

# Spring Gateway

## Gold - Chapter 7 - Topic 2

Selamat datang di **Chapter 7 Topic 2**  
online course **Back End Java** dari  
**Binar Academy!**



### Welcome Back! ☺ ✨

Masih inget nggak pada topic sebelumnya kita belajar apa? Buat kalian yang menjawab Spring Kafka, selamat ya kalian emezing~

Di topic ini, kita akan kupas tuntas materi Spring Gateway mulai dari konsep API Gateway, fungsi penggunaannya, jenis-jenisnya sampai dengan pengaplikasiannya menggunakan tools Spring Gateway.

Beli pita sambil makan kue cucur. Yuk kita langsung meluncur~



**Dari sesi ini, kita bakal bahas hal-hal berikut:**

- Spring Gateway Introduction
- Penggunaan Spring Gateway pada use case
- Konfigurasi dan implementasi Spring Gateway



Kamu tahu nggak siapa yang meneruskan request dari user ke aplikasi? Yup, API Gateway!

Nah, kali ini kita bakal bahas tentang salah satu API Gateway, yakni Spring Gateway.

Penasaran? Yuk, langsung menyelam ke materi **Spring Gateway introduction~**



### Microservices itu keren banget, tapi ada rawannya juga ☺

Dengan menerapkan arsitektur microservices, mau nggak mau kita akan berhadapan dengan banyaknya microservices yang akan kita kelola secara bersamaan, guys.

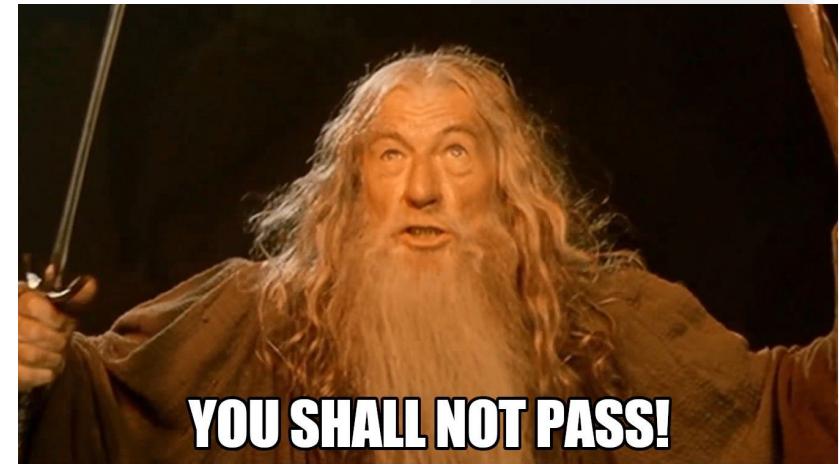
Selain itu, jumlah service yang banyak dan lokasinya (host+port) yang dapat berubah sewaktu-waktu akan menyulitkan user (mobile app, web frontend, service lain) ketika mereka mengakses service-service tersebut secara individual.



Berdasarkan masalah tersebut, **maka dibutuhkan entitas yang bertindak sebagai penerus serta penengah di antara user & aktual service kita.**

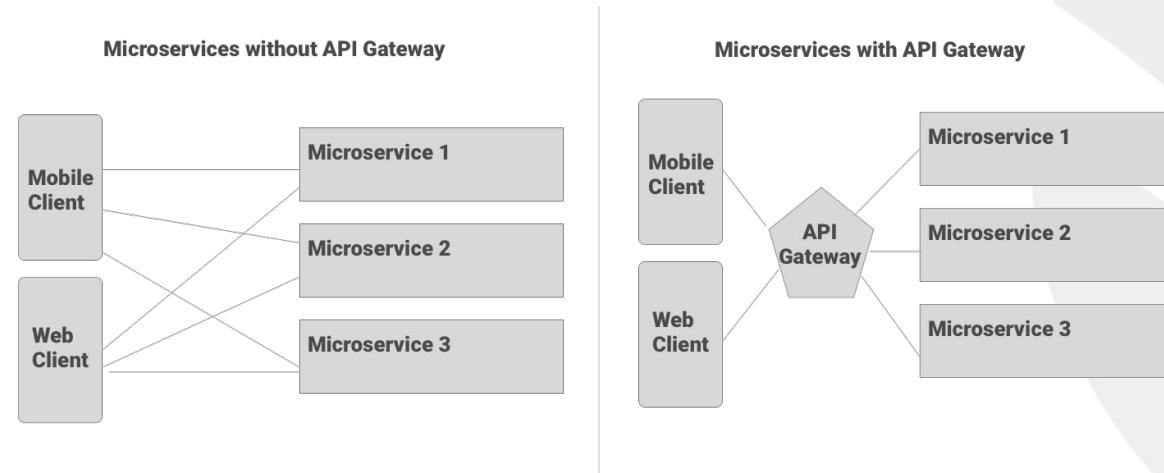
Penengah ini disebut dengan API Gateway.

API Gateway adalah aplikasi/server yang bertindak sebagai single entry point pada sistem yang kita bangun. Dengan kata lain, API Gateway bertindak sebagai satu-satunya “gerbang” dari luar ke dalam.



## Luar? Dalam? Coba jelaskan lagi dong gimana maksudnya?

Kalem-kalem, sini duduk dulu sembari kita bahas perlahan-lahan. Luar di sini bisa kita ibaratkan sebagai user, sedangkan dalam adalah aplikasi microservices.



Simpelnya, seluruh aktivitas traffic dari luar harus melewati API Gateway sebelum menuju ke aplikasi microservices yang kita buat.

Nggak cuma itu, API Gateway dapat menangani request dari user, guys.

Nah sebagai lapisan keamanan, API Gateway juga melakukan pengecekan apakah setiap request dari user diperbolehkan untuk dilanjutkan atau tidak.



### Cuma itu aja? Tentu tidak dong□

Selain yang sudah disebutin tadi, API Gateway juga bertindak sebagai middleware, sehingga sistem internal yang kita bangun di belakangnya bisa kita enkapsulasi deh!

Lebih jauh lagi, API Gateway bisa kita beri tanggung jawab lain seperti logging, authentication, rate limiting, caching, transforming, dan load balancing loh! Canggih banget nggak tuh□



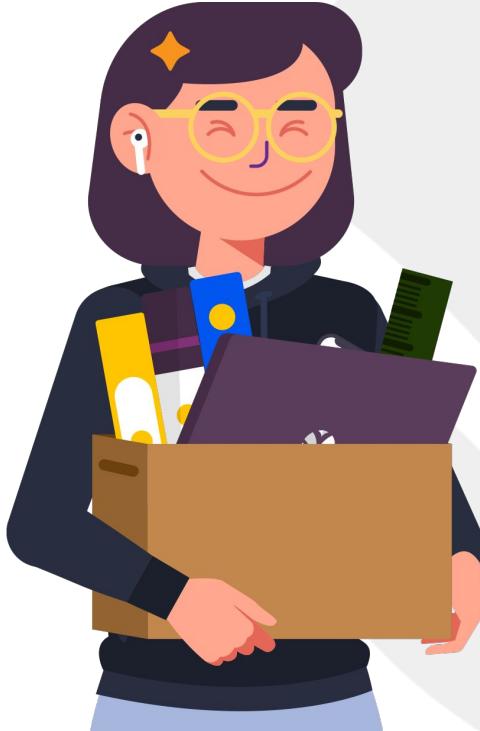
## Keuntungan dari API Gateway

Tentu kita menggunakan API Gateway karena ada banyak keuntungannya, kan? Nah, ini dia beberapa di antaranya.

- Meningkatkan keamanan microservices karena dapat membatasi akses eksternal ke service kita
- Masalah cross-cutting seperti authentication, monitoring/metrics, dan resilience perlu diterapkan hanya di API Gateway karena semua panggilan akan dialihkan ke API Gateway saja



- User nggak bisa mengetahui arsitektur internal system microservices yang kita bangun
- Menyederhanakan interaksi user karena mereka hanya perlu mengakses satu layanan



Kalo ngomongin tentang API gateway, ada beberapa opsi layanan yang bisa kita pakai, yaitu:

- [Zuul from Netflix](#)
- [Kong](#)
- [NGINX](#)
- [HAProxy](#)
- [Traefik](#)
- [Amazon API Gateway](#)
- [Google Cloud Endpoints](#)
- [Spring Cloud Gateway](#)



Eittss tenang, khusus di course ini, kita cuma bahas Spring Cloud Gateway aja~

### Spring Gateway, API Gateway khusus buat kamu si Java lovers ☐

Spring Gateway adalah API Gateway yang dibangun di atas Spring Ecosystem, yaitu Spring 5, Spring Boot 2 dan Project Reactor.

Seperti apa sih itu? Jadi, Spring Gateway menyediakan mekanisme routing out-of-the-box yang sering digunakan di microservices sebagai cara untuk menyembunyikan multiple service di belakang single facade.

Menariknya lagi, Spring Gateway ini open source dan free. Mantap betul! ☐



Spring Gateway ini adalah non blocking API. Artinya, **ketika digunakan, thread selalu tersedia untuk memproses request masuk.**

Request ini nantinya akan diproses secara asynchronous. Setelah selesai diproses, response pun akan dikembalikan.

Jadi, nggak ada request masuk yang diblock ketika kita pakai Spring Gateway. Udah kepo mau cobain ya? Yuk, simak sampai habis dulu □



Spring Gateway memiliki 3 bagian penting, yaitu:

- **Route**

Route adalah building block utama dari gateway yang berisi hal-hal di bawah ini.

1. ID atau nama dari route
2. Destination, yang menjelaskan ke mana harus meneruskan request
3. Predicate dan filter



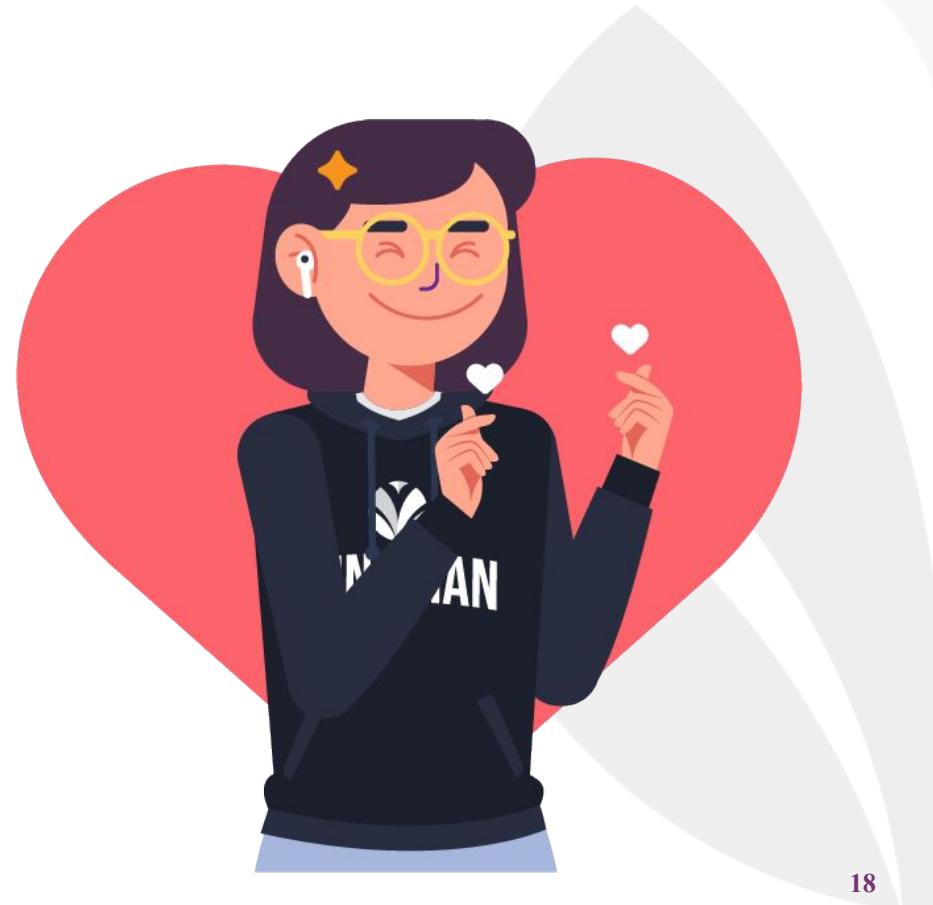
- **Predicate**  
Predicate merujuk pada kumpulan kriteria yang harus cocok dengan request masuk untuk diteruskan ke internal microservices.
- **Filter**  
Sementara filter adalah tempat untuk memodifikasi request masuk sebelum request dikirim ke internal microservices atau sebelum direspon kembali ke user.



### Fitur-fitur yang ada di Spring Gateway

Agar bisa memanfaatkannya, kita juga harus kenal fitur di dalam Spring gateway dong~

- Bisa mencocokkan route pada request attribute apapun
- Predicate dan filter khusus untuk route
- Integrasi circuit breaker
- Integrasi Spring Cloud DiscoveryClient
- Mudah untuk menulis Predicate dan Filter
- Request time limiting
- Path rewriting



Sip, kita udah tahu nih Spring Gateway fungsinya untuk apa.

Kali ini, kita bakal kupas tuntas penggunaan Spring Gateway pada use case. Mari kita meluncur ke TKP~



**Berikut ini adalah use case yang bisa diselesaikan oleh Spring Gateway**

- API Service discovery dan routing
- A&A Security
- API Rate limiting untuk client
- Impose common policies
- API Caching
- Mengontrol API traffic



Lanjut~

- Circuit breaker dan monitoring
- Path filtering
- API performance untuk redundant data request
- High cost and heavy H/W
- Throttling API
- Loose security



Selanjutnya apa ya? Di depan ada materi yang membahas **Spring Gateway configuration & implementation** nih!

Nggak usah berebut, semua kebagian kok □

Yuk, digeser lagi slidenya~

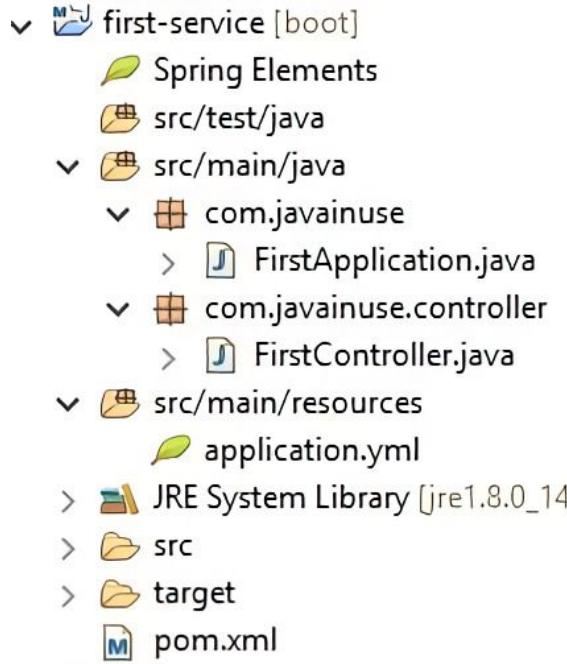


Dengan Spring Gateway, kita bisa membuat route dengan salah satu dari dua cara berikut.

- **Menggunakan konfigurasi property based** yaitu application.properties atau application.yml
- **Menggunakan konfigurasi Java-based** untuk membuat route secara terprogram

Nah, di topic ini kita bakal melakukan implementasi menggunakan kedua cara konfigurasi tersebut untuk route request dua layanan microservices tergantung pada pola URL-nya. Are you ready? □





## Kita mulai dengan implementasi microservices yang pertama

Untuk melakukannya, ikuti project Maven seperti gambar di samping~

Lalu, ikuti pom.xml seperti gambar di bawah ini~

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.javainuse</groupId>
  <artifactId>first-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

</project>
```

Kemudian tentukan application.yml sebagai berikut.

```
spring:  
  application:  
    name: first-service  
server:  
  port: 8081
```

Setelah itu, buat Controller class yang mengekspos GET REST service!

```
package com.javainuse.controller;

import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/employee")
public class FirstController {

    @GetMapping("/message")
    public String test() {
        return "Hello JavaInUse Called in First Service";
    }
}
```

Yang terakhir, buat Bootstrap class dengan annotation @SpringBootApplication seperti pada gambar.

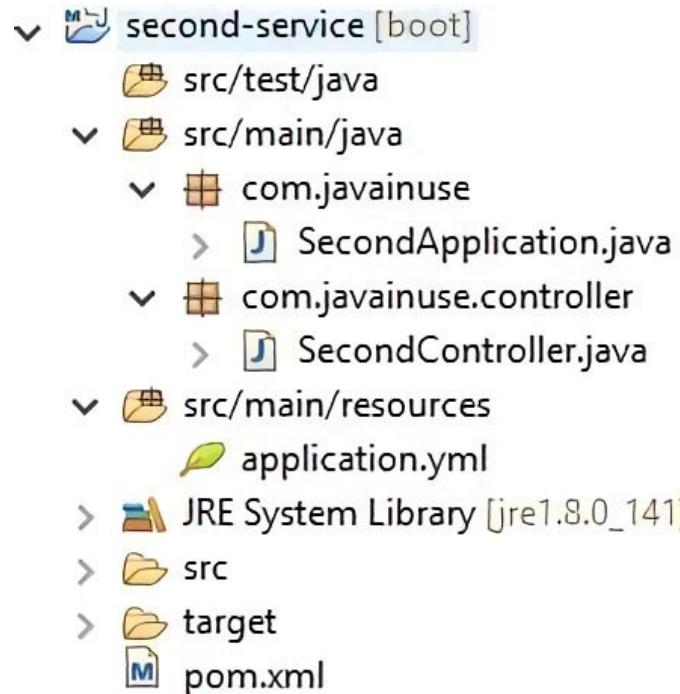
```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class FirstApplication {

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

}
```



Yang pertama sudah selesai □  
Kita lanjut dengan implementasi  
microservices yang kedua

Seperti tadi, ikuti project Maven seperti gambar di samping

Selanjutnya ikuti pom.xml seperti gambar di bawah ini~

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.7.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.javaInuse</groupId>
  <artifactId>second-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

</project>
```

Kemudian, tentukan application.properties sebagai berikut.

```
spring:  
  application:  
    name: second-service  
server:  
  port: 8082
```

Buat Controller class yang mengekspos GET REST service.

```
package com.javainuse.controller;

import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/consumer")
public class SecondController {

    @GetMapping("/message")
    public String test() {
        return "Hello JavaInUse Called in Second Service";
    }

}
```

Terakhir, buat Bootstrap class dengan annotation  
@SpringBootApplication.

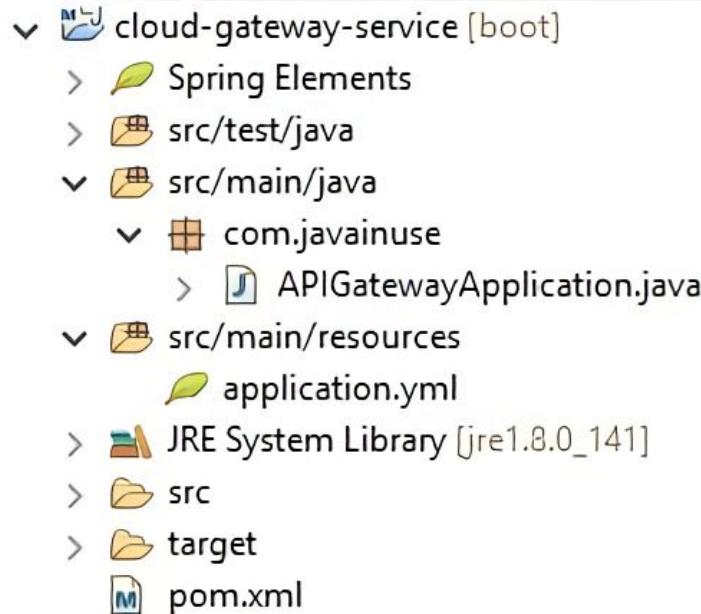
Selesai deh untuk mengimplementasikan dua service pada  
microservices!

```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SecondApplication {

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



**Selanjutnya kita implementasikan Spring Gateway menggunakan konfigurasi property based**

Caranya kayak gimana? Ikuti project Maven seperti gambar di samping

Ikuti pom.xml seperti gambar di samping.

Kamu juga dapat mengakses code pada [link berikut](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.java1nuses</groupId>
  <artifactId>cloud-gateway-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>gateway-service</name>

  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
  </properties>

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

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>${spring-cloud.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <repositories>
    <repository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
    </repository>
  </repositories>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Tentukan application.yml seperti gambar di samping ya!

```
server:
  port: 8080

spring:
  cloud:
    gateway:
      routes:
        - id: employeeModule
          uri: http://localhost:8081/
          predicates:
            - Path=/employee/**
        - id: consumerModule
          uri: http://localhost:8082/
          predicates:
            - Path=/consumer/**
```

Lalu, buat Bootstrap class dengan annotation  
@SpringBootApplication.

```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

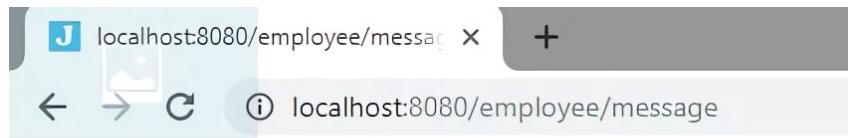
@SpringBootApplication
public class APIGatewayApplication {

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

}
```

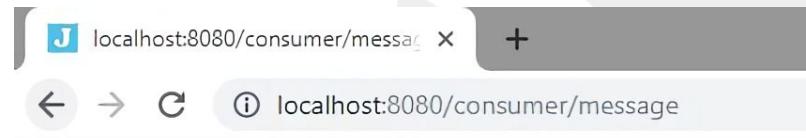
Yes! Kita bisa mulai layanan microservices yang telah kita buat. Ikuti instruksi di bawah ya!

Buka URL - localhost:8080/employee/message



Hello JavaInUse Called in First Service

Buka URL - localhost:8080/consumer/message



Hello JavaInUse Called in Second Service

```
cloud-gateway-service [boot]
  src/test/java
  src/main/java
    com.javainuse
      APIGatewayApplication
    com.javainuse.config
      SpringCloudConfig.java
  src/main/resources
    application.yml
  Referenced Libraries
  JRE System Library [jre1.8.0_181]
  src
  target
  pom.xml
```

**Yang terakhir, kita coba  
mengimplementasikan Spring Gateway  
menggunakan konfigurasi Java based**

Ikuti project Maven seperti gambar di samping.

Buat Bootstrap class dengan annotation @SpringBootApplication.

```
package com.javainuse;

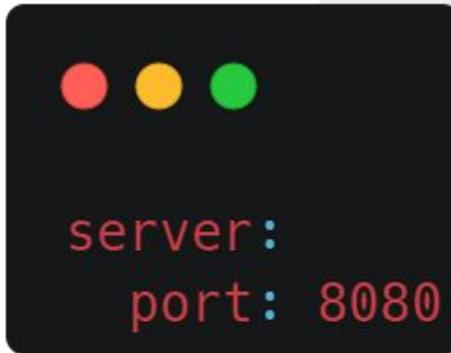
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class APIGatewayApplication {

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

}
```

Tentukan application.yml seperti gambar di samping.



Buat konfigurasi class di mana kita mendefinisikan konfigurasi route.

Sedangkan Gateway Handler menyelesaikan konfigurasi route dengan menggunakan bean RouteLocator.

```
package com.javainuse.config;

import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SpringCloudConfig {

    @Bean
    public RouteLocator gatewayRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
            .route(r -> r.path("/employee/**"))
            .uri("http://localhost:8081/")
            .id("employeeModule")

            .route(r -> r.path("/consumer/**"))
            .uri("http://localhost:8082/")
            .id("consumerModule")
        .build();
    }
}
```

Buat Bootstrap class dengan annotation @SpringBootApplication.

```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

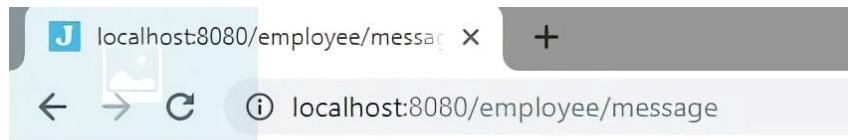
@SpringBootApplication
public class APIGatewayApplication {

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

}
```

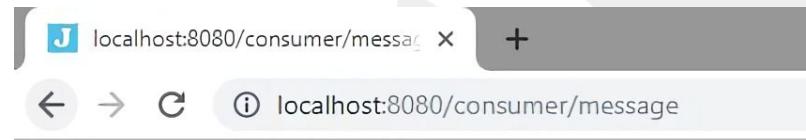
Selesai! Sama seperti sebelumnya, kita bisa mulai layanan microservices yang telah kita buat deh~

Buka URL - localhost:8080/employee/message



Hello JavaInUse Called in First Service

Buka URL - localhost:8080/consumer/message

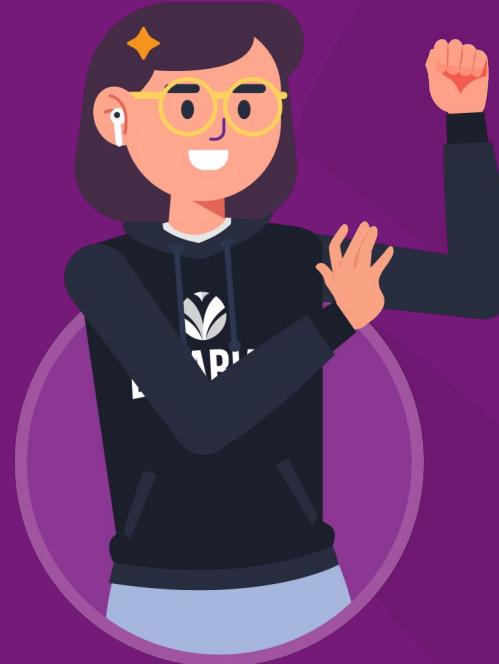


Hello JavaInUse Called in Second Service

**Gimana guys impresi kamu terhadap Spring Gateway ini?**

Kelihatan kan bahwa Spring Gateway punya banyak manfaat bagi developer khususnya yang menggunakan struktur microservices dalam mengembangkan aplikasinya □

Selain itu, menerapkan API Gateway salah satunya bisa kamu lakukan jika kamu mau memonetisasi APImu dengan menawarkan ke consumer atau organisasi.



Nah, selesai sudah pembahasan kita di Chapter 7 Topic 2 ini.

Selanjutnya, kita bakal bahas tentang **Microservices** alias arsitektur aplikasi dengan struktur yang dipecah.

Penasaran kayak gimana? Yuk, langsung ke topik selanjutnya~

