Zuul – API Gateway for Microservices

Zuul is an edge service that provides dynamic routing, monitoring, resiliency, security, and more.

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.

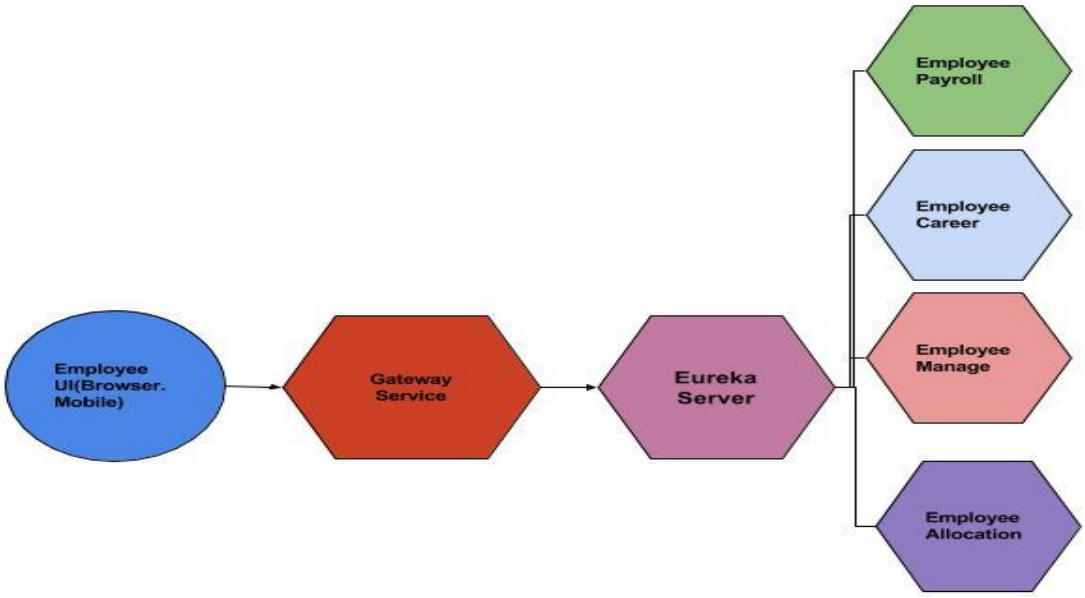
Zuul uses a range of different types of filters that enables us to quickly and nimbly apply functionality to our edge service. These filters help us perform the following functions:

- 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

Edge Service: Zuul acts as an **API gateway** or **Edge service**. It receives all the request comes from UI and then delegates the request to internal Microservices. So we have to create a brand new Microservice which is Zuul enabled and this service sits on top of all other Microservices. It acts as an Edge service or Client facing service. Its Service API should be exposed to client/UI, Client calls this service as a proxy for Internal Microservice then this service delegates the request to appropriate service.

The advantage of this type of design is common aspects like CORS, Authentication, Security can be put into a centralized service, so all common aspects will be applied on each request, and if any changes occur in the future we just have to update the business logic of this Edge Service.

Also, we can implement any Routing rules or any Filter implementation says we want to append a special tag into the request header before it reaches to internal Microservices we can do it in the Edge service. As Edge service itself is a Microservice so it can be independently scalable, deployable. So we can perform some load testing also.

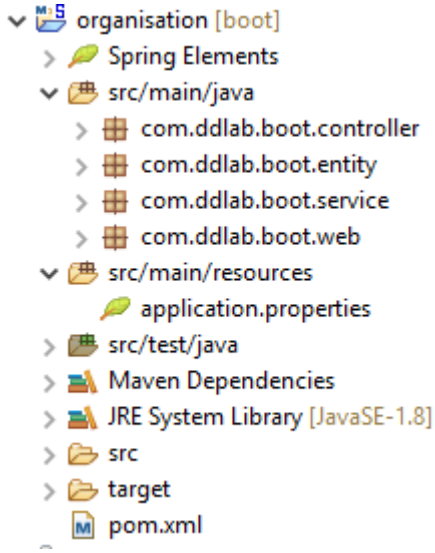


Zuul Enabled Edge Service

Let us consider a practical example. There are two microservices like **Organisation** and **Employee**. Both the microservices provide REST end points and provide data and these microservices can accessed directly without using Zuul. In case the microservices move to different location or different port, we have to change our application with REST end point URI. In case of Zuul, we configure all the microservices and access the rest end points through Zuul API gate way.

Source code for Organisation – A simple Microservice

Project Structure



Maven (pom.xml)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>organisation</groupId>
	<artifactId>organisation</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>organisation</name>
	<url>http://maven.apache.org</url>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.0.RELEASE</version>
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

SpringBoot Main Application – Web Layer

OrganisationBootApplication.java

```
package com.ddlab.boot.web;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class OrganisationBootApplication {

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

BootConfig.java

```
package com.ddlab.boot.web;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@ComponentScan(basePackages= {"com.ddlab.boot.*"})
@Configuration
public class BootConfig {

}
```

Entity Layer

Location.java

```
package com.ddlab.boot.entity;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@JsonPropertyOrder({"officeName", "streetName", "city", "pincode"})
public class Location {

  @JsonProperty private String officeName;
  @JsonProperty private String streetName;
  @JsonProperty private String city;
  @JsonProperty private String pincode;
}
```

Service Layer

OrgService.java

```
package com.ddlab.boot.service;
import com.ddlab.boot.entity.Location;

public interface OrgService {
      Location getLocationByPincode(String pincode);
}
```

OrgServiceImpl.java

```
package com.ddlab.boot.service;
import org.springframework.stereotype.Service;
import com.ddlab.boot.entity.Location;

@Service
public class OrgServiceImpl implements OrgService {

  @Override
  public Location getLocationByPincode(String pincode) {
    Location location = new Location();
    location.setOfficeName("DDLAB INC");
    location.setCity("Bangalore, Karnataka");
    location.setPincode(pincode);
    location.setStreetName("13th main road, 5th block");
    return location;
  }
}
```

Resource or Controller Layer

OrgController.java

```
package com.ddlab.boot.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.ddlab.boot.entity.Location;
import com.ddlab.boot.service.OrgService;

@RestController
@RequestMapping("/home")
@CrossOrigin
public class OrgController {

  @Autowired private OrgService orgService;

  //http://localhost:8081/org/home/location/12345
  @GetMapping(path = "/location/{pincode}", produces =
MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Location> getLocationByPincode(@PathVariable("pincode")
String pincode) {
    System.out.println("Org Service :::" + orgService);
    return new ResponseEntity<Location>(orgService.getLocationByPincode(pincode),
HttpStatus.OK);
  }
}
```

Spring Boot Configuration

Application.properties

```
spring.application.name=org
server.servlet.context-path=/org
server.port=8081
server.error.whitelabel.enabled=false
```

How to use

Start the server and access the following URL in browser

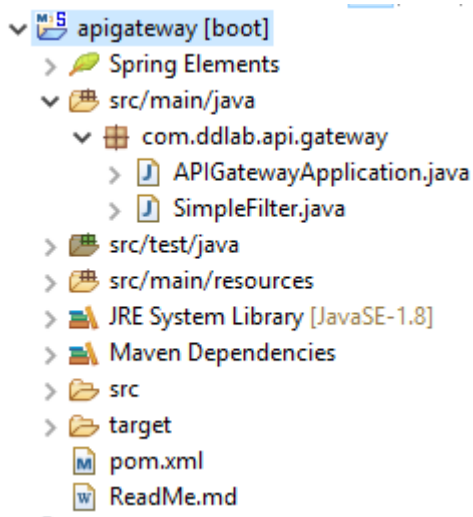
```
http://localhost:8081/org/home/location/12345.
```

Now accordingly develop employee service, it is just similar to Organisation service, the only difference is port number which 8082.

Zuul API Gateway

It is also a SpringBoot microservice with Netflix specific libraries.

Project Structure



Maven Configuration (pom.xml)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>apigateway</groupId>
	<artifactId>apigateway</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>apigateway</name>
	<url>http://maven.apache.org</url>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.0.RELEASE</version>
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>
```

```
	<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>org.springframework.cloud</groupId>
				<artifactId>spring-cloud-dependencies</artifactId>
				<version>Finchley.RELEASE</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
	</dependencyManagement>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-config</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-zuul</artifactId>
		</dependency>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
</project>
```

SpringBoot Main Application

APIGatewayApplication.java

```
package com.ddlab.api.gateway;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import org.springframework.context.annotation.Bean;

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

  @Bean
  public SimpleFilter simpleFilter() {
    return new SimpleFilter();
  }
}
```

SimpleFilter.java

```
package com.ddlab.api.gateway;
import javax.servlet.http.HttpServletRequest;
import com.netflix.zuul.context.RequestContext;
import com.netflix.zuul.ZuulFilter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SimpleFilter extends ZuulFilter {
  private static Logger log = LoggerFactory.getLogger(SimpleFilter.class);

  public String filterType() {
    return "pre";
  }

  public int filterOrder() {
    return 1;
  }

  public boolean shouldFilter() {
    return true;
  }

  public Object run() {
    RequestContext ctx = RequestContext.getCurrentContext();
    HttpServletRequest request = ctx.getRequest();
    log.info( String.format("%s request to %s", request.getMethod(),
request.getRequestURL().toString()));
    return null;
  }
}
```

Zuul API Gateway Configuration

Application.properties

```
#API Getway Configuration for Organization service
zuul.routes.org.url=http://localhost:8081
zuul.routes.org.path=/org/**
zuul.routes.org.serviceId=org
zuul.routes.org.stripPrefix=false

#API Getway Configuration for Employee service
zuul.routes.emp.url=http://localhost:8082
zuul.routes.emp.path=/emp/**
zuul.routes.emp.serviceId=emp
zuul.routes.emp.stripPrefix=false

zuul.prefix=/api
server.port=8085
server.error.whitelabel.enabled=false
```

How to use

1. Start organization service in port 8081
2. Start employee service in port 8082
3. Finally start apigateway service in port 8085.

Now access the following URIs

http://localhost:8085/api/org/home/location/12345

http://localhost:8085/api/emp/home/id/11