explain this in depth resilience4j: circuitbreaker: configs: default: register-health-indicator: true sliding-window-type: COUNT\_BASED sliding-window-size: 10 permitted-number-of-calls-in-half-open-state: 2 automatic-transition-from-open-to-half-open-enabled: true failure-rate-threshold: 50 wait-duration-in-open-state: 10s max-wait-duration-in-half-open-state: 5s

###### ChatGPT said:

Perfect 👍 You’ve shared a **Resilience4j Circuit Breaker config** in YAML, and you want a **deep-dive explanation** of what each property means and how it influences behavior. Let’s break it down carefully.

## 🔹 Circuit Breaker Basics (Resilience4j)

A **circuit breaker** protects your microservice from repeatedly calling a failing downstream service (DB, REST API, Kafka, etc.).  
It works like an **electrical circuit breaker**:

* **Closed** → normal operation (calls allowed).
* **Open** → calls are blocked (fail fast).
* **Half-open** → test a few calls before deciding to close or open again.

## 🔹 Your Config in Detail

resilience4j:

circuitbreaker:

configs:

default:

register-health-indicator: true

sliding-window-type: COUNT\_BASED

sliding-window-size: 10

permitted-number-of-calls-in-half-open-state: 2

automatic-transition-from-open-to-half-open-enabled: true

failure-rate-threshold: 50

wait-duration-in-open-state: 10s

max-wait-duration-in-half-open-state: 5s

### 1. register-health-indicator: true

* Registers this circuit breaker with **Spring Boot Actuator**.
* You can check its status at /actuator/health or /actuator/circuitbreakerevents.
* Helps monitoring systems (Prometheus, Grafana, etc.) detect failures automatically.

### 2. sliding-window-type: COUNT\_BASED

* Defines how failures are calculated.
* **COUNT\_BASED** → looks at the last **N calls** (where N = sliding-window-size).
* Alternative is **TIME\_BASED** (e.g., last 10 seconds).

📌 In this config: We’re using a **window of 10 calls**.

### 3. sliding-window-size: 10

* Number of calls to evaluate when calculating the **failure rate**.
* Here → Circuit breaker checks the **last 10 calls**.
* If 5 of them fail (≥ 50% threshold), the breaker **opens**.

### 4. failure-rate-threshold: 50

* Threshold (%) to trigger **OPEN state**.
* Here → If **≥ 50% of last 10 calls fail**, breaker opens.
* Example:
  + Last 10 calls → 6 succeed, 4 fail → Failure rate = 40% → Still Closed.
  + Last 10 calls → 5 succeed, 5 fail → Failure rate = 50% → Breaker Opens.

### 5. wait-duration-in-open-state: 10s

* How long the breaker stays **open** before trying to go **half-open**.
* During this 10s → all requests fail immediately with CallNotPermittedException.

### 6. automatic-transition-from-open-to-half-open-enabled: true

* Enables **automatic recovery attempt**.
* After 10s in open state, the breaker will **transition to half-open** without waiting for a manual trigger.

### 7. permitted-number-of-calls-in-half-open-state: 2

* Number of test calls allowed in **half-open** state.
* Example:
  + Circuit breaker moves to **half-open** after 10s.
  + It lets through **2 calls** to test if the service has recovered.
  + If both succeed → Circuit breaker closes (normal again).
  + If one fails → Circuit breaker reopens immediately.

### 8. max-wait-duration-in-half-open-state: 5s

* Maximum time the breaker can stay in **half-open** before deciding.
* Prevents a situation where **test calls never happen** (e.g., low traffic).
* If 5s passes with no calls → breaker **goes back to open**.

## 🔹 Flow Example with Your Config

1. Start in **Closed** state.
   * Tracks last 10 calls.
2. If 5 of 10 calls fail → Failure rate = 50% → Switch to **Open**.
3. Stay **Open** for 10 seconds.
   * All calls blocked during this time.
4. After 10 seconds → Automatically transition to **Half-Open**.
   * Allow **2 test calls**.
5. Case A: Both succeed → Switch to **Closed** (normal operation).  
   Case B: At least 1 fails → Switch back to **Open** for 10 seconds.  
   Case C: No call comes in within 5s → Go back to **Open**.

## 🔹 Why This Config is Useful in Microservices

* **Prevents cascading failures** → if a downstream service is unhealthy, you stop hammering it.
* **Fast failure** → users don’t wait on timeouts.
* **Self-healing** → automatically retries after cool-off.
* **Health visibility** → integrates with monitoring.

✅ **In short**: This config creates a **smart, self-healing circuit breaker** that:

* Evaluates last 10 requests,
* Opens if 50% fail,
* Waits 10s before retry,
* Tests with 2 calls,
* Falls back if service is still broken.

👉 Do you want me to also draw an **ASCII timeline diagram** showing how requests move between **Closed → Open → Half-Open → Closed** with your settings?

##### You said:

ASCII dia yes

###### ChatGPT said:

Alright 🚀 — here’s an **ASCII diagram** that shows how your Resilience4j circuit breaker config behaves step by step.

## ⚡ Circuit Breaker State Transitions (Your Config)

┌─────────────────────┐

│ CLOSED │

│ (normal operation) │

│ last 10 calls count │

└─────────┬───────────┘

│

failure-rate ≥ 50% │

▼

┌─────────────────────┐

│ OPEN │

│ all calls blocked │

│ (10s wait timer) │

└─────────┬───────────┘

│

after 10s wait │

auto-transition enabled ▼

┌─────────────────────┐

│ HALF-OPEN │

│ allow 2 test calls │

│ max wait = 5s │

└─────────┬───────────┘

┌──────────────┴───────────────┐

│ │

both calls succeed any call fails

│ │

▼ ▼

┌─────────────────────┐ ┌─────────────────────┐

│ CLOSED │ │ OPEN │

│ normal operation │ │ block calls again │

│ reset counters │ │ wait 10s again │

└─────────────────────┘ └─────────────────────┘

## 🔑 With Your Parameters:

* **Sliding window size = 10 calls** → decision made after each set of 10 requests.
* **Failure threshold = 50%** → 5 out of 10 failures = breaker opens.
* **Open duration = 10s** → block all requests for 10s.
* **Half-open test = 2 calls** → let exactly 2 requests try.
* **Max half-open wait = 5s** → if no calls come in, go back to open.