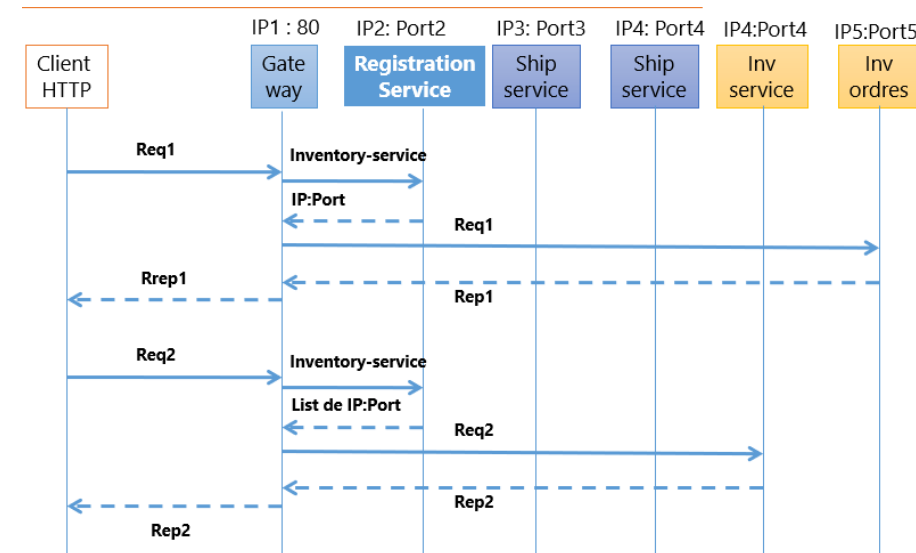
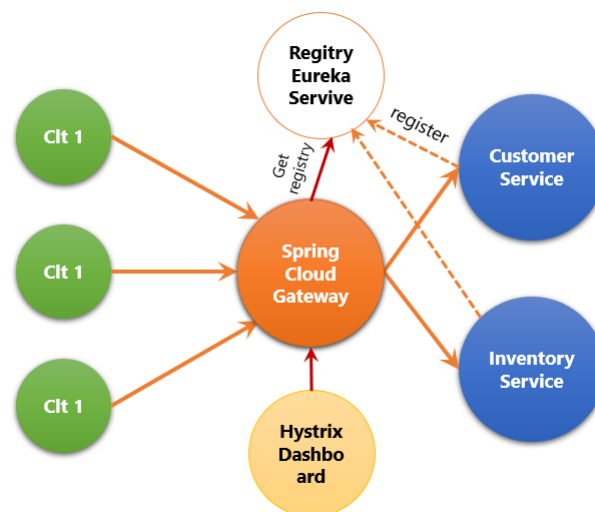
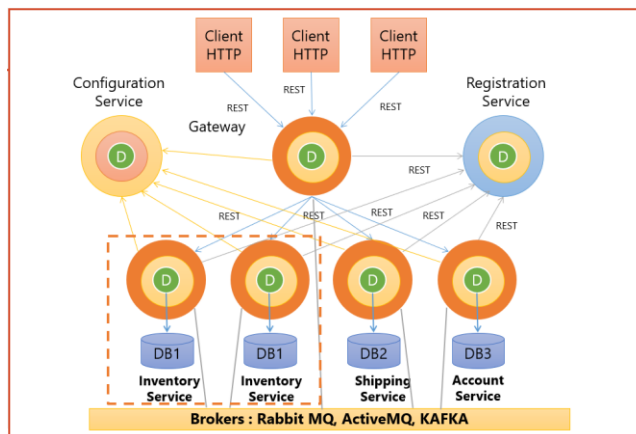


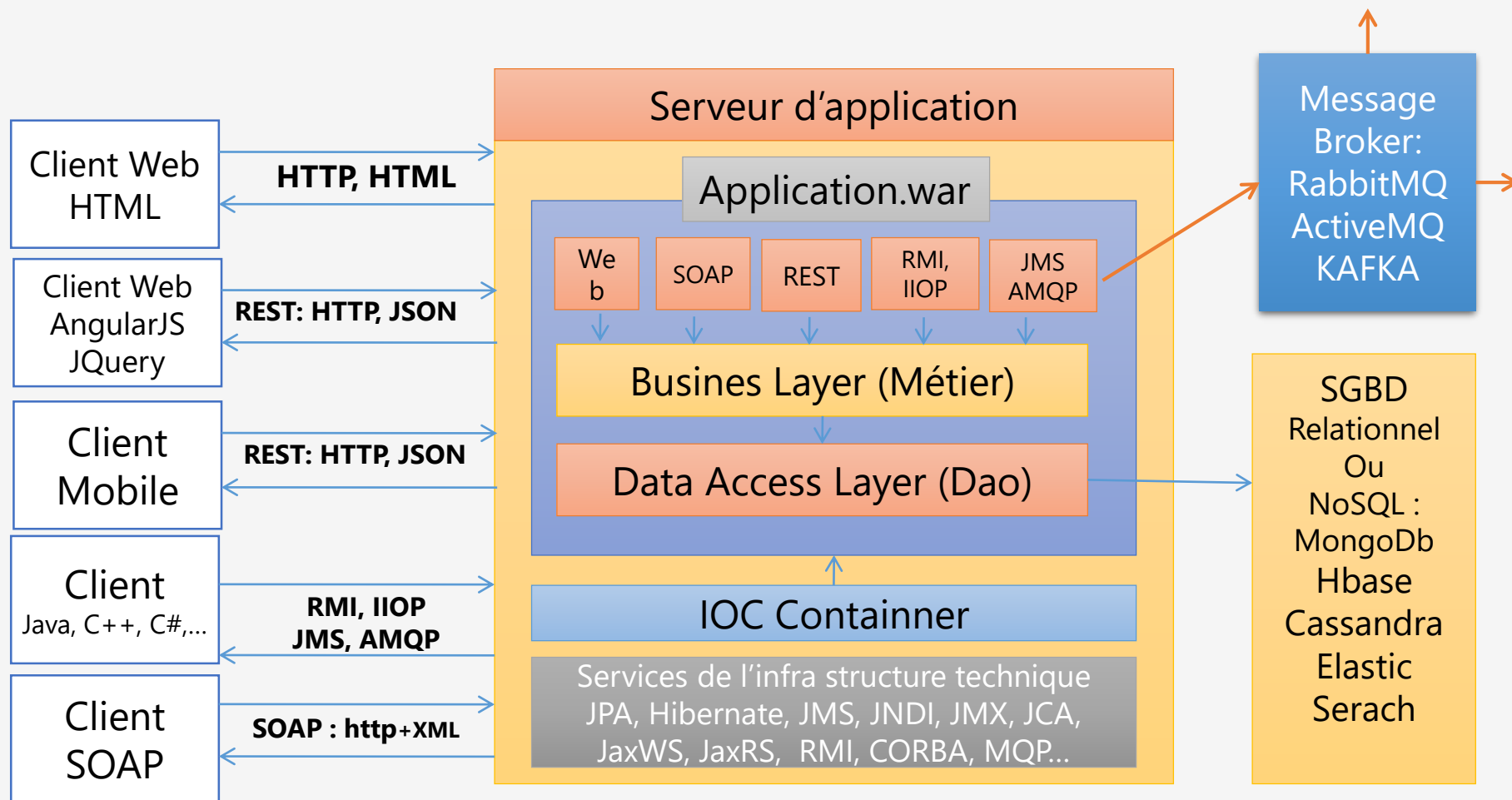
Systemes Distribués basés sur les Micro services



Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@yousfi.net
Supports de cours : <http://fr.slideshare.net/mohamedyousfi9>
Chaîne vidéo : <http://youtube.com/mohamedYoussfi>
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

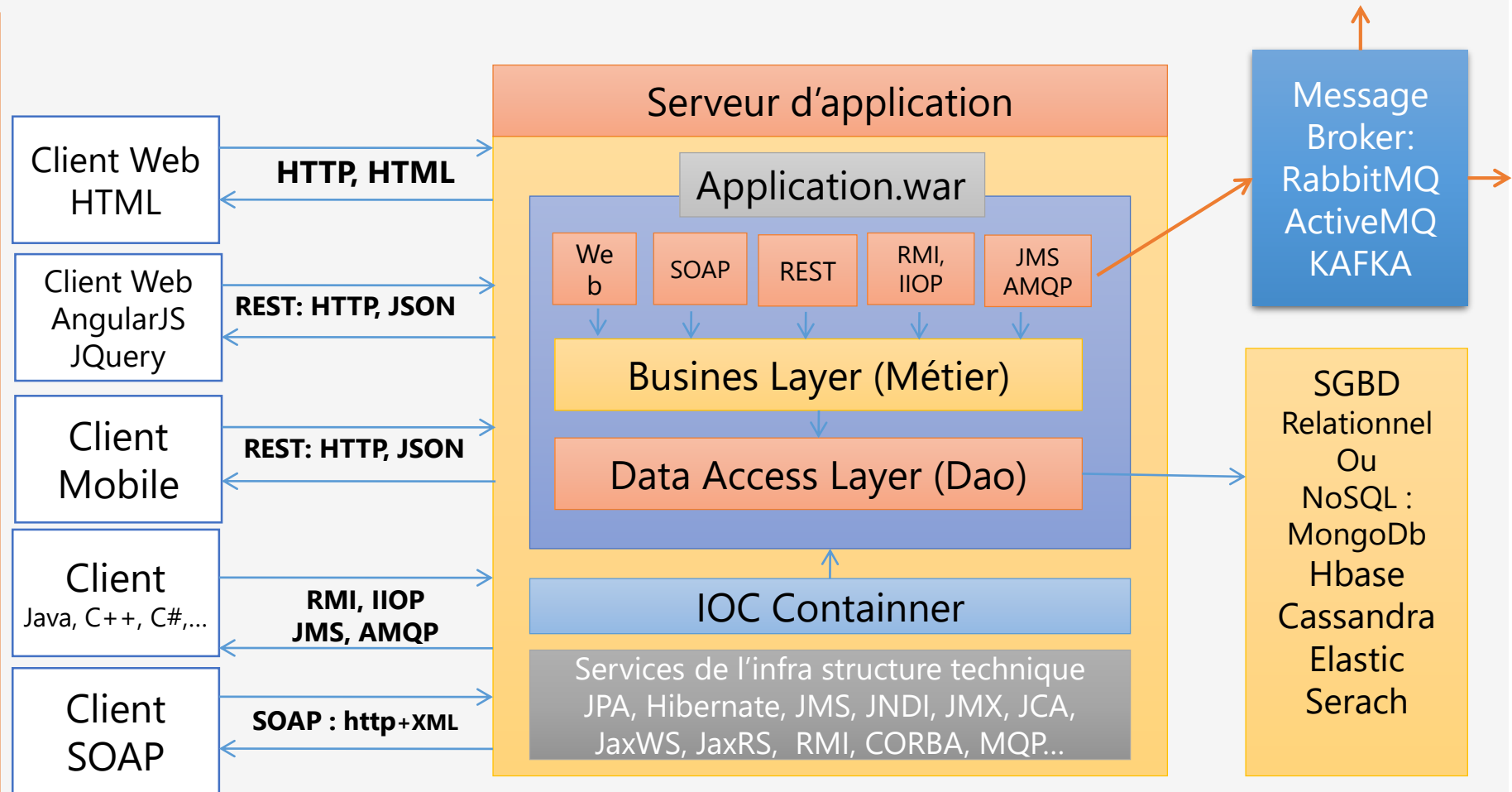
Approche Monolithique Vers Approche Micro-services

Une application monolithique est une application qui est développée en un seul bloc (war, jar, Ear, dll), avec une même technologie et déployée dans un serveur d'application



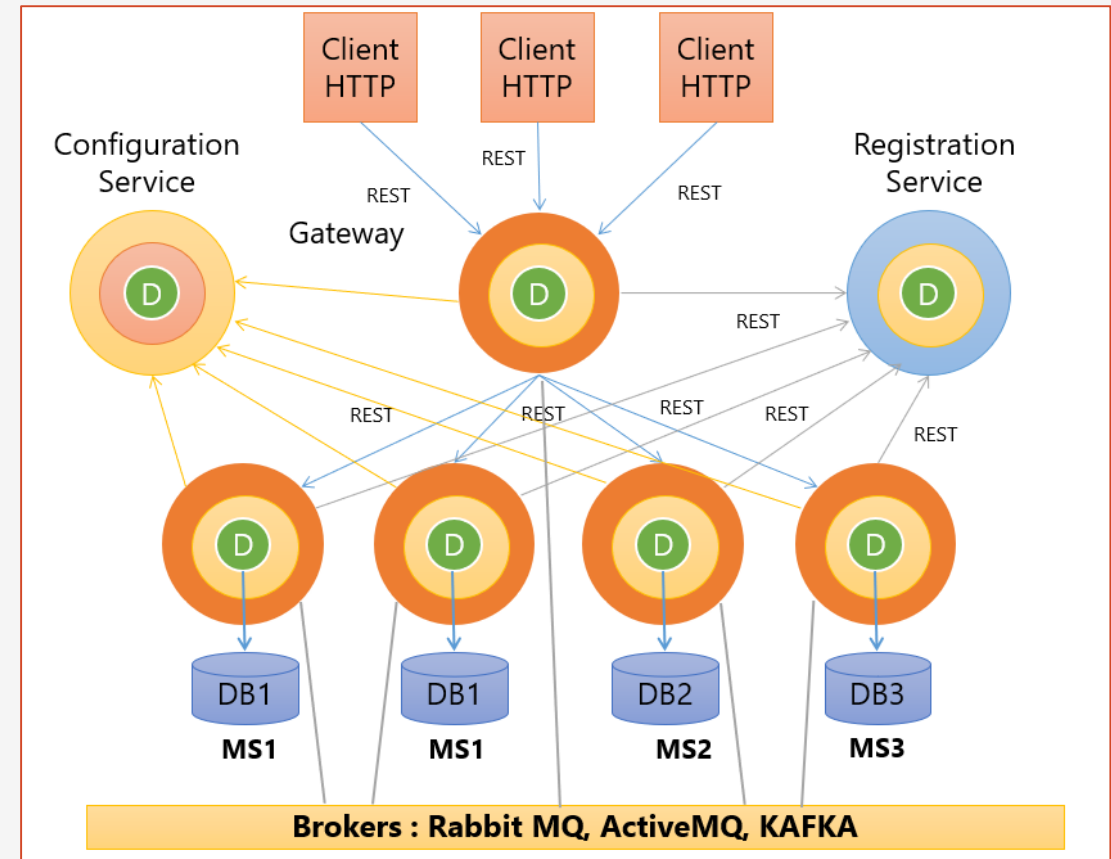
Principaux Problèmes d'une approche monolithique

- Elles centralisent tous les besoins fonctionnels
- Elles sont réalisées dans une seule technologie.
- Chaque modification nécessite de :
- Tester les régressions
- Redéployer toute l'application
- Difficile à faire évoluer au niveau fonctionnel
- Livraison en bloc (Le client attend beaucoup de temps pour commencer à voir les premières versions)



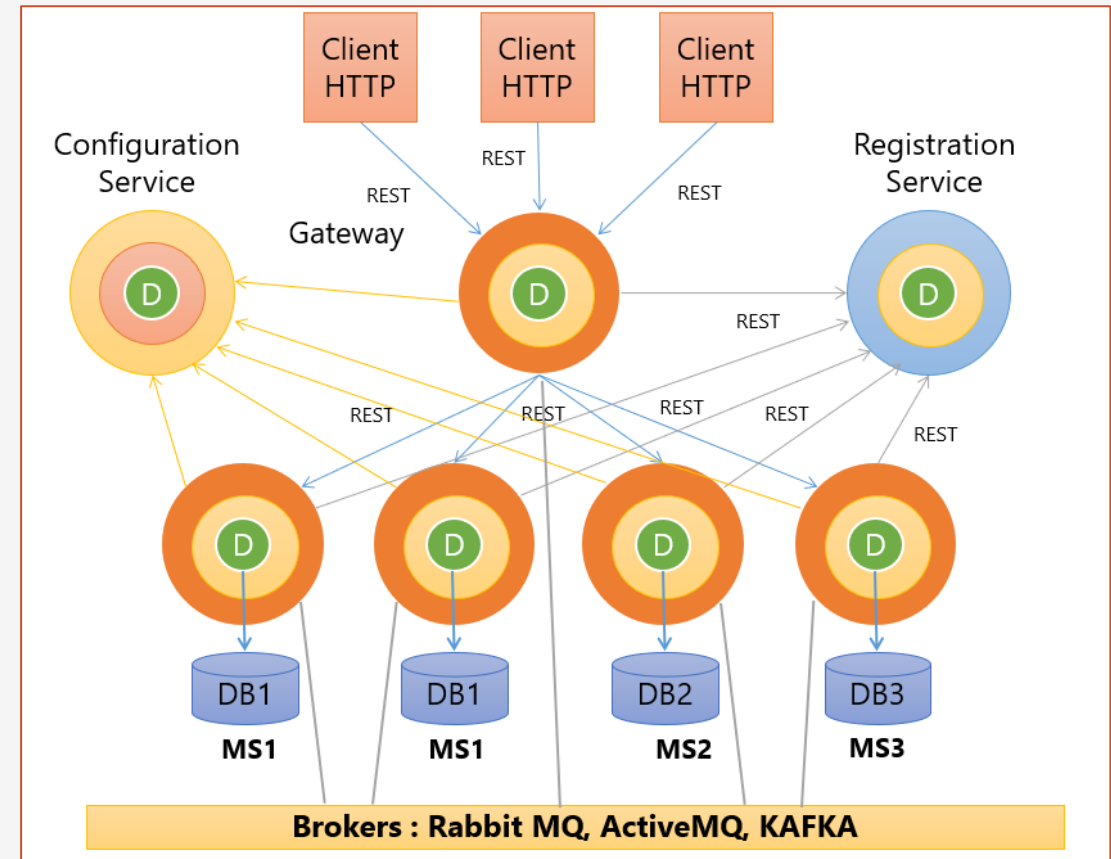
Approche Micro services

- Les micro services sont une approche d'architecture et de développement d'une **application composées de petits services**.
- L'idée étant de découper un grand problème en petites unités implémentée sous forme de micro-services
- Chaque service est **responsable d'une fonctionnalité**,
- Chaque micro-service est **développé, testé et déployé séparément** des autres.
- Chaque micro service **est développé en utilisant une technologie qui peut être différente des autres**. (Java, C++, C#, PHP, NodeJS, Python, ...)
- Chaque service **tourne dans un processus séparé**.
- Utilisant des mécanismes de communication légers (REST)
- La seule relation entre les différents micro services est l'échange de données effectué à travers les différentes APIs qu'ils exposent. (**SOAP, REST, RMI, CORBA, JMS, MQP, ...**)
- Lorsqu'on **les combinent**, ces micro services peuvent **réaliser des opérations très complexes**.



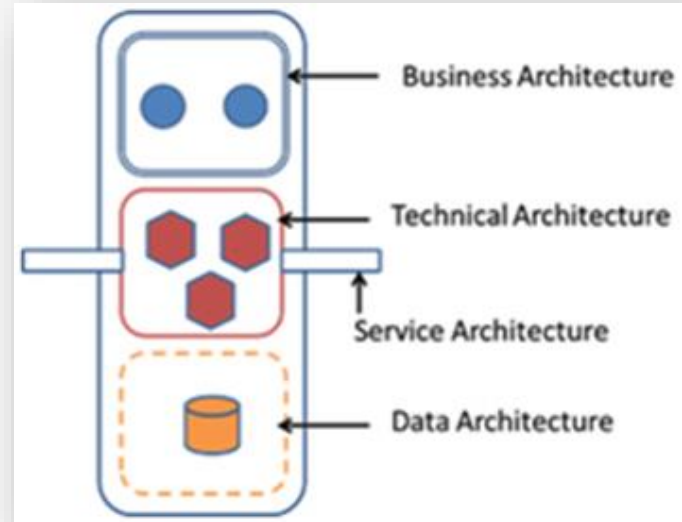
Approche Micro services

- Ils sont **faiblement couplés** puisque chaque micro service est physiquement séparé des autres,
- **Indépendance relative entre les différentes équipes** qui développent les différents micro services.
- **Facilité des tests** et du **déploiement**
- **Livraison continue.**
- S'apprête bien à au processus du GL : **TDD** (Test Driver Développement) et **les méthodes agiles**



Approche Micro services

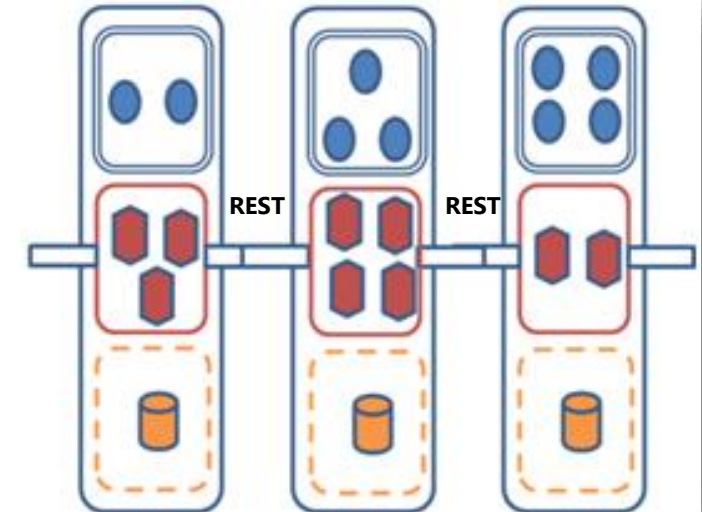
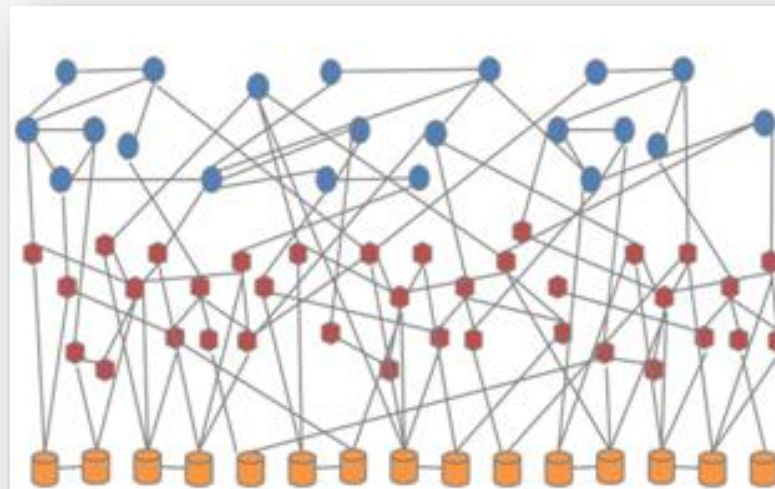
- Comme pour le cas d'une application monolithique, un micro service peut être composé de plusieurs très petites couches:
 - Couche DAO
 - Couche Métier,
 - Couches Techniques (REST, SOAP, RMI, JMS, AMQP, Sécurité, etc...)

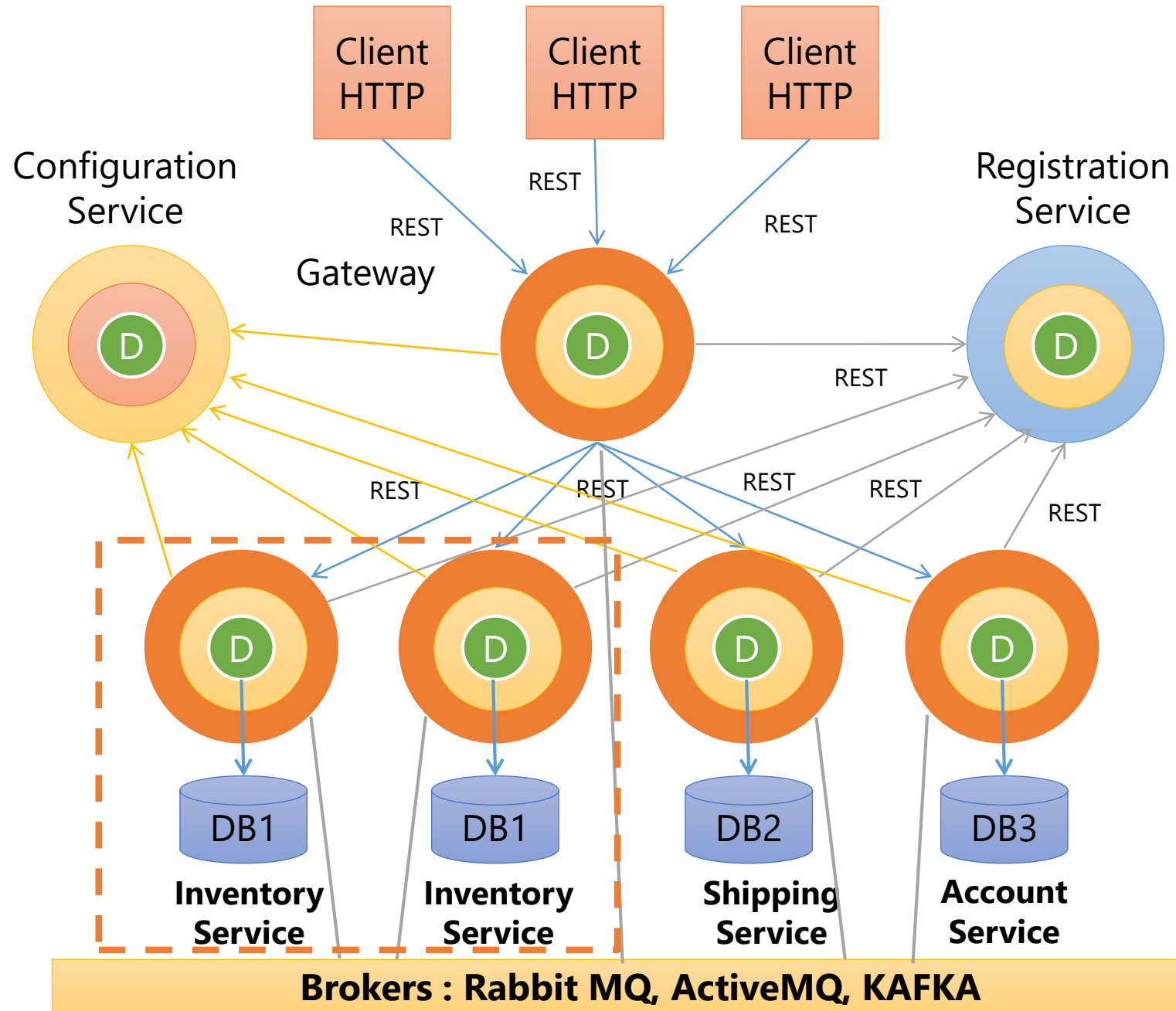


Client
Service

Compte
Service

Opération
Service

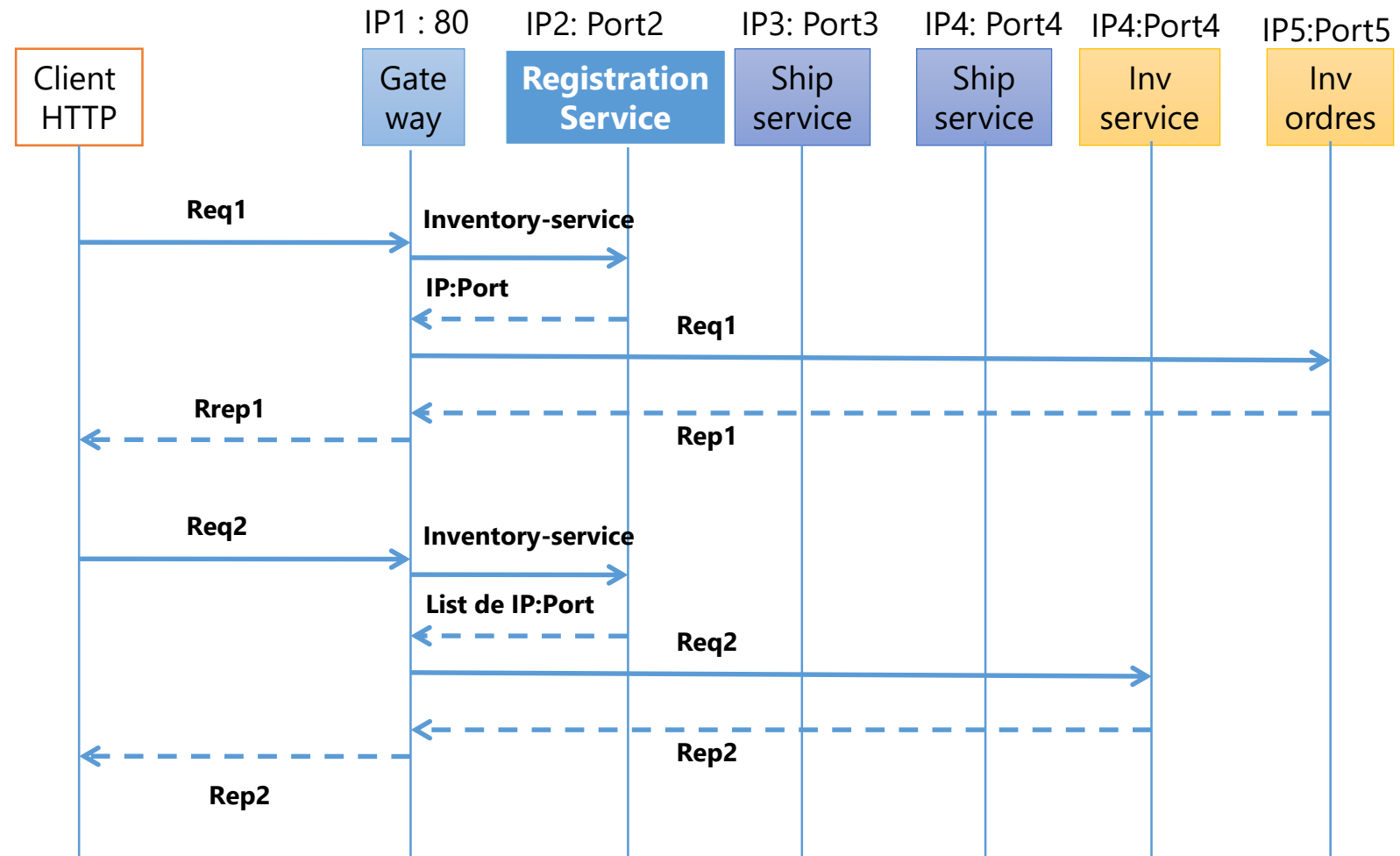




Consulter les services via le service proxy

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

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



Spring Boot

Spring Boot est un Micro Framework qui permet de créer des applications basées sur des micro services.

Atouts de Spring Boot :

- Faciliter le développement d'applications complexes.
- Faciliter à l'extrême l'injection des dépendances
- Réduire à l'extrême les fichiers de configurations
- Faciliter la gestion des dépendances Maven.
- Auto Configuration : la plupart des beans sont créés si le ou les jar(s) adéquats sont dans le classpath.
- Fournir un conteneur de servlet embarqué (Tomcat, Jetty)
- Créer une application autonome (jar ou war)



EXEMPLE DE MICRO SERVICE AVEC SPRING BOOT

<https://www.youtube.com/watch?v=zBLXWIhrg7U>

Premier Exemple d'application

On souhaite créer une application qui permet de gérer des produits.

Chaque produit est défini par :

- Sa référence de type Long
- Sa désignation de type String
- Son prix

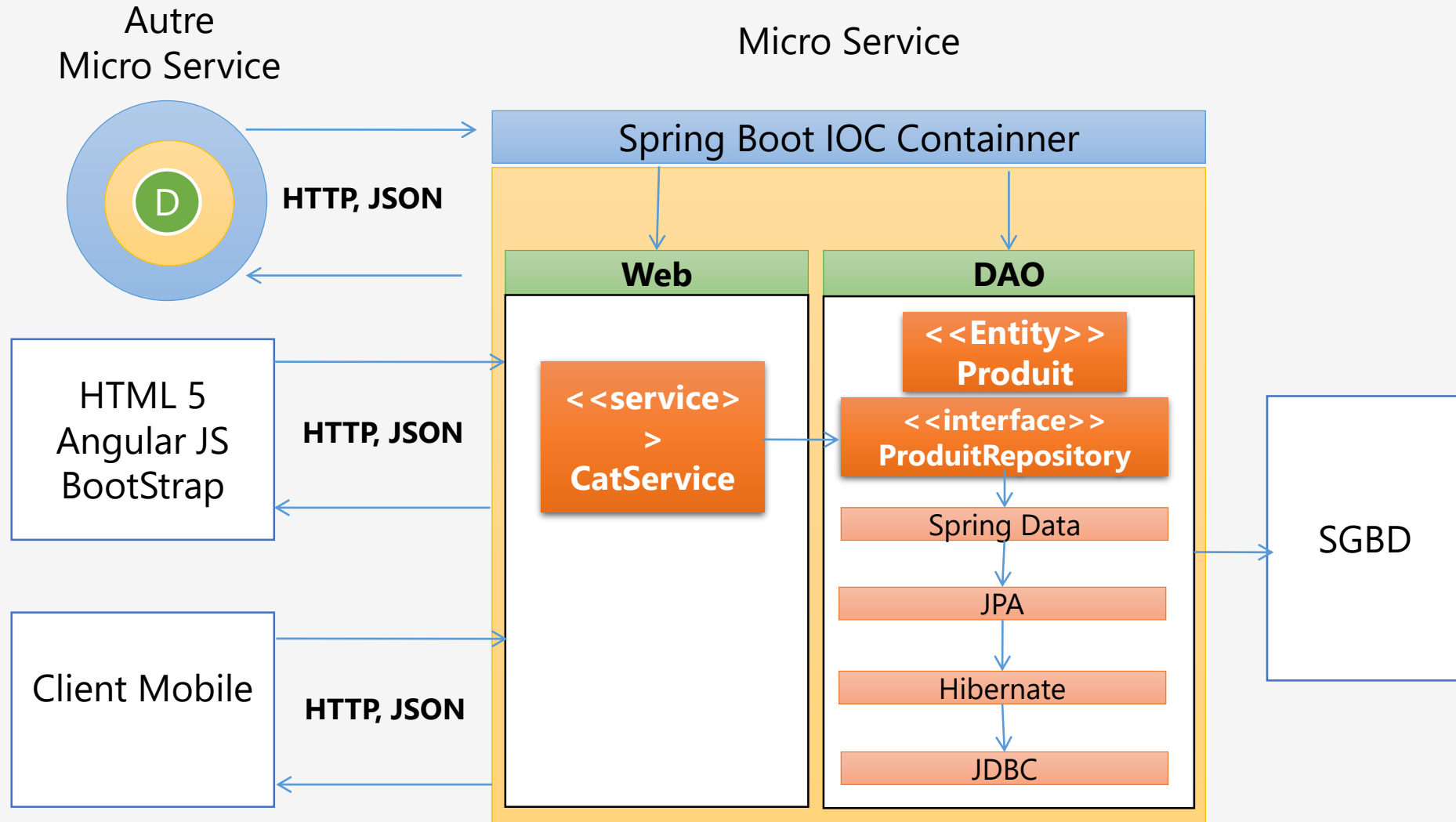
L'application de permettre de :

- Ajouter des produits
- Chercher les produits par mot clé

Les données sont stockées dans une base de données MySQL

L'application est un micro service Restful basée sur Spring Boot

Architecture



```

@RestController
public class ProduitRestService {
    @Autowired
    private ProduitRepository produitRepository;
    @RequestMapping(value="/produits",method=RequestMethod.GET)
    public List<Produit> produits(){
        return produitRepository.findAll();
    }
    @RequestMapping(value="/produits",method=RequestMethod.POST)
    public Produit save(@RequestBody Produit p){
        return produitRepository.save(p);
    }
}

```

Entité produit

```

@Entity
@Data
public class Produit
implements Serializable {
    @Id @GeneratedValue
    private Long id;
    private String designation;
    private double prix;
}

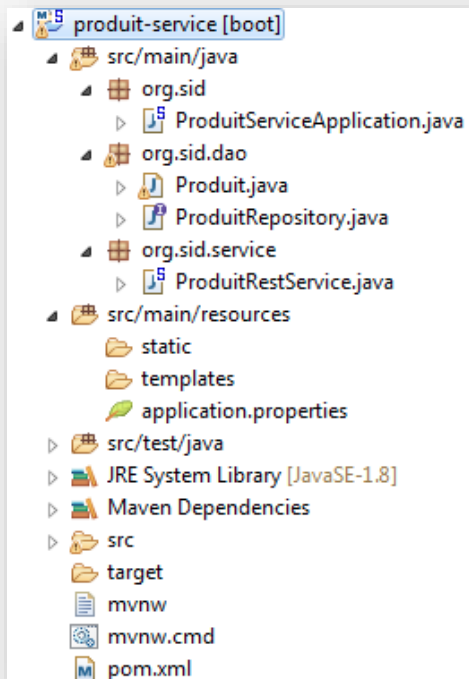
```

Interface DAO basée sur Spring data

```

public interface ProduitRepository extends JpaRepository<Produit, Long> {
}

```



@SpringBootApplication

```

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

```

Application Spring Boot

```

spring.datasource.url = jdbc:mysql://localhost:3306/prod-services
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

```

Application.properties

Service Restful

Micro Service : Spring Boot

@RestController

```
public class ProduitRestService {  
    @Autowired  
    private ProduitRepository produitRepository;  
    @RequestMapping(value="/produits",method=RequestMethod.GET)  
    public List<Produit> produits(){  
        return produitRepository.findAll();  
    }  
    @RequestMapping(value="/produits",method=RequestMethod.POST)  
    public Produit save(@RequestBody Produit p){  
        return produitRepository.save(p);  
    }  
}
```

Entité produit

@Entity

```
public class Produit  
implements Serializable {  
    @Id @GeneratedValue  
    private Long id;  
    private String designation;  
    private double prix;  
  
    // Getters et Setters  
}
```

Interface DAO basée sur Spring data

@RepositoryRestResource

```
public interface ProduitRepository extends JpaRepository<Produit, Long> {  
}
```

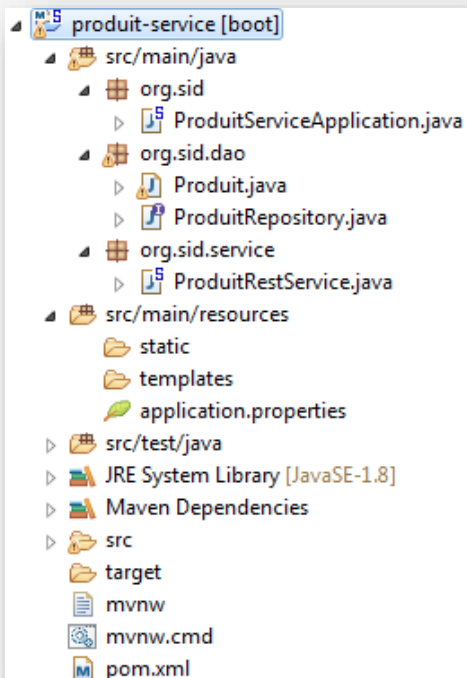
@SpringBootApplication

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

Application Spring Boot

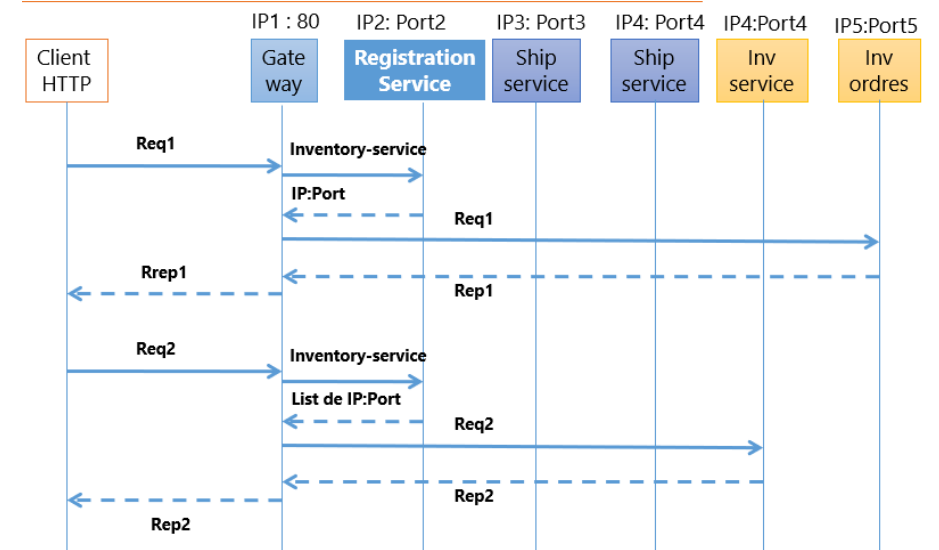
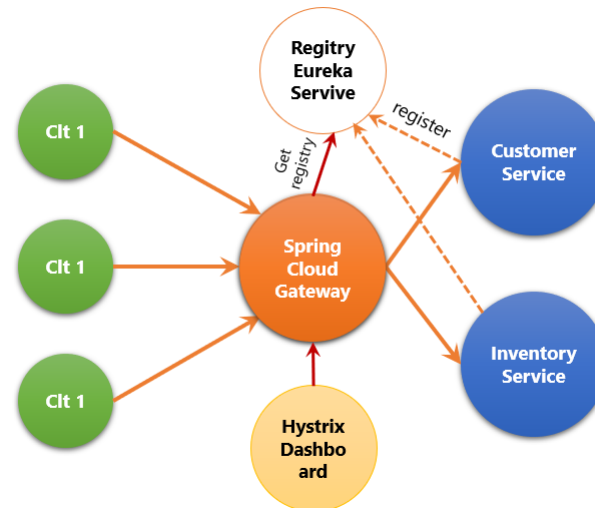
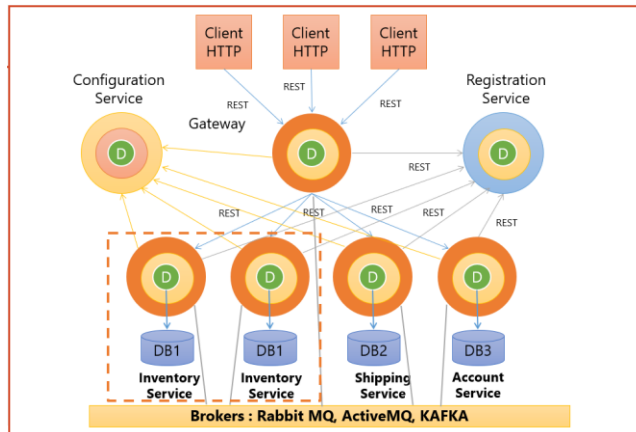
```
spring.datasource.url = jdbc:mysql://localhost:3306/prod-services  
spring.datasource.username = root  
spring.datasource.password =  
spring.datasource.driverClassName = com.mysql.jdbc.Driver  
spring.jpa.hibernate.ddl-auto = update  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Application.properties

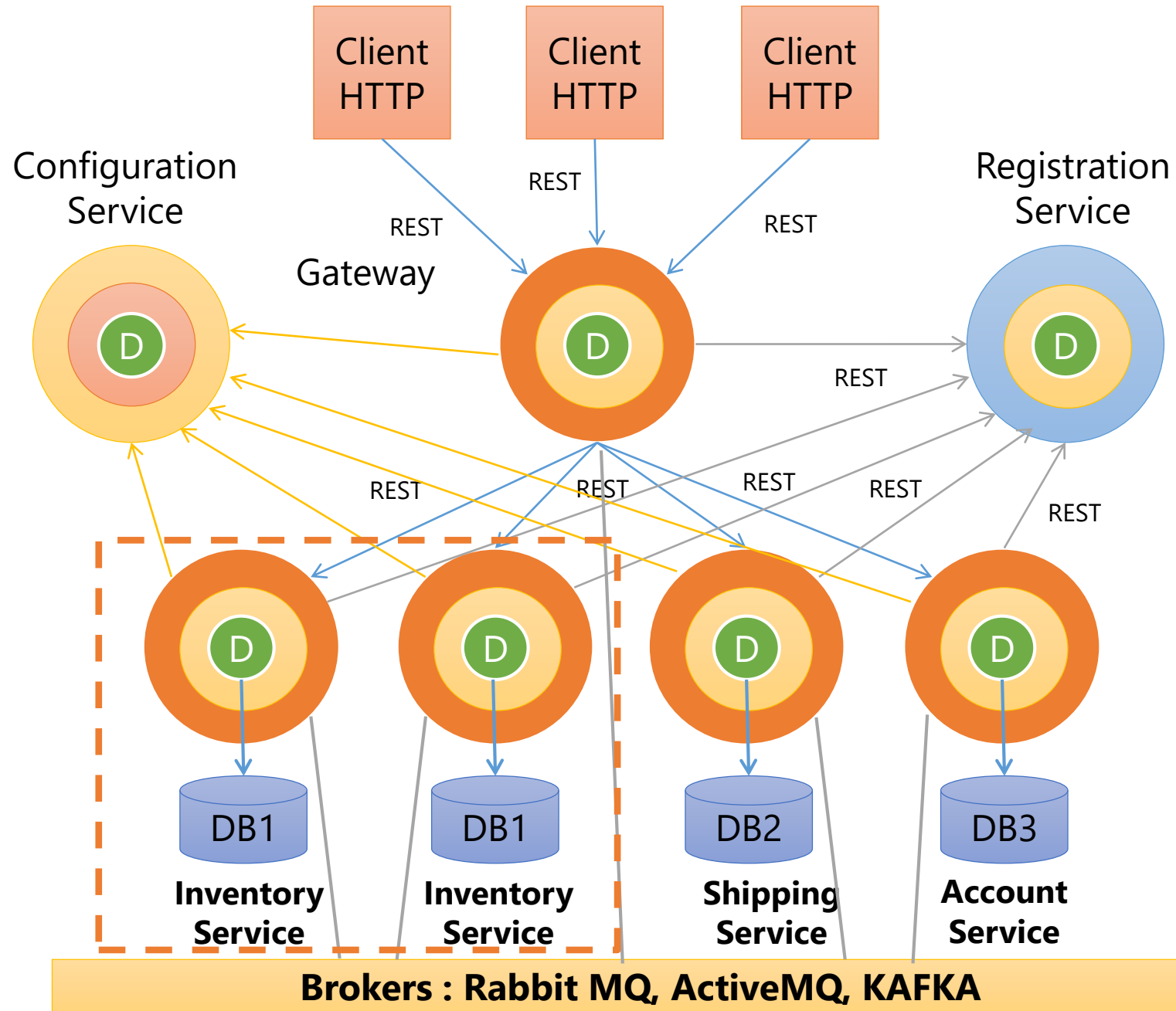


Architectures

Systemes Distribués basés sur les Micro services avec Spring Cloud



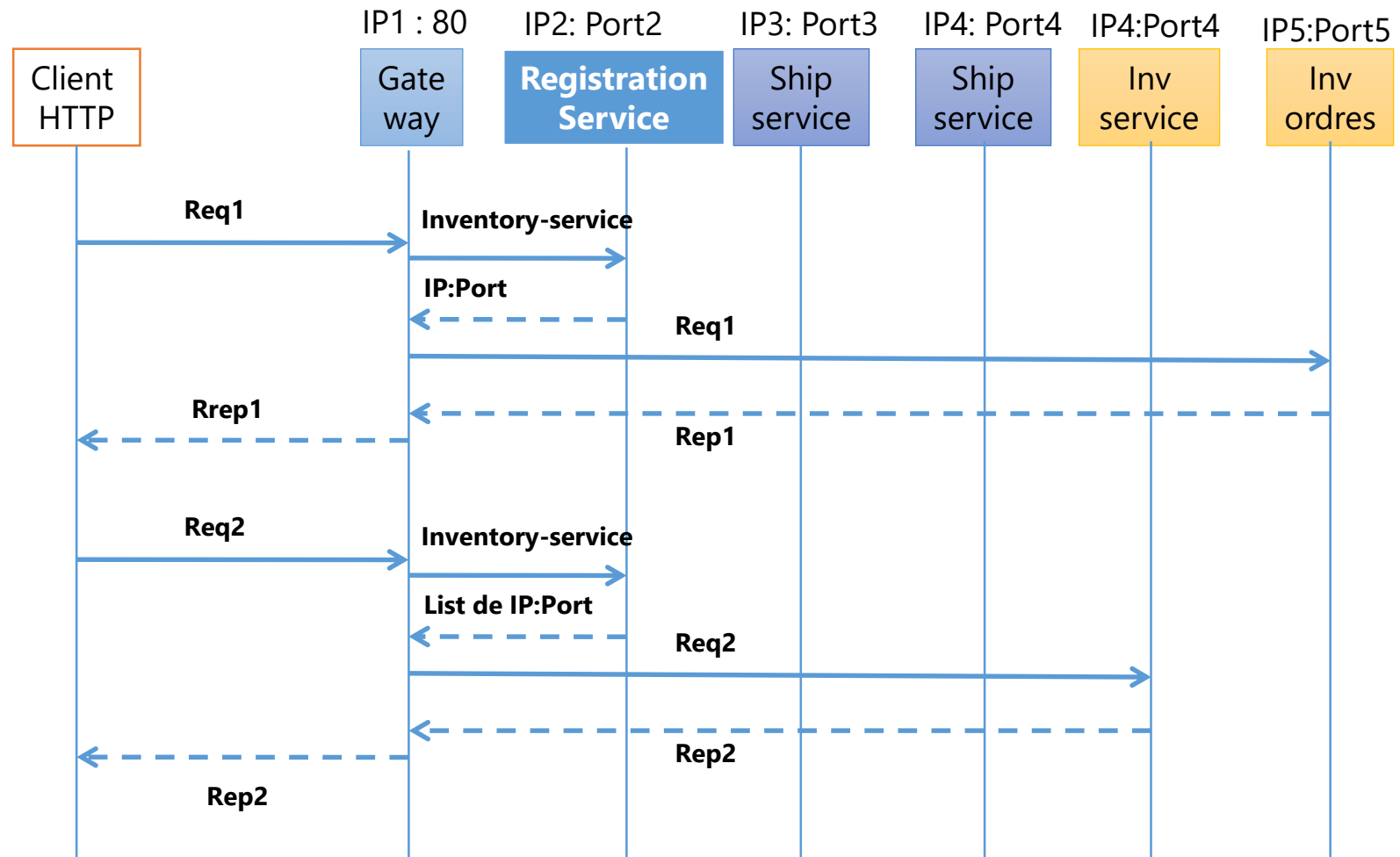
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@yousfi.net
Supports de cours : <http://fr.slideshare.net/mohamedyousfi9>
Chaîne vidéo : <http://youtube.com/mohamedYousfi>
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications



Consulter les services via le service proxy

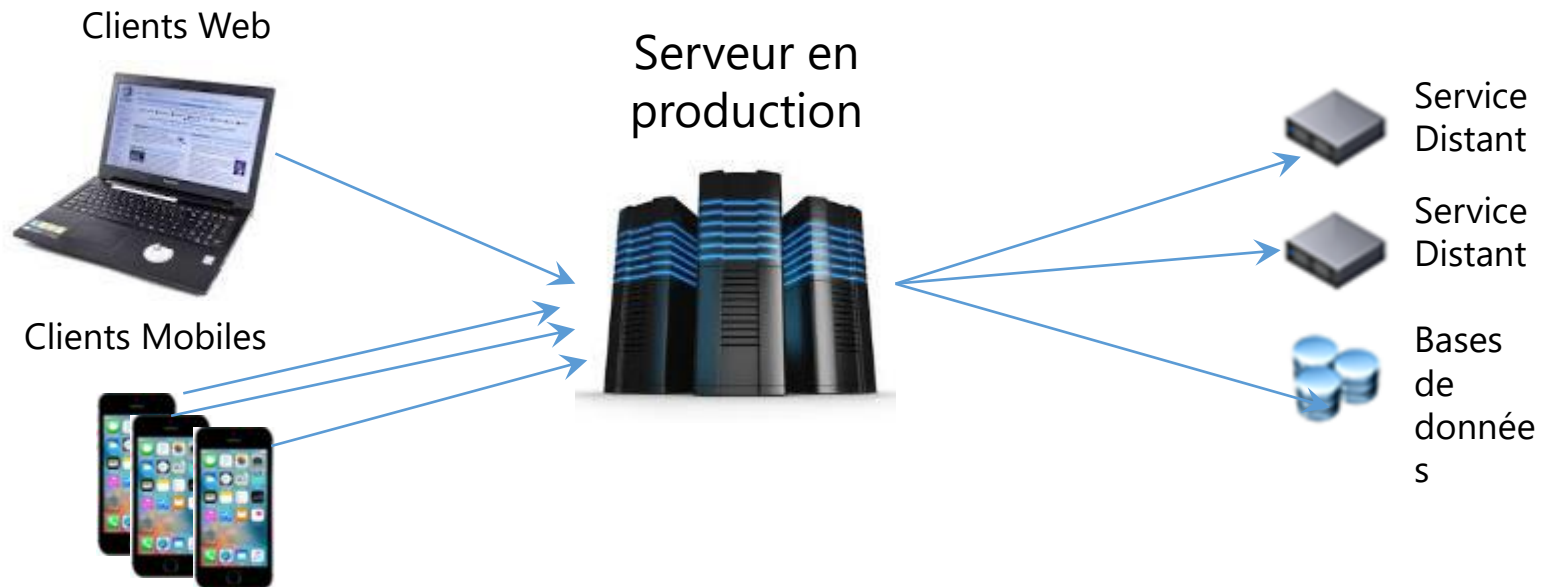
Req 1 : GET http://gateway/inventory-service/products

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



Blocking IO Model : Latency Problem

- 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

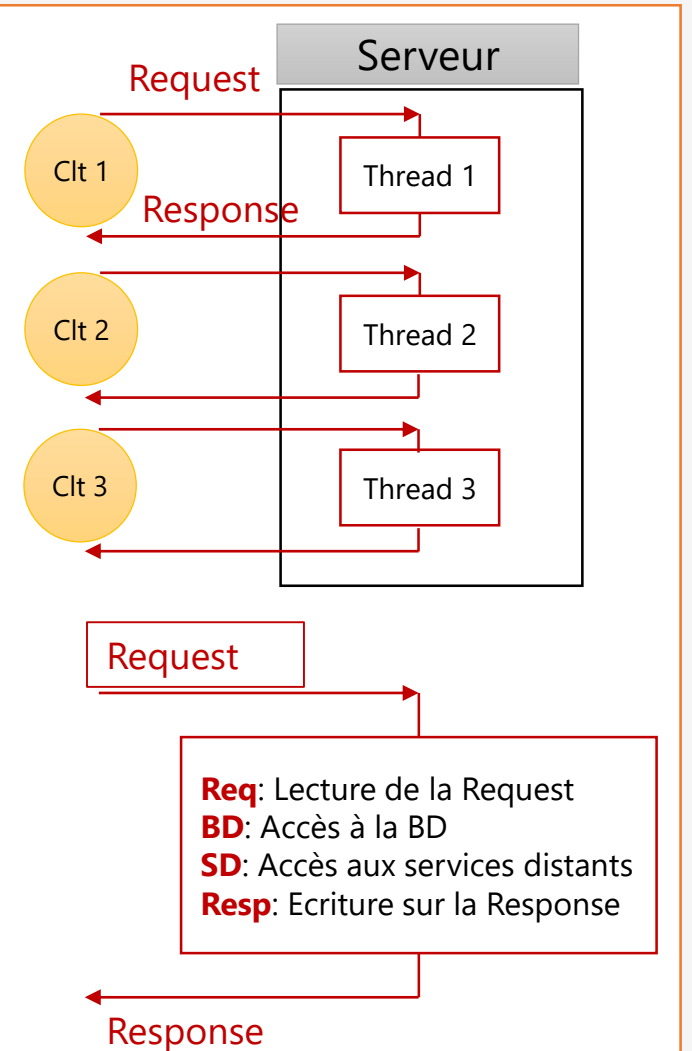


Modèle Multi Threads Bloquant

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

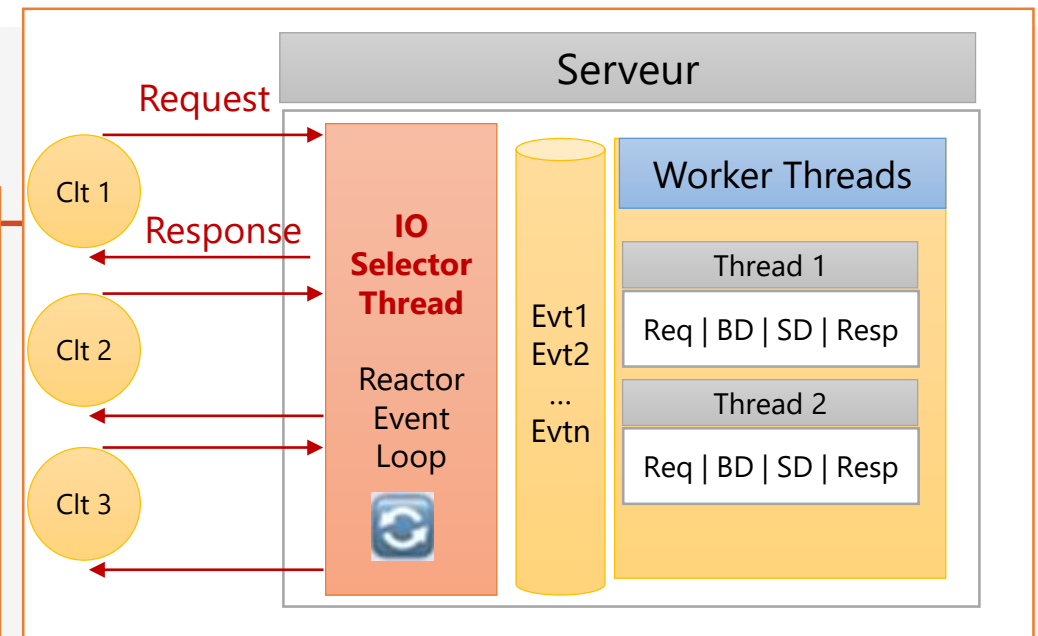
- Pendant ce traitement on peut avoir :
 1. Lecture des données de la requête
 2. Accéder à une base de données
 3. Accéder à des services distants
 4. Ecriture sur la response
- Toutes ces Entrées Sorties sont bloquantes
- Le thread attend la lecture et l'écriture sur les IO
- Dans le cas d'une connexion 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 :

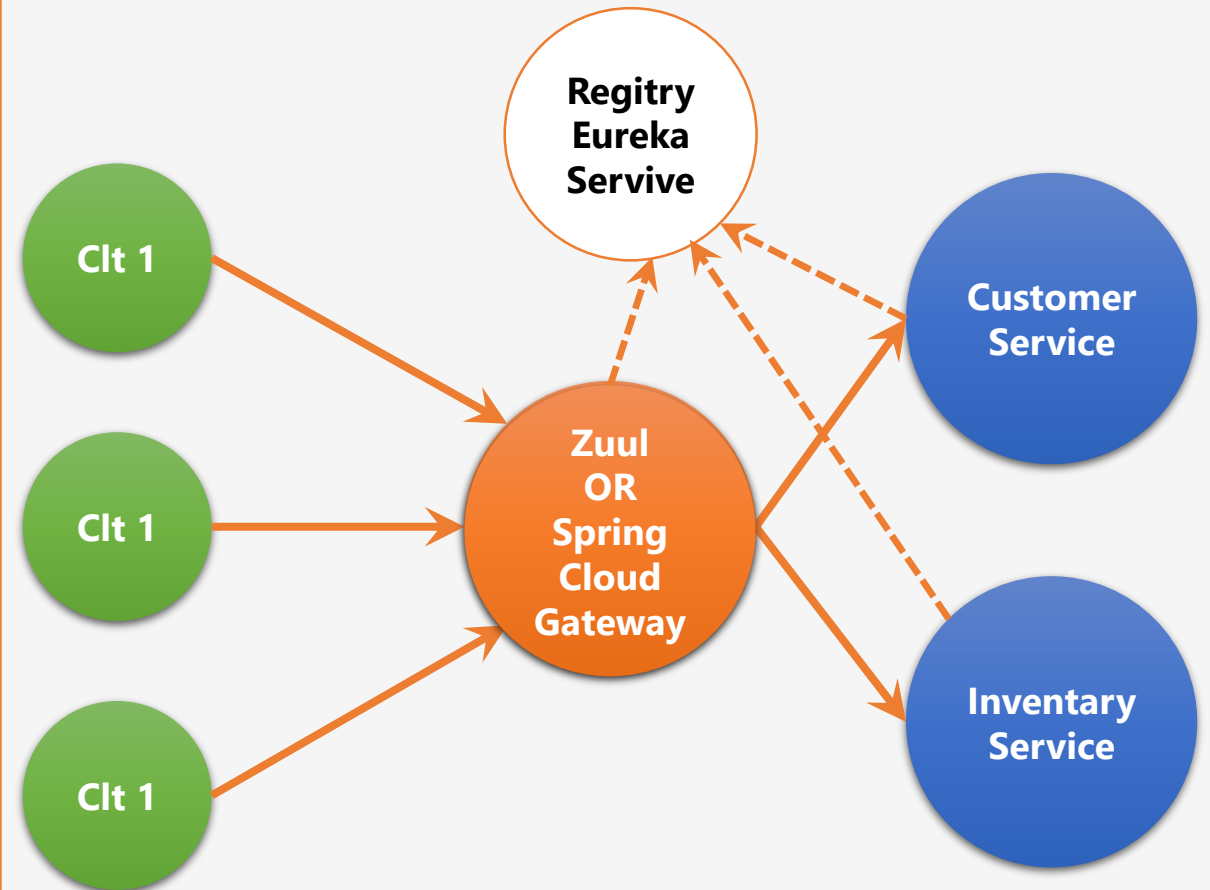
- Dans ce modèle on n'aura pas besoin d'un Thread par requête / réponse



- On a un modèle qui utilise un nombre beaucoup plus réduit de threads
 - Un IO Selector Thread dont le rôle est d'orchestrer les entrées sorties Non bloquantes.
 - Cette fois-ci tous les IO 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 événements** 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 Worker Threads vont s'occuper de traiter les requêtes de manière non bloquantes. Ils ne vont jamais attendre. Ils seront toujours entrain de travailler et exploiter aux maximum les ressources du serveur
 - Ce modèle assure la scalabilité verticale : les performances augmentent avec la capacité du serveur (CPUs, Mémoire, Stockage, etc...)
 - La latence des IO ne va pas impacter les performances du serveur.

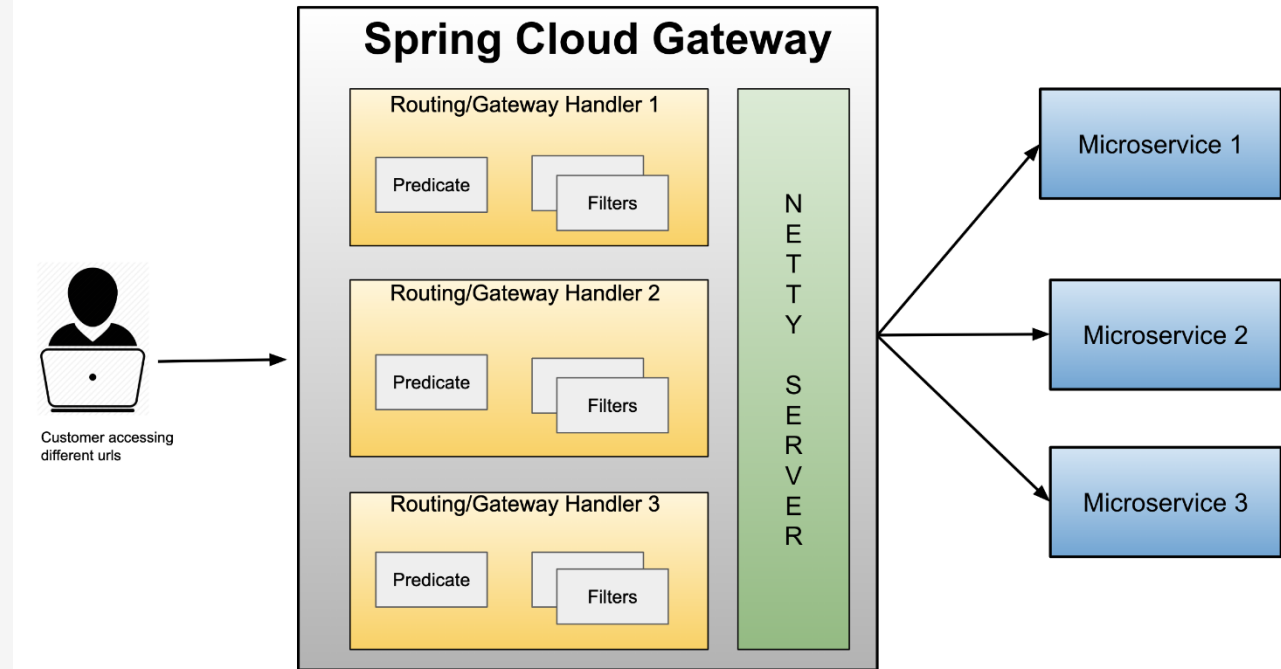
Spring Cloud Gateway

- 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.



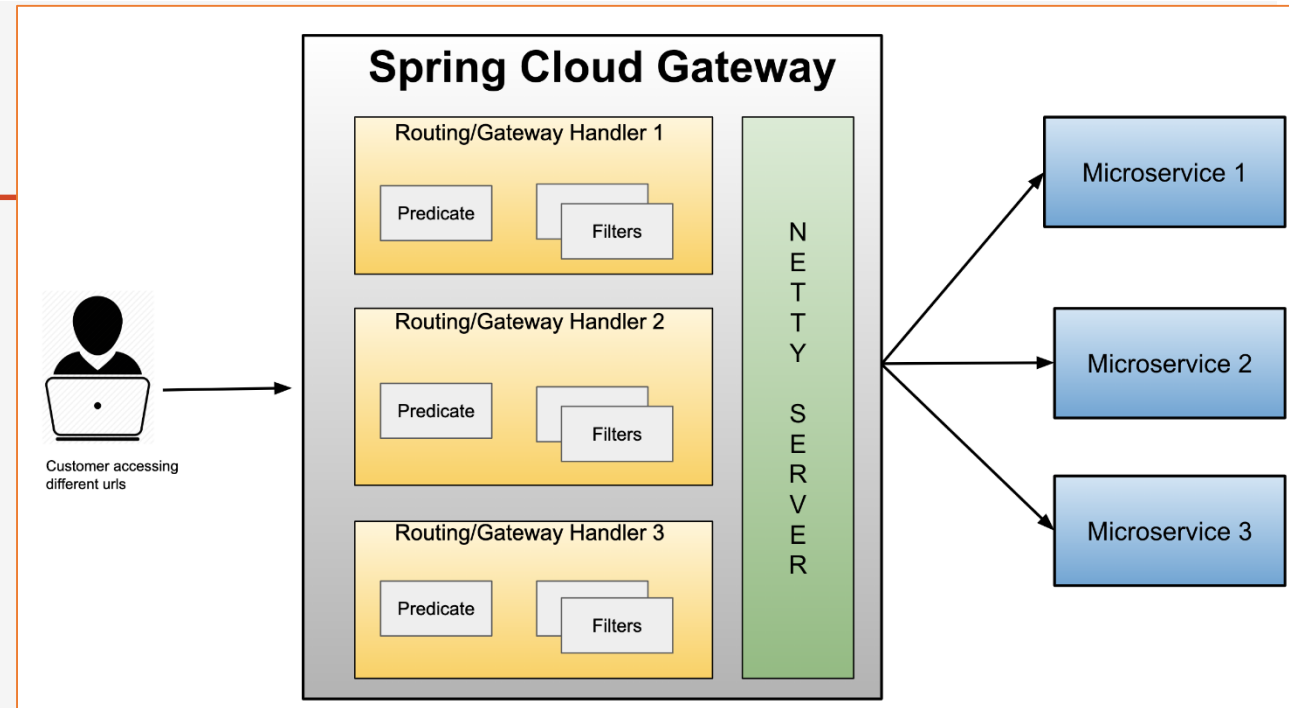
Spring Cloud Gateway

- 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:
- Route: Destination vers laquelle nous voulons qu'une requête particulière soit acheminée. Une route comprend :
 - l'URI de destination,
 - Predicate : Une condition qui doit satisfaire
 - Filters : Un ou plusieurs filtres qui peuvent intervenir pour apporter des traitements et des modifications des requêtes et des réponses HTTP



Predicates :

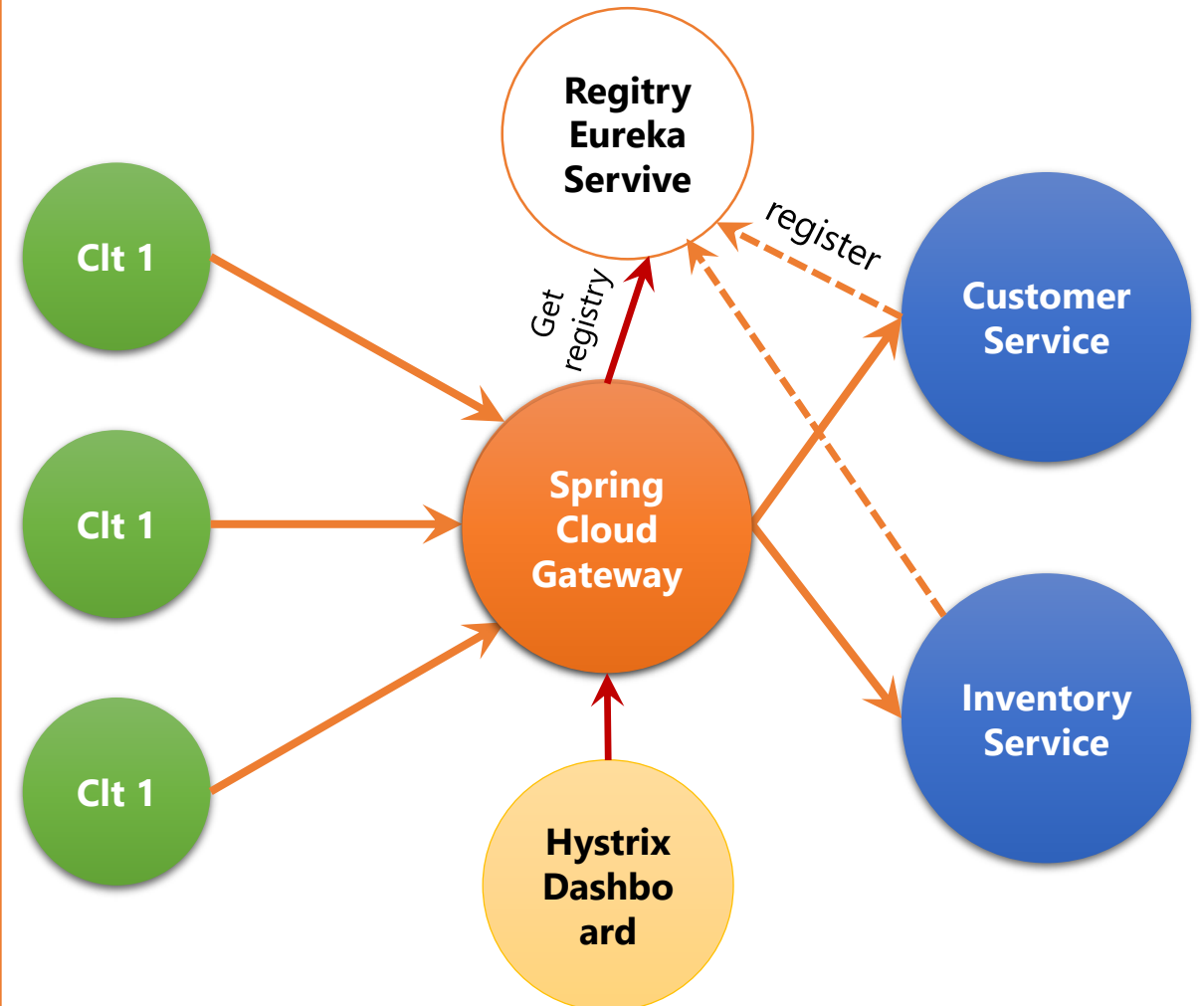
- Host, Path, Method
- After, Before, Between
- Cookie, Header, Query
- RemoteAddr
- Etc ...

Filters :

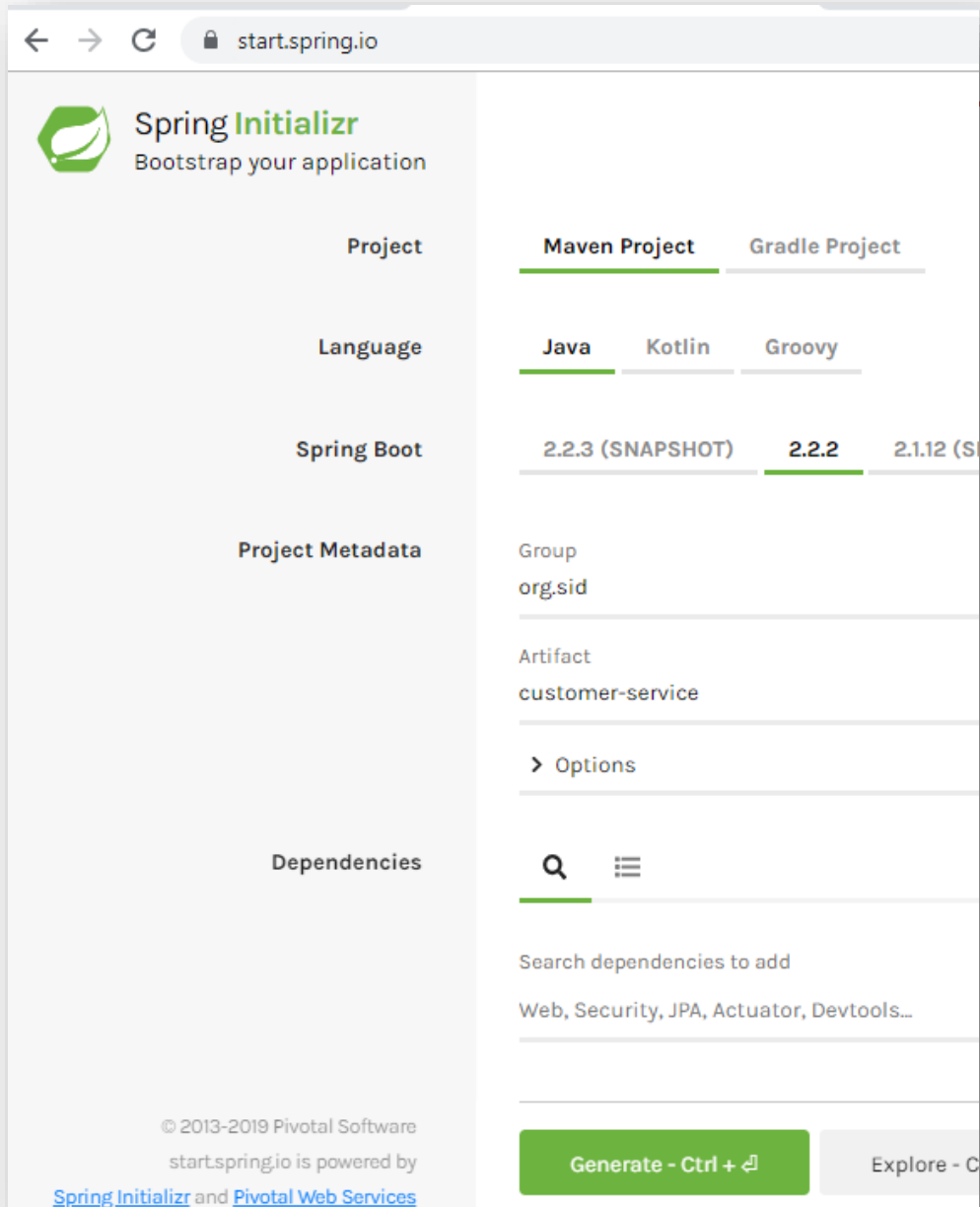
- AddRequestHeader
- AddRequestParameter
- AddResponseHeader
- DedupeResponseHeader
- Hystrix
- CircuitBreaker
- RewritePath
- Etc ...

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



Customer-service



The screenshot shows the Spring Initializr web application at start.spring.io. The interface is divided into several sections:

- Project:** Maven Project (selected) and Gradle Project.
- Language:** Java (selected), Kotlin, and Groovy.
- Spring Boot:** 2.2.3 (SNAPSHOT), 2.2.2 (selected), and 2.1.12 (S).
- Project Metadata:**
 - Group: org.sid
 - Artifact: customer-service
 - Options: expandable section
- Dependencies:** Search bar with a magnifying glass icon and a menu icon. Below the search bar, it says "Search dependencies to add" and lists "Web, Security, JPA, Actuator, Devtools...".

At the bottom, there is a green button labeled "Generate - Ctrl + ⌘" and a grey button labeled "Explore - C".

© 2013-2019 Pivotal Software
start.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)

Selected dependencies

- **Spring Web** : Build web, including RESTful, applications using Spring MVC. 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.

Customer-service : CustomerServiceApplication.java

```
package org.id.customerservice;
import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor; import lombok.ToString; import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.JpaRepository; import org.springframework.data.rest.core.annotation.RepositoryRestResource; import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;
```

application.properties

```
spring.cloud.discovery.enabled=false
server.port=8081
spring.application.name=customer-service
#management.endpoints.web.exposure.include=*
```

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

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

```
@SpringBootApplication
public class CustomerServiceApplication {
```

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

```
@Bean
CommandLineRunner start(CustomerRepository customerRepository){
    return args -> {
        customerRepository.save(new Customer(null,"Enset","contact@enset-media.ma"));
        customerRepository.save(new Customer(null,"FSTM","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM","contact@ensam.ma"));
        customerRepository.findAll().forEach(System.out::println);
    };
}
```

```
@Projection(name = "fullCustomer",types =
Customer.class)
interface CustomerProjection extends Projection{
    public Long getId();
    public String getName();
    public String getEmail();
}
```

Customer-service

localhost:8081/customers

```
{
  "_embedded": {
    "customers": [
      {
        "name": "Enset",
        "email": "contact@enset-media.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1"
          }
        }
      },
      {
        "name": "FSTM",
        "email": "contact@fstm.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2"
          }
        }
      },
      {
        "name": "ENSAM",
        "email": "contact@ensam.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/3"
          }
        }
      }
    ]
  }
}
```

localhost:8081/customers?projection=fullCustomer

```
{
  "_embedded": {
    "customers": [
      {
        "name": "Enset",
        "id": 1,
        "email": "contact@enset-media.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1?projection=fullCustomer",
            "templated": true
          }
        }
      },
      {
        "name": "FSTM",
        "id": 2,
        "email": "contact@fstm.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2?projection=fullCustomer",
            "templated": true
          }
        }
      },
      {
        "name": "ENSAM",
        "id": 3,
        "email": "contact@ensam.ma",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/3"
          },
          "customer": {
            "href": "http://localhost:8081/customers/3?projection=fullCustomer",
            "templated": true
          }
        }
      }
    ]
  }
}
```

localhost:8081/customers/1

```
{
  "name": "Enset",
  "email": "contact@enset-media.ma",
  "_links": {
    "self": {
      "href": "http://localhost:8081/customers/1"
    },
    "customer": {
      "href": "http://localhost:8081/customers/1"
    }
  }
}
```

localhost:8081/actuator

```
{
  "_links": {
    "self": {
      "href": "http://localhost:8081/actuator",
      "templated": false
    },
    "archaius": {
      "href": "http://localhost:8081/actuator/archaius",
      "templated": false
    },
    "beans": {
      "href": "http://localhost:8081/actuator/beans",
      "templated": false
    }
  }
}
```

localhost:8081/actuator/health

```
{
  "status": "UP"
}
```

Customer-service : Base de données H2 (<http://localhost:8081/h2-console>)

← → ↻ ⓘ localhost:8081/h2-console/login.jsp?jsessionId=b62a346ed406e8b4c23322a7bc54c9ec

English ▼ Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded) ▼

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:testdb

User Name: sa

Password:

Connect Test Connection

← → ↻ ⓘ localhost:8081/h2-console/login.do?jsessionId=b62a346ed406e8b4c23322a7bc54c9ec

Auto commit Max rows: 1000

Run Run Selected Auto complete Clear

jdbc:h2:mem:testdb

- CUSTOMER
 - ID
 - EMAIL
 - NAME
 - Indexes
- INFORMATION_SCHEMA
- Sequences
- Users

H2 1.4.200 (2019-10-14)

```
SELECT * FROM CUSTOMER
```

```
SELECT * FROM CUSTOMER;
```

ID	EMAIL	NAME
1	contact@enset-media.ma	Enset
2	contact@fstm.ma	FSTM
3	contact@ensam.ma	ENSAM

(3 rows, 4 ms)

Edit

Inventory-service

The screenshot shows the Spring Initializr web application at start.spring.io. The interface is divided into several sections:

- Project:** Maven Project (selected), Gradle Project
- Language:** Java (selected), Kotlin, Groovy
- Spring Boot:** 2.2.3 (SNAPSHOT), 2.2.2 (selected)
- Project Metadata:**
 - Group: org.id
 - Artifact: inventory-service
 - > Options
- Dependencies:** Search dependencies to add. A search bar with a magnifying glass icon and a menu icon is visible. Below it, the text "Web, Security, JPA, Actuator, Dev" is partially visible.

At the bottom, there is a green button labeled "Generate - Ctrl + ⌘".

© 2013-2019 Pivotal Software
start.spring.io is powered by
[Spring Initializr](#) and [Pivotal Web Services](#)

Selected dependencies

- **Spring Web** : Build web, including RESTful, applications using Spring MVC. 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.

Inventory-service : InventoryServiceApplication.java

```
package org.id.inventoryservice;
```

```
import ...
```

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

```
@RepositoryRestResource  
interface ProductRepository extends JpaRepository<Product,Long> { }
```

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

```
@Bean  
CommandLineRunner start(ProductRepository productRepository){  
    return args -> {  
        productRepository.save(new Product(null,"Computer Desk Top HP",900));  
        productRepository.save(new Product(null,"Printer Epson",80));  
        productRepository.save(new Product(null,"MacBook Pro Lap Top",1800));  
        productRepository.findAll().forEach(System.out::println);  
    };  
}
```

```
}
```

application.properties

```
spring.application.name=inventory-service  
spring.cloud.discovery.enabled=false  
server.port=8082
```

Inventory-service

```
localhost:8082/products

{
  "_embedded": {
    "products": [
      {
        "name": "Computer Desk Top HP",
        "price": 900,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/1"
          },
          "product": {
            "href": "http://localhost:8082/products/1"
          }
        }
      },
      {
        "name": "Printer Epson",
        "price": 80,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/2"
          },
          "product": {
            "href": "http://localhost:8082/products/1"
          }
        }
      }
    ]
  }
}
```

```
localhost:8082/products/1

{
  "name": "Computer Desk Top HP",
  "price": 900,
  "_links": {
    "self": {
      "href": "http://localhost:8082/products/1"
    },
    "product": {
      "href": "http://localhost:8082/products/1"
    }
  }
}
```

localhost:8082/h2-console/login.do?jsessionId=...

Auto commit Max rows: 1000

jdbc:h2:mem:testdb

- PRODUCT
 - ID
 - NAME
 - PRICE
 - Indexes
- INFORMATION_SCHEMA
- Sequences
- Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear

SELECT * FROM PRODUCT

SELECT * FROM PRODUCT;

ID	NAME	PRICE
1	Computer Desk Top HP	900.0
2	Printer Epson	80.0
3	MacBook Pro Lap Top	1800.0

(3 rows, 7 ms)

Edit

Gateway-service



Spring Initializr

Bootstrap your application

Project

Maven Project

Gradle Project

Language

Java

Kotlin

Groovy

Spring Boot

2.2.3 (SNAPSHOT)

2.2.2

Project Metadata

Group

org.id

Artifact

gateway-service

> Options

Dependencies



Search dependencies to add

[Web](#), [Security](#), [JPA](#), [Actuator](#), [DevTools](#)

Generate - Ctrl + ⌘

© 2013-2019 Pivotal Software

start.spring.io is powered by

[Spring Initializr](#) and [Pivotal Web Services](#)

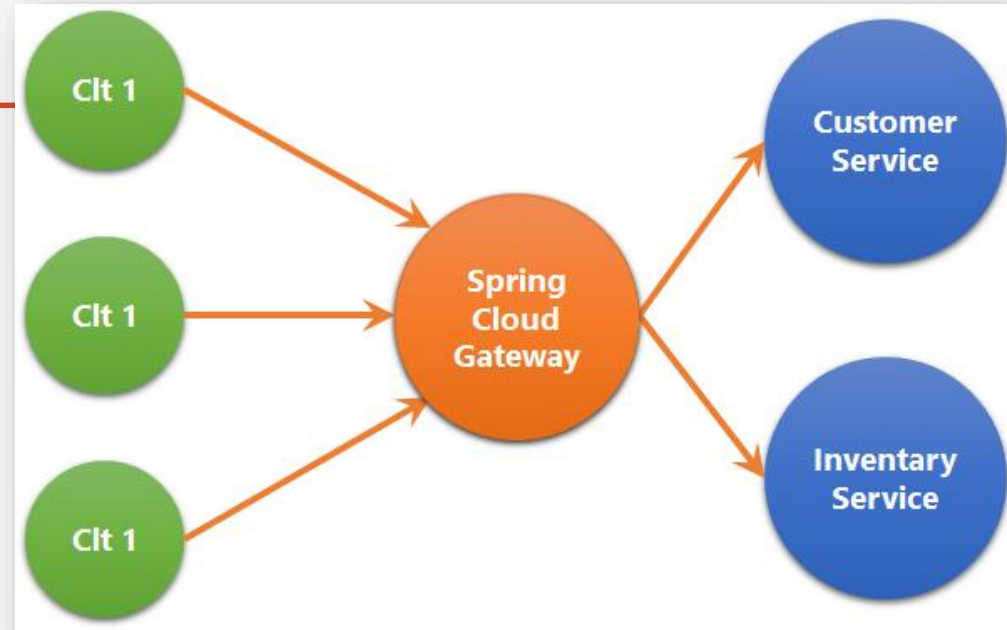
Selected dependencies

- **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.
- **Hystrix** : Circuit breaker with Spring Cloud Netflix Hystrix.
- **Eureka Discovery Client** : a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

Static routes configuration: application.yml

application.yml

```
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/customers/1

```
{
  "name": "Enset",
  "email": "contact@enset-media.ma",
  "_links": {
    "self": {
      "href": "http://localhost:8081/customers/1"
    },
    "customer": {
      "href": "http://localhost:8081/customers/1"
    }
  }
}
```

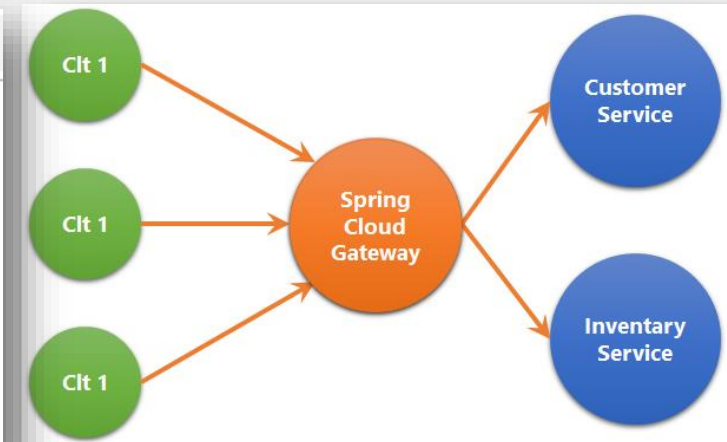
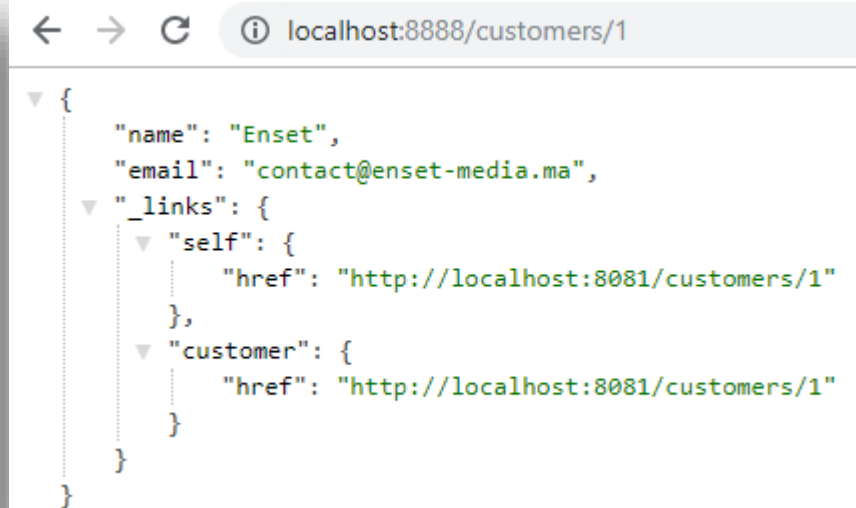
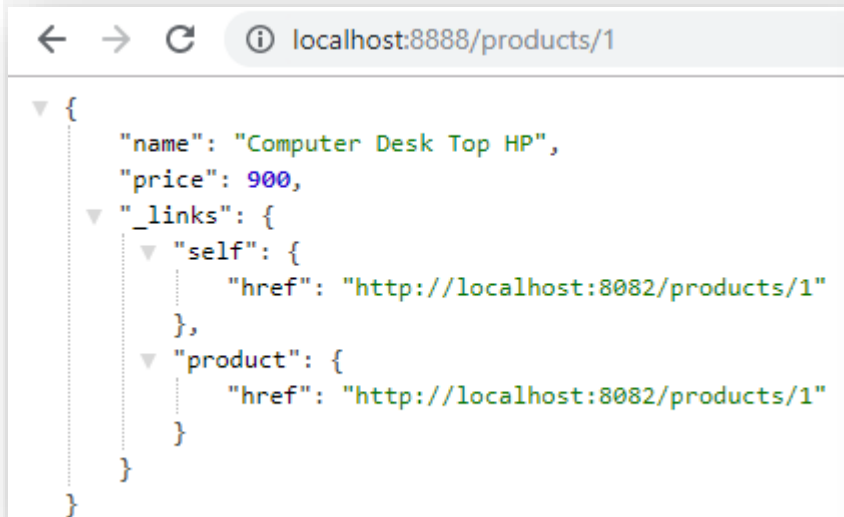
localhost:8888/products/1

```
{
  "name": "Computer Desk Top HP",
  "price": 900,
  "_links": {
    "self": {
      "href": "http://localhost:8082/products/1"
    },
    "product": {
      "href": "http://localhost:8082/products/1"
    }
  }
}
```

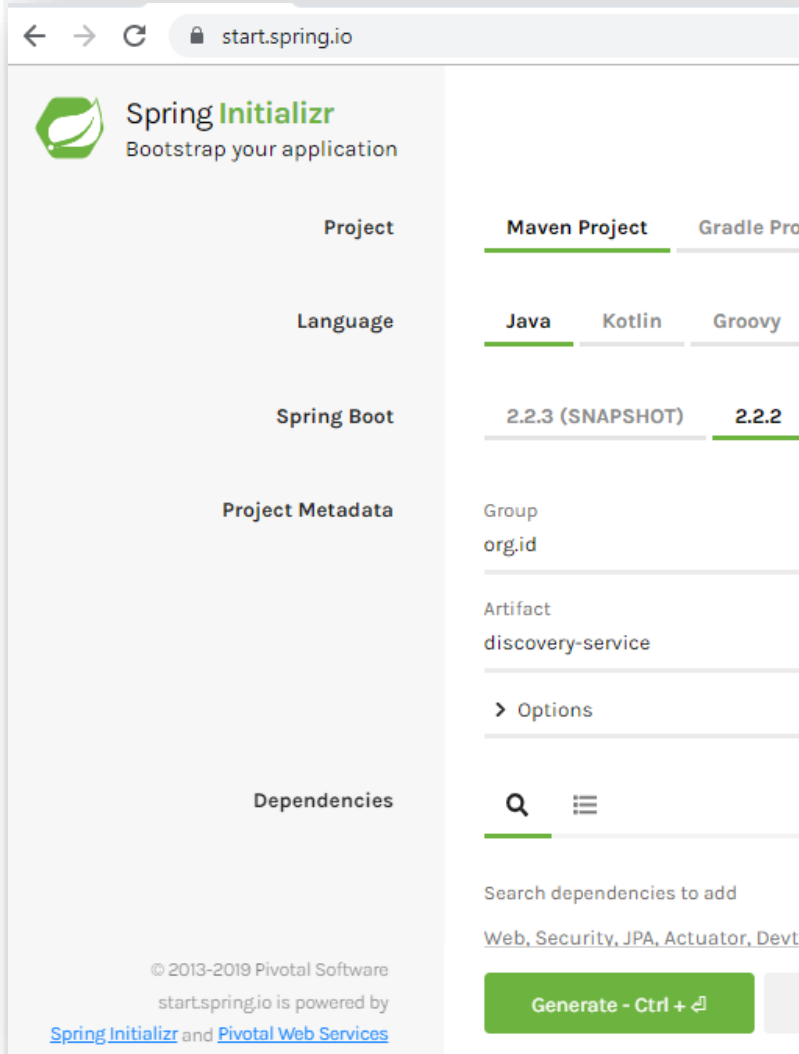
Static routes configuration: Java Config Class

@Bean

```
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){  
    return builder.routes()  
        .route(r->r.path("/customers/**").uri("http://localhost:8081/").id("r1"))  
        .route(r->r.path("/products/**").uri("http://localhost:8082/").id("r2"))  
        .build();  
}
```

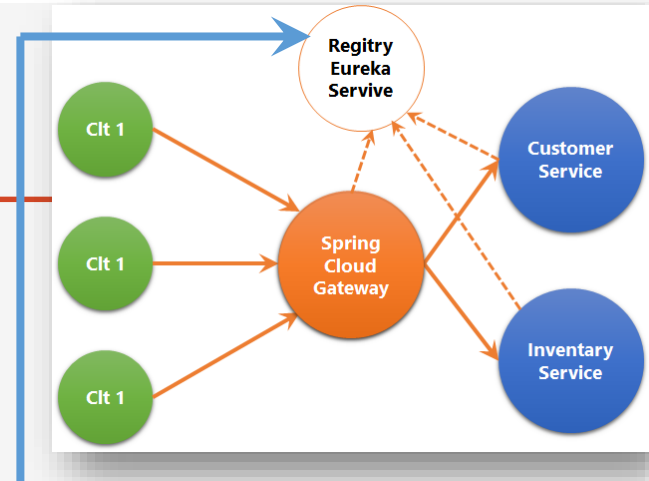


Eureka Discovery Service : Dynamic Routing



Selected dependencies

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



```
package org.id.discovery.service; import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServiceApplication.class, args);
    }
}
```

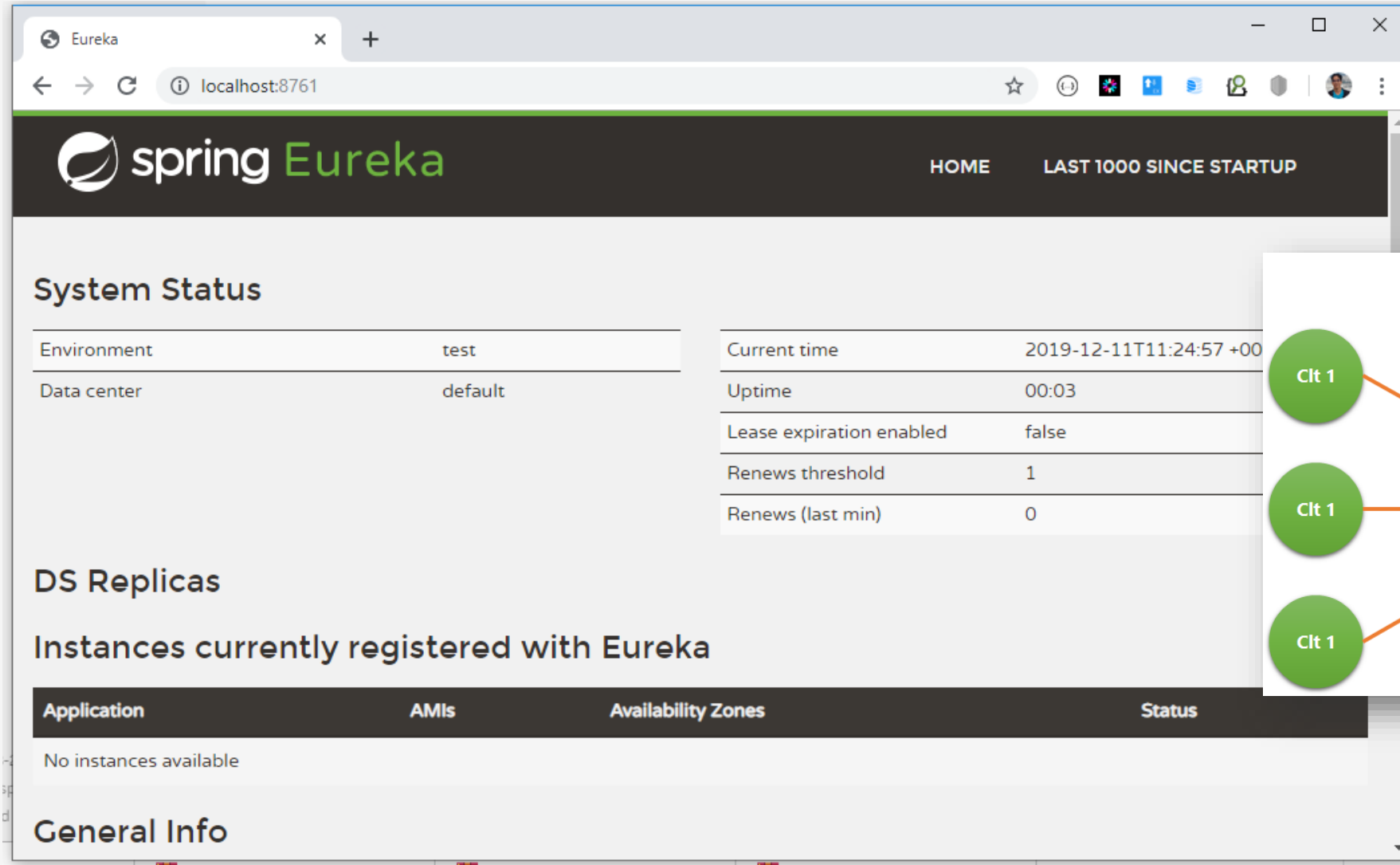
server.port=8761

application.properties

eureka.client.fetch-registry=false

eureka.client.register-with-eureka=false

Eureka Discovery Service : Dynamic Routing



The screenshot shows the Spring Eureka web interface in a browser window. The address bar shows `localhost:8761`. The page has a dark header with the Spring Eureka logo and navigation links for `HOME` and `LAST 1000 SINCE STARTUP`.

System Status

Environment	test
Data center	default

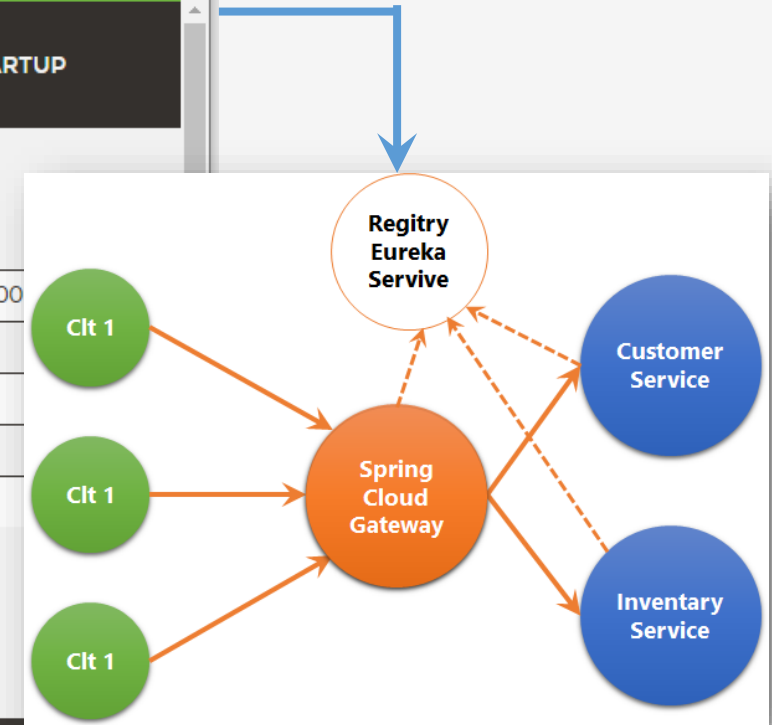
Current time	2019-12-11T11:24:57 +00
Uptime	00:03
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info



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

Customer-service

```
spring.cloud.discovery.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

Inventory-service

```
spring.cloud.discovery.enabled=true
server.port=8082
spring.application.name=inventory-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

application.properties

Eureka Discovery Service : Dynamic Routing

← → ↺ ⓘ localhost:8761

spring Eureka

HOME LAST 1000 S

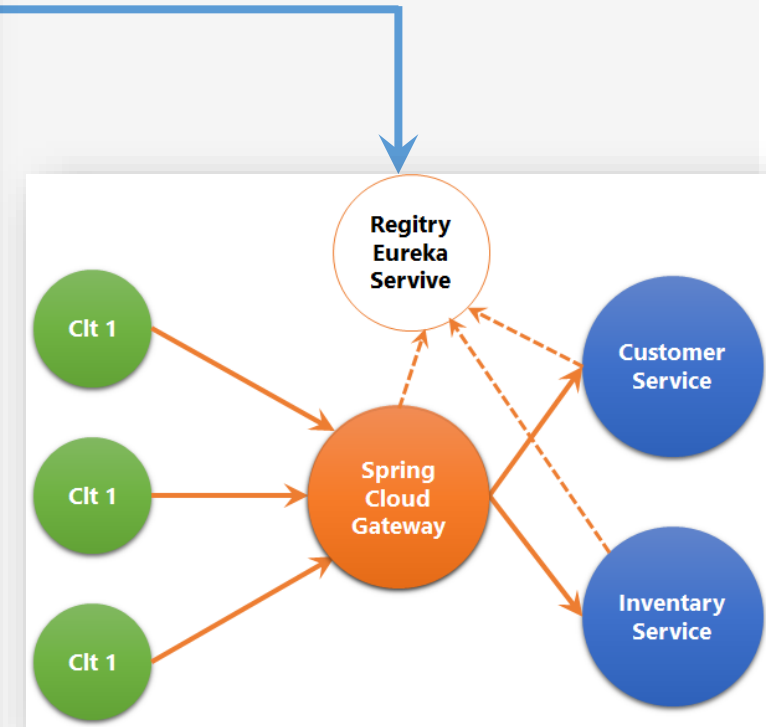
System Status

Environment	test	Current time	2019-12-11T13:
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

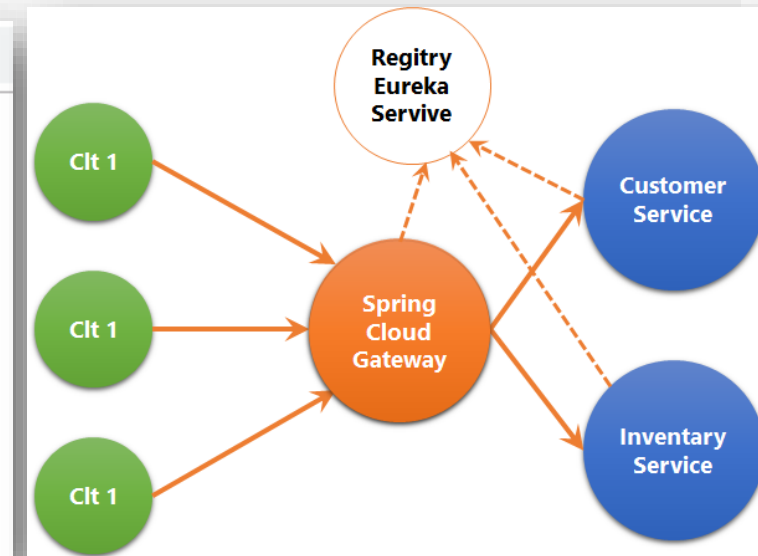
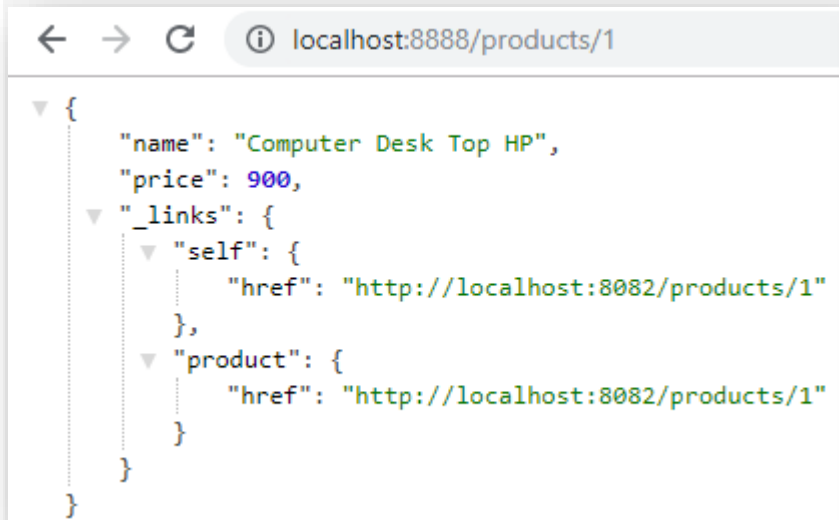
Application	AMIs	Availability Zones	Status
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - localhost:customer-service:8081
INVENTORY-SERVICE	n/a (1)	(1)	UP (1) - localhost:inventory-service:8082



Static routes configuration with Discovery Service

@Bean

```
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){  
    return builder.routes()  
        .route(r->r.path("/customers/**").uri("lb://CUSTOMER-SERVICE").id("r1"))  
        .route(r->r.path("/products/**").uri("lb://INVENTORY-SERVICE").id("r2"))  
        .build();  
}
```



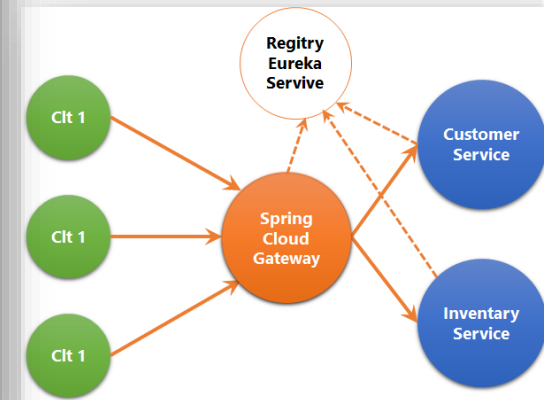
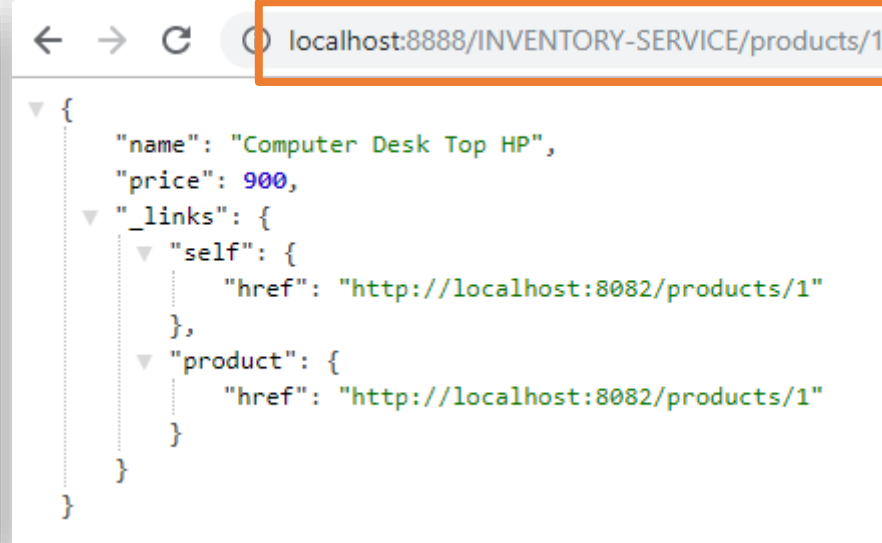
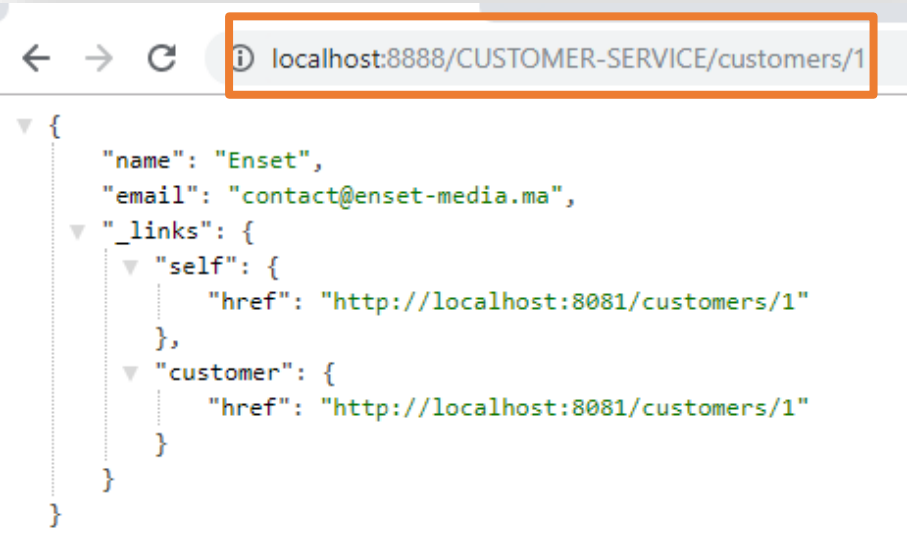
Dynamic routes configuration with Discovery Service

application.properties

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

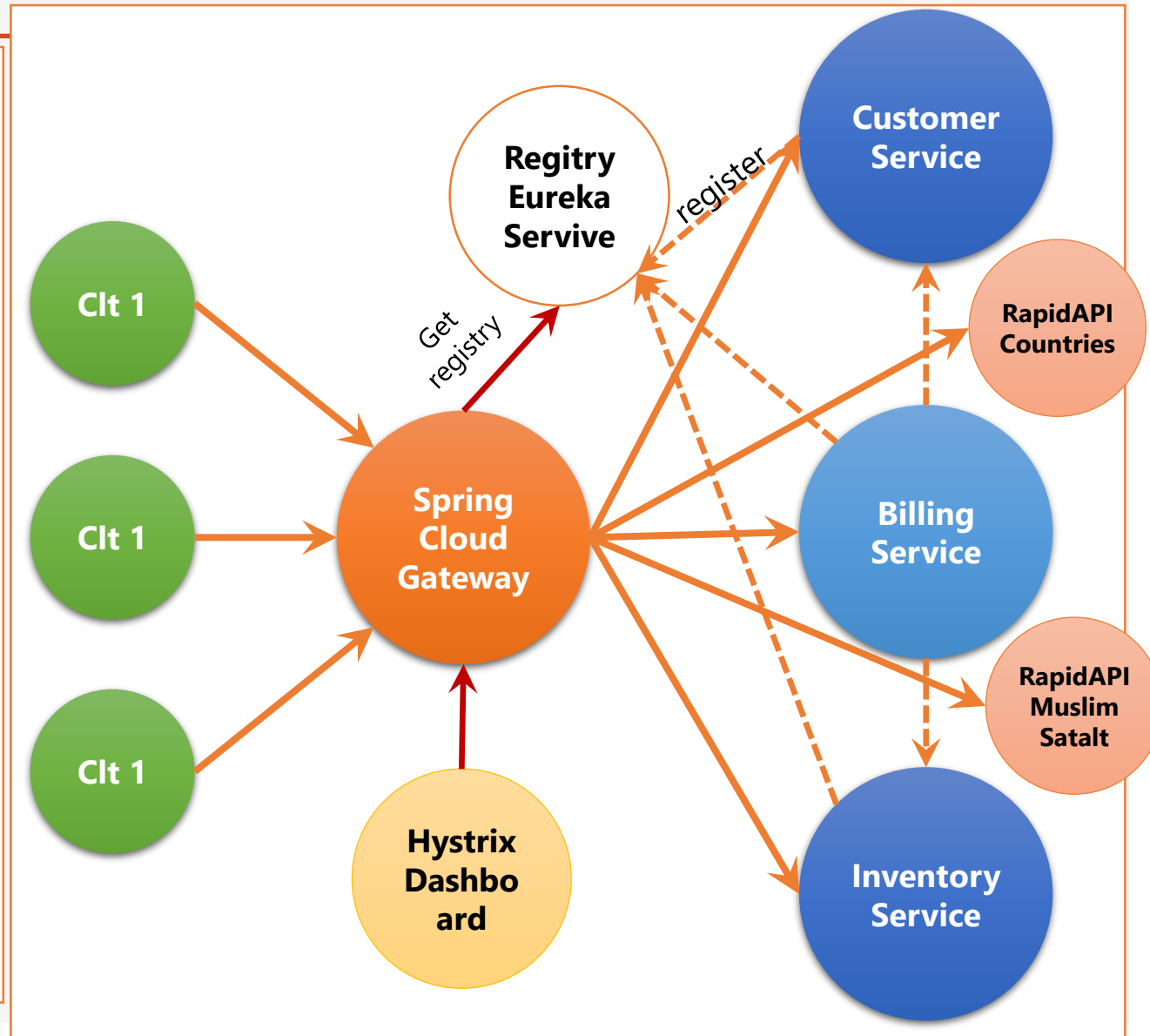
@Bean

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



Autres services à ajouter

1. Accès aux services externes en utilisant des filtres au niveau du gateway service :
 - RapidAPI Countries
 - Rapid API Mulsim Salat
2. Utilisation de Circuit Breaker avec Hystrix
3. Utilisation de Hystrix Dashboard pour surveiller l'état du trafic au niveau du service Gateway
4. Ajouter un service de facturation (Billing Service), qui communique avec les services Clients et Inventaire en utilisant Spring cloud OpenFeign Rest Client



Exemple de : Routes Filters

@Bean

```
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){  
    return builder.routes()  
        .route(r->r.path("/restcountries/**")  
            .filters(f->f  
                .addRequestHeader("x-rapidapi-host", "restcountries-v1.p.rapidapi.com")  
                .addRequestHeader("x-rapidapi-key", "fe5e774996msh4eb6e863d457420p1d2ffbjsnee0617ac5078")  
                .rewritePath("/restcountries/(?<segment>.*)", "/${segment}")  
            )  
            .uri("https://restcountries-v1.p.rapidapi.com").id("countries")  
        )  
        .route(r->r.path("/muslimsalat/**")  
            .filters(f->f  
                .addRequestHeader("x-rapidapi-host", "muslimsalat.p.rapidapi.com")  
                .addRequestHeader("x-rapidapi-key", "fe5e774996msh4eb6e863d457420p1d2ffbjsnee0617ac5078")  
                .rewritePath("/muslimsalat/(?<segment>.*)", "/${segment}")  
            )  
            .uri("https://muslimsalat.p.rapidapi.com")  
            .id("countries")  
        )  
        .build();  
}
```

Static Routes with Filters

localhost:8888/muslimsalat/marrakech/daily/5.json

```
{
  "title": "",
  "query": "marrakech",
  "for": "daily",
  "method": "5",
  "prayer_method_name": "Muslim World League",
  "daylight": "1",
  "timezone": "1",
  "map_image": "https://maps.google.com/maps/api/staticmap?c",
  "sealevel": "451",
  "today_weather": {
    "pressure": "1023",
    "temperature": "11"
  },
  "link": "http://muslimsalat.com/marrakech",
  "qibla_direction": "91.44",
  "latitude": "31.633333",
  "longitude": "-8.000000",
  "address": "",
  "city": "Marrakesh",
  "state": "Marrakesh-Tensift-Al Haouz",
  "postal_code": "",
  "country": "Morocco",
  "country_code": "MA",
  "items": [
    {
      "date_for": "2019-12-14",
      "fajr": "7:56 am",
      "shurooq": "9:15 am",
      "dhuhr": "2:26 pm",
      "asr": "5:11 pm",
      "maghrib": "7:37 pm",
      "isha": "8:51 pm"
    }
  ],
  "status_valid": 1,
  "status_code": 1,
  "status_description": "Success."
}
```

localhost:8888/muslimsalat/rabat/weekly/1.json

```
{
  "query": "rabat",
  "for": "weekly",
  "method": "1",
  "prayer_method_name": "Egyptian General Authority of Su",
  "daylight": "1",
  "timezone": "1",
  "map_image": "https://maps.google.com/maps/api/staticma",
  "sealevel": "72",
  "today_weather": {
    "pressure": "1024",
    "temperature": "13"
  },
  "link": "http://muslimsalat.com/rabat",
  "qibla_direction": "94.66",
  "latitude": "34.015049",
  "longitude": "-6.832720",
  "address": "",
  "city": "Rabat",
  "state": "Rabat-Sale-Zemmour-Zaer",
  "postal_code": "",
  "country": "Morocco",
  "country_code": "MA",
  "items": [
    {
      "date_for": "2019-12-14",
      "fajr": "7:45 am",
      "shurooq": "9:18 am",
      "dhuhr": "2:21 pm",
      "asr": "5:01 pm",
      "maghrib": "7:25 pm",
      "isha": "8:48 pm"
    }
  ],
  { ... }, // 7 items
  { ... }, // 7 items
  { ... }, // 7 items
  { ... }, // 7 items
  { ... }, // 7 items
  { ... } // 7 items
}
```

localhost:8888/restcountries/all

```
{ ... }, // 22 items
{ ... }, // 22 items
{
  "name": "Morocco",
  "topLevelDomain": [
    ".ma"
  ],
  "alpha2Code": "MA",
  "alpha3Code": "MAR",
  "callingCodes": [
    "212"
  ],
  "capital": "Rabat",
  "altSpellings": [
    "MA",
    "Kingdom of Morocco",
    "Al-Mamlakah al-Mağribiyah"
  ],
  "region": "Africa",
  "subregion": "Northern Africa",
  "population": 33337529,
  "latlng": [
    32,
    -5
  ],
  "demonym": "Moroccan",
  "area": 446550,
  "gini": 40.9,
  "timezones": [
    "UTC"
  ],
  "borders": [
    "DZA",
    "ESH",
    "ESP"
  ],
  "nativeName": "المغرب",
  "numericCode": "504",
}
```

Static Routes with Filters

localhost:8888/muslimsalat/marrakech/daily/5.json

```
{
  "title": "",
  "query": "marrakech",
  "for": "daily",
  "method": "5",
  "prayer_method_name": "Muslim World Le",
  "daylight": "1",
  "timezone": "1",
  "map_image": "https://maps.google.com/",
  "sealevel": "451",
  "today_weather": {
    "pressure": "1023",
    "temperature": "11"
  },
  "link": "http://muslimsalat.com/marrak",
  "qibla_direction": "91.44",
  "latitude": "31.633333",
  "longitude": "-8.000000",
  "address": "",
  "city": "Marrakesh",
  "state": "Marrakesh-Tensift-Al Haouz",
  "postal_code": "",
  "country": "Morocco",
  "country_code": "MA",
  "items": [
    {
      "date_for": "2019-12-14",
      "fajr": "7:56 am",
      "shurooq": "9:15 am",
      "dhuhr": "2:26 pm",
      "asr": "5:11 pm",
      "maghrib": "7:37 pm",
      "isha": "8:51 pm"
    }
  ],
  "status_valid": 1,
  "status_code": 1,
  "status_description": "Success."
}
```

localhost:8888/muslimsalat/rabat/weekly/1.json

```
{
  "query": "rabat",
  "for": "weekly",
  "method": "1",
  "prayer_method_name": "Egyptian General Authority of Su",
  "daylight": "1",
  "timezone": "1",
  "map_image": "https://maps.google.com/maps/api/staticma",
  "sealevel": "72",
  "today_weather": {
    "pressure": "1024",
    "temperature": "13"
  },
  "link": "http://muslimsalat.com/rabat",
  "qibla_direction": "94.66",
  "latitude": "34.015049",
  "longitude": "-6.832720",
  "address": "",
  "city": "Rabat",
  "state": "Rabat-Sale-Zemmour-Zaer",
  "postal_code": "",
  "country": "Morocco",
  "country_code": "MA",
  "items": [
    {
      "date_for": "2019-12-14",
      "fajr": "7:45 am",
      "shurooq": "9:18 am",
      "dhuhr": "2:21 pm",
      "asr": "5:01 pm",
      "maghrib": "7:25 pm",
      "isha": "8:48 pm"
    }
  ],
  // 7 items
  // 7 items
  // 7 items
  // 7 items
  // 7 items
  // 7 items
}
```

localhost:8888/restcountries/all

```
{ ... }, // 22 items
{ ... }, // 22 items
{
  "name": "Morocco",
  "topLevelDomain": [
    ".ma"
  ],
  "alpha2Code": "MA",
  "alpha3Code": "MAR"
  "callingCodes": [
    "212"
  ],
  "capital": "Rabat",
  "altSpellings": [
    "MA",
    "Kingdom of Mor",
    "Al-Mamlakah al"
  ],
  "region": "Africa",
  "subregion": "North",
  "population": 33337
  "latlng": [
    32,
    -5
  ],
  "demonym": "Morocca",
  "area": 446550,
  "gini": 40.9,
  "timezones": [
    "UTC"
  ],
  "borders": [
    "DZA",
    "ESH",
    "ESP"
  ],
  "nativeName": "المغرب",
  "numericCode": "504"
}
```

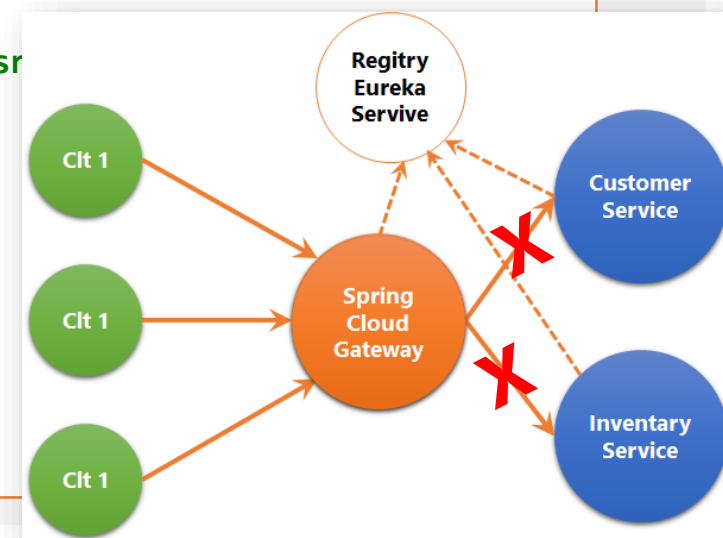
localhost:8888/restcountries/region/africa

```
{
  "name": "Algeria",
  "topLevelDomain": [
    ".dz"
  ],
  "alpha2Code": "DZ",
  "alpha3Code": "DZA",
  "callingCodes": [
    "213"
  ],
  "capital": "Algiers",
  "altSpellings": [
    "DZ",
    "Dzayer",
    "Algérie"
  ],
  "region": "Africa",
  "subregion": "Northern Africa",
  "population": 39500000,
  "latlng": [
    28,
    3
  ],
  "demonym": "Algerian",
  "area": 2381741,
  "gini": 35.3,
  "timezones": [
    "UTC+01:00"
  ],
  "borders": [
    "TUN",
    "LBY",
    "NER",
    "ESH",
    "MRT"
  ]
}
```


Circuit Breaker avec Hystrix

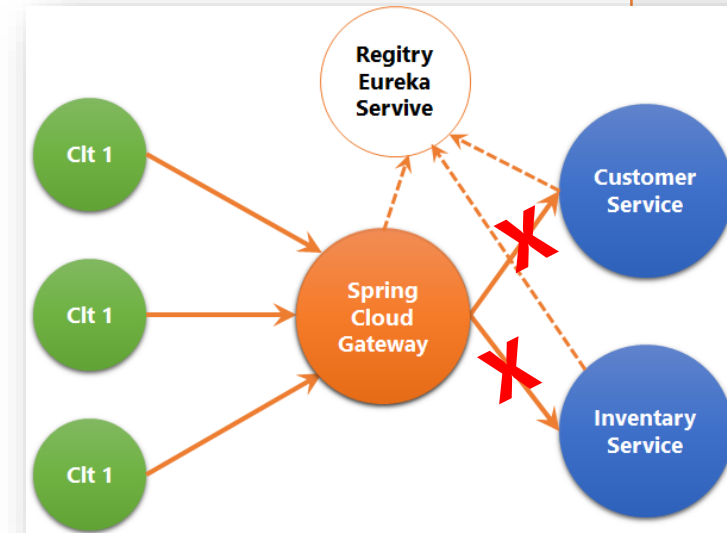
@Bean

```
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){  
    return builder.routes()  
        .route(r->r.path("/restcountries/**")  
            .filters(f->f  
                .addRequestHeader("x-rapidapi-host", "restcountries-v1.p.rapidapi.com")  
                .addRequestHeader("x-rapidapi-key", "fe5e774996msh4eb6e863d457420p1d2ffbjsnee0617ac5078")  
                .rewritePath("/restcountries/(?<segment>.*)", "/${segment}")  
                .hystrix(h->h.setName("rest-countries")  
                    .setFallbackUri("forward:/restCountriesFallback"))  
            )  
        .uri("https://restcountries-v1.p.rapidapi.com").id("countries")  
        .route(r->r.path("/muslimsalat/**")  
            .filters(f->f  
                .addRequestHeader("x-rapidapi-host", "muslimsalat.p.rapidapi.com")  
                .addRequestHeader("x-rapidapi-key", "fe5e774996msh4eb6e863d457420p1d2ffbjsnee0617ac5078")  
                .rewritePath("/muslimsalat/(?<segment>.*)", "/${segment}")  
                .hystrix(h->h.setName("muslimsalat")  
                    .setFallbackUri("forward:/muslimsalatFallback"))  
            )  
        .uri("https://muslimsalat.p.rapidapi.com").id("countries")  
    )  
    .build();  
}
```



Circuit Breaker avec Hystrix

```
@RestController
class FallBackRestController{
    @GetMapping("/restCountriesFallback")
    public Map<String,String> restCountriesFallback(){
        Map<String,String> map=new HashMap<>();
        map.put("message","Default Rest Countries Fallback service");
        map.put("countries","Algeria, Morocco");
        return map;
    }
    @GetMapping("/muslimsalatFallback")
    public Map<String,String> muslimsalatback(){
        Map<String,String> map=new HashMap<>();
        map.put("message","Default Muslim Fallback service");
        map.put("Fajr","07:00");
        map.put("DOHR","14:00");
        return map;
    }
}
```



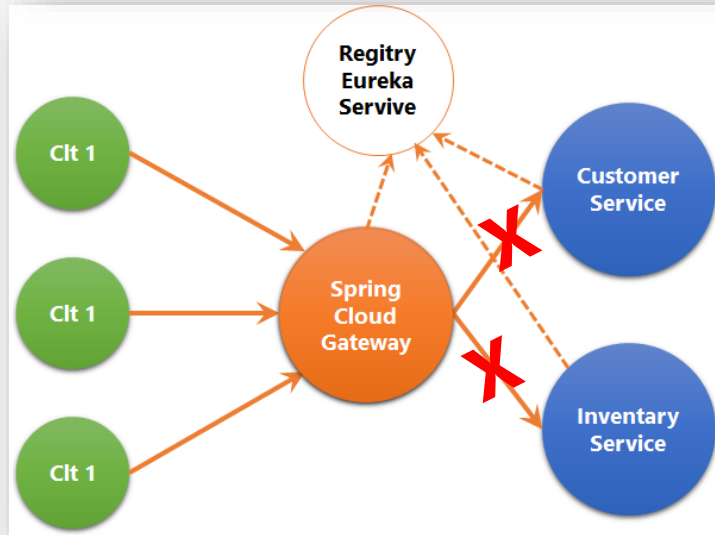
application.properties

`management.endpoints.web.exposure.include=hystrix.stream`

`hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=1000`

Circuit Breaker With Hystrix

```
Hystrix Monitor x localhost
localhost:8888/actuator/hystrix.stream
data:{"type":"ping"}
data:{"type":"ping"}
data:{"type":"ping"}
data:{"type":"ping"}
data:{"type":"ping"}
data:{"type":"ping"}
```



```
localhost:8888/restcountries/all
{
  "countries": "Algeria, Morocco",
  "message": "Default Rest Countries Fallback service"
}
```

```
localhost:8888/actuator/hystrix.stream
data:{"type":"ping"}
data:{"type":"HystrixCommand","name":"rest-countries","group":"HystrixGatewayFilterFactor
itBreakerOpen":false,"errorPercentage":0,"erro
tBadRequests":0,"rollingCountCollapsedRequest
xceptionsThrown":0,"rollingCountFailure":0,"r
FallbackFailure":0,"rollingCountFallbackMissi
"rollingCountFallbackSuccess":0,"rollingCount
horeRejected":0,"rollingCountShortCircuited":
hreadPoolRejected":0,"rollingCountTimeout":0,
lingMaxConcurrentExecutionCount":0,"latencyEx
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99
":0,"latencyTotal":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99
cuitBreakerRequestVolumeThreshold":20,"proper
seconds":5000,"propertyValue_circuitBreakerEn
e_circuitBreakerForceOpen":false,"propertyVal
propertyValue_circuitBreakerEnabled":true,"prope
MAPHORE","propertyValue_executionIsolationThr
```

```
localhost:8888/restcountries/all
[
  {
    "name": "Afghanistan",
    "topLevelDomain": [
      ".af"
    ],
    "alpha2Code": "AF",
    "alpha3Code": "AFG",
    "callingCodes": [
      "93"
    ],
    "capital": "Kabul",
    "altSpellings": [
      "AF",
      "Afgānistān"
    ],
    "area": 652230,
    "population": 32946012,
    "gdp": 17045000000,
    "currency": "AFN",
    "languages": "Pashto, Dari"
  }
]
```

Hystrix Dashboard

Spring Initializr

Bootstrap your application

Project

Language

Spring Boot

Project Metadata

Dependencies

Maven Project

Gradle Project

Java

Kotlin

Groovy

2.2.3 (SNAPSHOT)

2.2.2

2.1.12 (SNAPSHOT)

2.1.11

Group

org.sid

Artifact

hystrix-dashboard

> Options

🔍

☰

1 selected

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

Hystrix Dashboard

Circuit breaker dashboard with Spring Cloud Netflix Hystrix.

✓

Circuit Breaker avec Hystrix

```
package org.sid.hystrixdashboard;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;

@SpringBootApplication
@EnableHystrixDashboard
public class HystrixDashboardApplication {

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


}
```

application.properties

server.port=9999

Circuit Breaker avec Hystrix

localhost:9999/hystrix



Hystrix Dashboard

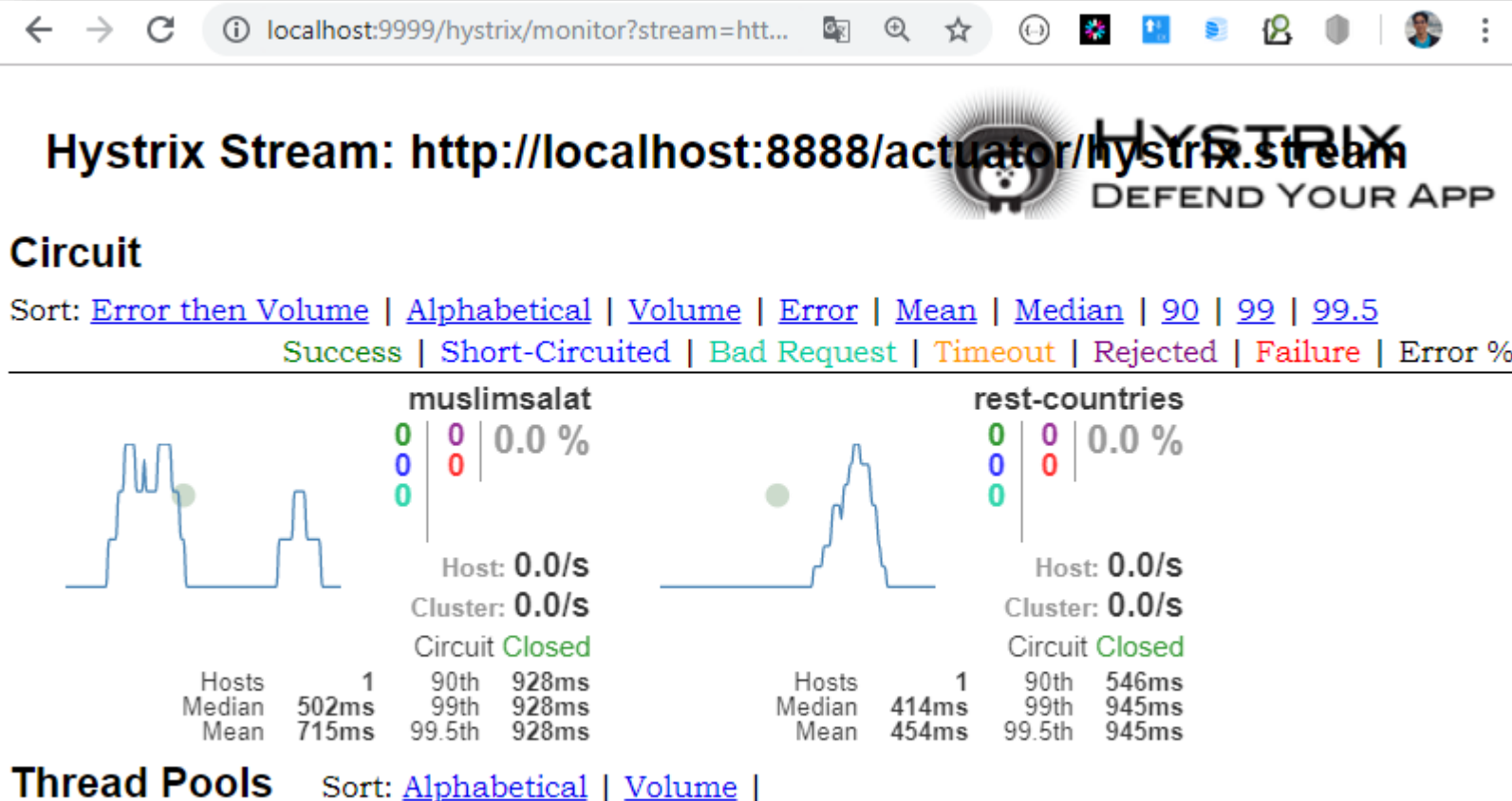
<http://localhost:8888/actuator/hystrix.stream>

Cluster via Turbine (default cluster): <https://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [https://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](https://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <https://hystrix-app:port/actuator/hystrix.stream>

Delay: ms Title:

Circuit Breaker avec Hystrix

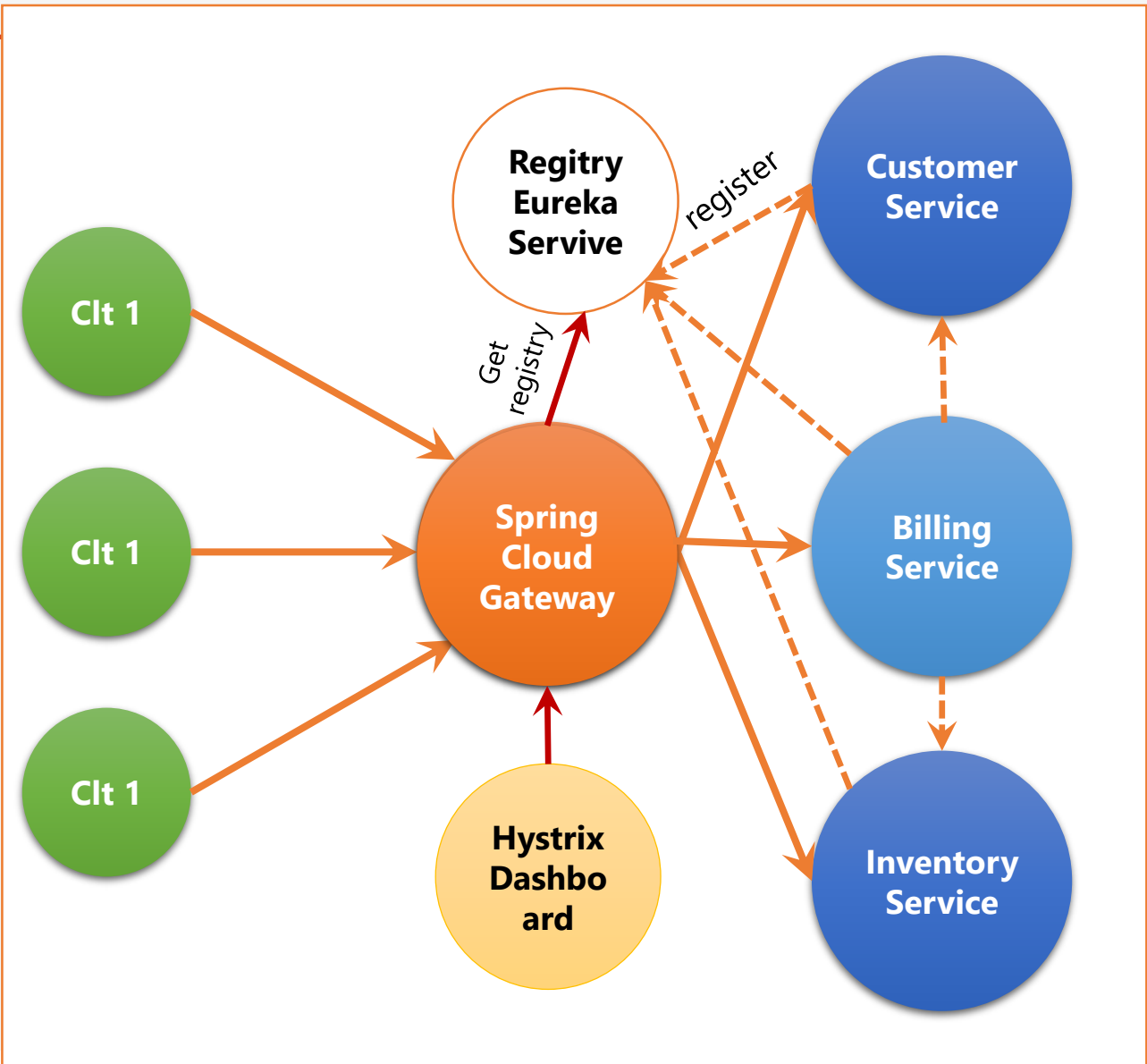
```
localhost:8888/muslimsalat/rabat/5.json  
  
{  
  "Fajr": "07:00",  
  "DOHR": "14:00",  
  "message": "Default Muslim Fallback service"  
}
```



```
localhost:8888/muslimsalat/rabat/5.json  
  
,  
  "link": "http://muslimsalat.com/rabat",  
  "qibla_direction": "94.66",  
  "latitude": "34.015049",  
  "longitude": "-6.832720",  
  "address": "",  
  "city": "Rabat",  
  "state": "Rabat-Sale-Zemmour-Zaer",  
  "postal_code": "",  
  "country": "Morocco",  
  "country_code": "MA",  
  "items": [  
    {  
      "date_for": "2019-12-17",  
      "fajr": "7:56 am",  
      "shurooq": "9:20 am",  
      "dhuhr": "2:23 pm",  
      "asr": "5:02 pm",  
      "maghrib": "7:25 pm",  
      "isha": "8:44 pm"  
    }  
  ]  
}
```

Communication REST entre les micro-services : Declarative Rest Client avec Spring Cloud Feign

- Feign est un Framework, introduite dans Spring cloud, qui permet de créer facilement un Client REST d'une manière déclarative.
- Feign peut être utilisée à la place de RestTemplate pour interagir avec d'autres services distants via des API Restful.
- Dans Notre cas, nous allons ajouter un autre service de facturation qui a besoin de communiquer avec els services d'inventaires et le service client pour récupérer les informations sur le client et les produits d'une facture



Billing-service

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

    @Bean
    CommandLineRunner start(BillRepository billRepository, ProductItemRepository productItemRepository,
        InventoryServiceClient inventoryServiceClient, CustomerServiceClient customerServiceClient){
        return args -> {
            Bill bill=new Bill();
            bill.setBillingDate(new Date());
            Customer customer=customerServiceClient.findCustomerById(1L);
            bill.setCustomerID(customer.getId());
            billRepository.save(bill);
            inventoryServiceClient.findAll().getContent().forEach(p->{
productItemRepository.save(new ProductItem(null,null,p.getId(),p.getPrice(),(int)(1+Math.random()*1000),bill));
            });
        };
    }
}
```

Billing-service

start.spring.io

Spring Initializr

Bootstrap your application

Project

Maven ProjectGradle Project

Language

JavaKotlinGroovy

Spring Boot

2.2.3 (SNAPSHOT)2.2.22.1.12 (SNAPSHOT)2.1.10

Project Metadata

Group

org.sid

Artifact

billing-service

> Options

Dependencies

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Generate - Ctrl + G

Explore - Ctrl + Space

© 2013-2019 Pivotal Software

start.spring.io is powered by

Initializr and Pivotal Web Services

Selected dependencies

- **Spring Web** : Build web, including RESTful, applications using Spring MVC. 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.
- **OpenFeign** : Declarative REST Client. OpenFeign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.
- **Spring HATEOAS** : Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC.

Billing-service

```
package org.sid.billingervice;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;import lombok.Data; import lombok.NoArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.hateoas.PagedModel;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import javax.persistence.*;import java.util.Collection; import java.util.Date;import java.util.List;
```

Billing-service

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
class Bill{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date billingDate;
    @Transient
    @OneToMany(mappedBy = "bill")
    private Collection<ProductItem> productItems;
    @Transient
    private Customer customer;
    private long customerID;
}

@RepositoryRestResource
interface BillRepository extends JpaRepository<Bill,Long>
{

}
```

```
@Entity @Data @NoArgsConstructor @AllArgsConstructor
class ProductItem{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Transient
    private Product product;
    private long productID;
    private double price;
    private double quantity;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Bill bill;
}

@RepositoryRestResource
interface ProductItemRepository extends
JpaRepository<ProductItem,Long>{
    List<ProductItem> findByBillId(Long billID);
}
```

Billing-service

```
@Data
class Product{
    private Long id;
    private String name;
    private double price;
}
```

```
@Data
class Customer{
    private Long id;
    private String name;
    private String email;
}
```

```
@FeignClient(name="customer-service")
interface CustomerServiceClient{
    @GetMapping("/customers/{id}?projection=fullCustomer")
    Customer findCustomerById(@PathVariable("id") Long id);
}

@FeignClient(name="inventory-service")
interface InventoryServiceClient{
    @GetMapping("/products/{id}?projection=fullProduct")
    Product findProductById(@PathVariable("id") Long id);
    @GetMapping("/products?projection=fullProduct")
    PagedModel<Product> findAll();
}
```

Billing-service

@RestController

```
class BillRestController{

    @Autowired private BillRepository billRepository;
    @Autowired private ProductItemRepository productItemRepository;
    @Autowired private CustomerServiceClient customerServiceClient;
    @Autowired private InventoryServiceClient inventoryServiceClient;
    @GetMapping("/bills/full/{id}")
    Bill getBill(@PathVariable(name="id") Long id){
        Bill bill=billRepository.findById(id).get();
        bill.setCustomer(customerServiceClient.findCustomerById(bill.getCustomerID()));
        bill.setProductItems(productItemRepository.findByBillId(id));
        bill.getProductItems().forEach(pi->{
            pi.setProduct(inventoryServiceClient.findProductById(pi.getProductID()));
        });
        return bill; }
}
```

Billing-service

localhost:8083/bills/full/1

```
{
  "id": 1,
  "billingDate": "2019-12-18T12:20:18.458+0000",
  "productItems": [
    {
      "id": 1,
      "product": {
        "id": 1,
        "name": "Computer Desk Top HP",
        "price": 900
      },
      "productID": 1,
      "price": 900,
      "quantity": 332
    },
    { ... }, // 5 items
    { ... } // 5 items
  ],
  "customer": {
    "id": 1,
    "name": "Enset",
    "email": "contact@enset-media.ma"
  },
  "customerID": 1
}
```

localhost:8083/h2-console/login.do?jsessionid=e6b579158ed775ed7

Auto commit Max rows: 1000 Auto complete

jdbc:h2:mem:testdb

BILL
ID
BILLING_DATE
CUSTOMERID
Indexes
PRODUCT_ITEM
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM BILL

SELECT * FROM BILL;

ID	BILLING_DATE	CUSTOMERID
1	2019-12-18 12:20:18.458	1

localhost:8083/h2-console/login.do?jsessionid=bacf481fcb11c8353

Auto commit Max rows: 1000 Auto complete

jdbc:h2:mem:testdb

PRODUCT_ITEM
ID
PRICE
PRODUCTID
QUANTITY
BILL_ID
Indexes
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PRODUCT_ITEM

SELECT * FROM PRODUCT_ITEM;

ID	PRICE	PRODUCTID	QUANTITY	BILL_ID
1	980.0	1	5.0	1
2	980.0	2	5.0	1

(2 rows, 7 ms)

Edit