## **Lab 5 - Using Ribbon Clients**

****Part 1, Run Config Server, Eureka, and the word servers****

1. Let's make a fresh start: stop all of the services that you may have running from previous exercises. If using an IDE you may also wish to close all of the projects that are not related to "lab-5".
2. Open the config-server and the eureka-server. Run the config-server, and run the eureka-server.
3. Create and Start 5 separate copies of the lab-4 word-server using the profiles "subject", "verb", "article", "adjective", and "noun". Rename as “lab-5 - <*application\_name*>” .There are several ways to do this, depending on your preference:

* If you wish to use Maven, open separate command prompts in the target directory and run these commands:
  + mvn spring-boot:run -Dspring.profiles.active=subject
  + mvn spring-boot:run -Dspring.profiles.active=verb
  + mvn spring-boot:run -Dspring.profiles.active=article
  + mvn spring-boot:run -Dspring.profiles.active=adjective
  + mvn spring-boot:run -Dspring.profiles.active=noun
* Or if you wish to run from directly within STS, right click on the project, Run As... / Run Configurations... . From the Spring Boot tab specify a Profile of "subject", UNCHECK JMX port / live bean support, and Run. Repeat this process (or copy the run configuration) for the profiles "verb", "article", "adjective", "noun".

1. Check the Eureka server running at [http://localhost:8010](http://localhost:8010/). Ignore any warnings about running a single instance; this is expected. Ensure that each of your 5 applications are eventually listed in the "Application" section, bearing in mind it may take a few moments for the registration process to be 100% complete.
2. Optional - If you wish, you can click on the link to the right of any of these servers. Replace the "/info" with "/" and refresh several times. You can observe the randomly generated words.

****Part 2, Modify sentence server to use Ribbon****

1. Copy and Rename the lab-4-sentence-server project to lab-5-sentence-server. Run the lab-5-sentence-server project. Refresh Eureka to see it appear in the list. Test to make sure it works by opening <http://localhost:8020/sentence>. You should see several random sentences appear. We will refactor this code to make use of Ribbon.
2. Stop the lab-5-sentence-server. Add the org.springframework.cloud / spring-cloud-starter-netflix-ribbon dependency.
3. Go to Application.java. Create a new @Bean method that instantiates and returns a new RestTemplate. The @Bean method should also be annotated with @LoadBalanced - this will associate the RestTemplate with Ribbon. Code should look something like this:

// This "LoadBalanced" RestTemplate

// is automatically hooked into Ribbon:

@Bean @LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

1. Open SentenceController.java. Replace the @Autowired DiscoveryClient with an @Autowired RestTemplate. Note that this will temporarily break the code.
2. Refactor the code in the getWord method. Use your restTemplate's getForObject method to call the given service. The first argument should be a concatenation of "http://" and the given service ID. The second argument should simply be a String.class; we want the restTemplate to yield a String containing whatever was returned to the server. The call should look like this:

return template.getForObject("http://" + service, String.class);

1. Run the project. Test it to make sure it works by opening <http://localhost:8020/sentence>. The application should work the same as it did before, though now it is using Ribbon client side load balancing.

****Multiple Clients**** At this point we have refactored the code to use Ribbon, but we haven’t really seen Ribbon’s full power as a client side load-balancer. To illustrate this we will run two copies of one of the “noun” word server with different words hard-coded. You’ll see the sentence adapt to make use of values from both servers.

1. Look at the bootstrap.yml file inside lab-5-word-server; there is an entry (shown here) that we have not seen or discussed. For the next part, we will need to run multiple copies of the noun server on the same host, and this entry will allow us to do this. There is nothing you need to change here, just understand that Eureka will be able to distinguish multiple servers of a given type running on the same host:

# Allow Eureka to recognize two apps of the same type on the same host as separate instances:

eureka:

instance:

instanceId: ${spring.cloud.client.hostname}:${spring.application.name}:${spring.application.instance\_id:${random.value}}

1. While in bootstrap.yml, note that there is another profile defined called "cold-nouns". It establishes the same application name as the 'noun' server.
2. Now look at application.yml, note that there is another set of nouns - but these nouns are all 'cold' words (there is nothing you need to change here):

---

spring:

profiles: cold-noun

words: icicle,refrigerator,blizzard,snowball

1. Start a copy of the lab-5-word-server using the "cold-noun" profile, similar to how you launched the application earlier. Eureka will see this as another instance of the 'noun' server. Because each runs on its own port, there will be no conflict. Each noun server will present a different lists of words. Both will register with Eureka, and the Ribbon load balancer in the sentence server will soon learn that both exist.
2. Return to the Eureka page running at [http://localhost:8010](http://localhost:8010/). Refresh it several times. Once registration is complete, you should see two “NOUN” services running, each with its own instance ID (this is the purpose for the setting you added a few steps back).
3. Refresh the sentence browser page at <http://localhost:8020/sentence>. Once it becomes aware of the new “NOUN” service, the loadbalancer will distribute the load between the two services, and half of the time your sentence will end with one of the “cold” nouns that you hard-coded above.
4. Stop one of the NOUN services (either one) and refresh your sentence browser page several times. You will see that it fails half the time as one of the instances is no longer available. In fact, since the default load balancer is based on a round-robin algorithm, the failure occurs every second time the noun service is used. If you continue refreshing long enough, you will see that the failures eventually stop as the ribbon client becomes updated with the revised server list from Eureka.

****Reflection:****

1. You may be wondering about the Eureka registration delay that occurs. After all, you can see from your application logs that each application registers itself with Eureka immediately. The cause results from the need to synchronize between Eureka clients and servers; they all need to have the same metadata. A 30 second heartbeat interval means that you could need up to three heartbeats for synchronization to occur. You can decrease this interval, but 30 seconds is probably fine for most production cases.
2. The registration delay also affects when you stopped the NOUN server, and you may be surprised that the Ribbon load balancer did not direct us away from the server that was clearly not available. We can address this by using different Ping, Rule, or LoadBalancer strategies. By default Ribbon relies on Eureka to provide a list of healthy servers, and we’ve seen that with Default settings Eureka can take a while to notice a server’s absence. We could use a different strategy, and also employ a rule that avoids non-functioning servers. We will discuss this more when we explore Hystrix.
3. Our application will still fail if we can’t find at least one of each kind of word server. We will improve this later when we discuss circuit breakers with Hystrix.
4. To improve performance, can we run each of the calls in parallel? We will improve this later when discussing Ribbon and Hystrix.
5. We will see an alternative to the RestTemplate when we discuss Feign.
6. Sometimes you may encounter an ominous message on the Eureka console saying "EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE." The simple explanation: Eureka expects clients to periodically renew their registrations - the default is 30 seconds. When services fail to renew Eureka removes them from the list. When a large % of services renew, Eureka assumes that the problem is with network connectivity, not the client services. In this situation it continues to list services that it has not heard from, but gives you this message to warn you that it may be wrong. When you are working on the labs you will be frequently stopping and restarting most services, and this large change triggers the warning.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*