redhat

WHAT ARE MY MICROSERVICES DOING?

OpenSource Summit EU 2018

2018-10-24

JURACI PAIXÃO KRÖHLING

Software Engineer

Kiali Project, Distributed Tracing team

Speaker notes

Talk about Red Hat's Microservices Survey, where "monitoring/diagnostics" is among the top concerns. Observability helps there!

MICROSERVICES

THEY ARE JUST DISTRIBUTED SYSTEMS, BUT IN THE HUNDREDS (THOUSANDS!)

DISTRIBUTED SYSTEMS FAIL. ALL THE TIME.

MULTIPLE VERSIONS PER SERVICE

A/B TESTS, CANARY RELEASES, ROLLING DEPLOYMENT, ...

MICROSERVICES

CHAOS

MICROSERVICES

CHAOS

YAY! AS LONG AS IT'S OBSERVABLE.

WITH CHAOS, WE GET RESILIENCY.

OBSERVABILITY

WHAT WENT WRONG, WHERE AND WHY

METRICS, LOGS, TRACING

DISTRIBUTED SYSTEMS AS CONTEXT

METRICS AND LOGS FOR INDIVIDUAL INSTANCES ARE NOT ENOUGH,

TRACES SHOULD REFLECT WHAT HAPPENED IN THE WHOLE TRANSACTION

OBSERVABILITY

by @peterbourgon

source: https://peter.bourgon.org/blog/2017/02/21/metrics-tracing-and-logging.html

Speaker notes

People often ask whether they still need logging when they have tracing, and whether they need tracing when they have logs already. Some are also not clear where "metrics" (or this Prometheus-thing) fit.

This image can help clarify that.

DISTRIBUTED TRACING

STORY OF A REQUEST ACROSS SERVICES

WHICH SERVICES WERE TOUCHED, WHEN, IN WHICH ORDER, WITH INSTANCE INFORMATION (ROUTING INFO, VERSION, TAGS, ...)

Speaker notes

Distributed Tracing tells the story of a request across the microservices.

With instrumentation via OS/Middleware/Platform, we can tell details about the request that aren't immediately apparent in logs, like the routing information from Istio.

DISTRIBUTED TRACING

ROOT CAUSE ANALYSIS

EASY TO SPOT WHERE THE FIRST FAILURE HAPPENED, HOPEFULLY WITH A "WHY" (ERROR FLAG + MESSAGE IN THE SPAN)

PERFORMANCE OPTIMIZATION

IS THERE A SERVICE BEHAVING BADLY?

IF I IMPROVE *THIS* SERVICE, HOW BIG WILL BE THE IMPACT?

DISTRIBUTED TRACING

MEASURES UNITS OF WORK

STORES IN A DATA STRUCTURE CALLED "SPAN"

REFERENCES OTHER SPANS

CAUSALITY!

CONTEXT PROPAGATION

MOST OF IT IS SIMILAR TO CORRELATION ID

Speaker notes

Distributed tracing isn't about "messages", but about "spans". Spans have a start and end time, so, they are "blocks", instead of events.

DISTRIBUTED TRACING

OUR CODE

```
chargeCreditCard();
changeOrderStatus();
dispatchEventToInventory();
```

DISTRIBUTED TRACING

OUR CODE WITH DISTRIBUTED TRACING

```
try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {

    chargeCreditCard();
    changeOrderStatus();
    dispatchEventToInventory();
}
```

Speaker notes

It's not that intrusive, is it? Logging would be as intrusive as tracing...

DISTRIBUTED TRACING

OUR CODE WITH DISTRIBUTED TRACING

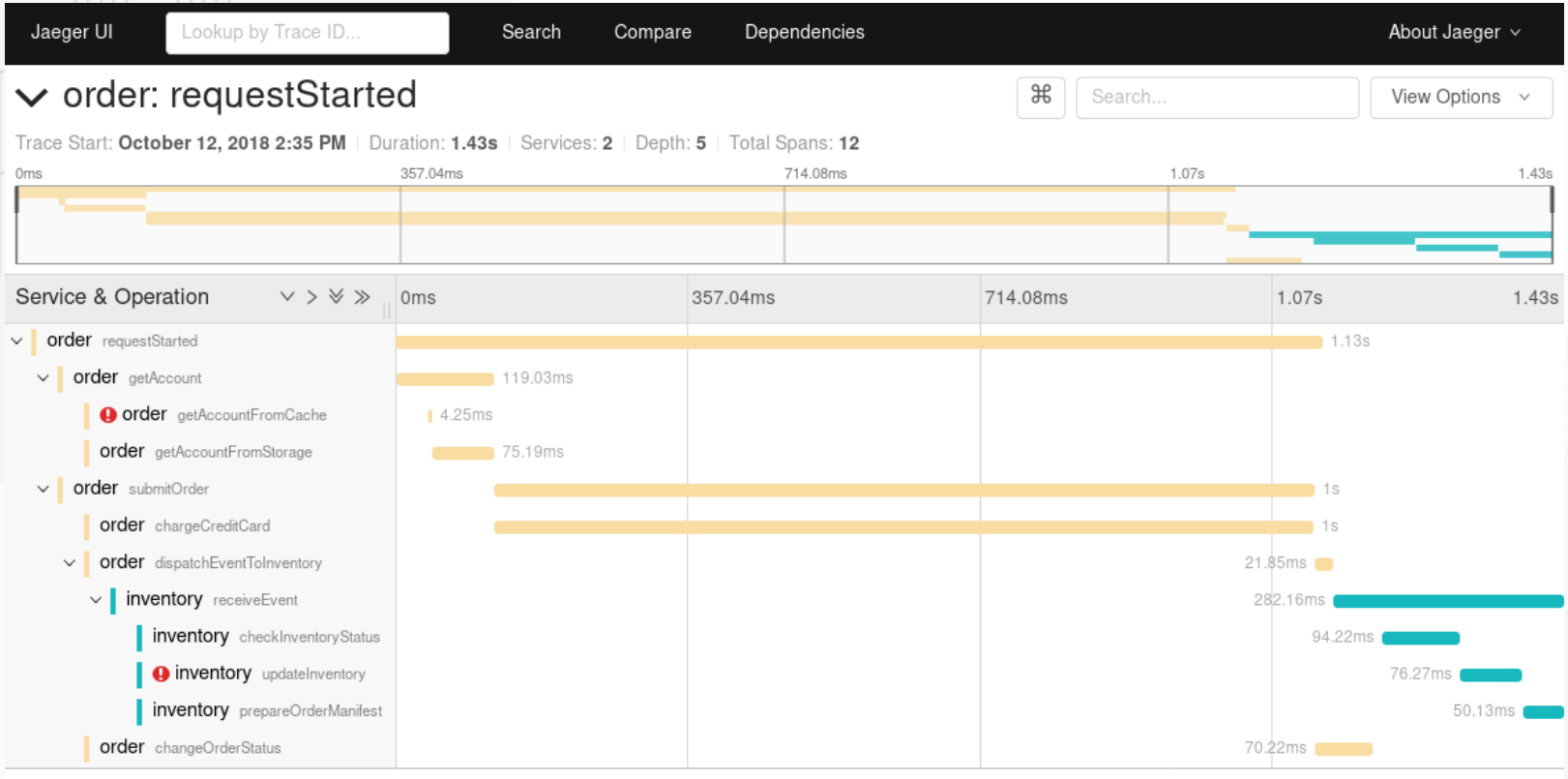
AND CUSTOM TAG, WITH BUSINESS-SPECIFIC INFORMATION

```
try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {
    scope.span().setTag("order-id", "c85b7644b6b5");
    chargeCreditCard();
    changeOrderStatus();
    dispatchEventToInventory();
}
```

Speaker notes

We can add business-related information to the span!

DISTRIBUTED TRACING

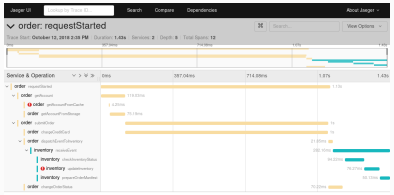


Trace as seen on Jaeger

Speaker notes

When we have properly instrumented services, this is what we get

DISTRIBUTED TRACING



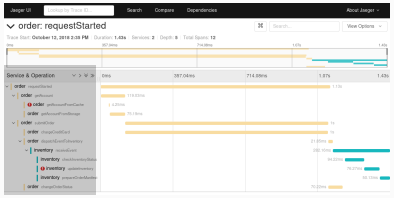
order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**

Speaker notes

Trace-level properties (computed based on the spans)

DISTRIBUTED TRACING

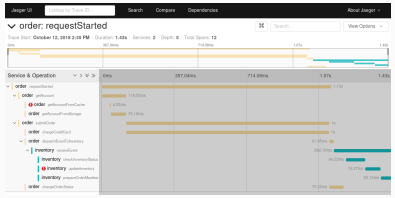


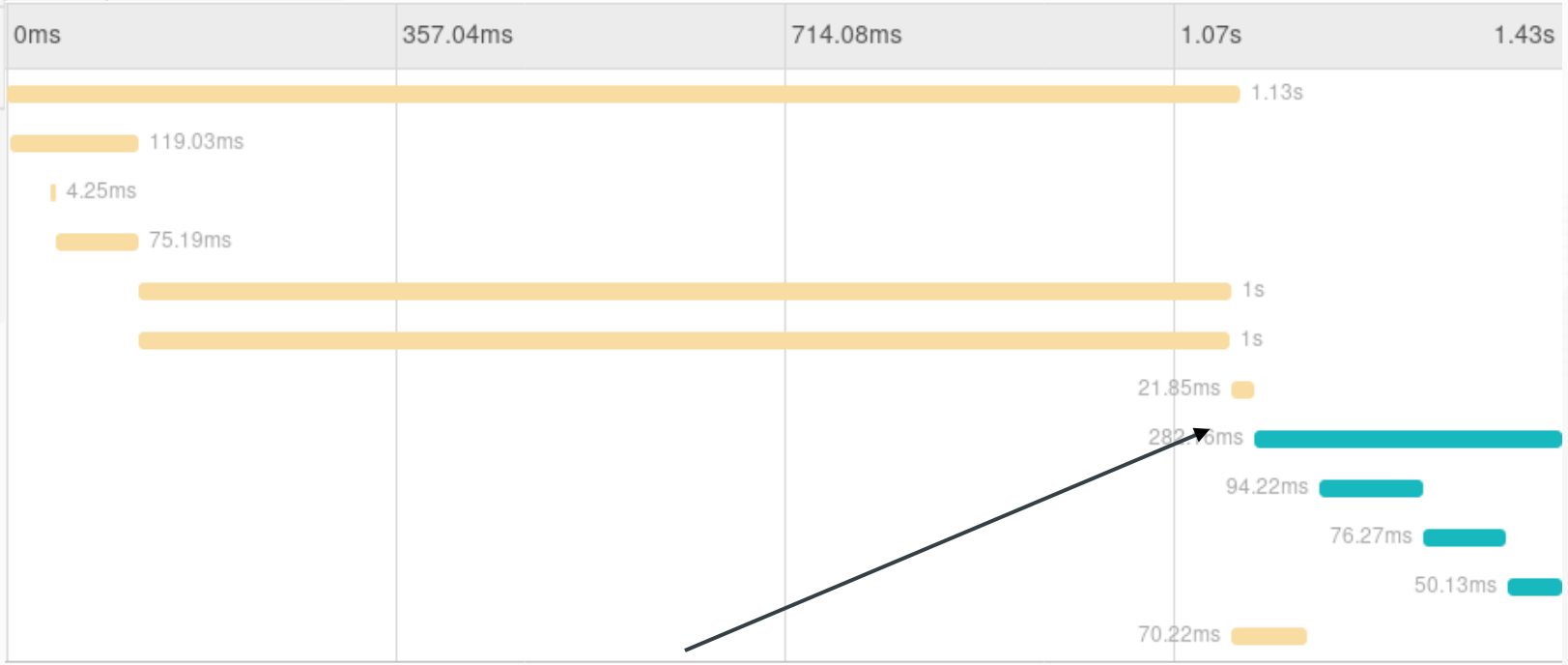
Service & Operation

- order requestStarted
  - order getAccount
    - order getAccountFromCache
    - order getAccountFromStorage
  - order submitOrder
    - order chargeCreditCard
    - order dispatchEventToInventory
      - inventory receiveEvent
        - inventory checkInventoryStatus
        - inventory updateInventory
        - inventory prepareOrderManifest
    - order changeOrderStatus

Our operation!

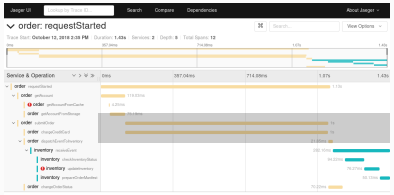
DISTRIBUTED TRACING

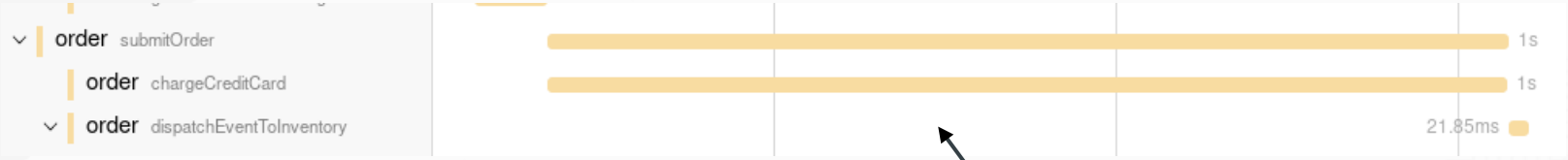




Process (service) boundary

DISTRIBUTED TRACING





Possible performance bottleneck

INSTRUMENTATION

EXPLICIT

IN MY CODE

IMPLICIT

NOT IN MY CODE

OPENTRACING

SEMANTICS FOR DISTRIBUTED TRACING

WHAT'S A TRACE, WHAT'S A SPAN, ...

INSTRUMENTATION API

OFFICIAL APIS FOR GO, JAVASCRIPT, JAVA, PYTHON, RUBY, PHP, OBJECTIVE-C, C++, C#

CAN BE USED WITH COMPATIBLE TRACERS, LIKE ZIPKIN, JAEGER, ...

HOSTED AT THE CNCF

CONTRIBUTORS WITH DIVERSE BACKGROUNDS

Speaker notes

OpenTracing is one API aiming to be a standard, so that applications written using Python have the same semantics for distributed tracing as Java applications.

It's also "vendor neutral", in the sense that applications can just instrument their applications without caring about which concrete tracer to use. This can be decided later.

The project has contributors with and without vendor affiliations.

OPENTRACING

FRAMEWORK, STACK, PLATFORM

JAX-RS, JDBC, SERVLET, ...

SPRING BOOT, MICROPROFILE, BYTECODE MANIPULATION, ...

AND MORE

GITHUB.COM/OPENTRACING-CONTRIB

Speaker notes

Infra -- routing can be done based on baggage items

JAEGER

CONCRETE TRACER IMPLEMENTATION

NATIVE OPENTRACING SEMANTICS

C++, C#, GO, JAVA, NODEJS, PYTHON, RUBY, ...

BACKEND COMPONENTS

AGENT, COLLECTOR, QUERY, UI

PRODUCTION-READY

BARE METAL, OPENSHIFT, KUBERNETES

Speaker notes

On the client-side, Jaeger implements the OpenTracing API.

It also provides the "missing pieces", like the client component that actually captures the spans, sends to an agent/collector, and so on.

It's battle tested in production by tracing multi-thousand architectures at unicorn startups (like Uber).

DEMO

THREE MICROSERVICES

BARE METAL

INSTRUMENTATION: FRAMEWORK

Speaker notes

Istio Tutorial running on localhost

Q&A

Speaker notes

Prepare for the "performance"/"overhead" question :-)

PERFORMANCE

NOTHING COMES FOR FREE

DEFINE BEFORE HAND WHAT'S AN ACCEPTABLE COMPROMISE

INSTRUMENTATION OVERHEAD

INSTRUMENTATION NEEDS CPU (BUT SHOULD BE LOW)

TRACER OVERHEAD

TRACER NEEDS MEMORY + NETWORK (MIGHT BE HIGH)

Speaker notes

Instrumentation knows "what" to measure (HTTP request, for instance), so, it needs to just read data from the context (HttpServletRequest) and put in the span. It's mostly CPU bound.

The tracer is the piece that actually builds and stores the span. It needs a place to store the spans in memory until they are finally dispatched to somewhere.

PERFORMANCE

USE SAMPLING STRATEGIES

FOR BUSY SERVICES, USE SAMPLING TO REDUCE THE OVERHEAD

MEASURE, MEASURE, MEASURE

LIKE EVERYTHING RELATED TO PERFORMANCE...

Speaker notes

Sampling is the technique of selecting which business transactions to measure. One simple way of doing that is randomly selecting one business transaction for every thousand transactions.

If you are concerned about performance, ship your service with a `NoopTracer` and with a concrete tracer and use middleware monitoring/metrics to compare the services.

Prefer to use the Jaeger Agent whenever possible. This way, the pressure on the client residing within your application is reduced, as it then sends the spans via UDP to the agent, close to a "fire and forget" mechanism. Spans then stays very short in memory and the TCP overhead is avoided. Usually, the agent is on `localhost`, so, the downsides of UDP are minimal.