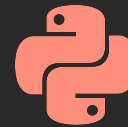


Logs e observabilidade!

Live de Python #266



1. O que são logs?

E quais os seus tipos?

2. Logs e OTel

Especificação / semântica

3. Instrumentação de logs

Um pouco mais complicado do que sempre

4. Instrumentação automática

Mostrando para que viemos

Título	Data
Observabilidade - Uma introdução	11/03
Observabilidade - Métricas (Opentelemetry / Prometheus)	25/03
Observabilidade - Rastreamento / Tracing (Opentelemetry / Tempo / Jeager)	08/04
Observabilidade - Logs (Opentelemetry / Loki)	Agora!
Mão na massa com OpenTelemetry	29/04



Isso é uma série! – Playlist na descrição!



Language Support

Logs are a [stable](#) signal in the OpenTelemetry specification. For the individual language specific implementations of the Logs API & SDK, the status is as follows:

Language	Logs
C++	Stable
C#/.NET	Stable
Erlang/Elixir	Experimental
Go	In development
Java	Stable
JavaScript	Experimental
PHP	Stable
Python	Experimental
Ruby	In development
Rust	Alpha
Swift	In development

<https://opentelemetry.io/docs/concepts/signals/logs/>



Um grande aviso antes de começar!





apoia.se/livedepython



pix.dunossauro@gmail.com



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, Cleyton 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, Mírian 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, Téó 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



O que são?
De que se
alimentam?

Logs

Logs



Logs são **registros de texto**, estruturados ou não, **contendo metadados** para **registrar eventos importantes** em um sistema.

Eventos podem ser: problemas, erros, informações sobre operações, alterações de configuração.

Normalmente são enviados para o **stdout** ou para **arquivos de log**.

Tentativa de login as XX:XX:XX com o e-mail
xxxxx@xxxx.com no serviço yyyy



Exemplo



Tipos de logs



- **Eventos: Entender como um sistema está operando**
 - Algum erro aconteceu?
 - Alguma chamada estranha foi feita?
 - Uma parte que não deveria ser chamada, foi?
- Sistema
- Acesso
- Servidor
- Alterações
- ...

Tipos de logs



- Eventos: Entender como um sistema está operando
- **Sistema: Eventos do sistema operacional**
 - Tivemos algum erro?
 - Acabou a memória?
 - Disco cheio?
- Acesso
- Servidor
- Alterações
- ...

Tipos de logs



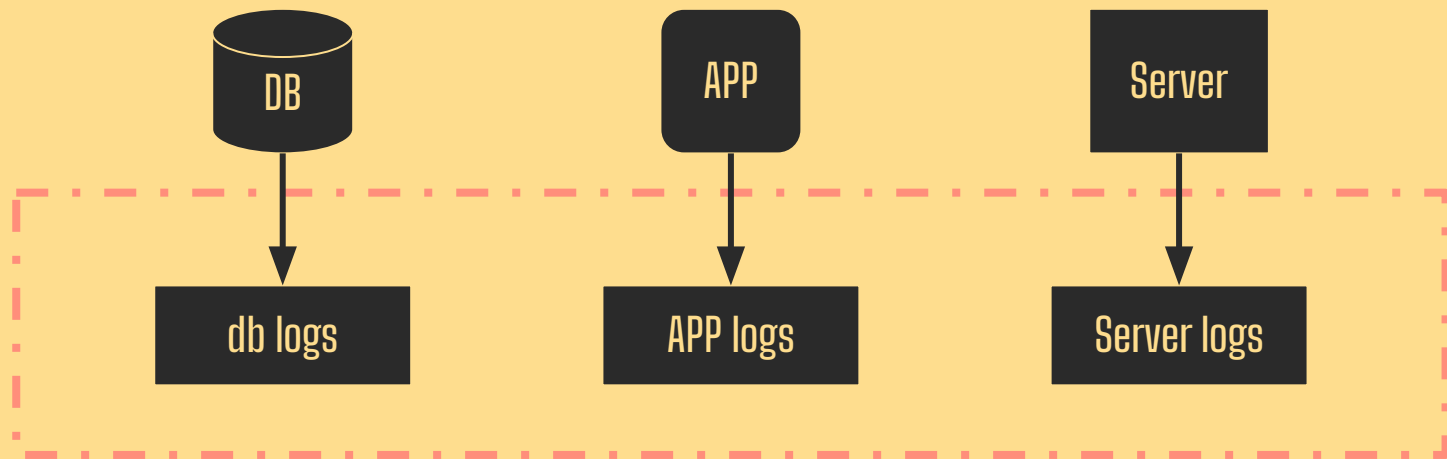
- Eventos: Entender como um sistema está operando
- Sistema: Eventos do sistema operacional
- **Acesso: Eventos relativos a recursos**
 - Algo foi solicitado? Quem solicitou?
 - Problemas de permissão?
- **Servidor: Eventos do servidor**
 - Uma requisição no endpoint X
 - Ips de quem solicitou
 - Quando foi solicitado
- Alterações
- ...

Tipos de logs



- Eventos: Entender como um sistema está operando
- Sistema: Eventos do sistema operacional
- Acesso: Eventos relativos a recursos
- Servidor: Eventos do servidor
- **Alterações: Alterações de configuração**
 - O serviço agora iniciou na versão 2.0
 - As variáveis de ambiente foram alteradas
 - A conexão com o banco de dados mudou
 - *Coisas elásticas*
- ...

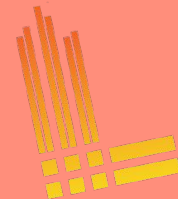
Logs são normalmente descentralizados



A proposta é que não sejam



A proposta é que não sejam





Como criar logs em Python? - Live de Python #198 [Logging, Loguru, Sentry]

Eduardo Mendes • 13 mil visualizações • Transmitido há 2 anos

Na live de hoje vamos conversar sobre como estruturar logs em python. Quais ferramentas usar. Vamos entender a estrutura da biblioteca padrão logging, dos handlers, formatters e filters. Também...



De volta aos logs



Logs, de maneira geral, tem dois conceitos importantes:

- Níveis / Leveis / Severity: Gravidade ou urgência da mensagem
- Handlers: Uma rotina associada ao evento
- Propagação de contexto: A configuração é global

Leveis



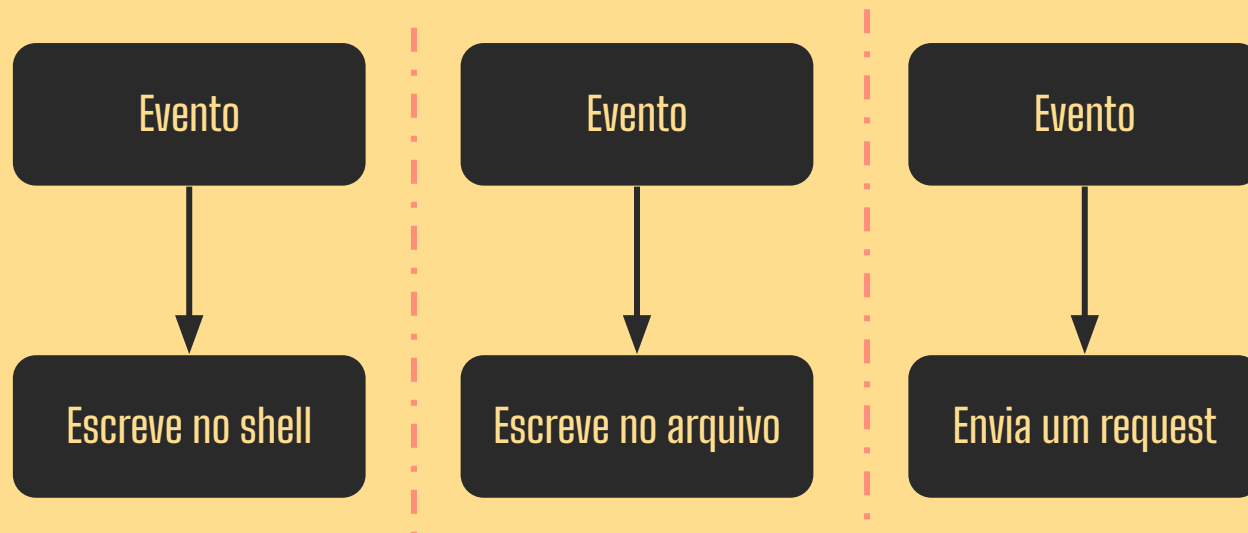
Nível	Quando é usado
DEBUG	Informação detalhada, tipicamente de interesse apenas quando diagnosticando problemas.
INFO	Confirmação de que as coisas estão funcionando como esperado.
WARNING	Uma indicação que algo inesperado aconteceu, ou um indicativo que algum problema em um futuro próximo (ex.: 'pouco espaço em disco'). O software está ainda funcionando como esperado.
ERROR	Por conta de um problema mais grave, o software não conseguiu executar alguma função.
CRITICAL	Um erro grave, indicando que o programa pode não conseguir continuar rodando.

<https://docs.python.org/pt-br/3/howto/logging.html#logging-basic-tutorial>

Handlers



Blocos de códigos específicos para executar determinadas ações em eventos. Por exemplo, quando um evento de log acontecer, o que fazer?



Um grupo de handlers





```
1  from logging import DEBUG, getLogger, StreamHandler, FileHandler
2
3  logger = getLogger( )
4  logger.setLevel(DEBUG)
5
6  logger.addHandler(StreamHandler( ))
7  logger.addHandler(FileHandler('arquivo.log'))
8
9  logger.info('Tudo certo até aqui!')
```

Filtro por níveis



Cada handler pode ter um nível específico, isso ajuda a filtrar o que vai para cada lugar.


```
1  from logging import DEBUG, INFO, FileHandler, StreamHandler, getLogger
2
3  logger = getLogger()
4  logger.setLevel(DEBUG)
5
6  shell_handler = StreamHandler()
7  shell_handler.setLevel(DEBUG)
8  logger.addHandler(shell_handler)
9
10 file_handler = FileHandler('arquivo.log')
11 file_handler.setLevel(INFO)
12 logger.addHandler(file_handler)
13
14 logger.info('Tudo certo até aqui!') # para os 2
15 logger.info('Passei aqui') # para os shell
```

E mais...



Existem diversos outros conceitos importantes como:

- Filters
- Formatters
- Outros handlers
- Configuração declarativa
- ...

A thumbnail image for a live stream. It shows a close-up of green leaves with water droplets. Overlaid on the image is the text 'Live de Python #48' in a large, bold, white font. Below it, in a smaller white font, is 'Criando logs para aplicações (logging)'. In the bottom right corner of the image, there is a white timestamp '1:23:11'.

Live de Python #48 - Criando logs para aplicações

Eduardo Mendes • 5,7 mil visualizações • Transmitido há 5 anos

Código: <https://github.com/dunossauro/live-de-python> Telegram do live: <https://t.me/livepython> Perguntas frequentes: <https://github.com/dunossauro/live-de-python/wiki>

Estruturas de dados em logs



Por padrão, a biblioteca logging não tem uma forma estruturada de emitir logs, não existe um handler nativo para isso. O que faz com que todos os logs sejam **não estruturados**.

Sempre existe a necessidade de fazer buscas textuais em arquivos ou em aplicações como *logstash*, boas em buscar em texto.

Json logger



Existem algumas alternativas para handlers, mas são ineficientes, na minha opinião

```
1  import logging
2  # pip install python-json-logger
3  from pythonjsonlogger import jsonlogger
4
5  logger = logging.getLogger()
6  logger.setLevel(logging.DEBUG)
7
8  shell_handler = logging.StreamHandler()
9  formatter = jsonlogger.JsonFormatter()
10 shell_handler.setFormatter(formatter)
11 logger.addHandler(shell_handler)
12
13 logger.info('Passei aqui!') # {"message": "Passei aqui!"}
```

Ferramentas externas



Podemos alterar a biblioteca de logs, por coisas que façam mais sentido no mundo moderno, como o **loguru**.

```
1  from sys import stderr
2  from loguru import logger
3
4  logger.add(stderr, level='DEBUG', serialize=True)
5
6  logger.info('Passei aqui!')
```

```
1 {
2   "text": "2024-04-15 18:18:50.413 | INFO      | __main__:<module>:6 - Passei aqui!\n",
3   "record": {
4     "elapsed": {
5       "repr": "0:00:00.004416",
6       "seconds": 0.004416
7     },
8     "exception": null,
9     "extra": {},
10    "file": {
11      "name": "exemplo_03.py",
12      "path": "/home/dunossauro/live_logs/intro/exemplo_03.py"
13    },
14    "function": "<module>",
15    "level": {
16      "icon": "i",
17      "name": "INFO",
18      "no": 20
19    },
20    "line": 6,
21    "message": "Passei aqui!",
22    "module": "exemplo_03",
23    "name": "__main__",
24    "process": {
25      "id": 349170,
26      "name": "MainProcess"
27    },
28    "thread": {
29      "id": 138163716003648,
30      "name": "MainThread"
31    },
32    "time": {
33      "repr": "2024-04-15 18:18:50.413768-03:00",
34      "timestamp": 1713215930.413768
35    }
36  }
37 }
```

OTel

Logs e
opentelemetry

Igual, mas diferente

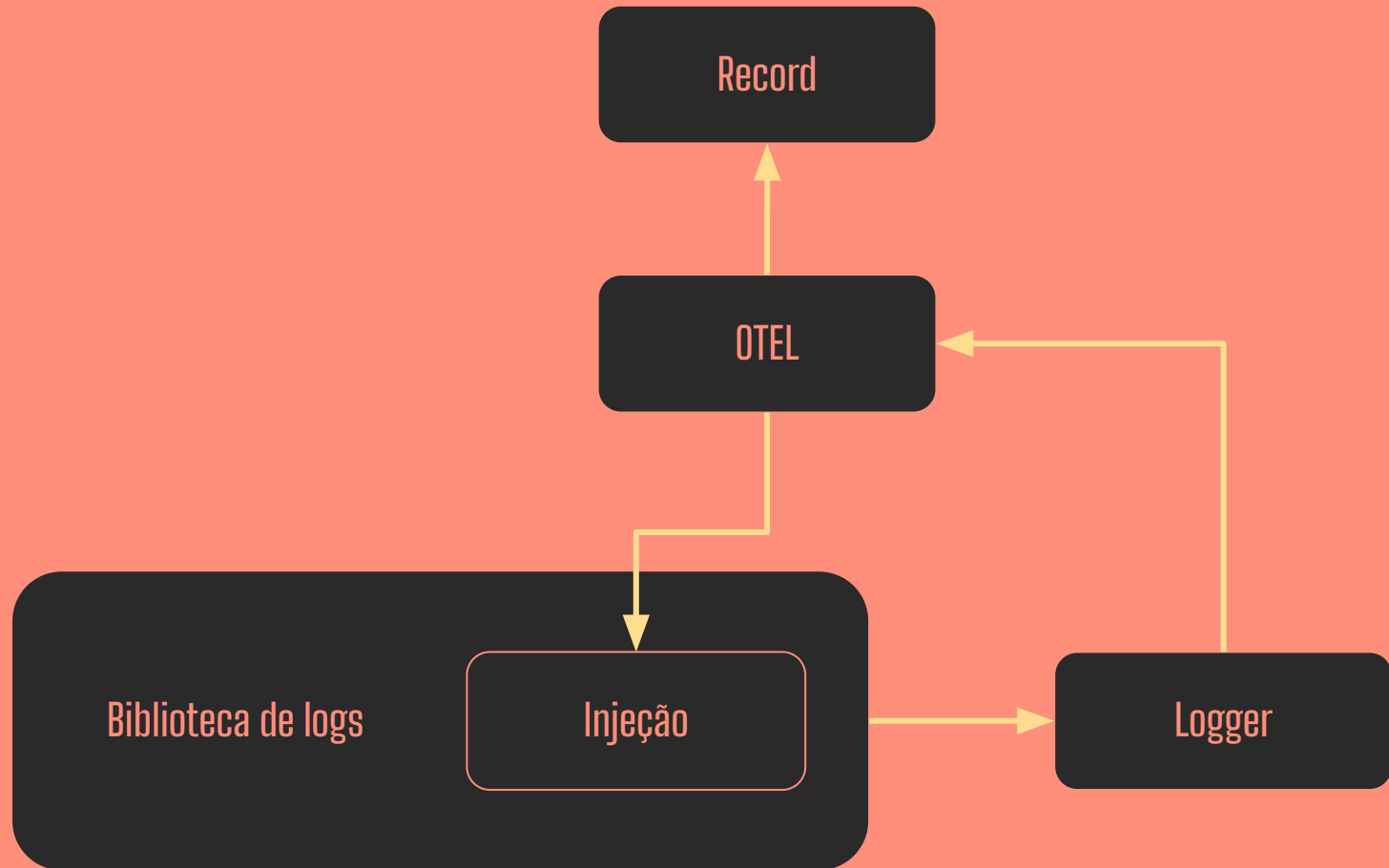


Diferente dos outros sinais, como métricas e traces, os logs tendem a ser **aditivos**.

Pois as linguagens já tem uma forma bem estabelecida de gerar logs. No caso do python até a biblioteca padrão faz isso.

O padrão é que exista uma ponte/bridge entre a instrumentação e o sistema de logs. Fazendo com que o OTel não seja chamado diretamente, como os outros sinais.

Schema



Records



A ideia dos records é que eles sejam **sempre estruturados** e carreguem os dados do sinal de trace além do corpo do log. Para existir uma associação.

Field Name	Description
Timestamp	Time when the event occurred.
ObservedTimestamp	Time when the event was observed.
TraceId	Request trace ID.
SpanId	Request span ID.
TraceFlags	W3C trace flag.
SeverityText	The severity text (also known as log level).
SeverityNumber	Numerical value of the severity.
Body	The body of the log record.
Resource	Describes the source of the log.
InstrumentationScope	Describes the scope that emitted the log.
Attributes	Additional information about the event.

<https://opentelemetry.io/docs/concepts/signals/logs/>



General Logs Attributes

Status: [Experimental](#)

The attributes described in this section are rather generic. They may be used in any Log Record they apply to.

The following semantic conventions for logs are defined:

- **[General](#):** General semantic attributes that may be used in describing Log Records.
- [Exceptions](#): Semantic attributes that may be used in describing exceptions in logs.
- [Feature Flags](#): Semantic attributes that may be used in describing feature flag evaluations in logs.

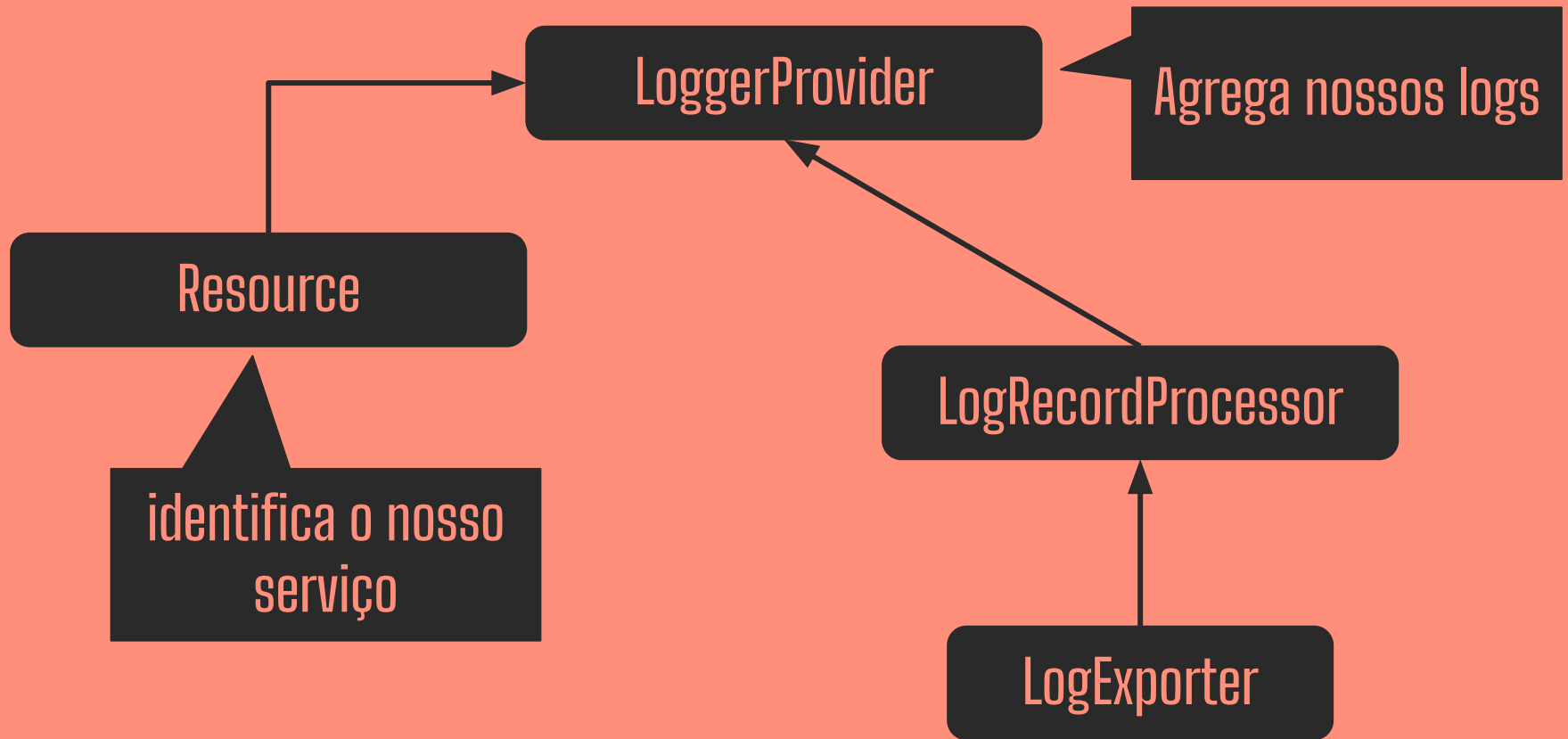
Apart from semantic conventions for logs, [events](#), [traces](#), and [metrics](#), OpenTelemetry also defines the concept of overarching [Resources](#) with their own [Resource Semantic Conventions](#).

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

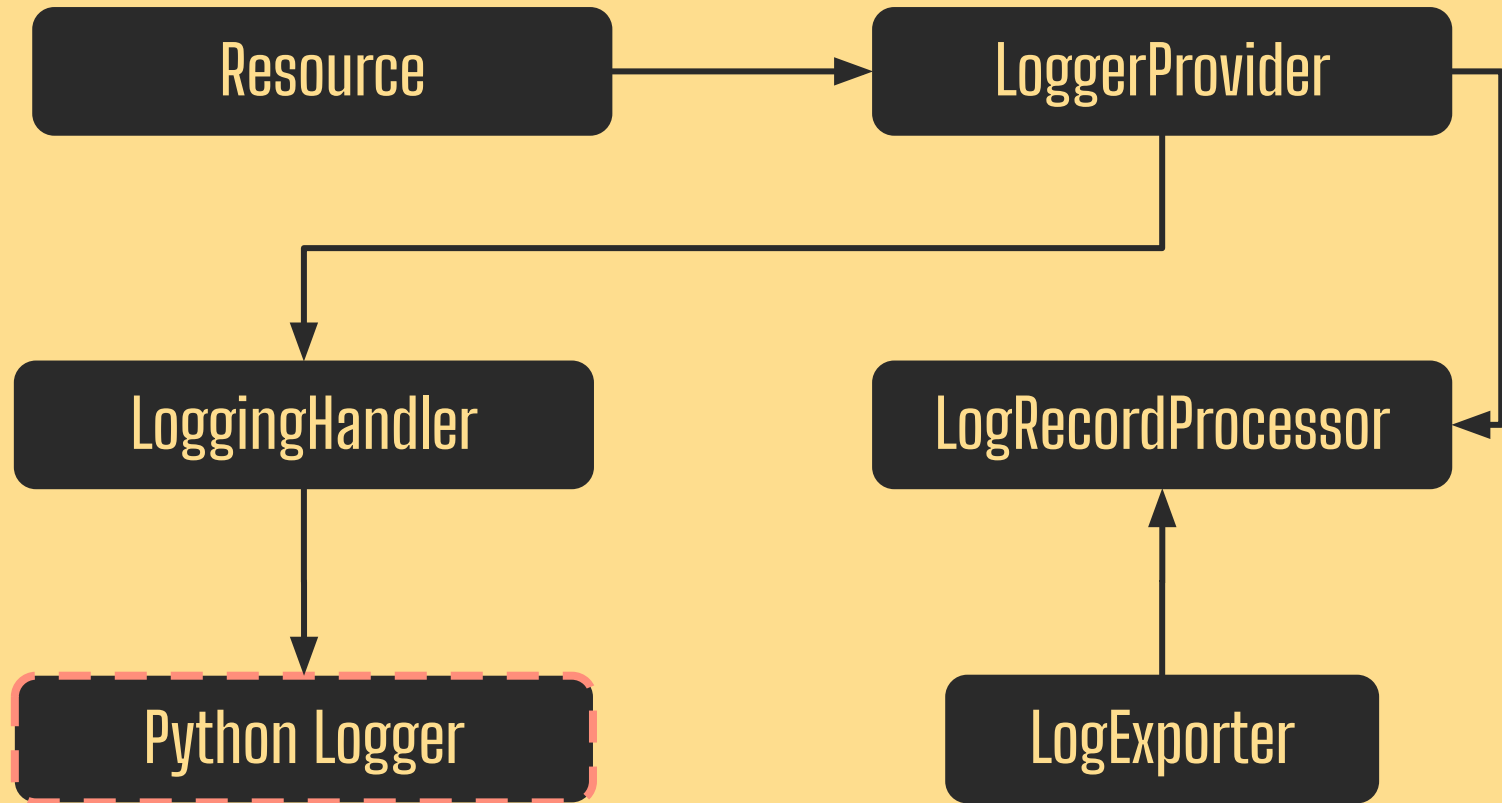
Instrumentação
manual

Manual

Objetos relacionados o Logs



Ligando ao logger



Language Support

Logs are a [stable](#) signal in the OpenTelemetry specification. For the individual language specific implementations of the Logs API & SDK, the status is as follows:

Language	Logs
C++	Stable
C#/.NET	Stable
Erlang/Elixir	Experimental
Go	In development
Java	Stable
JavaScript	Experimental
PHP	Stable
Python	Experimental
Ruby	In development
Rust	Alpha
Swift	In development

<https://opentelemetry.io/docs/concepts/signals/logs/>



REPETINDO O AVISO,
você pode ter chegado depois



Muitos underlines



Pelo fato do sinal de log ainda ser instável na linguagem. Todos os imports referentes a logs se iniciam com `__`.

Quando isso for estável, provavelmente exigirá manutenção

```
1 from opentelemetry.__logs import set_logger_provider
2 from opentelemetry.sdk.__logs import LoggerProvider, LoggingHandler
3 from opentelemetry.sdk.__logs.export import (
4     BatchLogRecordProcessor,
5     ConsoleLogExporter,
6 )
```

O menor boilerplate que você vai ver hoje



```
1 import logging
2
3 from opentelemetry._logs import set_logger_provider
4 from opentelemetry.sdk._logs import LoggerProvider, LoggingHandler
5 from opentelemetry.sdk._logs.export import (
6     BatchLogRecordProcessor,
7     ConsoleLogExporter,
8 )
9 from opentelemetry.sdk.resources import SERVICE_NAME, Resource
10
11 resource = Resource({SERVICE_NAME: 'LogAPP'})
12 provider = LoggerProvider(resource=resource)
13 processor = BatchLogRecordProcessor(ConsoleLogExporter())
14 provider.add_log_record_processor(processor)
15
16 set_logger_provider(provider)
17
18 handler = LoggingHandler(level=logging.INFO, logger_provider=provider)
19
20 logger = logging.getLogger()
21
22 # OTel Handler
23 logger.addHandler(handler)
24
25 # Default Handler
26 logger.addHandler(logging.StreamHandler())
27
28 logger.setLevel(logging.DEBUG)
29
30 logger.critical('aaaaaaaa')
```



Via OTLP



```
1 import logging
2
3 from opentelemetry._logs import set_logger_provider
4 from opentelemetry.sdk._logs import LoggerProvider, LoggingHandler
5 from opentelemetry.sdk._logs.export import BatchLogRecordProcessor
6 from opentelemetry.exporter.otlp.proto.grpc._log_exporter import (
7     OTLPLogExporter
8 )
9 from opentelemetry.sdk.resources import SERVICE_NAME, Resource
10
11 resource = Resource({SERVICE_NAME: 'LogAPP'})
12 provider = LoggerProvider(resource=resource)
13
14 # Exporter via OTLP
15 processor = BatchLogRecordProcessor(OTLPLogExporter())
16
17 provider.add_log_record_processor(processor)
18
19 set_logger_provider(provider)
20
21 handler = LoggingHandler(level=logging.INFO, logger_provider=provider)
22
23 logger = logging.getLogger(__name__)
24
25 # OTel Handler
26 logger.addHandler(handler)
27
28 # Default Handler
29 logger.addHandler(logging.StreamHandler())
30
31 logger.setLevel(logging.DEBUG)
32
33 logger.critical('aaaaaaa')
```




Embora exista a solução padrão e experimental, existe a instrumentação manual no contrib para a biblioteca de Logging. O que se torna uma forma alternativa de manusear os logs. (*é confuso, eu sei*)

 / OpenTelemetry Logging Instrumentation  Edit on GitHub

OpenTelemetry Logging Instrumentation

The OpenTelemetry `logging` integration automatically injects tracing context into log statements.

The integration registers a custom log record factory with the the standard library logging module that automatically inject tracing context into log record objects. Optionally, the integration can also call `logging.basicConfig()` to set a logging format with placeholders for span ID, trace ID and service name.

The following keys are injected into log record objects by the factory:

- `otelSpanID`
- `otelTraceID`
- `otelServiceName`
- `otelTraceSampled`

<https://opentelemetry-python-contrib.readthedocs.io/en/latest/instrumentation/logging/logging.html>

Instrumentação

Autom
ática

Ufa... Tudo faz sentido novamente!



```
1 import logging
2
3 logger = logging.getLogger()
4
5 logger.critical('PEI')
```

```
1 export OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true
2 export OTEL_EXPORTER_OTLP_ENDPOINT=0.0.0.0:4317
3 export OTEL_EXPORTER_OTLP_INSECURE=true
4 export OTEL_LOGS_EXPORTER=otlp
5 export OTEL_SERVICE_NAME=sample
6
7 opentelemetry-instrument python exemplo_01.py
```

Bora subir o Loki +
grafana + tempo!



Uma pequena demo!



Via compose



```
1  app:
2    depends_on:
3      - otel
4    build: 'dt/.'
5    environment:
6      - OTEL_PYTHON_LOGGING_AUTO_INSTRUMENTATION_ENABLED=true
7      - OTEL_EXPORTER_OTLP_ENDPOINT=otel:4317
8      - OTEL_EXPORTER_OTLP_INSECURE=true
9      - OTEL_SERVICE_NAME=appzin
10     - OTEL_TRACES_EXPORTER=otlp
11     - OTEL_LOGS_EXPORTER=otlp
```

Bora subir o Loki +
grafana + tempo!



O demo completo





apoia.se/livedepython



pix.dunossauro@gmail.com



patreon.com/dunossauro



Ajude o projeto <3

