Variations

In this lesson, we'll discuss some variations to the approaches we've already looked at.

We'll cover the following

- Zuul2
- resilience4j
- Consul
- Kubernetes
- HTTP proxy
- Atom
- Frontend integration

Netflix is only one technological option for implementing synchronous microservices.

There are **various alternatives** to the technologies of the Netflix stack.

Zuul2

The Zuul project is not maintained very well anymore. An alternative might be Zuul2 (https://github.com/Netflix/zuul). It is based on asynchronous I/O (https://medium.com/netflix-techblog/zuul-2-the-netflix-journey-to-asynchronous-non-blocking-systems-45947377fb5c) so it consumes less resources and is more stable. However, Spring Cloud won't support Zuul2 (https://github.com/spring-cloud/spring-cloud-netflix/issues/1498). Another alternative might be Spring Cloud Gateway

netflix/issues/1198). Another alternative might be Spring Cloud Gateway (https://spring.io/projects/spring-cloud-gateway). The approach with

Apache for routing shown in chapter 11 (https://www.educative.io/collection/page/10370001/5441945024331776/4625149481451520) and Kubernetes in chapter 13 (https://www.educative.io/collection/page/10370001/5441945024331776/4922985196552192) is probably even better.

resilience4j

Netflix does not invest in Hystrix that much anymore. They suggest using resilience4j (https://github.com/resilience4j/resilience4j) instead, which is a very similar Java library that supports typical resilience patterns.

Consul

The **Consul example** (chapter 11 (https://www.educative.io/collection/page/10370001/5441945024331776/4625149481451520)) uses:

- **Consul** instead of **Eureka** for **service discovery**
- **Apache httpd** instead of **Zuul** for **routing**
- However, this project also uses **Hystrix** for **resilience**
- and **Ribbon** for **load balancing**.

Consul supports DNS and can handle any programming language as implemented in the Consul DNS example. Consul Template offers the possibility to configure services with Consul by filling a configuration file template with the data from Consul. In the example, Apache httpd is configured this way.

Eureka has quite a few advantages over Consul. Apache httpd as a web

server is familiar to many developers and might therefore be less risky compared to Zuul. On the other hand, Zuul provides dynamic filters that Apache httpd does not support.

Kubernetes

Kubernetes (see chapter 13 (https://www.educative.io/collection/page/10370001/5441945024331776/4922985196552192/)) and a *PaaS*, such as *Cloud Foundry* (see chapter 14 (https://www.educative.io/collection/page/10370001/5441945024331776/5727581686988800)), offer **service discovery, routing, and load balancing**. At the same time, **the code remains independent** of the infrastructure.

Nevertheless, the examples in those chapters use **Hystrix** for resilience, too.

These solutions require the use of a Kubernetes or PaaS environment. Thus, it is no longer possible to just deploy some Docker containers on a Linux server.

HTTP proxy

Functionalities like load balancing and resilience can be implemented with an **HTTP proxy** instead of Ribbon.

This is a further development of the sidecar concept. An example is Envoy (https://github.com/lyft/envoy). This proxy implements some resilience patterns. Envoy is also part of Istio and is used as a sidecar in Istio application.

> **Istio** is a service mesh that supports many technologies for the operation of microservice systems.

With a proxy, the application itself can be kept free of these aspects. Apache httpd or nginx can at least implement load balancing. They could also provide basic features of a sidecar.

Atom

Asynchronous communication (see chapter 6 (https://www.educative.io/collection/page/10370001/5441945024331776/4516786718375936)) seems to be a contradiction to communication via a synchronous protocol like REST. But *Atom* (see chapter 8 (https://www.educative.io/collection/page/10370001/5441945024331776/5669712774037504)) can be combined with concepts from this chapter.

Atom uses REST, so the microservices only need to implement other types of REST resources.

A collaboration with messaging systems like **Kafka** (see chapter 11 (https://www.educative.io/collection/page/10370001/5441945024331776/6652565976514560)) is also conceivable.

However, in this case, the system not only has the complexity of the messaging system but must also offer a REST environment.

Frontend integration

Frontend integration (chapter 3 (https://www.educative.io/collection/page/10370001/5441945024331776/6168726367895552)) works on a different level than REST and can be combined with the Netflix stack.

In particular, integration with **links and JavaScript** (Chapter 4 (https://www.educative.io/collection/page/10370001/5441945024331776/4860469534785536)) is possible without any problems.

With *ESI* (see chapter 5 (https://www.educative.io/collection/page/10370001/5441945024331776/4867833454395392)) Varnish instead of Zuul implements routing. Varnish would have to extract the IP addresses of the microservices from Eureka.

However, this is not possible.

QUIZ

1 Zuul and Hystrix are not maintained that much anymore.

- A) True
- B) False

Submit Answer

Question 1 of 3
0 attempted

Reset Quiz

In the next lesson, we'll look at some experiments that can be conducted with the Netflix stack.

Back

Resilience: Hystrix Implementation

Next

Experiments

Mark as Completed

Report an Issue