Métricas de observabilidade

Live de Python #263

Título	Data
Observabilidade - Uma introdução *[Link na descrição]*	11/03
Observabilidade - Métricas (Opentelemetry / Prometheus)	**Agora!**
Observabilidade - Rastreamento / Tracing (Opentelemetry / Tempo / Jeager)	08/04
Observabilidade - Logs (Opentelemetry / Loki)	15/04

Isso é uma série!

Roteiro

1. Introdução a métricas

Um pouco dos conceitos por trás

2. OpenTelemetry e Metricas

Os tipos de métricas suportadas pelo OTel

3. Instrumentação manual

Como instrumentar métricas na mão?

4. Instrumentação automática

O que o OTel já tem pronto para python

apoia.se/livedepython

pix.dunossauro@gmail.com

patreon.com/dunossauro

Ajude o projeto <3

Ademar Peixoto, Adilson Herculano, Adriano Casimiro, Alemao, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alfredo Braga, Alisson Souza, Alysson Oliveira, Andre Paula, Antônio Filho, Apc 16, Aslay Clevisson, Aurelio Costa, Bárbara Grillo, Ber_dev_2099, Bernarducs, Bruno Almeida, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Ramos, Bruno Santos, Caio Nascimento, Carlos Ramos, Clara Battesini, Cristian Firmino, Daniel Bianchi, Daniel Freitas, Daniel Real, Daniel Wojcickoski, Danilo Boas, Danilo Silva, David Couto, David Kwast, Davi Souza, Dead Milkman, Denis Bernardo, Dgeison Peixoto, Diego Guimarães, Dino, Edgar, Edilson Ribeiro, Eduard0ml, Elias Moura, Emerson Rafael, Ennio Ferreira, Erick Andrade, Érico Andrei, Everton Silva, Fabio Barros, Fábio Barros, Fabio Valente, Fabricio Biazzotto, Felipe Augusto, Felipe Rodrigues, Fernanda Prado, Francisco Silvério, Frederico Damian, Gabriel Espindola, Gabriel Mizuno, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Giovanna Teodoro, Giuliano Silva, 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, Joseíto 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, Lengo, 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 Eduardo, Luiz Carlos, Luiz Duarte, Luiz Lima, Luiz Martins, Luiz Paula, Mackilem Laan, Marcelo Araujo, Marcio Silva, Marco Mello, Marcos Almeida, Marcos Gomes, Marcos Oliveira, Marina Passos, Mateus Lisboa, Matheus Silva, Matheus Vian, Mírian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Pedro Henrique, Pedro Pereira, Peterson Santos, Philipe Vasconcellos, Pytonyc, Rafael, Rafael Araújo, Rafael Faccio, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Renato José, Renato Moraes, Rene Pessoto, Renne Rocha, Ricardo Combat, Ricardo Silva, Rinaldo Magalhaes, Riverfount, Rjribeiro, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Santana, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rui Jr, Samanta Cicilia, Santhiago Cristiano, Selmison Miranda, Téo Calvo, Thiago Araujo, Thiago Borges, Thiago S, Tiago Ferreira, Tiago Henrique, Tony Dias, Tyrone Damasceno, Valdir, Valdir Tegon, Varlei Menconi, Vinicius Bego, Vinicius Silva, Vinicius Stein, Vladimir Lemos, Wagner Gabriel, Washington Teixeira, Willian Lopes, Willik Araujo, Wilson Duarte, Zeca Figueiredo

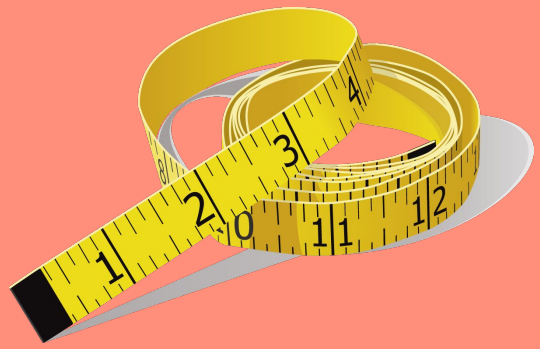
Obrigado você

O que são? Do que se alimentam?

Métricas!

O que são métricas?

Métricas são *basicamente* **números coletados da sua aplicação em tempo de execução** para responder perguntas sobre o que está acontecendo durante a execução



O que são métricas?

Métricas são *basicamente* **números coletados da sua aplicação em tempo de execução** para responder perguntas sobre o que está acontecendo durante a execução. Exemplos:

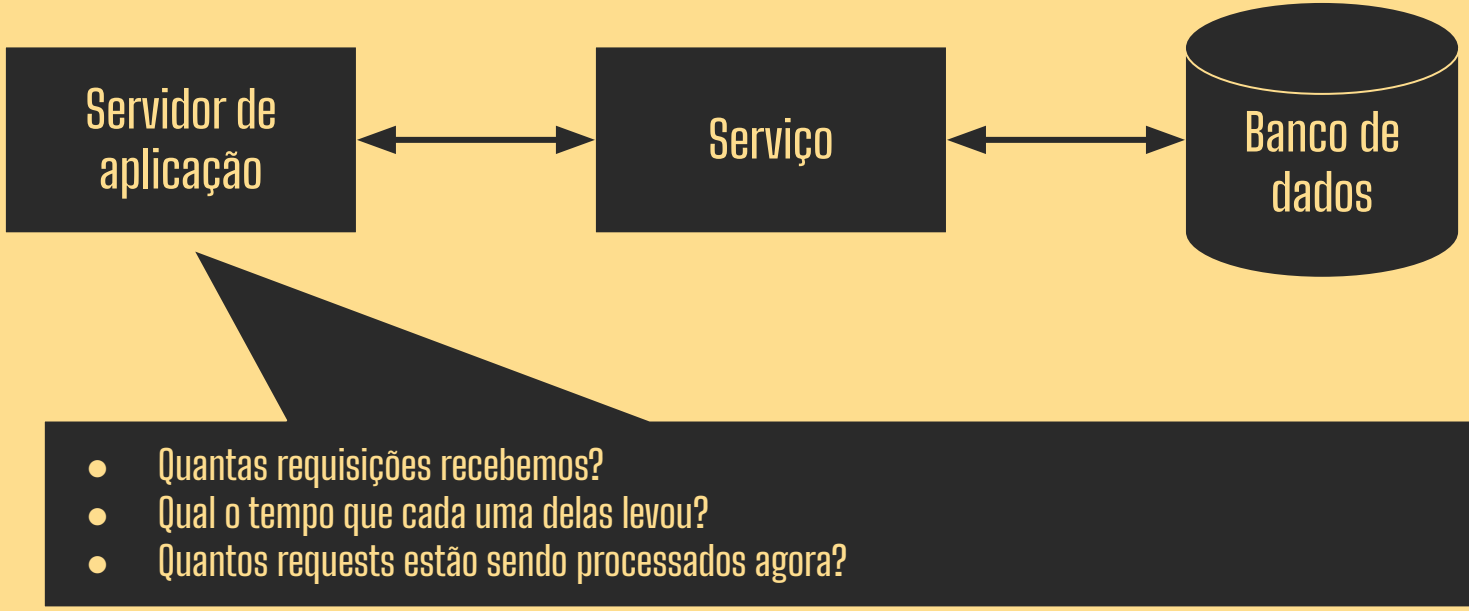
- Quantos requests recebemos?
- Quantas vezes algo foi chamado?
- Quantos deploys foram feitos?
- Quantos erros nossa aplicação gerou?
- Quantas chamadas externas fizemos?
- Quanto tempo uma chamada gerou?
- Qual a média de requisições por hora?
- Quanto a aplicação está consumindo de memória?
- Quanto a aplicação está consumindo de CPU?
- Qual o meu pior tempo de resposta?

Telemetria e métricas

A ideia é adicionar "sensores" em componentes da aplicação para entender o comportamento com números.

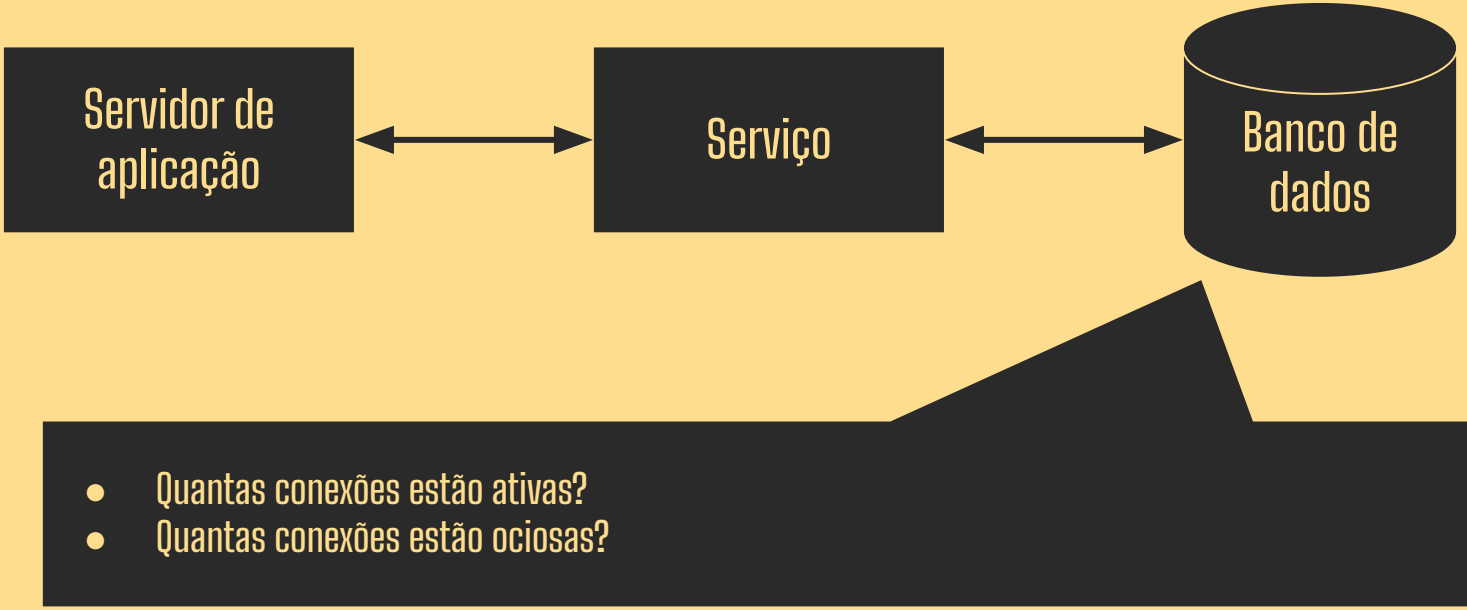
Telemetria e métricas

A ideia é adicionar "sensores" em componentes da aplicação para entender o comportamento com números.



Telemetria e métricas

A ideia é adicionar "sensores" em componentes da aplicação para entender o comportamento com números.



Telemetria e métricas

A ideia é adicionar "sensores" em componentes da aplicação para entender o comportamento com números.

- Quanto de Memória / CPU / Disco / Swap o serviço está consumindo?
- Quantas threads estão ativas?
- Quantos erros o serviço teve?
- Quantas requisições foram feitas para fontes externas?
- Medidas te tempo internas da aplicação ...

Métricas são muito mais do que números sobre "metal"

Embora esse seja o sentimento padrão

Implantação!

Existem vários outros casos importantes de mensurar. Se olharmos para o mundo de operações, temos coisas importantes que se relacionaram:

- Quantos deploy fizemos em uma janela de tempo?
- Quantas vezes a configuração do pod/vm foi alterada?
- Quais são as medidas de tempo de um deploy?
- Ocorreram erros na implantação? Quantos?
- Se a escala for elástica, quantas vezes foi *esticada* ou *contraída*?
- As FaaS foram ativadas? Quantas vezes? Quais delas?

Sistemas complementares

Nem só de serviço vive um produto. Imagine:

- ETL
- Sistemas des raspagem
- Automações
- Tarefas em segundo plano em filas

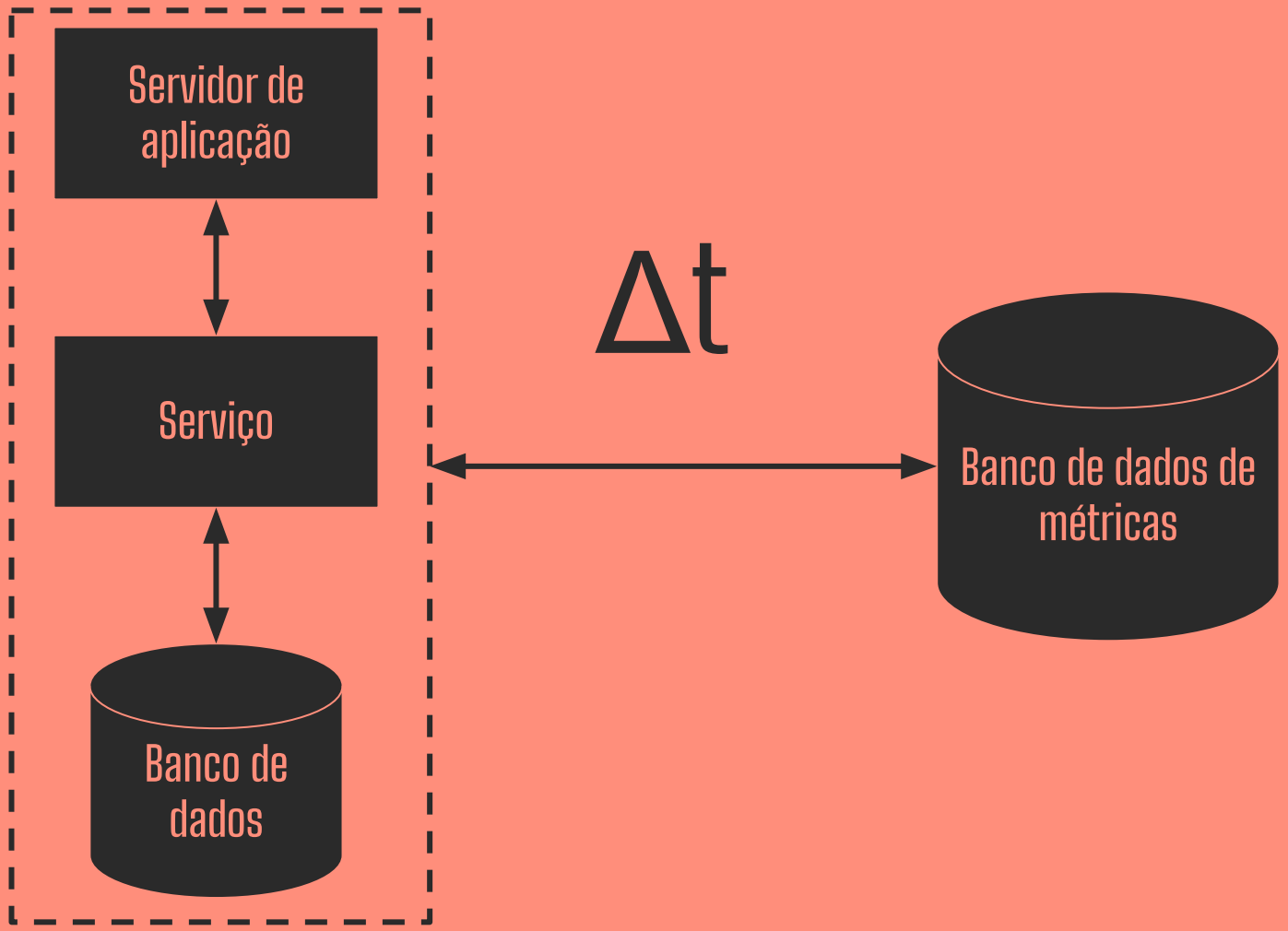
Tudo isso pode ser notificado!

Regras de negócio

Problemáticas de negócio também podem ser medidas:

- Quantas vendas fizemos em uma janela de tempo?
- Quantas pessoas foram até o checkout e desistiram?
- Quantas pessoas passaram pela aplicação?
- Qual dos clientes usa mais a aplicação?
- Qual o tempo médio de uso por cliente?
- ...

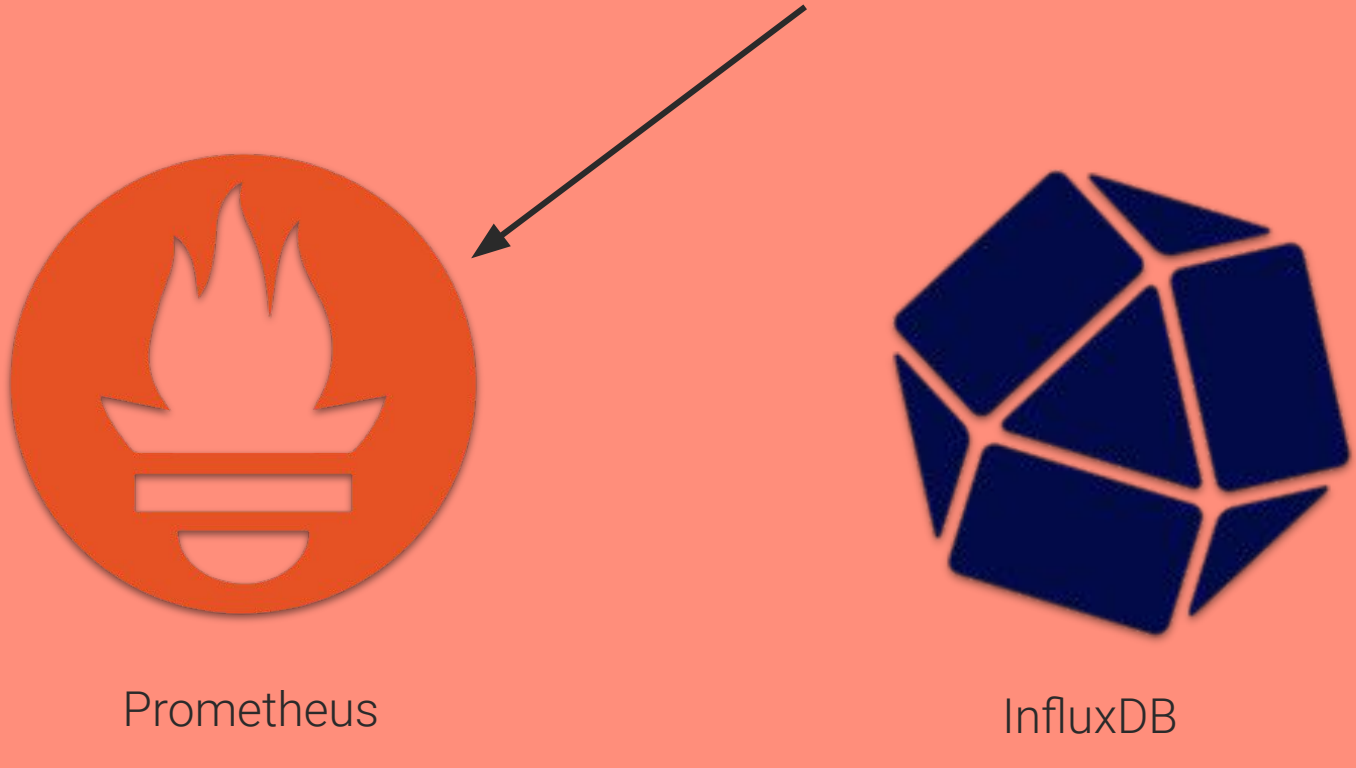
Blz, gostei. Como funciona?



Uma visão ingênua do armazenamento

Métrica	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ7	Δ8	Δn
http.server.active_requests	...	2	1	0	7	3	1	7	9	...
http.server.requests	1	6	7	13	17	21	30	32	38	...
db.client.connections.usage	1	10	13	7	8	3	0	13	17	...
process.memory.usage	3	25	78	32	17	15	12	18	96	...
faas.errors	...	...	...	...	...	...	2	5	...	...

Soluções para armazenamento de métricas

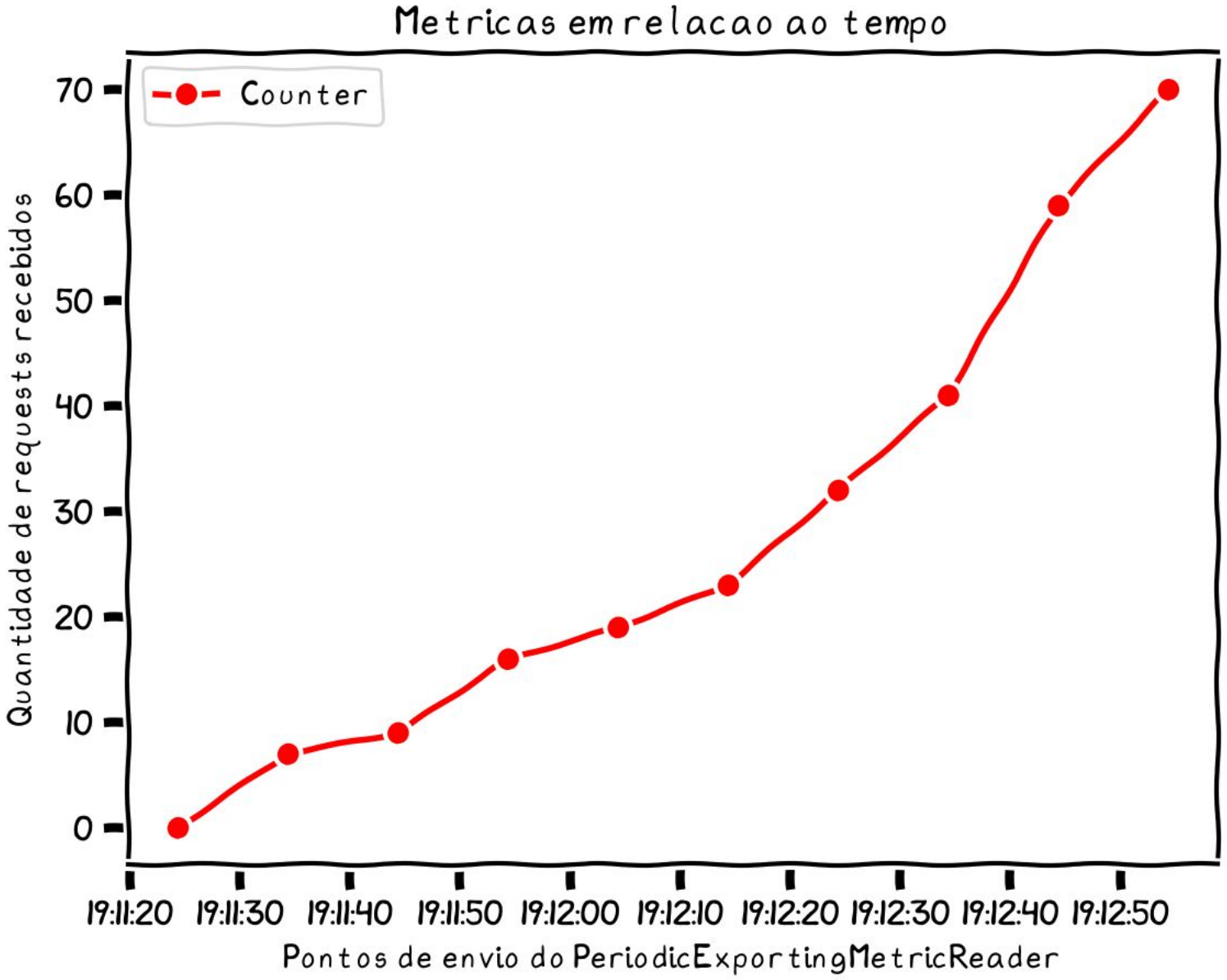


Prometheus

InfluxDB

Outra visão ingênua do armazenamento

Métrica	Atributos e valores	Δ1	Δ2	Δ3
http_server_requests	{job="app1", method="GET", route="/docs"}	...	2	1
http.server.requests	{job="app1", method="POST", route="/create"}	1	6	7
http.server.requests	{job="app1", method="GET", route="/user/{id}"}	1	10	13
http.server.requests	{job="app2", method="GET", route="/docs"}	3	25	78
http.server.requests	{job="app2", method="GET", route="/view_tasks"}	7	3	1


Metricas em relacao ao tempo
Counter
Quantidade de requests recebidos
0
10
20
30
40
50
60
70
19:11:20
19:11:30
19:11:40
19:11:50
19:12:00
19:12:10
19:12:20
19:12:30
19:12:40
19:12:50
Pontos de envio do PeriodicExportingMetricReader

Especificações de métricas!

Open telemetry

Métricas sob a ótica do OTel

As métricas do OTel seguem quatro regras (tem mais, mas pra nós isso é o que é necessário):

- **Nome**: O nome da métrica
- **Unidade de medida** (unit): Como a métrica é mensurada
  - Bytes? Int? Float? Segundos? Milissegundos?
- **Descrição**: Uma explicação sobre a métrica (um texto arbitrário)
- **Tipo** (kind): O que delimita como a métrica será coletada e armazenada

```
counter = meter.create_counter(
    'contador',
    description='Contagem de coisas!',
    unit='????',
)
```

Tipos de métricas

A especificação do OTel suporta até o momento 7 tipos de métricas divididas em duas categorias:

- **Síncronas**:
  - Counter
  - Histograma
  - Gauge
  - UpDownCounter
- **Assíncronas**:
  - Counter
  - Gauge
  - UpDownCounter

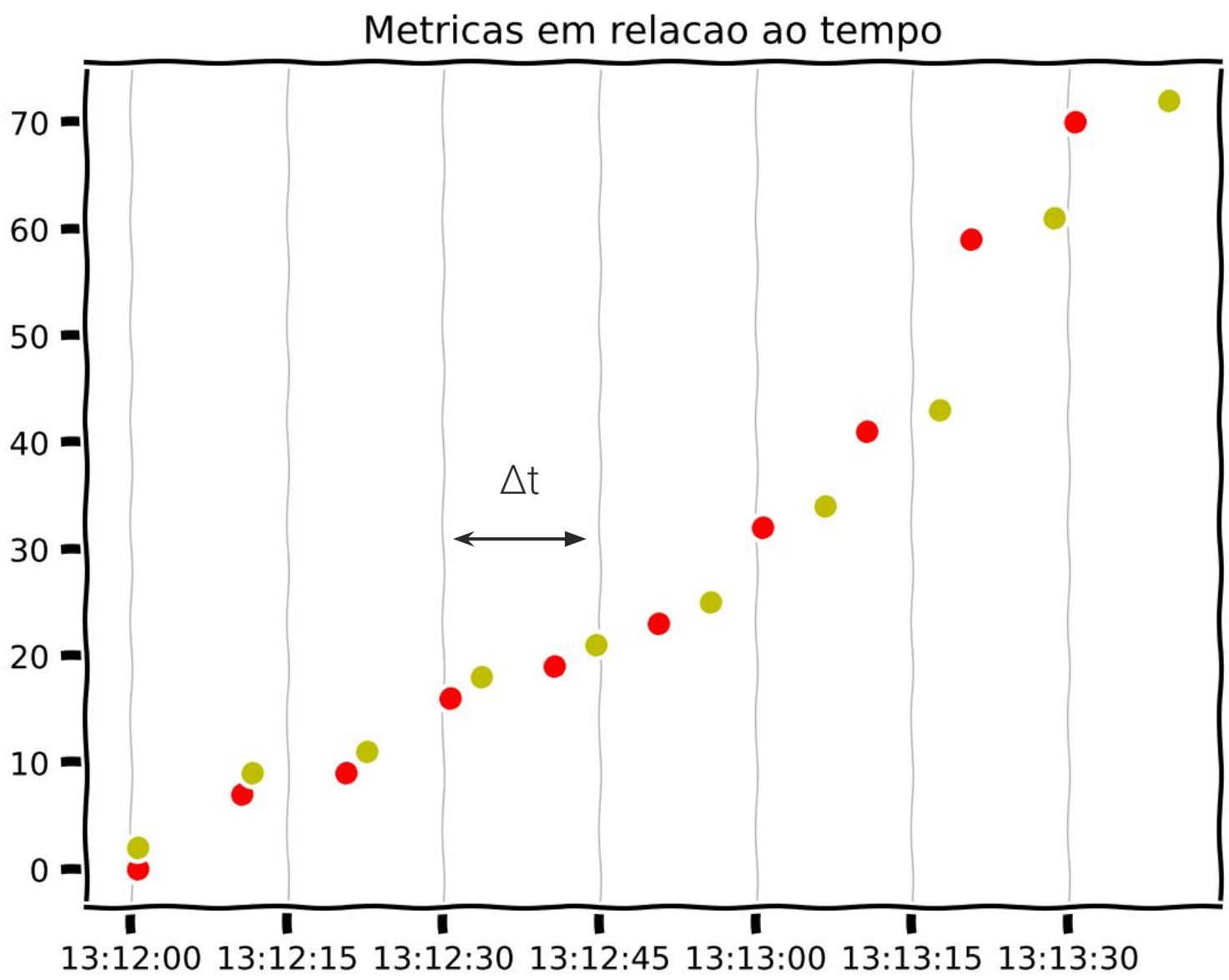
Métricas síncronas

As métricas síncronas são manipuladas pelo código de forma direta. Podemos alterar seu valor durante a execução do código.

```python
counter = meter.create_counter(
    'contador',
    description='Contagem de coisas!',
    unit='????',
)

counter.add(
    amount=1,
  attributes={'atributo': 'valor'}
)

counter.add(1, {'atributo': 'valor'})
```


Metricas em relacao ao tempo
70
60
50
40
30
20
10
0
Δt
13:12:00
13:12:15
13:12:30
13:12:45
13:13:00
13:13:15
13:13:30

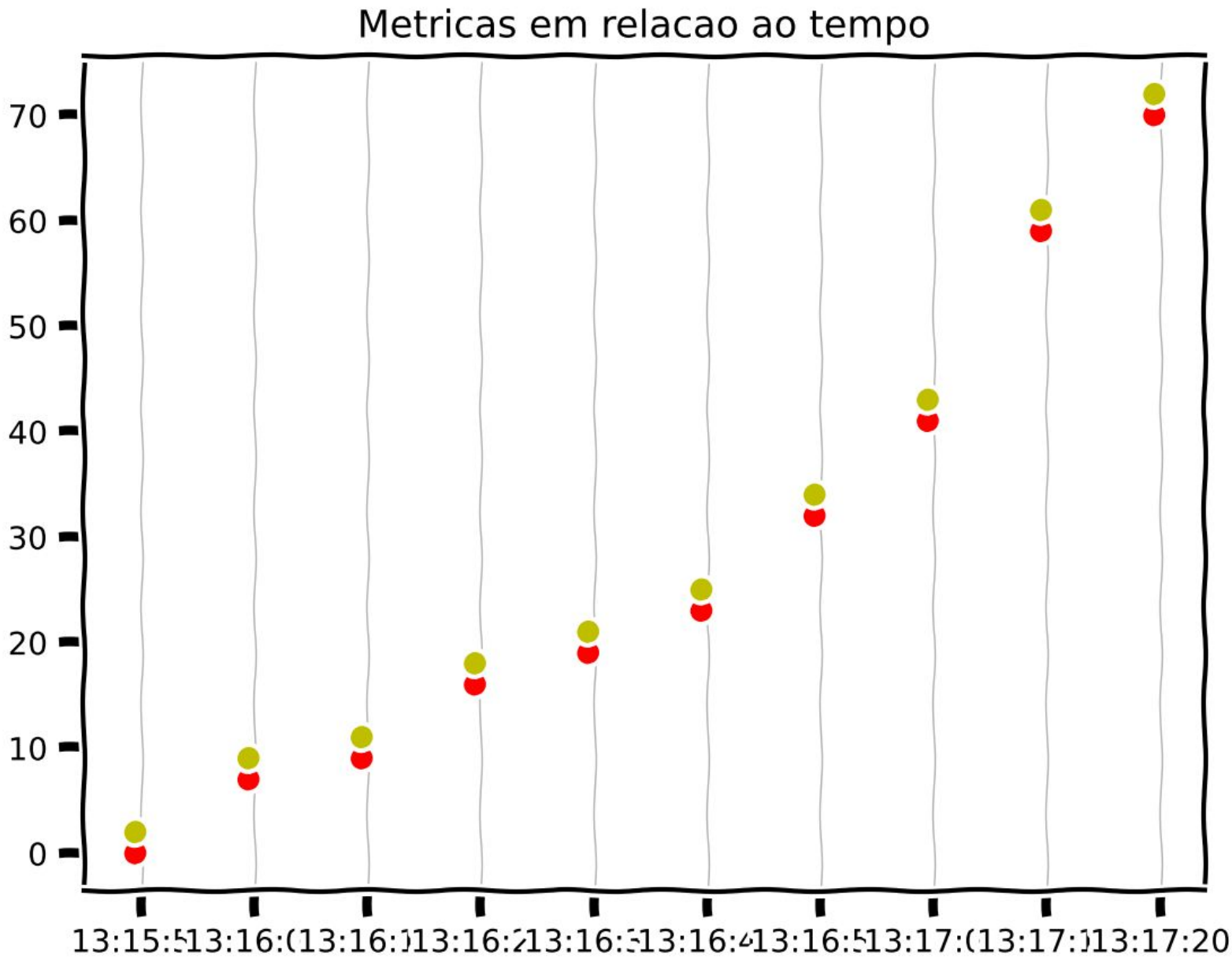
Métricas assíncronas

Definidas por meio de callbacks que serão chamados na hora da coleta temporal da métrica (Δ). A medida só muda uma vez por coleta.

```python
from opentelemetry.metrics import Observation

def callback(observer):
    # coleta algo
    yield Observation(1, {'dado': 'Alguma coisa!'})

async_counter = meter.create_observable_counter(
    'contador asyncrono',
    description='Contagem de coisas assincronamente!',
    unit='???',
    callbacks=[callback]
)
```


Metricas em relacao ao tempo
70
60
50
40
30
20
10
0
13:15:5
13:16:(
13:16:(
13:16:2
13:16:3
13:16:4
13:16:5
13:17:(
13:17:(
13:17:20

Métricas síncronas

- **Counter**: Contador monotônico, poder ser somente acrescentado
  - Requisições efetuadas
  - Requisições recebidas
  - Quantidade de bytes recebidos
  - Número de contas criadas
  - Número de pontos de verificação executados
  - Quantidade de erros HTTP 5xx
- **Histograma**: Valores que podem ser estatisticamente interessantes
- **Gauge**: Valores não aditivos
- **UpDownCounter**: Contador decrementável

Métricas síncronas

- **Counter**: Contador monotônico, poder ser somente acrescentado
- **Histograma**: Valores que podem ser estatisticamente interessantes
  - Quanto tempo o banco demorou pra fazer uma busca?
  - Quanto tempo uma requisição levou?
  - Quanto tempo uma ação específica demorou?
- **Gauge**: Valores não aditivos
- **UpDownCounter**: Contador decrementável

Métricas síncronas

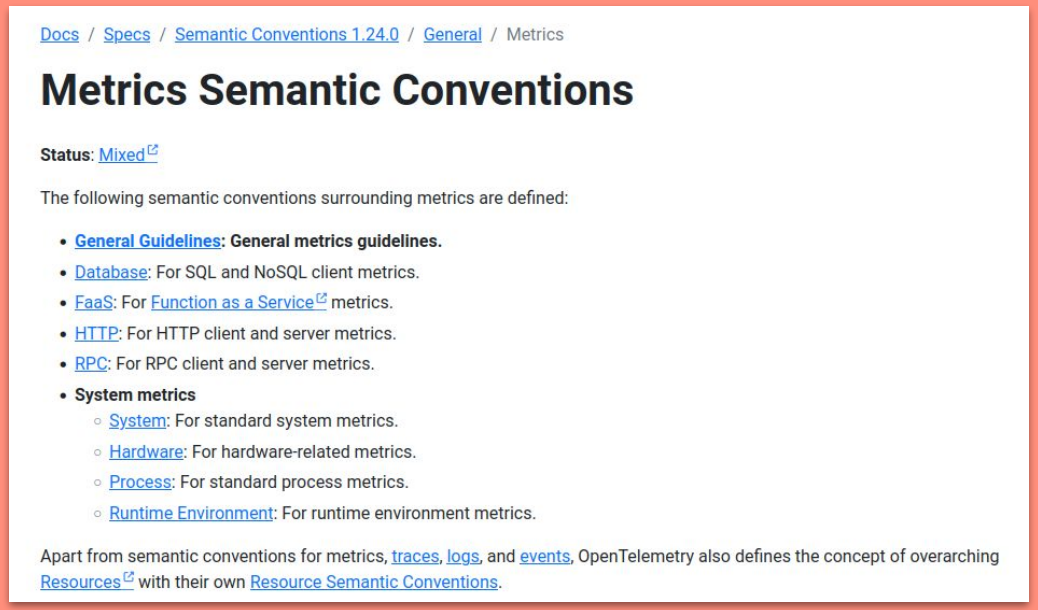
- **Counter**: Contador monotônico, poder ser somente acrescentado
- **Histograma**: Valores que podem ser estatisticamente interessantes
- **Gauge**: Valores não aditivos
  - Quanto de cpu está em uso agora?
  - Quanto de memória está em uso?
  - Quanto a aplicação está usando de memória?
- **UpDownCounter**: Contador decrementável

Métricas síncronas

- **Counter**: Contador monotônico, poder ser somente acrescentado
- **Histograma**: Valores que podem ser estatisticamente interessantes
- **Gauge**: Valores não aditivos
- **UpDownCounter**: Contador decrementável
  - Quantas pessoas estão logadas agora?
  - Quantos requests estão sendo processados?
  - Quantas conexões estão abertas com o banco de dados?
  - Quantos jobs estão na fila?

Convenções semânticas

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



Docs / Specs / Semantic Conventions 1.24.0 / General / Metrics

Metrics Semantic Conventions

Status: Mixed

The following semantic conventions surrounding metrics are defined:

- **General Guidelines: General metrics guidelines.**
- Database: For SQL and NoSQL client metrics.
- FaaS: For Function as a Service metrics.
- HTTP: For HTTP client and server metrics.
- RPC: For RPC client and server metrics.
- **System metrics**
  - System: For standard system metrics.
  - Hardware: For hardware-related metrics.
  - Process: For standard process metrics.
  - Runtime Environment: For runtime environment metrics.

Apart from semantic conventions for metrics, traces, logs, and events, OpenTelemetry also defines the concept of overarching Resources with their own Resource Semantic Conventions.

https://opentelemetry.io/docs/specs/semconv/general/metrics/

Instrumentação

Manual

Instrumentação manual

A instrumentação é a parte de fornecer os insumos no nosso código para que o opentelemetry consiga interagir com a nossa aplicação.

Para isso precisamos de 4 componentes da biblioteca do OTel:

- Resource: Explica para o OTel o que é nosso serviço
- Provider: Provedor de métricas (de onde criaremos as métricas)
- Reader: Um leitor periódico das métricas
- Exporter: Um exportador das métricas lidas

pip install

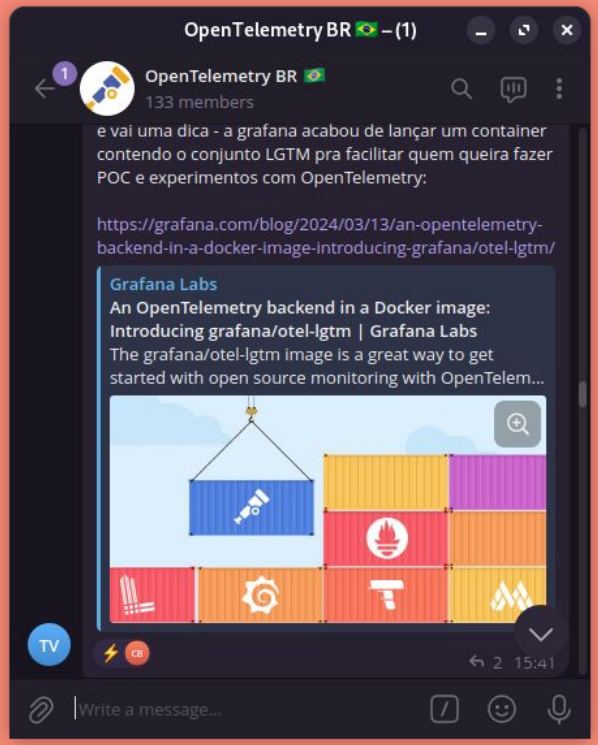
opentelemetry-distro

opentelemetry-exporter-otlp

Bora instalar as coisas

Simplificando a vida

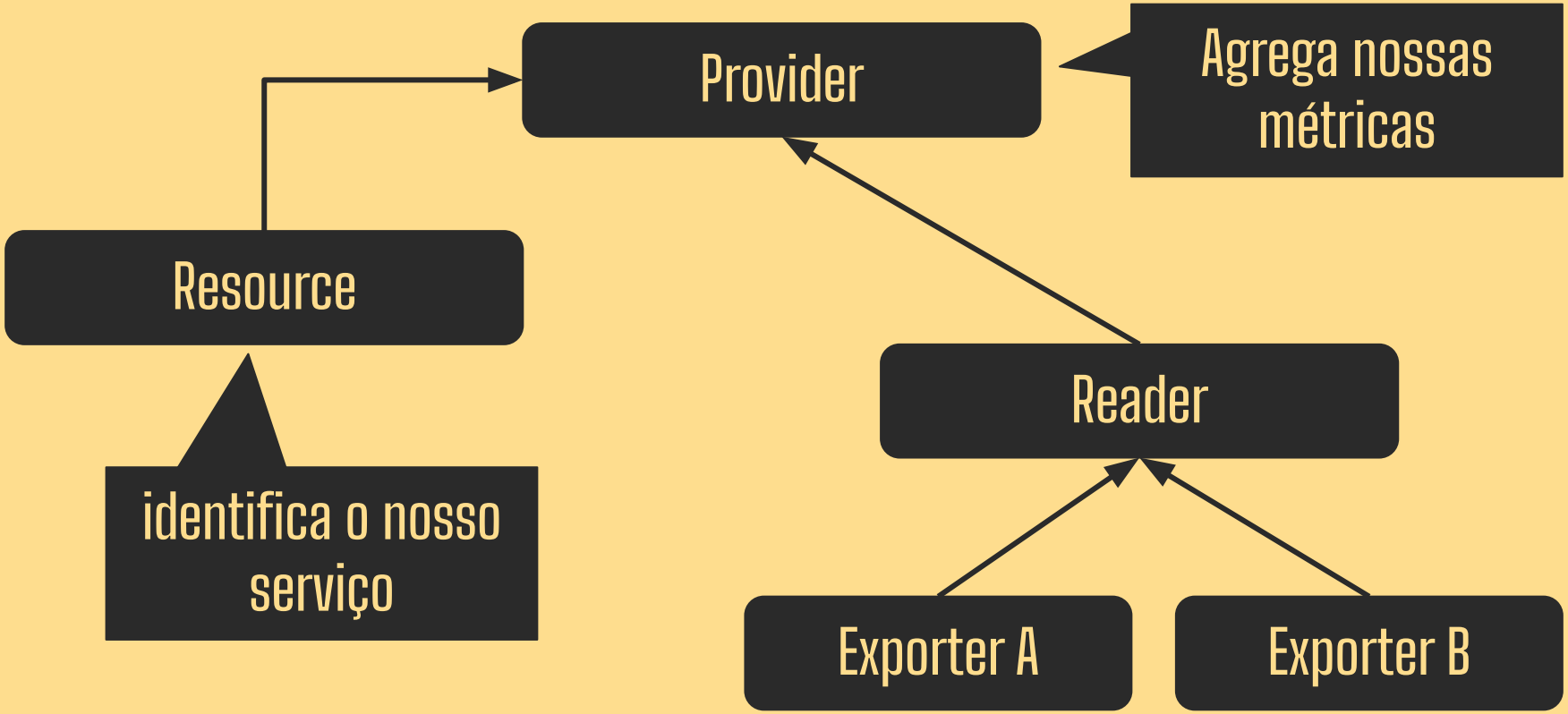
Da última live pra cá, a grafana lançou uma imagem com todas as ferramentas juntas. Collector, Tempo, Loki, Prometheus e Grafana (grafana/otel-lgtm)



https://grafana.com/blog/2024/03/13/an-opentelemetry-backend-in-a-docker-image-introducing-grafana/otel-lgtm/

grafana/otel-lgtm

```yaml
# compose.yaml
services:
  olgtm:
    image: grafana/otel-lgtm
    ports:
      - 3000:3000
      - 9090:9090
      - 4318:4318
      - 4317:4317
```


Provider
Agrega nossas métricas
Resource
identifica o nosso serviço
Reader
Exporter A
Exporter B

```python
from opentelemetry.sdk.resources import SERVICE_NAME, SERVICE_VERSION, Resource

resource = Resource(
    attributes={SERVICE_NAME: 'meu-app', SERVICE_VERSION: '1.0.0'}
)
```

```python
from opentelemetry.metrics import get_meter
from opentelemetry.sdk.metrics import MeterProvider

provider = MeterProvider(resource=resource)
meter = get_meter('meter', meter_provider=provider)

metrica_a = meter.create_counter('A')
metrica_b = meter.create_up_down_counter('B')
metrica_c = meter.create_gauge('C')
metrica_d = meter.create_histogram('D')
```

```python
from opentelemetry.sdk.metrics.export import (
    PeriodicExportingMetricReader,
    ConsoleMetricExporter,
)
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import (
    OTLPMetricExporter,
)

reader_console = PeriodicExportingMetricReader(ConsoleMetricExporter())
reader_grpc = PeriodicExportingMetricReader(OTLPMetricExporter())

provider = MeterProvider(
    resource=resource, metric_readers=[reader_console, reader_grpc]
)
```

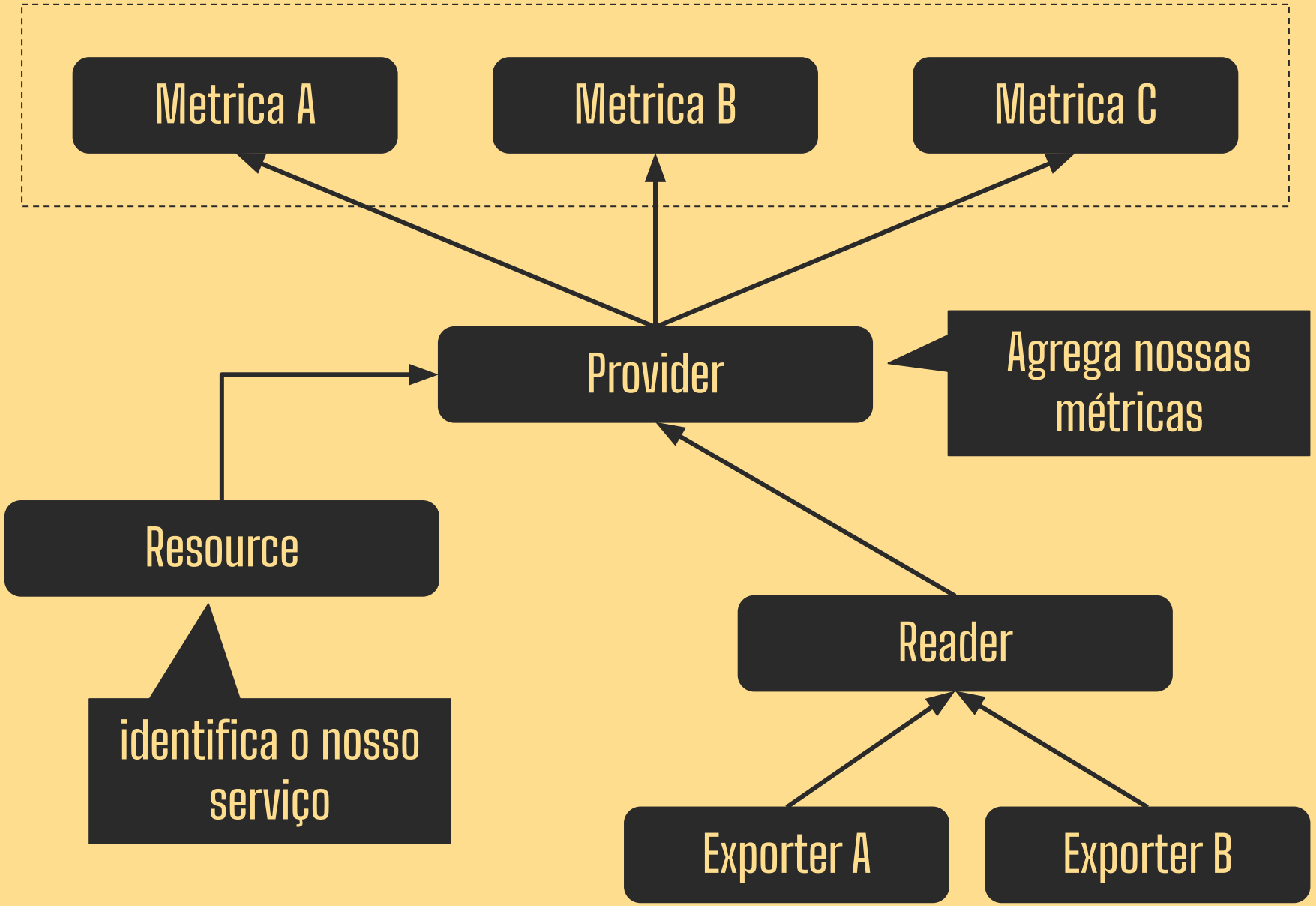
```python
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import (
    OTLPMetricExporter,
)
from opentelemetry.metrics import get_meter
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import (
    ConsoleMetricExporter,
    PeriodicExportingMetricReader,
)
from opentelemetry.sdk.resources import SERVICE_NAME, SERVICE_VERSION, Resource

resource = Resource(
    attributes={SERVICE_NAME: 'meu-app', SERVICE_VERSION: '1.0.0'}
)

reader_console = PeriodicExportingMetricReader(ConsoleMetricExporter())
reader_grpc = PeriodicExportingMetricReader(OTLPMetricExporter())

provider = MeterProvider(
    resource=resource, metric_readers=[reader_console, reader_grpc]
)

meter = get_meter('meter', meter_provider=provider)
metrica_a = meter.create_counter('A')
metrica_a.add(1)
```


Metrica A
Metrica B
Metrica C
Provider
Agrega nossas métricas
Resource
identifica o nosso serviço
Reader
Exporter A
Exporter B

Automática

Instrumentação simplificada!

Instrumentação automática

Na instrumentação automática não nos preocupamos em definir nada dentro do código. O OTel vai se encarregar de instalar as extensões necessárias para metrificar nossa aplicação!

Dentro da comunidade do opentelemetry existe um repositório focado em instrumentação de ferramentas:

https://github.com/open-telemetry/opentelemetry-python-contrib/

Usando instrumentação automática

```
export OTEL_SERVICE_NAME='meu-app'
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=0.0.0.0:4317
export OTEL_METRIC_EXPORT_INTERVAL=1000

opentelemetry-bootstrap -a install

opentelemetry-instrument python arquivo.py
```

Bootstrap

Inspeciona o projeto e baixa as dependências existentes no contrib

```
1  export OTEL_SERVIC
2  export OTEL_METRIC
3  export OTEL_EXPORT                                    :4317
4  export OTEL_METRIC_EXPORT_    ERVAL=1000
5
6  opentelemetry-bootstrap -a install
7
8  opentelemetry-instrument python arquivo.py
```

Métricas para serviços comuns

- Databases
  - `db.client.connections.usage` - Req
    - SQLAlchemy (1)
    - Database API - PEP 249 ()
    - pyMySQL ()
    - Pymongo ()
    - Tortoise ()
    - Psyco ()
    - AsyncPG ()
    - Django ()

HTTP

1. `http.server.request.duration`
2. `http.server.active_requests`
    a. ASGI (1, 2)
    b. WSGI (1, 2)
3. `http.client.request.duration`
4. `http.client.request.size`
5. `http.client.response.size`
    a. Requests (3)
    b. HTTPX ()
    c. aioClient ()
    d. urllib (3, 4 ,5)

Sistema

- Hardware
  - System - https://opentelemetry.io/docs/specs/semconv/system/system-metrics/
    - CPU
    - Memory
    - Swap
    - Disk
    - Filesystem
    - Network
    - Process
    - OS
  - Process - https://opentelemetry.io/docs/specs/semconv/system/process-metrics/
  - Runtime - https://opentelemetry.io/docs/specs/semconv/runtime/#metrics
    - CPython

      https://github.com/open-telemetry/opentelemetry-python-contrib/tree/main/instrumentation/opentelemetry-instrumentation-system-metrics

A vida real é hibrida!

No fundo, você quer aproveitar o contrib, mas também quer criar métricas customizadas.

take_3

Título	Data
Observabilidade - Uma introdução *[Link na descrição]*	11/03
Observabilidade - Métricas (Opentelemetry / Prometheus)	**Agora!**
Observabilidade - Rastreamento / Tracing (Opentelemetry / Tempo / Jeager)	08/04
Observabilidade - Logs (Opentelemetry / Loki)	15/04

Isso é uma série!

apoia.se/livedepython

pix.dunossauro@gmail.com

patreon.com/dunossauro

Ajude o projeto <3