

07/10/2023

Version 2

# SPRING CLOUD

JÉRÉMY PERROUULT



# DISJONCTEUR

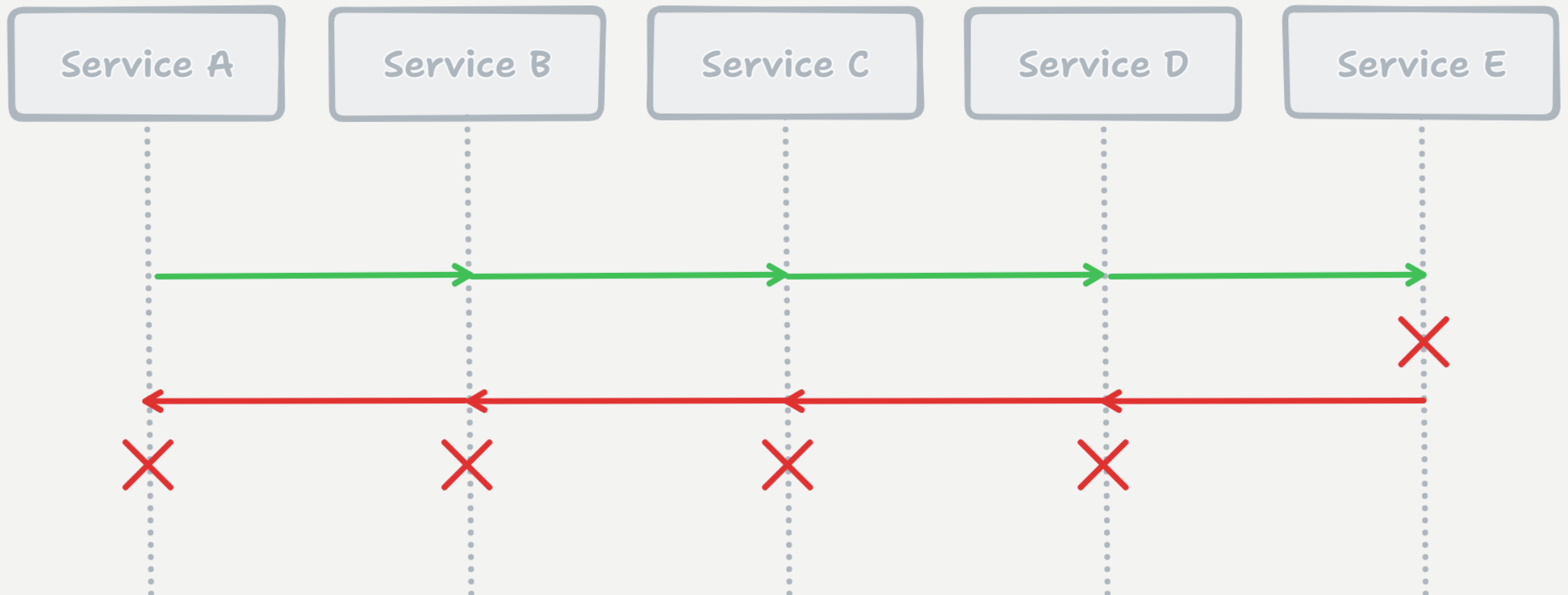
SERVICE RÉSILIENT

# DISJONCTEUR

- « Service A » a besoin de « Service B » pour composer une réponse
- « Service B » met plus de temps qu'habituellement pour répondre
  - Doit-on continuer à consommer ce service ? Sachant que son délai de réponse alonge celui de « A »
  - Doit-on temporiser sa consommation ? Et mettre en place un mécanisme de contournement ?
- Si « Service B » répond en 250ms au lieu de 10ms habituellement
  - Si on continue de l'appeler, le service risque de saturer encore plus ...
  - ... Il vaut donc mieux ne pas surcharger les appels, et le laisser reprendre ses capacités
  - Et de répondre avec un traitement alternatif en attendant
- Il nous faut des services résilients

# DISJONCTEUR

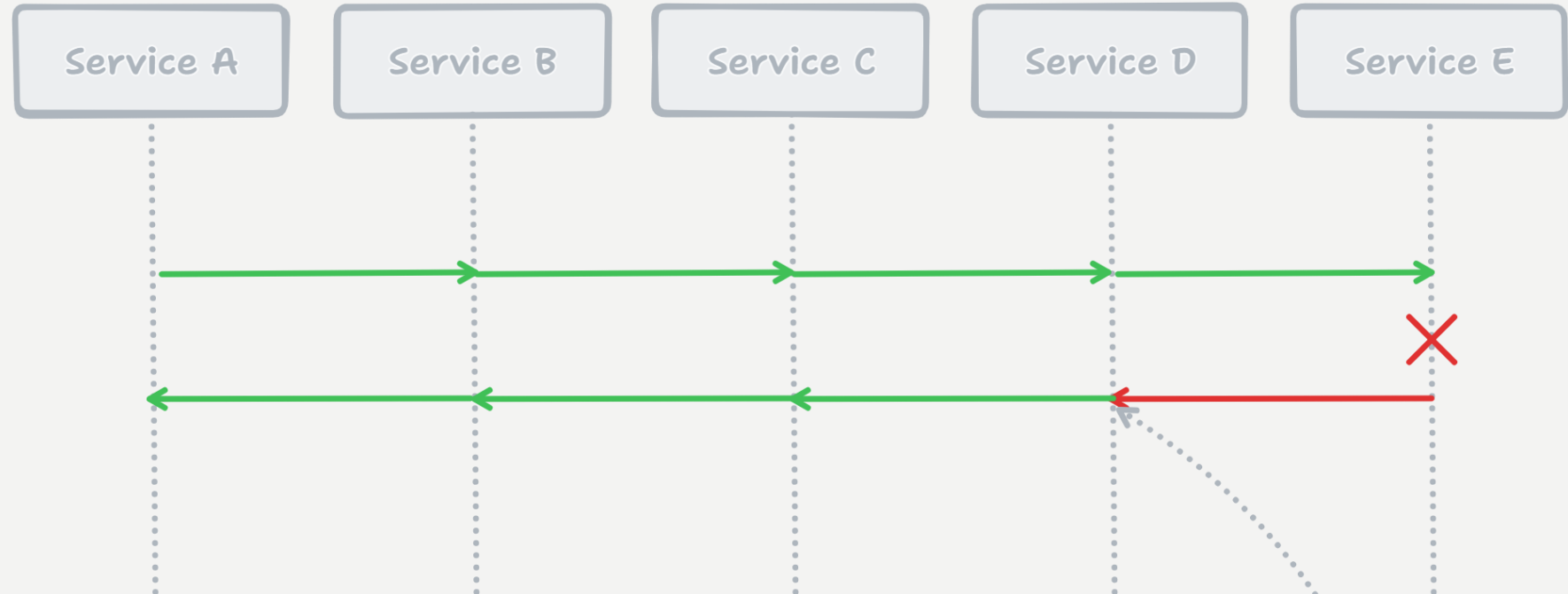
- Reprendre le scénario précédent, avec plus de services ...
  - Le service en erreur, finira par mettre en erreur toute la chaîne d'appel (effet domino)



# DISJONCTEUR

- Utilisation du Pattern **Circuit Breaker**
  - L'idée, c'est de fixer des règles pour lesquelles un service est « indisponible »
    - Temps de latence trop long
    - Nombre d'erreurs atteint
  - Si une des règles est détectée, mise en place d'une réponse alternative (**Fallback**)
    - Cette solution de repli doit être rapide et toujours disponible
    - Cette solution a une durée, c'est la durée pendant laquelle le circuit est « ouvert »

# DISJONCTEUR

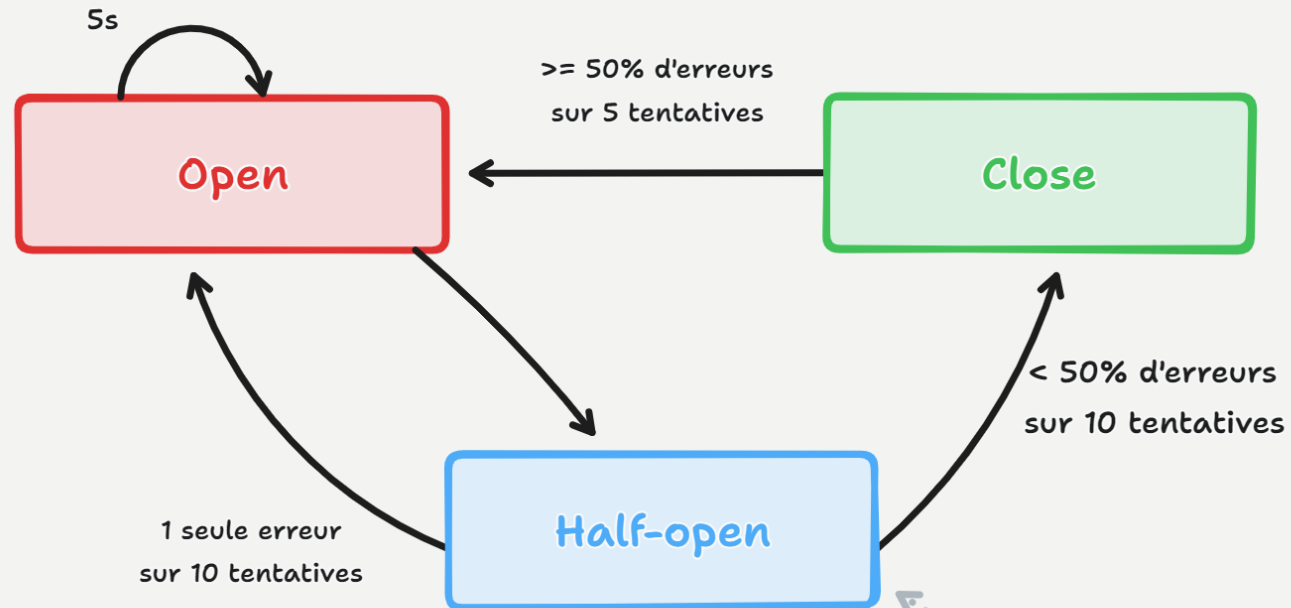


Solution  
alternative ici :  
un contenu par  
défaut par  
exemple

# DISJONCTEUR

Le circuit est ouvert après la 6ème tentatives, ayant eu 3 tentatives infructueuses. Cet état sera actif pendant 5s avant de basculer en état "semi-ouvert", puis en état "fermé".

Les erreurs peuvent être des temps de latence > durée fixée, ou un statut HTTP 5xx ou 408



Dans cet état, les requêtes sont de nouveaux exécutées. Si une seule erreur est détectée, retour dans l'état "ouvert".

# DISJONCTEUR

- En complément, il est possible d'utiliser le Pattern **Bulkhead**
  - Permet de limiter les accès concurrentiels au service
- Les deux sont complémentaires
  - **Circuit Breaker** permet de contourner un problème (implémenté côté appelant)
  - **Bulkhead** permet de limiter les appels simultanés pour alléger la charge (implémenté côté appelé)
- Lorsque le Pattern **Bulkhead** est implémenté, un nombre d'appels simultanés trop important peut faire augmenter le taux d'échec, et le **Circuit Breaker** entrera plus rapidement en état « ouvert »



# DISJONCTEUR

- Il y a deux façons d'implémenter le Pattern **Bulkhead**
  - Semaphore
    - On limite le nombre de requêtes simultanées vers le service, on rejette les requêtes si on dépasse le seuil
  - `FixedThreadPoolBulkhead`
    - On utilise un pool de Thread pour le service, ainsi qu'une file d'attente (Queue)
    - Si le pool de Threads et la liste d'attente sont pleins, on rejette les requêtes suivantes

# DISJONCTEUR

- Utilisation d'une technologie existante ...
  - **Hystrix (JAVA Netflix)**
  - **Resilience4j (JAVA)**
  - **Polly (DOTNET)**
  - ...

# DISJONCTEUR

- Mettre en place un coupe circuit sur les services concernés



# SPRING BOOT

EXEMPLE D'IMPLÉMENTATIONS

# SPRING BOOT – CIRCUIT BREAKER

- Ajouter la dépendance *spring-cloud-starter-circuitbreaker-resilience4j*

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>  
</dependency>
```

# SPRING BOOT – CIRCUIT BREAKER

- Utiliser cette Factory pour paramétrer un **Fallback**

```
String result = this.circuitBreakerFactory.create("nomService").run(  
    () -> restTemplate.getForObject("...", String.class),  
    t -> "défaut"  
);
```

# SPRING BOOT – CIRCUIT BREAKER

- Injecter `CircuitBreakerFactory` à l'endroit nécessaire
- Utiliser cette Factory pour paramétrer un **Fallback**
  - Exemple avec `RestTemplate`

```
String result = this.circuitBreakerFactory.create("nomService").run(  
    () -> this.restTemplate.getForObject("...", String.class),  
    t -> "défaut"  
);
```

- Exemple avec **OpenFeign**

```
String result = this.circuitBreakerFactory.create("nomService").run(  
    () -> this.serviceClient.getForObject("...", String.class),  
    t -> "défaut"  
);
```

# SPRING BOOT – CIRCUIT BREAKER

- Dans le cas où **OpenFeign** est utilisé, il est possible de créer une classe de `Fallback`

```
@FeignClient(value = "nom-service", fallback = ServiceClient.Fallback.class)
public interface ServiceClient {
    @GetMapping("/api/endpoint/by-produit-id/{produitId}")
    public List<String> uneRequete(@PathVariable String produitId);

    @Component
    public static class Fallback implements ServiceClient {
        @Override
        public List<String> uneRequete(String produitId) {
            return List.of("valeur par défaut");
        }
    }
}
```

- Puis, dans la configuration

```
spring.cloud.openfeign.circuitbreaker.enabled = true
spring.cloud.openfeign.circuitbreaker.alphanumeric-ids.enabled = true
```



# SPRING BOOT – CIRCUIT BREAKER

**NOTE** `resilience4j.circuitbreaker.configs.default.*`  
Pour une configuration générale

- Adapter les paramètres par défaut dans la configuration

```
resilience4j.circuitbreaker.instances.nomService.failure-rate-threshold = 50
resilience4j.circuitbreaker.instances.nomService.minimum-number-of-calls = 3
resilience4j.circuitbreaker.instances.nomService.permitted-number-of-calls-in-half-open-state = 10
resilience4j.circuitbreaker.instances.nomService.wait-duration-in-open-state = 5s
resilience4j.circuitbreaker.instances.nomService.sliding-window-size = 5
resilience4j.circuitbreaker.instances.nomService.sliding-window-type = count-based
resilience4j.circuitbreaker.instances.nomService.slow-call-duration-threshold = 1s
resilience4j.circuitbreaker.instances.nomService.slow-call-rate-threshold = 50
resilience4j.circuitbreaker.instances.nomService.record-exceptions = java.util.concurrent.TimeoutException
```

- |  |   |
|--|---|
| • failure-rate-threshold                       | Seuil d'échecs, en %                            |
| • minimum-number-of-calls                      | Nombre d'appels minimum                         |
| • permitted-number-of-calls-in-half-open-state | Nombre d'appels en état semi-ouvert             |
| • wait-duration-in-open-state                  | Durée d'attente en état ouvert                  |
| • sliding-window-size                          | Nombre d'enregistrements ou durée en état fermé |
| • sliding-window-type                          | Type (nombre ou temps)                          |
| • slow-call-duration-threshold                 | Seuil de temps de réponse                       |
| • slow-call-rate-threshold                     | Seuil au-delà duquel un temps est lent, en %    |

# SPRING BOOT – BULKHEAD

- Ajouter la dépendance *resilience4j-spring-boot3*

```
<dependency>  
  <groupId>io.github.resilience4j</groupId>  
  <artifactId>resilience4j-spring-boot3</artifactId>  
</dependency>
```

# SPRING BOOT – BULKHEAD

- Utiliser cette `@Bulkhead` et paramétrer un **Fallback**

```
@ResponseStatus(HttpStatus.TOO_MANY_REQUESTS)
public List<ProduitResponse> bulkheadFallback(BulkheadFullException e) {
    return new ArrayList<>();
}
```

```
@GetMapping
@Bulkhead(name = "produitsService", fallbackMethod = "bulkheadFallback")
public List<ProduitResponse> findAll() {
    // ...
}
```

# SPRING BOOT – BULKHEAD

- Configurer **Bulkhead**

```
resilience4j.bulkhead.instances.produitsService.max-wait-duration = 1  
resilience4j.bulkhead.instances.produitsService.max-concurrent-calls = 10
```

```
resilience4j.thread-pool-bulkhead.instances.produitsService.max-thread-pool-size = 10  
resilience4j.thread-pool-bulkhead.instances.produitsService.core-thread-pool-size = 10  
resilience4j.thread-pool-bulkhead.instances.produitsService.queue-capacity = 5
```

- |                         |                                      |
|-------------------------|--------------------------------------|
| • max-wait-duration     | Durée d'attente maximum              |
| • max-concurrent-calls  | Nombre d'appels simultanés autorisés |
| • max-thread-pool-size  | Taille du pool de thread maximum     |
| • core-thread-pool-size | Taille du pool de thread principal   |
| • queue-capacity        | Taille de la queue maximum           |