

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



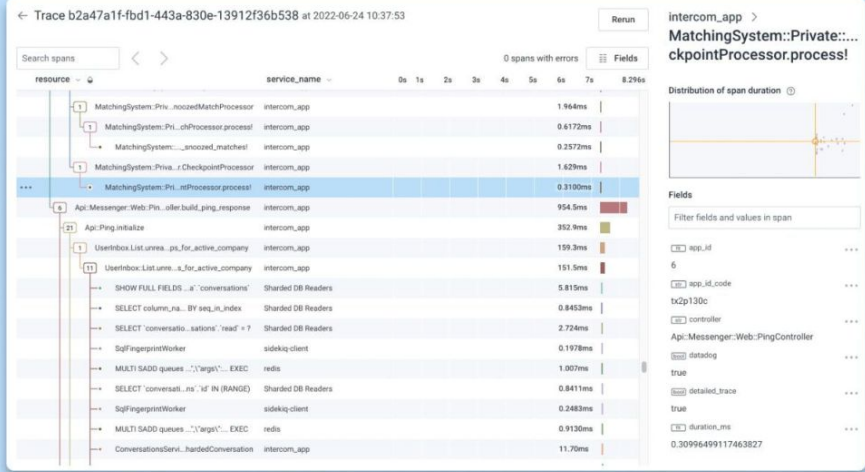
Tracing for a single service

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*





9

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!



13	GenerateQueryFromPrompt	poodle	8.149s						
	▪ launchdarkly.BoolVariation	poodle	40.6µs						
	▪ launchdarkly.NumberVariation	poodle	58.4µs						
	▪ launchdarkly.StringVariation	poodle	31.0µs						
	▪ launchdarkly.StringVariation	poodle	19.1µs						
	▪ Schema Store Get	poodle	23.9µs						
	▪ Schema Store Get	poodle	10.7µs						
20	queryml.FindAllSuggestedQueriesForDataset	poodle	106.7ms						
1	queryml.MostRelevantColumns	poodle	5.815s						
2	queryml.filterColumnsUsingEmbeddings	poodle	5.813s						
4	queryml.embeddings.Embeddings	poodle	188.7ms						
1	queryml.embeddings.getFromCache	poodle	0.8656ms						
	▪ queryml.embedd...shalEmbeddings	poodle	23.3µs						
	▪ TruncateColumnList	poodle	72.0µs						
	▪ openai.Embeddings	poodle	186.7ms						
	▪ queryml.embeddings.saveInCache	poodle	0.8410ms						
4	queryml.embeddings.Embeddings	poodle	5.612s						
1	queryml.embeddings.getFromCache	poodle	727.2ms						
	▪ queryml.embedd...shalEmbeddings	poodle	608.2ms						
	▪ TruncateColumnList	poodle	11.60ms						
	▪ openai.Embeddings	poodle	4.782s						
	▪ queryml.embeddings.saveInCache	poodle	84.99ms						
	▪ CreateChatPrompt	poodle	24.47ms						
	▪ CreateCustomExamplesPromptText	poodle	0.1182ms						
	▪ TruncateColumnList	poodle	0.7687ms						
1	queryml.GenerateQuery	poodle	2.193s						
	▪ openai.ChatCompletion	poodle	2.193s						
	render.JSON	poodle	44.1µs						

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