The **Bulkhead pattern** in Resilience4j is used to **limit the number of concurrent calls** to a particular service or method, preventing a single overloaded dependency from exhausting all system resources.

Resilience4j provides **two types** of bulkheads:

1. 🧵 **Semaphore-based bulkhead** → limits concurrent calls using semaphores (lightweight, most common).
2. 🧱 **ThreadPool-based bulkhead** → runs calls in a separate thread pool (useful for isolating blocking calls).

## 🧰 1️⃣ ****Add Dependency****

If not already added:

<dependency>

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

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

</dependency>

(Use resilience4j-spring-boot2 if you're on Spring Boot 2.x)

## ⚙️ 2️⃣ ****Configure in**** application.yml

### 🔸 ****Semaphore-based Bulkhead**** (most common)

resilience4j:

bulkhead:

instances:

currencyExchangeService:

maxConcurrentCalls: 3 # Max concurrent calls allowed

maxWaitDuration: 5s # How long to wait for a permit if max reached

writableStackTraceEnabled: true

#### 📝 Explanation:

| **Property** | **Description** |
| --- | --- |
| maxConcurrentCalls | Number of concurrent calls allowed at the same time. |
| maxWaitDuration | How long to wait to acquire a permit if limit is reached. |
| writableStackTraceEnabled | If false, stack traces are not written (optimization). |

👉 Example: If maxConcurrentCalls = 3 and 4th request comes while all 3 are busy → it will wait up to maxWaitDuration. If no permit becomes available in time → it triggers fallback.

## 🧠 3️⃣ ****Annotate Your Method****

Example in a CurrencyConversionController:

import io.github.resilience4j.bulkhead.annotation.Bulkhead;

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

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

@RestController

public class CurrencyConversionController {

@GetMapping("/convert")

@Bulkhead(name = "currencyExchangeService", fallbackMethod = "bulkheadFallback")

public String convertCurrency() throws InterruptedException {

// Simulate long processing

Thread.sleep(3000);

return "Currency conversion completed at " + System.currentTimeMillis();

}

public String bulkheadFallback(Throwable t) {

return "Too many concurrent requests — please try again later";

}

}

### 🧪 Test it:

* Make **4+ simultaneous calls** (e.g., using Postman runner, Apache Bench, or multiple browser tabs).
* First 3 will be processed. Others will fallback if maxWaitDuration expires.

## 🧵 4️⃣ ****ThreadPool Bulkhead (Advanced)****

Useful if your method is **blocking** and you want to run it in a separate thread pool:

resilience4j:

thread-pool-bulkhead:

instances:

currencyExchangeService:

coreThreadPoolSize: 5

maxThreadPoolSize: 10

queueCapacity: 20

And annotate using:

import io.github.resilience4j.bulkhead.annotation.ThreadPoolBulkhead;

@ThreadPoolBulkhead(name = "currencyExchangeService", fallbackMethod = "bulkheadFallback")

public String convertCurrency() {

return "Processed in separate thread pool";

}

📝 With ThreadPoolBulkhead:

* Requests are queued and processed in a separate thread pool.
* Protects the **main request threads** from being blocked by slow downstream calls.

## 📊 5️⃣ ****Monitor via Actuator****

If actuator is enabled:

GET http://localhost:8080/actuator/bulkheads

GET http://localhost:8080/actuator/bulkhead-events

You'll see metrics like how many calls are permitted, rejected, etc.

## ✅ Summary Table

| **Type** | **Use Case** |
| --- | --- |
| **Semaphore Bulkhead** | Limits concurrent access using permits. Good for **non-blocking** calls. |
| **ThreadPool Bulkhead** | Runs calls in isolated thread pool. Good for **blocking** operations. |

## 🌟 Combine with CircuitBreaker

You can stack annotations like:

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

@Bulkhead(name = "currencyExchangeService")

@GetMapping("/convert")

public String convertCurrency() { ... }

👉 Resilience4j will apply **Bulkhead first**, then **CircuitBreaker**, depending on the order of annotations.

Would you like me to give a **real microservice example** (e.g., currency-conversion calling currency-exchange via Feign/RestTemplate) with **Bulkhead + CircuitBreaker + RateLimiter** applied together?