OpenTelemetry Tracing for Monoliths

Phillip Carter, Principal PM @ Honeycomb.io

Monoliths vs. Microservies

Often a meaningless distinction

A common “monolith”

- Most of the app is a single process
- Running behind a load balancer
- Talks to a database server (or two)
- Maybe talks to an auth server somewhere
- Might have an external vendor’s API somewhere
- …
- **It’s already a distributed system**

Microservices grow too!

- Just a little more code here and there…
- “This code is poorly organized, let’s modularize it”
- …
- Not uncommon to end up with *distributed monoliths*
- …
- And that’s okay!

You already need distributed tracing

- Tracing is how you correlate events happening in different parts of a system
  - How many database calls did this request make?
  - What's my average overall system latency per request? P95? P50?
  - Is that API we're calling the cause of a slowdown?
  - etc.
- Reign in problems at the *edges* of your systems with automatic instrumentation

Example: Intercom

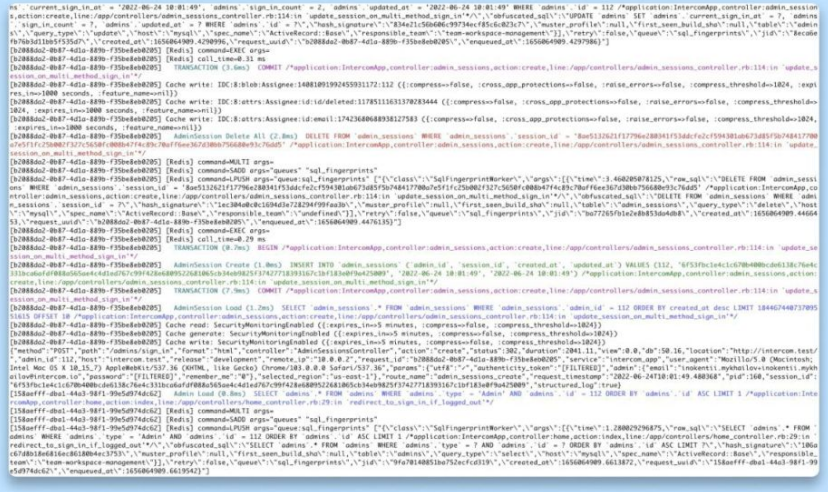
- Several services, but one big ‘ole monolith doing the bulk of work
- Added instrumentation for the edges of their systems
  - API calls
  - DB calls
  - Calling other services
- Iterated to add quality tracing instrumentation for their main monolithic service

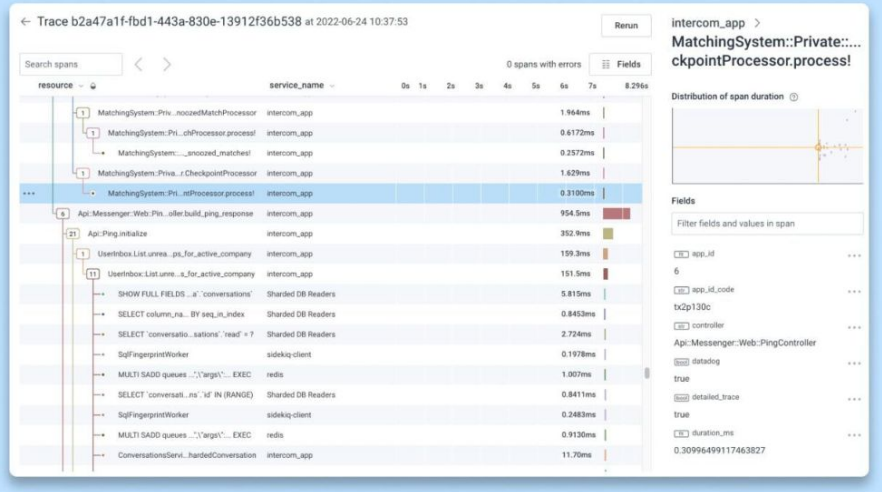
Tracing for a single service

Call it a monolith if you like!

Traces are better structured logs

- A trace is a collection of **spans** correlated by a **trace id** that encode an **order of operations** and **parent-child** hierarchy
- A span is just a structured log with some stuff on it
  - Name
  - Span ID
  - Parent Span ID
  - **Key-value pairs of data**
  - A list of *span events*
  - A list of *span links*





Credit: Intercom Engineering Blog

How to trace a single service

- Add autoinstrumentation / instrumentation libraries to track incoming/outgoing requests
- Use the **opentelemetry api** in your language to create more spans
- Change thinking from “log these things” to “track this operation with a span”
  - Start the span at the start of a function/method
  - Add relevant “log data” as attributes to the span
  - Use Span Events to capture “this thing happened at this time!”, like exceptions
  - End the span at the end of the function

Organizing spans in big services

- Big services organized into modules/bounded contexts/etc. can have that organization represented in telemetry
- Create a wrapper function/method for starting a span
- Calling the wrapper adds a name for your module/bounded context/etc.
  - Wrapper always adds a *bounded.context* attribute with a configured name
  - Bounded Context name can be passed in, configuration-driven, or defined by something else (e.g., assembly name for .NET apps)
- Now every span has the right organizational attribute on it to let you differentiate them!

Bring your logs along for the ride!

Don’t throw away your past work

OpenTelemetry Logs Bridge

- Set up OTel Logs in your SDK setup code
- Have a trace, such as from request autoinstrumentation
- And that's it!
  - Each log gets emitted as an OTel log
  - Each log record has the **span id** and **trace id** for the current active span/trace at the time that the log gets created
  - Automatic correlation between existing logs and a trace

Create new spans for new things

- Track operations with spans in code for new instrumentation
  - Remember: spans are like structured logs, but better
  - New instrumentation -> spans
  - Old instrumentation -> logs
- Keep your existing logs, or not, up to you!
  - Migrate to spans if you feel like it
  - Don’t migrate these logs if they work fine

Takeaways for your team

Traces are for monoliths!

- Distributed traces doesn’t mean you need tons of services
- You can add tracing to **one** service and get a lot of value out of it
- Traces are the superior, evolved form of structured logging
- You can bring your existing logs with you

Questions?

www.honeycomb.io