

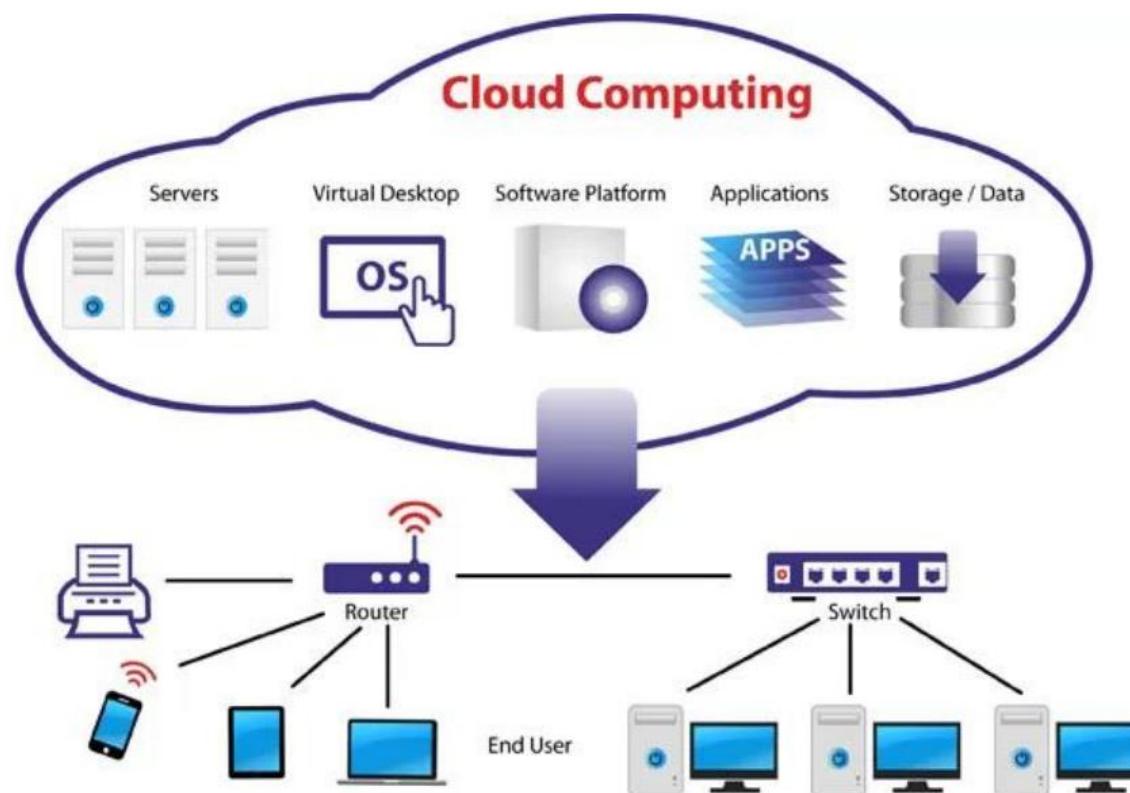


# LẬP TRÌNH JAVA SPRING BOOT

Bài 9: Spring Cloud, Microservice, API Gateway,  
Spring Cloud Config

- ❑ Kết thúc bài học này bạn có khả năng
  - ❖ Hiểu Spring Cloud
  - ❖ Hiểu mô hình lập trình Microservice (MSA) với Spring Boot
  - ❖ Biết cách triển khai dự án microservice với Spring Boot
  - ❖ Biết cách triển khai Spring Cloud Gateway
  - ❖ Biết cách triển khai Spring Cloud Config

- Điện toán đám mây (**Cloud Computing**) theo định nghĩa của **IBM** là việc cung cấp các tài nguyên máy tính cho người dùng tùy theo mục đích sử dụng thông qua kết nối Internet. Nguồn tài nguyên đó có thể là bất kì thứ gì liên quan đến điện toán và máy tính, ví dụ như phần mềm, phần cứng, hạ tầng mạng cho đến các máy chủ và mạng lưới máy chủ cỡ lớn

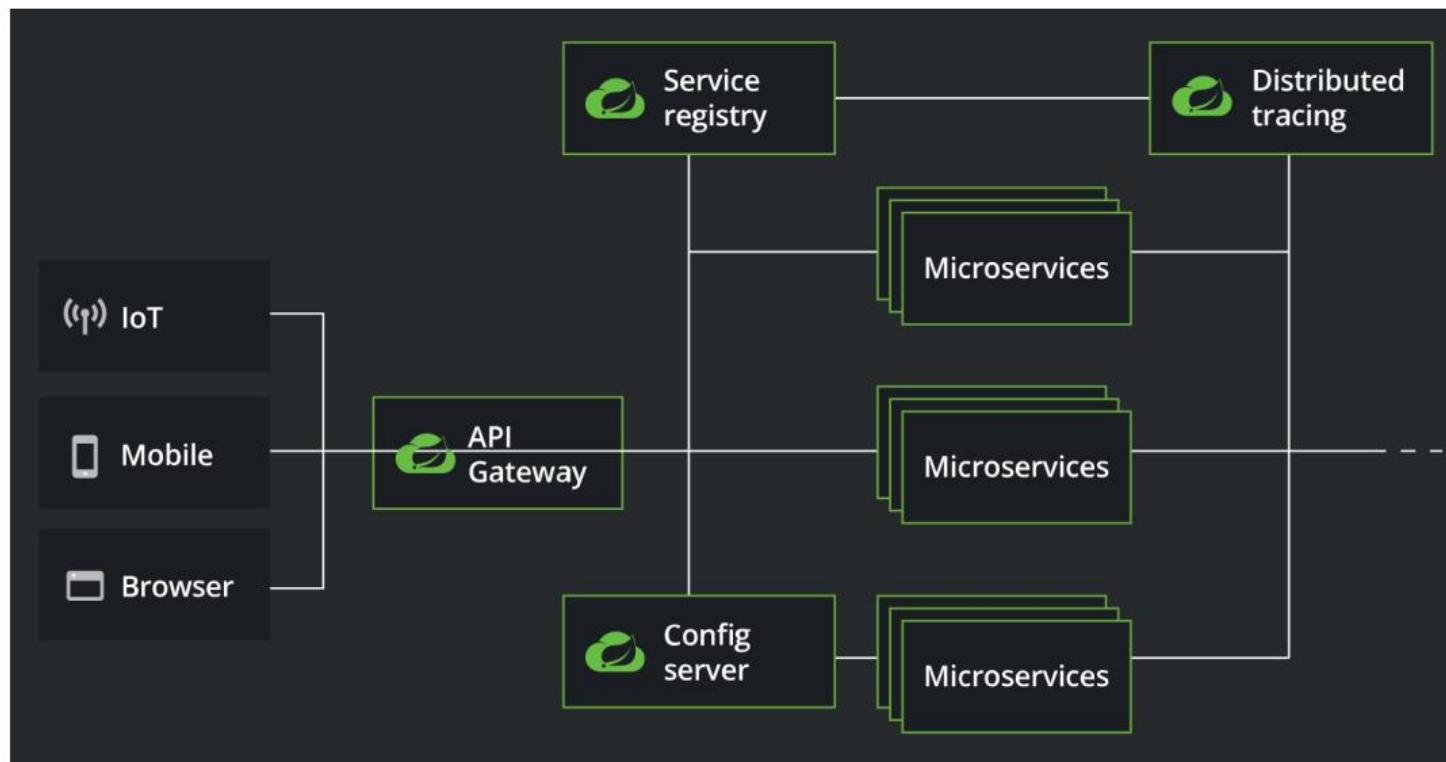


- ❑ Netflix là một công ty dịch vụ giải trí của Mỹ, cung cấp dịch vụ về các Video trực tuyến (online Video streaming) và Video theo yêu cầu (Video on demand), được thành lập năm 1997, và có trụ sở tại Los Gatos, California
- ❑ Ban đầu họ là một công ty phân phối DVD, hình thức bán hàng của họ là gửi các DVD cho khách hàng thông qua email (DVD Emailing)
- ❑ Họ cũng nhận thấy rằng trước sau gì nó cũng bị thay thế (supplant) bởi các video trên Internet (Video streaming over the internet), chính vì vậy họ quyết định đổi mới
- ❑ Đúng một năm trước thời điểm Netflix có ý định tái cơ cấu. Năm 2006, ở một nơi khác Amazon, một công ty chuyên về thương mại điện tử công bố một dự án lớn của họ và chẳng có liên quan gì tới lĩnh vực mà họ đang kinh doanh, đó chính là "điện toán đám mây" (Cloud computing)
- ❑ Dịch vụ mà sau này được đặt tên là Amazon S3 (Amazon Simple Storage Service) cho phép người dùng lưu trữ dữ liệu của họ tại các máy chủ đám mây, và cho phép bạn truy xuất bất kỳ lúc nào và ở đâu
- ❑ Sau khi chính thức sử dụng dịch vụ của Amazon, Netflix bắt đầu trở thành người tiên phong trong công nghệ phần mềm cho điện toán đám mây, làm những thứ mà chưa có một người nào làm trước đó
- ❑ Hầu như tất cả mọi thứ của họ được chạy trên Amazon Web Service (AWS), mà thực chất là hạ tầng máy chủ

- ❑ Thời điểm đó AWS không thực sự chú tâm đến các thành phần phần mềm
- ❑ Không ai làm, vì vậy họ phải tự sáng tạo ra công nghệ phần mềm cần thiết để ứng dụng của họ làm việc tốt trên môi trường đám mây
- ❑ **Thật thú vị**, các công nghệ phần mềm trên nền tảng đám mây không tới từ các **công ty IT truyền thống** mà bạn có thể đang nghĩ tới như **Oracle, Microsoft, IBM** hoặc các công ty khác, mà nó được phát minh bởi các công ty làm việc thực tế, đó là **Netflix, Amazone** và sau đó là cả **Facebook**
- ❑ Netflix đã quyết định rằng nhiều đổi mới của họ là hệ thống hữu ích và mục đích chung có thể được sử dụng cho nhiều mục đích khác nhau
- ❑ Chính vì vậy họ quyết định mở nguồn công nghệ này cho phép những người khác có thể tiếp cận với nó
- ❑ Ngay sau khi Netflix mở nguồn công nghệ của họ, nó ngay lập tức được chú ý bởi nhóm phát triển của Spring
- ❑ Họ ngay lập tức nghiên cứu Netflix OSS để đánh giá nhận thấy rằng nó rất tốt và không cần thiết phải cải tiến gì cả
- ❑ Tuy nhiên Netflix OSS cũng là một sản phẩm được phát triển cho các mục đích của chính Netflix. Vì vậy nhóm phát triển Spring tập trung vào việc làm cho các lập trình viên dễ dàng sử dụng các thư viện của Netflix hơn. Trong nhiều trường hợp, tất cả các điều cần thiết đó là thêm vào các sự phụ thuộc (Dependency) và các Annotation

- ❑ **Microservices** là một cách tiếp cận hiện đại đối với phần mềm, theo đó mã ứng dụng được phân phối thành các phần nhỏ, có thể quản lý được, độc lập với các phần mềm khác
- ❑ Tại sao sử dụng **Microservices** ?
  - ❖ Quy mô nhỏ và sự cô lập tương đối của chúng có thể dẫn đến nhiều lợi ích bổ sung, chẳng hạn như bảo trì dễ dàng hơn, cải thiện năng suất, khả năng chịu lỗi cao hơn, liên kết kinh doanh tốt hơn và hơn thế nữa
- ❑ Microservice và Spring Cloud
  - ❖ Bản chất phân tán của microservices mang đến nhiều thách thức. Spring giúp giảm thiểu những điều này. Với một số mẫu cloud ready-to-run, Spring Cloud có thể giúp khám phá dịch vụ, cân bằng tải, ngắt mạch, theo dõi phân tán và giám sát. Nó thậm chí có thể hoạt động như một cổng API

- Spring Cloud là một công nghệ phần mềm sử dụng để phát triển các ứng dụng phân tán. Một ứng dụng được gọi là phân tán (**Distributed application**) khi các phần của nó có thể được phát triển trên các ngôn ngữ khác nhau, và được triển khai trên các máy chủ khác nhau. Vì vậy mục tiêu của Spring Cloud là làm sao để các thành phần của ứng dụng có thể giao tiếp với nhau



- ❑ Bạn cần một nơi để chứa tất cả dự án microservice
- ❑ Tạo một thư mục và mở cmd tại thư mục đó và gõ dòng lệnh sau

*mvn archetype:generate -DgroupId=com.likelion -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false*

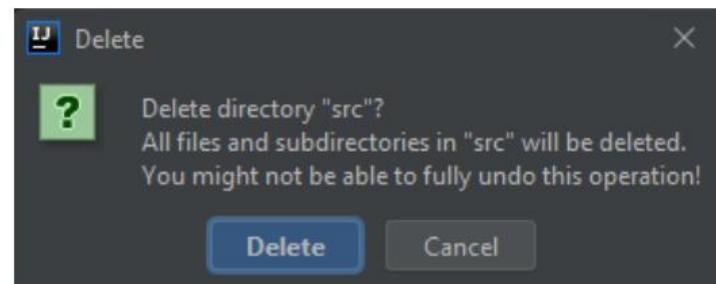
```
C:\Windows\System32\cmd.e Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

D:\springboot\projects>mvn archetype:generate -DgroupId=com.likelion
-DartifactId=microservice -DarchetypeArtifactId=maven-archetype-quick
start -DarchetypeVersion=1.4 -DinteractiveMode=false

[INFO] Parameter: groupId, Value: com.likelion
[INFO] Parameter: artifactId, Value: microservice
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.likelion
[INFO] Parameter: packageInPathFormat, Value: com/likelion
[INFO] Parameter: package, Value: com.likelion
[INFO] Parameter: groupId, Value: com.likelion
[INFO] Parameter: artifactId, Value: microservice
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: D:\springboot\projects\microservice
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:43 min
[INFO] Finished at: 2022-07-20T16:24:54+07:00
```

Bạn có thể thay đổi tên groupId và artifactid

- ☐ Mở thư mục vừa tạo và thực hiện delete thư mục src



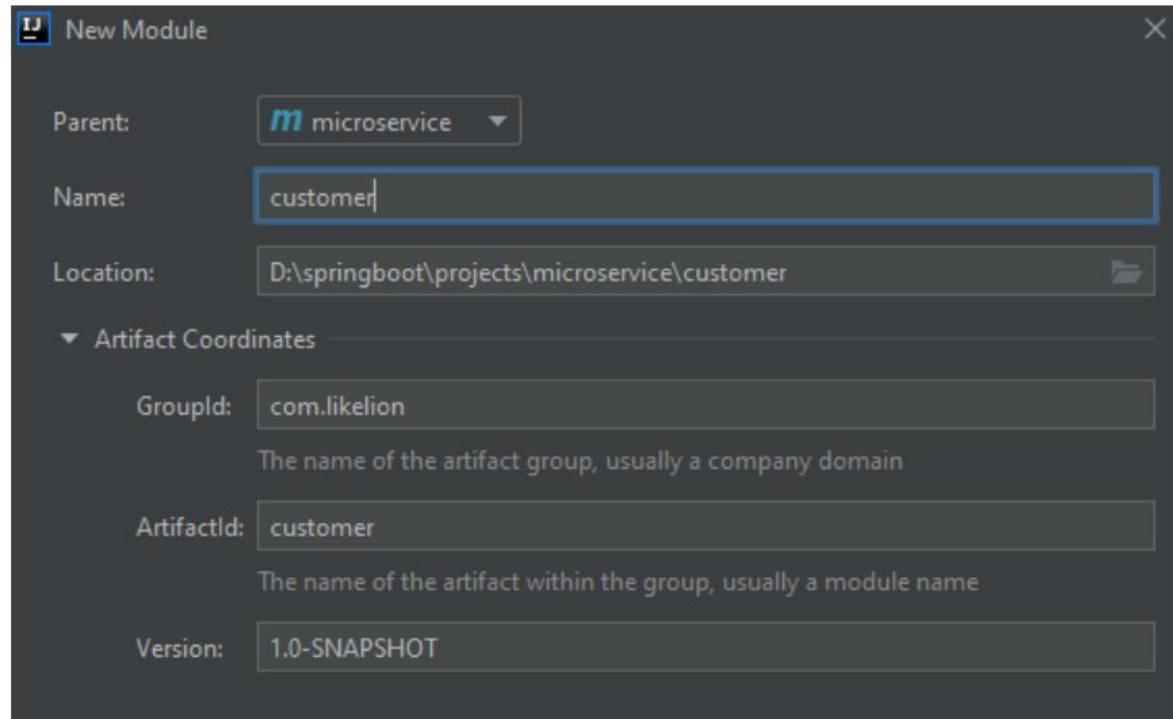
- ☐ Thêm tag <parent> vào file pom.xml gốc (microservice)

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.1</version>
</parent>
```

- ☐ Tiếp tục thêm như sau

```
<dependencies>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
</dependencies>
```

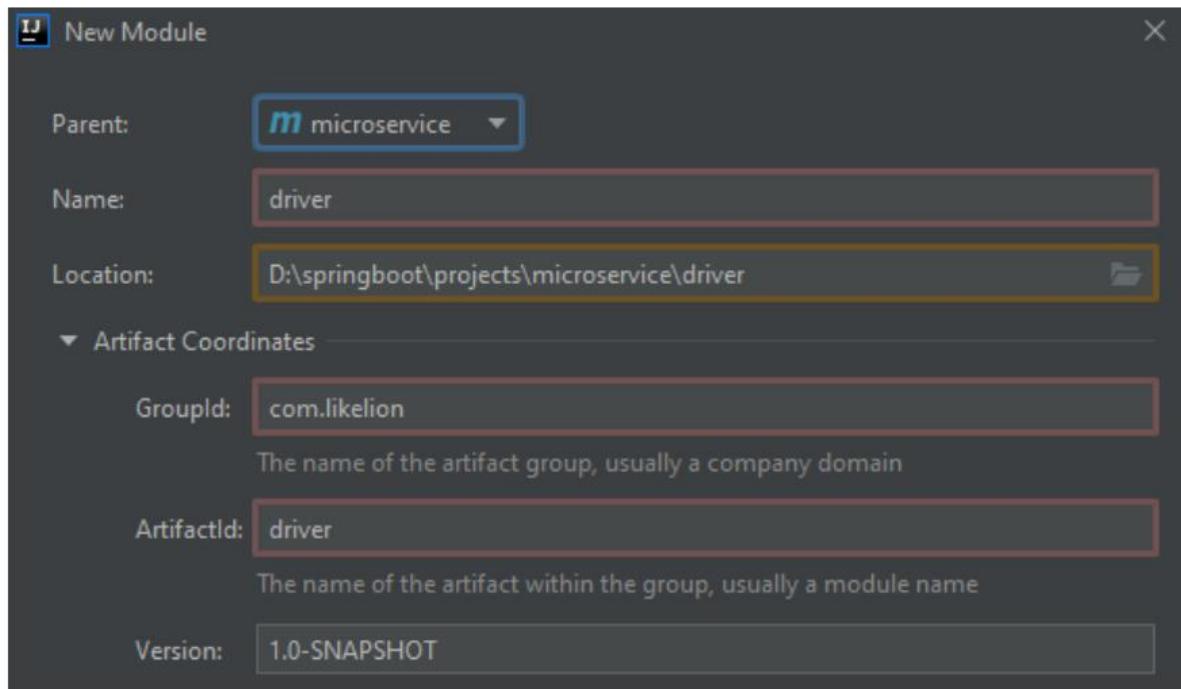
## ☐ Tiếp theo tạo module **customer** cho microservice



## ☐ Sau khi thêm xong kiểm tra file pom.xml gốc (microservice) được thêm module vào chưa

```
...
<modules>
  <module>customer</module>
</modules>
...
```

## ☐ Tiếp theo tạo module **driver** cho microservice



## ☐ Sau khi thêm xong kiểm tra file pom.xml gốc (microservice) được thêm module vào chưa

```
...
<modules>
  <module>customer</module>
  <module>driver</module>
</modules>
...
```

- ❑ Thêm dependency cho mỗi module **customer**, **driver** vào mỗi file pom.xml tương ứng với module như sau

```
<artifactId>customer</artifactId>

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

```
<artifactId>driver</artifactId>

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

- ❑ Tạo class main và controller cho mỗi module **customer**
- ❑ Từ thư mục java của module tạo package **com.likelion.customer**
- ❑ Từ package vừa tạo, tạo class **CustomerApplication**

The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is displayed with a tree view. The root project is 'microservice' located at 'D:\springboot\projects\microservice'. It contains '.idea', 'customer', and 'resources' directories. 'customer' has a 'src' folder which contains 'main' and 'java' folders. 'java' contains a 'com.likelion.customer' package with a 'CustomerApplication' class. This 'CustomerApplication' class is highlighted with a red rectangle. On the right, the code editor shows the 'CustomerApplication.java' file. The code is as follows:

```
1 package com.likelion.customer;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class CustomerApplication {
8     public static void main(String[] args) {
9         SpringApplication.run(CustomerApplication.class, args);
10    }
11 }
```

- ❑ Thực hiện tương tự với module **driver**

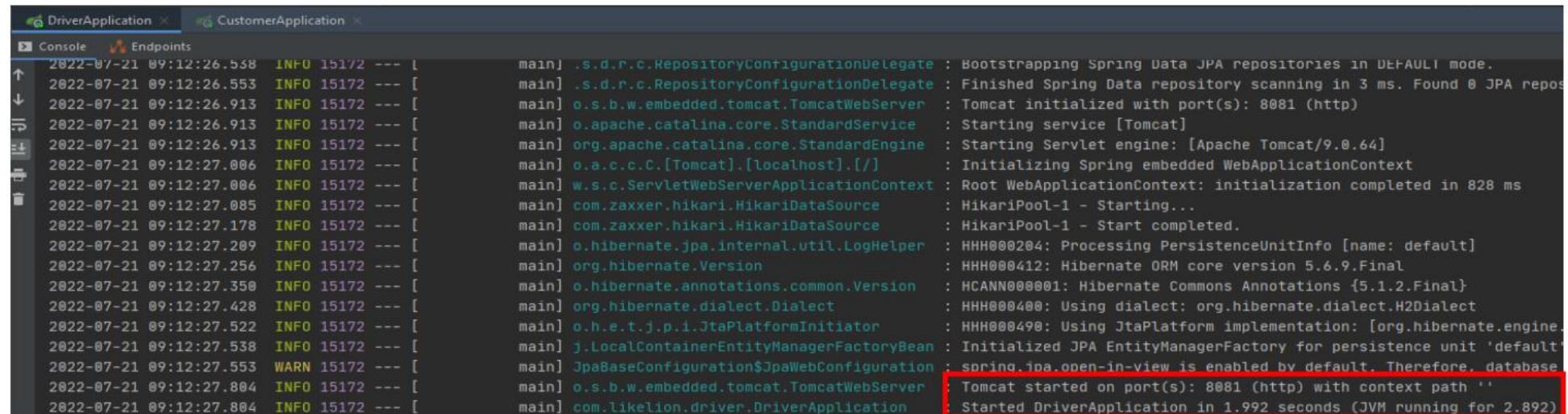
- ❑ Tạo file application.yml tại thư mục src/main/resources cho module **customer** nội dung file như sau

```
server:  
  port: 8080  
  
spring:  
  application:  
    name: customer
```

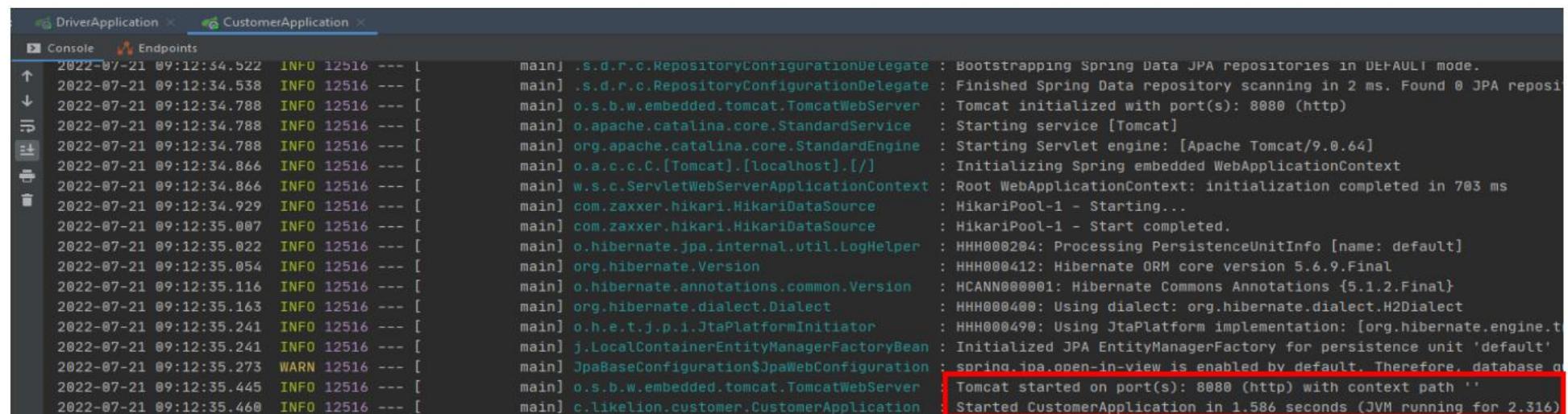
- ❑ Tạo file application.yml tại thư mục src/main/resources cho module **driver** nội dung file như sau

```
server:  
  port: 8081  
  
spring:  
  application:  
    name: driver
```

## ☐ Thực hiện khởi chạy hai module



```
DriverApplication x CustomerApplication x
Console Endpoints
2022-07-21 09:12:26.538 INFO 15172 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-07-21 09:12:26.553 INFO 15172 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 3 ms. Found 0 JPA repos.
2022-07-21 09:12:26.913 INFO 15172 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8081 (http)
2022-07-21 09:12:26.913 INFO 15172 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-07-21 09:12:26.913 INFO 15172 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-07-21 09:12:27.006 INFO 15172 --- [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2022-07-21 09:12:27.006 INFO 15172 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 828 ms
2022-07-21 09:12:27.085 INFO 15172 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-07-21 09:12:27.178 INFO 15172 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-07-21 09:12:27.209 INFO 15172 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-07-21 09:12:27.256 INFO 15172 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.9.Final
2022-07-21 09:12:27.350 INFO 15172 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-07-21 09:12:27.428 INFO 15172 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-07-21 09:12:27.522 INFO 15172 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformImpl]
2022-07-21 09:12:27.538 INFO 15172 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-07-21 09:12:27.553 WARN 15172 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries will be executed via the current transaction.
2022-07-21 09:12:27.804 INFO 15172 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http) with context path ''
2022-07-21 09:12:27.804 INFO 15172 --- [main] com.likelion.driver.DriverApplication : Started DriverApplication in 1.992 seconds (JVM running for 2.892)
```



```
DriverApplication x CustomerApplication x
Console Endpoints
2022-07-21 09:12:34.522 INFO 12516 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-07-21 09:12:34.538 INFO 12516 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 2 ms. Found 0 JPA repos.
2022-07-21 09:12:34.788 INFO 12516 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-07-21 09:12:34.788 INFO 12516 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-07-21 09:12:34.788 INFO 12516 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-07-21 09:12:34.866 INFO 12516 --- [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2022-07-21 09:12:34.866 INFO 12516 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 703 ms
2022-07-21 09:12:34.929 INFO 12516 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-07-21 09:12:35.007 INFO 12516 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-07-21 09:12:35.022 INFO 12516 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-07-21 09:12:35.054 INFO 12516 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.9.Final
2022-07-21 09:12:35.116 INFO 12516 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-07-21 09:12:35.163 INFO 12516 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-07-21 09:12:35.241 INFO 12516 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformImpl]
2022-07-21 09:12:35.241 INFO 12516 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-07-21 09:12:35.273 WARN 12516 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries will be executed via the current transaction.
2022-07-21 09:12:35.445 INFO 12516 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-07-21 09:12:35.460 INFO 12516 --- [main] c.likelion.customer.CustomerApplication : Started CustomerApplication in 1.586 seconds (JVM running for 2.316)
```

- ❑ Sử dụng object RestTemplate để giao tiếp giữa các service
- ❑ Đầu tiên chuẩn bị các api giữa các service để có thể giao tiếp với nhau
- ❑ Tại thư mục chính (com.likelion.driver). Tạo class **DriverController**

```
@RestController  
@RequestMapping("api/v1/driver")  
public class DriverController {  
  
    @GetMapping("taxi")  
    public String getTaxi() {  
        return "You found a taxi. the taxi is comming!";  
    }  
}
```

- ❑ Tạo controller và dùng **RestTemplate** giao tiếp cho module **customer**
- ❑ Tại thư mục chính (com.likelion.customer). Tạo class **CustomerConfig**

```
@Configuration  
public class CustomerConfig {  
  
    @Bean  
    public RestTemplate restTemplate() {  
        return new RestTemplate();  
    }  
}
```

- ❑ Tại thư mục chính (com.likelion.customer). Tạo class **CustomerController**

```
@RestController  
@RequestMapping("api/v1/customer")  
public class CustomerController {  
  
    @Autowired  
    RestTemplate restTemplate;  
  
    @GetMapping("/findTaxi")  
    public String findTaxi() {  
        String result = restTemplate.getForObject("http://localhost:8081/api/v1/driver/taxi", String.class);  
        return result;  
    }  
}
```

## ❑ Diễn giải

- ❖ Vì xử lý logic tìm chiếc taxi gần khách hàng nằm bên module **driver**
- ❖ Nên cần gọi đến module **driver** để nhận kết quả và trả về cho người dùng
- ❖ Điều đó thể hiện ý nghĩa việc sử dụng mô hình microservice tách nhỏ từng phần service thành từng module chuyên biệt

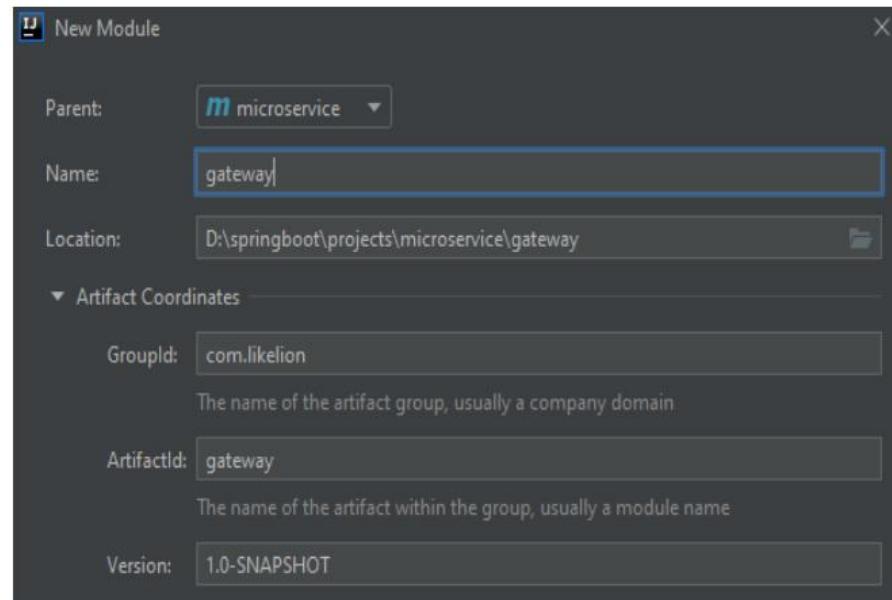
## ❑ Mở Postman và thực hiện test api như sau

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** `http://localhost:8080/api/v1/customer/findTaxi`
- Headers:** (6)
- Body:** Body dropdown is set to "Pretty".
- Response Status:** 200 OK, 109 ms, 202 B
- Response Body:** You found a taxi. the taxi is comming!

- ❑ **Spring Cloud Gateway** hỗ trợ chúng ta các filter, giúp chúng ta có thể thay đổi một request bất kỳ tới các service bên trong như thêm request parameter, thay đổi request URL, thêm header, ...
- ❑ **Spring Cloud Gateway** trong microservice đơn giản bạn không cần quan tâm tới domain của các service, bạn chỉ cần gọi đến api của gateway sau đó nó sẽ tự động định tuyến đến các service phù hợp trong dự án microservice
- ❑ **Spring Cloud Gateway** thuộc hệ sinh thái Spring framework, cũng giống như một số các API Gateway như Zuul Proxy của Spring Cloud Netflix...

## ☐ Tạo module **gateway** cho microservice



## ☐ Thêm các dependency sau module **gateway**

```
<artifactId>gateway</artifactId>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>io.projectreactor</groupId>
        <artifactId>reactor-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

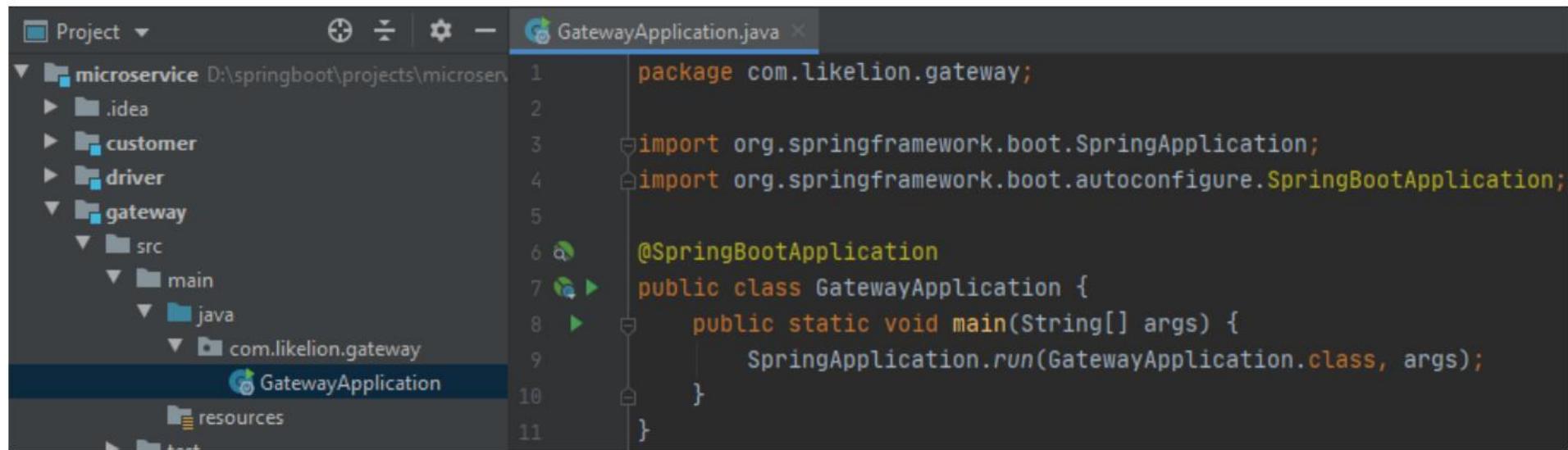
## ☐ **Lưu ý:** Cập nhật thêm các dependency cho file pom.xml gốc (microservice)

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <spring.boot.maven.plugin.version>2.7.1</spring.boot.maven.plugin.version>
    <spring.boot.dependencies.version>2.7.1</spring.boot.dependencies.version>
    <spring-cloud.version>2021.0.3</spring-cloud.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>{spring.boot.dependencies.version}</version>
            <scope>runtime</scope>
            <type>pom</type>
        </dependency>

        <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>
```

## ☐ Tạo class main cho module gateway



```
package com.likelion.gateway;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class GatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(GatewayApplication.class, args);
    }
}
```

## ☐ Tạo file application.yml tại thư mục src/main/resources cho module gateway nội dung file như sau

```
server:
port: 9088

spring:
cloud:
gateway:
routes:
- id: customerService
  uri: http://localhost:8080/
  predicates:
  - Path=/api/v1/customer/**
```

## ☐ Khởi chạy ba module và mở Postman test như sau

The screenshot shows a Postman collection named "Microservice / Find Taxi By Gateway". A GET request is selected with the URL `http://localhost:9088/api/v1/customer/findTaxi`. The "Params" tab is active. The response body shows the message "You found a taxi. the taxi is comming!".

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

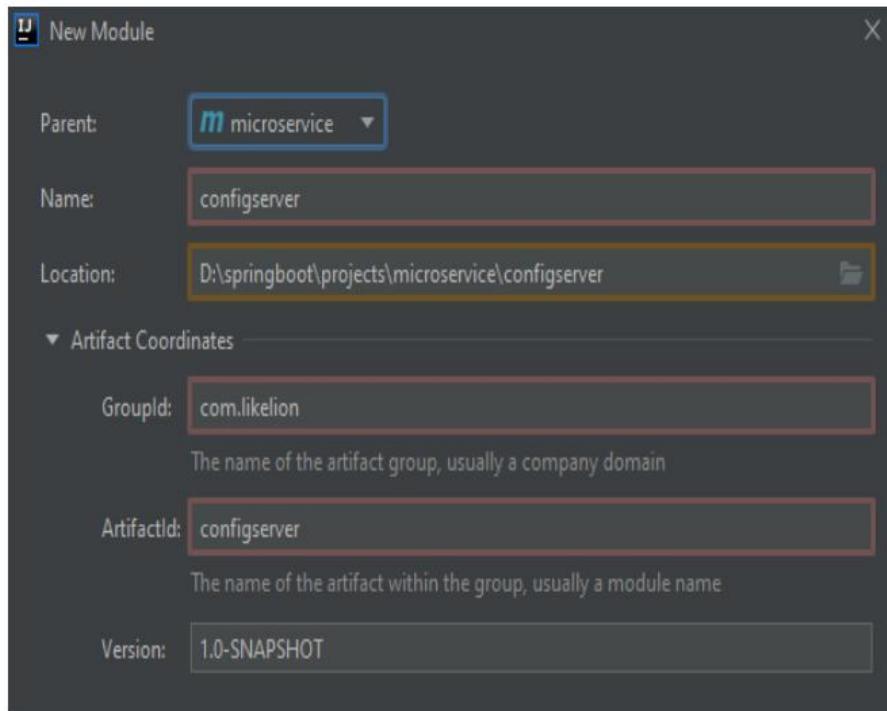
Body 200 OK 270 ms 154 B Save Response

Pretty Raw Preview Visualize Text 1 You found a taxi. the taxi is comming!

- ❑ Trong các hệ thống **phân tán**, Spring Cloud Config được dùng cho mục đích **externalized configuration** và **centralized configuration** - tách biệt và tập trung các property của các ứng dụng Spring tại một nơi. Đây là các property có giá trị khác nhau trên các môi trường phát triển (dev, staging, product, ...)
- ❑ **Spring Cloud Config** hoạt động theo mô hình kiến trúc client - server, bao gồm: Spring Cloud **Config Server** và Spring Cloud **Config Client**
- ❑ **Spring Cloud Config Server**: Các property của các ứng dụng Spring (được lưu trong các property source như file properties hoặc file YAML) được tập trung trên một hệ thống backend: **Git repository** (mặc định), File System, Vault, JDBC, Redis, ...
- ❑ Nhiệm vụ của Spring Config Server là pull các property này về và dùng **EnvironmentRepository** để lưu trữ. **EnvironmentRepository** cung cấp các đối tượng Spring Environment
- ❑ Sau đó cung cấp các property cho Config Client thông qua các HTTP resource-based API (HTTP Method là GET)

- ❑ **Spring Cloud Config Client:** Đây chính là các ứng dụng Spring đã tách biệt các property. Khi khởi chạy, **Config Client** sẽ đọc các property từ API của **Config Server** và khởi tạo đối tượng **Environment** với property source phù hợp
- ❑ Các API này có các path sau
  - ❖ /{application}/{profile}[/{label}]
  - ❖ /{application}-{profile}.yml
  - ❖ /{label}/{application}-{profile}.yml
  - ❖ /{application}-{profile}.properties
  - ❖ /{label}/{application}-{profile}.properties
- ❑ **Diễn giải**
  - ❖ {application}: map với giá trị **spring.application.name** trong file **application.yml** (hoặc **bootstrap.yml**) của Config Client
  - ❖ {profile}: map với giá trị **spring.profiles.active** trong file **application.yml** của Config Client. Giá trị mặc định là default.
  - ❖ {label} (tùy chọn): Git label dùng để gán nhãn version. Giá trị mặc định là master

## ☐ Tạo module configserver cho microservice



## ☐ Thêm các dependency sau module configserver

```
<artifactId>configserver</artifactId>

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

## ☐ Tạo class main cho module configserver

The screenshot shows the IntelliJ IDEA interface. On the left, the project structure is displayed for a 'microservice' project containing a 'configserver' module. The 'src' folder under 'configserver' contains a 'main' folder with a 'java' subfolder containing the file 'ConfigServerApplication.java'. This file contains the following Java code:

```
package com.likelion.configserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

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

## ☐ Tạo file application.yml tại thư mục src/main/resources cho module configserver nội dung file như sau

```
server:
port: 8888

spring:
cloud:
config:
server:
git:
# thay bằng đường dẫn tới git của bạn
uri: https://github.com/your-git-account/your-config-repository.git
```

## ☐ Khởi chạy configserver và truy cập vào file đã tạo sẵn trên git

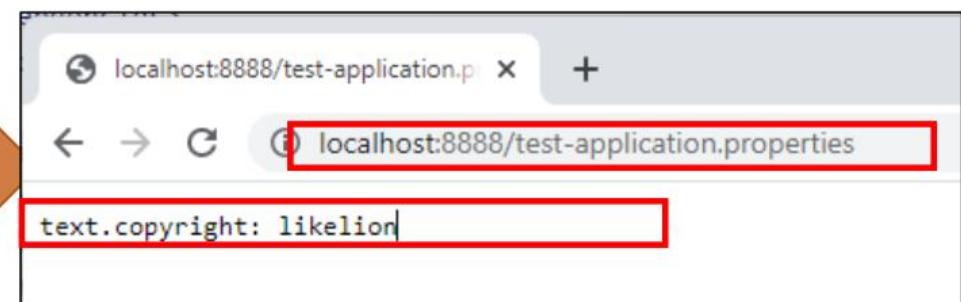
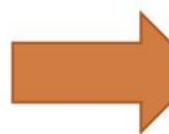


A screenshot of a GitHub repository page for 'spring-cloud-config-server'. The repository has one commit, '5da628c' made 1 hour ago. The file 'test-application.properties' is displayed, containing the line 'text.copyright=likelion'. The URL of the file is highlighted with a red box.

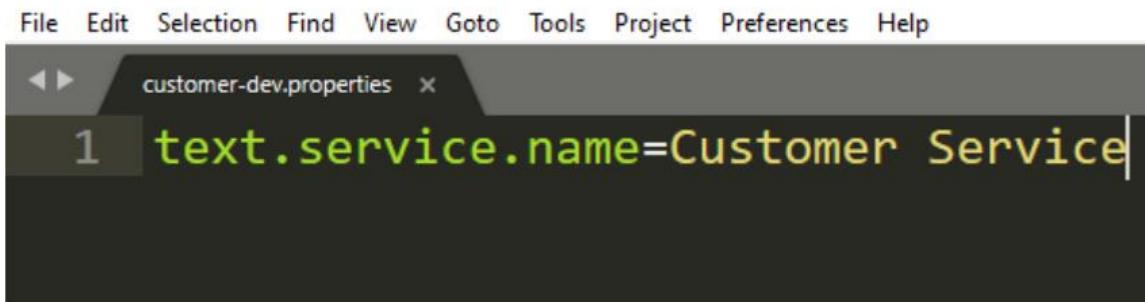
```
text.copyright=likelion
```

## ☐ Truy cập vào

<http://localhost:8888/test-application.properties>

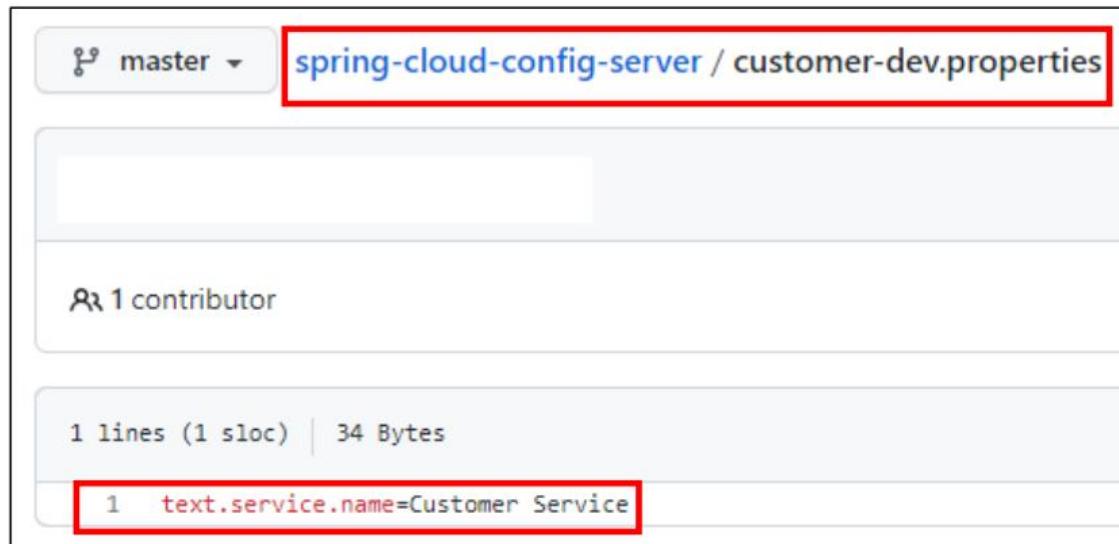


## ☐ Tạo file **customer-dev.properties** có nội dung sau



```
File Edit Selection Find View Goto Tools Project Preferences Help  
customer-dev.properties x  
1 text.service.name=Customer Service|
```

## ☐ Đưa file vừa tạo lên git config server



master spring-cloud-config-server / customer-dev.properties

1 contributor

1 lines (1 sloc) | 34 Bytes

```
1 text.service.name=Customer Service
```

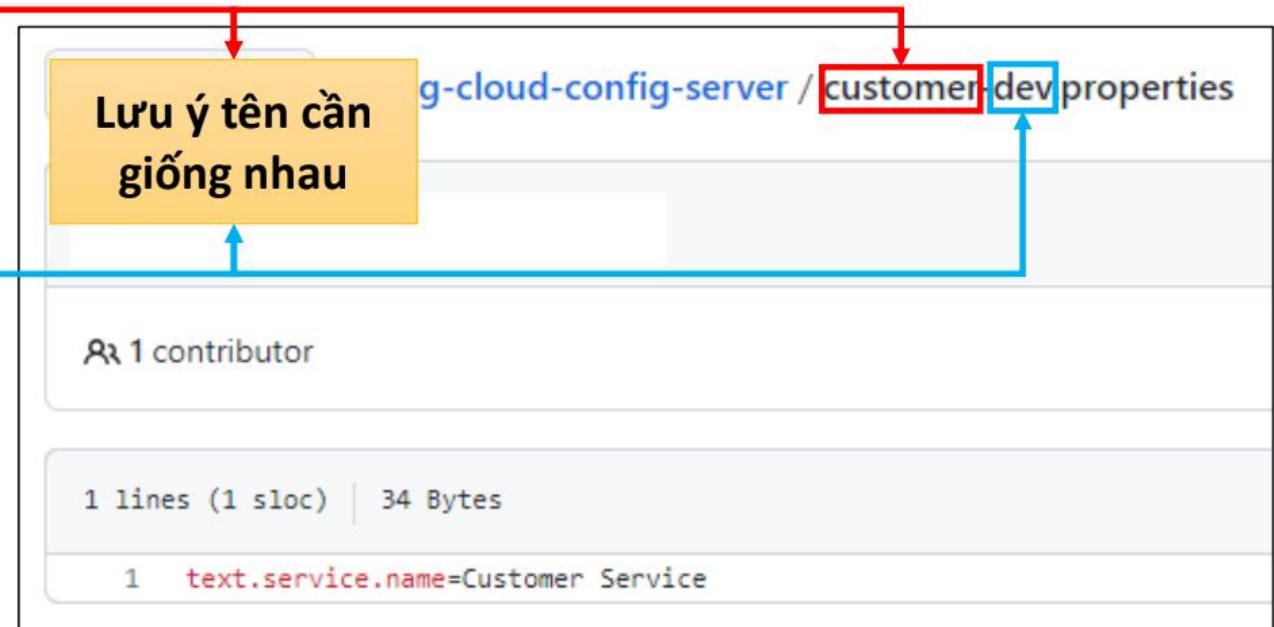
- ❑ Tại module **customer**
- ❑ Thêm dependency sau để module **customer** như là một config client

```
<artifactId>customer</artifactId>

<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-bootstrap</artifactId>
    </dependency>
</dependencies>
```

- ☐ Tại module **customer** xoá file **application.properties**, **application.yml**
- ☐ Tạo file **bootstrap.yml** có nội dung như sau

```
spring:  
  application:  
    name: customer  
  profiles:  
    active: dev  
  cloud:  
    config:  
      uri: http://localhost:8888  
      enabled: true  
      #   username: user  
      #   password: secret  
  
  server:  
    port: 8080  
    #   request-read-timeout: 10
```



- ☐ **Lưu ý:** đặt **application.name** và **profiles.active** phải giống với tên trên git repository

- ❑ Tại module **customer** tạo class **MessageRestController** như sau

```
@RefreshScope  
@RestController  
public class MessageRestController {  
    @Value("${text.service.name}")  
    private String message;  
  
    @RequestMapping("/message")  
    String getMessage() {  
        return this.message;  
    }  
}
```

- ❑ Mặc định, các property sẽ chỉ được đọc một lần duy nhất khi Config Client khởi chạy. Tuy nhiên, với các bean cần cập nhật giá trị mới nhất từ Config Server, chúng ta có thể khai báo thêm annotation **@RefreshScope**

- ❑ Trước tiên khởi chạy module **configserver**
- ❑ Tiếp theo khởi chạy module **customer**
- ❑ Truy cập vào <http://localhost:8080/message> để kiểm tra kết quả

The screenshot shows a POSTMAN interface. At the top, the URL `http://localhost:8080/message` is highlighted with a red box. Below the URL, under the 'Params' tab, there is a table:

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

At the bottom of the interface, the response body is displayed with the text "Customer Service" highlighted by a red box.

- ❑ Điện toán đám mây (**Cloud Computing**) là cung cấp các tài nguyên máy tính cho người dùng tùy theo mục đích sử dụng thông qua kết nối Internet
- ❑ Nguồn tài nguyên đó có thể là bất kì thứ gì liên quan đến điện toán và máy tính, ví dụ như **phần mềm, phần cứng, hạ tầng mạng** cho đến các **máy chủ và mạng lưới** máy chủ cỡ lớn
- ❑ Thật thú vị, các công nghệ phần mềm trên nền tảng đám mây không tới từ các **công ty IT truyền thống** mà bởi các công ty làm việc thực tế, đó là **Netflix, Amazone** và sau đó là cả **Facebook**
- ❑ **Spring Cloud** là một công nghệ phần mềm sử dụng để phát triển các ứng dụng **phân tán**, khi các phần của nó có thể được phát triển và triển khai trên các máy chủ **khác nhau**
- ❑ Mục tiêu của **Spring Cloud** là làm sao để các thành phần của ứng dụng có thể **giao tiếp với nhau**
- ❑ **Microservices** là một cách tiếp cận hiện đại đối với phần mềm, theo đó mã ứng dụng được phân phối thành các **phần nhỏ**, có thể quản lý được, độc lập với các phần mềm khác
- ❑ Nó mang lại nhiều **lợi ích** như: bảo trì dễ dàng hơn, cải thiện năng suất, khả năng chịu lỗi cao hơn, liên kết kinh doanh tốt hơn và hơn thế nữa
- ❑ Nhưng bản chất **phân tán** của **microservices** mang đến nhiều **thách thức**

- ❑ Spring giúp **giảm thiểu** những điều này. Với một số mẫu cloud ready-to-run, **Spring Cloud** có thể giúp khám phá dịch vụ, cân bằng tải, ngắt mạch, theo dõi phân tán và giám sát
- ❑ **Spring Cloud Gateway** hỗ trợ chúng ta các filter, giúp chúng ta có thể thay đổi một request bất kỳ tới các service bên trong như thêm request parameter, thay đổi request URL, thêm header, ...
- ❑ **Spring Cloud Gateway** thuộc hệ sinh thái Spring framework, cũng giống như một số các **API Gateway** như **Zuul Proxy** của **Spring Cloud Netflix**...
- ❑ **Spring Cloud Config** được dùng cho mục đích **externalized configuration** và **centralized configuration** - **tách biệt** và **tập trung** các **property** của các ứng dụng **Spring** tại một nơi. Đây là các **property** có giá trị khác nhau trên các môi trường phát triển (**dev, staging, product, ...**)
- ❑ **Spring Cloud Config** bao gồm: **Config Server** và **Config Client**
- ❑ **Config Server**: Các **property** của các ứng dụng Spring (được lưu trong các **property source** như file **properties** hoặc file **YAML**) được **tập trung** trên một hệ thống backend: **Git repository** (mặc định), **File System**, **Vault**, **JDBC**, **Redis**, ...
- ❑ **Config Client**: Đây chính là các ứng dụng **Spring** đã tách biệt các **property**. Khi khởi chạy, **Config Client** sẽ **đọc** các **property** từ **API** của **Config Server** và khởi tạo đối tượng **Environment** với **property source** phù hợp

# Cảm Ơn Bạn Đã Chăm Chỉ!

