Your Telemetry Has a Story - Write It Down

Liudmila Molkova

About me

@lmolkova @neskazu.bsky.social lmolkova@linkedin

- Staff Developer Advocate @ Grafana Labs
- Member of the OTel Technical Committee and OTel SemConv maintainer

"There is a lot of bad telemetry out there."

- Juraci

There is even more **undocumented** telemetry.

Metrics documentation in the wild: Exhibit 1

1.2.4. Tasks Submitted

> *Counter that increments by one each time a task is submitted (via any of the schedule methods on both Scheduler and Scheduler.Worker).*
>
> *Note that there are actually 4 counters, which can be differentiated by the SubmittedTags#SUBMISSION tag. The sum of all these can thus be compared with the TASKS_COMPLETED counter.*

Metric name `%s.scheduler.tasks.submitted` - since it contains `%s`, the name is dynamic and will be resolved at runtime. **Type** `counter`.

IMPORTANT

KeyValues that are added after starting the Observation might be missing from the *.active metrics.

Table 6. Low cardinality Keys

Name	Description
`submission.type` *(required)*	The type of submission: • `"direct"` for `Scheduler#schedule(Runnable)` • `"delayed"` for `Scheduler#schedule(Runnable,long,TimeUnit)` • `"periodic_initial"` for `Scheduler#schedulePeriodically(Runnable,long,long,TimeUnit)` after the initial delay • `"periodic_iteration"` for `Scheduler#schedulePeriodically(Runnable,long,long,TimeUnit)` further periodic iterations

link

Metrics documentation in the wild: Exhibit 2

Telemetry metric name	OTLP metric data point type	Labels	Description
`producer.connection.creation.rate`	Gauge		The rate of connections established per second.
`producer.connection.creation.total`	Sum		The total number of connections established.
`producer.node.request.latency.avg`	Gauge	node_id	The average request latency in ms for a node.
`producer.node.request.latency.max`	Gauge	node_id	The maximum request latency in ms for a node.
`producer.produce.throttle.time.avg`	Gauge		The average time in ms a request was throttled by the broker.
`producer.produce.throttle.time.max`	Gauge		The maximum time in ms a request was throttled by the broker.
`producer.record.queue.time.avg`	Gauge		The average time in ms record batches spent in the send buffer.
`producer.record.queue.time.max`	Gauge		The maximum time in ms record batches spent in the send buffer.

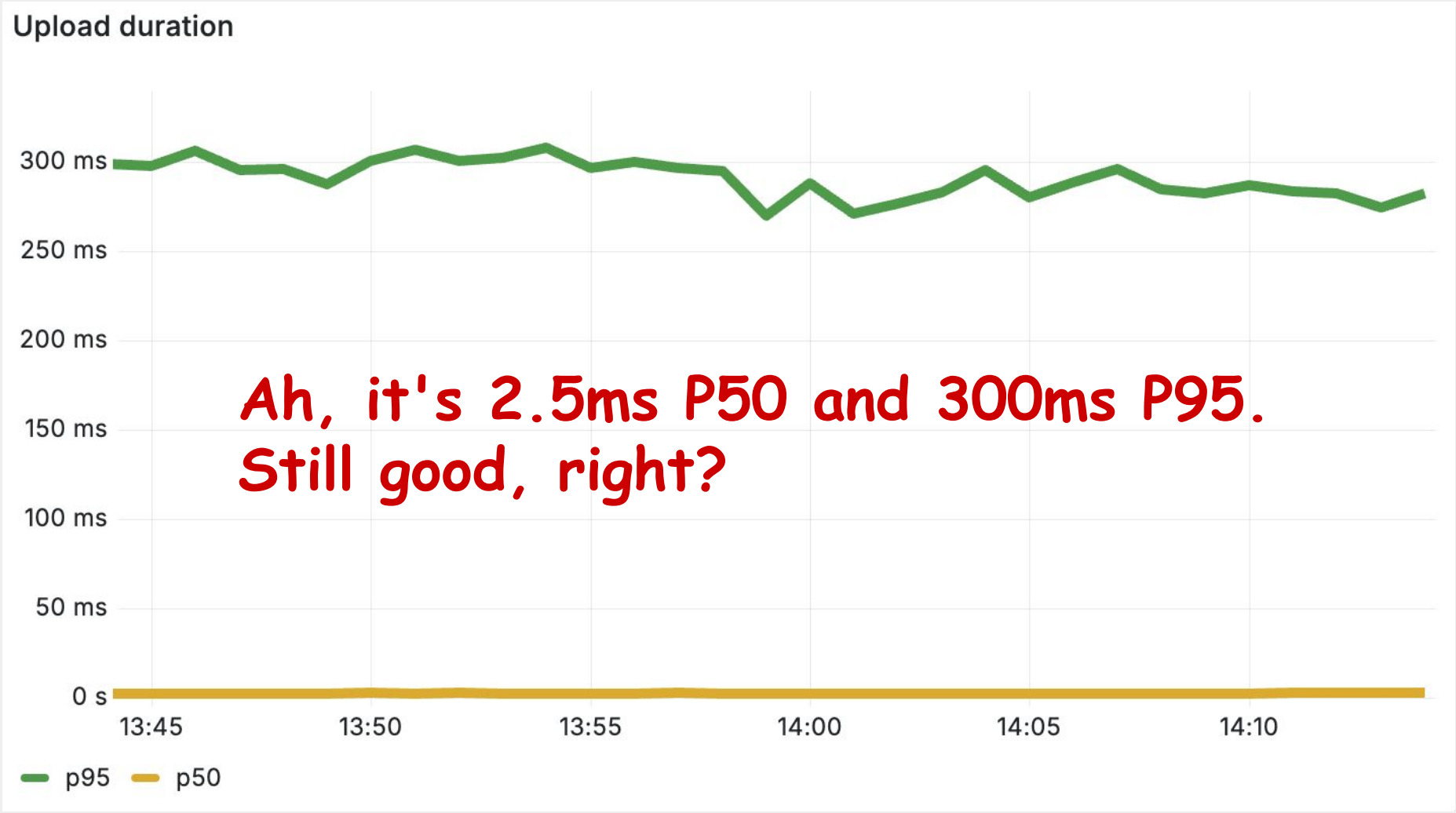
So imagine a metric defined like this

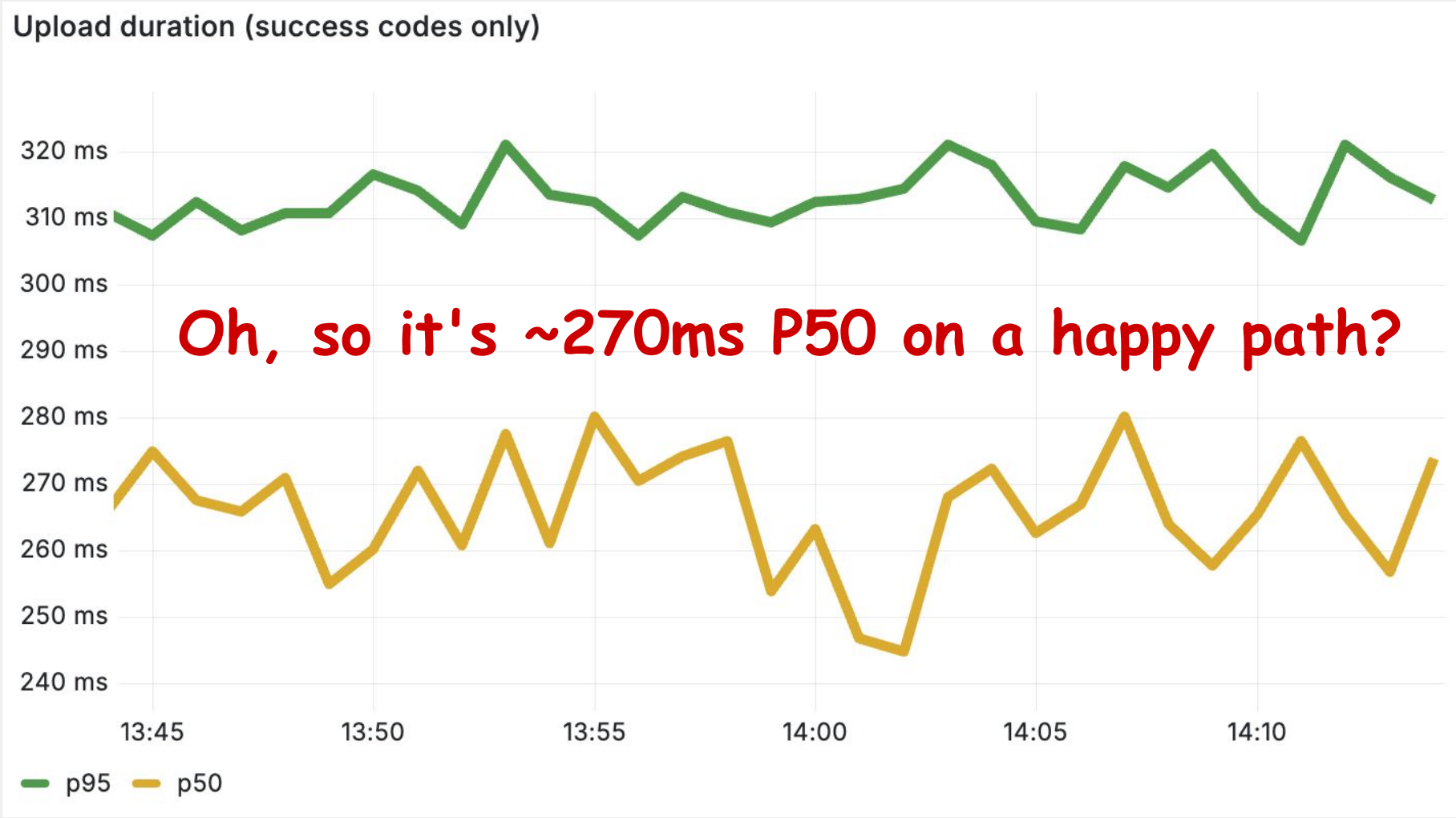
`storage.client.operation.duration` - measures the duration of a storage call

Attributes:

- **`storage.bucket`** - bucket name
- **`storage.operation.name`** - operation name, like `upload`, `download`
- **`server.address`** & **`server.port`** - host name and port


Upload duration
2.8 ms
2.7 ms
2.6 ms
2.5 ms
2.4 ms
2.3 ms
2.5ms seems quite good, right?
13:55
14:00
14:05
{server_address="s3mock", storage_bucket="demo"}


Upload duration
300 ms
250 ms
200 ms
150 ms
100 ms
50 ms
0 s
13:45
13:50
13:55
14:00
14:05
14:10
p95
p50
Ah, it's 2.5ms P50 and 300ms P95.
Still good, right?


Upload duration (success codes only)
320 ms
310 ms
300 ms
290 ms
280 ms
270 ms
260 ms
250 ms
240 ms
13:45
13:50
13:55
14:00
14:05
14:10
p95
p50
Oh, so it's ~270ms P50 on a happy path?

How This Metric *Should* Be Defined

`storage.client.operation.duration` - measures the duration of a storage call

unit: s
instrument: histogram

Attributes:

- **`storage.bucket`** (string) - bucket name
- **`storage.operation.name`** (string) - operation name, like `upload` or `download`
- **`server.address`** (string) & **`server.port`** (int) - host name and port
- **`error.type`** - error category (if operation failed)

Imagine logging approach like this

```
14:49:01.681 info download.start {"storage.object.key":"6c5cfa",
"trace_id":"c7fdc23ffcda...", "span_id":"60d1e2..."}
14:49:01.692 warn download.fail {"exception.type":"NoSuchKey",
"trace_id":"c7fdc23ffcda...", "span_id":"60d1e2..."}


14:49:01.724 info download.start {"storage.object.key":"3b4a21",
"trace_id":"a07488ac1279...", "span_id":"cda074..."}
14:49:01.831 info download.end {"trace_id":"a07488ac1279...",
"span_id":"cda074..."}
```

A query that joins logs to bring the context together

```
SELECT
 s.storage_object_key,
 if(r.event_name = 'download.end', 'ok', 'fail') AS status,
 r.exception_type,
 dateDiff('millisecond', s.start_ts, r.end_ts) AS duration_ms
FROM
(
 SELECT ts AS start_ts, `storage.object.key` AS storage_object_key
 FROM logs
 WHERE trace_id = tid AND span_id = sid AND event_name = 'download.start' AND storage_object_key != ''
) AS s
JOIN logs r ON r.trace_id = tid AND r.span_id = sid
 AND r.event_name IN ('download.end', 'download.fail')
 AND r.`storage.object.key` = s.storage_object_key
 AND r.ts > s.start_ts
ORDER BY s.start_ts;
```

Please don't write it!

Wouldn't life be better if...

The *same* end event was recorded for *all* outcomes and contained *all the context*

```
14:49:01.681 debug download.start {"storage.object.key":"6c5cfa",
"trace_id":"c7fdc23ffcda...", "span_id":"60d1e2..."}

14:49:01.693 warn download.end {"exception.type":"NoSuchKey",
"duration":0.012, "storage.object.key":"6c5cfa", "trace_id":"c7fdc23ffcda...",
"span_id":"60d1e2..."}
```

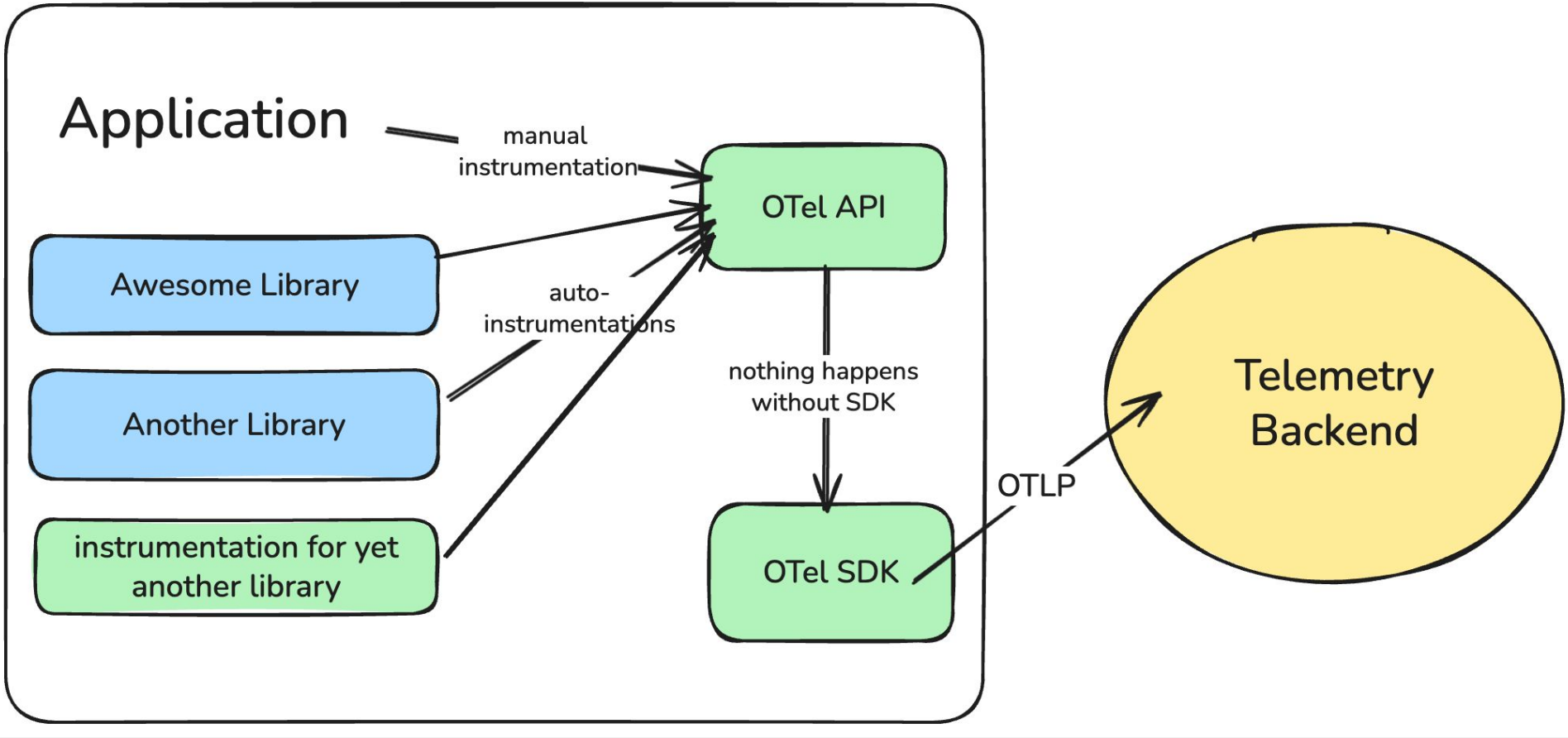
Have I convinced you that **designing** and **documenting** your telemetry **matters**?

Let’s build **tooling for telemetry schemas**?

Turns out OpenTelemetry already did

OpenTelemetry

is a **vendor-neutral** open source **observability framework** for **instrumenting, collecting**, and exporting telemetry data.


Application
manual instrumentation
OTel API
Awesome Library
auto-instrumentations
Another Library
nothing happens without SDK
instrumentation for yet another library
OTel SDK
OTLP
Telemetry Backend

OTLP data model


```
{ "resourceSpans": [ {
   "resource": {"attributes": [ {"key":"service.name", "value": {"stringValue":"demo" }}] },
   "scopeSpans": [
     {
       "scope": { "name":"demo-app.main.py", "schema_url":"https://demo.com/schema/1.0.0-dev" },
       "spans": [
         {
           "traceId": "c7fdc23ffcda07488ac12797ccb7292a",
           "spanId": "c9ec4fde54d6aa5e",
           "parentSpanId": "60d1e29f4560f593",
           "name": "important operation!",
           "startTimeUnixNano": "1544712660000000000",
           "endTimeUnixNano": "1544712661000000000",
           "kind": 2,
           "attributes": [
             { "key": "foo", "value": { "stringValue": "bar" } }
           ]
```

Semantic Conventions

The OTLP data model defines strict format,
Semantic Conventions define how to fill it

HTTP server span in OTLP data model


```
{ "resourceSpans": [ {
  "resource": {"attributes": [ {"key":"service.name", "value": {"stringValue":"demo" }}] },
  "scopeSpans": [
    {
      "scope": {"name":"otel.instr.fastapi","version":"0.60b1","schema_url":"https://otel.io/schemas/1.39.0
      "spans": [
        {
          "traceId": "c7fdc23ffcda07488ac12797ccb7292a" , "spanId": "c9ec4fde54d6aa5e",
          "parentSpanId": "60d1e29f4560f593",
          "startTimeUnixNano" : "1544712660000000000", "endTimeUnixNano" : "1544712661000000000",
          "name": "GET /download/{key}",
          "kind": 2,
          "attributes": [
            { "key": "http.request.method", "value": { "stringValue": "GET" } },
            { "key": "http.response.status_code", "value": { "intValue": 200 } },
            { "key": "http.route", "value": { "stringValue": "/download/{key}" } },
            ...
          ]
        }]}]}]}
```

scale23x: GET /download/{key}

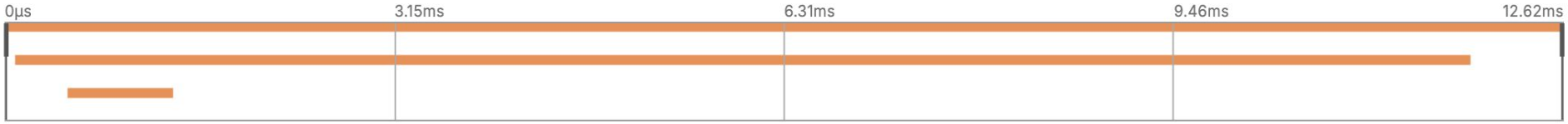
GET 200

Feedback Share

Trace ID 81ea4bd4d9550e138b95a5961fd2fb86 Start time 2026-02-26 18:07:22.583 (9 minutes ago) Duration 12.62ms Services 1 Route /download/{key}

Span Filters

3 spans Prev Next



Service & Operation	0µs	3.15ms	6.31ms	9.46ms	12.62ms
scale23x GET /download/{key} (12.62ms)					

GET /download/{key} Service: scale23x Duration: 12.62ms Start Time: 0µs (18:07:22.583) Child Count: 1

Share

Kind: server Status: unset Library Name: opentelemetry.instrumentation.fastapi

Library Version: 0.60b1

Logs for this span

Span attributes client.address 142.250.69.187 client.port 26348 http.request.method GET http.response.status_code 200 http.route /download/{key} net...

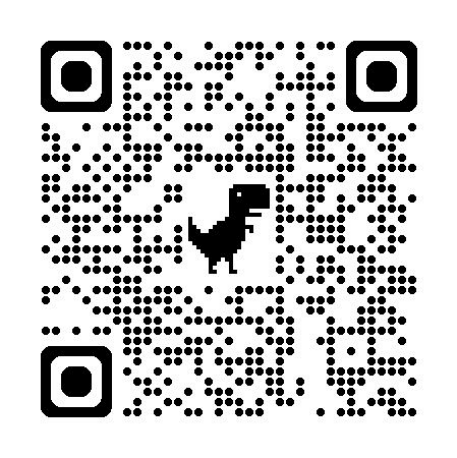
Resource attributes service.name scale23x telemetry.sdk.language python telemetry.sdk.name opentelemetry telemetry.sdk.version 1.39.1

SpanID: 69475de512e4f192

download demo (11.78ms)		11.78r
GET (853.54µs)	853.54µs	

How we document conventions in OTel

https://opentelemetry.io/docs/specs/semconv/



More importantly: they are machine readable

https://github.com/open-telemetry/semantic-conventions/tree/main/model

Demo

https://github.com/lmolkova/semconv-scale23x-demo

So markdown docs are generated, right?

```
$ weaver registry generate \
        --registry ./conventions \
        --v2 \
        markdown \
        ./docs
```

Demo

Docs are cool, what about code?

```
$ weaver registry generate \
        --registry ./conventions \
        --v2 \
        python
        ./conventions_py
```

Demo

Even better: validate telemetry against schema

```
$ weaver registry live-check \
    --registry ./conventions \
    --emit-otlp-logs \
    --otlp-logs-endpoint=http://endpoint:4317 \
    --v2 --output==none --inactivity-timeout=0
```

Use it in your unit or integration tests, automate compliance testing in CI or continuously monitor your production telemetry

Demo

Versioning and evolving the schema

- Telemetry schema is like public API
- Avoid breaking changes
- Use SemVer 2.0.0
- Stamp `schema_url` on all telemetry (includes version)
- Test your schemas including backward compatibility with Rego policies

Validating your schema

```
$ weaver registry check \
    --policy ./policies \
    --registry ./conventions \
     --baseline-registry https://github/repo... \
    --v2
```

Use common checks in otel-weaver-packages or write your own

Demo

Summary

- Telemetry schema is like public API
- It should be documented, validated, versioned, and enforced
- OpenTelemetry has language and tooling (weaver) to make it all possible

Where to find us

- weaver and semantic-conventions on GitHub
- #otel-weaver CNCF slack channel
- OTel SemConv Tooling SIG meetings

We’d love to know how you’re using Weaver, come say hi!

Want to contribute?

Join us in building:

- Live checks
- MCP server
- Code generation
- Telemetry version upgrades
- And more

Love Rust? Contribute to **Weaver**.

Prefer another language? Help build **code generation for your ecosystem**.

Thank you!