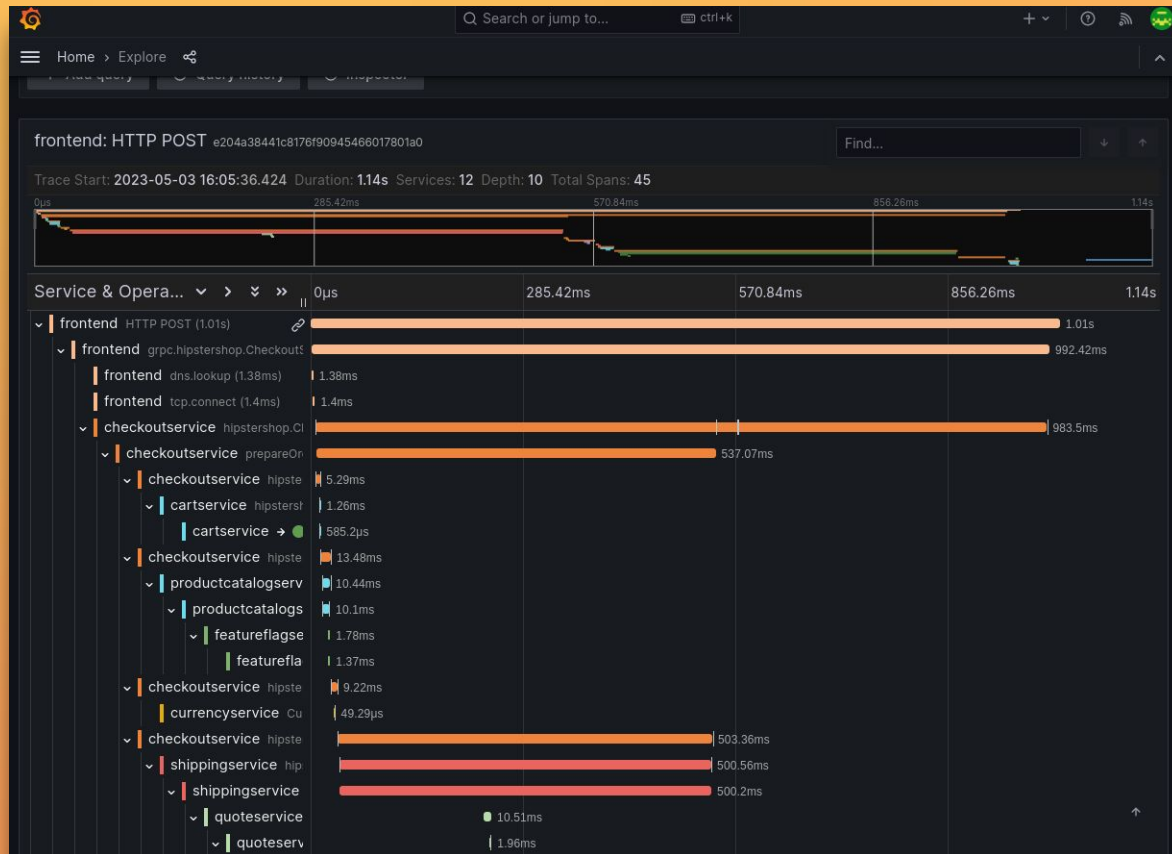
At the bottom of the interface, there are buttons for '+ Add query', 'Query history', and 'Inspector'.



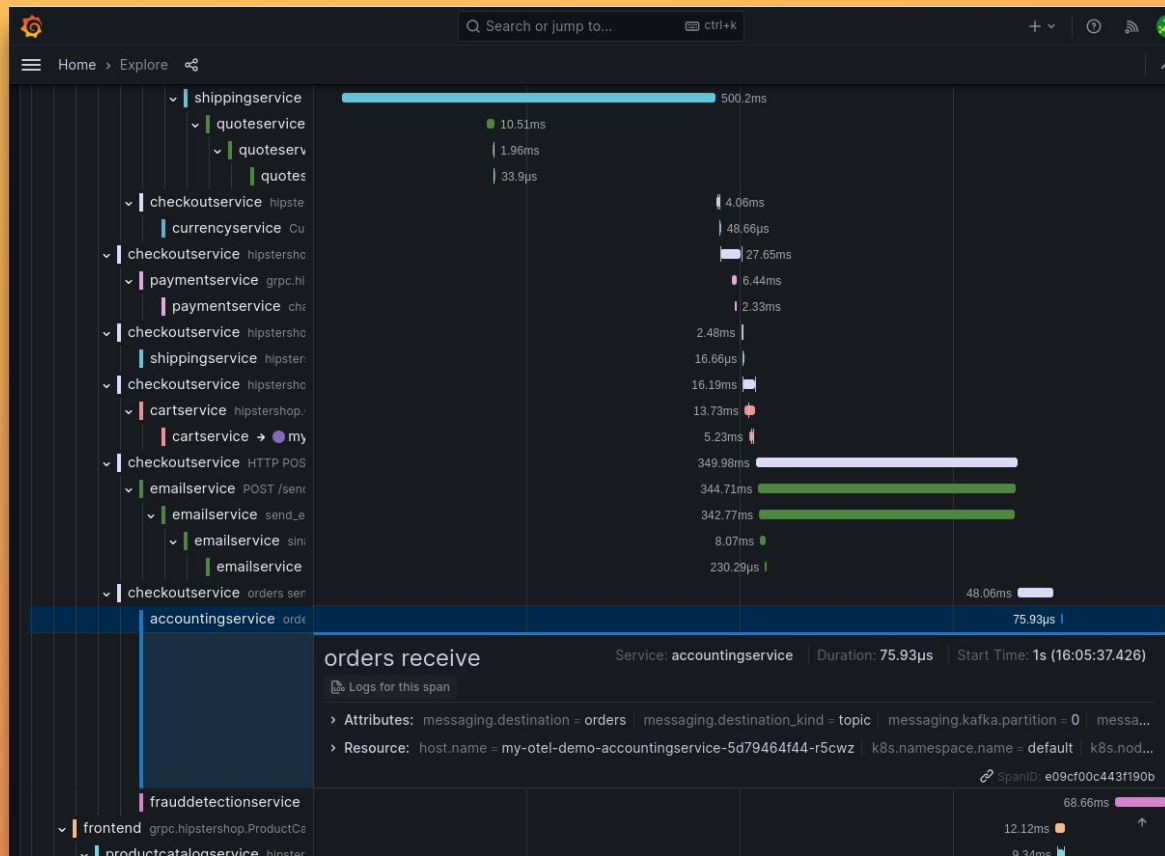
Todos os eventos relacionados



Todos os eventos relacionados



O trecho do accountingservice



Desafios



Desafios

- Não temos como saber se um rastro foi finalizado, novos trechos podem chegar a qualquer momento
- Volume de dados é enorme: trabalhar com amostragem é comum
- Visualização e busca de rastros ainda podem melhorar
- Armazenamento com soluções genéricas geralmente são ineficientes
- Falha na coleta de um item na cadeia de eventos é o suficiente pra gerar incertezas



Resumo



Pontos chave

- Rastreamento distribuído é essencial quando temos um arquitetura distribuída, como em microsserviços, para compreender o que se passa por trás de cada requisição.
- OpenTelemetry é o padrão de mercado atual no que se refere a rastreamento distribuído
- As APIs do OpenTelemetry podem ser usadas para instrumentar serviços
- As bibliotecas de instrumentação te dão super poderes, auxiliando na compreensão do que sua aplicação está fazendo como um todo
- Instrumentação automática, como a fornecida por agentes Java, são úteis para ter resultados imediatos



Perguntas e respostas



Muito obrigado pela
atenção!