Resilience: Hystrix Implementation

In this lesson, we'll study how Hystrix is used in code.

We'll cover the following

Implementation

Hystrix (https://github.com/Netflix/Hystrix/) offers an implementation of most resilience patterns as a Java library.

The Hystrix API requires **command objects instead of simple method calls**. These classes supplement the method call with the necessary Hystrix functionalities.

When using Hystrix with Spring Cloud, it is not necessary to implement commands. Instead, the methods are annotated with `@HystrixCommand`. It activates Hystrix for this method and the attributes of the annotation configure Hystrix.

```
@HystrixCommand(
  fallbackMethod = "getItemsCache",
  commandProperties = {
    @HystrixProperty(
      name = "circuitBreaker.requestVolumeThreshold",
      value = "2") })
public Collection<Item> findAll() {
...
  this.itemsCache = pagedResources.getContent();
...
  return itemsCache;
}
```

The listing shows the access from the order microservice to the catalog microservice.

- **Lines 5-6:** `circuitBreaker.requestVolumeThreshold` specifies how many calls in a time window must cause errors for the circuit breaker to open.
- **Line 2:** The `fallbackMethod` attribute of the annotation configures the method `getItemsCache()` as a fallback method.
- **Line 7:** `findAll()` stores the data returned by the catalog microservice in the instance variable `itemsCache` .
- **Line 2:** The `getItemsCache()` method serves as a fallback and reads the result of the last call from the instance variable `itemsCache` and returns it. Have a look at the listing below for its implementation.

```
private Collection<Item> getItemsCache() {
  return itemsCache;
}
```

The reasoning behind this is that **it is better that the service continues to work with outdated data than a service that does not work at all.**.

This can lead to orders being charged at an outdated price. However, this is **better than accepting no orders at all**.

In general, **if a service fails, a default value can be used or an error can be reported**. Reporting an error is the correct solution if incorrect data cannot be accepted under any circumstances.

Which approach is correct in the end is a decision that **depends on the domain logic**. It should be avoided that in case of an error, the REST call burdens the server or blocks the client for too long.

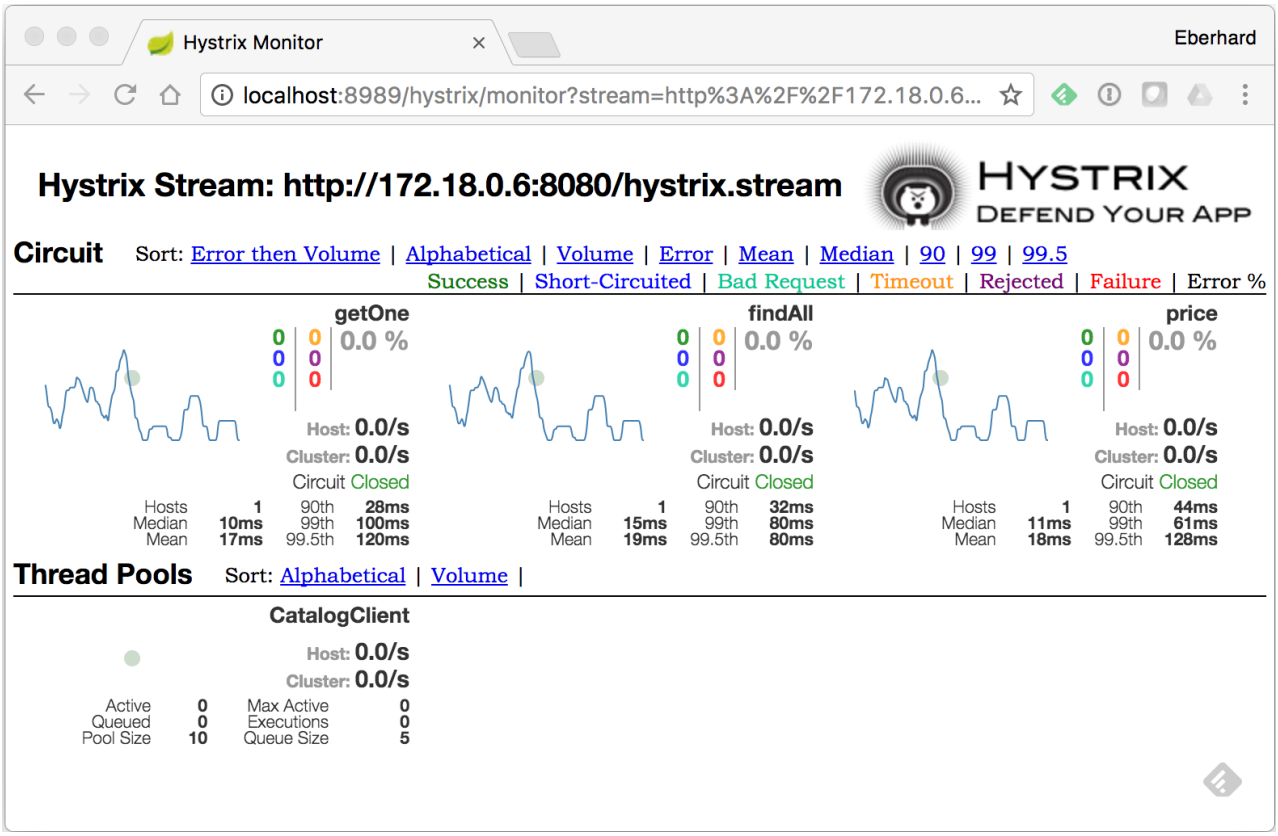
Monitoring

The state of the circuit breakers provides a good overview of the state of the system. An open circuit breaker is an indication of a problem.

Hystrix is a good source of metrics as it provides information about the circuit breaker state via HTTP as a stream of JSON data.

Hystrix dashboard

The Hystrix dashboard can display this data on a web page and thereby show what is happening in the system at the moment, see the screenshot below.



Hystrix Dashboard

The upper area shows the state of the circuit breaker for the functions `getOne()`, `findAll()`, and `price()`. The circuit breakers of all three functions are `closed` and there are no errors at the moment. The dashboard also shows information about the average latency of the requests and current throughput.

Hystrix executes the calls in a separate thread pool. The state of this thread pool is also shown on the dashboard. It contains ten threads and is currently processing no requests.

Other monitoring options

The Hystrix metrics are also available via the Spring Boot mechanisms and can be exported to other monitoring systems.

Thereby, Hystrix metrics can **seamlessly integrate into an existing monitoring infrastructure**.

monitoring infrastructure.

Turbine

- The metrics of a single microservice instance are not particularly meaningful.
- Microservices can be scaled independently.
- This means that many instances can exist for each microservice.
- This means that the Hystrix metrics of all instances must be displayed together.

This can be done with Turbine (https://github.com/Netflix/Turbine/wiki). This tool queries the HTTP data streams of the Hystrix servers and consolidates them into a **single stream of data displayed by the dashboard**.

Spring Cloud offers a simple way to implement a Turbine server, see for instance: https://github.com/ewolff/microservice/tree/master/microservice-demo/microservice-demo-turbine-server (https://github.com/ewolff/microservice/tree/master/microservice-demo/microservice-demo-turbine-server).

QUIZ

1 In the given code, what happens if the `findAll()` method fails?

```
@HystrixCommand(
  fallbackMethod = "getItemsCache",
  commandProperties = {
    @HystrixProperty(
      name = "circuitBreaker.requestVolumeThreshold",
      value = "2") })
public Collection<Item> findAll() {
...
  this.itemsCache = pagedResources.getContent();
...
  return itemsCache;
}
```

A) `circuitBreaker.requestVolumeThreshold` gets called and the second instance of `findAll()` gets used.

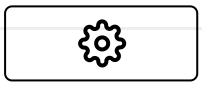
B) The `fallbackMethod`, `getItemsCache` gets called and returns an older version of the cache.

C) The microservice fails and results in a failure of the entire system.

Submit Answer

Question 1 of 3
0 attempted

Reset Quiz



In the next lesson, we'll look at some interesting variations.

← Back

Next →

Resilience: Hystrix

Variations

Mark as Completed

Report an Issue