Load Balancing: Ribbon

In this lesson, we'll check out load balancing with Ribbon.

WE'LL COVER THE FOLLOWING

Introduction

Microservices have the advantage that each microservice can be scaled independently of the other microservices.

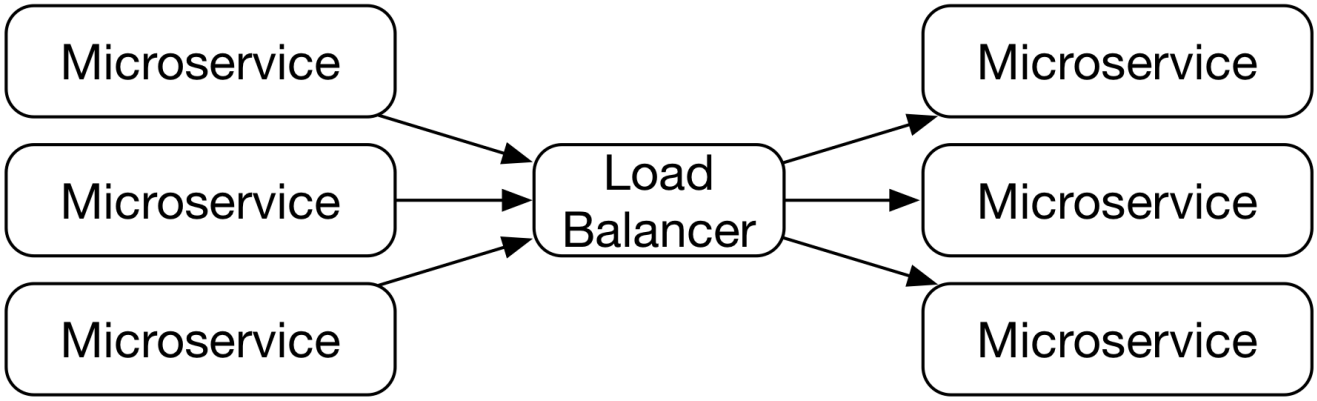
To do this, it is necessary that the call to a microservice can be **distributed to several instances by a load balancer**.

Central load balancer

Typically, a single load balancer is used for all calls. Therefore, a single load balancer, which processes all requests from all microservices, can also be used for an entire microservices system.

However, such an approach leads to a **bottleneck** since all network traffic must be routed through this single load balancer.

The load balancer is also a single point of failure. If the load balancer fails, all network traffic stops functioning and the **entire microservices system fails**.



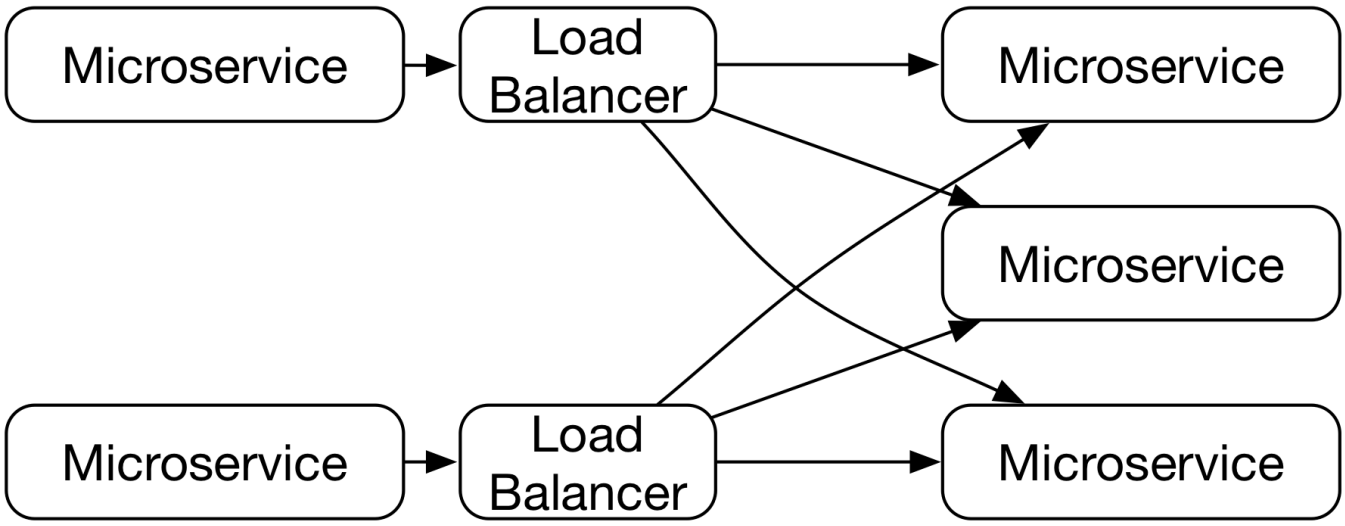
Central Load Balancer

Decentralized load balancing would be better. For this, each microservice must have its own load balancer.

If such a load balancer fails, **only one** microservice will fail.

Client-side load balancing

The idea of client-side load balancing can be implemented by a “normal” load balancer such as Apache httpd or nginx. A load balancer is deployed for each microservice. The load balancer must obtain the information about the currently available microservices from the service discovery.



Client-side Load Balancer

It is also possible to write a library that distributes requests to other microservices to different instances. This library must read the currently available microservice instances from the service discovery and then, for each request, select one of the instances. This is how Ribbon Ribbon works.

Ribbon API

Ribbon offers a relatively simple API for load balancing.

```
private LoadBalancerClient loadBalancer;
  // Spring injects a LoadBalancerClient
ServiceInstance instance = loadBalancer.choose("CUSTOMER");
url = String.format("http://%s:%s/customer/",
 instance.getHost(), instance.getPort());
```

Spring Cloud injects an implementation of the interface `LoadBalancerClient`. First, a call to the `LoadBalancerClient` selects an instance of a microservice. This information is then used to fill a URL to which the request can be sent.

Ribbon supports various strategies for selecting an instance. Thus, approaches other than a simple round robin are feasible.

Ribbon with Consul

As part of the Netflix stack, Ribbon supports Eureka as a service discovery tool, but it also supports Consul. In the Consul example (see chapter 11) the access to the microservices is implemented identical to the Netflix example.

RestTemplate

Spring contains the `RestTemplate` to easily implement REST calls. If a `RestTemplate` is created by Spring and annotated with `@LoadBalanced`, Spring makes sure that a URL like `http://order/` is forwarded to the order microservice. Internally, Ribbon is used for this. https://spring.io/guides/gs/client-side-load-balancing/ shows how this approach can be implemented with a `RestTemplate`.

QUIZ

1 In the following code,

```
private LoadBalancerClient loadBalancer;
```

```
  // Spring injects a LoadBalancerClient
ServiceInstance instance = loadBalancer.choose("ORDER");

url = String.format("http://%s:%s/order/",
 instance.getHost(), instance.getPort());
```

Load balancing is done for which microservice?

COMPLETED 0%

1 of 3

In the next lesson, we'll discuss resilience with Hystrix.