Understanding the circuit breaker pattern

In this lesson we will learn about resource exhaustion in microservices and how to deal with it using the circuit breaker pattern.

Resource exhaustion

Having a retry policy in place is good, but you also must be mindful about the limited resources available to your service. Imagine once again a situation where there is an ongoing issue with your service dependency. This might not be just a transient issue but instead some prolonged down time caused, perhaps, by a broad network outage.

Now the client calls our service and this one in turn invokes the already failing dependency, which will hopelessly start waiting for a reply. While this is happening, more clients keep sending requests to our service and this results in more requests being sent to the failing service.

One thing you have to realize is that each of these requests are making use of your service threads, of which there is only a limited amount. Once enough threads are in use, there are no more resources available and you can reach what we call resource exhaustion. When this happens, your entire service becomes unavailable for any future requests potentially causing a lot of trouble in the system.

The circuit breaker pattern

One approach we can use to properly handle this issue is implementing the circuit breaker pattern. A circuit breaker prevents the service from performing an operation that's likely to fail. Here's how it works.

Once again, we are in a situation where our dependent service is already in a bad state, unable to provide successful replies. Our client then makes a request to our service. However, this time, instead of invoking the failing dependency directly, there is an intermediary that we will call the circuit breaker.

The circuit breaker will now start monitoring the results of each of the requests that go through it to the external dependency and, when it detects that the rate of failure goes beyond the configured threshold, it will immediately stop letting any more request to go out and will fail them right away. This is what we call opening the circuit.

After this, requests will keep failing immediately during the configured wait time, which would hopefully give the dependent service enough time to go back to a healthy state. Eventually, the circuit breaker will let some requests go out to verify if they succeed and, if that is the case, it will close the circuit again, letting all further requests reach the dependent service.

That's how the circuit breaker prevents our service from reaching resource exhaustion while at the same time avoid overwhelming dependent services until they get a chance to recover.

Let's go ahead and implement the circuit breaker pattern in our Inventory microservice.