Spring Boot Integration with Sleuth and Zipkin

DIVYA JAYAPRAKASH

What is Sleuth?

- **Tracing** requests propagating through microservices
- Specially between **two or more** microservices (Auction Ms wants to fetch data from OPS DB)
- Sleuth adds the trace **id** and the span id to the log statements
- Span id is for a basic **unit** if work
- Trace id is **common** for a set of span ids

Why is Sleuth Used?

- We don't know the **implementation** of the microservice that we are calling
- In general, in **production**, there will be several requests running on multiple threads in automated batch processes
- It will be like, in the morning, we can see request logged as failed in DB or dust the server logs. We have to rerun, see if the same issue is **replicable**.
- In webservice testing or operation, if a request fails, we have to do **path** tracing
- We can **manually** generate request trace id taking the last request number from the DB (tedious). Sleuth beautifully does it for us.

How is Sleuth Implemented

- Created new spring boot application with dependency **spring-cloud-starter-sleuth**
- //Copied the dependencies to the old car demo application
- **Spring.application.name**= DJ_Test_Sleuth_MicroS_001, will appear in the log trace
- Log statements importing **log4j** or slf4j
- Tomcat started on localhost default port **8080**
- [DJ_Test_Sleuth_Missile_007,**44462edc42f2ae73,44462edc42f2ae7**3,false]

Extension of Sleuth across Microservices

- Deployed another spring boot restful webservice(DJ_Test_Sleuth_MicroS_002) in Tomcat running on local host port number **8081.** From the previous webservice on localhost8080, **calling** webservice method in localhost 8081
- **RestTemplate** class is in the downloaded jar in org.springframework.web.client
- **GetForObject** - URL as input, it does a GET and returns an representation object
- One simple way of calling from one microservice to another in resteasy API

  (**SOAP service** call - old)

Extension of Sleuth across Microservices

- We created Carservice, then autowired it in carcontroller class
- **@Bean** – produces the independent bean to be managed by the Spring container
- `<bean/>` in spring XML schema
- **@Autowire** – dependency injection of the independent bean- ids stored and carried in headers
- 2016-06-17 16:12:36.902 INFO [DJ_Test_Sleuth_MicroS_001,432943172b958030,432943172b958030,false]
- 2016-06-17 16:12:36.940 INFO [DJ_Test_Sleuth_MicroS_002,432943172b958030,b4d88156bc6a49ec,false]

What is Zipkin?

- Time tracing
- Latency is the duration of a particular request
- Show how many spans a trace has
- Time taken by each span
- Which is the calling webservice
- Chain of links hit

Why is Zipkin used?

- Development phase for **code optimization**
- In the last two days, this is the time taken for post request!
- Webservices are used by clients in real time environment- that should work in **a swish**

How is Zipkin implemented?

- Spring boot application with **Zipkin UI and Zipkin Server** dependencies
- Deployed the application on tomcat running on localhost 9411(default used by zipkin)
- **Server.port=9411**
- http://localhost:9411 - Zipkin UI
- Applications **export** information to zipkin
- 2016-06-17 16:12:36.902 INFO

DJ_Test_Sleuth_MicroS_001,432943172b958030,432943172b958030,**true**]

2016-06-17 16:12:36.940 INFO
[DJ_Test_Sleuth_MicroS_002,432943172b958030,b4d88156bc6a49ec,true]

Point A to Point B to Point C

Thank You