Load Balancing – Netflix Zuul

Introduction

The concept of Microservices is one that has forever changed application development and deployment lifecycles by allowing developers to focus on building and maintaining smaller components without sacrificing integration capabilities. However, this is not without drawbacks, as it usually introduces additional complexity which often manifests through routing, load balancing, additional security challenges.

In this context, Netflix Zuul is one of the solutions that system might employ in order to ensure better scalability, resiliency and efficiency of a miscoservice-based system. Similarly to other existing tools (eg. Nginx), Netflix Zuul represents the main entrypoint for client requests. In this way, it ensures additional security, and balances the work-load across existing instances in order to minimize potential downtime due to overloading the system.

How to?

Step 1: Create the Eureka Server Project

1. Create a new project
2. Select building tool (this example will use gradle) and Java Version (at least Java 8)
3. Finish creation
4. Add required dependencies in build.gradle:

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-
netflix-eureka-server'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

5. Create main class for the Spring Boot Application

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServiceApplication {

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

}
```

6. Edit application.properties

```
spring.application.name=eureka-service
server.port=8761

#telling the server not to register himself in the service
eureka.client.register-with-eureka=false
```

```
#Eureka clients fetch the service registry (ServiceInstance: {URL, PORT,
HOST}) from the Eureka server
eureka.client.fetch-registry=false
```

7. Build the project (eg. gradle build)
8. Create docker image using the following docker file:

```
FROM openjdk:17-jdk-alpine
RUN addgroup -S springdocker && adduser -S springdocker -G springdocker
USER springdocker:springdocker
ARG JAR_FILE=build/libs/eureka-service-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar","/app.jar"]
EXPOSE 8761
```

Step 2: Create the Zuul Gateway Project

1. Create new project
2. Select building tool (this example will use gradle) and Java Version (recommended 8 to avoid additional configurations)
3. Finish creation
4. Add required dependencies in build.gradle:

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-
netflix-eureka-client'
    implementation 'org.springframework.cloud:spring-cloud-starter-
netflix-zuul'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

5. Create main application class. Make sure to add cors configuration to avoid being blocked by same-origin policy

```
@SpringBootApplication
@EnableZuulProxy
@EnableEurekaClient
public class ZuulGatewayServiceApplication {

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

    @Bean
    public WebMvcConfigurer corsConfigurer() {
       return new WebMvcConfigurer() {
          @Override
          public void addCorsMappings(CorsRegistry registry) {
             registry.addMapping("/**").allowedOrigins("*");
          }
       };
    }

}
```

6. Edit application.properties

```
spring.application.name=gateway-service
server.port=8765

zuul.ignored-headers=Access-Control-Allow-Credentials, Access-Control-
Allow-Origin
#Pass the headers from gateway to sub-microservices.
zuul.sensitiveHeaders=Cookie,Set-Cookie

zuul.prefix=/api
#When path starts with /api/user/**, redirect it to user-service.
zuul.routes.user.path=/user/**
zuul.routes.user.serviceId=user-management
#When path starts with /api/**, redirect it to event-service.
zuul.routes.event.path=/service/**
zuul.routes.event.serviceId=events-management

#eureka
eureka.client.service-url.default-zone=http://eureka-
container:8761/eureka/
#indicates the frequency the client sends heartbeats to indicate that it
is still alive.
eureka.instance.lease-renewal-interval-in-seconds=30
#indicates the duration the server waits since it received the last
heartbeat before it can evict an instance from its registry
eureka.instance.lease-expiration-duration-in-seconds=90

#load balancing
ribbon.eureka.enabled=true
eureka.client.fetch-registry=true
eureka.client.register-with-eureka=true
ribbon.ServerListRefreshInterval=5000

#timeout
#this will help you load services eagerly. Otherwise for first time, we
will get timeout exception.
zuul.ribbon.eager-load.enabled=true
#The read timeout in milliseconds. Default is 1000ms
ribbon.ReadTimeout=60000
#The Connection timeout in milliseconds. Default is 1000ms.
ribbon.ConnectTimeout=10000



management.endpoints.web.exposure.include=*
management.endpoint.routes.enabled=true
management.endpoint.gateway.enabled=true
management.endpoint.health.show-details=always
```

7. Build the project (eg. gradle build)
8. Create docker image using the following docker file:

```
FROM openjdk:8-jdk-alpine
RUN addgroup -S springdocker && adduser -S springdocker -G springdocker
USER springdocker:springdocker
ARG JAR_FILE=build/libs/zuul-gateway-service-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
```

```
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8765
```

Step 3: Create a microservice

1. Create a new project
2. Select building tool (this example will use gradle) and Java Version (at least Java 8)
3. Finish creation
4. Add required dependencies in build.gradle

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.cloud:spring-cloud-starter-
netflix-eureka-client'
    }
```

5. Edit application.properties

```
spring.application.name=user-management
server.port=8080


#eureka
eureka.client.service-url.default-zone=http://eureka-
container:8761/eureka/
#indicates the frequency the client sends heartbeat to server to indicate
that it is alive.
eureka.instance.lease-renewal-interval-in-seconds=30
#indicates the duration the server waits since it received the last
heartbeat before it can evict an instance from its registry
eureka.instance.lease-expiration-duration-in-seconds=90

#load balancing
ribbon.eureka.enabled=true
```

6. Build the project (eg. gradle build)
7. Create docker image using the following docker file (optional, but this is assumed to be done for the rest of the tutorial)

```
FROM openjdk:17-jdk-alpine
RUN addgroup -S springdocker && adduser -S springdocker -G springdocker
USER springdocker:springdocker
ARG JAR_FILE=build/libs/user-management-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8080
```

Step 4: Run and test the setup

Create a docker-compose file in order to ease the process of setting up the environment and run the configuration using docker-compose up -d

```
services:
    eureka-container:
        image: eureka-image
        ports:
            - "8761:8761"
        networks:
            - app-network

    user-container:
        image: user-image
        ports:
            - "8080:8080"
        depends_on:
            - eureka-container
        networks:
            - app-network
        environment:
            server.port: 8080
            eureka.client.serviceUrl.defaultZone: http://eureka-
container:8761/eureka/


    event-container:
        image: event-image
        ports:
            - "8081:8081"
        depends_on:
            - eureka-container
        networks:
            - app-network
        environment:
            server.port: 8081
            eureka.client.serviceUrl.defaultZone: http://eureka-
container:8761/eureka/

    zuul-container:
        image: zuul-image
        ports:
            - "8765:8765"
        depends_on:
            - eureka-container
        networks:
            - app-network
        environment:
            server.port: 8765
            eureka.client.serviceUrl.defaultZone: http://eureka-
container:8761/eureka/
networks:
    app-network:
        driver: bridge
```

In this example, we have the eureka and zuul project setup and ready to ensure mapping and communication between the outside and the system, as well as ensure communication between the 2 sub-systems: user-management and event-management. It is important to notice that the mapping of our endpoints is provided by zuul. For example, in order to reach a @GetMapping("/users") in our user management example, the actual url would look something like this: **http://localhost:8765/api/user/users**.

And just like that, a basic application using microservices architecture has been setup and ready for extension.