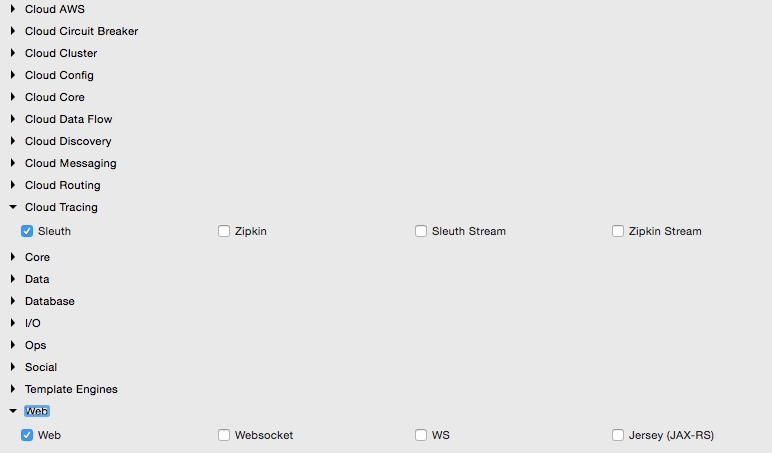
Lab 22 - Sleuth

* Update Search and Search API Gateway. Before this, the Sleuth dependency needs to be added to the respective POM files, which can be done via the following code:
* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-sleuth</artifactId>  
  </dependency>
* In the case of building a new service, select **Sleuth** and **Web**, as shown here:
* 
* Add the Logstash dependency to the Search service as well as the Logback configuration, as in the previous example.
* The next step is to add two more properties in the Logback configuration, as follows:
* <property name="spring.application.name" value="search-service"/>  
  <property name="CONSOLE\_LOG\_PATTERN" value="%d{yyyy-MM-dd HH:mm:ss.SSS} [${spring.application.name}] [trace=%X{X-Trace-Id:-},span=%X{X-Span-Id:-}] [%15.15t] %-40.40logger{39}: %m%n"/>
* The first property is the name of the application. The names given in this are the service IDs: search-service and search-apigateway in Search and Search API Gateway, respectively. The second property is an optional pattern used to print the console log messages with a trace ID and span ID. The preceding change needs to be applied to both the services.
* Add the following piece of code to advise Sleuth when to start a new span ID in the Spring Boot Application class. In this case, AlwaysSampler is used to indicate that the span ID has to be created every time a call hits the service. This change needs to be applied in both the services:
* @Bean  
   public AlwaysSampler defaultSampler() {  
   return new AlwaysSampler();  
   }
* Add a new endpoint to Search API Gateway, which will call the Search service as follows. This is to demonstrate the propagation of the trace ID across multiple microservices. This new method in the gateway returns the operating hub of the airport by calling the Search service, as follows:
* @RequestMapping("/hubongw")  
   String getHub(HttpServletRequest req){  
   logger.info("Search Request in API gateway for getting Hub, forwarding to search-service ");  
   String hub = restTemplate.getForObject("http://search-service/search/hub", String.class);  
   logger.info("Response for hub received, Hub "+ hub);  
   return hub;   
   }
* Add another endpoint in the Search service, as follows:
* @RequestMapping("/hub")  
   String getHub(){  
   logger.info("Searching for Hub, received from search-apigateway ");  
   return "SFO";   
   }
* Once added, run both the services. Hit the gateway's new hub on the gateway (/hubongw) endpoint using a browser ( http://localhost:8095/hubongw).
* As mentioned earlier, the Search API Gateway service is running on 8095 and the Search service is running on 8090.
* Look at the console logs to see the trace ID and span IDs printed. The first print is from Search API Gateway, and the second one came from the Search service. Note that the trace IDs are the same in both the cases, as follows:
* 2016-04-02 17:24:37.624 [**search-apigateway**] [trace=**8a7e278f-7b2b-43e3-a45c-69d3ca66d663**,span=8a7e278f-7b2b-43e3-a45c-69d3ca66d663] [io-8095-exec-10] c.b.p.s.a.SearchAPIGatewayController : Response for hub received, Hub DXB  
    
    
  2016-04-02 17:24:37.612 [**search-service**] [trace=**8a7e278f-7b2b-43e3-a45c-69d3ca66d663**,span=fd309bba-5b4d-447f-a5e1-7faaab90cfb1] [nio-8090-exec-1] c.b.p.search.component.SearchComponent : Searching for Hub, received from search-apigateway
* Open the Kibana console and search for the trace ID using this trace ID printed in the console. In this case, it is 8a7e278f-7b2b-43e3-a45c-69d3ca66d663. As shown in the following screenshot, with a trace ID, one can trace service calls that span multiple services:
* 