# Spring Cloud & Spring Cloud Netflix

# Agenda

- Configuration Management (Config Server/Client)
- Service Discovery (Eureka)
- Client Side Load Balancing (Ribbon)
- Declarative REST Client (Feign)
- Fault Tolerance & Monitoring (Hystrix)
- Router and Filter (ZUUL)

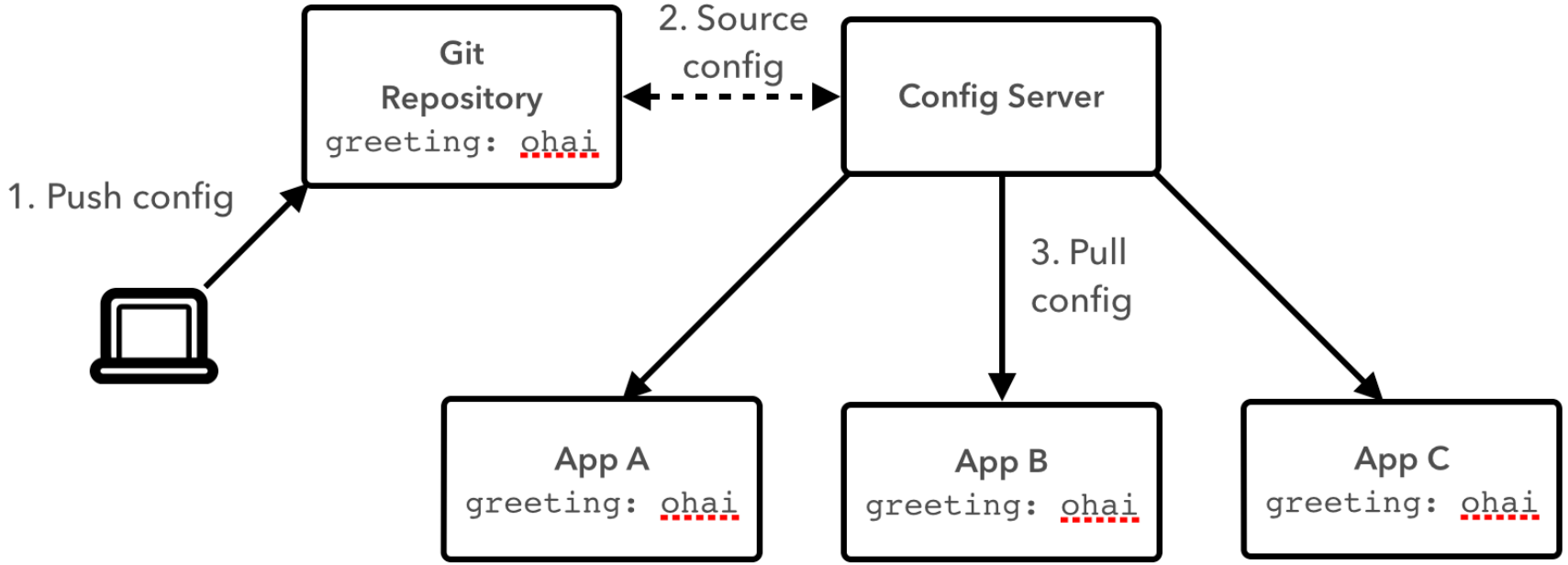
# Overview

- Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems
- Spring Cloud Netflix is the Netflix OSS which was lately moved under spring’s hood
- Spring Cloud Netflix alleviates the mundane concerns associated with building cloud native distributed apps by encapsulating the boiler plate into efficient framework components and also by providing well laid recommended patterns

# Configuration Management

- Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system
- With the Config Server you have a central place to manage external properties for applications across all environments

# Configuration Management Contd.



# Config Server

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-config-server</artifactId>
<version>1.0.4.RELEASE</version>
</dependency>
```

```
spring:
  cloud:
    config:
      server:
        git:
          uri: [config repo]
          search-paths: [name of folder in case any]
```

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
public static void main(String[] args) {
SpringApplication.run(ConfigServerApplication.class,
args);
}
}
```

# Config Client

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-config-client</artifactId>
</dependency>
```

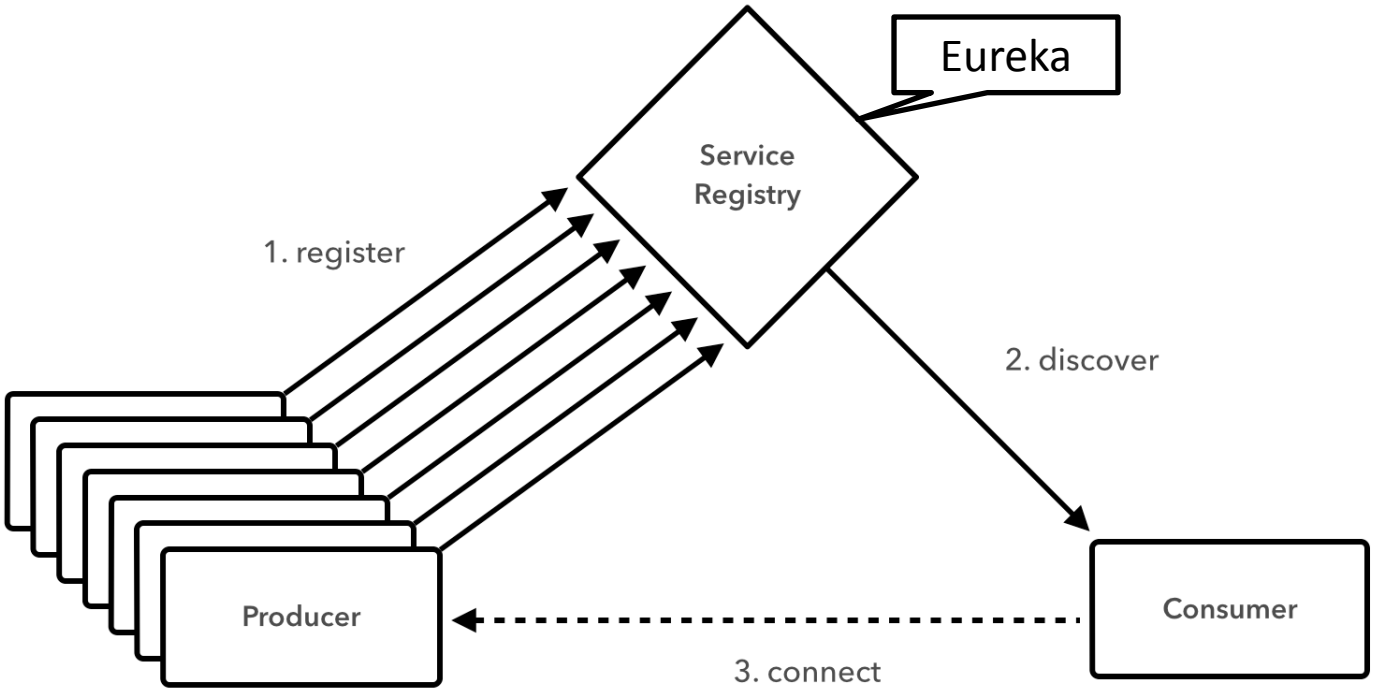
```
spring:
 cloud:
    config:
     uri: [config server root]
```

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
public static void main(String[] args) {
SpringApplication.run(ConfigServerApplication.class,
args);
}
}
```

# Service Discovery

- Service Discovery is one of the key tenets of a micro-service based architecture. Trying to hand configure each client or some form of convention can be very difficult to do and can be very brittle.
- Eureka is the Netflix Service Discovery Server and Client. The server can be configured and deployed to be ***highly available***, with each server replicating state about the registered services to the others.

# Service Discovery Contd.



# Eureka Server

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-config</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-
server</artifactId>
</dependency>
```

```
eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone:
http://${eureka.instance.hostname}:${server.port}/eure
ka/
```

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

public static void main(String[] args) {
SpringApplication.run(EurekaServerApplication.class,
args);
}
}
```

# Eureka Client – Provider Service

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

```
spring:
  application:
    name: exchange-rate-provider
eureka:
  client:
    service-url:
      defaultZone: http://localhost:9090/eureka/
```

```
@SpringBootApplication
@EnableEurekaClient
public class ConversionApplication {

public static void main(String[] args) {
SpringApplication.run(ConversionApplication.class,
args);
}
}
```

# Eureka Client – Consumer Service

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

```
spring:
  application:
    name: conversion-app
eureka:
  client:
    service-url:
      defaultZone: http://localhost:9090/eureka/
```

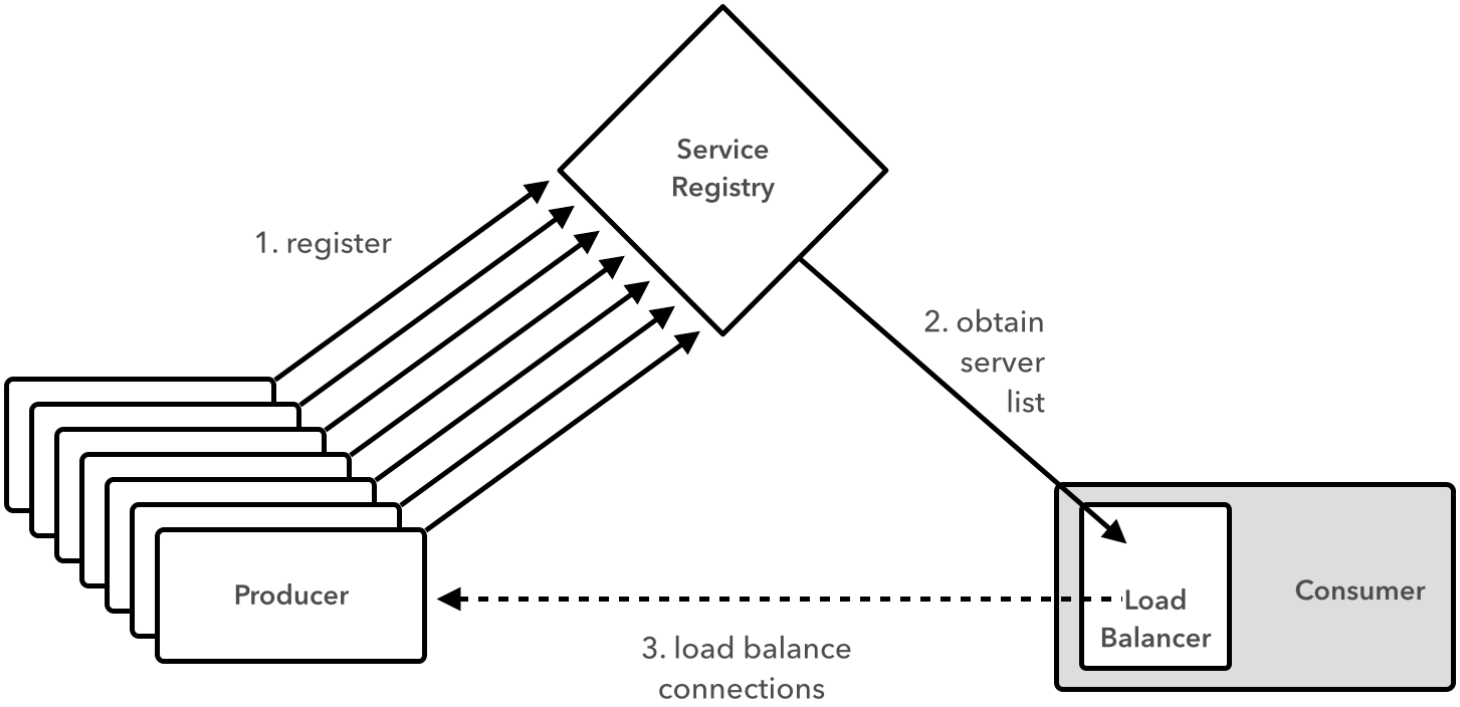
```
@SpringBootApplication
@EnableEurekaClient
public class ConversionApplication {

public static void main(String[] args) {
SpringApplication.run(ConversionApplication.class, args);
}
}
```

```
InstanceInfo instance =
discoveryClient.getNextServerFromEureka("exchange-rate-provider",
false);

RestTemplate restTemplate = new RestTemplate();
conversionFactor =
restTemplate.getForObject(instance.getHomePageUrl()+"exchangeRate?cou
ntryCode={countryCode}”, Float.class, countryCode);
```

# Load Balance (Ribbon)

- Ribbon is a client side load balancer
- Plays well with the rest of Netflix core components especially Eureka

# Load Balance (Ribbon) Contd.



# Consumer Service

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-
parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-
eureka</artifactId>
</dependency>
```

```
@Autowired
private RestTemplate restTemplate;

@RequestMapping(value="/convert")
public Float convert(@RequestParam("amt") Float amount, @RequestParam("cc")
String countryCode, @RequestParam("sit") Integer serviceInvocationType) {

return conversionFactor = restTemplate.getForObject("http://exchange-rate-
provider/exchangeRate?countryCode={countryCode}”, Float.class,
countryCode);
}
```

```
@Autowired
private LoadBalancerClient loadBalancerClient;

@RequestMapping(value="/convert")
public Float convert(@RequestParam("amt") Float amount, @RequestParam("cc")
String countryCode, @RequestParam("sit") Integer serviceInvocationType) {

ServiceInstance serviceInstance = loadBalancerClient.choose("exchange-rate-
provider");
RestTemplate restTemplate = new RestTemplate();
return
restTemplate.getForObject(serviceInstance.getUri()+"/exchangeRate?countryCo
de="+countryCode, Float.class);
}
```

# Declarative REST Client

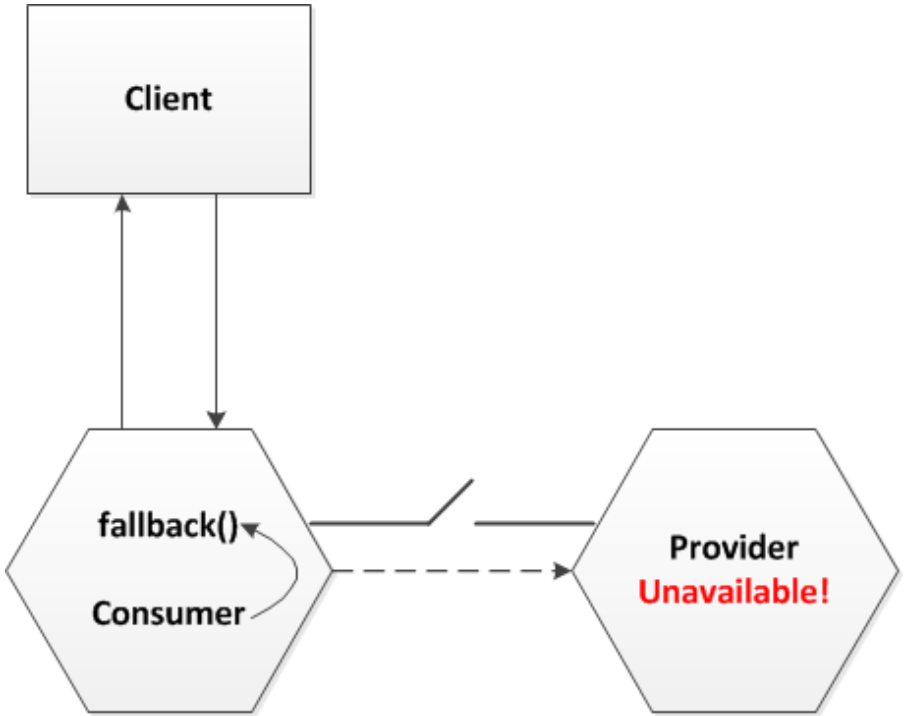
- Feign is a declarative web service client that comes in handy with service to service communication in micro services based architecture
- Spring Cloud integrates Ribbon and Eureka to provide a load balanced http client when using Feign

# Declarative REST Client

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>


<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class ConversionApplication {

public static void main(String[] args) {
SpringApplication.run(ConversionApplication.class, args);
}
}


@FeignClient("exchange-rate-provider")
public interface ExchangeRateClient {

@RequestMapping(value="/exchangeRate", method=RequestMethod.GET)
public String getConversionFactor(@RequestParam("countryCode")
String countryCode);
}
```

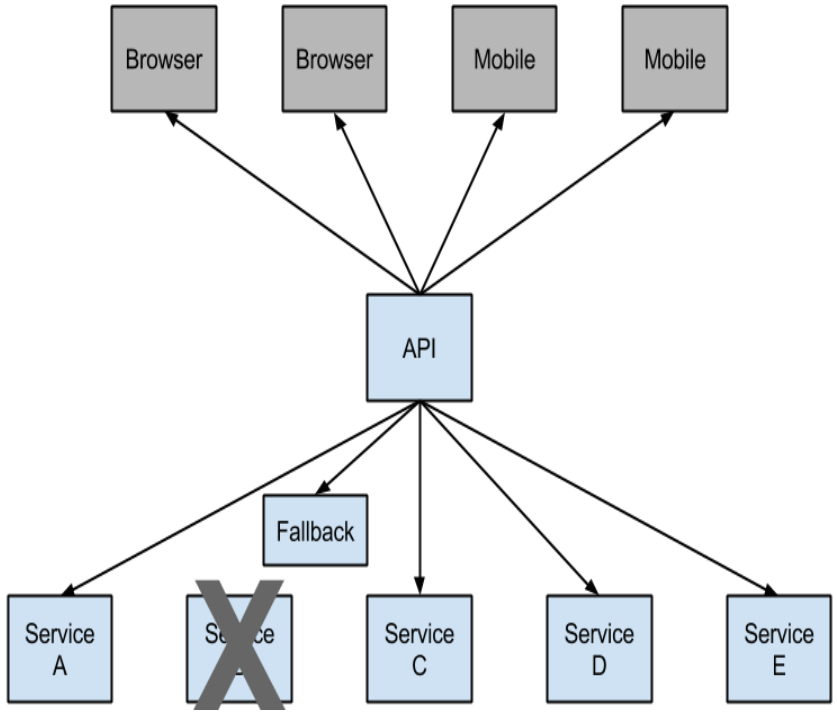
# Fault Tolerance (Circuit Breaker)



# Circuit Breaker Pattern – Hystrix Clients

- Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable

- A service failure in the lower level of services can cause cascading failure all the way up to the user. When calls to a particular service reach a certain threshold (20 failures in 5 seconds is the default in Hystrix), the circuit opens and the call is not made. In cases of error and an open circuit a fallback can be provided by the developer.



# Fault Tolerance – Hystrix Clients

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

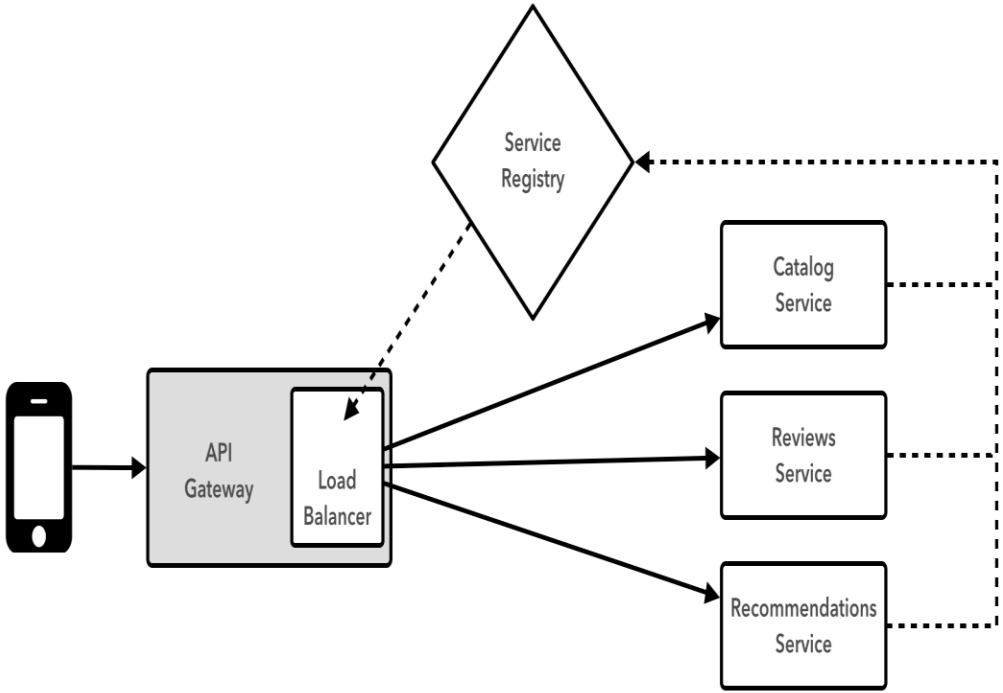
```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker
public class ConversionApplication {

public static void main(String[] args) {
SpringApplication.run(ConversionApplication.class, args);
}
}
```

```
@Component
public class ExchangeRateHystrixClient {

@Autowired
@LoadBalanced
RestTemplate restTemplate;

@HystrixCommand(fallbackMethod = "getConversionFactorFallback")
public String getConversionFactor(String countryCode) {

return restTemplate.getForObject("http://exchange-rate-
provider/exchangeRate?countryCode="+countryCode, String.class);
}

public String getConversionFactorFallback(String countryCode) {
return "0";
}
}
```

# API Gateway Pattern

- Routing in an integral part of a microservice architecture.
- Zuul is a JVM based router and server side load balancer by Netflix



# Reverse Proxy Using ZUUL

```
<parent>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-
parent</artifactId>
<version>1.0.2.RELEASE</version>
<relativePath/>
</parent>
```

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-
zuul</artifactId>
</dependency>
```

```
server:
  port: 8000
zuul:
  routes:
    exchange-rate:
      path: /exchangeRate/**
      strip-prefix: false
      url: http://localhost:7070
    exchange-rate-root:
      path: /
      strip-prefix: false
      url: http://localhost:7070
```

```
@SpringBootApplication
@EnableZuulProxy
public class ZuulReverseProxyApplication {

public static void main(String[] args) {
SpringApplication.run(ZuulReverseProxyApplication.class, args);
}
}
```