AGENDA

- WARM UP | 10 min
- INTRO DT & OT | 30 min
- Q&A | 10 min
- BREAK | 10 min
- HANDS-ON | 90 min
- Q&A | 30 min
WHY?

We replaced our monolith with micro services so that every outage could be more like a murder mystery.

1:10 AM - Oct 8, 2015

2,546 3,054 people are talking about this
MODERN MICROSERVICES ARCHITECTURES

Amazon internal service dependency visualization
WHAT IS DISTRIBUTED TRACING?

- A trace tells the **story of a transaction or workflow as it propagates** through a (potentially distributed) system.

- It’s basically a directed acyclic graph (DAG), with a **clear start** and a **clear end** - no loops.

- Focused on **business operations instead of services**

- Advantages:
  - It provides useful insight into the behavior of the system including the sources of **latency** and **errors** (think troubleshooting or performance analysis)
  - It enables us to see how an individual business operation is handled by other **applications that we don't know** (yet)
A trace tells the **story of a transaction or workflow as it propagates** through a (potentially distributed) system.

It’s basically a directed acyclic graph (DAG), with a **clear start** and a **clear end** - no loops.

OpenTelemetry is made up of an integrated set of APIs and libraries as well as a collection mechanism via an agent and collector. It also does **Distributed Tracing**.
WHAT IS OPENTRACING?

Vendor-neutral APIs and code instrumentation standard for distributed tracing
Span: a named operation which records the duration, usually a remote procedure call, with optional Tags and Logs.
**Tag:** A "mostly" arbitrary Key:Value pair (value can be a string, number or bool)

- **error:** true
- **span.kind:** client
- **region:** eu-central-1
- **account_id:** 123456789
**OPENTRACING CONCEPTS - SPAN LOGS**

**Log:** A "mostly" arbitrary Key:Value pair (like Tag) but of any type and with an accompanying timestamp (Structured Logging)

1527250021100 event: timed out
1527250021100 message: someone set us up the bomb
1527250021100 stack: <platform-stacktrace-here>
**Span Context**: Used to propagate the context across process boundaries. A vendor dependent element that, usually, contains a **traceId**, **spanId** and **baggage**.
**Baggage Items**: An arbitrary Key:Value pair that crosses process boundaries. Every process will copy Baggage items when continuing a trace.

---

**Context Propagation**

```plaintext
ot-baggage-environment: production
ot-baggage-steering-ids: foo=bar
```
Trace: A collection of Spans. Formally, a directed acyclic graph (DAG) of Spans, where the edges between Spans are called References.
ChildOf: A Span may be the ChildOf a parent Span. In a ChildOf reference, the parent Span depends on the child Span in some capacity.

- A Span representing the server side of an RPC may be the ChildOf a Span representing the client side of that RPC
- A Span representing a SQL insert may be the ChildOf a Span representing an ORM save method
- Many Spans doing concurrent (perhaps distributed) work may all individually be the ChildOf a single parent Span that merges the results for all children that return within a deadline
FollowsFrom: Some parent Spans do not depend in any way on the result of their child Spans. In these cases, we say merely that the child Span followsFrom the parent Span in a causal sense.

- Service A sends a request to Service B; Service B responds with an ACK. In the meantime, it will work on the request it got.
- Event based systems are also good candidates. But beware of batch processing of events.
SEMANTIC CONVENTIONS

- The OpenTracing community defined some semantic conventions to guide the usage of Tags and Logs, helping create a standardized approach to them.
  - Tags: `http.status_code`, `error`, `peer.service`, `span.kind`
  - Logs: `error.kind`, `event`, `message`
- It also describes how to combine some of the Tags and Logs for a richer context of the Spans
  - Adding context to RPCs or Database Client Calls
- Consider having specific Semantic Conventions in your own organization
  - Provide extra guidance in the use of OpenTracing
  - Ensure a standardized experience across different teams and services
1. Always try to **extract** the span context from the inbound request (if any)
2. **Start** a new span (reference to span ctx when present)
3. Add tags (optional)
4. Add logs (optional)
5. **Inject** the span context on remote procedure calls
6. **Finish** the span
The key to building value is to balance completing some well-articulated high value traces with the notion of total code coverage.
OPENTRACING - CRAWL, WALK, RUN

- **Crawl**: Visualize your instrumentation as early as possible. This will help you identify areas that need further visibility.

- **Walk**: Once you have an end-to-end trace, you can evaluate and prioritize areas where greater visibility will be worth the level of effort for the additional instrumentation.

- **Run**: Look for the units of work that can be reused. An example of this would be instrumenting a library that is used across multiple services.
AGENDA

WARM UP | 10 min
INTRO DT & OT | 30 min
Q&A | 10 min
BREAK | 10 min
HANDS-ON | 90 min
Q&A | 30 min
AGENDA

WARM UP | 10 min
INTRO DT & OT | 30 min
Q&A

BREAK | 10 min
HANDS-ON | 90 min
Q&A | 30 min
AGENDA

- WARM UP | 10 min
- INTRO DT & OT | 30 min
- Q&A | 10 min
- BREAK | 10 min
- HANDS-ON | 90 min
- Q&A | 30 min
HANDS-ON EXERCISE

Repository:
https://github.com/sarslanhan/sre-con-19-effective-distributed-tracing-workshop
HANDS-ON EXERCISE

Website

Cart API

Stock API

add item to cart

check stock

Send spans

Fault injection

JAEGGER
AGENDA

WARM UP | 10 min
INTRO DT & OT | 30 min
Q&A | 10 min
BREAK | 10 min
HANDS-ON | 90 min
Q&A | 30 min