Chasing Down Performance Issues Using Distributed Tracing

Richard Seroter
SENIOR DIRECTOR OF PRODUCT, PIVOTAL

@rseroter

Overview

The role of tracing in microservices

Problems with the status quo

What is Spring Cloud Sleuth?

Anatomy of a trace

What is automatically instrumented?

Adding Spring Cloud Sleuth to a project

Visualizing latency with Zipkin

Adding Zipkin to a solution

Working with samplers

Manually creating spans

Summary

The Role of Tracing in Microservices

Locate misbehaving components

Observe end-to-end latency

Understand actual, not specified, behavior

Problems with the Status Quo

Instrumenting all communication paths

Collecting logs across components, threads

Correlating and querying logs

Seeing the bigger picture / graph

Spring Cloud Sleuth

Automatic instrumentation of communication channels.

Glossary of Spring Cloud Sleuth Terms

Span

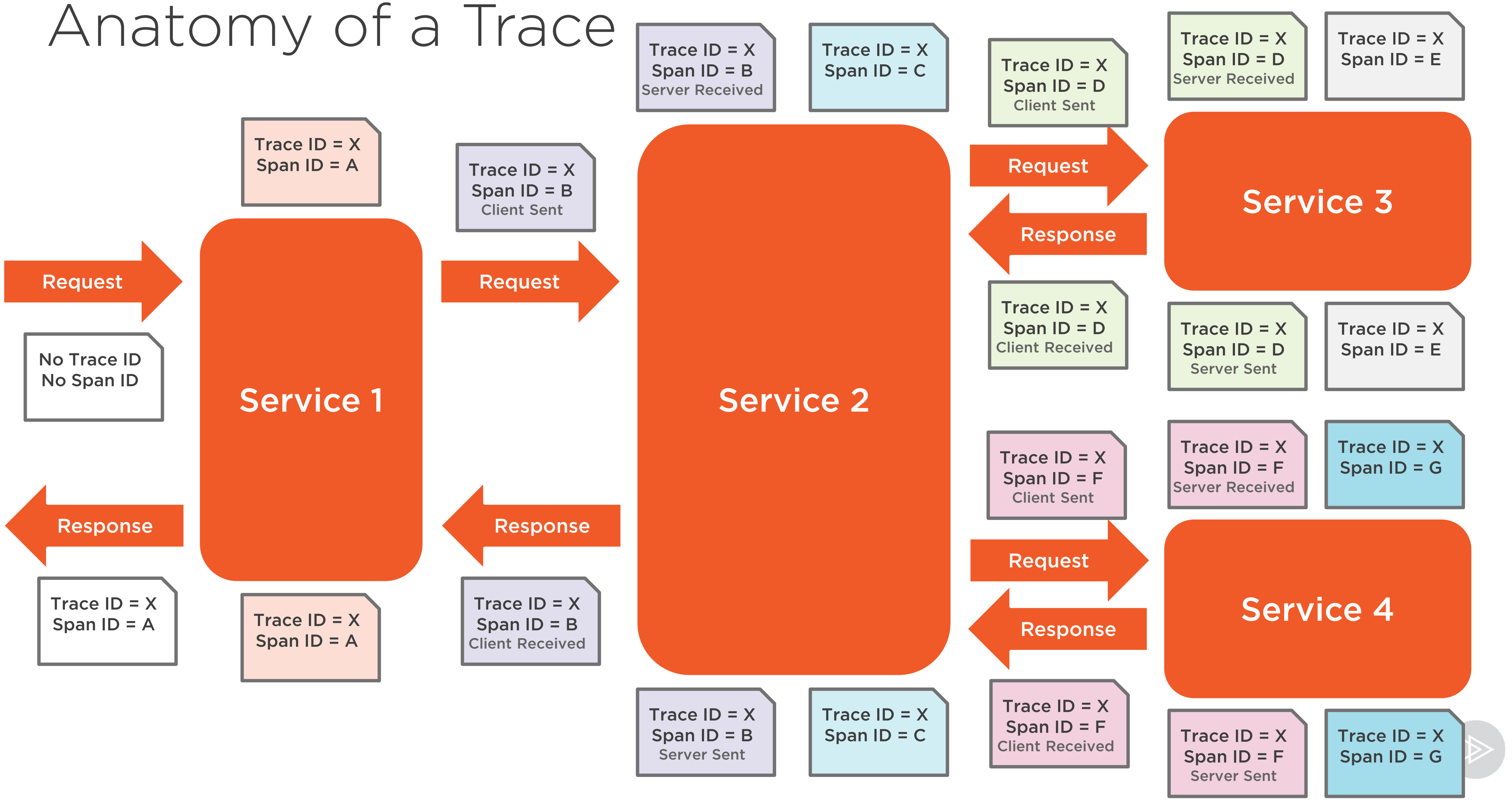
Trace

Annotation

- Client Sent
- Server Received
- Server Sent
- Client Received

Tracer

Anatomy of a Trace



What Is Automatically Instrumented?

- Runnable / Callable operations
- Spring Cloud Hystrix, Zuul
- RxJava
- Synchronous / Asychronous RestTemplate
- Spring Integration
- @Async, @Scheduled operations

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

Adding Spring Cloud Sleuth to a Project

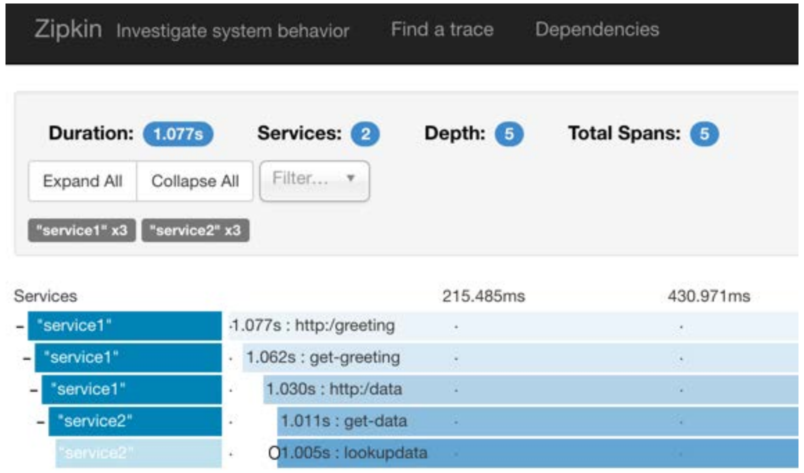
Demo

Adding Spring Cloud Sleuth to services

Updating properties files to reveal traces

Testing services and observing output

Visualizing Latency with Zipkin



Duration: 1.077s Services: 2 Depth: 5 Total Spans: 5

"service1" x3 "service2" x3

```
Services                           215.485ms        430.971ms
- "service1"   1.077s : http:/greeting
 - "service1"  1.062s : get-greeting
  - "service1"  1.030s : http:/data
   - "service2"  1.011s : get-data
     "service2"   1.005s : lookupdata
```

Created by Twitter,OpenZipkin public fork

Collects timing data

Shows service dependencies

Visualize latency for spans in a trace

Many integrations, besides Spring

```xml
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

Add Sleuth with Zipkin Over HTTP

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-sleuth-stream</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<!– an example binding for RabbitMQ -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-stream-binder-rabbit</artifactId>
</dependency>
```

Add Sleuth with Zipkin Over Spring Cloud Stream

Demo

Creating new Spring project

Adding Zipkin Server annotations

Starting up a Zipkin Server

Changing services to use Zipkin dependency

Visualizing and Querying Traces in Zipkin

View dependencies

Find a trace, view details

Perform annotations query

Look for durations

Demo

Viewing the dependencies between our services

Analyzing the details of a trace

Filtering by time duration

Sleuth exports 10% of spans by default

Can set property for spring.sleuth.sampler.percentage = 1.0

Custom samplers give fine-grained control

Demo

Experimenting with sampler percentages

Creating new Sampler class

Reviewing Sampler “span” properties

Viewing logs and Zipkin results

Manually Creating Spans

Create new spans

Continue existing spans

Associate with explicit parent

Add tags, events to span

Demo

Adding span to data query service

Including tags and events

Calling the microservice

Observing new span in Zipkin

Summary

Overview

The role of tracing in microservices

Problems with the status quo

What is Spring Cloud Sleuth?

Anatomy of a trace

What is automatically instrumented?

Adding Spring Cloud Sleuth to a project

Visualizing latency with Zipkin

Adding Zipkin to a solution

Working with samplers

Manually creating spans