Let’s talk about **Resilience4j Retry** — a very handy mechanism for **automatically retrying a failed call** a certain number of times before giving up and possibly triggering a fallback.

👉 Think of it like this:

“If the service call fails, try again up to 3 times with some delay before returning an error.”

**🧰 1️⃣ Add Dependency (if not added)**

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot3</artifactId>

</dependency>

**⚙️ 2️⃣ Configure in application.yml**

Here’s a **Retry config for currencyConversionService**:

resilience4j:

retry:

instances:

currencyConversionService:

maxAttempts: 3 # 🔁 Total attempts = initial + 2 retries

waitDuration: 2s # ⏱ Wait time between retries

enableExponentialBackoff: true # 📈 Optional: exponential increase in wait time

exponentialBackoffMultiplier: 2 # Wait doubles each time: 2s → 4s → 8s

retryExceptions:

- java.io.IOException

- org.springframework.web.client.HttpServerErrorException

ignoreExceptions:

- java.lang.NullPointerException

**📝 Config Explanation**

| **Property** | **Description** |
| --- | --- |
| maxAttempts | Total number of attempts = initial + retries |
| waitDuration | Time to wait between retries |
| enableExponentialBackoff | If true, wait time grows exponentially on each retry |
| exponentialBackoffMultiplier | Factor by which the wait time increases after each retry |
| retryExceptions | Which exceptions should **trigger retry** |
| ignoreExceptions | Exceptions which should **NOT trigger retry** (they fail immediately) |

**🧠 3️⃣ Annotate Your Method**

You can use @Retry on any method.  
For example, in a **Currency Conversion Service** calling the **Currency Exchange Service**:

import io.github.resilience4j.retry.annotation.Retry;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.client.RestTemplate;

@RestController

public class CurrencyConversionController {

private final RestTemplate restTemplate = new RestTemplate();

@GetMapping("/convert")

@Retry(name = "currencyConversionService", fallbackMethod = "retryFallback")

@CircuitBreaker(name = "currencyConversionService", fallbackMethod = "retryFallback")

public String convertCurrency() {

System.out.println("👉 Trying to call currency-exchange-service...");

String response = restTemplate.getForObject("http://localhost:8000/exchange", String.class);

return "Conversion Result: " + response;

}

public String retryFallback(Throwable t) {

return "⚠ Exchange service unavailable after retries. Please try again later.";

}

}

**🧠 How it works:**

* If the first call fails → Resilience4j waits 2s → retries again.
* It will retry up to maxAttempts.
* If still failing → fallback method executes.

**🧪 4️⃣ Test It**

1. Stop the currency-exchange service (to simulate failure).
2. Hit the endpoint:

GET http://localhost:8100/convert

✅ You’ll see in logs:

👉 Trying to call currency-exchange-service...

👉 Trying to call currency-exchange-service...

👉 Trying to call currency-exchange-service...

Then after retries → fallback response.

**🧠 5️⃣ Combine with CircuitBreaker**

**Retry + CircuitBreaker** is a common pattern:

* **Retry** handles **transient errors** (like network glitches).
* **CircuitBreaker** prevents repeated failures from overloading the system.

@Retry(name = "currencyConversionService", fallbackMethod = "fallback")

@CircuitBreaker(name = "currencyConversionService", fallbackMethod = "fallback")

public String callExchangeService() {

return restTemplate.getForObject("http://localhost:8000/exchange", String.class);

}

**📊 6️⃣ Monitor via Actuator**

If you have Actuator enabled, you can check retry events:

GET http://localhost:8100/actuator/retries

GET http://localhost:8100/actuator/retry-events

You’ll see retry attempts, successes, and failures.

**✅ Summary Table**

| **Feature** | **Retry 🔁** |
| --- | --- |
| **Main Goal** | Automatically retry failed calls |
| **When to Use** | Transient failures (temporary network issues, timeouts) |
| **Key Props** | maxAttempts, waitDuration, exceptions |
| **Fallback** | After retries are exhausted |
| **Best with** | CircuitBreaker + TimeLimiter |

**🧠 Real-World Example:**

In a currency conversion microservice:

* Retry the call to the exchange service 3 times if it fails due to network issues.
* Use TimeLimiter to not hang too long.
* Use CircuitBreaker to stop retrying if the downstream service is completely down.

👉 This gives you **resilient, production-grade fault tolerance** 🚀