Spring Cloud Distributed Tracing

Distributed tracing, also called distributed request tracing, is a method used to profile and monitor applications, especially those built using a microservices architecture. Distributed tracing helps pinpoint where failures occur and what causes poor performance.

Outline

Spring Cloud Tracing using Sleuth and Zipkin
by Pratap Kumar

Chasing Down Performance Issues Using Distributed Tracing

- In this module , we will look into how to chase down the latency issues in the services you build, which is a big part in building microservices

Content

Spring Cloud Tracing using Sleuth and Zipkin
by Pratap Kumar

- The Role of Tracing in microservices
- Problem with status quo
- What is Spring Cloud Sleuth?
- Anatomy of Trace
- What is automatically instrumented?
- Adding Spring Cloud Sleuth to a project
- Visualizing latency with Zipkin
- Add Zipkin to a Solution
- Working with samplers
- Manually creating spans
- Summary

Role of Tracing

- One of the worst thing you can hear in distributed system is, “Something is Slow”

- What is slow?
  - Is that behaviour abnormal, do I have a baseline.
  - This is a very hard question in distributed system.
  - The call graph is very complicated and trying to figure out what cause that latency is more dizzy.

Role of Tracing

- **Locate misbehaving Components**
  - Where is Latency actually happening
  - Is it unusual compare to base line
- **Observe end-to-end latency**
  - The performance of one service no longer tells the story
  - How the overall latency shapes up.
- **Understand actual, not specified, behaviour**

Problem with Status Quo

- Instrumenting all communication paths
- Collecting log across components, threads
- Correlating and querying logs
- Seeing the bigger picture / graph

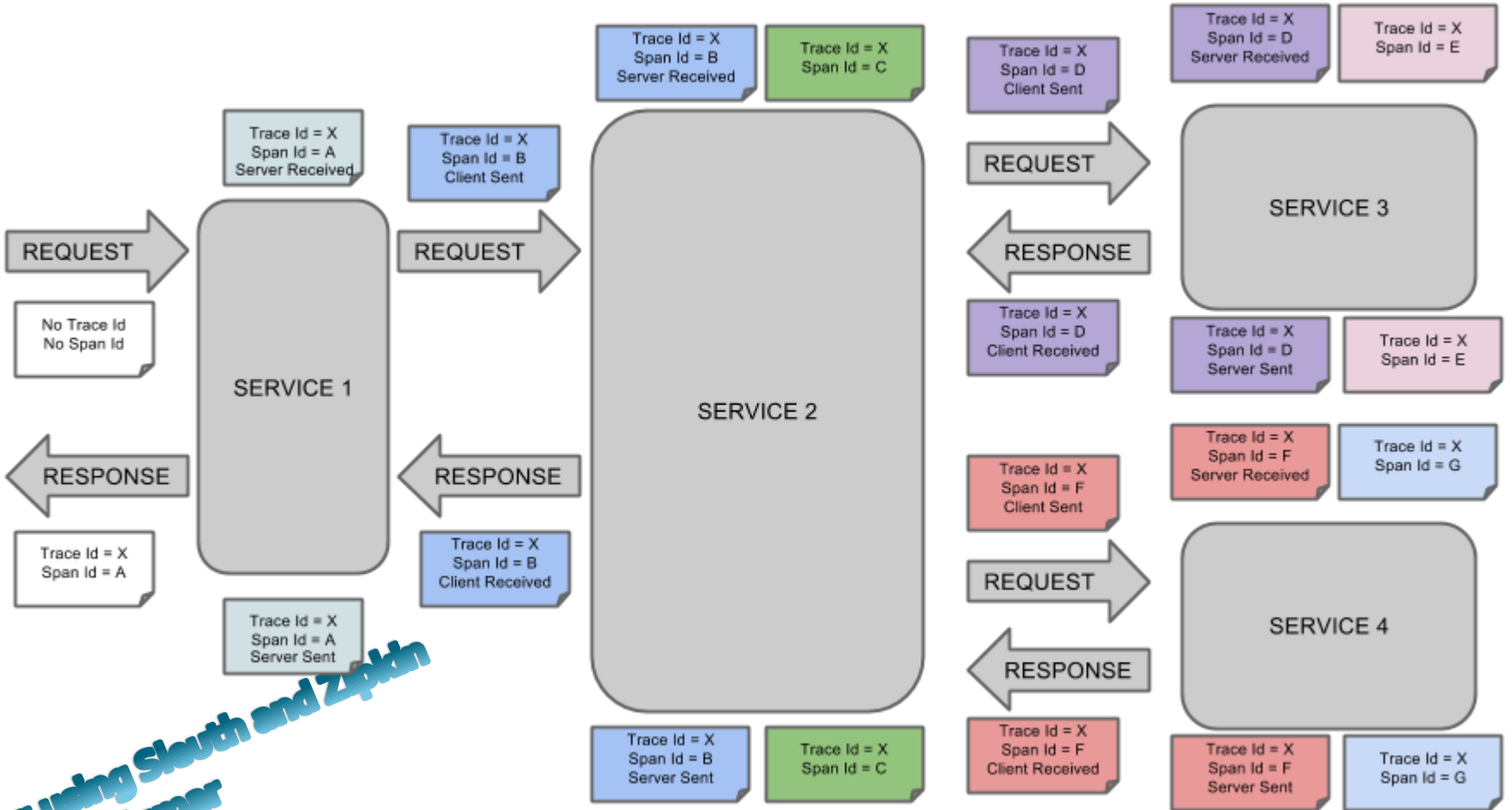
Spring Cloud Sleuth

Automatic Instrumentation of communication channels

- We get near real time data collection of timing information
- We are able to trace that information and generate traces that can flow through the path of a particular call tree
- Great Integration with Zipkin

Spring Cloud Sleuth Terms

- Trace
- Span
- Annotation
  - Client Sent
  - Server Received
  - Server Sent
  - Client Received
- Tracer



Spring Cloud Tracing using Sleuth and Zipkin
by Pratap Kumar

What is Automatically Instrumented?

- Runnable / Callable Operation
- Spring Cloud Hystrix, Zuul
- RxJava
- Synchronous / Asynchronous RestTemplate
- Spring Integration
- @Async , @Scheduled Operations

Adding Spring Cloud Sleuth to a Project

- Import DataService1
- Import DataService2
- Import CustomerService

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-sleuth</artifactId>
   <version>2.1.3.RELEASE</version>
</dependency>
```

Demo

- Adding Spring Cloud Sleuth to services
- Updating properties files to reveal traces
- **Testing services and observing outputs**

application.properties

- DataService1
  - server.port=8081
  - server.servlet.context-path=/fastpass
  - spring.application.name=ContactDetails
  - logging.level.org.springframework.cloud.sleuth=debug
- DataService1
  - server.port=8082
  - server.servlet.context-path=/fastpass
  - spring.application.name=VehicleDetails
  - logging.level.org.springframework.cloud.sleuth=debug

application.properties

- CustomerService
  - server.port=8083
  - server.servlet.context-path=/fastpass

  - spring.application.name=ContactService
  - logging.level.org.springframework.cloud.sleuth=debug

Run

- Run DataService1
- localhost:8081/fastpass/customer/100/contactdetails

- Run DataService2
- localhost:8082/fastpass/customer/100/vehicledetails

- Run CustomerService
- localhost:8083/fastpass/customer/102

Visualizing Latency with Zipkin

- Created by Twitter, OpenZipkin public fork
- Collects timing data
- Show service dependencies
- Visualize latency for spans in a trace
- Many integrations, besides spring

Demo

- **Create Zipkin Server**
  - **Creating new Spring boot starter project**

```
<dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-server</artifactId>
        <version>2.11.7</version>
</dependency>
<dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-autoconfigure-ui</artifactId>
        <version>2.11.7</version>
</dependency>
```

  - **Adding Zipkin Server annotations**
    - @EnableZipkinServer
  - **Starting up a Zipkin Server**

Demo

Spring Cloud Tracing using Sleuth and Zipkin
by Pratap Kumar

- **Changing services to use Zipkin dependency**
  - DataService1
  - DataService2
  - CustomerService

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
  <version>2.1.3.RELEASE</version>
</dependency>
```

Dependency

Spring Cloud Tracing using Sleuth and Zipkin
by Pratap Kumar

- **Add Sleuth with Zipkin Over HTTP**

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-zipkin</artifactId>
   <version>2.1.3.RELEASE</version>
</dependency>
```

Demo

- **Changing services to use Zipkin dependency**
  - DataService1
  - DataService2
  - CustomerService
- Edit application.properties file of these application
  - spring.zipkin.base-url=http://localhost:9411
  - spring.sleuth.sampler.probability=1.0
  - View the Zipking Dashboard

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
- Analysing the details of a service
- Filtering by time duration

Sampler

- Sleuth exports 10% of spans by default
- Can set property for
  - spring.sleuth.sampler.probability=1.0
- Custom samplers give fine-grainer control

res

- https://opentracing.io/docs/overview/what-is-tracing/
- https://microservices.io/patterns/observability/distributed-tracing.html
- https://callistaenterprise.se/blogg/teknik/2017/07/29/building-microservices-part-7-distributed-tracing/
- https://www.youtube.com/watch?v=Vdwn7LvIPlo