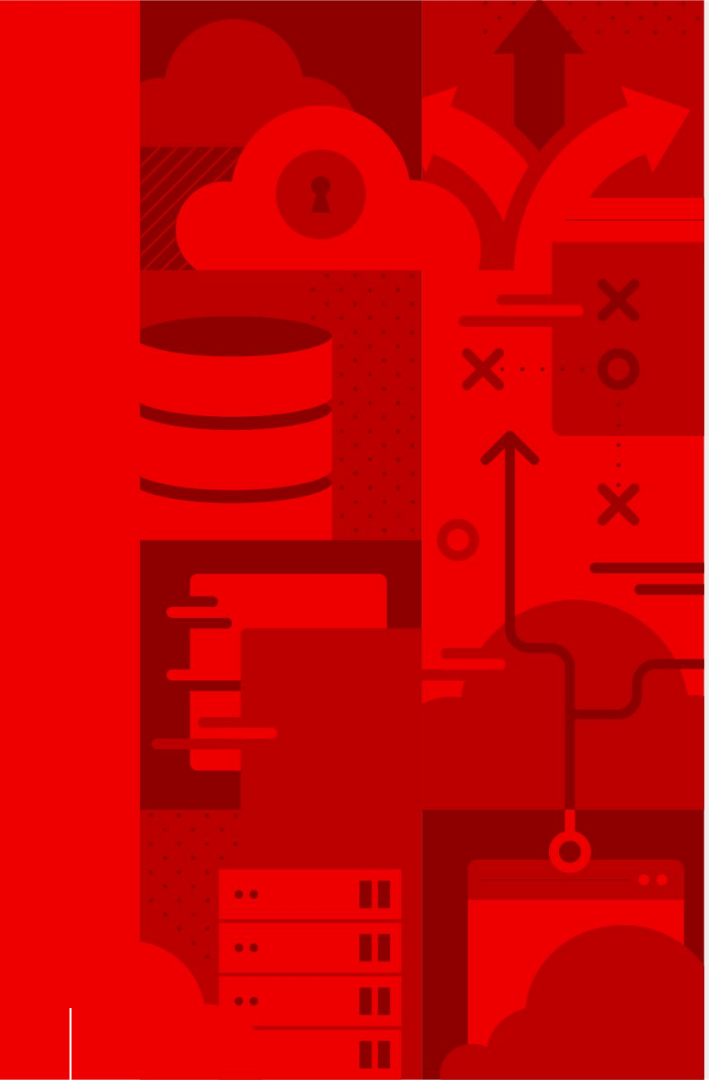


Observing chaos

How distributed tracing brings observability to a service mess

Juraci Paixão Kröhling
Software Engineer
@jpkrohling

Red Hat

Microservices

They are just distributed systems. But in the hundreds.

Distributed systems fail. All the time!

Each service at a potentially different version.

A/B tests, canary releases, ...

Microservices

Chaos.

Microservices

Chaos. YAY!

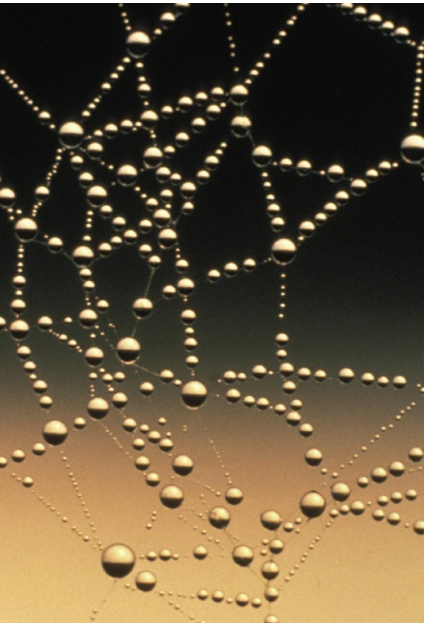
(as long as we can observe it…)

Observability

What went wrong, where, and why

Metrics, logs, distributed tracing, …

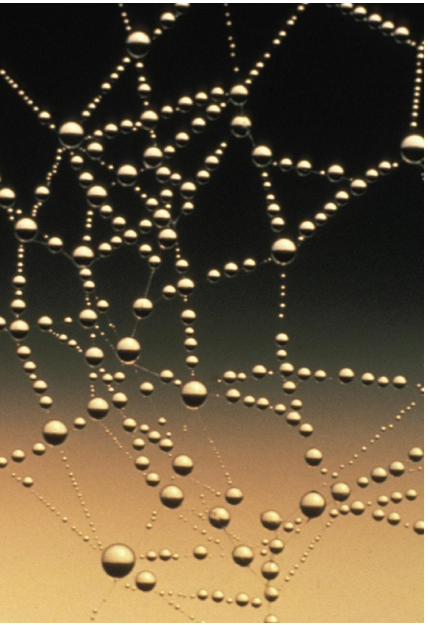
Distributed systems as context



Distributed Tracing

Tells the story of a request across services

Which services were touched, when, and in which order



Distributed Tracing

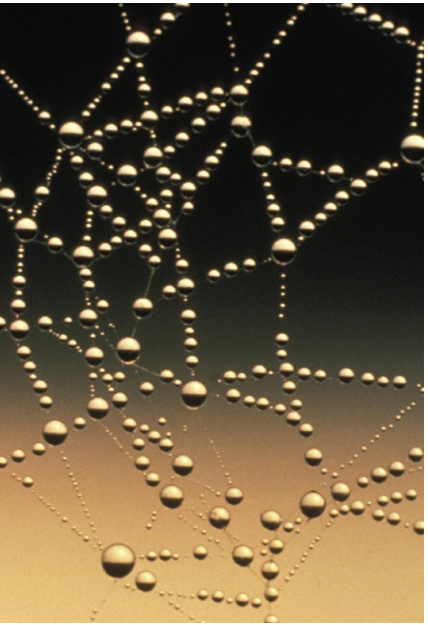
Measures units of work (“spans”)

- When did it start, how long it took, …

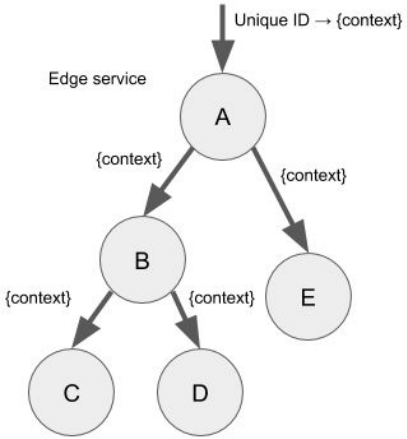
Causality

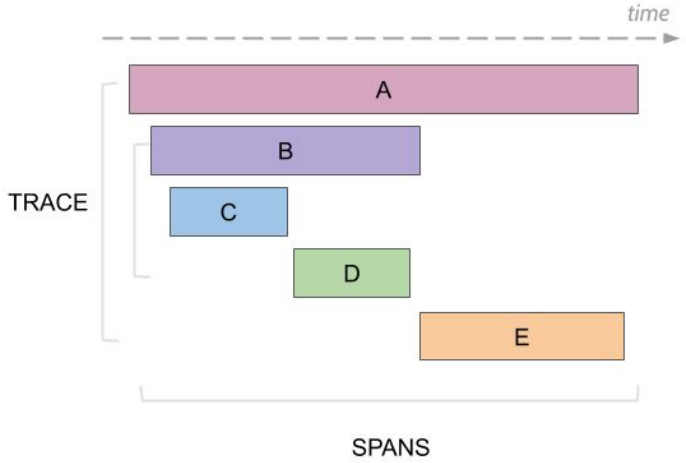
- Spans may contain references to other spans

Context propagation

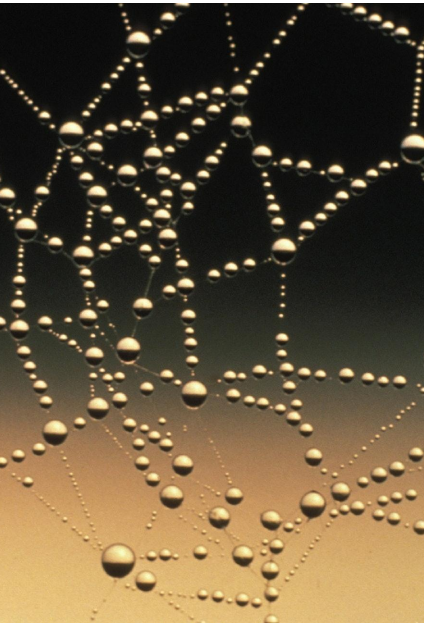


Distributed Tracing





Source: https://www.jaegertracing.io/docs/1.14/architecture/



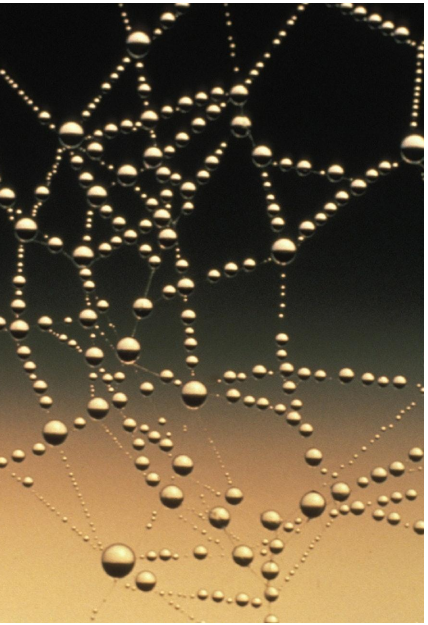
Distributed Tracing

Our code

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String submit() {

        Account account = accountsService.getAccount();
        Order order = new Order(UUID.randomUUID().toString(), account);
        inventoryService.processOrder(order);
        return String.format("Order submitted: %s", order);

}
```

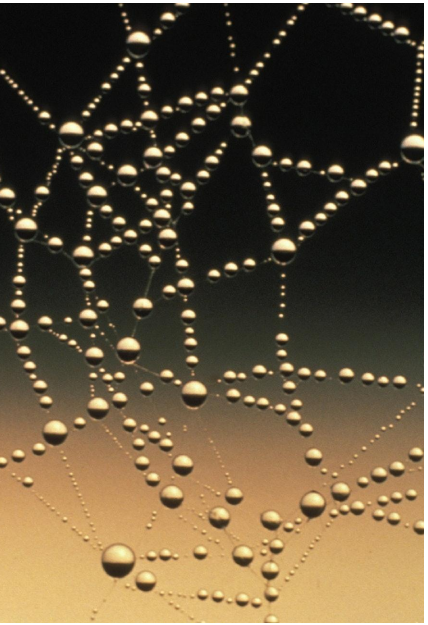


Distributed Tracing

Our code with distributed tracing

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String submit() {
    try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {

        Account account = accountsService.getAccount();
        Order order = new Order(UUID.randomUUID().toString(), account);
        inventoryService.processOrder(order);
        return String.format("Order submitted: %s", order);
    }
}
```



Distributed Tracing

Our code with distributed tracing and custom tags

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String submit() {
    try (Scope scope = tracer.buildSpan("submitOrder").startActive(true)) {
        scope.span().setTag("priority", "high");
        Account account = accountsService.getAccount();
        Order order = new Order(UUID.randomUUID().toString(), account);
        inventoryService.processOrder(order);
        return String.format("Order submitted: %s", order);
    }
}
```

Jaeger

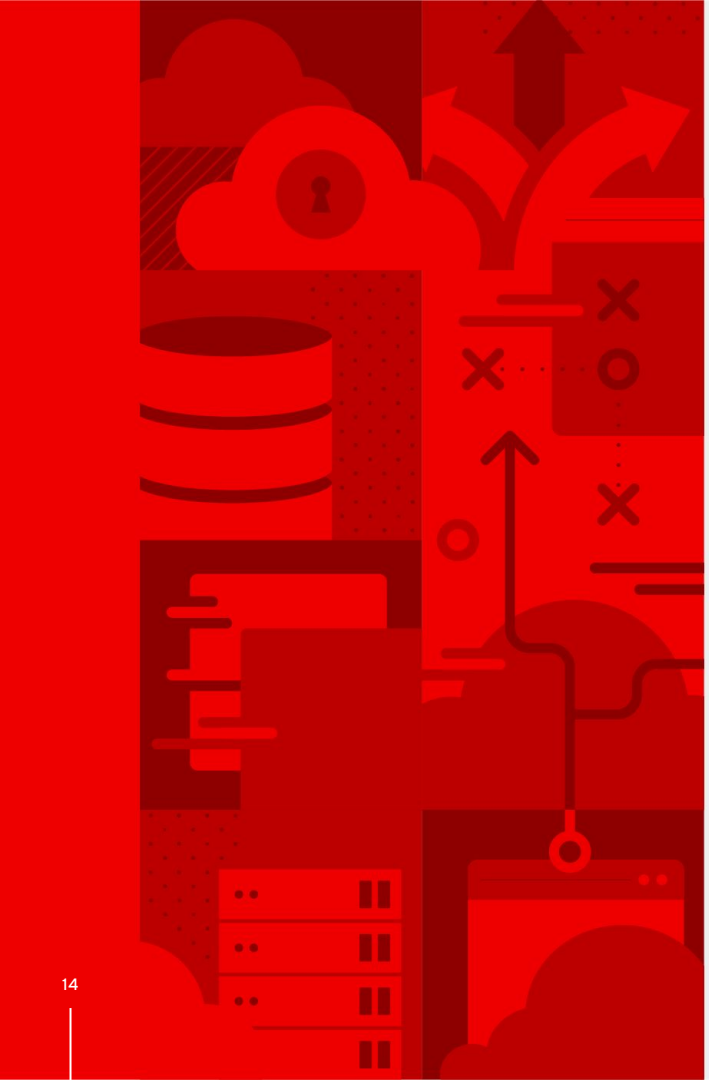
Complete tracing solution

- Client libraries (aka Tracers)
- Backend components (Collector, Ingester, Query)
- User interface

Jaeger

Available as part of OpenShift Service Mesh

- Jaeger Operator
- Uses Elasticsearch (via Elasticsearch Operator)
- Seamless integration with Istio
- Deep integration with Kiali



Thank you

Contact info:

jpkroehling@redhat.com

@jpkroehling

Recommended resources:

Jaeger, Istio, Kiali, Istio by example

Photos from Pixabay and Pexels:

Microservices, Observability, Distributed Tracing

Red Hat