Resilience: Hystrix

In this lesson, we'll look at resilience with Hystrix in detail.

WE'LL COVER THE FOLLOWING

Introduction

With synchronous communication between microservices, it is important that the failure of one microservice does not cause other microservices to fail as well.

Otherwise, the unavailability of a single microservice can cause further microservices to fail until the entire system is no longer available.

The microservices may return errors because they cannot deliver reasonable results due to a failed microservice.

However, it must not happen that a microservice waits for the result of another microservice for an infinite period of time and thereby becomes unavailable itself.

Resilience patterns

Michael T. Nygard's book, *Release It!* describes different patterns with which the resilience of a system can be increased. Hystrix implements some of these patterns. Let's discuss them.

Timeout

A **timeout** prevents a microservice from waiting too long for another microservice.

Without a timeout, **a thread can block for a very long time** because the thread does not get a response from another microservice. If all threads are blocked, the microservice will fail because there are no more threads available for new tasks.

Hystrix executes a request in a separate thread pool. Hystrix controls these threads and can terminate the request to implement the timeout.

Fail fast

Fail Fast describes a similar pattern. It is better to generate an error as quickly as possible.

The code can check at the beginning of an operation whether all necessary resources are available.

If this is not the case, the request can be terminated immediately with an error. This reduces the time that the caller has to block a thread or other resources.

Bulkhead

Hystrix can use its own thread pool for each type of request. For example, a separate thread pool can be set up for each called microservice.

If the call of a particular microservice takes too long, only the thread pool for that particular microservice is emptied, while the others still contain threads.

This will limit the impact of the problem and is called a **bulkhead**. This term was coined in analogy to a watertight bulkhead in a ship which divides the ship into different segments. If a leak occurs, only part of the ship is flooded with water so that the ship does not sink.

Circuit breaker

Finally, Hystrix implements a **circuit breaker**.

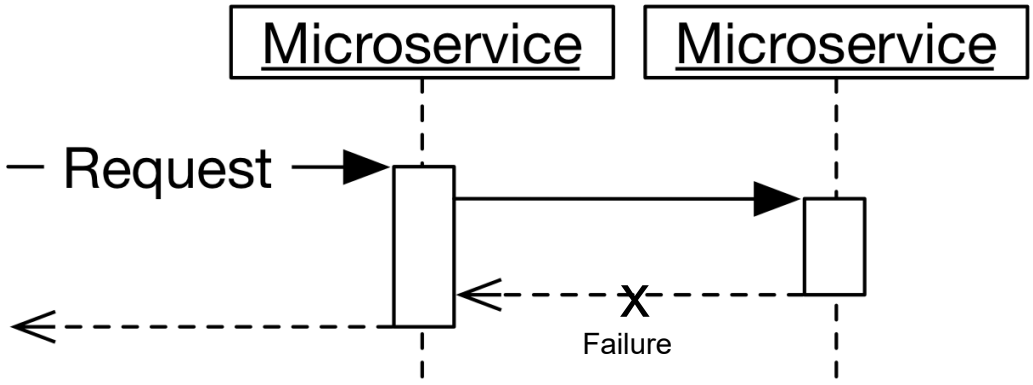
This is a fuse analogous to the ones used in the electrical system of a house.

There, a circuit breaker is used to cut off the current flow if there is a short circuit, preventing a fire from breaking out.

The Hystrix circuit breaker has a different approach. If a **system call results in an error, the circuit breaker is opened and does not allow any calls to pass through**.

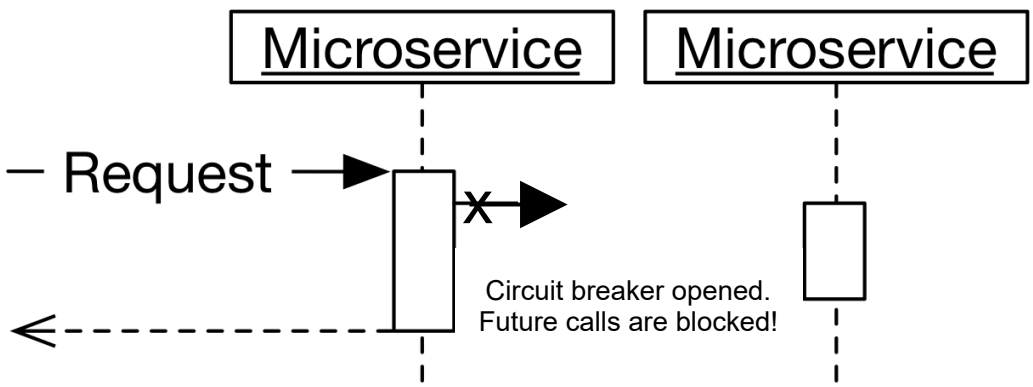
After some time, a call is allowed to pass through again. Only when this call is successful, is the circuit breaker closed again. This prevents a faulty microservice from being called. This **saves resources and avoids blocked threads**.

In addition, the circuit breakers of the different clients are closing one by one so that a failed and recovered microservice gradually handles the full load. This **reduces the probability that it will fail again immediately** after starting up.



Circuite breaker parttern in Hystrix

1 of 2



Circuite breaker parttern in Hystrix

2 of 2

QUIZ

1 The timeout pattern _____.

COMPLETED 0%

1 of 2

In the next lesson, we'll discuss some variations of the approaches that we've discussed in this lesson.