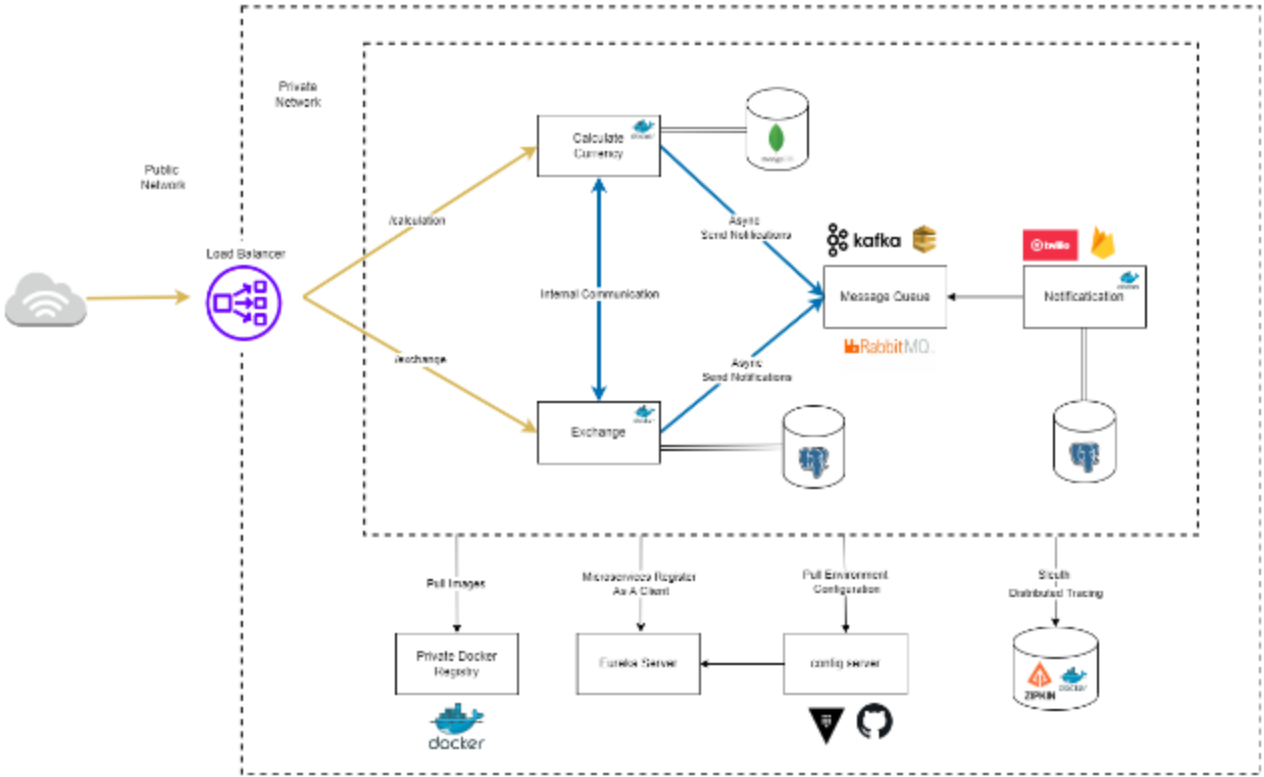
LẬP TRÌNH JAVA SPRING BOOT

Bài 10: Thực hành dự án với Microservice và Spring Cloud

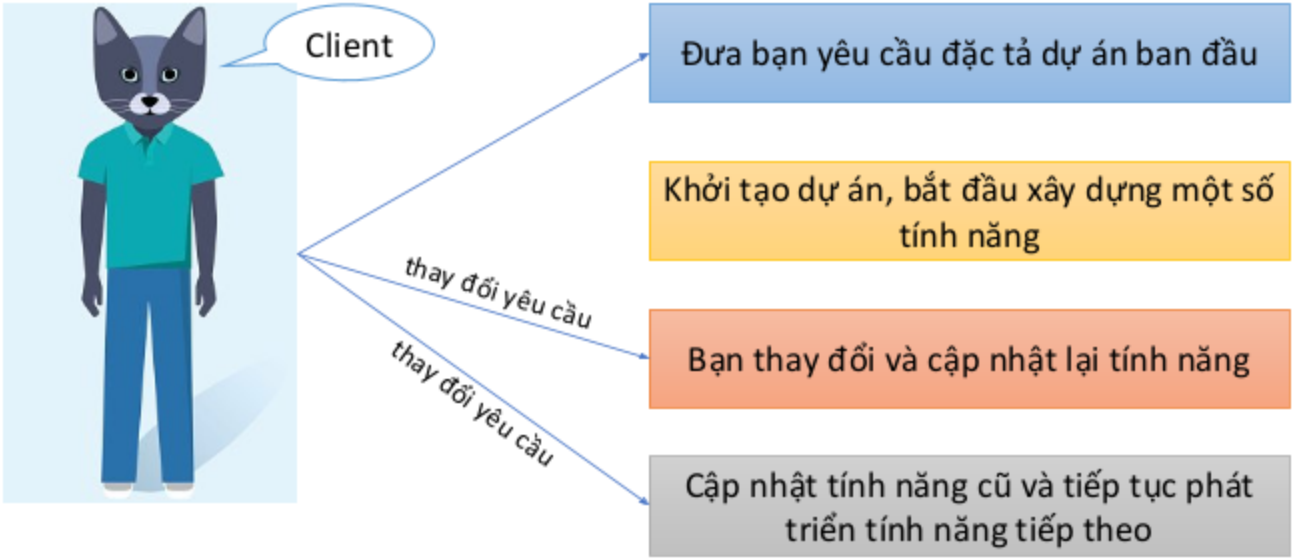
Đề bài: Xây dựng ứng dụng chuyển đổi tiền tệ

- Kết thúc bài thực hành này bạn có khả năng
  - Xây dựng dự án với kiến trúc Microservice. Mỗi service bao gồm ba tầng **controller**, **service**, **repository**
  - Ứng dụng các công nghệ Spring Cloud vào dự án như: gateway, config…
  - Kết hợp với các công nghệ khác như
    - Service Registry (Eureka Server), Load Balance Requests

- **MSA** là cách phát triển ứng dụng bằng cách chia nhỏ các nghiệp vụ thành các dự án nhỏ

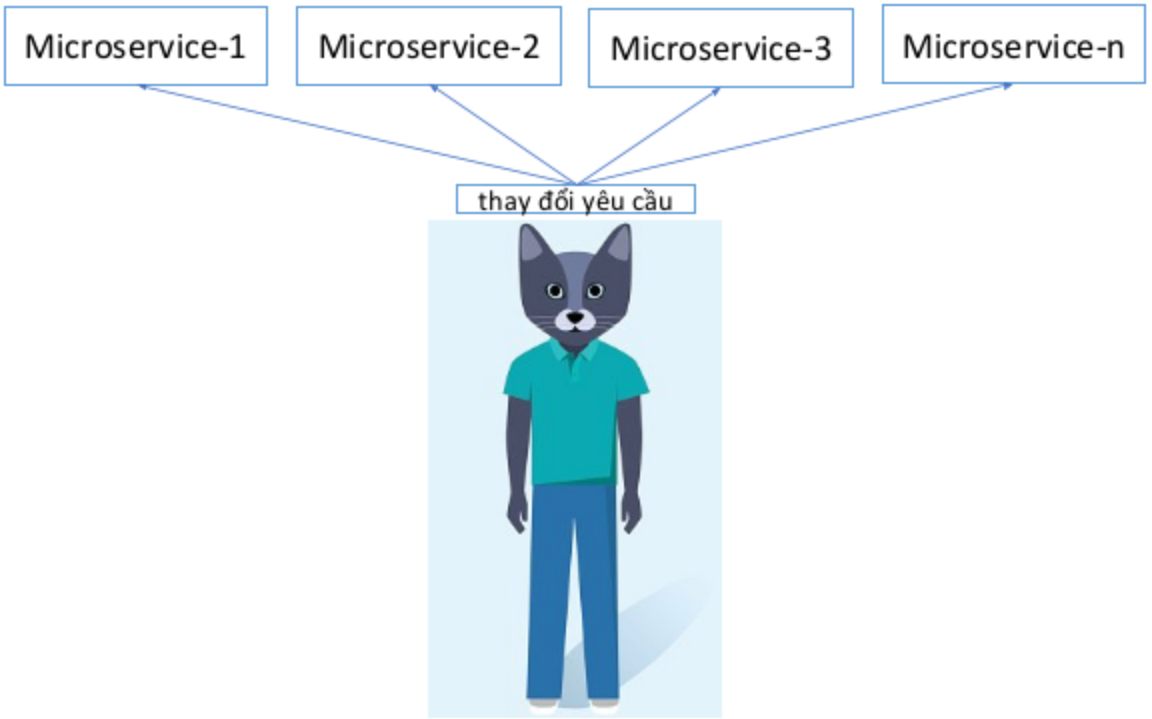


Cách Phát Triển Dự Án Truyền Thống



- Theo cách tiếp cận này dự án sẽ gặp vấn đề lớn khi về cuối tiến độ
- Tốn nhiều chi phí coding và testing
- Rất khó bảo trì dự án sau này

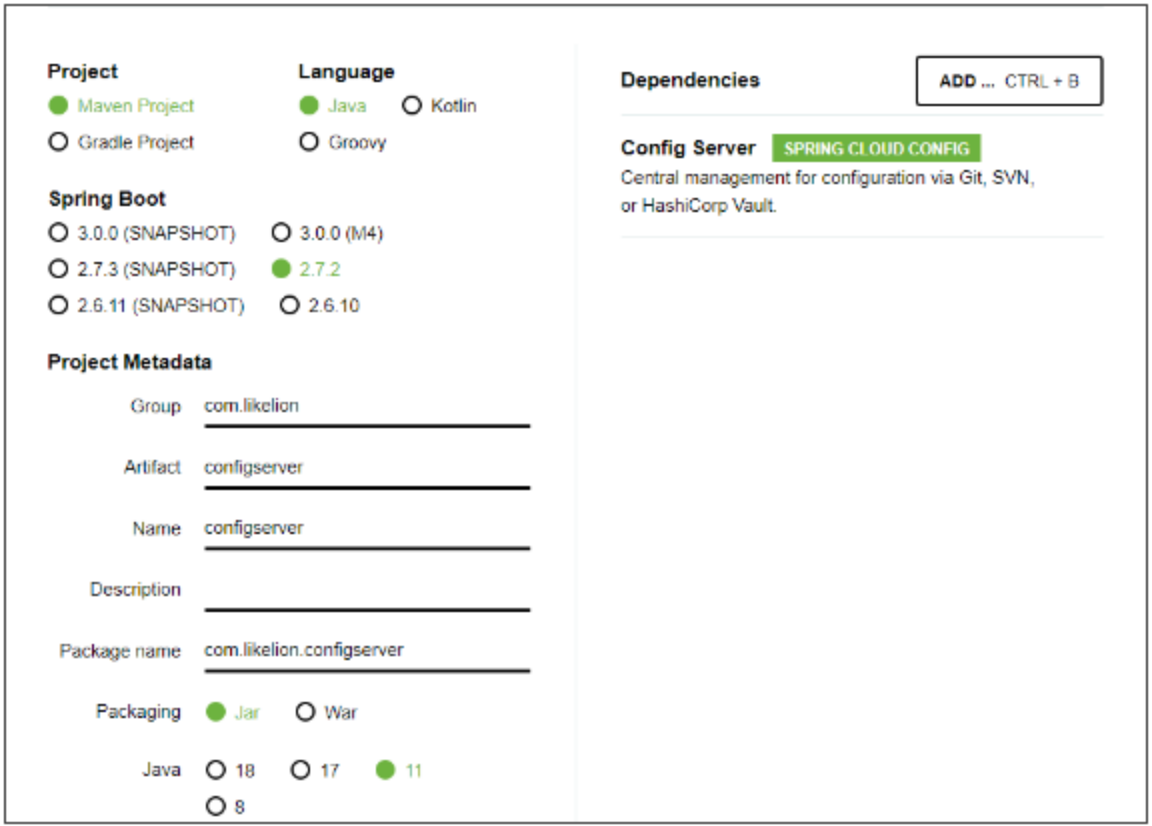
- Mỗi một thành viên trong dự án sẽ phụ trách một service



- Khi coding hoàn tất tích hợp các service lại và testing

- **Giới hạn Context**: ban đầu thiết kế chỉ có 16 service nhưng sau thời gian có thể tăng lên 32 service và có thể nhiều hơn thế
- **Cấu hình**: việc có nhiều service thì cấu hình cho mỗi service là một thách thức
- **Khả năng mở rộng và thu nhỏ dự án**: những ngày nghỉ lễ thì số lượng người truy cập vào tăng mạnh, nên cần có số lượng service đủ nhiều để đáp ứng. Ngược lại những ngày có số lượng ít người truy cập thì cần thu hẹp số lượng service đang hoạt động để giảm bớt tài nguyên bị lãng phí
- **Stack of card**: khi một trong các service có vấn đề thì tất cả các service khác có thể bị ảnh hưởng theo
- …
- **Dó đó**: Các nhà phát triển **Spring** đã phát triển **Spring Cloud** để giải quyết các các thách thức trên cho lập trình viên

- Truy cập vào https://start.spring.io/ tạo dự án như hình



- Kiểm tra file pom.xml như sau

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </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>
```

- Thêm annotation **@EnableConfigServer** vào class main module

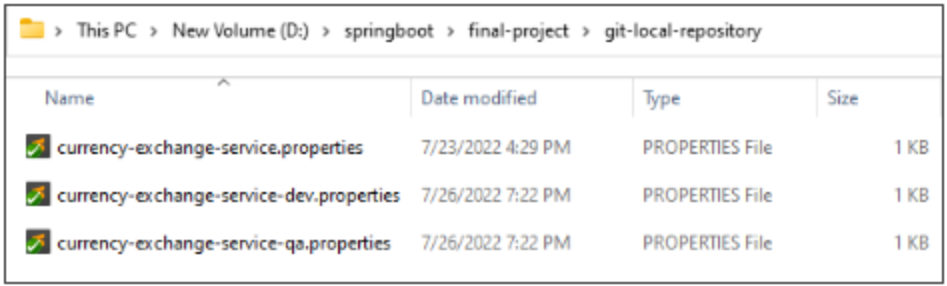
```
@SpringBootApplication
@EnableConfigServer
public class ConfigserverApplication {

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

}
```

- Thêm property ở file **application.properties** như sau

```
spring.application.name=config-server
server.port=8888
# đường dẫn đến git repository
# có thể link đến thư mục git máy local
# hoặc link đến đường dẫn của git remote
spring.cloud.config.server.git.uri=file://D:/springboot/final-project/git-local-repository
spring.cloud.config.server.git.default-label=master
```

Tạo File Properties Tại Máy Local

- Tạo 3 file properties ở local như sau, lưu ý đặt tên 3 file giống hình



- Nội dung file **currency-exchange-service.properties** (default)

```
text.currency.exchange.service=Currency Exchange Service
```

Tạo File Properties Tại Máy Local

- Nội dung file **currency-exchange-service-dev.properties** (dev)

```
# Enabling H2 Console
spring.h2.console.enabled=true

# =======================================
# DB
# =======================================

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# =======================================
# JPA / HIBERNATE
# =======================================

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization: 'true'
```

- Nội dung file **currency-exchange-service-qa.properties** (qa)

```
# Enabling H2 Console
spring.h2.console.enabled=true

# =======================================
# DB
# =======================================

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=qa
spring.datasource.password=

# =======================================
# JPA / HIBERNATE
# =======================================

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization: 'true'
```

Push File Properties Lên Git

- Tạo một repository trên git và đưa các file properties từ local lên repository trên git
- Thực hiện dùng lệnh git đưa lên repository
  - Thực hiện lệnh commit

    ```
    git add . && git commit -m "first commit"
    ```

  - Thực hiện lệnh push

    ```
    git push -u origin master
    ```

- **Lưu ý**: trước đó thư mục chứa các file properties đã phải thực hiện những lệnh init, tạo branch, tạo remote
- Kết quả sau khi đưa lên repository

Chạy Module Config Server

- Khởi chạy với cổng **8888**

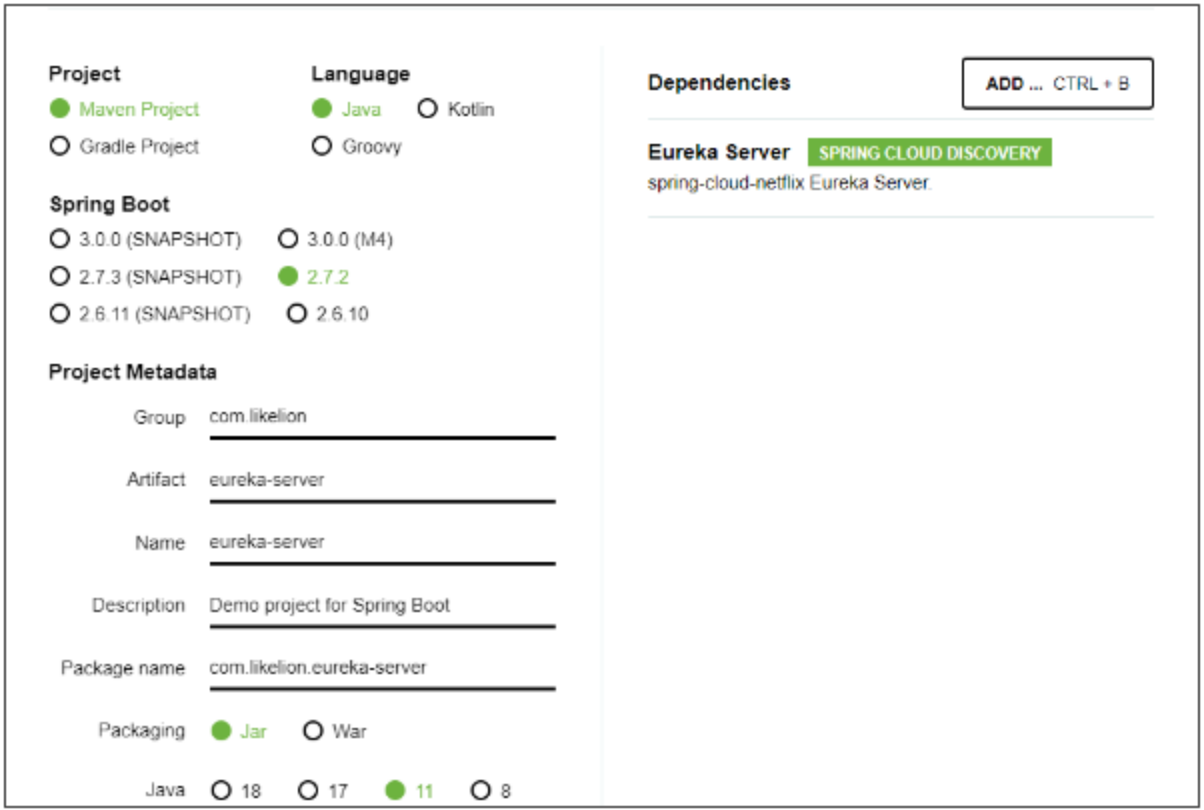
```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.2)

2022-07-28 09:46:39.881  INFO 11996 --- [           main] c.l.c.ConfigserverApplication            : Starting ConfigserverApplication using Java 11.0.15 on MSI with PID 11996
2022-07-28 09:46:39.881  INFO 11996 --- [           main] c.l.c.ConfigserverApplication            : No active profile set, falling back to 1 default profile: "default"
2022-07-28 09:46:40.428  INFO 11996 --- [           main] o.s.cloud.context.scope.GenericScope     : BeanFactory id=95380acb-b192-3266-a1d8-f999d9c3350d
2022-07-28 09:46:40.599  INFO 11996 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8888 (http)
2022-07-28 09:46:40.599  INFO 11996 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-07-28 09:46:40.599  INFO 11996 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-07-28 09:46:40.740  INFO 11996 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-07-28 09:46:40.756  INFO 11996 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 844 ms
2022-07-28 09:46:41.332  INFO 11996 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8888 (http) with context path ''
2022-07-28 09:46:41.441  INFO 11996 --- [           main] c.l.c.ConfigserverApplication            : Started ConfigserverApplication in 2.039 seconds (JVM running for 2.739)
```

- Truy cập vào http://localhost:8888/currency-exchange-service/dev


```
{
  "name": "file://D:/springboot/final-project/git-local-repository/file:D:\\springboot\\final-project\\git-local-repository\\currency-exchange-service-dev.properties",
  "source": {
    "spring.h2.console.enabled": "true",
    "spring.datasource.url": "jdbc:h2:mem:testdb",
    "spring.datasource.driverClassName": "org.h2.Driver",
    "spring.datasource.username": "sa",
    "spring.datasource.password": "",
    "spring.jpa.show-sql": "true",
    "spring.jpa.hibernate.ddl-auto": "update",
    "spring.jpa.properties.hibernate.dialect": "org.hibernate.dialect.H2Dialect",
    "spring.jpa.defer-datasource-initialization": "'true'"
  }
},
```

- **Eureka Server** là một máy chủ dùng để quản lý, đặt tên cho các service, hay còn gọi là **service registry**. Nhưng tại sao chúng ta lại cần một server để đặt tên cho mỗi service
  - Chúng ta không muốn **hardcode** địa chỉ IP của mỗi microservice
  - Khi mà các service có **nhiều** instances, nó sẽ hỗ trợ tự động map tới các instances tướng ứng thay vì chỉ định địa chị IP cho mỗi lần gọi
- **Service Registry** giữ các thực thể microservices và địa chỉ của chúng. Thực thể microservices được đăng kí với service registry khi bắt đầu chạy và hủy đăng kí khi tắt
- Các **Eureka client service** là một service độc lập trong kiến trúc microservice. Tương ứng với mỗi module trong microservice
- Các **Eureka client service** sẽ phải đăng ký với **Eureka Server** để có thể biết được sự tồn tại của nó và cung cấp ip, port để có thể giao tiếp

- Truy cập vào https://start.spring.io/ tạo dự án như hình



- Kiểm tra file pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </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>
```

- Thêm annotation **@EnableEurekaServer** vào class main module

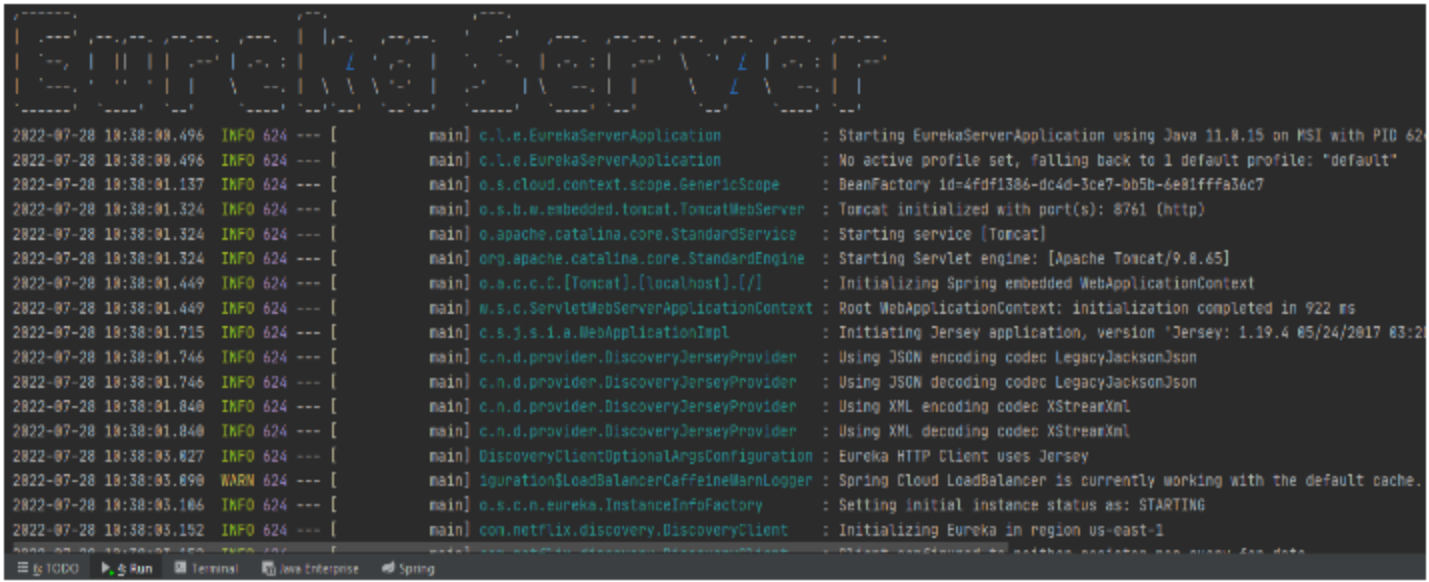
```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

}
```

- Đổi tên file **application.properties** bằng **application.yml** như sau

```
spring:
  application:
    name: eureka-server

server:
  port: 8761

eureka:
  client:
    fetch-registry: false
    register-with-eureka: false
```

Chạy Module Eureka Server

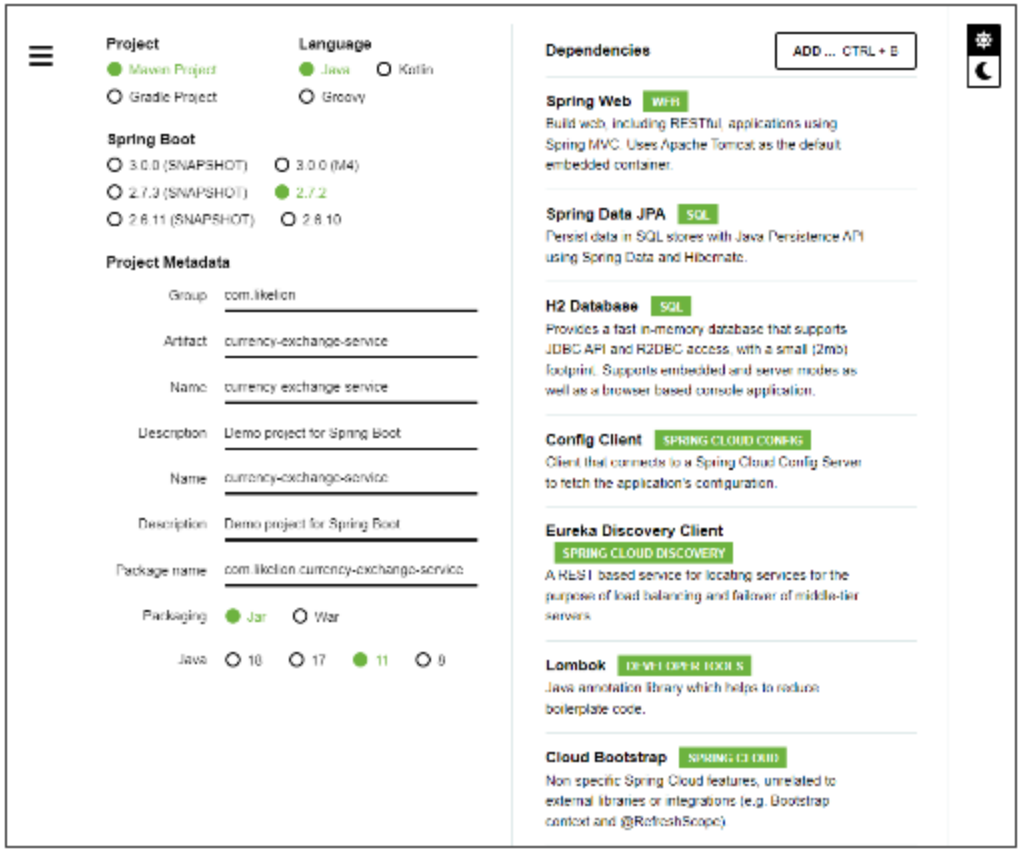
- Khởi chạy với cổng **8761**



- Truy cập vào http://localhost:8761/ để kiểm tra

Tạo Module Currency Exchange Service

- Truy cập vào https://start.spring.io/ tạo dự án như hình



- Kiểm tra file pom.xml như sau

1

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <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-eureka-client</artifactId>
  </dependency>
```

2

```
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Thêm annotation **@EnableEurekaClient** vào class main module

```
@SpringBootApplication
@EnableEurekaClient
public class CurrencyExchangeServiceApplication {

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

}
```

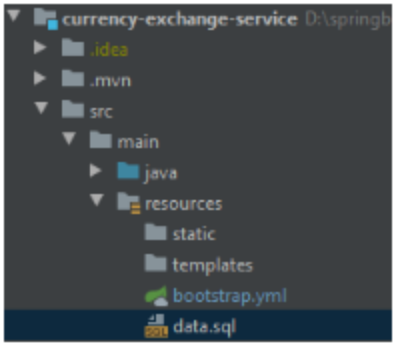
- Đổi tên file **application.properties** bằng **bootstrap.yml** như sau

```
spring:
  cloud:
    config:
      uri: http://localhost:8888 # đăng ký với config server
  application:
    name: currency-exchange-service
  profiles:
    active: dev # khi khởi chạy sẽ lấy thông tin file tương ứng là currency-exchange-service-dev.properties

server:
  port: 8000

# đăng ký với eureka server đây là một eureka client
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```

- Tại thư mục chính module tạo các package **controller, model, repository**
- Từ package **model** tạo class **ExchangeValue**

```
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class ExchangeValue {
    @Id
    private Long id;
    @Column(name="currency_from")
    private String from;
    @Column(name="currency_to")
    private String to;
    private BigDecimal conversionMultiple;
    private int port;
}
```

- Tạo file **data.sql** tại thư mục *src/main/resources* như sau

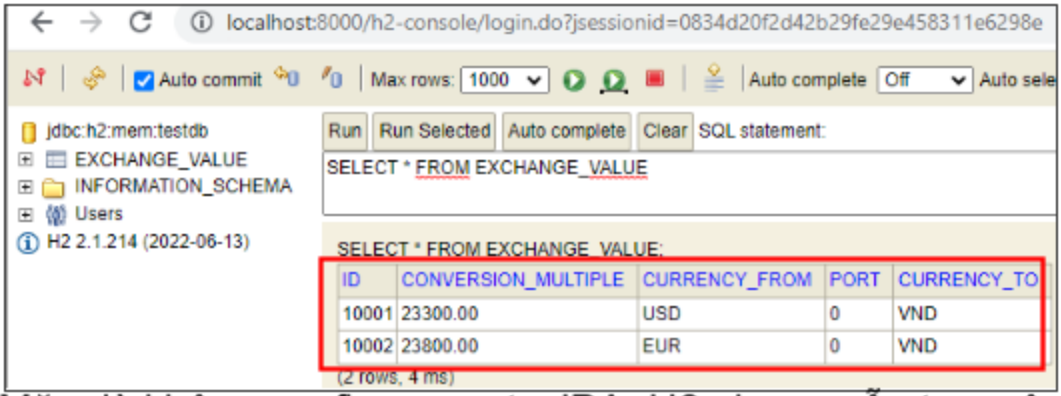



```
insert into
exchange_value(id,currency_from,currency_to,conversion_multiple,port)
values(10001,'USD','VND',23300,0);
insert into
exchange_value(id,currency_from,currency_to,conversion_multiple,port)
values(10002,'EUR','VND',23800,0);
```

- Khi khởi chạy module spring boot sẽ tự động chạy file **data.sql**
- Tạo hai record chuyển đổi tiền tệ từ **usd** và **eur** đến **vnd**

Chạy Module Currency Exchange Service

- Trước tiên chạy module **eureka server**
- Tiếp theo chạy module **config server**
- Tiếp theo chạy module **currency exchange service**
- Truy cập vào http://localhost:8000/h2-console login bằng account **sa**



- Mặc dù không config property JPA, H2 nhưng vẫn truy cập được có nghĩa là đã lấy file **currency-exchange-service-dev.properties** từ module **config server** thành công

- Tại package **repository** tạo interface **ExchangeValueRepository** như sau

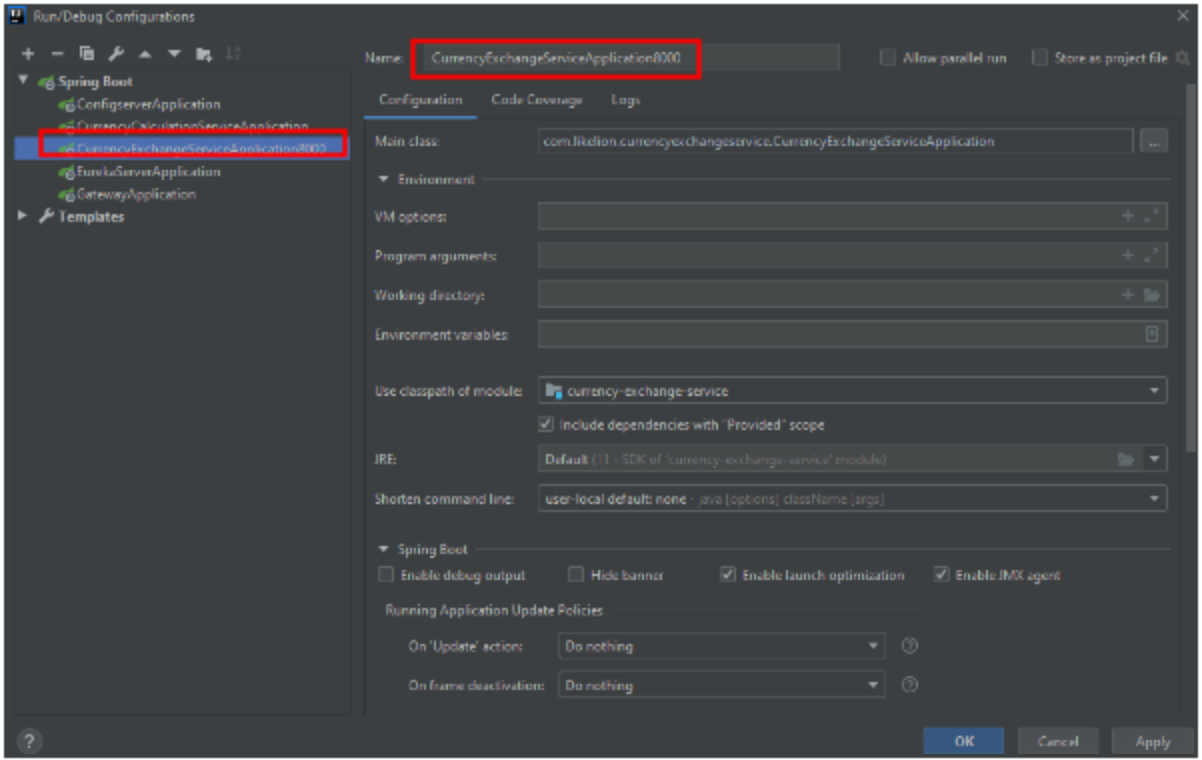
```
@Repository
public interface ExchangeValueRepository extends JpaRepository<ExchangeValue, Long> {
    ExchangeValue findByFromAndTo(String from,String to);
}
```

- Tại package **controller** tạo class **CurrencyExchangeController** như sau

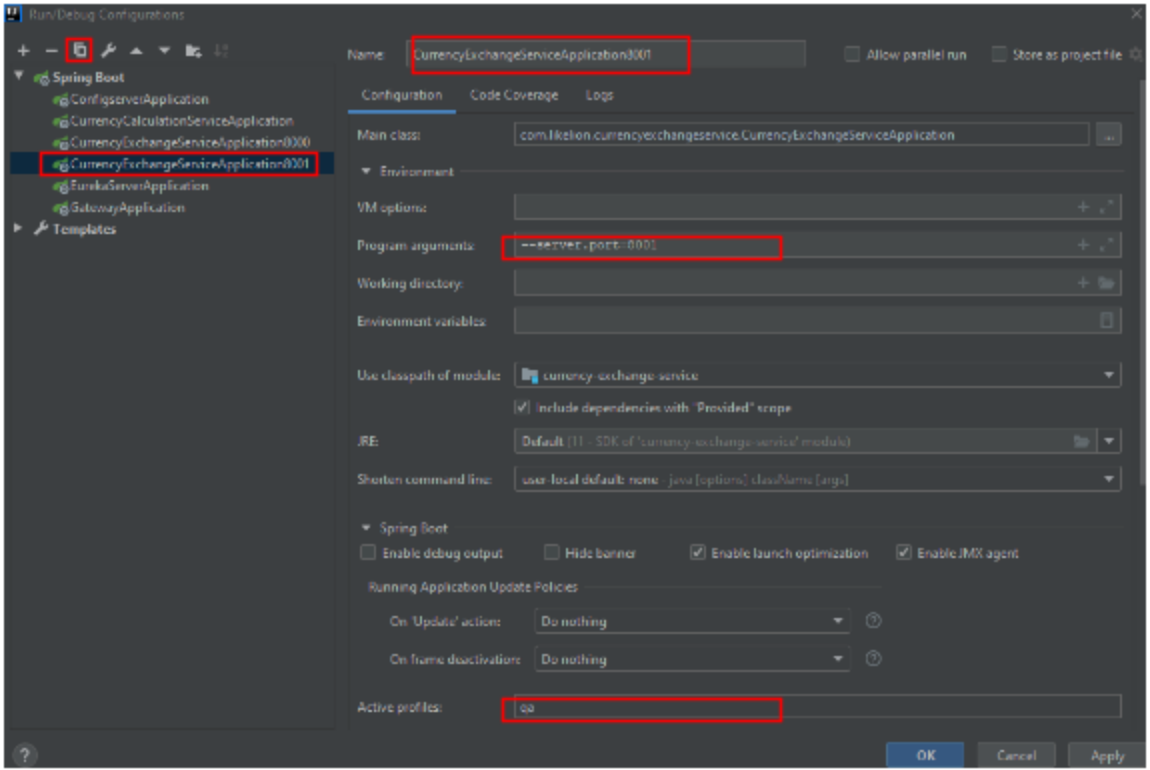
```
@RestController
public class CurrencyExchangeController {

    @Autowired
    private ExchangeValueRepository exchangeValueRepository;

    @Autowired
    private Environment environment;

    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public ExchangeValue retrieveExchangeValue(@PathVariable String from, @PathVariable String to) {
        ExchangeValue exchangeValue = exchangeValueRepository.findByFromAndTo(from, to);
        exchangeValue.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
        return exchangeValue;
    }
}
```

Tạo Instance Currency Exchange Service

- Tạo một instance mới từ module **currency exchange service** như sau
- Mở cửa sổ Run/Debug Configurations và đổi tên item như hình dưới

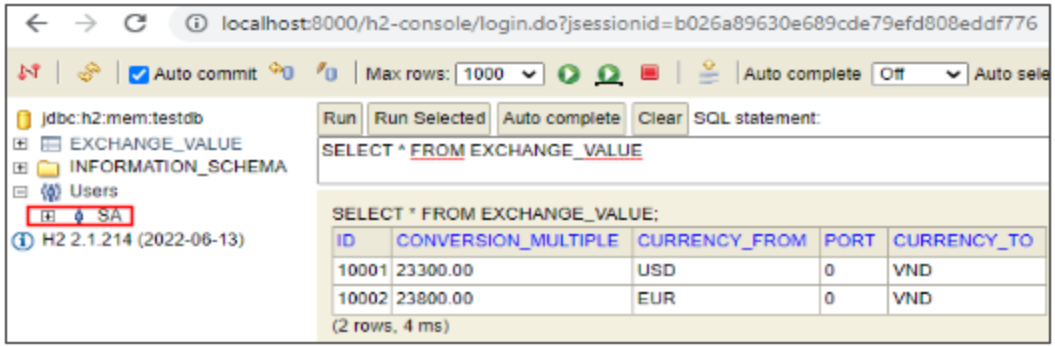


- Chọn item **CurrencyExchangeServiceApplication8000** và bấm vào nút ***copy*** đổi tên (8001) và thêm một vài cấu hình như sau

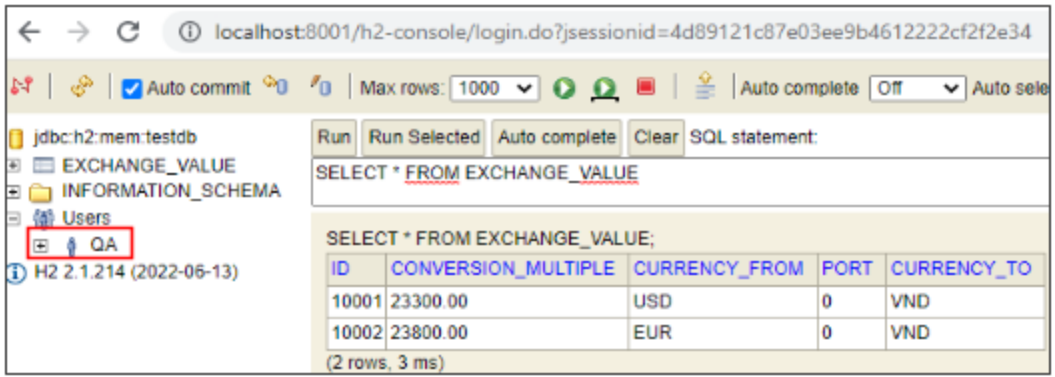


Chạy Module Currency Exchange Service

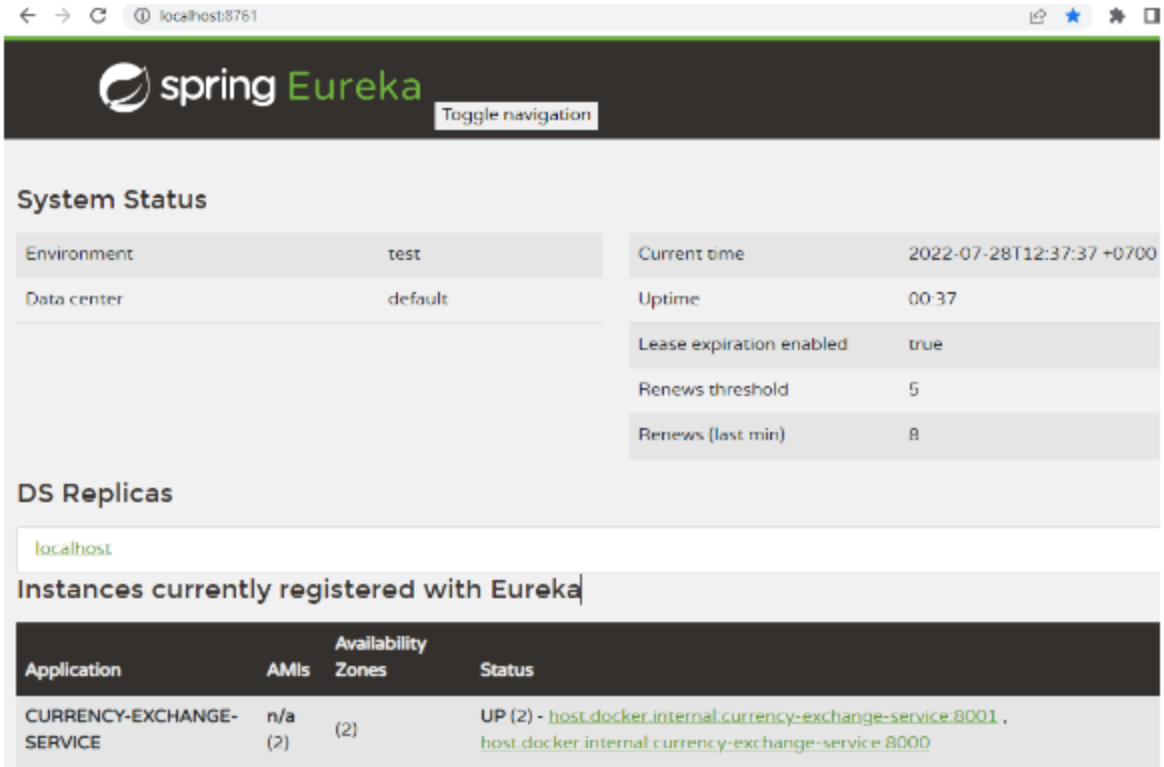
- Khi khởi chạy hai instance ta có thể truy cập vào như sau
- Với cổng **8000** login bằng account **sa**



- Với cổng **8001** login bằng account **qa**

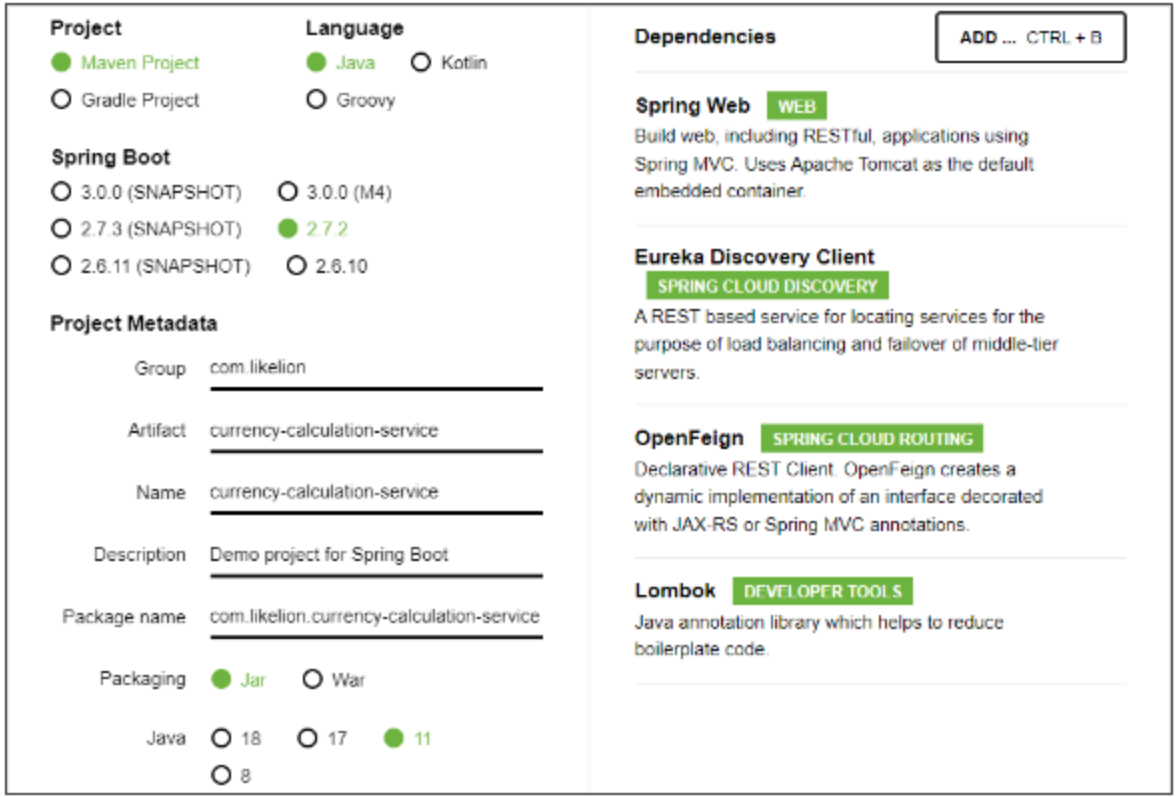


- Truy cập vào http://localhost:8761/ để kiểm tra các eureka client



Tạo Module Currency Calculation Service

- Truy cập vào https://start.spring.io/ tạo dự án như hình



- Kiểm tra file pom.xml như sau

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Thêm hai annotation **@EnableEurekaClient** và **@EnableFeignClients** vào class main module

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class CurrencyCalculationServiceApplication {

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

}
```

- Đổi tên file **application.properties** bằng **application.yml** như sau

```
server:
  port: 8100

spring:
  application:
    name: currency-calculation-service

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka # đăng ký với eureka server đây là một eureka client
    fetch-registry: true
    register-with-eureka: true
```

- **Feign** là một HTTP client cho Java, được phát triển bởi Netflix. Mục tiêu của Fiegn là giúp **đơn giản hóa** HTTP API Client
- Tương tự với các thư viện khác, Feign giúp bạn dễ dàng xử lý dữ liệu JSON hoặc XML sau đó phân tích cú pháp thành Plain Old Java Objects (POJOs)
- Tất cả các yêu cầu **GET**, **POST**, **PUT**, và **DELETE** đều có thể được thực thi
- Ý tưởng của Feign là sử dụng **interface** và các **annotation** để định nghĩa các phương thức **request** đến **API**
- Các Annotations để mô tả yêu cầu HTTP
  - Request method: Mỗi phương thức phải có Annotation HTTP cung cấp request method và URL. Feign sử dụng **@RequestLine**
  - Header manipulation: Gán thông tin static header bằng cách sử dụng annotation **@Header**
  - Url manipulation: Sử dụng URL 1 cách động dựa vào biến truyền vào, bằng cách sử dụng anotation **@Path**
  - Request body: Một đối tượng có thể được chỉ định để sử dụng làm phần thân yêu cầu HTTP với Annotation **@Body**

Cài Đặt Module Currency Calculation Service

- Tại thư mục chính module **currency calculation service** tạo các package **config, controller, model, facade**
- Tại package **config** tạo class **CurrencyCalculationConfig**

```
@Configuration
public class CurrencyCalculationConfig {
    @Bean
    @LoadBalanced // annotation này sẽ cân bằng tải cho mỗi requests
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

- Tại package **model** tạo class **CalculatedAmount**

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class CalculatedAmount {
    private Long id;
    private String from;
    private String to;
    private BigDecimal conversionMultiple;
    private BigDecimal quantity;
    private BigDecimal TotalCalculatedAmount;
    private int port;
}
```

- Tại package **facade** tạo interface **CurrencyExchangeProxy**

```
// Định nghĩa tên của module Currency Exchange Service đăng ký tại Eureka Server
@FeignClient(name ="CURRENCY-EXCHANGE-SERVICE")
public interface CurrencyExchangeProxy {
    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    CalculatedAmount retrieveExchangeValue(@PathVariable("from") String from,
                                           @PathVariable("to") String to); // Định nghĩa request gọi đến method của module
}
```

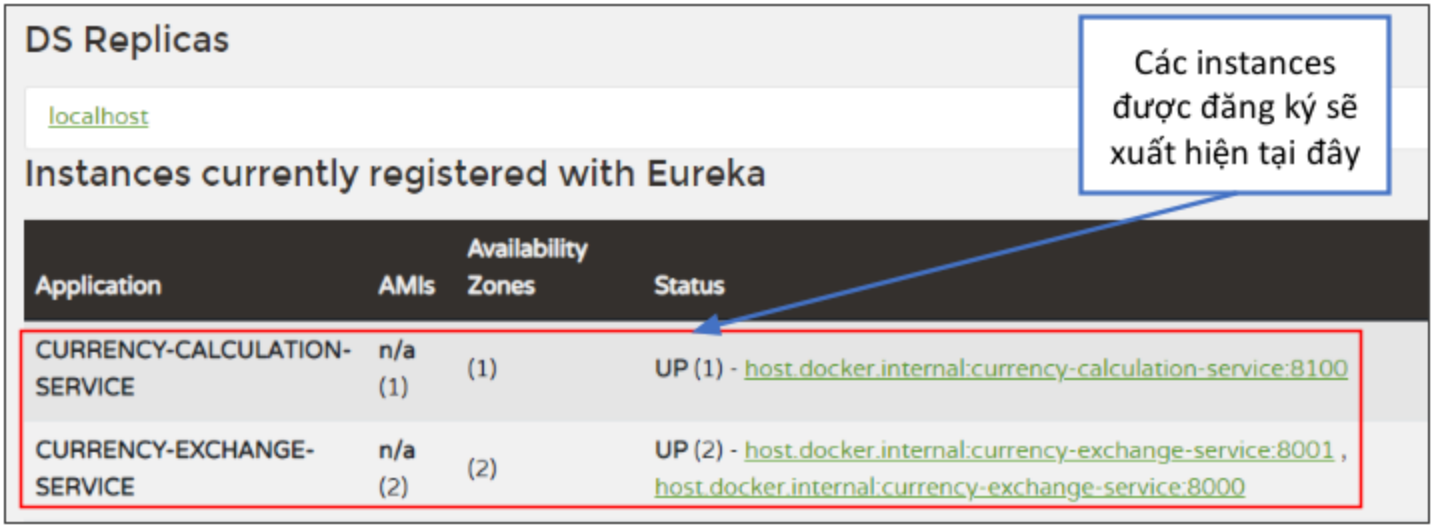
- Interface **CurrencyExchangeProxy** hoạt động như một proxy, cung cấp các giải pháp khác cho **RestTemplate**
- Việc gọi nhiều method đến các service sẽ tạo ra nhiều dòng code khi dùng **RestTemplate**
- Vì vậy với các method ta đưa vào interface và định nghĩa nó là **Feign Client** mỗi khi dùng thì có thể **@Autowired** và gọi tên method. Giúp cho code được tối giản hơn

- Tại package **controller** tạo class **CurrencyCalculationController**

```
@RestController
public class CurrencyCalculationController {
   private Logger logger = LoggerFactory.getLogger(CurrencyCalculationController.class);

   @Autowired
   private CurrencyExchangeProxy proxy;
   @Autowired
   private RestTemplate restTemplate;

   @GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
   public CalculatedAmount calculateAmount(@PathVariable String from, @PathVariable String to,
                                           @PathVariable BigDecimal quantity) {
      Map<String, String> uriVariables = new HashMap<>();
      uriVariables.put("from", from);
      uriVariables.put("to", to);
      ResponseEntity<CalculatedAmount> responseEntity = restTemplate.getForEntity("http://CURRENCY-EXCHANGE-SERVICE/currency-exchange/from/{from}/to/{to}"
            , CalculatedAmount.class, uriVariables);
      CalculatedAmount calculatedAmount =responseEntity.getBody();
      return new CalculatedAmount(calculatedAmount.getId(), calculatedAmount.getFrom(),
            calculatedAmount.getTo(), calculatedAmount.getConversionMultiple(),
            quantity, quantity.multiply(calculatedAmount.getConversionMultiple()),
            calculatedAmount.getPort());
   }

   @GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")
   public CalculatedAmount calculateAmountFeign(@PathVariable String from, @PathVariable String to,
                                                @PathVariable BigDecimal quantity) {
      CalculatedAmount calculatedAmount= proxy.retrieveExchangeValue(from, to);
      logger.info("Port: " + calculatedAmount.getPort());
      return new CalculatedAmount(calculatedAmount.getId(), calculatedAmount.getFrom(),
            calculatedAmount.getTo(), calculatedAmount.getConversionMultiple(),
            quantity, quantity.multiply(calculatedAmount.getConversionMultiple()),
            calculatedAmount.getPort());
   }
}
```

Chạy Module Currency Calculation Service

- Trước tiên chạy module **eureka server**
- Tiếp theo chạy module **config server**
- Tiếp theo chạy module **currency exchange service 8000**
- Tiếp theo chạy module **currency exchange service 8001**
- Cuối cùng chạy module **currency calculation service**
- Truy cập vào http://localhost:8761/ để kiểm tra các eureka client



Chạy Module Currency Calculation Service

- Truy cập vào hai API để kiểm tra kết quả
- Truy cập vào API **RestTemplate**
  http://localhost:8100/currency-converter/from/USD/to/VND/quantity/100

```
// http://localhost:8100/currency-converter/from/USD/to/VND/quantity/100

{
  "id": 10001,
  "from": "USD",
  "to": "VND",
  "conversionMultiple": 23300.00,
  "quantity": 100,
  "port": 8000,
  "totalCalculatedAmount": 2330000.00
}
```

```
// http://localhost:8100/currency-converter/from/USD/to/VND/quantity/100

{
  "id": 10001,
  "from": "USD",
  "to": "VND",
  "conversionMultiple": 23300.00,
  "quantity": 100,
  "port": 8001,
  "totalCalculatedAmount": 2330000.00
}
```

Khi truy cập lần thứ nhất ta nhận được port 8000
Lần thứ hai ta nhận được port 8001
→ **Load Balancer** đang hoạt động

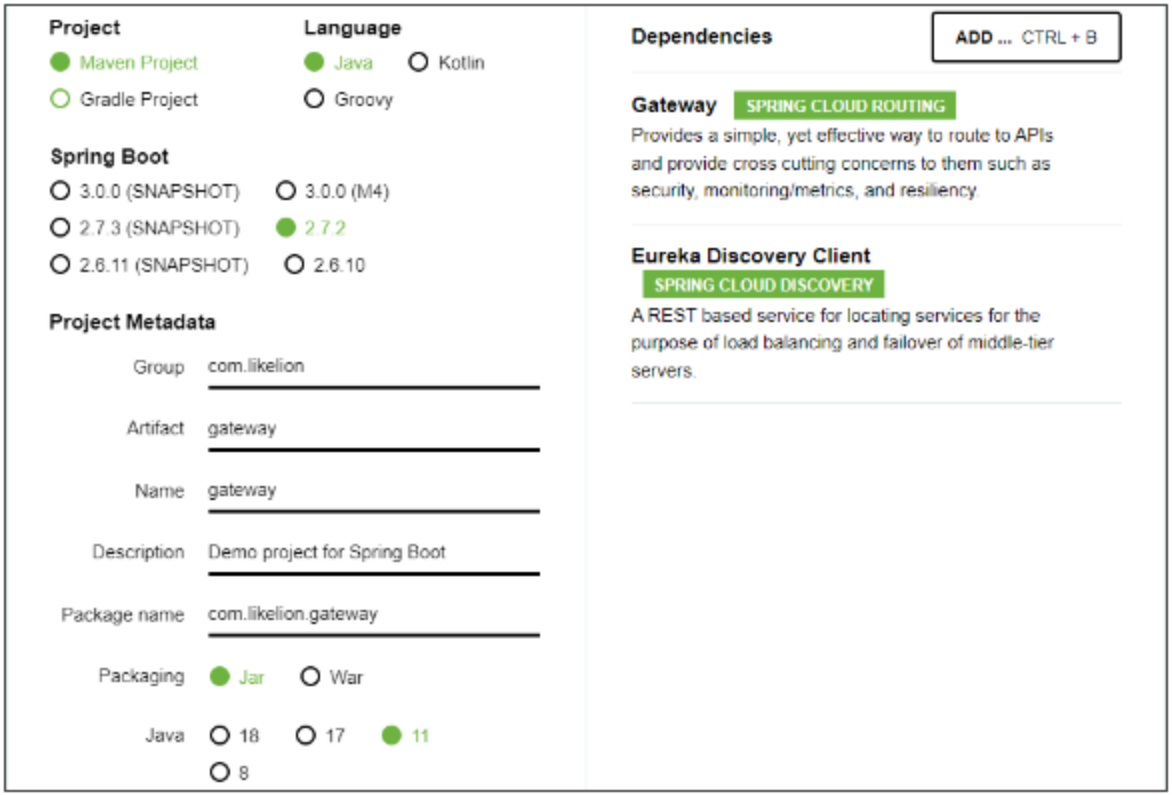
- Tương tự như vậy, truy cập vào API **FeignClient** http://localhost:8100/currency-converter-feign/from/USD/to/VND/quantity/1000

```
// http://localhost:8100/currency-converter-feign/from/USD/to/VND/quantity/1000

{
  "id": 10001,
  "from": "USD",
  "to": "VND",
  "conversionMultiple": 23300.00,
  "quantity": 1000,
  "port": 8000,
  "totalCalculatedAmount": 23300000.00
}
```

```
// http://localhost:8100/currency-converter-feign/from/USD/to/VND/quantity/1000

{
  "id": 10001,
  "from": "USD",
  "to": "VND",
  "conversionMultiple": 23300.00,
  "quantity": 1000,
  "port": 8001,
  "totalCalculatedAmount": 23300000.00
}
```

Khi truy cập lần thứ nhất ta nhận được port 8000
Lần thứ hai ta nhận được port 8001
→ **Load Balancer** đang hoạt động

- Truy cập vào https://start.spring.io/ tạo dự án như hình



- Kiểm tra file pom.xml như sau

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- Thêm hai annotation **@EnableEurekaClient** vào class main module

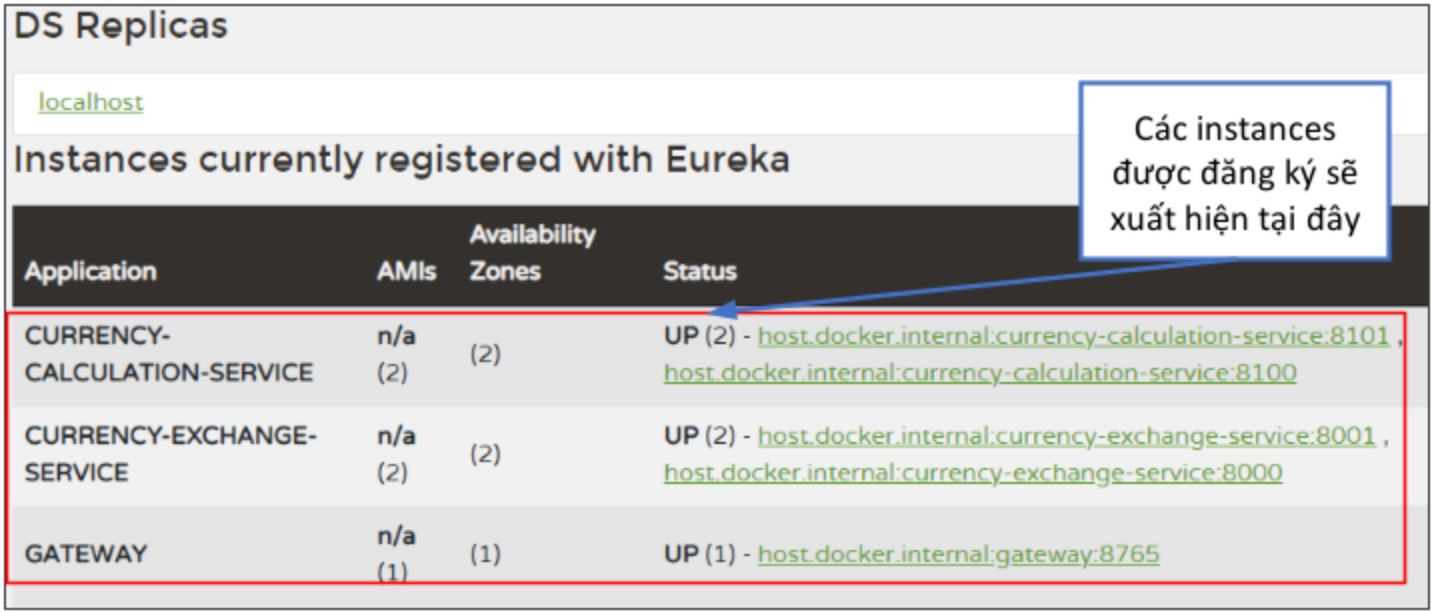
```
@SpringBootApplication
@EnableEurekaClient
public class GatewayApplication {

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

}
```

- Đổi tên file **application.properties** bằng **application.yml** như sau

```
server:
  port: 8765

spring:
  application:
    name: gateway
  cloud:
    gateway:
      routes: # đăng ký gateway cho module currency calculation service. Lưu ý property uri sử dụng lb:// <tên module> được đăng ký tại Eureka Server
        - id: calculation-service
          uri: lb://CURRENCY-CALCULATION-SERVICE/
          predicates:
            - Path=/**

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```

Chạy Module Gateway

- Trước tiên chạy module **eureka server**
- Tiếp theo chạy module **config server**
- Tiếp theo chạy các instances module **currency exchange service**
- Tiếp theo chạy các instances module **currency calculation service**
- Cuối cùng chạy module **gateway**



- Truy cập vào API của module **gateway** để kiểm tra

```
// http://localhost:8765/currency-converter-feign/from/USD/to/VND/quantity/1000

{
  "id": 10001,
  "from": "USD",
  "to": "VND",
  "conversionMultiple": 23300.00,
  "quantity": 1000,
  "port": 8000,
  "totalCalculatedAmount": 23300000.00
}
```

Tổng Kết Nội Dung

- **MSA** là cách phát triển ứng dụng bằng cách **chia nhỏ** các nghiệp vụ thành các dự án nhỏ
- Theo cách tiếp cận truyền thống dự án sẽ gặp **vấn đề lớn** khi về cuối tiến độ. **Tốn nhiều** chi phí coding và testing. Rất khó **bảo trì** dự án sau này
- Đối với MSA Mỗi một thành viên trong dự án sẽ **phụ trách** một service
- **Eureka Server** là một máy chủ dùng để **quản lý**, **đặt tên** cho các service, hay còn gọi là **service registry**
- Service Registry giữ các **thực thể** microservices và **địa chỉ** của chúng. Thực thể microservices được đăng kí với service registry khi bắt đầu **chạy** và hủy đăng kí khi **tắt**
- Các **Eureka client service** sẽ phải đăng ký với **Eureka Server** để có thể biết được sự tồn tại của nó và cung cấp **ip**, **port** để có thể **giao tiếp**
- **Feign** là một **HTTP** client cho **Java**, được phát triển bởi **Netflix**. Mục tiêu của Fiegn là giúp **đơn giản hóa** HTTP API Client
- Ý tưởng của Feign là sử dụng **interface** và các **annotation** để định nghĩa các phương thức **request** đến API
- Biết cách xây dựng các module trong microservice. Xây dựng các tầng cho mỗi module
- Cấu hình cho các module bằng các file properties, yml. Biết cách kết nối và thực hiện giao tiếp giữa các module

Cảm Ơn Bạn Đã Chăm Chỉ!