Circuit Breaker

Prashant Bansal
Edison, New Jersey, USA

Abstract:- In the realm of microservices architecture, ensuring reliability and resilience is paramount due to the distributed nature and interdependencies of services. A key pattern employed to enhance system robustness is the circuit breaker. When a microservice experiences failures, the circuit breaker pattern detects these faults and transitions between three states: closed, open, and half-open. During this period, requests are automatically redirected or fail fast, preventing additional strain on the troubled service. Once the service stabilizes, the circuit breaker transitions to the half-open state, where it allows a limited number of test requests to verify recovery. If these succeed, the circuit breaker closes, restoring normal operation. This abstract explores the implementation, benefits, and challenges of circuit breakers in microservices, emphasizing their role in maintaining high availability and robust performance in modern distributed systems.

Keywords:- *CB, Circuit, Circuit Breaker, Microservices.*

I. INTRODUCTION

It is a design pattern to prevent calling a microservice in case of failure or abnormal behavior in that service. This idea can help the client's side ignore handling failed requests and would prevent number of failed calls giving the service time to recover.

Hystrix is one of the popular implementation libraries. Hystrix is a Netflix library that provides several features for building resilient systems, including circuit breakers.

II. STATES OF CIRCUIT BREAKER

Let's say, Service 1 calls Service 2, but Service 2 unfortunately does not respond. Service 1 will either wait for response OR will handle the time out exception for every failure. Subsequent requests directed at Service 2 will encounter similar challenges, leading to a bad user experience.

In such cases, circuit breaker can help by stopping request sending for a specific time, waiting for the timeout ends, enable a limited number of requests to check whether Service 2 is working.

➢ *Circuit Breaker has 3 States.*

- *Open*

During the open state of circuit braker, requests will be blocked and will not be sent further to the service layer. This handles the overwhelming requests to the service, after a certain predefined time, circuit braker switched to half open state from open state.

- *Closed*

When the circuit breaker is in closed condition, it will allow calls to happen until a failure threshold is reached. It will keep track of number of failures and will switch to open state once the failure threshold is exceeded.

- *Half-Open*

After a cool down period from open state, a single call to the service is allowed. If the call succeeds, the circuit breaker will transition to the closed state. If the call fails, the circuit breaker will transition to the open state.

III. NECESSITY OF CIRCUIT BREAKERS

In the world of Microservices which rely on communication between multiple systems, even a single failure within the ecosystem could lead to multiple failures due to ripple effect. Here comes circuit breaker to the rescue, not only it restricts the number of failure calls giving the system time to recover but also makes sure, system is not overloaded due to higher number of timeouts and failures.

In a large ecosystem containing number of integrations, circuit breakers play a vital role to maintain the resilience and robustness.

IV. CIRCUIT BREAKER IMPLEMENTATION WITH HYSTRIX

- Please add annotation to you base class @EnableCircuitBreaker. Without this annotation, circuit breakers won't be available to use.
- In order to let your method know about handling of circuit breaker in the class, we need to add *@HystrixCommand* annotation.
- Now depending on business needs, where in you might need fall back plan for post your failure of service.

V. CIRCUIT BREAKER IMPLEMENTATION (OWN CODE)

```
public class Main {
    public static void main(String[] args) {
        CircuitBreaker circuitBreaker = new CircuitBreaker();

        // Simulate calls to a remote service
        for (int i = 0; i < 10; i++) {
            if (circuitBreaker.allowCalls()) {
                if (callRemoteService()) {
                    // On success, reset the circuit breaker
                    circuitBreaker.recordSuccess();
                } else {
                    circuitBreaker.increaseFailedCount();
                }
            } else {
                System.out.println("Circuit breaker is open,
request not allowed");
            }

            try {
                Thread.sleep(1000); // Simulate a delay between
calls
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
```

VI. CONCLUSION

The Circuit Breaker pattern is an excellent way to increase the reliability of your microservices architecture.

Circuit breaker ensures less failure load and more robust system providing systems time to recover while not catering to unnecessary failed service calls. Circuit breaker should be part of microservice design architecture specially when heavy traffic via those services is expected.

REFERENCES

[1]. https://medium.com/@abhishekranjandev/the-circuit-breaker-pattern-in-microservices-an-in-depth-look-fa3c4e40b6eb

[2]. https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/circuit-breaker.html