(../..)

Purpose of this lab

- How to use a Ribbon enabled `RestTemplate`
- Estimated Time: 25 minutes

Start the `config-server`, `service-registry`, and `fortune-service`

1. Start the `config-server` in a terminal window. You may have terminal windows still open from previous labs. They may be reused for this lab.

```
cd config-server
mvn clean spring-boot:run
```

2. Start the `service-registry`

```
cd service-registry
mvn clean spring-boot:run
```

3. Start the `fortune-service`

```
cd fortune-service
mvn clean spring-boot:run
```

Set up `greeting-ribbon-rest`

No additions to the pom.xml

In this case, we don't need to explicitly include Ribbon support in the `pom.xml`. Ribbon support is pulled in through transitive dependencies (dependencies of the dependencies we have already defined).

1. Review the the following file: `greeting-ribbon-rest/src/main/java/io/pivotal/GreetingRibbonRestApplication.java`. In addition to the standard `@EnableDiscoveryClient` annotation, we're also configuring a `RestTemplate` bean. It is not the usual `RestTemplate`, it is load balanced by Ribbon. The `@LoadBalanced` annotation is a qualifier to ensure we get the load balanced `RestTemplate` injected. This further simplifies application code.

```
@SpringBootApplication
@EnableDiscoveryClient
public class GreetingRibbonRestApplication {

  public static void main(String[] args) {
    SpringApplication.run(GreetingRibbonRestApplication.class,
args);
  }

  @LoadBalanced
  @Bean
  RestTemplate restTemplate() {
    return new RestTemplate();
  }

}
```

2. Review the the following file: `greeting-ribbon-rest/src/main/java/io/pivotal/greeting/GreetingController.java`. Here we autowire the restTemplate we configured in the previous step. Note also that the spring cloud API is smart enough to dynamically substitute the name of the service `fortune-service` in the url parameter for `getForObject` with its load-balanced, discovered url.

```
@Controller
public class GreetingController {

  Logger logger = LoggerFactory.getLogger(GreetingController.class
);

  @Autowired
  private RestTemplate restTemplate;

  @RequestMapping("/")
  String getGreeting(Model model) {

    logger.debug("Adding greeting");
    model.addAttribute("msg", "Greetings!!!");

    String fortune = restTemplate.getForObject("http://fortune-ser
vice", String.class);

    logger.debug("Adding fortune");
    model.addAttribute("fortune", fortune);

    //resolves to the greeting.vm velocity template
    return "greeting";
  }

}
```

3. Open a new terminal window. Start the `greeting-ribbon-rest` app.

```
cd greeting-ribbon-rest
mvn clean spring-boot:run
```

4. After the a few moments, check the `service-registry` dashboard at http://localhost:8761 (http://localhost:8761). Confirm the `greeting-ribbon-rest` app is registered.

5. Browse to http://localhost:8080/ (http://localhost:8080/) to the `greeting-ribbon-rest` application. Confirm you are seeing fortunes. Refresh as desired. Also review the terminal output for the `greeting-ribbon-rest` app.

6. When done, stop the `config-server`, `service-registry`, `fortune-service` and `greeting-ribbon-rest` applications.

Deploy the `greeting-ribbon-rest` to PCF

1. Package and push the `greeting-ribbon-rest` application.

```
mvn clean package
cf push greeting-ribbon-rest -p target/greeting-ribbon-rest-0.0.1-
SNAPSHOT.jar -m 512M --random-route --no-start
```

2. Bind services for the `greeting-ribbon-rest` application.

```
cf bind-service greeting-ribbon-rest config-server
cf bind-service greeting-ribbon-rest service-registry
```

You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don't need to restage at this time.

1. Set the `TRUST_CERTS` environment variable for the `greeting-ribbon-rest` application (our PCF instance is using self-signed SSL certificates).

```
cf set-env greeting-ribbon-rest TRUST_CERTS <your api endpoint>
```

You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don't need to restage at this time.

1. Start the `greeting-ribbon-rest` app.

```
cf start greeting-ribbon-rest
```

2. After the a few moments, check the `service-registry`. Confirm the `greeting-ribbon-rest` app is registered.

3. Refresh the `greeting-ribbon-rest` `/` endpoint.

Note About This Lab

If services (e.g. `fortune-service`) are registering using the first Cloud Foundry URI (using the `route` registration method) this means that requests to them are being routed through the `router` and subsequently load balanced at that layer. Therefore, client side load balancing doesn't occur.

Pivotal Cloud Foundry has recently added support for allowing cross container communication. This will allow applications to communicate with each other without passing through the `router`. As applied to client-side load balancing, services such as `fortune-service` would register with Eureka using their container IP addresses. Allowing clients to reach them without going through the `router`. This is known as using the `direct` registration method.

For more details, please read the following (http://docs.pivotal.io/spring-cloud-services/1-2/service-registry/writing-client-applications.html#register-a-service).

(https://pivotal.io)