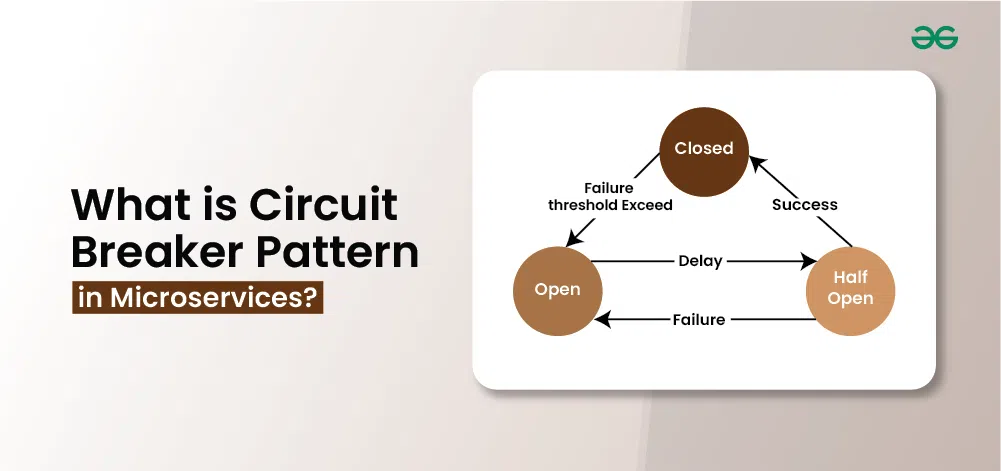
To implement circuit breaker pattern add following dependencies in pom.xml in consumer application;

**What is Circuit Breaker Pattern in Microservices?**

The Circuit Breaker pattern is a design pattern used in [microservices](https://www.geeksforgeeks.org/microservices/)to enhance system [resilience](https://www.geeksforgeeks.org/resilient-system-system-design/)and [fault tolerance](https://www.geeksforgeeks.org/fault-tolerance-in-system-design/). It acts like an electrical circuit breaker by preventing an application from repeatedly trying to execute an operation that is likely to fail, which can lead to cascading failures across the system.

**Table of Content**



**What is a Circuit Breaker Pattern?**

The Circuit Breaker pattern is like a safety switch for your microservices. Imagine you have an online store that relies on a payment service. If that payment service starts failing repeatedly, instead of your store trying to contact it over and over (which could make things worse), the circuit breaker “trips” and stops any further attempts for a while. let's understand circuit breaker pattern with this example:

* Your store makes a request to the payment service to process a payment. Everything works fine.
* Suddenly, the payment service has issues and fails three times in a row.
* The circuit breaker trips and enters an "open" state. Now, when your store tries to contact the payment service, it immediately gets an error response instead of trying to connect again.
* After a set time, the circuit breaker changes to a "half-open" state. It allows a few test requests to see if the payment service is back online.
* If those requests succeed, the circuit breaker resets to "closed," and everything goes back to normal. If they fail, it stays open longer, giving the payment service more time to recover.

**Characteristics of Circuit Breaker Pattern**

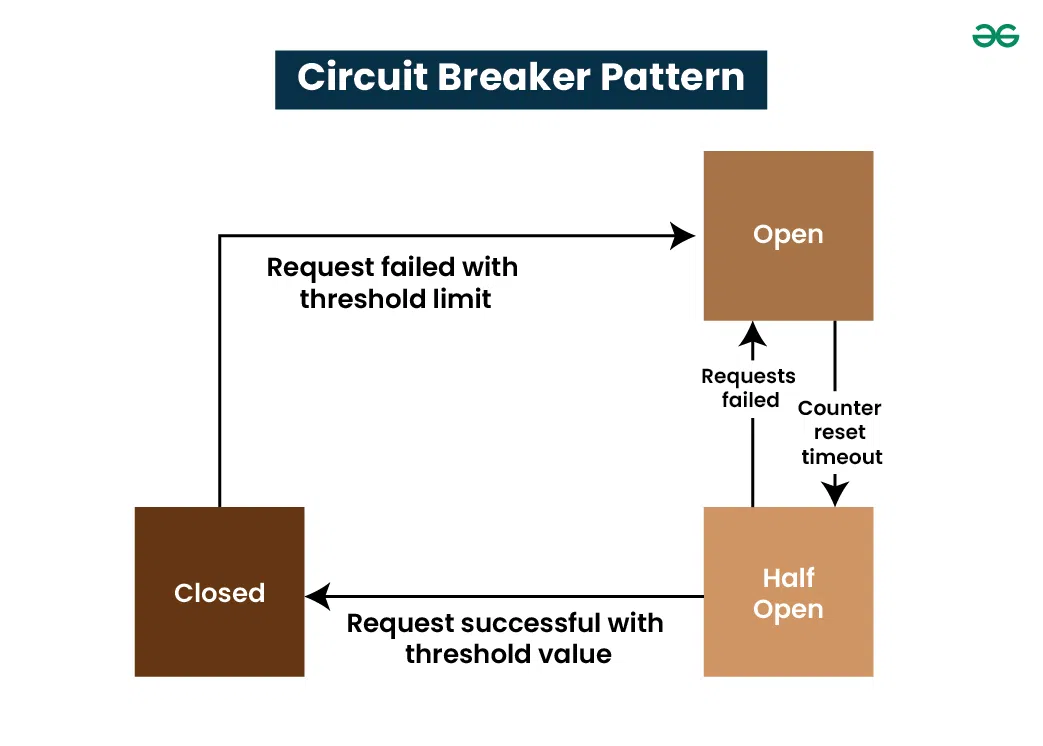
Below are some of the characteristics of Circuit Breaker Patterns in Microservices include:

* Circuit Breaker enhances[fault tolerance](https://www.geeksforgeeks.org/fault-tolerance-in-system-design/) by isolating and managing failures in individual services.
* It continuously monitors interactions between services to detect issues in real time.
* Also useful in temporarily stops requests to failing services, preventing cascading failures and minimizing disruptions.
* It Provides fallback responses or error messages to clients during service failures, ensuring graceful degradation.
* It Automatically transitions back to normal operation when the failing service recovers, improving system reliability

**Working and Different States in Circuit Breaker Pattern**

The Circuit Breaker pattern typically operates in three main states: Closed, Open, and Half-Open. Each state represents a different phase in the management of interactions between services. Here's an explanation of each state:

* **Closed State**
  + In the Closed state, the circuit breaker operates normally, allowing requests to flow through between services.
  + During this phase, the circuit breaker monitors the health of the downstream service by collecting and analyzing metrics such as response times, error rates, or timeouts..
* **Open State**
  + When the monitored metrics breach predetermined thresholds, signaling potential issues with the downstream service, the circuit breaker transitions to the Open state.
  + In the Open state, the circuit breaker immediately stops forwarding requests to the failing service, effectively isolating it.
  + This helps prevent cascading failures and maintains system stability by ensuring that clients receive timely feedback, even when services encounter issues.
* **Half-Open State**
  + After a specified timeout period in the Open state, transitions to Half-Open state.
  + Allows a limited number of trial requests to pass through to the downstream service.
  + Monitors responses to determine service recovery.
  + If trial requests succeed, indicating service recovery, transitions back to Closed state.
  + If trial requests fail, service issues persist.
  + May transition back to Open state or remain in Half-Open state for further evaluation.



<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>

<version>2.1.3</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

Controller2

package com.ict.controller;

import java.util.List;

import java.util.ArrayList;

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

import org.springframework.cloud.client.ServiceInstance;

import org.springframework.cloud.client.discovery.DiscoveryClient;

import org.springframework.stereotype.Component;

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

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

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

import org.springframework.web.client.RestTemplate;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;

import io.github.resilience4j.retry.annotation.Retry;

@RestController

public class ConsumerControllerApp {

@Autowired

DiscoveryClient discoveryClient;

//need to comment method in producer

@GetMapping("/data1")

@Retry(name="producer", fallbackMethod="sendDummyData")

@CircuitBreaker(name="producer", fallbackMethod="sendDummyData")

public String getCources() { // http://localhost:8083/data1

List<ServiceInstance> siList = discoveryClient.getInstances("PRODUCERAPP");

ServiceInstance si = siList.get(0);

String url = si.getUri() +"/producer";

RestTemplate rt = new RestTemplate();

String response = rt.getForObject(url, String.class);

System.out.println("using discovery client"+ response);

return response;

}

public String sendDummyData(Exception e) {

return "Producer is unvailable,please go for other option";

}

}

application.yml:

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka

instance:

preferIpAddress: true

spring:

application:

name: consumerapp

server:

port: 8083

resilience4j:

retry:

instances:

producer:

max-attempts: 5

instance:

producerPort:

max-attempts: 6