Observability Day

NORTH AMERICA

Observability Day
NORTH AMERICA

There's a Lot of Bad Telemetry Out There

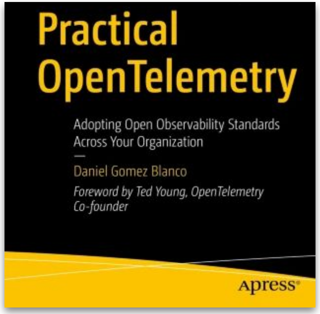
Dan Gomez Blanco, New Relic
Juraci Paixão Kröhling, OllyGarden

Speakers

Observability Day
NORTH AMERICA

Dan Gomez Blanco

- Principal Observability Architect @ New Relic
- OTel SIG End-User Maintainer
- Emeritus OTel GC
- https://dangb.me
- Author of Practical OpenTelemetry



Practical OpenTelemetry

Adopting Open Observability Standards Across Your Organization

Daniel Gomez Blanco

Foreword by Ted Young, OpenTelemetry Co-founder

Apress®

Juraci Paixão Kröhling

- Software Engineer @ OllyGarden
- OTel Governance Committee
- CNCF Ambassador
- Organizer OTel Night Berlin
- Emeritus: OTel Collector, Jaeger
- Telemetry Drops: LinkedIn, YouTube

Agenda

- What is *telemetry*?
- What is *bad telemetry*?
- Case #1: Fast responses, long traces in async workloads
- Case #2: Health check traces

What is *telemetry*?

What is *telemetry*?

observability /əbˌzɜːvəˈbɪləti/

noun

[1] Extent to which the internal state of a system can be inferred from observations of its external outputs.

telemetry /təˈlɛmətri/

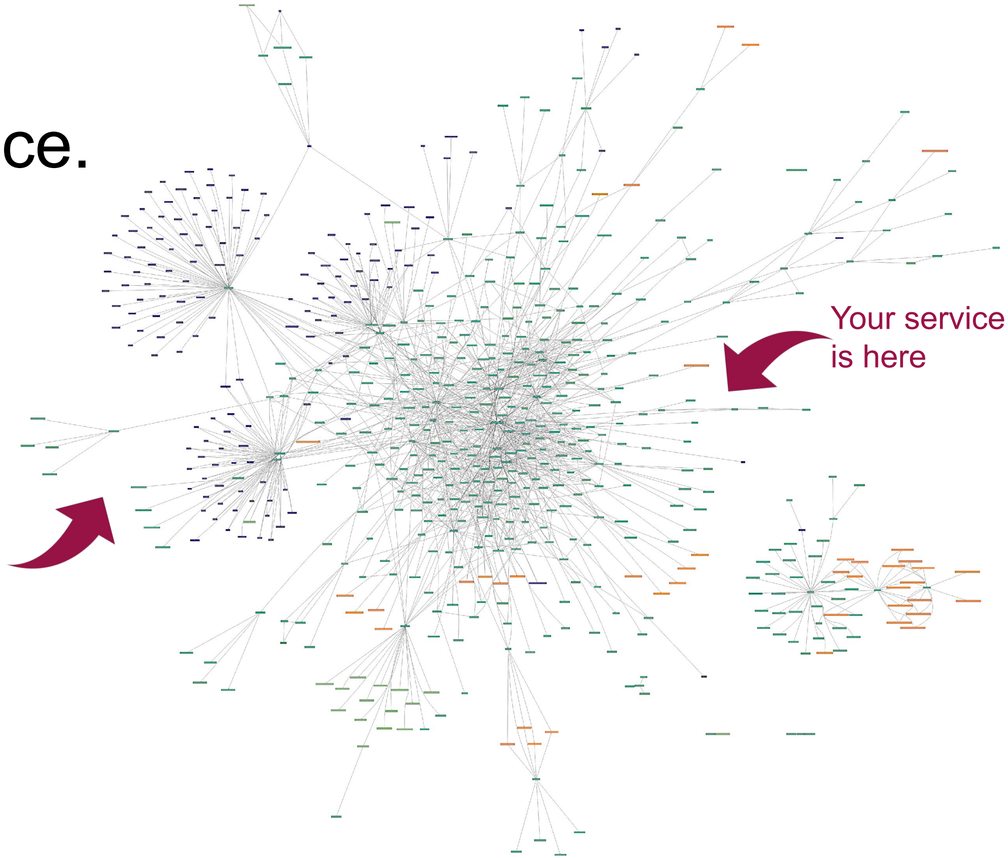
noun

[1] The external outputs... (*see* ***observability***).

What is *a system*?

Could be a single service.

However, it rarely is…

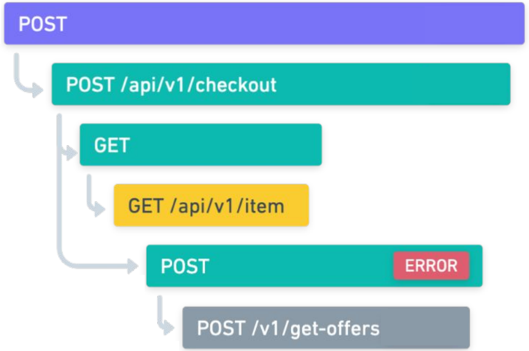


Your service is here

Your users are here

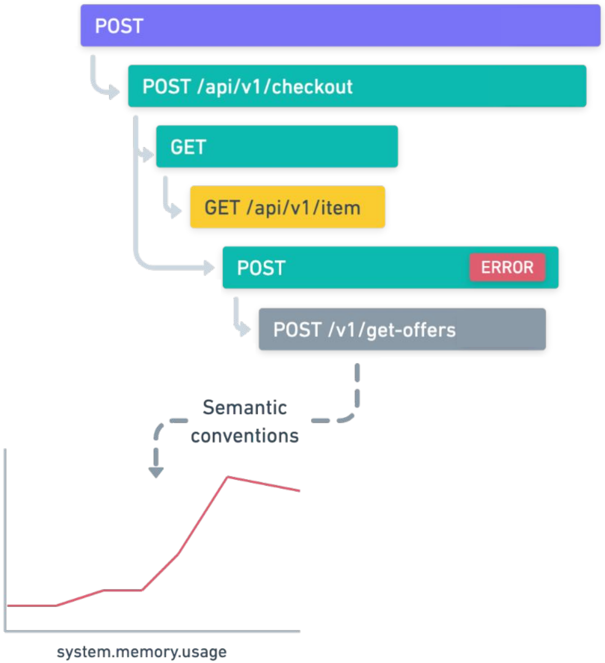
What telemetry do we need?

Observability Day
NORTH AMERICA

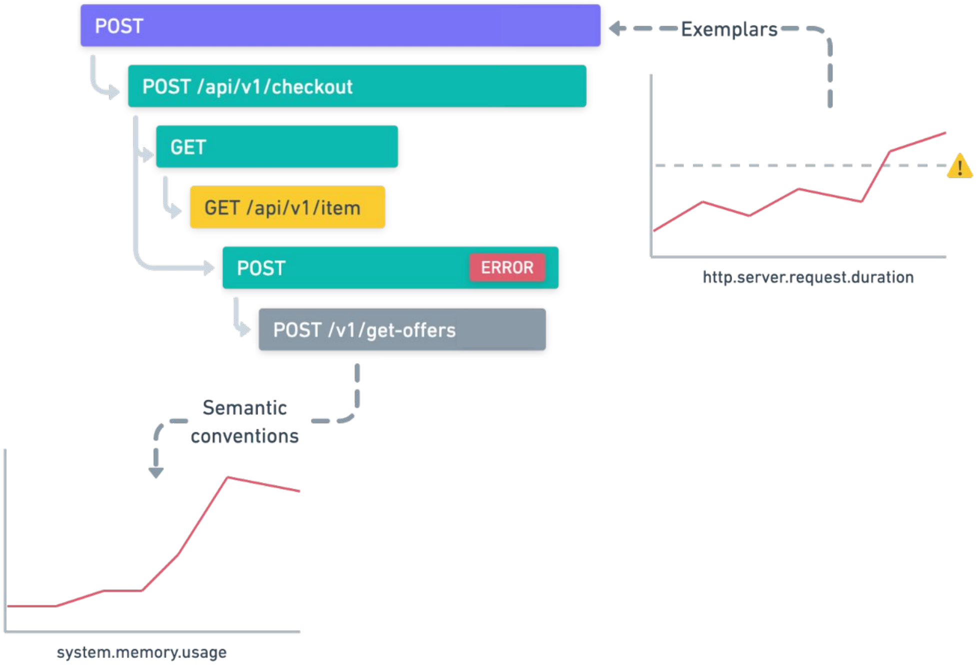


What telemetry do we need?

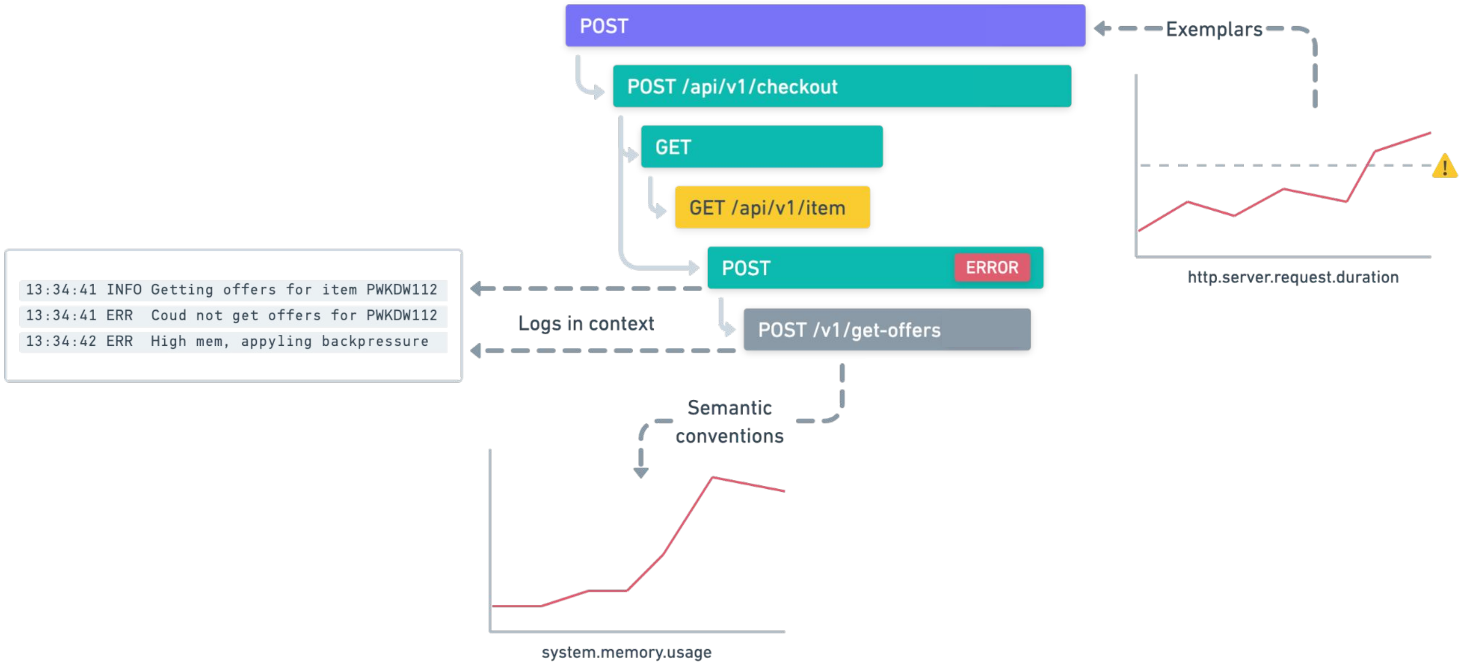
Observability Day
NORTH AMERICA



What telemetry do we need?

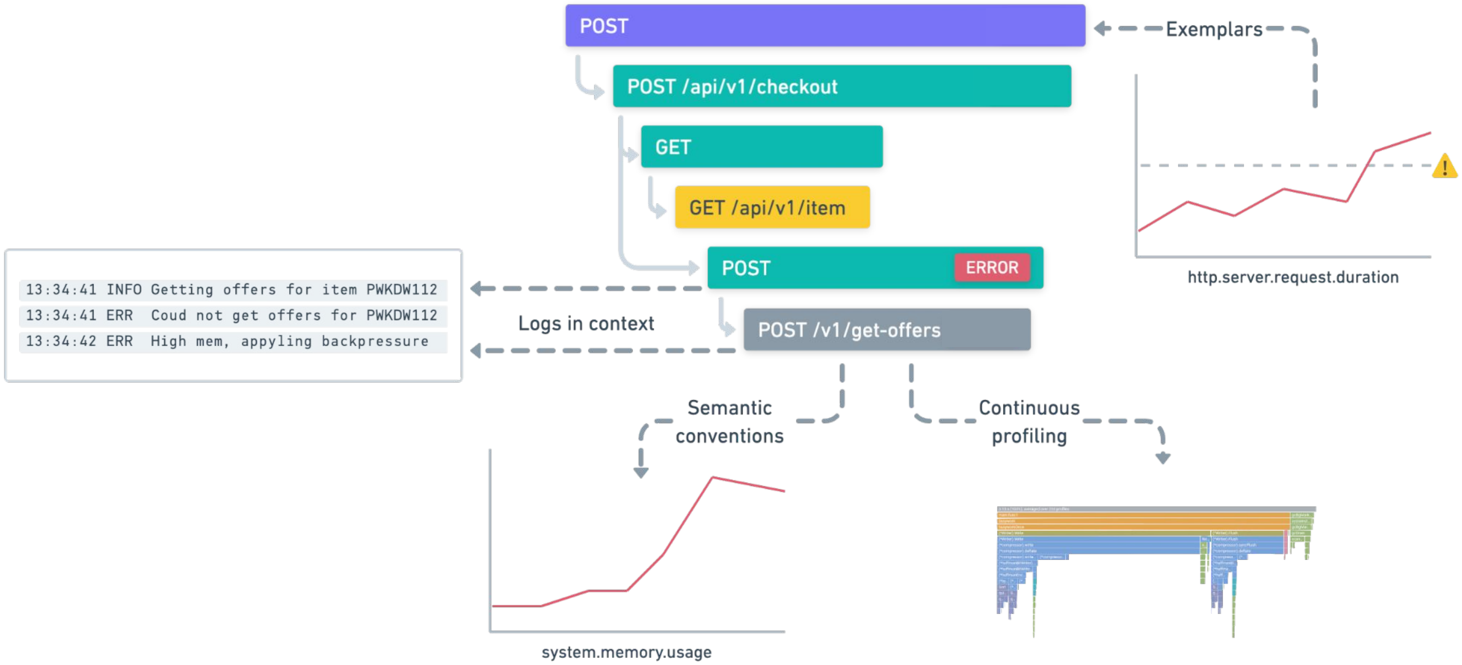


What telemetry do we need?



```
13:34:41 INFO Getting offers for item PWKDW112
13:34:41 ERR  Coud not get offers for PWKDW112
13:34:42 ERR  High mem, appyling backpressure
```

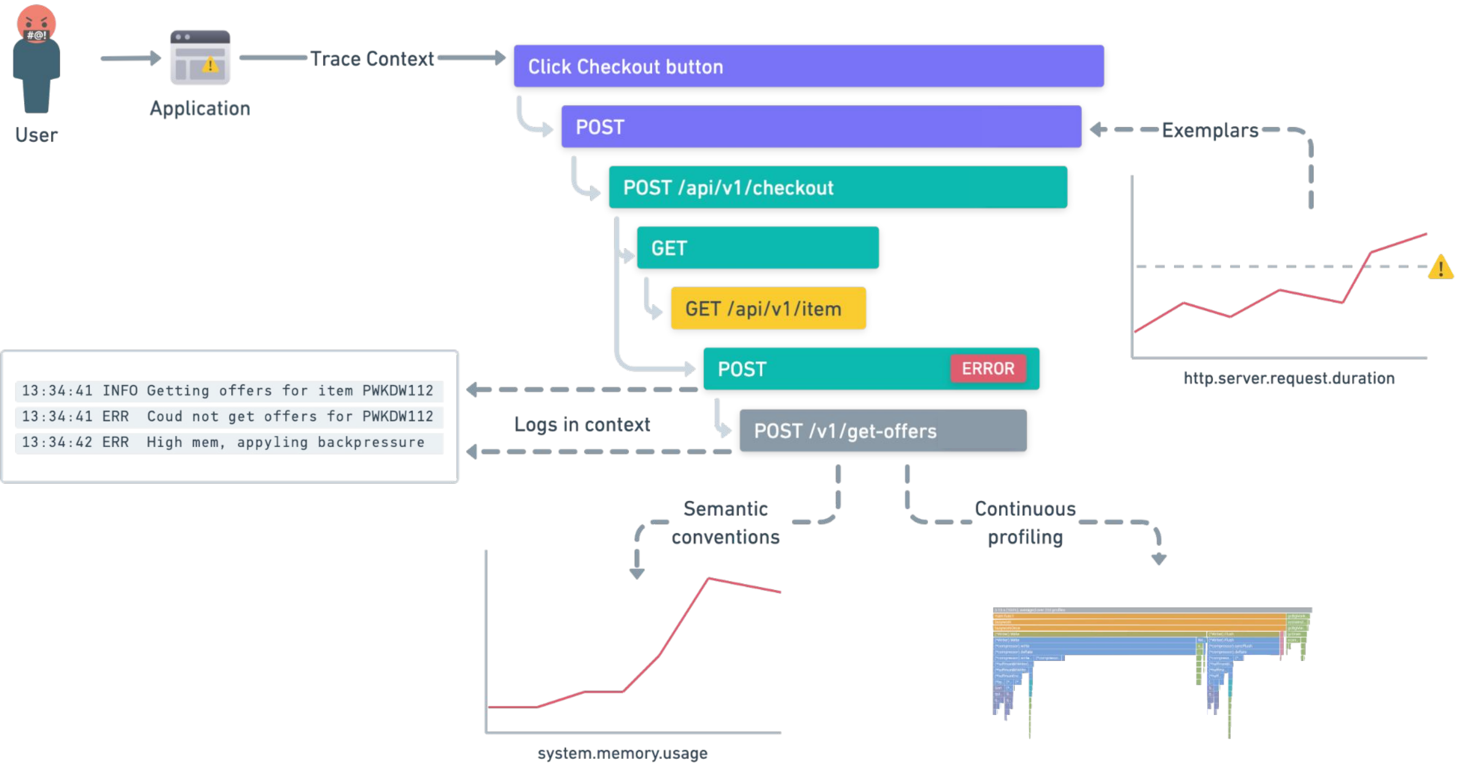
What telemetry do we need?



```
13:34:41 INFO Getting offers for item PWKDW112
13:34:41 ERR  Coud not get offers for PWKDW112
13:34:42 ERR  High mem, appyling backpressure
```

What telemetry do we need?

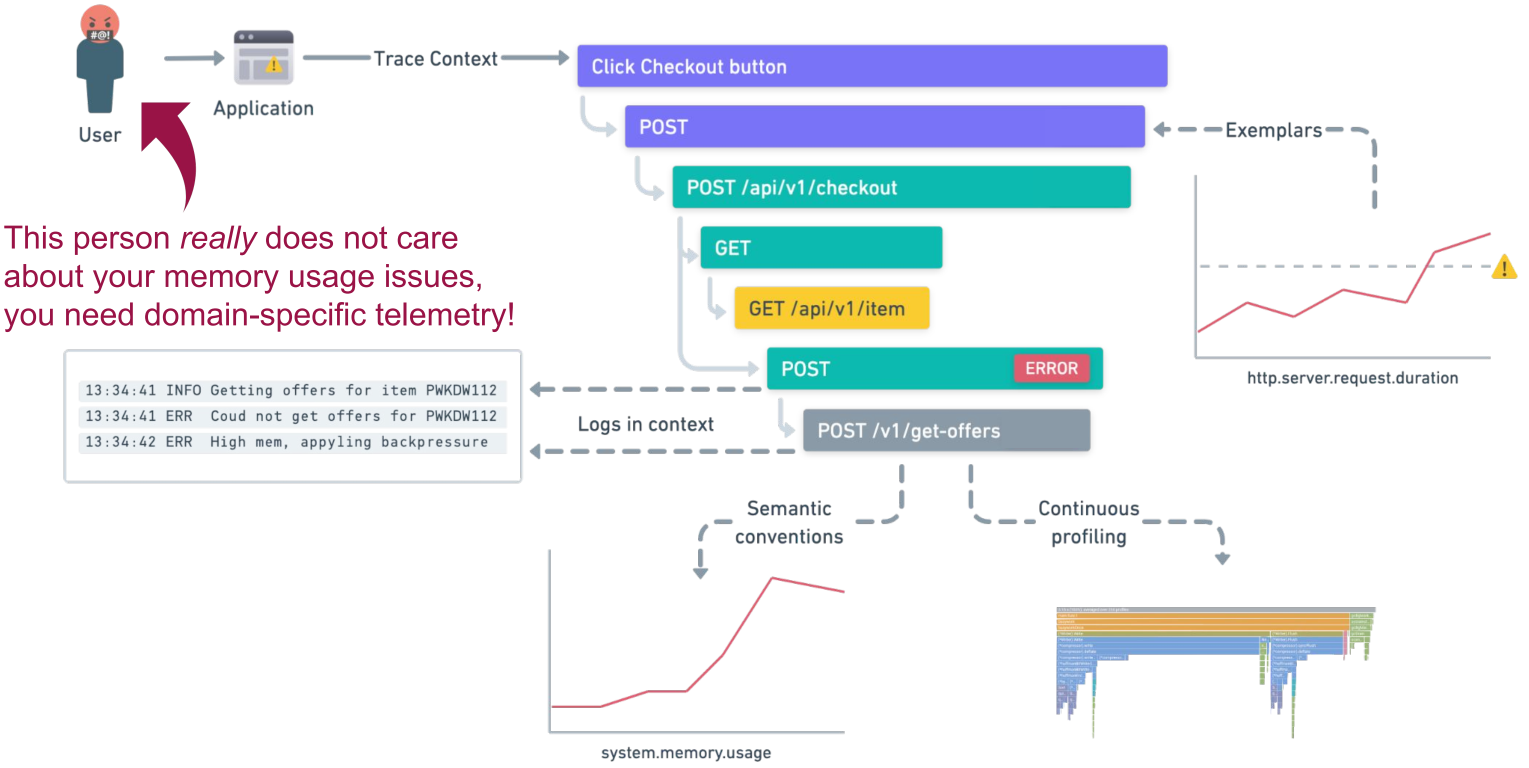
Observability Day
NORTH AMERICA



```
13:34:41 INFO Getting offers for item PWKDW112
13:34:41 ERR  Coud not get offers for PWKDW112
13:34:42 ERR  High mem, applying backpressure
```

What telemetry do we need?

Observability Day
NORTH AMERICA



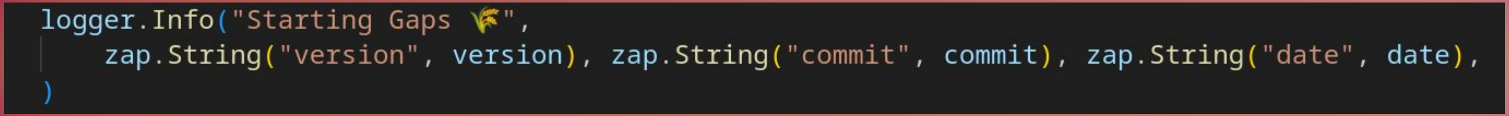
This person *really* does not care about your memory usage issues, you need domain-specific telemetry!

```
13:34:41 INFO Getting offers for item PWKDW112
13:34:41 ERR  Coud not get offers for PWKDW112
13:34:42 ERR  High mem, applying backpressure
```

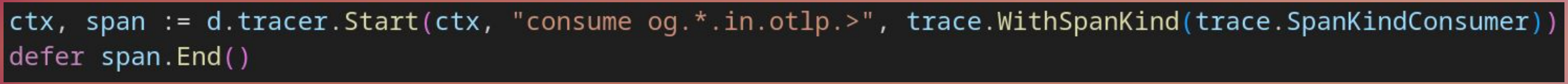
Instrumentation

Instrumentation

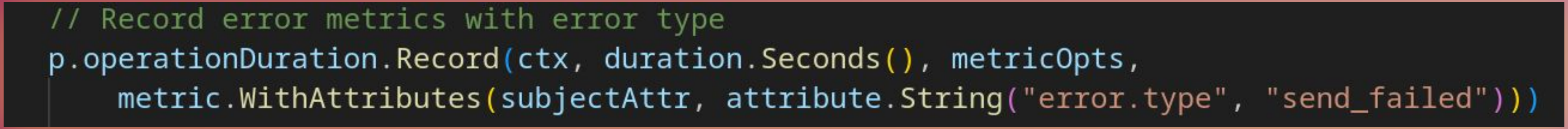
Code that generates telemetry from our programs, with the purpose of enabling humans and agents to understand application behavior at runtime.



```
logger.Info("Starting Gaps 🌾",
    zap.String("version", version), zap.String("commit", commit), zap.String("date", date),
)
```

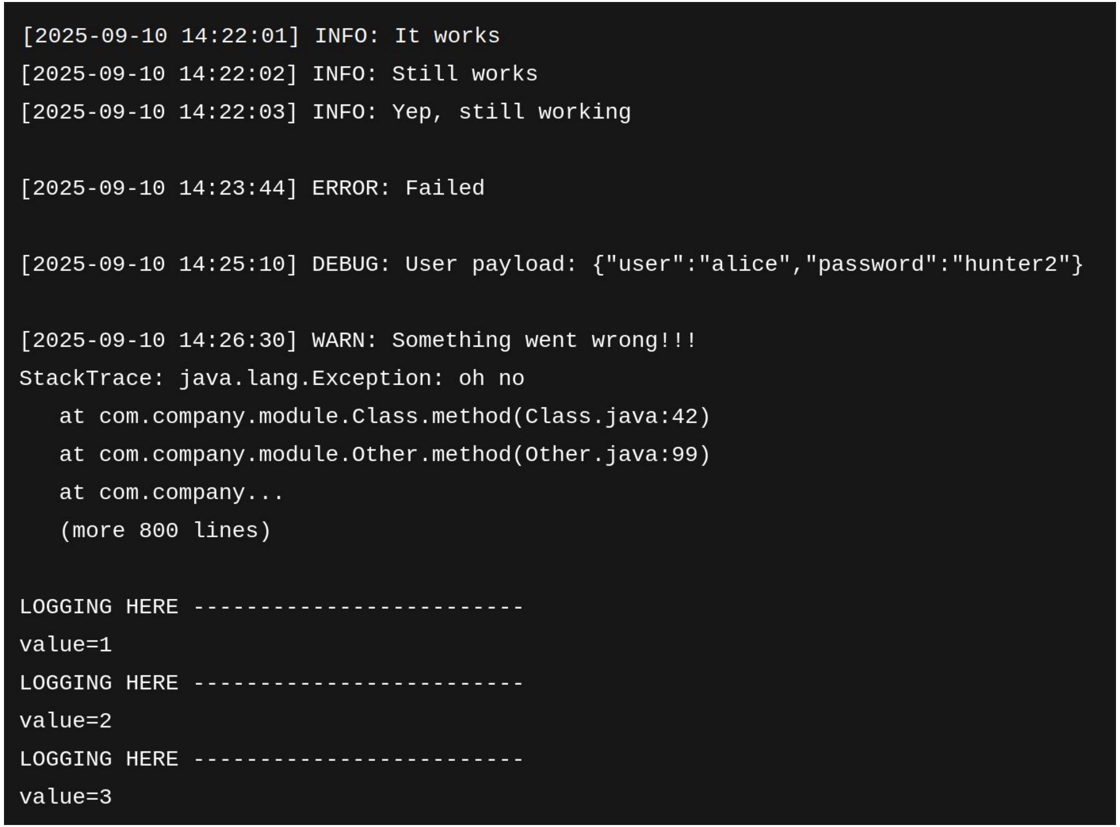


```
ctx, span := d.tracer.Start(ctx, "consume og.*.in.otlp.>", trace.WithSpanKind(trace.SpanKindConsumer))
defer span.End()
```

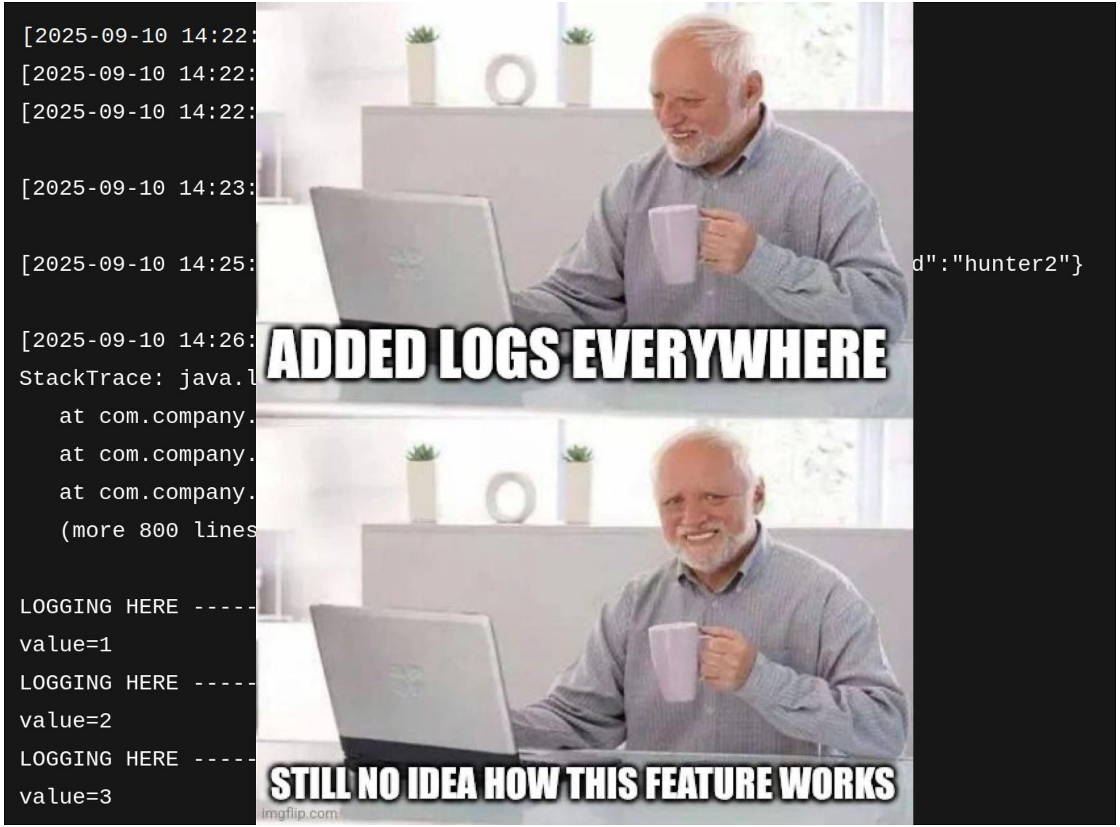


```
// Record error metrics with error type
p.operationDuration.Record(ctx, duration.Seconds(), metricOpts,
    metric.WithAttributes(subjectAttr, attribute.String("error.type", "send_failed")))
```

What is *bad telemetry?*



```
[2025-09-10 14:22:01] INFO: It works
[2025-09-10 14:22:02] INFO: Still works
[2025-09-10 14:22:03] INFO: Yep, still working

[2025-09-10 14:23:44] ERROR: Failed

[2025-09-10 14:25:10] DEBUG: User payload: {"user":"alice","password":"hunter2"}

[2025-09-10 14:26:30] WARN: Something went wrong!!!
StackTrace: java.lang.Exception: oh no
   at com.company.module.Class.method(Class.java:42)
   at com.company.module.Other.method(Other.java:99)
   at com.company...
   (more 800 lines)

LOGGING HERE -------------------------
value=1
LOGGING HERE -------------------------
value=2
LOGGING HERE -------------------------
value=3
```



```
[2025-09-10 14:22:
[2025-09-10 14:22:
[2025-09-10 14:22:

[2025-09-10 14:23:

[2025-09-10 14:25:                                              d":"hunter2"}

[2025-09-10 14:26:
StackTrace: java.l
   at com.company.
   at com.company.
   at com.company.
   (more 800 lines

LOGGING HERE -----
value=1
LOGGING HERE -----
value=2
LOGGING HERE -----
value=3
```

Bad Telemetry

Usefulness

Telemetry that doesn’t help in the goal of understanding the state of an application

⚠️ Noise

Worse yet if it delays resolutions

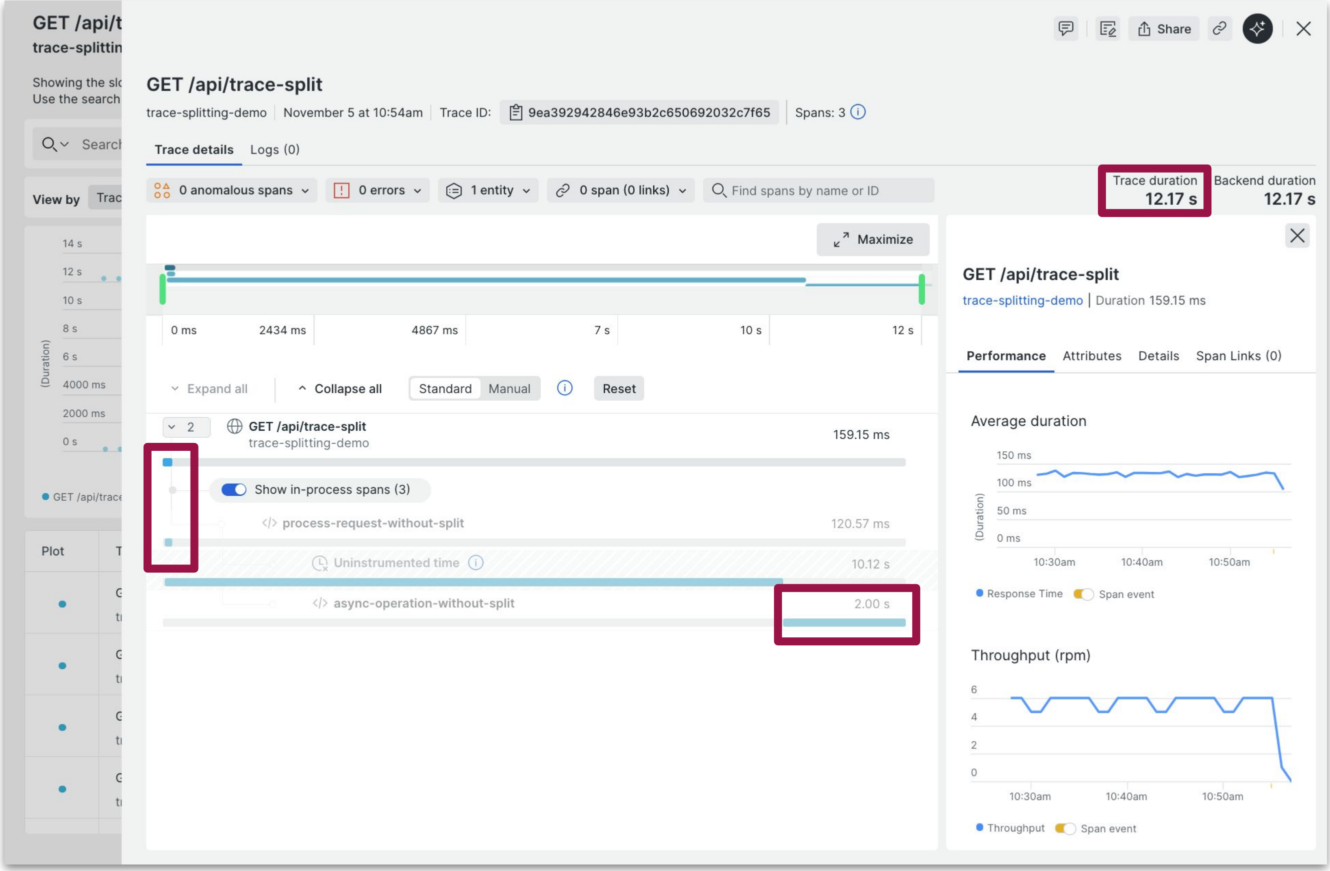
Costs and PII

Even worse: if it breaks your bank or risks penalties

Case #1
Fast responses, long traces

Fast responses, long traces

Observability Day
NORTH AMERICA



GET /api/trace-split

trace-splitting-demo | November 5 at 10:54am | Trace ID: 9ea392942846e93b2c650692032c7f65 | Spans: 3

Trace duration 12.17 s

Backend duration 12.17 s

0 anomalous spans | 0 errors | 1 entity | 0 span (0 links)

GET /api/trace-split — trace-splitting-demo — 159.15 ms

Show in-process spans (3)

process-request-without-split — 120.57 ms

Uninstrumented time — 10.12 s

async-operation-without-split — 2.00 s

GET /api/trace-split

trace-splitting-demo | Duration 159.15 ms

Why is this bad telemetry?

⏱️ Misleading duration

The trace duration has no relation to the main operation, or user experience

Difficult visualisation

Long traces and big gaps are not readable

Flawed parent/child model

Spans are only

ith

Inconsistent tail sampling

Orphan spans, storing half of a trace or, even worse, storing lots of long traces that are perfectly “normal”

Some ~~past nightmares~~ common cases

Messaging instrumentation

Producer/consumer spans not following Semantic Conventions for messaging systems

Gotchas in async frameworks

E.g. reusing JS *Promise* instances in lazy-loaded recursive pollers = infinite traces

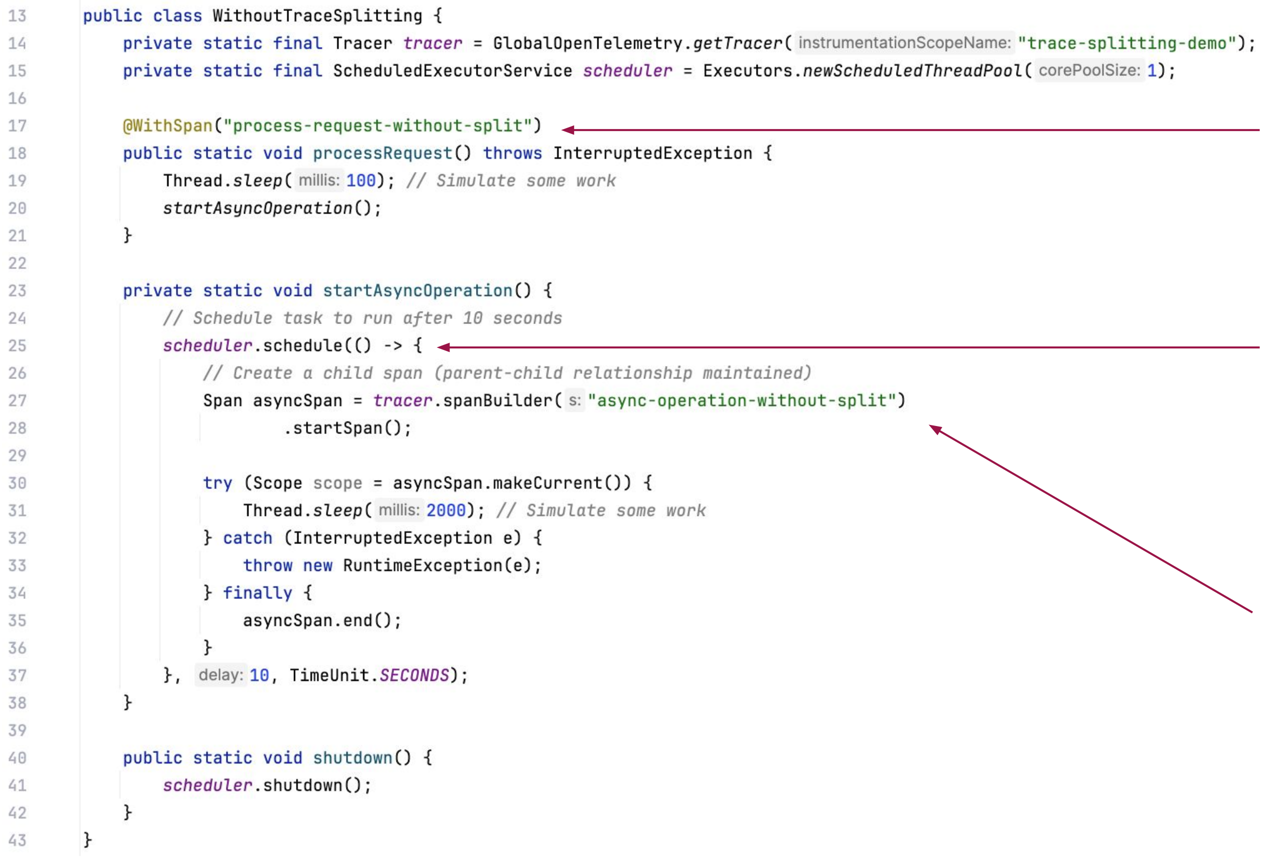
Busy/scheduled thread pools

Fire-and-forget tasks not not awaited by the caller, picked up by another thread later

Tinkering with Context state

Honestly, just use the OTel APIs for each signal unless you *really* know what you're doing

Auto-propagated context



```
public class WithoutTraceSplitting {
    private static final Tracer tracer = GlobalOpenTelemetry.getTracer( instrumentationScopeName: "trace-splitting-demo");
    private static final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool( corePoolSize: 1);

    @WithSpan("process-request-without-split")
    public static void processRequest() throws InterruptedException {
        Thread.sleep( millis: 100); // Simulate some work
        startAsyncOperation();
    }

    private static void startAsyncOperation() {
        // Schedule task to run after 10 seconds
        scheduler.schedule(() -> {
            // Create a child span (parent-child relationship maintained)
            Span asyncSpan = tracer.spanBuilder( s: "async-operation-without-split")
                    .startSpan();

            try (Scope scope = asyncSpan.makeCurrent()) {
                Thread.sleep( millis: 2000); // Simulate some work
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            } finally {
                asyncSpan.end();
            }
        },  delay: 10, TimeUnit.SECONDS);
    }

    public static void shutdown() {
        scheduler.shutdown();
    }
}
```

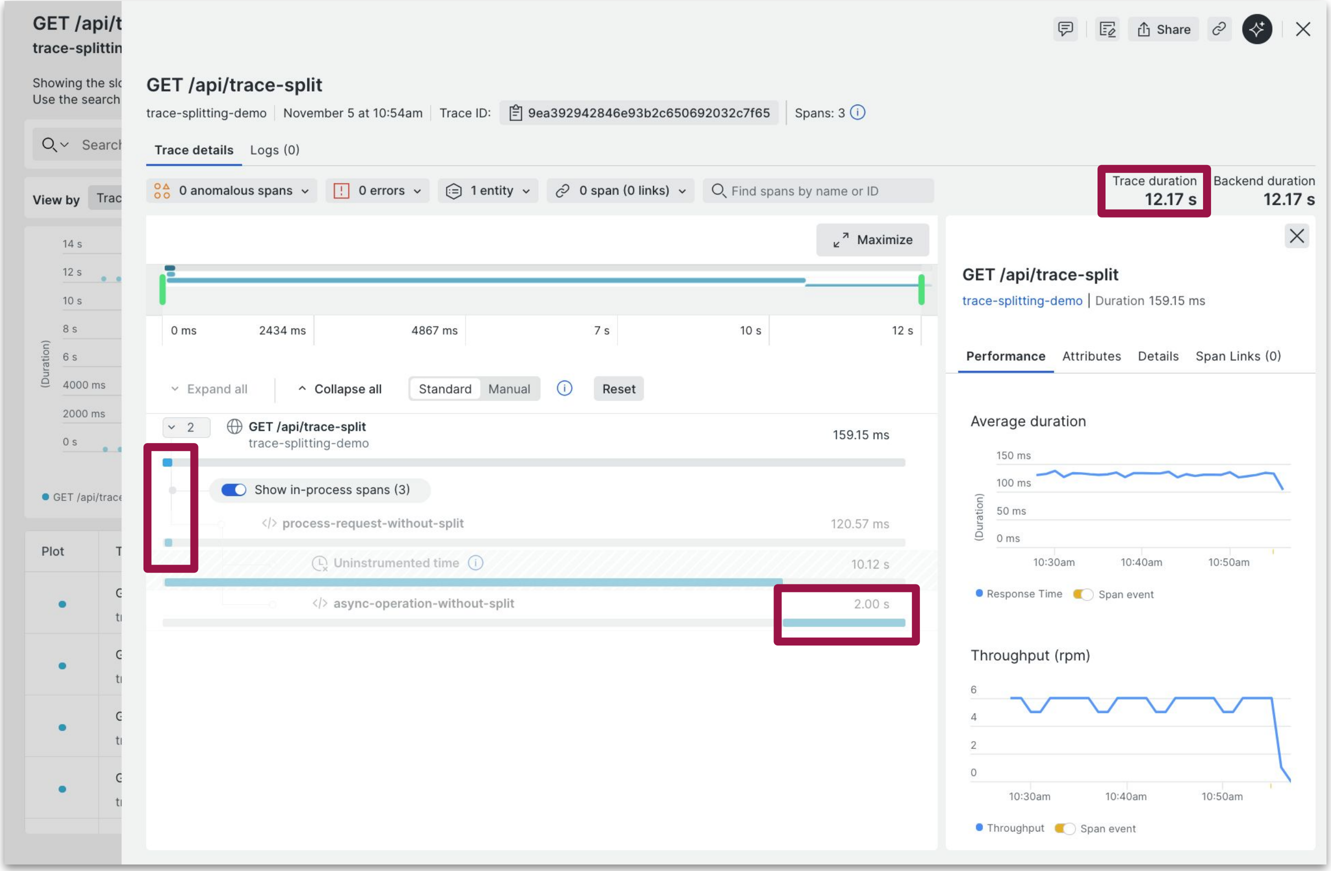
Span wraps the whole method execution

The `executors` instrumentation lib automatically wraps the `Runnable` and propagates context (i.e. thread-local vars) to the thread executing async task

`asyncSpan` inherently gets its context from the active *Span Context*, which is the one propagated from the caller

Fast responses, long traces

Observability Day
NORTH AMERICA



GET /api/trace-split

trace-splitting-demo | November 5 at 10:54am | Trace ID: 9ea392942846e93b2c650692032c7f65 | Spans: 3

Trace details | Logs (0)

0 anomalous spans | 0 errors | 1 entity | 0 span (0 links) | Find spans by name or ID

Trace duration 12.17 s

Backend duration 12.17 s

Maximize

0 ms | 2434 ms | 4867 ms | 7 s | 10 s | 12 s

Expand all | Collapse all | Standard | Manual | Reset

2 GET /api/trace-split
trace-splitting-demo — 159.15 ms

Show in-process spans (3)

process-request-without-split — 120.57 ms

Uninstrumented time — 10.12 s

async-operation-without-split — 2.00 s

GET /api/trace-split

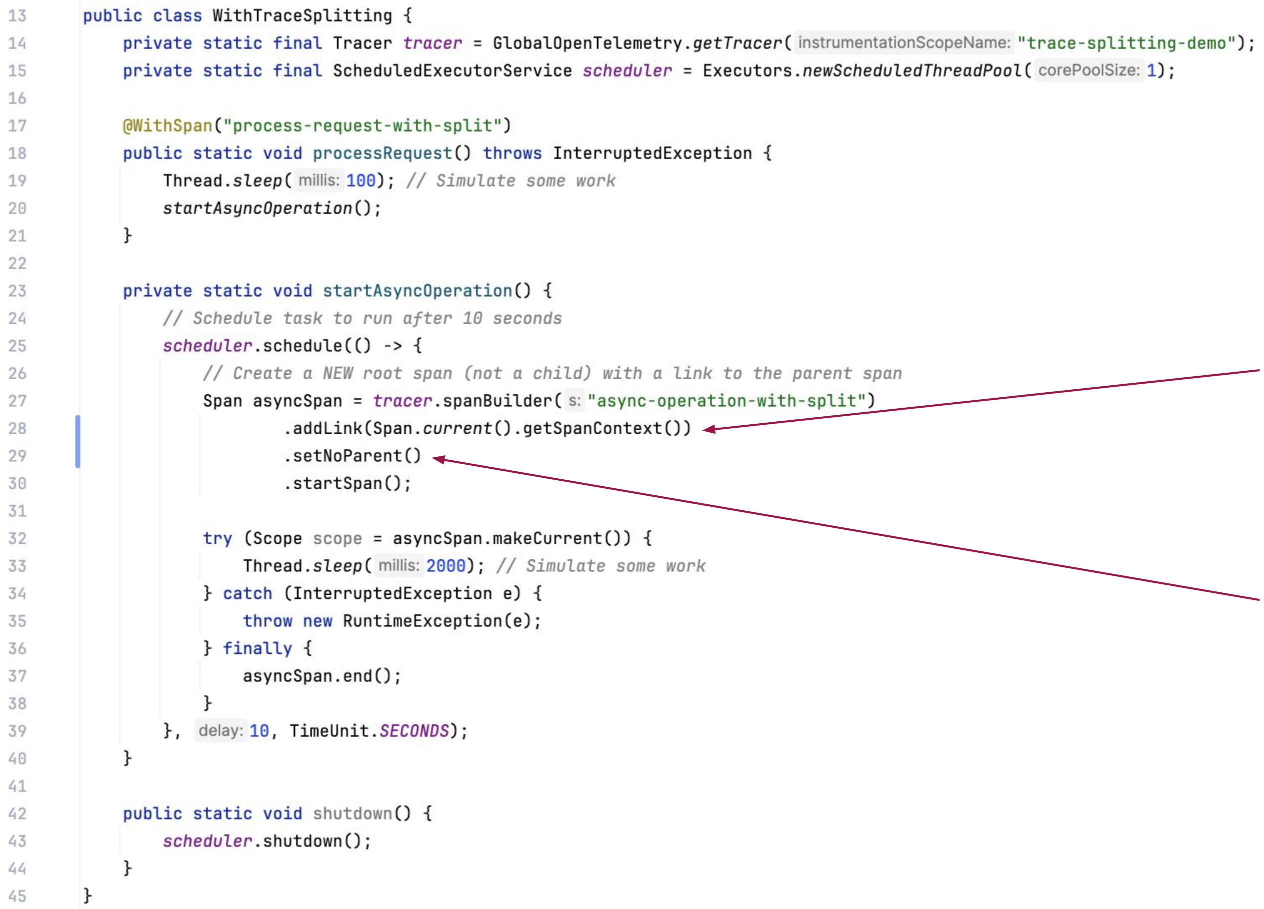
trace-splitting-demo | Duration 159.15 ms

Performance | Attributes | Details | Span Links (0)

Average duration

Throughput (rpm)

Split trace and link spans



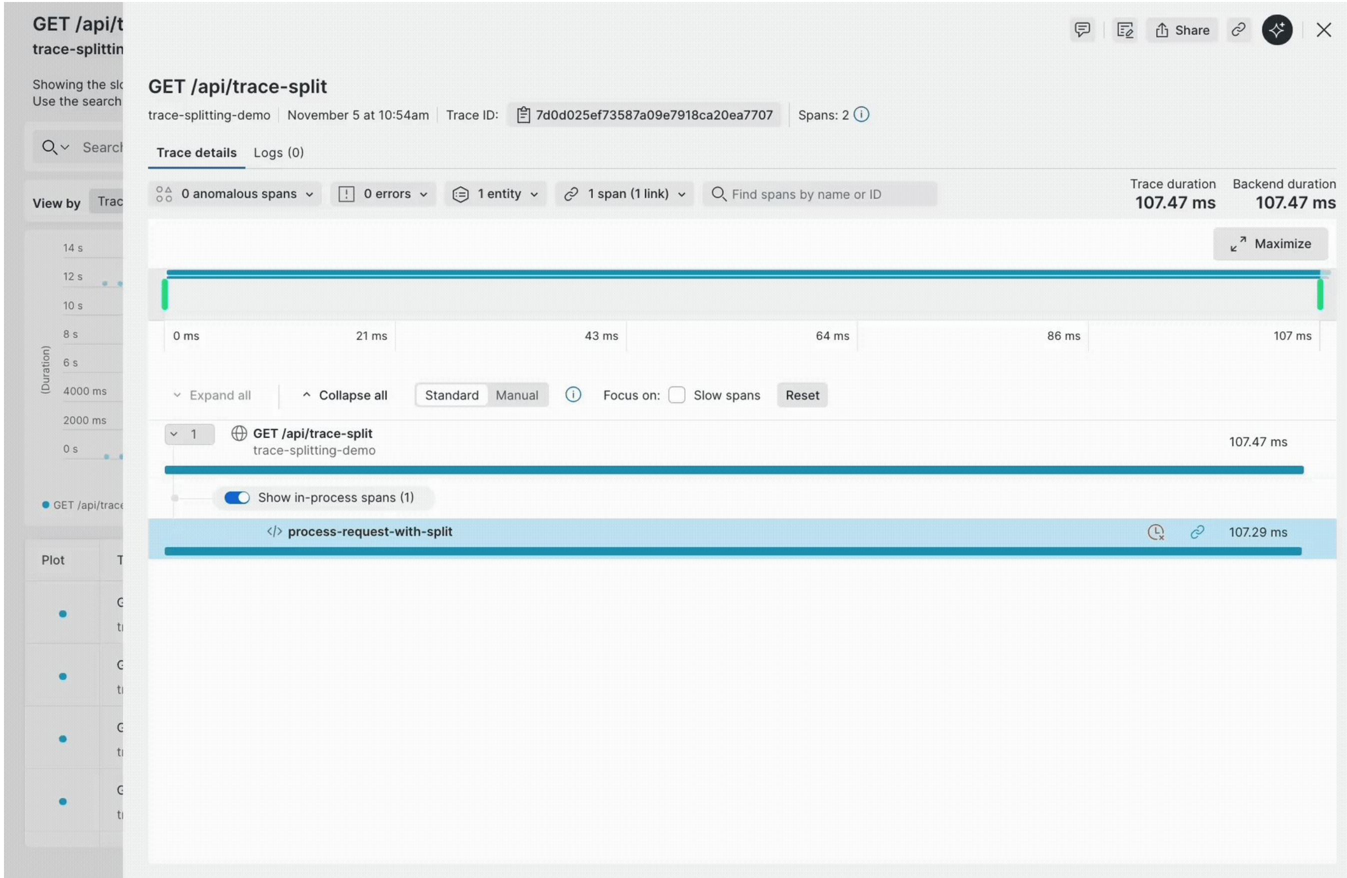
```java
public class WithTraceSplitting {
    private static final Tracer tracer = GlobalOpenTelemetry.getTracer( instrumentationScopeName: "trace-splitting-demo");
    private static final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool( corePoolSize: 1);

    @WithSpan("process-request-with-split")
    public static void processRequest() throws InterruptedException {
        Thread.sleep( millis: 100); // Simulate some work
        startAsyncOperation();
    }

    private static void startAsyncOperation() {
        // Schedule task to run after 10 seconds
        scheduler.schedule(() -> {
            // Create a NEW root span (not a child) with a link to the parent span
            Span asyncSpan = tracer.spanBuilder( s: "async-operation-with-split")
                    .addLink(Span.current().getSpanContext())
                    .setNoParent()
                    .startSpan();

            try (Scope scope = asyncSpan.makeCurrent()) {
                Thread.sleep( millis: 2000); // Simulate some work
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            } finally {
                asyncSpan.end();
            }
        },  delay: 10, TimeUnit.SECONDS);
    }

    public static void shutdown() {
        scheduler.shutdown();
    }
}
```

Add a *Span Link* to the current span to describe the causal relationship

Explicitly make this a root *Span*, thus starting a new trace

Traces represent units of work

Observability Day
NORTH AMERICA



GET /api/trace-split

trace-splitting-demo | November 5 at 10:54am | Trace ID: 7d0d025ef73587a09e7918ca20ea7707 | Spans: 2

Trace details | Logs (0)

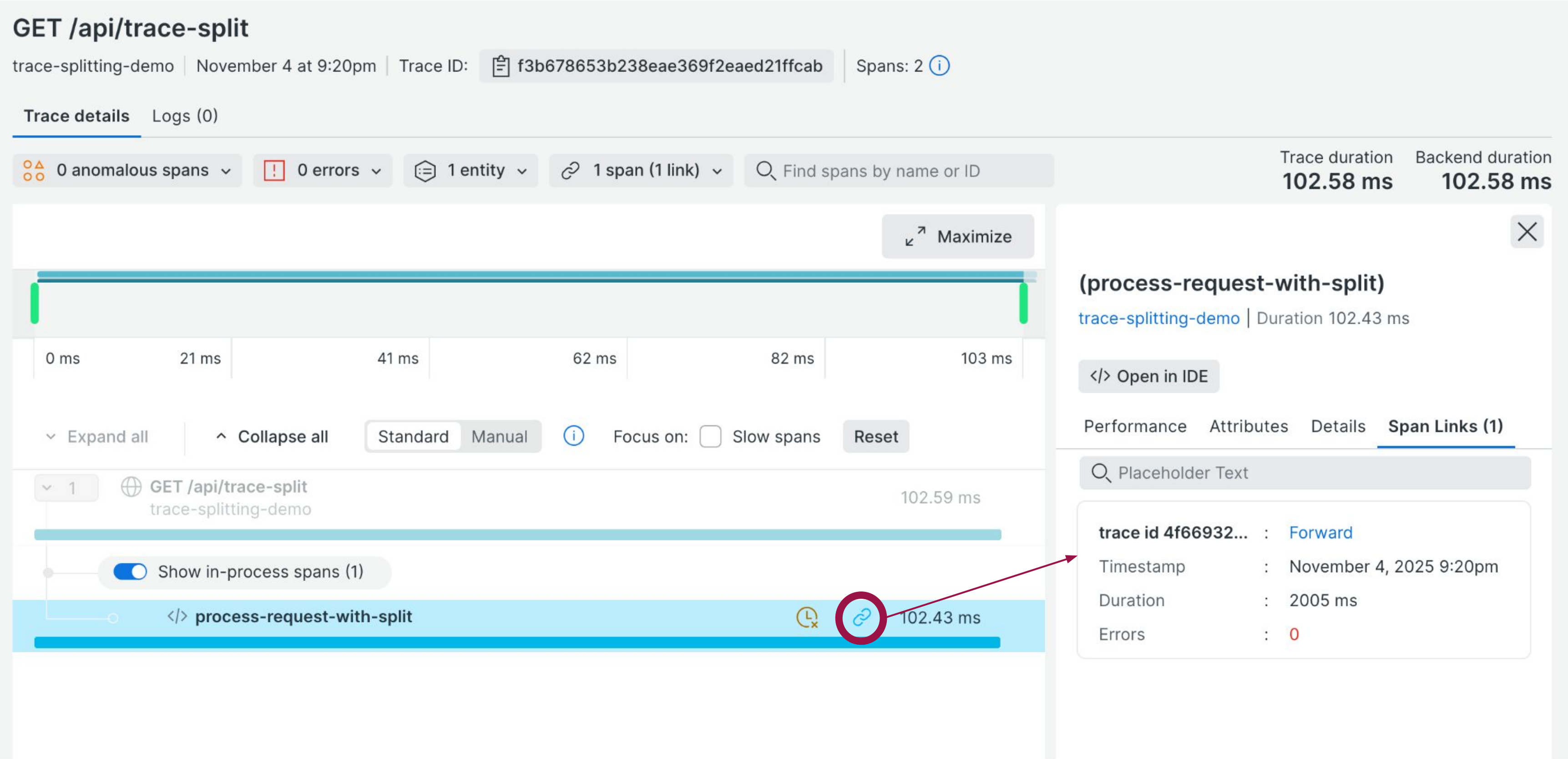
0 anomalous spans | 0 errors | 1 entity | 1 span (1 link) | Find spans by name or ID

Trace duration 107.47 ms | Backend duration 107.47 ms

1	GET /api/trace-split trace-splitting-demo	107.47 ms
	Show in-process spans (1)	
	process-request-with-split	107.29 ms

Traces represent units of work

Observability Day
NORTH AMERICA



GET /api/trace-split

trace-splitting-demo | November 4 at 9:20pm | Trace ID: f3b678653b238eae369f2eaed21ffcab | Spans: 2

Trace details | Logs (0)

0 anomalous spans | 0 errors | 1 entity | 1 span (1 link) | Find spans by name or ID

Trace duration 102.58 ms | Backend duration 102.58 ms

1 GET /api/trace-split
trace-splitting-demo — 102.59 ms

Show in-process spans (1)

process-request-with-split — 102.43 ms

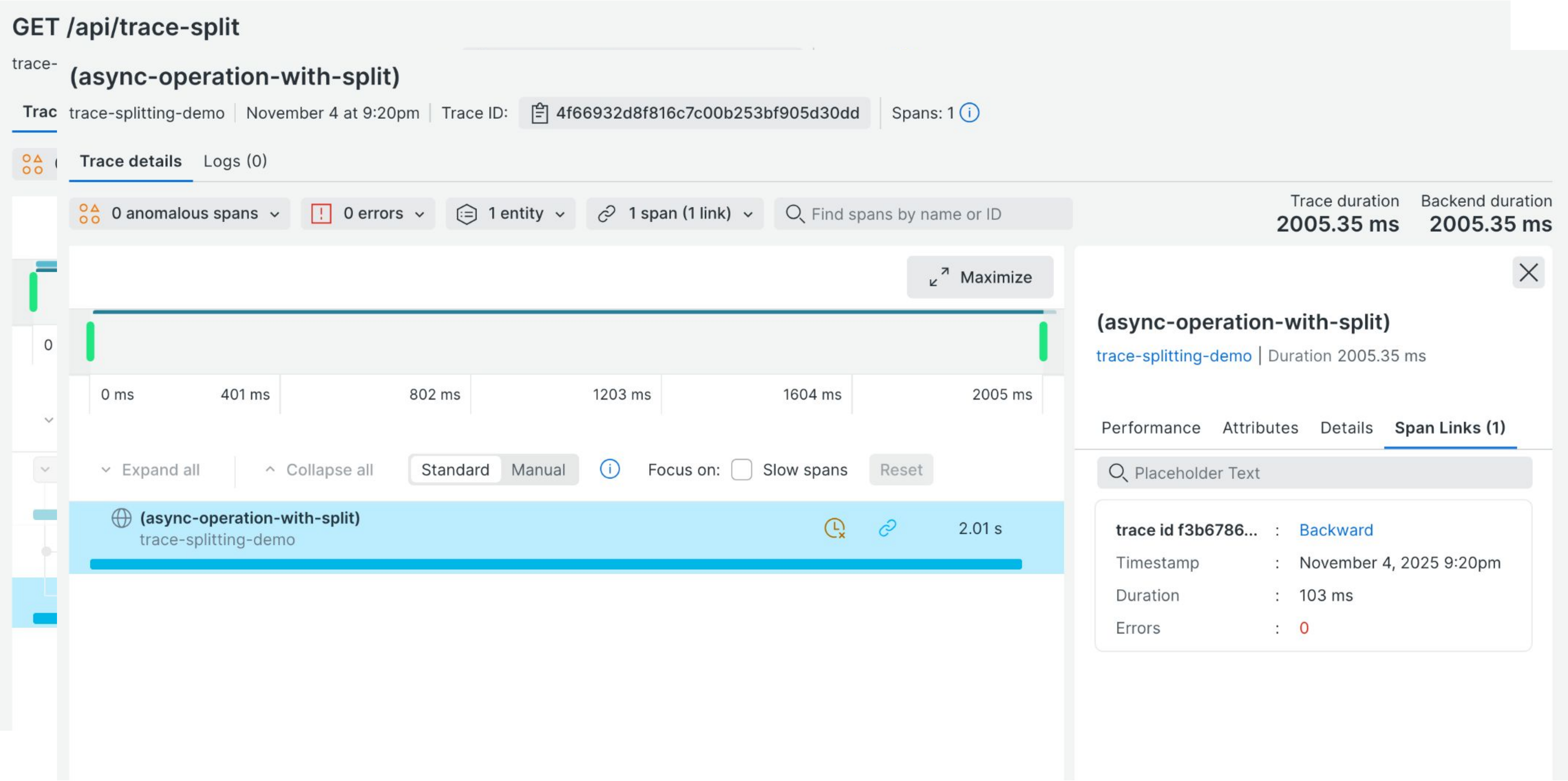
(process-request-with-split)

trace-splitting-demo | Duration 102.43 ms

trace id 4f66932...	:	Forward
Timestamp	:	November 4, 2025 9:20pm
Duration	:	2005 ms
Errors	:	0

Traces represent units of work

Observability Day
NORTH AMERICA



GET /api/trace-split

trace-

(async-operation-with-split)

Trac trace-splitting-demo | November 4 at 9:20pm | Trace ID: 4f66932d8f816c7c00b253bf905d30dd | Spans: 1

Trace details | Logs (0)

0 anomalous spans | 0 errors | 1 entity | 1 span (1 link) | Find spans by name or ID

Trace duration 2005.35 ms

Backend duration 2005.35 ms

(async-operation-with-split)
trace-splitting-demo 2.01 s

(async-operation-with-split)

trace-splitting-demo | Duration 2005.35 ms

Performance | Attributes | Details | Span Links (1)

Placeholder Text

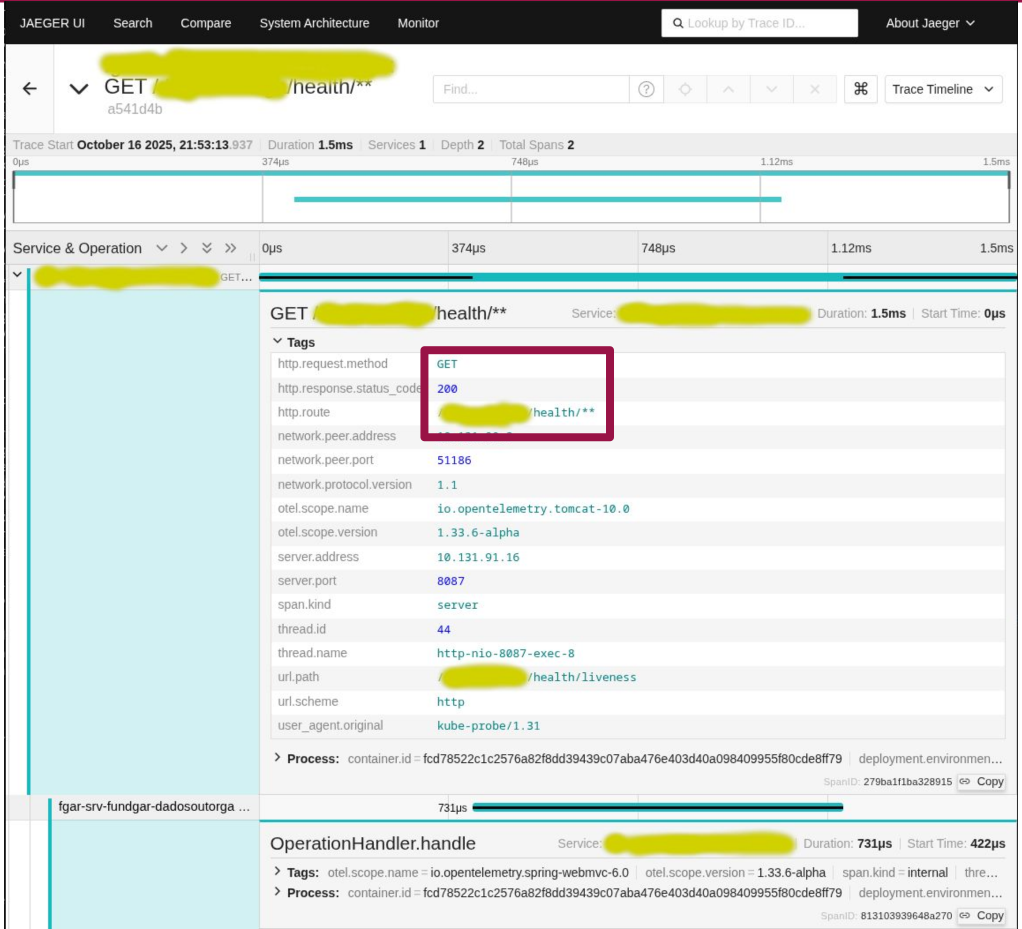
trace id f3b6786...	:	Backward
Timestamp	:	November 4, 2025 9:20pm
Duration	:	103 ms
Errors	:	0

Case #2

Case #2

- Health-check traces
  - Often single-span traces (but not always!)
  - Almost always with a 200 status code
  - Low business value
  - Noisy: once every few seconds per container/pod
  - Comes with different names, shapes, and forms

Case #2



GET /[redacted]/health/**
a541d4b

Trace Start **October 16 2025, 21:53:13**.937 | Duration **1.5ms** | Services **1** | Depth **2** | Total Spans **2**

[redacted] GET...

GET /[redacted]/health/** Service: [redacted] | Duration: **1.5ms** | Start Time: **0μs**

Tags

http.request.method	GET
http.response.status_code	200
http.route	/[redacted]/health/**
network.peer.address	[illegible]
network.peer.port	51186
network.protocol.version	1.1
otel.scope.name	io.opentelemetry.tomcat-10.0
otel.scope.version	1.33.6-alpha
server.address	10.131.91.16
server.port	8087
span.kind	server
thread.id	44
thread.name	http-nio-8087-exec-8
url.path	/[redacted]/health/liveness
url.scheme	http
user_agent.original	kube-probe/1.31

Process: container.id = fcd78522c1c2576a82f8dd39439c07aba476e403d40a098409955f80cde8ff79 | deployment.environmen...

SpanID: 279ba1f1ba328915 Copy

fgar-srv-fundgar-dadosoutorga ... 731μs

OperationHandler.handle Service: [redacted] | Duration: **731μs** | Start Time: **422μs**

Tags: otel.scope.name = io.opentelemetry.spring-webmvc-6.0 | otel.scope.version = 1.33.6-alpha | span.kind = internal | thre...

Process: container.id = fcd78522c1c2576a82f8dd39439c07aba476e403d40a098409955f80cde8ff79 | deployment.environmen...

SpanID: 813103939648a270 Copy

Case #2

```
1 | GET /health/**
8 | GET /health/{*path}
14 | GET /health
17 | GET /api/health/**
18 | GET /actuator/health/**
25 | GET /v1/health/**
26 | GET /health/liveness
28 | GET /health/readiness
30 | GET /v1/health
```

```
34 | GET /REDACTED/health/**
39 | GET /REDACTED/health/**
41 | GET /REDACTED/health/**
42 | GET /REDACTED/health/**
43 | GET /REDACTED/health/**
47 | GET /REDACTED/health/**
49 | GET /REDACTED/health/**
50 | GET /REDACTED/actuator/health/**
```

Case #2

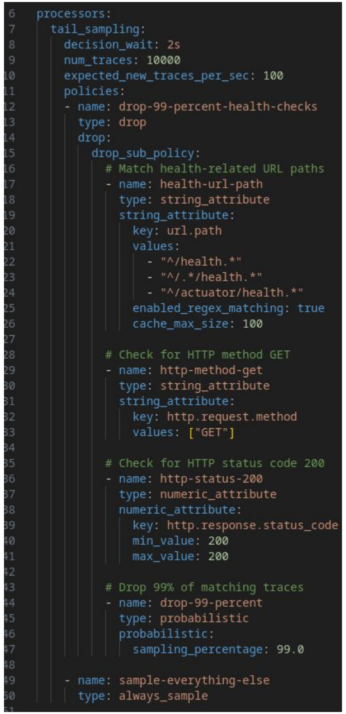
- Hint: “Why it took 5 years to ignore health check endpoints in tracing”
  - https://opentelemetry.io/blog/2025/declarative-config/

Case #2

```yaml
tracer_provider:
  sampler:
    rule_based_routing:
      fallback_sampler:
        always_on:
      span_kind: SERVER
      rules:
        - action: DROP
          attribute: url.path
          pattern: /actuator.*
```

Case #2

- OTel Collector with tail-sampling, removing health checks:



```yaml
processors:
  tail_sampling:
    decision_wait: 2s
    num_traces: 10000
    expected_new_traces_per_sec: 100
    policies:
    - name: drop-99-percent-health-checks
      type: drop
      drop:
        drop_sub_policy:
          # Match health-related URL paths
          - name: health-url-path
            type: string_attribute
            string_attribute:
              key: url.path
              values:
                - "^/health.*"
                - "^/.*/health.*"
                - "^/actuator/health.*"
              enabled_regex_matching: true
              cache_max_size: 100

          # Check for HTTP method GET
          - name: http-method-get
            type: string_attribute
            string_attribute:
              key: http.request.method
              values: ["GET"]

          # Check for HTTP status code 200
          - name: http-status-200
            type: numeric_attribute
            numeric_attribute:
              key: http.response.status_code
              min_value: 200
              max_value: 200

          # Drop 99% of matching traces
          - name: drop-99-percent
            type: probabilistic
            probabilistic:
              sampling_percentage: 99.0

    - name: sample-everything-else
      type: always_sample
```

Key takeaways

Bad telemetry

Has many side-effects

Reduced observability, debugging noise, PII leaks, cost…

Comes in many shapes

Leaky context propagation, verbosity, inconsistency with semantic conventions

Can be fixed in many ways

API, SDK, Collector… you can choose the most appropriate level

What about governance and compliance?

OpenTelemetry Weaver

https://github.com/open-telemetry/weaver

Observability by design.
Treat your telemetry like a public API.

instrumentation-score

🏗️ Under construction…

https://github.com/instrumentation-score

Measuring and evaluating quality of telemetry across software systems

Instrumentation Score

- Instrumentation Score: The Difference Between Telemetry and Good Telemetry - Juraci Paixão Kröhling, OllyGarden & Michele Mancioppi, Dash0

Thank you!

- We’ll be at the Observatory at different times during KubeCon
- Come talk to us about instrumentation score!
- Got any feedback about this session? Here’s the QR code.

Thank you!