**Day4: Circuit-Breaker Pattern using Resilience4j**

**Circuit Breaker Pattern in Java Spring Boot microservices** using **Resilience4j**, which is the recommended fault tolerance library integrated with Spring Cloud.

**Goal of This Lab:**

* Build two microservices: order-service and product-service
* Use Resilience4j Circuit Breaker in order-service to protect it from product-service failure
* Simulate a failure and observe circuit breaker behavior

**Prerequisites**

* Java 17+
* Maven
* IntelliJ IDEA or any IDE
* Spring Boot (3.x recommended)
* Postman or browser to test

**Step 1: Create product-service**

**🔹 Generate using Spring Initializr:**

* Dependencies: Spring Web, Spring Boot Actuator

**🔹 Create Controller:**

@RestController

@RequestMapping("/api/products")

public class ProductController {

@GetMapping("/{id}")

public ResponseEntity<String> getProduct(@PathVariable String id) {

if ("fail".equalsIgnoreCase(id)) {

throw new RuntimeException("Product service is down!");

}

return ResponseEntity.ok("Product ID: " + id);

}

}

**🔹 Run on Port 8081**

In application.properties:

server.port=8081

**Step 2: Create order-service**

**🔹 Generate using Spring Initializr:**

* Dependencies: Spring Web, Spring Boot Actuator, Resilience4j, Spring Cloud CircuitBreaker, Spring Boot DevTools

**🔹 Add RestTemplate Bean:**

@Configuration

public class AppConfig {

@Bean

public RestTemplate restTemplate() {

return new RestTemplate();

}

}

**🔹 Create Service with Circuit Breaker:**

@Service

public class OrderService {

@Autowired

private RestTemplate restTemplate;

@CircuitBreaker(name = "productServiceBreaker", fallbackMethod = "fallbackProduct")

public String getProductDetails(String id) {

return restTemplate.getForObject("http://localhost:8081/api/products/" + id, String.class);

}

public String fallbackProduct(String id, Throwable t) {

return "Product Service is currently unavailable. Please try again later.";

}

}

**🔹 Create Controller:**

@RestController

@RequestMapping("/api/orders")

public class OrderController {

@Autowired

private OrderService orderService;

@GetMapping("/{productId}")

public ResponseEntity<String> getOrderDetails(@PathVariable String productId) {

String response = orderService.getProductDetails(productId);

return ResponseEntity.ok("Order created with: " + response);

}

}

**🔹 Run on Port 8080**

server.port=8080

**Step 3: Configure Resilience4j Circuit Breaker**

Add to application.yml:

resilience4j.circuitbreaker:

instances:

productServiceBreaker:

registerHealthIndicator: true

slidingWindowSize: 5

permittedNumberOfCallsInHalfOpenState: 2

minimumNumberOfCalls: 5

waitDurationInOpenState: 10s

failureRateThreshold: 50

**Step 4: Run and Test**

**Normal Test:**

Call:

GET http://localhost:8080/api/orders/123

Response should be:

Order created with: Product ID: 123

**❌ Trigger Failure:**

Call:

GET http://localhost:8080/api/orders/fail

After 5 failed calls (based on config), circuit breaker opens.

**🛑 While Open:**

Subsequent calls immediately trigger fallback method.

Wait 10 seconds (open state timeout), and then test again.

**Summary of What You Built**

| **Component** | **Role** |
| --- | --- |
| product-service | Simulates a product backend |
| order-service | Depends on product-service |
| Resilience4j | Applies circuit breaker pattern |
| Fallback method | Handles failures gracefully |