WHAT ARE MY MICROSERVICES DOING?

OpenSource Summit EU 2018

2018-10-24

JURACI PAIXÃO KRÖHLING

Software Engineer

Kiali Project, Distributed Tracing team
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.

twitter.com/jpkrohling
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
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, ...)

twitter.com/jpkrohling
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?

twitter.com/jpkrohling
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
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

```java
try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {
    chargeCreditCard();
    changeOrderStatus();
    dispatchEventToInventory();
}
```
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

```java
try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {
    scope.span().setTag("orderid", "c85b7644b6b5");
    chargeCreditCard();
    changeOrderStatus();
    dispatchEventToInventory();
}
```
We can add business-related information to the span!
DISTRIBUTED TRACING

Trace as seen on Jaeger
When we have properly instrumented services, this is what we get
Trace-level properties (computed based on the spans)
DISTRIBUTED TRACING

Our operation!
DISTRIBUTED TRACING

<table>
<thead>
<tr>
<th>Time</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0ms</td>
<td>119.03ms</td>
</tr>
<tr>
<td></td>
<td>119.03ms</td>
</tr>
<tr>
<td>357.04ms</td>
<td>4.25ms</td>
</tr>
<tr>
<td>714.08ms</td>
<td>75.19ms</td>
</tr>
<tr>
<td>1.07s</td>
<td>1.13s</td>
</tr>
<tr>
<td>1.43s</td>
<td>1s</td>
</tr>
<tr>
<td>1s</td>
<td>21.85ms</td>
</tr>
<tr>
<td>1s</td>
<td>94.22ms</td>
</tr>
<tr>
<td>262.6ms</td>
<td>76.27ms</td>
</tr>
<tr>
<td></td>
<td>50.13ms</td>
</tr>
<tr>
<td>70.22ms</td>
<td>50.13ms</td>
</tr>
</tbody>
</table>

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, JAEGGER, ...

HOSTED AT THE CNCF

CONTRIBUTORS WITH DIVERSE BACKGROUNDS

twitter.com/jpkrohling
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

twitter.com/jpkrohling
Infra -- routing can be done based on baggage items
JAEGGER

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

twitter.com/jpkrohling
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
Istio Tutorial running on localhost
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)

twitter.com/jpkrohling
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_EVERYTHINGRELATED_TO_PERFORMANCE...
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.