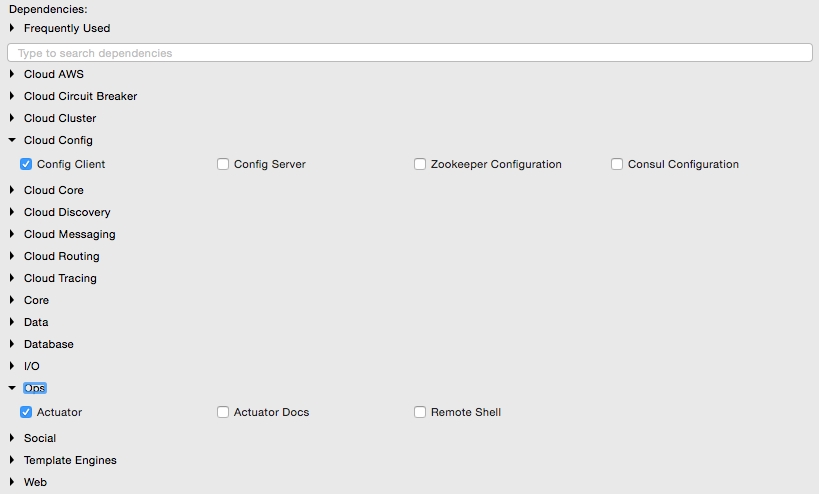
Lab 12 - Accessing the Config server from clients

Follow these steps to use the Config server instead of reading properties from the application.properties file:

* Add the Spring Cloud Config dependency and the actuator (if the actuator is not already in place) to the pom.xml file. The actuator is mandatory for refreshing the configuration properties:
* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-config</artifactId>  
   </dependency>
* Since we are modifying the Spring Boot Search microservice from the earlier chapter, we will have to add the following to include the Spring Cloud dependencies. This is not required if the project is created from scratch:
* <dependencyManagement>  
   <dependencies>  
   <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-dependencies</artifactId>  
   <version>Brixton.RELEASE</version>  
   <type>pom</type>  
   <scope>import</scope>  
   </dependency>  
   </dependencies>  
   </dependencyManagement>
* The next screenshot shows the Cloud starter library selection screen. If the application is built from the ground up, select the libraries as shown in the following screenshot:
* 
* Rename application.properties to bootstrap.properties, and add an application name and aconfiguration server URL. The configuration server URL is not mandatory if the Config server is running on the default port (8888) on the local host:
* The new bootstrap.properties file will look as follows:
* spring.application.name=search-service   
  spring.cloud.config.uri=http://localhost:8888  
    
  server.port=8090  
    
  spring.rabbitmq.host=localhost  
  spring.rabbitmq.port=5672  
  spring.rabbitmq.username=guest  
  spring.rabbitmq.password=guest
* search-service is a logical name given to the Search microservice. This will be treated as service ID. The Config server will look for search-service.properties in the repository to resolve the properties.
* Create a new configuration file for search-service. Create a new search-service.propertiesunder the config-repo folder where the Git repository is created. Note that search-service is the service ID given to the Search microservice in the bootstrap.properties file. Move service-specific properties from bootstrap.properties to the new search-service.properties file. The following properties will be removed from bootstrap.properties, and added to search-service.properties:
* spring.rabbitmq.host=localhost  
  spring.rabbitmq.port=5672  
  spring.rabbitmq.username=guest  
  spring.rabbitmq.password=guest
* In order to demonstrate the centralized configuration of properties and propagation of changes, add a new application-specific property to the property file. We will add originairports.shutdown to temporarily take out an airport from the search. Users will not get any flights when searching for an airport mentioned in the shutdown list:
* originairports.shutdown=SEA
* In this example, we will not return any flights when searching with SEA as origin.
* Commit this new file into the Git repository by executing the following commands:
* **git add –A .**  
  **git commit –m "adding new configuration"**
* The final search-service.properties file should look as follows:
* spring.rabbitmq.host=localhost  
  spring.rabbitmq.port=5672  
  spring.rabbitmq.username=guest  
  spring.rabbitmq.password=guest  
  originairports.shutdown:SEA
* The chapter5.search project's bootstrap.properties should look like the following:
* spring.application.name=search-service  
  server.port=8090  
  spring.cloud.config.uri=http://localhost:8888
* Modify the Search microservice code to use the configured parameter, originairports.shutdown. A RefreshScope annotation has to be added at the class level to allow properties to be refreshed when there is a change. In this case, we are adding a refresh scope to the SearchRestController class:
* @RefreshScope
* Add the following instance variable as a place holder for the new property that is just added in the Config server. The property names in the search-service.properties file must match:
* @Value("${originairports.shutdown}")  
   private String originAirportShutdownList;
* Change the application code to use this property. This is done by modifying the search method as follows:
* @RequestMapping(value="/get", method = RequestMethod.POST)  
   List<Flight> search(@RequestBody SearchQuery query){  
   logger.info("Input : "+ query);  
   if(Arrays.asList(originAirportShutdownList.split(",")).contains(query.getOrigin())){  
   logger.info("The origin airport is in shutdown state");  
   return new ArrayList<Flight>();  
   }  
   return searchComponent.search(query);  
   }
* The search method is modified to read the parameter originAirportShutdownList and see whether the requested origin is in the shutdown list. If there is a match, then instead of proceeding with the actual search, the search method will return an empty flight list.
* Start the Config server. Then start the Search microservice. Make sure that the RabbitMQ server is running.
* Modify the chapter5.website project to match the bootstrap.properties content as follows to utilize the Config server:
* spring.application.name=test-client  
  server.port=8001  
  spring.cloud.config.uri=http://localhost:8888
* Change the run method of CommandLineRunner in Application.java to query SEA as the origin airport:
* SearchQuery = new SearchQuery("SEA","SFO","22-JAN-16");
* Run the chapter5.website project. The CommandLineRunner will now return an empty flight list. The following message will be printed in the server:
* **The origin airport is in shutdown state**