Intro to Instrumentation

Kubernetes Community Day, SCaLEx20

What was your *first* monitoring tool?

Set Up

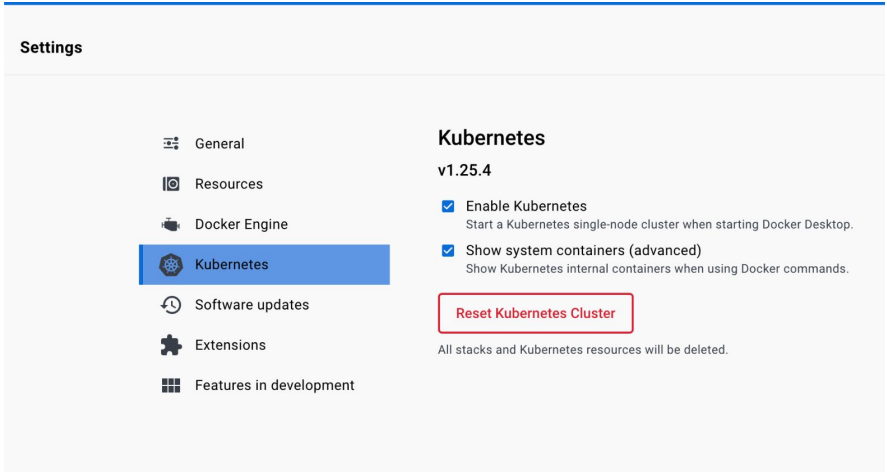
https://github.com/paigerduty/intro-to-instrumentation

Docker Desktop

- MAC Intel Chip
- Mac Apple Chip
- Windows
- Linux

Docker Desktop: Enable Kubernetes

https://docs.docker.com/desktop/kubernetes/#enable-kubernetes



Kubectl

- Linux
- Windows
- Mac

Helm

- MacOS: brew install helm
- Windows: choco install kubernetes-helm
- Linux: https://helm.sh/docs/intro/install

Observability Primer

Me + Tracing

new relic

2015-2018

SWE @ NR working on their 1st tracing product

2019-2021

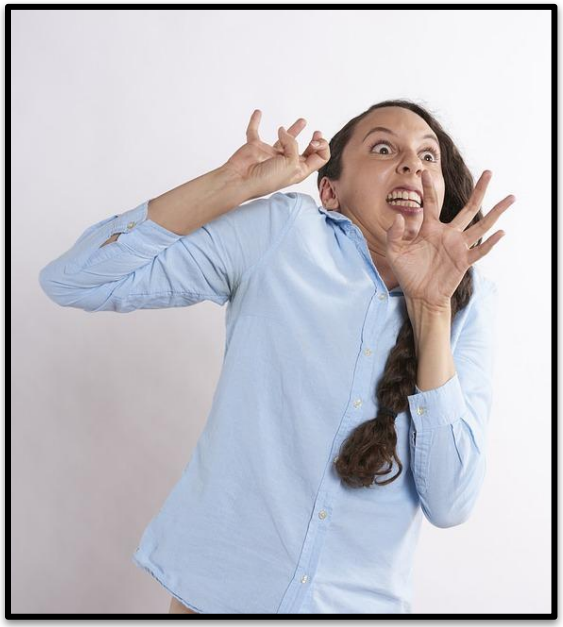
SRE @ Lightstep (a tracing and metrics platform) co-founded OTel

chronosphere

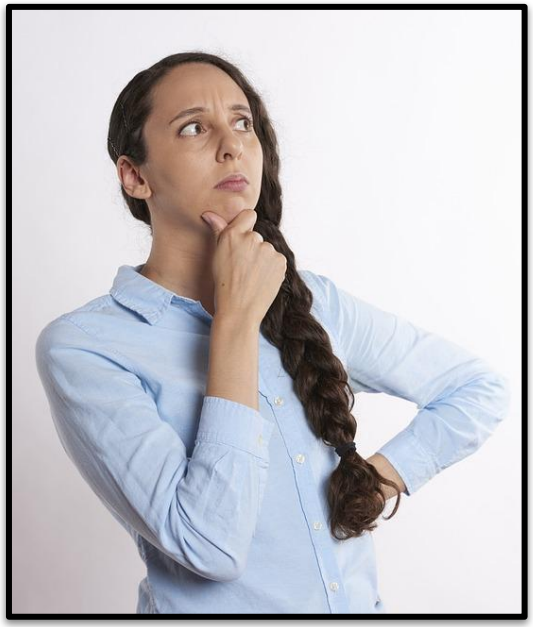
Now

OSS Instrumentation Advocate @ Chronosphere

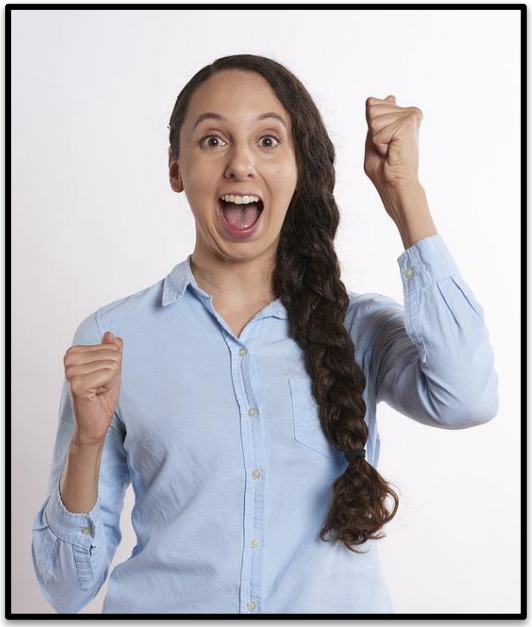
MFW: I think about tracing…



Never!!!



Considering it



Love it

What brought you to SCaLE?

I want to learn _____

Our agenda

- What is observability?
- What are key tracing concepts?
- What concepts do I need to use OpenTelemetry?
- How do I record data using OpenTelemetry?
- Where can I see my data?
- Next steps…

Observability Primer

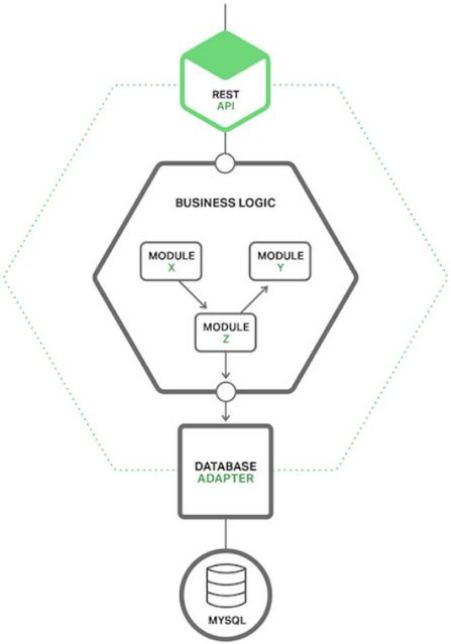
What is observability?

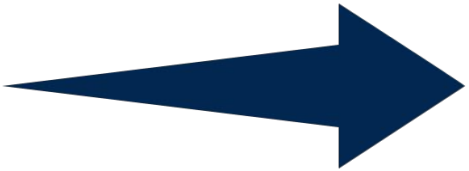
How effectively you can understand system behavior from the outside

Why observability?

- Microservices create complex interactions.
- Failures don't exactly repeat.
- Debugging has become painstaking.
- As architectures evolve so should the approaches for making sense of interactions

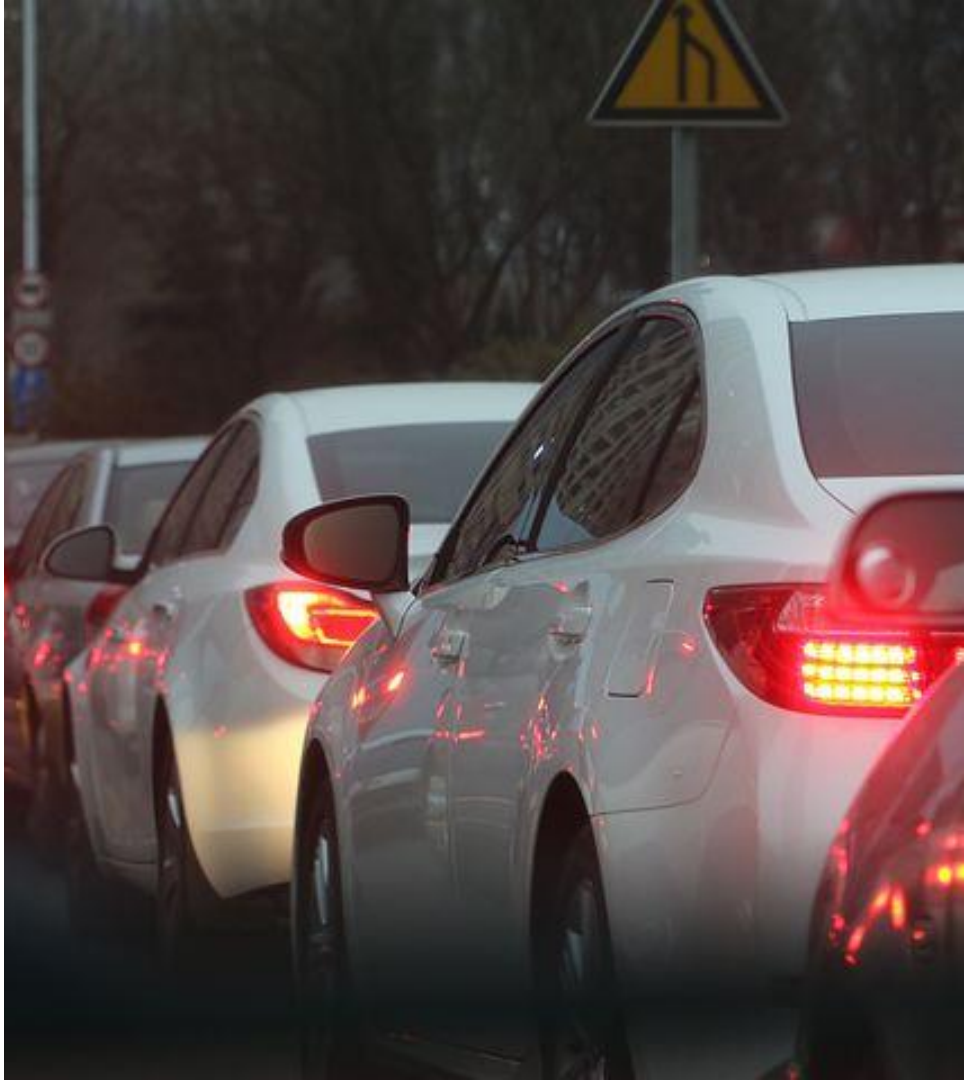
The Rise of Microservices on the Cloud





Evolving visibility

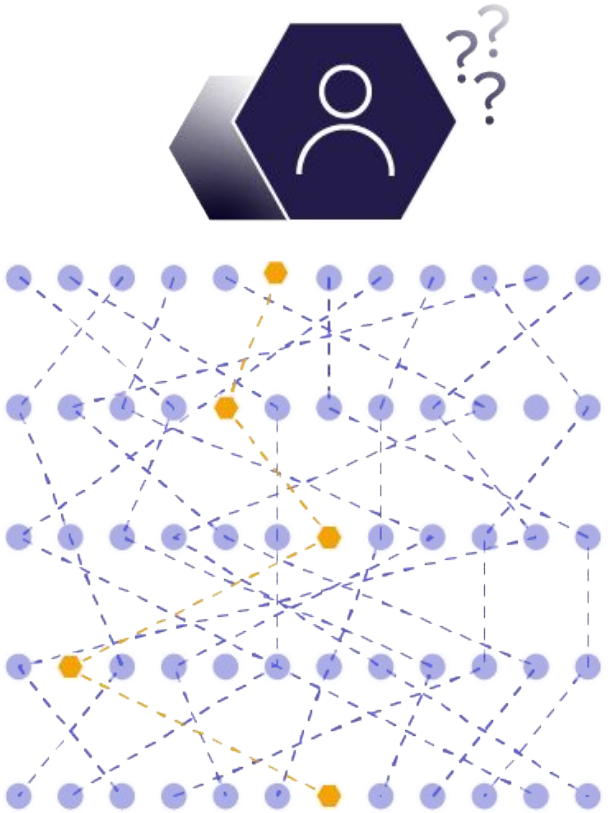
View of APM /Traditional Monitoring

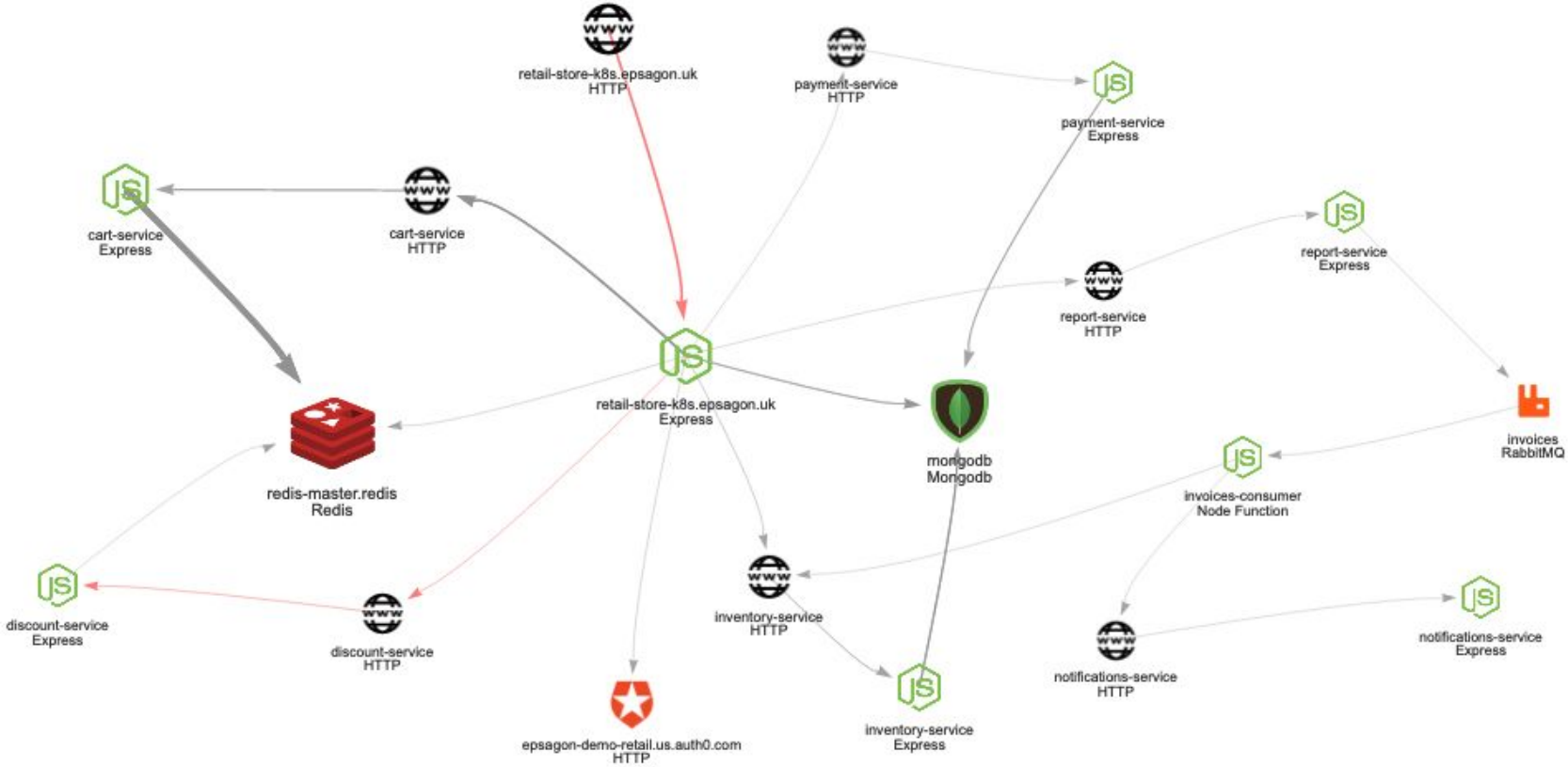


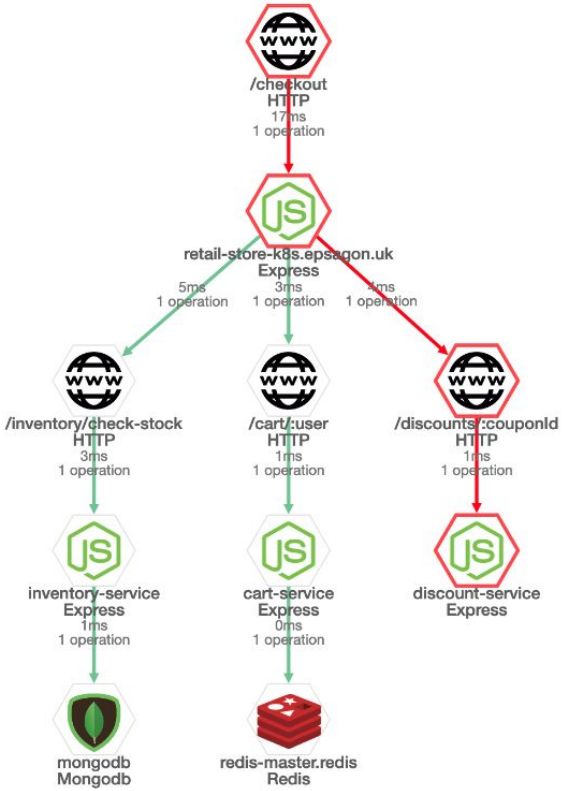
View with Distributed Tracing

Challenges

- Observe
- Troubleshoot
- Optimize




retail-store-k8s.epsagon.uk
HTTP
payment-service
HTTP
payment-service
Express
cart-service
Express
cart-service
HTTP
report-service
Express
report-service
HTTP
retail-store-k8s.epsagon.uk
Express
mongodb
Mongodb
invoices
RabbitMQ
redis-master.redis
Redis
invoices-consumer
Node Function
discount-service
Express
discount-service
HTTP
inventory-service
HTTP
notifications-service
Express
notifications-service
HTTP
epsagon-demo-retail.us.auth0.com
HTTP
inventory-service
Express

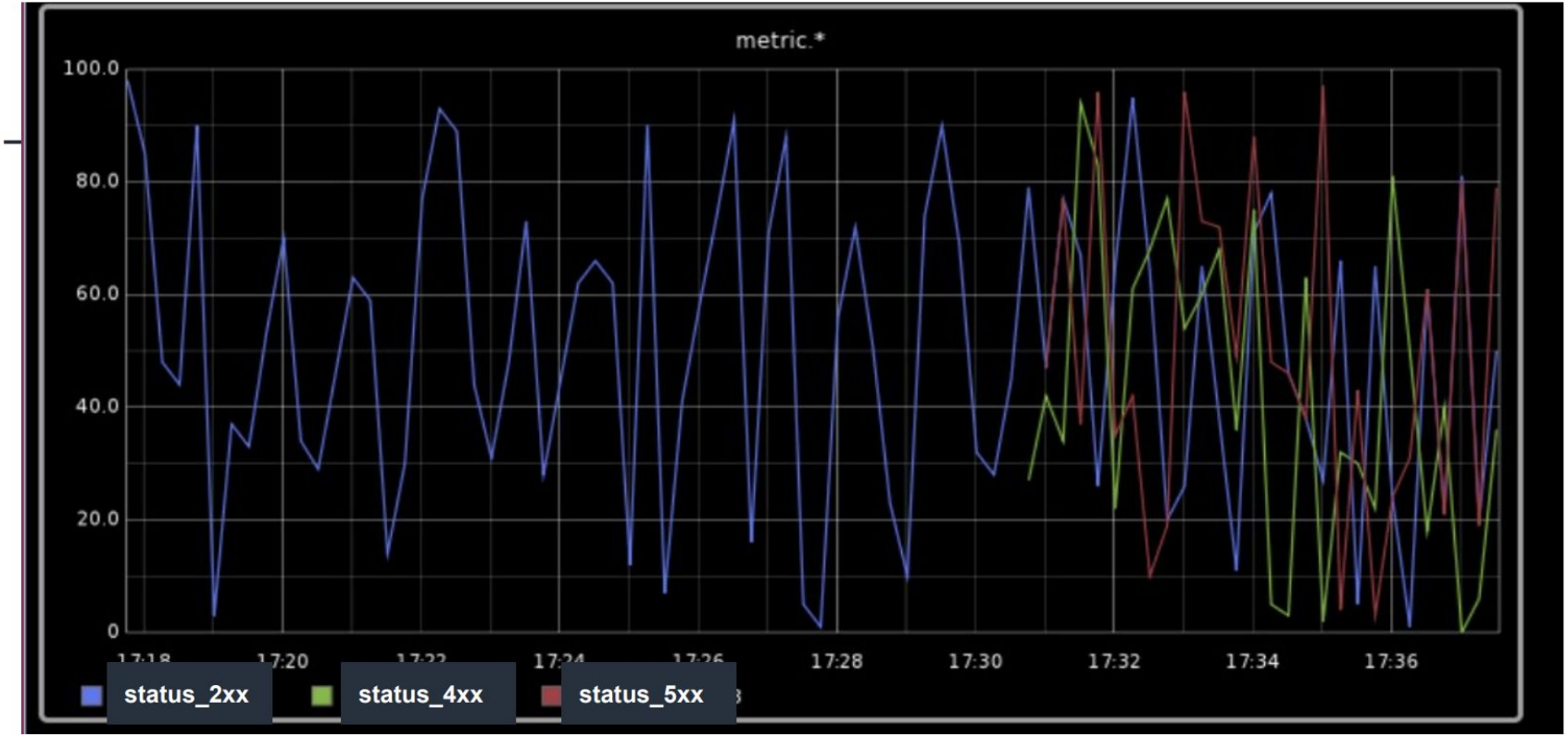

/checkout
HTTP
17ms
1 operation
retail-store-k8s.epsagon.uk
Express
5ms
1 operation
3ms
1 operation
1ms
1 operation
/inventory/check-stock
HTTP
3ms
1 operation
/cart/:user
HTTP
1ms
1 operation
/discounts/:couponId
HTTP
1ms
1 operation
inventory-service
Express
1ms
1 operation
cart-service
Express
0ms
1 operation
discount-service
Express
mongodb
Mongodb
redis-master.redis
Redis

Understanding Your System

What traits did the requests that timed out at 500ms share in common? Service versions? Browser plugins? Node pool?

- Instrumentation produces data
- Querying data answers our questions.

The oncall sends you this chart showing a new spike in 400 & 500s from your service...



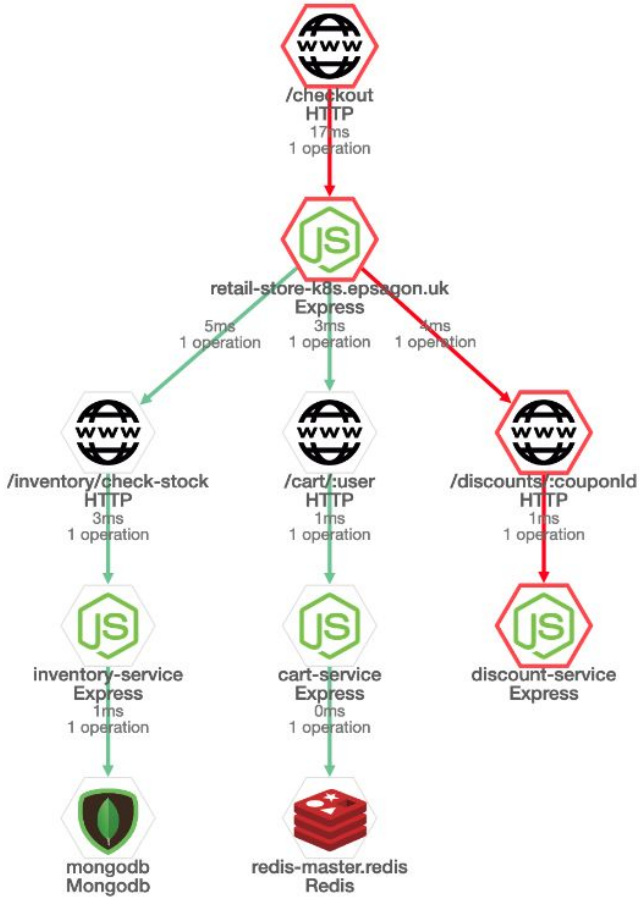
Telemetry aids observability

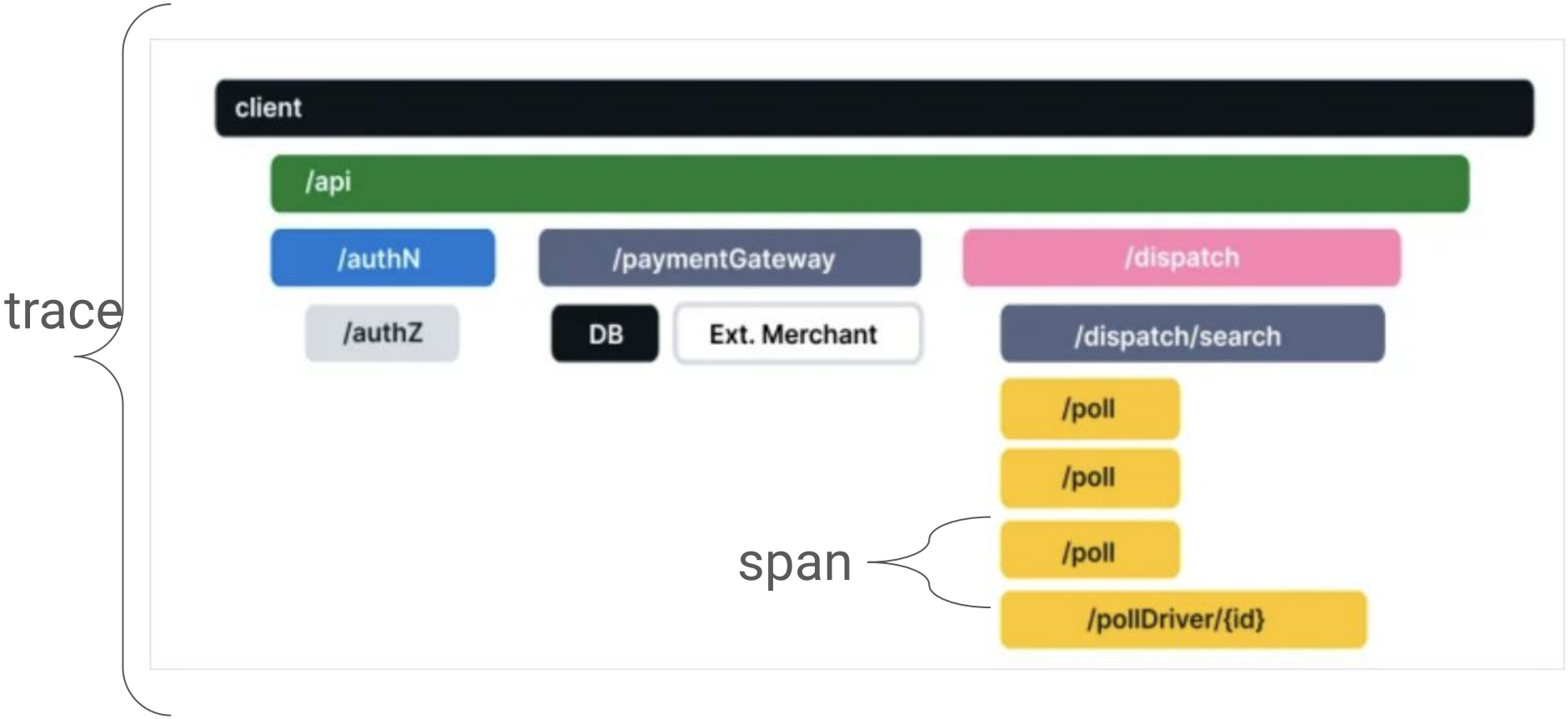
- Telemetry data **isn't** observability itself.
- Instrumenting code is *how* we get telemetry.
- Telemetry data can include traces, logs, and/or metrics.

Our agenda

- ~~What is observability?~~
- What are key tracing concepts?
- What do I know to use OpenTelemetry?
- How do I record data using OpenTelemetry?
- Where can I see my data?
- Next steps…

Tracing Concepts


/checkout
HTTP
17ms
1 operation
retail-store-k8s.epsagon.uk
Express
5ms
1 operation
3ms
1 operation
4ms
1 operation
/inventory/check-stock
HTTP
3ms
1 operation
/cart/:user
HTTP
1ms
1 operation
/discounts/:couponId
HTTP
1ms
1 operation
inventory-service
Express
1ms
1 operation
cart-service
Express
0ms
1 operation
discount-service
Express
mongodb
Mongodb
redis-master.redis
Redis


client
/api
/authN
/paymentGateway
/dispatch
/authZ
DB
Ext. Merchant
/dispatch/search
/poll
/poll
/poll
/pollDriver/{id}
trace
span

Tracing concepts in a nutshell

Trace

- Defined implicitly by its **spans**. A **trace** can be thought of as a directed acyclic graph of **spans** where the edges between **spans** are defined as parent/child relationships.

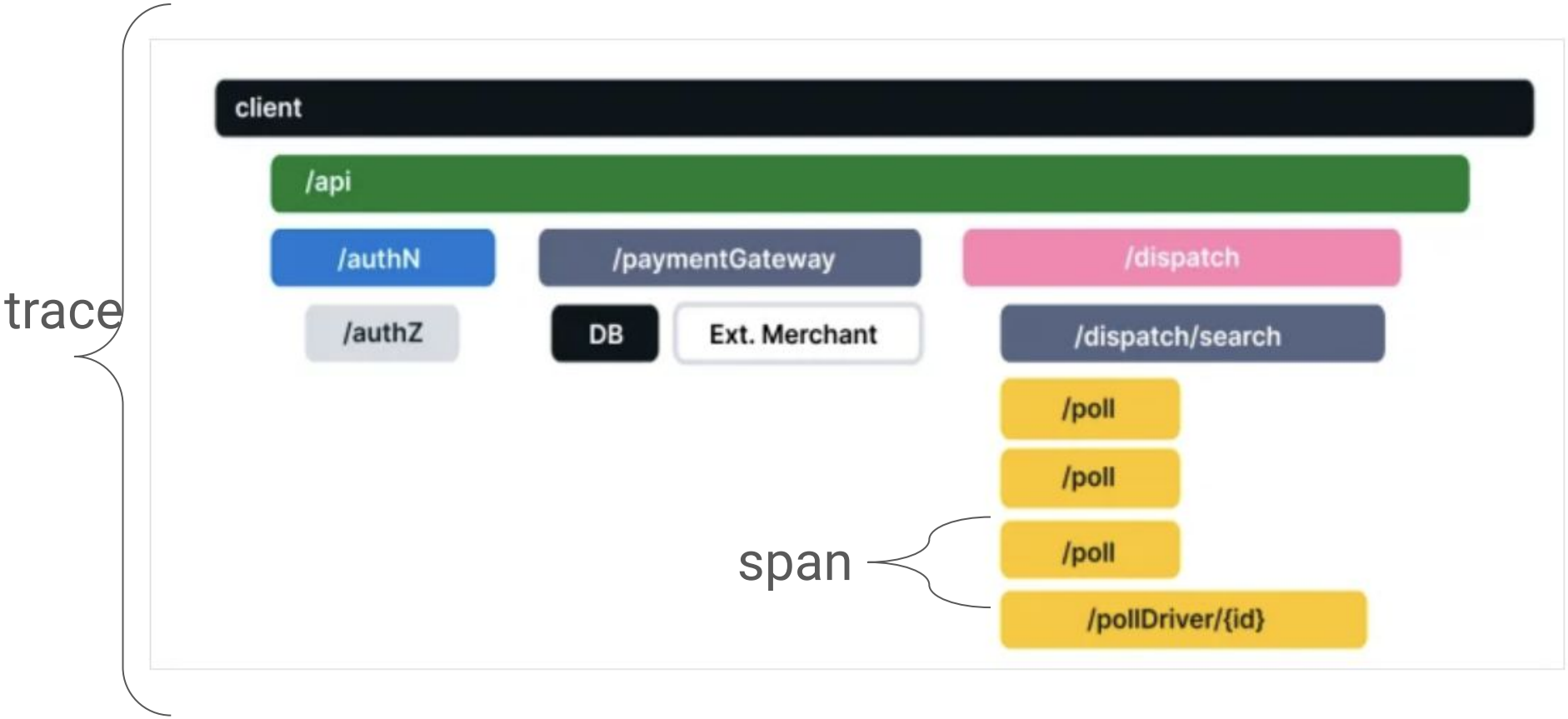
Tracing concepts in a nutshell

Span

- Represents a **single unit of work** in a system.
- Typically encapsulates: operation name, a start and finish timestamp, the parent span identifier, the span identifier, and context items.

Span


```
{
   "name": "Hello-Greetings",
   "context": {
       "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
       "span_id": "0x5fb397be34d26b51",
   },
   "parent_id": "0x051581bf3cb55c13",
   "start_time": "2022-04-29T18:52:58.114304Z",
   "end_time": "2022-04-29T22:52:58.114561Z",
   "attributes": {
       "http.route": "some_route1"
   },
   "events": [
       {
           "name": "hey there!",
           "timestamp": "2022-04-29T18:52:58.114561Z",
           "attributes": {
               "event_attributes": 1
           }
       },
       {
           "name": "bye now!",
           "timestamp": "2022-04-29T18:52:58.114585Z",
           "attributes": {
               "event_attributes": 1
           }
       }
   ],
}
```



client
/api
/authN
/paymentGateway
/dispatch
/authZ
DB
Ext. Merchant
/dispatch/search
/poll
/poll
/poll
/pollDriver/{id}
trace
span

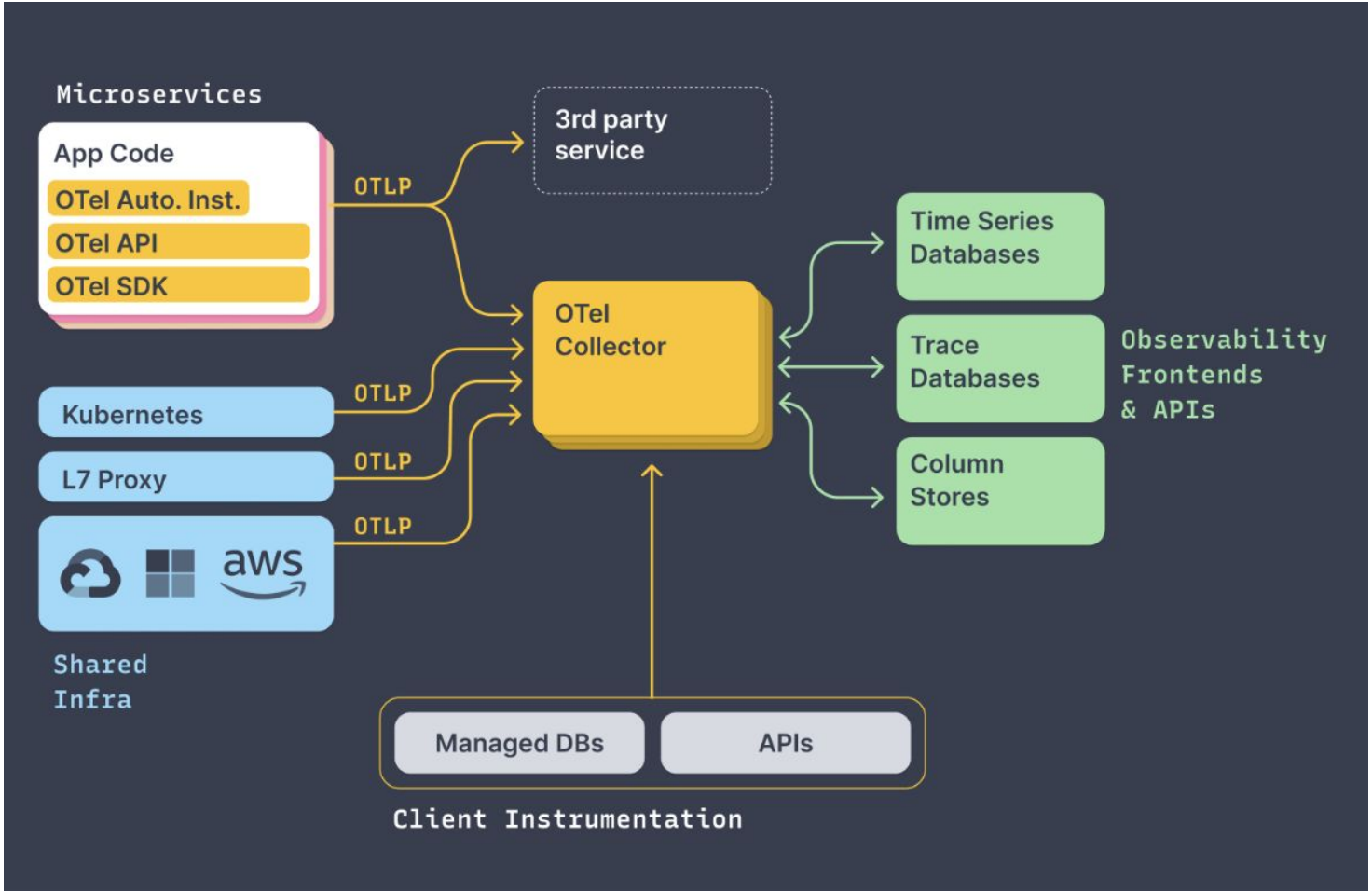
How do I implement these?

- You need an instrumentation framework!
- and a place to send the data!
- and a way to visualize the data!

Our agenda

- ~~What is observability?~~
- ~~What are key tracing concepts?~~
- What do I know to use OpenTelemetry?
- How do I record data using OpenTelemetry?
- Where can I see my data?
- Next steps…

Hello, OpenTelemetry!


Microservices
App Code
OTel Auto. Inst.
OTel API
OTel SDK
OTLP
3rd party service
OTel Collector
Time Series Databases
Trace Databases
Column Stores
Observability Frontends & APIs
Kubernetes
L7 Proxy
aws
OTLP
OTLP
OTLP
Shared Infra
Managed DBs
APIs
Client Instrumentation


VMware Aria
UPTRACE
servicepilot
sumo logic
DaoCloud
logz.io
aws
solarwinds
Aspecto
splunk>
observIQ
Promscale
f5
Coralogix
chronosphere
LogicMonitor
INSTANA
new relic
DATADOG
dynatrace
lumigo
APPDYNAMICS
elastic
Grafana Labs
logscale
Helios
SigNoz
Azure
honeycomb.io

Tracing

- **API**: stable, feature-freeze
- **SDK**: stable
- **Protocol**: stable
- Notes:
  - The tracing specification is now completely stable, and covered by long term support.
  - The tracing specification is still extensible, but only in a backwards compatible manner.
  - OpenTelemetry clients are versioned to v1.0 once their tracing implementation is complete.

Metrics

- **API**: stable
- **SDK**: mixed
- **Protocol**: stable
- Notes:
  - OpenTelemetry Metrics is currently under active development.
  - The data model is stable and released as part of the OTLP protocol.
  - Experimental support for metric pipelines is available in the Collector.
  - Collector support for Prometheus is under development, in collaboration with the Prometheus community.

Logging

- **API:** draft
- **SDK:** draft
- **Protocol:** stable
- Notes:
  - OpenTelemetry Logging is currently under active development.
  - The logs data model is released as part of the OpenTelemetry Protocol.
  - Log processing for many data formats has been added to the Collector, thanks to the donation of Stanza to the OpenTelemetry project.
  - Log appenders are currently under development in many languages. Log appenders allow OpenTelemetry tracing data, such as trace and span IDs, to be appended to existing logging systems.
  - An OpenTelemetry logging SDK is currently under development. This allows OpenTelemetry clients to ingest logging data from existing logging systems, outputting logs as part of OTLP along with tracing and metrics.
  - An OpenTelemetry logging API is not currently under development. We are focusing first on integration with existing logging systems. When metrics is complete, focus will shift to development of an OpenTelemetry logging API.

Is Your Framework Instrumented?

Think of the tech stack you use for projects or at work …

Head to the Registry
https://opentelemetry.io/ecosystem/registry/

Share what you find!
Any languages or frameworks missing? Any that surprised you?

Our agenda

- ~~What is observability?~~
- ~~What are key tracing concepts?~~
- ~~What do I know to use OpenTelemetry?~~
- How do I record data using OpenTelemetry?
- Where can I see my data?
- Next steps…

Instrumenting (aka recording data)

SDKs, Exporters, and Collector Services, Oh My!

- OpenTelemetry's **SDK** implements trace & span creation.
- An **exporter** can be instantiated to send the data collected by OpenTelemetry to the backend of your choice.
- OpenTelemetry **collector** proxies data between instrumented code and backend service(s). The exporters can be reconfigured without changing instrumented code.

Manual vs Automatic Instrumentation

Automatic Instrumentation

```
@app.route("/server_request")
def server_request():
    print(request.args.get("param"))
    return "served"
```

Manual Instrumentation

```
@app.route("/server_request")
def server_request():
    with tracer.start_as_current_span(
        "server_request",
        context=extract(request.headers),
        kind=trace.SpanKind.SERVER,
        attributes=collect_request_attributes(request.environ),
    ):
        print(request.args.get("param"))
        return "served"
```

Tracer methods, & when to call

- tracer.start_span(name, parent=, ...)
  - This method returns a child of the specified span.
- with tracer.start_as_current_span(name)
  - Starts a new span, sets it to be active. Optionally, can get a reference to the span.
- tracer.get_current_span()
  - Used to access & add information to the current span

Span methods, & when to call

- span.add_event(name, attributes)
  - Adds structured annotations (e.g. "logs") about what is currently happening.
- span.set_attribute(key, value)
  - Adds an attribute to the current span. This may include a user id, a build id, a user-agent, etc.
- span.end()
  - Manually closes a span.

Add more context to traces with Span Events

- Span Events are context-aware logging.
- An `event` contains timestamped information added to a span. You can think of this as a structured log, or a way to annotate your spans with specific details about what happened along the way.
  - Contains:
    - the name of the event
    - one or more attributes
    - a timestamp

Code examples: Current Span & Span

- Get the current span
  - `span = tracer.get_current_span()`
- Update the span status
  - `span.set_status(Status(StatusCanonicalCode.UNKNOWN, error))`
- Add events
  - `span.add_event("foo", {"customer": "bar"})`
- Add attributes
  - `span.set_attribute("error", True)`

Auto-instrument

pip3 install opentelemetry-distro

```
pip3  install opentelemetry-distro

opentelemetry-bootstrap -a install

opentelemetry-instrument \
    --traces_exporter console \
    --metrics_exporter console \
    flask run
```

```
opentelemetry-instrument \
    --traces_exporter console \
    --metrics_exporter console \
    flask run
```

```
{
    "name": "/visit",
    "context": {
        "trace_id": "0xa2f924bb3a05138820631f9913c84487",
        "span_id": "0xa79dd2bc621c37fc",
        "trace_state": "[]"
    },
    "kind": "SpanKind.SERVER",
    "parent_id": null,
    "start_time": "2023-03-09T21:19:12.509247Z",
    "end_time": "2023-03-09T21:19:12.510387Z",
    "status": {
        "status_code": "UNSET"
    },
    "attributes": {
        "http.method": "GET",
        "http.server_name": "127.0.0.1",
        "http.scheme": "http",
        "net.host.port": 5000,
        "http.host": "localhost:5000",
        "http.target": "/visit",
        "net.peer.ip": "127.0.0.1",
        "http.user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36",
        "net.peer.port": 56044,
        "http.flavor": "1.1",
        "http.route": "/visit",
        "http.status_code": 200
    },
    "events": [],
    "links": [],
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python",
            "telemetry.sdk.name": "opentelemetry",
            "telemetry.sdk.version": "1.16.0",
            "telemetry.auto.version": "0.37b0",
            "service.name": "unknown_service"
        },
        "schema_url": ""
    }
}
```


Manually instrument

app.py

```
1   from flask import Flask
2   from opentelemetry import trace
3   from opentelemetry.sdk.trace import TracerProvider
4   from opentelemetry.sdk.trace.export import BatchSpanProcessor
5   from opentelemetry.exporter.otlp.proto.http.trace_exporter import OTLPSpanExporter
6
7
8   import random
9
10  trace.set_tracer_provider(TracerProvider())
11  trace.get_tracer_provider().add_span_processor(
12      BatchSpanProcessor(OTLPSpanExporter())
13  )
14
15  # Acquire a tracer
16  tracer = trace.get_tracer(__name__)
17
```

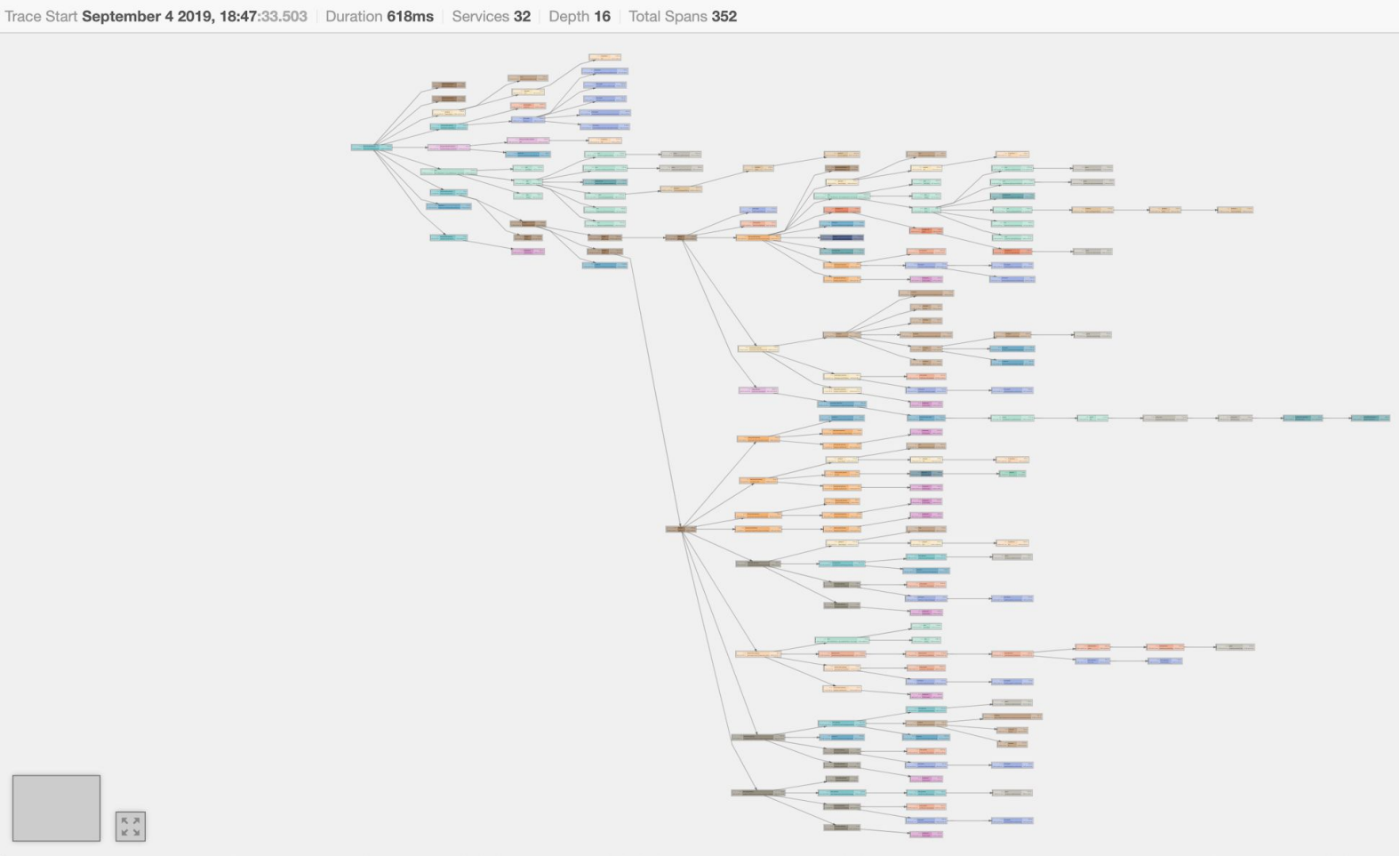
app.py

```
31
32  @app.route("/rolldice")
33  def roll_dice():
34      return str(do_roll())
35
36  def do_roll():
37      with tracer.start_as_current_span("do_roll") as rollspan
38          result = random.randint(1, 6)
39          rollspan.set_attribute("roll.value", result)
40          return result
41
42  if __name__ == "__main__":
43      app.run(host="0.0.0.0", port=8000, debug=True)
44
```

```
pip3  install -r requirements.txt
```

OTEL_SERVICE_NAME=appy flask run

Jaeger!


Trace Start September 4 2019, 18:47:33.503 | Duration 618ms | Services 32 | Depth 16 | Total Spans 352

```
docker run -d --name jaeger \
  -e COLLECTOR_OTLP_ENABLED=true \
  -p 16686:16686 \
  -p 4317:4317 \
  -p 4318:4318 \
  jaegertracing/all-in-one:1.42
```

Go look for your trace!

The Jaeger visualization URL is at (notice the port number):

http://localhost:16686/search

Put in your `SERVICE_NAME` value into the service name, and search for your recent traces!

Our agenda

- ~~What is observability?~~
- ~~What are key tracing concepts?~~
- ~~What do I know to use OpenTelemetry?~~
- ~~How do I record data using OpenTelemetry?~~
- ~~Where can I see my data?~~
- Next steps…

Community

- Documentation
- GitHub
- CNCF Slack
- SIG meetings

TIL _____