**Lab 3 - Create a Spring Cloud Config Server and Client**

**\*\*Part 1 - Config Server:\*\***

1. Create a new Spring Boot application using Spring Starter Project tool in STS. Name the project "lab-3-server”, and use this value for the Artifact. Use Jar packaging and the latest versions of Java. Use a version of Boot > 2.0.x. No need to select any dependencies.

2. Add a dependency for group "org.springframework.cloud" and artifact "spring-cloud-config-server". You do not need to specify a version -- this is already defined in the spring-cloud-dependencies POM.

3. Edit the main Application class (probably named Lab3ServerApplication). Add the @EnableConfigServer to this class.

4. Create a new repository on GitHub to hold your application configuration data. Call the repository "ConfigData" or a name of your choosing. Note the full URI of the repository, you will need this in a following step.

5. Add a new file to your GitHub repository called "lab-3-client.yml” (or lab-3-client.properties). Add a key called "lucky-word" and a value of "Irish", "Rabbit's Foot", "Serendipity", or any other value of your choosing.

6. Back in your project, create an application.yml (or application.properties) file in the root of your classpath (src/main/resources recommended). Add the key "spring.cloud.config.server.git.uri" and the value "https://github.com/"YOUR-GITHUB-ID"/ConfigData", substituting the value for Github ID and repository name as needed.

7. Also within application.yml (or application.properties), set the “server.port” to 8001.

8. Run the application. Open the URL http://localhost:8001/lab-3-client/default/. You should see the JSON result that will actually be used by Spring. If the server is not working, review the prior steps to find the issue before moving on.

**\*\*Part 2 - Config Client:\*\***

9. Create a new, separate Spring Boot application. Use a version of Boot > 2.0.x. Name the project "lab-3-client", and use this value for the Artifact. Add the web dependency. You can make this a JAR or WAR project, but the instructions here will assume JAR.

10. Add a dependency for group "org.springframework.cloud" and artifact "spring-cloud-starter-config”. You do not need to specify a version -- this is already defined in the parent pom in the dependency management section.

11. Add a bootstrap.yml (or bootstrap.properties) file in the root of your classpath (src/main/resources recommended). Add the following key/values using the appropriate format:

```

spring.application.name=lab-3-client

spring.cloud.config.uri=http://localhost:8001

server.port=8002

```

> (Note that this file must be "bootstrap" -- not "application" -- so that it is read early in the application startup process. The server.port could be specified in either file, but the URI to the config server affects the startup sequence.)\_

12. Add a REST controller to obtain and display the lucky word:

```

@RestController

public class LuckyWordController {

@Value("${lucky-word}") String luckyWord;

@GetMapping("/lucky-word")

public String showLuckyWord() {

return "The lucky word is: " + luckyWord;

}

}

```

13. Start your client. Open http://localhost:8002/lucky-word. You should see the lucky word message in your browser.

> (Note: if you receive an error, and your server (above) is working properly, first confirm that your client is actually contacting your server. The Client's console output should contain lines similar to this:

```

... c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at : http://localhost:8001

... c.c.c.ConfigServicePropertySourceLocator : Located environment: name=lab-3-client, profiles=[default], label=null, version=d200aa331ac9945354579204ea816157251059f6, state=null

... b.c.PropertySourceBootstrapConfiguration : Located property source: CompositePropertySource {name='configService', propertySources=[MapPropertySource {name='configClient'}, ...]}

```

> If you do not see these lines, your client is NOT contacting your server. Check the items above (application name, config server URI, name / location of your config files, etc.) before proceeding.

**\*\*Profiles:\*\***

14. Create a separate file in your GitHub repository called "lab-3-client-northamerica.yml” (or .properties). Populate it with the "lucky-word" key and a different value than used in the original file.

15. Stop the client application. Modify the bootstrap file to contain a key of spring.profiles.active: northamerica. Save, and restart your client. Access the URL. Which lucky word is displayed? (You could also run the application with -Dspring.profiles.active=northamerica rather than changing the bootstrap file)

**### Reflection:**

1. Notice that the client needed some dependencies for Spring Cloud, and the URI of the Spring Cloud server, but no code.

2. What happens if the Config Server is unavailable when the “lucky word” application starts? To mitigate this possibility, it is common to run multiple instances of the config server in different racks / zones behind a load balancer.

3. What happens if we change a property after client applications have started? The server picks up the changes immediately, but the client does not. Later we will see how Spring Cloud Bus and refresh scope can be used to dynamically propagate changes.

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