Savu Ana
258/2

Zuul Tutorial

Microservices have revolutionized the way we construct apps, allowing us to create smaller, more focused components that integrate smoothly. While microservices have numerous benefits, they also present issues in routing, load distribution, and keeping a coherent interface. Netflix Zuul is an effective tool for addressing these difficulties and creating a durable, scalable, and efficient microservices environment.

Netflix Zuul is a popular API gateway and load balancer that makes routing and load distribution easier in microservices systems. It serves as the primary entry point for all client requests, offering functionality like as authentication, routing, rate restriction, and others. Zuul achieves load balancing by evenly spreading incoming requests over many instances of a service, hence improving performance and dependability.

Prerequisites

- Java version 8 or higher
- Maven or Gradle (the latest will be used for this tutorial)
- Optional (but better) an IDE (Intellij, Visual Studio Code)

Implementation of Netflix Zuul for load balancing

Step 1. Create Eureka Server project

1. *Generate Project with Spring Initializr*

- Go to Spring Initializr
- Choose your preferred project metadata (Group, Artifact, Name, Description) and the package name.
- Select the Spring Boot version (make sure it's compatible with Spring Cloud).
- Add the "Eureka Server" dependency by searching for it in the dependencies section.
- Click on "Generate" to download your project template.

2. *Configure Eureka Server*

After extracting the downloaded project:

- ✓ Open the `build.gradle` (for Gradle) file and ensure the Eureka Server dependency is present:

```
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
```

- ✓ Enable the Eureka Server in your Spring Boot application. Open the `@SpringBootApplication` class (usually `Application.java` or `Application.kt`) and annotate it with `@EnableEurekaServer`. For example:

```
@SpringBootApplication
@EnableEurekaServer
public class YourApplication {
```

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

3. *Configure Application Properties*

In `src/main/resources/application.properties` or `application.yml`, add the necessary Eureka server configuration. For a basic setup, you might use:

- For `application.properties`:

```
server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false
```

4. *Run your Eureka Server* (`gradlew clean build  gradlew bootRun`)

Step 2. Create Zuul Gateway project

1. *Generate Project with Spring Initializr*

- Go to Spring Initializr
- Choose your preferred project metadata (Group, Artifact, Name, Description) and the package name.
- Select the Spring Boot version (make sure it's compatible with Spring Cloud).
- Add the “Zuul" dependency by searching for it in the dependencies section.
- Click on "Generate" to download your project template.

2. *Configure Zuul Proxy*

✓ If Zuul was not directly selectable in Spring Initializr or if you're setting up the project manually, ensure the following dependencies are present in your `build.gradle` (for Gradle) file:

```
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-zuul'
```

✓ Modify your main application class to enable Zuul Proxy. Open the `@SpringBootApplication` class (usually named `Application.java` or similar) and add `@EnableZuulProxy` annotation:

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

3. *Configure Application Properties*

Configure routes in `src/main/resources/application.properties` or `application.yml`. Here's an example configuration:

```
zuul.prefix=/api
zuul.routes.movies.url=http://localhost:8081
```

```
zuul.routes.movies.path=/movies/**
zuul.routes.movies.serviceId=movie-service
```

4. *Run your Zuul Gateway*(`gradlew clean build  gradlew bootRun`)

Step 3. Create a Microservice

1. When creating a new microservice, don't forget to add the necessary Spring Boot and Eureka client dependencies in build.gradle:

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

2. Configure application.properties:

```
eureka.client.service-url.default-
zone=http://host.docker.internal:8761/eureka/
eureka.instance.lease-renewal-interval-in-seconds=30
eureka.instance.lease-expiration-duration-in-seconds=90
```

Step 4. Run and Test your Setup

1. Run the Eureka server
2. Run the Zuul gateway
3. Run the microservice
4. Access the microservice through the Zuul gateway: Open a browser or use a tool like curl to send a request to an implemented endpoint in the microservice. You should see the response from the microservice

This configuration enables you to implement a simple microservices architecture using Eureka for service discovery and Zuul for routing. You can add more microservices and configure Zuul to handle more complex routing and load balancing as needed.