**how you can set up Spring Cloud LoadBalancer**

**1. Create a Spring Boot Application**

Use [Spring Initializr](https://start.spring.io/) to generate a project with the following dependencies:

* **Spring Web**
* **Spring Cloud LoadBalancer**
* **Spring Boot Actuator** (optional, for health checks)

**2. Add Dependencies**

Include the required dependencies in your pom.xml:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-loadbalancer</artifactId>

</dependency>

If you're using **Eureka** for service discovery, include:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

**3. Configure application.properties**

**3.1 Static Service List**

If you're not using a service registry, define a static list of service instances in your application.properties file:

# Enable Spring Cloud LoadBalancer

spring.cloud.loadbalancer.enabled=true

# Define static service instances

spring.cloud.loadbalancer.instances.my-service[0].uri=http://localhost:8081

spring.cloud.loadbalancer.instances.my-service[1].uri=http://localhost:8082

spring.cloud.loadbalancer.instances.my-service[2].uri=http://localhost:8083

**3.2 Dynamic Service Discovery (Eureka)**

If you are using Eureka, configure Eureka in application.properties:

# Enable Spring Cloud LoadBalancer

spring.cloud.loadbalancer.enabled=true

# Eureka server configuration

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

# Application name (used to discover other services)

spring.application.name=my-client-service

Spring Cloud LoadBalancer will automatically fetch service instances from Eureka.

**4. Use LoadBalancer with RestTemplate**

In your application, annotate a RestTemplate bean with @LoadBalanced to enable load balancing:

**RestTemplate Configuration**

@Configuration

public class RestTemplateConfig {

@Bean

@LoadBalanced

public RestTemplate restTemplate() {

return new RestTemplate();

}

}

**Controller Example**

@RestController

@RequestMapping("/api")

public class MyController {

@Autowired

private RestTemplate restTemplate;

@GetMapping("/call-service")

public String callService() {

String response = restTemplate.getForObject("http://my-service/endpoint", String.class);

return "Response from service: " + response;

}

}

* **http://my-service**: The service name (my-service) is resolved to one of the configured instances (static list or from Eureka).

**5. Configure Load Balancer Rules (Optional)**

To customize the load-balancing strategy, configure it in application.properties:

# Default Load Balancer Strategy

spring.cloud.loadbalancer.strategy=round-robin # Default: round-robin (alternatively, use random)

* round-robin: Cycles through available instances.
* random: Selects a random instance.

**6. Enable Retry (Optional)**

You can enable retries for failed requests:

# Enable retries

spring.cloud.loadbalancer.retry.enabled=true

spring.cloud.loadbalancer.retry.max-attempts=3

spring.cloud.loadbalancer.retry.backoff=200ms

**7. Test the Application**

1. Run multiple instances of the target service on ports 8081, 8082, and 8083 (for a static service list).
2. Run your Spring Boot application.
3. Access the endpoint, e.g., http://localhost:8080/api/call-service.
4. Observe that the requests are balanced among the available instances.