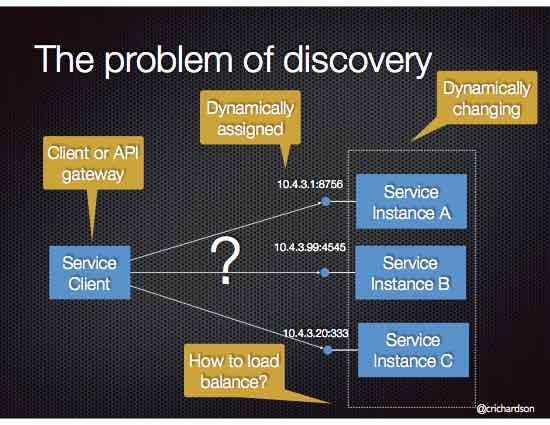
**Client-side service discovery**



a modern microservice-based application typically runs in a virtualized or containerized environments where the number of instances of a service and their locations changes dynamically.

It provides **Eureka**, which is a **Service Registry**, and **Ribbon**, which is an HTTP client that queries Eureka in order to route HTTP requests to an available service instance.

**Service registry**

The Eureka Server is a small Spring Boot application:

@SpringBootApplication

@EnableEurekaServer

**public** **class** **EurekaServer** **{**

**public** **static** **void** **main(**String**[]** args**)** **{**

**new** **SpringApplicationBuilder(**EurekaServer**.**class**).**web**(true).**run**(**args**);**

**}**

**}**

It is deployed using Docker:

**Circuit Breaker**

**Create a Separate Data Store for Each Microservice**

Do not use the same backend data store across microservices. You want the team for each microservice to choose the database that best suits the service. Moreover, with a single data store it’s too easy for microservices written by different teams to share database structures, perhaps in the name of reducing duplication of work. You end up with the situation where if one team updates a database structure, other services that also use that structure have to be changed too.

**Do a Separate Build for Each Microservice**

Do a separate build for each microservice, so that it can pull in component files from the repository at the revision levels appropriate to it. This sometimes leads to the situation where various microservices pull in a similar set of files, but at different revision levels.

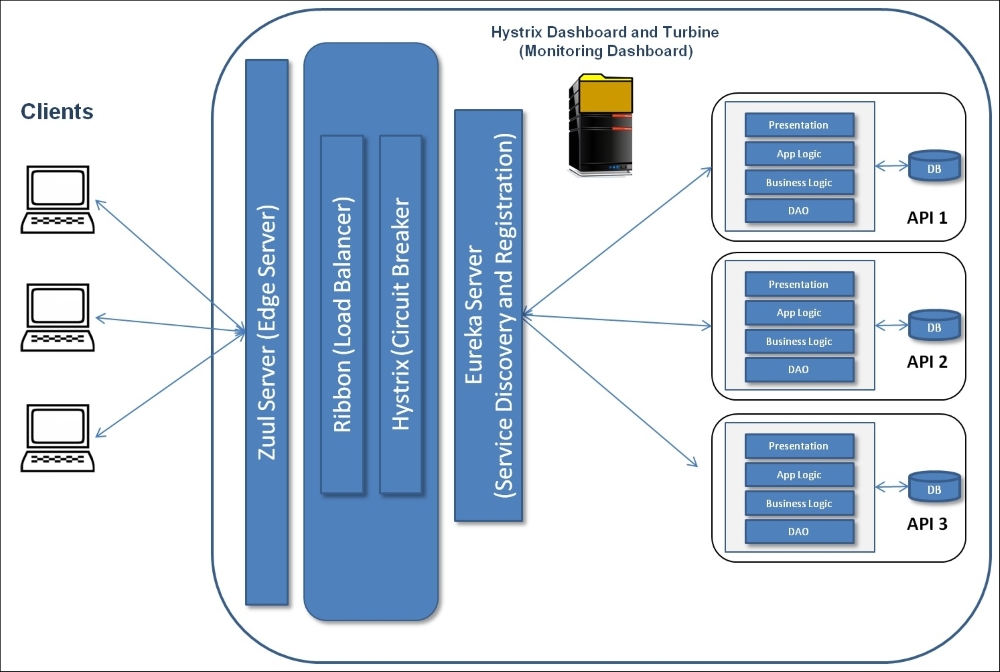
**Deploy in Containers**

Deploying microservices in containers is important because it means you just need just one tool to deploy everything. As long as the microservice is in a container, the tool knows how to deploy it. It doesn’t matter what the container is. That said, Docker seems very quickly to have become the de facto standard for containers.

**Treat Servers as Stateless**

**An overview of microservice architecture using Netflix OSS**

Spring took the opportunity to integrate many Netflix OSS projects, such as Zuul, Ribbon, Hystrix, Eureka Server, and Turbine, into Spring Cloud. This is one of the reasons Spring Cloud provides a ready-made platform for developing production-ready microservices.



## Load balancing

**Load balancing** is required to service requests in a manner that maximizes speed, capacity utilization, and it makes sure that no server is overloaded with requests. The load balancer also redirects requests to the remaining host servers if a server goes down. In microservice architecture, a microservice can serve internal or external requests. Based on this, we can have two types of load balancing – **client-side** and **server-side load balancing**.

## Client-side load balancing

Microservices need interprocess communication so that services can communicate with each other. Spring Cloud uses Netflix Ribbon, a client-side load balancer that plays this critical role and can handle both HTTP and TCP. Ribbon is cloud-enabled and provides built-in failure resiliency. Ribbon also allows you to use multiple and pluggable load balancing rules. It integrates clients with load balancers.

In the last chapter, we added Eureka Server. Ribbon is integrated with Eureka Server in Spring Cloud by default. This integration provides the following features:

## Circuit breaker and monitoring

It tracks the availability of external services such as Eureka Server, API services such as restaurant-service, and so on, and prevents service consumers from performing any action on any service that is not available.

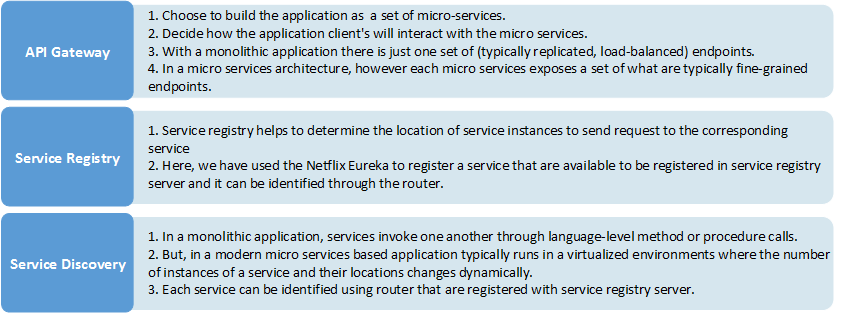
It is another important aspect of microservice architecture, a safety measure (failsafe mechanism) when the service does not respond to a call made by the service consumer – circuit breaker.

## Spring Cloud Netflix features

<https://cloud.spring.io/spring-cloud-netflix/>

<https://cloud.spring.io/spring-cloud-netflix/single/spring-cloud-netflix.html>

**Patterns in Micro services Architecture**



**Major Components of Netflix**



**Netflix Eureka – Service Discovery Server**

Netflix Eureka allows micro services to register themselves at runtime as they appear in the system landscape.



**Netflix Ribbon – Dynamic Routing & Load Balancer**

Netflix Ribbon can be used by service consumers to look up services at runtime. Ribbon uses the information available in Eureka to locate appropriate service instances. If more than one instance is found, Ribbon will apply load balancing to spread the requests over the available instances. Ribbon does not run as a separate service but instead as an embedded component in each service consumer.



**Netflix Zuul – Edge Server**

Zuul is (of course) our gatekeeper to the outside world, not allowing any unauthorized external requests pass through. Zulu also provides a well-known entry point to the micro services in the system landscape. Using dynamically allocated ports is convenient to avoid port conflicts and to minimize administration but it makes it of course harder for any given service consumer. Zuul uses Ribbon to look up available services and routes the external request to an appropriate service instance.