Lightstep

An introduction to OpenTelemetry

CNCF Meetup London – 7th September 2022 – v3
Dimitris Finas, Sr Advisory Solution Consultant

Lightstep

Agenda

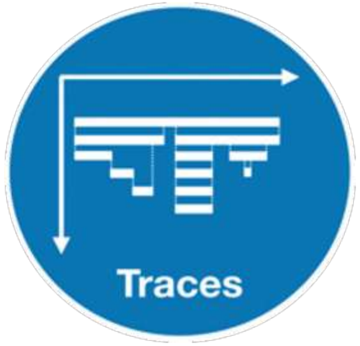
- What is OpenTelemetry?
    - A Brief History
- Distributed Tracing Overview
- What does it change?
    - Time for change
- New capabilities & workflows illustrated by demo
- Q&A

Lightstep

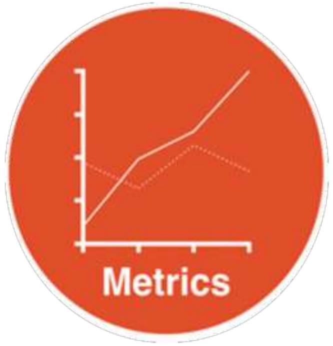
What is OpenTelemetry?

OpenTelemetry is an open source project. It is a set of APIs, SDKs, tooling and integrations that are designed for the creation and management of *telemetry data,* such as traces, metrics, and logs.

OpenTelemetry's Mission is to enable effective observability by making high-quality, portable telemetry ubiquitous and vendor-agnostic.



A trace represents a single user's journey across multiple applications and systems (usually microservices).



Numeric data measured at various time intervals (time series data); SLI's (request rate, error rate, duration, CPU%, etc.)

Timestamped records of discrete events that happened within an application or system, such as a failure, an error, or a state transformation

You want more? See Otel vision in https://github.com/open-telemetry/community/blob/main/mission-vision-values.md#otel-mission-vision-and-values

Lightstep

OpenTelemetry - A brief history

2015

Lightstep is founded by Ben Sigelman, Daniel Spoonhower, and Ben Cronin

2016

Lightstep helps to found **OpenTracing**, the first common, portable API for distributed tracing

2017

Jaeger, an open-source tracing tool developed by Yuri Shkuro, is open-sourced by **Uber**

2018

Google releases a new portable instrumentation project called **OpenCensus**

2019

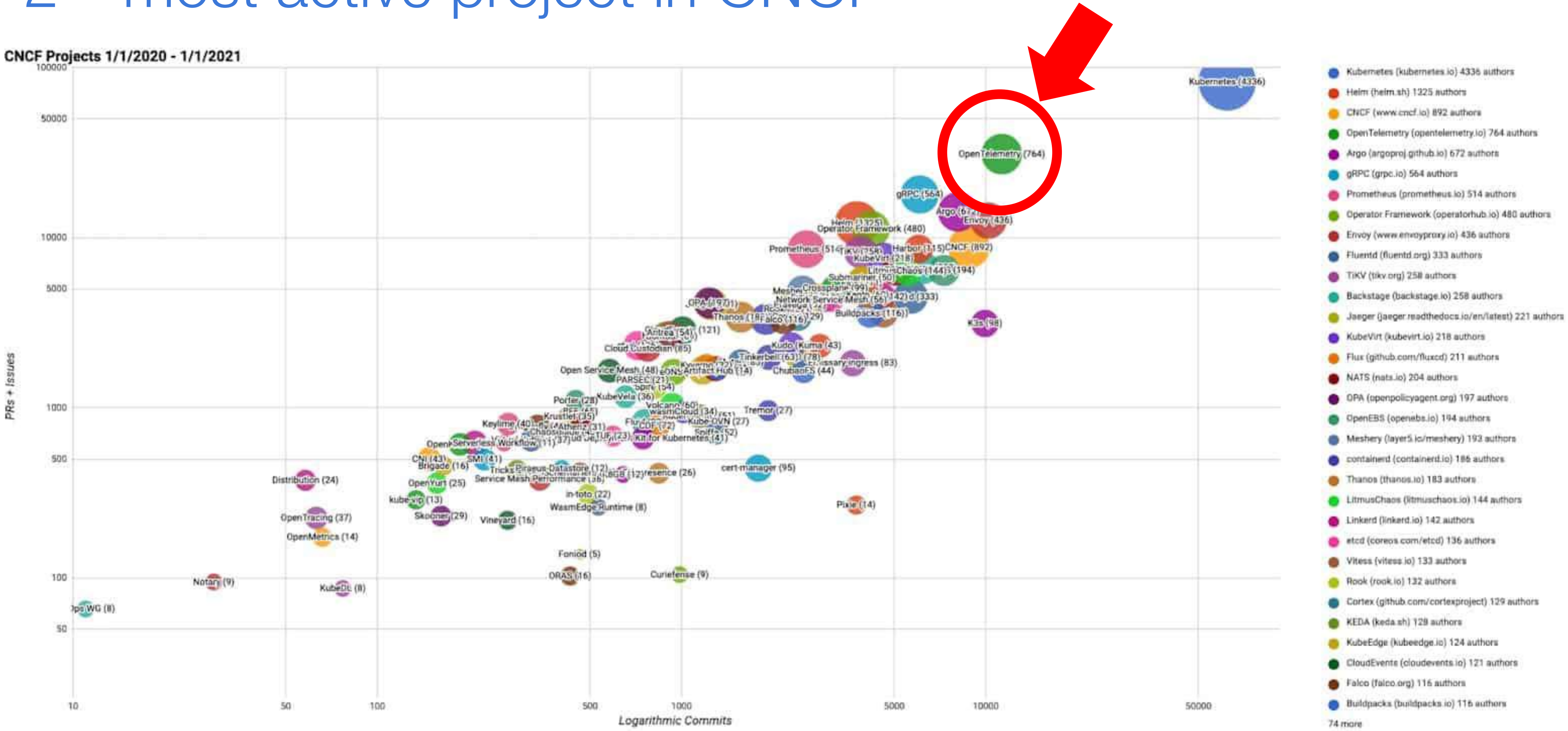
The **OpenTracing** and **OpenCensus** projects announce a merger, forming **OpenTelemetry** to be a part of CNCF

2020

OpenTelemetry is released in beta

Lightstep

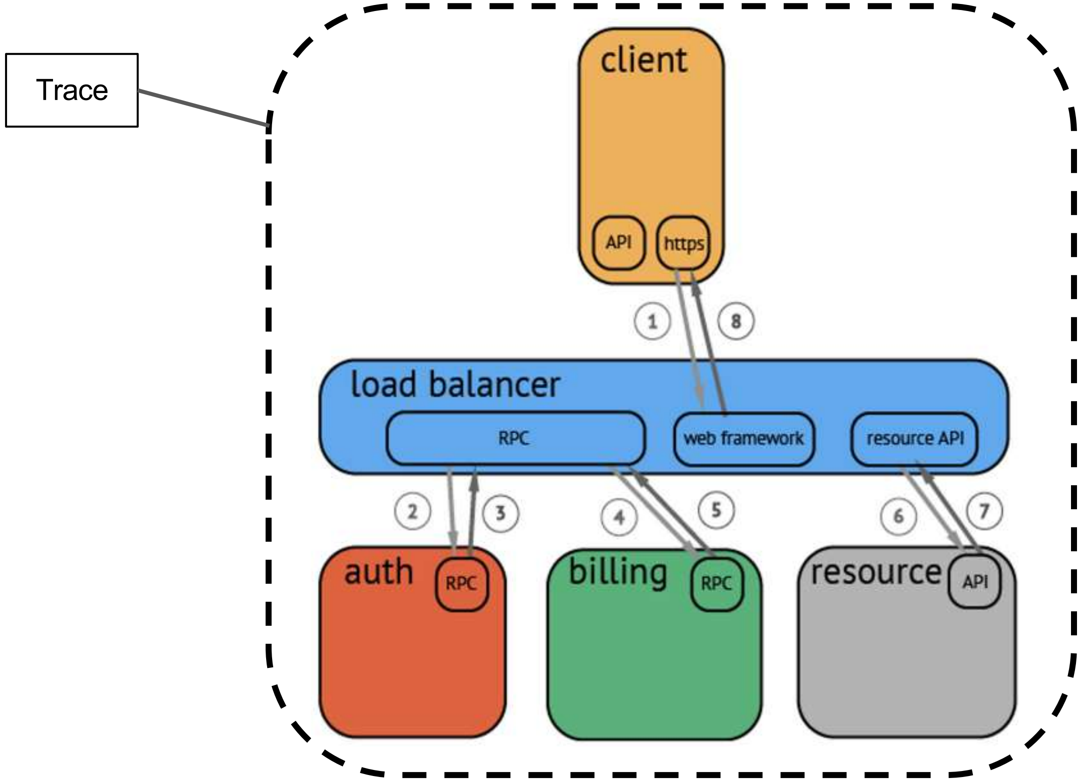
2nd most active project in CNCF



source: https://www.cncf.io/blog/2021/08/02/update-on-cncf-and-open-source-project-velocity-2020

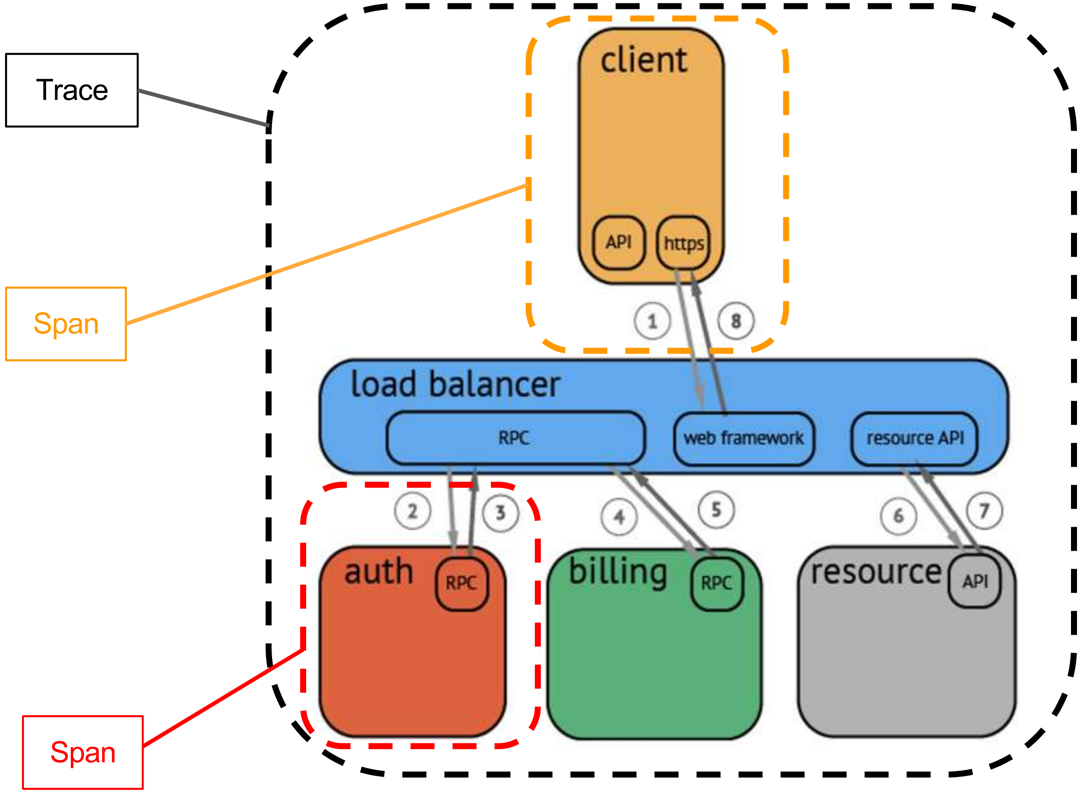
What are **distributed traces**?

Introduction to distributed traces



A “**trace**” is a view into the request lifecycle as a whole as it moves through a distributed system.

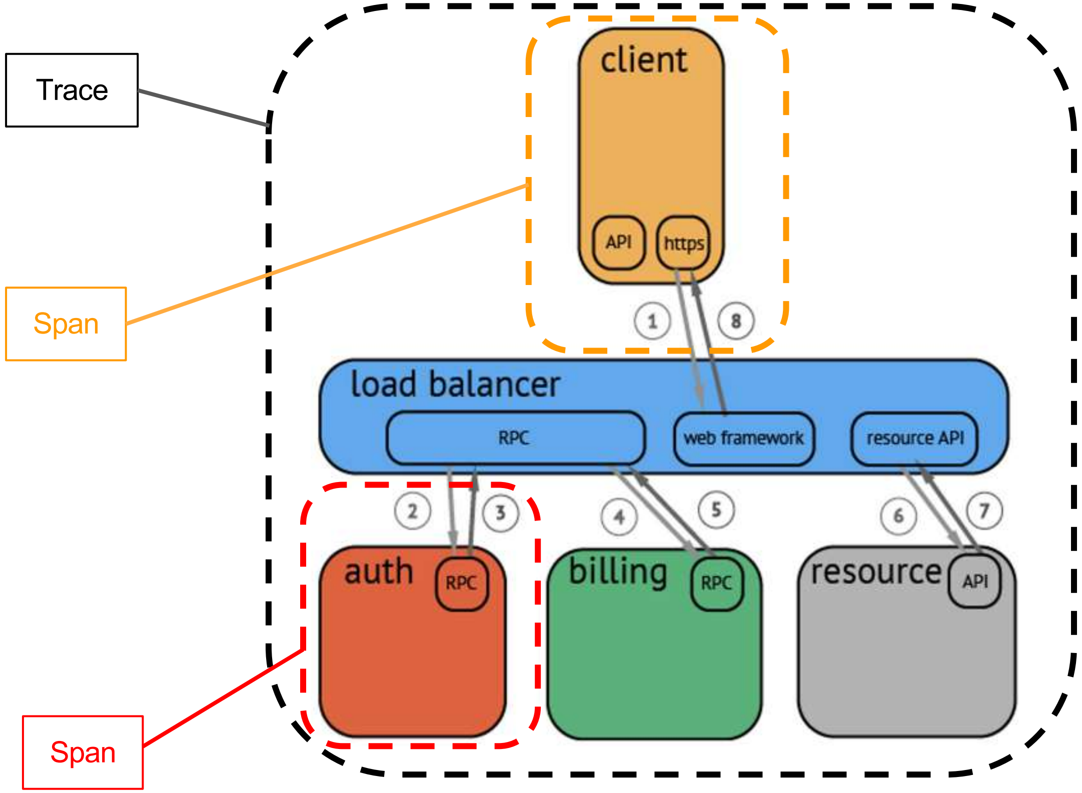
Introduction to distributed traces



A **trace** is a collection of spans.

Each component of the distributed system contributes a “**span**” - a named, timed operation representing a piece of the workflow.

Introduction to distributed traces



Trace

Span

Span

client

API

https

1

8

load balancer

RPC

web framework

resource API

2

3

4

5

6

7

auth

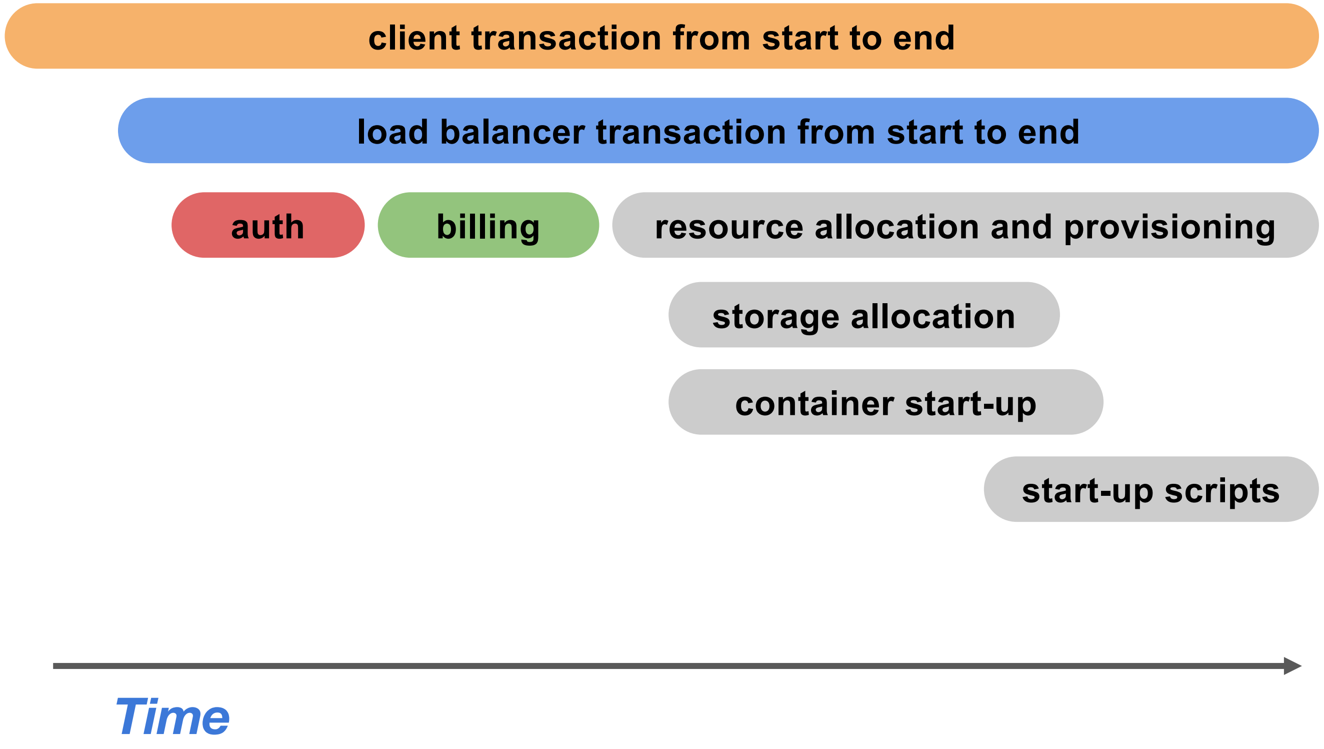
RPC

billing

RPC

resource

API



client transaction from start to end

load balancer transaction from start to end

auth

billing

resource allocation and provisioning

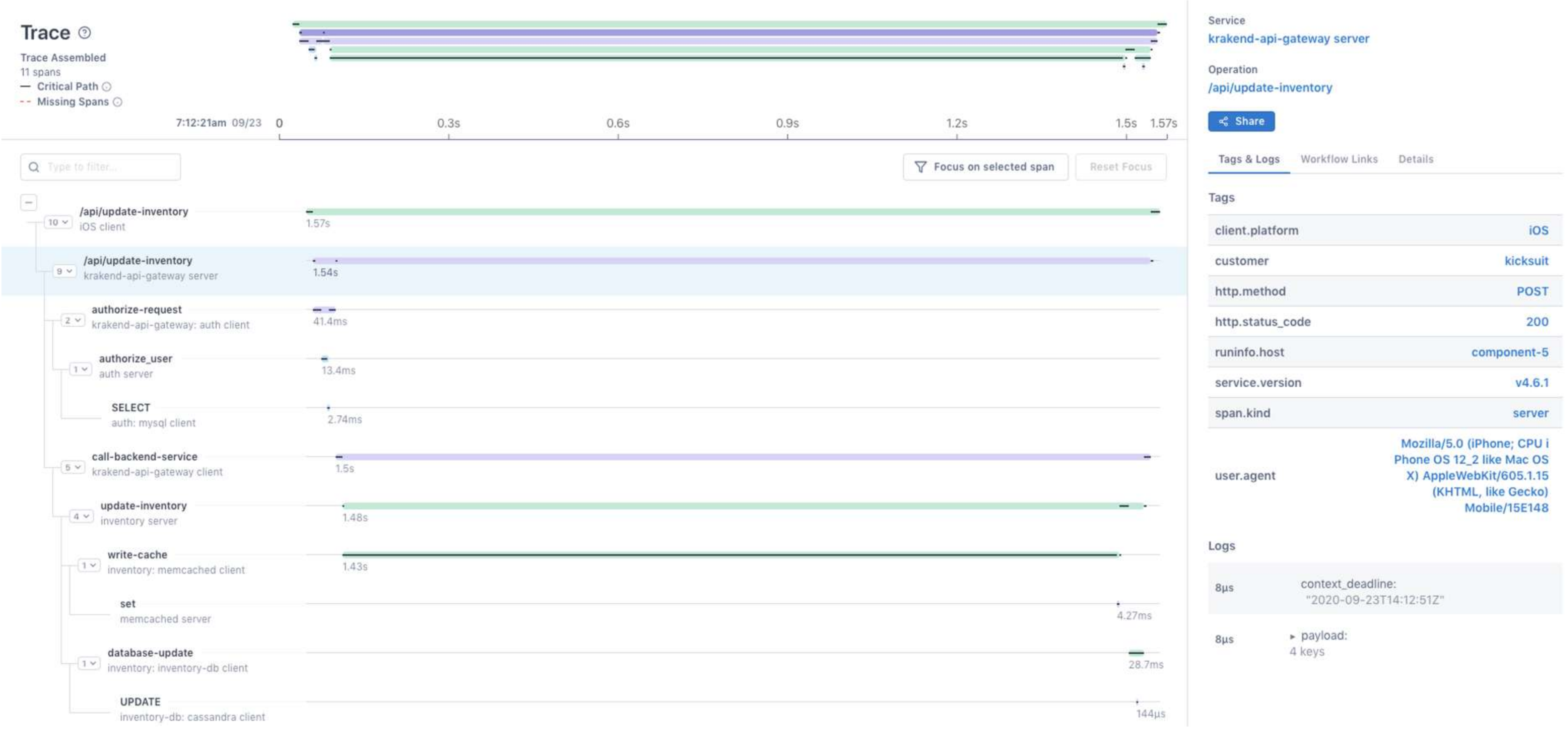
storage allocation

container start-up

start-up scripts

Time

Visualizing a trace



Trace

Trace Assembled
11 spans
— Critical Path
-- Missing Spans

7:12:21am 09/23 0 0.3s 0.6s 0.9s 1.2s 1.5s 1.57s

Type to filter...

Focus on selected span | Reset Focus

Operation	Service	Children	Duration
/api/update-inventory	iOS client	10	1.57s
/api/update-inventory	krakend-api-gateway server	9	1.54s
authorize-request	krakend-api-gateway: auth client	2	41.4ms
authorize_user	auth server	1	13.4ms
SELECT	auth: mysql client		2.74ms
call-backend-service	krakend-api-gateway client	5	1.5s
update-inventory	inventory server	4	1.48s
write-cache	inventory: memcached client	1	1.43s
set	memcached server		4.27ms
database-update	inventory: inventory-db client	1	28.7ms
UPDATE	inventory-db: cassandra client		144µs

Service
krakend-api-gateway server

Operation
/api/update-inventory

Share

Tags & Logs | Workflow Links | Details

Tags

client.platform	iOS
customer	kicksuit
http.method	POST
http.status_code	200
runinfo.host	component-5
service.version	v4.6.1
span.kind	server
user.agent	Mozilla/5.0 (iPhone; CPU iPhone OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148

Logs

8µs	context_deadline: "2020-09-23T14:12:51Z"
8µs	▸ payload: 4 keys

Lightstep

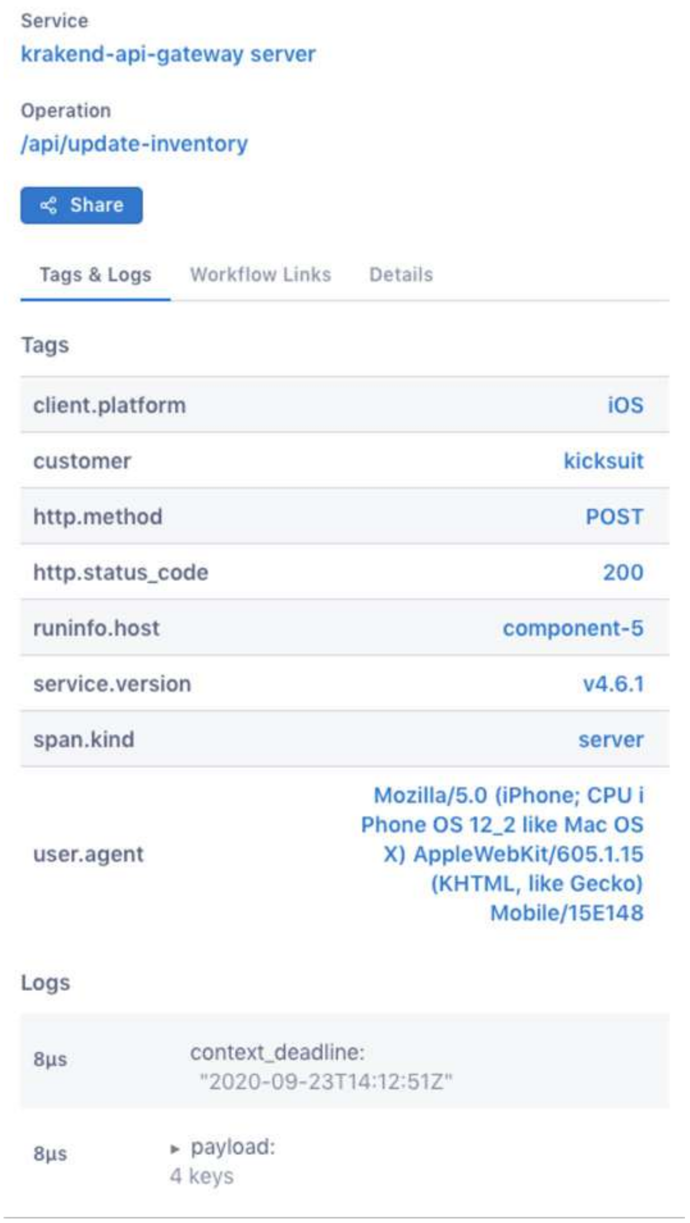
Visualizing a trace

Each span also has **span context**

Span context is composed of ***attributes (tags)*** and ***events (logs)***. These are added during instrumentation.

Attributes allow you to search and segment your data within LightStep

Events (logs) add information that is useful during root cause and debugging



Service
krakend-api-gateway server

Operation
/api/update-inventory

Tags

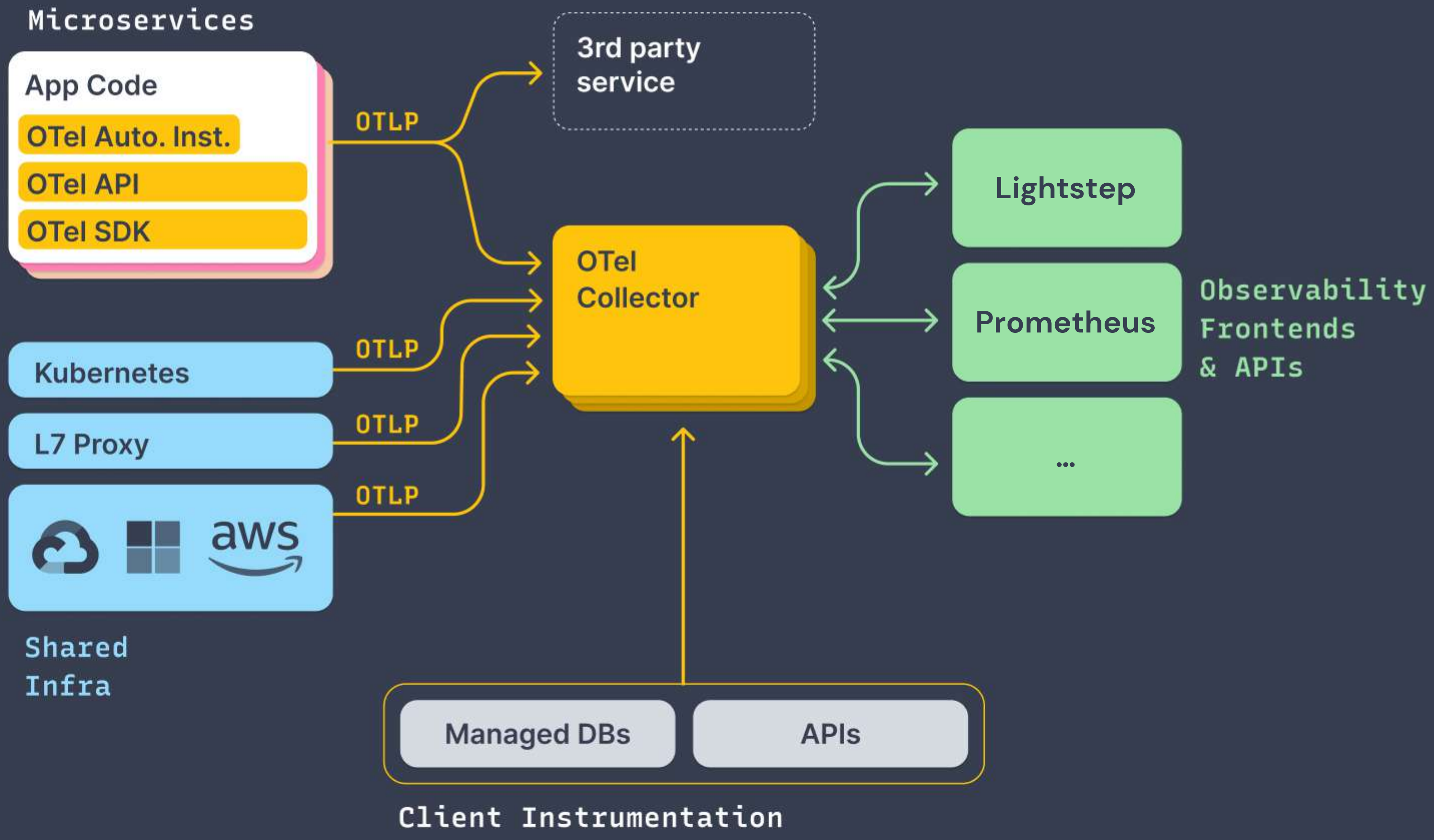
client.platform	iOS
customer	kicksuit
http.method	POST
http.status_code	200
runinfo.host	component-5
service.version	v4.6.1
span.kind	server
user.agent	Mozilla/5.0 (iPhone; CPU iPhone OS 12_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148

Logs

8μs	context_deadline: "2020-09-23T14:12:51Z"
8μs	▸ payload: 4 keys

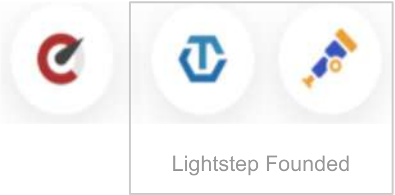
Time for
Change

A performant solution, agnostic from any vendor



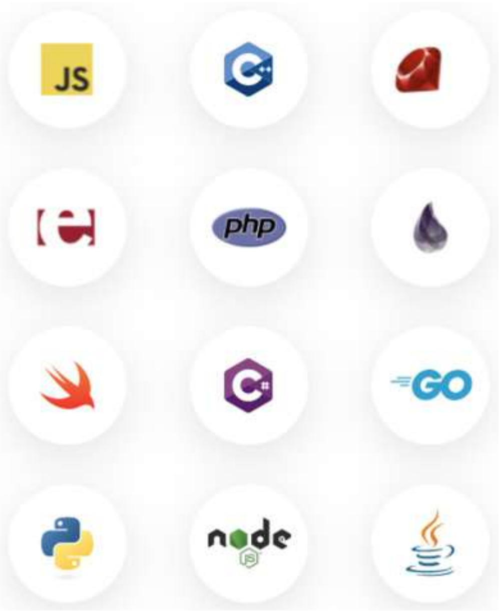
Integrations

Standards

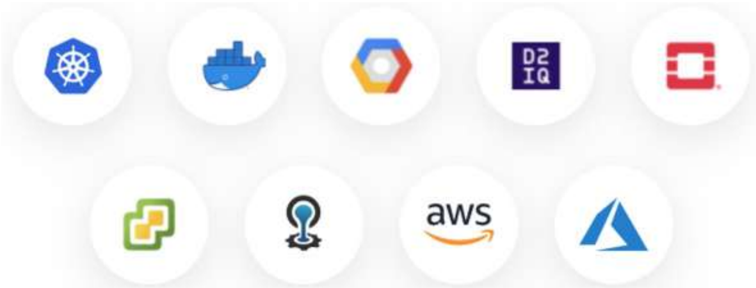


Tracers

Languages



Containers, Platforms and Clouds



Data Streaming and Storage

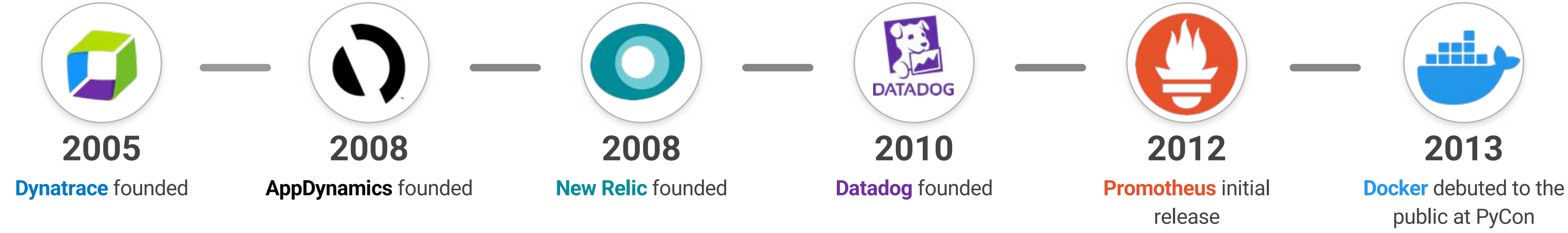
Service Meshes / Proxies

Deployment Automation (CI/CD)

Alerting and Tools

See up to date list in https://opentelemetry.io/registry/

History vs Legacy vendors



2005
Dynatrace founded

2008
AppDynamics founded

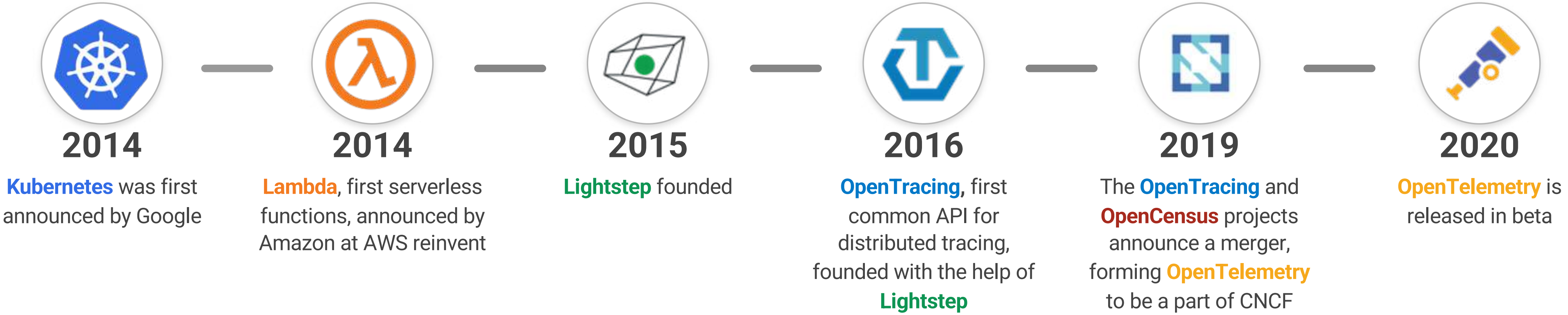
2008
New Relic founded

2010
Datadog founded

2012
Promotheus initial release

2013
Docker debuted to the public at PyCon

All legacy vendors exists even before Docker was invented!



2014
Kubernetes was first announced by Google

2014
Lambda, first serverless functions, announced by Amazon at AWS reinvent

2015
Lightstep founded

2016
OpenTracing, first common API for distributed tracing, founded with the help of **Lightstep**

2019
The **OpenTracing** and **OpenCensus** projects announce a merger, forming **OpenTelemetry** to be a part of CNCF

2020
OpenTelemetry is released in beta

Cultural Shift 1:

It’s less about

USE

(**U**sage + **S**aturation + **E**rror)

It’s more about

RED

(**R**ate + **E**rror + **D**elay)

Cultural Shift 2:

It’s less about

LOGS

It’s more about

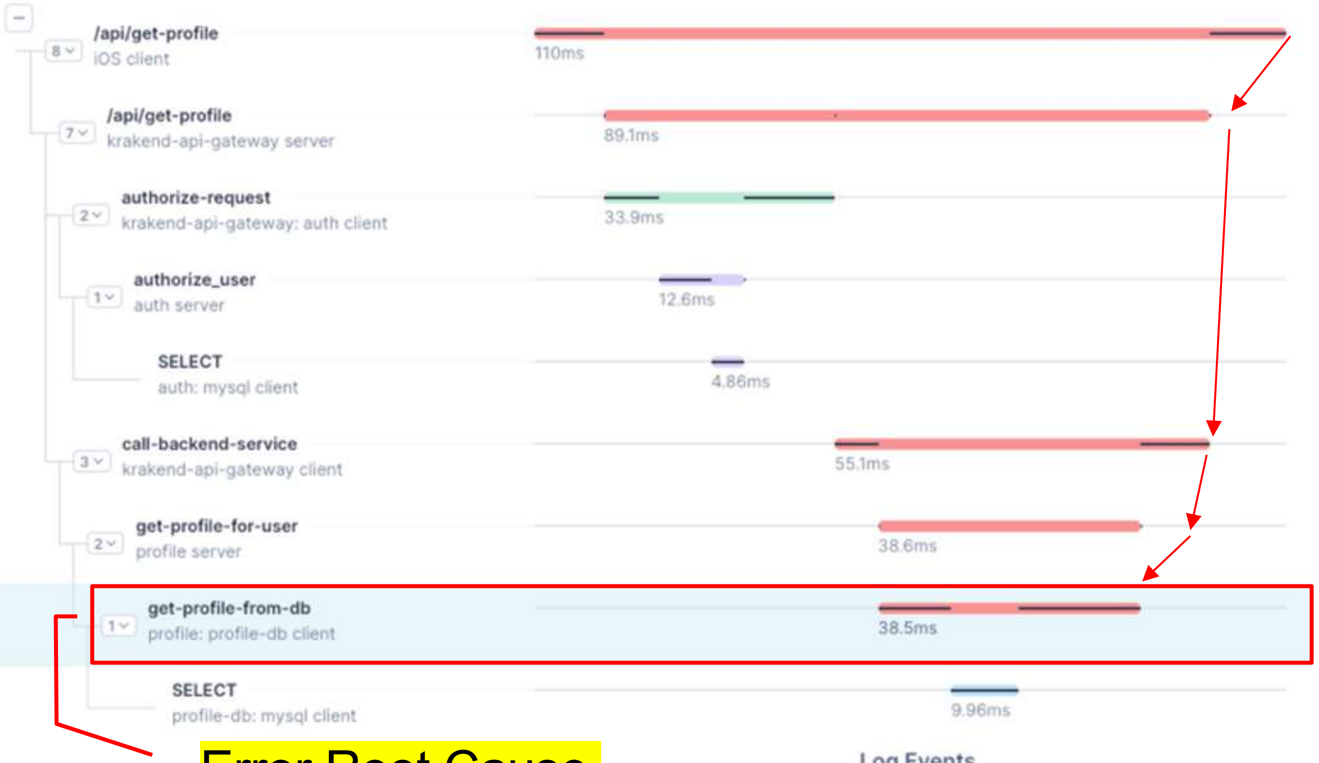
DISTRIBUTED TRACES

New capabilities & workflows offered by Lightstep on top of OpenTelemetry data

New Capabilities => New Workflows

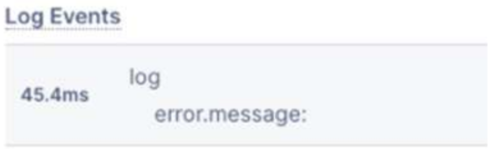
Distributed Traces

Distributed traces help you understand details of your call stack for each transaction



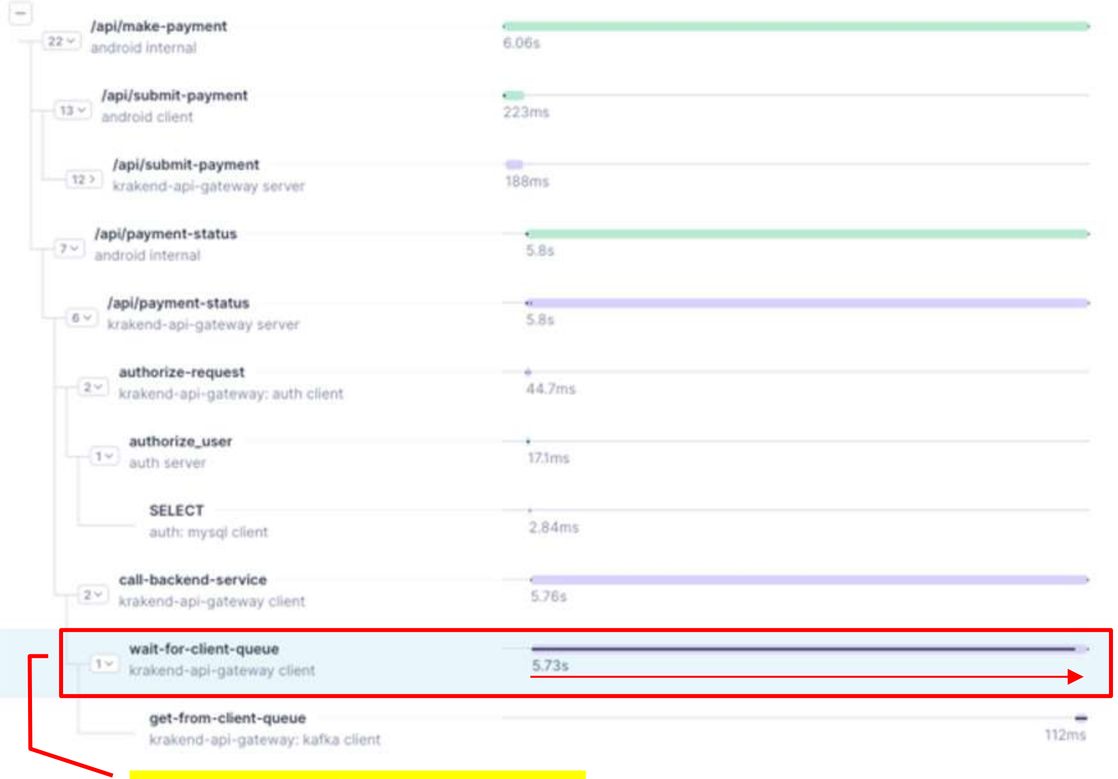
```
/api/get-profile                     110ms
iOS client
  /api/get-profile                   89.1ms
  krakend-api-gateway server
    authorize-request                33.9ms
    krakend-api-gateway: auth client
      authorize_user                 12.6ms
      auth server
        SELECT                       4.86ms
        auth: mysql client
    call-backend-service             55.1ms
    krakend-api-gateway client
      get-profile-for-user           38.6ms
      profile server
        get-profile-from-db          38.5ms
        profile: profile-db client
          SELECT                     9.96ms
          profile-db: mysql client
```

Error Root Cause,
with attached logs:



Log Events

```
45.4ms   log
           error.message:
```



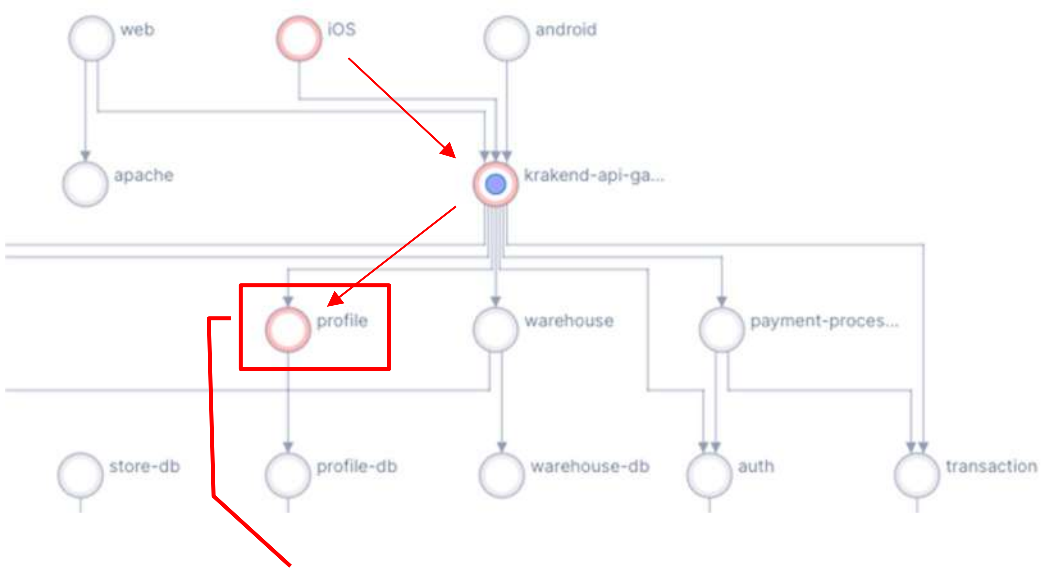
```
/api/make-payment                    6.06s
android internal
  /api/submit-payment                223ms
  android client
    /api/submit-payment              188ms
    krakend-api-gateway server
  /api/payment-status                5.8s
  android internal
    /api/payment-status              5.8s
    krakend-api-gateway server
      authorize-request              44.7ms
      krakend-api-gateway: auth client
        authorize_user               17.1ms
        auth server
          SELECT                     2.84ms
          auth: mysql client
      call-backend-service           5.76s
      krakend-api-gateway client
        wait-for-client-queue        5.73s
        krakend-api-gateway client
          get-from-client-queue      112ms
          krakend-api-gateway: kafka client
```

Latency Critical Path

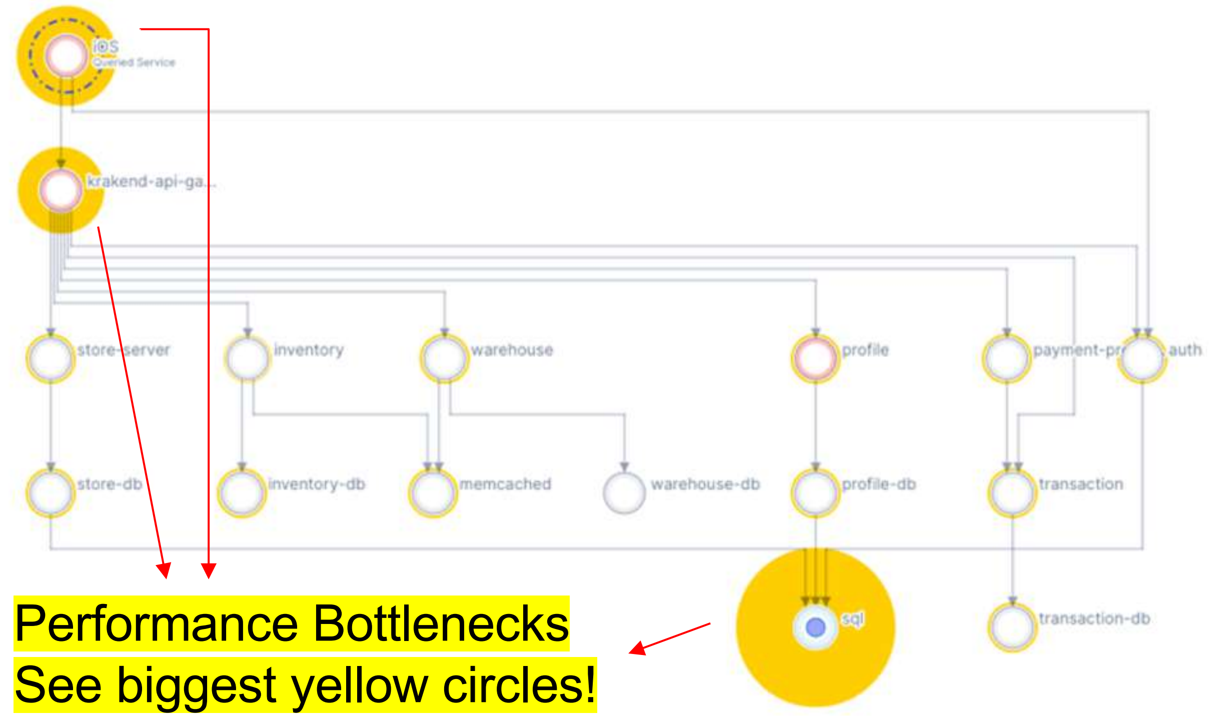
New Capabilities => New Workflows

Graphical visibility of dependencies

Captured traces allow to draw dependencies between your services with understanding of errors and latency information in real time!



Error Root Cause
Follow the red circles!



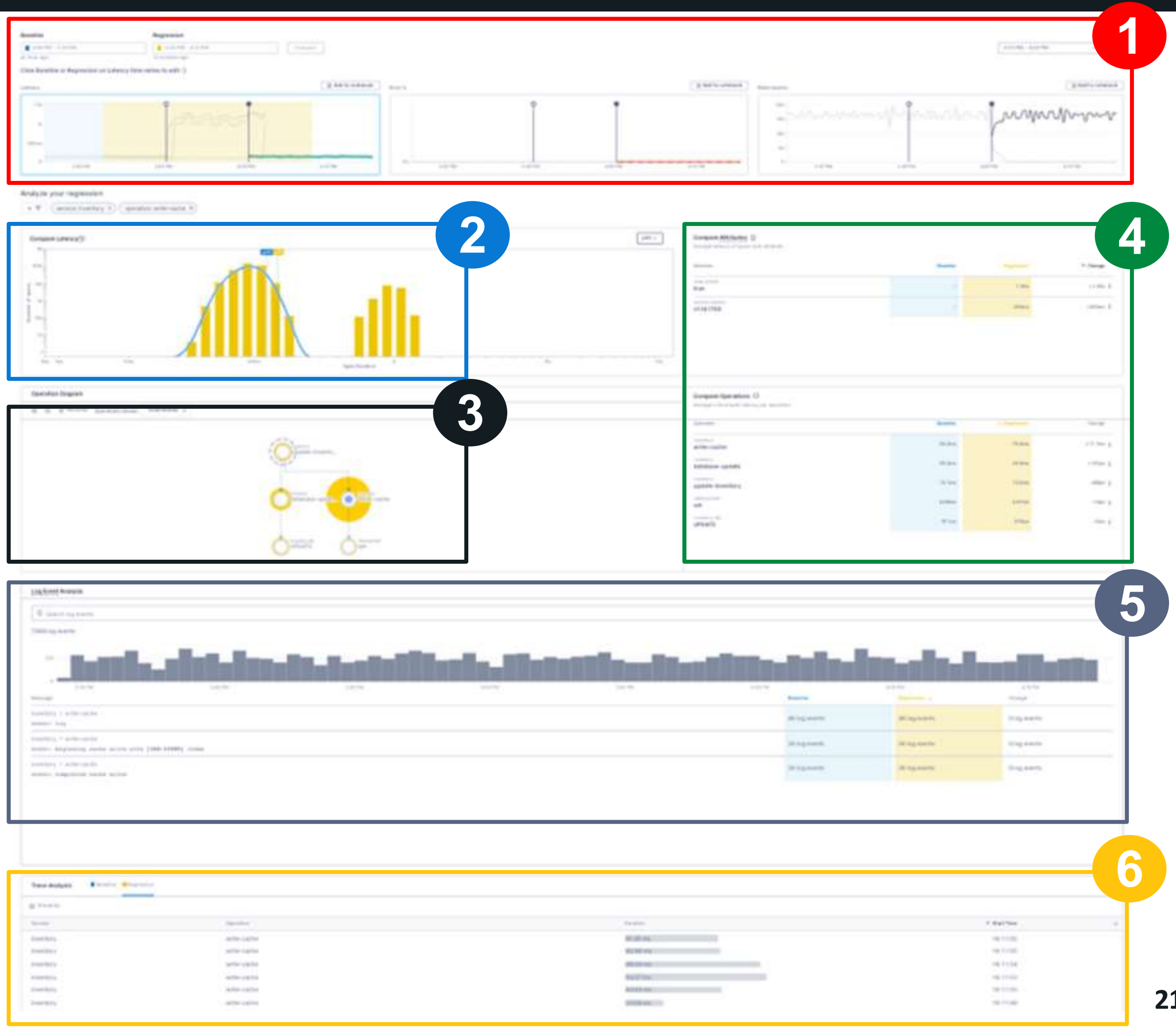
Performance Bottlenecks
See biggest yellow circles!

New Capabilities => New Workflows

Unified telemetry (traces, metrics, logs)

Get a view of all your telemetry data in a single page

1. Metrics & KPIs help detect anormal behavior
2. Latency histogram give your performance distribution
3. Architecture diagram give your dependencies & bottlenecks
4. Otel attributes give you the context of the anomaly
5. Log events explain the root cause
6. Traces explain the root cause



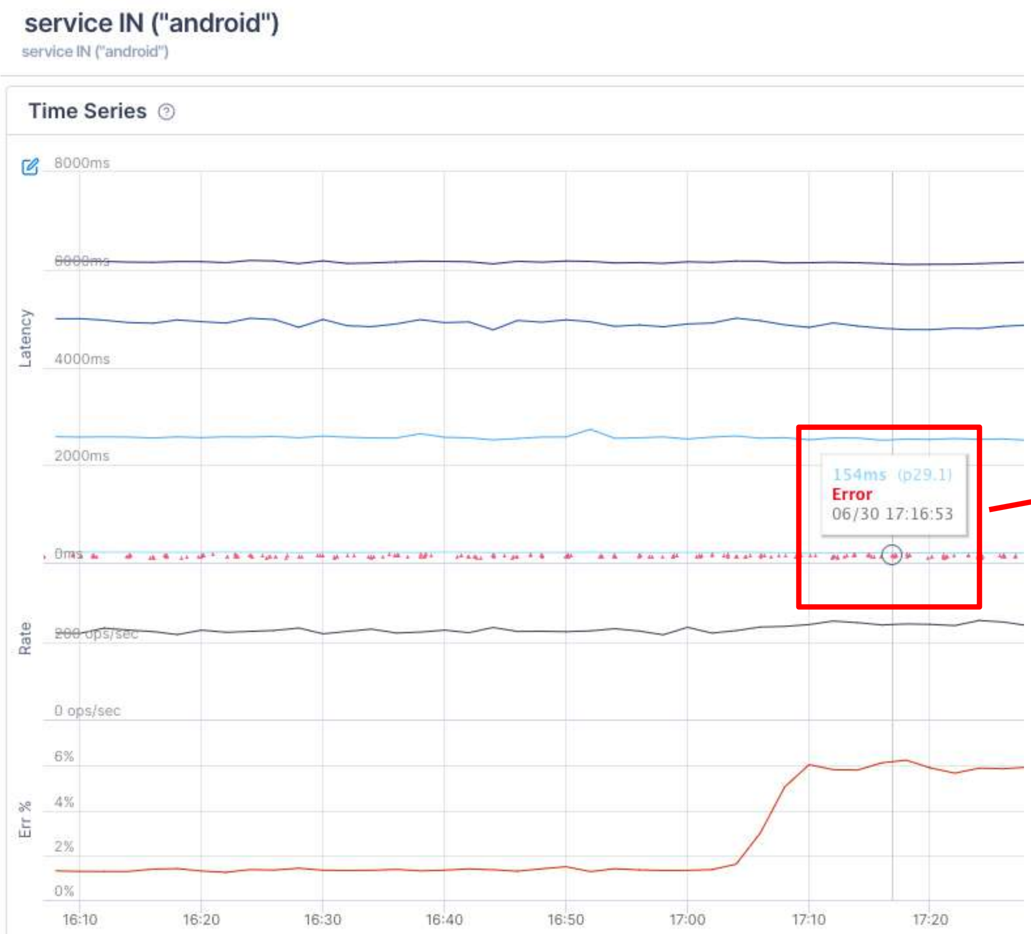
New Capabilities => New Workflows

Follow the RED ~~butterfly~~ method

R

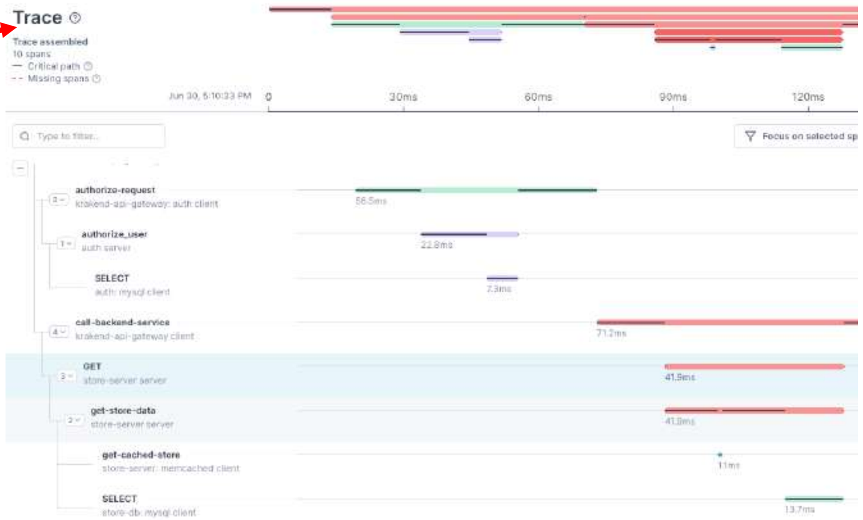
E

D



New reports are based on **RED** (Rate, Error, Delay) best practise to follow metrics linked to customer perception

Each point give access to a trace to pinpoint specific issues



authorize-request
krakend-api-gateway: auth client
56.5ms

authorize_user
auth server
22.8ms

SELECT
auth: mysql client
7.3ms

call-backend-service
krakend-api-gateway client
71.2ms

GET
store-server server
41.9ms

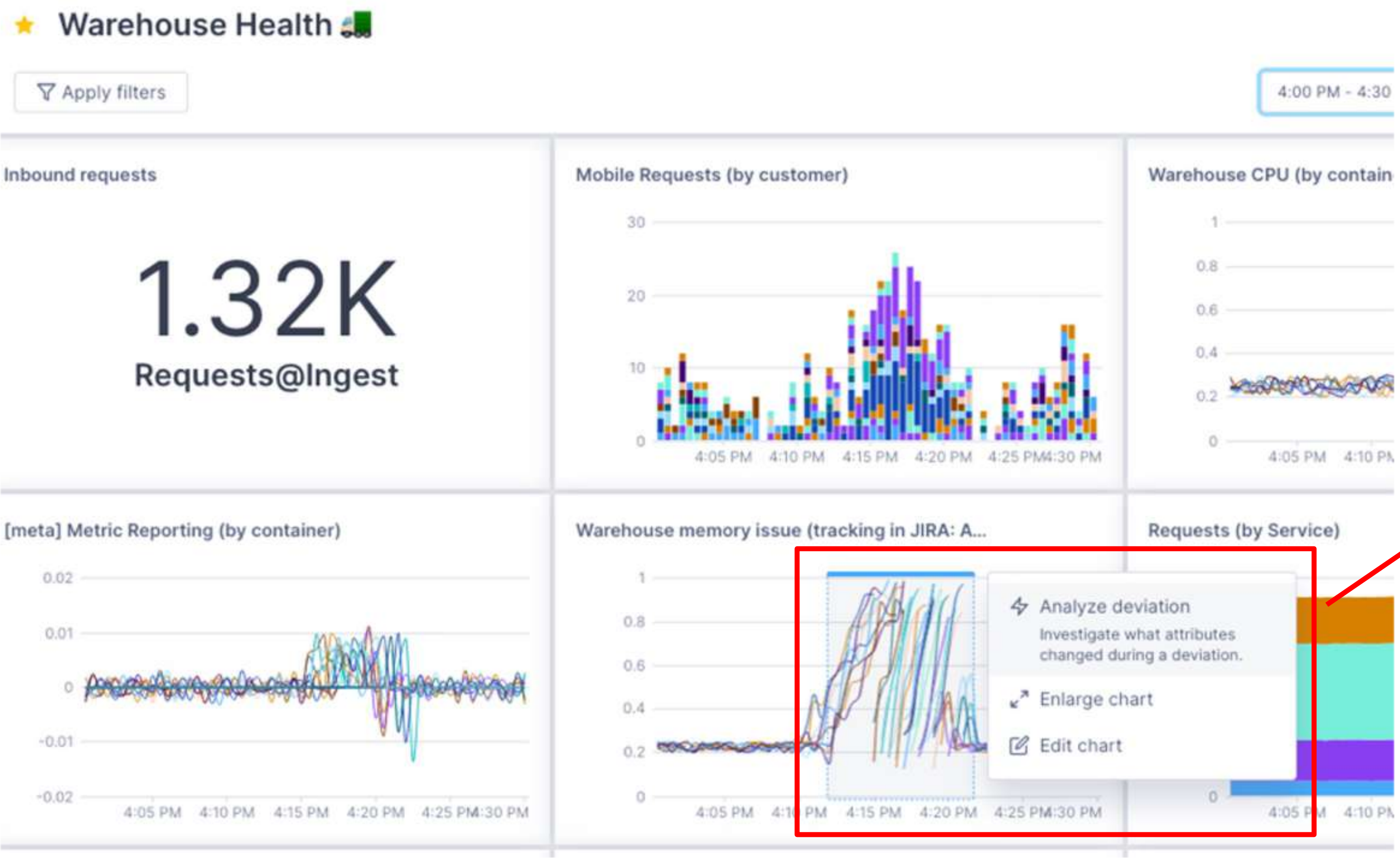
get-store-data
store-server server
41.9ms

get-cached-store
store-server: memcached client
1.1ms

SELECT
store-db: mysql client
13.7ms

New Capabilities => New Workflows

Analytics & Correlations



Correlate metrics & traces
to find
the impact & root cause
of an anomaly

Next Steps

01 – Look at OpenTelemetry

… to collect all traces, metrics and logs in vendor agnostic way.

02 – Use Distributed Traces for RCA

… as it is the best way to do root cause analysis (RCA) for micro-services and cloud natives applications.

03 – Get New Observability Frontend

… simple & intuitive to be used by Dev & Ops, but also unified to show traces, metrics and logs in single view & correlate them.

Learn OpenTelemetry

Zalando Testimony

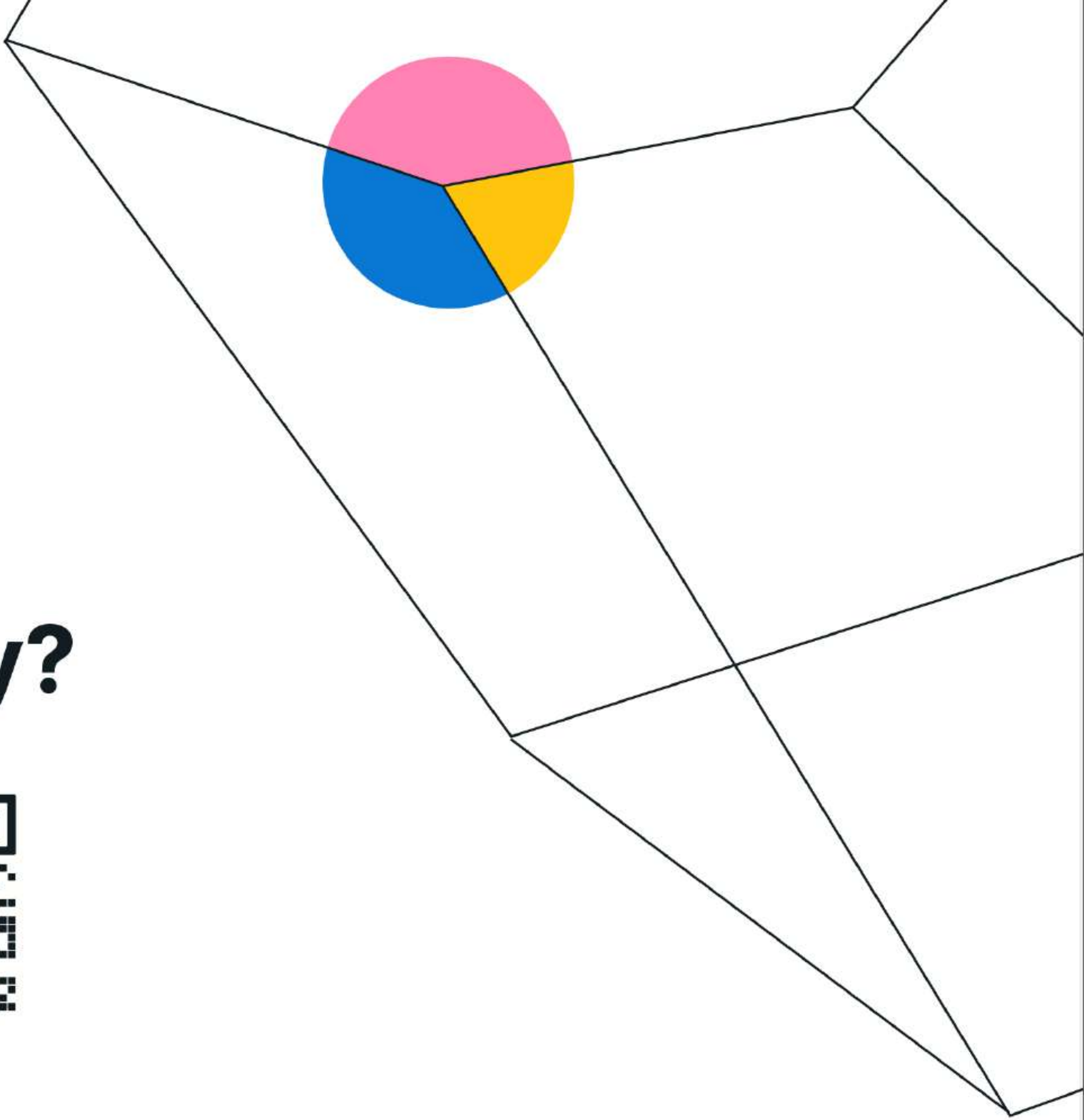
Lightstep

from ServiceNow

Interested in

Cloud Native Observability with OpenTelemetry?

Register today to learn more →



@LightstepHQ @lightstep @lightstephq @lightstephq

APPENDIX

Lightstep

Further readings

- OpenTelemetry documentation https://opentelemetry.io/docs/ and https://opentelemetry.lightstep.com/
- OpenTelemetry registry to get up-to-date list of supported technologies and projects: https://opentelemetry.io/registry/
- How to choose your Observability solution: https://medium.com/dzerolabs/unpacking-observability-how-to-choose-an-observability-vendor-aa0e6d80b71d

OpenTelemetry support of dev languages

LANGUAGE	TRACE STATUS*	INSTRUMENTATION MANUAL/AUTO**
C++	stable	manual
C# / .NET	stable	manual & auto
Erlang / Elixir	stable	manual
Go	stable	manual
Java	stable	manual & auto
Javascript / Node	stable	manual & auto
Javascript / Browser	stable	manual & auto

LANGUAGE	TRACE STATUS*	INSTRUMENTATION MANUAL/AUTO**
PHP	pre-alpha	manual
Python	stable	manual & auto
Ruby	stable	manual & auto
Rust	beta	manual
Swift	beta	manual

(*) Trace implementation status as of end of April 2022
(**) Automatic instrumentation means quick wins as no need to update existing code

Supported languages versions:

- .NET & .NET Framework all supported versions except .NET Fwk v3.5 as https://github.com/open-telemetry/opentelemetry-dotnet
- Java >= v1.8
- NodeJS >=v10 as https://github.com/open-telemetry/opentelemetry-js
- Python, only latest versions as of https://github.com/open-telemetry/opentelemetry-python

You want to know more?
See https://opentelemetry.io/docs/instrumentation/

Lightstep