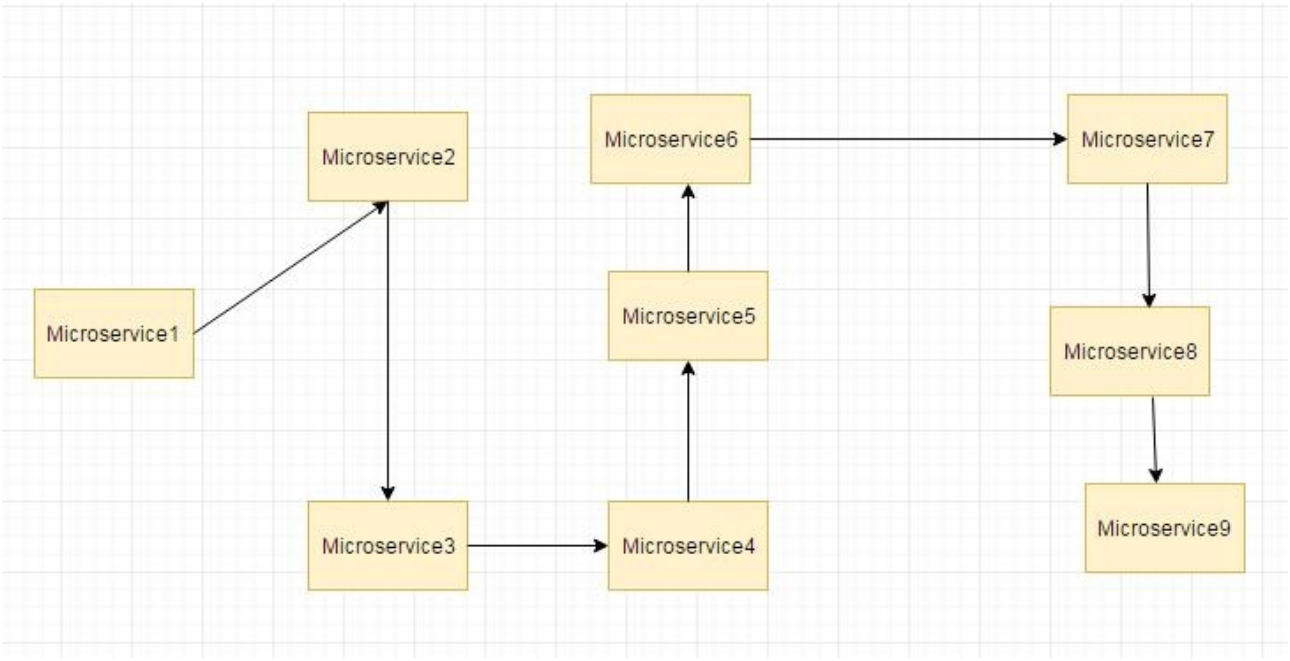
Distributed Log Tracing using Sleuth and Zipkin

Microservices architecture involve multiple services which interact with each other.
So a functionality may involve call to multiple microservices. Usually for systems developed using Microservices architecture, there are many microservices involved.

These microservices collaborate with each other.

Consider the following microservices-



If suppose during such calls there are some issues like exception has occurred.
Or may be there are latency issues due to a particular service taking more than expected time.
How do we identify where the issue is occurring.

In regular project we would have used logging to analyze the logs to know more about occurred exceptions and also performance timing. But since microservices involves multiple services we cannot use regular logging.

Each Service will be having its own separate logs.

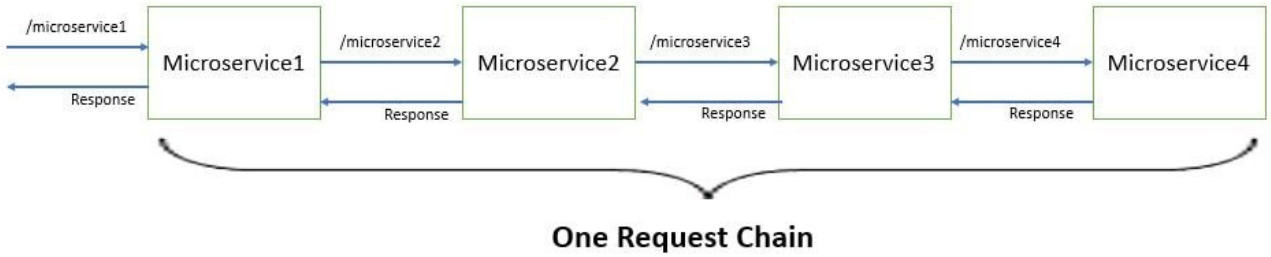
So we will need to go through the logs of each service. Also how do we correlate the logs to a request call chain i.e which logs of microservices are related to Request1, which are related to Request2.

To resolve these issues we make use of Spring Cloud Sleuth and Zipkin

Spring Cloud Sleuth is used to generate and attach the trace id, span id to the logs so that these can then be used by tools like Zipkin and ELK for storage and analysis

Zipkin is a distributed tracing system.
It helps gather timing data needed to troubleshoot latency problems in service architectures.
Features include both the collection and lookup of this data.

Example getting started create 4 ms :



ms1---> ms2------> ms3--->ms4
how to handle distributed logging and tracing?

ms1

```
@SpringBootApplication
public class Ms1Application {

	public static void main(String[] args) {
		SpringApplication.run(Ms1Application.class, args);
	}

	@Bean
	public RestTemplate getRestTemplate() {
		return new RestTemplate();
	}
}


@RestController
public class MyController {

	@Autowired
	private RestTemplate restTemplate;

	private Logger logger=LoggerFactory.getLogger(MyController.class);
	@GetMapping("helloms1")
	public String hello() {
		logger.info("inside mycontroller ms1");
		String value =restTemplate.getForObject("http://localhost:8082/helloms2",
String.class);
		logger.info("value return from ms2"+value);
		return "hello from ms1";
	}
}
```

```
server.port=8081
spring.application.name=ms1
```

ms2

```
@RestController
public class MyController {

	@Autowired
	private RestTemplate restTemplate;

	private Logger logger=LoggerFactory.getLogger(MyController.class);
	@GetMapping("helloms2")
	public String hello() {
		logger.info("inside mycontroller ms2");
		String value =restTemplate.getForObject("http://localhost:8083/helloms3",
String.class);
		logger.info("value return from ms3"+value);
		return "hello from ms2";
	}
}
```

```
server.port=8082
spring.application.name=ms2
```

ms3

```
@RestController
public class MyController {

	@Autowired
	private RestTemplate restTemplate;

	private Logger logger=LoggerFactory.getLogger(MyController.class);
	@GetMapping("helloms3")
	public String hello() {
		logger.info("inside mycontroller ms3");
		String value =restTemplate.getForObject("http://localhost:8084/helloms4",
String.class);
		logger.info("value return from ms4"+value);
		return "hello from ms3";
	}
}
```

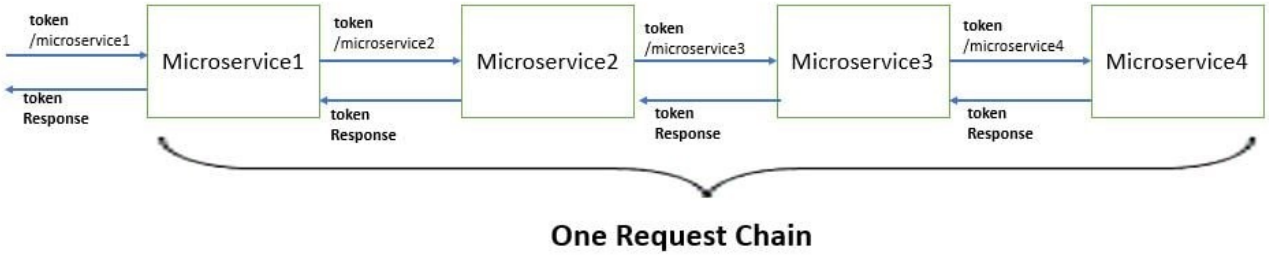
```
server.port=8083
spring.application.name=ms3
```

ms4

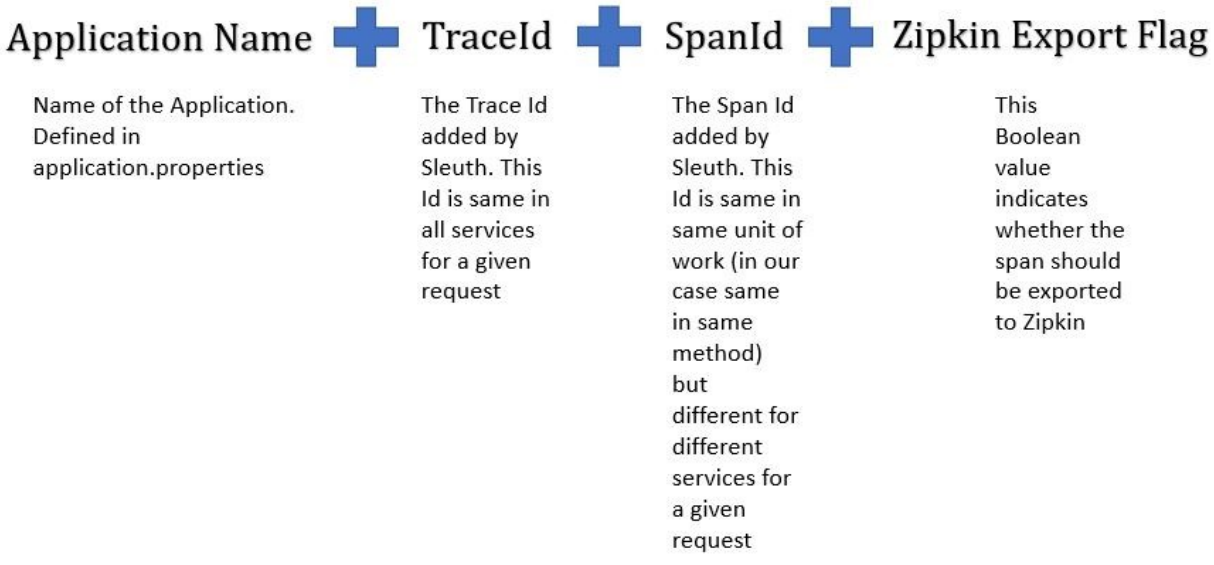
```
@RestController
public class MyController {

	private Logger logger=LoggerFactory.getLogger(MyController.class);
	@GetMapping("helloms4")
	public String hello() {
		logger.info("inside mycontroller ms4");
		return "hello from ms4";
	}
}
```

```
server.port=8084
spring.application.name=ms4
```

Implement distributed tracing using Spring Cloud Sleuth




```
[microservice2,92bb6a46d4c8c452,640756ff93630bc1,true]
[microservice2,92bb6a46d4c8c452,640756ff93630bc1,true]
[microservice3,92bb6a46d4c8c452,d7a5a2098395f621,true]
[microservice3,92bb6a46d4c8c452,d7a5a2098395f621,true]
```



Next we will be adding the spring cloud sleuth for all the microservices.
Using Spring Cloud Sleuth we will be adding a unique token to all requests.
Spring Cloud Sleuth is used to generate and attach the trace id, span id to the logs so that these can then be used by tools like Zipkin and ELK for storage and analysis.

step 1: add dependency: spring cloud sleuth and zipkins
spring.zipkin.enabled=false

```
@Bean
public Sampler defaultSampler() {
        return Sampler.ALWAYS_SAMPLE;
}
```

In the Microservice1Application class

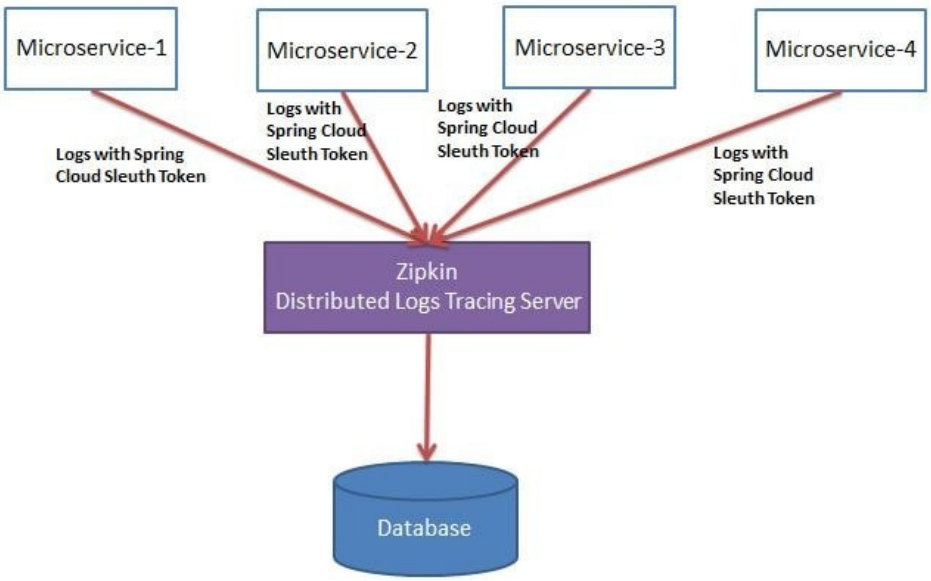
In distributed tracing the data volumes can be very high so sampling can be important.
This determines what amount of data you want to send to a centralized log analysis tool.

If you want to send all the data or only a part of it.
If you are exporting span data to Zipkin or Spring Cloud Stream, there is also an AlwaysSampler that exports everything and a PercentageBasedSampler that samples a fixed fraction of spans.

We will be making use of the Always Sampler for this example.

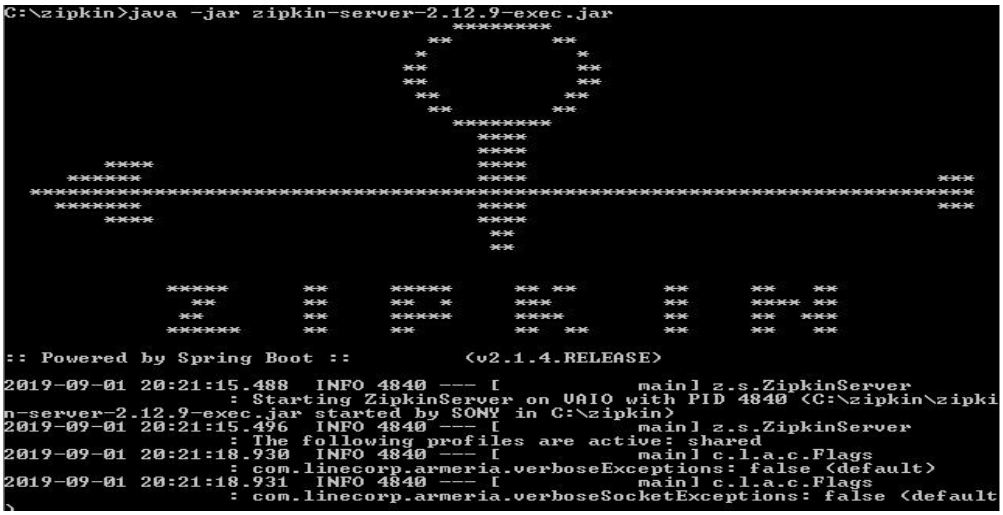
Use Zipkin for distributed log analysis

Zipkin is a distributed tracing system.
It helps gather timing data needed to troubleshoot latency problems in service architectures.
Features include both the collection and lookup of this data.



From maven repository get the latest zipkin jar.
Once you have downloaded this jar using the command prompt run :

java -jar zipkin-server-2.12.9-exec.jar



Also from the application.properties remove the property we had defined earlier

- spring.zipkin.enabled=false

open
http://localhost:9411/