## **Lab 4 - Create a Spring Cloud Eureka Server and Client**

**Part 1, create server**

1. Create a new Spring Boot application.

* Name the project "lab-4-eureka-server”, and use this value for the Artifact.
* Use JAR packaging and the latest versions of Java.
* Boot version 1.5.x is the most recent at the time of this writing, but you can use the latest stable version available.
* No need to select any dependencies.

2. Edit the POM (or Gradle) file. Add a “Dependency Management” section (after , before ) to identify the spring cloud parent POM. **"Camden.SR5"** is the most recent stable version at the time of this writing, but you can use the latest stable version available. Example:

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-parent</artifactId>  
 <version>Camden.SR5</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>

3. Add a dependency for group "org.springframework.cloud" and artifact "spring-cloud-starter-eureka-server". You do not need to specify a version -- this is already defined in the parent project.

4. Save an application.yml (or properties) file in the root of your classpath (src/main/resources recommended). Add the following key / values (use correct YAML formatting):

* server.port: 8010

5. (optional) Save a bootstrap.yml (or properties) file in the root of your classpath. Add the following key / values (use correct YAML formatting):

* spring.application.name=lab-4-eureka-server

6. Add @EnableEurekaServer to the Application class. Save. Start the server. Temporarily ignore the warnings about running a single instance (i.e. connection refused, unable to refresh cache, backup registry not implemented, etc.). Open a browser to [http://localhost:8010](http://localhost:8010/) to see the server running.

**Part 2, create clients**

In this next section we will create several client applications that will work together to compose a sentence. The sentence will contain a subject, verb, article, adjective and noun such as “I saw a leaky boat” or “You have the reasonable book”. 5 services will randomly generate the word components, and a 6th service will assemble them into a sentence.

7. Create a new Spring Boot web application.

* Name the project "lab-4-subject”, and use this value for the Artifact.
* Use JAR packaging and the latest versions of Java.
* Use Boot version 1.5.x or the latest stable version available.
* Add actuator and web as a dependencies.

8. Modify the POM (or Gradle) file.

* Add the same dependency management section you inserted into the server POM. (You could simply change the parent entries, but most clients will probably be ordinary applications with their own parents.)
* Add a dependency for group "org.springframework.cloud" and artifact "spring-cloud-starter-eureka".

9. Modify the Application class. Add @EnableDiscoveryClient.

10. Save an application.yml (or properties) file in the root of your classpath (src/main/resources recommended). Add the following key / values (use correct YAML formatting):

* eureka.client.serviceUrl.defaultZone=<http://localhost:8010/eureka/>
* words=I,You,He,She,It
* server.port=${PORT:${SERVER\_PORT:0}} (this will cause a random, unused port to be assigned if none is specified)

11. Save a bootstrap.yml (or properties) file in the root of your classpath. Add the following key / values (use correct YAML formatting):

* spring.application.name=lab-4-subject

12. Add a Controller class

* Place it in the 'demo' package or a subpackage of your choice.
* Name the class anything you like. Annotate it with @RestController.
* Add a String member variable named “words”. Annotate it with @Value("${words}”).
* Add the following method to serve the resource (optimize this code if you like):

@RequestMapping("/")  
 public @ResponseBody String getWord() {  
 String[] wordArray = words.split(",");  
 int i = (int)Math.round(Math.random() \* (wordArray.length - 1));  
 return wordArray[i];  
 }

13. Repeat steps 7 thru 12 (copy the entire project if it is easier), except use the following values:

* Name of application: “lab-4-verb”
* spring.application.name: “lab-4-verb”
* words: “ran,knew,had,saw,bought”

14. Repeat steps 7 thru 12, except use the following values:

* Name of application: “lab-4-article”
* spring.application.name: “lab-4-article”
* words: “a,the”

15. Repeat steps 7 thru 12, except use the following values:

* Name of application: “lab-4-adjective”
* spring.application.name: “lab-4-adjective”
* words: “reasonable,leaky,suspicious,ordinary,unlikely”

16. Repeat steps 7 thru 12, except use the following values:

* Name of application: “lab-4-noun”
* spring.application.name: “lab-4-noun”
* words: “boat,book,vote,seat,backpack,partition,groundhog”

17. Create a new Spring Boot web application.

* Name the application “lab-4-sentence”, and use this value for the Artifact.
* Use JAR packaging and the latest versions of Java and Boot.
* Add actuator and web as a dependencies.
* Alter the POM (or Gradle) just as you did in step 8.

18. Add @EnableDiscoveryClient to the Application class.

19. Save an application.yml (or properties) file in the root of your classpath (src/main/resources recommended). Add the following key / values (use correct YAML formatting):

* eureka.client.serviceUrl.defaultZone=<http://localhost:8010/eureka/>
* server.port: 8020

20. Add a Controller class to assemble and return the sentence:

* Name the class anything you like. Annotate it with @RestController.
* Use @Autowired to obtain a DiscoveryClient (import from Spring Cloud).
* Add the following methods to serve the sentence based on the words obtained from the client services. (feel free to optimize / refactor this code as you like:

@RequestMapping("/sentence")  
 public @ResponseBody String getSentence() {  
 return   
 getWord("LAB-4-SUBJECT") + " "  
 + getWord("LAB-4-VERB") + " "  
 + getWord("LAB-4-ARTICLE") + " "  
 + getWord("LAB-4-ADJECTIVE") + " "  
 + getWord("LAB-4-NOUN") + "."  
 ;  
 }  
   
 public String getWord(String service) {  
 List<ServiceInstance> list = client.getInstances(service);  
 if (list != null && list.size() > 0 ) {  
 URI uri = list.get(0).getUri();  
 if (uri !=null ) {  
 return (new RestTemplate()).getForObject(uri,String.class);  
 }  
 }  
 return null;  
 }

21. Run all of the word services and sentence service. (Run within your IDE, or build JARs for each one (mvn clean package) and run from the command line (java -jar name-of-jar.jar), whichever you find easiest). (If running from STS, uncheck “Enable Live Bean support” in the run configurations). Since each service uses a separate port, they should be able to run side-by-side on the same computer. Open <http://localhost:8020/sentence> to see the completed sentence. Refresh the URL and watch the sentence change.

**BONUS - Refactor to use Spring Cloud Config Server.**

We can use Eureka together with the config server to eliminate the need for each client to be configured with the location of the Eureka server

22. Add a new file to your GitHub repository (the same repository used in the last lab) called “application.yml” (or properties). Add the following key / values (use correct YAML formatting):

* eureka.client.serviceUrl.defaultZone=<http://localhost:8010/eureka/>

23. Open the common-config-server project. This is essentially the same config server that you produced in lab 3. Alter the application.yml to point to your own github repository. Save all and run this server. (You can use it as the config server for almost all of the remaining labs in this course.)

24. Edit each client application’s application.properties file. Remove the eureka client serviceUrl defaultZone key/value. We will get this from the config server.

25. In each client project, add the following key/value to bootstrap.yml (or bootstrap.properties), using correct YAML formatting:

* spring.cloud.config.uri: [http://localhost:8001](http://localhost:8001/).

26. Add an additional dependency for spring-cloud-config-client.

27. Make sure the Eureka server is still running. Start (or restart) each client. Open <http://localhost:8020/sentence> to see the completed sentence.

28. If you like, you can experiment with moving the “words” properties to the GitHub repository so they can be served up by the config server. You’ll need to use separate profile sections within the file (yml) or files with names that match the application names (yml or properties). A single application.yml file would look something like this:

---  
spring:  
 profiles: subject  
words: I,You,He,She,It  
  
---  
spring:  
 profiles: verb  
words: ran,knew,had,saw,bought  
  
---  
spring:  
 profiles: article  
words: a,the  
  
---  
spring:  
 profiles: adjective  
words: reasonable,leaky,suspicious,ordinary,unlikely  
  
---  
spring:  
 profiles: noun  
words: boat,book,vote,seat,backpack,partition,groundhog

**BONUS - Refactor to use multiple Eureka Servers**

To make the application more fault tolerant, we can run multiple Eureka servers. Ordinarily we would run copies on different racks / data centers, but to simulate this locally do the following:

29. Stop all of the running applications.

30. Edit your computer's /etc/hosts file (c:\WINDOWS\system32\drivers\etc\hosts on Windows). Add the following lines and save your work:

# START section for Microservices with Spring Course  
127.0.0.1 eureka-primary  
127.0.0.1 eureka-secondary  
127.0.0.1 eureka-tertiary  
# END section for Microservices with Spring Course

31. Within the lab-4-server project, add application.yml with multiple profiles: primary, secondary, tertiary. The server.port value should be 8011, 8012, and 8013 respectively. The eureka.client.serviceUrl.defaultZone for each profile should point to the "eureka-\*" URLs of the other two; for example the primary value should be: <http://eureka-secondary:8012/eureka/,http://eureka-tertiary:8013/eureka/>

32. Run the application 3 times, using -Dspring.profiles.active=primary (and secondary, and tertiary) to activate the relevant profile. The result should be 3 Eureka servers which communicate with each other.

33. In your GitHub project, modify the application.properties eureka.client.serviceUrl.defaultZone to include the URIs of all three Eureka servers (comma-separated, no spaces).

34. Start all clients. Open <http://localhost:8020/sentence> to see the completed sentence.

35. To test Eureka’s fault tolerance, stop 1 or 2 of the Eureka instances. Restart 1 or 2 of the clients to ensure they have no difficulty finding Eureka. Note that it may take several seconds for the clients and servers to become fully aware of which services are up / down. Make sure the sentence still displays.

**Reflection:** There are a number of remaining issues with the current application which can be addressed.

1. These services contain duplicated code. This was done only to make the instructions straightforward. You can easily implement this system using a single ‘word’ server which selects different words based on a @Profile. (This is done in the solution)
2. What happens if one of the “word” servers is down? Right now our entire application will fail. We will improve this later when we discuss circuit breakers with Hystrix.
3. To improve performance, can we run each of the calls in parallel? We will improve this later when discussing Ribbon and Hystrix.
4. We will see an alternative to the RestTemplate when we discuss Feign.