CIRCUIT BREAKER DESIGN PATTERNS

1. In a microservices architecture, failure is inevitable.
2. A single failing service can cause a cascading effect, leading to system-wide failures.
3. To prevent such scenarios, implementing [circuit breakers](https://www.javaguides.net/2024/05/spring-cloud-circuit-breaker-tutorial.html) is crucial

What is a Circuit Breaker?

A circuit breaker is a design pattern used in microservices architecture to prevent cascading failures and provide fallback mechanisms. When a service call fails repeatedly, the circuit breaker trips and directs the calls to a fallback method, thereby preventing further failures and allowing the system to recover gracefully.

**Key Benefits**

1. Prevents cascading failures by stopping the flow of requests to a failing service.
2. Provides alternate responses when a service fails.
3. Enhances the overall resilience of the system by isolating failures.

Example project  
  
PRODUCT- MICROSERVICE  
  
dependencies required

* **Spring Web**

ORDER-MICROSERVICES

dependencies required:

* **Spring Web**
* **Spring Cloud Circuit Breaker**
* **Spring Boot Starter Actuator** (for monitoring)

Why circuit breaker added in this order service?  
1. **Order Service depends on Product Service**

* If Product Service is **down**, **slow**, or throwing errors, then without a circuit breaker the **Order Service** will also fail or hang while waiting.

2. **To prevent cascading failures**

* Imagine Product Service is down.
* If thousands of requests come into Order Service, all will try to reach Product Service → causing timeouts, slow responses, and eventually crashing Order Service.
* The **circuit breaker stops making calls after repeated failures**, so Order Service stays healthy.

3. **To provide a fallback (graceful response)**

* Instead of returning an error when Product Service is unavailable, the **fallback method** (fallbackGetProductById) is triggered.
* This gives a **default response** (an Order with "Fallback Product") so the system can still respond to users.

So in simple , The circuit breaker is added in **Order Service** because it calls **Product Service**. If Product Service fails, the circuit breaker prevents repeated failures, protects Order Service from crashing, and routes the request to a fallback method for a safe response.  
  
  
NOTE: if the instances are not showing in the server make sure added all the dependencies mainly “spring-cloud-dependencies”

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
</dependency>

PHASES OF CIRCUIT BREAKER PATTERN  
  
1. WHEN ORDER SERVICE – UP and PRODUCT SERVICE IS -DOWN  
  
  
  

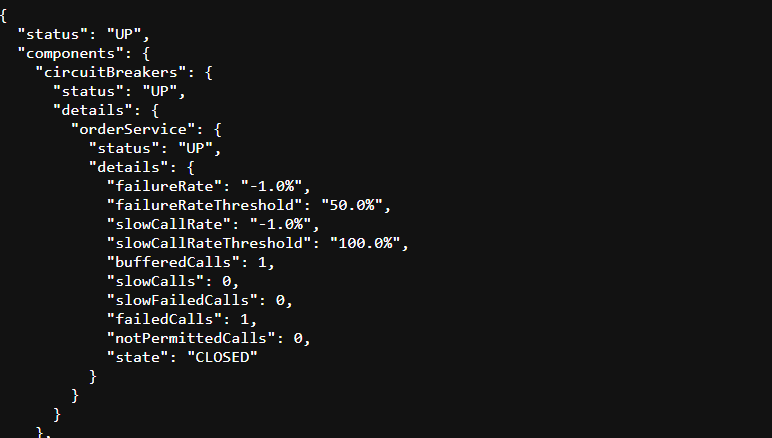

When doing rest call from orderservice to product service  
[localhost:8081/orders/1](http://localhost:8081/orders/1)  
so instead of giving 500 internal server error because productservice is down it will give some response which is dummy fallback method response  
{

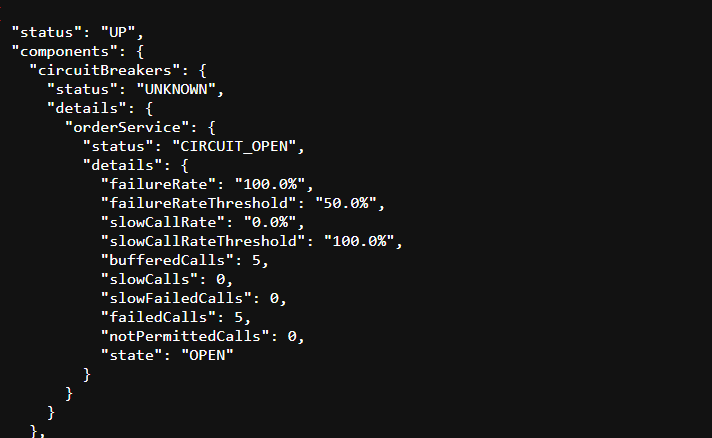
"id": 1,

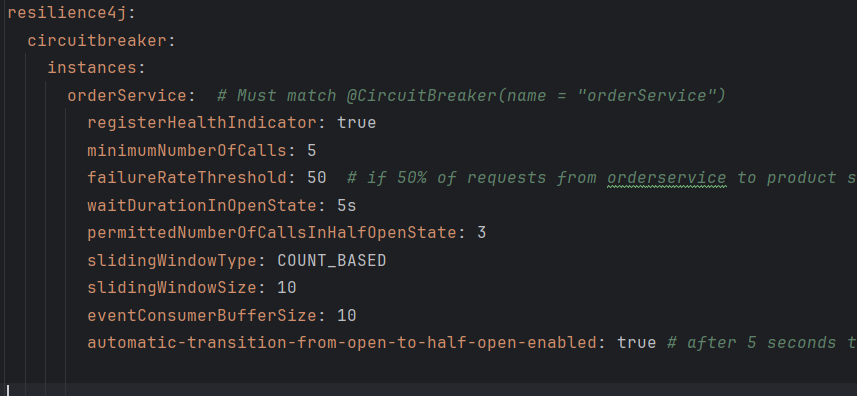
"productName": "Fallback Product",

"quantity": 1

}



as of now failedcalls = 1, which does not exceeds the thresholdrate which is “50%” and also failure rate.  


If failure rate reaches to 100% then status changes to “open”  
  
  
   
changed to half open after 5seconds  
  
as we configured the above in application where  
“waitDurationInOpenState: 5s” that mean when the status changes to “ open” it will be only for 5 sec , then after 5sec the status will changed to “half-open”  
  
as we mentioned this In application.yml file  
permittedNumberOfCallsInHalfOpenState: 3

That means when the status is halfopen only 3 calls are permitted .

After making three class the status will changed to “open”  
  
immediately the status will be changed to halfopen again because it will be only for 5 seconds  
  
  
when the status is in halfopen and after 3 retry attempts of hitting the request url, then status changes to closed like below  
  
then if u hit the url [localhost:8081/orders/1](http://localhost:8081/orders/1) then u must get the actual response  
{

"id": 1,

"productName": "iphone 17promax",

"quantity": 3

}  
   
[localhost:8081/orders/3](http://localhost:8081/orders/3)  
  
{

"id": 3,

"productName": "IPhone 15 promax",

"quantity": 3

}