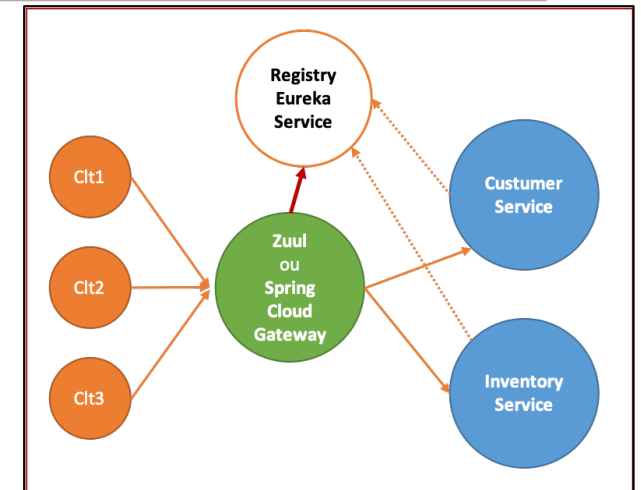


TP2 (suite)


Application: Eureka Discovery Service: Dynamic Routing

Représente un annuaire qui permet d'enregistrer la localisation de toutes les instances des micro-services de l'application. Au démarrage, chaque micro-service se connecte à ce micro-service pour enregistrer le nom du micro-service et l'URI du micro-service incluant l'adresse IP de la machine et le numéro de port. Un exemple de Discovery service fourni par Spring Cloud est Eureka Discovery, une implémentation de Netflix.



Application: Eureka Discovery Service: Dynamic Routing

start.spring.io



Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Java** ☐ Kotlin ☐ Groovy

☒ **Maven**

Spring Boot

☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (RC1) ☐ 3.1.6 (SNAPSHOT) ☐ 3.1.5

☐ 3.0.13 (SNAPSHOT) ☐ 3.0.12 ☐ 2.7.18 (SNAPSHOT) ☒ **2.7.17**

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ **Jar** ☐ War

Java ☐ 21 ☒ **17** ☐ 11 ☐ 8

Selected dependencies:

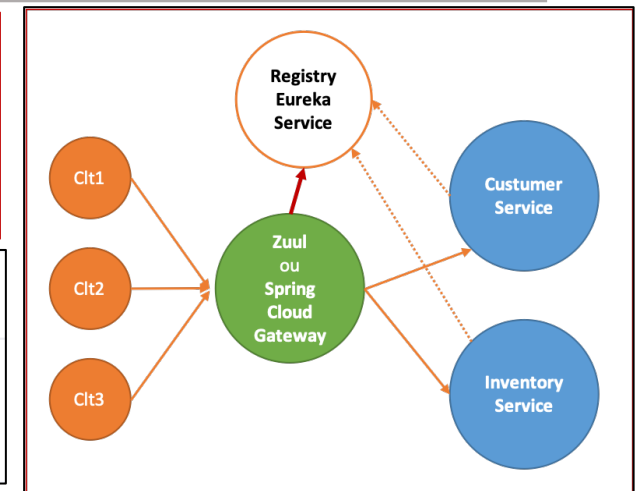
- **Eureka Server** : spring-cloud-netflix Eureka Server.

application.properties

Ne pas cacher les clients locaux

```
1 server.port=8761
2 eureka.client.fetch-registry=false
3 eureka.client.register-with-eureka=false
```

Ne pas s'auto-enregistrer



```
@SpringBootApplication
```

```
@EnableEurekaServer
```

```
public class DiscoveryServiceApplication {
```

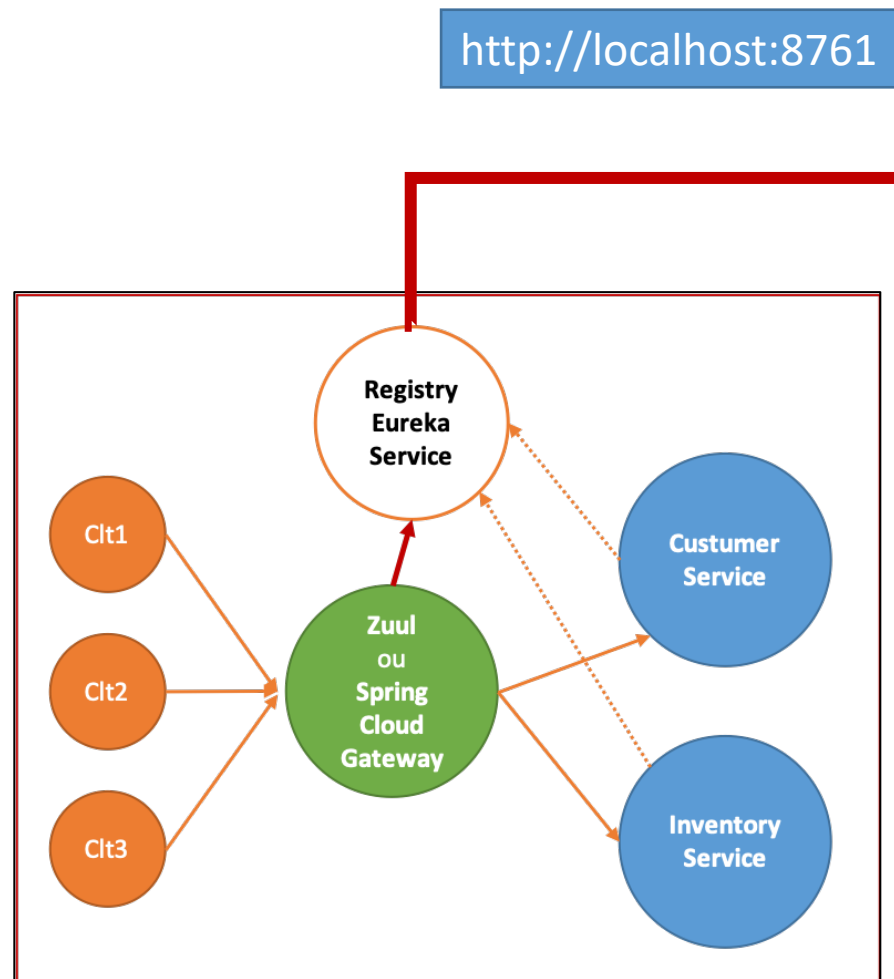
```
    public static void main(String[] args) {
```

```
        SpringApplication.run(DiscoveryServiceApplication.class, args);
```

```
    }
```

```
}
```

Application: Eureka Discovery Service: Dynamic Routing



Screenshot of the Spring Eureka web interface at `localhost:8761`.

System Status

Environment	test	Current time	2023-10-21T19:18:08+0100
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

DS Replicas

[localhost](#)

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	128mb
num-of-cpus	10
current-memory-usage	59mb (46%)
server-uptime	00:00
registered-replicas	<code>http://localhost:8761/eureka/</code>
unavailable-replicas	<code>http://localhost:8761/eureka/</code>
available-replicas	

Instance Info

Application: Eureka Discovery Service: Dynamic Routing

Permettre à Customer-service et Inventory-service de s'enregistrer chez Eureka server

Customer-service

```
spring.datasource.url=jdbc:h2:mem:inventory-db
spring.h2.console.enabled=true
server.port=8082
spring.application.name=inventory-service
management.endpoints.web.exposure.include=*
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

application.properties

Inventory-service

```
spring.datasource.url=jdbc:h2:mem:customer-db
spring.h2.console.enabled=true
server.port=8081
spring.application.name=customer-service
management.endpoints.web.exposure.include=*
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

application.properties

The screenshot shows the Spring Eureka web interface. At the top, there's a header with the 'spring Eureka' logo and navigation links 'HOME' and 'LAST 1000 SINCE STARTUP'. Below the header, the 'System Status' section displays a table with system metrics: Environment (test), Data center (default), Current time (2023-10-21T19:39:14+0100), Uptime (00:21), Lease expiration enabled (true), Renewal threshold (5), and Renewal (last min) (8). Below this, the 'DS Replicas' section shows a dropdown menu with 'localhost' selected. The 'Instances currently registered with Eureka' section contains a table listing the registered services.

Application	AMIs	Availability Zones	Status
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.104:customer-service:8081
INVENTORY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.104:inventory-service:8082

Application: Eureka Discovery Service: Dynamic Routing

Routage dynamique de la gateway

GatewayServiceApplication.java

```
@Bean
DiscoveryClientRouteDefinitionLocator dynamicRoutes (ReactiveDiscoveryClient rdc,
                                                    DiscoveryLocatorProperties dlp){
    return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
}
```

application.properties

```
spring.application.name=gateway-service
spring.cloud.discovery.enabled=true
server.port=8888
```

← → ↻ ⓘ localhost:8888/INVENTORY-SERVICE/products/1

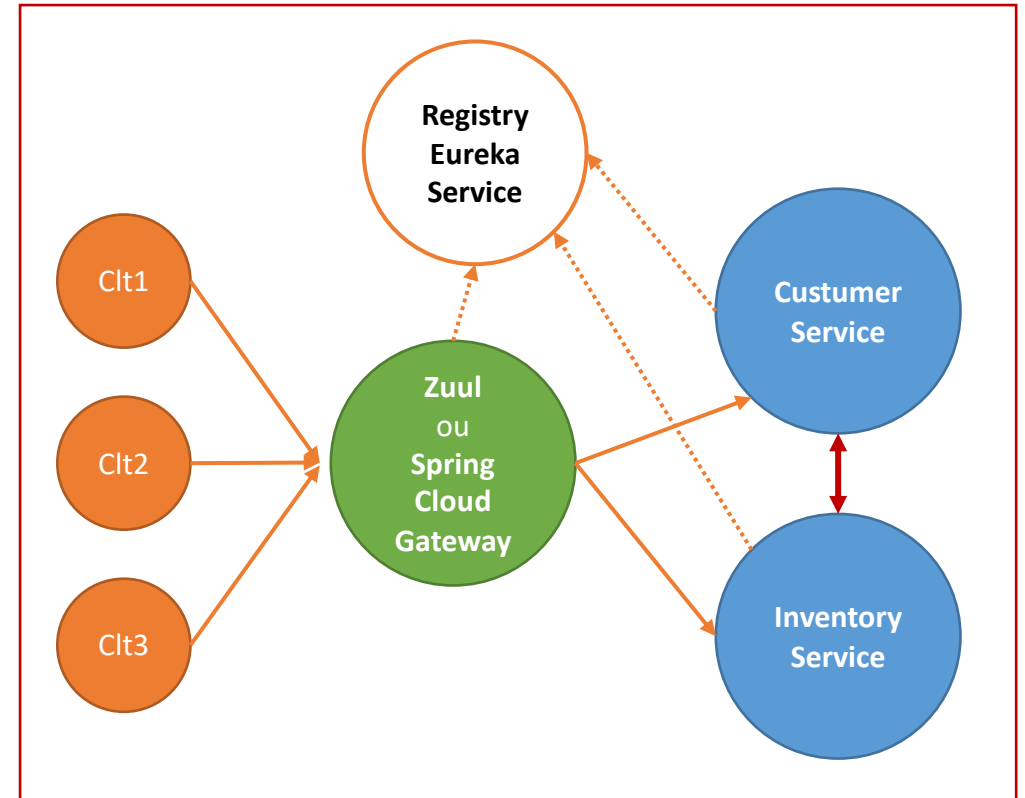
```
{
  "name" : "HP Computer",
  "price" : 900.0,
  "_links" : {
    "self" : {
      "href" : "http://192.168.1.104:8082/products/1"
    },
    "product" : {
      "href" : "http://192.168.1.104:8082/products/1"
    }
  }
}
```

← → ↻ ⓘ localhost:8888/CUSTOMER-SERVICE/customers/1

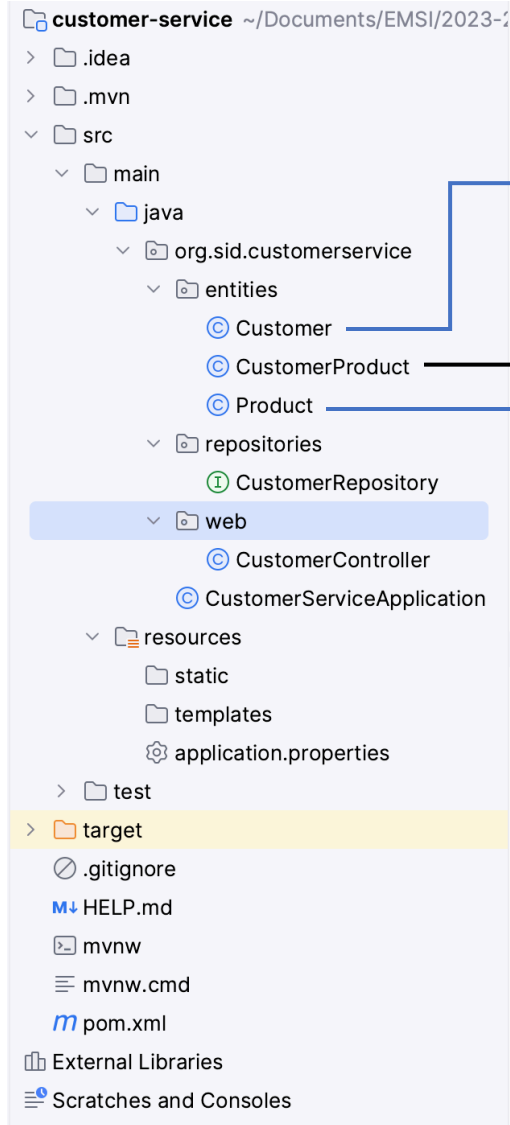
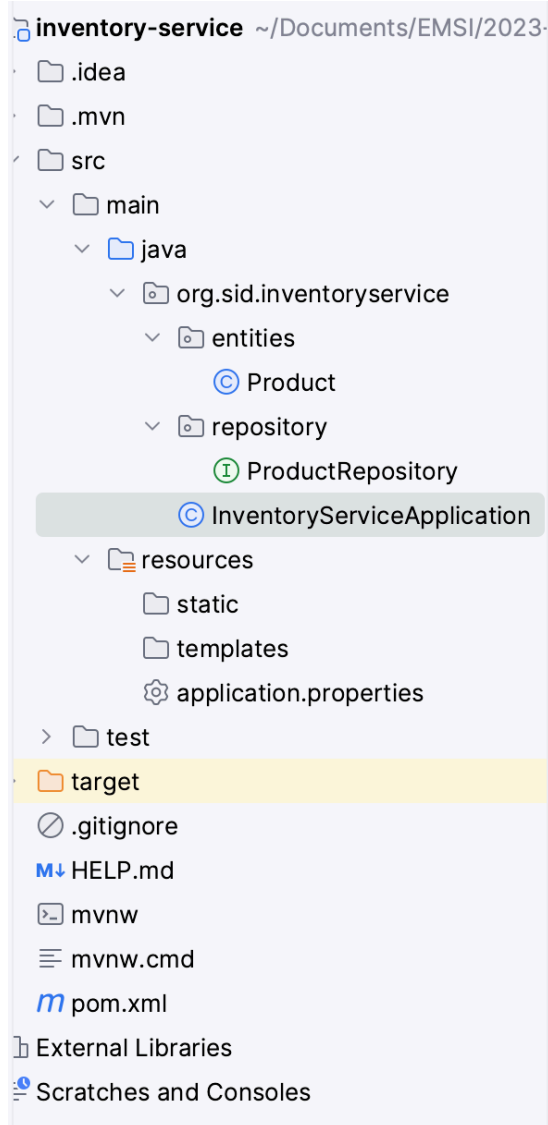
```
{
  "name" : "imane",
  "email" : "imane@gmail.com",
  "_links" : {
    "self" : {
      "href" : "http://192.168.1.104:8081/customers/1"
    },
    "customer" : {
      "href" : "http://192.168.1.104:8081/customers/1"
    }
  }
}
```

Application: RestTemplate

RestTemplate est un outil de communication utilisé pour envoyer des requêtes HTTP de manière synchrone en utilisant une interface simple de type modèle. En d'autres termes, c'est un composant conçu pour appeler des services REST de manière synchronisée. Il joue un rôle essentiel dans la communication entre les microservices de Spring Boot.



Application: RestTemplate



```
@Entity @Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    private Long productId;
}
```

N'oubliez pas d'ajouter les
productId dans les champs
cree dans
CustomerServiceApplication

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CustomerProduct {
    private Long id;
    private String customerName;
    private String email;
    private String productName;
    private double price;
}
```

```
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
}
```


Application: RestTemplate

```
@RestController
@RequestMapping("/api")
public class CustomerController {
    2 usages
    @Autowired
    private final ApplicationPropertiesConfiguration applicationPropertiesConfiguration;
    3 usages
    @Autowired
    CustomerRepository customerRepository;
    1 usage
    @Autowired
    RestTemplate restTemplate;
    no usages
    public CustomerController(ApplicationPropertiesConfiguration applicationPropertiesConfiguration) {
        this.applicationPropertiesConfiguration = applicationPropertiesConfiguration;
    }
    no usages
    @RequestMapping("/customers")
    public List<Customer> getCustomers() {
        List<Customer> customers = customerRepository.findAll();
        List<Customer> limitCustomers = customers.subList(0,applicationPropertiesConfiguration.getLimitCustomers());
        return limitCustomers;
    }

    no usages
    @RequestMapping("/getCustomerProducts/{id}")
    public CustomerProduct getCustomerProducts(@PathVariable("id") Long id) {
        Optional<Customer> customer = customerRepository.findById(id);

        Product product = restTemplate.getForObject( url: "http://localhost:8888/INVENTORY-SERVICE/products/" + customer.get().getProductId(), Product.class);
        return new CustomerProduct(customer.get().getId(), customer.get().getName(), customer.get().getEmail(), product.getName(), product.getPrice());
    }

    no usages
    @RequestMapping("/customers/{id}")
    public Optional<Customer> getCustomer(@PathVariable("id") Long id) {
        return customerRepository.findById(id);
    }
}
```



localhost:8888/CUSTOMER-SERVICE/api/getCustomerProducts/2

{"id":2,"customerName":"laila","email":"laila@gmail.com","productName":"Ipad","price":500.0}



localhost:8888/CUSTOMER-SERVICE/api/getCustomerProducts/3

{"id":3,"customerName":"ahmed","email":"ahmed@gmail.com","productName":"Keyboard","price":20.0}

Récupérer le Customer

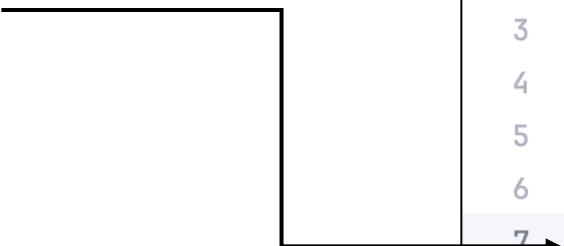
Récupérer le Produit
du Customer

Rassembler les
données dans un seul
objet

Application: CONFIGURATION NON EXTERNALISÉE

Créer une valeur limite du nombre de customers à retourner quand on fait appel à l'URI /customers

- Au niveau du fichier « application.properties »
Ajouter la propriété :



```
1 spring.datasource.url=jdbc:h2:mem:customer-db
2 spring.h2.console.enabled=true
3 server.port=8081
4 spring.application.name=customer-service
5 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
6 #management.endpoints.web.exposure.include=*
7 mes-configs.limitCustomers= 2
8
```

- Veiller à ce que tous les noms des propriétés soient précédés d'un préfixe afin qu'elles soient identifiables: Dans notre cas, le préfixe est `mes-configs`

Application: CONFIGURATION NON EXTERNALISÉE

- Afin de récupérer les valeurs que nous avons indiqué dans application.properties , on va utiliser `@ConfigurationProperties` dans une classe de configuration dédiée `ApplicationPropertiesConfiguration`



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a `java` package with `org.sid.customerservice` and `configurations` sub-packages. The `configurations` package contains `ApplicationPropertiesConfiguration`. The `resources` package contains `application.properties`. The code editor shows the following code:

```
no usages
7  @Component @ConfigurationProperties("mes-configs")
8  public class ApplicationPropertiesConfiguration {
9      private int limitCustomers;
10
11     public int getLimitCustomers() {
12         return limitCustomers;
13     }
14
15     public void setLimitCustomers(int limitCustomers) {
16         this.limitCustomers = limitCustomers;
17     }
18 }
```

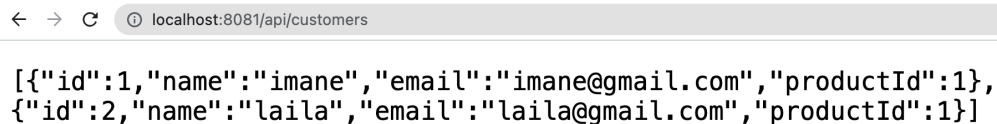
An arrow points from the `limitCustomers` property in the `getLimitCustomers()` method to the `limitCustomers` property in the `setLimitCustomers()` method.

- @Component** : demande à Spring de scanner cette classe à la recherche de configurations.
- @ConfigurationProperties("mes-configs")** : précise que cette classe de configuration va récupérer des propriétés dans `application.properties` dont le préfixe est `mes-configs`.
- Déclarer des propriétés avec les mêmes noms que celles du fichier de configuration. Dans notre cas, il s'agit de `limitCustomers`

Application: CONFIGURATION NON EXTERNALISÉE

Il suffit de retourner dans le contrôleur pour accéder aux valeurs de manière simple.

- **`applicationPropertiesConfiguration.getLimitCustomers()`** va retourner le chiffre **2** défini dans le **fichier de configuration**.
- On le passe a **`subList`** qui coupe une liste donnée à la limite donnée en 2e argument.
- `http://localhost:8081/api/customers`, va retourner les 2 premiers produits



```
[{"id":1,"name":"imane","email":"imane@gmail.com","productId":1},
{"id":2,"name":"laila","email":"laila@gmail.com","productId":1}]
```

```
@RestController
@RequestMapping("/api")
public class CustomerController {

    2 usages
    @Autowired
    private final ApplicationPropertiesConfiguration applicationPropertiesConfiguration;

    3 usages
    @Autowired
    CustomerRepository customerRepository;

    1 usage
    @Autowired
    RestTemplate restTemplate;

    no usages

    public CustomerController(ApplicationPropertiesConfiguration applicationPropertiesConfiguration) {
        this.applicationPropertiesConfiguration = applicationPropertiesConfiguration;
    }

    no usages
    @RequestMapping("/customers")
    public List<Customer> getCustomers() {
        List<Customer> customers = customerRepository.findAll();
        List<Customer> limitCustomers = customers.subList(0, applicationPropertiesConfiguration.getLimitCustomers());
        return limitCustomers;
    }
}
```