



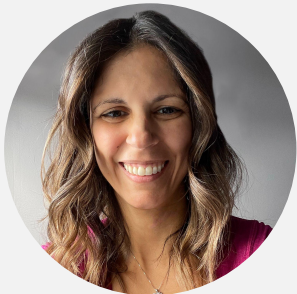
Dude, Where's My **ERROR?**



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/adrianavillela

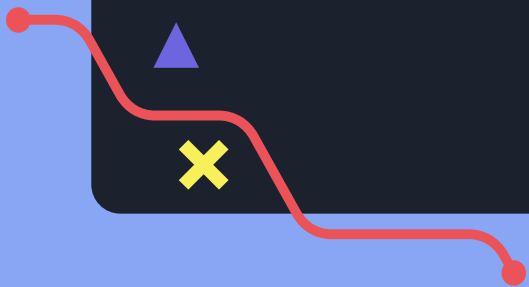



Reese Lee

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'.”

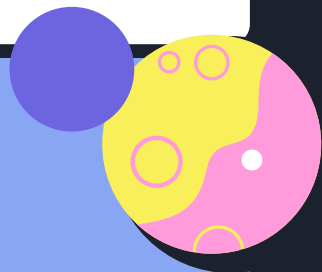
Dude, where's
my **ERROR?**



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



Table of Contents



01

Background

03

Show Me!

02

**Error handling
in OTel**

04

Wrap-up



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?

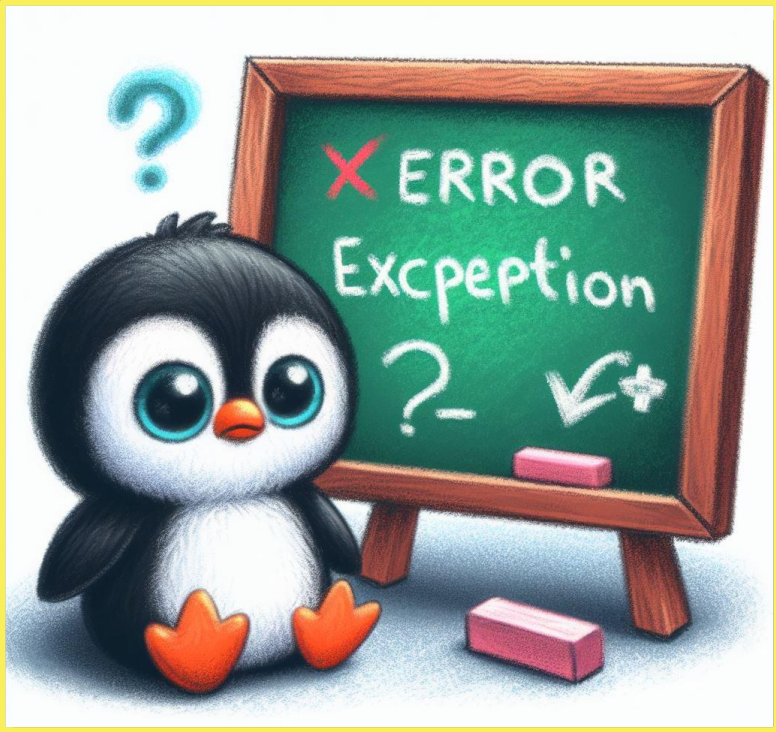




OpenTelemetry Refresher

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

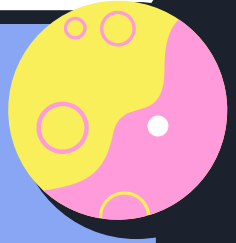




Errors vs Exceptions

What's the difference?





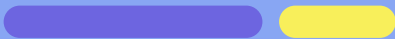
Errors vs Exceptions

Error

An unexpected issue in a program that hinders its execution.

Exception

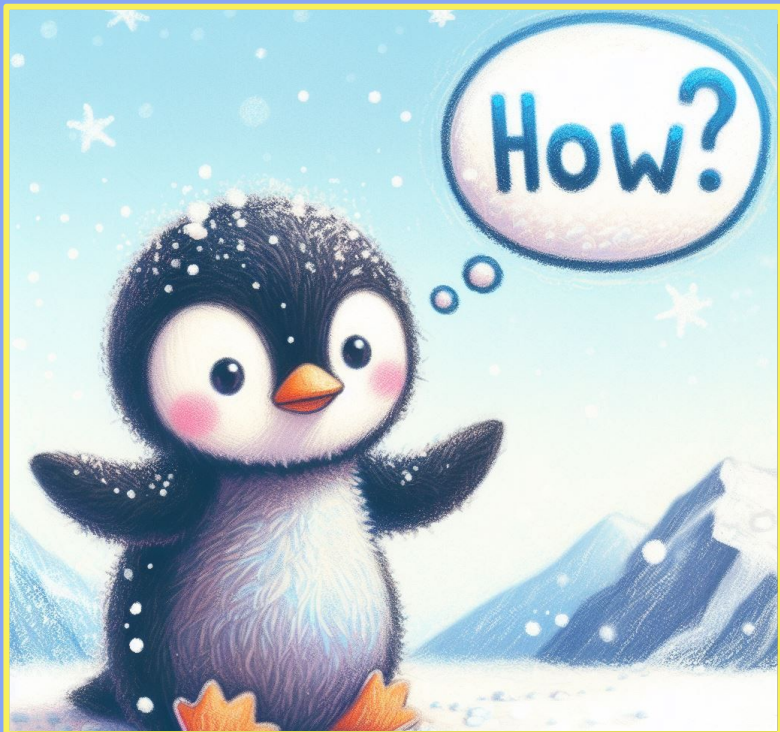
A type of runtime error that disrupts the normal flow of a program.



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



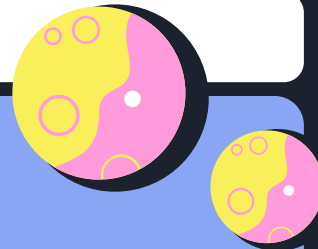
An OTel Span

Trace

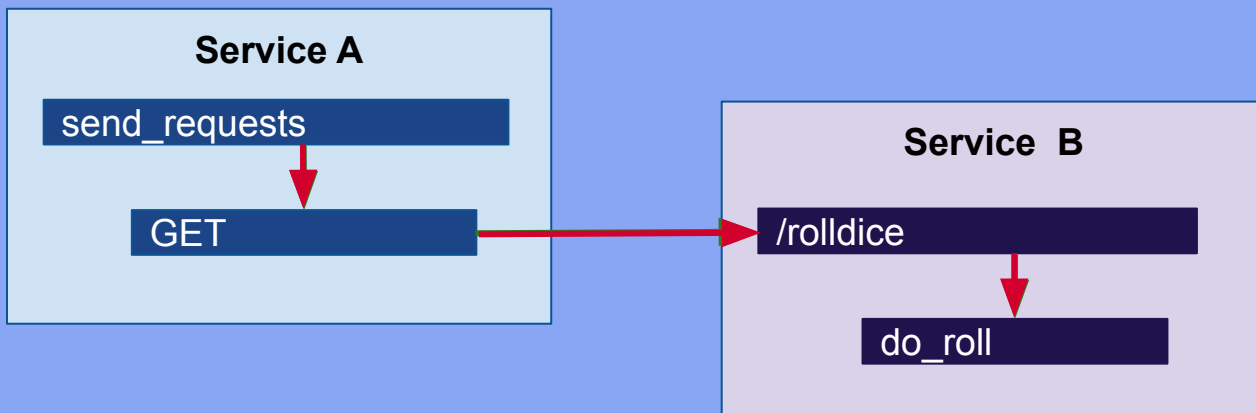


Root Span

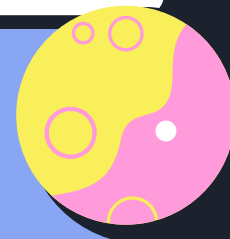
Spans



OpenTelemetry Trace Context

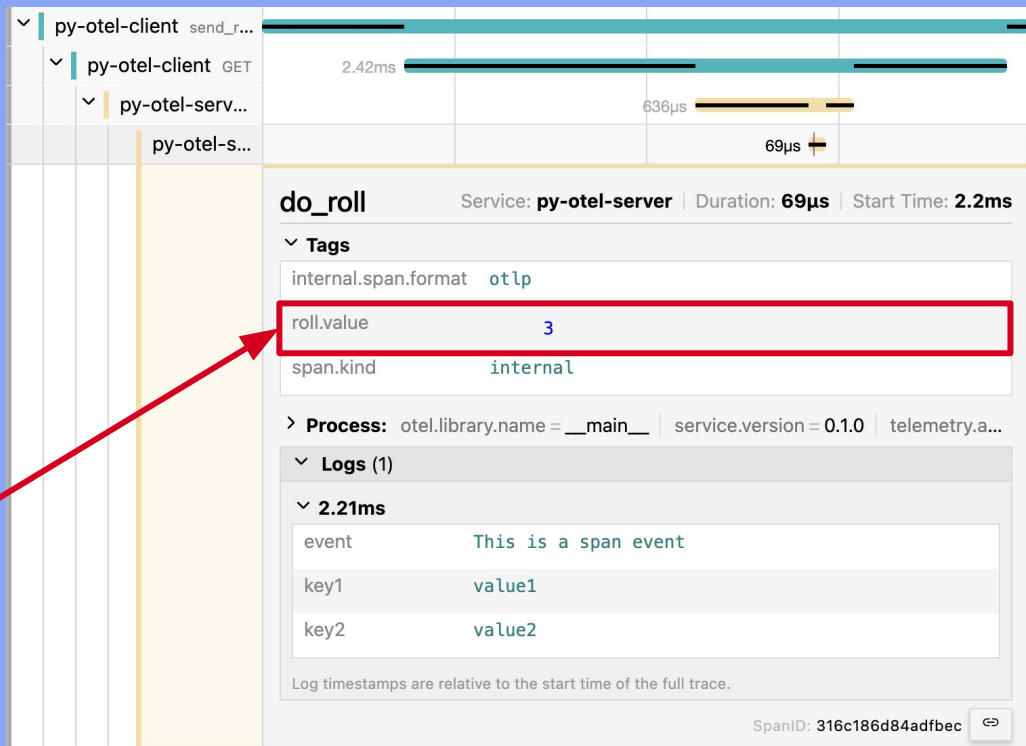


Context propagation

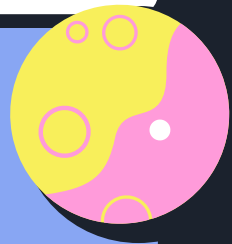


Enhancing spans with metadata

Attribute
(metadata)

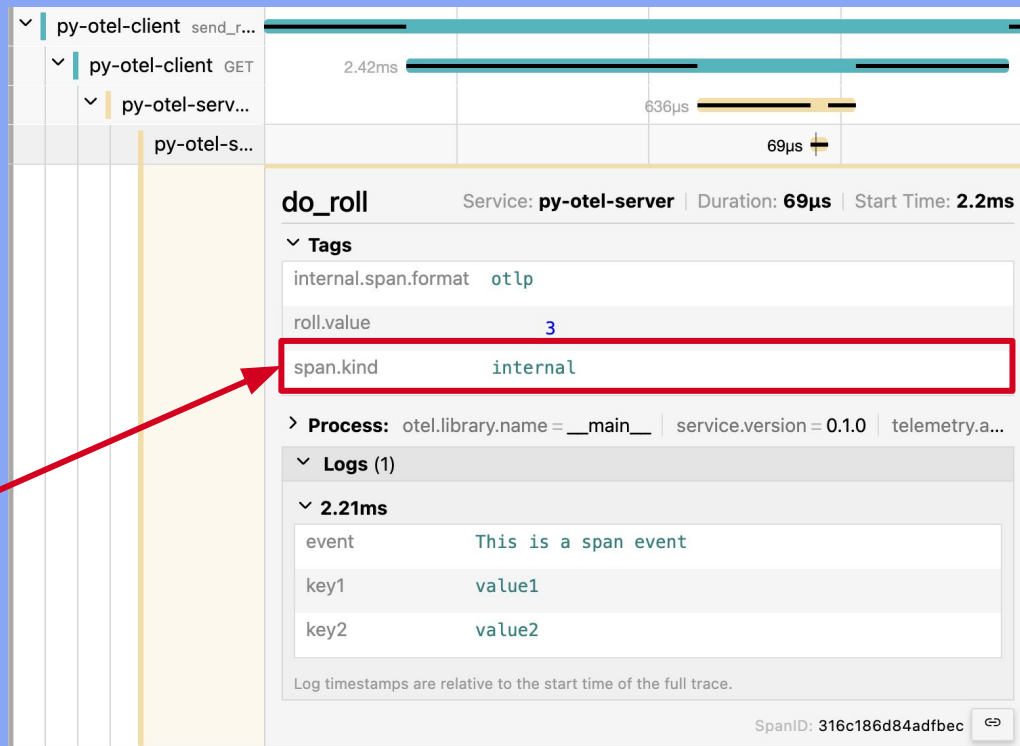


The screenshot displays a distributed tracing interface. The top section shows a tree view of spans: `py-otel-client` (send_r...) with a duration of 2.42ms, `py-otel-client` (GET) with a duration of 636µs, and `py-otel-serv...` (py-otel-s...) with a duration of 69µs. The selected span is `do_roll` from `Service: py-otel-server`, with a duration of 69µs and a start time of 2.2ms. Under the `Tags` section, the `roll.value` attribute is highlighted with a red box and has a value of 3. Other tags include `internal.span.format` (otlp) and `span.kind` (internal). The `Process` section shows `otel.library.name = __main__`, `service.version = 0.1.0`, and `telemetry.a...`. The `Logs (1)` section shows a log at 2.21ms with the event `This is a span event` and two key-value pairs: `key1 = value1` and `key2 = value2`. A red arrow points from the text 'Attribute (metadata)' to the `roll.value` attribute. The bottom right corner shows the `SpanID: 316c186d84adfbec`.



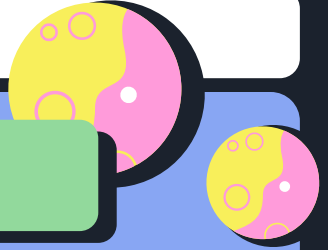
Enhancing spans with span kind

span kind



The screenshot displays a distributed tracing interface. The top section shows a tree view of spans: `py-otel-client send_r...` (2.42ms), `py-otel-client GET` (636µs), and `py-otel-s...` (69µs). The selected span is `do_roll` from service `py-otel-server`, with a duration of 69µs and a start time of 2.2ms. The `Tags` section includes `internal.span.format` (otlp), `roll.value` (3), and `span.kind` (internal). The `Process` section shows `otel.library.name = __main__`, `service.version = 0.1.0`, and `telemetry.a...`. The `Logs (1)` section shows a log at 2.21ms: `event This is a span event`, with `key1 value1` and `key2 value2`. The `SpanID` is 316c186d84adfbec.





Client

Server

Internal

Producer

Consumer

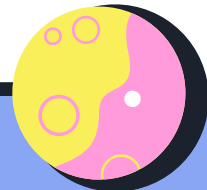
Enhancing spans with span status

span status

The screenshot displays a distributed tracing interface. On the left, a tree view shows a span hierarchy: `py-otel-client send_requ...` (2.86ms), `py-otel-client GET` (1.14ms), and `py-otel-server /...` (340µs). The `py-otel-se...` span is highlighted in yellow and has a red error icon. A red arrow points from the text 'span status' to this span. The right pane shows the details for the `do_roll` span (Service: `py-otel-server`, Duration: `340µs`, Start Time: `2.02ms`). Under the 'Tags' section, the following tags are listed:

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

Below the tags, there are sections for 'Process' (otel.library.name = `__main__`, service.version = `0.1.0`, telemetry.auto.version...) and 'Logs (2)'. The SpanID is `30c5447aa2509143`.



Unset (no status)

Error

OK (no error)

Enhancing spans with span events

span event

The screenshot displays a tracing interface with a tree view on the left and a detailed view on the right. The tree view shows a hierarchy of spans: `py-otel-client` (send_r...), `py-otel-client` (GET, 2.42ms), `py-otel-serv...` (636µs), and `py-otel-s...` (69µs). The detailed view for the `do_roll` span (Service: `py-otel-server`, Duration: `69µs`, Start Time: `2.2ms`) includes the following sections:

- 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`

A red arrow points from the text "span event" to the event entry in the logs section.

Enhancing spans with span events

message attributes

The screenshot displays a tracing interface with a tree view on the left and a detailed view on the right. The tree view shows a hierarchy of spans: 'py-otel-client send_r...', 'py-otel-client GET' (2.42ms), 'py-otel-serv...' (636µs), and 'py-otel-s...' (69µs). The detailed view for the 'do_roll' span (Service: py-otel-server, Duration: 69µs, Start Time: 2.2ms) includes the following sections:

- 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

Red arrows point from the text 'message attributes' to the 'event', 'key1', and 'key2' entries in the logs section, which are enclosed in a red rectangular box.

Enhancing spans with span events

error status

span event + attributes

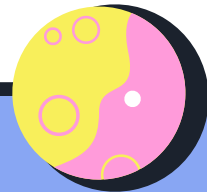
The screenshot displays a web browser window with a tab titled 'py-otel-server...'. The main content shows a detailed view of a span named 'do_roll' with a duration of 340µs and a start time of 2.02ms. The span is categorized as an 'internal' kind. A red box highlights the 'otel.status_code' attribute, which is set to 'ERROR'. Below this, the 'otel.status_description' is shown as 'Exception: Divisible by 2!'. The 'Process' section indicates the library name is '__main__', service version is '0.1.0', and telemetry auto version is '0.40b0...'. The 'Logs (2)' section shows two log entries: one at 2.04ms with the message 'event = This is a span event' and key-value pairs 'key1 = value1' and 'key2 = value2'; and another at 2.35ms with the event 'exception'. A second red box highlights the 'exception' log entry, which includes attributes for 'exception.escaped' (False), 'exception.message' ('Divisible by 2!'), and 'exception.stacktrace'. The stacktrace shows the error occurred in the 'do_roll' function at line 45 of 'app/server.py'.

```
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/opentelemetry/trace/__init__.py", line 573, in use_span
      yield span
    File "/usr/local/lib/python3.11/site-packages/opentelemetry/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



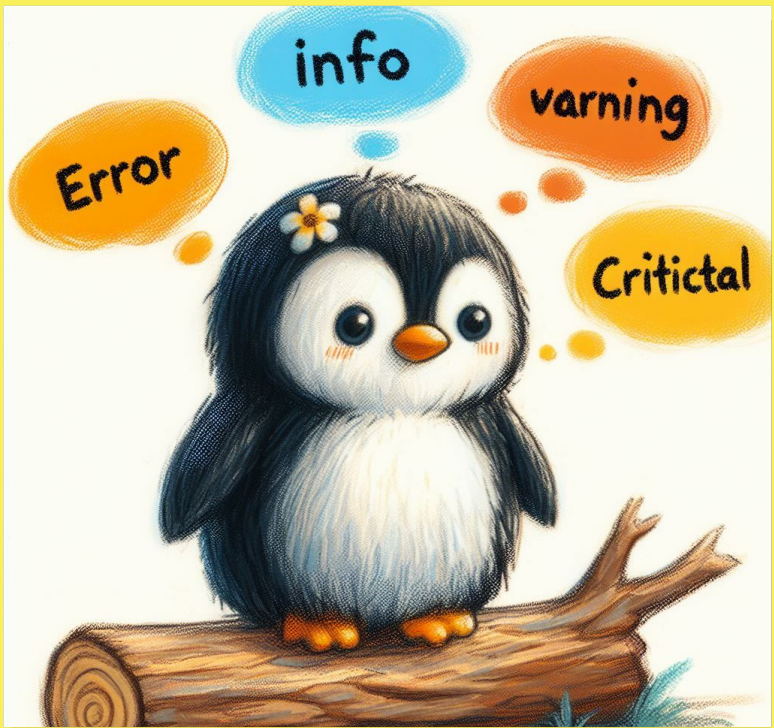
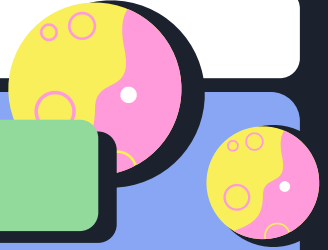
Errors in OTel Logs

Another way to report errors!

Errors in OTel Logs

Message
Severity
Timestamp

```
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
```



Debug

Info

Warning

Error

Critical

Same Span
Same Trace

```
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    : 015528908a7b83fc
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.

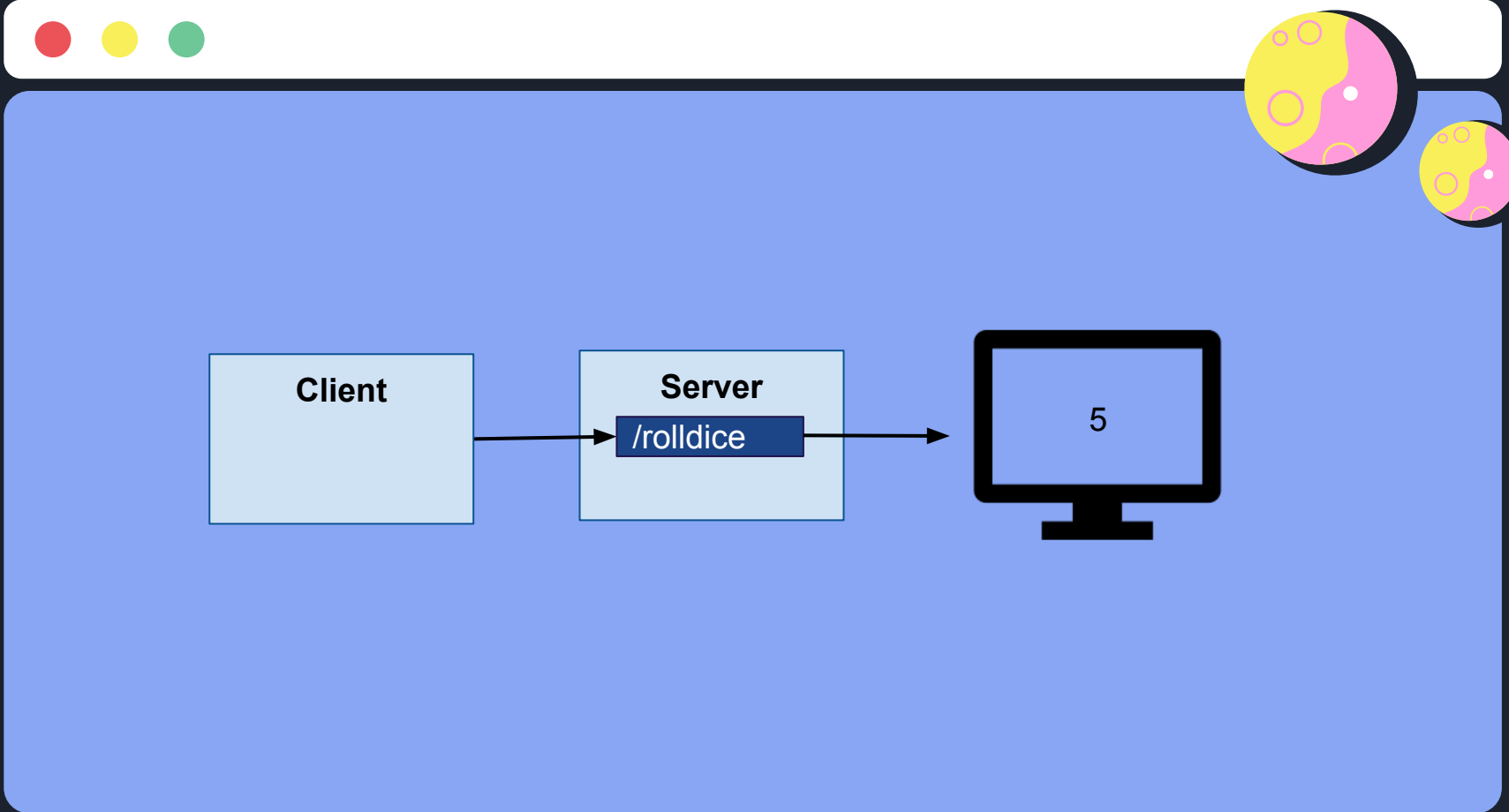




03

Show Me!





```
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
```

```
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)
```

```

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

```



```

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)

```

```

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

```

```

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)

```

```

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

```

```

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)

```

```

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

```

```

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

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](#)

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 count



Trace duration (ms)



Traces with errors



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

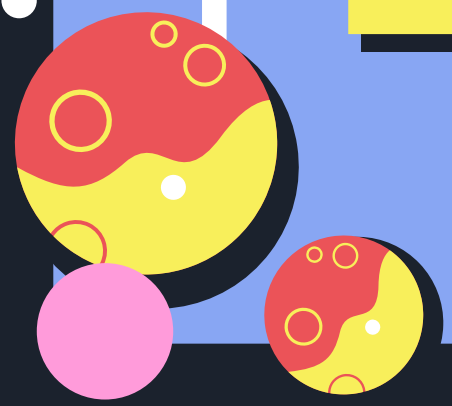
26.25 ms
p95

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)





Errors Example
[Repo on GitHub](#)



The inspirational post



Dude, Where's My
Error Blog Post



OpenTelemetry Spec
Compliance Matrix





Thanks!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik

