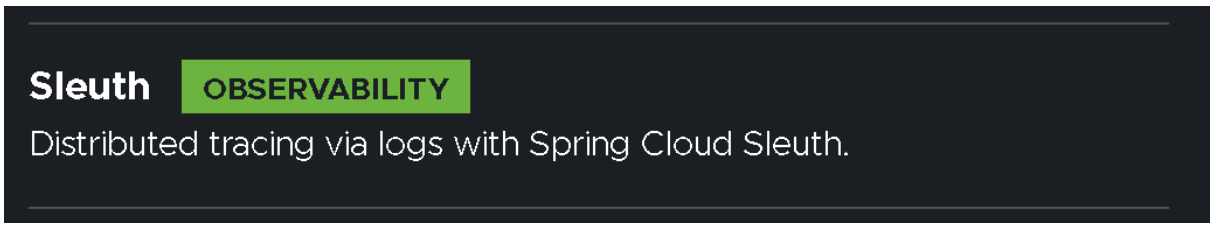
Certified Tech Developer
The Ultimate Degree

Distributed Tracing

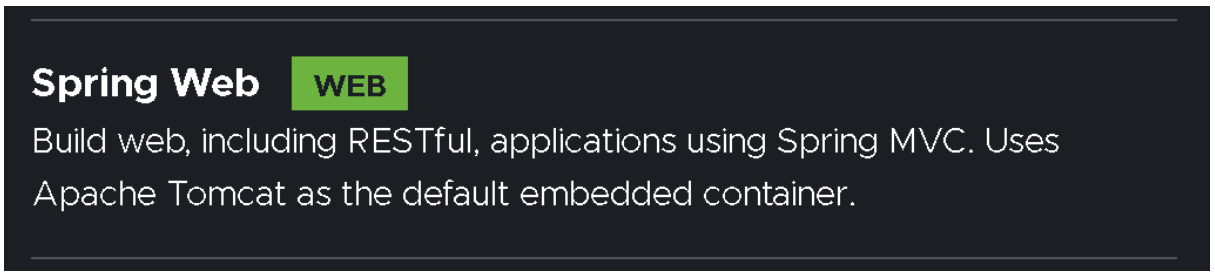
Implementação de Spring Cloud Sleuth

Vamos deixar aqui um passo a passo bem fácil e prático para a implementação do Srping Cloud Sleuth.

Para essa implementação do Spring Cloud Sleuth, podemos adicionar sua dependência no momento em que criamos nosso projeto através do site **https://start.spring.io/**.



Vamos adicionar também a dependência Spring Web.



Somente para fins de testes, vamos criar um controller com mapeamento de duas requisições. Nesse controller, teremos dois caminhos, Path1 que chamará o Path2 em uma porta fixa 8090. E a ideia aqui é executarmos duas instâncias separadas do mesmo aplicativo.

```java
public class Controller {
    private static final Logger logger = LoggerFactory.getLogger(Controller.class);
    private RestTemplate restTemplate;

    @Value("${spring.application.name}")
    private String applicationName;

    public Controller(RestTemplate restTemplate) {this.restTemplate = restTemplate;}

    @GetMapping("/path1")
    public ResponseEntity path1() {
        logger.info("Request at {} for request /path1 ", applicationName);
        String response = restTemplate
                .getForObject( url: "http://localhost:8090/service/path2", String.class);
        return ResponseEntity.ok("response from /path1 + " + response);
    }

    @GetMapping("/path2")
    public ResponseEntity path2() {
        logger.info("Request at {} at /path2", applicationName);
        return ResponseEntity.ok("response from /path2 ");
    }
}
```

Precisamos fazer com que o RestTemplate seja injetado como um bean em vez de inicializá-lo diretamente, para que o sleuth injete cabeçalhos na solicitação de saída. E dessa forma, o sleuth irá adicionar um interceptor ao RestTemplate para injetar um cabeçalho com o trace e span id na requisição de saída.

```java
    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
```

Agora, vamos iniciar as duas instâncias, e para fazer isso, primeiramente compile o aplicativo através do comando mvn clean verify e então, abra o prompt de comando e execute o seguinte comando para iniciar o “Serviço 1”.

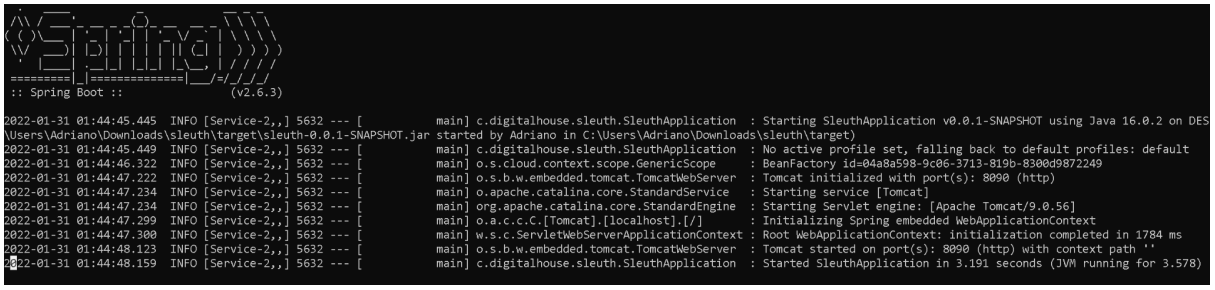
```
java -jar .\nomeDaSuaAplicação-0.0.1-SNAPSHOT.jar .\ --spring.application.name=Service-1 .\ --server.port=8081
```

Ao executar o comando, é importante que você esteja dentro da pasta target do seu projeto. E caso tenha sucesso ao executar o comando, teremos o seguinte resultado:

E em outro terminal, devemos executar o “Serviço 2”, através do comando:

```
java -jar .\sleuth-0.0.1-SNAPSHOT.jar .\ --spring.application.name=Service-2 .\ --server.port=8090
```

Caso tenha sucesso ao executar o comando, teremos o seguinte resultado:



Assim que o aplicativo for iniciado, chame o “Serviço 1” da /path1 através do comando:

```
curl -i http://localhost:8081/service/path1
```

E então, poderemos ver o Distributed Tracing na prática:

```
2022-02-24 17:16:29.355  INFO [Servico-1,02942c5a6b337b84,02942c5a6b337b84]
 7624 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Init
ializing Spring DispatcherServlet 'dispatcherServlet'
2022-02-24 17:16:29.356  INFO [Servico-1,02942c5a6b337b84,02942c5a6b337b84]
 7624 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Init
ializing Servlet 'dispatcherServlet'
2022-02-24 17:16:29.360  INFO [Servico-1,02942c5a6b337b84,02942c5a6b337b84]
 7624 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Comp
leted initialization in 1 ms
```

Aqui, podemos identificar o nome do serviço (conteúdo do retângulo vermelho), o trace id (conteúdo do retângulo amarelo) e o span id (conteúdo do retângulo verde).