Chapter Conclusion

In this lesson, we'll look at a quick conclusion to the chapter.

WE'LL COVER THE FOLLOWING

Setting up a microservices system with Consul is another option for a synchronous system.

Summary

This infrastructure meets the typical challenges of synchronous microservices as follows.

Service discovery

Service discovery is covered by Consul. Consul is very flexible; due to the DNS interface and Consul Template it can be used with many technologies. This is particularly important in the context of microservices. While a system might not need to use a variety of technologies from the start, in the long term it is advantageous to be able to integrate new technologies.

Increased transparency

Consul is **more transparent** to use than Eureka. The Spring Cloud applications still need special Consul configurations, but Consul offers a configuration in Apache format for Apache httpd so at least in this case Consul is transparent.

If Registrator is used to register the microservices, and Consul is used as a DNS server, Consul is **fully transparent** and can be used without any code dependencies. With Envconsul, Consul can even configure the microservices without code dependencies.

Configuration

Consul can be used to **configure** the microservices. In this way, with only one technological approach, both service discovery and configuration can be implemented.

Resilience

Resilience is not implemented in this example.

Routing

Routing with Apache httpd is a relatively common approach. This reduces the technological complexity, which is quite high in a microservices system anyway. With the large number of new technologies and a new architectural approach, it is helpful to cover some areas with established approaches.

Load balancing

Load balancing is implemented with Ribbon, like in the Netflix example (see Load Balancing: Ribbon). However, it is not a problem to provide each microservice instance with an Apache httpd which is configured by Consul Template in such a manner that it provides load balancing for outbound calls. For Consul DNS, Consul even implements load balancing transparently with the DNS server. In that case, no additional technology for load balancing is needed.

Comparison to Netflix

- The technology stack from this example has the **advantage** that it also

- The technology stack from this example has the **advantage** that it also supports heterogeneous microservices systems because there are no more code dependencies on Consul if using DNS.
- Consul as a service discovery technology is more powerful than Eureka with the DNS interface it provides and Consul Template.
- Apache is a standard reverse proxy that is widely used. It is therefore more mature than Zuul.
- Zuul is not supported any more while Apache is still one of the most broadly used web servers.
- For resilience, the stack does not offer a good solution. However, it can still be combined with libraries such as Hystrix.
- The main benefit of the Consul technology stack is its **independence from a concrete language** and environment.
- The Netflix stack is based on Java and it is hard to integrate other languages.
- Netflix has discontinued several of the projects, i.e., Hystrix and Zuul.
- The Consul stack is usually preferable over the Netflix stack.

Advantages

- Consul does not have a Java focus but supports many different technologies.
- Consul supports DNS.
- Consul Template can configure many services (Apache httpd) transparently via configuration files.
- Entirely transparent registration and service discovery with Registrator and DNS are possible.
- The use of well-established technologies such as Apache httpd reduces the risk.

Challenges

Challenges

- Consul is written in Go. Therefore, monitoring and deployment differ from Java microservices.

That’s it for this chapter! From the next chapter onwards, we’ll discuss microservices platforms.