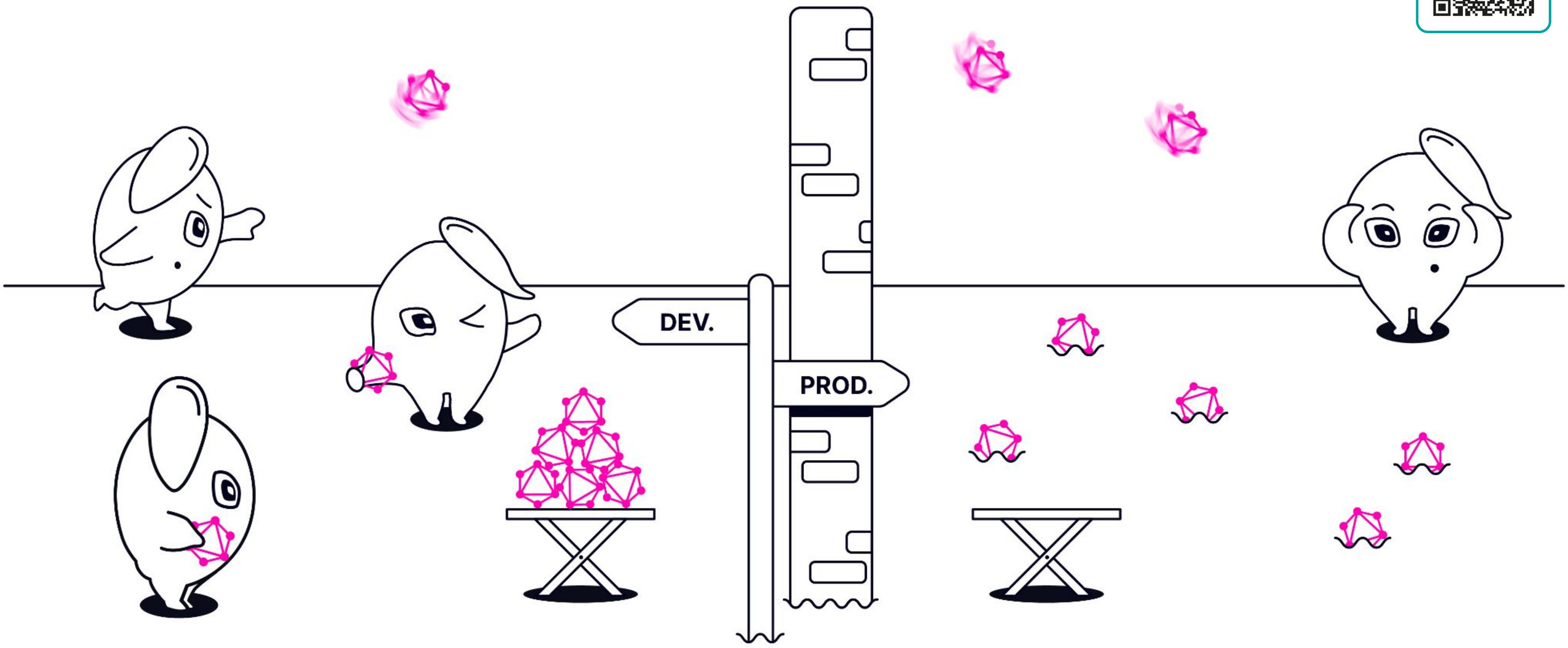
Tyk

What could go wrong with a GraphQL query and can OpenTelemetry help?

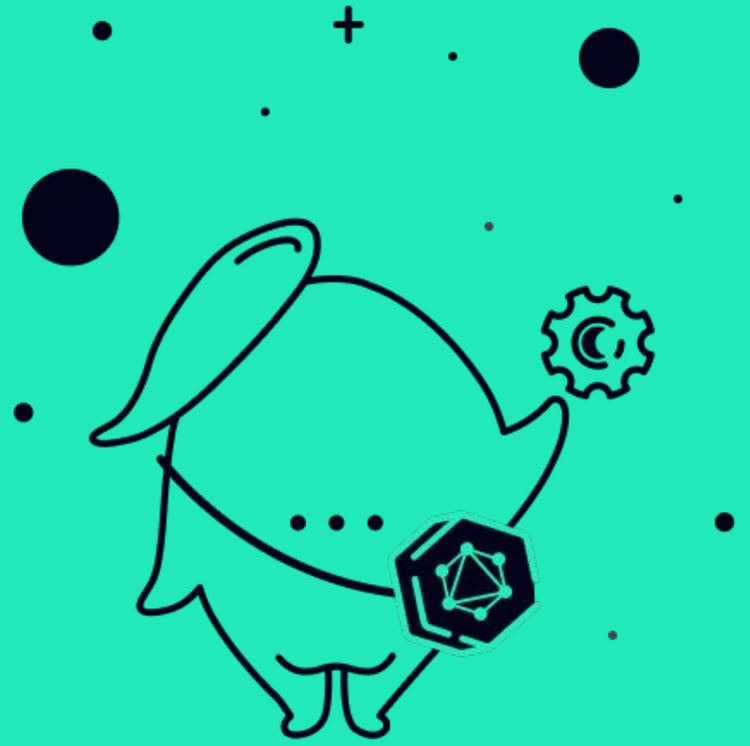
Sonja Chevre & Ahmet Soormally


DEV.
PROD.

DECLARATIVE QUERY LANGUAGE FOR THE INTERNET

GraphQL 101

1. What kinds of problems does GraphQL solve?
2. What are the general use-cases for GraphQL?
3. What does a query & schema look like?
4. How is GraphQL different from your typical REST API?



Tyk

GraphQL 101

SQL

```
SELECT
    authors.name,
    posts.title
FROM
    posts
LEFT JOIN authors
  ON authors.id = posts.author_id
WHERE posts.status = ‘published’;
```

REST

```
GET /posts?status=published

GET /authors/2
GET /authors/7
GET /authors/9
...
```

GRAPHQL

```
query {
  postsByStatus(status: "published") {
    title
    author {
      name
    }
  }
}
```

Tyk

GRAPHQL 101

Overfetching

SQL

```
SELECT
    authors.name,
    posts.title
FROM
    posts
LEFT JOIN authors
  ON authors.id = posts.author_id
WHERE posts.status = ‘published’;
```

REST

GET /posts?status=published

GET /authors/2
GET /authors/7
GET /authors/9

GRAPHQL

```
query {
  postsByStatus(status: "published") {
    title
    author {
      name
    }
  }
}
```

Tyk

Underfetching

SQL

```
SELECT
    authors.name,
    posts.title
FROM
    posts
LEFT JOIN authors
  ON authors.id = posts.author_id
WHERE posts.status = ‘published’;
```

REST

```
GET /posts?status=published
```

GET /authors/2
GET /authors/7
GET /authors/9
...

GRAPHQL

```
query {
  postsByStatus(status: "published") {
    title
    author {
      name
    }
  }
}
}
```

Tyk

GraphQL 101

Schema

```
type Query {
  getPostsByStatus(status: String!): [Post]
}

type User {
  id: ID!
  name: String!
  email: String!
  posts: [Post]
  comments: [Comment]
}

type Post {
  title: String
  content: String
  status: String
  author: User
  comments: [Comment]
}

type Comment {
  id: ID!
  author: User
  comment: String
}
```

Operation

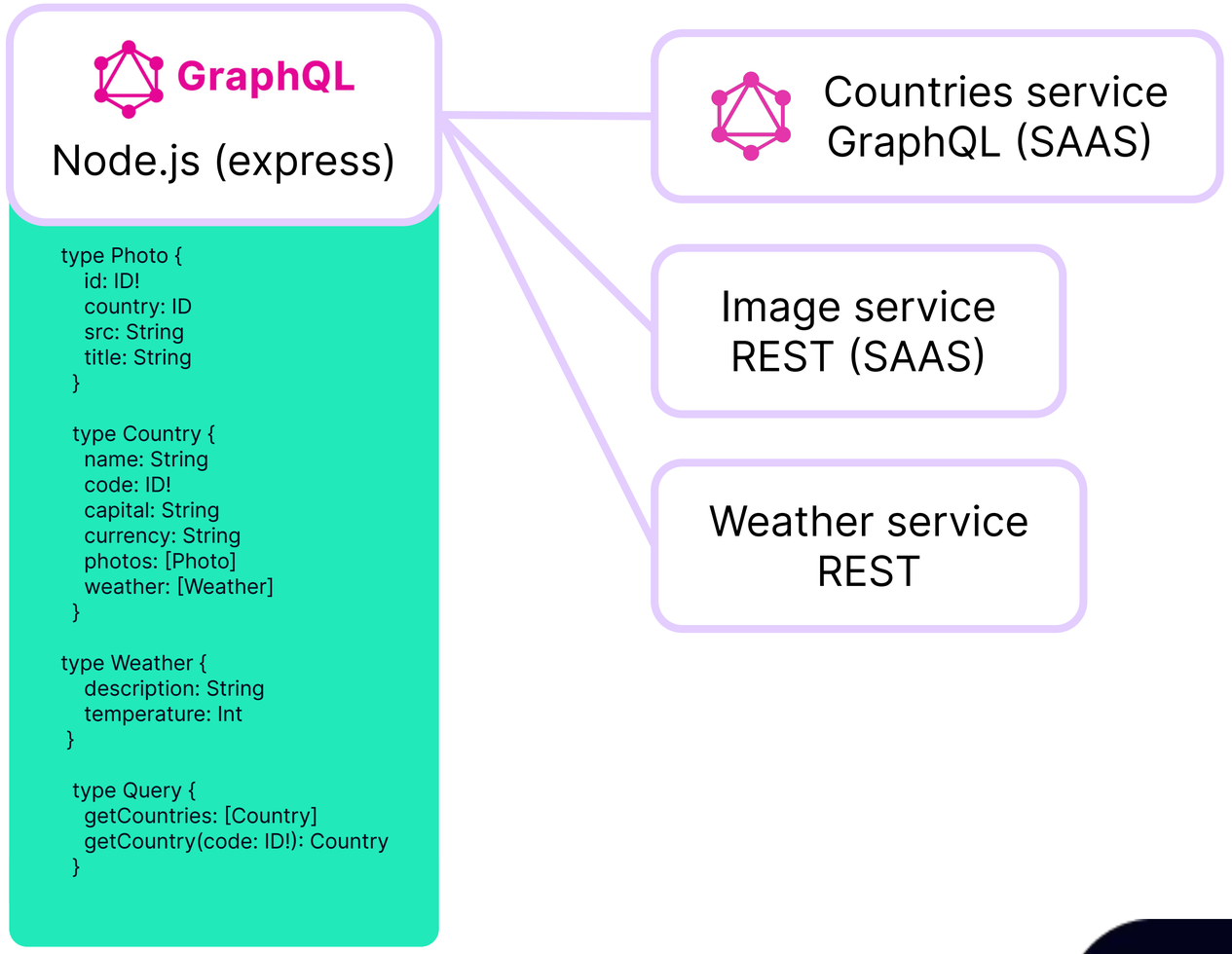
```
query {
  getPostsByStatus(status: "published") {
    title
    author {
      name
    }
  }
}
```

Response

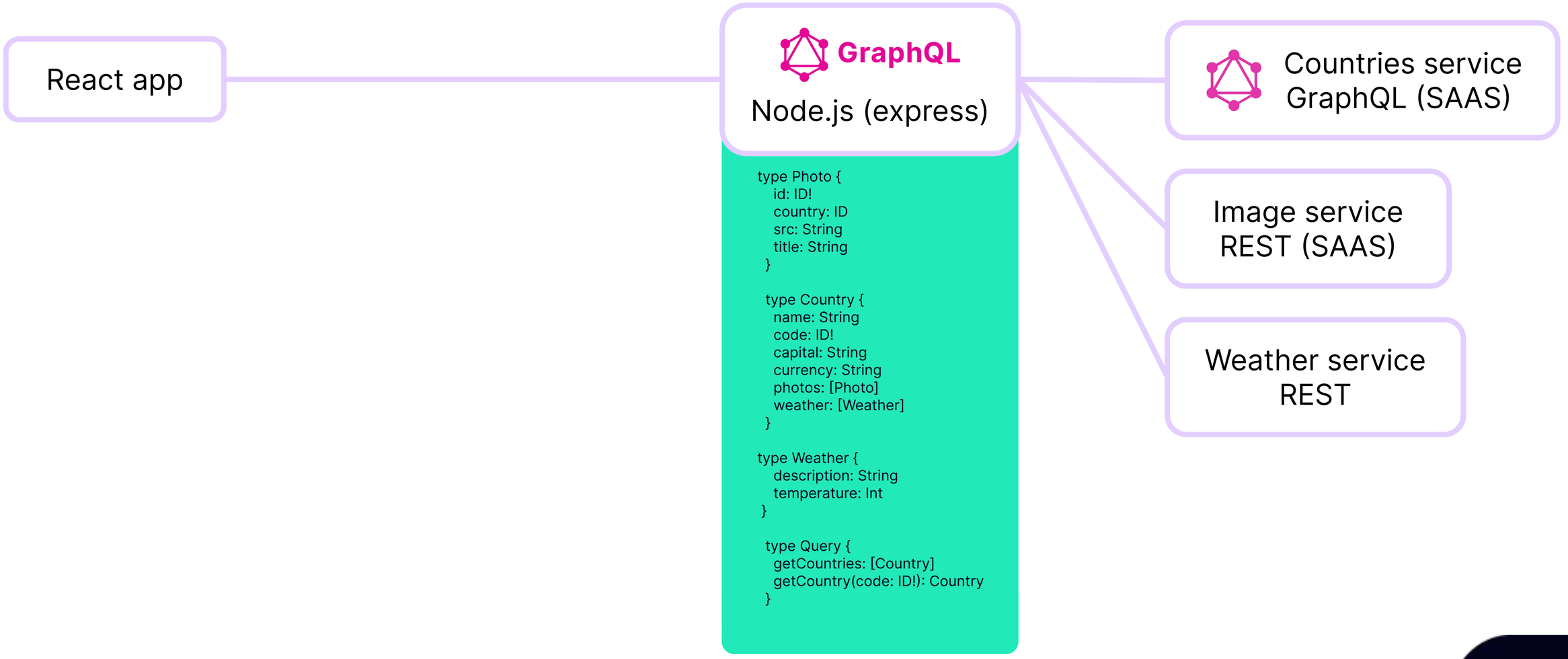
```
{
  “data”: [
    {
      “title”: “abc”,
      “author”: {
        “name”: “Ahmet”
      }
    },
    {
      “title”: “def”,
      “author”: {
        “name”: “Sonja”
      }
    },
  ]
}
```

Tyk

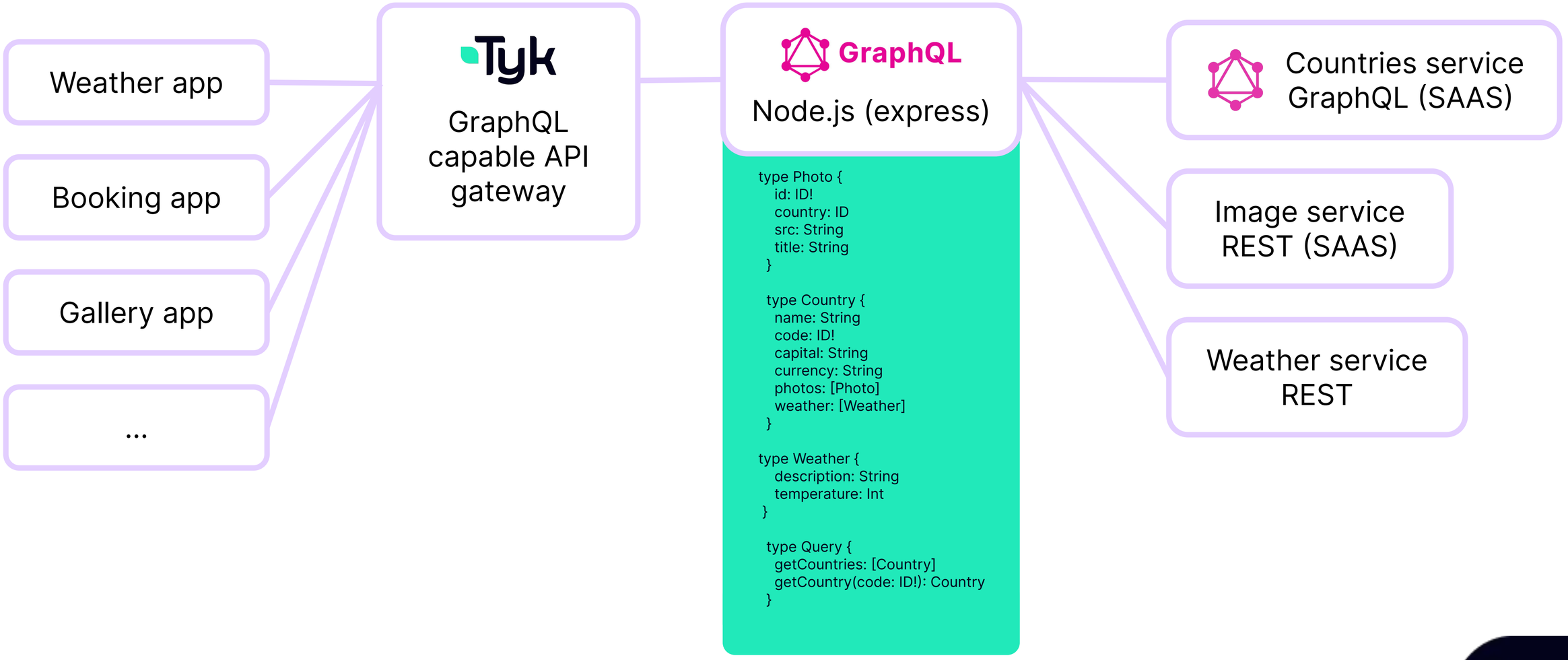
Kube travel - Server side



Kube travel app



Kube travel API product



How do we monitor GraphQL in production?

Let’s apply the **RED** method to GraphQL

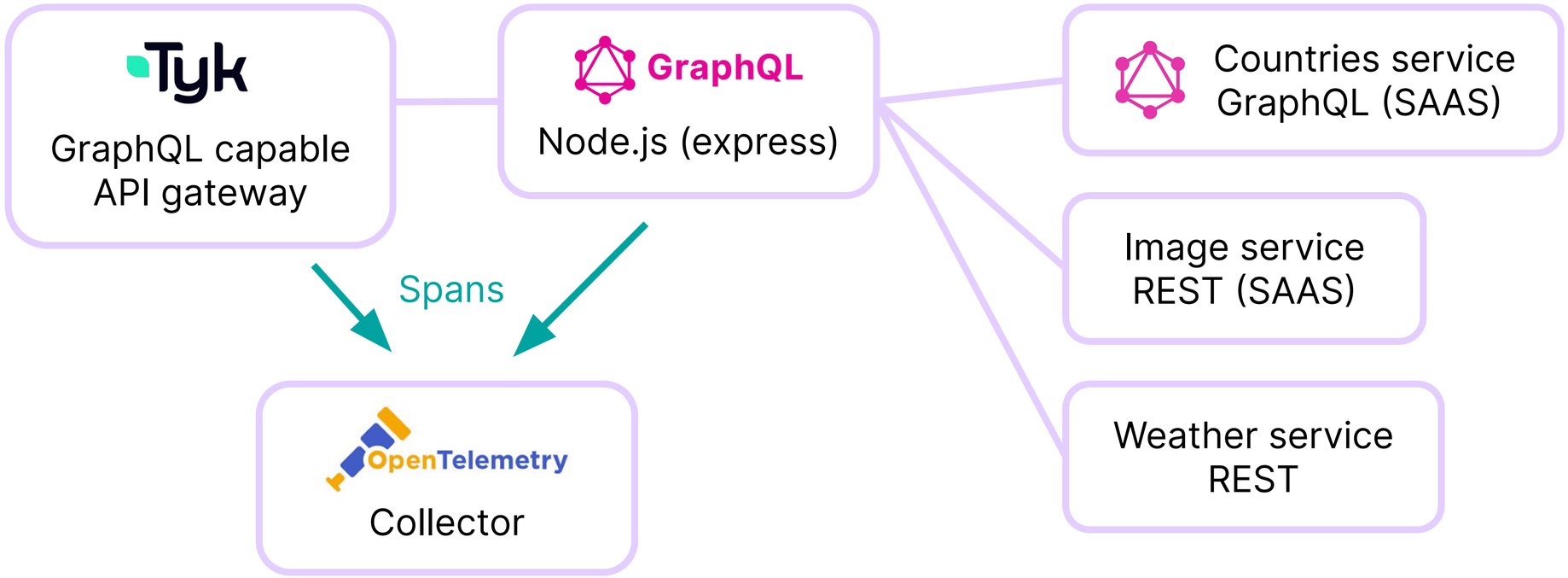
R Rate - the number of requests, per second, your services are serving.

E Errors - the number of failed requests per second.

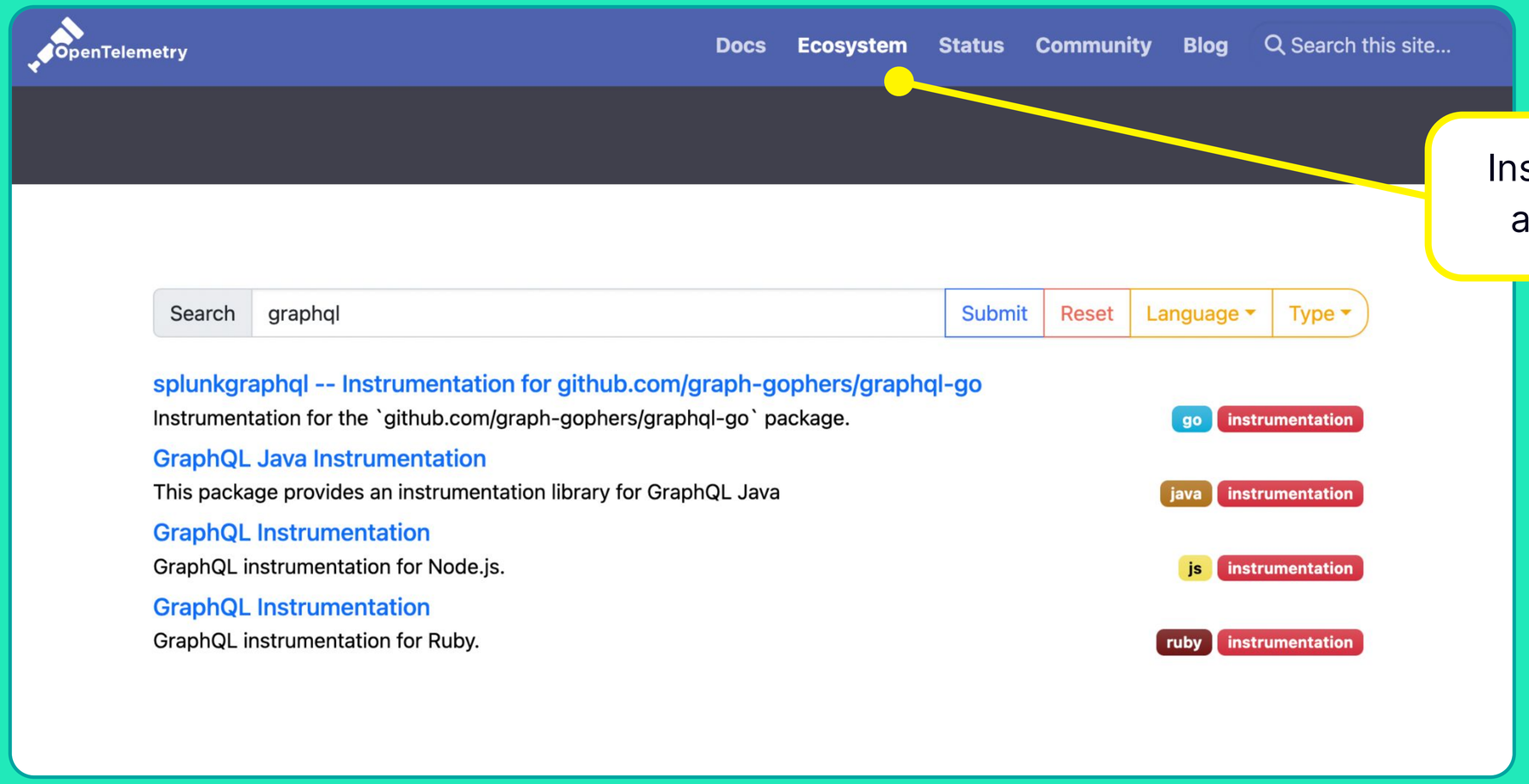
D Duration - distributions of the amount of time each request takes.

Tyk

Adding OpenTelemetry to Kube Travel



Instrumenting GraphQL with OpenTelemetry



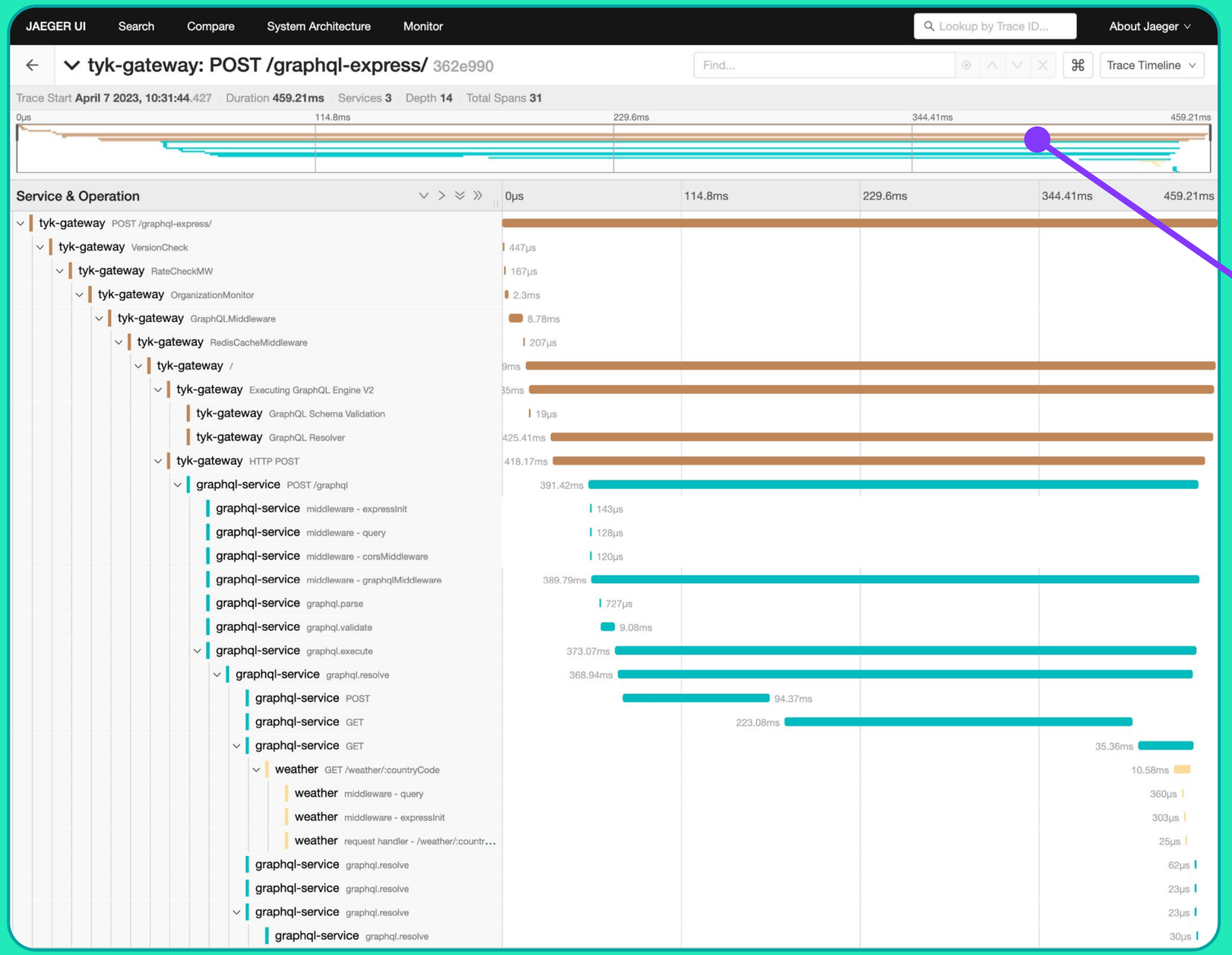
Instrumentation libraries available for GraphQL

Tyk

GraphQL instrumentation

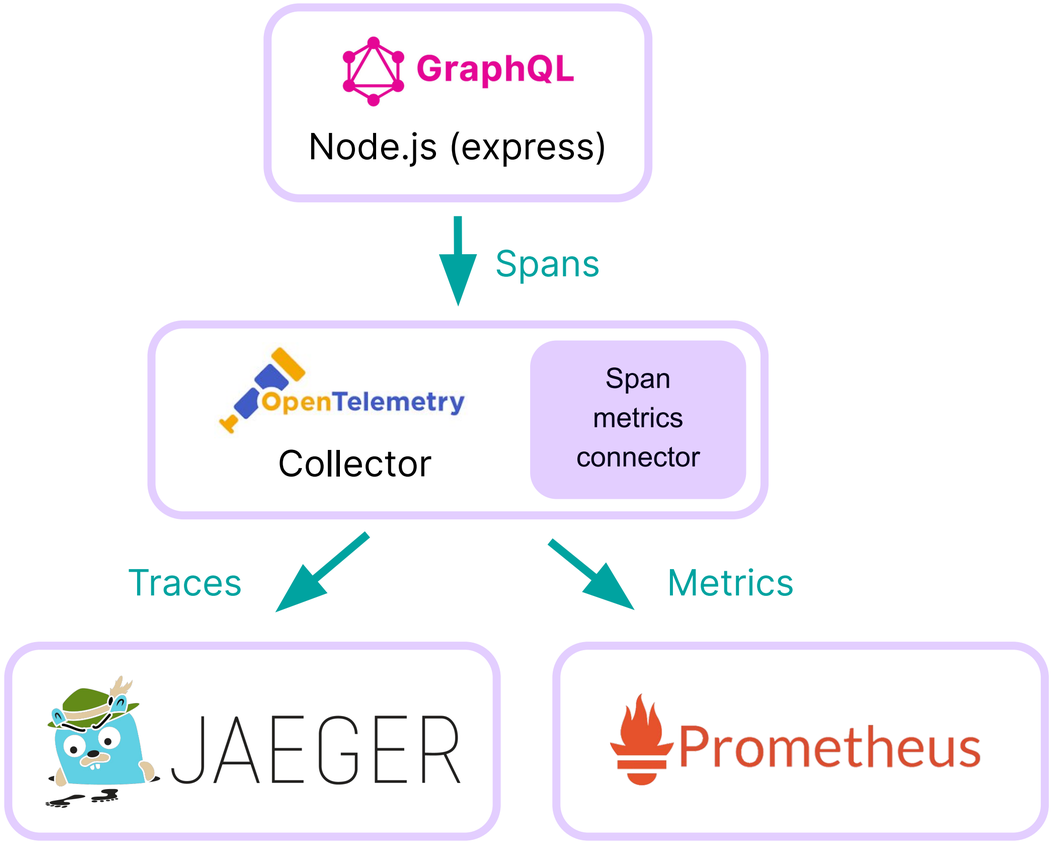
graphql-server > JS tracer.js > ...

```
/*tracing.js*/
const opentelemetry = require("@opentelemetry/sdk-node");
const { OTLPTraceExporter } = require("@opentelemetry/exporter-trace-otlp-http");
const { GraphQLInstrumentation } = require('@opentelemetry/instrumentation-graphql');
const { HttpInstrumentation } = require('@opentelemetry/instrumentation-http');
const { ExpressInstrumentation } = require('@opentelemetry/instrumentation-express');
const { Resource } = require('@opentelemetry/resources');
const { SemanticResourceAttributes } = require('@opentelemetry/semantic-conventions');


// Receiving service
const { context, propagation, trace } = require('@opentelemetry/api');

// Assume "input" is an object with 'traceparent' & 'tracestate' keys
const input = {};

// Extracts the 'traceparent' and 'tracestate' data into a context object.
//
// You can then treat this context as the active context for your
// traces.
let activeContext = propagation.extract(context.active(), input);


const sdk = new opentelemetry.NodeSDK({
  traceExporter: new OTLPTraceExporter({
    url: "http://localhost:4318/v1/traces",
    // optional - collection of custom headers to be sent with each request, empty by default
    headers: {},
  }),
  instrumentations: [
      new GraphQLInstrumentation({
        // allowAttributes: true,
        // depth: 2,
        // mergeItems: true,
      }),
      new HttpInstrumentation(),
      new ExpressInstrumentation()
      ],
  resource: new Resource({
      [SemanticResourceAttributes.SERVICE_NAME]: 'graphql-service',
    }),
});
sdk.start();
```



An end-to-end distributed trace in Jaeger

RED metrics from traces with Jaeger



Span Metrics Connector creates two metrics based on spans:

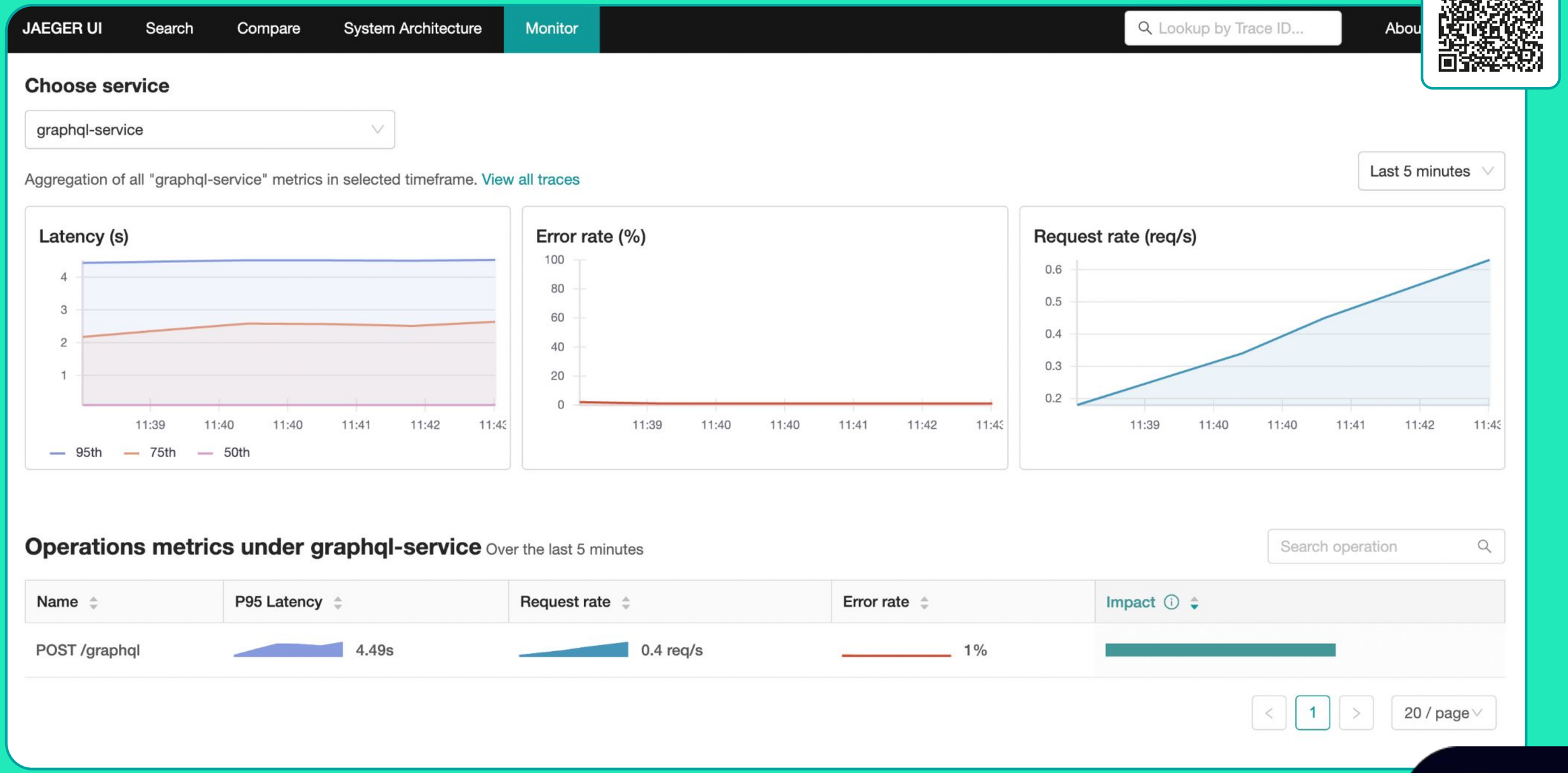
`Calls_total`

- Counts the total number of spans, including error spans. Call counts are differentiated from errors via the `status_code` label. Errors are identified as any time series with the label `status_code = "STATUS_CODE_ERROR"`.

`Latency`

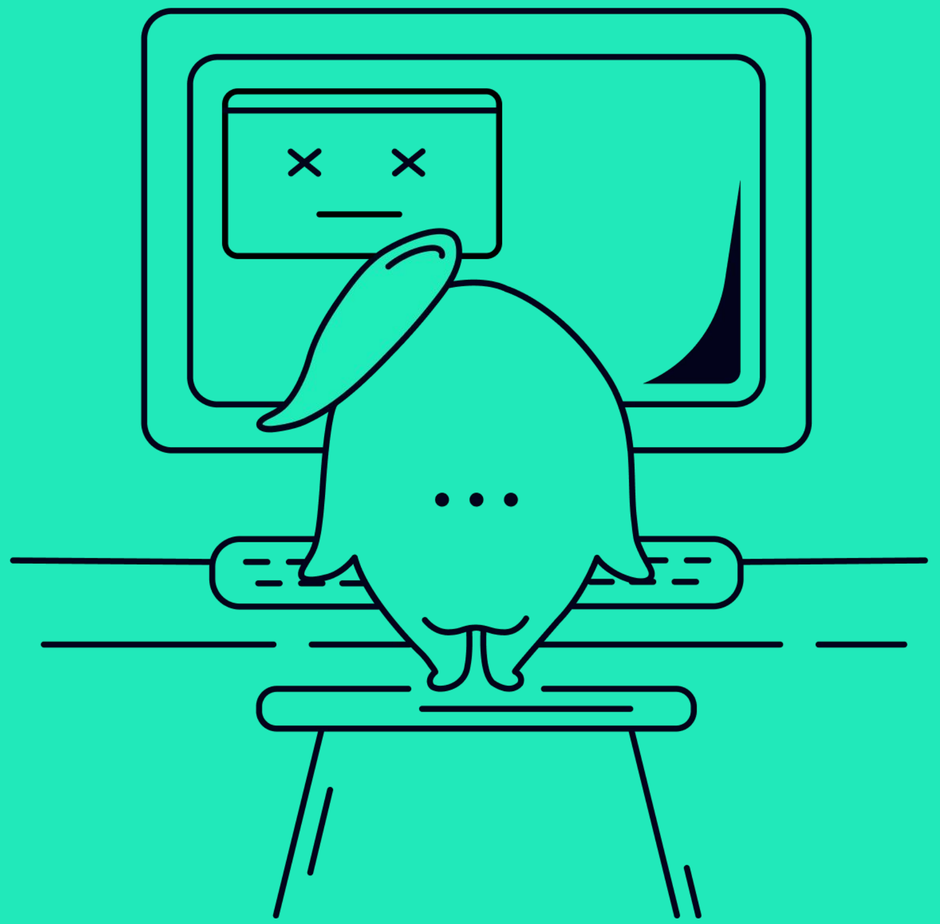
- `latency_count`: The total number of data points across all buckets in the histogram.
- `latency_sum`: The sum of all data point values.
- `latency_bucket`: A collection of `n` time series (where `n` is the number of latency buckets) for each latency bucket identified by an `le` (less than or equal to) label. The `latency_bucket` counter with lowest `le` and `le >= span latency` will be incremented for each span.

Tyk


JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
Abou
Choose service
graphql-service
Aggregation of all "graphql-service" metrics in selected timeframe. View all traces
Last 5 minutes
Latency (s)
4
3
2
1
11:39
11:40
11:40
11:41
11:42
95th
75th
50th
Error rate (%)
100
80
60
40
20
0
11:39
11:40
11:40
11:41
11:42
Request rate (req/s)
0.6
0.5
0.4
0.3
0.2
11:39
11:40
11:40
11:41
11:42
Operations metrics under graphql-service Over the last 5 minutes
Search operation
Name
P95 Latency
Request rate
Error rate
Impact
POST /graphql
4.49s
0.4 req/s
1%
1
20 / page

Errors

1. Upstream errors
2. Resolver Errors



Tyk

Tyk

Explorer Prettify

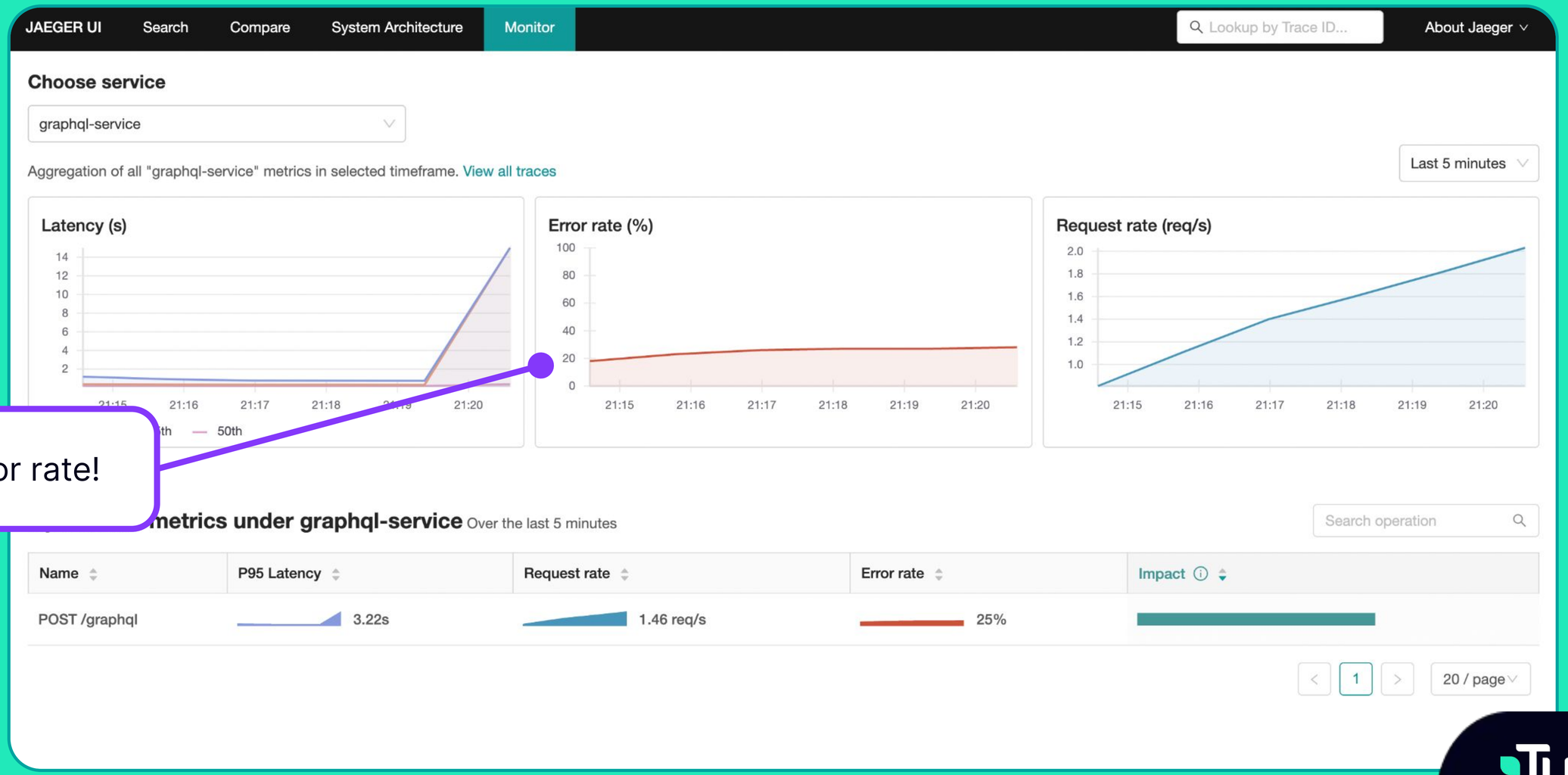
```
{ country (code: "IT")
  {
    code
    name
    capital
    weather {
      temperature
    }
  }
}
```

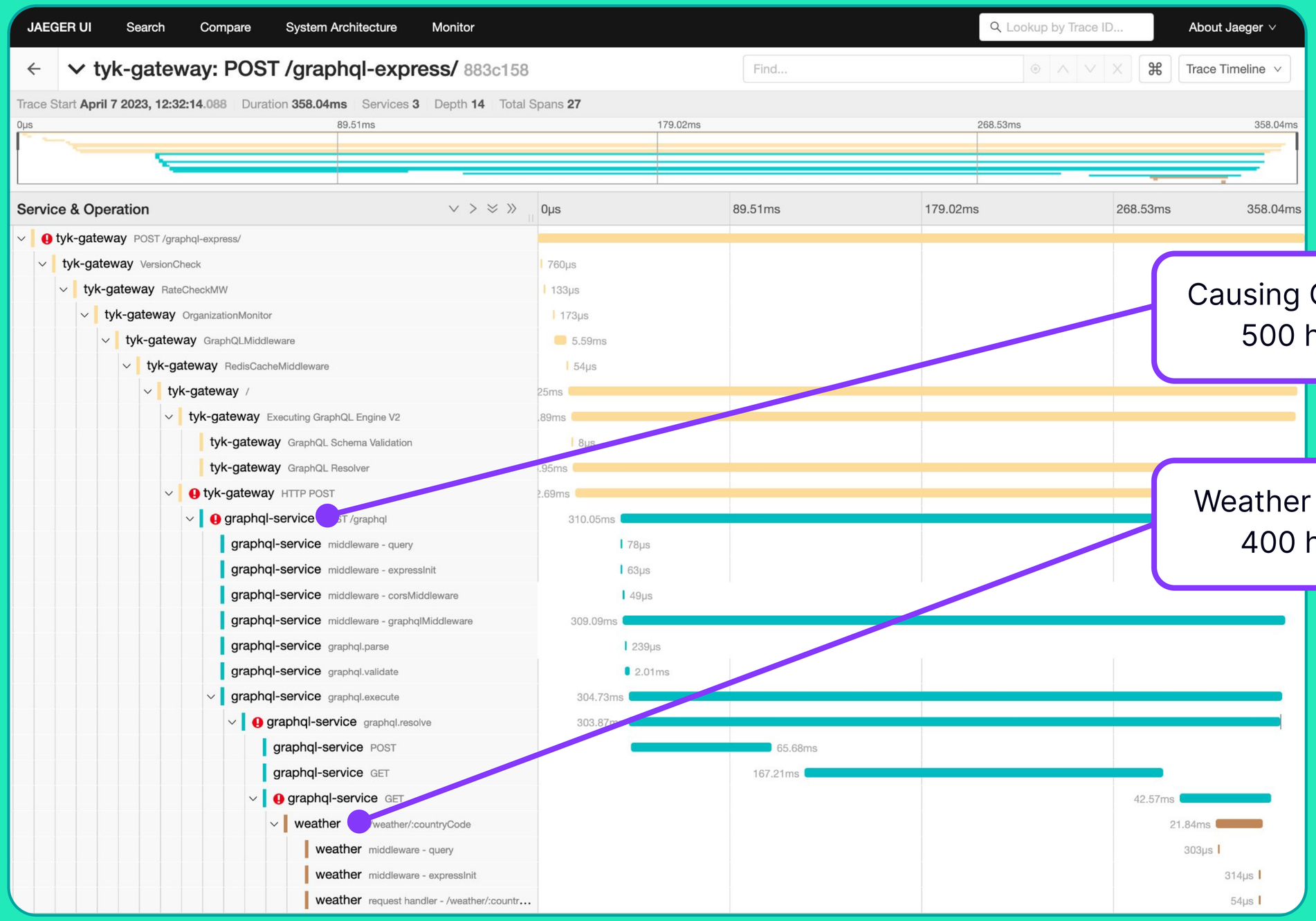
QUERY VARIABLES REQUEST HEADERS


```
{
  "errors": [
    {
      "message": "Failed to fetch country data",
      "locations": [
        {
          "line": 1,
          "column": 16
        }
      ],
      "path": [
        "country"
      ]
    },
    {
      "message": "unable to resolve",
      "locations": [
        {
          "line": 1,
          "column": 3
        }
      ],
      "path": [
        "country"
      ]
    }
  ],
  "data": null
}
```


Tyk

Something is going on...

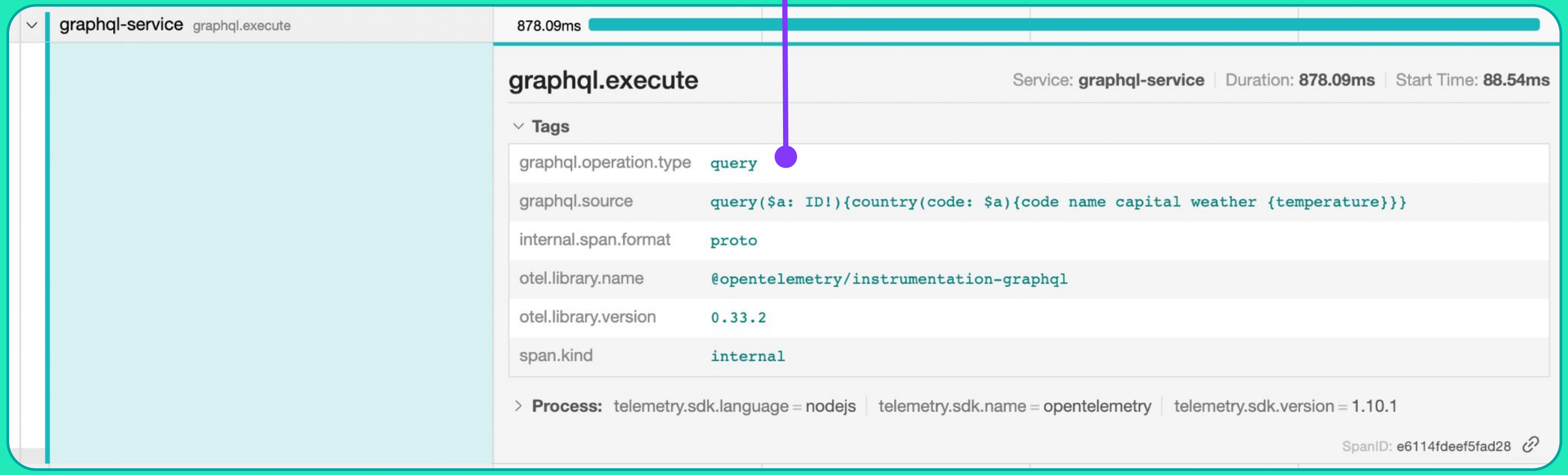




Causing GraphQL to return a 500 http status code

Weather service returning a 400 http status code

The GraphQL query that is causing the error



Tyk ▶ Explorer Prettify

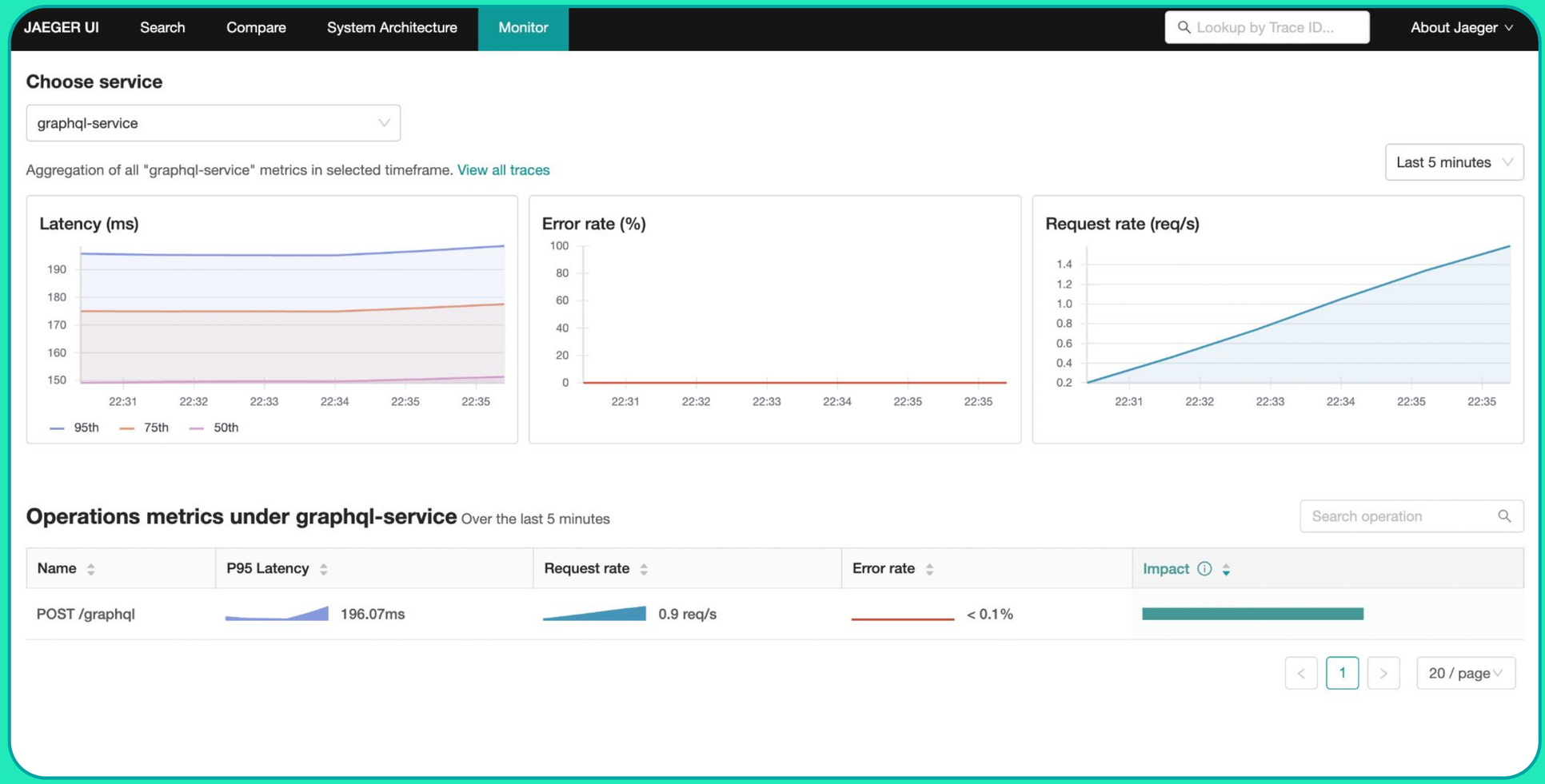
```
{ country (code: "NL")
  {
    code
    name
    weather {
      temperature
    }
  }
}
```

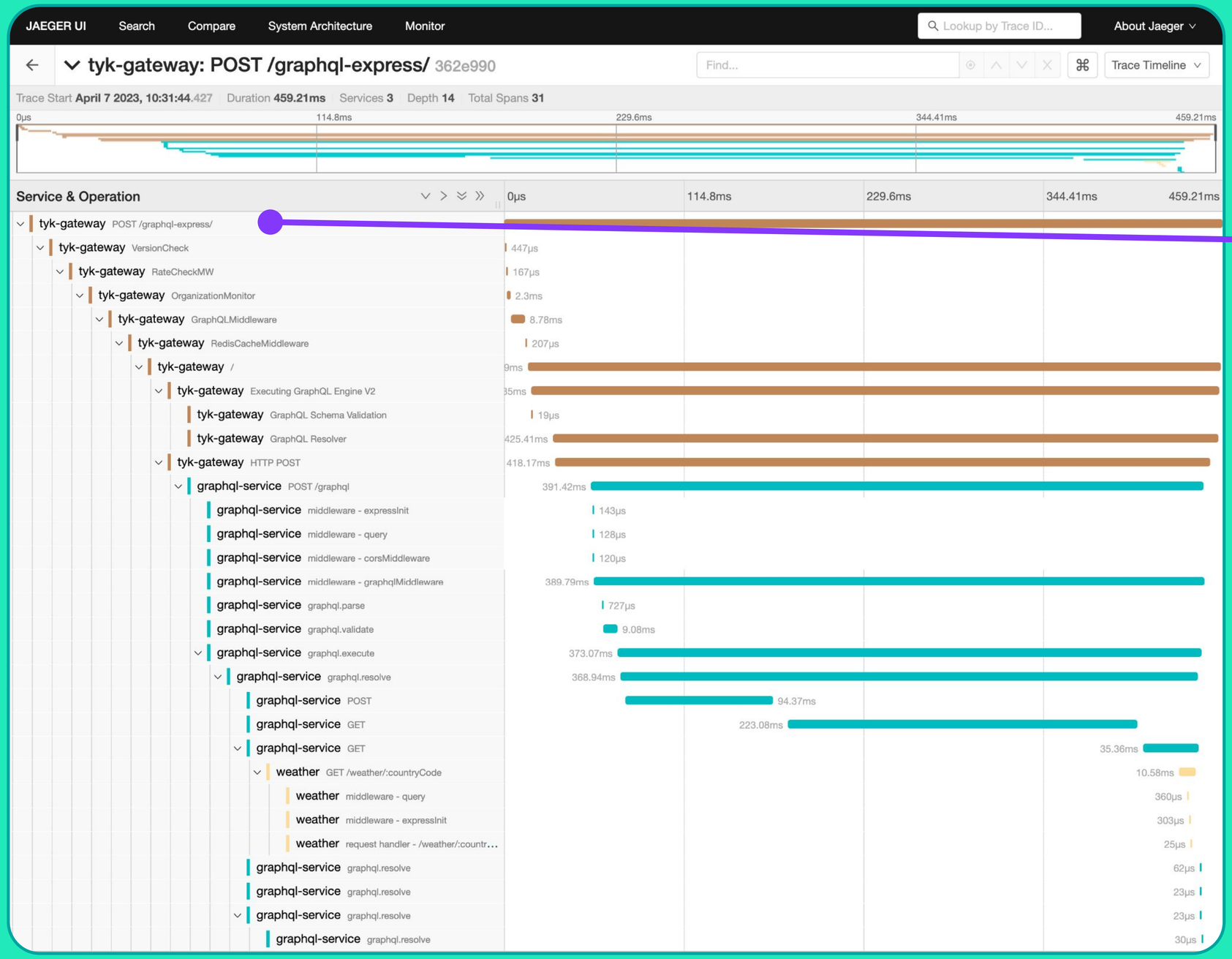
QUERY VARIABLES REQUEST HEADERS


```
{
  "errors": [
    {
      "message": "Cannot return null for non-nullable field Weather.temperature.",
      "locations": [
        {
          "line": 1,
          "column": 53
        }
      ],
      "path": [
        "country",
        "weather",
        "temperature"
      ]
    }
  ],
  "data": {
    "country": {
      "code": "NL",
      "name": "Netherlands",
      "weather": null
    }
  }
}
```


Tyk

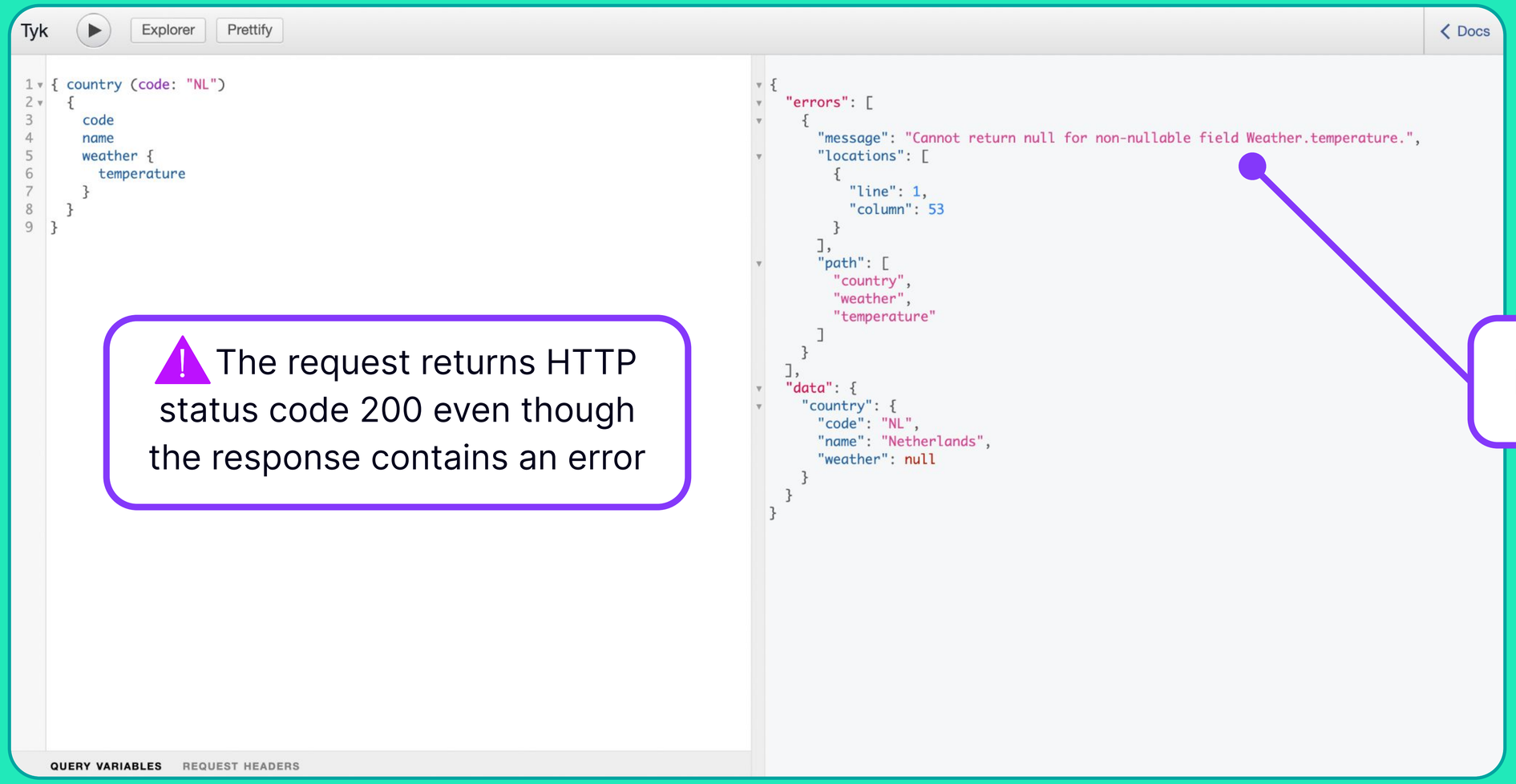
Everything looks fine




JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaeger
tyk-gateway: POST /graphql-express/ 362e990
Find...
Trace Timeline
Trace Start April 7 2023, 10:31:44.427 Duration 459.21ms Services 3 Depth 14 Total Spans 31
0µs
114.8ms
229.6ms
344.41ms
459.21ms
Service & Operation
tyk-gateway POST /graphql-express/
tyk-gateway VersionCheck 447µs
tyk-gateway RateCheckMW 167µs
tyk-gateway OrganizationMonitor 2.3ms
tyk-gateway GraphQLMiddleware 8.78ms
tyk-gateway RedisCacheMiddleware 207µs
tyk-gateway / 9ms
tyk-gateway Executing GraphQL Engine V2 35ms
tyk-gateway GraphQL Schema Validation 19µs
tyk-gateway GraphQL Resolver 425.41ms
tyk-gateway HTTP POST 418.17ms
graphql-service POST /graphql 391.42ms
graphql-service middleware - expressInit 143µs
graphql-service middleware - query 128µs
graphql-service middleware - corsMiddleware 120µs
graphql-service middleware - graphqlMiddleware 389.79ms
graphql-service graphql.parse 727µs
graphql-service graphql.validate 9.08ms
graphql-service graphql.execute 373.07ms
graphql-service graphql.resolve 368.94ms
graphql-service POST 94.37ms
graphql-service GET 223.08ms
graphql-service GET 35.36ms
weather GET /weather/:countryCode 10.58ms
weather middleware - query 360µs
weather middleware - expressInit 303µs
weather request handler - /weather/:countr... 25µs
graphql-service graphql.resolve 62µs
graphql-service graphql.resolve 23µs
graphql-service graphql.resolve 23µs
graphql-service graphql.resolve 30µs

All good!

GraphQL errors



Adding GraphQL to error spans

Span Status MUST be left unset if HTTP status code was in the 1xx, 2xx or 3xx ranges, unless there was another error (e.g., network error receiving the response body; or 3xx codes with max redirects exceeded), in which case status MUST be set to `Error`.

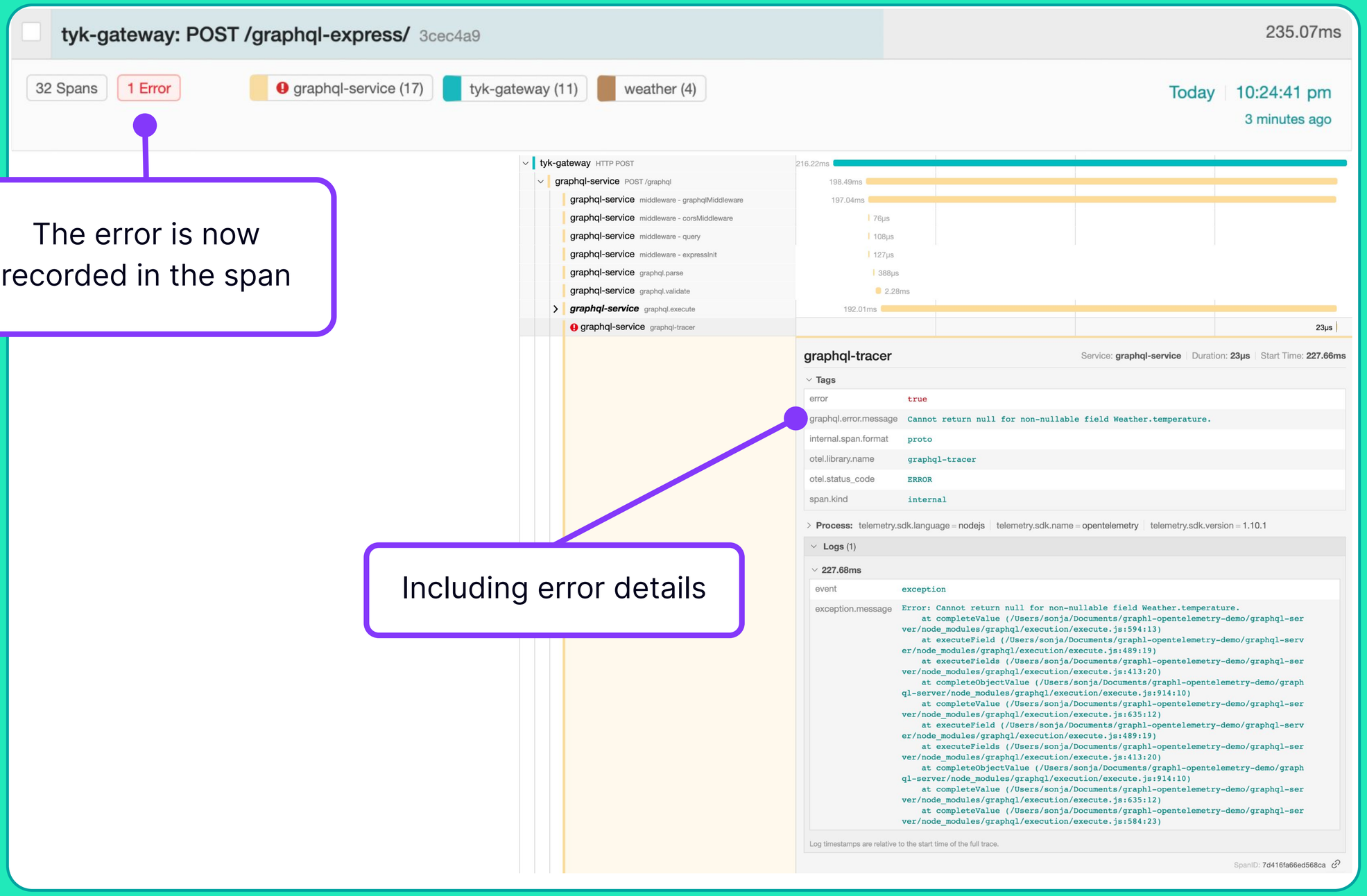
Attribute	Type	Examples
`graphql.operation.name`	string	`findBookById`
`graphql.operation.type`	string	`query`; `mutation`; `subscription`
`graphql.document`	string	`query findBookById { bookById(id: ?) { name } }`

Is there a semantic convention we can use?

Tyk

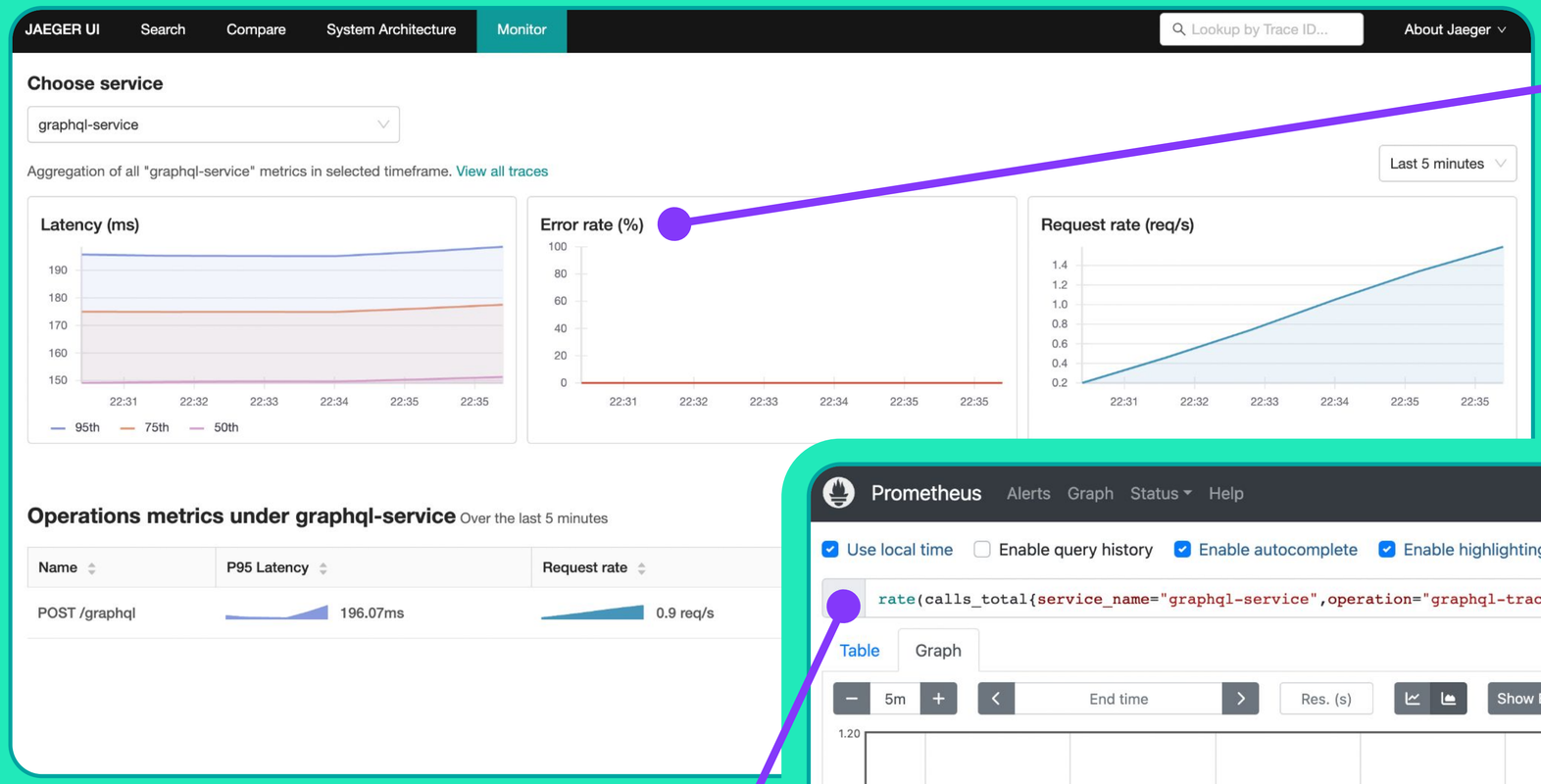
Report the error to the active span with manual instrumentation

```
280  // Create an Express app
281  const app = express();
282  app.use(cors());
283
284  // Add a GraphQL API endpoint
285  app.use(
286    '/graphql',
287    graphqlHTTP({
288      schema: schema,
289      rootValue: root,
290      graphiql: true,
291      formatError: error => {
292        const params = {
293          message  : error.message,
294          stack    : error.stack
295        };
296
297        tracer.startActiveSpan('graphql-tracer', (span) => {
298          span.setAttribute('graphql.error.message', error.message);
299          span.recordException(error.stack);
300          span.setStatus({ code: opentelemetry.SpanStatusCode.ERROR });
301          span.end();
302
303        });
304
305        return (params);
306      }
307    }),
308  );
309
310  // Start the server
311  app.listen(4000, () => {
312    console.log('GraphQL server running on http://localhost:4000/graphql');
313  });
```



The error is now recorded in the span

Including error details

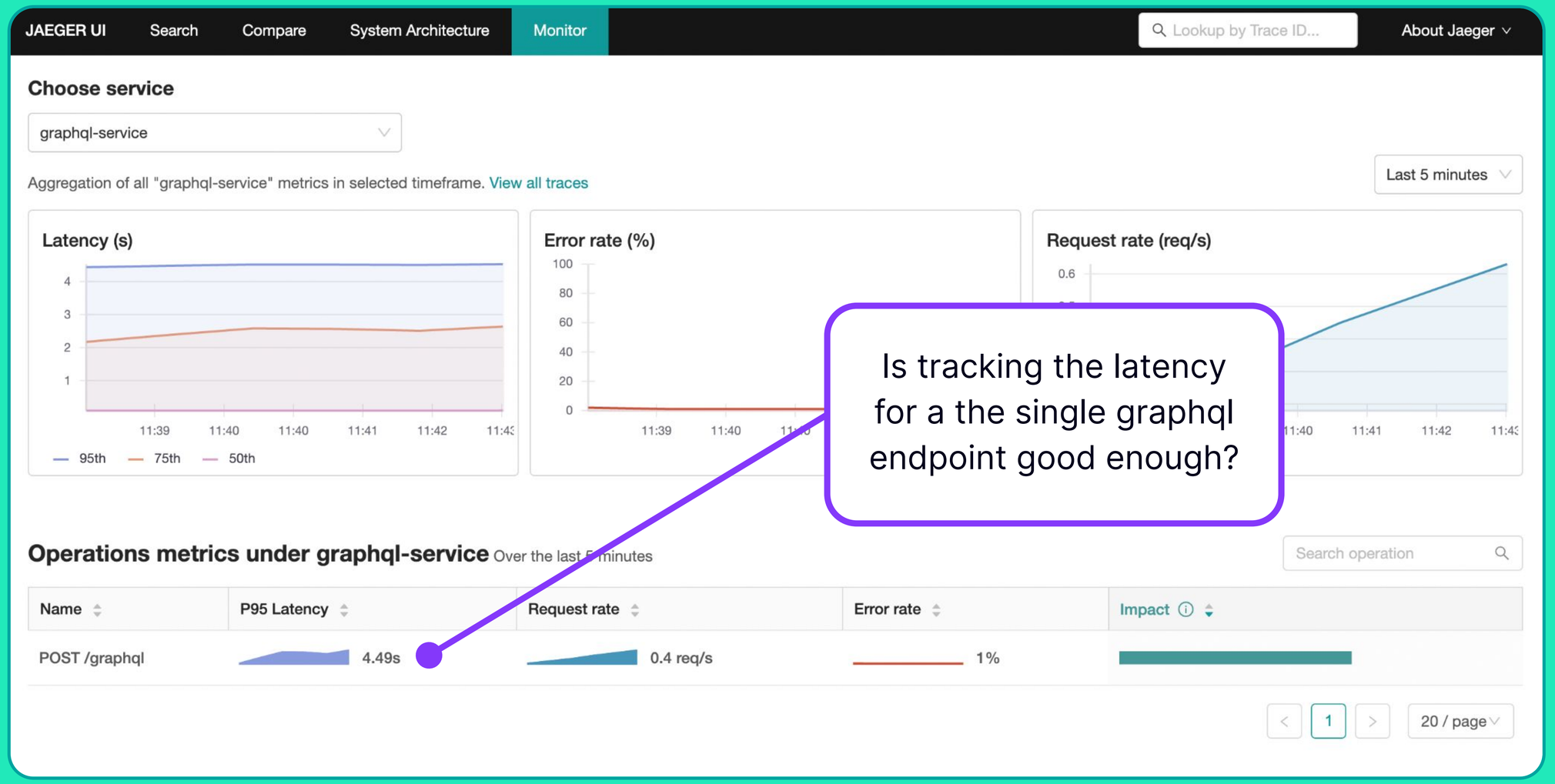


Still no error

But we can use Prometheus to report error rates for GraphQL queries

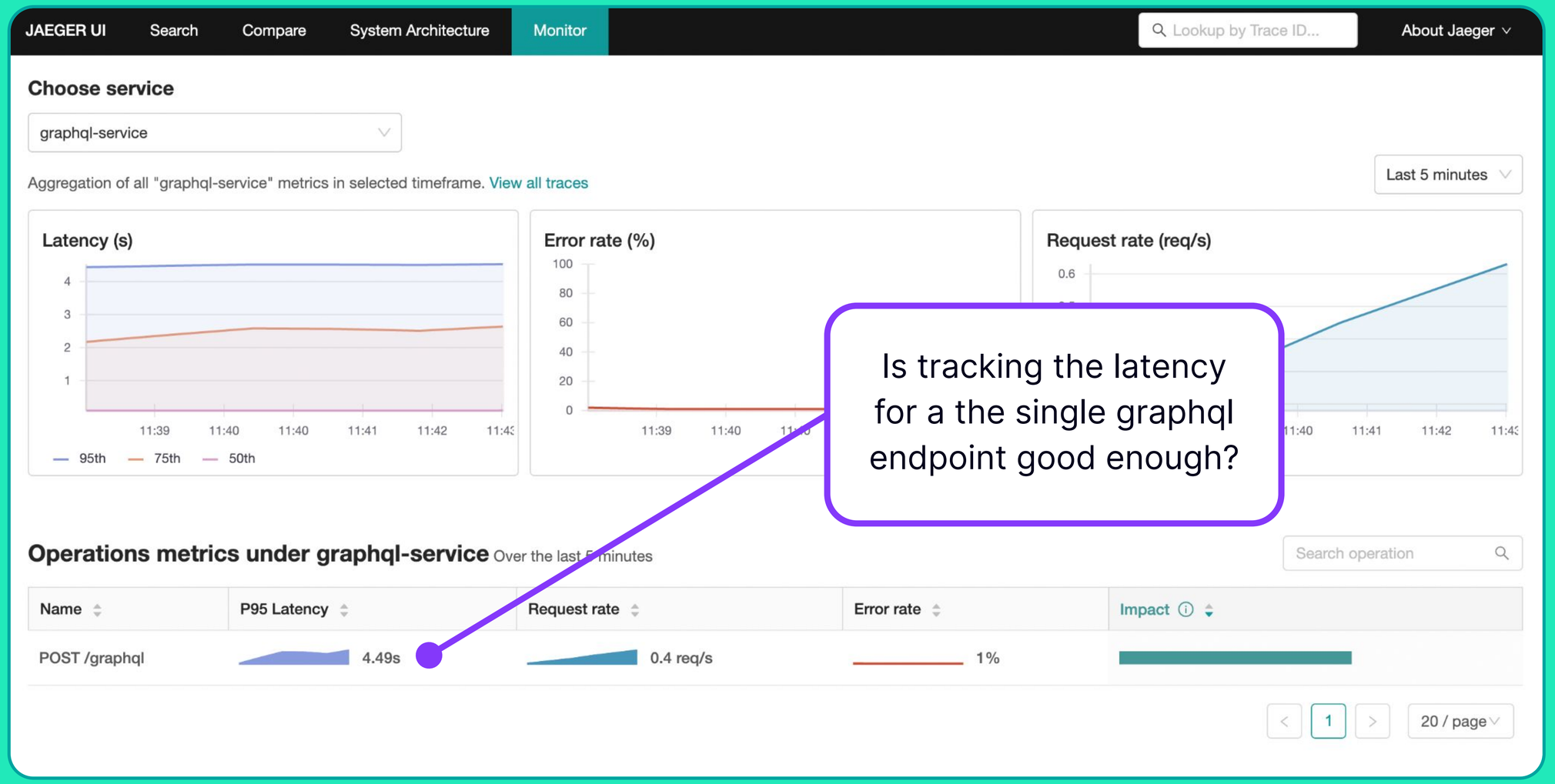
PERFORMANCE

How can we detect performance issues?



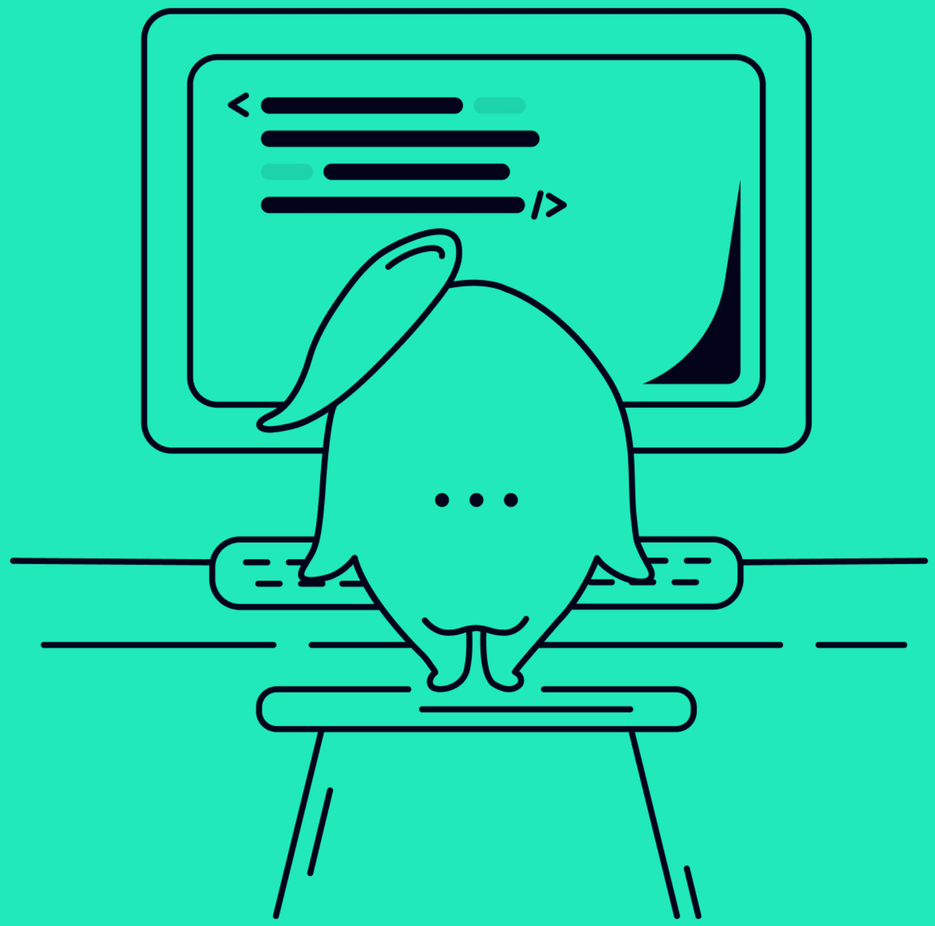
PERFORMANCE

How can we detect performance issues?



Performance issues

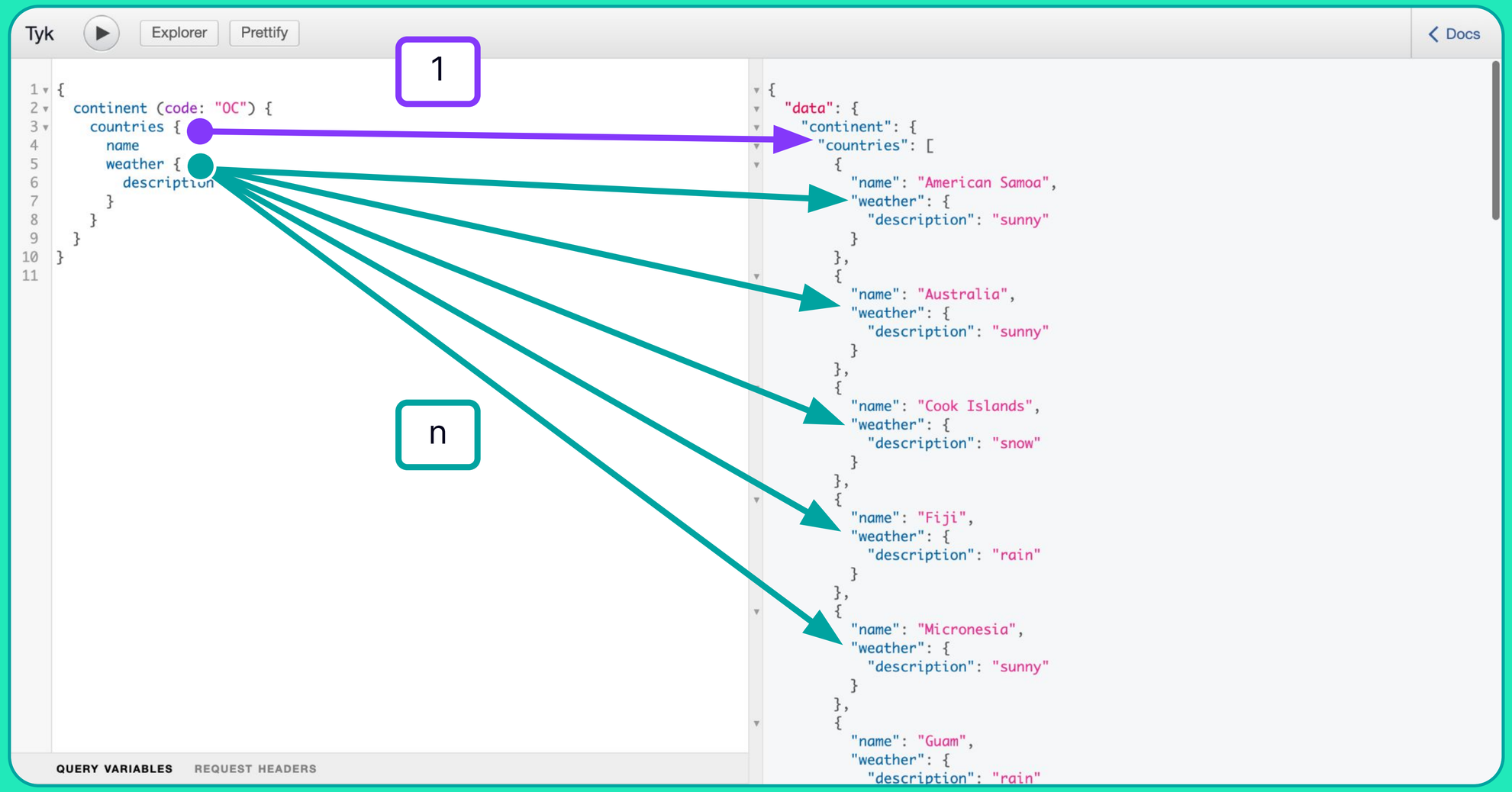
1. N+1
2. Cyclic queries
3. Query complexity
4. Query depth

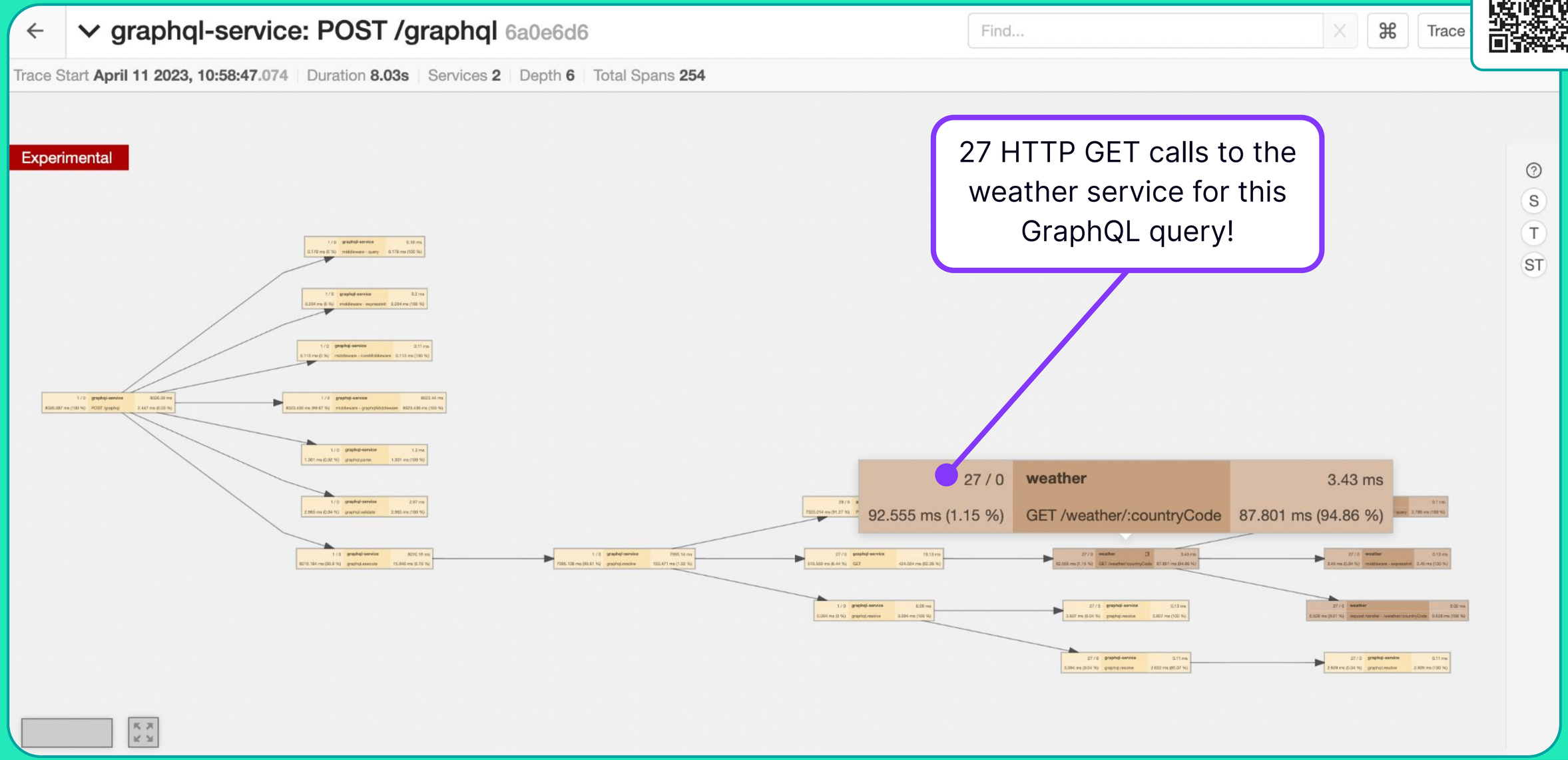


Tyk

PERFORMANCE

N+1 issue




graphql-service: POST /graphql 6a0e6d6
Find...
Trace
Trace Start April 11 2023, 10:58:47.074 | Duration 8.03s | Services 2 | Depth 6 | Total Spans 254
Experimental
27 HTTP GET calls to the weather service for this GraphQL query!
27 / 0
weather
3.43 ms
92.555 ms (1.15 %)
GET /weather/:countryCode
87.801 ms (94.86 %)

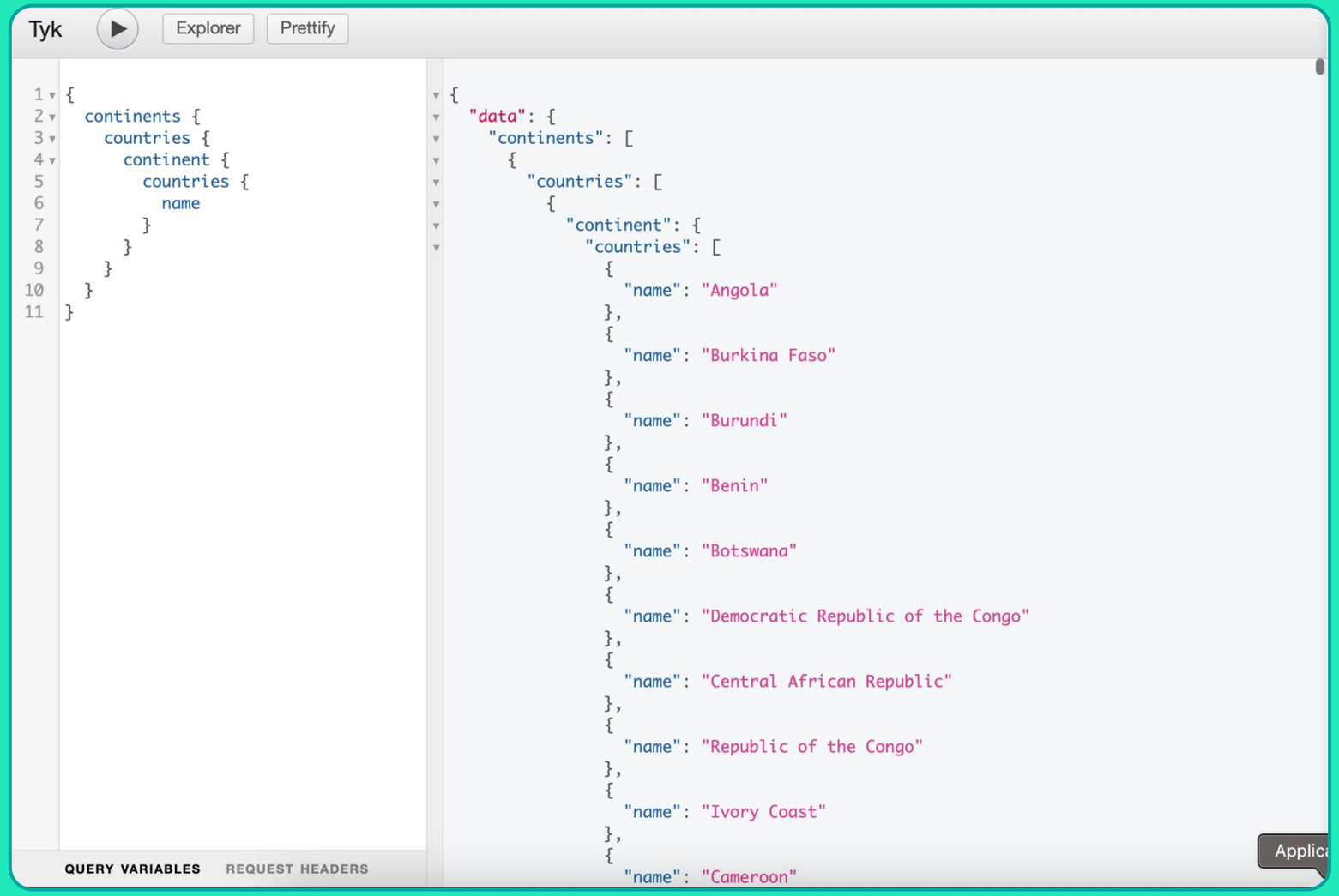
PERFORMANCE

Detecting N+1 queries

On average, 1 GraphQL query makes 24 outgoing requests

PERFORMANCE

Cyclic queries



PERFORMANCE

Expensive Queries

Depth 3

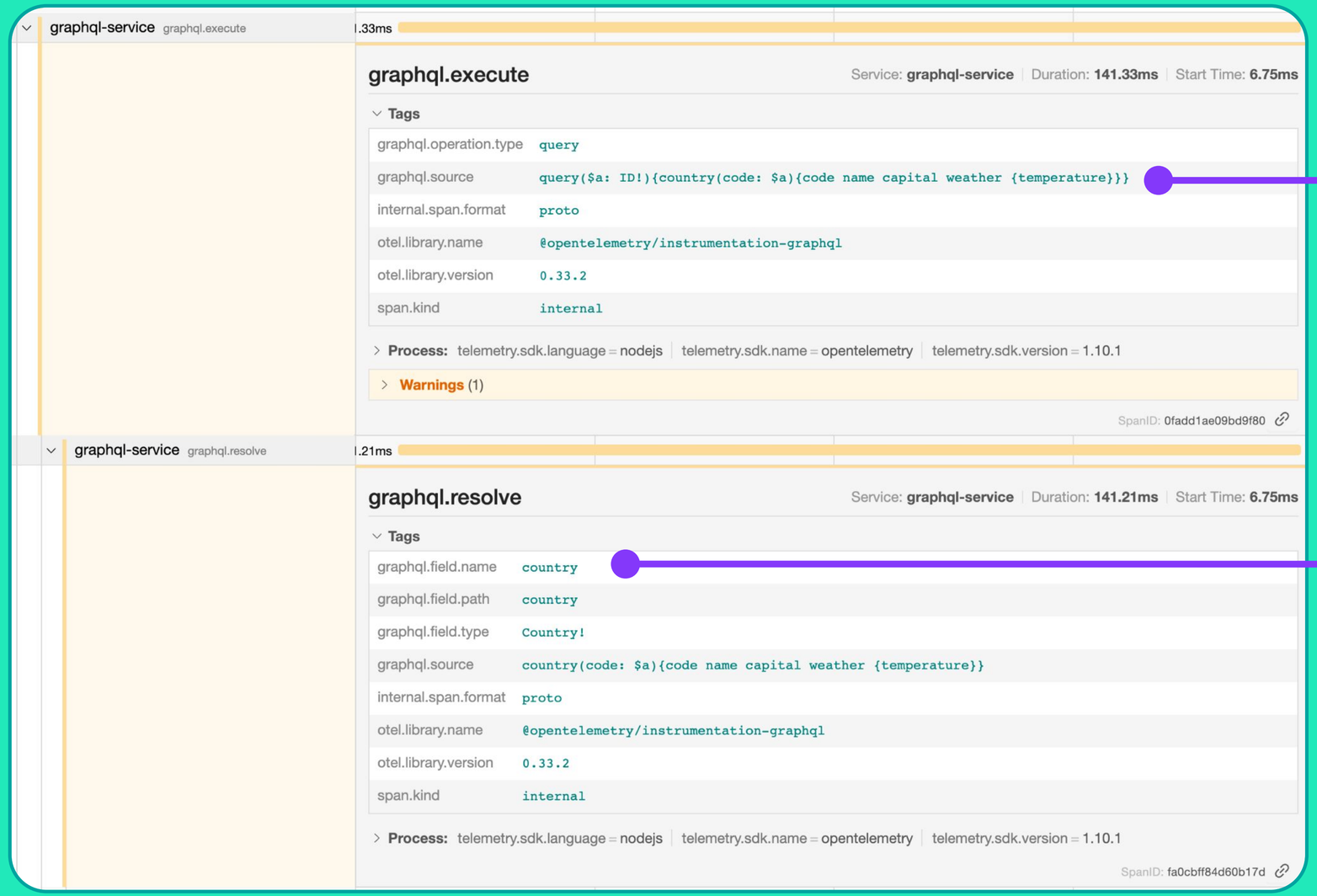
```
{
  a {
    b {
      c
    }
  }
}
```

Complexity 6

```
{
  a b c {
    d e {
      f
    }
  }
}
```

PERFORMANCE

What information do we have on our spans?



Full query (doesn't match the semantic conventions)

Field name (doesn't match the semantic conventions either)

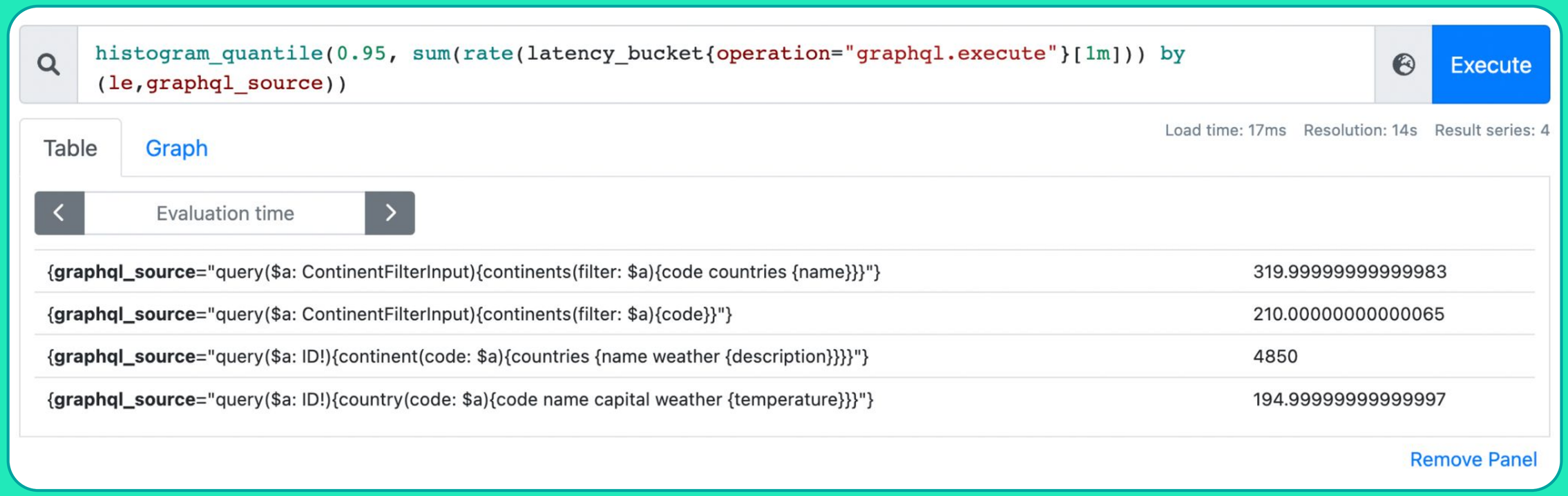
OpenTelemetry collector configuration to export additional span metrics

```
21  processors:
22    batch:
23    spanmetrics:
24      dimensions:
25        - name: graphql.source
26        - name: graphql.field.name
27        - name: graphql.field.path
28      metrics_exporter: prometheus
29
30  service:
31    pipelines:
32      traces:
33        receivers: [otlp]
34        processors: [spanmetrics, batch]
35        exporters: [otlp]
36      metrics:
37        receivers: [otlp]
38        processors: [batch]
39        exporters: [prometheus]
40
```

Tyk

PERFORMANCE

Let try to group them by query



Using operation type and name

Span Status MUST be left unset if HTTP status code was in the 1xx, 2xx or 3xx ranges, unless there was another error (e.g., network error receiving the response body; or 3xx codes with max redirects exceeded), in which case status MUST be set to `Error`.

Attribute	Type	Examples
`graphql.operation.name`	string	`findBookById`
`graphql.operation.type`	string	`query`; `mutation`; `subscription`
`graphql.document`	string	`query findBookById { bookById(id: ?) { name } }`

but not available in our instrumentation library

but not available in our instrumentation library

Tyk

Using operation type and name

query continents	300 ms
query continents	2400 ms
mutation updateCountry	500 ms

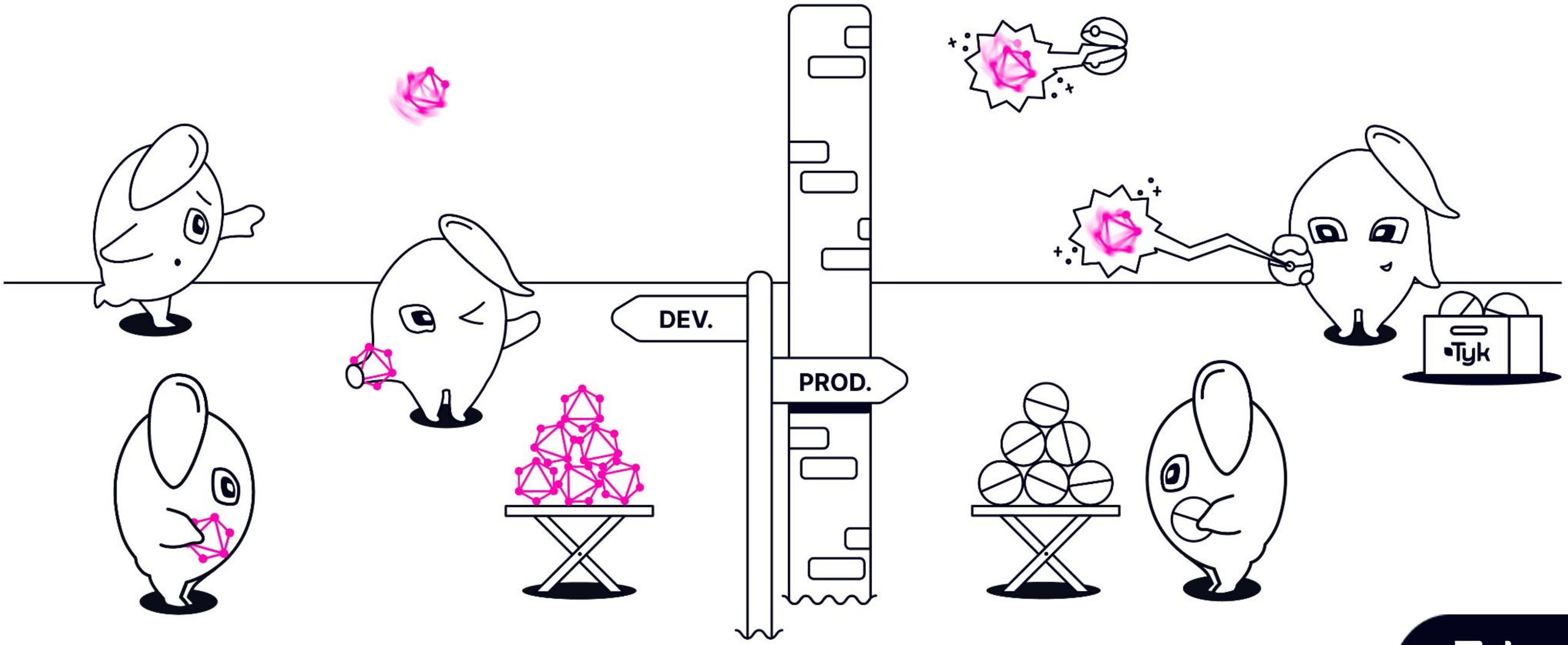
Adding a client identification

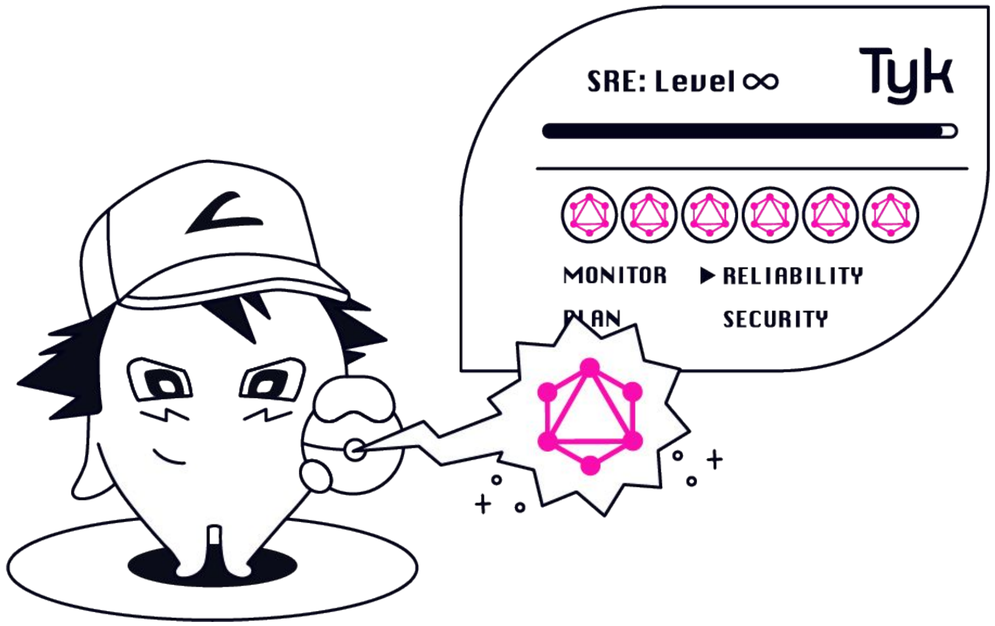
query continents mobile	600 ms
query continents react	3500 ms
query continents weather-app	550 ms

What have we learnt?

1. OpenTelemetry is helpful for monitoring and troubleshooting GraphQL queries, BUT:
2. Semantic conventions are not always respected by instrumentation libraries
3. GraphQL errors are missing from semantic conventions and from instrumentation
   - needed to detect for errors
   - and for error based sampling
4. RED metrics approach of monitoring needs to be extended to report GraphQL specifics (query type, operation name, query depth, query complexity)

Tyk


DEV.
PROD.
Tyk



Thank you!

Come talk to us to continue the discussion or reach out:

- Sonja Chevre
  - @SonjaChevre
  - linkedin.com/in/sonjachevre
- Ahmet Soormally
  - @SoormallyAhmet
  - linkedin.com/in/ahmet-soormally

Tyk