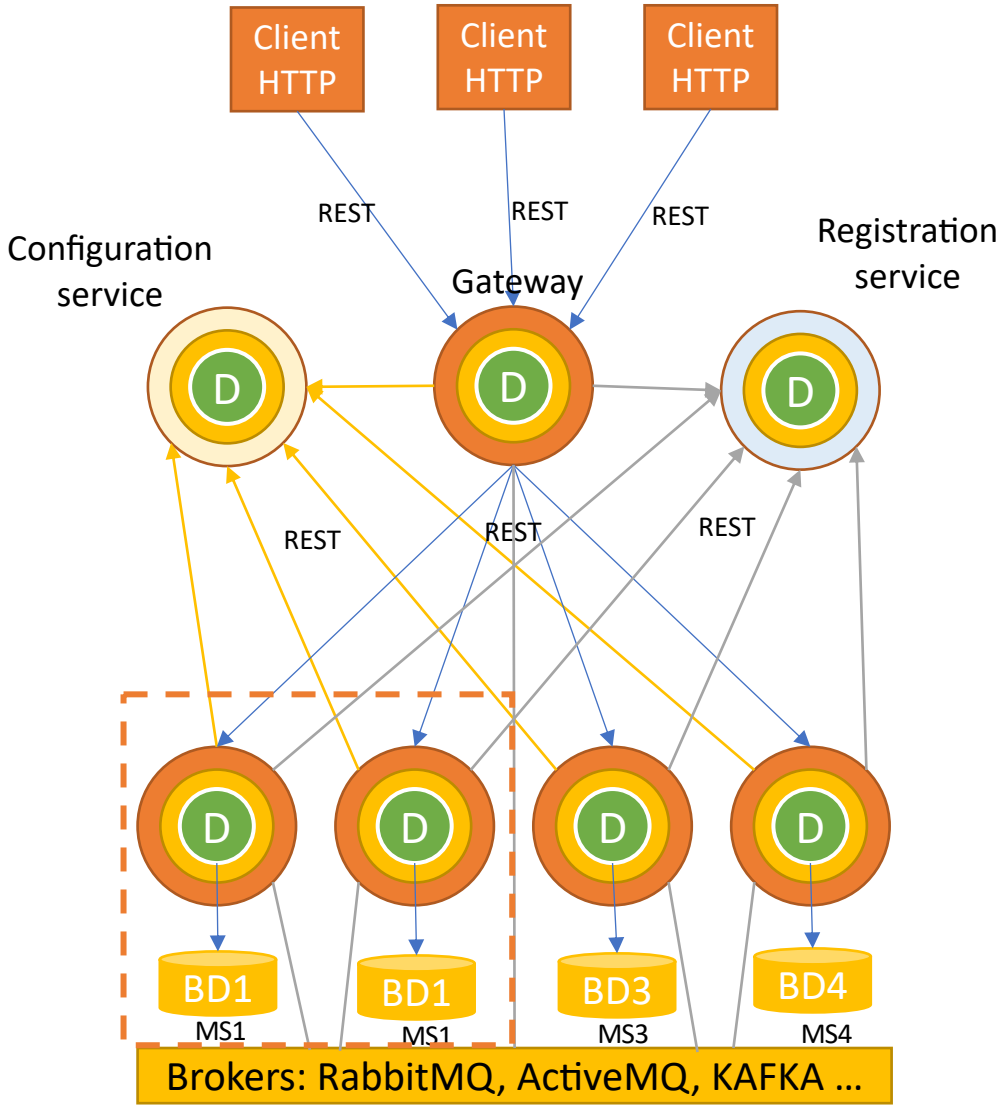
ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

JEE 2

Pr. HACHCHANE Imane

hachchane.emsi@gmail.com

Approche Micro services

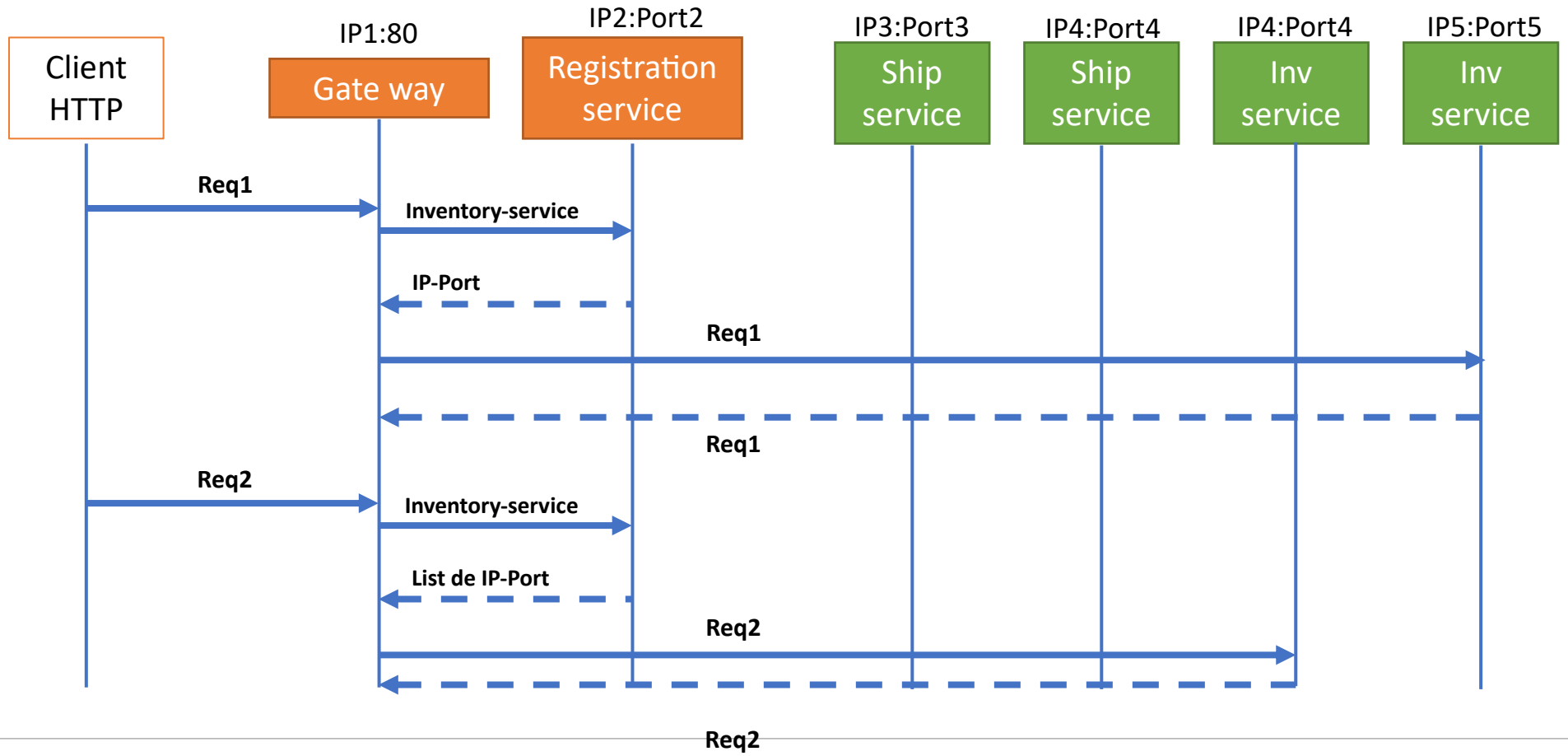


Approche Micro services

Consulter le service via le service proxy

Req 1: GET http://gateway/inventory-service/products

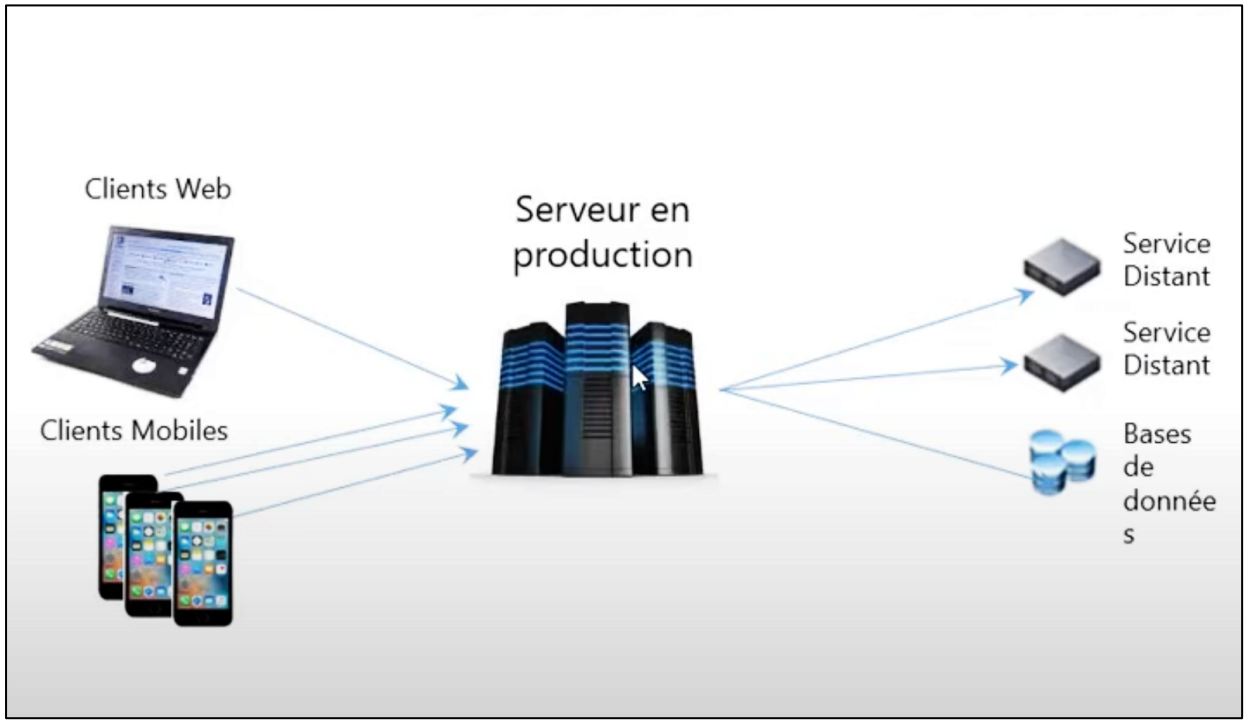
Req 2: GET http://gateway/inventory-service/products



Blocking IO Model : Latency Problem

Le modèle d'entrée/sortie (IO) bloquante : Problème de latence

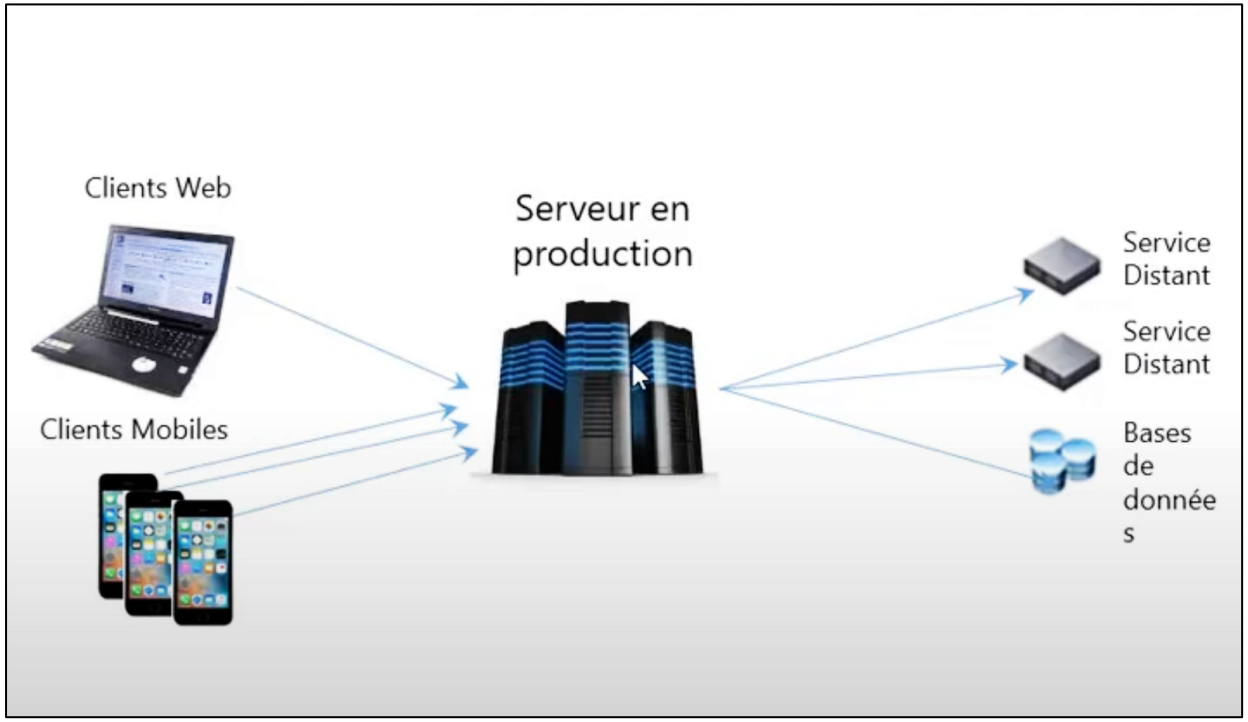
- Les applications qui tournent en production
- Une variété de clients et une variété de services distants qui peuvent être (Bases de données, d'autres services web)
- Problème et contraintes :
  - Des clients qui ont des connexions lentes (Long lived) et qui monopolisent des ressources sur notre serveur
  - Une API distante avec un problème de latence.
  - Ce qui peut ralentir notre service.
  - Voir le rendre complètement indisponible



Blocking IO Model : Latency Problem

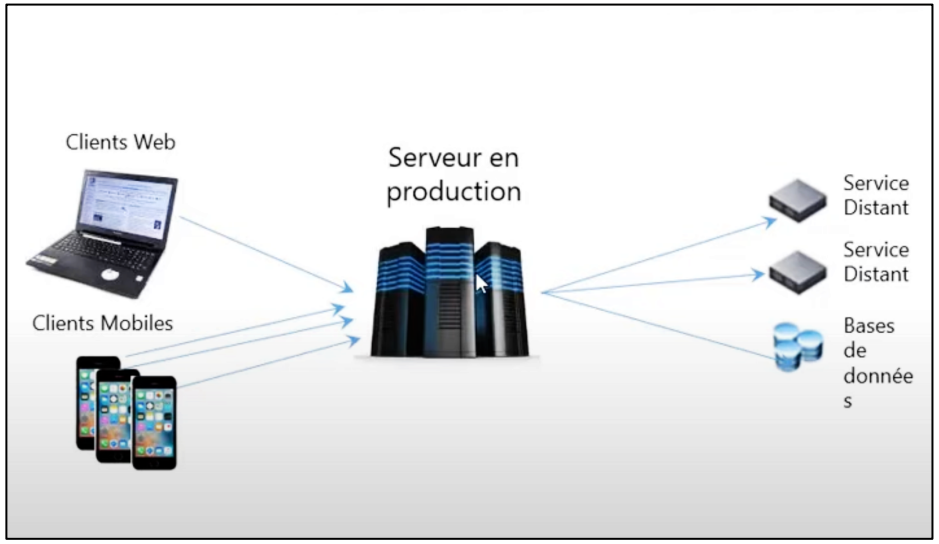
Le modèle d'entrée/sortie (IO) bloquante : Problème de latence

Le modèle d'entrée/sortie bloquante est une approche de gestion des opérations d'entrée/sortie dans un système informatique. Dans ce modèle, lorsqu'un programme effectue une opération d'entrée/sortie, il est bloqué et attend que l'opération se termine avant de pouvoir continuer. Cela peut poser des problèmes de latence, c'est-à-dire des retards dans l'exécution des opérations.



Blocking IO Model : Latency Problem

Le modèle d'entrée/sortie (IO) bloquante : Problème de latence



Les problèmes de latence dans le modèle d'entrée/sortie bloquante peuvent survenir pour plusieurs raisons, notamment :

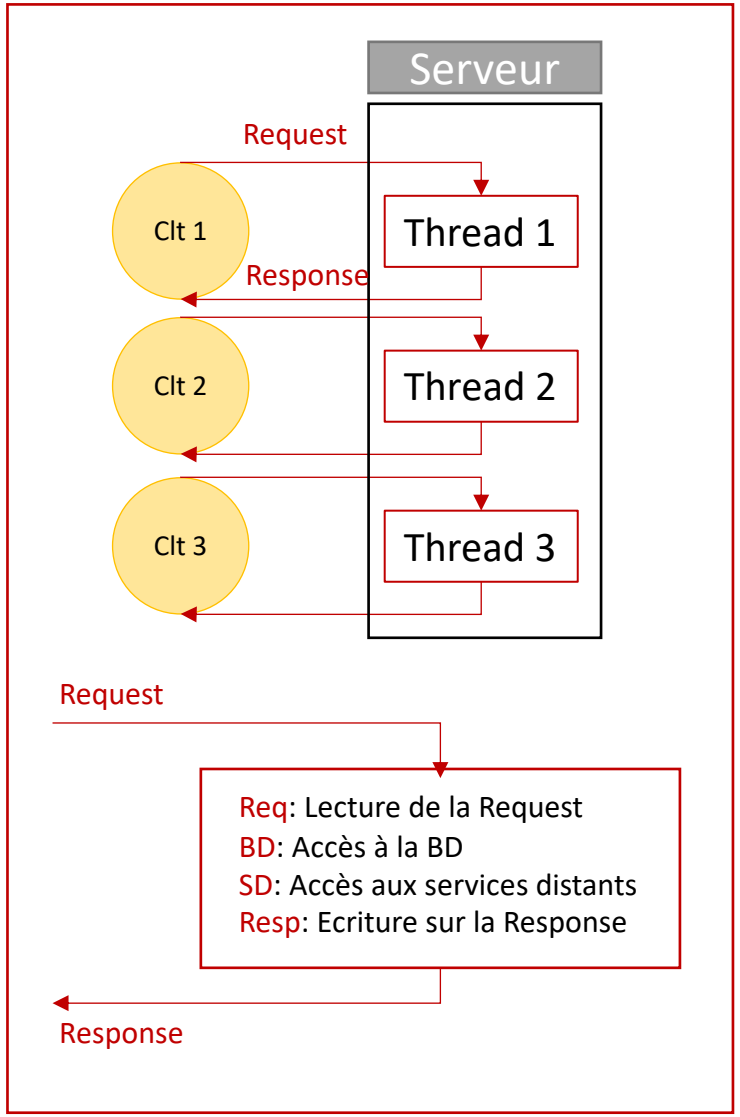
- Attente des opérations de lecture/écriture sur des dispositifs lents : Lorsqu'un programme effectue une opération d'entrée/sortie sur un dispositif de stockage lent, comme un disque dur traditionnel, il peut devoir attendre un certain temps avant que l'opération ne soit terminée, ce qui entraîne une latence.
- Attente des opérations réseau : Lorsqu'un programme communique avec d'autres systèmes via un réseau, il peut y avoir des retards dus aux temps de transmission et de réponse, ce qui peut augmenter la latence.
- Concurrence : Lorsque plusieurs threads ou processus effectuent des opérations d'entrée/sortie en même temps, ils peuvent se bloquer mutuellement, ce qui entraîne une latence supplémentaire.

Multi Threads Bloquant

Le modèle classique Bloquant basé sur une Pool de Threads.

- Marche très bien pour de nombreux cas
- A chaque requête, on affecte un Thread tiré du pool de centaines de threads.
- Le rôle de ce thread étant de gérer le traitement de la requête en question

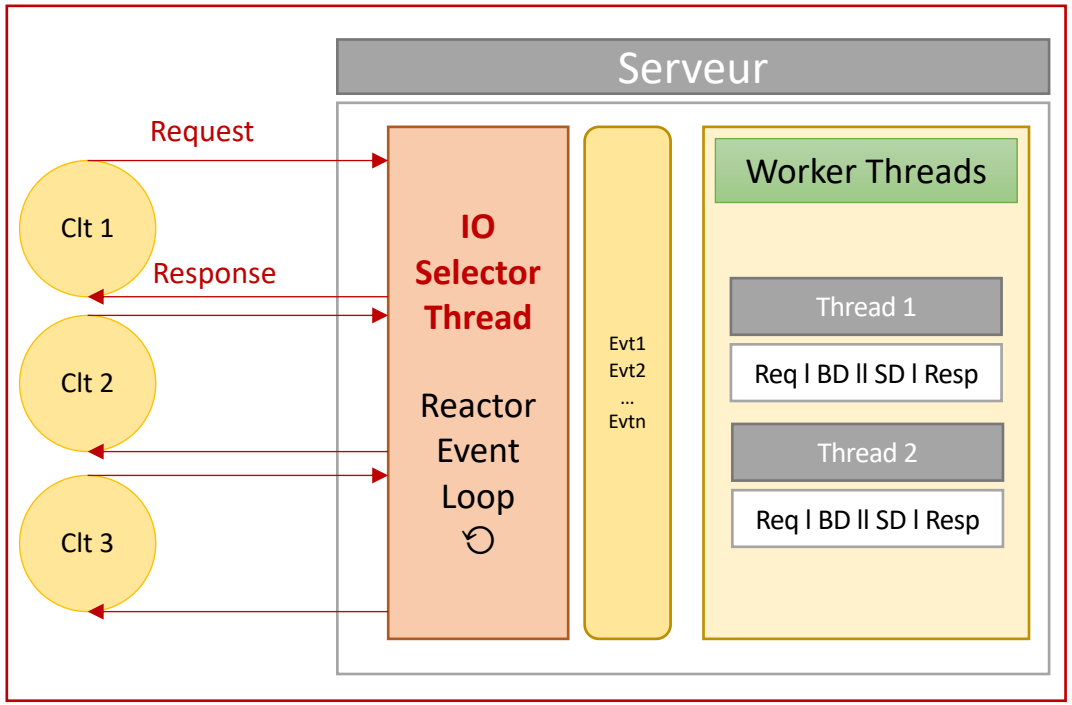
- Pendant ce traitement on peut avoir :
  1. Lecture des données de la requête
  2. Accéder à une base de données
  3. Accéder à des services distants
  4. Ecriture sur la response
- ×Toutes ces Entrées Sorties sont bloquantes
- ×Le thread attend la lecture et l'écriture sur les IO
- Dans le cas d'une connexion lente, le thread est mobilisé pour longtemps coté serveur qui empêche d'exploiter les capacités des ressources du serveur.



Modèle Single Thread Non Bloquant

On peut utiliser un autre modèle de Runtime qui permet de mieux gérer les ressources du serveur :

- Dans ce modèle on n'aura pas besoin d'un Thread par requête / response

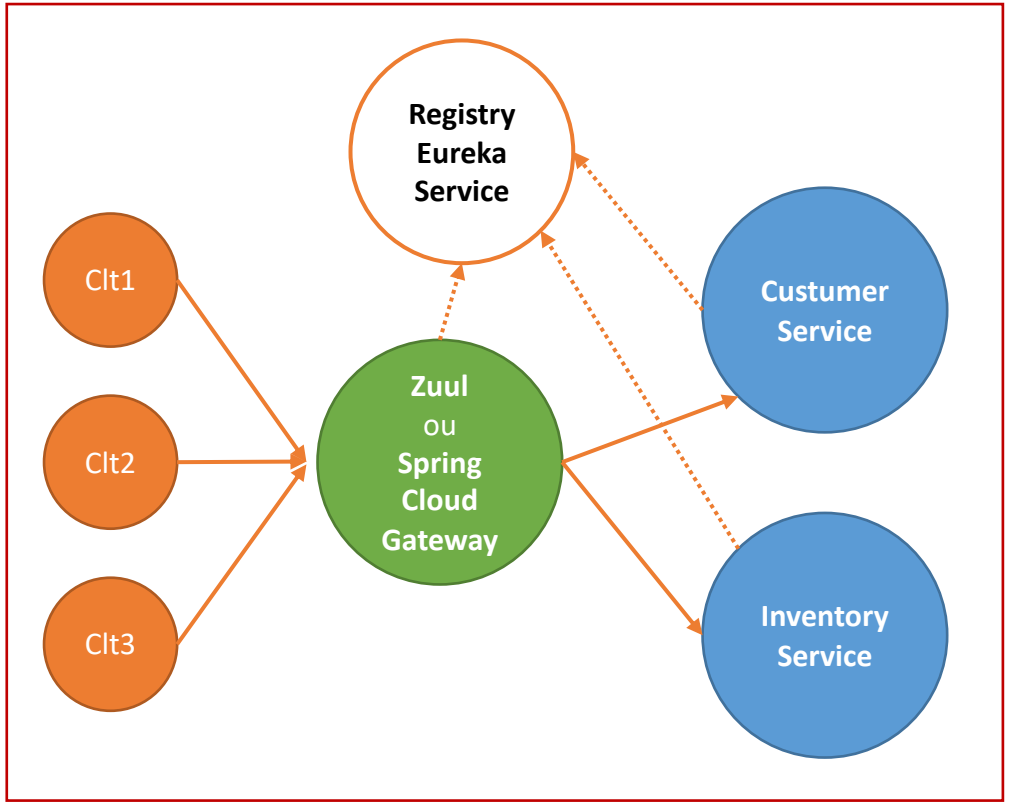


On a un modèle qui utilise un nombre beaucoup plus réduit de threads

- Un IO Selector Thread dont le rôle est d'orchestrer les entrée sorties Non bloquantes.
- Cette fois ci tous les I0 doivent être faites d'une manière non bloquantes. Ce qui fait qu'on va jamais attendre
- Cet IO thread va gérer les lectures et les écritures comme des évènements qu'il va empiler et dépiler dans une Queue d'une manière non bloquante.
- Un nombre réduit de Worker Threads (en fonction du nombre de CPU du serveur)
- Ces Workers Threads vont s'occuper de traiter les requêtes de manière non bloquantes. Il ne vont jamais attendre. Ils seront toujours entrain de travaillr et exploiter aux maximum les ressources du serveur
- Ce modèle assure la scalabilité verticale : les performances augmente avec la capacité du serveur (CPUs, Mémoire, Stockage, etc...)
- La latence des I0 ne va pas impacter les performances du serveur.

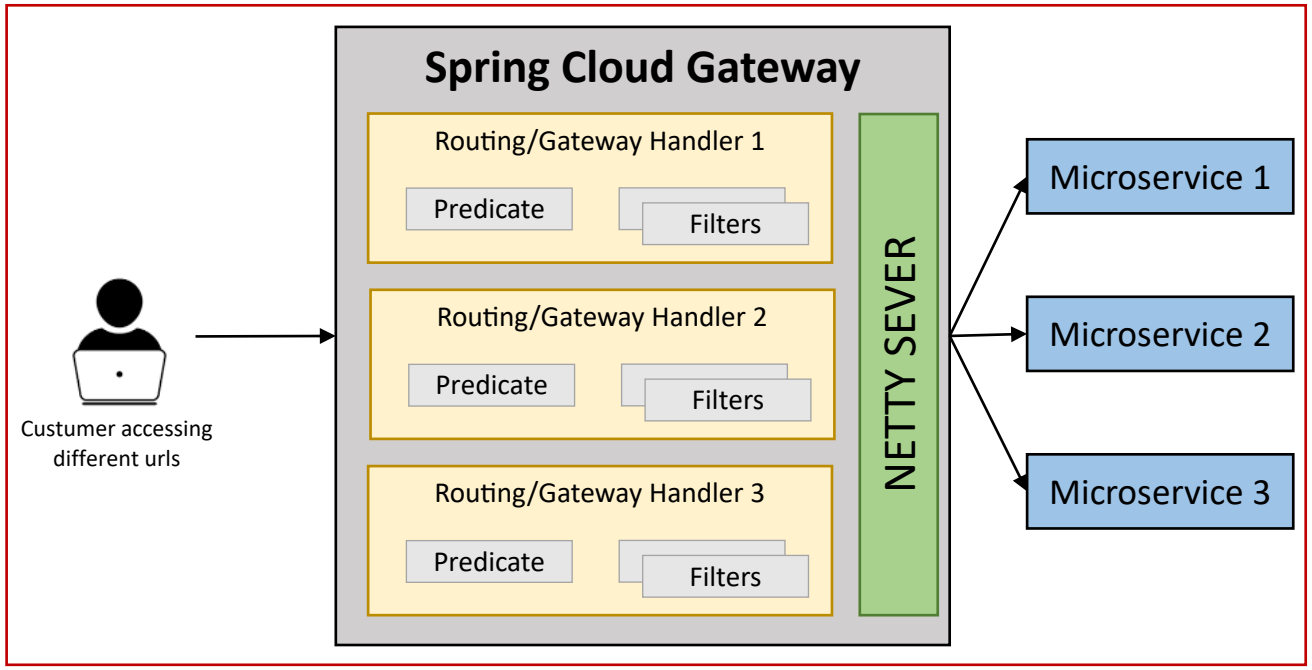
Spring Cloud Gateway

- Gateway API est un reverse proxy amélioré avec des fonctionnalités plus avancées, y compris **l'orchestration** et la **sécurité** et le **monitoring**.
- Quelques implémentations de API Gateway : Netflix Zuul Proxy, Amazon Gateway API, et Spring Cloud Gateway
- **Zuul** est un proxy utilisant une API qui utilise des ***entrées sorties bloquantes.***
  - Une api de passerelle bloquante utilise autant de threads que le nombre de requêtes entrantes.
  - Si aucun thread n'est disponible pour traiter la requête entrante, celle-ci doit attendre dans la file d'attente.
- **Spring Cloud Gateway** est un proxy utilisant une API ***non bloquante***.
  - Un thread est toujours disponible pour traiter requête entrante.
  - Ces requêtes sont ensuite traitées de manière asynchrone en arrière-plan et une fois complétées. la réponse est renvoyée.
  - Ainsi. aucune requête entrante n'est jamais bloquée lors de l'utilisation de Spring Cloud Gateway sauf si les ressources CPU et mémoires sont saturées.



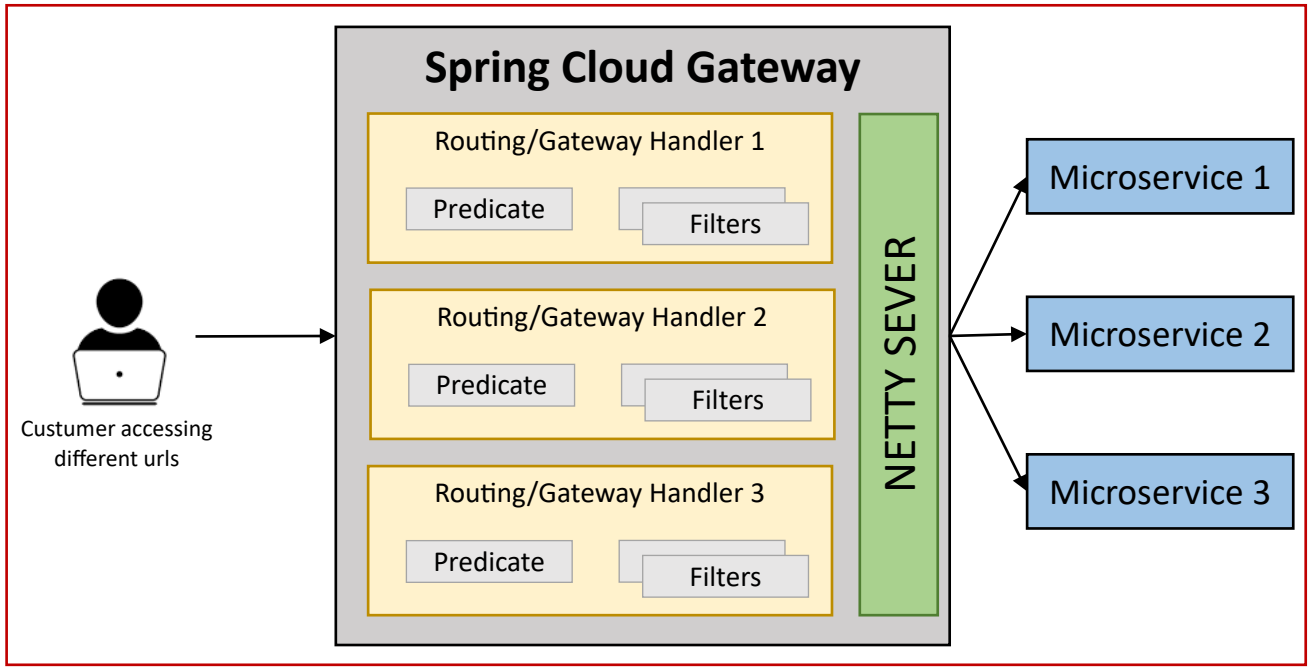
Spring Cloud Gateway

- Spring Cloud Gateway a été introduite dans Spring Cloud 2.x, au-dessus de l'écosystème Reactive Spring.
- Il fournit un moyen simple et efficace d'acheminer les requêtes entrantes vers la destination appropriée à l'aide du Gateway Handler Mapping.
- Et Spring Cloud Gateway utilise le serveur Netty pour fournir un traitement asynchrone non bloquant des requêtes.



Spring Cloud Gateway

- Spring Cloud Gateway se compose de 3 blocs de construction principaux:
  - ×□□W !"#$%&'()*+,-./0123456789:;□ÿ□□ voulons qu'une requête particulière soit acheminée. Une route comprend :
    - ×□□W !"#$%&'()*+,-./
    - Predicate : Une condition qui doit
    - satisfaire
    - Filters : Un ou plusieurs filtres qui peuvent intervenir pour apporter des traitement et des modifications des requêtes et des réponses HTTP



Predicates :

- ×□□W !"#$%&'()*+,
- ×□□W !"#$%&'()*+,-./0
  Cookie, Header, Query
- ×□□W !"#
- ×□□W !

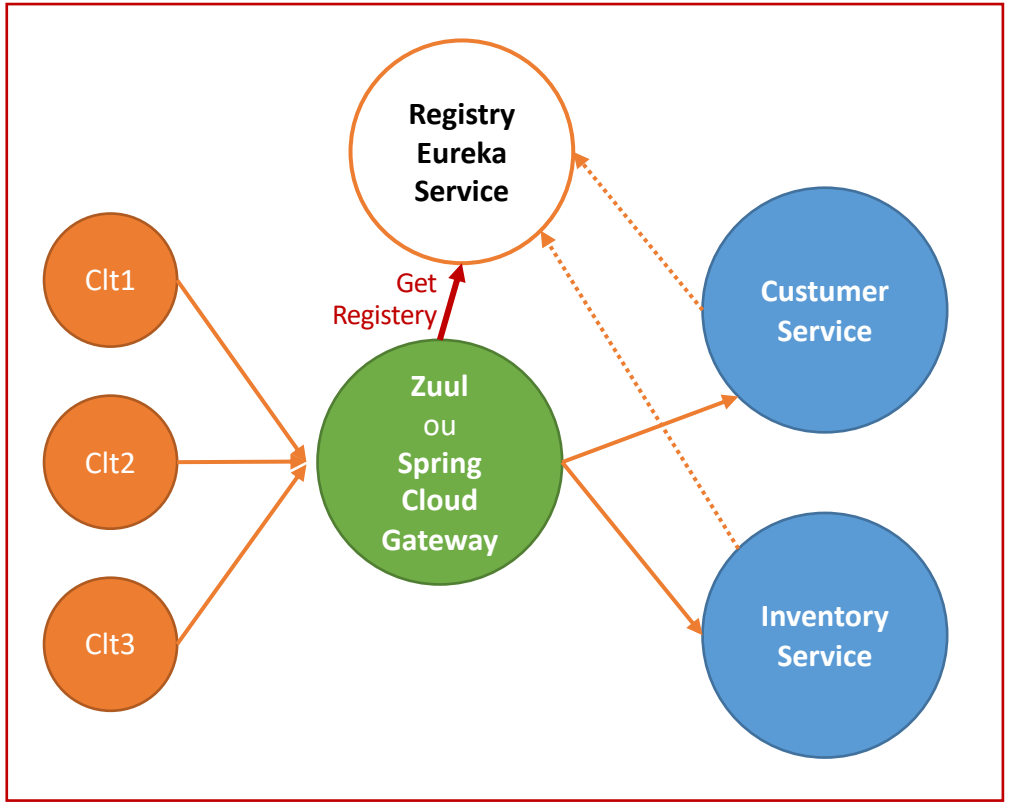
Filters :

•×□□W !"#$%&'()*
•×□□W !"#$%&'()*+,-
AddResponseHeader
DedupeResponseHeader
•Hystrix

•×□□W !"#$%&'(
•×□□W !"#$%
•×□□W !

Application

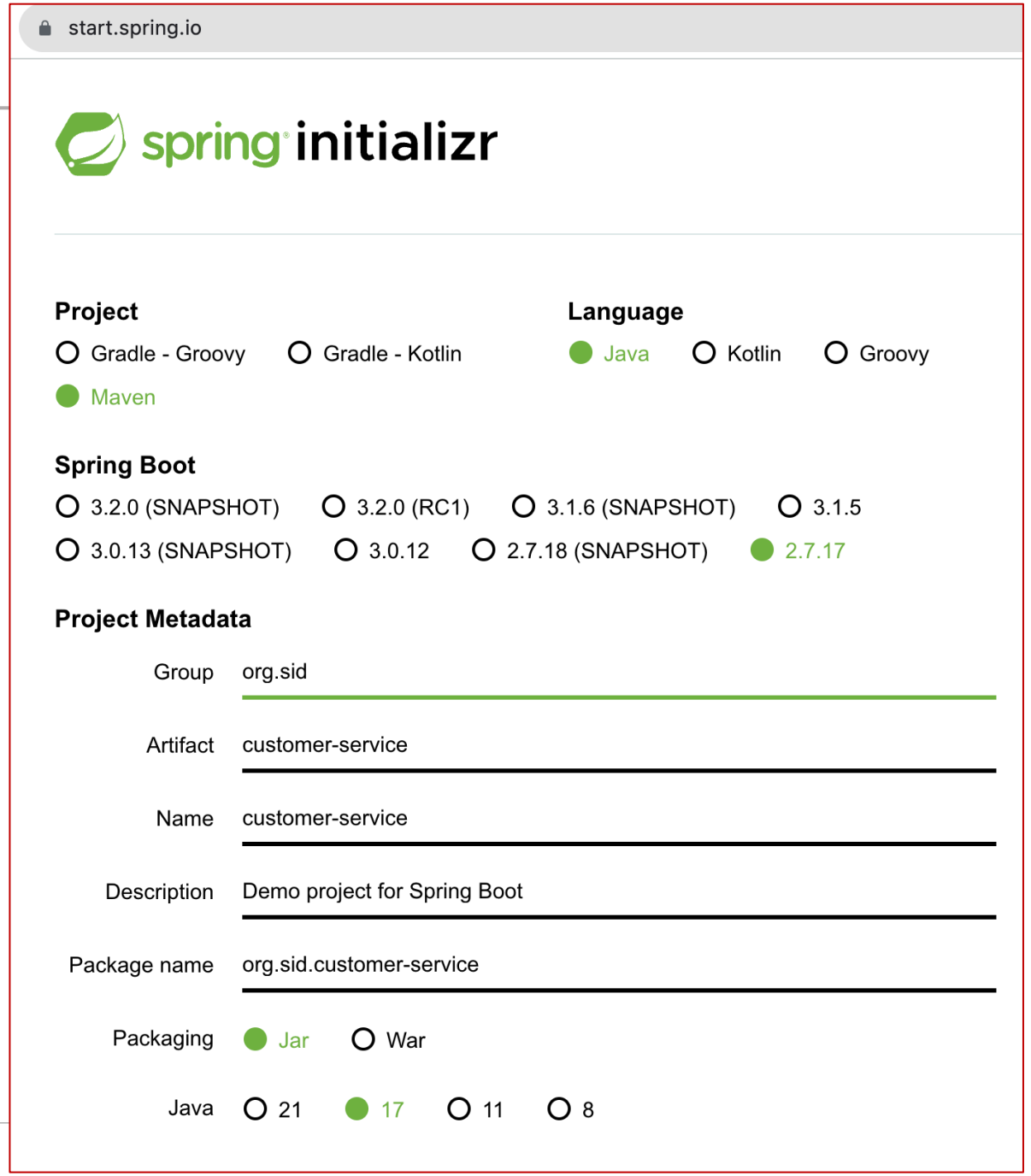
- Créer une application basée sur deux services métiers:
  - Service des clients
  - Service d'inventaire
- L'orchestration des services se fait via deux services techniques de Spring Cloud :
  - Spring Cloud Gateway Service comme service proxy
  - Registry Eureka Service comme annuaire d'enregistrement et de découverte des services de l'architecture



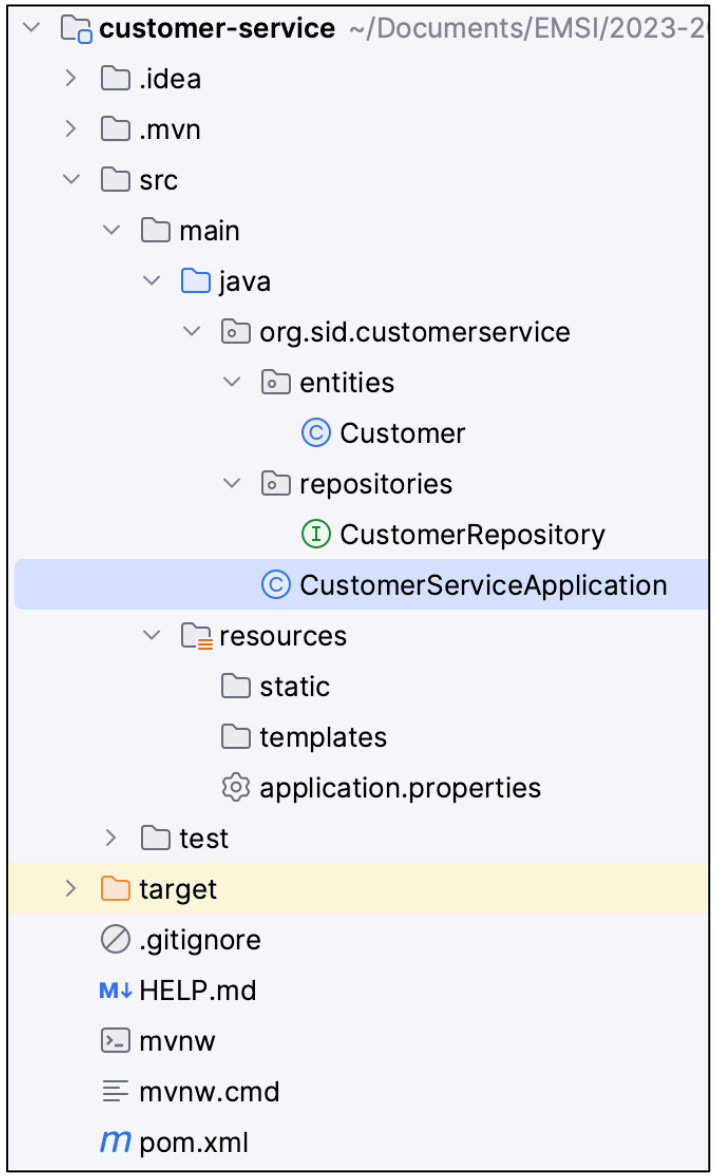
Application: customer-service

Selected dependencies:

- **Spring Web**: Build web, including RESTful, applications using Spring MC. Uses Apache Tomcat as the default embedded container.
- **Spring Data JPA**: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- **H2 Database** : Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
- **Rest Repositories**: Exposing Spring Data repositories over REST via Spring Data REST.
- **Lombok**: Java annotation library which helps to reduce boilerplate code.
- **Spring Boot DevTools**: Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- **Eureka Discovery Client**: a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.
- **Spring Boot Actuator**: Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.



Application: customer-service




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

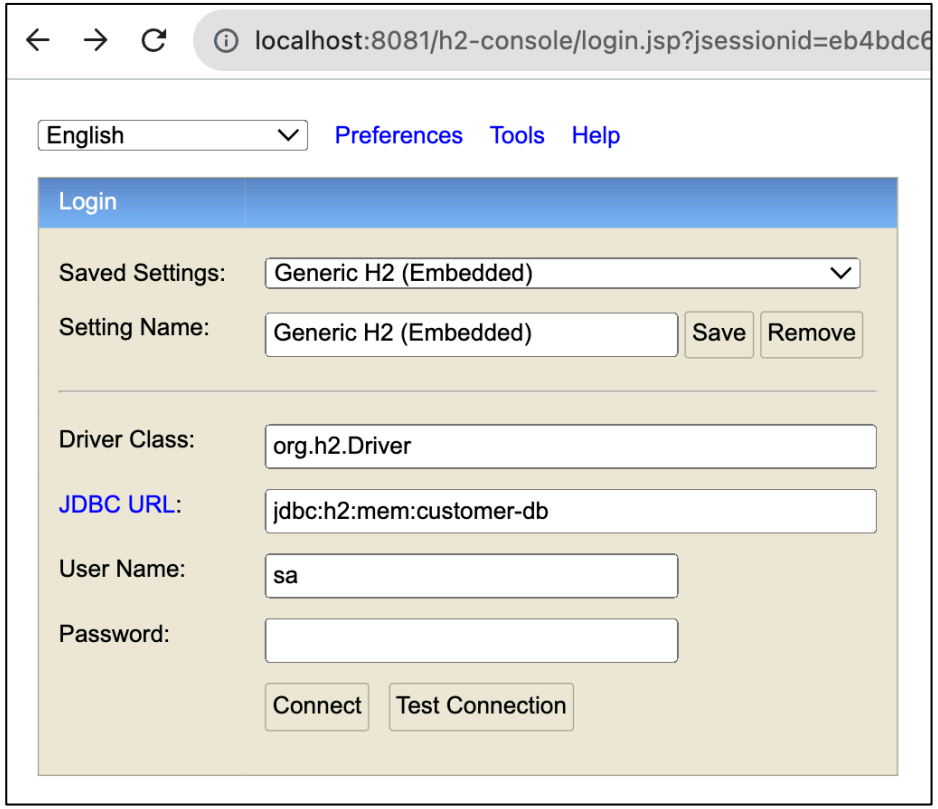
```
spring.datasource.url=jdbc:h2:mem:customer-db
spring.h2.console.enabled=true
server.port=8081
spring.cloud.discovery.enabled=false
spring.application.name=customer-service
#management. endpoints.web.exposure.include=*
```

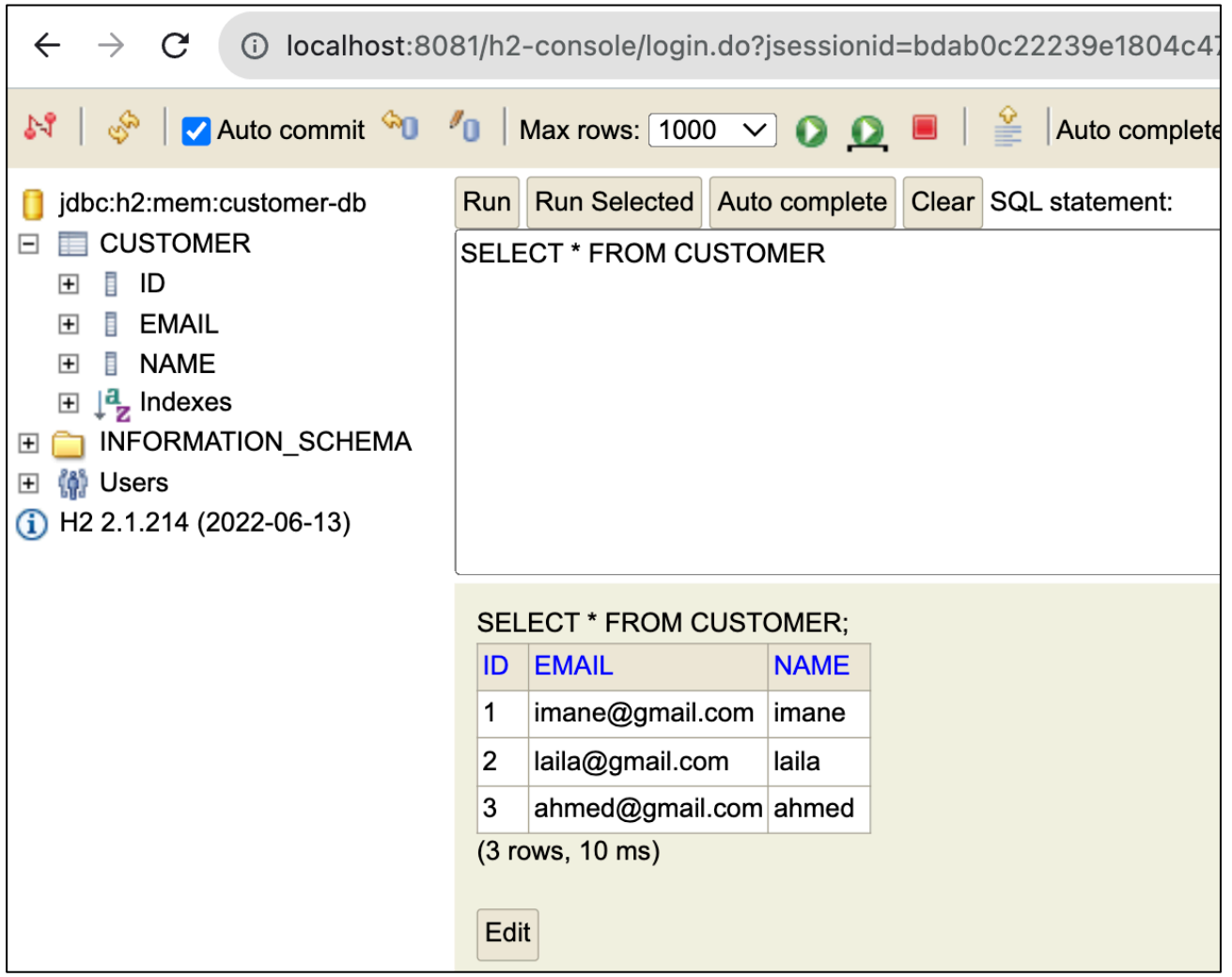
```
@RepositoryRestResource
public interface CustomerRepository extends JpaRepository<Customer,Long> {
}
```

```
@SpringBootApplication
public class CustomerServiceApplication {
    public static void main(String[] args) { SpringApplication.run(CustomerServiceApplication.class, args); }
    no usages
    @Bean
    CommandLineRunner start( CustomerRepository customerRepository){
        return args -> {
            customerRepository.save(new Customer( id: null, name: "imane", email: "imane@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "laila", email: "laila@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "ahmed", email: "ahmed@gmail.com"));

        };
    }
}
```

Application: customer-service

http://localhost:8081/h2-console/login.jsp





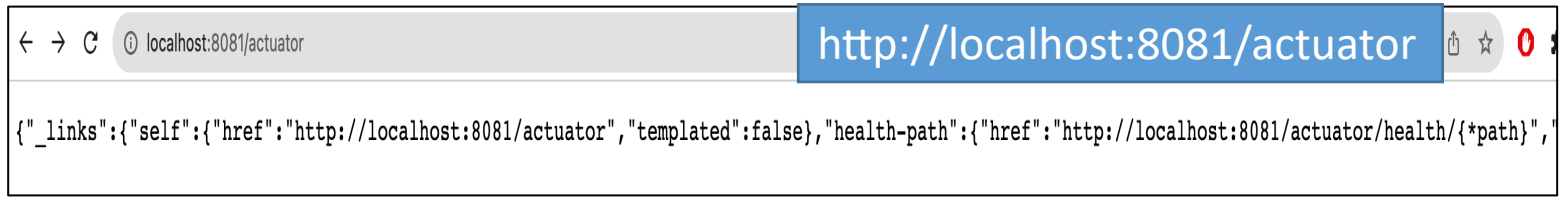
Application: customer-service

http://localhost:8081/customers


```
{
  "_embedded" : {
    "customers" : [ {
      "name" : "imane",
      "email" : "imane@gmail.com",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/customers/1"
        },
        "customer" : {
          "href" : "http://localhost:8081/customers/1"
        }
      }
    }, {
      "name" : "laila",
      "email" : "laila@gmail.com",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/customers/2"
        },
        "customer" : {
          "href" : "http://localhost:8081/customers/2"
        }
      }
    }, {
      "name" : "ahmed",
      "email" : "ahmed@gmail.com",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8081/customers/3"
        },
        "customer" : {
          "href" : "http://localhost:8081/customers/3"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8081/customers"
    },
    "profile" : {
      "href" : "http://localhost:8081/profile/customers"
```


http://localhost:8081/customers/1/

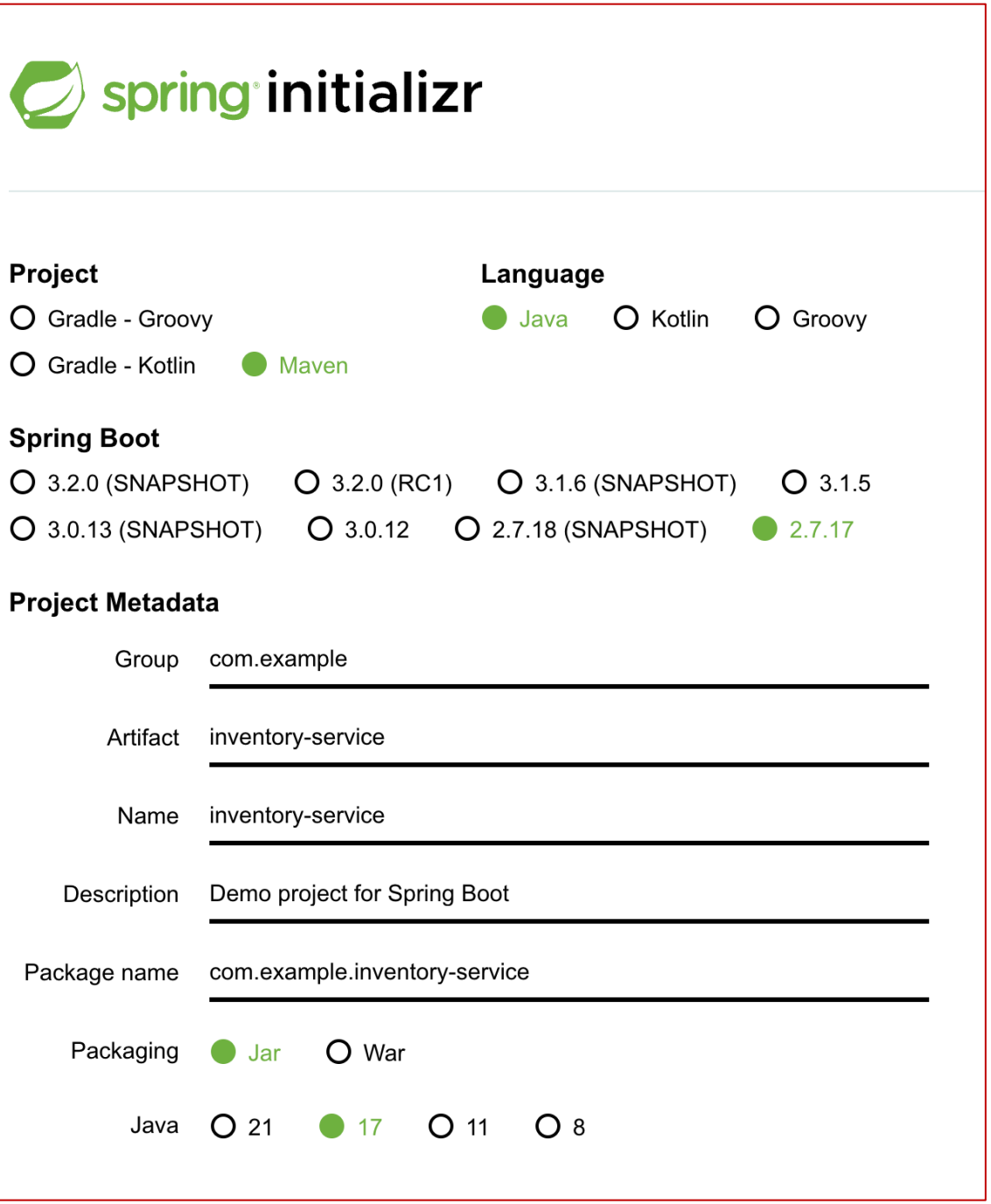

```
{
  "name" : "imane",
  "email" : "imane@gmail.com",
  "_links" : {
    "self" : {
      "href" : "http://localhost:8081/customers/1"
    },
    "customer" : {
      "href" : "http://localhost:8081/customers/1"
    }
  }
}
```

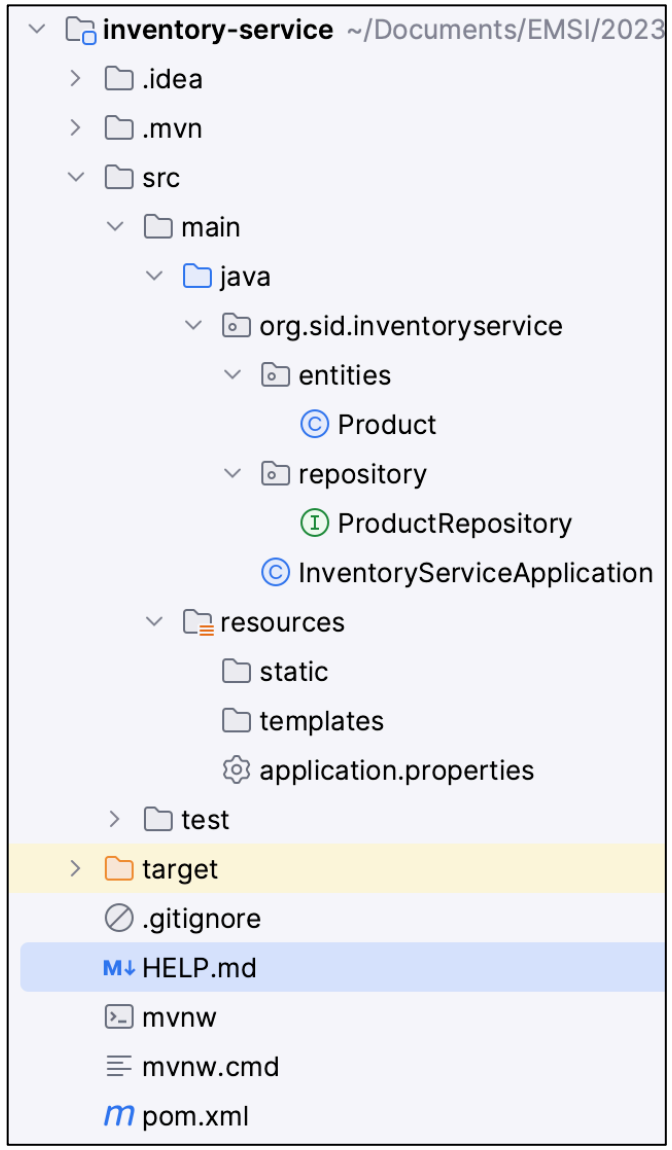
Application: inventory-service

Selected dependencies:

- **Spring Web**: Build web, including RESTful, applications using Spring MC. Uses Apache Tomcat as the default embedded container.
- **Spring Data JPA**: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- **H2 Database** : Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
- **Rest Repositories**: Exposing Spring Data repositories over REST via Spring Data REST.
- **Lombok**: Java annotation library which helps to reduce boilerplate code.
- **Spring Boot DevTools**: Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- **Eureka Discovery Client**: a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.
- **Spring Boot Actuator**: Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.



Application: inventory-service




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

```
spring.datasource.url=jdbc:h2:mem:inventory-db
spring.h2.console.enabled=true
server.port=8082
spring.cloud.discovery.enabled=false
spring.application.name=inventory-service
#management. endpoints.web.exposure.include=*
```

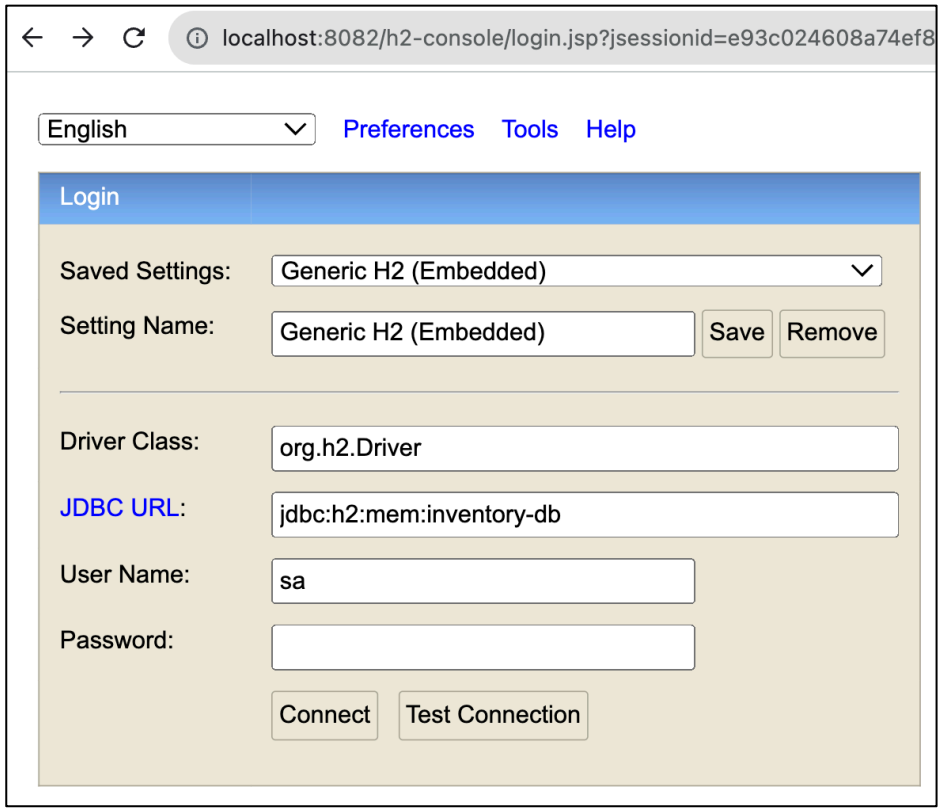
Ajouter tous les endpoints de actuator ou bien une fct particuliere

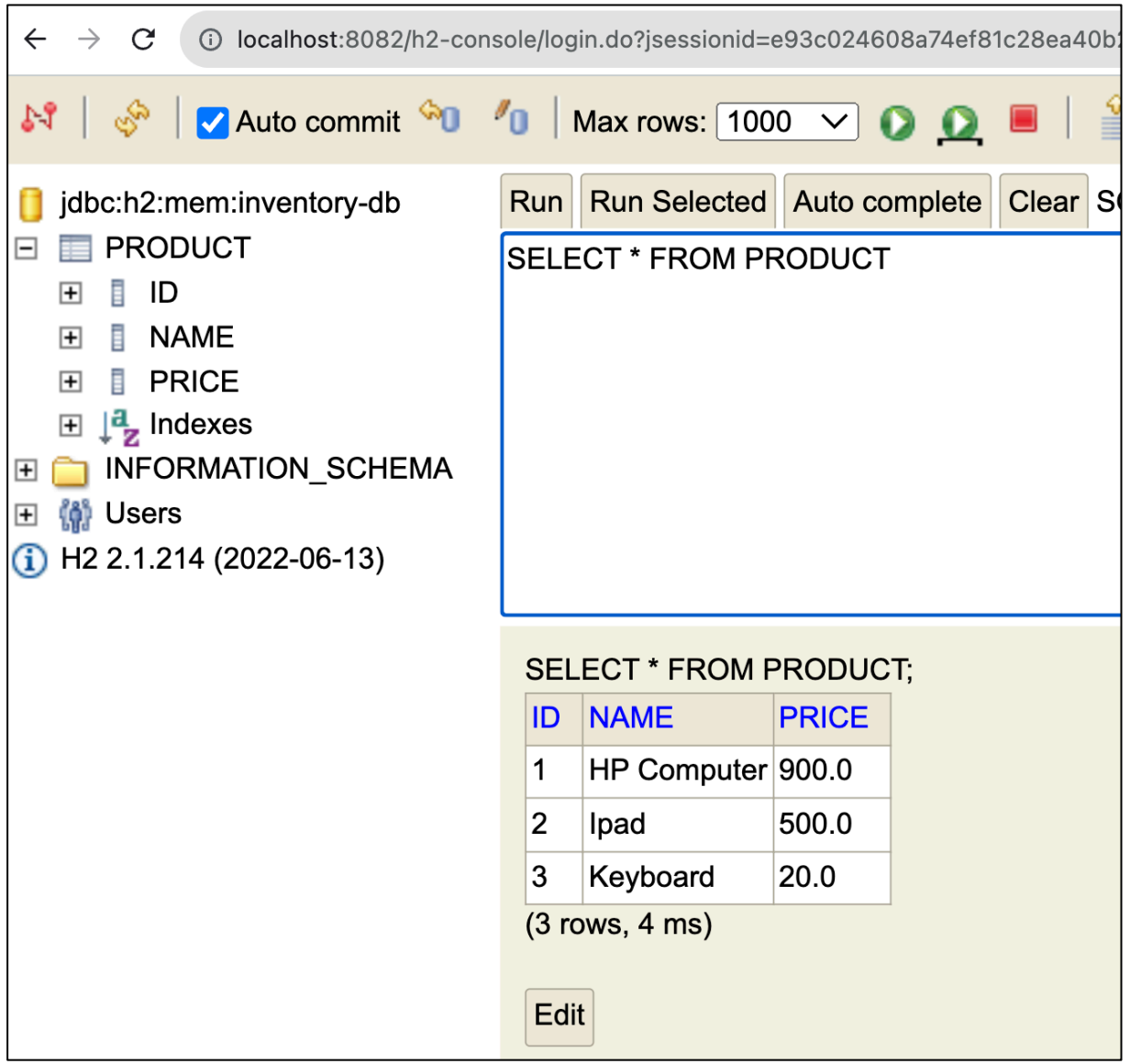
```java
2 usages
public interface ProductRepository extends JpaRepository<Product,Long> {
}
```

```java
@SpringBootApplication
public class InventoryServiceApplication {

    public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
    no usages
    @Bean
    CommandLineRunner start(ProductRepository productRepository){
        return args -> {
            productRepository.save(new Product( id: null, name: "HP Computer", price: 900));
            productRepository.save(new Product( id: null, name: "Ipad", price: 500));
            productRepository.save(new Product( id: null, name: "Keyboard", price: 20));

        };
    }
}
```

Application: inventory-service

http://localhost:8082/h2-console/login.jsp





Application: inventory-service

http://localhost:8082/products


```
{
  "_embedded" : {
    "products" : [ {
      "name" : "HP Computer",
      "price" : 900.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/1"
        },
        "product" : {
          "href" : "http://localhost:8082/products/1"
        }
      }
    }, {
      "name" : "Ipad",
      "price" : 500.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/2"
        },
        "product" : {
          "href" : "http://localhost:8082/products/2"
        }
      }
    }, {
      "name" : "Keyboard",
      "price" : 20.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/3"
        },
        "product" : {
          "href" : "http://localhost:8082/products/3"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8082/products"
    },
```

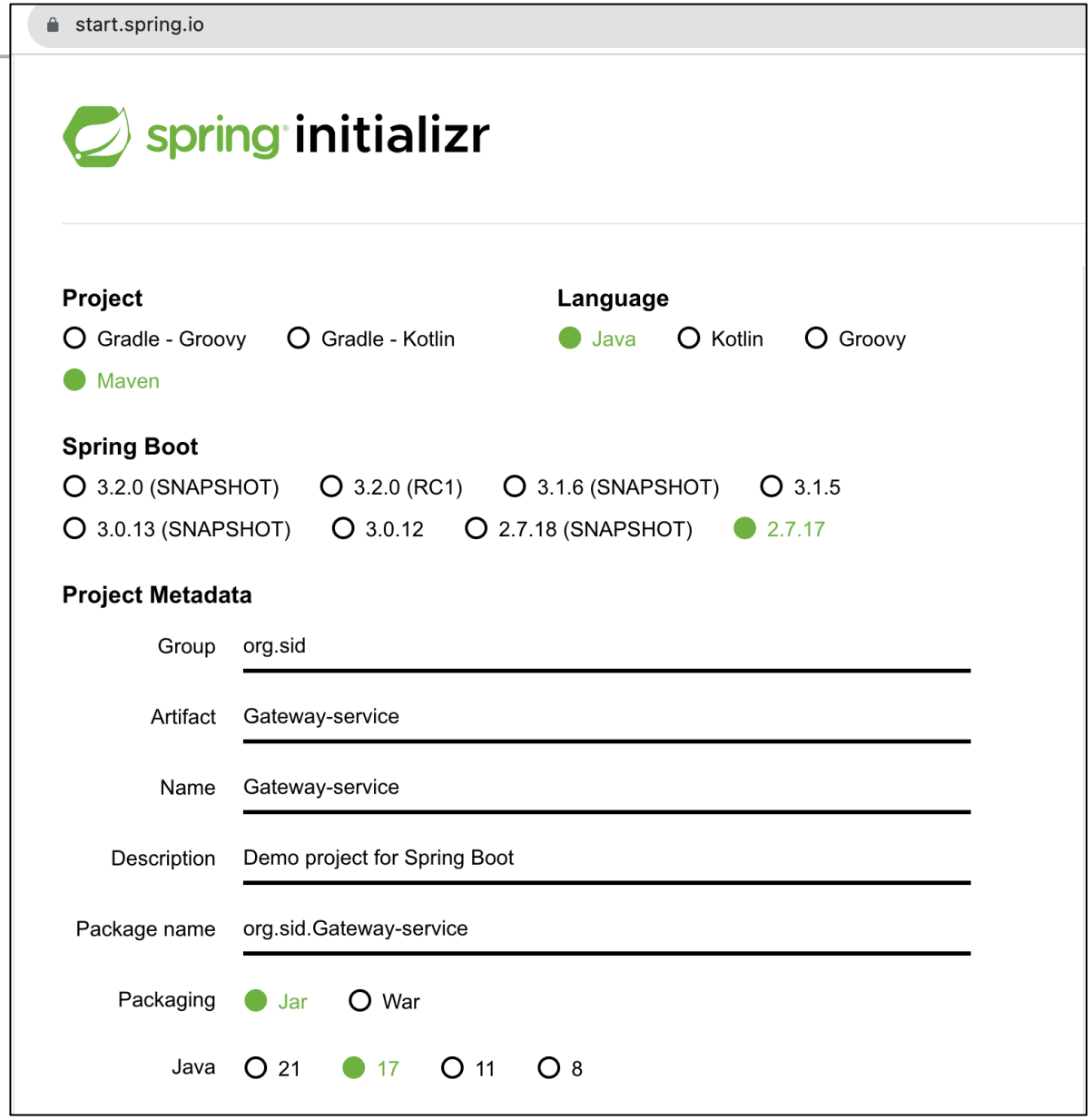

http://localhost:8082/products/1/


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


Application: Gateway-service

Selected dependencies:

- **Gateway**: Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.
- **Spring Boot Actuator** : Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.
- **Eureka Discovery Client** : a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

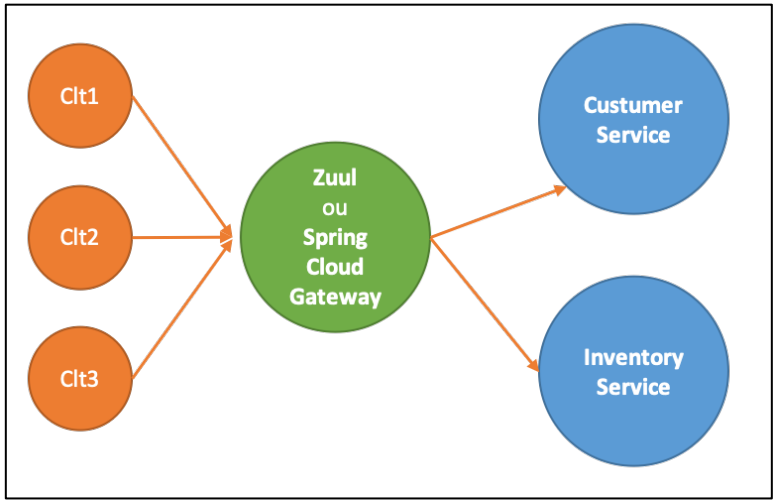


Application: Gateway-service

Configuration statique on utilisant:

Application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: r1
          uri: http://localhost:8081/
          predicates:
            - Path=/customers/**
        - id: r2
          uri: http://localhost:8082/
          predicates:
            - Path=/products/**
    discovery:
      enabled: false
server:
  port: 8888
```



localhost:8888/products/1


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


localhost:8888/products


```
{
  "_embedded" : {
    "products" : [ {
      "name" : "HP Computer",
      "price" : 900.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/1"
        },
        "product" : {
          "href" : "http://localhost:8082/products/1"
        }
      }
    }, {
      "name" : "Ipad",
      "price" : 500.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/2"
        },
        "product" : {
          "href" : "http://localhost:8082/products/2"
        }
      }
    }, {
      "name" : "Keyboard",
      "price" : 20.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/3"
        },
        "product" : {
          "href" : "http://localhost:8082/products/3"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8082/products"
    },
```

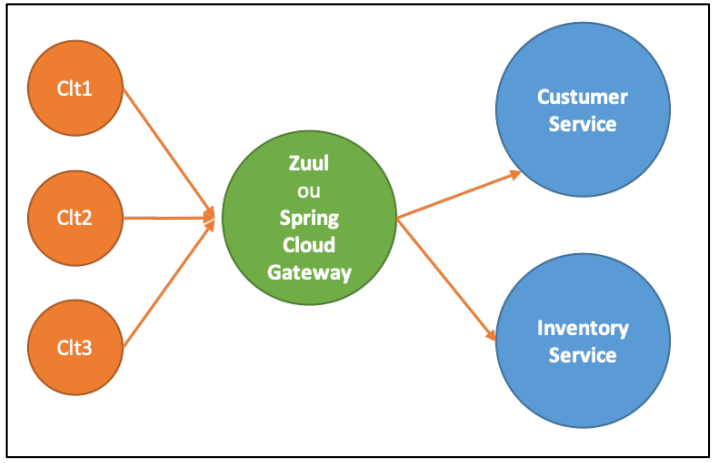
Application: Gateway-service

Configuration statique par programme:

```java
@SpringBootApplication
public class GatewayServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayServiceApplication.class, args);
    }
    no usages
    @Bean
    RouteLocator gatewayRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(r->r.path( ...patterns: "/customers/**").uri("http://localhost:8081/"))
                .route(r->r.path( ...patterns: "/products/**").uri("http://localhost:8082/"))
                .build();
    }
}
```

localhost:8888/products/1


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

localhost:8888/products


```
{
  "_embedded" : {
    "products" : [ {
      "name" : "HP Computer",
      "price" : 900.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/1"
        },
        "product" : {
          "href" : "http://localhost:8082/products/1"
        }
      }
    }, {
      "name" : "Ipad",
      "price" : 500.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/2"
        },
        "product" : {
          "href" : "http://localhost:8082/products/2"
        }
      }
    }, {
      "name" : "Keyboard",
      "price" : 20.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8082/products/3"
        },
        "product" : {
          "href" : "http://localhost:8082/products/3"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8082/products"
    },
```

Application: Eureka Discovery Service: Dynamic Routing

start.spring.io

spring initializr

Project
- ○ Gradle - Groovy
- ○ Gradle - Kotlin
- ● Maven

Language
- ● Java
- ○ Kotlin
- ○ Groovy

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	org.sid
Artifact	Discovery-service
Name	Discovery-service
Description	Demo project for Spring Boot
Package name	org.sid.Discovery-service
Packaging	● Jar ○ War
Java	○ 21 ● 17 ○ 11 ○ 8

Selected dependencies:

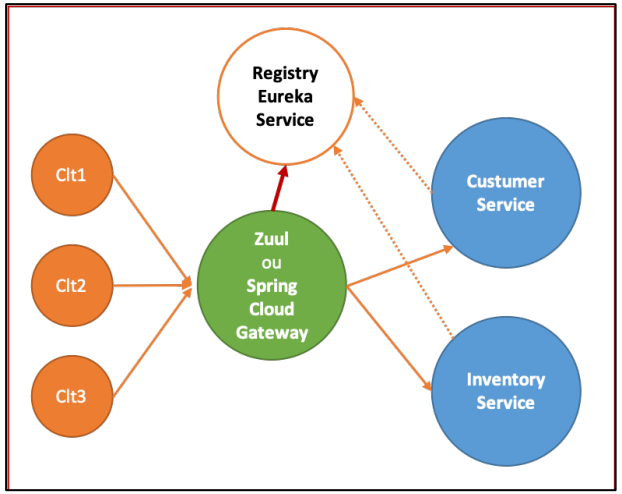
- **Eureka Server** : spring-cloud-netflix Eureka Server.

application.properties

Ne pas cacher les clients locaux

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

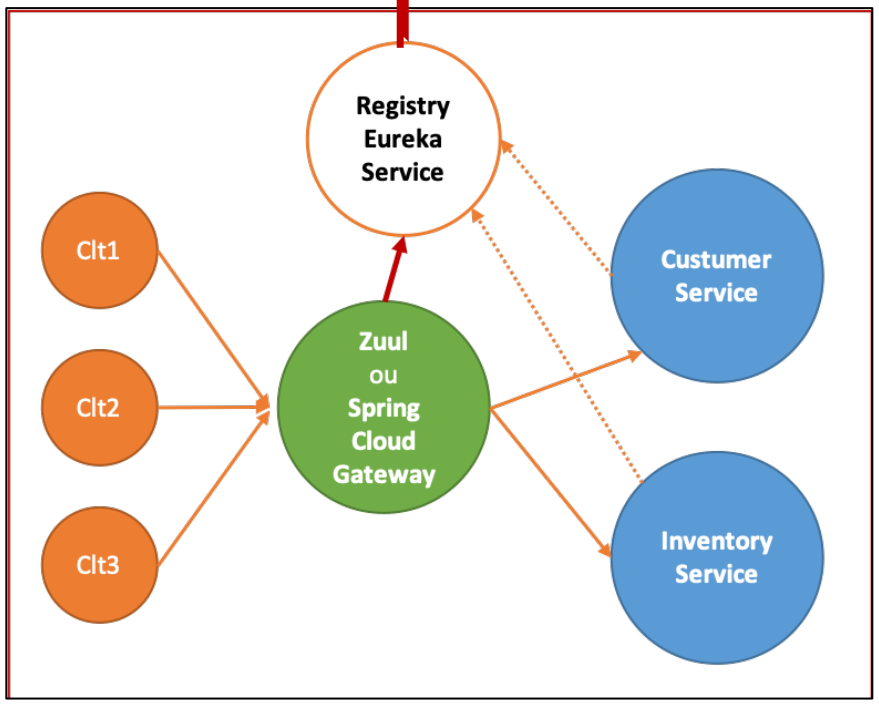
Ne pas s'auto-enregistrer




```
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {

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

Application: Eureka Discovery Service: Dynamic Routing

http://localhost:8761



localhost:8761 — Update

spring Eureka — HOME — LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2023-10-21T19:18:08 +0100
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	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Instance Info

Application: Eureka Discovery Service: Dynamic Routing

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

Customer-service

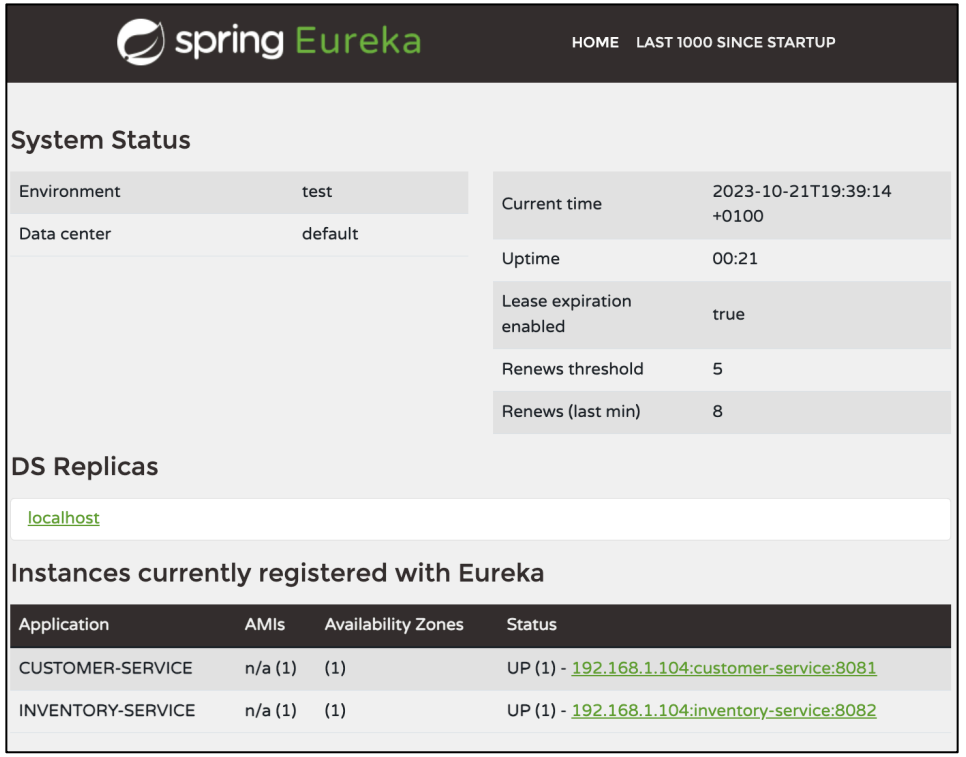
application.properties

```
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
```

Inventory-service

application.properties

```
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: Eureka Discovery Service: Dynamic Routing

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

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: Eureka Discovery Service: Dynamic Routing

Exemple de : Routes Filters

GatewayServiceApplication.java

```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"
    }
  }
}
```