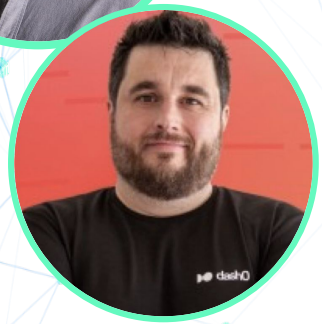
BUOYANT

Creators of LINKERD

+ dash0

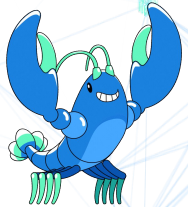




# Linkerd and OpenTelemetry


Michele Mancioppi, Head of Product @ Dash0

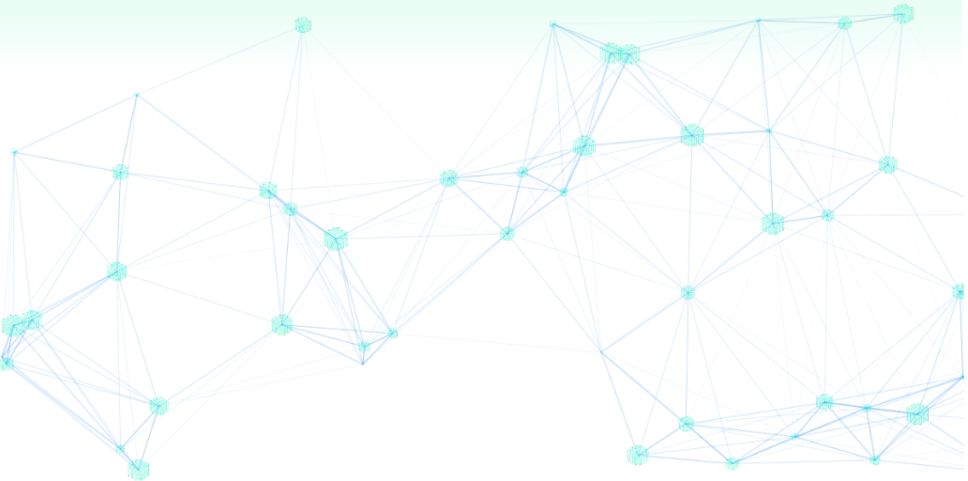
Flynn, Technical Evangelist for Linkerd

@BuoyantIO

buoyant.io

# What's on the agenda?



- What is Observability?
- What is OpenTelemetry?
  - and why should you care?
- Which telemetry do you need for your systems?
- How do you get OpenTelemetry going?
- OpenTelemetry + Linkerd = ❤️
- DEMO!

dash0 + BUOYANT

# How do you follow along?



- https://github.com/BuoyantIO/service-mesh-academy/tree/main/opentelemetry-and-linkerd
- **For this demo, you must use edge-25.2.1 or later.**
  - It'll work with Buoyant Enterprise for Linkerd 2.18 and later, when that ships!
- I'll be using a k3d cluster, but really any cluster should work fine.

dash0 + BUOYANT

# How do you follow along?



- **k3d**
  https://k3d.io/
- **kubectl**
  https://kubernetes.io/docs/tasks/tools/
- **linkerd CLI**
  https://linkerd.io/2/getting-started
- **bat**
  https://github.com/sharkdp/bat
- **jq**
  https://github.com/jqlang/jq
- **yq**
  https://github.com/mikefarah/yq

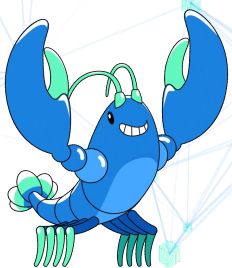
dash0 + BUOYANT

BUOYANT
SERVICE MESH
ACADEMY

# Prelude

# Observa... what??

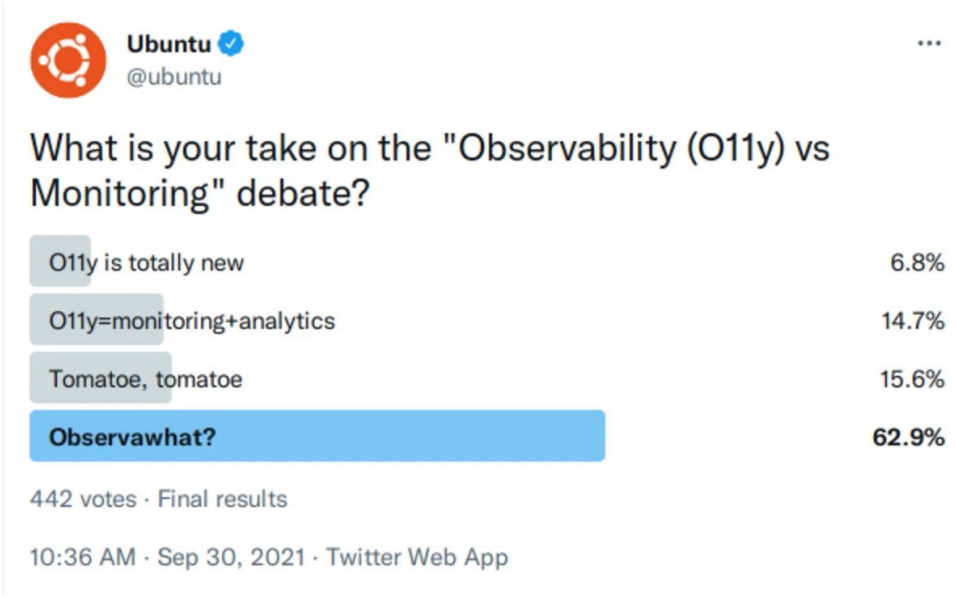
dash0

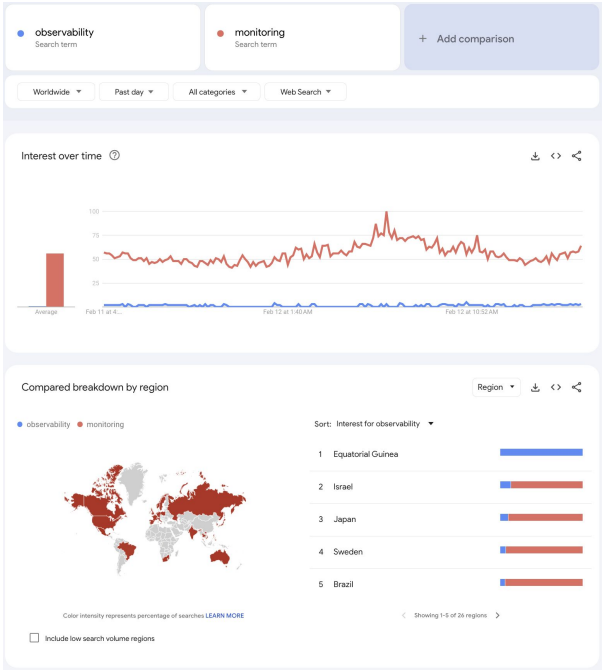


# If an application crashes but nobody is watching…



# Observability: depends on ~~who~~ where you ask




"[Observability vs. monitoring debate: An irreverent view]" *Michele Mancioppi, November 2021*

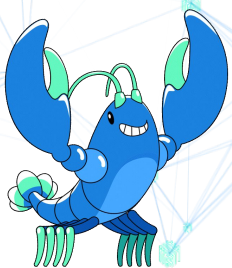
*Google Trends: Observability vs. Monitoring 2025-02-12, 16:30 CET*

BUOYANT
SERVICE MESH
ACADEMY

# Part 1

# What is OpenTelemetry?

dash0



# OpenTelemetry in a nutshell

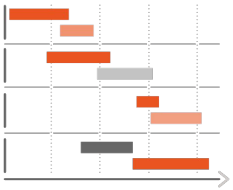
It is:

- A set of various things focused on letting you **collect** telemetry about systems:
  - Data models
  - API specifications
  - Semantic conventions
  - Library implementations in many languages
  - Utilities
  - and much more
- 2nd largest CNCF project
- Very, very cool

It isn't:

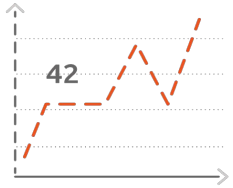
- Proprietary
- Driven by single or few vendors
- A query language
- Tools to ingest and query telemetry
- Feature complete
- Ever boring to work on/with
  - Not all excitement is good in software btw

# OpenTelemetry supports many *signals*



## Distributed tracing

*Rock solid, implemented in most languages.*



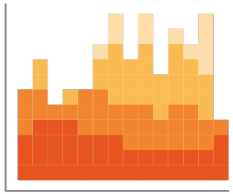

## Metrics

*Has cool things going on, but most people keep doing Prometheus.*




## Logs & events

*Usable in many languages, won't replace your logging agents any time soon.*



## Profiling

*Data model WIP, will be cool when it's done.*

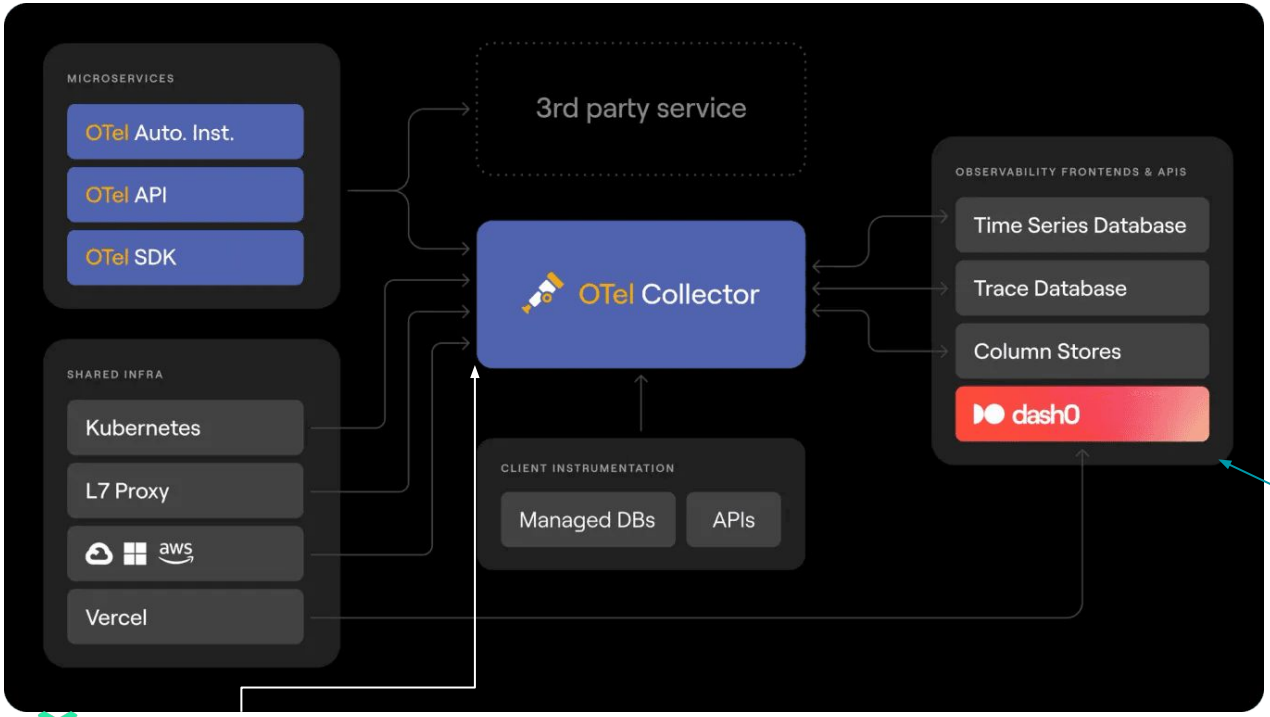


## Real-user monitoring

*Problem child, long and saaad story. Being reworked.*

*Images from: "What is observability?" ubuntu.com*

# OpenTelemetry: a 1000 miles view



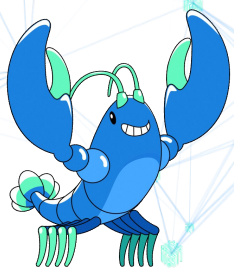

Shameless plug

BUOYANT
SERVICE MESH
ACADEMY

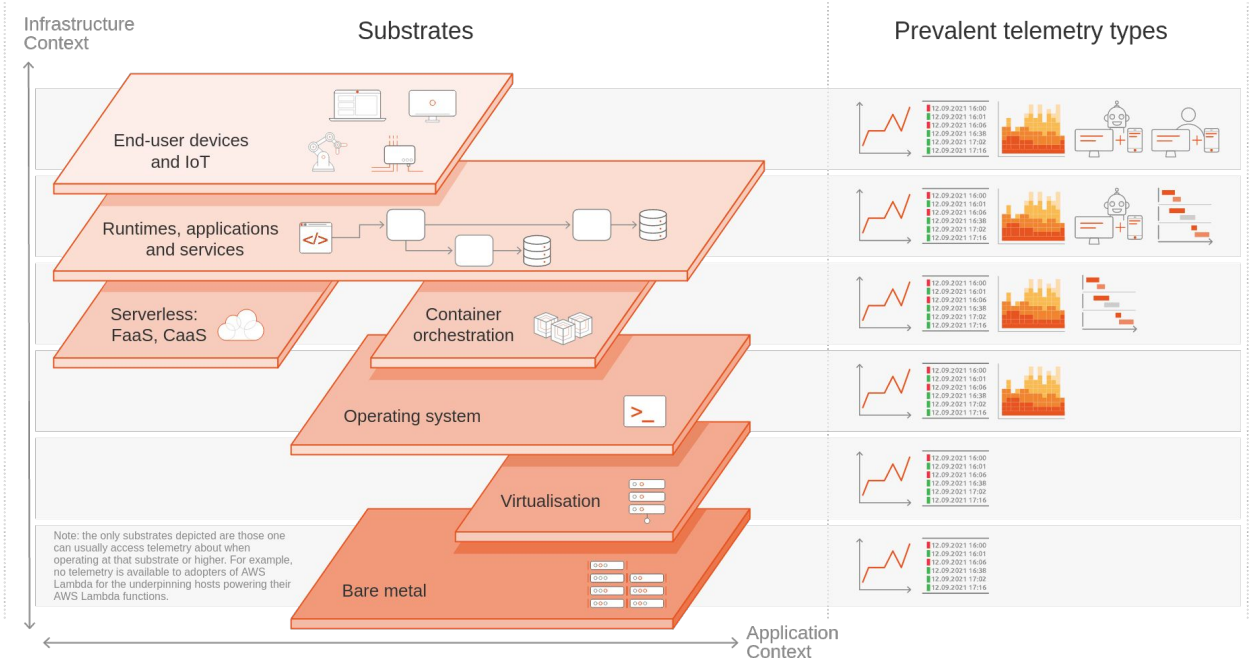
# Part 2

# Which telemetry do I need?

dash0



# Which types of telemetry do I need?



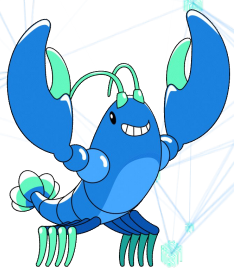

*From: "What is observability?" ubuntu.com*

BUOYANT
SERVICE MESH
ACADEMY

## Part 3

# How do I get myself some OpenTelemetry?

dash0



You have seen this coming a mile away, haven't you?



# Ok, for real now

For languages with runtime and bytecode instrumentation-like capabilities, start with **automatic instrumentation**:

- Java
- .NET Core
- Node.js
- Python
- Ruby
- …

For languages without runtime and bytecode instrumentation-like capabilities, **manual instrumentation**:

- Go
- Rust
- C++

Also, there's (some) **eBPF**, e.g. the (very much WIP) OpenTelemetry Go Automatic Instrumentation.

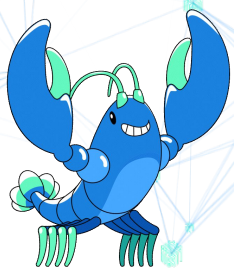
*eBPF likely won't be an option if you do not control the Linux kernel powering your apps, like on Amazon EKS/ECS on Fargate, AWS Lambda, etc.*

BUOYANT
SERVICE MESH
ACADEMY

# Part 4

# How do I get myself some OpenTelemetry with Linkerd?



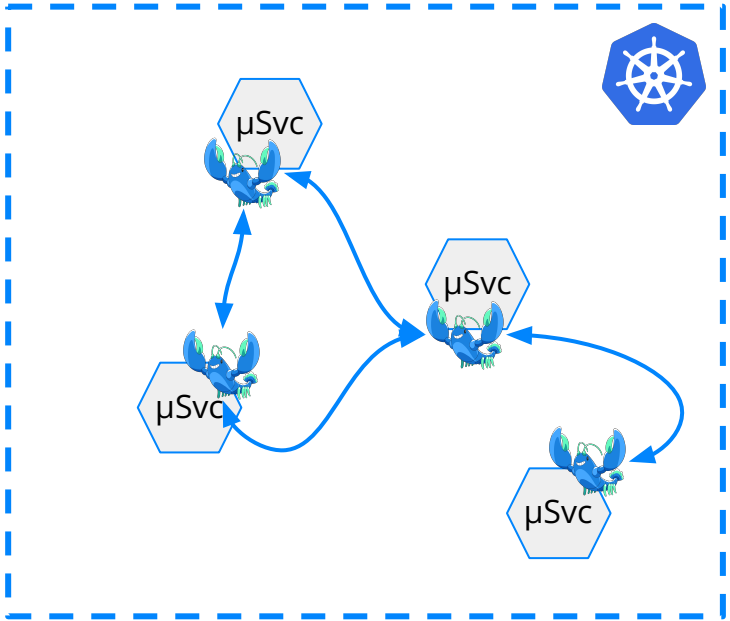



# You did *not* see that one coming, right?



# OpenTelemetry & Linkerd: EASY!

- Linkerd is a service mesh: its purpose in life is to add security, reliability, and **observability** to your application
- Like most meshes, Linkerd does this by adding proxies next to every application pod and letting them mediate and measure all application traffic
- This means that Linkerd is in a great place to help with OpenTelemetry…
- … and you need to integrate Linkerd in your observability setup



# OpenTelemetry & Linkerd: EASY!

Getting distributed tracing with OpenTelemetry going in Linkerd is as simple as installing and configuring the `jaeger` (?!?) extension

- Comes with a built-in Jaeger and OpenTelemetry collector, if you want it
- But most likely you already have your own collector, e.g., via the OpenTelemetry Operator

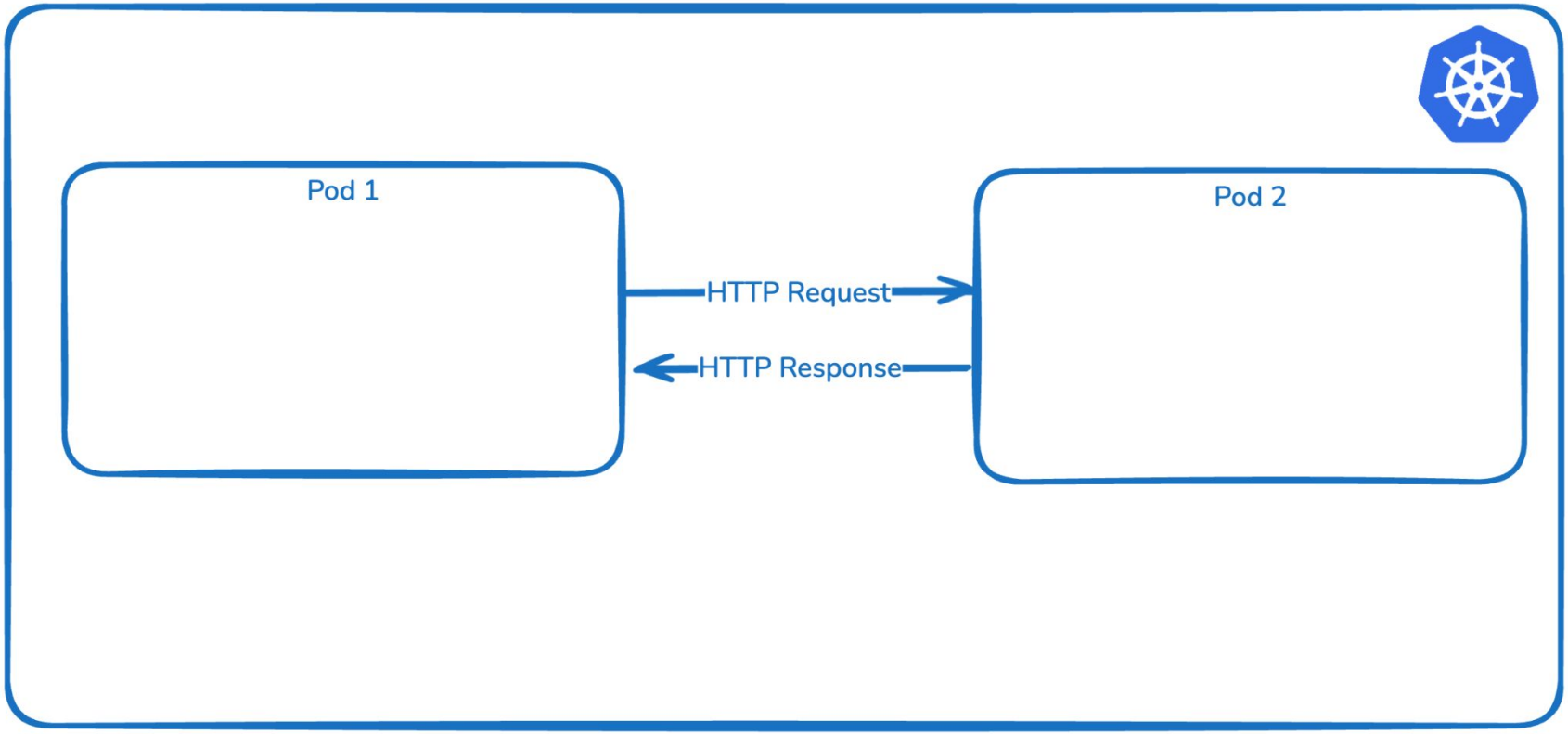
```
# Don't drink and YAML, folks!

jaeger:
  # We will use dash0 instead
  enabled: false

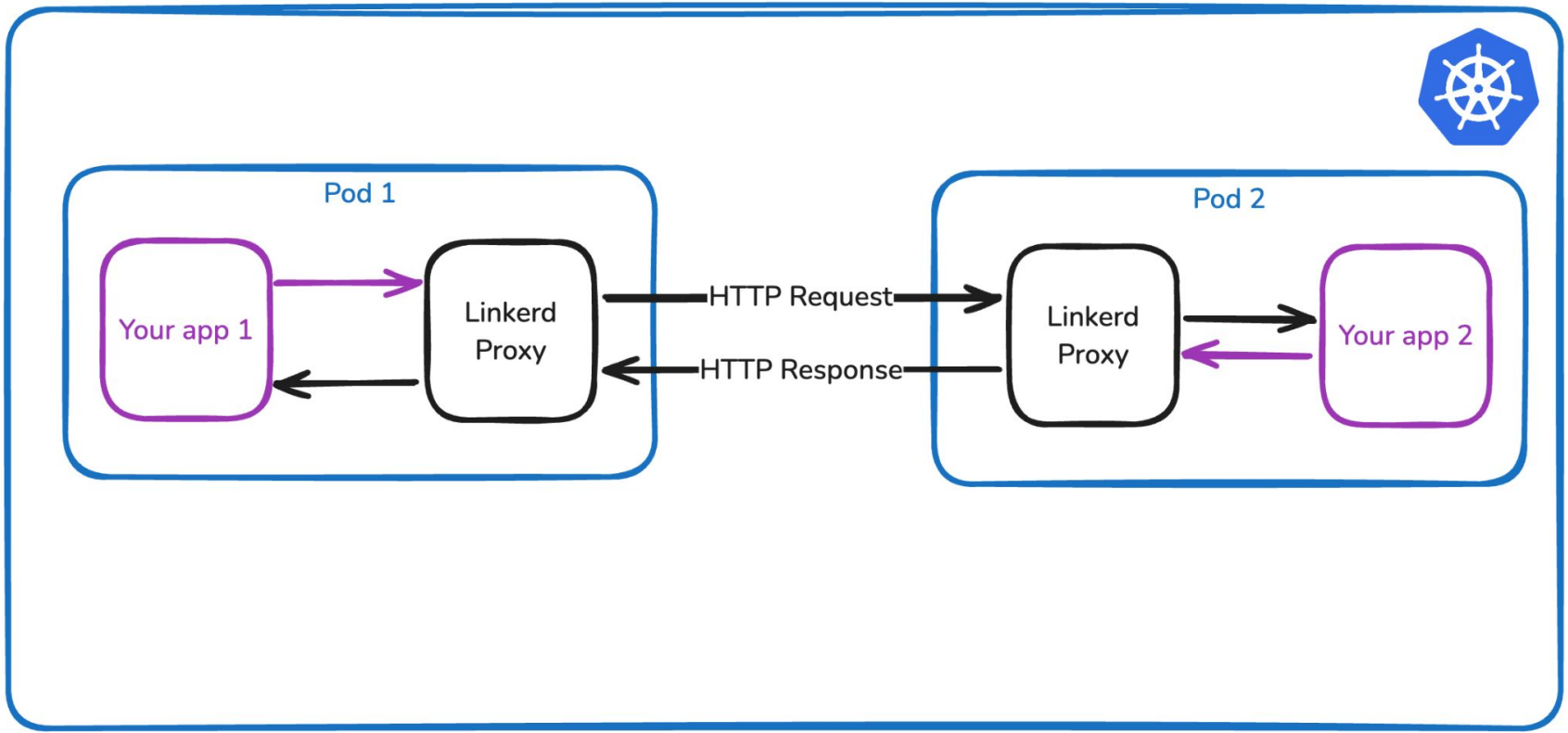
collector:
  # We will use the collector configured in the OTel Demo
  enabled: false

webhook:
  # Send tracing data to the collector configured
  # in the OTel Demo
  collectorSvcAddr: otel-demo-otelcol.otel-demo:4317
  collectorSvcAccount: otel-demo-otelcol
  collectorTraceProtocol: opentelemetry
```

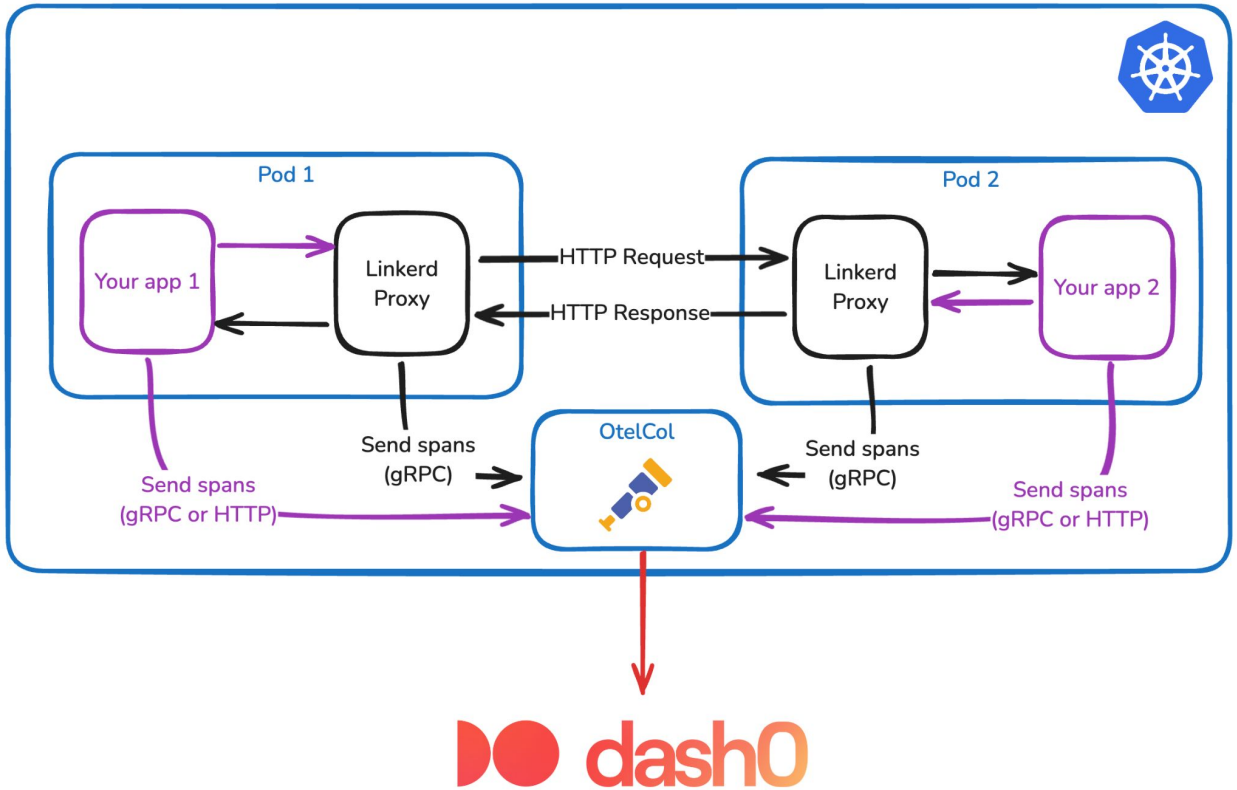
# Pods talking /1: If only the world was so simple…



# Pods talking /2: Linkerd joins the fray



# Pods talking /3: Now with distributed tracing!



# What's a span?

Spans are a bit like logs, except:

- Have a **duration** (start time and end time)
- Are sent to the backend only when they are **closed** (i.e., the end is recorded)
- Are part of a **trace**
- Can have a **parent**:
  - the span that happened "before it" in the trace
- Has a **name** and **attributes** that describe what the span represents

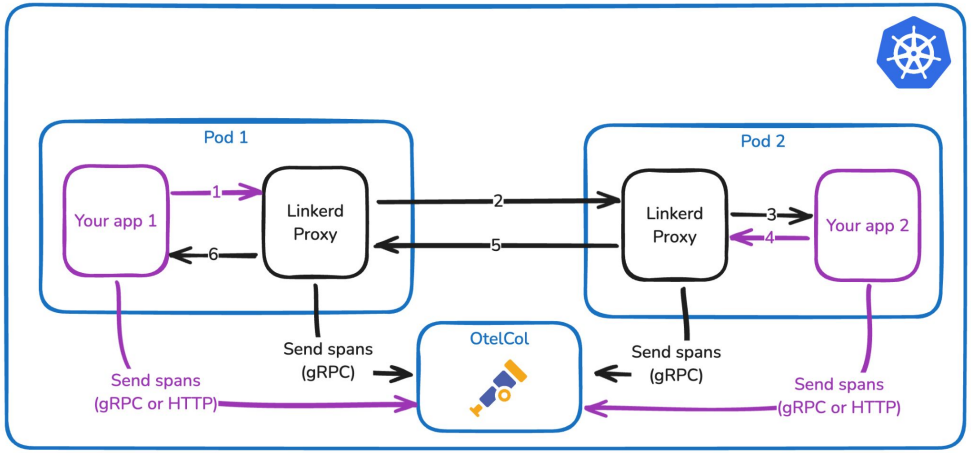
```
// [1] Create a new span
Span span = tracer.spanBuilder("jdbc query").startSpan();
try (Scope ignored = span.makeCurrent()) {
  String query = "select COF_NAME, SUP_ID, PRICE, SALES, TOTAL from COFFEE";
  // [2] Record which query is being executed
  span.recordException(t, Attributes.of(SemanticAttributes.DB_QUERY_TEXT, query));
  return stmt.executeQuery(query);
} catch (Throwable t) {
  // Usually you would handle checked exceptions and runtime exceptions separately
  // with different catch blocks. This example simplifies error handling due to how
  // checked expressions work in Java.
  // [3] Record that the query execution failed
  span.setStatus(StatusCode.ERROR);
  // [4] Record WHY the query execution failed
  span.recordException(t, Attributes.of(SemanticAttributes.EXCEPTION_ESCAPED, true));
  throw t;
} finally {
  // [5] Close the span and queue it for delivery to your distributed tracing system
  span.end();
}
```

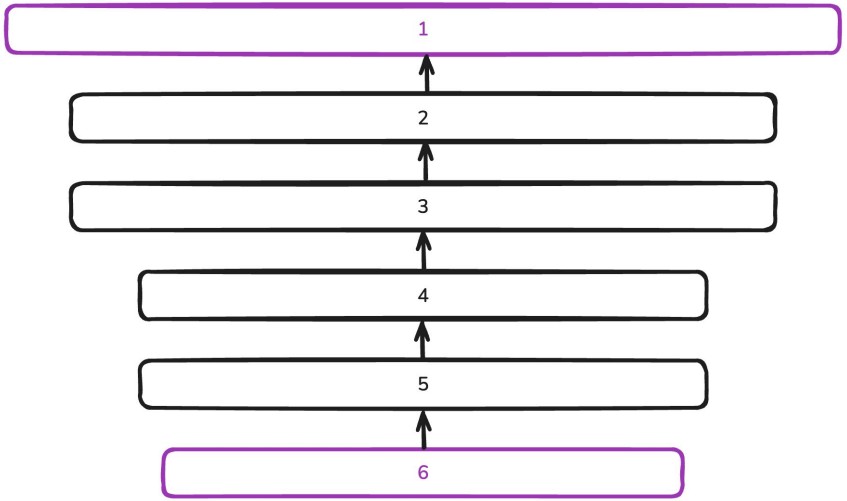
Want to know more? "What is distributed tracing" **dash0.com**

# Anatomy of an OpenTelemetry span

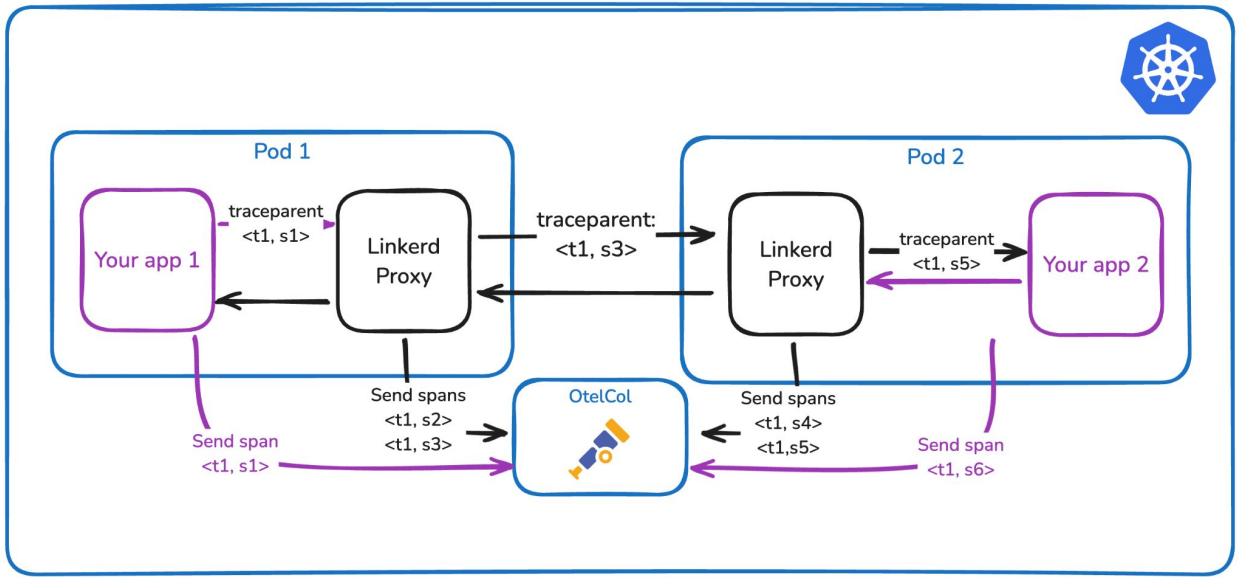
TraceId: 25a082c7fb6615209ed6d2d848bf0700
Id: a3192e5e9aa1c4fa
ParentId: 5748f1eb5659a186
Name: GET /images/products/:filename
Start: 2025-02-11 17:43:02.001
End: 2025-02-11 17:43:02.004
Attribute:
- Key: url.full
  Value: http://my-app/images
- Key: http.request.method
  Value: GET
- Key: http.response.status_code
  Value: 200

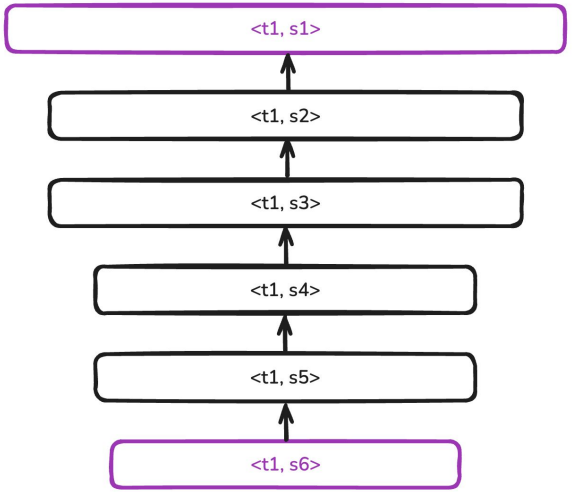
# The making of a trace



# Trace context propagation




Tracer context is propagated *forwards*

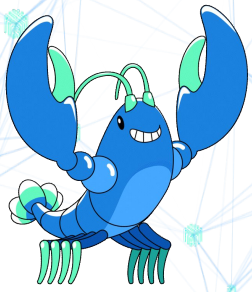
through HTTP headers, message metadata, etc.

Spans link *backwards* (child → parent)

BUOYANT
SERVICE MESH
ACADEMY

# DEMO

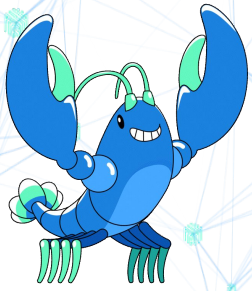
https://github.com/open-telemetry/opentelemetry-demo



BUOYANT
SERVICE MESH
ACADEMY

# DEMO

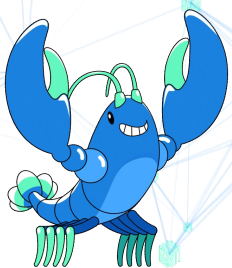




BUOYANT
SERVICE MESH
ACADEMY

# Part 5
# Gotchas

dash0



# Is Linkerd all you need to do distributed tracing with OpenTelemetry?

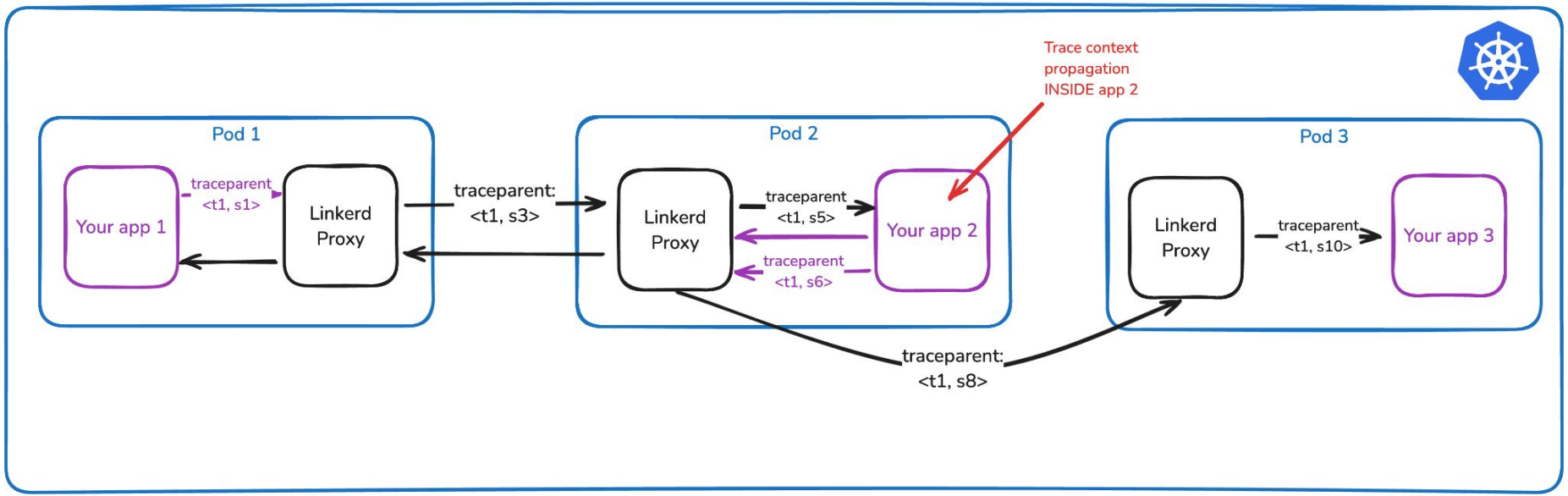
No, but it helps **massively**.

Your application still needs to:

1. create spans
2. update trace context as spans are created
3. propagate trace context from one end of the app to the other

If you don't add tracing to your application, you'll see traces between proxies, but they won't be connected in any way!

# Is Linkerd all you need to do distributed tracing with OpenTelemetry? (With pictures!)



# Distributed tracing is “eventually consistent”

Different **components** in your distributed trace will export spans at different **rates** and with different **delays**.

The structure of the trace is pre-defined when spans are created. Some spans may take a little longer than other to arrive.

Also: you generally never know when a trace is *over*. There is no such thing as **EOF** in distributed tracing. (Some traces involving messaging systems will be “alive” for longer than you expect!)

# Tell us how we can improve!

## Your feedback matters!

(We promise it won't take more than a few minutes, and it will help us tremendously — thank you! 🙂)



BUOYANT
Creators of LINKERD

# Buoyant Enterprise for LINKERD

**Rust-based network security and reliability for modern applications. Built on open source and designed for the enterprise.**

- ➔ Zero-trust security and compliance across your entire network
- ➔ Global traffic management and control
- ➔ Full L7 application observability
- ➔ Built for the enterprise

Learn more & try it for free at buoyant.io/**enterprise-linkerd**

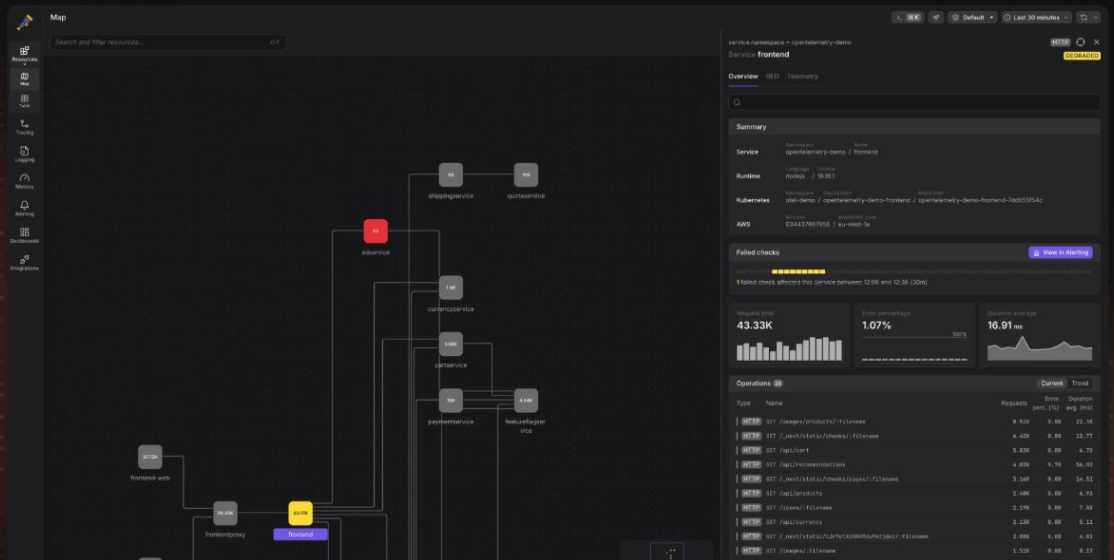
BUOYANT
Creators of LINKERD

# Observability, Simplified

## The Fastest Path to Clarity and Control

Email Address Start for free

Start for free or Book a demo





Updated Course!

BUOYANT
Creators of LINKERD

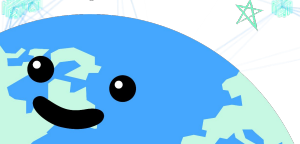


# Get Certified!
# Service Mesh 101, Online Course

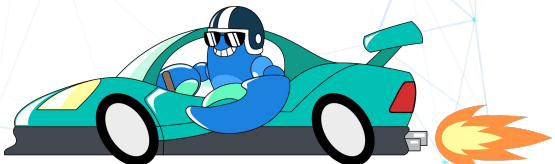
With hands-on LINKERD labs at learn.buoyant.io











SERVICE MESH ACADEMY

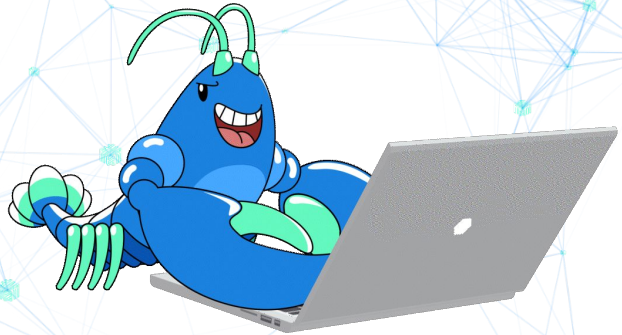


Up Next on March 13

# Gateway API 101

## with Gateway API maintainer Shane Utt

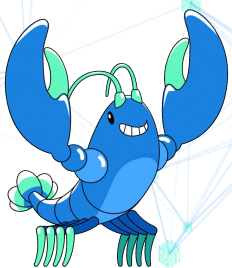
**SIGN UP TODAY!**
buoyant.io/sma



BUOYANT
SERVICE MESH
ACADEMY

# Q&A

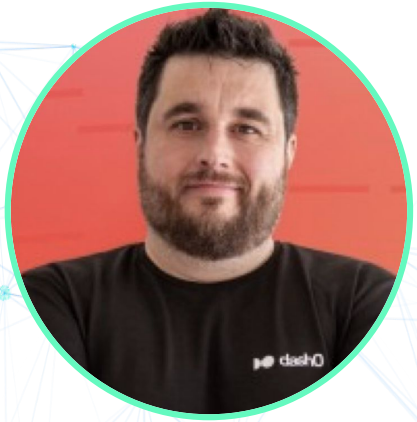
dash0



BUOYANT
SERVICE MESH
ACADEMY

# Thanks much!






flynn@buoyant.io
@flynn on slack.linkerd.io

michele.mancioppi@dash0.com
@??? on CNCF slack?