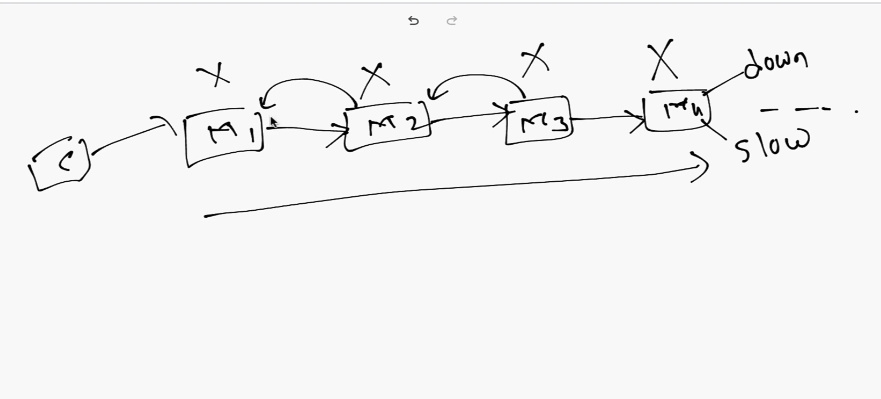
Fie ca avem 4 microservices, si M1 il cheama pe M2 si M2 pe M3 si M3 pe M4



Asa ceva poate duce la urmatoarele probleme:

- Daca microservice4 a cazut, cand 3 va face un call la el, va primi eroare, insa el va face iar si iar requesturi, si asta poate face si ca 3 sa cada, si deci si 2 poate sa cada i 1 si toate deci

- Daca 4 functioneaza foarte incet, atunci si 3 va trebui sa astepte mult si 2 si 1 si clientul

**Circuit breaker and Retry mechanim**

* **Circuit breaker** – pattern care se asigura ca daca un microservice1 il apeleaza pe 2, si 2 e down, 1 nu il va apela iar si iar pe 2, fara sens, ci va limita numarul de calluri pe care un microservice il poate face catre altele
* **Retry mechanism** – un microservice1 il apeleaza pe 2, si daca 2 e cazut sau nu raspunde, il poate apela de un numar stabilit de ori, de ex de 5 ori, intr-un anumit interval de timp, si asa se evita apelarea de foarte multe ori fara sens.

**Circuit breaker**

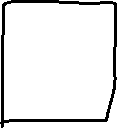
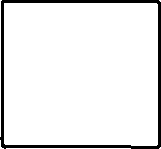
* Circuit breaker mentine 3 states:

- **open** – trece din closed in open cand o rata de succes nu e atinsa si apoi asteapta o perioada de timp cat e in open

- **closed** – permite requesturile liber

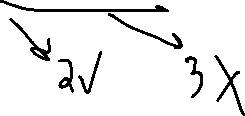
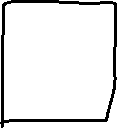
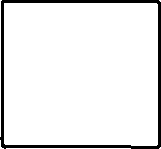
- **half open** – trece din open peste o perioada setata de timp si limiteaza numarul de calluri ce mai pot fi facute catre alt microservice.

* **closed**:



Avem microservice1 si 2, si circuit breaker e implementant in M1. Deci, M1 poate face liber un call catre M2 cat circuit breaker e in closed state

* **open**

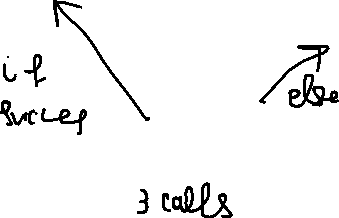
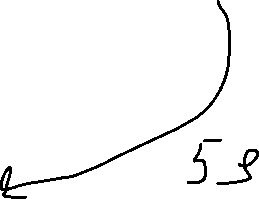
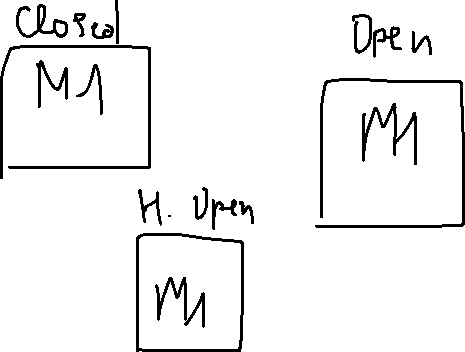
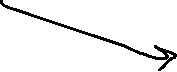
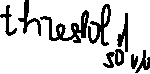


Sa zicem ca M1 e in closed state. Insa, de data asta setam un threshold(prag), de ex de 50%, ceea ce inseamna ca daca mai mult de 50% din calluri catre m2 nu reusesc, adica dau fail, atunci M1 trece in open state,daca reusesc, ramane in closed in acest timp. Mai sus, 2 au trecut, dar 3 au dat fail, deci M1 trece in open state si va trebui sa astepte o perioada de timp setata

* **Half-open**

sa zicem ca M1 este in open state. Daca se seteaza un timp, gen de 5 secunde, atunci peste 5 secunde circuit breaker din M1 trece din open in half in open state, si in aceste 5 secunde nu se face niciun request la M2. Half open state limiteaza numarul de requests pe care un microservice le poate face catre altul. De ex, daca numarul maxim e setat la 3, M1 va putea face doar 3 calluri catre M2 in half-open

- Acum, daca facand 3 calluri, M1 nu primeste niciun raspuns de la M2, el trece inapoi in Open state, iar daca primeste, in closed



si tot asa mai departe

* In asa mode, circuit breaker limiteaza numarul de calluri pe care M1 le poate face catre M2, si se asigura sa le faca iar peste un anumit itnerval de timp.

**Implementing Circuit Breaker**

* Nu il punem in acele services care nu folosesc altele. Employee apeleaza Department service, dar Department service nu apeleaza Employee, deci in Department nu e sens de pus acest sistem de la circuit breaker.
* Imp,ementarea lui Circuit Breaker se face cu **Resilience4J**

1. **Adaugam dependentele in service necesare, adica in Employee:**

- Resilience4J

- Actutors

- Spring AOP

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

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>

Endpoint va avea nevoie de AOP pentru a aduna anumite informatii

1. **Folosim @CircuitBreaker(atentie sa nu fie de la retry module!) in metodele din Service ce apeleaza un service extern**

Putem sa setam o metoda care sa returneze un obiect in caz ca metoda ce apeleaza alt service nu poate primi raspuns de la acel service. Pentru asta folosim method=””

Avem asa metoda in serviciu:

@Override  
@CircuitBreaker(name = "EMPLOYEE-SERVICE",fallbackMethod = "getDefaultResponse")  
public APIResponseDto getEmployeeById(Long id) {  
 Employee employee = employeeRepository.findById(id)  
 .orElseThrow(() ->  
 new ResourceNotFoundException("User","id",String.*valueOf*(id)));  
 DepartmentDto departmentDto = departmentFeignClient.getDepartmentByDepartmentCode(employee.getDepartmentCode());  
  
 return new APIResponseDto(employeeMapper.employeeToEmployeeDto(employee),departmentDto);  
}

public APIResponseDto getDefaultResponse(Long id, Throwable exception){  
 Employee employee = employeeRepository.findById(id)  
 .orElseThrow(() ->  
 new ResourceNotFoundException("User","id",String.*valueOf*(id)));  
 DepartmentDto departmentDto = DepartmentDto.*builder*()  
 .departmentCode("XXX")  
 .departmentName("Nonexistent")  
 .departmentDescription("No department existent")  
 .build();  
  
 return new APIResponseDto(employeeMapper.employeeToEmployeeDto(employee),departmentDto);  
}

in getDefaultResponse punem si un parametru de tip Throwable, si atentie sa fie identic tipul de return si primul parametru

1. **adaugam proprietati in .properties**

management.health.circuitbreakers.enabled=true  
management.endpoint.health.show-details=*always*resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.register-health-indicator=true  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.failure-rate-threshold=50  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.wait-duration-in-open-state=5s  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.permitted-number-of-calls-in-half-open-state=5  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.minimum-number-of-calls=3  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.automatic-transition-from-open-to-half-open-enabled=true  
resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.sliding-window-size=4

**management.health.circuitbreakers.enabled = true** – activeaza circuitbreaker in healt, sa fie aratat

**resilience4j.circuibreakr.instances.NUMELE CE E PUS IN ANOTATIA @CircuitBreaker(name=””)**

**register-health-indicator** – activeaza health indicator pentru circuit breaker

**failure-rate-treshold** – in % arata cate procente din calluri trebuie sa cada ca sa treaca din CLOSED state in OPEN

**wait-duration-in-open-state** – timpul cat va sta in open state

**permitted-number-of-calls-in-half-open-state** – numarul de calluri permise in half open state

**minimum-number-of-calls** – numarul minim de calluri ca in CLOSED state sa calculeze rata de esuare, adica threshold sau in HALF OPEN ca sa treaca in alta

**automatic-transition-from-open-to-half-open-enabled**

**sliding-window-size –** numarul arata care calluri ultimele vor fi luate pentru a calcula failure rate din CLOSED state

[CircuitBreaker (readme.io)](https://resilience4j.readme.io/docs/circuitbreaker)

1. **Testam prin a inchide Department service**

Vedem ca returneaza ce am setat in metoda de defaultResponse.

1. **Apelam** [**localhost:8081/actuator/health**](http://localhost:8081/actuator/health)

Si vedem si asta:

"circuitBreakers": **{**

"status": "UNKNOWN",

"details": **{**"EMPLOYEE-SERVICE": **{**

"status": "CIRCUIT\_HALF\_OPEN",

"details": **{**

"failureRate": "1.0%",

"failureRateThreshold": "50.0%",

"slowCallRate": "1.0%",

"slowCallRateThreshold": "100.0%",

"bufferedCalls": **0**,

"slowCalls": **0**,

"slowFailedCalls": **0**,

"failedCalls": **0**,

"notPermittedCalls": **0**,

"state": "HALF\_OPEN"**}**

**}**

**}**

**}**,

Ea apare datorita la

resilience4j.circuitbreaker.instances.EMPLOYEE-SERVICE.register-health-indicator=true

**Facand diferite calluri, cat e Dpeartment inchis, vom vedeam cum state se modifica.**

* Odata ce am facut 3 requesturi cu fail, el a trecut in OPEN state
* Peste 5 secunde e in HALP OPEN state
* Apoi daca mai facem 3(desi e 5, am pus minim 3 ce afectaza si HALF OPEN) si iar esueaza, iar e in OPEN

**Cand se trece dintr-o stare in alta, valorile salvate mereu se reseteaza! Gen calls, failedCalls etc.**

**Cu ce ne-a ajutat**

- Circuit Breaker ne-a ajutat cu optimizarea. De ex, cat era in CLOSED state, el facea requesturi la Department service care era inchis, insa pana la urma a trecut in OPENED. In Opened cat era, nici un request nu se mai ducea la Department service, se returna valoarea default pusa in metoda. Apoi, half open a permis cateva calluri ca sa il puna din nou in OPEN si iar sa nu apeleze pentru 5 secunde deloc Department service.

- In plus, Emloyee la sigur nu va cadea, caci Circuit Breaker se ocupa de asta si pana la urma returneaza din metoda default pusa in anotatie.

**Retry Pattern – Resilience4J**

* Circuit breaker doar limiteaza numarul de calluri ce poate fi facut si se asigura ca un service sa nu cada din cauza ca altul e deja cazut
* **Retry Pattern** – face mai multe calluri daca vede ca un service este down
* **Pasi de dezvoltare:**

1. **Punem anotatia @Retry deasupra la metoda ce apeleaza alt service si stergem @CircuitBreaker daca e prezent. La fel are 2 atribute: name si fallbackMethod**

@Override  
@Retry(name = "EMPLOYEE-SERVICE", fallbackMethod = "getDefaultResponse")  
public APIResponseDto getEmployeeById(Long id) {  
 Employee employee = employeeRepository.findById(id)  
 .orElseThrow(() ->  
 new ResourceNotFoundException("User","id",String.*valueOf*(id)));  
 DepartmentDto departmentDto = departmentFeignClient.getDepartmentByDepartmentCode(employee.getDepartmentCode());  
  
 return new APIResponseDto(employeeMapper.employeeToEmployeeDto(employee),departmentDto);  
}  
public APIResponseDto getDefaultResponse(Long id,Throwable throwable){  
 Employee employee = employeeRepository.findById(id)  
 .orElseThrow(() ->  
 new ResourceNotFoundException("User","id",String.*valueOf*(id)));  
 DepartmentDto departmentDto = DepartmentDto.*builder*()  
 .departmentCode("XXX")  
 .departmentName("Nonexistent")  
 .departmentDescription("No department existent")  
 .build();  
  
 return new APIResponseDto(employeeMapper.employeeToEmployeeDto(employee),departmentDto);  
}

1. **Adaugam properties:**

Ele seamana mult cu cele de la circuit breaker

resilience4j.retry.instances.EMPLOYEE-SERVICE.max-attempts=5  
resilience4j.retry.instances.EMPLOYEE-SERVICE.wait-duration=1s

[Retry (readme.io)](https://resilience4j.readme.io/docs/retry)

**maxRetryAttempts** – numarul maxim de incercari ce se vor face asupra la acel service

**waitDuration** – intervalul de timp intre fiecare incercare

Acum daca vom opri department service, si vom face acest call, vom vedea ca metoda deasupra la care sta anotatia @Retry va fi executata de 5 ori, fiecare call fiind facut in interval de 1s, si daca nici al 5 nu reuseste, se returneaza raspunsul default.