Instrumentation 201

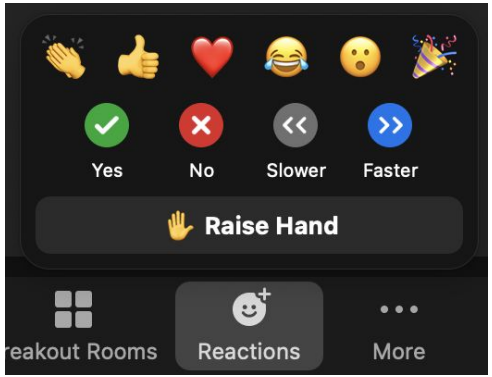
Pierre Tessier ✨
Michael Sickles ✨

Some quick Housekeeping

Everyone can speak.
Please remain on mute unless called upon to speak.

Asking questions: Raise your hand or use chat

Do not unmute to say “ok”, use reactions instead.



Slack channel: #advanced-instrumentation-workshop

Live captions available at: https://www.streamtext.net/player?event=honeycombiocaptions

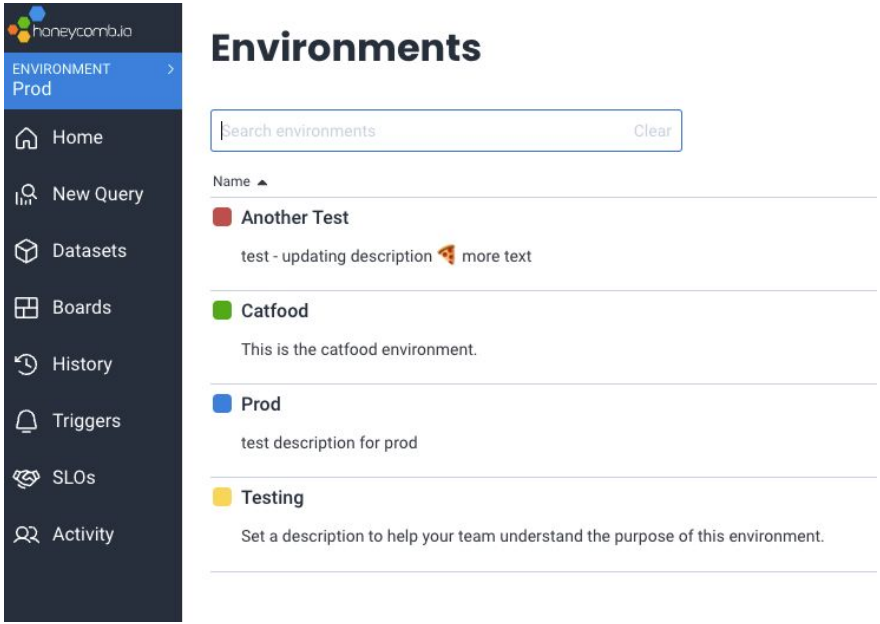
Honeycomb – Environments and Services

A better way to organize your data with Services as a first class citizen

All new teams created after March 26th

Dataset is automatically inferred based on service name

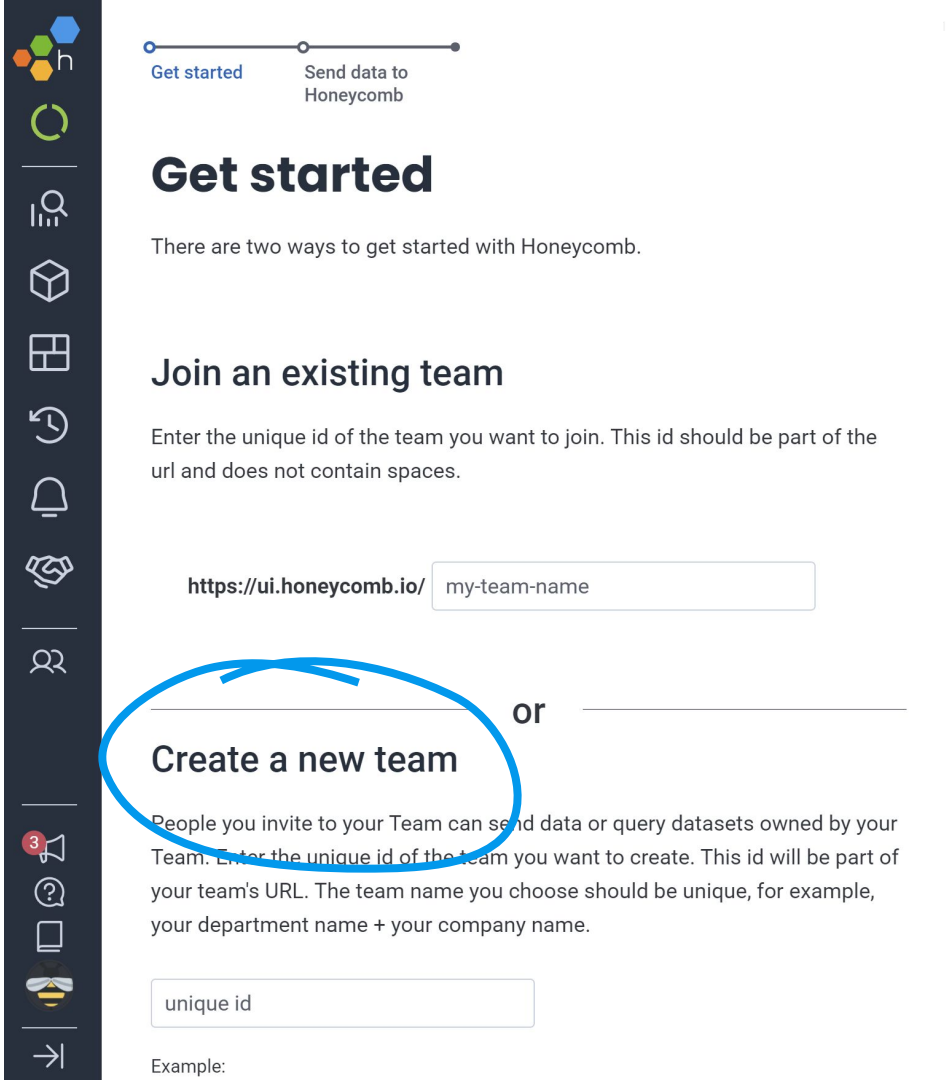
`HONEYCOMB_DATASET` is not required for these teams.



Create a team

When you first sign in, you can join a team or create a team.

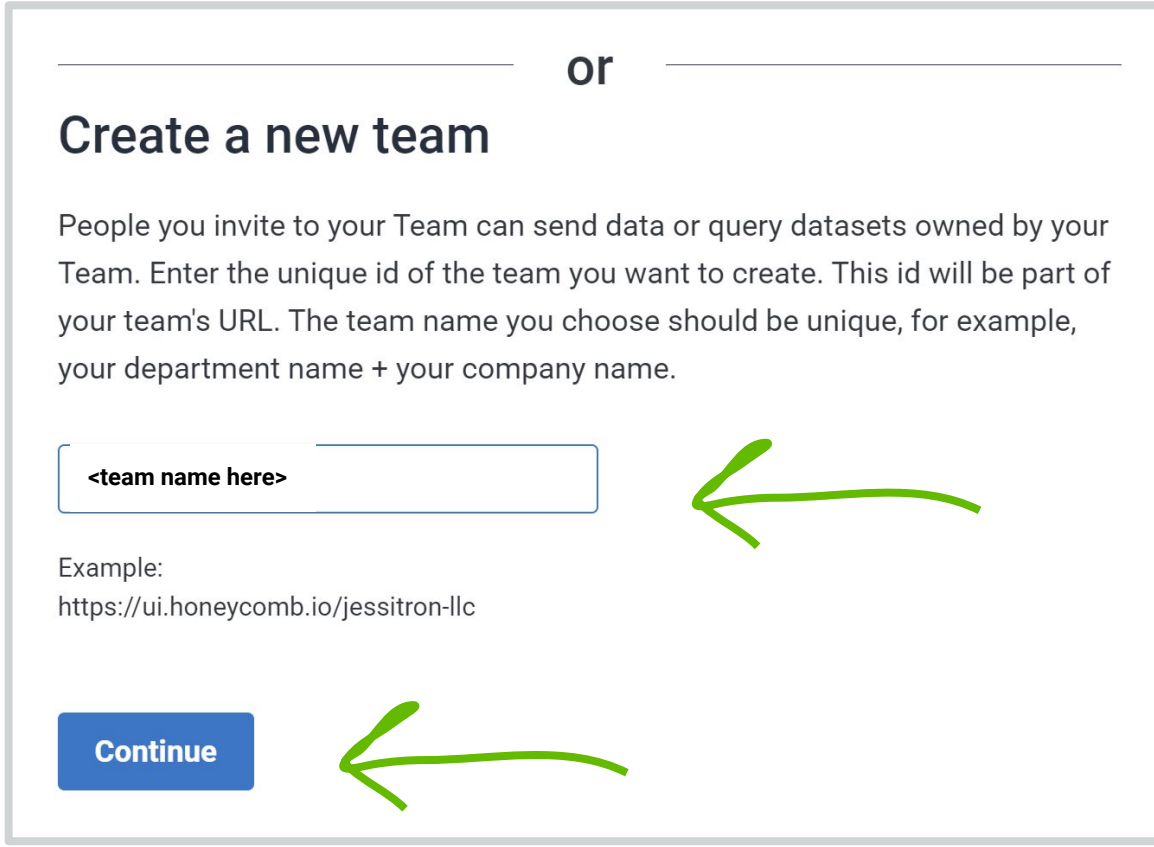
Create a new team!



Create a team

When you first sign in, you can join a team or create a team.

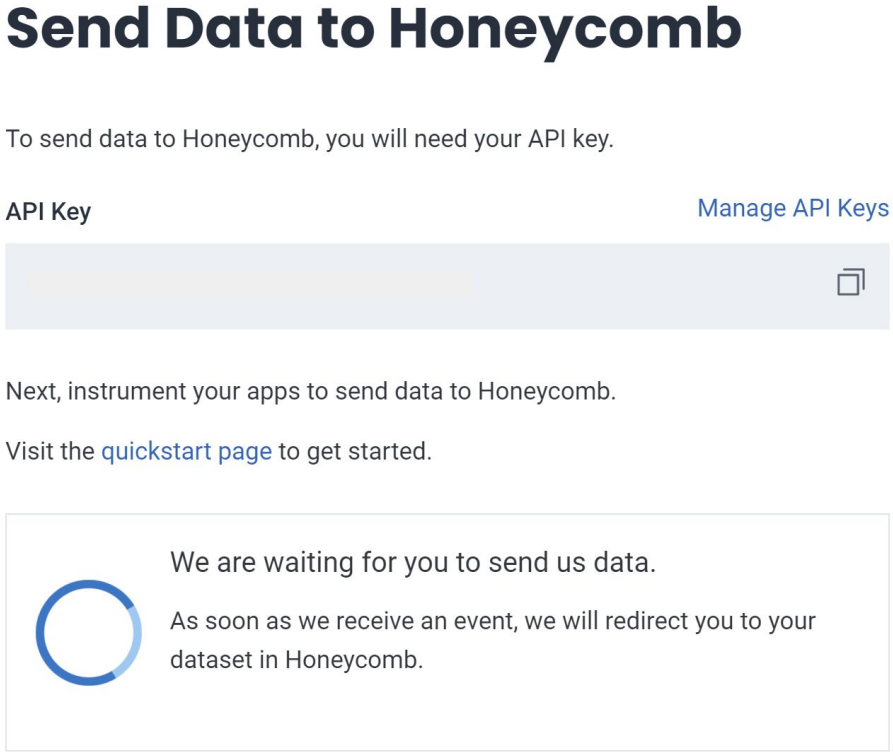
Create a new team!



Get a Honeycomb API Key

Get started — Send data to Honeycomb

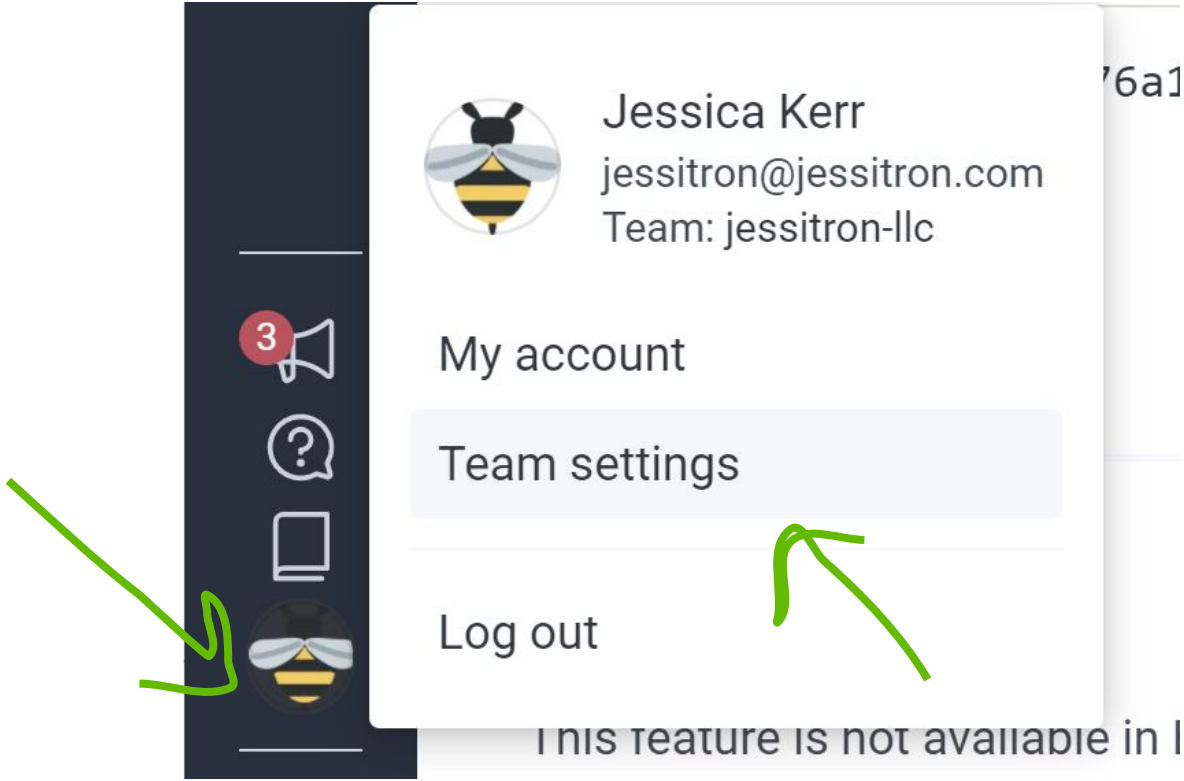
The first time you create a team, it takes you directly to your API Key.



Honeycomb API Key (classic teams)

If you already have an account:

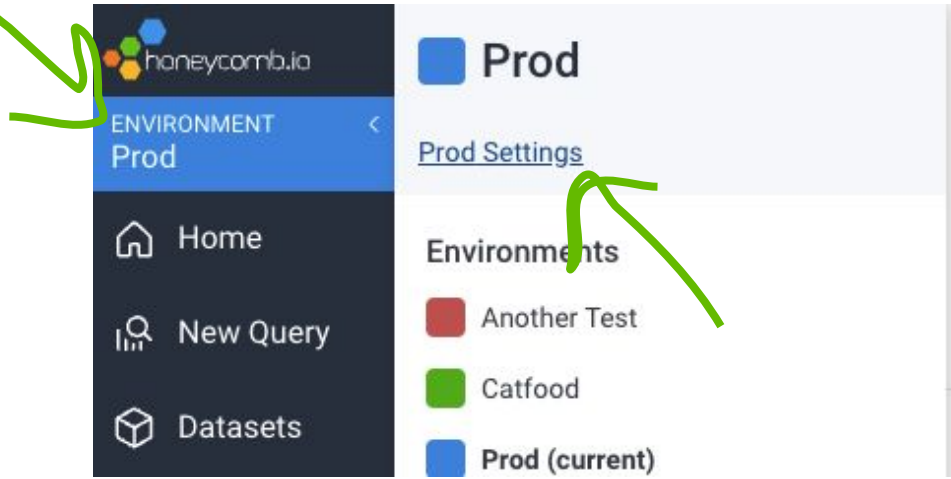
1. Select your profile picture in the lower left corner
2. Choose **Team settings**



Honeycomb API Key (E&S teams)

If you already have an account:

1. Click on the Environment name at the top left on the navigation bar
2. Click on **[name] settings**



Before we get started: create account

Create a GitPod user account

gitpod.io

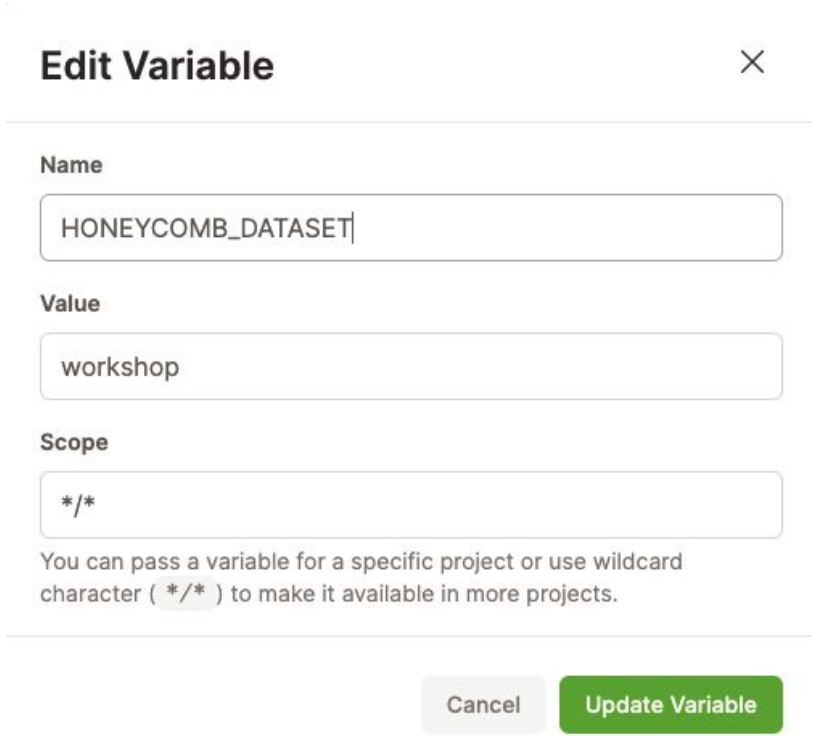
Before we get started: set up environment

Create 2 user environment variables
gitpod.io/variables

`HONEYCOMB_API_KEY` - set this to your Honeycomb Team API Key

`HONEYCOMB_DATASET` (classic)- set this to `workshop` (or any name you like)

Set Scope to `*/*` for both variables



Before we get started: Initialize workspace

GitPod
gitpod.io/#https://github.com/honeycombio/workshop-advanced-instrumentation

Local system: requires Java 11+, Gradle 7+, Go 1.14+, Node 12+, Python 3.7+

```
git clone https://github.com/honeycombio/workshop-advanced-instrumentation
```

Auto-instrumentation

Quick recap

Auto-instrumentation is...

Easy to Use

Fast to Value

Attributes for quick context

… but it only goes so far

Auto-instrumentation does not provide the deep **context** required to understand **your unique** application.

Manual Instrumentation

Creating new spans and traces

What does OpenTelemetry provide

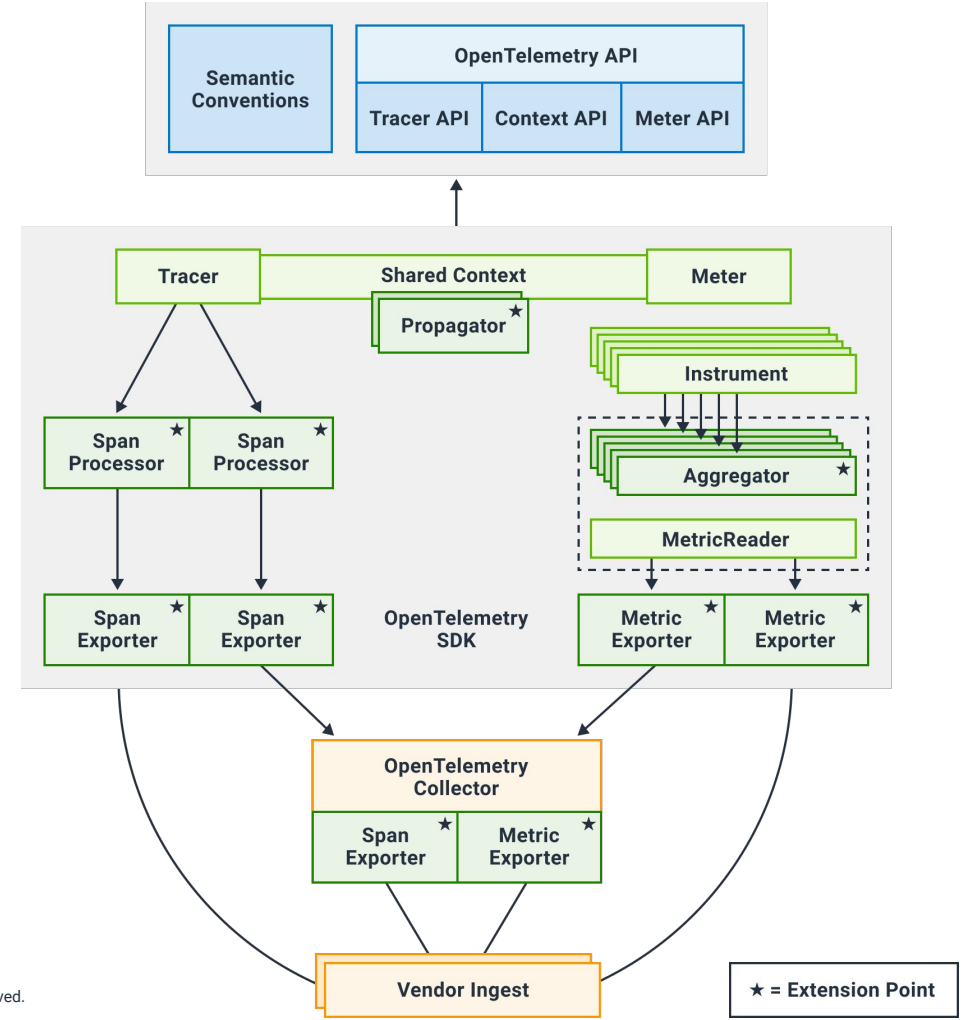
an API

… and an SDK

… and a Collector

… that can also run as an Agent

… all with Extension points



Context … in the right context

Different types of **Contexts**

Runtime: Java thread runtime, Go `Context`, Node global space

Trace SDK: What connects spans to create traces

Span: Specific to a span

Trace Propagation: Inter-process communication

Application: Your application's domain

OpenTelemetry Trace SDK Context

Is used to connects spans, creating a trace

Working with Trace Context differs between languages

- **Java**: implicit thread context
- **Go**: explicit context
- **Node**: implicit context
- **Python**: implicit context

Why do you need a new span?

Understand critical application logic

Understand intensive processing routines

Wrap auto-instrumented code

Wrapping a function the easy way in Java

```
@WithSpan("name-this")
public void myFunction() {
    // app logic
}
```

Wrapping code with a Span (Java)

```
Span span = tracer.spanBuilder("name-this").startSpan();
try (Scope scope = span.makeCurrent()) {

    // app logic

} catch (Throwable t) {
    span.setStatus(StatusCode.ERROR);
} finally {
    span.end();
}
```

Wrapping code with a Span (Go)

```
ctx, span := tracer.Start(ctx, "getYear")
defer span.End()

// app logic

if err != nil {
    span.SetStatus(codes.Error, “my error message”)
}
```

Wrapping code with a Span (Node)

```
const tracer = trace.getTracer("");
const span = tracer.startSpan("getYear");

// app logic

span.end();
```

Wrapping code with a Span (Python)

```
tracer = trace.get_tracer(__name__);

span = tracer.start_span("getYear")

# app logic

span.end()
```

Why do you need a new trace?

Auto-instrumentation is not available

Scheduled / batch jobs

Background process

Starting a trace (any language)

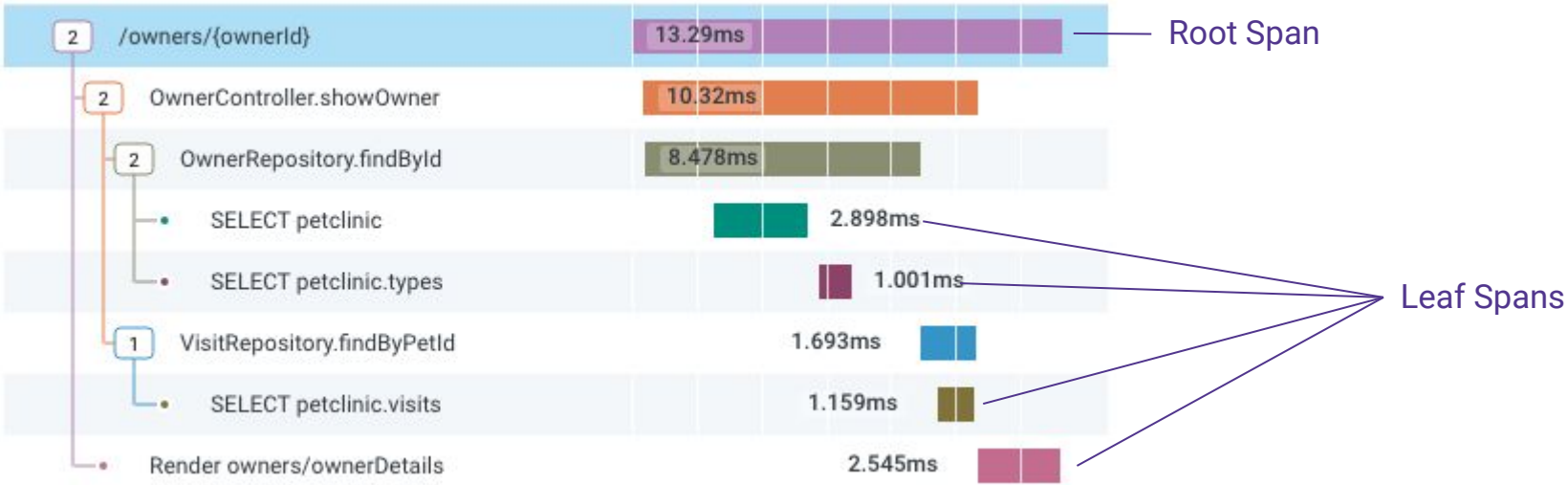
Create and start a span without context

Manual Instrumentation

Example #01

Trace Structure

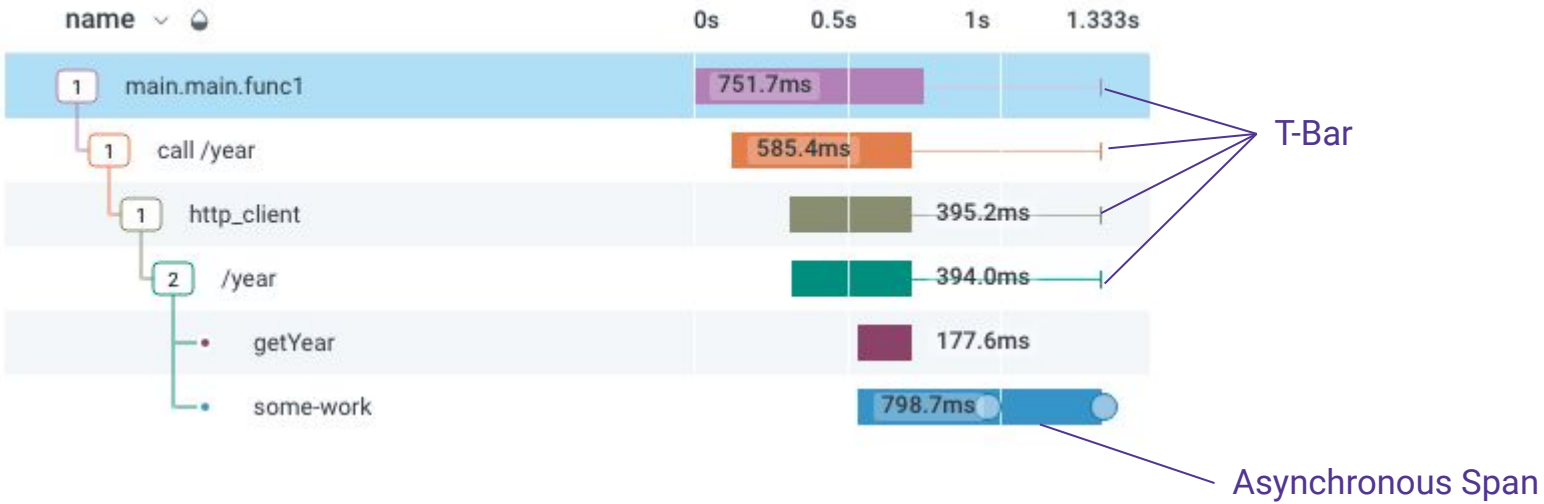
Parent / Child relationship



Asynchronous processing

A unit of work that happens on a **different thread** within the **trace context**

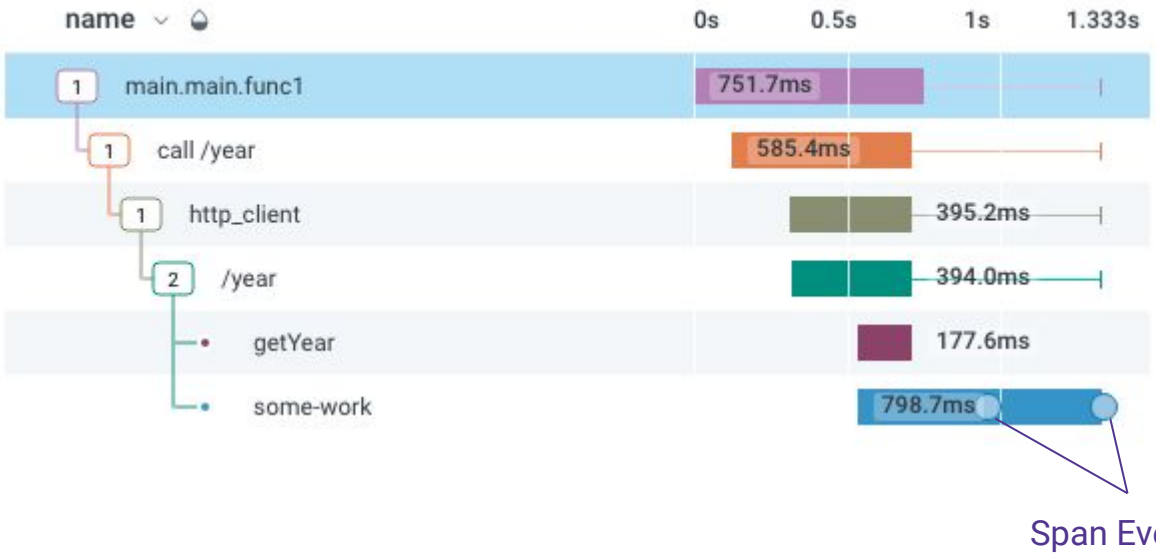
Asynchronous processing



Span Events

Something **notable** happened

Span Events



Span Events – how to

```
span.addEvent(name, attributes)
```

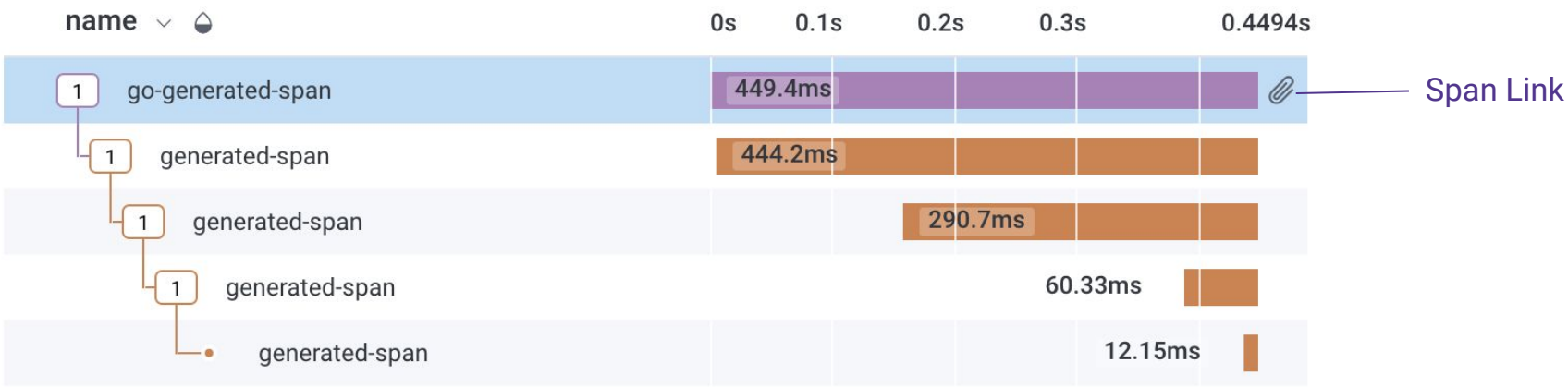
Span Links

Casual links between distinct traces

Batch -> controller creates multiple linked children

Batch -> single job linked to multiple inputs

Span Links



Span Links - how to Java

```
Span sourceSpan = Span.current();

Span span = tracer.spanBuilder("name-this")
            .addLink(sourceSpan.getSpanContext())
            .startSpan();
```

Span Links - how to Go

```
srcSpanCtx := trace.SpanContextFromContext(ctx)

ctx, span := tracer.Start(ctx, "name-this",
   trace.WithLinks(trace.Link{SpanContext: srcSpanCtx}))
```

Span Links - how to Node

```
let sourceSpan = trace.getSpan(context.active());

tracer.startSpan("name-this", {
   links: [{ context: sourceSpan.spanContext() }]
})
```

Span Links - how to Python

```
source_span = trace.get_current_span()
link = trace.Link(context=source_span.get_span_context())

tracer.start_span("name-this",
                  links=[link])
```

Trace Structure

Examples #02, 03, 04

Trace Propagation

Trace Context Explained

Identifiers

Trace Id
Uniquely identifies the trace
Present on every span

Span Id
Uniquely identifies the span
Present on every span

Glue

Parent Id
Span Id for this Span's parent
Present on every non-root span

Intra-Process Propagation

OpenTelemetry SDKs will provide this for you

Accomplished using thread or explicit context (in memory)

Inter-Process Propagation

Accomplished using headers/meta in communication between services

Requires SDKs to propagate and parse known **propagation types**

Propagation types (header formats)

Honeycomb

B3

Jaeger

Xray

W3C

(Otel default)

W3C - tracestate

Can be Used to propagate details beyond trace/span id

ie: `userid` should be known to all downstream spans

Zero to many key/value pairs passed to downstream spans

Accomplished using headers

Inter-process spans will have **network impact**

Enabled using **Baggage** in OpenTelemetry

Multi-Span Attributes

Using Baggage we can propagate attributes to descendant spans

but Baggage is not exported via OTLP 😰

(cue womp womp music)

Honeycomb's OpenTelemetry Distributions

A collection OpenTelemetry SDK wrappers

Quick Honeycomb bootstrapping with **100% OpenTelemetry API compatibility**

Baggage is exported as span attributes by default

Support for Java and .NET (coming soon)

more languages will be added in the coming months

Trace Propagation

Examples #05, 06

Tracing a message pipeline

Or a streaming events pipeline

Tracing a pipeline

Kafka, ActiveMQ, Amazon SQS, etc.

Understand the time spent in the pipeline

Continue a trace through the pipeline

Tracing a pipeline

Using OpenTelemetry, how do we propagate a trace through a pipeline?


IT JUST
WORKS

Tracing Kafka example

Sample application

Uses Spring Boot Getting Started for Kafka producer and consumer

GitPod:
https://gitpod.io/#https://github.com/McSick/confluent-otel-example.git

GitHub:
https://github.com/McSick/confluent-otel-example

Event Survey + i test in prod t-shirt!

We value your feedback and read every comment!

All respondents will receive a t-shirt upon completion. The survey closes tomorrow, April 29 at 12 p.m. PT.

Password:
OpenTelemetry

Please see the survey link in the Zoom chat window now!

Questions?

Upcoming events....

QCon Plus

May 11, 2022

Visibility into production behavior benefits product planning, organization design, and business decisions. With observability, we can get the software to help us change it smoothly and safely. Jessica Kerr, Honeycomb Developer Advocate, will explain how at QCon Plus.

SLOconf

May 9-12, 2022

Liz, Honeycomb Principal Developer Advocate, is speaking at the SLOConf. Come learn from our mistakes and be better able to implement streaming SLO evaluation at scale for true real-time SLO computation, action, and iteration on existing SLOs.

DevOpsDays Zurich

May 31, 2022

Charity, Honeycomb CTO & Co-founder, is keynoting at DevOpsDays Zurich. Join this in-person event to hear her talk about "The Sociotechnical Path to High-Performing Teams."

Thank you

Pierre Tessier - @PuckPuck

Michael Sickles - @McSick90


honeycomb.io