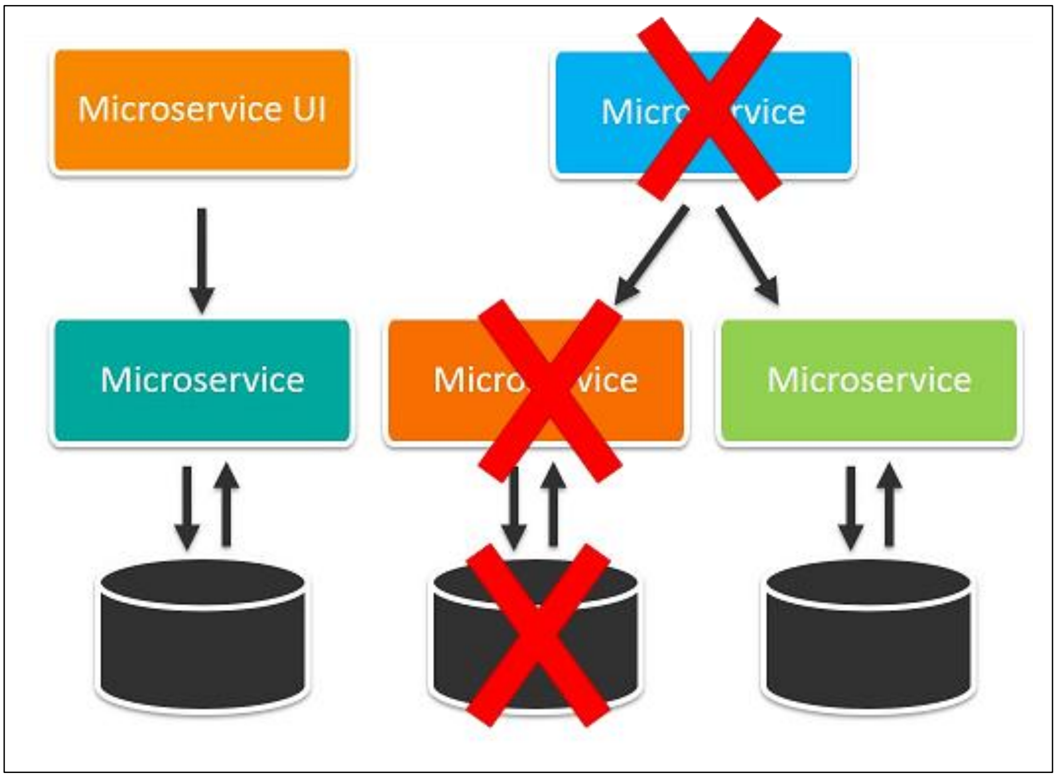
Circuit Breaker Pattern

Problem

In case if system calls any external components synchronously (subordinary microservice, database, messaging system), there is a risk that external system doesn't response in expected time. It can happen because of number of reasons: network issues, failure or high load on external component.

In case if such external calls are not protected, the failure will be propagated to top layer and make the whole system not available. Resources such as threads might be consumed in the caller while waiting for the other service to respond. This might lead to resource exhaustion, which would make the calling service unable to handle other requests. The failure of one service can potentially cascade to other services throughout the application.

This situation is typical in microservice architecture, using cascading microservices calls.



Solutions

There are several solutions to prevent cascading failure:

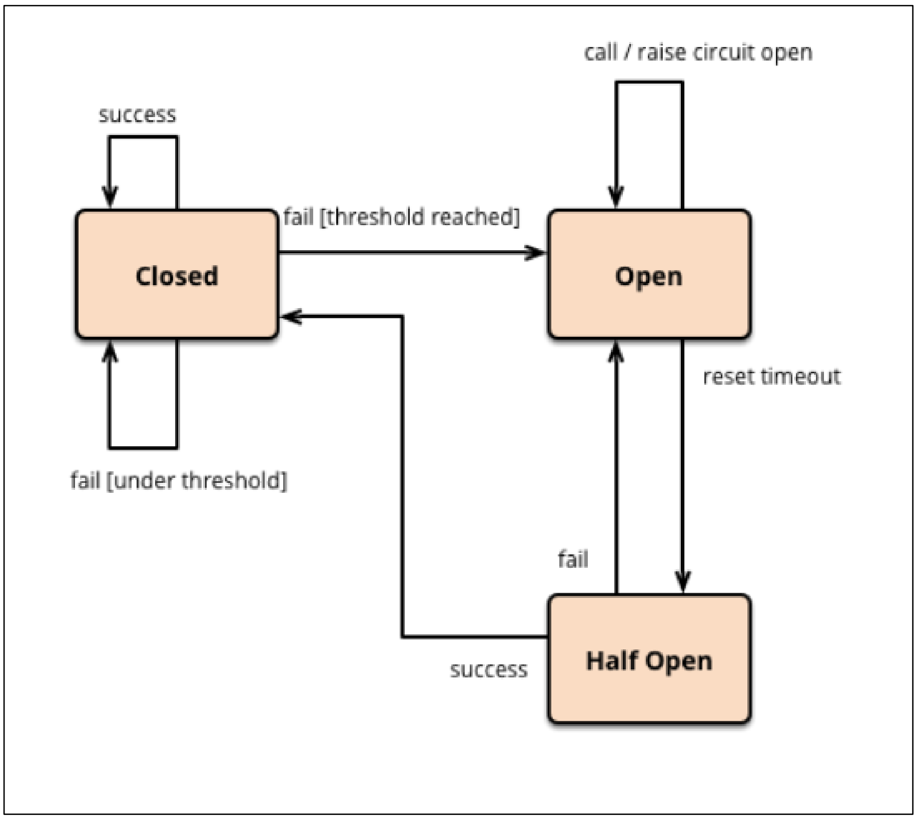
1. Timeout
2. Circuit Breaker
3. Redundancy and Duplications

Circuit Breaker is the most effective and important patterns to prevent vertical failure propagation.

Circuit Breaker Pattern

A service client should invoke a remote service via a proxy that functions in a similar fashion to an electrical circuit breaker. When the number of consecutive failures crosses a threshold, the circuit breaker trips, and for the duration of a timeout period all attempts to invoke the remote service will fail immediately. After the timeout expires the circuit breaker allows a limited number of test requests to pass through. If those requests succeed the circuit breaker resumes normal operation. Otherwise, if there is a failure the timeout period begins again.

The Circuit Breaker states are depicted bellow:



1. CB is closed
2. External system failure threshold is reached
3. CB moves into open state. In this state all calls of external system will be immediately rejected and fallback value (if any) will be returned
4. After the configured time, CB goes into half open state
5. In this state CB tries to invoke external system
6. In success case, CB moves into closed state
7. In failure case, CB returns into open state

Pros of Circuit Breaker

- ensures fail fast in case if external system is not available
- isolates external calls using own thread pool
- provides option to use default fallback, if external system is not available
- relaxes external system and gives more chances to recover

Cons of Circuit Breaker

- requires external framework
- requires additional configuration and monitoring

Tutorial

Tutorial demonstrates using Circuit Breaker in Microservices chain calls based on SpringBoot 2 and Hystrix implementation.

Important configurations

1. SpringBoot application class is annotated with @EnableCircuiteBreaker annotation
2. Method protected by Circuit Breaker is annotated with @HystrixCommand annotation
3. @HystrixCommad annotation contains link to fallback method, command and group keys
4. Additional configurations are listed in table bellow

Parameter	**Description**	**Sample value**
hystrix.command.default.circuitbreaker.enabled	Enable/disable CB	True
hystrix.command.default.circuitbreaker.requestValumeThreshold	minimum number of requests in a rolling window that will trip the circuit.	20
hystrix.command.default.circuitbreaker.sleepWindowInMilliseconds	amount of time, after tripping the circuit, to reject requests before allowing attempts again to determine if the circuit should again be closed	5000
hystrix.command.default.circuitbreaker.errorThresholdPercentage	error percentage at or above which the circuit should trip open and start short-circuiting requests to fallback logic	50

How to run demo

1. Clone projects from circuitbreaker
2. Import basic and composite service projects in your favorite IDE
3. Start both basic and composite microservices
4. Call composite microservice using [`curl http://localhost:8081/composite/hello`]
5. Expected result is 200 “`Hello from basic service`”
6. Stop basic microservice and invoke composite microservice using [`curl http://localhost:8081/composite/hello`]
7. Expected result is 200 “`Hello from composite fallback: com.netflix.hystrix.exception.HystrixTimeoutException`”. That means, basic microservice call is failed due hystrix timeout, but fallback value was returned

8. Make the 6 invocations of composite service using [`curl http://localhost:8081/composite/hello`]
9. The 6-th call will return “`Hello from composite fallback: java.lang.RuntimeException: Hystrix circuit short-circuited and is` OPEN”. This means, minimum value of calls (5) and failure threshold (50%) is reached and CB moves into OPEN state. During `sleepWindowInMilliseconds (20 sec)` time CB will reject all requests immediately and return fallback value.
10. After 20 sec, composite microservice will try to call basic service again. If it is not available, CB returns into OPEN state.
11. Start basic microservice.
12. Invoke composite microsevice using [`curl http://localhost:8081/composite/hello`]
13. Invocation is successful, CB moves into CLOSED state and call returns 200 “`Hello from basic service`”

Links

- https://microservices.io/patterns/reliability/circuit-breaker.html
- https://github.com/Netflix/Hystrix/wiki/How-it-Works
- https://github.com/ashakirin-talend/microservice-patterns/tree/master/circuitbreaker