Lab 14 - Spring Cloud Bus for propogating configuration changes

In this example, RabbitMQ is used as the AMQP message broker. Implement this by following the steps documented as follows:

* Add a new dependency in the chapter5.search project's pom.xml file to introduce the Cloud Bus dependency:
* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-bus-amqp</artifactId>  
  </dependency>
* The Search microservice also needs connectivity to the RabbitMQ, but this is already provided in search-service.properties.
* Rebuild and restart the Search microservice. In this case, we will run two instances of the Search microservice from a command line, as follows:
* **java -jar -Dserver.port=8090 search-1.0.jar**   
  **java -jar -Dserver.port=8091 search-1.0.jar**
* The two instances of the Search service will be now running, one on port 8090 and another one on 8091.
* Rerun the website project. This is just to make sure that everything is working. The Search service should return one flight at this point.
* Now, update search-service.properties with the following value, and commit to Git:
* originairports.shutdown:SEA
* Run the following command to /bus/refresh. Note that we are running a new bus endpoint against one of the instances, 8090 in this case:
* **curl –d {} localhost:8090/bus/refresh**
* Immediately, we will see the following message for both instances:
* **Received remote refresh request. Keys refreshed [originairports.shutdown]**
* The bus endpoint sends a message to the message broker internally, which is eventually consumed by all instances, reloading their property files. Changes can also be applied to a specific application by specifying the application name like so:
* /bus/refresh?destination=search-service:\*\*