# ribbon with eureka

In this Spring cloud tutorial, learn to use **client side load balancing using Netflix Ribbon** in spring boot/cloud projects. Learn to build microservice based applications which use **ribbon** as client side load balancer and **eureka** as registry service. Learn how we can dynamically add new instances of microservices under the load balancer.

Netflix ribbon from Spring Cloud family provides such facility to set up client side load balancing along with the service registry component. Spring boot has very nice way of configuring ribbon client side load balancer with minimal effort. It provides the following features

1. Load balancing
2. Fault tolerance
3. Multiple protocol (HTTP, TCP, UDP) support in an asynchronous and reactive model
4. Caching and batching

**Zuul Api gateway**

**What is Zuul?**

Zuul is the front door for all requests from devices and web sites to the backend of the Netflix streaming application. As an edge service application, Zuul is built to enable dynamic routing, monitoring, resiliency and security. It also has the ability to route requests to multiple Amazon Auto Scaling Groups as appropriate.

## Why did we build Zuul?

The volume and diversity of Netflix API traffic sometimes results in production issues arising quickly and without warning. We need a system that allows us to rapidly change behavior in order to react to these situations.

Zuul uses a range of different types of filters that enables us to quickly and nimbly apply functionality to our edge service. These filters help us perform the following functions:

* **Authentication and Security** - identifying authentication requirements for each resource and rejecting requests that do not satisfy them.
* **Insights and Monitoring** - tracking meaningful data and statistics at the edge in order to give us an accurate view of production.
* **Dynamic Routing** - dynamically routing requests to different backend clusters as needed.
* **Stress Testing** - gradually increasing the traffic to a cluster in order to gauge performance.
* **Load Shedding** - allocating capacity for each type of request and dropping requests that go over the limit.
* **Static Response handling** - building some responses directly at the edge instead of forwarding them to an internal cluster
* **Multiregion Resiliency** - routing requests across AWS regions in order to diversify our ELB usage and move our edge closer to our members

**Zipkin**

[Zipkin](http://zipkin.io/) is very efficient tool for **distributed tracing** in [microservices](https://howtodoinjava.com/microservices/microservices-definition-principles-benefits/) ecosystem. Distributed tracing, in general, is latency measurement of each component in a distributed transaction where multiple microservices are invoked to serve a single business usecase. Let’s say from our application, we have to call 4 different services/components for a transaction. Here with distributed tracing enabled, we can measure which component took how much time.

This is useful during debugging when lots of underlying systems are involved and the application becomes slow in any particular situation. In such case, we first need to identify see which underlying service is actually slow. Once the slow service is identified, we can work to fix that issue. Distributed tracing helps in identifying that slow component among in the ecosystem.

# **Hystrix**

Learn to leverage the one of the [Spring cloud Netflix](https://howtodoinjava.com/spring/spring-cloud/spring-cloud-service-discovery-netflix-eureka/) stack component called [Hystrix](https://github.com/Netflix/Hystrix" \t "_blank) to implement **circuit breaker** while invoking underlying [microservice](https://howtodoinjava.com/microservices/microservices-definition-principles-benefits/). It is generally required to enable fault tolerance in the application where some underlying service is down/throwing error permanently, we need to fall back to different path of program execution automatically. This is related to distributed computing style of Eco system using lots of underlying Microservices. This is where circuit breaker pattern helps and Hystrix is an tool to build this circuit breaker.