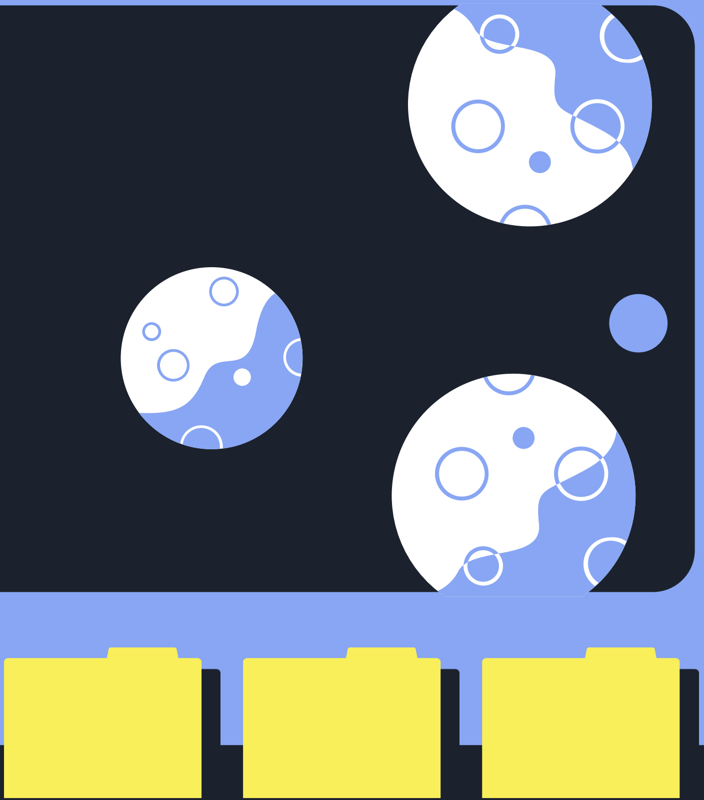
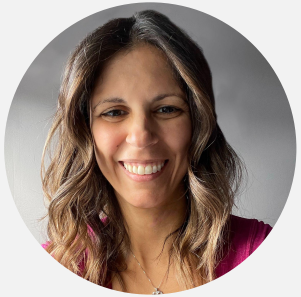
Dude, Where's My ERROR?

How OpenTelemetry Records Errors, and Why it Does it Like That



About US



Adriana Villela

Sr. Staff Developer Advocate

ServiceNow Cloud Observability

(formerly Lightstep)

bento.me/adrianamvillela

Reese Lee

Senior Developer Relations Engineer

New Relic

x.com/reesesbytes

It started with a simple statement…

“There was an interesting Slack discussion I was part of recently where someone asked how OpenTelemetry deals with 'error recording'.”

Dude, where's my ERROR?

Image source: https://images.app.goo.gl/sXK8vuGWcymcZbcC7

Table of Contents

01

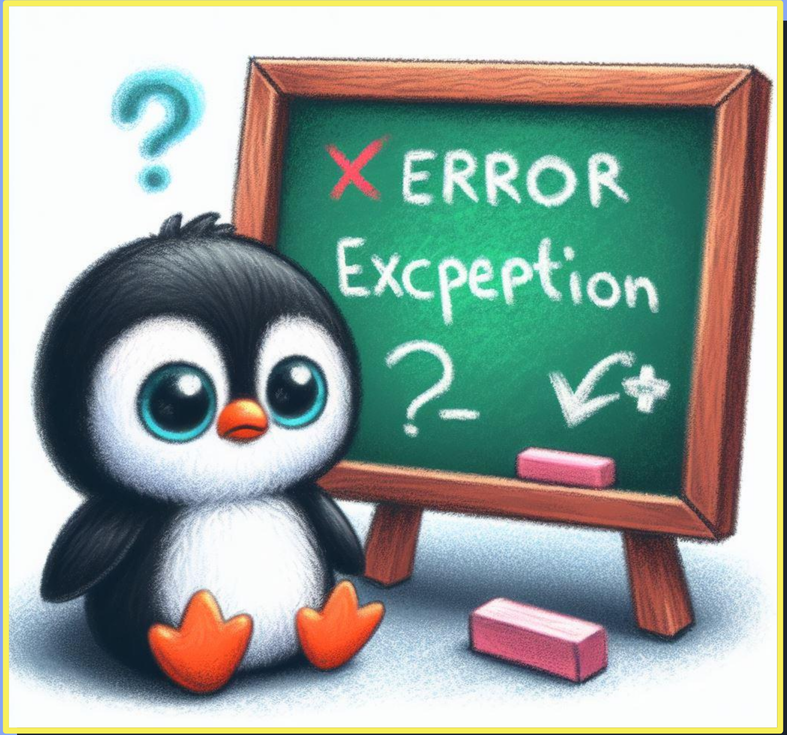
Background

Languages Approach Errors Differently

When languages don't agree on errors, what do you use to get consistent telemetry for microservices in those languages?

OpenTelemetry Refresher

An open source, vendor-neutral observability framework for instrumenting, generating, collecting, and exporting telemetry data.



Errors vs Exceptions

What's the difference?

Errors vs Exceptions

Error

An unexpected issue in a program that hinders its execution.

Exception

A type of runtime error that disrupts the normal flow of a program.

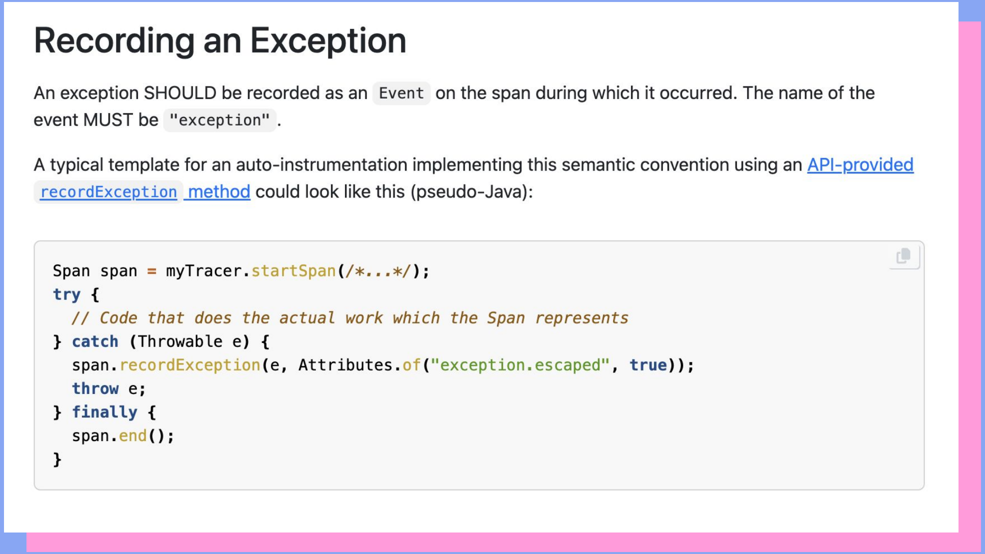
02

Error handling in OTel

How does OTel deal with these differences?

With the OTel specification!

The "spec" provides a blueprint...



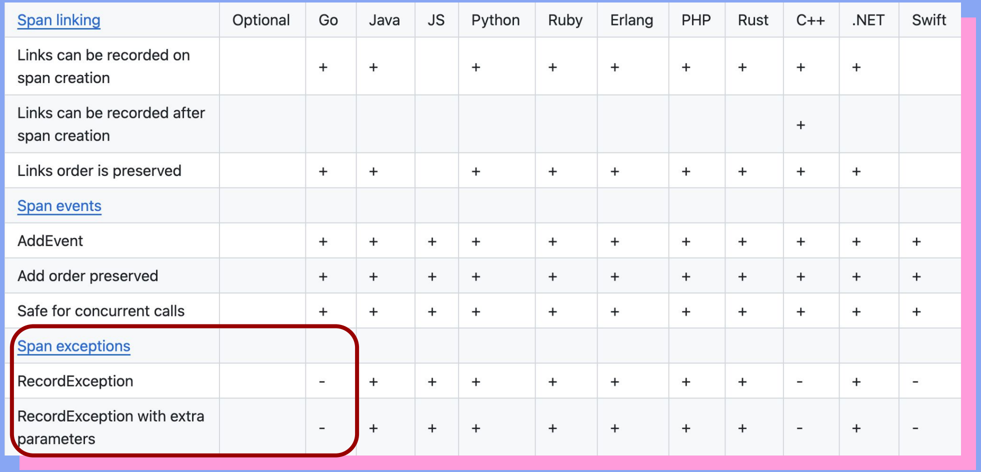
Recording an Exception

An exception SHOULD be recorded as an `Event` on the span during which it occurred. The name of the event MUST be `"exception"`.

A typical template for an auto-instrumentation implementing this semantic convention using an API-provided `recordException` method could look like this (pseudo-Java):

```
Span span = myTracer.startSpan(/*...*/);
try {
  // Code that does the actual work which the Span represents
} catch (Throwable e) {
  span.recordException(e, Attributes.of("exception.escaped", true));
  throw e;
} finally {
  span.end();
}
```

…but allows for some **flexibility**



Span linking	Optional	Go	Java	JS	Python	Ruby	Erlang	PHP	Rust	C++	.NET	Swift
Links can be recorded on span creation		+	+		+	+	+	+	+	+	+	
Links can be recorded after span creation										+		
Links order is preserved		+	+		+	+	+	+	+	+	+	
Span events												
AddEvent		+	+	+	+	+	+	+	+	+	+	+
Add order preserved		+	+	+	+	+	+	+	+	+	+	+
Safe for concurrent calls		+	+	+	+	+	+	+	+	+	+	+
Span exceptions												
RecordException		-	+	+	+	+	+	+	+	-	+	-
RecordException with extra parameters		-	+	+	+	+	+	+	+	-	+	-

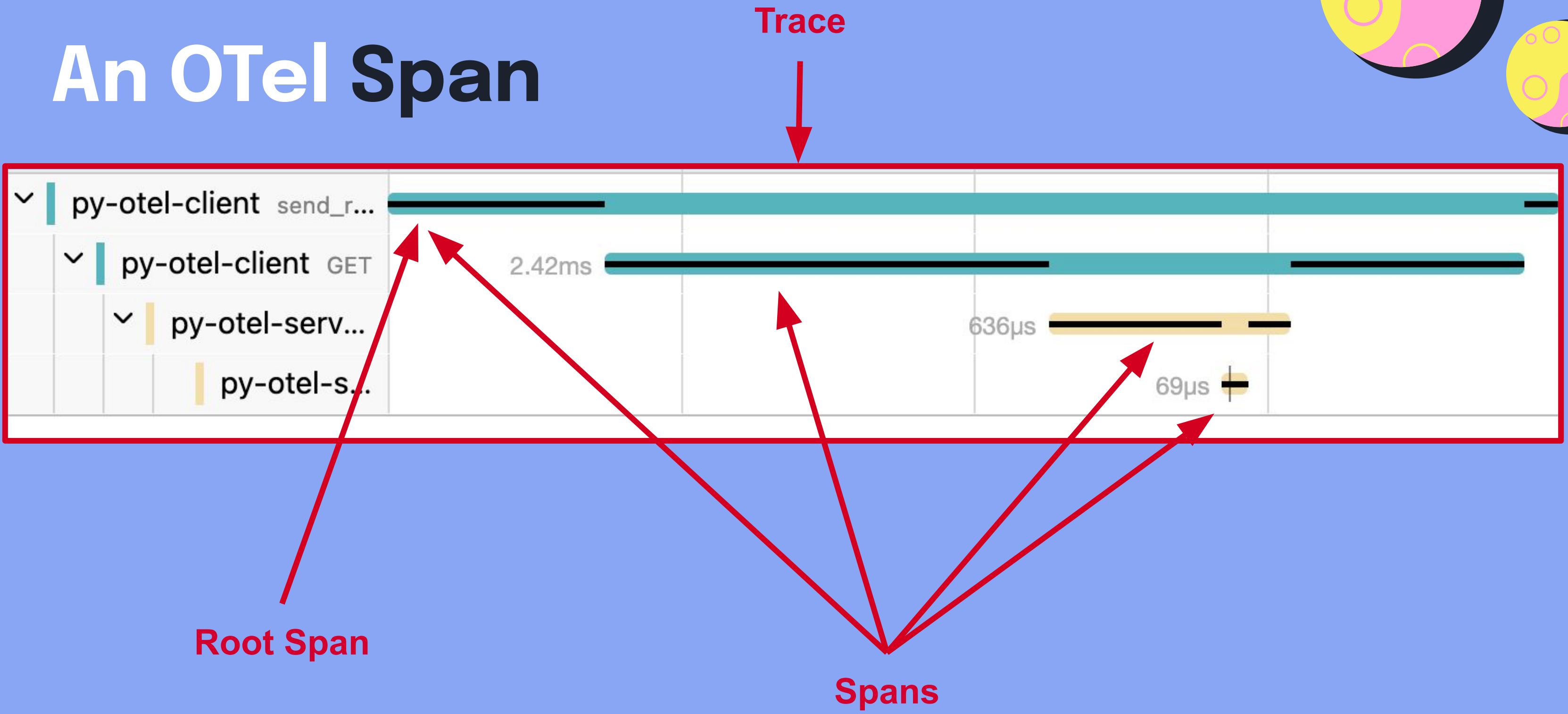
Now that we have a unified framework...

Let's learn what options OTel provides for handling errors!

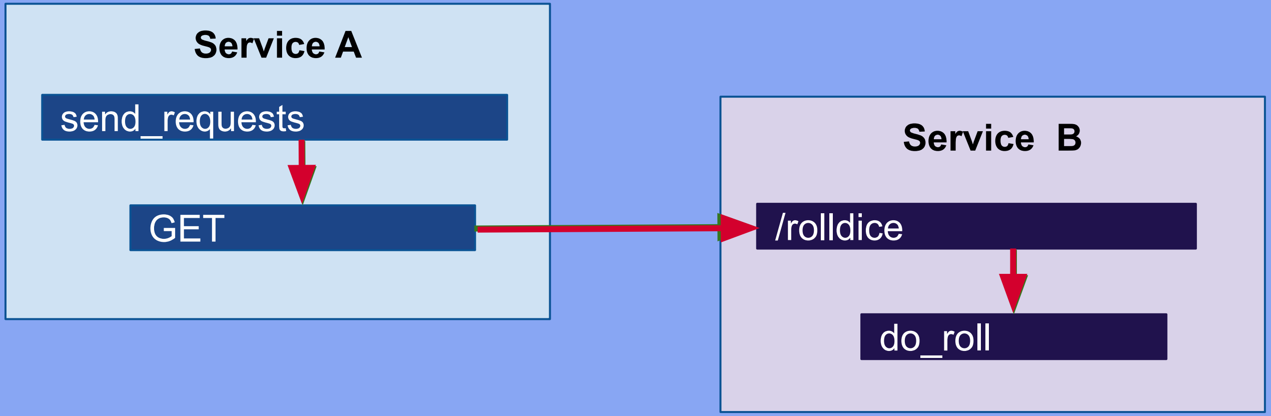
Spans

Logs

An OTel Span


Trace
py-otel-client send_r...
py-otel-client GET
2.42ms
py-otel-serv...
636µs
py-otel-s...
69µs
Root Span
Spans

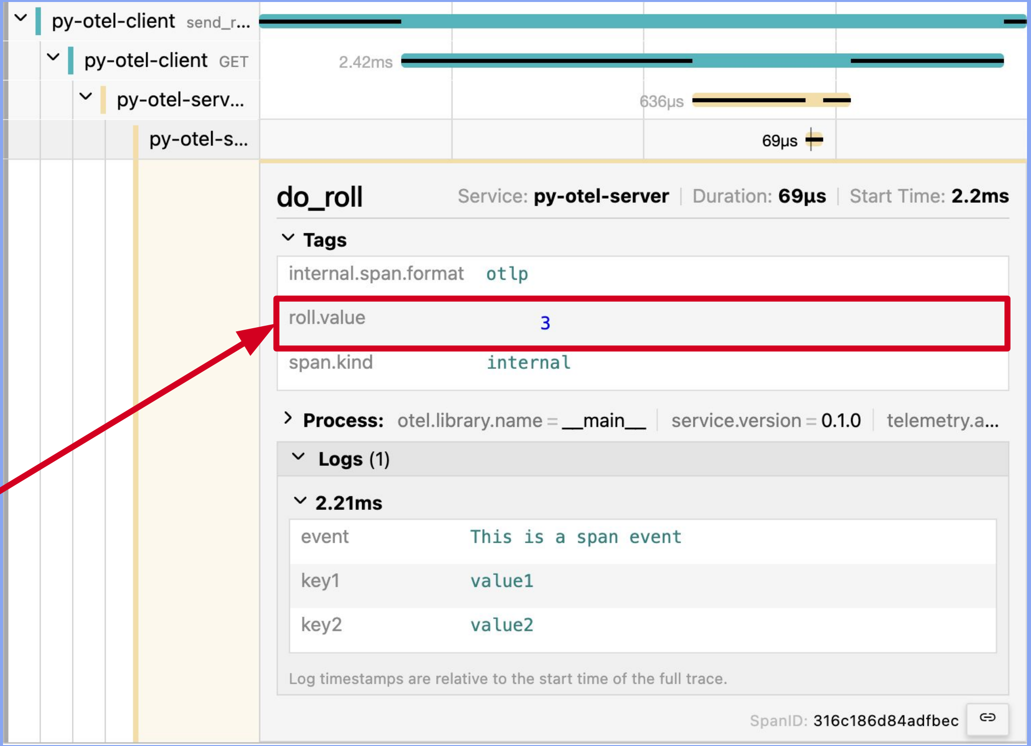
OpenTelemetry Trace Context



Context propagation

Enhancing spans with metadata

Attribute (metadata)



do_roll Service: **py-otel-server** | Duration: **69µs** | Start Time: **2.2ms**

Tags

internal.span.format	otlp
roll.value	3
span.kind	internal

Process: otel.library.name = __main__ | service.version = 0.1.0 | telemetry.a...

Logs (1)

2.21ms

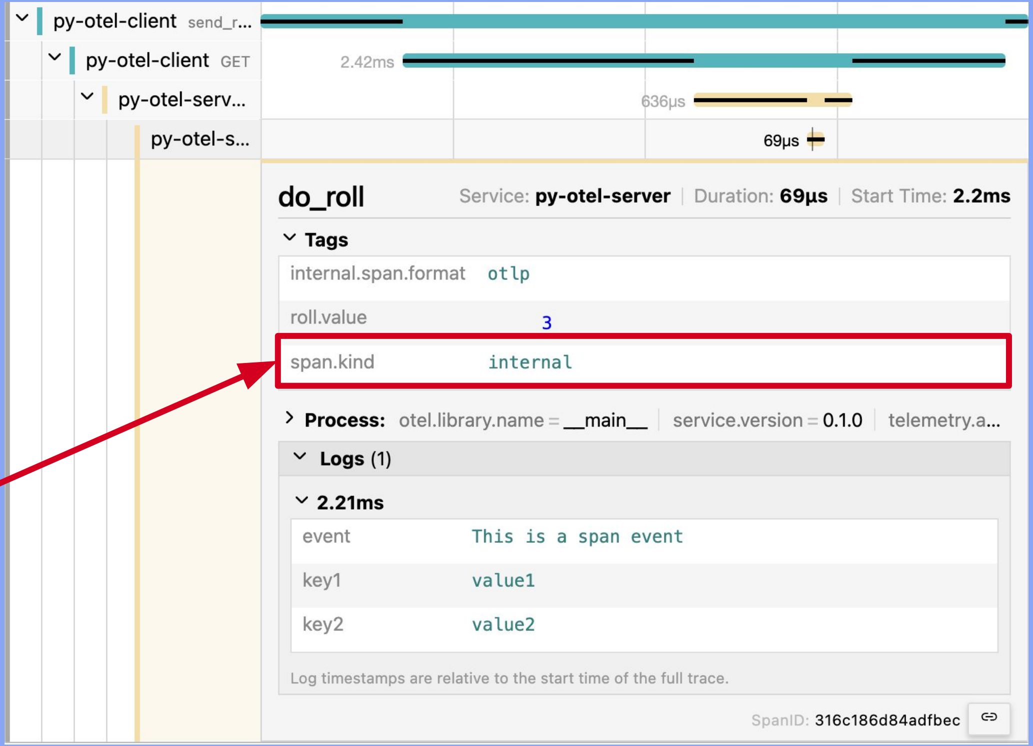
event	This is a span event
key1	value1
key2	value2

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

SpanID: 316c186d84adfbec

Enhancing spans with span kind

span kind



py-otel-client	send_r...
py-otel-client	GET 2.42ms
py-otel-serv...	636µs
py-otel-s...	69µs

do_roll Service: py-otel-server | Duration: 69µs | Start Time: 2.2ms

Tags

internal.span.format	otlp
roll.value	3
span.kind	internal

Process: otel.library.name = __main__ | service.version = 0.1.0 | telemetry.a...

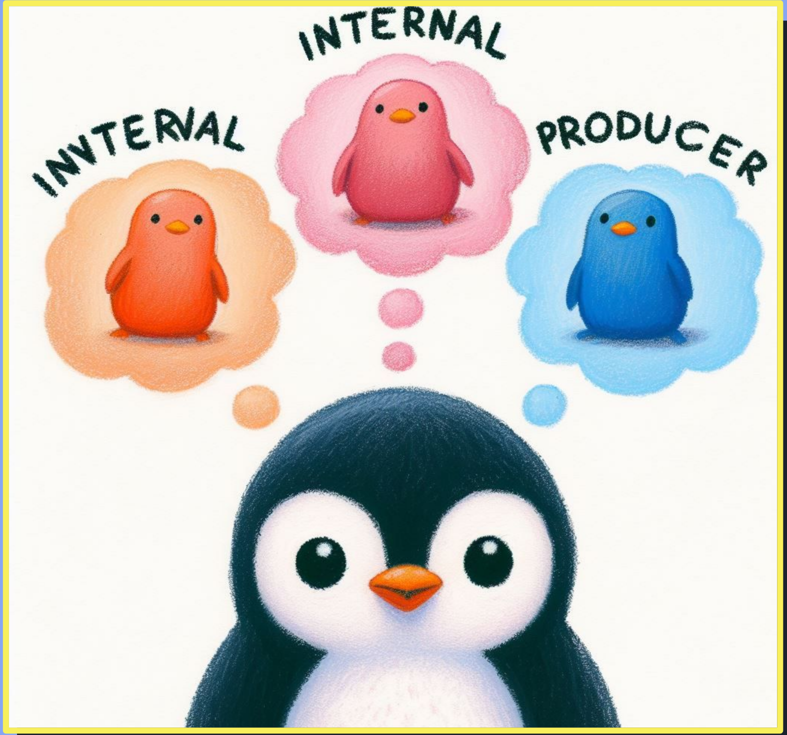
Logs (1)

2.21ms

event	This is a span event
key1	value1
key2	value2

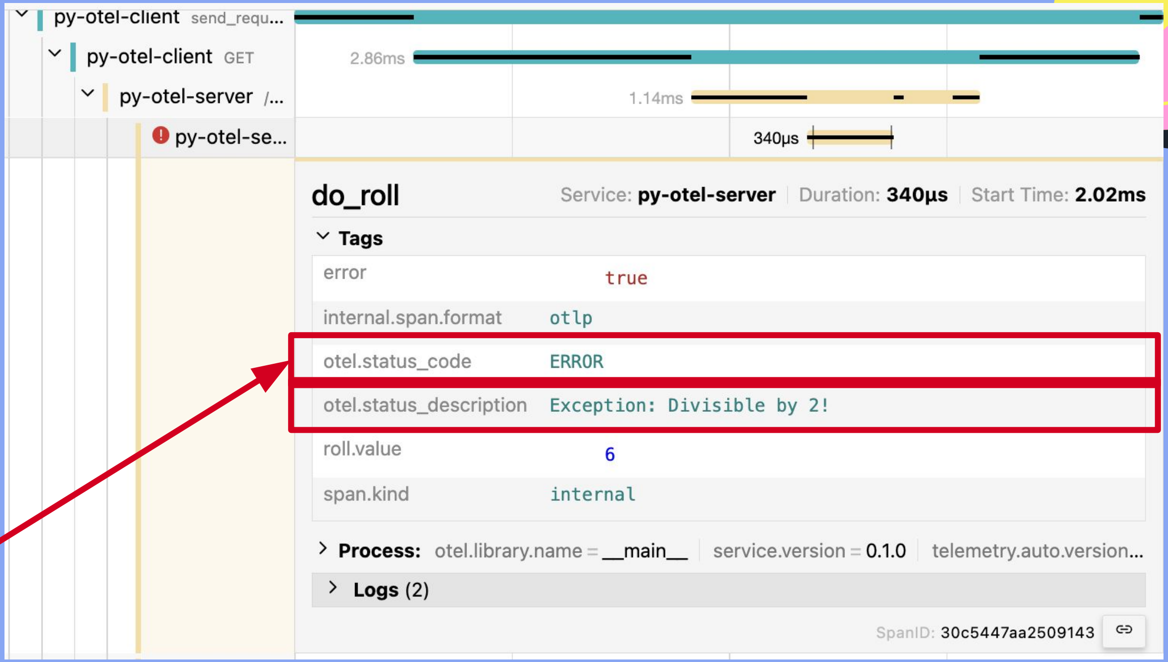
Log timestamps are relative to the start time of the full trace.

SpanID: 316c186d84adfbec



- Client
- Server
- Internal
- Producer
- Consumer

Enhancing spans with span status



py-otel-client send_requ...
py-otel-client GET 2.86ms
py-otel-server /... 1.14ms
py-otel-se... 340µs

do_roll Service: **py-otel-server** Duration: **340µs** Start Time: **2.02ms**

Tags

error	true
internal.span.format	otlp
otel.status_code	ERROR
otel.status_description	Exception: Divisible by 2!
roll.value	6
span.kind	internal

Process: otel.library.name = __main__ service.version = 0.1.0 telemetry.auto.version...

Logs (2)

SpanID: 30c5447aa2509143

span status

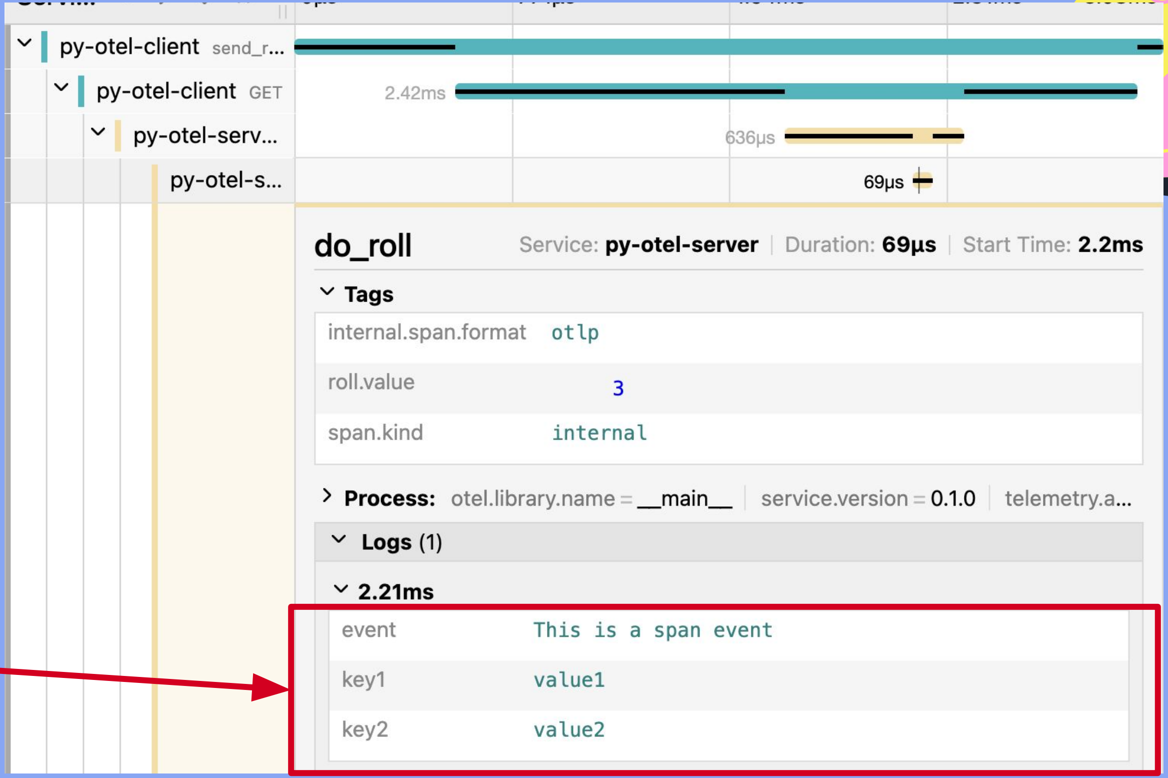
Unset (no status)

Error

OK (no error)

Enhancing spans with span events

span event



do_roll Service: **py-otel-server** | Duration: **69μs** | Start Time: **2.2ms**

Tags

internal.span.format	otlp
roll.value	3
span.kind	internal

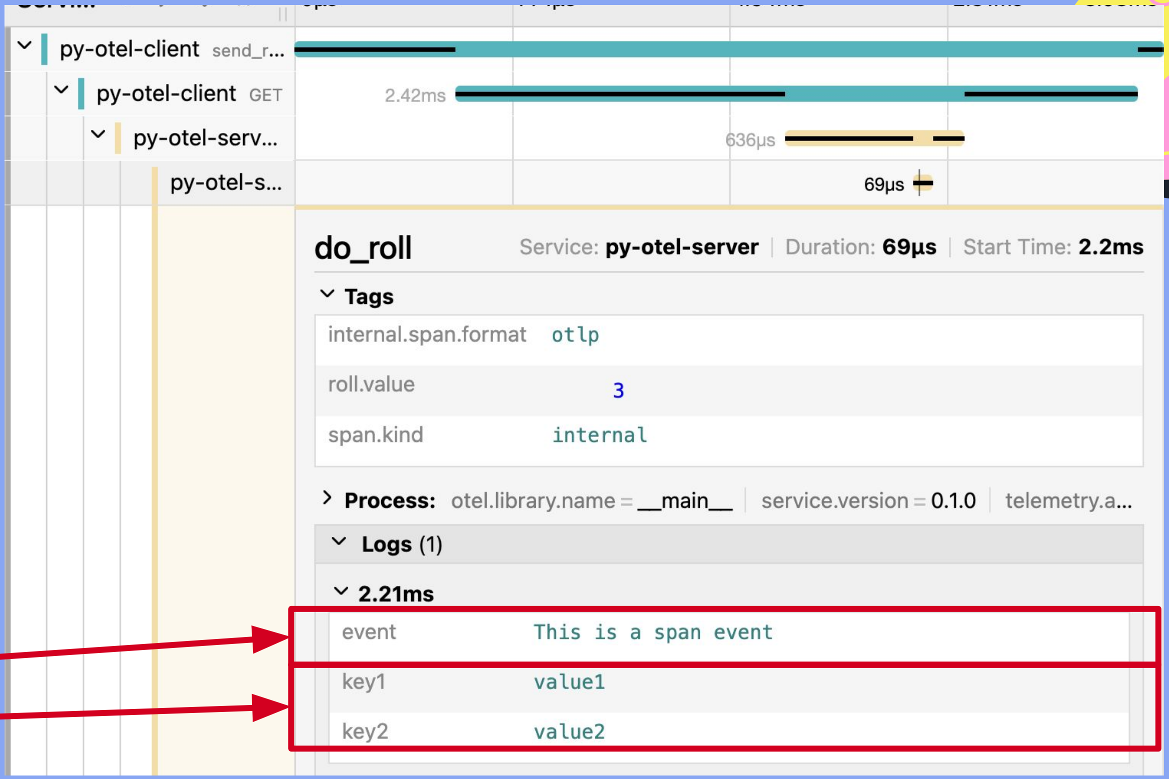
Process: otel.library.name = __main__ | service.version = 0.1.0 | telemetry.a...

Logs (1)

2.21ms

event	This is a span event
key1	value1
key2	value2

Enhancing spans with span events



do_roll Service: **py-otel-server** | Duration: **69μs** | Start Time: **2.2ms**

Tags

internal.span.format	otlp
roll.value	3
span.kind	internal

Process: otel.library.name = __main__ | service.version = 0.1.0 | telemetry.a...

Logs (1)

2.21ms

event	This is a span event
key1	value1
key2	value2

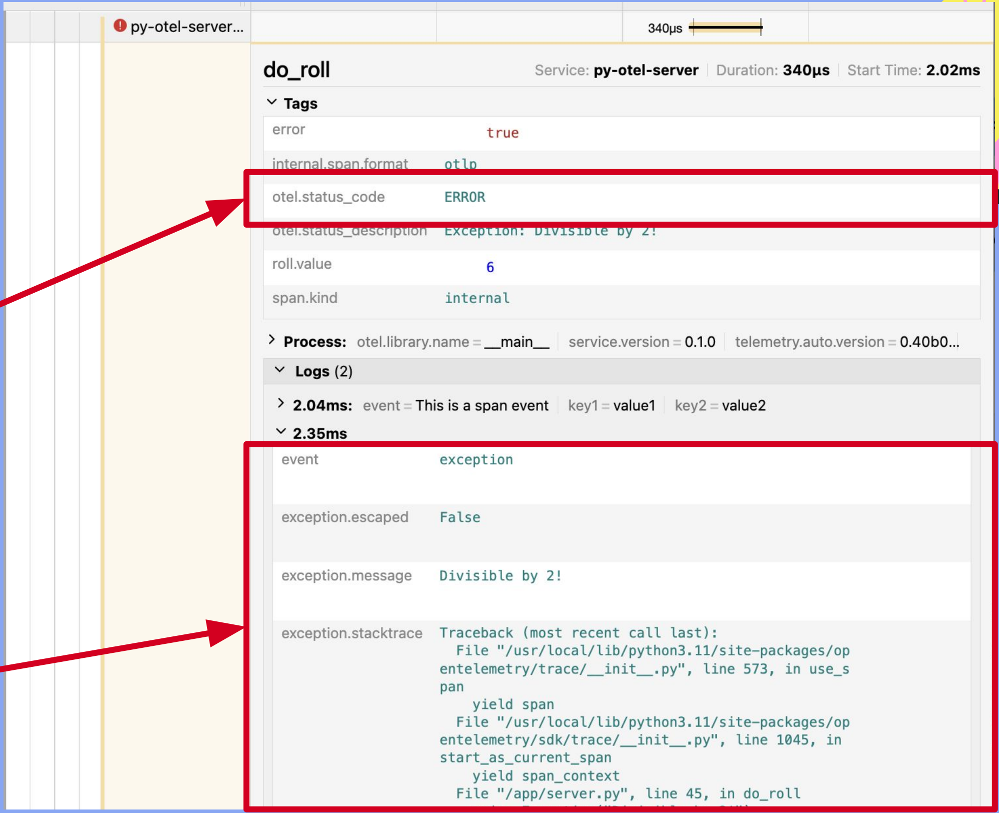
message

attributes

Enhancing spans with span events

error status

span event + attributes



py-otel-server...

340µs

```
do_roll                         Service: py-otel-server | Duration: 340µs | Start Time: 2.02ms
Tags
error                       true
internal.span.format        otlp
otel.status_code            ERROR
otel.status_description     Exception: Divisible by 2!
roll.value                  6
span.kind                   internal
> Process: otel.library.name = __main__   service.version = 0.1.0   telemetry.auto.version = 0.40b0...
Logs (2)
> 2.04ms: event = This is a span event   key1 = value1   key2 = value2
  2.35ms
event                  exception
exception.escaped      False
exception.message      Divisible by 2!
exception.stacktrace   Traceback (most recent call last):
                         File "/usr/local/lib/python3.11/site-packages/op
                       entelemetry/trace/__init__.py", line 573, in use_s
                       pan
                           yield span
                         File "/usr/local/lib/python3.11/site-packages/op
                       entelemetry/sdk/trace/__init__.py", line 1045, in
                       start_as_current_span
                           yield span_context
                         File "/app/server.py", line 45, in do_roll
```

RecordException
RecordError

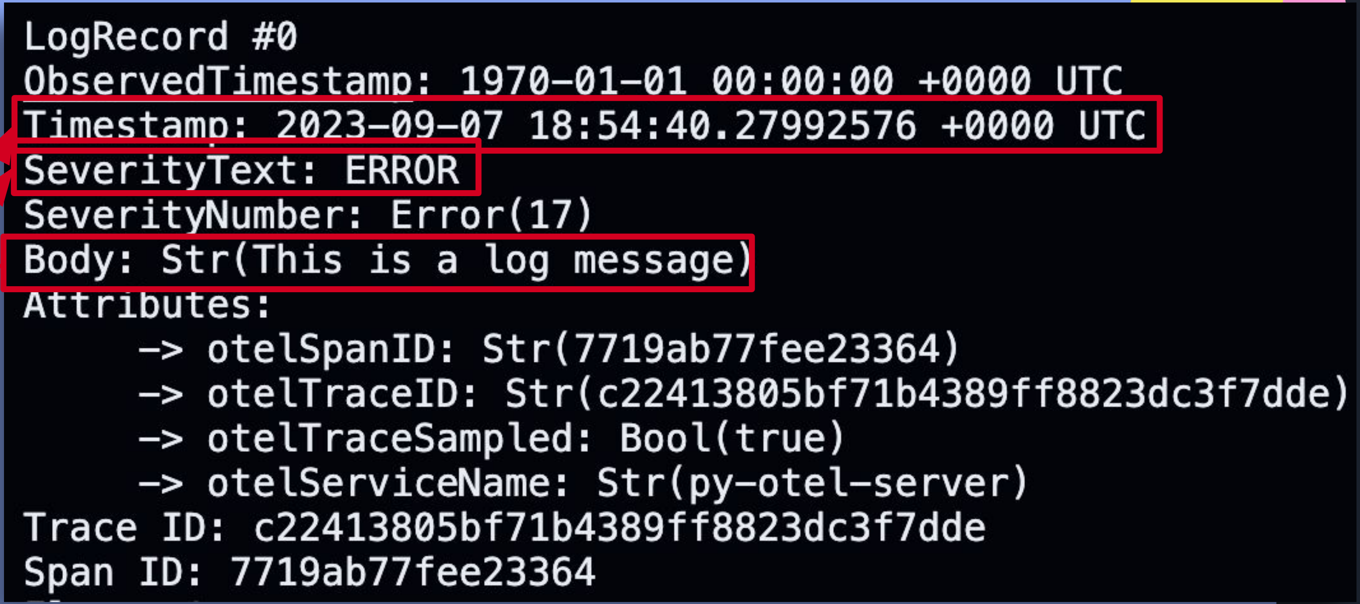
Set span status to `Error` if necessary

Can be used to record additional information

Errors in OTel Logs

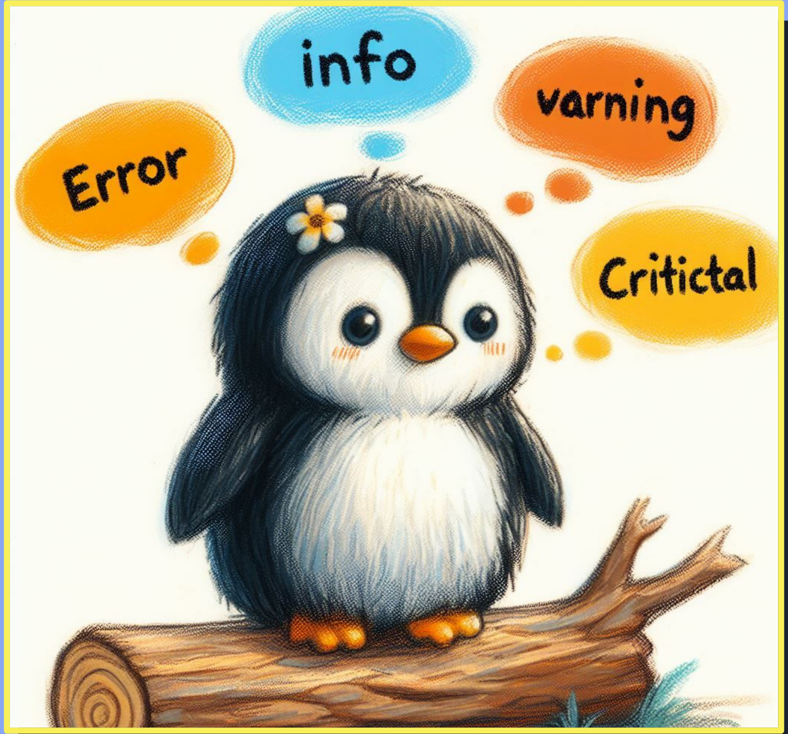
Another way to report errors!

Errors in OTel Logs



```
LogRecord #0
ObservedTimestamp: 1970-01-01 00:00:00 +0000 UTC
Timestamp: 2023-09-07 18:54:40.27992576 +0000 UTC
SeverityText: ERROR
SeverityNumber: Error(17)
Body: Str(This is a log message)
Attributes:
     -> otelSpanID: Str(7719ab77fee23364)
     -> otelTraceID: Str(c22413805bf71b4389ff8823dc3f7dde)
     -> otelTraceSampled: Bool(true)
     -> otelServiceName: Str(py-otel-server)
Trace ID: c22413805bf71b4389ff8823dc3f7dde
Span ID: 7719ab77fee23364
```

Message
Severity
Timestamp



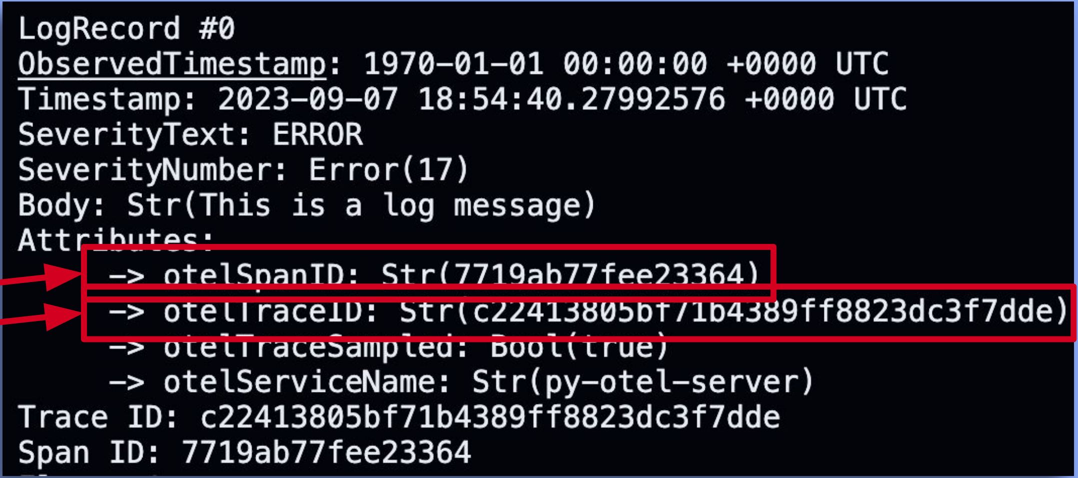
Debug

Info

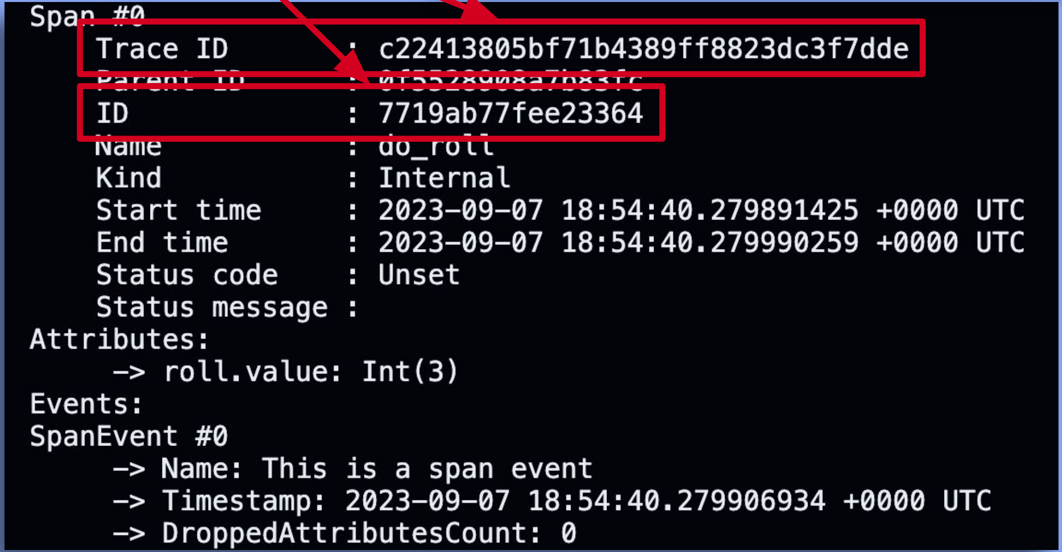
Warning

Error

Critical



```
LogRecord #0
ObservedTimestamp: 1970-01-01 00:00:00 +0000 UTC
Timestamp: 2023-09-07 18:54:40.27992576 +0000 UTC
SeverityText: ERROR
SeverityNumber: Error(17)
Body: Str(This is a log message)
Attributes:
     -> otelSpanID: Str(7719ab77fee23364)
     -> otelTraceID: Str(c22413805bf71b4389ff8823dc3f7dde)
     -> otelTraceSampled: Bool(true)
     -> otelServiceName: Str(py-otel-server)
Trace ID: c22413805bf71b4389ff8823dc3f7dde
Span ID: 7719ab77fee23364
```



```
Span #0
    Trace ID       : c22413805bf71b4389ff8823dc3f7dde
    Parent ID      : 0f5528908a7b83fc
    ID             : 7719ab77fee23364
    Name           : do_roll
    Kind           : Internal
    Start time     : 2023-09-07 18:54:40.279891425 +0000 UTC
    End time       : 2023-09-07 18:54:40.279990259 +0000 UTC
    Status code    : Unset
    Status message :
Attributes:
     -> roll.value: Int(3)
Events:
SpanEvent #0
     -> Name: This is a span event
     -> Timestamp: 2023-09-07 18:54:40.279906934 +0000 UTC
     -> DroppedAttributesCount: 0
```

Spans or Logs?

Is one better than the other?

OTel data vs APM agent data

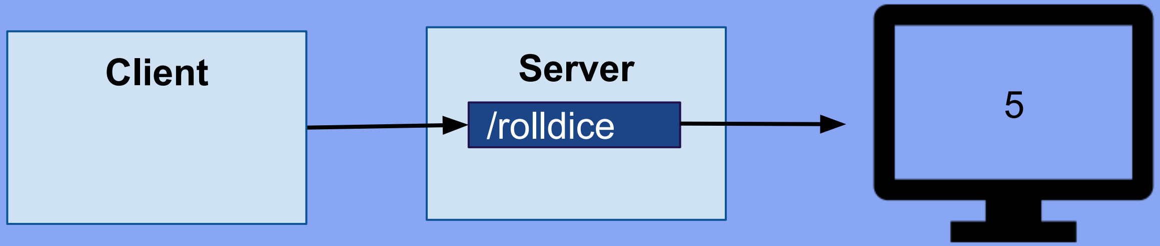
OTel models errors differently.

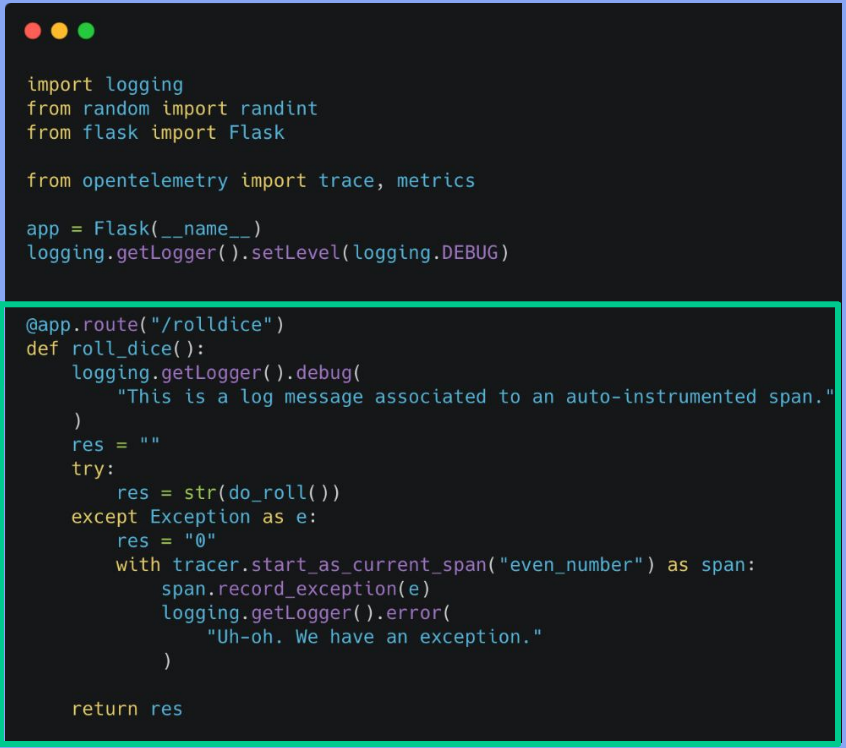
Transactions mean something slightly different.

Span kind impacts error rate reporting.

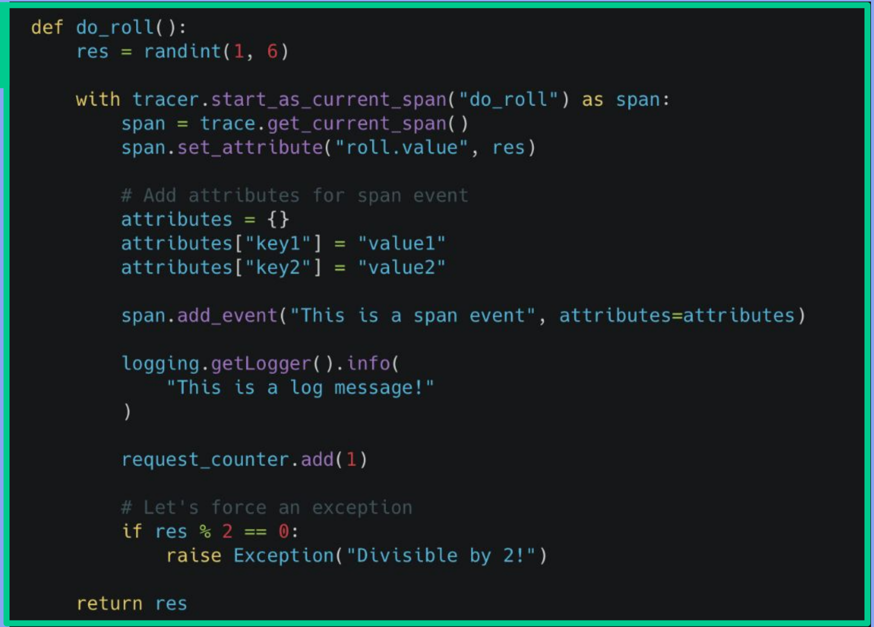
03

Show Me!

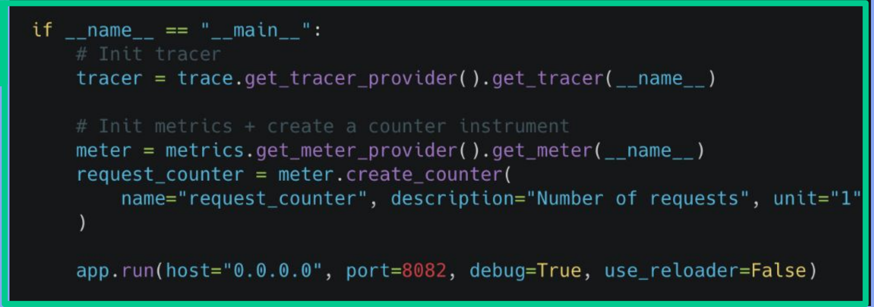

Client
Server
/rolldice
5



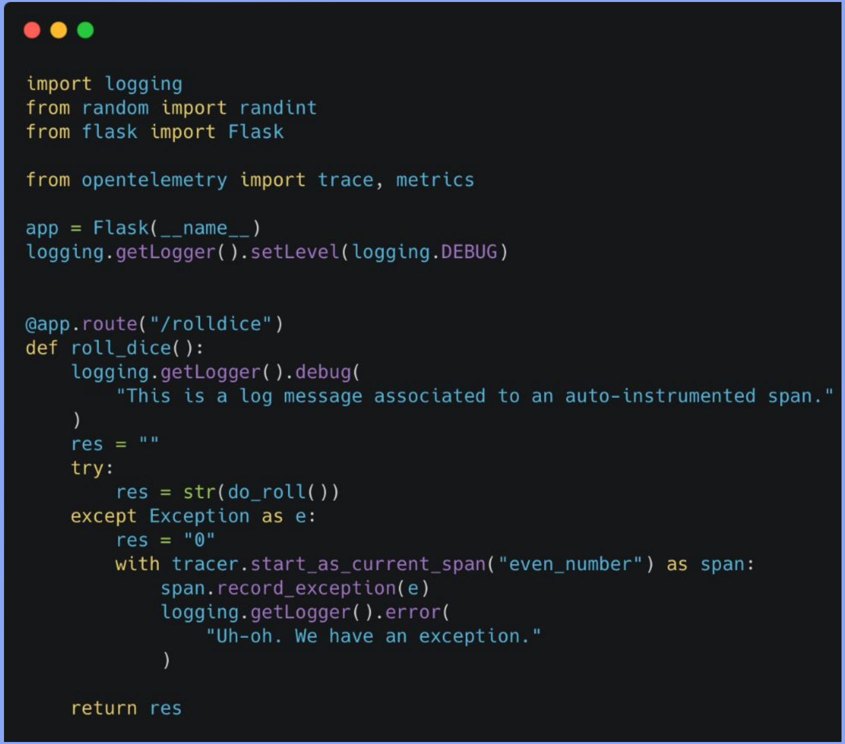
```python
import logging
from random import randint
from flask import Flask

from opentelemetry import trace, metrics

app = Flask(__name__)
logging.getLogger().setLevel(logging.DEBUG)


@app.route("/rolldice")
def roll_dice():
    logging.getLogger().debug(
        "This is a log message associated to an auto-instrumented span."
    )
    res = ""
    try:
        res = str(do_roll())
    except Exception as e:
        res = "0"
        with tracer.start_as_current_span("even_number") as span:
            span.record_exception(e)
            logging.getLogger().error(
                "Uh-oh. We have an exception."
            )

    return res
```



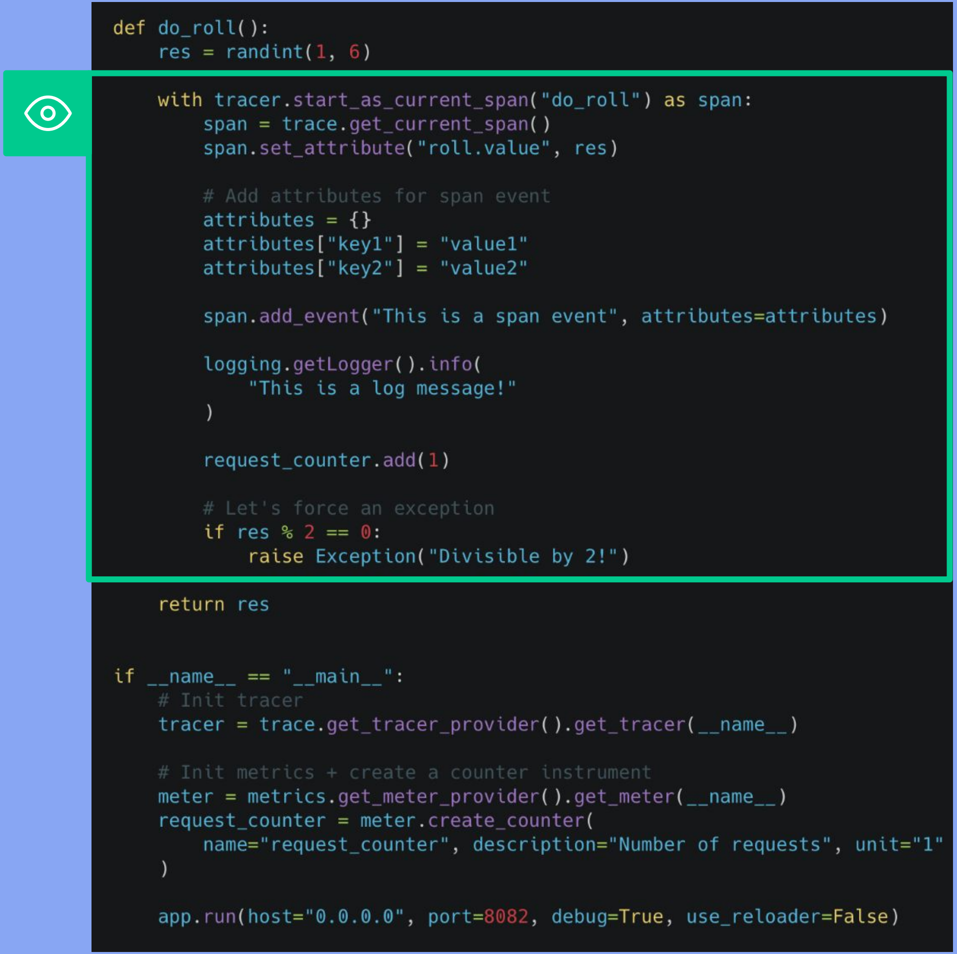
```python
def do_roll():
    res = randint(1, 6)

    with tracer.start_as_current_span("do_roll") as span:
        span = trace.get_current_span()
        span.set_attribute("roll.value", res)

        # Add attributes for span event
        attributes = {}
        attributes["key1"] = "value1"
        attributes["key2"] = "value2"

        span.add_event("This is a span event", attributes=attributes)

        logging.getLogger().info(
            "This is a log message!"
        )

        request_counter.add(1)

        # Let's force an exception
        if res % 2 == 0:
            raise Exception("Divisible by 2!")

    return res
```



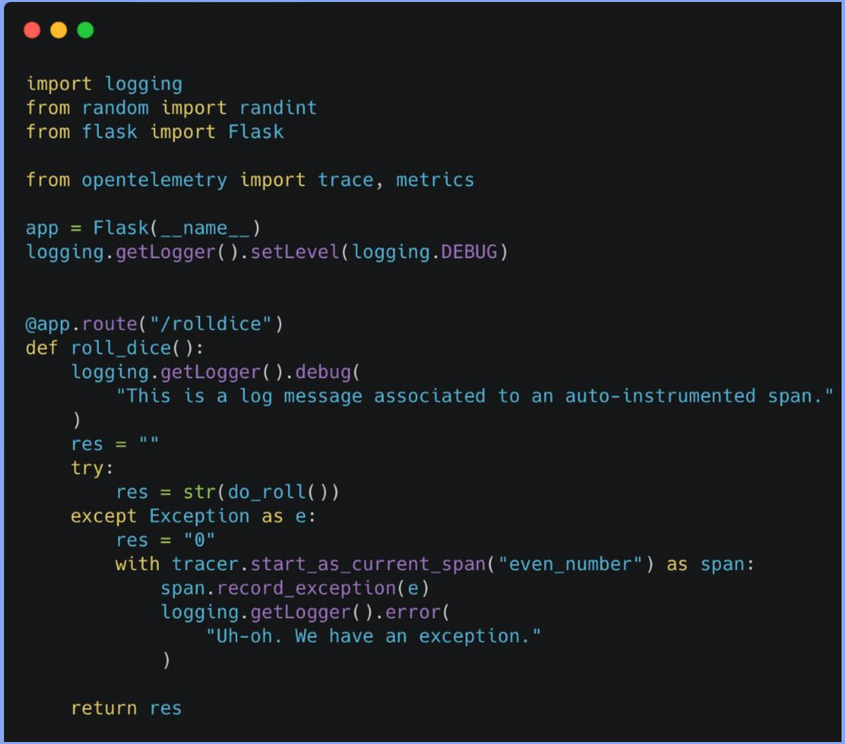
```python
if __name__ == "__main__":
    # Init tracer
    tracer = trace.get_tracer_provider().get_tracer(__name__)

    # Init metrics + create a counter instrument
    meter = metrics.get_meter_provider().get_meter(__name__)
    request_counter = meter.create_counter(
        name="request_counter", description="Number of requests", unit="1"
    )

    app.run(host="0.0.0.0", port=8082, debug=True, use_reloader=False)
```



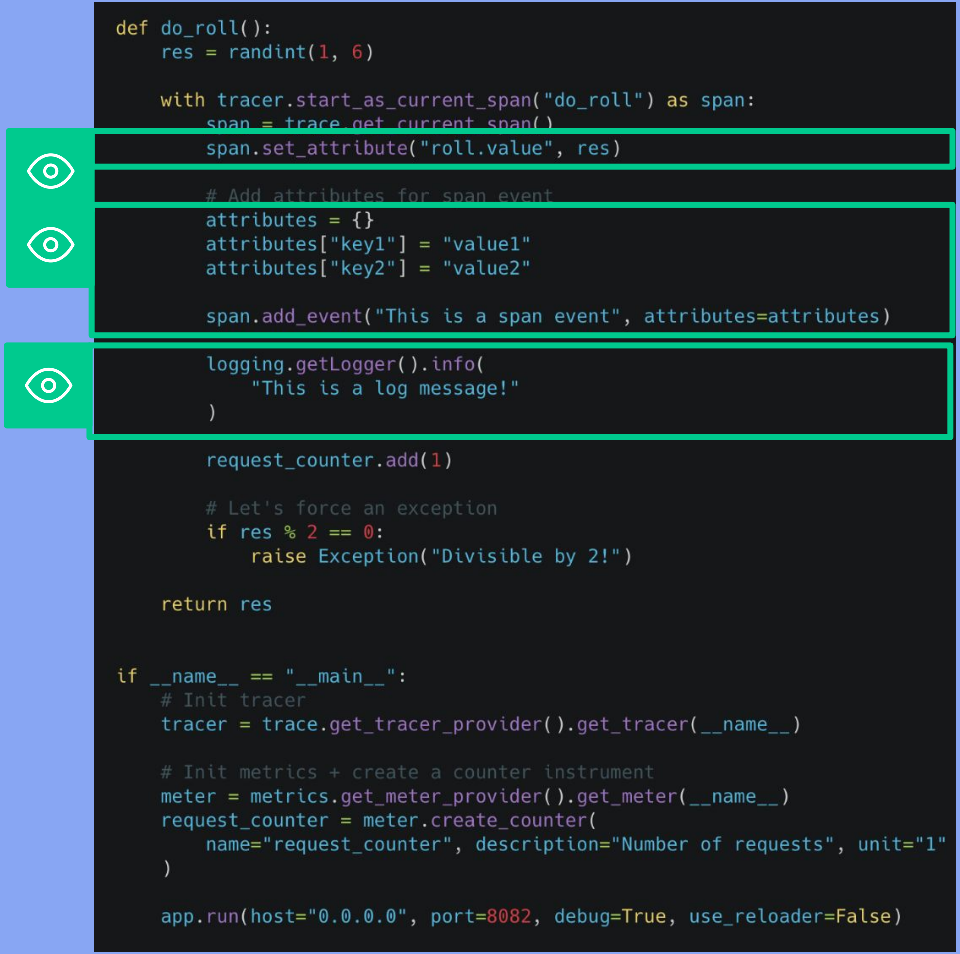
```python
import logging
from random import randint
from flask import Flask

from opentelemetry import trace, metrics

app = Flask(__name__)
logging.getLogger().setLevel(logging.DEBUG)


@app.route("/rolldice")
def roll_dice():
    logging.getLogger().debug(
        "This is a log message associated to an auto-instrumented span."
    )
    res = ""
    try:
        res = str(do_roll())
    except Exception as e:
        res = "0"
        with tracer.start_as_current_span("even_number") as span:
            span.record_exception(e)
            logging.getLogger().error(
                "Uh-oh. We have an exception."
            )

    return res
```



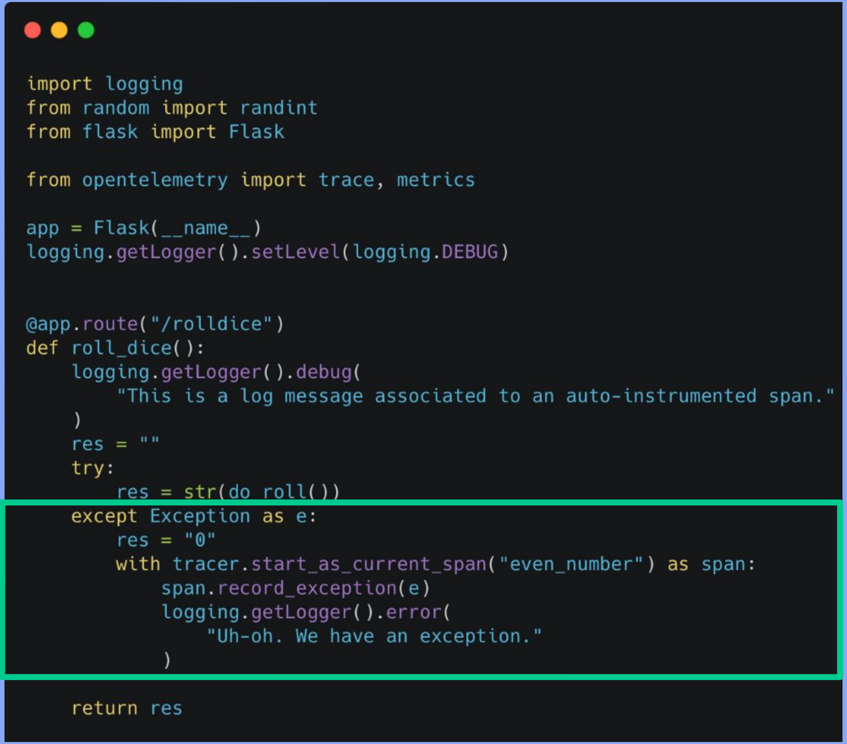
```python
def do_roll():
    res = randint(1, 6)

    with tracer.start_as_current_span("do_roll") as span:
        span = trace.get_current_span()
        span.set_attribute("roll.value", res)

        # Add attributes for span event
        attributes = {}
        attributes["key1"] = "value1"
        attributes["key2"] = "value2"

        span.add_event("This is a span event", attributes=attributes)

        logging.getLogger().info(
            "This is a log message!"
        )

        request_counter.add(1)

        # Let's force an exception
        if res % 2 == 0:
            raise Exception("Divisible by 2!")

    return res


if __name__ == "__main__":
    # Init tracer
    tracer = trace.get_tracer_provider().get_tracer(__name__)

    # Init metrics + create a counter instrument
    meter = metrics.get_meter_provider().get_meter(__name__)
    request_counter = meter.create_counter(
        name="request_counter", description="Number of requests", unit="1"
    )

    app.run(host="0.0.0.0", port=8082, debug=True, use_reloader=False)
```



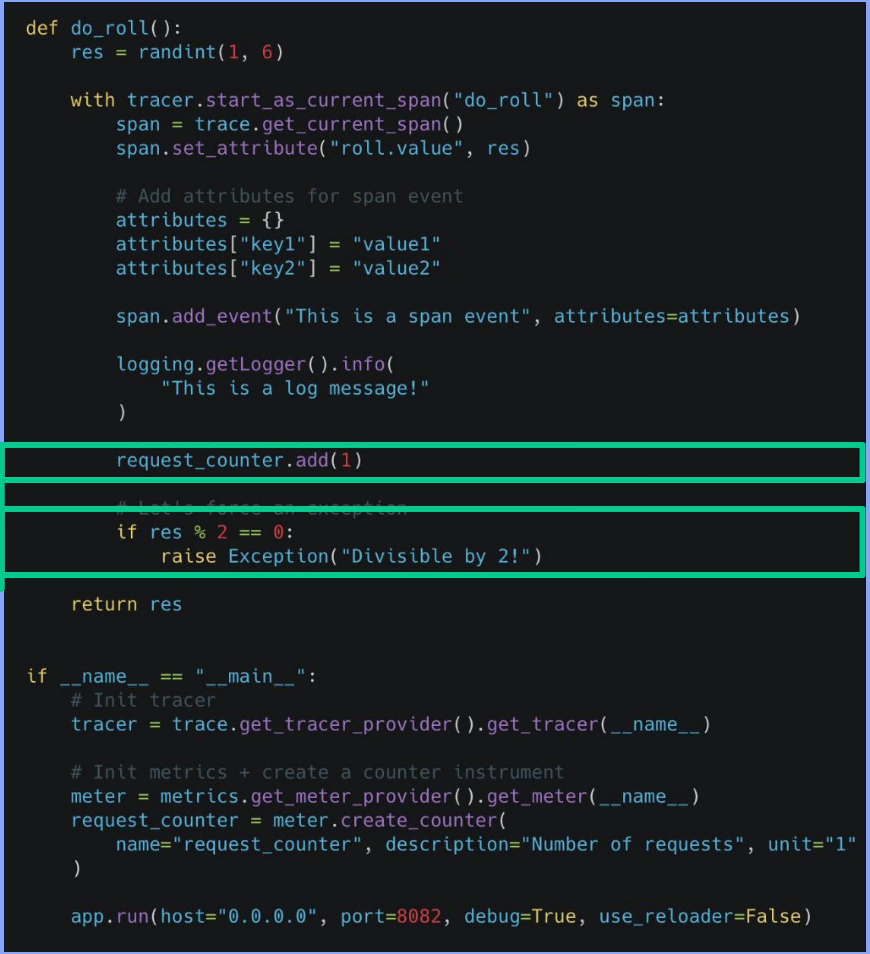
```python
import logging
from random import randint
from flask import Flask

from opentelemetry import trace, metrics

app = Flask(__name__)
logging.getLogger().setLevel(logging.DEBUG)


@app.route("/rolldice")
def roll_dice():
    logging.getLogger().debug(
        "This is a log message associated to an auto-instrumented span."
    )
    res = ""
    try:
        res = str(do_roll())
    except Exception as e:
        res = "0"
        with tracer.start_as_current_span("even_number") as span:
            span.record_exception(e)
            logging.getLogger().error(
                "Uh-oh. We have an exception."
            )

    return res
```



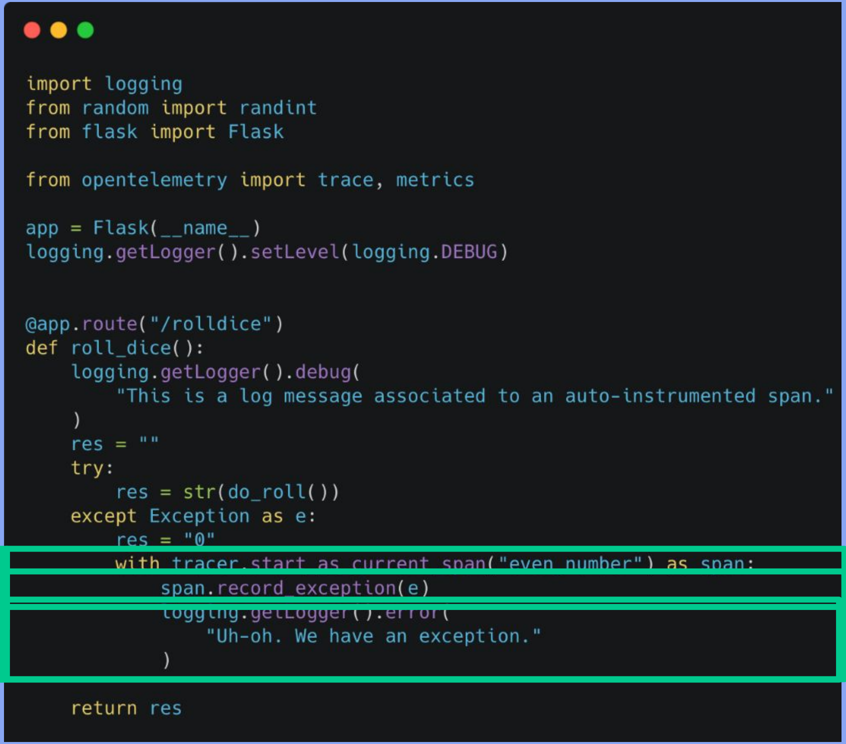
```python
def do_roll():
    res = randint(1, 6)

    with tracer.start_as_current_span("do_roll") as span:
        span = trace.get_current_span()
        span.set_attribute("roll.value", res)

        # Add attributes for span event
        attributes = {}
        attributes["key1"] = "value1"
        attributes["key2"] = "value2"

        span.add_event("This is a span event", attributes=attributes)

        logging.getLogger().info(
            "This is a log message!"
        )

        request_counter.add(1)

        # Let's force an exception
        if res % 2 == 0:
            raise Exception("Divisible by 2!")

    return res


if __name__ == "__main__":
    # Init tracer
    tracer = trace.get_tracer_provider().get_tracer(__name__)

    # Init metrics + create a counter instrument
    meter = metrics.get_meter_provider().get_meter(__name__)
    request_counter = meter.create_counter(
        name="request_counter", description="Number of requests", unit="1"
    )

    app.run(host="0.0.0.0", port=8082, debug=True, use_reloader=False)
```



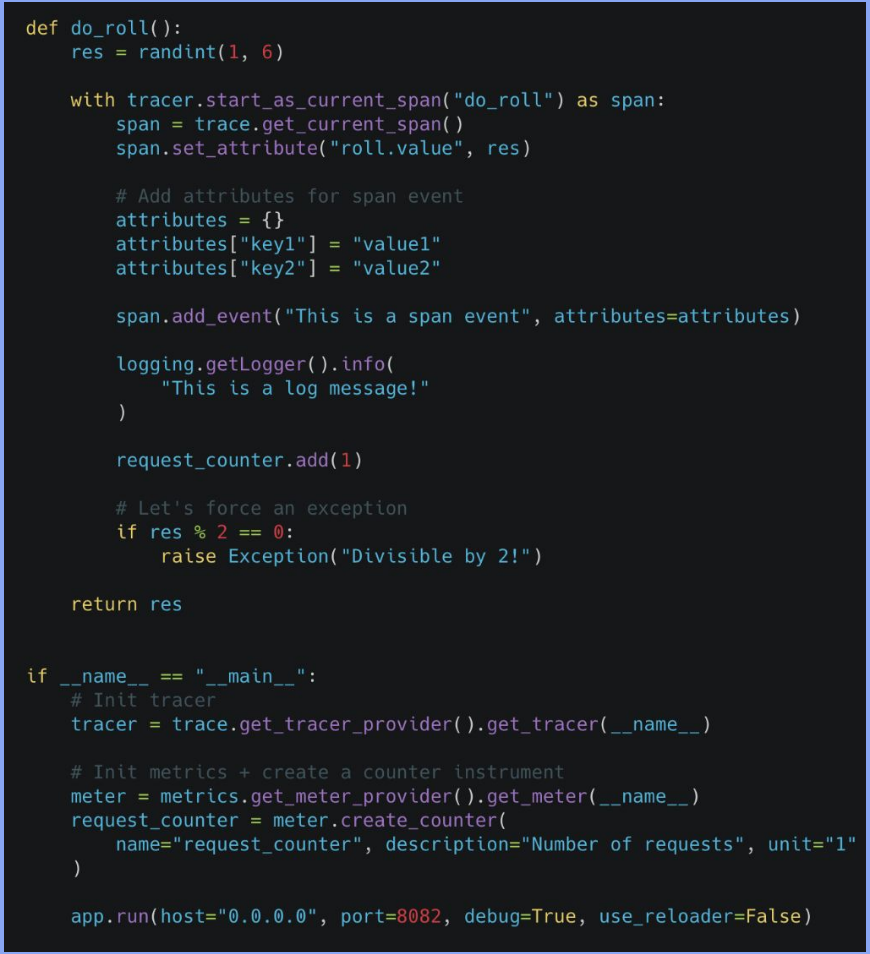
```python
import logging
from random import randint
from flask import Flask

from opentelemetry import trace, metrics

app = Flask(__name__)
logging.getLogger().setLevel(logging.DEBUG)


@app.route("/rolldice")
def roll_dice():
    logging.getLogger().debug(
        "This is a log message associated to an auto-instrumented span."
    )
    res = ""
    try:
        res = str(do_roll())
    except Exception as e:
        res = "0"
        with tracer.start_as_current_span("even_number") as span:
            span.record_exception(e)
            logging.getLogger().error(
                "Uh-oh. We have an exception."
            )

    return res
```



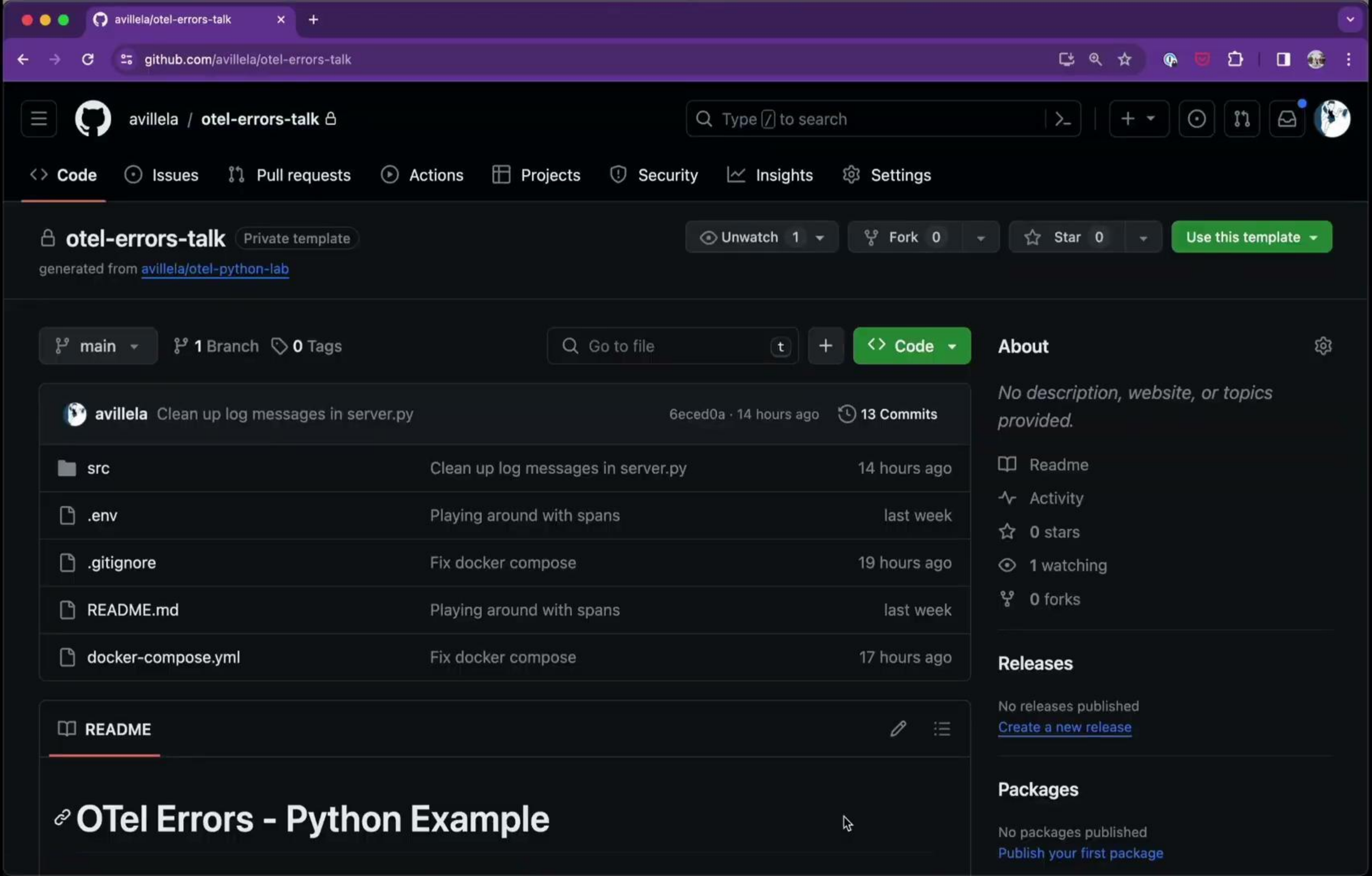
```python
def do_roll():
    res = randint(1, 6)

    with tracer.start_as_current_span("do_roll") as span:
        span = trace.get_current_span()
        span.set_attribute("roll.value", res)

        # Add attributes for span event
        attributes = {}
        attributes["key1"] = "value1"
        attributes["key2"] = "value2"

        span.add_event("This is a span event", attributes=attributes)

        logging.getLogger().info(
            "This is a log message!"
        )

        request_counter.add(1)

        # Let's force an exception
        if res % 2 == 0:
            raise Exception("Divisible by 2!")

    return res


if __name__ == "__main__":
    # Init tracer
    tracer = trace.get_tracer_provider().get_tracer(__name__)

    # Init metrics + create a counter instrument
    meter = metrics.get_meter_provider().get_meter(__name__)
    request_counter = meter.create_counter(
        name="request_counter", description="Number of requests", unit="1"
    )

    app.run(host="0.0.0.0", port=8082, debug=True, use_reloader=False)
```



```python
import logging
from random import randint
from flask import Flask

from opentelemetry import trace, metrics

app = Flask(__name__)
logging.getLogger().setLevel(logging.DEBUG)


@app.route("/rolldice")
def roll_dice():
    logging.getLogger().debug(
        "This is a log message associated to an auto-instrumented span."
    )
    res = ""
    try:
        res = str(do_roll())
    except Exception as e:
        res = "0"
        with tracer.start_as_current_span("even_number") as span:
            span.record_exception(e)
            logging.getLogger().error(
                "Uh-oh. We have an exception."
            )

    return res
```



```python
def do_roll():
    res = randint(1, 6)

    with tracer.start_as_current_span("do_roll") as span:
        span = trace.get_current_span()
        span.set_attribute("roll.value", res)

        # Add attributes for span event
        attributes = {}
        attributes["key1"] = "value1"
        attributes["key2"] = "value2"

        span.add_event("This is a span event", attributes=attributes)

        logging.getLogger().info(
            "This is a log message!"
        )

        request_counter.add(1)

        # Let's force an exception
        if res % 2 == 0:
            raise Exception("Divisible by 2!")

    return res


if __name__ == "__main__":
    # Init tracer
    tracer = trace.get_tracer_provider().get_tracer(__name__)

    # Init metrics + create a counter instrument
    meter = metrics.get_meter_provider().get_meter(__name__)
    request_counter = meter.create_counter(
        name="request_counter", description="Number of requests", unit="1"
    )

    app.run(host="0.0.0.0", port=8082, debug=True, use_reloader=False)
```



avillela/otel-errors-talk

github.com/avillela/otel-errors-talk

avillela / otel-errors-talk

Type / to search

Code Issues Pull requests Actions Projects Security Insights Settings

otel-errors-talk Private template

generated from avillela/otel-python-lab

Unwatch 1 Fork 0 Star 0 Use this template

main 1 Branch 0 Tags Go to file t + Code

avillela Clean up log messages in server.py 6eced0a · 14 hours ago 13 Commits

src	Clean up log messages in server.py	14 hours ago
.env	Playing around with spans	last week
.gitignore	Fix docker compose	19 hours ago
README.md	Playing around with spans	last week
docker-compose.yml	Fix docker compose	17 hours ago

README

OTel Errors - Python Example

About

No description, website, or topics provided.

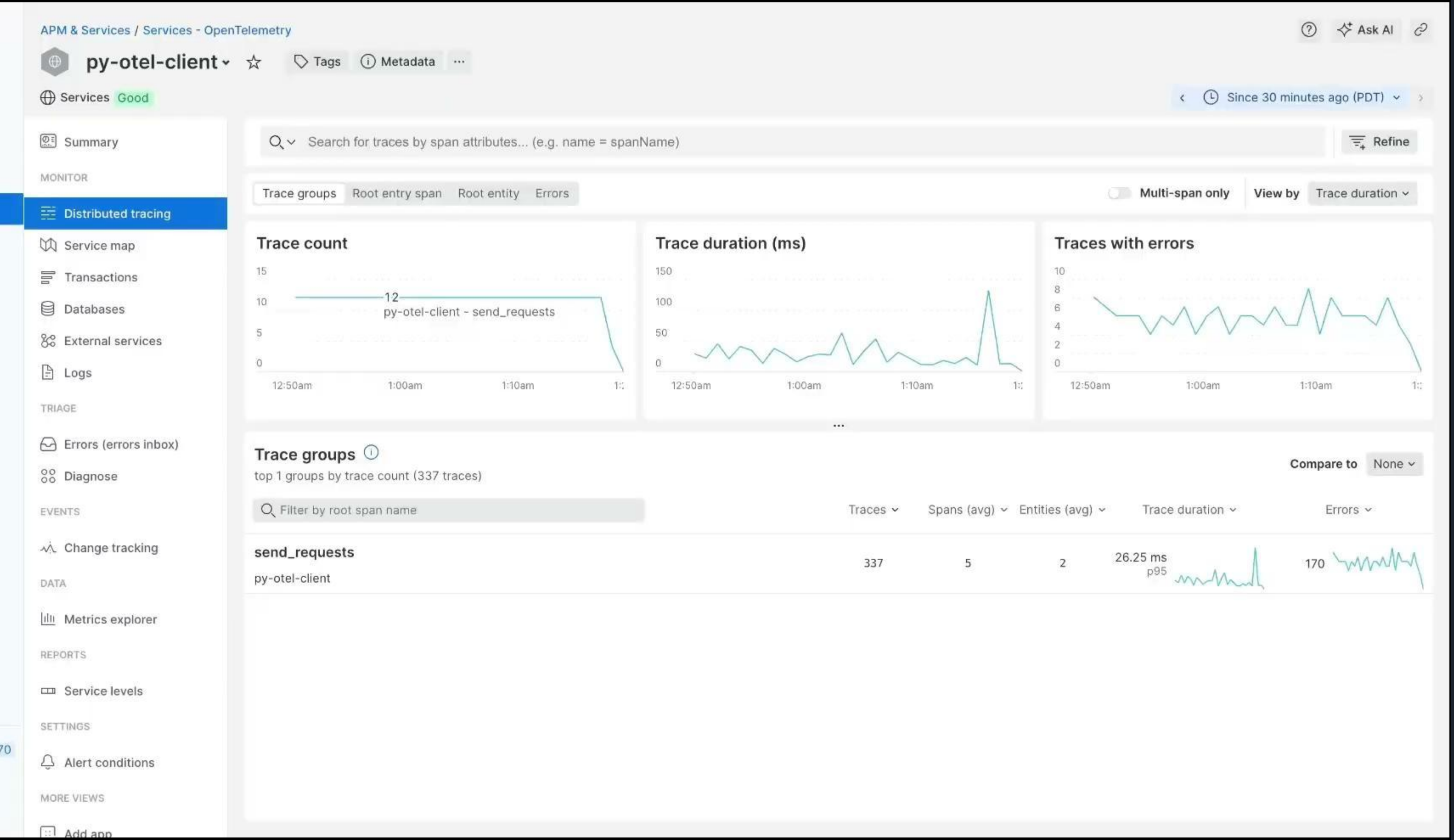
Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package



APM & Services / Services - OpenTelemetry

py-otel-client

Tags | Metadata

Services Good

Since 30 minutes ago (PDT)

Summary

MONITOR

Distributed tracing

Service map

Transactions

Databases

External services

Logs

TRIAGE

Errors (errors inbox)

Diagnose

EVENTS

Change tracking

DATA

Metrics explorer

REPORTS

Service levels

SETTINGS

Alert conditions

MORE VIEWS

Add app

Search for traces by span attributes... (e.g. name = spanName)

Refine

Trace groups | Root entry span | Root entity | Errors

Multi-span only

View by: Trace duration

Trace groups

top 1 groups by trace count (337 traces)

Compare to: None

Filter by root span name

	Traces	Spans (avg)	Entities (avg)	Trace duration	Errors
send_requests py-otel-client	337	5	2	26.25 ms p95	170

04

Wrap-up

In summary

- Error handling is challenging
- OTel can help!
- Record errors through spans and logs - correlation!
- Enhance spans with metadata & span events
- Data visualization differences

Not all Images were created by humans

Thanks, DALL·E3!

Also check out…

Geeking Out Podcast

Cool tech topics! (Guests include Kelsey Hightower)

Handy resources

Errors Example Repo on GitHub

The inspirational post

Dude, Where's My Error Blog Post

OpenTelemetry Spec Compliance Matrix

Thanks!