**Resilience in Microservices: A Scenario**

In a microservices architecture, consider a scenario where the **Accounts microservice** relies on both the **Loans** and **Cards microservices**. If the **Cards microservice** experiences issues such as network latency, high traffic, or downtime, it may respond slowly or become unreachable. Since the **Accounts microservice** depends on responses from both services, it waits for the **Cards microservice**, causing delays in responding to the client.

This situation highlights the importance of **resilience in microservices**—the system's ability to handle disruptions, manage delays gracefully, and minimize impact on the user experience.

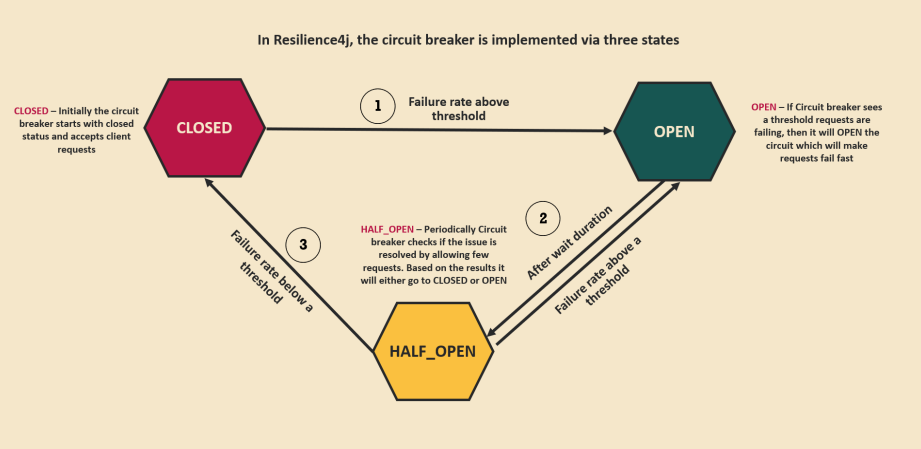
**What Are Resilient Microservices?**

Resilient microservices are designed to remain functional during network failures, service downtimes, or high loads, ensuring service availability and performance. A key pattern that enhances resilience is the Circuit Breaker Pattern.

**The Circuit Breaker Pattern**

Inspired by electrical circuit breakers that prevent damage by stopping excessive current flow, the **Circuit Breaker Pattern** in microservices acts as a safeguard. It stops repeated failed calls to a service by "tripping" after a defined threshold of failures. This prevents overloading the failing service and allows the system to handle issues more gracefully.

In microservices, the **Resilience4j library** is commonly used to implement circuit breakers, operating in three states: **CLOSED**, **OPEN**, and **HALF\_OPEN**.



**Circuit Breaker States and Transitions**

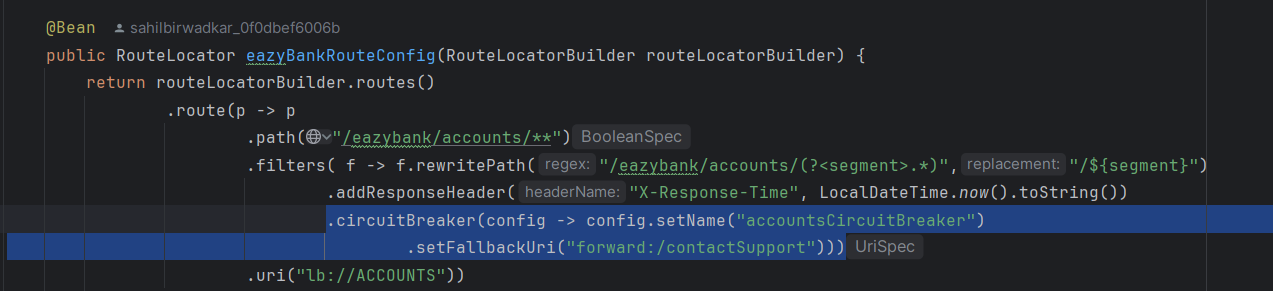
1. **CLOSED State**
   * All requests are allowed to pass through to the dependent service.
   * If the failure rate (e.g., timeouts or errors) exceeds the configured threshold, the circuit breaker transitions to the **OPEN** state.
2. **OPEN State**
   * Requests to the failing service are blocked.
   * This state is triggered when the failure rate crosses the threshold.
   * While in the **OPEN** state, requests are immediately rejected to prevent overloading the service.
   * After a predefined wait duration, the circuit breaker transitions to the **HALF\_OPEN** state.
3. **HALF\_OPEN State**
   * A limited number of requests are allowed to test if the service has recovered.
   * If these requests succeed and the failure rate drops below the threshold, the circuit breaker moves back to the **CLOSED** state.
   * If failures persist, it transitions back to the **OPEN** state.

The **Circuit Breaker Pattern** enhances system stability and performance by controlling the flow of requests to problematic services.

**Implementing a Circuit Breaker**

There are two common ways to implement circuit breakers in a microservices setup:

1. **Using an Edge Server**
   * Add the required dependency in the pom.xml of the **gateway server**:
   * Set up the circuit breaker by configuring its name and fallback URI in the **RouteLocator** class to customize API routes.



* + Define parameters in the application.yml file:
* **Sliding Window Size**: Specifies the size of the window used to monitor metrics (e.g., failure rate).
* **Permitted Calls in Half-Open State**: Defines the number of requests allowed during the **HALF\_OPEN** state to test recovery.
* **Failure Threshold**: The percentage of failed requests that triggers the **OPEN** state.
* **Wait Duration in OPEN State**: The duration the circuit breaker remains **OPEN** before moving to the **HALF\_OPEN** state.

