



# Distributed Tracing

Live de Python #265



## 1. Vocabulário de nicho

Rastro e Distribuído

## 2. OTel

Os conceitos do tracing, componentes e semântica

## 3. Instrumentação Manual

Como fazer na mão

## 4. Instrumentação Automática

A forma mais simples

Título	Data
Observabilidade - Uma introdução <i>[Link na descrição]</i>	11/03
Observabilidade - Métricas (Opentelemetry / Prometheus)	25/03
Observabilidade - Rastreamento / Tracing (Opentelemetry / Tempo / Jeager)	<b>Agora!</b>
Observabilidade - Logs (Opentelemetry / Loki)	15/04
Mão na massa com OpenTelemetry	29/04



Isso é uma série!





# Mango Bits

@mangobits · 66 inscritos · 6 vídeos

Este é um canal destinado a falar sobre observabilidade e coisas nativas da nuvem. >

 Inscrito 

<https://www.youtube.com/@mangobits>



[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



[patreon.com/dunossauro](https://patreon.com/dunossauro)



Ajude o projeto <3



Ademar Peixoto, Adriano Casimiro, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alfredo Braga, Alisson Souza, Alysson Oliveira, Andre Paula, Antônio Filho, Apc 16, Aslay Clevisson, Aurelio Costa, Ber\_dev\_2099, Bernarducs, Bruno Almeida, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Ramos, Bruno Santos, Caio Nascimento, Carlos Ramos, Christian Fischer, Clara Battesini, Cleiton Gomes, Controlado, Cristian Firmino, Daniel Bianchi, Daniel Real, Daniel Wojcickoski, Danilo Boas, Danilo Silva, David Couto, David Kwast, Davi Souza, Dead Milkman, Denis Bernardo, Dgeison Peixoto, Diego Guimarães, Dino, Edgar, Eduard0ml, Elias Moura, Emerson Rafael, Ennio Ferreira, Erick Andrade, Érico Andrei, Everton Silva, Fabio Barros, Fábio Barros, Fabio Valente, Fabricio Biazotto, Felipe Augusto, Felipe Rodrigues, Fernanda Prado, Francisco Silvério, Frederico Damian, Fsouza, Gabrieimoreira, Gabriel Espindola, Gabriel Mizuno, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Giovanna Teodoro, Giuliano Silva, Guibeira, Guilherme, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Guilherme Silva, Gustavo Almeida, Gustavo Suto, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Henrique Andrade, Henrique Machado, Henrique Sebastião, Hiago Couto, Igor Taconi, Jairo Lenfers, Janael Pinheiro, Jean Victor, Jefferson Antunes, Jlx, Joelson Sartori, Jonatas Leon, Jônatas Silva, Jorge Silva, Jose Barroso, Jose Edmario, José Gomes, Joseito Júnior, Jose Mazolini, Josir Gomes, Jrborba, Juan Felipe, Juliana Machado, Julio Batista-silva, Julio Franco, Júlio Gazeta, Júlio Sanchez, Kaio Peixoto, Leandro Silva, Leandro Vieira, Leonan Ferreira, Leonardo Mello, Leonardo Nazareth, Lucas Carderelli, Lucas Lattari, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Schneider, Lucas Simon, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Leite, Luiz Carlos, Luiz Duarte, Luiz Lima, Luiz Paula, Mackilem Laan, Marcelo Araujo, Marcio Novaes, Marcio Silva, Marco Mello, Marcos Almeida, Marcos Gomes, Marcos Oliveira, Marina Passos, Mateus Lisboa, Matheus Silva, Matheus Vian, Mirian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Natália Araújo, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Pedro Gomes, Pedro Henrique, Peterson Santos, Philipe Vasconcellos, Pytonyc, Rafael, Rafael Araújo, Rafael Faccio, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Renan, Renan Sebastião, Renato José, Renato Moraes, Rene Pessoto, Renne Rocha, Ricardo Combat, Ricardo Silva, Ricardo Viana, Richard Costa, Rinaldo Magalhaes, Riverfount, Rjribeiro, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Santana, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rondismar Gomes, Rui Jr, Samanta Cicilia, Santhiago Cristiano, Selmison Miranda, Shinodinho, Téo Calvo, Thiago Araujo, Thiago Borges, Tiago Ferreira, Tiago Henrique, Tony Dias, Tyrone Damasceno, Valdir, Valdir Tegen, Varlei Menconi, Vinicius Bego, Viniciusccosta, Vinicius Silva, Vinicius Stein, Vladimir Lemos, Wagner Gabriel, Washington Teixeira, Willian Lopes, Wilson Duarte, Zeca Figueiredo

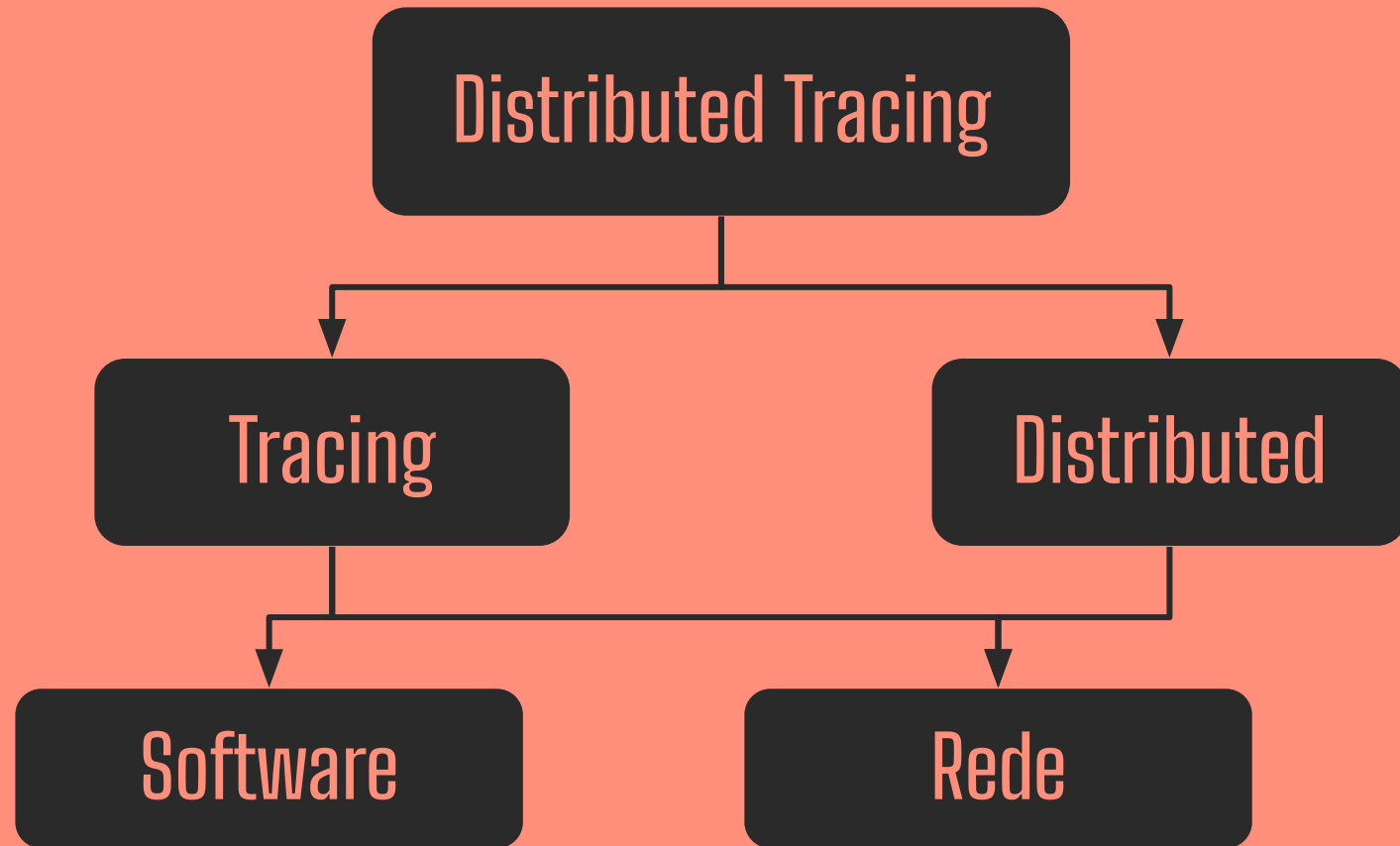


Obrigado você



Distributed Tracing

Vocab  
ulário





# Traces



- Trace Verbo:
  - Encontrar alguém ou algo que se perdeu
  - Traçar algo é também descobrir a sua causa ou origem
- Trace substantivo:
  - Um sinal de que algo aconteceu ou existiu



<https://dictionary.cambridge.org/dictionary/english/tracing>

<https://dictionary.cambridge.org/dictionary/english/trace>

# Trace de software

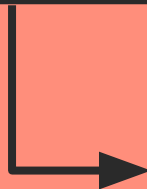


Subprograma A

Subprograma B

Subprograma C

Subprograma D



# Exemplo de trace



```
1  def d(): return 42
2  def c(): return d()
3  def b(): return c()
4  def a(): return b()
5
6  a()
```

# Exemplo de trace



```
1  def d(): return 42
2  def c(): return d()
3  def b(): return c()
4  def a(): return b()
5
6  a()
```

```
$ python -m trace -T exemplo_01.py
```

```
$ python -m trace -T exemplo_01.py
```

```
42
```

```
calling relationships:
```

```
*** /usr/lib/python3.11/trace.py ***
```

```
--> exemplo_01.py
```

```
    trace.Trace.runcctx -> exemplo_01.<module>
```

```
*** exemplo_01.py ***
```

```
    exemplo_01.<module> -> exemplo_01.a
```

```
    exemplo_01.a -> exemplo_01.b
```

```
    exemplo_01.b -> exemplo_01.c
```

```
    exemplo_01.c -> exemplo_01.d
```

# Traceback



Subprograma A

Subprograma B

Subprograma C

Subprograma D





```
1  def d(): return 1/0
2  def c(): return d()
3  def b(): return c()
4  def a(): return b()
5
6  a()
```

1

2

3

4

5

6

```
python exemplo_01.py
```

```
Traceback (most recent call last):
```

```
File "/home/dunossauro/live_traces/exemplo_01.py", line 7, in <module>
```

```
    print(a())
```

```
    ^^^
```

```
File "/home/dunossauro/live_traces/exemplo_01.py", line 4, in a
```

```
    def a(): return b()
```

```
    ^^^
```

```
File "/home/dunossauro/live_traces/exemplo_01.py", line 3, in b
```

```
    def b(): return c()
```

```
    ^^^
```

```
File "/home/dunossauro/live_traces/exemplo_01.py", line 2, in c
```

```
    def c(): return d()
```

```
    ^^^
```

```
File "/home/dunossauro/live_traces/exemplo_01.py", line 1, in d
```

```
    def d(): return 1/0
```

```
    ~^^
```

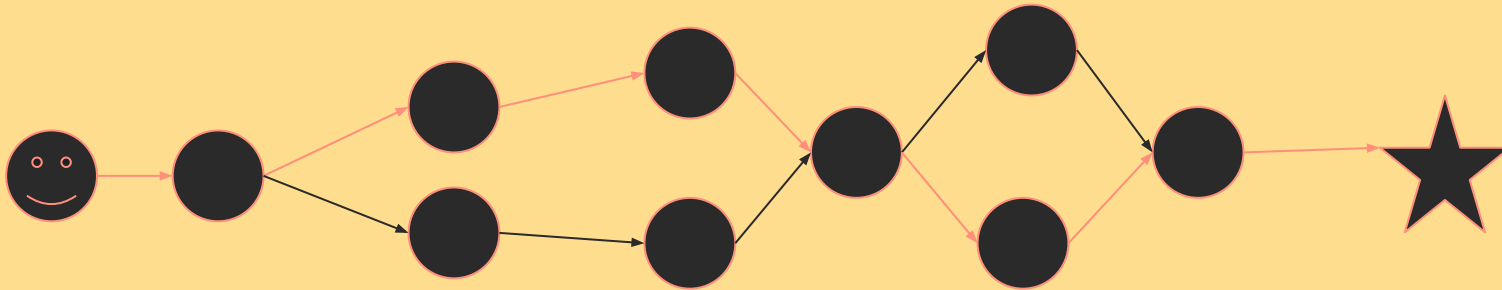
```
ZeroDivisionError: division by zero
```



# Trace de rede



Quando pensamos em um rede como a internet, quando precisamos acessar um site, por exemplo, temos um longo caminho a percorrer entre meios guiados e não guiados.



Precisamos sair do nosso computador, ir para o nosso roteador, que vai para o poste, que provavelmente tem outro roteador, que se liga em outro roteador, até chegar no computador (**servidor**) que está hospedando o site que queremos acessar.

# Traceroute



Uma boa forma de ver o caminho que uma requisição faz do seu computador até chegar no computador "requerido" é via o comando *traceroute*:

```
$ traceroute ddg.gg
```

```
traceroute to ddg.gg (40.89.244.232), 30 hops max, 60 byte packets
```

```
1  _gateway (192.168.15.1)  8.881 ms  8.983 ms  9.253 ms
```

```
2  * * *
```

```
6  * * *
```

```
7  * * *
```

```
8  * * *
```

```
19  be-4-0.ibr03.dsm05.ntwk.msn.net (104.44.28.248)  194.578 ms
```

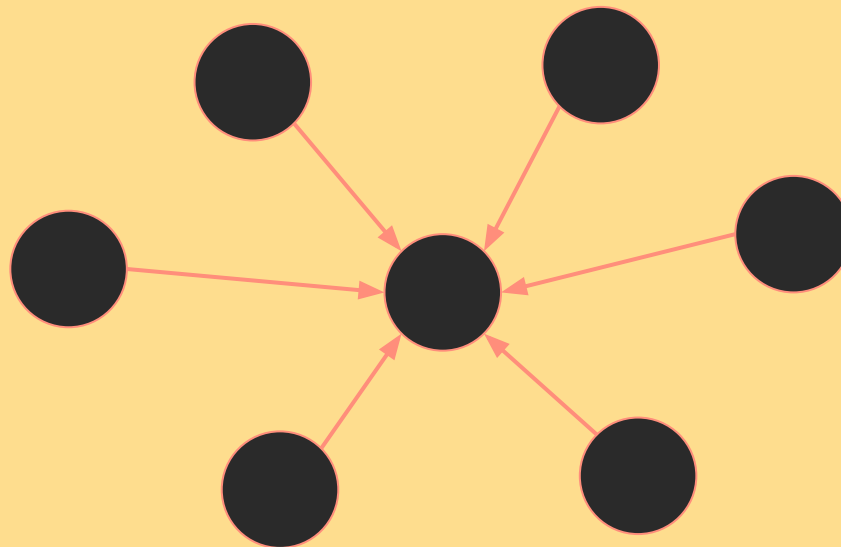
```
29  * * *
```

```
30  * * *
```

# Sistema distribuído



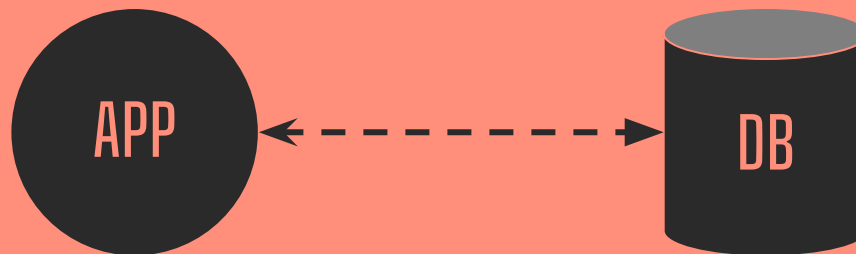
Quando escrevemos aplicações, geralmente fazemos uma única aplicação que resolve todos os nossos problemas. Chamamos isso de sistema centralizado.



# Sistema distribuído



Em alguns casos, vamos imaginar uma aplicação web que precisa de um banco de dados. Isso forma um sistema **simbiótico**. Onde os dados, para serem manipulados precisam de uma aplicação e a aplicação precisa do banco de dados para executar suas operações.

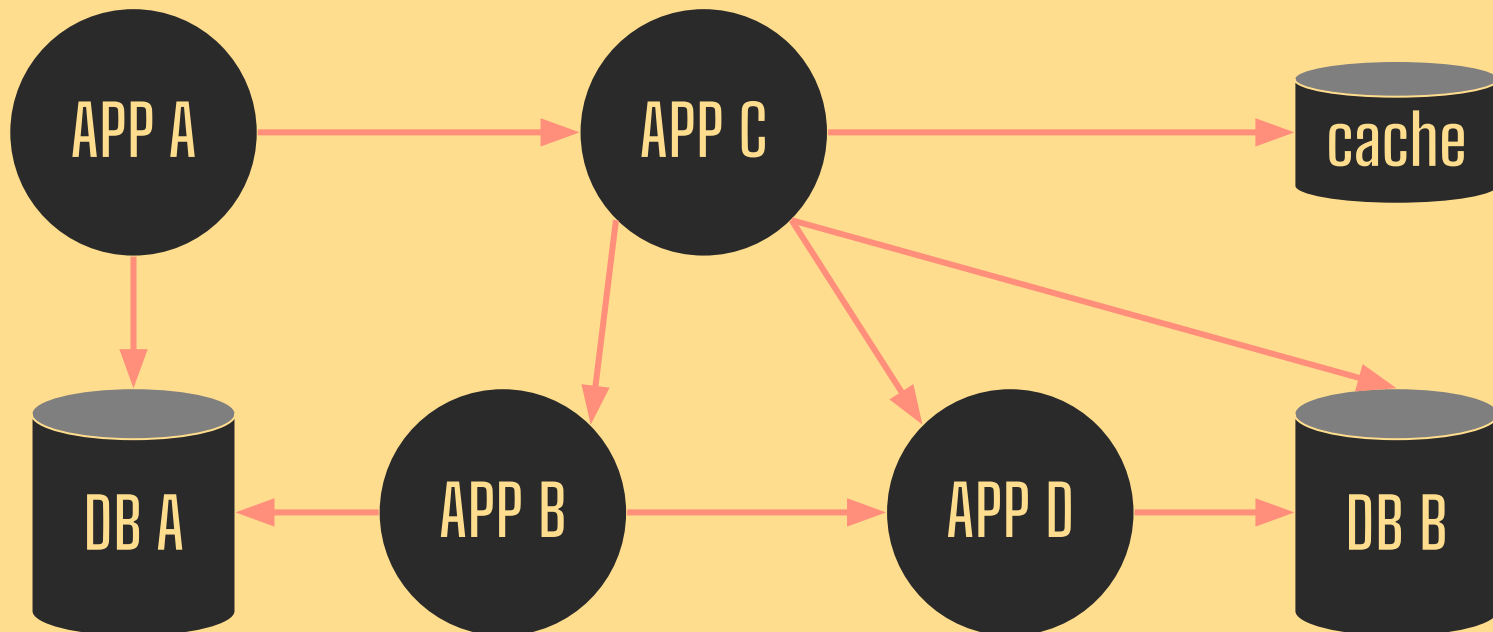


Por mais simples que esse sistema seja, ele é um **sistema distribuído**, pois os dois sistemas precisam trabalhar juntos para resolver uma tarefa.

# Sistema distribuído



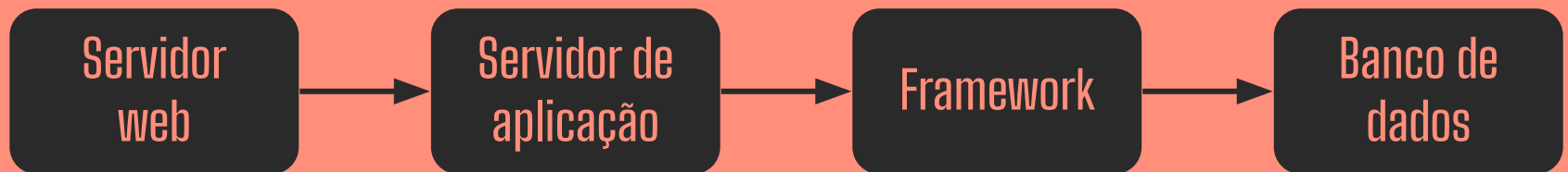
Outro termo comum que costumamos ouvir com uma certa frequência, são microsserviços. Que nada mais são do que pequenas aplicações compartilhando a resolução de tarefas específicas.



# O sistema básico tradicional



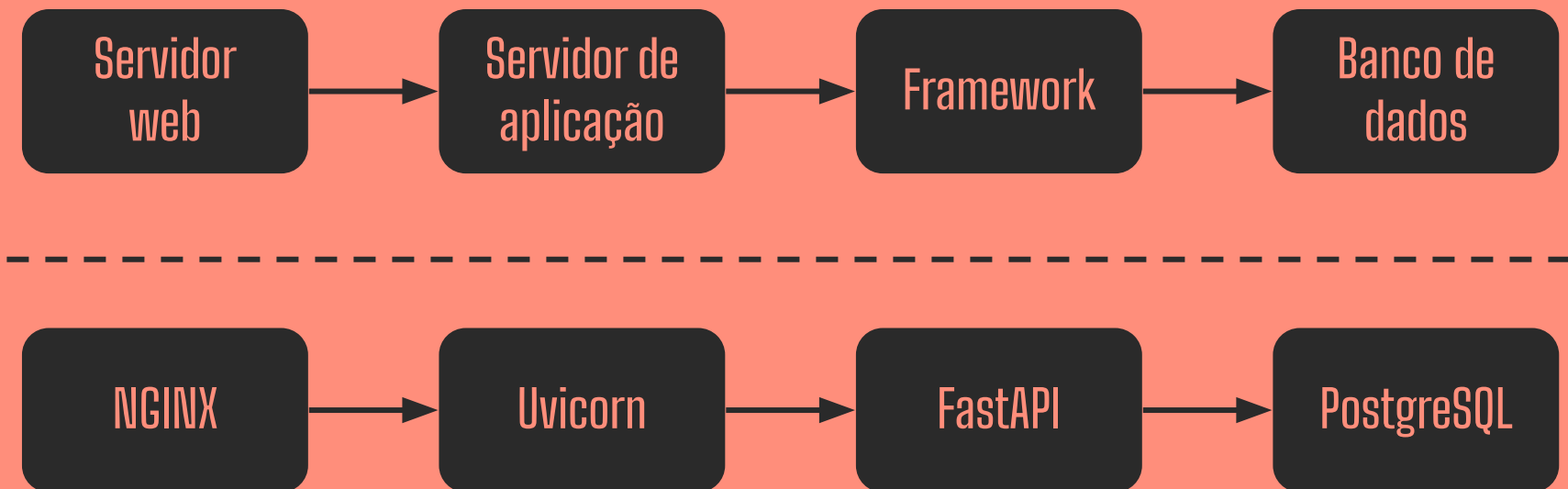
Talvez o ponto mais comum, mesmo quando estamos trabalhando com uma única aplicação web, é que existe uma distribuição de responsabilidades (SoC). Onde temos diferentes componentes interagindo:



# O sistema básico tradicional



Talvez o ponto mais comum, mesmo quando estamos trabalhando com uma única aplicação web, é que existe uma distribuição de responsabilidades (SoC). Onde temos diferentes componentes interagindo:



# Trace distribuído



A ideia do trace distribuído é conseguir juntar os dois conceitos que vimos anteriormente. O trace individual das aplicações com os rastros de rede entre elas. Por exemplo:

1. Cliente fez a requisição no servidor web
2. Servidor web delegou para servidor de aplicação
3. Servidor de aplicação delega para aplicação
4. A aplicação fez um **caminho** para resolver o problema
5. A aplicação faz algumas comunicações com o banco de dados

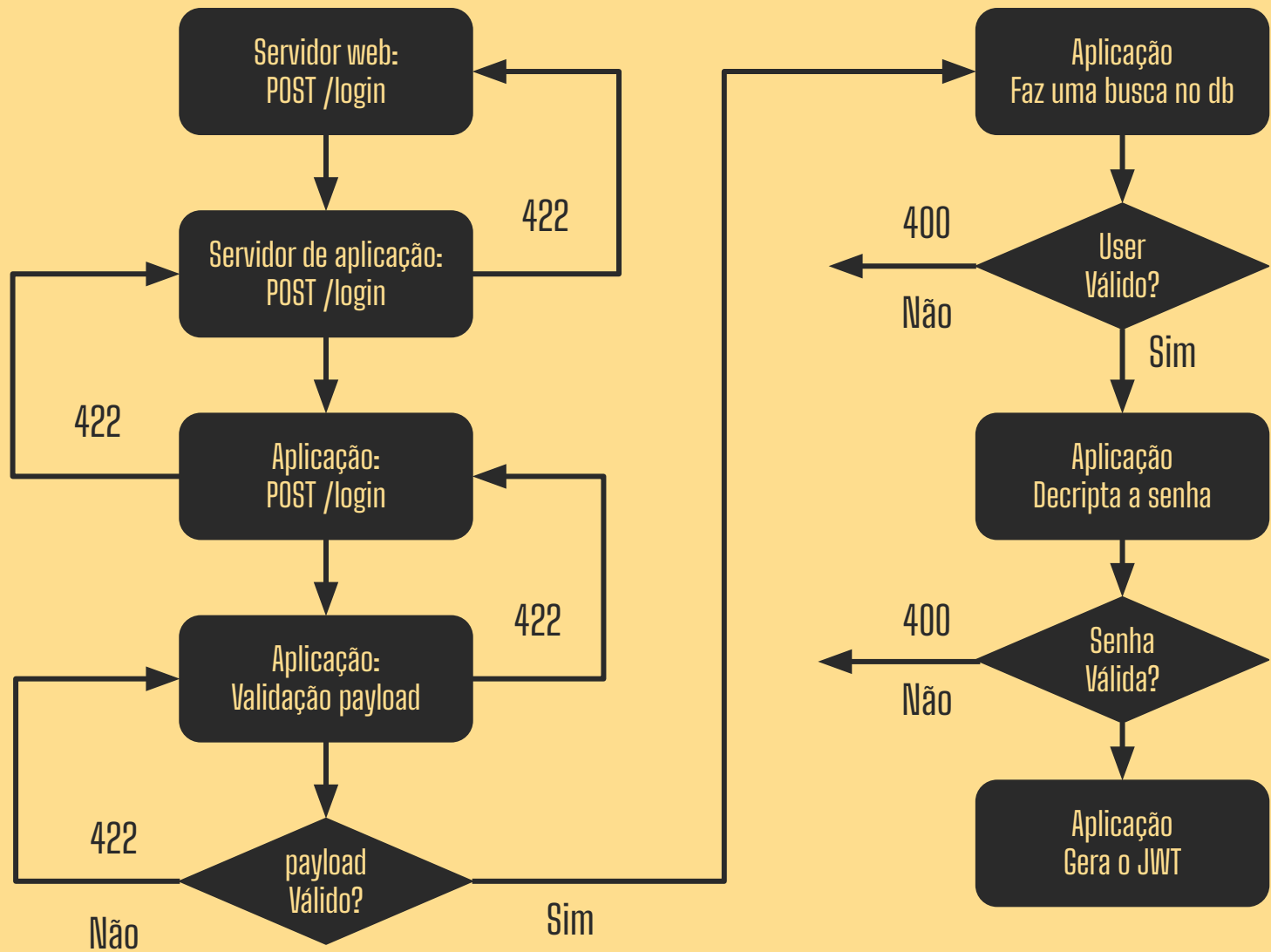
No final, o que queremos ver é o rastro gerado por um tipo de requisição específica.

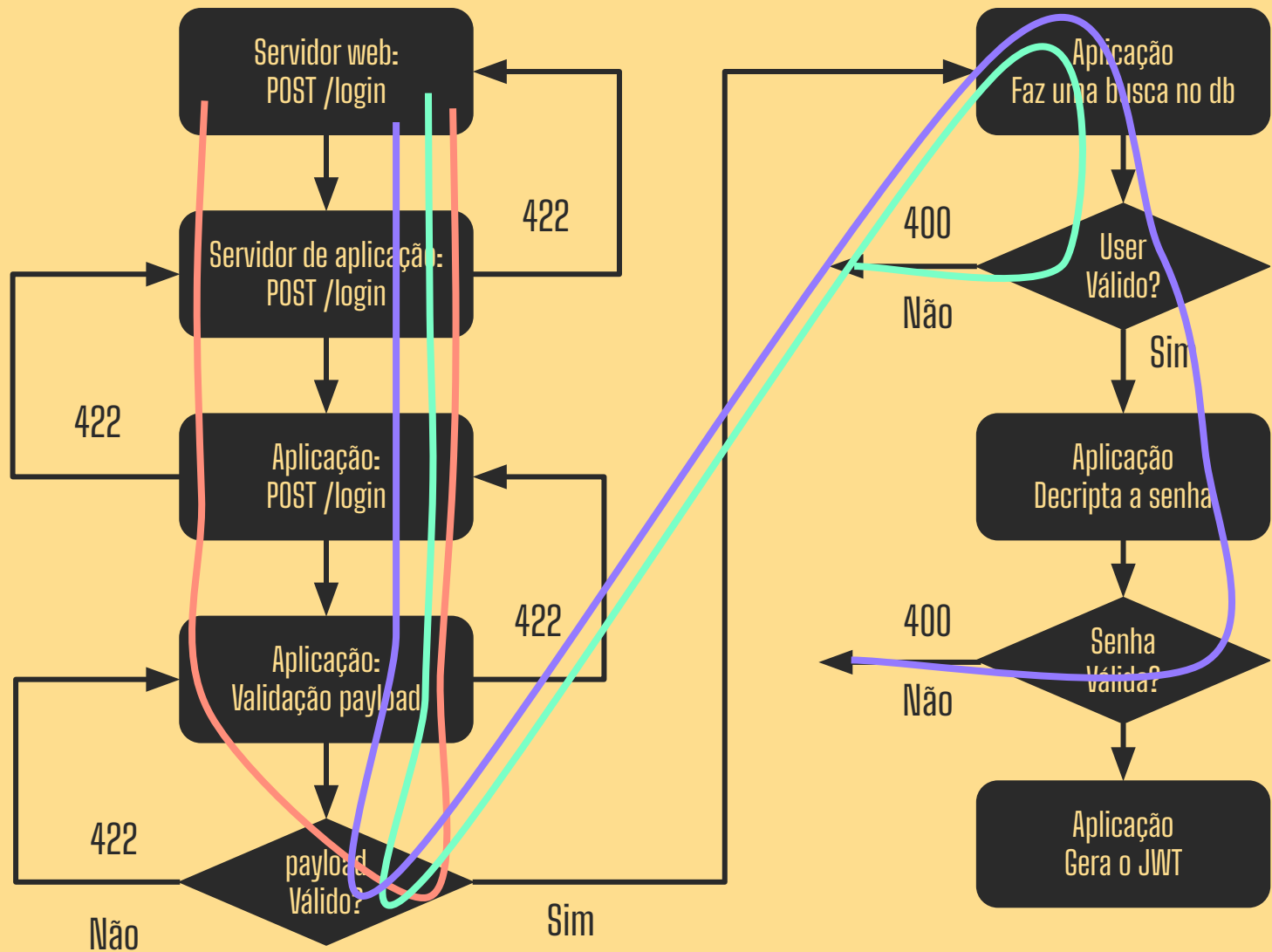


# Exemplo prático



1. Servidor web recebe o request POST em */login*
2. Servidor web repassa o POST em */login* para servidor de aplicação
3. Servidor de aplicação repassa o POST em */login* para a aplicação
4. A aplicação executa o bloco de código responsável pelo método POST no endpoint de */login*
5. Esse bloco de código pega os dados recebidos nos payload e faz a validação.
  - a. Se a validação deu errado, retorna 422
  - b. Se a validação deu certo, parte para 6
6. É feita a chamada para o banco de dados com os valores
7. O banco de dados retorna a chamada
8. É feita uma checagem se os dados estavam corretos
  - a. Caso estejam corretos, vamos para 9
  - b. Caso estejam incorretos, retornamos 400





# Bancos de dados de rastreamento

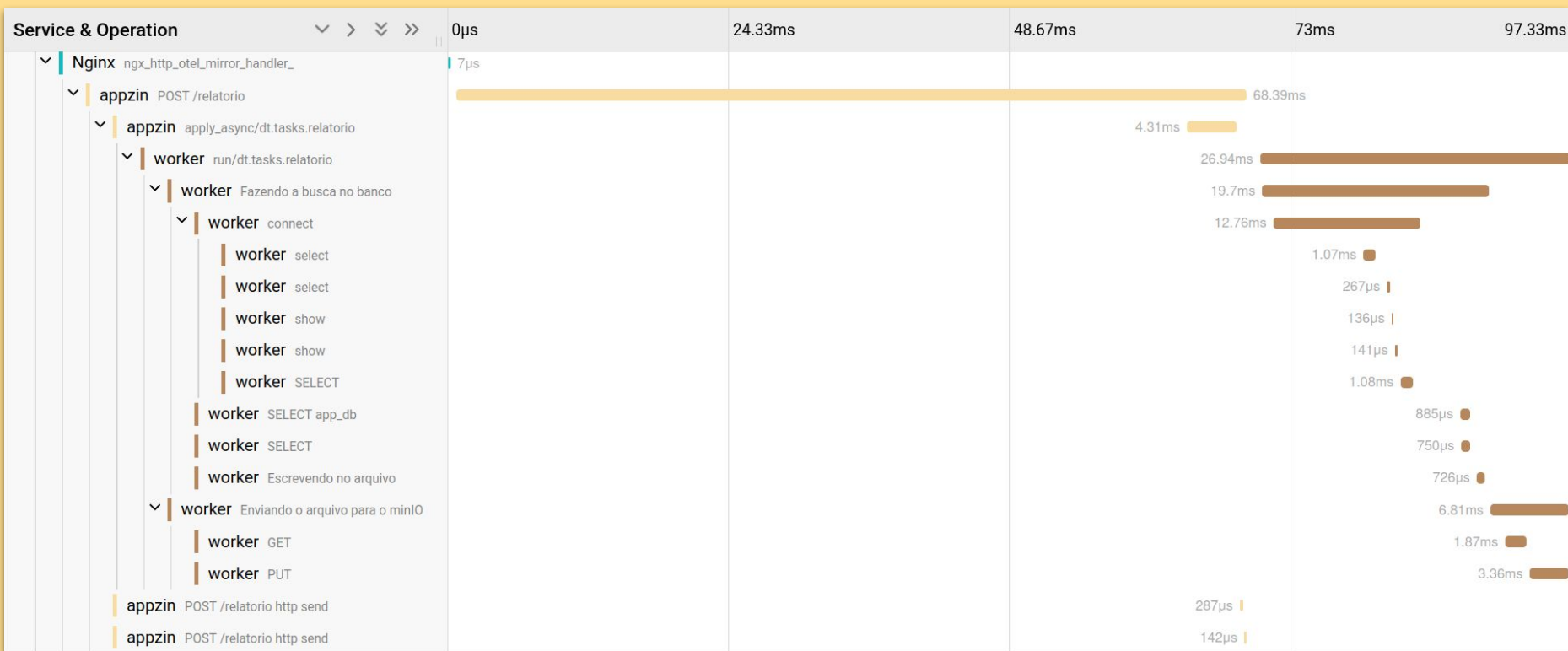


Existem algumas alternativas, mas acredito que as mais famosas sejam:

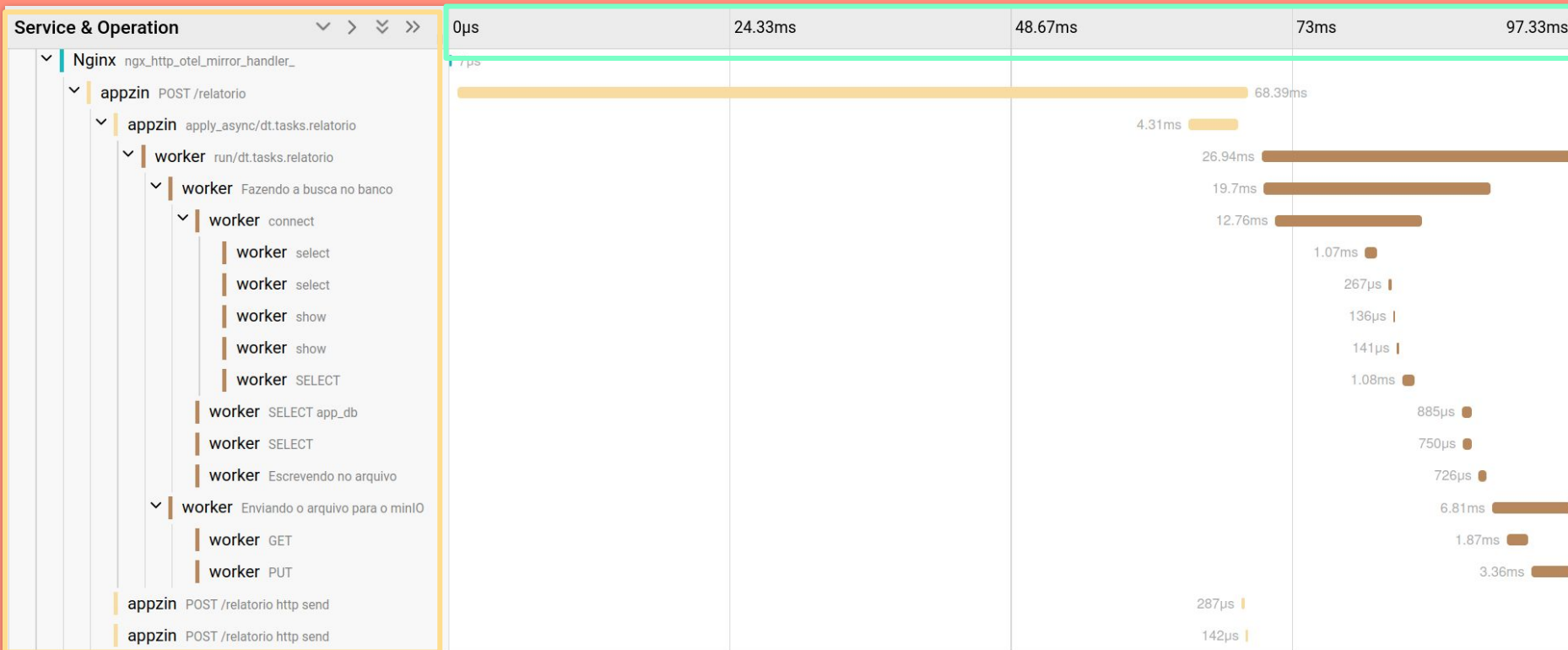
- Jaeger
- Grafana Tempo



# Exemplo de visualização



# Exemplo de visualização



Rastreamento  
distribuído e

OTel

# Conceitos importantes



Dentro do OTel temos alguns conceitos que precisamos entender para fazer tracing:

- Trace
- Span
- Context Propagation
- Events
- Sampling

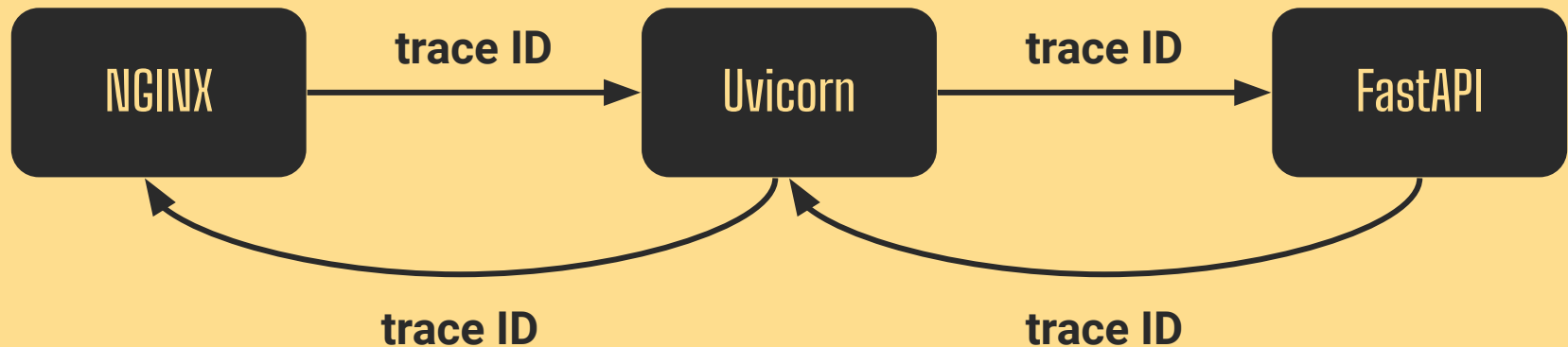


# Traces



Trace é o caminho de uma requisição observada pelo OTel.

Contém um **identificador único** para uma requisição. Esse identificador será passado entre as chamadas dos sistemas.



# Spans



Os Spans são os "**vãos**" em que o sistema fez uma determinada tarefa.

Spans contém:

- ID: Um identificador do vão
- Trace ID: Qual o rastro que originou o span
- Nome: Para identificarmos o vão como humanos
- Tempo de início e final: Quando começou e terminou um "vão"
- Atributos: Chaves e valores que gostaríamos de adicionar ao span

# Trace

```
graph TD; Trace[Trace] --> SPAN[SPAN]; SPAN --> ID[ID]; SPAN --> Nome[Nome]; SPAN --> Duração[Duração]; SPAN --> TraceID[Trace ID]; SPAN --> Atributos[Atributos];
```

SPAN

ID

Nome

Duração

Trace ID

Atributos

# Propagação de contexto



O contexto é **relação entre dois Spans**, que é propagado por meio de uma requisição http.

A W3C propõem a propagação por meio do campo *traceparent* no header da requisição.

<https://www.w3.org/TR/trace-context/#trace-context-http-headers-format>

```
{
  'host': 'app:8000',
  'traceparent': '00-aa2e3ad537bfe44abf88b27922d605c8-c86814fea24dc9d7-01',
  'tracestate': ''
}
```

```
{  
  'host': 'app:8000',  
  'traceparent': '00-aa2e3ad537bfe44abf88b27922d605c8-c86814fea24dc9d7-01',  
  'tracestate': ''  
}
```

# Trace

SPAN

Request (parentid)

SPAN



Serviço 1



Serviço 2



Distribuição

```
{
  'host': 'app:8000',
  'traceparent': '00-aa2e3ad537bfe44abf88b27922d605c8-c86814fea24dc9d7-01',
  'tracestate': ''
}
```

Trace ID

Trace

SPAN

Request (parentid)

SPAN

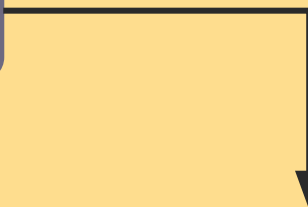
Span ID

Parent ID

Serviço 1

Serviço 2

Distribuição



# Eventos



Usamos um evento quando queremos notificar a telemetria um acontecimento significativo ao trace.

Em um momento específico do tempo, dentro de um span, algo relevante aconteceu.

- Uma requisição interna falhou
- Algo não aconteceu da maneira que deveria (uma exception suprimida)

Eventos também contém atributos, que podem adicionar mais valor a uma ocorrência.

```
with tracer.start_as_current_span(  
    'Iniciando a criação da task',  
    attributes=task.model_dump()  
) as span:
```

```
    with Session() as s:  
        _task = Task(**task.model_dump())  
        s.add(_task)  
        s.commit()  
        s.refresh(_task)
```

```
    span.add_event(  
        'Registro inserido no banco de dados',  
        attributes={'id': _task.id}  
    )
```

```
    return _task
```





Como grande parte das traces se repete em todos os serviços, se quisermos comparar as traces entre serviços e também manter um padrão de nomenclatura entre os rastros, precisamos de uma convenção.

## Trace Semantic Conventions

Status: [Mixed](#)

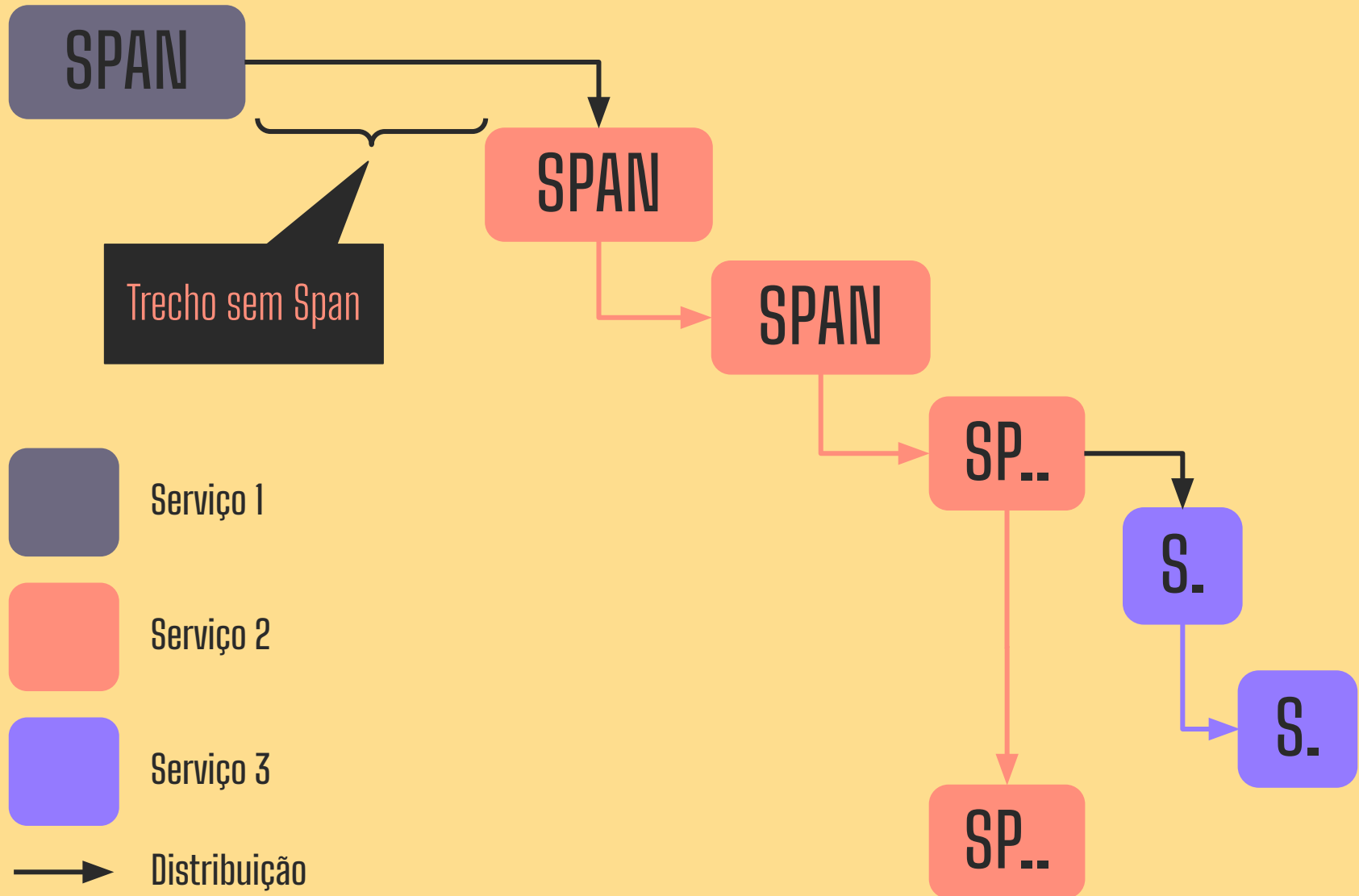
In OpenTelemetry spans can be created freely and it's up to the implementor to annotate them with attributes specific to the represented operation. Spans represent specific operations in and between systems. Some of these operations represent calls that use well-known protocols like HTTP or database calls. Depending on the protocol and the type of operation, additional information is needed to represent and analyze a span correctly in monitoring systems. It is also important to unify how this attribution is made in different languages. This way, the operator will not need to learn specifics of a language and telemetry collected from polyglot (multi-language) micro-service environments can still be easily correlated and cross-analyzed.

The following semantic conventions for spans are defined:

- **General:** General semantic attributes that may be used in describing different kinds of operations.
- [Compatibility](#): For spans generated by compatibility components, e.g. OpenTracing Shim layer.
- [CloudEvents](#): Semantic Conventions for the CloudEvents spans.
- [Cloud Providers](#): Semantic Conventions for cloud providers spans.
- [Database](#): For SQL and NoSQL client call spans.
- [Exceptions](#): For recording exceptions associated with a span.
- [FaaS](#): For [Function as a Service](#) (e.g., AWS Lambda) spans.
- [Feature Flags](#): For recording feature flag evaluations associated with a span.
- [HTTP](#): For HTTP client and server spans.
- [Messaging](#): For messaging systems (queues, publish/subscribe, etc.) spans.
- [Object Stores](#): Semantic Conventions for object stores spans.
- [RPC/RMI](#): For remote procedure call (e.g., gRPC) spans.

<https://opentelemetry.io/docs/specs/semconv/general/trace/>

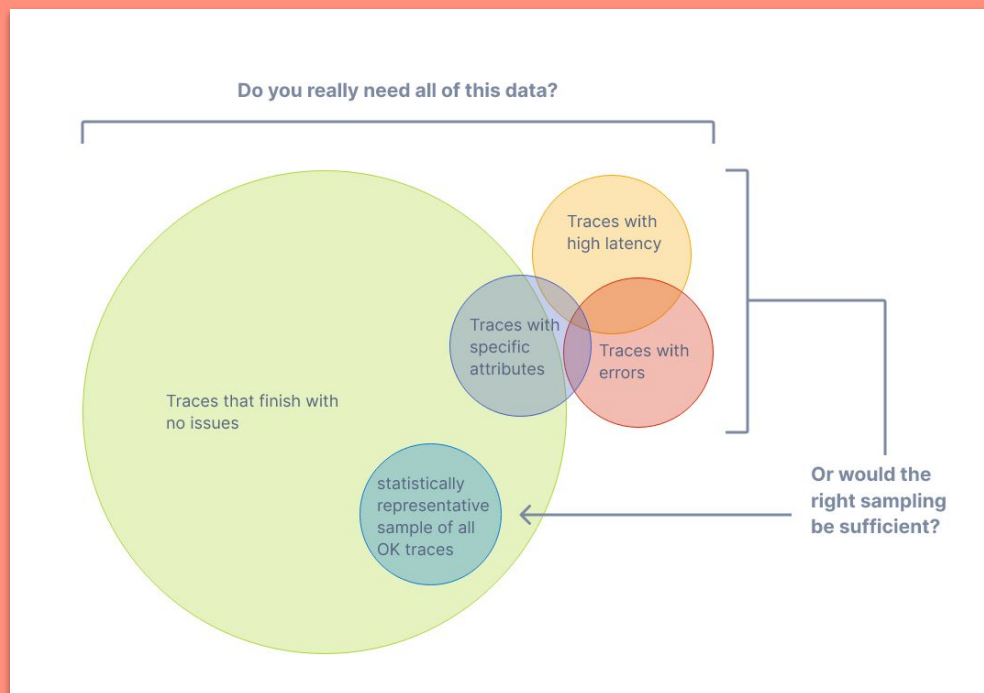
# Trace



# Sampling



A tradução para sampling é **amostragem**. Geralmente queremos guardar traces, mas o volume de requisição pode ser extremamente alto. Então, precisamos fazer isso de forma saudável.

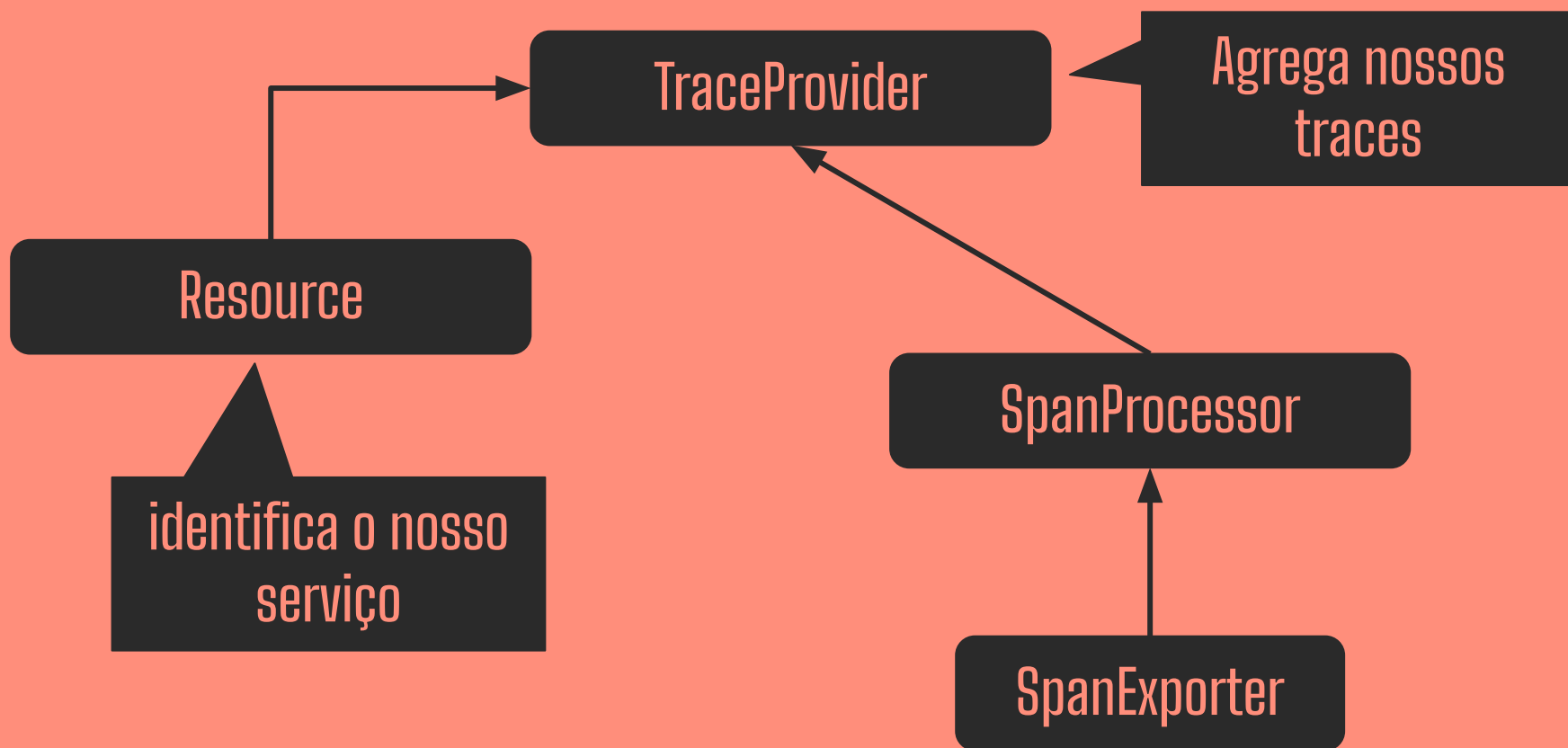


<https://opentelemetry.io/docs/concepts/sampling/>

Instrumentação

Manual

# Objetos relacionados o Trace



# Boilerplate



```
1  from opentelemetry.trace import get_tracer, set_tracer_provider
2  from opentelemetry.sdk.resources import Resource, SERVICE_NAME
3  from opentelemetry.sdk.trace import TracerProvider
4  from opentelemetry.sdk.trace.export import BatchSpanProcessor
5  from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import (
6      OTLPSpanExporter
7  )
8
9  resource = Resource({SERVICE_NAME: 'blah'})
10 provider = TracerProvider(resource=resource)
11 processor = BatchSpanProcessor(
12     OTLPSpanExporter(endpoint='http://localhost:4317', insecure=True)
13 )
14 provider.add_span_processor(processor)
15 set_tracer_provider(provider)
16 tracer = get_tracer(__name__)
```

# Exemplo de uso do trace



```
1  @tracer.start_as_current_span('fazendo algo')
2  def teste_spam():
3      return 42
4
5
6  with tracer.start_as_current_span('test'):
7      teste_spam()
```

```
{  
  "name": "test",  
  "context": {  
    "trace_id": "0xd679d965422fd3a9210830c5c28a0f9b",  
    "span_id": "0x894c33993a0caded",  
    "trace_state": ""  
  },  
  "kind": "SpanKind.INTERNAL",  
  "parent_id": null,  
  "start_time": "2024-04-08T00:26:28.312675Z",  
  "end_time": "2024-04-08T00:26:28.312740Z",  
  "status": {  
    "status_code": "UNSET"  
  },  
  "attributes": {},  
  "events": [],  
  "links": [],  
  "resource": {  
    "attributes": {  
      "service.name": "blah"  
    },  
    "schema_url": ""  
  }  
}
```




```
{
  "name": "test",
  "context": {
    "trace_id": "0xd679d965422fd3a9210830c5c28a0f9b",
    "span_id": "0x894c33993a0caded",
    "trace_state": "[]"
  },
  "kind": "SpanKind.INTERNAL",
  "parent_id": null,
  "start_time": "2024-04-08T00:26:28.312675Z",
  "end_time": "2024-04-08T00:26:28.312740Z",
  "status": {
    "status_code": "UNSET"
  },
  "attributes": {
    "key": "value"
  },
  "events": [],
  "links": [],
  "resource": {
    "attributes": {
      "key": "value"
    },
    "schema_url": "https://opentelemetry.io/schemas/1.11.0"
  },
}
```

```
1 @tracer.start_as_current_span('fazendo algo')
2 def teste_spam():
3     return 42
4
5
6 with tracer.start_as_current_span('test'):
7     teste_spam()
```

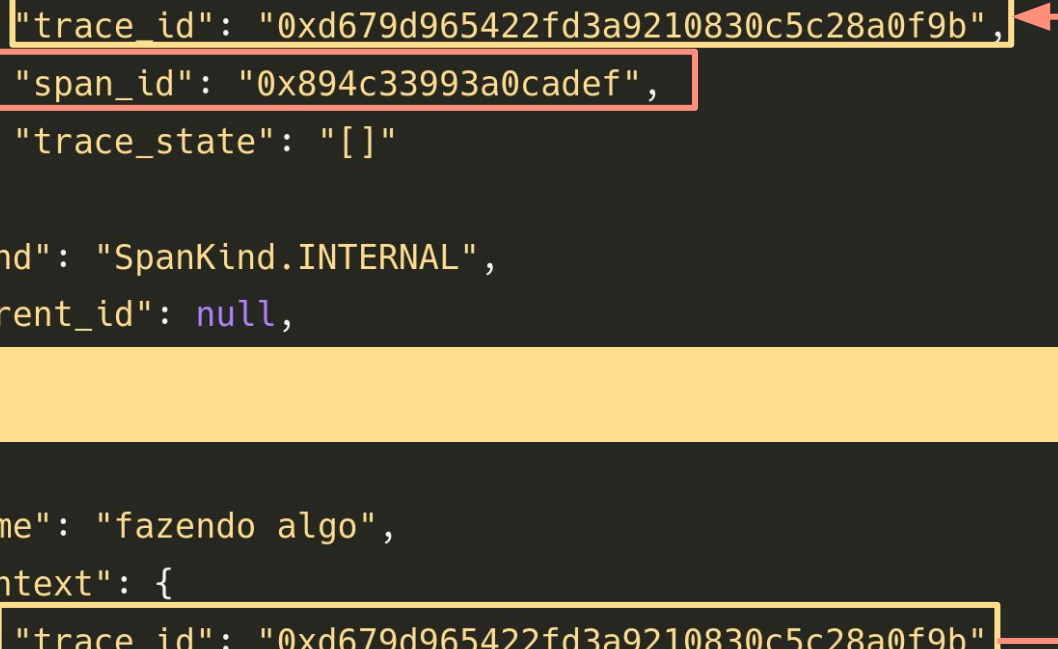
```
{
  "name": "fazendo algo",
  "context": {
    "trace_id": "0xd679d965422fd3a9210830c5c28a0f9b",
    "span_id": "0xc204091c77e34d60",
    "trace_state": "[]"
  },
  "kind": "SpanKind.INTERNAL",
  "parent_id": "0x894c33993a0caded",
  "start_time": "2024-04-08T00:26:28.312716Z",
  "end_time": "2024-04-08T00:26:28.312726Z",
  "status": {
    "status_code": "UNSET"
  },
  "attributes": {},
  "events": [],
  "links": [],
  "resource": {
    "attributes": {
      "service":
    },
    "schema_url":
  }
}
```

```
1 @tracer.start_as_current_span('fazendo algo')
2 def teste_spam():
3     return 42
4
5
6 with tracer.start_as_current_span('test'):
7     teste_spam()
```

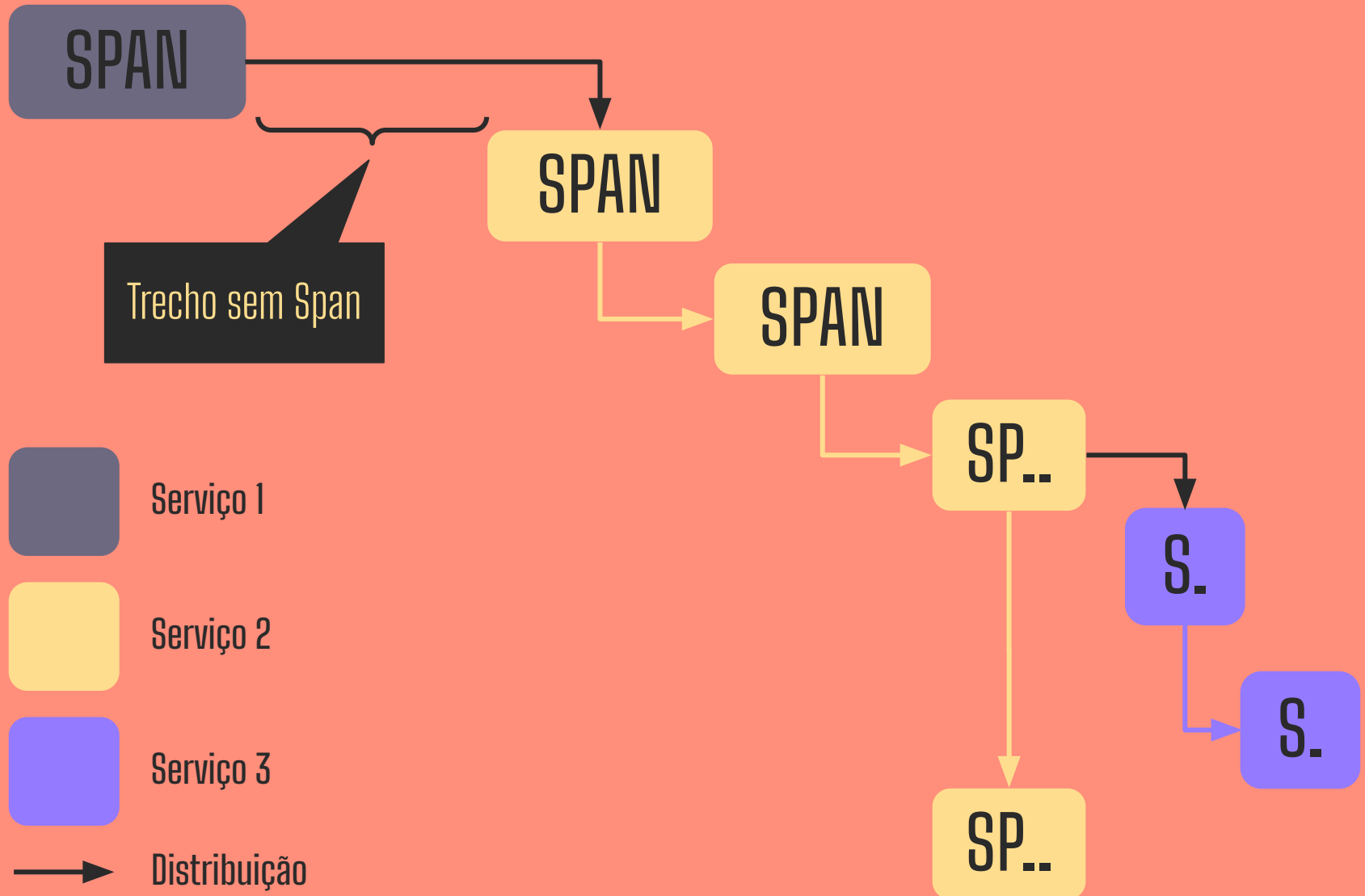
```
{
  "name": "test",
  "context": {
    "trace_id": "0xd679d965422fd3a9210830c5c28a0f9b",
    "span_id": "0x894c33993a0caded",
    "trace_state": "[]"
  },
  "kind": "SpanKind.INTERNAL",
  "parent_id": null,
```



```
{
  "name": "fazendo algo",
  "context": {
    "trace_id": "0xd679d965422fd3a9210830c5c28a0f9b",
    "span_id": "0xc204091c77e34d60",
    "trace_state": "[]"
  },
  "kind": "SpanKind.INTERNAL",
  "parent_id": "0x894c33993a0caded",
```



# Trace





Bora subir as coisas!



Take 1



Autom  
ática

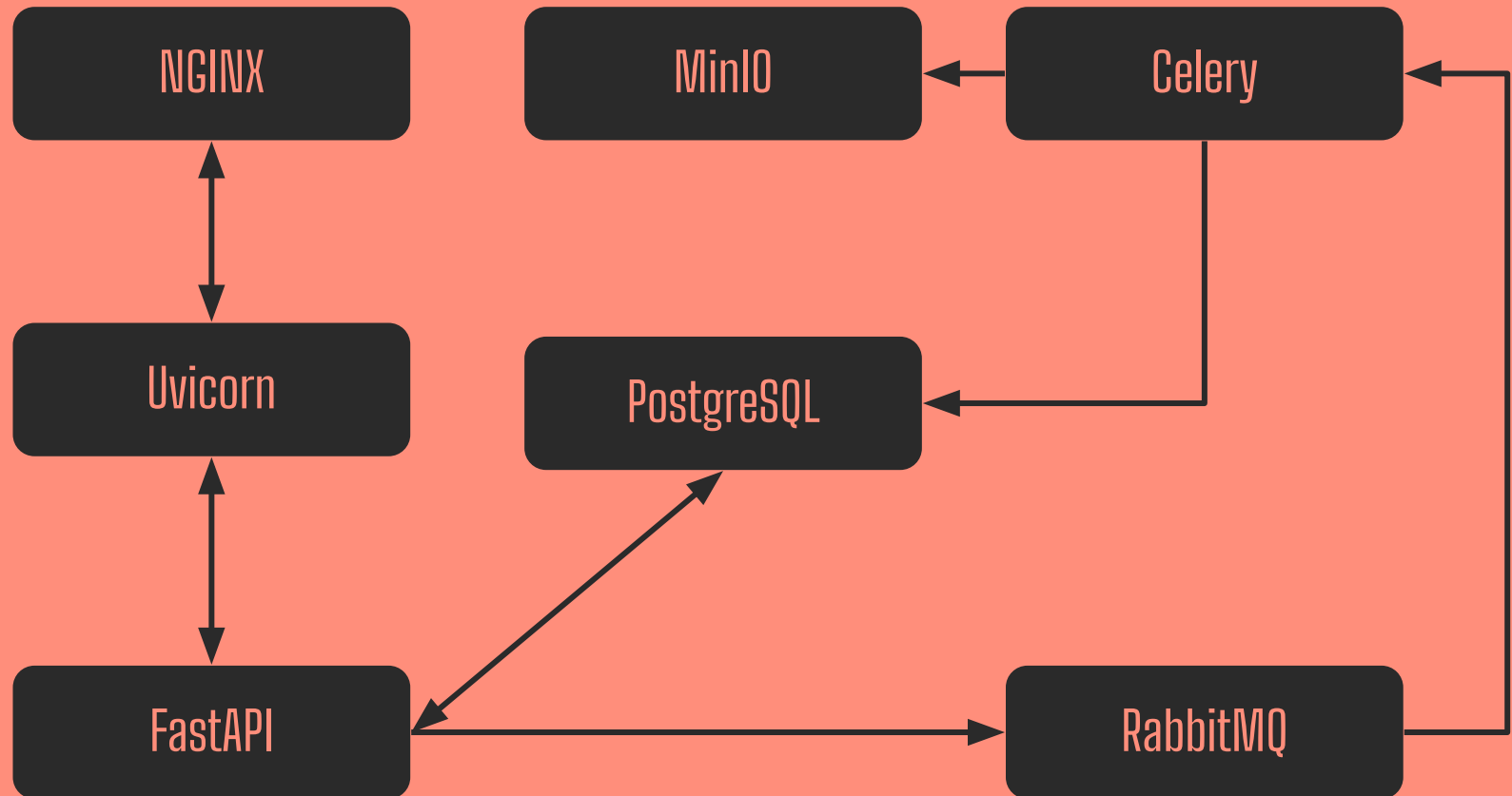
Instrumentação

# A configuração



```
— □ ×  
  
RUN pip install opentelemetry-distro  
RUN pip install opentelemetry-exporter-otlp  
  
RUN opentelemetry-bootstrap -a install  
  
ENV OTEL_SERVICE_NAME=appzin  
ENV OTEL_TRACES_EXPORTER=otlp  
ENV OTEL_EXPORTER_OTLP_INSECURE=true  
ENV OTEL_EXPORTER_OTLP_ENDPOINT=otel:4317  
  
CMD opentelemetry-instrument \  
    uvicorn dt.app:app \  
    --host 0.0.0.0
```

# O demo de hoje!







[apoia.se/livedepython](https://apoia.se/livedepython)



[pix.dunossauro@gmail.com](mailto:pix.dunossauro@gmail.com)



[patreon.com/dunossauro](https://patreon.com/dunossauro)



Ajude o projeto <3



# Referências



- <https://dictionary.cambridge.org/dictionary/english/tracin>
- <https://dictionary.cambridge.org/dictionary/english/trace>
- <https://docs.python.org/3/library/trace.html>
- <https://www.jaegertracing.io/>
- <https://grafana.com/docs/tempo/latest/>
- <https://docs.celeryq.dev/en/stable/userguide/tasks.html>
- <https://www.rabbitmq.com/docs>
- <https://opentelemetry.io/docs/concepts/signals/traces/>
- <https://opentelemetry.io/docs/specs/semconv/general/trace/>
- <https://opentelemetry.io/docs/concepts/context-propagation/>
- <https://opentelemetry.io/docs/specs/semconv/general/metrics/>
- <https://opentelemetry.io/docs/concepts/sampling/>
- <https://opentelemetry.io/blog/2022/instrument-nginx/>
- <https://www.w3.org/TR/trace-context/#traceparent-header>
- <https://www.w3.org/TR/trace-context/>