How To Implement Distributed Logging & Tracing Using Sleuth & Zipkin?

java Microservices Sleuth & Zipkin Spring Cloud *by devs5003 - April 21, 2021* 1

In Microservices Architecture based Applications, there are multiple services which interact with each other. In order to serve a client request, one request may involve multiple microservices call to get the response. If any exception occurs or any latency issue appears during such calls, then how will we identify the root cause of the issue? Of course, we will make use of Spring Cloud Sleuth & Zipkin to support distributed logging & Tracing. Therefore, we are going to learn about 'How to implement Distributed Logging & Tracing using Sleuth & Zipkin' and related concepts accordingly.

If we are working on a Standard application, we can implement logging by adding one log file to identify any exception or other issues like response timings or latency etc. However, in case of Microservices based applications, there are multiple small applications. Therefore, we will have to maintain multiple log files such as at least one log file per application. In this case, correlating the logs to a particular request chain becomes difficult. Hence, identifying the root cause of the issue also becomes complicated. In order to overcome from such kind of issues, we make use of Spring Cloud Sleuth and Zipkin. Let's discuss our topic 'How to Implement Distributed Logging & Tracing using

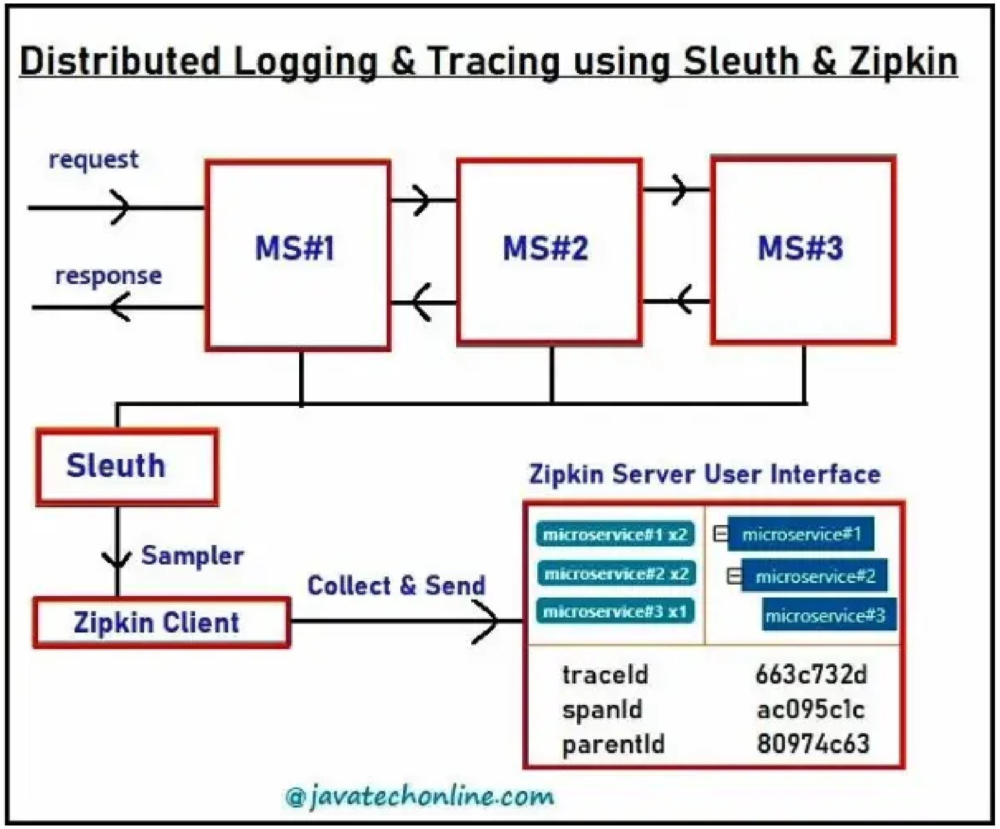
FOLLOW US

RECENTLY PUBLISHED POSTS

- How to Implement Feign Client in Spring Boot Microservices?
- How to implement Fault Tolerance in Microservices using Resilience4j?
- How to monitor Spring Boot Microservices using ELK Stack?
- Java 8 Features
- How to work with Apache Kafka in Spring Boot?
- Design Patterns in Java -3
- Design Patterns in Java -2

Sleuth & Zipkin?' And related concepts accordingly.

Table of Contents (Click on links below to navigate) [hide]



» Design Patterns in Java
» What is Spring Boot ?
» SOLID Principles : The Dependency Inversion Principle

SUBSCRIBE TO GET UPDATES

Email *

Subscribe!

DO YOU HAVE A QUERY ?

Submit your Query

What is Logging & Tracing?

Tracing : Tracing is a process of finding an execution path or a flow of multiple microservices to serve a request.

Logging : Logging is a process of admitting all log messages for a trace. These messages can be of any type such as DEBUG, INFO, WARN, ERROR etc.

Here, we will be implementing Logging and Tracing for multiple Microservices involved in serving one request. We also call it ‘Distributed Tracing and Logging’. Once it is implemented one time, then developer doesn’t need to check the code again to find the flow either for testing purpose or for debug purpose.

What is Sleuth and Zipkin used for ?

Sleuth : Sleuth provides unique ids for the request flow. Further, the developer uses these ids to find out the flow of execution with the help of tools like Zipkin, ELK etc. Generally it has two ids : Trace id and Span Id. Trace Id is applicable for a complete flow. If the developer gets this Id, he/she can find the flow of execution in all microservices involved. However, Span Id is applicable to the flow of one microservice. If the developer gets this Id, he/she can find the flow of execution in a particular microservice. Moreover, there is a concept of parent id which is applicable to a particular microservice like span id. During the flow of execution, span id of the previous microservice becomes the parent id of the next microservice as shown below.

Zipkin : We use Zipkin in two parts : Zipkin Client and Zipkin Server. Zipkin Client contains Sampler which collects data from microservices with the help of Sleuth and provides it to the Zipkin Server. In order to utilize the benefits of both tools, we should always add Zipkin Client’s dependency along with Sleuth in every microservice. However, there must be only one centralized Zipkin Server, which collects all data from Zipkin Client and display it as a UI. Hence, after making a request developer should look into Zipkin Server to find trace id, span id and even flow of execution. From here itself, Open Log files to check Log lines that are related to the current trace Id.

How to download & setup Zipkin Server?

In order to check the flow of execution in microservices chain, we need to download & setup Zipkin Server in our system. Please follow below steps to get it done.

1) Download ZIPKIN from the Link below:
Zipkin Download Link

2) Extract to a folder ‘ZipkinExecutable’. The downloaded file name should be like ‘ZipkinExecutable.zip’

3) Open command prompt at the above location where your extracted folder exists.
For example: Let’s assume that your location is C:/Downloads/ZipkinExecutable/>

4) Run Jar file by using below command:
```
>java -jar zipkin-server-2.12.9-exec.jar
```
If Zipkin Server started successfully, you will see message something like “Started ZipkinServer in 14.094 seconds (JVM running for 15.452)”

5) Open your Browser and Enter URL:
http://localhost:9411/zipkin/

6) Click on ‘Find Traces’ button. It will display ‘Showing: 0 of 0‘

How to Implement Distributed Logging & Tracing using Sleuth & Zipkin?

In order to implement Distributed Logging & Tracing, we will do three major tasks as below:

1. Develop three Microservices applications using Spring Boot which will interact with each other.
2. Implement distributed logging & tracing using Spring Cloud Sleuth
3. Test distributed logging & tracing using Zipkin Server

Let’s start with developing application step by step.

Step#1 : Create a Spring Boot Project using STS(Spring Tool Suite)

Here, we will use STS(Spring Tool Suite) to create our Spring Boot Project. If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot using STS. While creating a project in STS add starters ‘Sleuth’, ‘Zipkin Client’, ‘Spring Web’ in order to get features of them.

Step#2: Modify application.properties file

```
server.port=8081
spring.application.name=microservice#1
logging.file.name=D:/myLogs/serviceOne.log
#can include eureka details also—-
```

Step#3: Create AppConfig.java to Setup Configurations

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
import brave.sampler.Sampler;
```

```
@Configuration
public class AppConfig {

    //Collects data from Sleuth and provides it to Zipkin Client
    @Bean
```

```
        public Sampler samplerOb() {
            //return Sampler.NEVER_SAMPLE;
           return Sampler.ALWAYS_SAMPLE;
        }

        //Creates RestTemplate Object
       @Bean
       public RestTemplate rt() {
           return new RestTemplate();
       }
   }
```

Step#4: Write a RestController to implement a Microservice

Write a RestController like below:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class AppOneRestController {

        private Logger log = LoggerFactory.getLogger(AppOneRestController.class);

        @Autowired
        private RestTemplate restTemplate;

        @GetMapping("/m1")
        public String methodOne() {
                log.info("Inside microservice#1");
                String response = restTemplate.getForObject("http://localhost:8082/m2", String.class);
                log.info("response by microservice#1 " + response);
                return "returning from microservice#1";
        }
}
```

Step#5: Write two more applications just like above

As similar to above code also define two more applications. For example, let’s say MyServiceTwo and MyServiceThree. However, Config Files will be similar but properties may differ. Also change the code of RestController accordingly.

AppConfig.java for Service #2

Same as for Service #1

application.properties for Service #2

```
server.port=8082
spring.application.name=microservice#2
logging.file.name=D:/myLogs/serviceTwo.log
```

RestController for Service #2

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class AppTwoRestController {

        private Logger log = LoggerFactory.getLogger(AppTwoRestController.class);

        @Autowired
        private RestTemplate restTemplate;

        @GetMapping("/m2")
        public String methodTwo() {
                log.info("Inside microservice#2");
                String response = restTemplate.getForObject("http://localhost:8083/m3", String.class);
                log.info("response by microservice#2 " + response);
                return "returning from microservice#2";
        }
}
```

AppConfig.java for Service #3

Same as for Service #1

application.properties for Service #3

```
server.port=8083
spring.application.name=microservice#3
logging.file.name=D:/myLogs/serviceThree.log
```

RestController for Service #3

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class AppThreeRestController {

        private Logger log = LoggerFactory.getLogger(AppThreeRestController.class);

        @Autowired
        private RestTemplate restTemplate;

        @GetMapping("/m3")
        public String methodThree() {
                log.info("Inside microservice#3");
                log.info("done with microservice#3 " );
                return "returning from microservice#3";
        }
}
```

How to test the application as a whole?

Lets follow below steps to test the whole application. Screenshots are also provided to make testing easier.

1) Start Zipkin Server
2) Start All Microservices one by one
3) Enter URL: http://localhost:8081/m1
4) Open a new browser window and enter http://localhost:9411/zipkin
5) Click on Find Trace. You will see details of three running microservices highlighted in green color.
6) Click on Blue color bar. You will see trace record of each microservice.
7) Click on any one option. Let’s click on ‘microservice#2’. A new popup window will appear with all the details.
8) Click on ‘Show IDs’, you will find details of traceId, spanId and parentId. Now, copy traceId from here.
9) Open Log files and search(ctrl+F) with traceId to see all log lines related to current request.

For example, once we enter URL ‘http://localhost:9411/zipkin’ and click on ‘Find traces’ button, we will see below screen.

Now, Let’s click on blue bar just above the microservices highlighted in green color. We will get a screen something like below.

In order to find out the trace Ids for a particular microservice click on any microservice as highlighted above. For example, we click on microservice#2. Now, We will get a popup window something like below.

Finally, copy the trace Id from here and look into respective log files to see the log messages. Furthermore, we have provided log messages generated after running this example for all three microservices in the next section.

Messages from Log Files

Below are the messages generated from Log files for respective Microservice.

Microservice#1 : Log messages

```
2021-04-21 23:53:27.880 INFO [microservice#1,663c732d1728af87,663c732d1728af87] 56972 --- [http-ni
2021-04-21 23:53:28.064 INFO [microservice#1,663c732d1728af87,663c732d1728af87] 56972 --- [http-ni
```

Microservice#2 : Log messages

```
2021-04-21 23:53:27.939 INFO [microservice#2,663c732d1728af87,ac095c1c5bce450e] 21528 --- [http-nio
2021-04-21 23:53:28.045 INFO [microservice#2,663c732d1728af87,ac095c1c5bce450e] 21528 --- [http-nio
```

Microservice#3 : Log messages

```
2021-04-21 23:53:28.020 INFO [microservice#3,663c732d1728af87,79a65229aacee4d2] 43752 --- [http-ni
2021-04-21 23:53:28.021 INFO [microservice#3,663c732d1728af87,79a65229aacee4d2] 43752 --- [http-ni
```

We can see occurrences of trace id : ‘663c732d1728af87’ in all three microservices as it is common for all. However, span Id is specific for each microservice. Moreover, since microservice#1 is the parent microservice, hence both ids are same for it.

Conclusion

After going through all the theoretical & example part of ‘How to Implement Distributed Logging & Tracing using Sleuth & Zipkin?’, finally, we should be able to implement logging & tracing concept using Sleuth and Zipkin as a whole. Similarly, we expect from you to further apply this knowledge in your project accordingly. Moreover, having familiarity of terminologies explained in this article is very crucial to implement microservices for a developer especially in Java. In addition, we will also be updating the article time to time accordingly if any need arises. Moreover, feel free to provide your comments in comments section below.

Tagged distributed logging distributed logging spring boot distributed tracing in microservices spring boot distributed tracing using sleuth and zipkin How to Implement Distributed Logging & Tracing using Sleuth & Zipkin? microservices distributed logging sleuth and zipkin example sleuth logging sleuth spring boot sleuth spring boot example sleuth zipkin sleuth zipkin spring boot example Spring Boot 2 sleuth zipkin example spring boot sleuth zipkin example spring boot tracing spring boot zipkin spring boot zipkin example spring boot zipkin server spring cloud sleuth spring cloud sleuth example spring cloud zipkin spring cloud zipkin client spring cloud zipkin server spring cloud zipkin server example spring sleuth spring sleuth example spring sleuth zipkin zipkin and sleuth spring boot example zipkin server spring boot 2 zipkin spring boot

‹ **Previous article**
Lambda Expression Java

Next article ›
Spring Boot MongoDB CRUD Example

One thought on "How to Implement Distributed Logging & Tracing using Sleuth & Zipkin?"

May 4, 2021 at 4:18 PM

Hendi Santika

Nice article.

But, It would be better if You put your code in GitHub so that We can compare the code.

Thanks

Reply

Leave a Reply

Name *

Email Address *

Website

Comment *

Post Comment

☐ Yes, add me to your mailing list.