cdumay-opentracing Documentation

Release 0.0.1

Cedric Dumay

Feb 19, 2021

Contents

cdumay-opentracing is a python library to facilitate opentracing integration.

CHAPTER 1

Example

The following example shows how to use it with Jaeger:

```
import opentracing
from cdumay_opentracing import OpenTracingDriver, OpenTracingSpan


class A(object):
    "A sample class"
    def __init__(self):
        self.a = 5
        self.b = "toto"
        self.trace = dict()


class B(A):
    """A class which inherit of A"""


class DriverA(OpenTracingDriver):
    @classmethod
    def extract(cls, data):
        return opentracing.tracer.extract(cls.FORMAT, data.trace)

    @classmethod
    def inject(cls, span, data):
        opentracing.tracer.inject(span, cls.FORMAT, data.trace)

    @classmethod
    def tags(cls, data):
        return dict(a=data.a, b=data.b)


def child():
    """A function that automatically recovers the current span"""
```

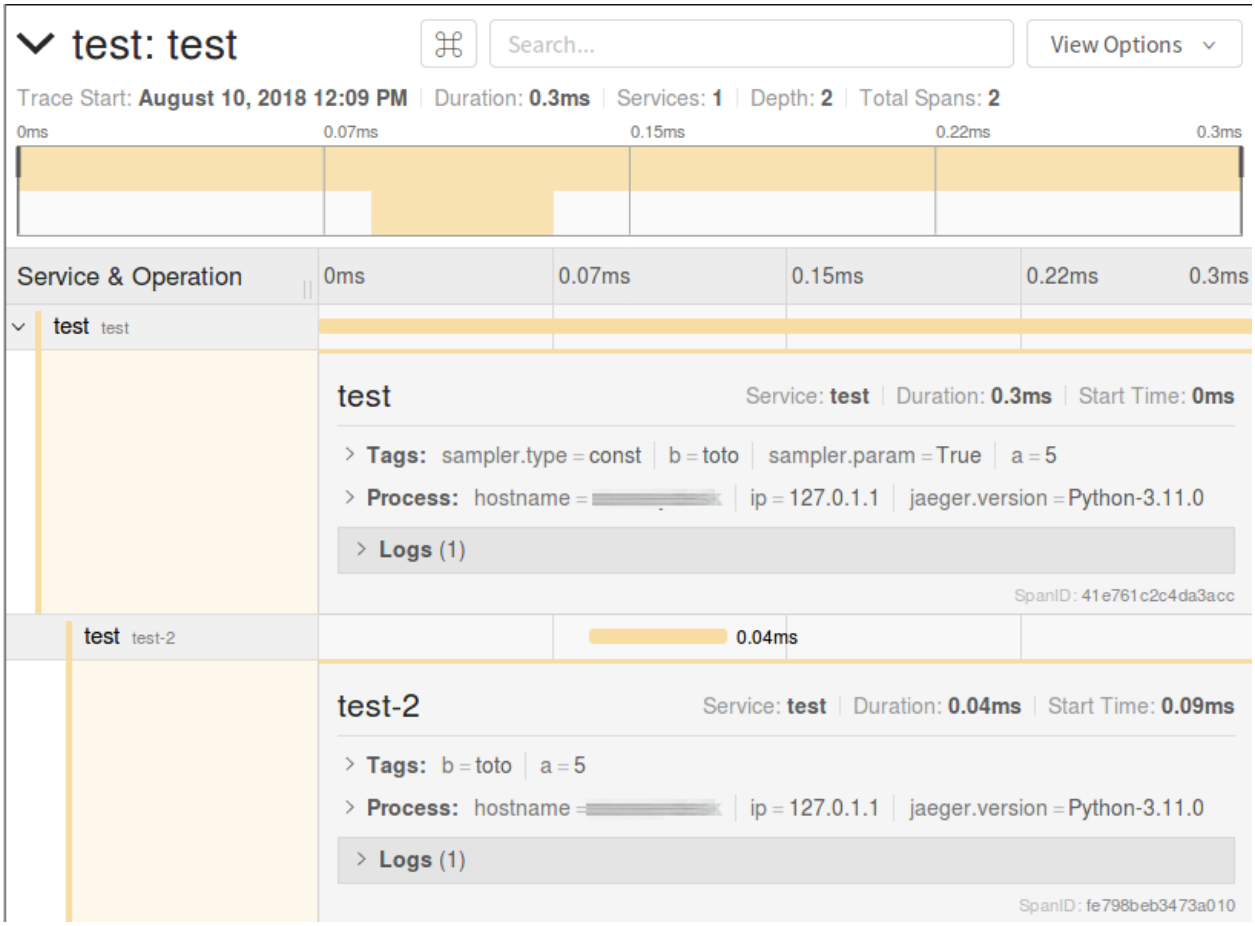
(continues on next page)

(continued from previous page)

```
33      with OpenTracingSpan(B(), "test-2") as span:
34          span.log_kv(dict(event="done"))
35
36
37  if __name__ == '__main__':
38      import os, time
39      from jaeger_client import Config
40      from cdumay_opentracing import OpenTracingManager
41
42      tracer = Config(service_name="test", config=dict(
43          sampler=dict(type='const', param=1), logging=True,
44          local_agent=dict(reporting_host=os.getenv('JAEGER_HOST', 'localhost'))
45      )).initialize_tracer()
46
47      OpenTracingManager.register(A, DriverA)
48
49      a = A()
50      with OpenTracingSpan(a, "test") as span:
51          child()
52          OpenTracingManager.log_kv(span, a, "done")
53
54      time.sleep(4)
55      tracer.close()
```

1.1 Explanations

- **line 5-14**: We create classes for our example.
- **line 17-28**: We create a driver which allows to automate the manipulation of the class `A`.
- **line 38-45**: We initialize tracing
- **line 47**: We register the driver into the Tracing manager. It will allow to manipulate `A` instances (and by extension `B`).
- **line 49-50**: We make a span using `a`.
- **line 51**: We call a function from a span which will create a sub-span.
- **line 54-55**: We sleep and close to make sure that the span is sent.


test: test
Search...
View Options
Trace Start: August 10, 2018 12:09 PM
Duration: 0.3ms
Services: 1
Depth: 2
Total Spans: 2
0ms
0.07ms
0.15ms
0.22ms
0.3ms
Service & Operation
test test
test
Service: test
Duration: 0.3ms
Start Time: 0ms
Tags: sampler.type = const
b = toto
sampler.param = True
a = 5
Process: hostname =
ip = 127.0.1.1
jaeger.version = Python-3.11.0
Logs (1)
SpanID: 41e761c2c4da3acc
test test-2
0.04ms
test-2
Service: test
Duration: 0.04ms
Start Time: 0.09ms
Tags: b = toto
a = 5
Process: hostname =
ip = 127.0.1.1
jaeger.version = Python-3.11.0
Logs (1)
SpanID: fe798beb3473a010

CHAPTER 2

Get It Now

Install the lib using pip:

```
pip install -U cdumay-opentracing
```

CHAPTER 3

Features & API Focus

3.1 `OpenTracingManager` — The trace manager

class cdumay_opentracing.**OpenTracingManager**

This class manages the opentracing context.

- It allows to recover using the `opentracing` library the initialized tracer.
- It manage spans stack

classmethod register(*clazz*, *driver*)

Register a driver for the given class

Parameters

- **clazz** (*Any*) – Class managed by the driver.
- **driver** (cdumay_opentracing.OpenTracingDriver) – Driver to register.

classmethod get_driver(*obj*)

Find registered driver for the given object

Parameters **obj** (*Any*) – object to manipulate

Returns registered driver for this class

Return type *cdumay_opentracing.OpenTracingDriver*

classmethod tags(*obj*)

Extract tags from *carrier* object using a registered driver.

Parameters **obj** (*Any*) – object to manipulate

Returns Tags to add on span

Return type dict

classmethod get_current_span()

Get the current span

Returns The current span

Return type opentracing.span.Span or None

classmethod create_span(*obj*, *name*, *tags*)
Create a new span

Parameters

- **obj** (*Any*) – Any object
- **name** (*str*) – Span name
- **tags** (*dict*) – Additional tags

Returns Span

Return type opentracing.span.Span

classmethod finish_span(*span*)
Terminate the given span

Parameters **span** (*opentracing.span.Span*) – Span to finish

classmethod log_kv(*span*, *obj*, *event*, ***kwargs*)
Adds a log record to the Span.

Parameters

- **span** (*opentracing.span.Span*) – the Span instance to use.
- **obj** (*Any*) – the *carrier* object.
- **event** (*str*) – Span event name.
- **kwargs** (*dict*) – A dict of string keys and values of any type to log

Returns Returns the Span itself, for call chaining.

Return type opentracing.span.Span

3.2 OpenTracingDriver - Classes driver

class cdumay_opentracing.**OpenTracingDriver**

FORMAT
`opentracing.Format` used to load and store span context

classmethod extract(*data*)

Extract span context from a *carrier* object

Parameters **data** (*Any*) – the *carrier* object.

Returns a SpanContext instance extracted from *carrier* or None if no such span context could be found.

classmethod inject(*span*, *data*)

Injects the span context into a *carrier* object.

Parameters

- **span** (*opentracing.span.SpanContext*) – the SpanContext instance to inject
- **data** (*Any*) – the *carrier* object.

classmethod tags(*data*)

Extract tags from *carrier* object.

Parameters **data** (*Any*) – the *carrier* object.

Returns Tags to add on span

Return type dict

classmethod log_kv(*span*, *data*, *event*, ***kwargs*)

Adds a log record to the Span.

Parameters

- **span** (*opentracing.span.Span*) – the Span instance to use.
- **data** (*Any*) – the *carrier* object.
- **event** (*str*) – Span event name.
- **kwargs** (*dict*) – A dict of string keys and values of any type to log

Returns Returns the Span itself, for call chaining.

Return type opentracing.span.Span

Index