

NETFLIX RIBBON



Ribbon is a client-side load balancer that gives you a lot of control over the behaviour of HTTP and TCP clients. Feign already uses Ribbon, so, if you use `@FeignClient`, this section also applies.

A central concept in Ribbon is that of the named client. Each load balancer is part of an ensemble of components that work together to contact a remote server on demand, and the ensemble has a name that you give it as an application developer (for example, by using the `@FeignClient` annotation). On demand, Spring Cloud creates a new ensemble as an `ApplicationContext` for each named client by using `RibbonClientConfiguration`. This contains (amongst other things) an `LoadBalancer`, a `RestClient`, and a `ServerListFilter`.

To include Ribbon in your project, use the starter with a group ID of **org.springframework.cloud** and an artifact ID of **spring-cloud-starter-netflix-ribbon**.

Note :

```
@LoadBalanced
@Bean
public RestTemplate getTemplate() { return new RestTemplate(); }
```

A load balanced `RestTemplate` can be configured to retry failed requests. By default this logic is disabled, you can enable it by adding Spring Retry to your application's class-path. The load balanced `RestTemplate` will honour some of the Ribbon configuration values related to retrying failed requests. Multiple resources can access our micro-services so better to annotate with **@LoadBalanced**

Running Multiple Instances Of Eureka Client Simultaneously

The screenshot displays the Spring Eureka client application running in a browser and an IDE. The browser shows the Eureka dashboard with system status and registered instances. The IDE shows the application code and the Run/Debug configuration panel.

System Status

Environment	test	Current time	2022-12-13T00:13:52+0530
Data center	default	Uptime	00:19
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	12

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PAYMENT-SERVICE	n/a	(3)	UP (3) - 192.168.29.46:PAYMENT-SERVICE-8082, 192.168.29.46:PAYMENT-SERVICE-8083, 192.168.29.46:PAYMENT-SERVICE-8081

General Info

Name	Value
total-avail-memory	138mb
num-of-cpus	12
current-memory-usage	89mb (64%)

Run/Debug Configurations

Name: Config2

Run on: Local machine

Build and run

```
java 17.0x of 'payment-service' module
cob_paymentservice.payment-service.Payments
```

Active profiles:

Environment variables: server.port=882

Open run/debug tool window when started

Add dependencies with "provided" scope to classpath

Background compilation enabled

Step 1 : Select **Run/Debug Configuration**

Step 2 : Using the plus icon , select **Spring Boot** option

Step 3 : Using **Modify Options** , select **Environment variables** and add **server.port=8081** there

Step 4 : Repeat above steps for port(s) **8082** and **8083**

Load Balancing Using Spring Cloud Ribbon

The screenshot displays a development environment with three main windows:

- Eureka:** Shows instances currently registered with Eureka. The table below summarizes the data:

Application	AMIs	Availability Zones	Status
PAYMENT-SERVICE	n/a (3)	(3)	UP (3) - 192.168.29.46:PAYMENT-SERVICE-8082, 192.168.29.46:PAYMENT-SERVICE-8083, 192.168.29.46:PAYMENT-SERVICE-8081
SHOPPING-SERVICE	n/a (1)	(1)	UP (1) - 192.168.29.46:SHOPPING-SERVICE-9999

General Info

Name	Value
total-avail-memory	138mb
num-of-cpus	12
current-memory-usage	99mb (71%)
server-uptime	06:20
registered-replicas	http://localhost:8761/eureka/

Code Editor: Shows the `PaymentController.java` file with the following code:

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/payment/order/")
public class PaymentController {

    @Value("${server.port}")
    private int port;

    @GetMapping("/{payNo}/order/")
    public String payNow(@PathVariable int price) {
        return "payment with " + price + " is successful @ service port : " + port;
    }
}
```

Services: Shows a list of active services:

- Spring Boot: 2022-12-13T09:12:11.258+05:30 INFO 11687 --- [...]
- Spring Boot: 2022-12-13T09:12:11.261+05:30 INFO 11687 --- [...]
- PaymentServiceApplication: 2022-12-13T09:12:11.263+05:30 INFO 11687 --- [...]
- Config 8082: 2022-12-13T09:12:11.264+05:30 INFO 11687 --- [...]
- Config 8083: 2022-12-13T09:12:11.276+05:30 INFO 11687 --- [...]
- Config 8082: 2022-12-13T09:12:11.276+05:30 INFO 11687 --- [...]
- Config 8083: 2022-12-13T09:12:11.276+05:30 INFO 11687 --- [...]
- InfoReplicator: 2022-12-13T09:12:11.292+05:30 INFO 11687 --- [...]
- InfoReplicator: 2022-12-13T09:12:11.293+05:30 INFO 11687 --- [...]
- Info-8081-exec: 2022-12-13T09:12:16.926+05:30 INFO 11687 --- [...]
- Info-8081-exec: 2022-12-13T09:12:16.927+05:30 INFO 11687 --- [...]
- Info-8081-exec: 2022-12-13T09:12:16.928+05:30 INFO 11687 --- [...]

We can observe here that Ribbon internally handles the load for 3 instances of payment service and one instance of shopping service.

Shopping service can make use of various instances of **Payment service** randomly.

Note:

Spring Cloud also lets us take full control of the ribbon client by declaring additional configuration (on top of the `RibbonClientConfiguration`) using `@RibbonClient`