
TP N°6 :

**Résilience des microservices avec
Spring Cloud : Hystrix**

1. Prérequis

- JDK.17
- Connexion internet

2. Objectifs

1. Mise en œuvre du framework **Hystrix** :
 - a. **@EnableCircuitBreaker**
 - b. **HystrixCommand**
2. Design pattern Fallback processing : **fallbackMethod**
3. Exploitation du tableau de bord de Hystrix : **@EnableHystrixDashboard**

3. Use case à étudier

En se basant sur le TP 2 : Communication entre les micorservices pour la gestion des employés:

WebApp frontal et API back end, on va simuler que la partie back end déclenche un timeout et

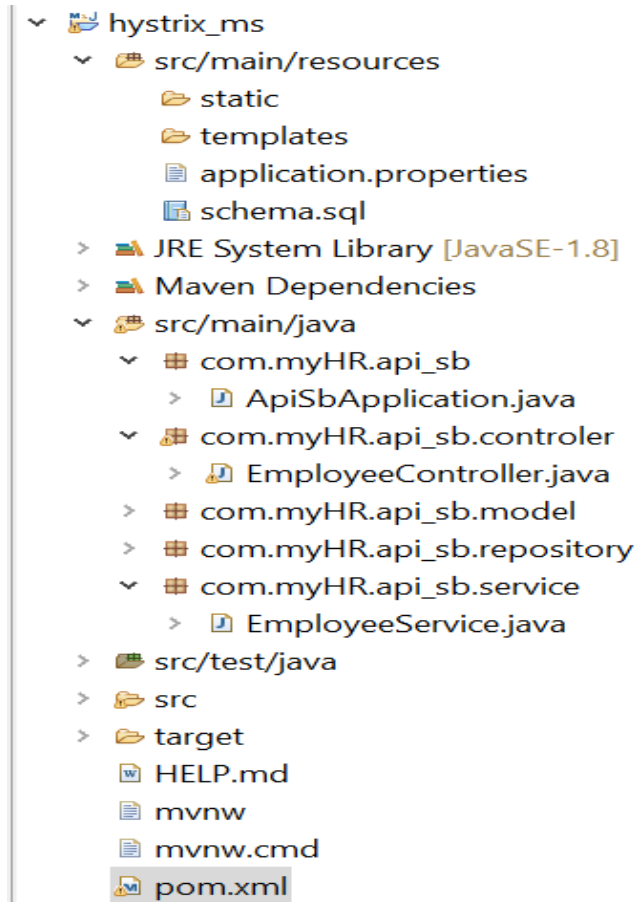
vérifier que le mécanisme de Hystrix a pu détecter le problème de timeout et va rediriger le

traitement vers une solution de contournement en se basant sur le Design pattern **Fallback**

Processing.

Pour arriver à cet objectif, on va simuler un timeout au niveau du Contrôleur Employee.

4. Adaptation du microservice « Employee » pour bénéficier de Hystrix



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.myHR</groupId>
  <artifactId>api_sb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>api_sb</name>
  <description>API with Spring Boot</description>
  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Finchley.RELEASE</spring-cloud.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
```

```

        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
        <version>1.4.7.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
        <version>2.2.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build><plugins><plugin><groupId>org.springframework.boot</groupId><artifactId>spring-
boot-maven-plugin</artifactId></plugin></plugins></build></project>

```

```

#Global configuration
spring.application.name=Api_sbHystrix
#Tomcat configuration
server.port=9000
#Log level configuration
logging.level.root=ERROR
logging.level.com.myHR=INFO
logging.level.org.springframework.boot.autoconfigure.h2=INFO
logging.level.org.springframework.boot.web.embedded.tomcat=INFO
#H2 Configuration
spring.jpa.show-sql=true

```

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:mytestdb
```

```
# Hystrix dashboard management
```

```
management.endpoints.web.exposure.include=hystrix.stream
```

```
package com.myHR.api_sb.controller;
```

```
import java.util.Optional;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.DeleteMapping;
```

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

```
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.PutMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
```

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

```
import com.myHR.api_sb.model.Employee;
```

```
import com.myHR.api_sb.service.EmployeeService;
```

```
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
```

```
import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;
```

```
import
```

```
org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;
```

```
@EnableCircuitBreaker
```

```
@Configuration
```

```
@EnableHystrixDashboard
```

```
@RestController
```

```
public class EmployeeController {
```

```
    @Autowired
```

```
    private EmployeeService employeeService;
```

```
    @GetMapping("/myMessage")
```

```
    @HystrixCommand(fallbackMethod = "myHystrixbuildFallbackMessage",
        commandProperties = {@HystrixProperty(name =
"execution.isolation.thread.timeoutInMilliseconds", value = "1000")},
        threadPoolKey = "messageThreadPool")
```

```
    public String getMessage() throws InterruptedException {
```

```
        System.out.println("Message from EmployeeController.getMessage():"
```

```

Begin To sleep for 3 scondes ");
    Thread.sleep(3000);
    return "Message from EmployeeController.getMessage(): End from sleep for 3
scondes ";
}

    private String myHistrixbuildFallbackMessage() {
        return "Message from myHistrixbuildFallbackMessage() : Hystrix
Fallback message ( after timeout : 1 second )";
    }

    @GetMapping("/employees")
    public Iterable<Employee> getEmployees() {
        return employeeService.getEmployees();
    }

    @DeleteMapping("/employee/{id}")
    public void deleteEmployee(@PathVariable("id") final Long id) {
        employeeService.deleteEmployee(id);
    }

    @PostMapping("/employee")
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.saveEmployee(employee);
    }

    @GetMapping("/employee/{id}")
    public Employee getEmployee(@PathVariable("id") final Long id) {
        Optional<Employee> employee = employeeService.getEmployee(id);
        if(employee.isPresent()) {
            return employee.get();
        } else {
            return null;
        }
    }

    @PutMapping("/employee/{id}")
    public Employee updateEmployee(@PathVariable("id") final Long id, @RequestBody
Employee employee) {
        Optional<Employee> e = employeeService.getEmployee(id);
        if(e.isPresent()) {
            Employee currentEmployee = e.get();

            String firstName = employee.getFirstName();
            if(firstName != null) {
                currentEmployee.setFirstName(firstName);
            }
            String lastName = employee.getLastName();
            if(lastName != null) {
                currentEmployee.setLastName(lastName);
            }
            String mail = employee.getMail();
            if(mail != null) {
                currentEmployee.setMail(mail);
            }
            String password = employee.getPassword();
            if(password != null) {

```

```

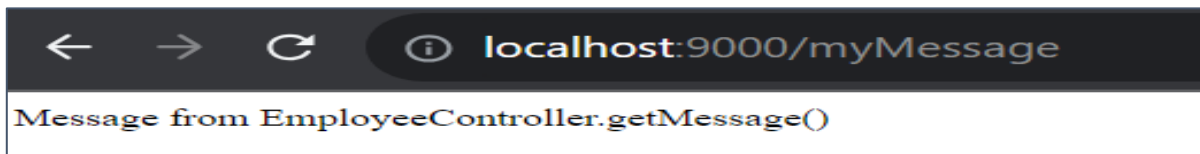
        currentEmployee.setPassword(password);
    }
    employeeService.saveEmployee(currentEmployee);
    return currentEmployee;
} else {
    return null;
}
}
}

```

- Résultat d'appel [sans activation](#) du Timeout:

Au niveau de la console:

Message from EmployeeController.getMessage(): Begin To sleep for 3 scondes

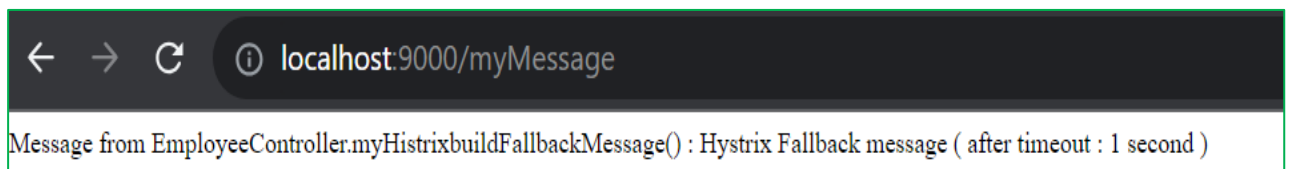


- Résultat d'appel avec activation du Timeout : **Hystrix** prend le contrôle après **1 seconde** définit dans :

```

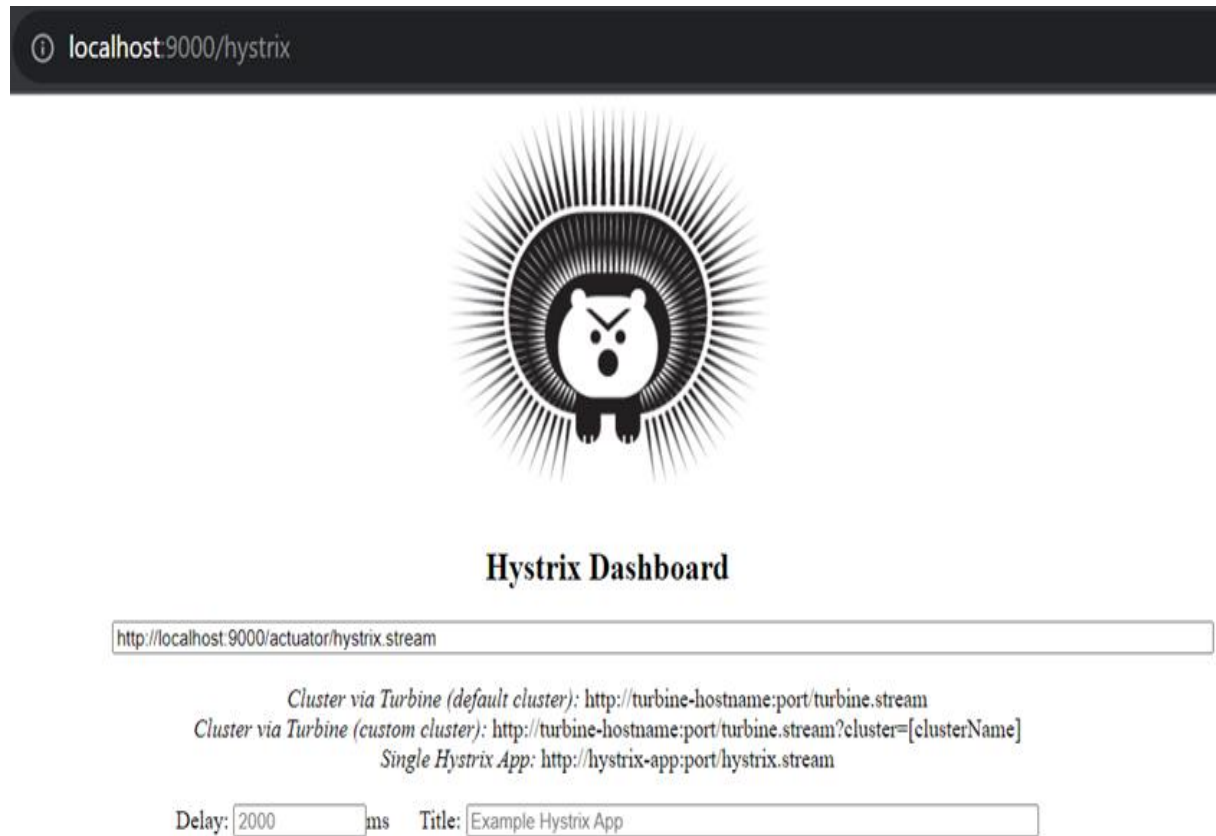
commandProperties ={@HystrixProperty(name =
"execution.isolation.thread.timeoutInMilliseconds",
value = "1000")

```



5. Exploitation du Dashbord de Hystrix

- Accès au dashbord Hystrix : <http://localhost:9000/hystrix>
- Dans le stream du dashbord entrer : <http://localhost:9000/actuator/hystrix.stream>



Hystrix Stream: http://localhost:9000/actuator/hystrix.stream



Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#)

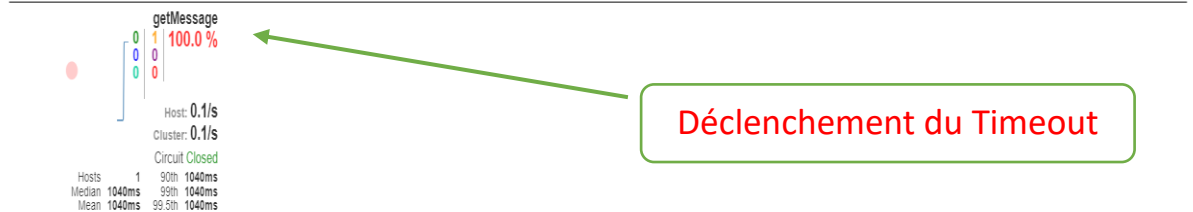


- Appel au microservice : <http://localhost:9000/myMessage>

Hystrix Stream: My Hystrix Application



Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#) [Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



Déclenchement du Timeout

Thread Pools Sort: [Alphabetical](#) | [Volume](#)



Hystrix Stream: My Hystrix Application



Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#) [Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#)

