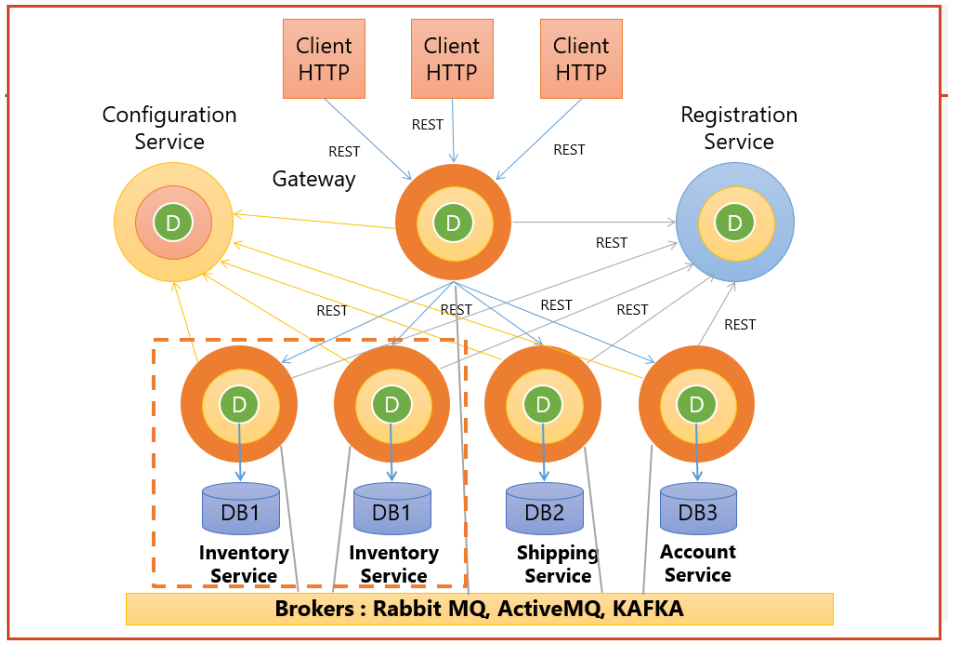
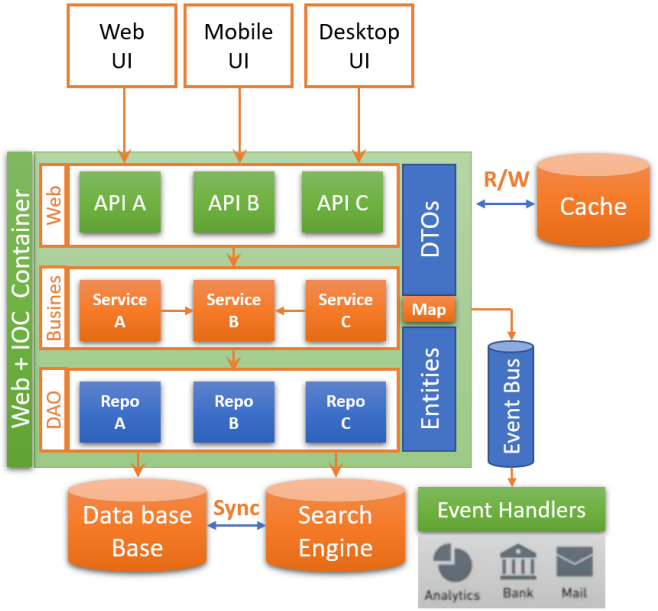
Systèmes Distribués Basés sur les Micro-services



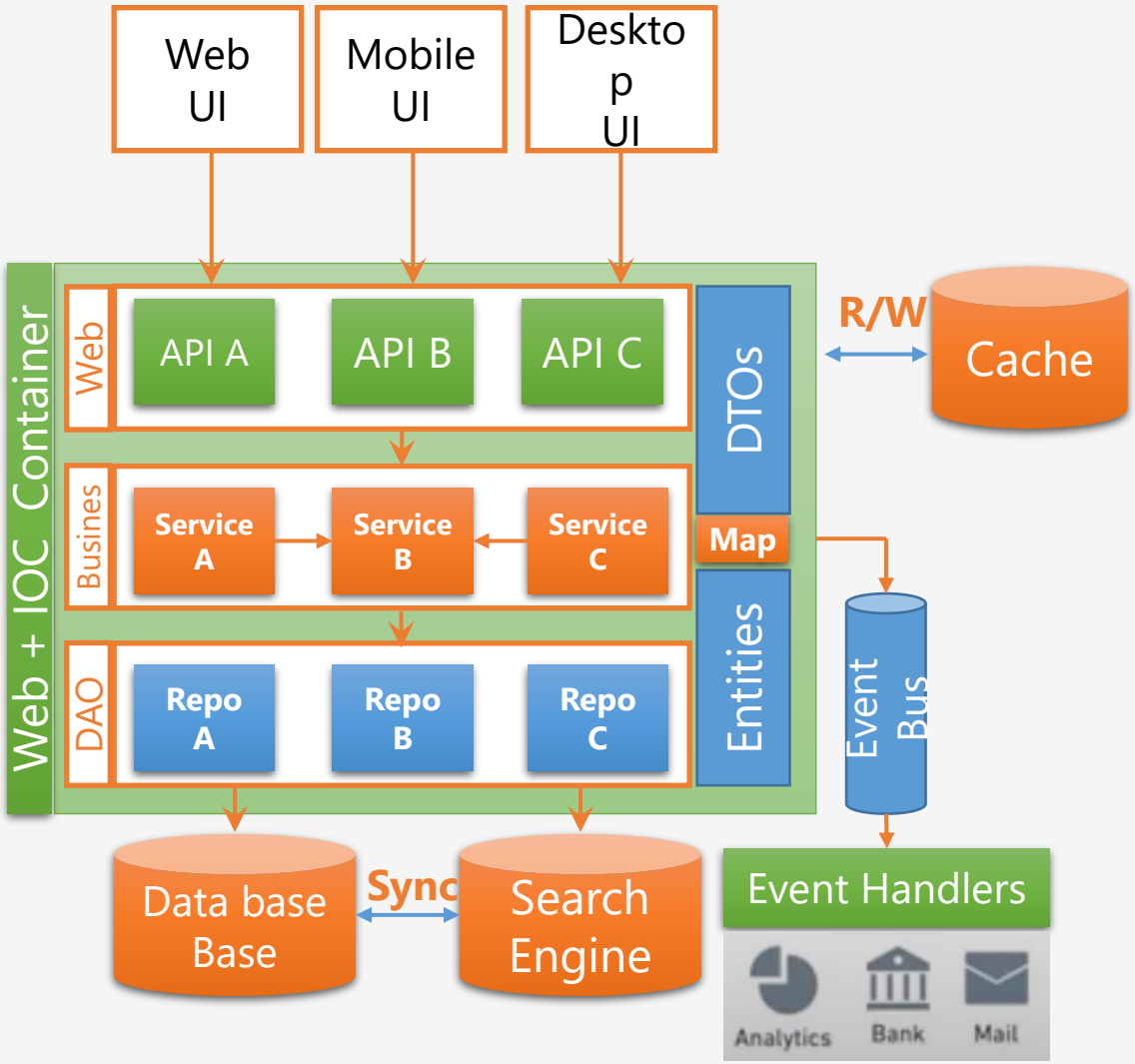
Spring Cloud

spring boot



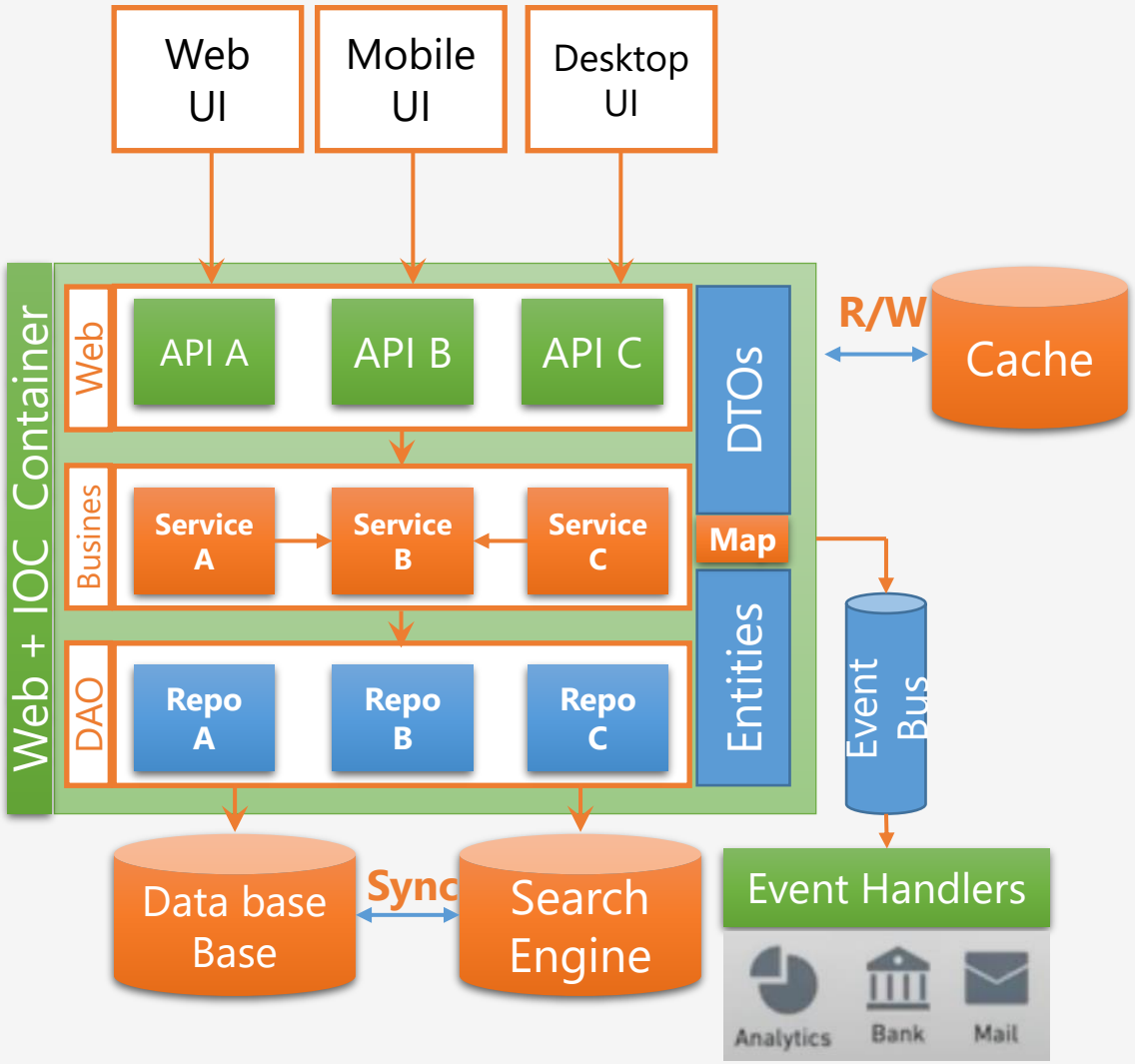
Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Université Hassan II de Casablanca,

Traditinnelle Architecture Monolithique



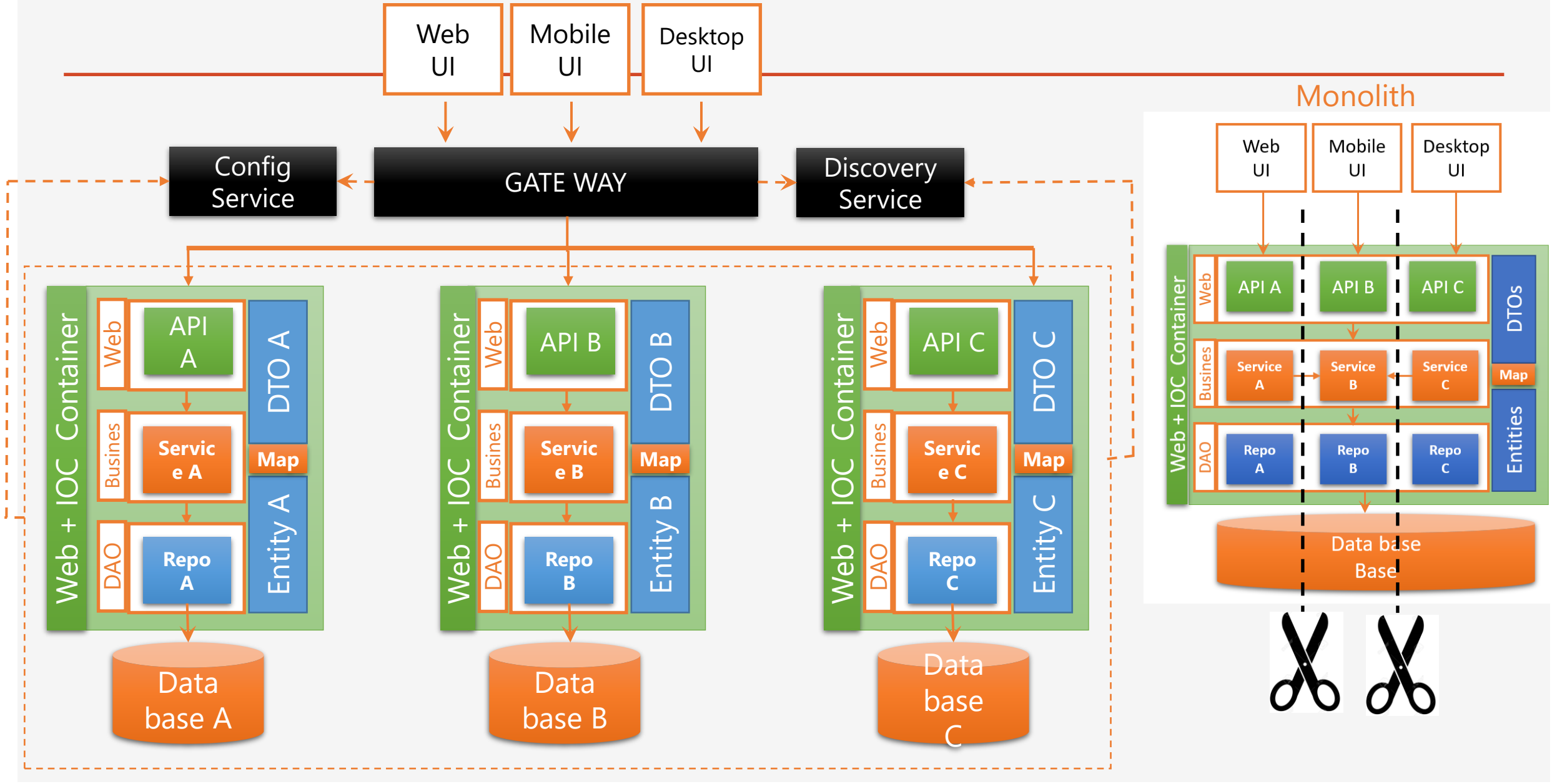
Une application monolithique est une application qui est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application

Contraintes principales des architectures Monolithiques :



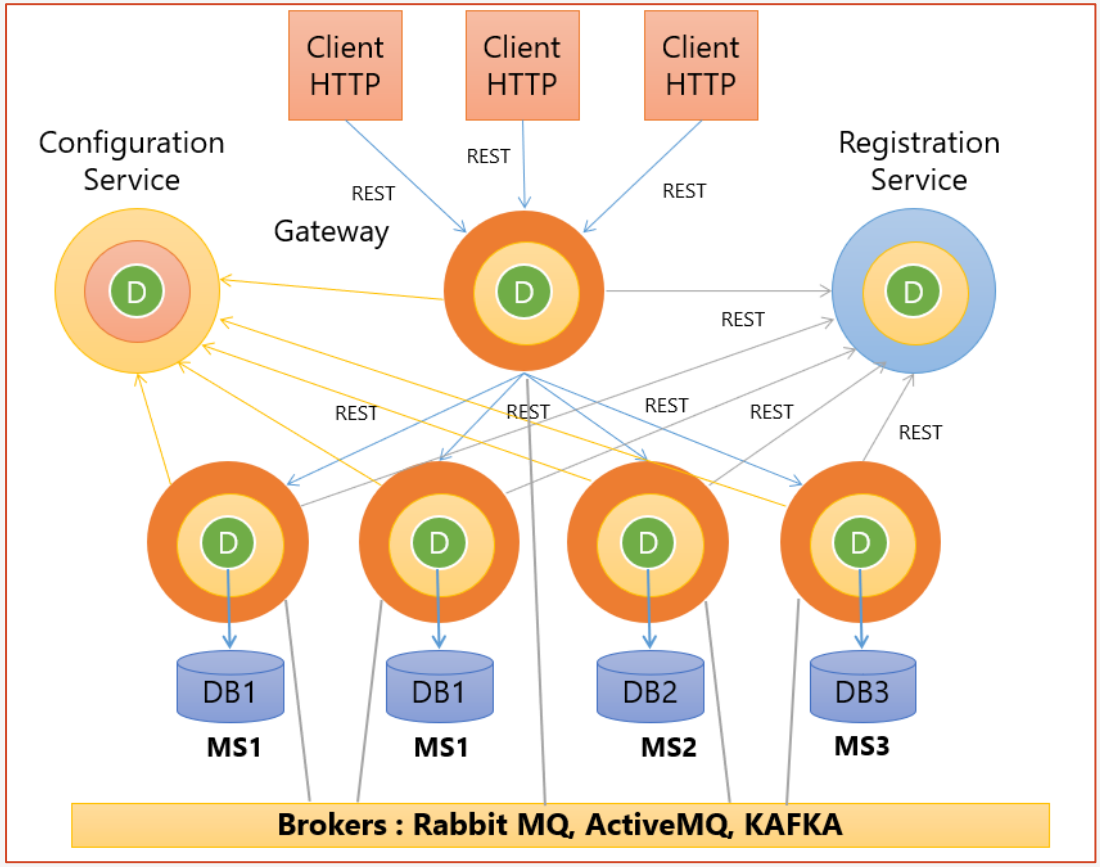
- Elles centralisent tous les besoins fonctionnels
- Elles sont réalisées dans une seule technologie.
- Chaque modification nécessite de :
  - Tester les régressions
  - Redéployer toute l'application
  - Difficile à faire évoluer au niveau fonctionnel
- Livraison en bloc (Le client attend beaucoup de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

Architecture Micro-Service



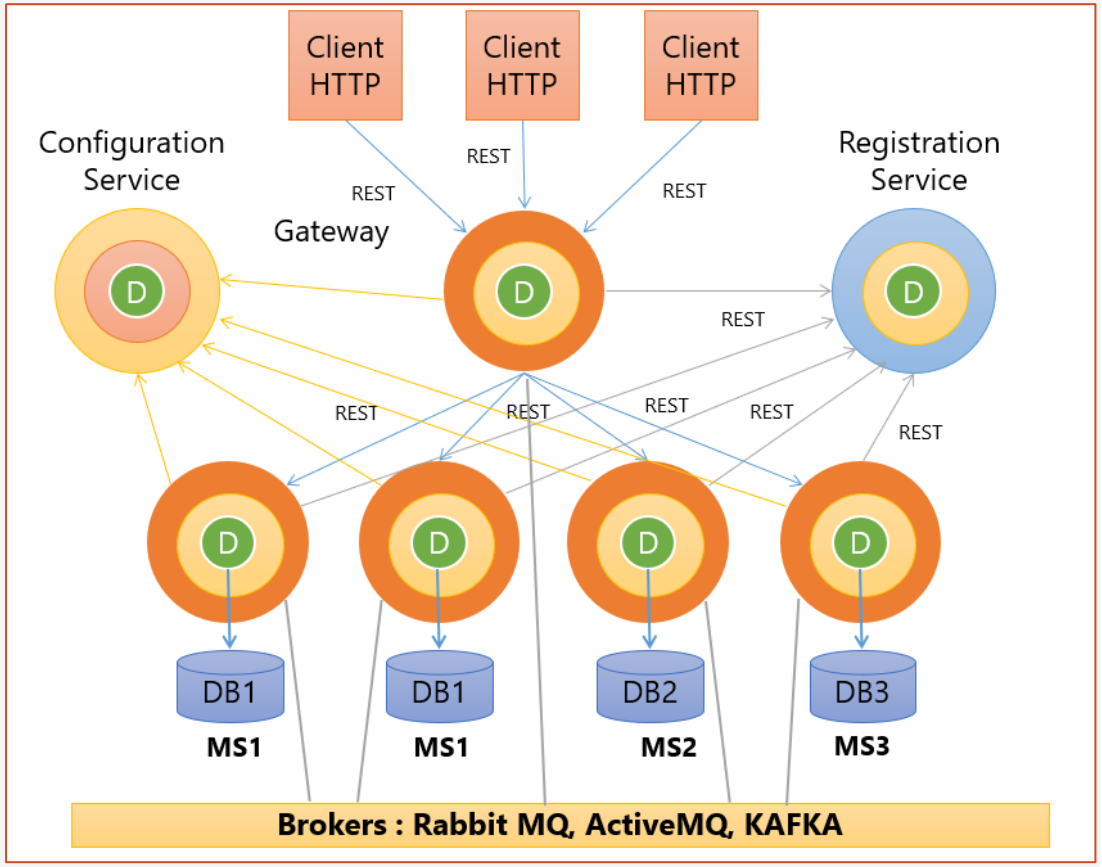
Approche Micro services

- 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, Pyton, …)
- 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**.



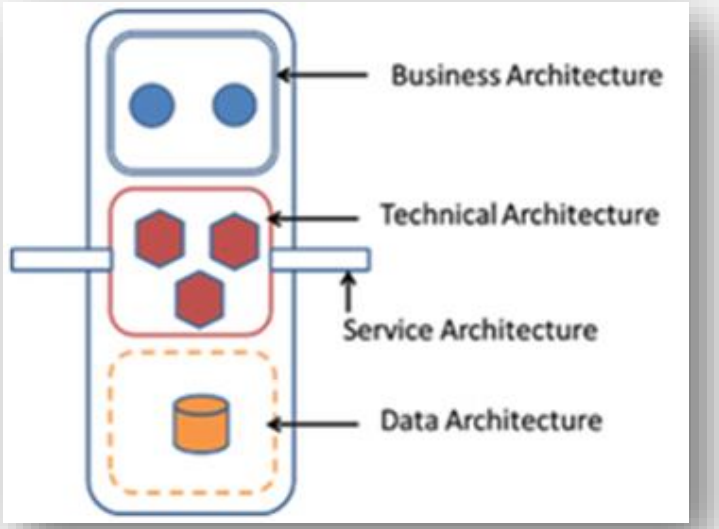
Approche Micro services

- Ils sont faiblement couplés puisque chaque micro service est physiquement séparé des autres,
- Indépendance relative entre les différentes équipes qui développement 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

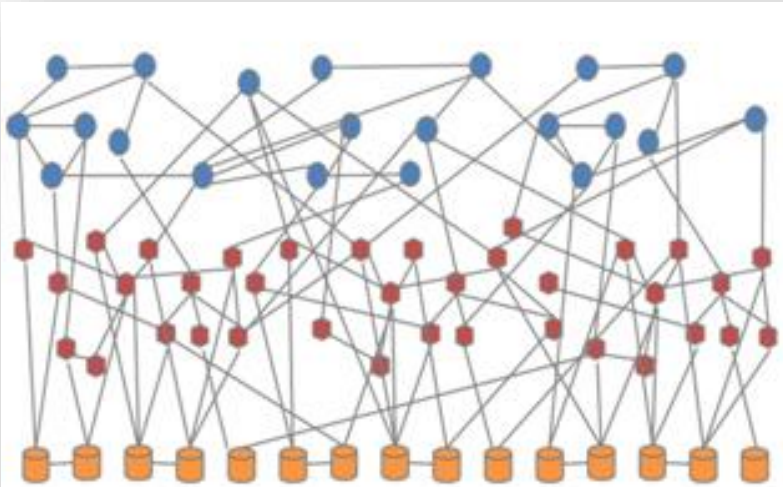
- Comme pour le cas d'une application monolithique, un micro service peut être composé de plusieurs très petites couches:
  - Couche DAO
  - Couche Métier,
  - Couches Techniques (REST, SOAP, RMI, JMS, AMQP, Sécurité, etc…)

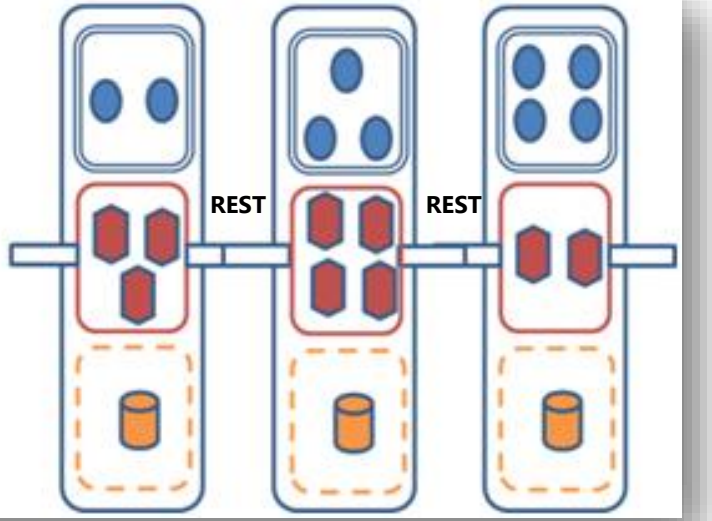


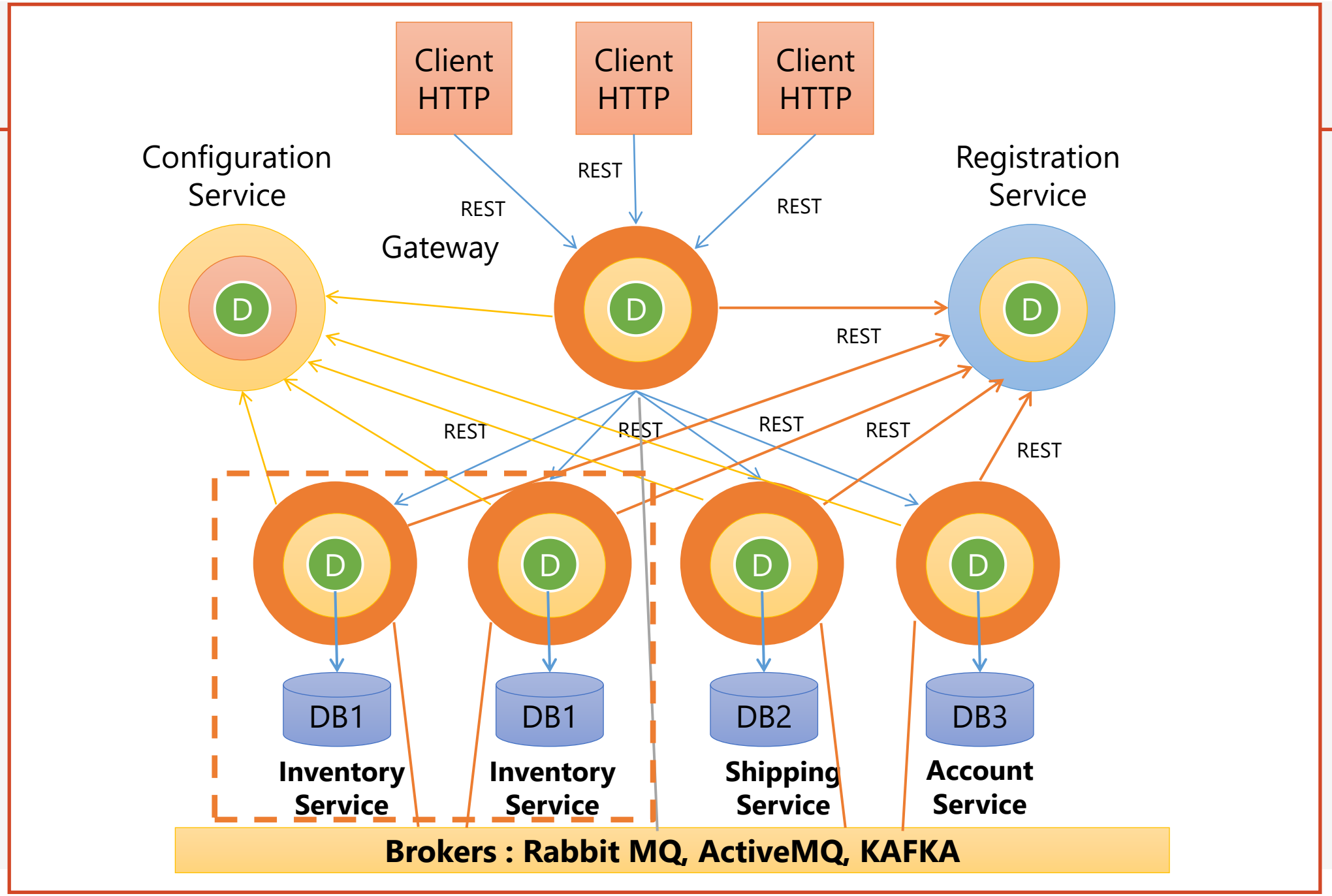
Client Service

Compte Service

Opération Service



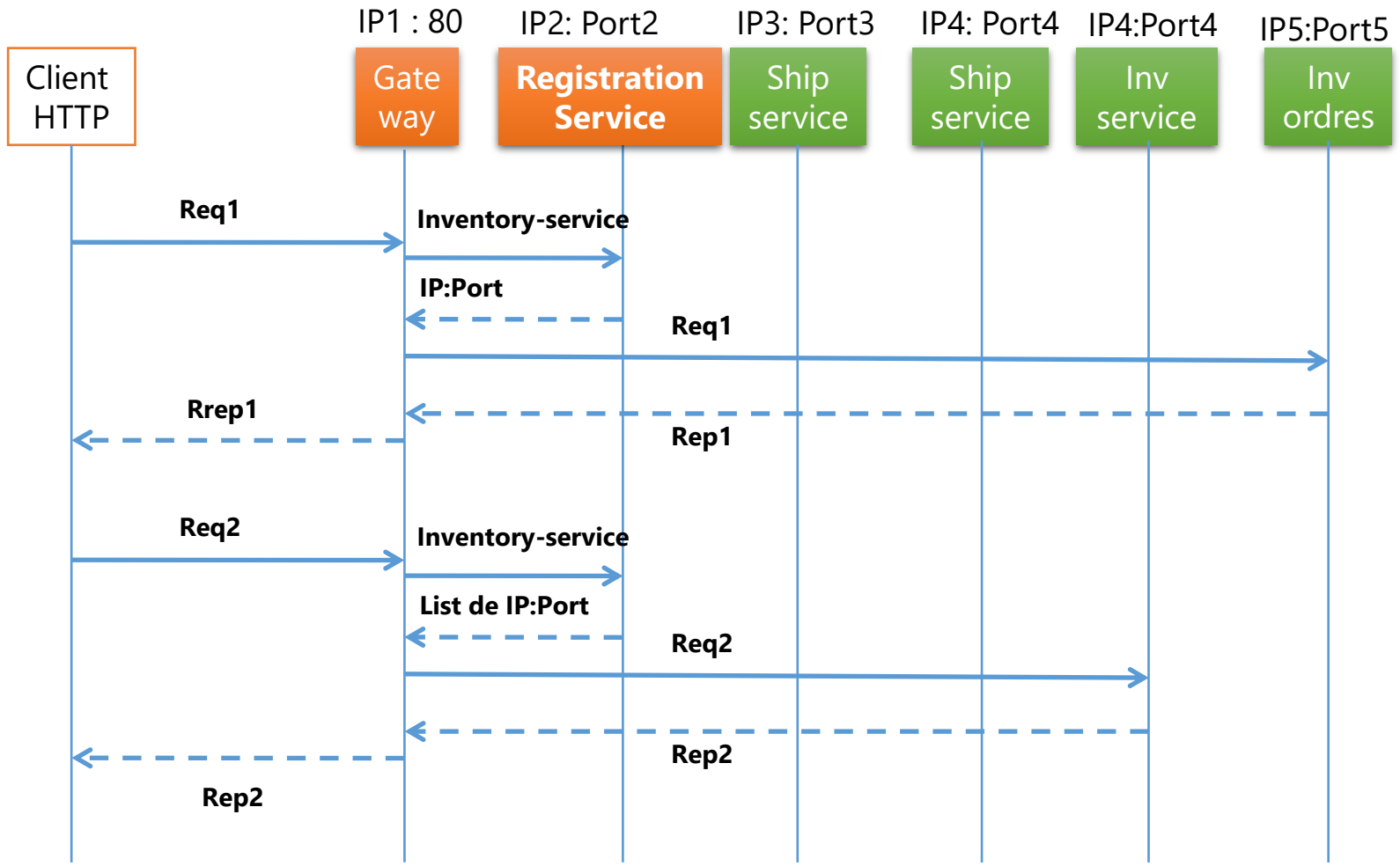



Client HTTP
Client HTTP
Client HTTP
Configuration Service
Gateway
Registration Service
REST
REST
REST
REST
REST
REST
REST
REST
REST
D
D
D
D
D
D
D
DB1
DB1
DB2
DB3
Inventory Service
Inventory Service
Shipping Service
Account Service
Brokers : Rabbit MQ, ActiveMQ, KAFKA

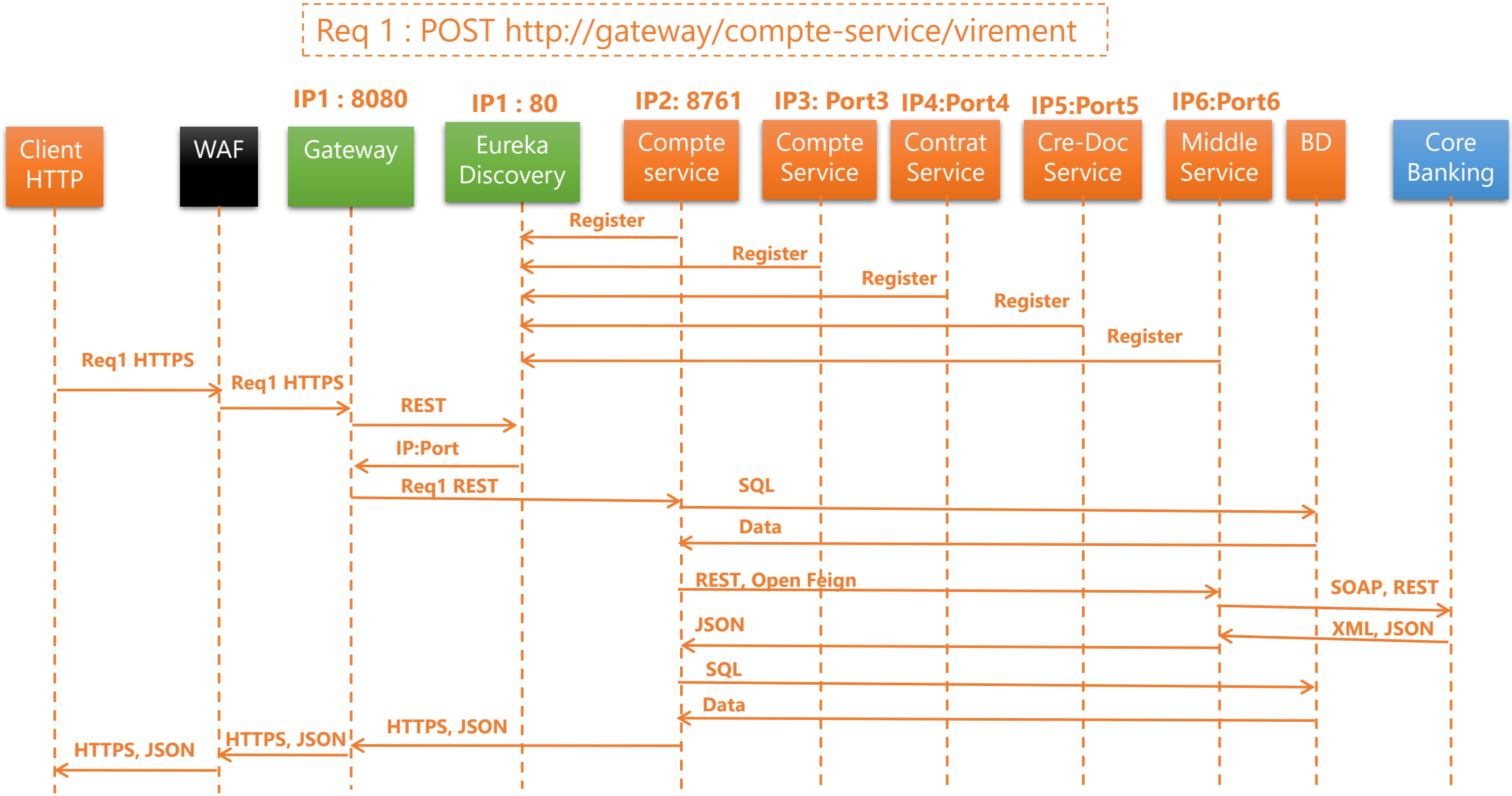
Consulter les services via le service proxy

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

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

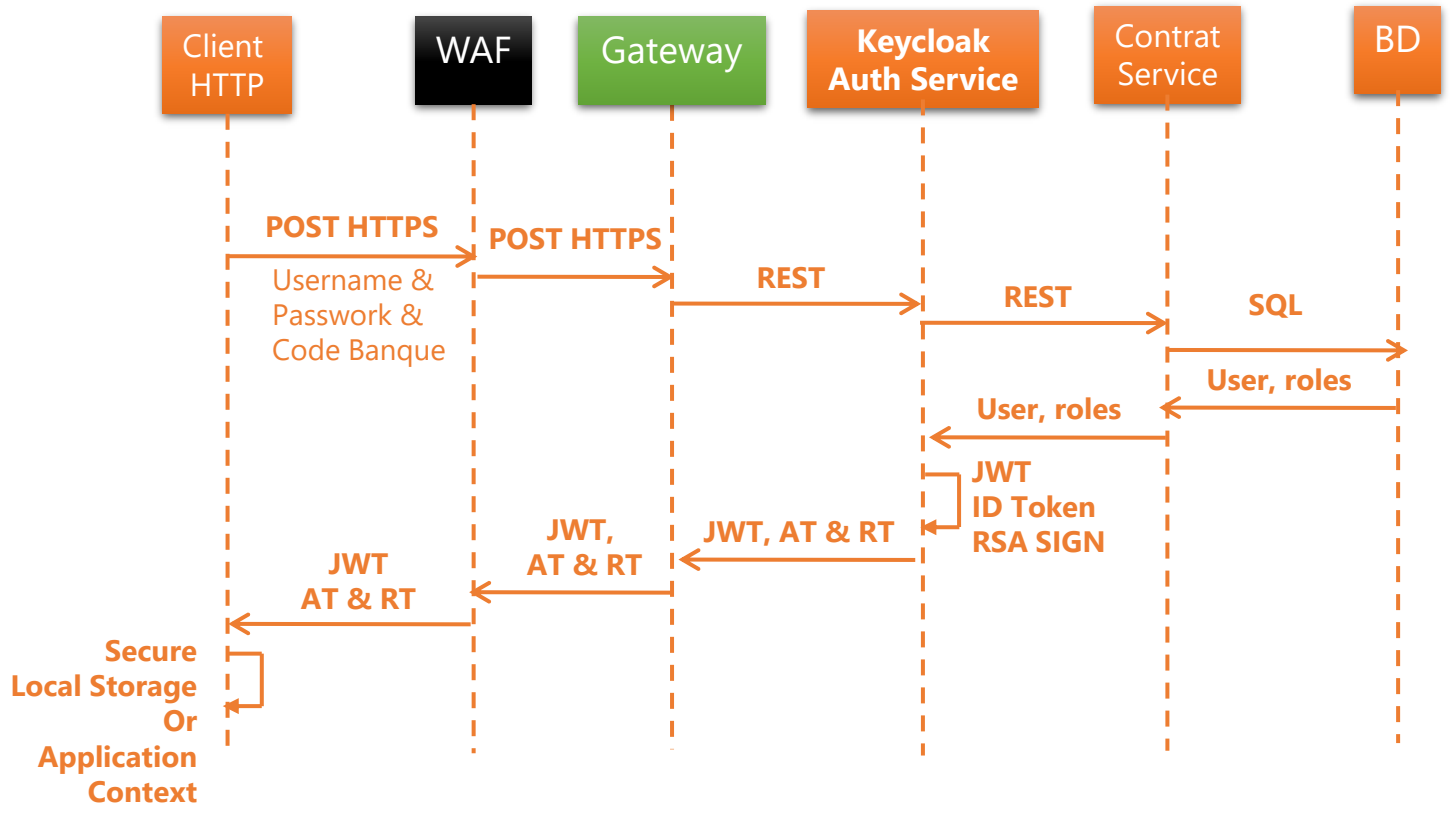


Consulter les services via le service proxy



Authentification

Req 1 : POST http://gateway/auth-service/auth



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 fichier 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

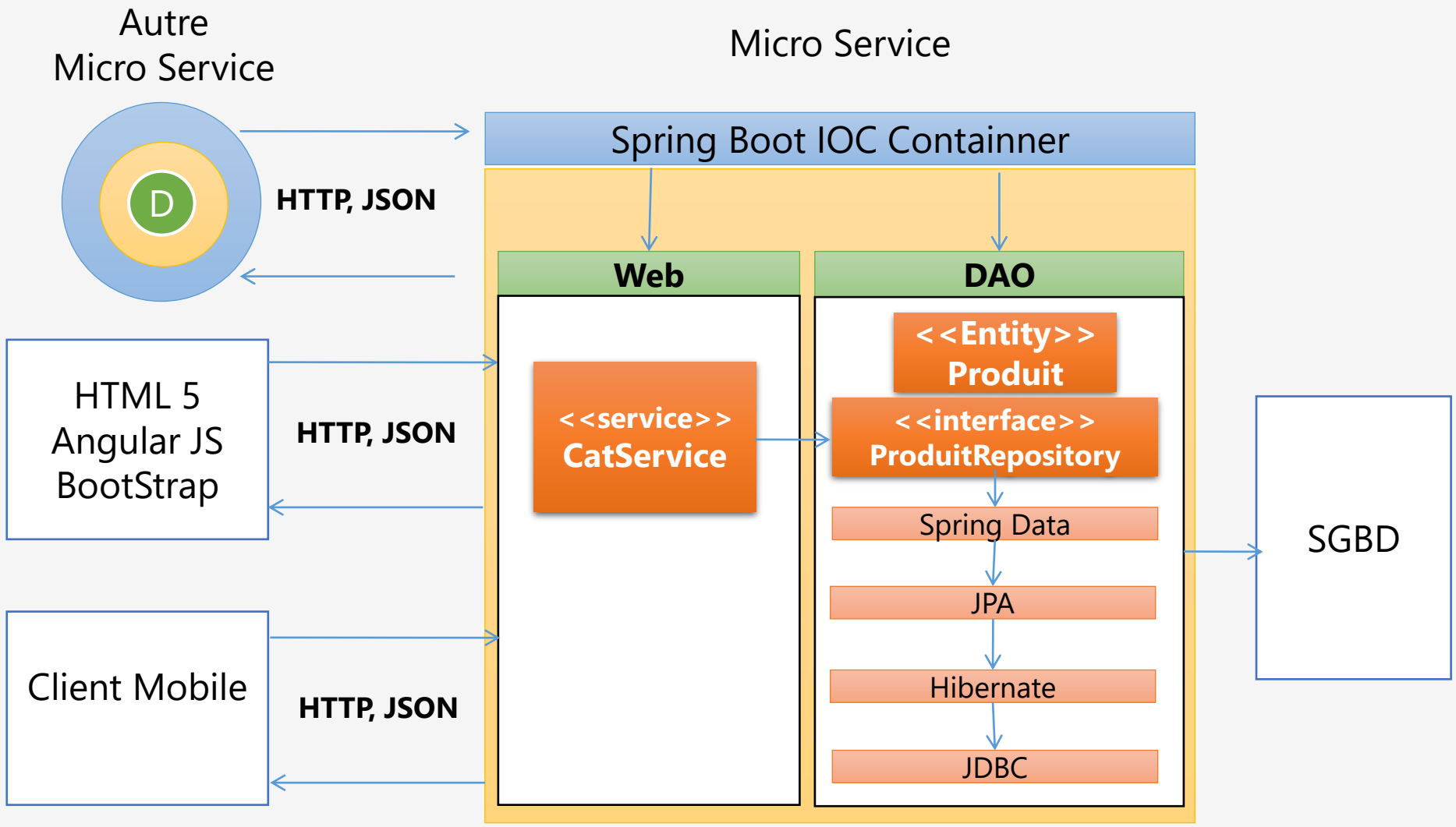
L'applications de permettre de :

- 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



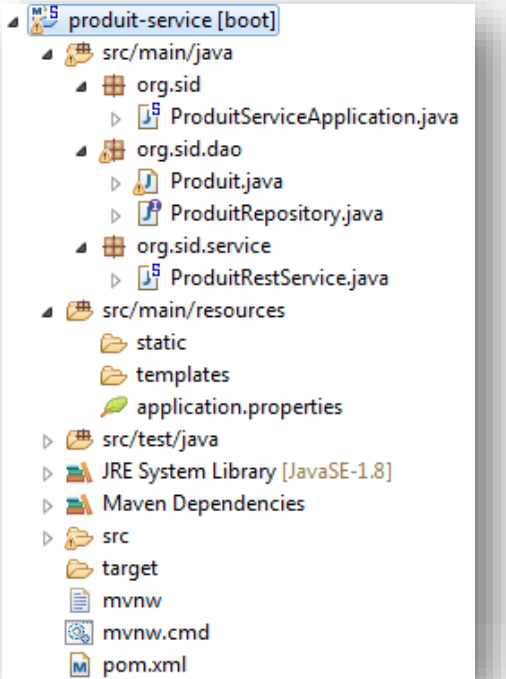
Micro Service : Spring Boot

Service Restful

```
@RestController
public class ProduitRestService {
    @Autowired private ProduitRepository produitRepository;
    @RequestMapping(value="/produits",method=RequestMethod.GET)
    public Lit<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 {
@Id @GeneratedValue
   private Long id;
   private String designation;
   private double prix;

}
```



Interface DAO basée sur Spring data

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

Application Spring Boot

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

Application.properties

```
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
```

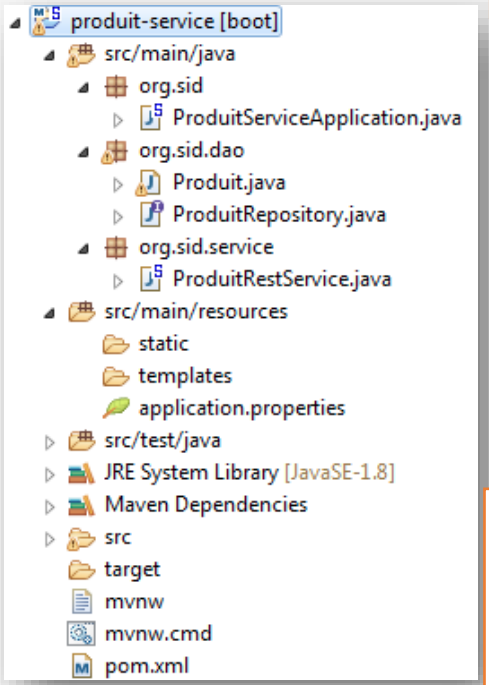
Micro Service : Spring Boot

Service Restful

```
@RestController
public class ProduitRestService {
    @Autowired
    private ProduitRepository produitRepository;
    @RequestMapping(value="/produits",method=RequestMethod.GET)
    public Lit<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> {
}
```

Application Spring Boot

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

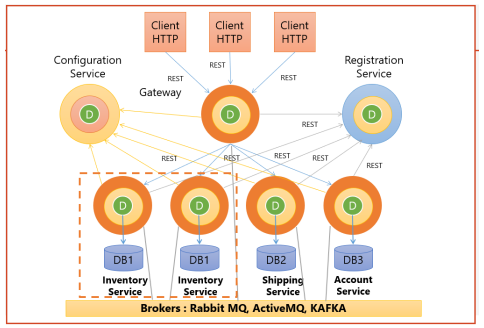
Application.properties

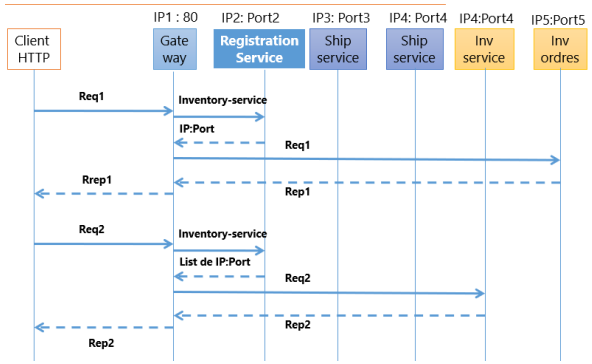
```
spring.datasource.url = jdbc:mysql://localhost:3306/prod-services
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect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

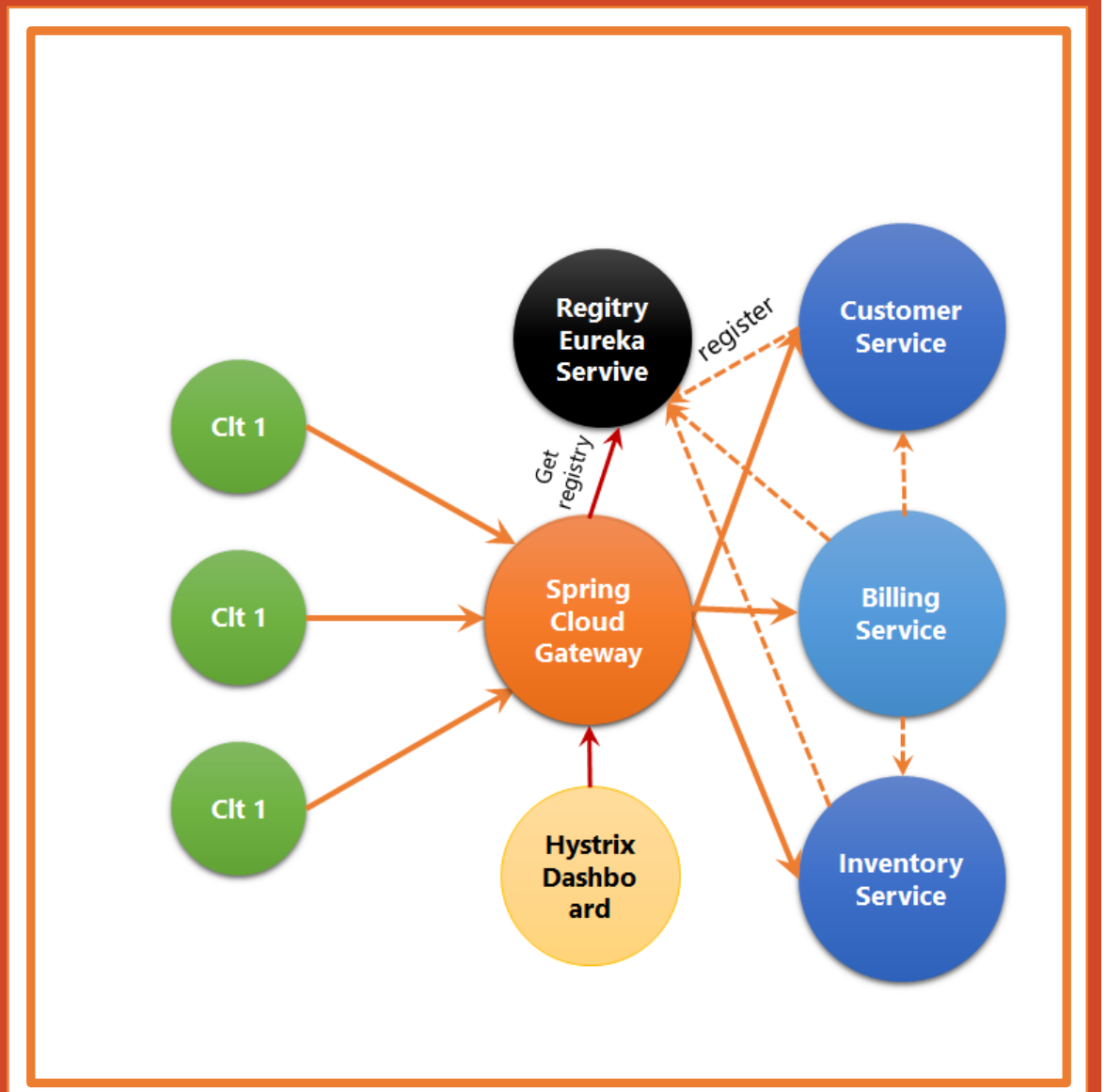
Architectures

Micro Services avec Spring Cloud

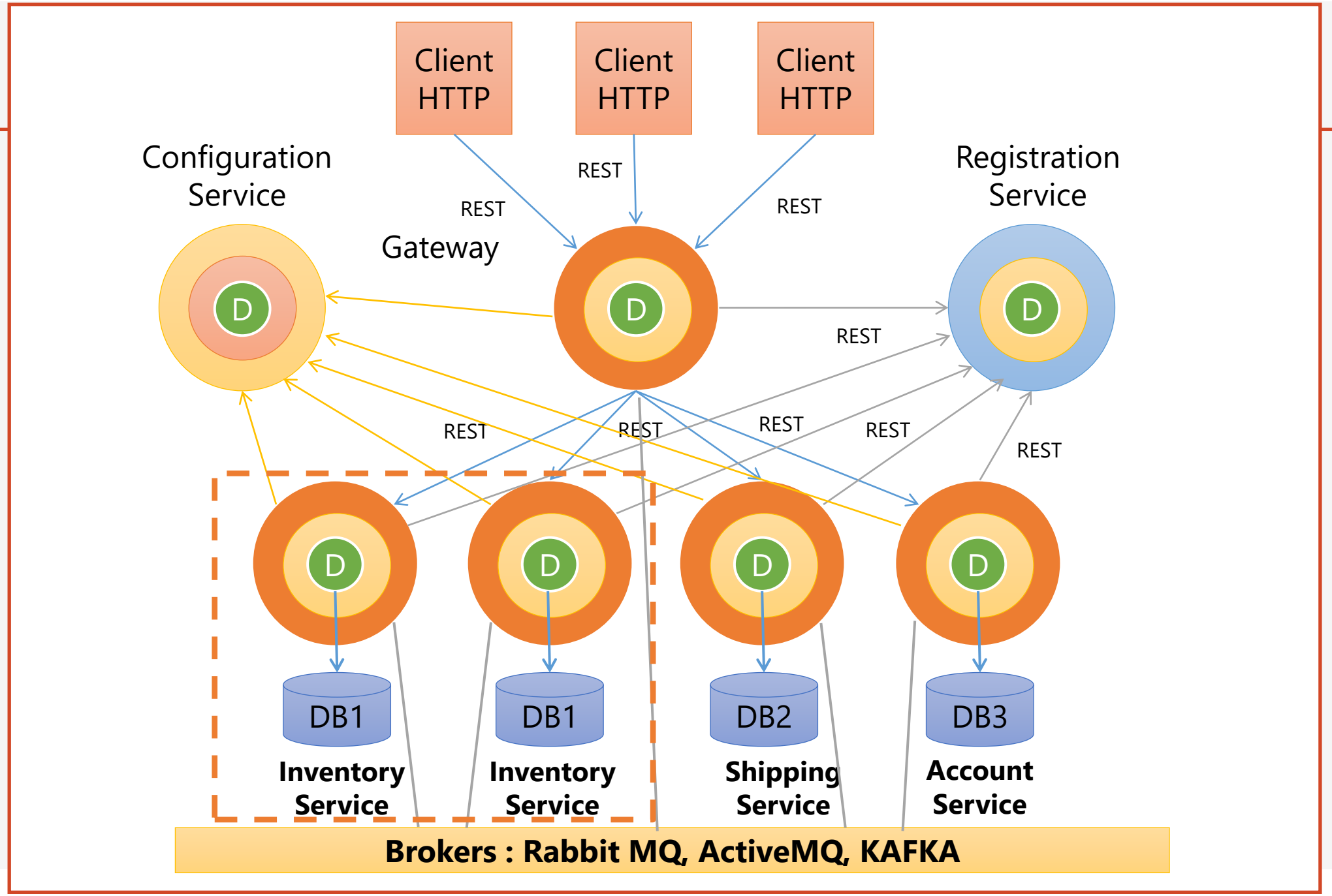
- Spring Cloud Gateway
- Eureka Discovery
- Open Feign Rest Client
- Hystrix DashBoard







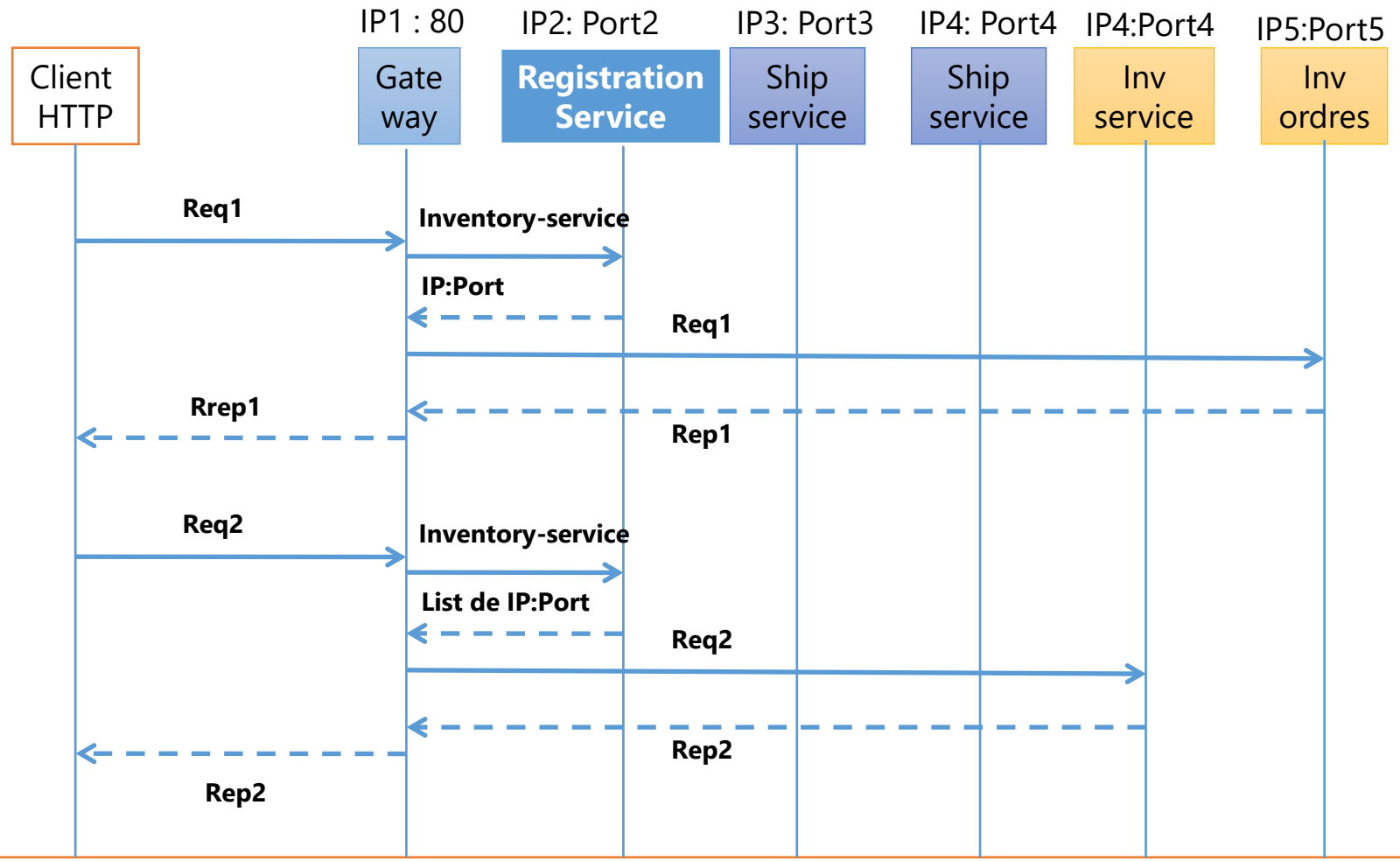
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications


Client HTTP
Client HTTP
Client HTTP
REST
REST
REST
Configuration Service
Gateway
Registration Service
D
REST
REST
REST
REST
REST
REST
DB1
DB1
DB2
DB3
Inventory Service
Inventory Service
Shipping Service
Account Service
Brokers : Rabbit MQ, ActiveMQ, KAFKA

Consulter les services via le service proxy

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

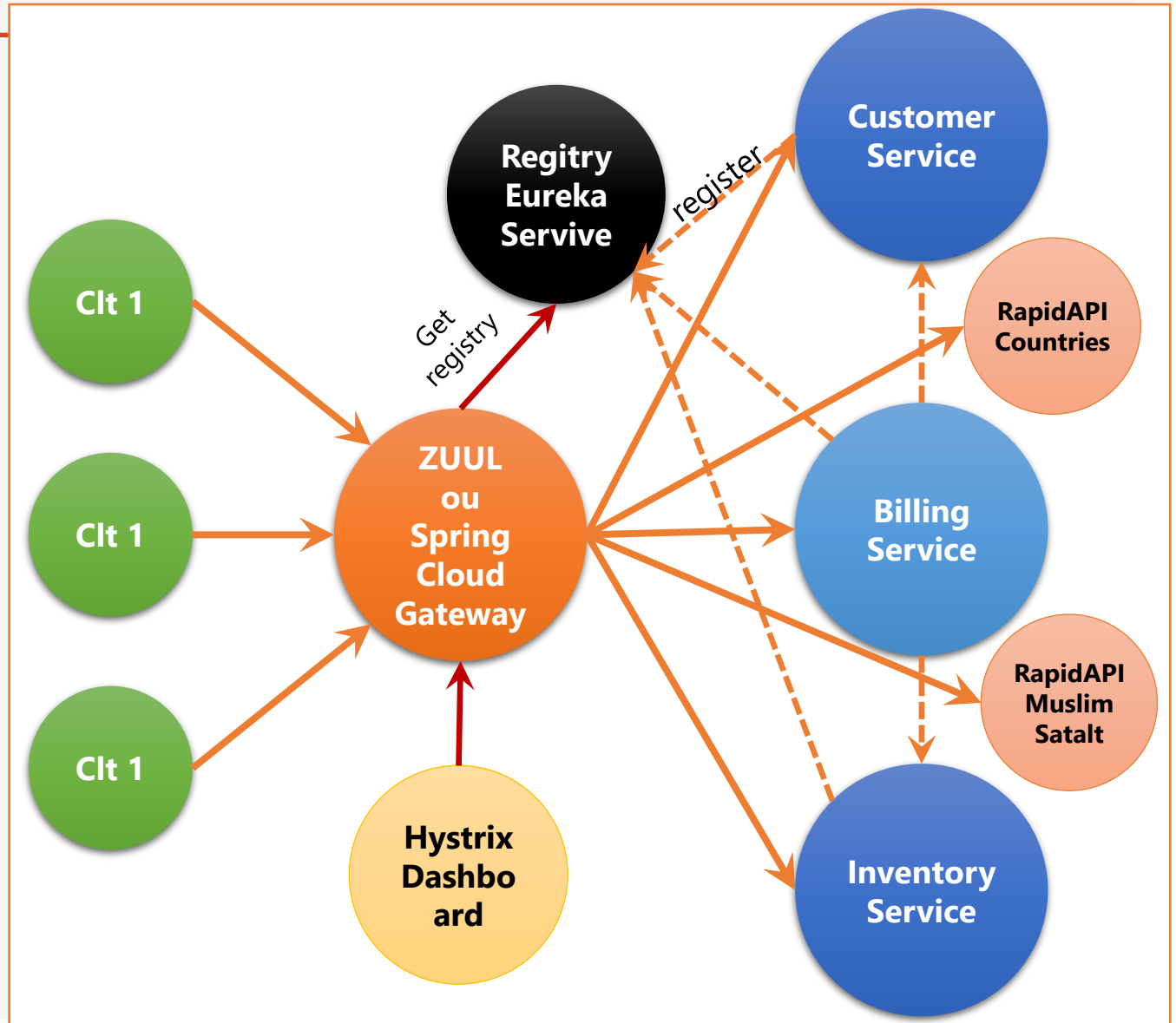
Req 2 : GET http://gateway/inventory-service/products



Spring Cloud Gateway

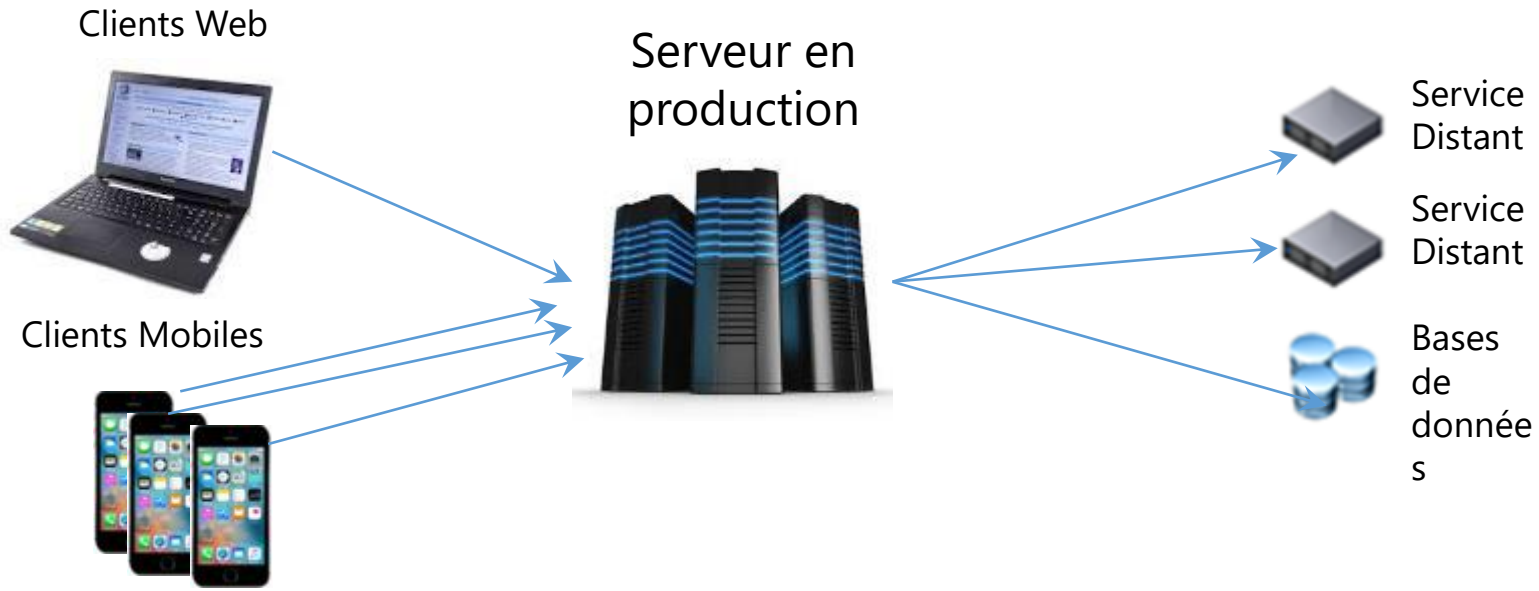
- Gateway API est un reverse proxy amélioré avec des fonctionnalités plus avancées, y compris l'**orchestration** et la **sécurité** et le **monitoring**.
- Quelques implémentations de API Gateway :

  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.



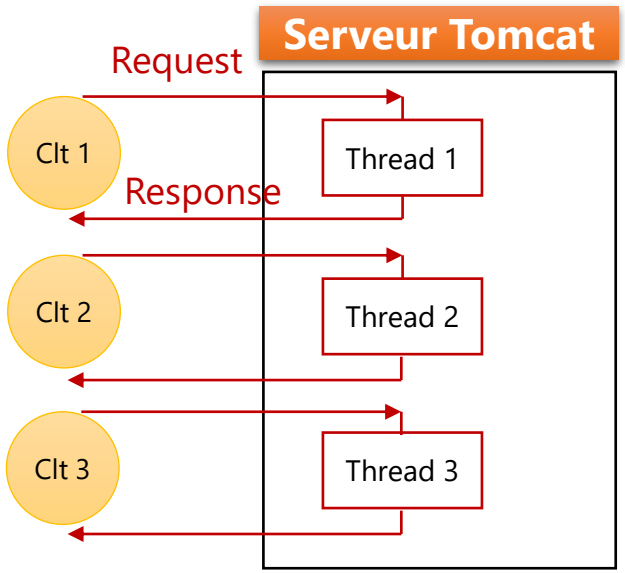
Blocking IO Model : Latency Problem

- 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



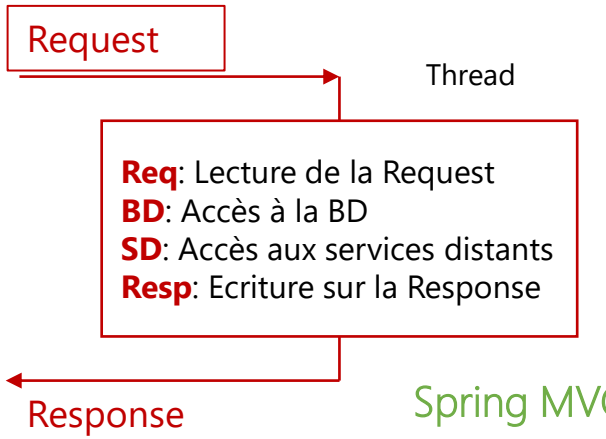
Modèles : Multi Threads avec IO Bloquantes Vs Single Thread avec IO Non Bloquantes

Multi Threads avec IO Bloquantes



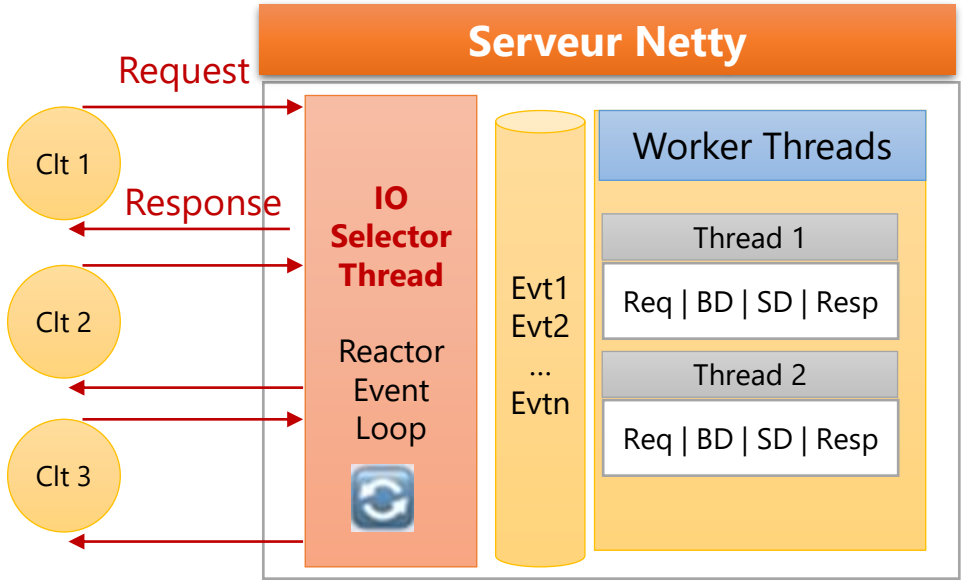
Package java.io

- InputStream
- OutputStream
- Reader
- Writer



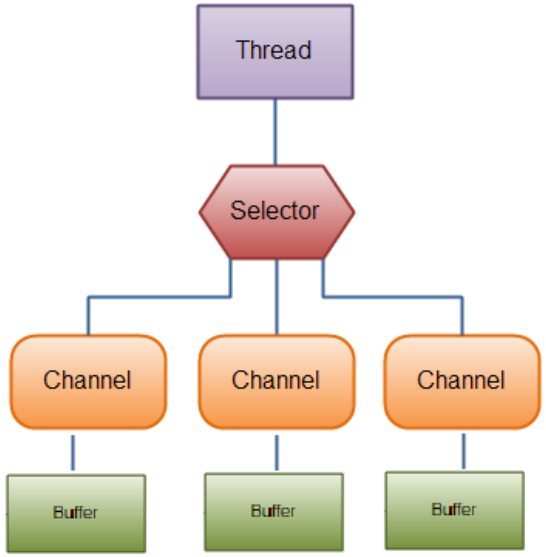
Spring MVC avec Tomcat

Single Thread avec IO Non Bloquantes



Package java.nio

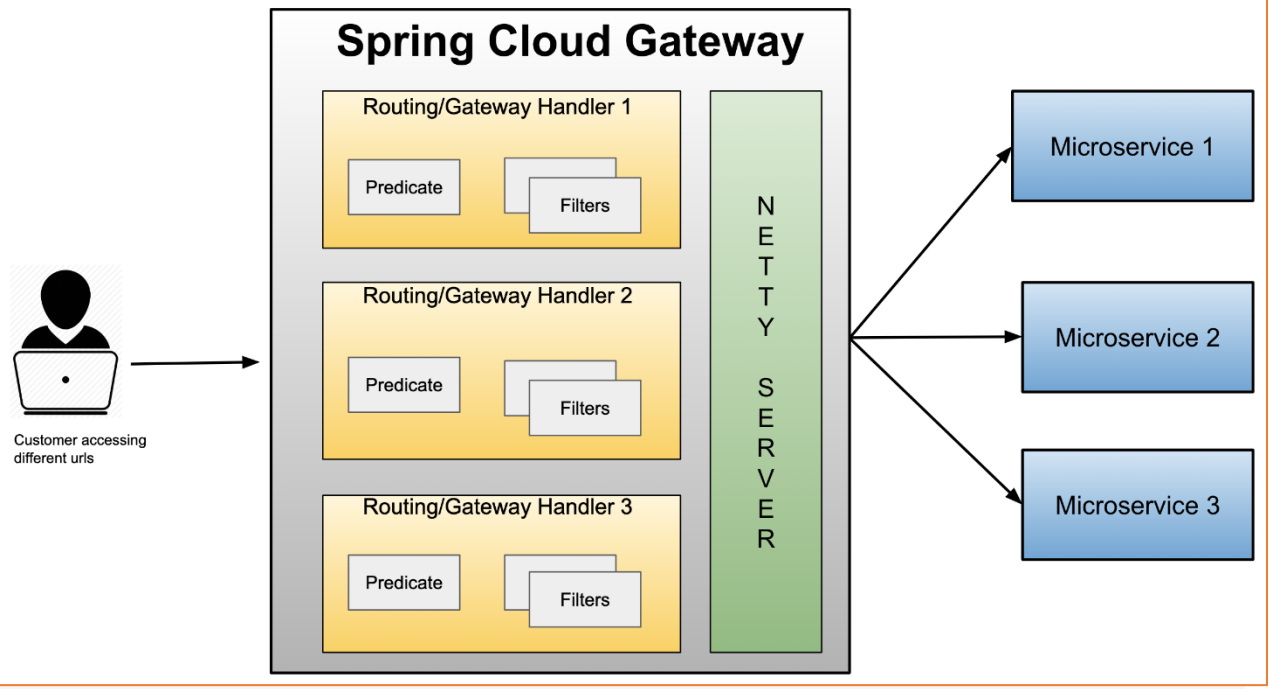
- Channels:
  - SocketChannel,
  - DataGramChannel
- Buffers
- Selector



Réactive Spring ou Spring Web Flux avec Netty

Spring Cloud Gateway

- 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 traitement et des modifications des requêtes et des réponses HTTP



Predicates :

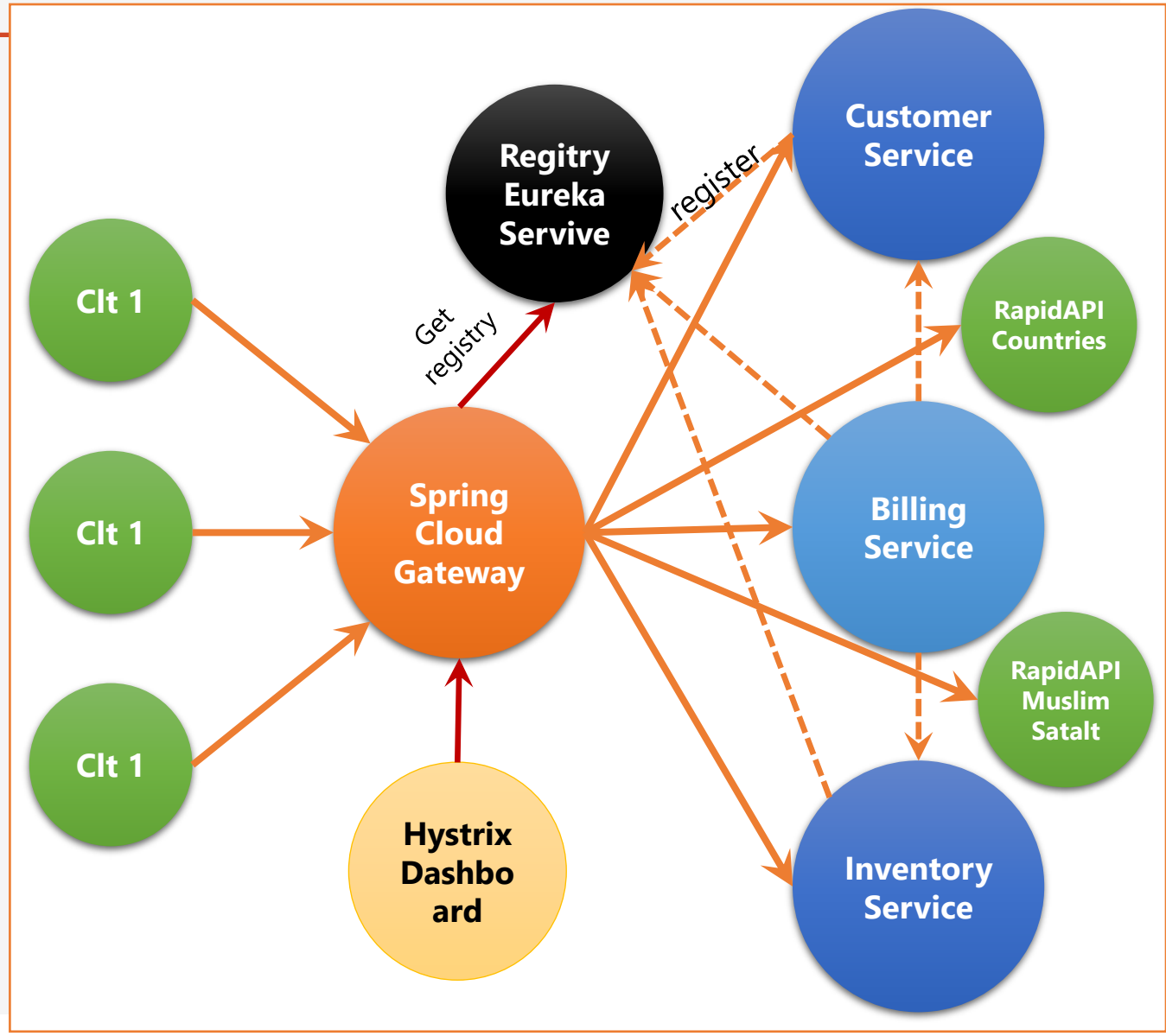
- Host, Path, Method
- After, Before, Between
- Cookie, Header, Query
- RmoteAddr
- Etc …

Filters :

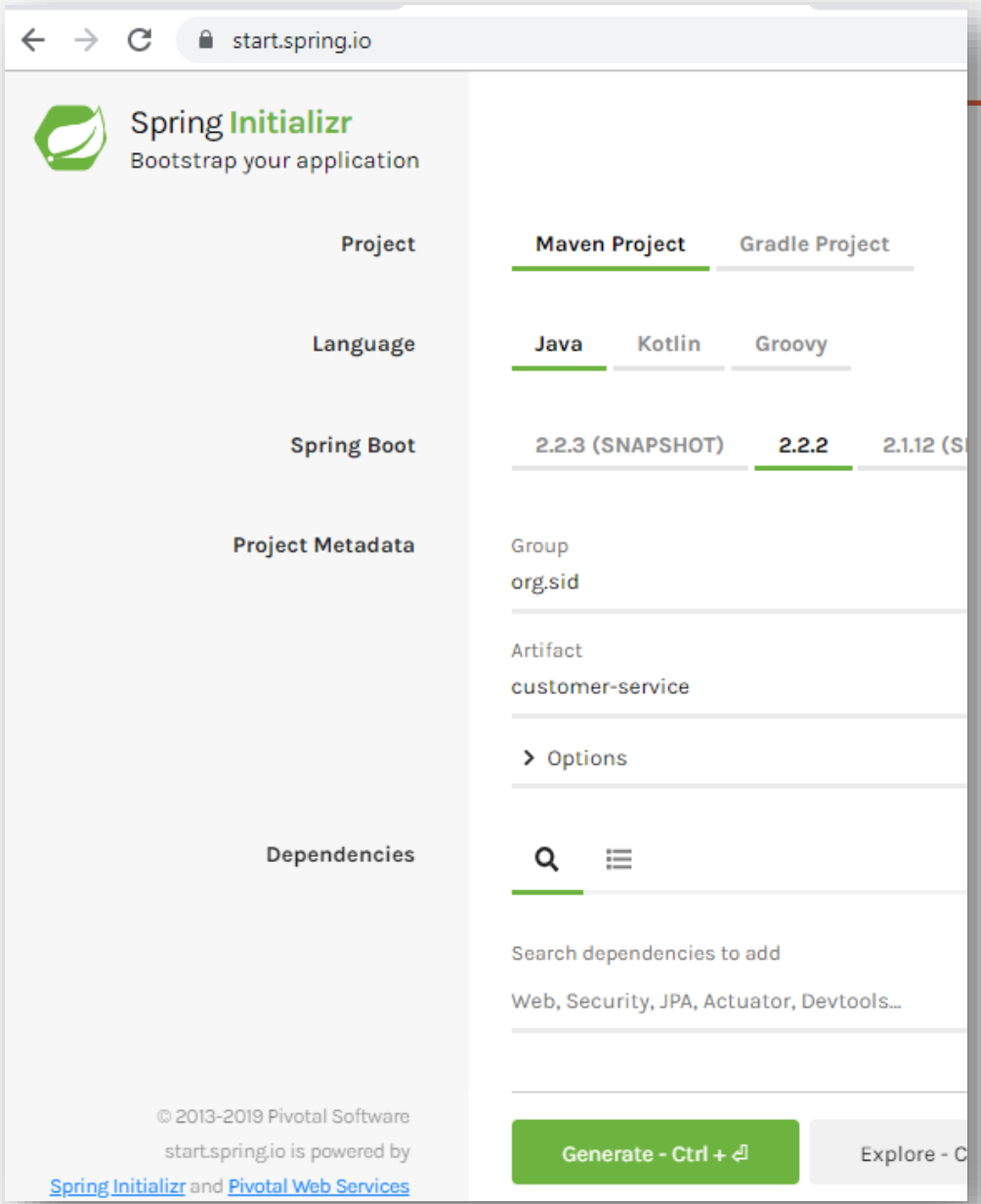
- AddRequestHeader
- AddRequestParameter
- AddResponseHeader
- DedupeResponseHeader
- Hystrix
- CircuitBreaker
- RewritePath
- Etc …

Application

- Créer une application basée sur deux services métiers:
  - Service des clients
  - Service d'inventaire
  - Service Facturation
  - Services Externes : RapidAPI
- L'orchestration des services se fait via les 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
  - Hystrix Circuit Breaker
  - Hystrix DashBoard



Customer-service



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=*
```

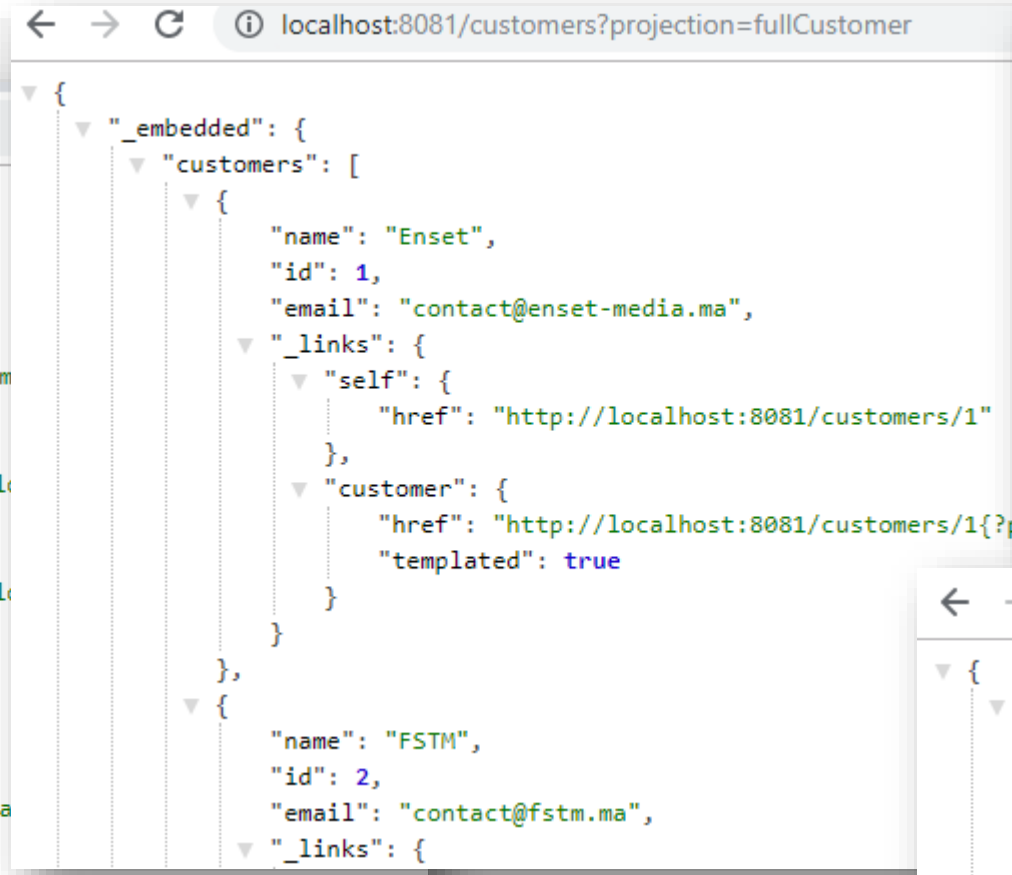
```
@Entity @Data @NoArgsConstructor @AllArgsConstructor @ToString
class Customer{
        @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;   private String name; private String email;
}
```

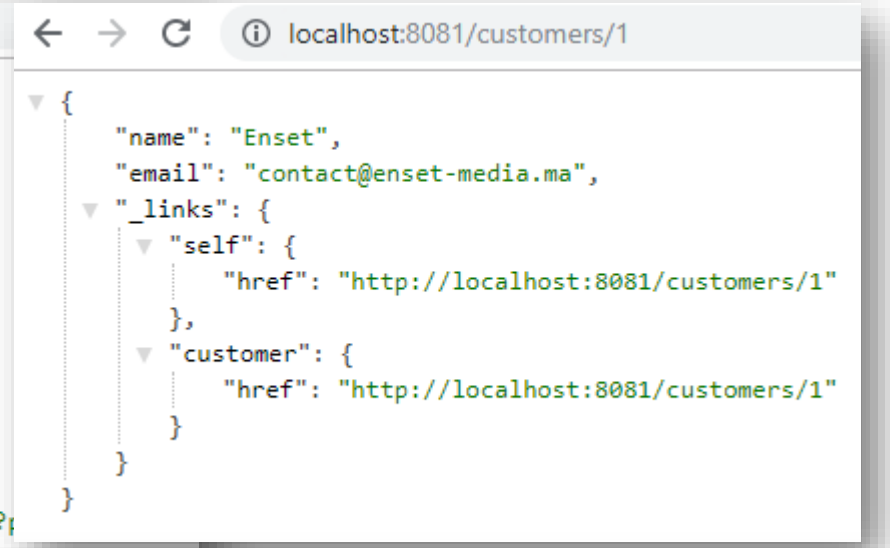
```
@RepositoryRestResource
interface CustomerRepository extends JpaRepository<Customer,Long> { }
```

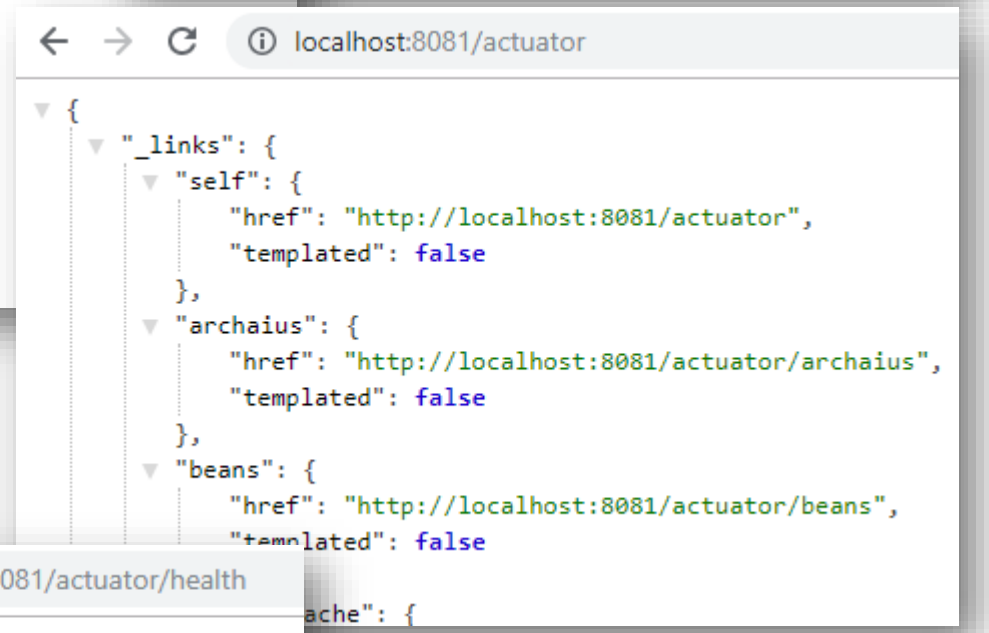
```
@Projection(name = "fullCustomer",types =
Customer.class)
interface CustomerProjection extends Projection{
          public Long getId();
          public String getName();
          public String getEmail();
}
```

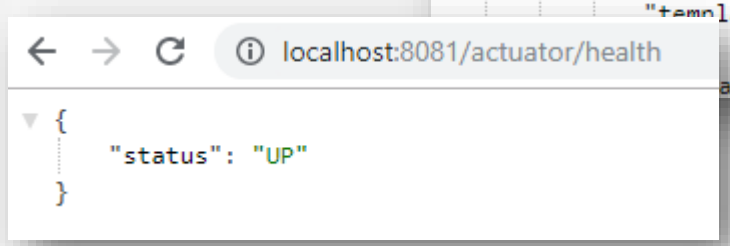
```
@SpringBootApplication
public class CustomerServiceApplication {

public static void main(String[] args) { pringApplication.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);
            };
    }
}
```

Customer-service

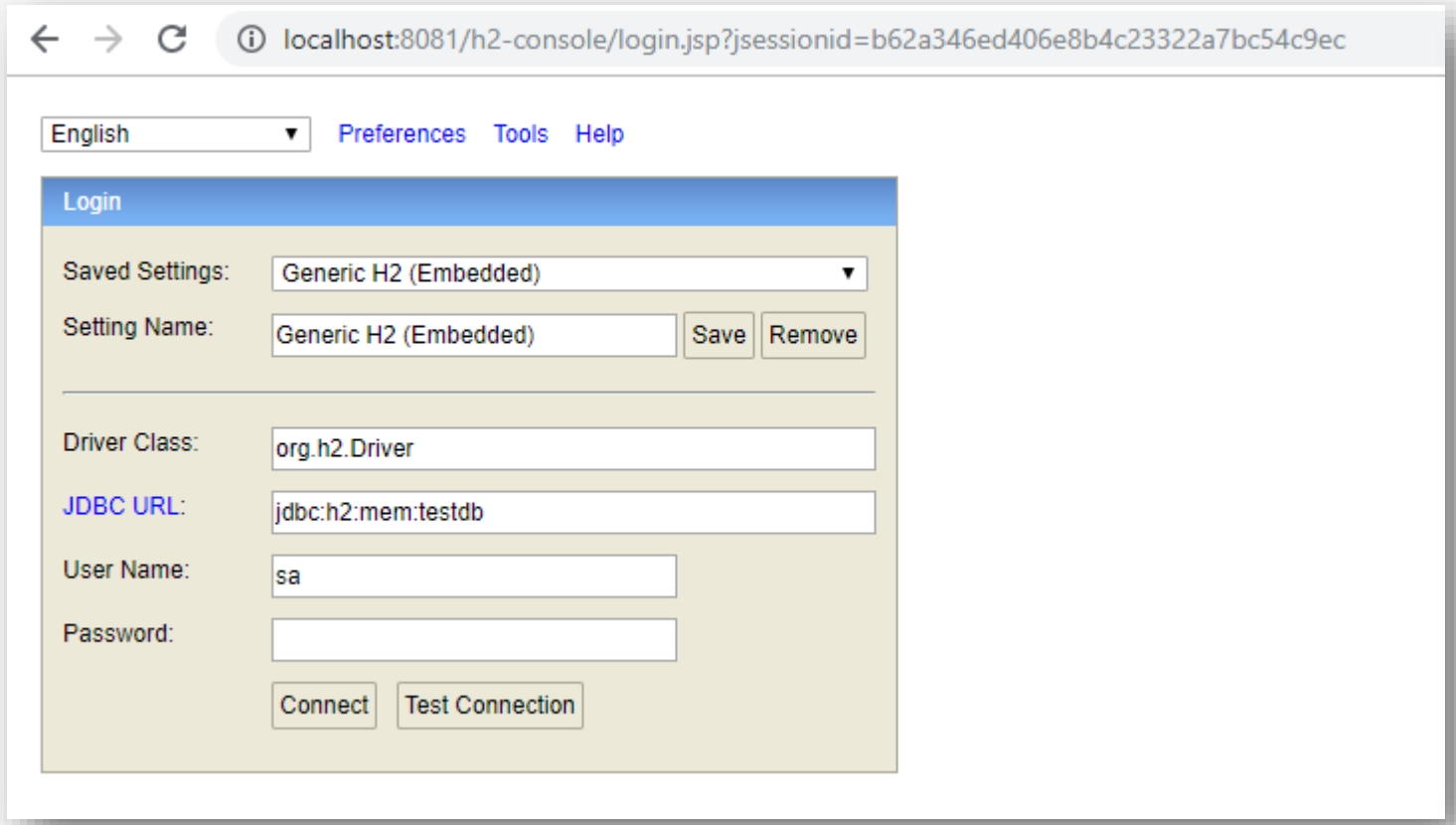


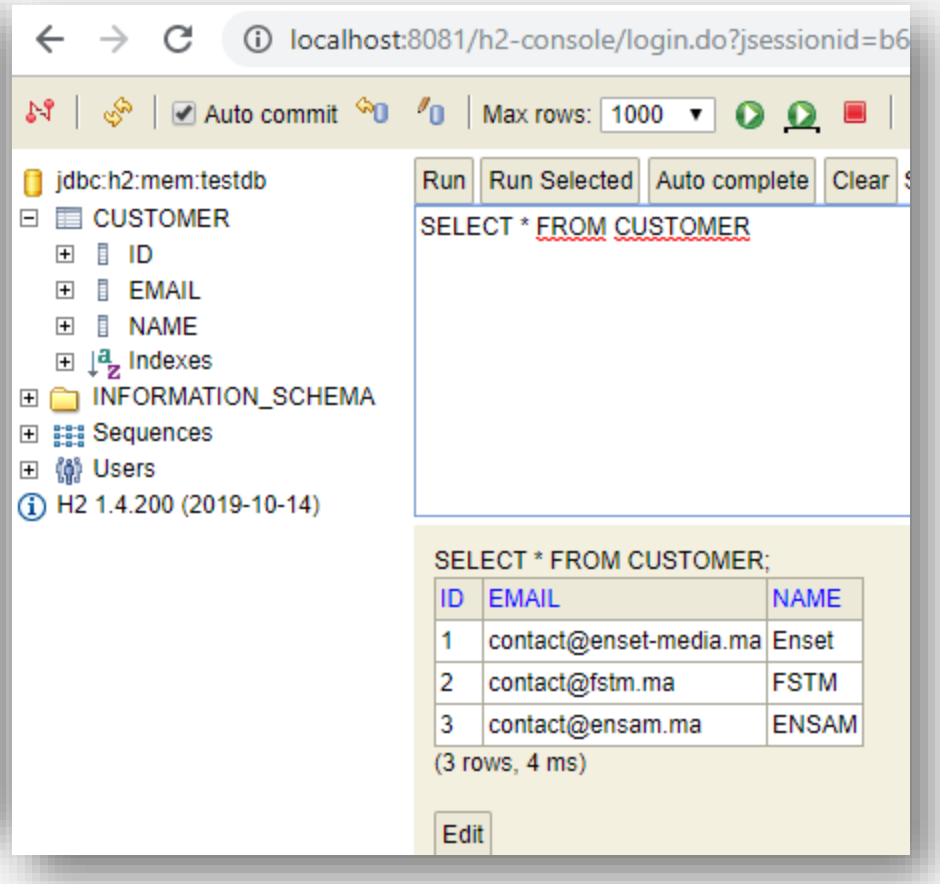




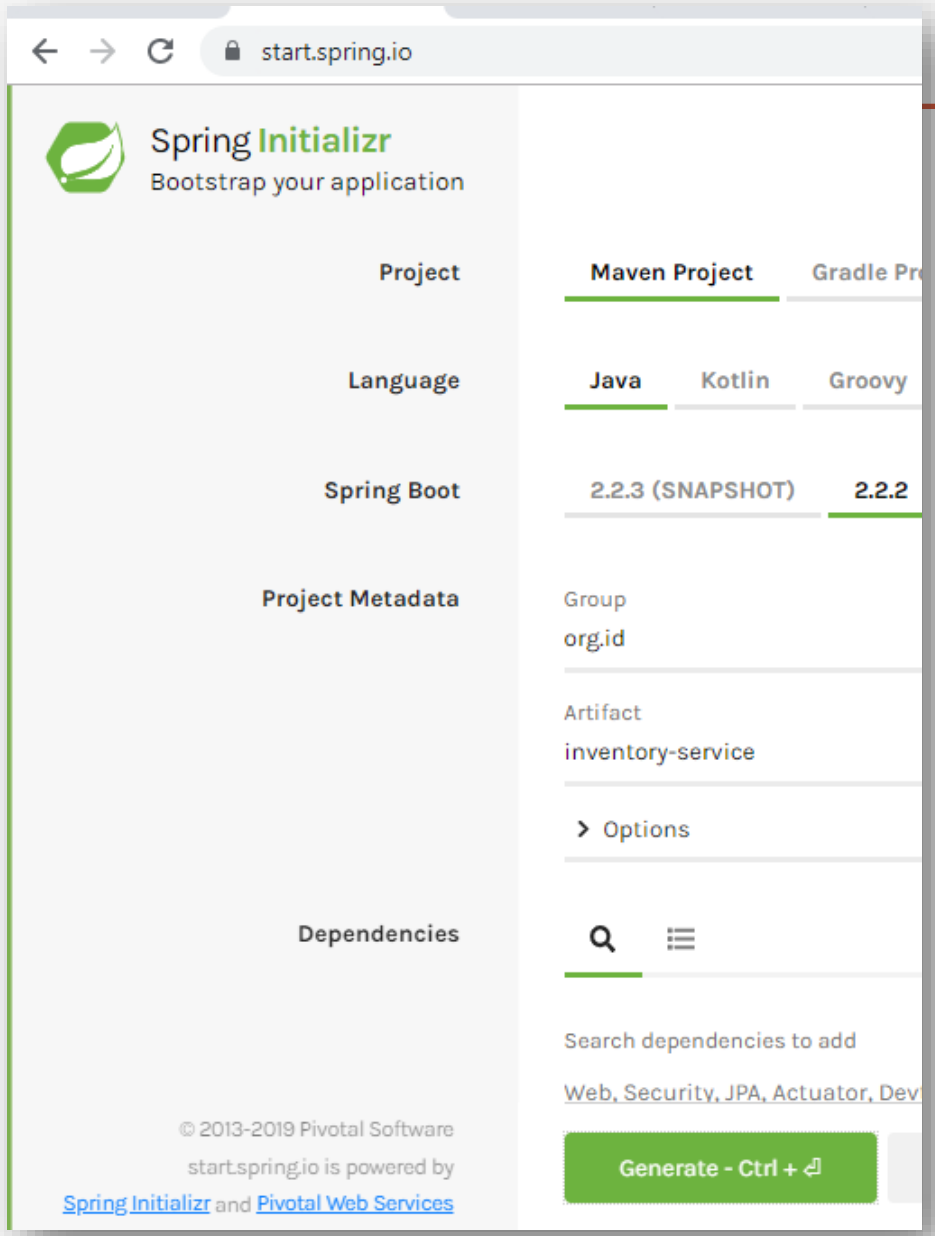


Customer-service : Base de données H2 (http://localhost:8081/h2-console)





Inventory-service



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
```

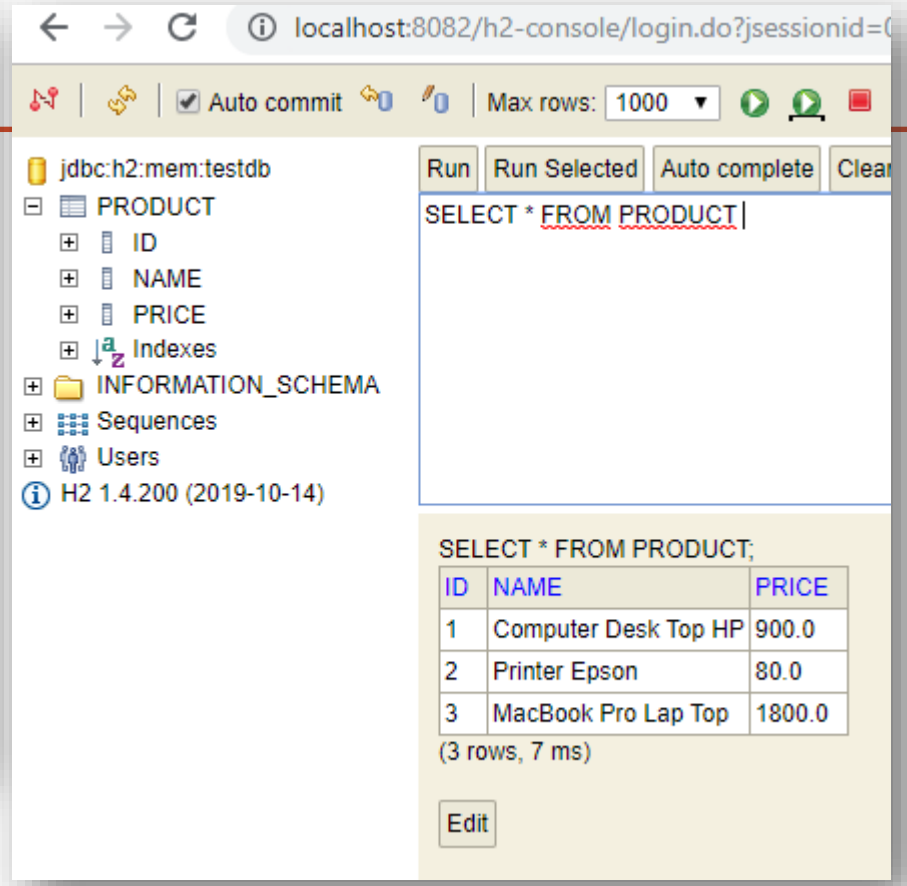
Inventory-service

localhost:8082/products

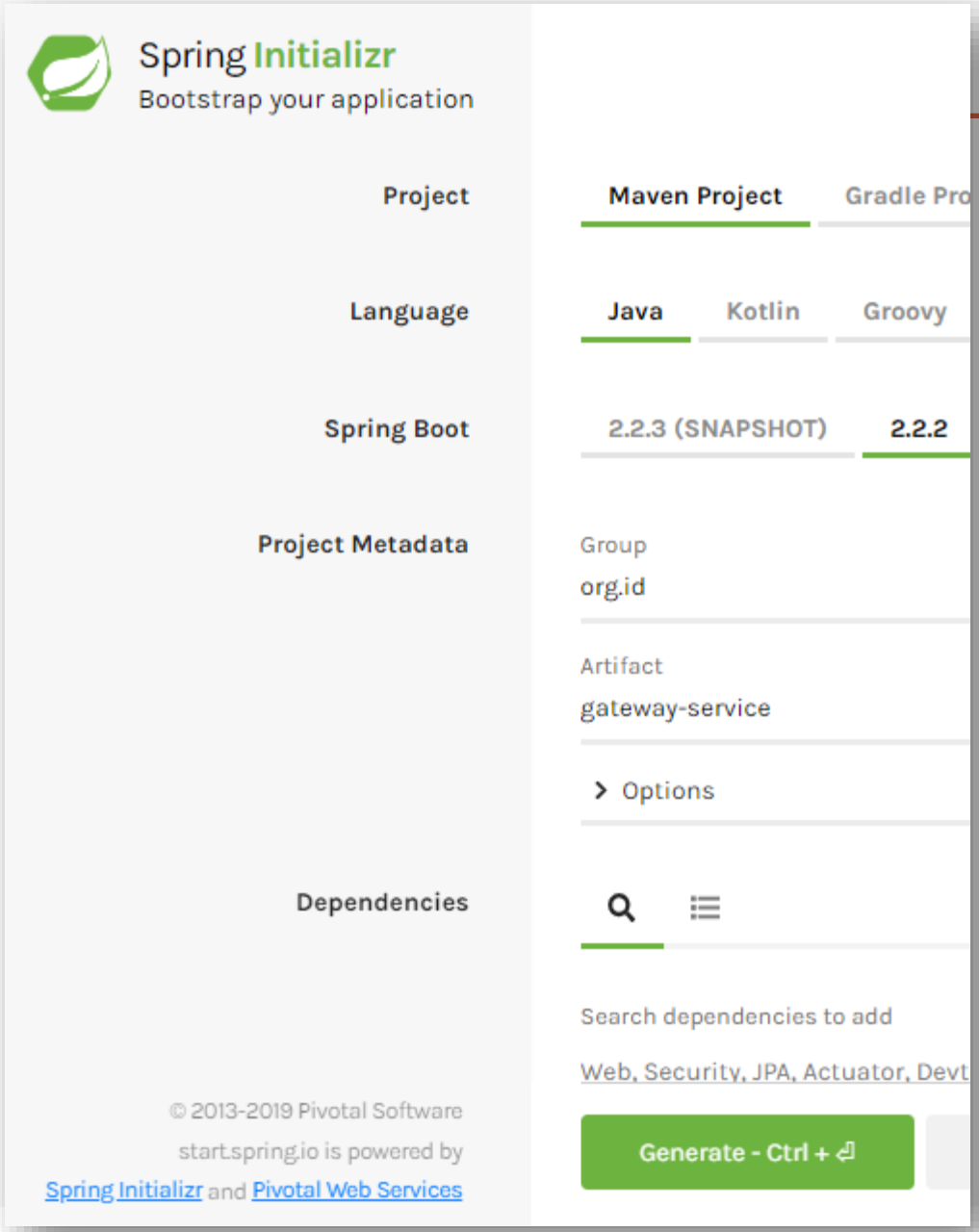

```
{
  "_embedded": {
    "products": [
      {
        "name": "Computer Desk Top HP",
        "price": 900,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/1"
          },
          "product": {
            "href": "http://localhost:8082/products/1"
   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


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"
    }
  }
}
```

Gateway-service



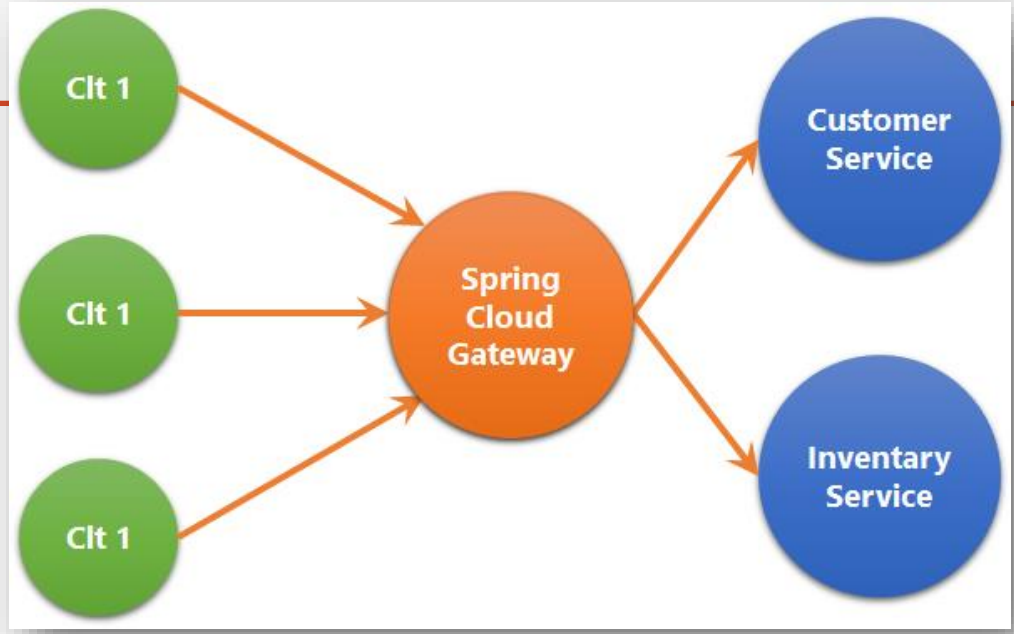
Selected dependencies

- **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.
- **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

```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/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

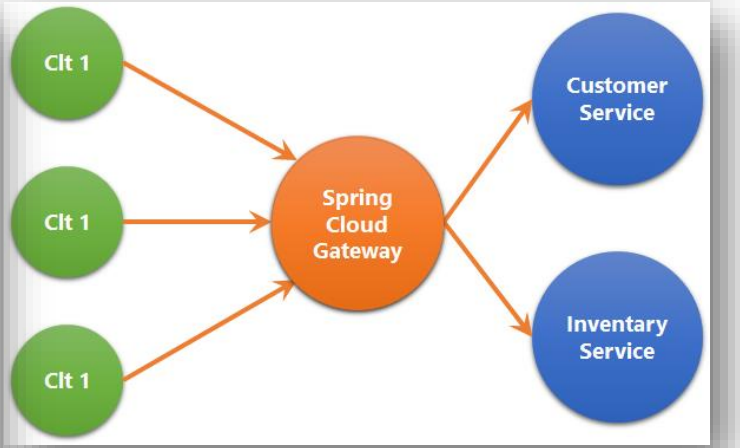
```
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
        return builder.routes()
                        .route(r->r.path("/customers/**").uri("http://localhost:8081/").id("r1"))
                        .route(r->r.path("/products/**").uri("http://localhost:8082/").id("r2"))
      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

localhost:8888/products/1

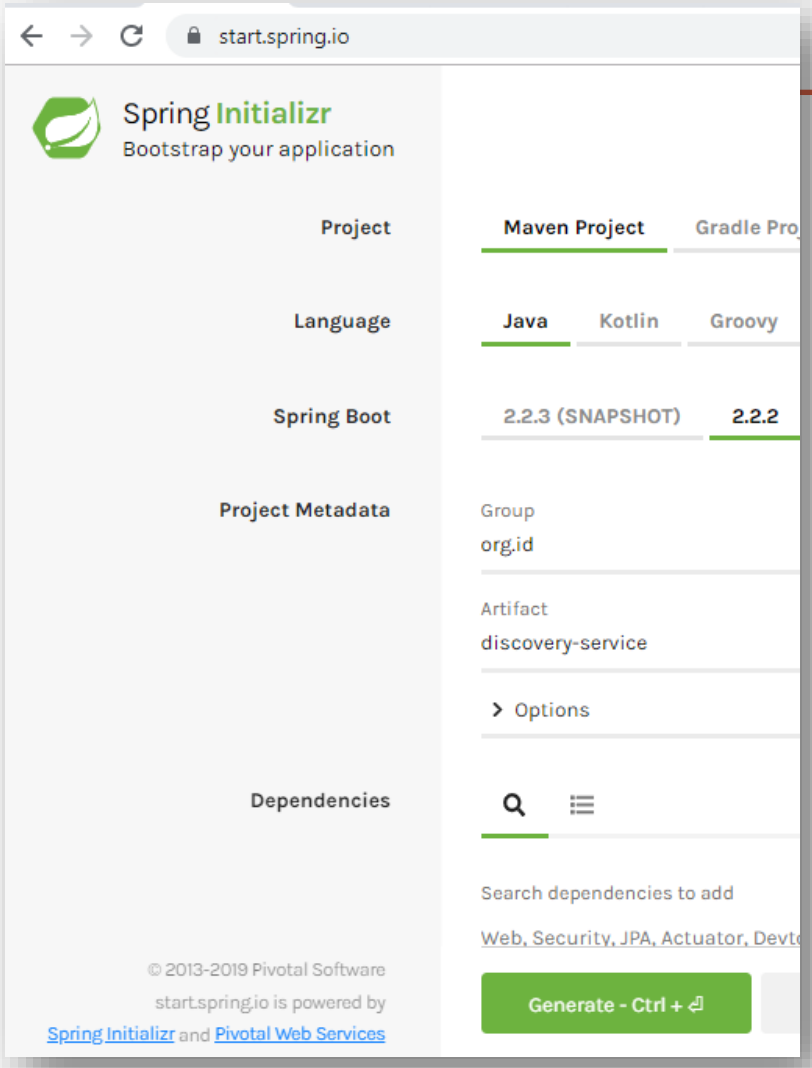

```
{
    "name": "Computer Desk Top HP",
    "price": 900,
    "_links": {
        "self": {
            "href": "http://localhost:8082/products/1"
        },
        "product": {
            "href": "http://localhost:8082/products/1"
        }
    }
}
```


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"
        }
    }
}
```

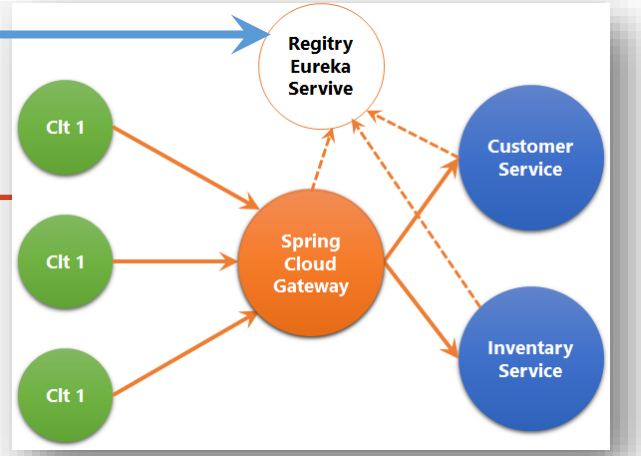



Eureka Discovery Service : Dynamic Routing



Selected dependencies

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

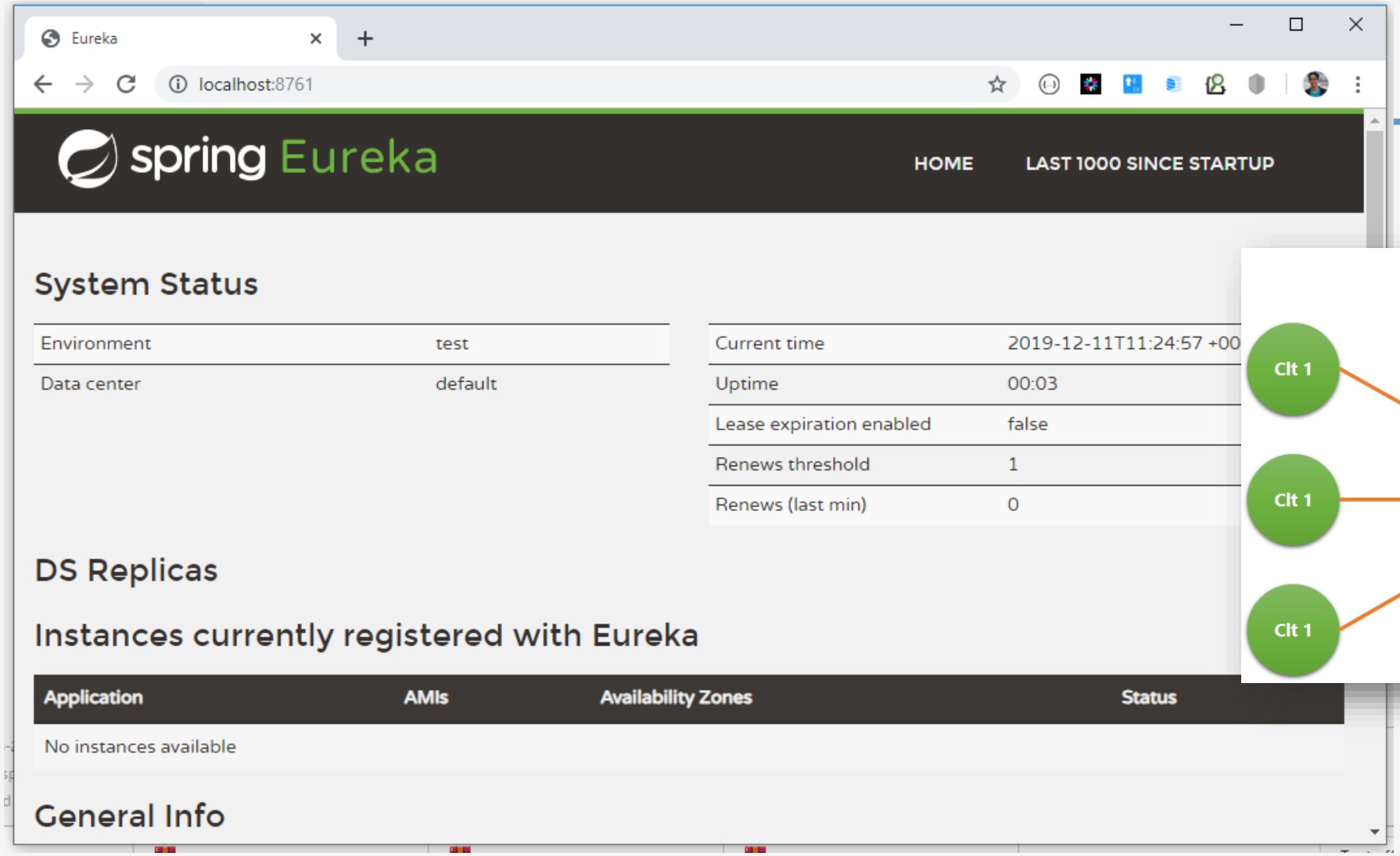


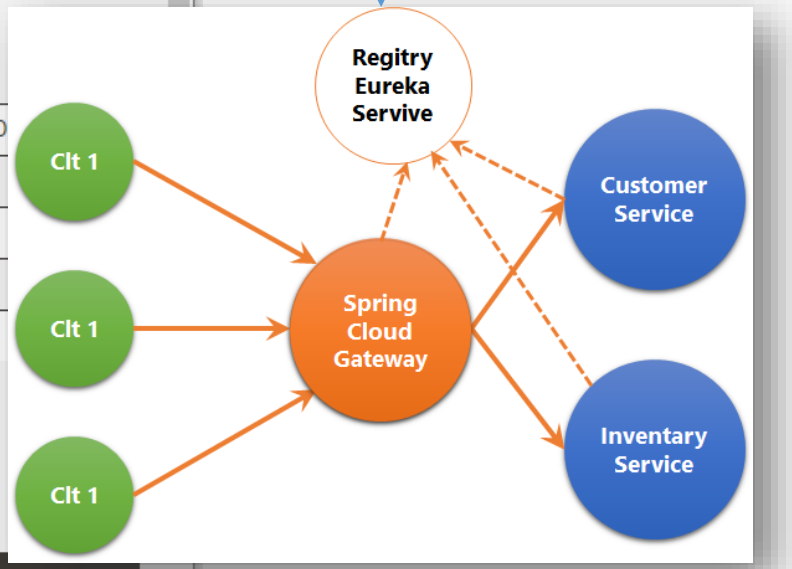

```java
package org.id.discoveryservice; import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication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

application.properties

```
server.port=8761
# dont register server itself as a client.
eureka.client.fetch-registry=false
# Does not register itself in the service registry.
eureka.client.register-with-eureka=false
```

Eureka Discovery Service : Dynamic Routing





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

Customer-service

application.properties

```
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
```

Inventory-service

application.properties

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

Eureka Discovery Service : Dynamic Routing

localhost:8761

spring Eureka

HOME LAST 1000 S

System Status

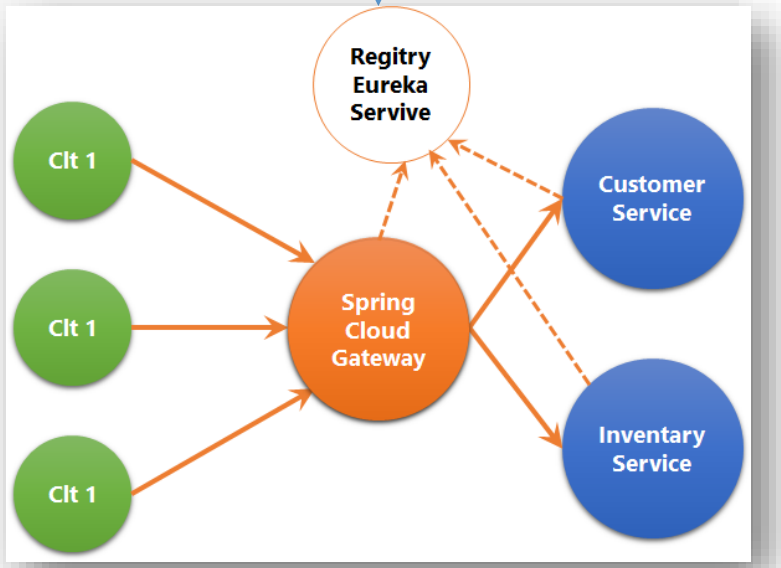
Environment	test
Data center	default

Current time	2019-12-11T13:
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

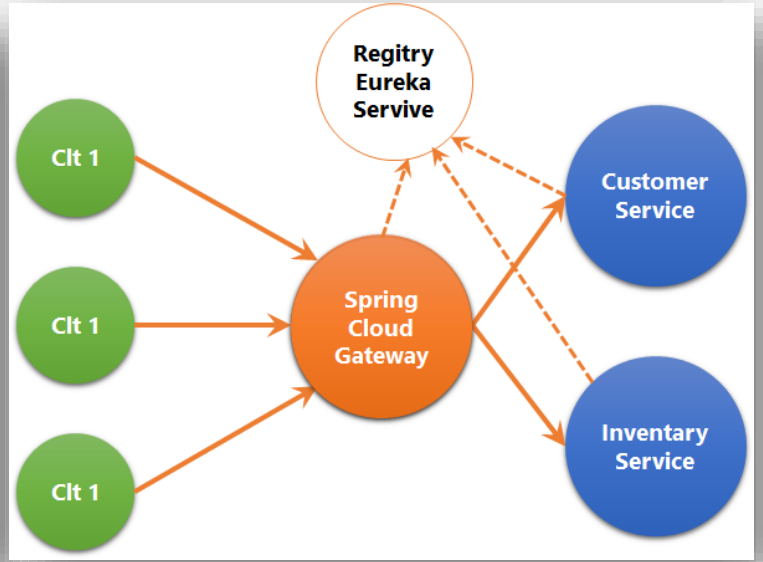
```
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
        return builder.routes()
                        .route(r->r.path("/customers/**").uri("lb://CUSTOMER-SERVICE") .id("r1"))
                         .route(r->r.path("/products/**").uri("lb://INVENTORY-SERVICE") .id("r2"))
    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

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"
        }
    }
}
```


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"
        }
    }
}
```

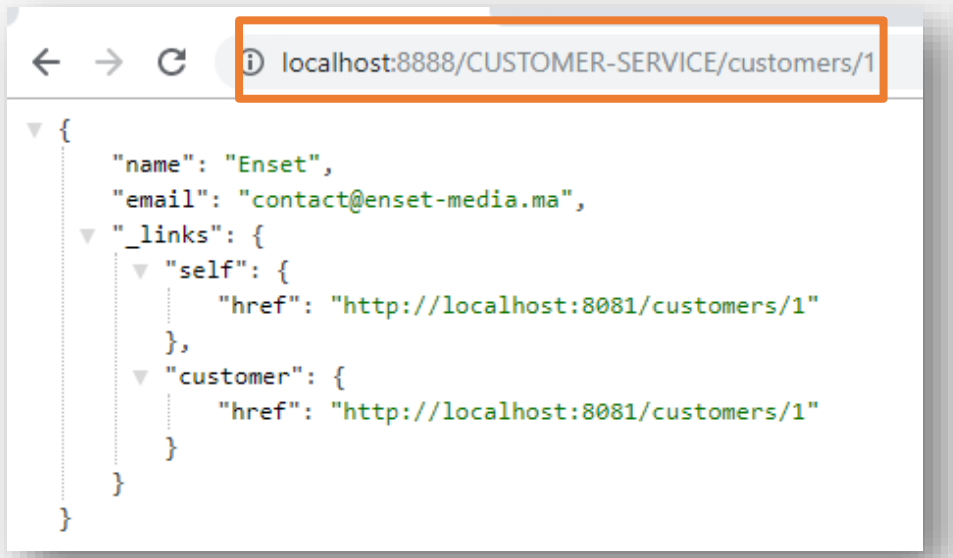



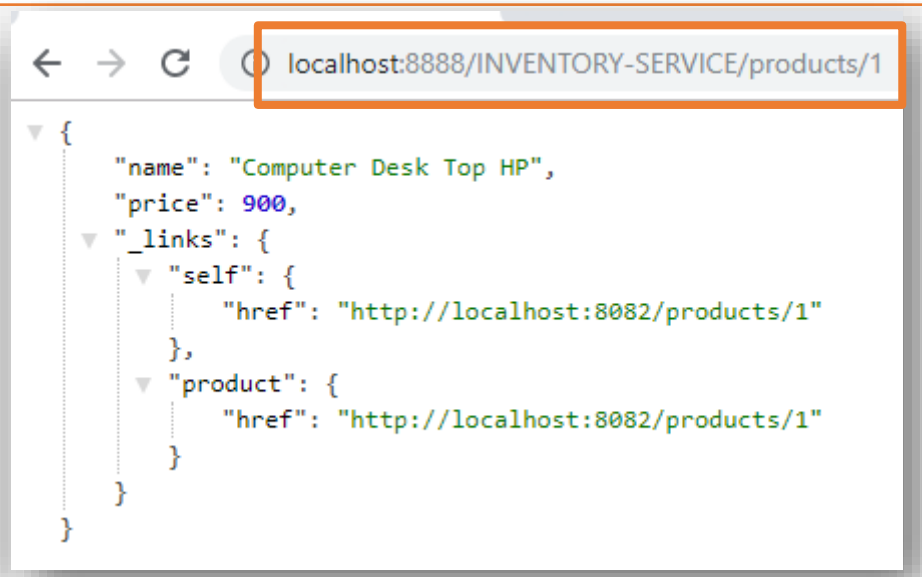
Dynamic routes configuration with Discovery Service

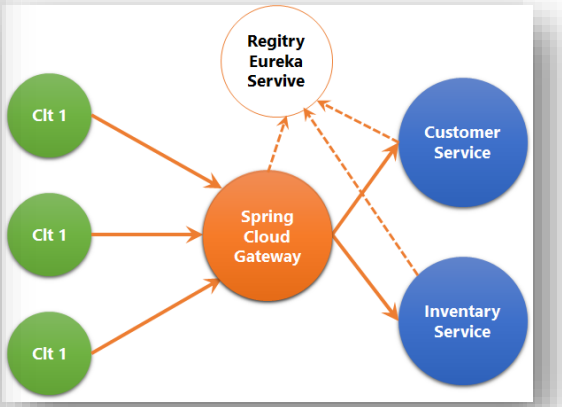
application.properties

```
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);
}
```

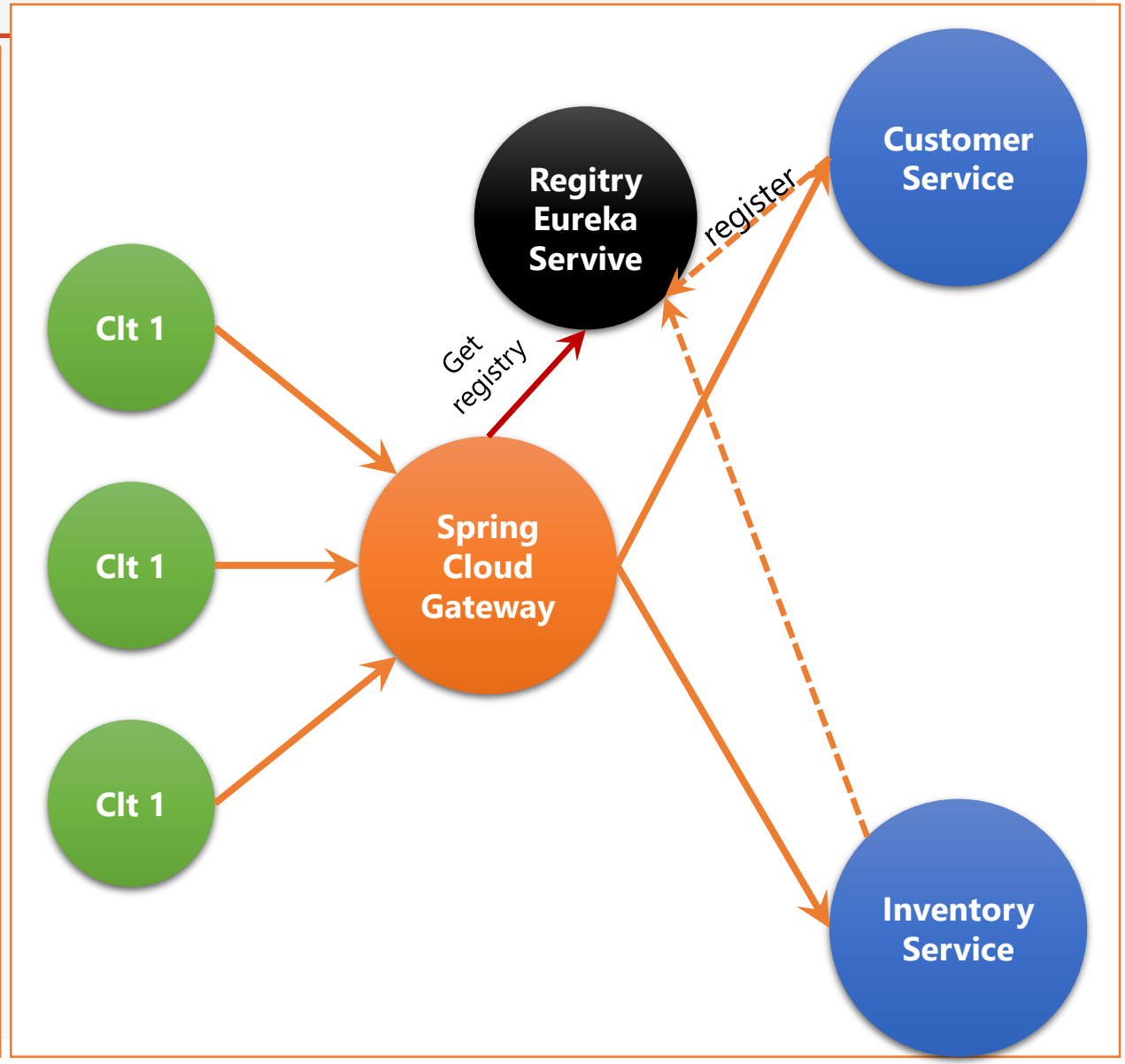






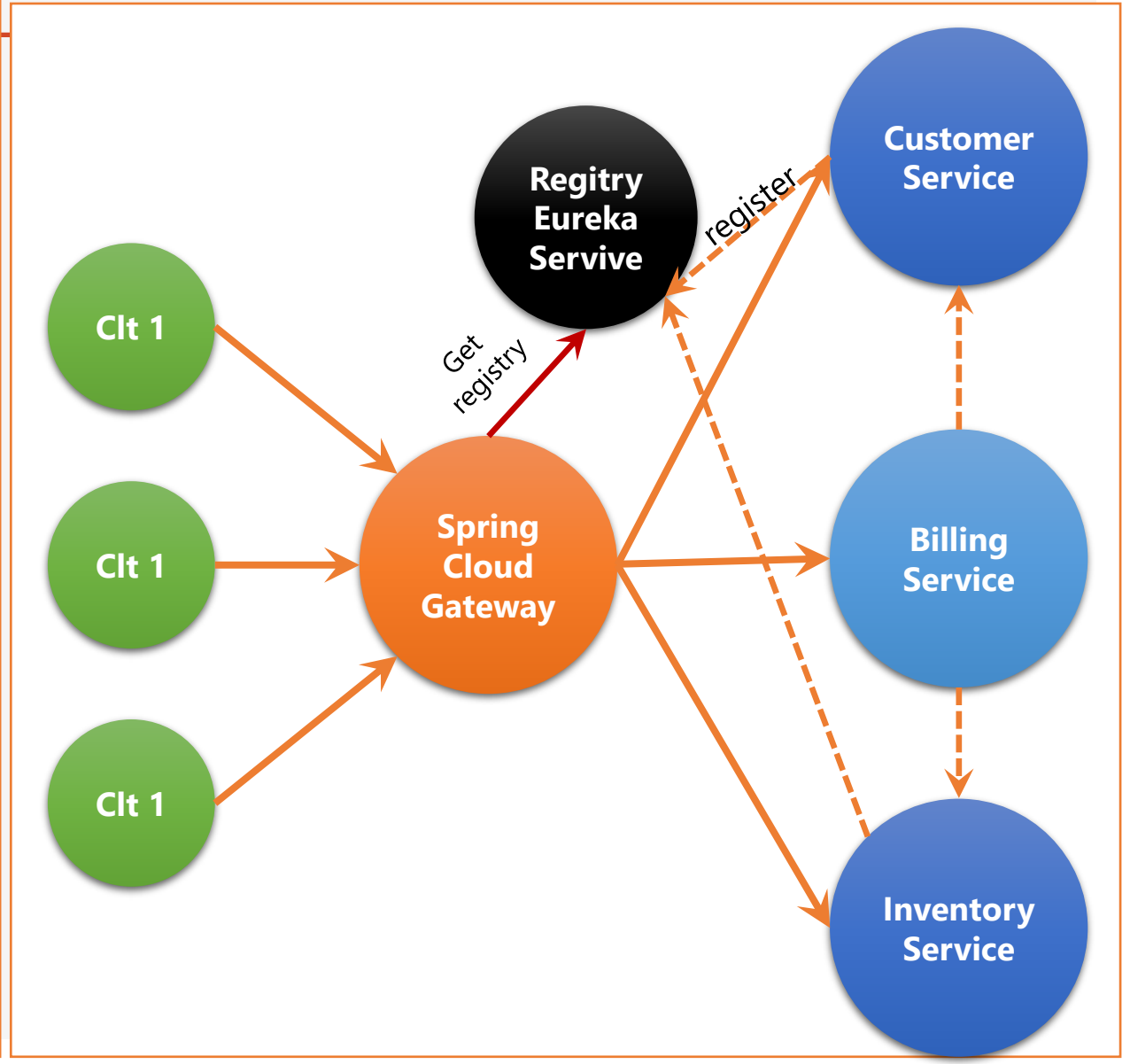
Activité Pratique 1 : Travail à faire

1. Créer le micro service Customer-service
   - Créer l'entité Customer
   - Créer l'interface CustomerRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
2. Créer le micro service Inventory-service
   - Créer l'entité Product
   - Créer l'interface ProductRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
3. Créer la Gateway service en utilisant Spring Cloud Gateway
   1. Tester la Service proxy en utilisant une configuration Statique basée sur le fichier application.yml
   2. Tester la Service proxy en utilisant une configuration Statique basée une configuration Java
4. Créer l'annuaire Registry Service basé sur NetFlix Eureka Server
5. Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micro services enregistrés dans l'annuaire Eureka Server



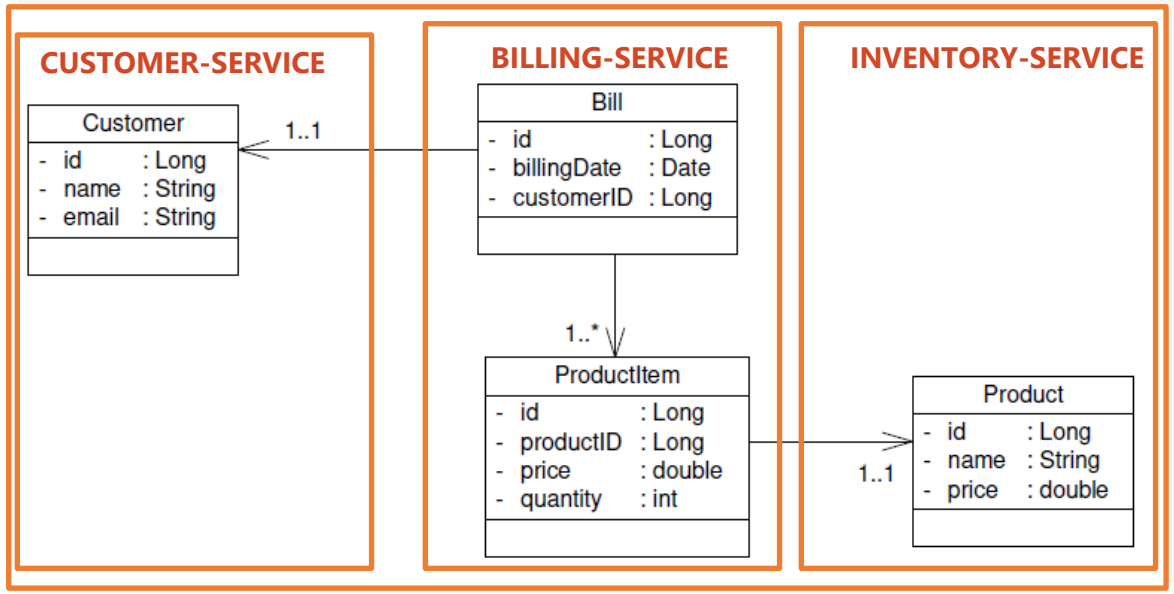
Activité Pratique : Travail à faire

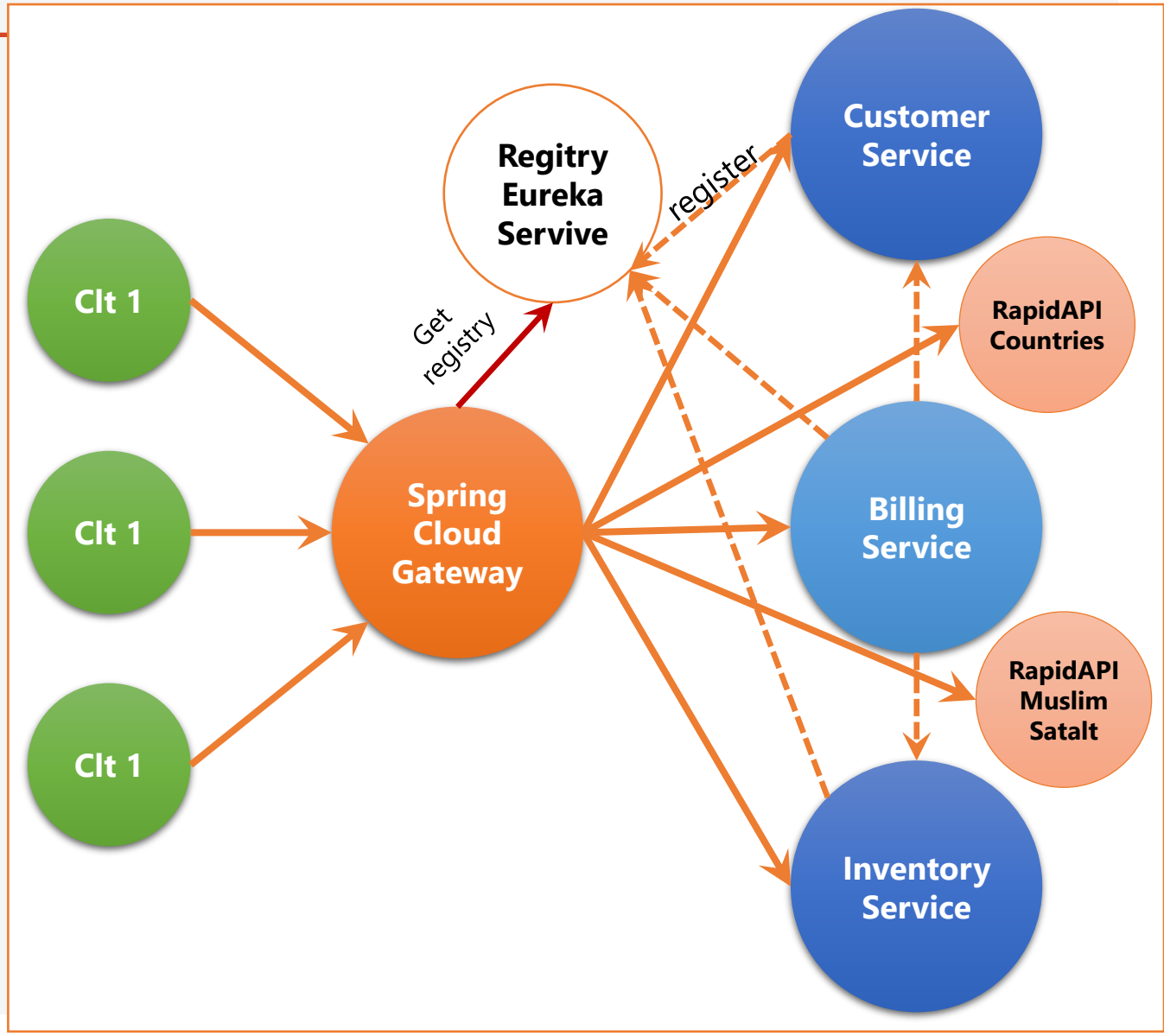
1. Créer le micro service Customer-service
   - Créer l'entité Customer
   - Créer l'interface CustomerRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
2. Créer le micro service Inventory-service
   - Créer l'entité Product
   - Créer l'interface ProductRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
3. Créer la Gateway service en utilisant Spring Cloud Gateway
   1. Tester la Service proxy en utilisant une configuration Statique basée sur le fichier application.yml
   2. Tester la Service proxy en utilisant une configuration Statique basée une configuration Java
4. Créer l'annuaire Registry Service basé sur NetFlix Eureka Server
5. Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micro services enregistrés dans l'annuaire Eureka Server
6. Créer Le service Billing-Service en utilisant Open Feign pour communiquer avec les services Customer-service et Inventory-service
7. Créer un client Angular qui permet d'afficher une facture



Autres services à ajouter

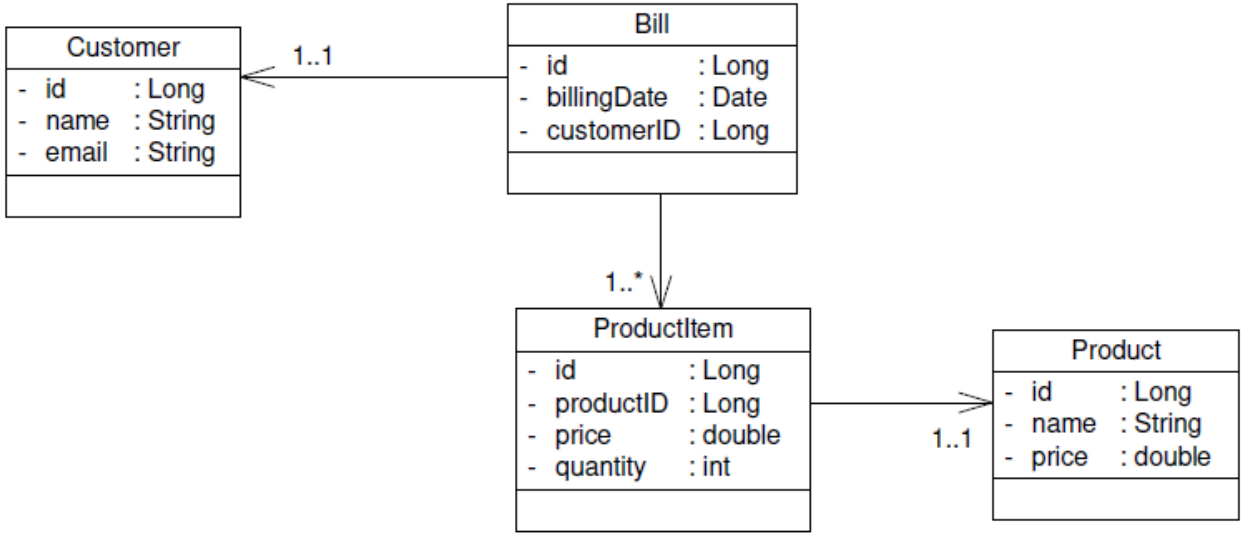
1. Accès aux services externes en utilisant des filtres au niveau du gateway service :
   - RapidAPI Countries
   - Rapid API Mulsim Salat
2. Utilisation de Circuit Breaker avec Hystrix
3. Utilisation de Hystrix Dashboard pour surveiller l'état du trafic au niveau du service Gateway
4. Ajouter un service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





Billing-service

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



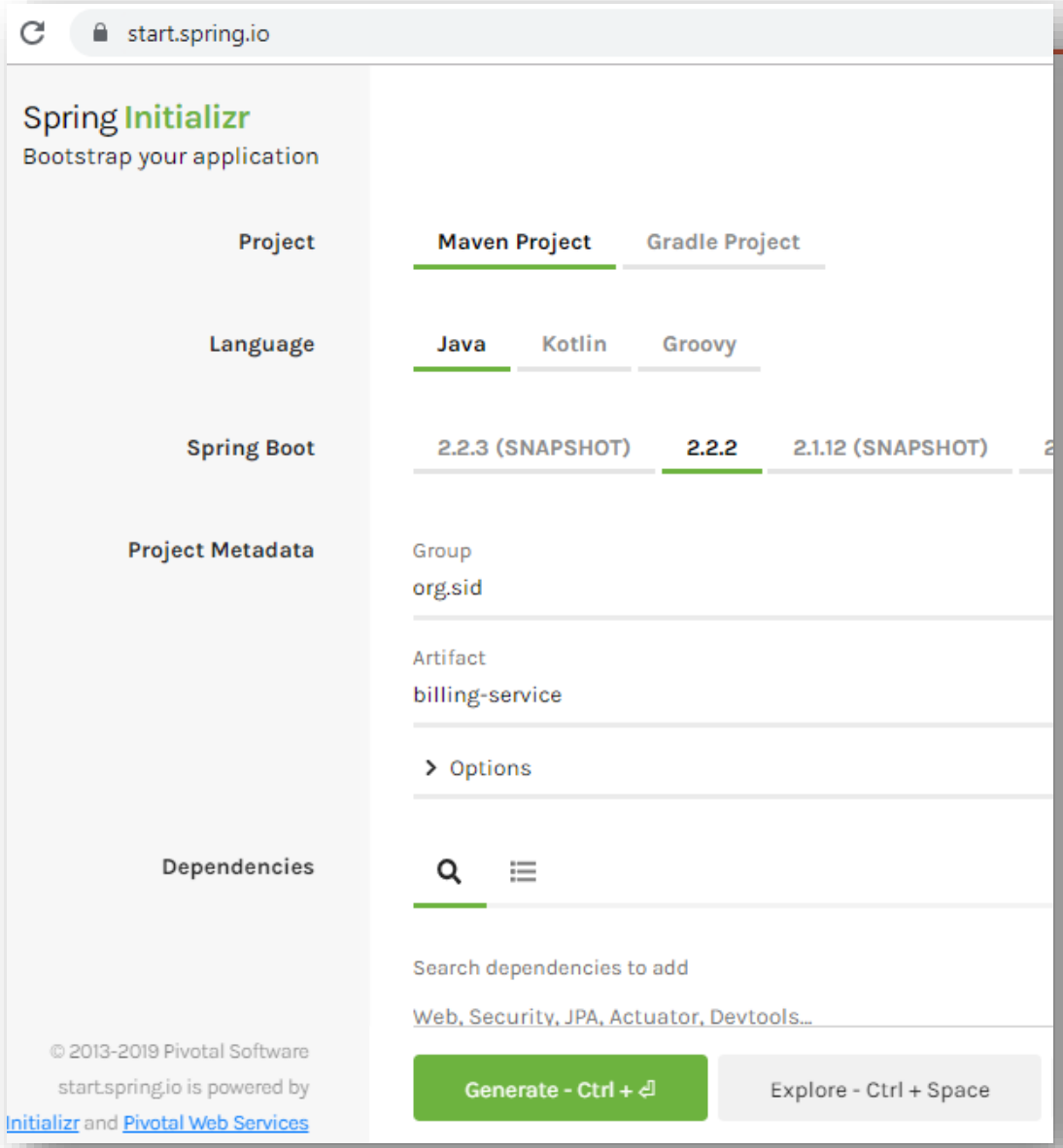

```java
@Entity @Data @NoArgsConstructor @AllArgsConstructor
class ProductItem{
 @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;
 private long productID;
 private double price; private double quantity;
 @ManyToOne
 private Bill bill;

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

Billing-service

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

```java
@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



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.

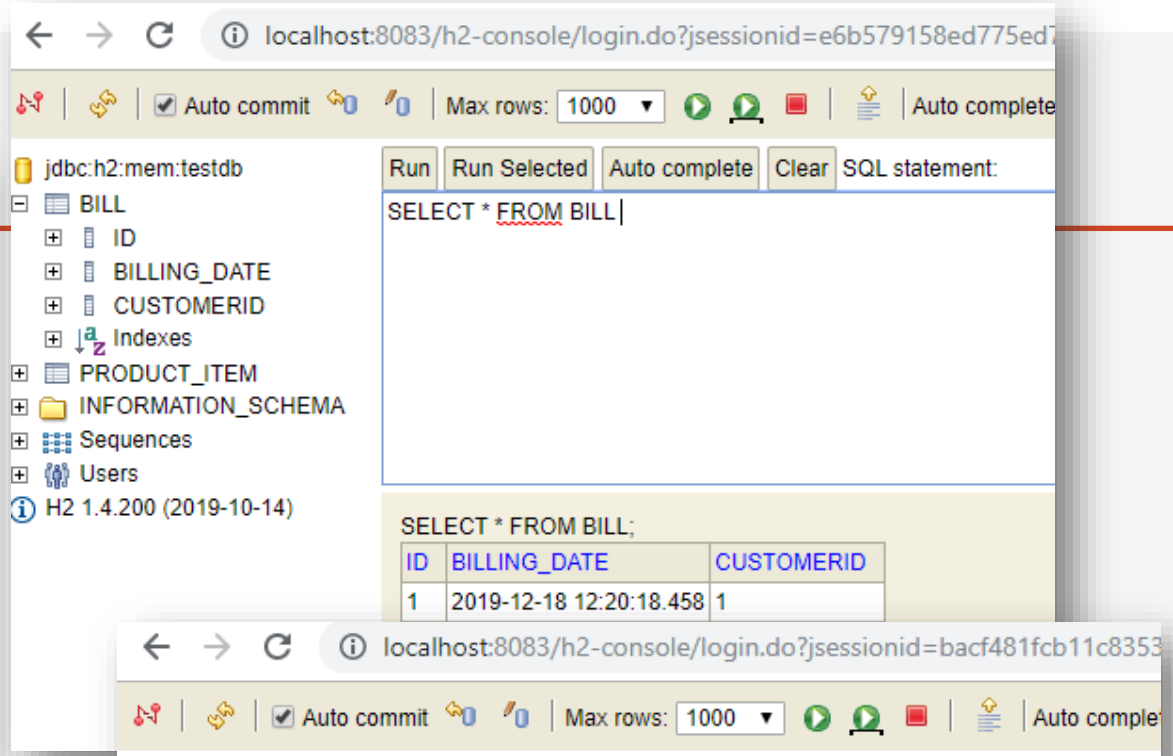
Billing-service

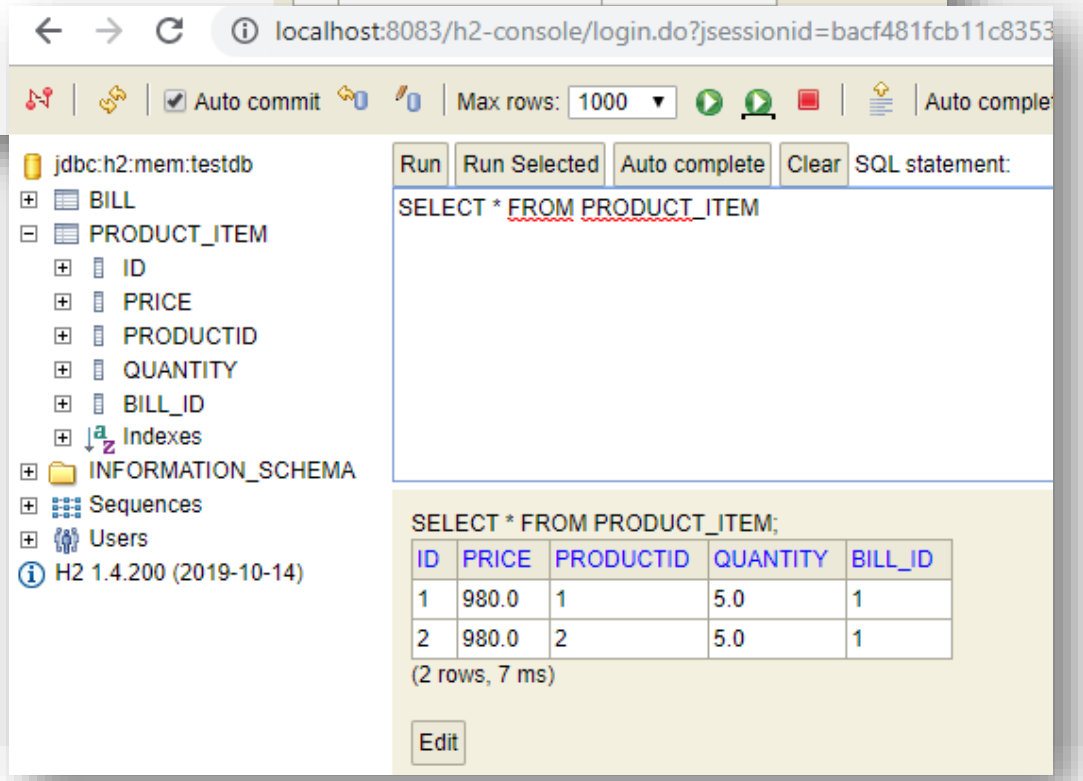
```
@RestController
class BillRestController{
        @Autowired private BillRepository billRepository;
        @Autowired private ProductItemRepository productItemRepository;
        @Autowired private CustomerServiceClient customerServiceClient;
        @Autowired private InventoryServiceClient inventoryServiceClient;
        @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
}
```



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-Maġribiyah"
        ],
        "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
```

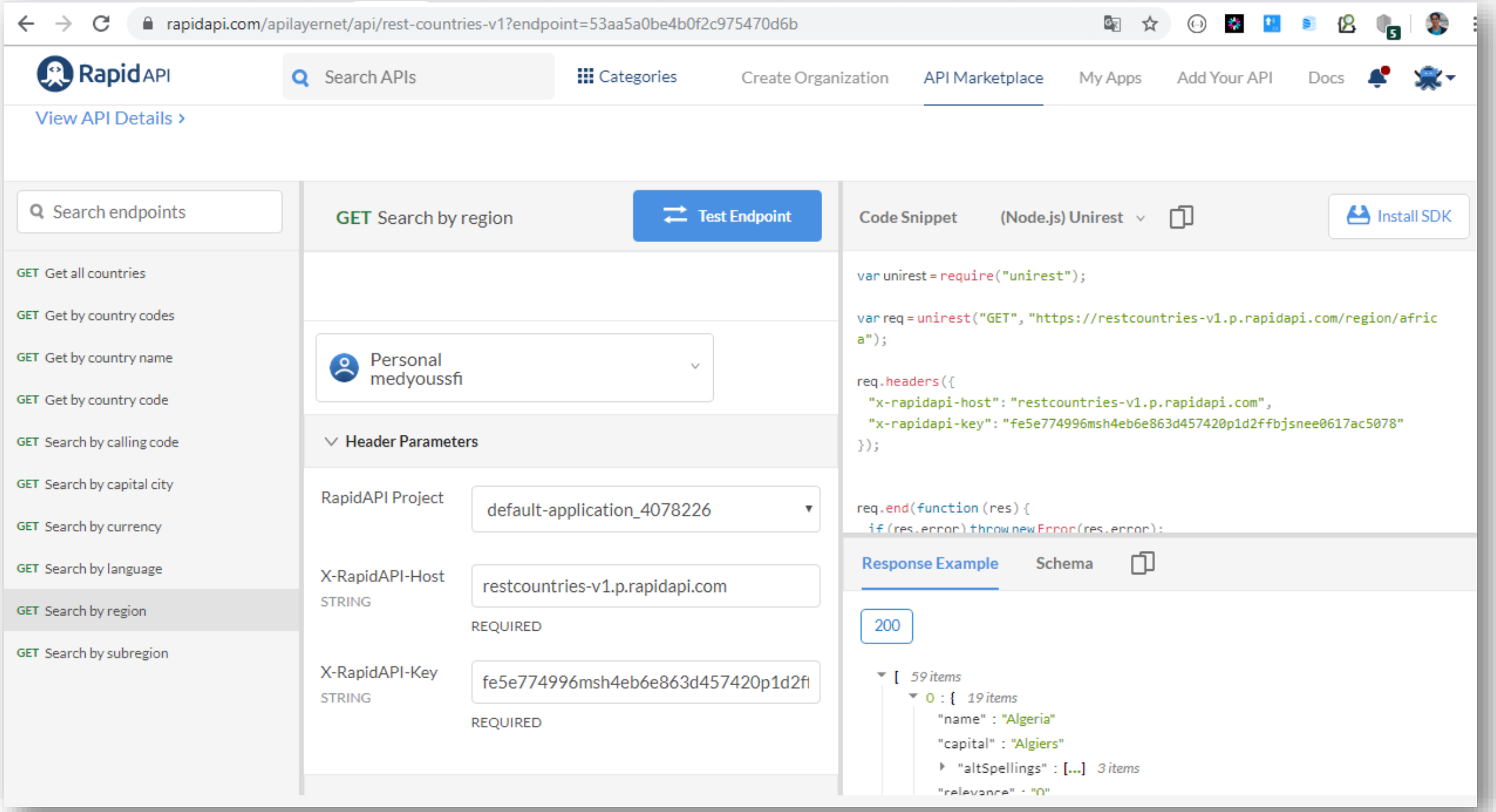

localhost:8888/restcountries/all


```
{ … }, // 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": "لمغرب
    "numericCode": "504
```


localhost:8888/restcountries/region/africa

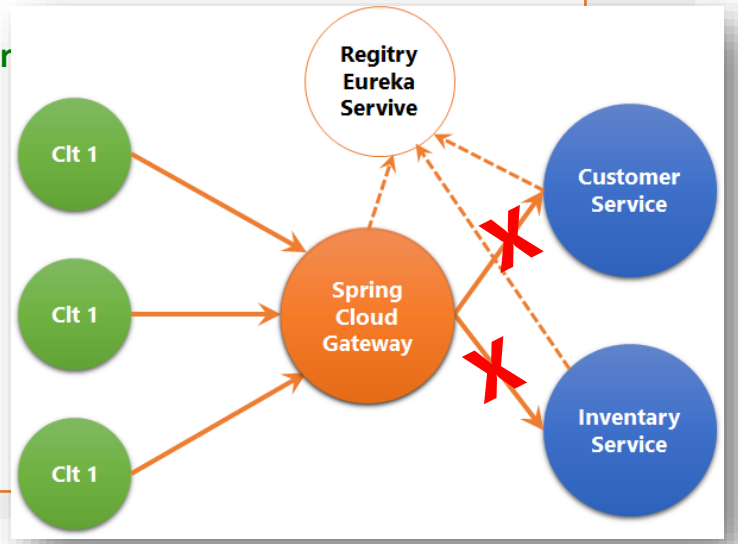

```
[
    {
        "name": "Algeria",
        "topLevelDomain": [
            ".dz"
        ],
        "alpha2Code": "DZ",
        "alpha3Code": "DZA",
        "callingCodes": [
            "213"
        ],
        "capital": "Algiers",
        "altSpellings": [
            "DZ",
            "Dzayer",
            "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"
```

Static Routes with Filters



Circuit Breaker avec Hystrix

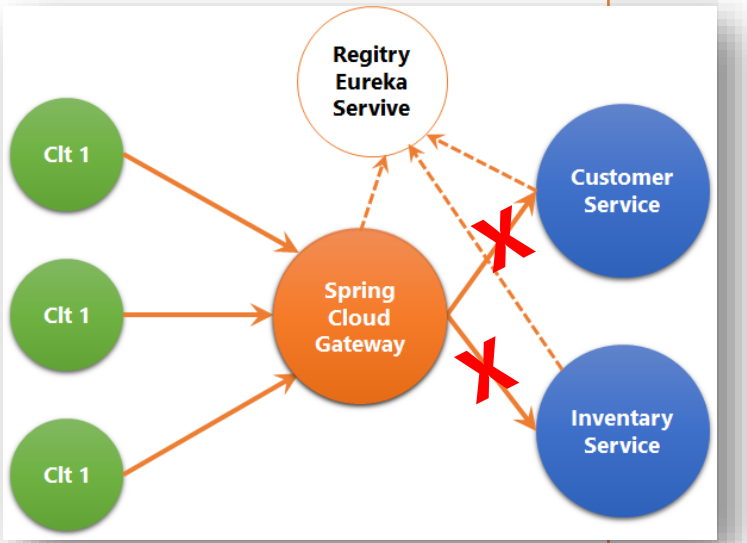
```
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
           return builder.routes()
           .route(r->r.path("/restcountries/**")
             .filters(f->f
             .addRequestHeader("x-rapidapi-host","restcountries-v1.p.rapidapi.com")
             .addRequestHeader("x-rapidapi-key", "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", "fe5e774996msh4eb6e863d457420p1d2ffbjsr
               .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

```java
@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;
        }
}
```

```java
@SpringBootApplication
@EnableHystrix
public class CloudGatewayApplication {
```



application.properties

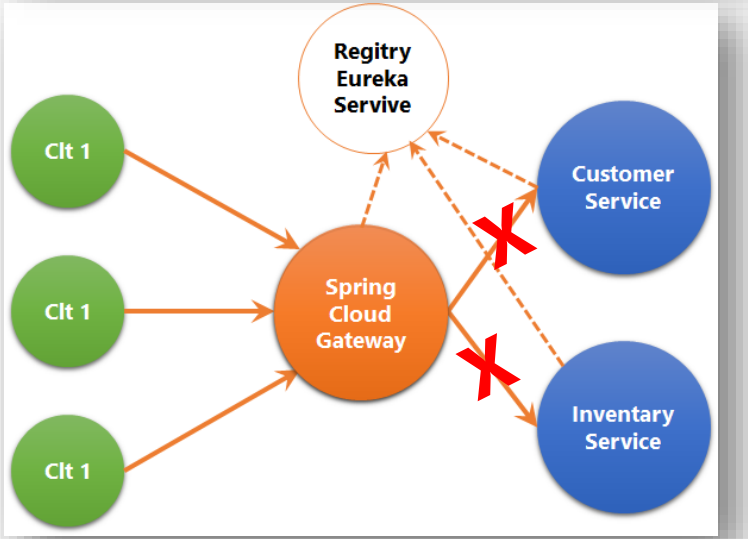
```
management.endpoints.web.exposure.include=hystrix.stream
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=1000
```

Circuit Breaker With Hystrix

Hystrix Monitor | 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"
}
```

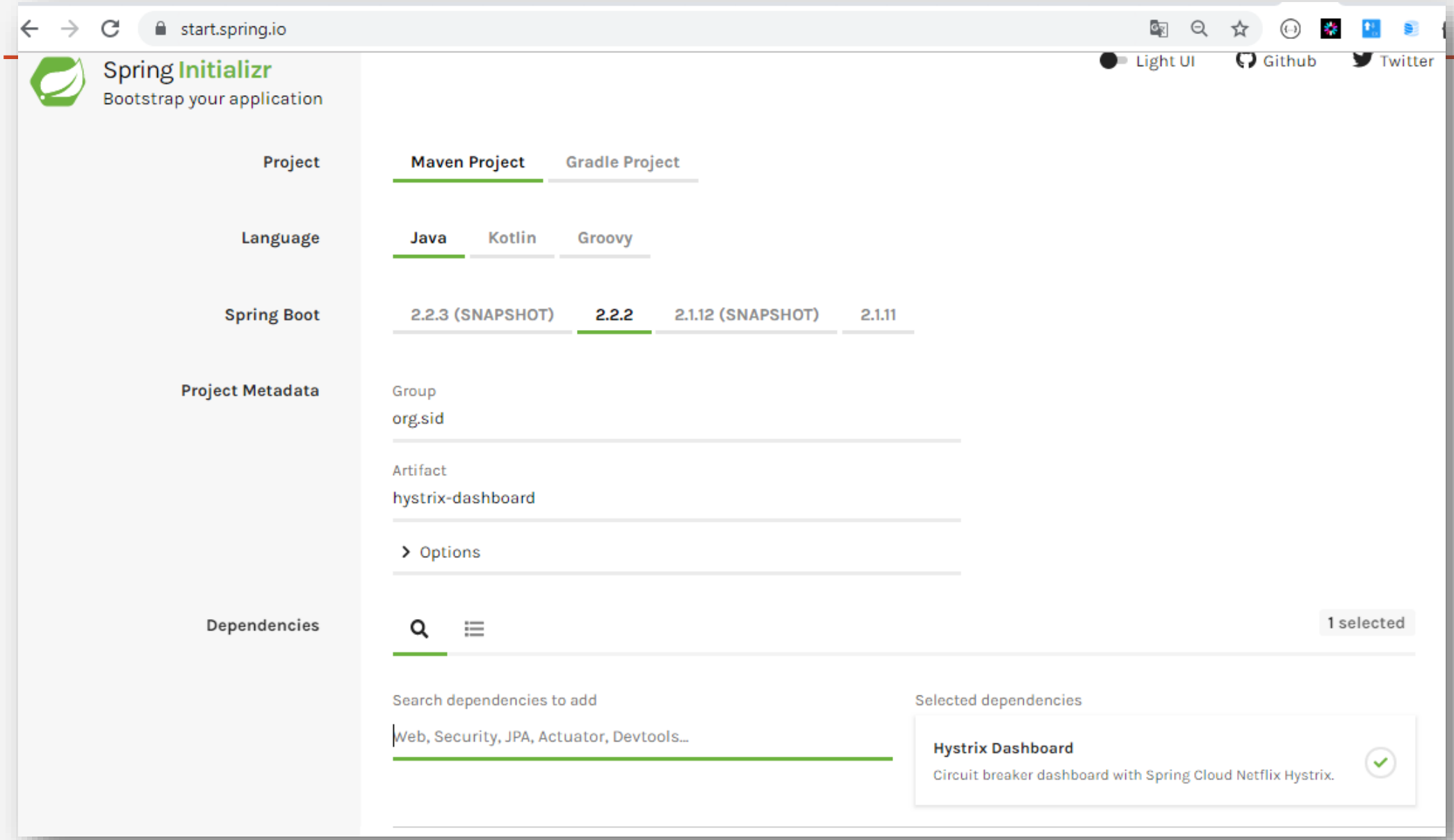

localhost:8888/actuator/hystrix.stream


```
y"}

data:{"type":"ping"}

data:{"type":"HystrixCommand","name":"rest-
countries","group":"HystrixGatewayFilterFacto
itBreakerOpen":false,"errorPercentage":0,"err
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_circuitBreakerEr
e_circuitBreakerForceOpen":false,"propertyVal
pertyValue_circuitBreakerEnabled":true,"prope
MAPHORE","propertyValue_executionIsolationThr
```


localhost:8888/restcountries/all


```
[
    {
        "name": "Afghanistan",
        "topLevelDomain": [
            ".af"
        ],
        "alpha2Code": "AF",
        "alpha3Code": "AFG",
        "callingCodes": [
            "93"
        ],
        "capital": "Kabul",
        "altSpellings": [
            "AF",
            "Afġānistān"
        ],
```

Hystrix Dashboard



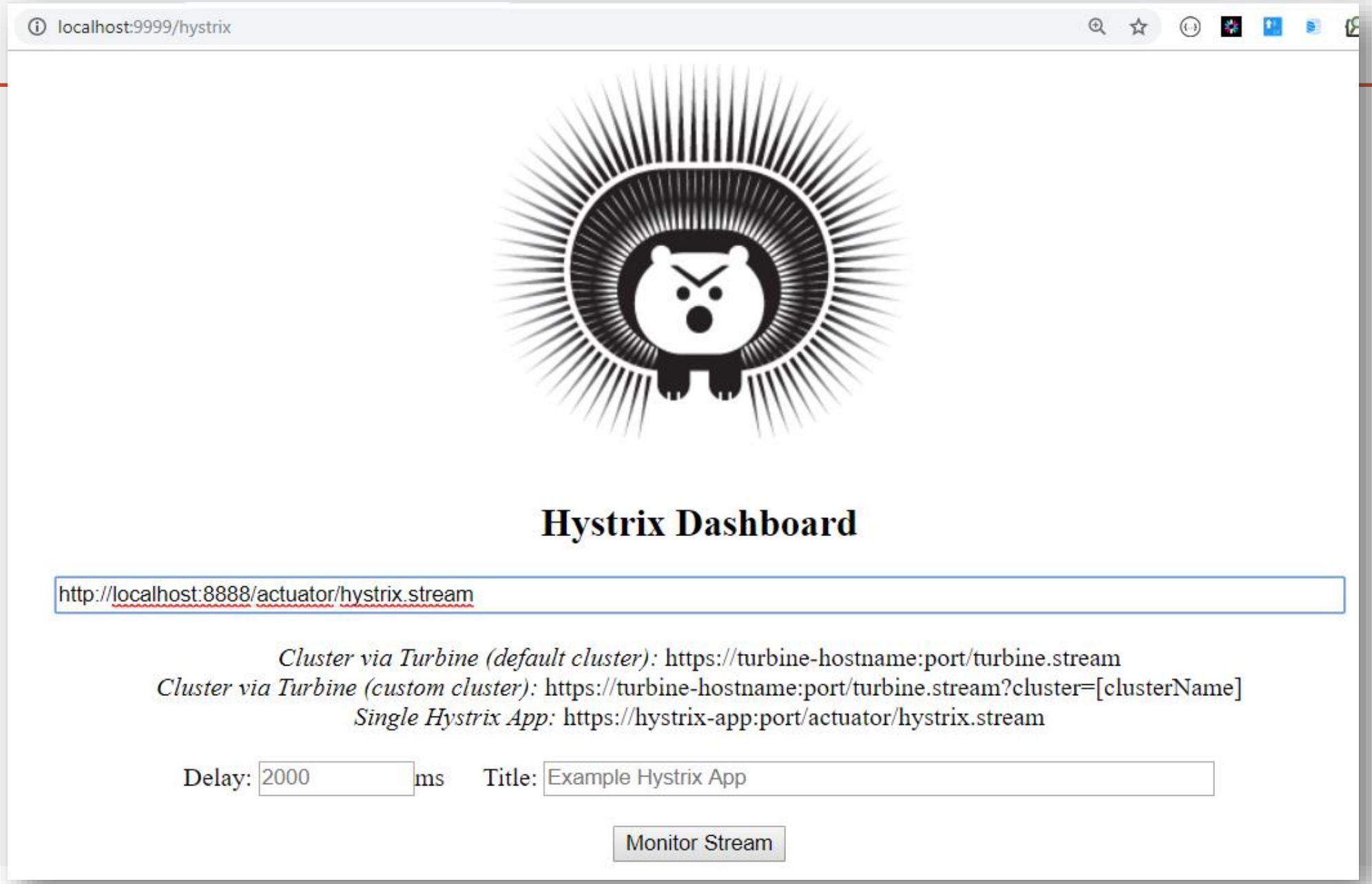
Circuit Breaker avec Hystrix

```java
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



Circuit Breaker avec Hystrix

localhost:8888/muslimsalat/rabat/5.json


```
{
    "Fajr": "07:00",
    "DOHR": "14:00",
    "message": "Default Muslim Fallback service"
}
```


localhost:9999/hystrix/monitor?stream=htt...

Hystrix Stream: http://localhost:8888/actuator/hystrix.stream

HYSTRIX
DEFEND YOUR APP

Circuit

Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

Success | Short-Circuited | Bad Request | Timeout | Rejected | Failure | Error %

muslimsalat

0 | 0 | 0.0 %
0 | 0
0

Host: 0.0/s
Cluster: 0.0/s
Circuit Closed

Hosts	1	90th	928ms
Median	502ms	99th	928ms
Mean	715ms	99.5th	928ms

rest-countries

0 | 0 | 0.0 %
0 | 0
0

Host: 0.0/s
Cluster: 0.0/s
Circuit Closed

Hosts	1	90th	546ms
Median	414ms	99th	945ms
Mean	454ms	99.5th	945ms

Thread Pools

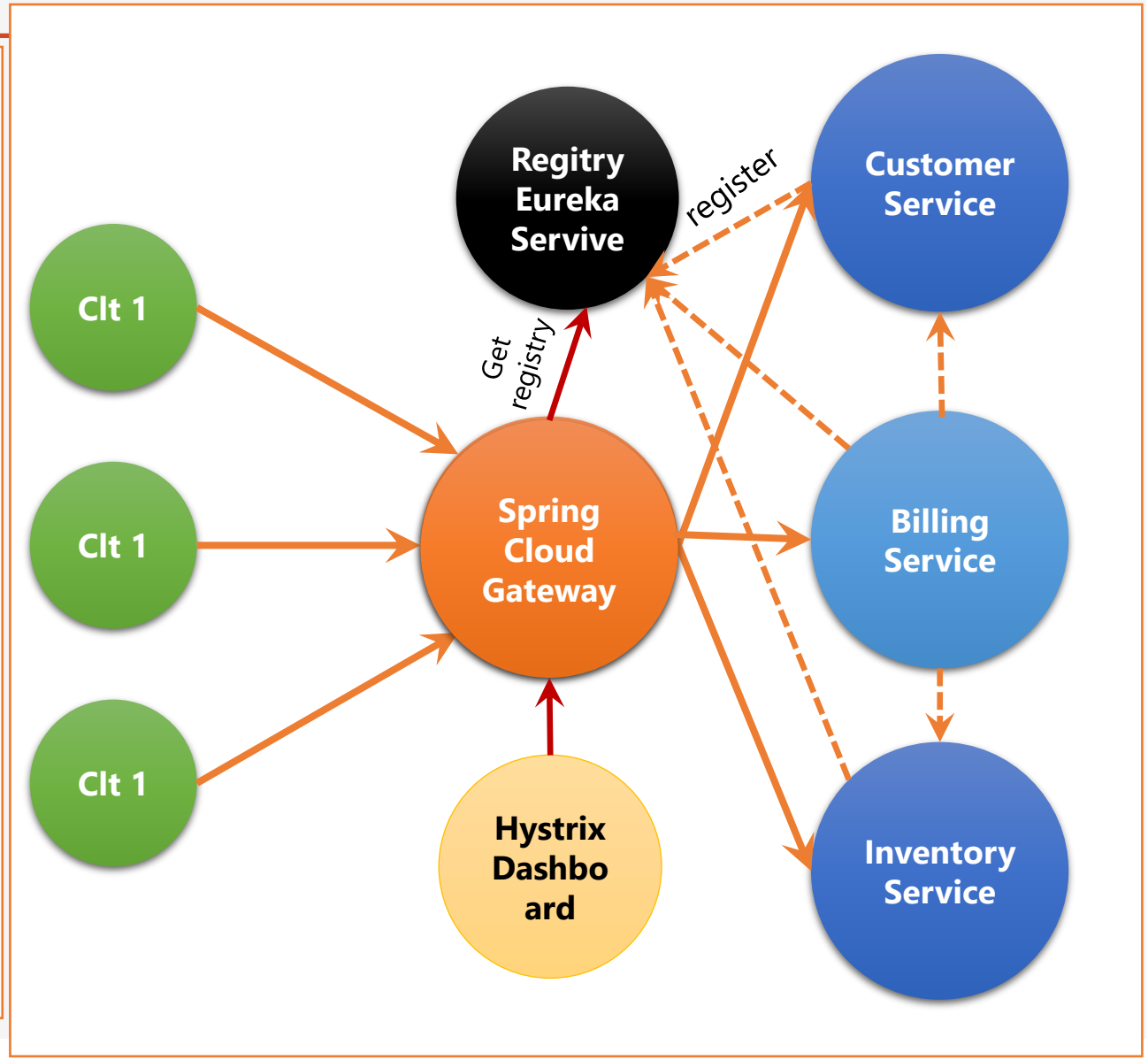
Sort: Alphabetical | Volume |

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 intéragir 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



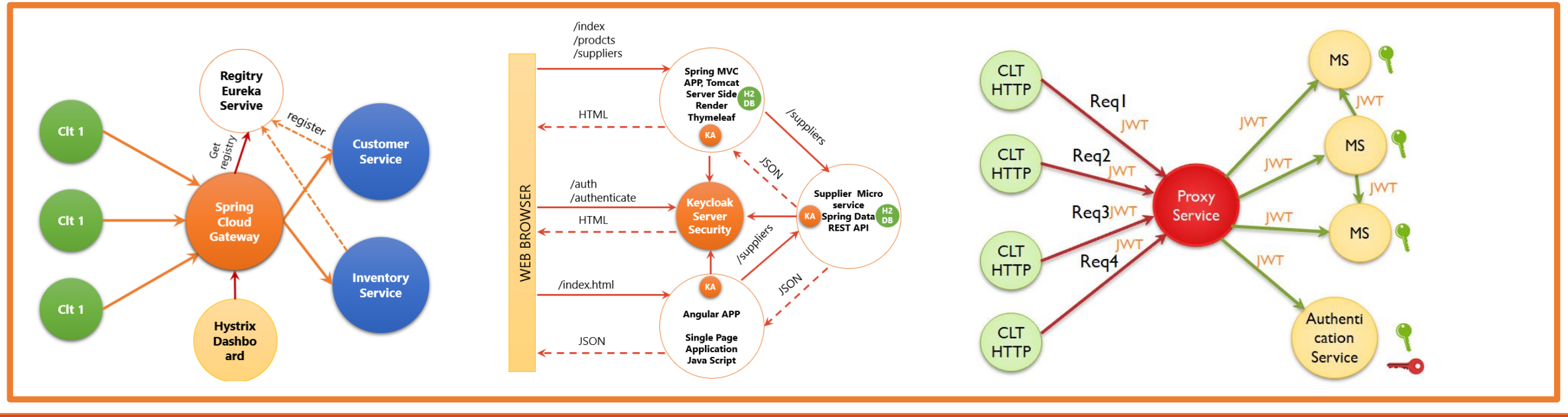
Billing-service

```
@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

```
package org.sid.billingservice;
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;
```

Distributed High Performance Computing
How to Secure Micro Services Architecture
JWT, OAuth2, Keycloack



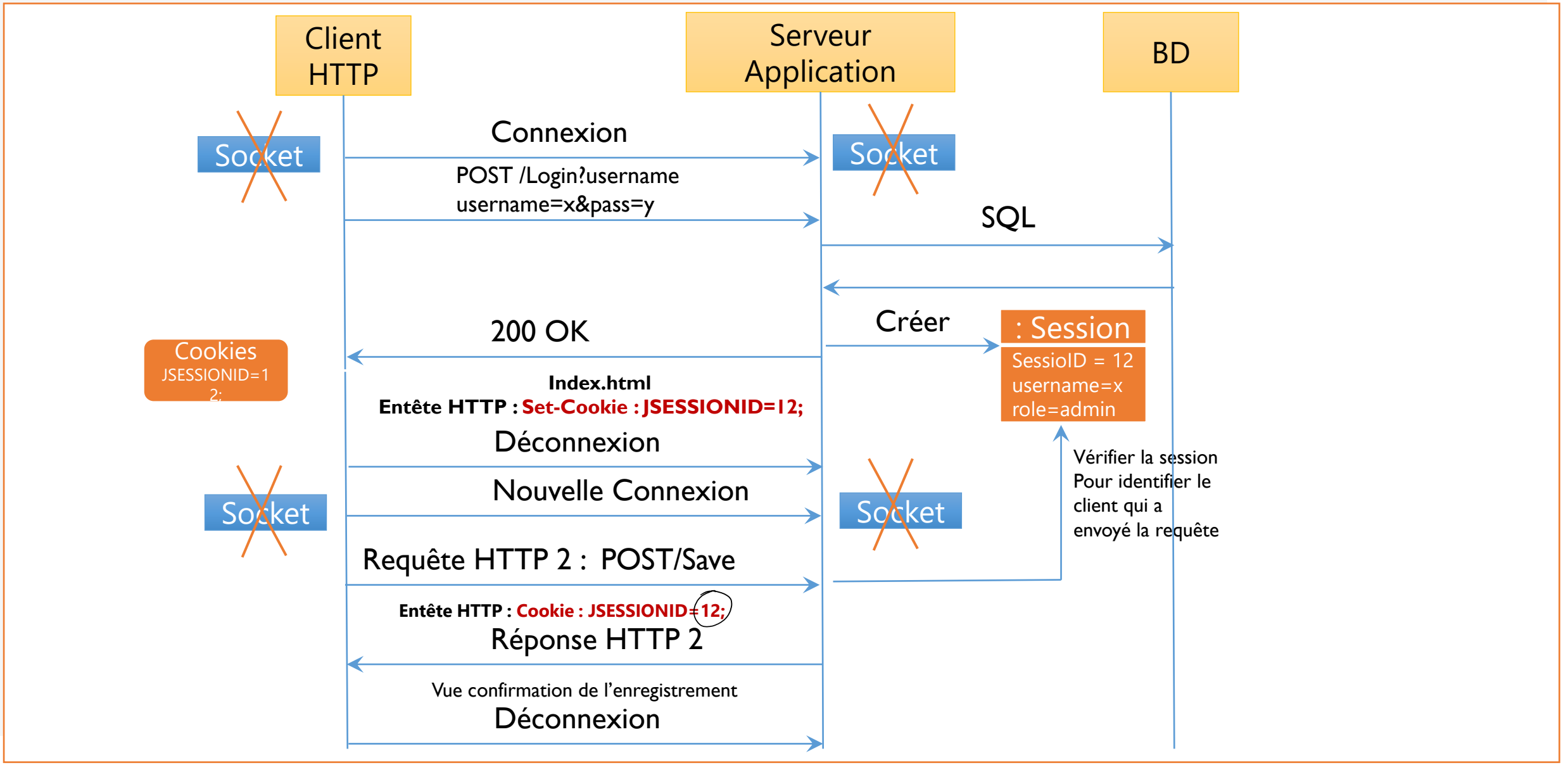
Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : http://fr.slideshare.net/mohamedyoussfi9
Chaîne vidéo : http://youtube.com/mohamedYoussfi
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Systèmes d'authentification

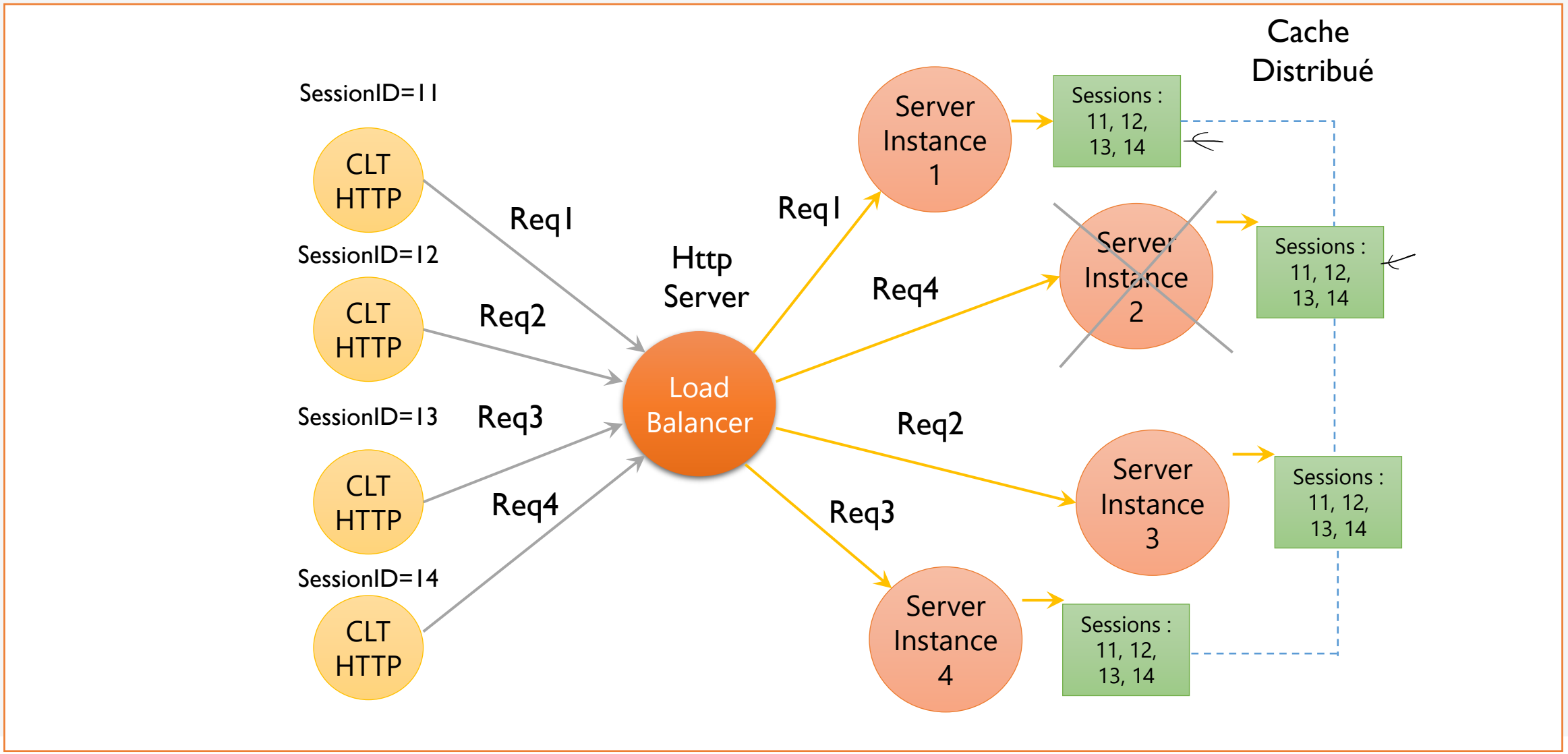
Deux types de modèles d'authentification :

- **Statful** : Les données de la session sont enregistrés coté serveur d'authentification
- **Statless** : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.

Authentification Statful basée sur les Sessions et Cookies



Problème de montée en charge : Nécessité d'un Cache Mémoire partagé ou Distribué pour les sessions



Type d’attaque : Cross Site Request Forgery (CSRF)

- En sécurité informatique, le ***Cross-Site Request Forgery***, abrégé **CSRF** est un type de vulnérabilité des services d'authentification web.
- L’objet de cette attaque est de transmettre à un utilisateur authentifié
  - Une requête HTTP falsifiée qui pointe sur une action interne au site,
  - Afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits.
  - L’utilisateur devient donc complice d’une attaque sans même s'en rendre compte.
  - L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

Cross Site Request Forgery : CSRF

- Exemple d'attaque CSRF :
  - Yassine possède un compte bancaire d'une banque qui lui donne entre autres la possibilité d'effectuer en ligne des virements de son compte vers d'autres comptes bancaires.
  - L'application de la banque utilise un système d'authentification basé uniquement sur les sessions dont les SessionID sont stockées dans les cookies.
  - Sanaa possède également un compte dans la même banque. Elle connait facilement la structure du formulaire qui permet d'effectuer les virements.
  - Sanaa envoie à Yassine, un email contenant un message présentant un lien hypertexte demandant Yassine de cliquer sur ce lien pour découvrir son cadeau d'anniversaire.
  - Ce message contient également un formulaire invisible permettant de soumettre les données pour effectuer un virement. Ce formulaire contient entre autres des champs cachés :
    - Le montant du virement à effectuer vers
    - Le numéro de compte de Sanaa
  - Au moment ou Yassine reçoie son message, la session de son compte bancaire est ouverte. Son Session ID est bien présent dans les cookies.
  - En cliquant sur le lien de l'email, Yassine, vient d'effectuer un virement de son compte vers celui de Sanaa, sans qu'il le sache.
  - En effet, en cliquant sur le lien, les données du formulaire caché sont envoyées au bon script coté serveur, et comme les cookies sont systématiquement envoyées au serveur du même domaine, le serveur autorise cette opération car, pour lui, cette requête vient d'un utilisateur bien authentifié ayant une session valide.

Préventions des attaques CSRF

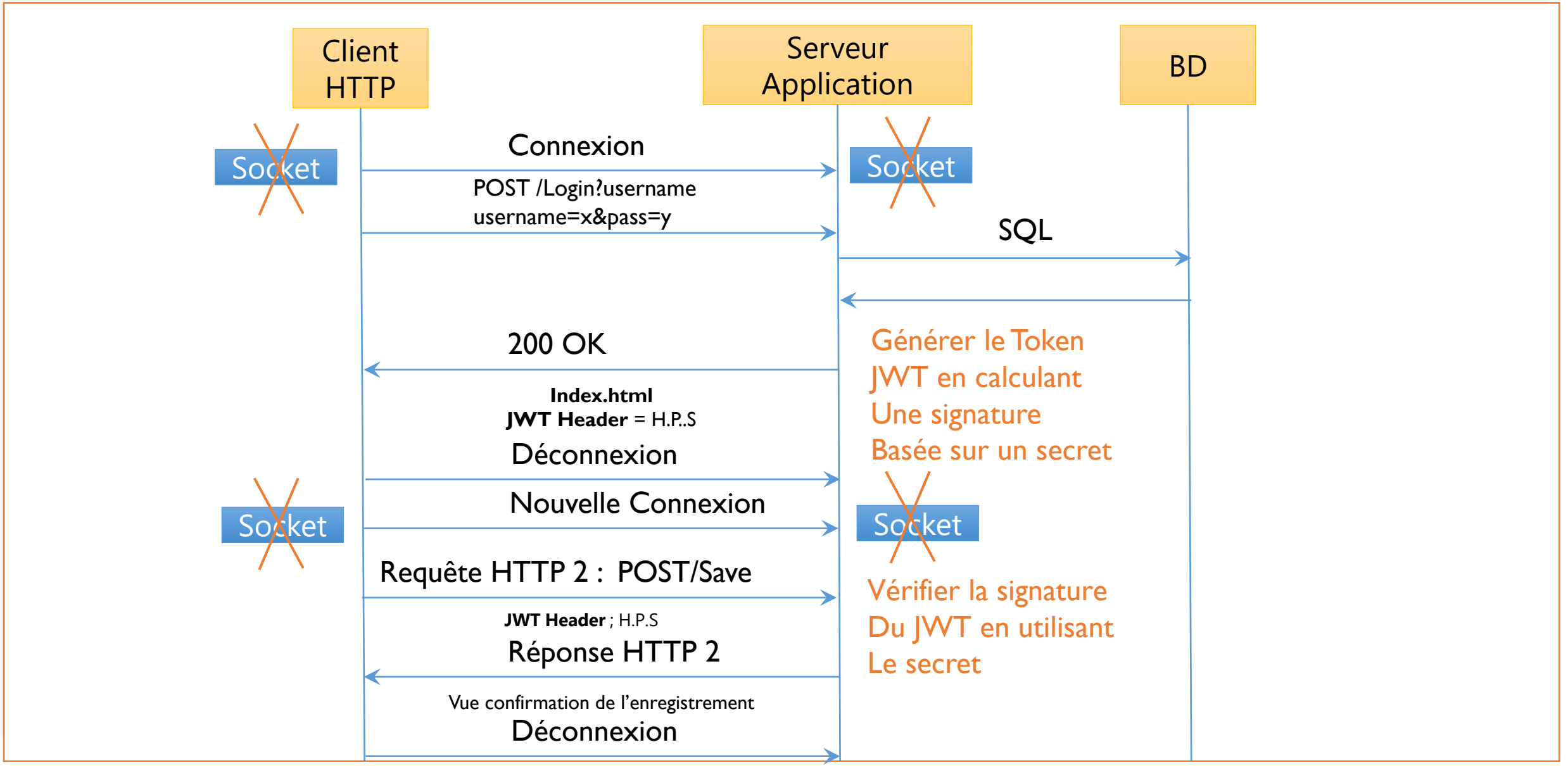
Utiliser des jetons de validité (**CSRF Synchronizer Token**) dans les formulaires :

- Faire en sorte qu'un formulaire posté ne soit accepté que s'il a été produit quelques minutes auparavant. Le jeton de validité en sera la preuve.
- Le jeton de validité doit être transmis souvent en paramètre (Dans un champs de type Hidden du formulaire) et vérifié côté serveur.

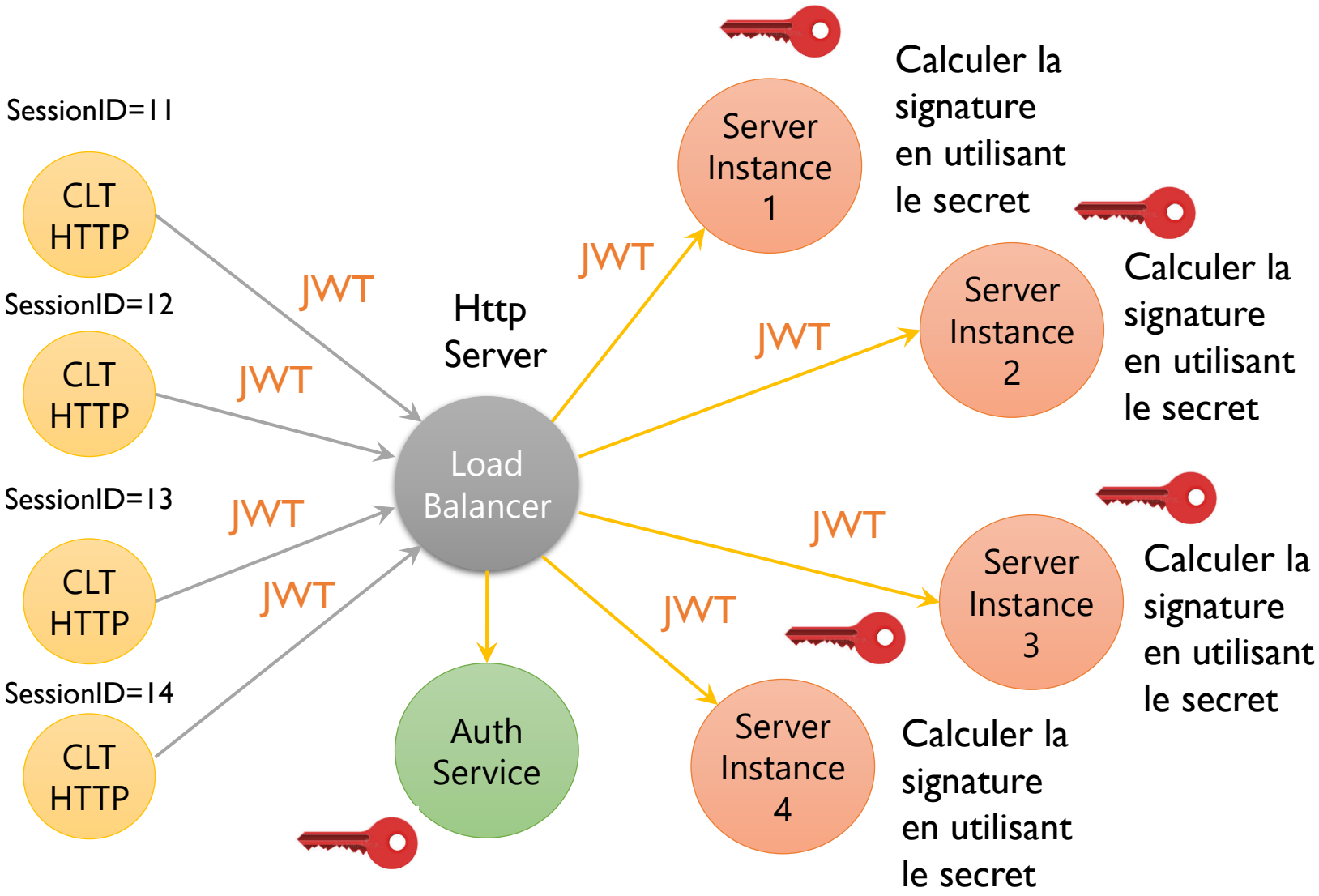
Autres présentions :

- Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions critiques de modification des données (Ajout, Mise à jour, Suppression) : cette technique va naturellement éliminer des attaques simples basées sur les liens hypertexte, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.
- Demander des confirmations à l'utilisateur pour les actions critiques, au risque d'alourdir l'enchaînement des formulaires.
- Demander une confirmation de l'ancien mot de passe à l'utilisateur pour changer celui-ci ou utiliser une confirmation par email ou par SMS.
- Effectuer une vérification du référent dans les pages sensibles : connaître la provenance du client permet de sécuriser ce genre d'attaques. Ceci consiste à bloquer la requête du client si la valeur de son référent est différente de la page d'où il doit théoriquement provenir.

Authentification Statless basée sur les Sessions et Cookies



Authentification Statless:
Problème de montée en charge : Pas de besoin de cache partagé ou distribué



Json Web Token (JWT)

- JSON Web Token (JWT) est un standard (RFC 7519) qui définit une solution **compacte** et **autonome** pour transmettre de manière sécurisée des informations entre les applications en tant qu'objet structuré au format JSON (Java Script Object Notation).
  - Compact : en raison de leur petite taille, les JWT peuvent être envoyés via une URL, un paramètre POST ou dans un en-tête HTTP. De plus, la plus petite taille signifie que la transmission est rapide.
  - Autonome : Le JWT contient toutes les informations requises sur l'utilisateur, ce qui évite d'avoir à interroger la base de données plus d'une fois pour connaitre le détail de l'identité d'un client authentifié.
- Le JWT est fiable car il est signé numériquement.
- JWT est constitué de trois parties séparées par un point « . » :
  - Header
  - Payload
  - Signature

La forme d'un JWT est donc :

- xxx.yyy.zzz

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL2F1dGgiLCJhdWQiOlsiV2ViIEZyb250IEVuZCIsIk1vYmlsZSBBcHAiXSwiZXhwIjo1NDc4OTAwNSwibmJmIjpudWxsLCJpYXQiOjQ5ODY1NDMyLCJqdGkiOiJpZHI1NjU0M2Z0dTg5MDk4NzYiLCJuYW1lIjoibWVkIiwicm9sZXMiOlsiYWRtaW4iLCJhdXRob3IiXX0.-V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0
```

Header

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

Encodage Base 64 URL

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

```
{
  "sub": "1234567890",
  "iss":"http://localhost:8080/auth",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Encodage Base 64 URL

eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIg ....
vciJdfQ

Signature : RSA (2 clés publiques et privée) ou HMAC (1 clé privée)

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0

JWT : Header

- L'en-tête se compose généralement de deux parties:
  - Le type du jeton, qui est JWT,
  - L'algorithme de hachage utilisé, tel que :
    - **HMAC** (**H**ash **M**essage **A**uthentication **C**ode) : Symétrique (Clé privé)
    - **RSA** (**R**onald Rivest (2015), Adi **S**hamir (2013) et Leonard **A**dleman (2010).) : Asymétrique avec clés Publique et Clé Privée
- La structure du Header est un objet JSON ayant la forme la suivante :

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Cet objet JSON est ensuite encodé en Base64URL. Ce qui donne la forme suivante :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

JWT : Payload

La deuxième partie du jeton est le Payload, qui contient les claims (revendications.)

Les claims sont des déclarations concernant :

- Une entité (généralement l'utilisateur)
- Des métadonnées supplémentaires.

Il existe trois types de claims :

- Enregistrées (Registred)
- publiques (Public)
- Privées (Private)

JWT : Payload

Registred Claims :

- Il s'agit d'un ensemble de revendications prédéfinies qui ne sont pas obligatoires mais recommandées pour fournir un ensemble de revendications utiles et interopérables. Certains d'entre eux sont:
  - iss (issuer : Origine du token),
  - exp (heure d'expiration),
  - sub (sujet),
  - aud (public cible),
  - nbf (Not Before : A ne pas utiliser avant cette date)
  - iat (issued at : date de création du token).
  - jti (JWT ID identifiant unique du JWT).

JWT : Payload

Public Claims :

- Celles-ci peuvent être définies à volonté par ceux qui utilisent des JWT.
- Mais pour éviter les collisions, elles doivent être définies dans le registre des jetons Web IANA JSON (IANA JSON Web Token Registry : https://www.iana.org/assignments/jwt/jwt.xhtml) ou être définies comme des URI contenant un espace de noms pour éviter les confusions.

JWT : Payload

Private Claims :

- Il s'agit des claims personnalisés créées pour partager des informations entre des parties qui acceptent de les utiliser et ne sont ni des réclamations enregistrées ni des réclamations publiques.

JWT : Exemple de Payload

```
{
  "sub": "1234567890",
  "iss":"Backend",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Le payload est encodé en Base64URL. Ce qui donne :

eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIgRnJvbnQgRW5kIiwiTW9iaWxlIEFwcCJdLCJleHAiOjU0Nzg5MDA1LCJuYmYiOm51bGwsImlhdCI6NDk4NjU0MzIsImp0aSI6ImlkcjU2NTQzZnR1ODkwOTg3NiIsIm5hbWUiOiJtZWQiLCJyb2xlcyI6WyJhZG1pbiIsImF1dGhvciJdfQ

JWT : Signature

La signature est utilisée pour :

- vérifier que l'expéditeur du JWT est celui qu'il prétend être.
- et pour s'assurer que le message n'a pas été modifié en cours de route.

Par exemple si vous voulez utiliser l'algorithme HMAC SHA256, la signature sera créée de la façon suivante:

- HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

jwt.io

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

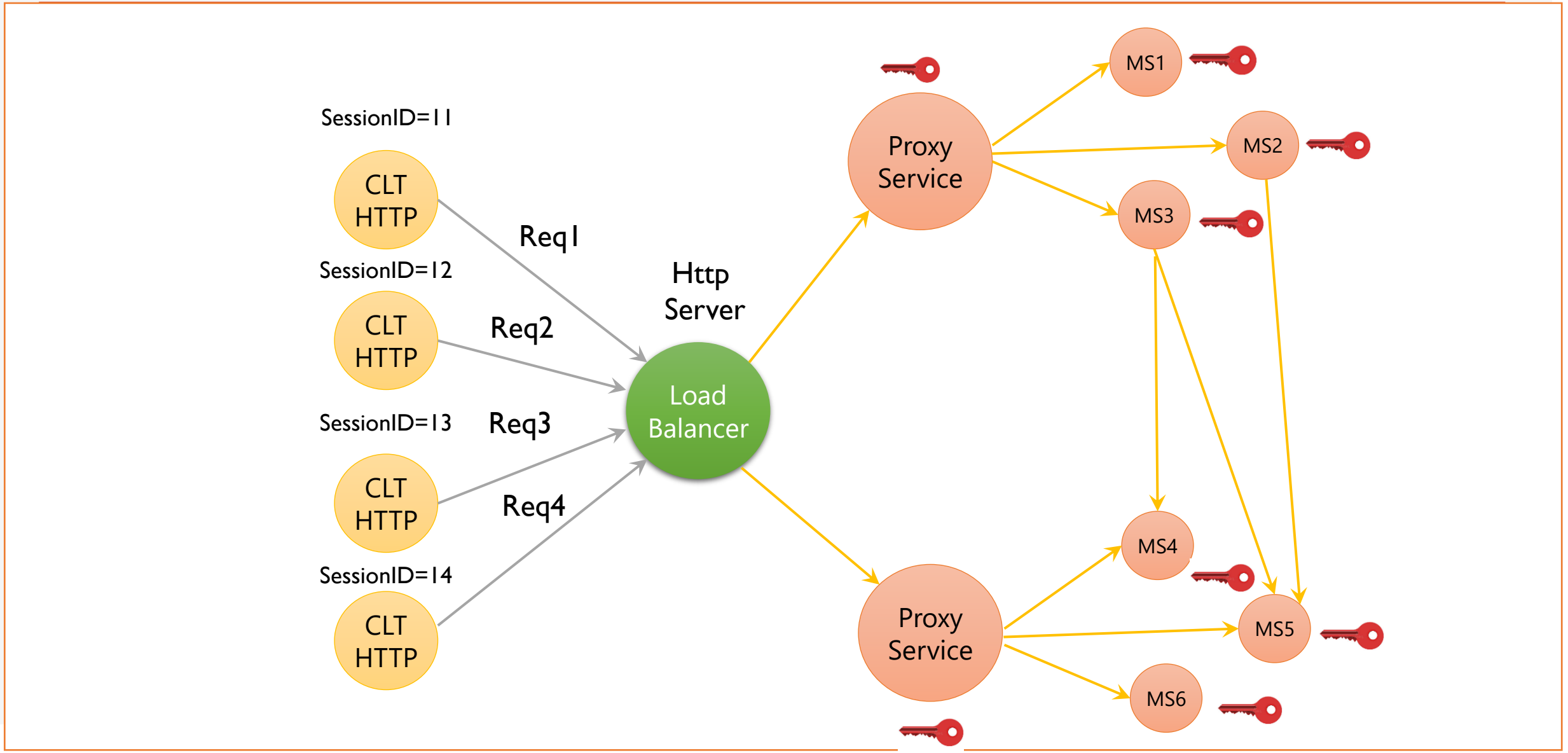
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

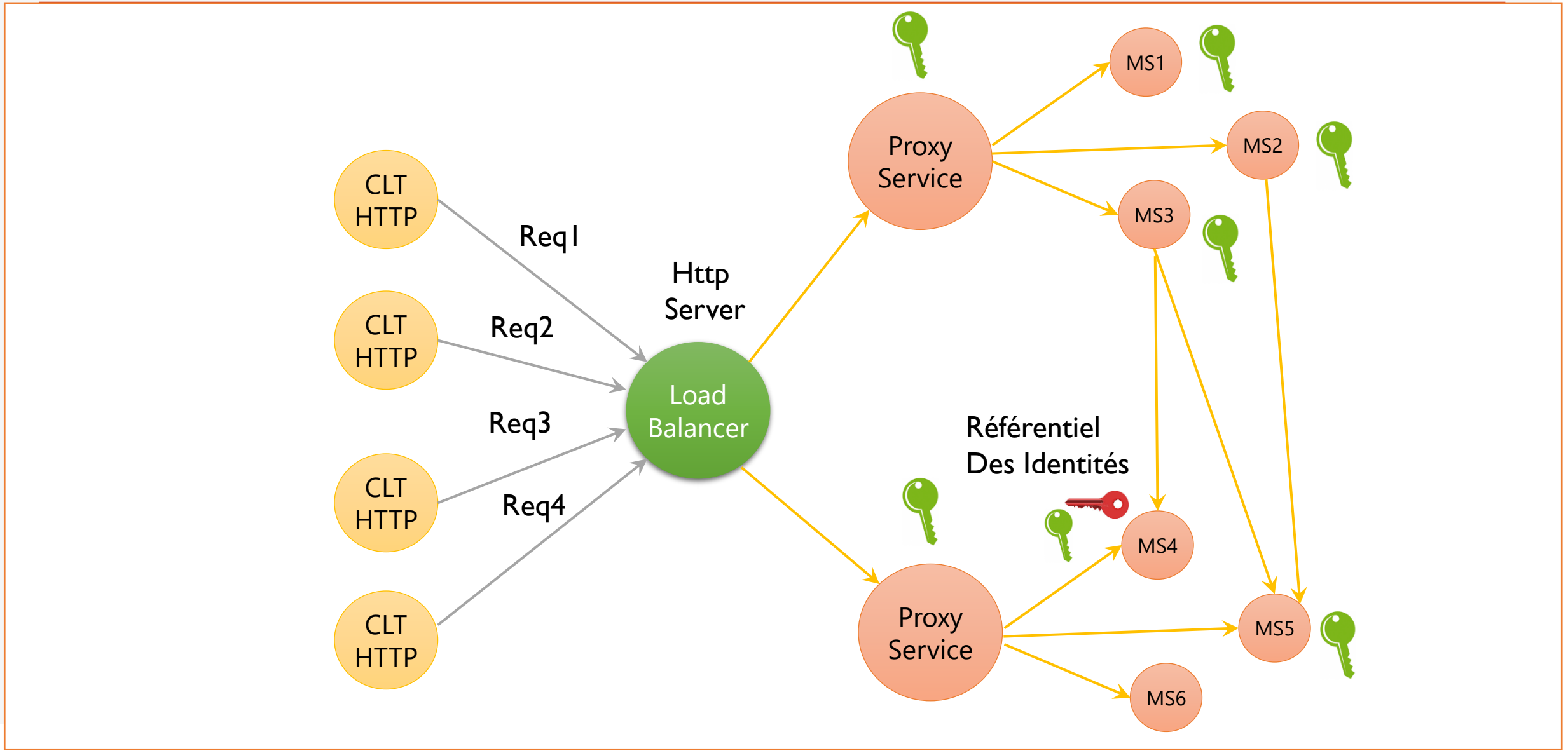
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

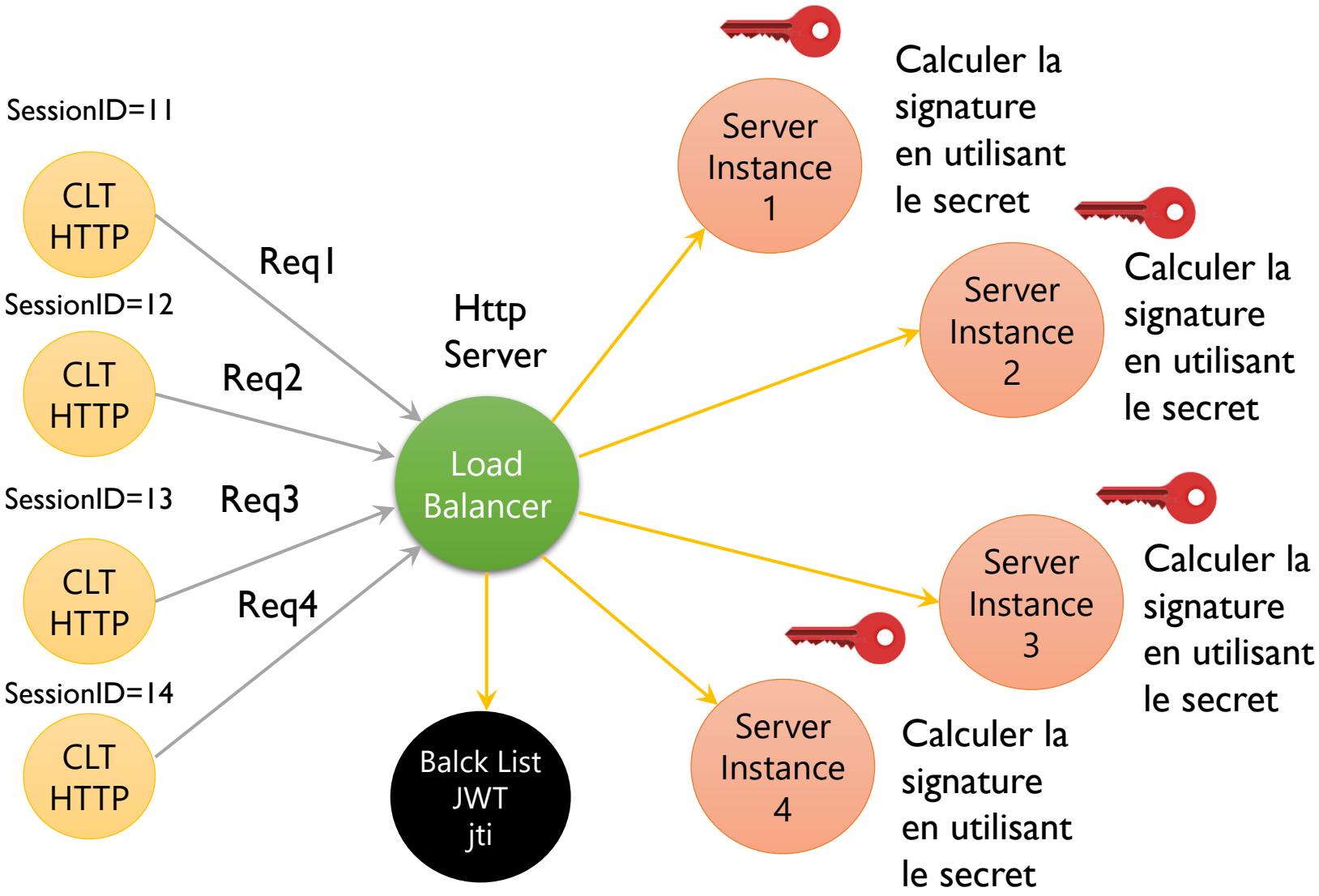
Bonne solution pour les architectures distribuées basées sur les micro services



JWT dans un système distribué avec clé privé et clé publique

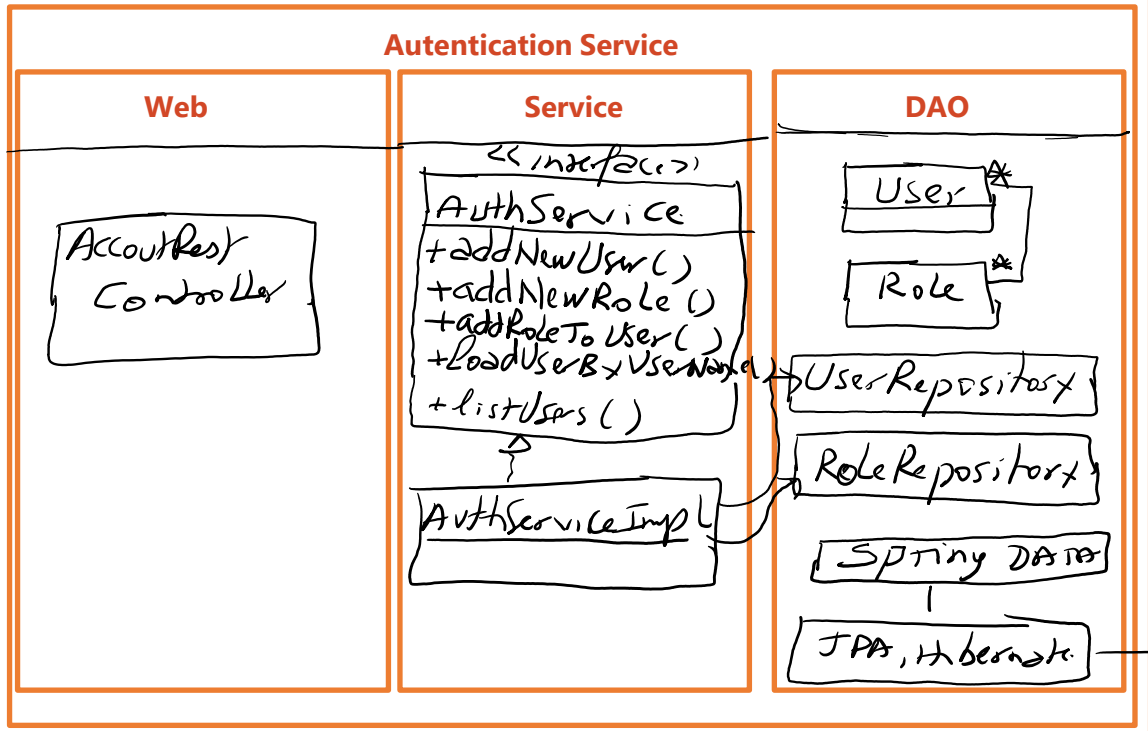


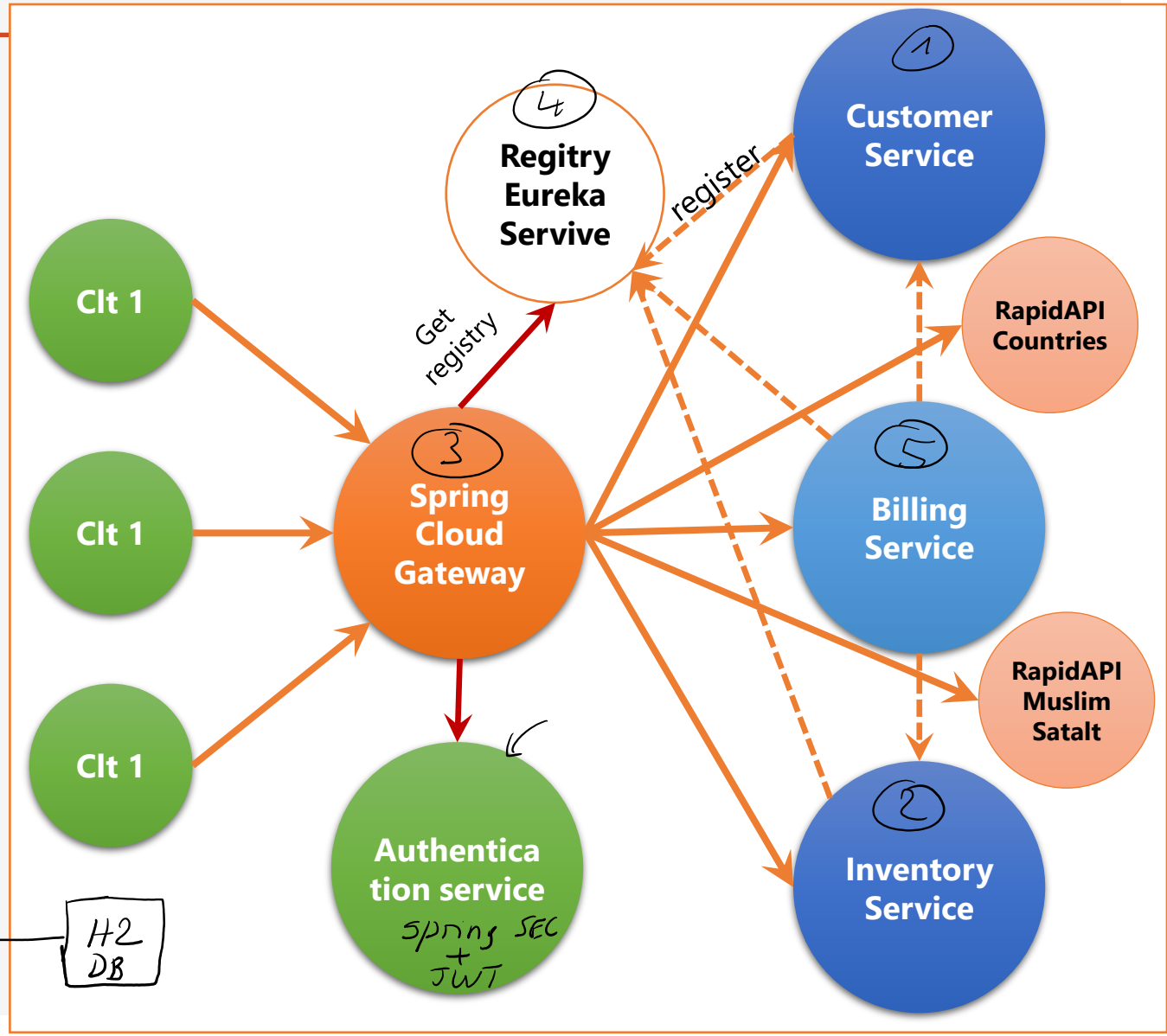
Problème de Révocation des Tokens



Service d'authentification avec Spring Security et JWT

- Créer un micro service d'authentification en utilisant Spring Security et JWT
- Ce services permet de gérer
  - Les utilisateurs
  - Les rôles (USER, ADMIN, CUSTOMER_MANAGER, PRODUCT_MANAGER, BILLS_MANAGER)
  - Un utilisateur peut avoir plusieurs rôles et chaque rôle peut être affecté à plusieurs utilisateurs

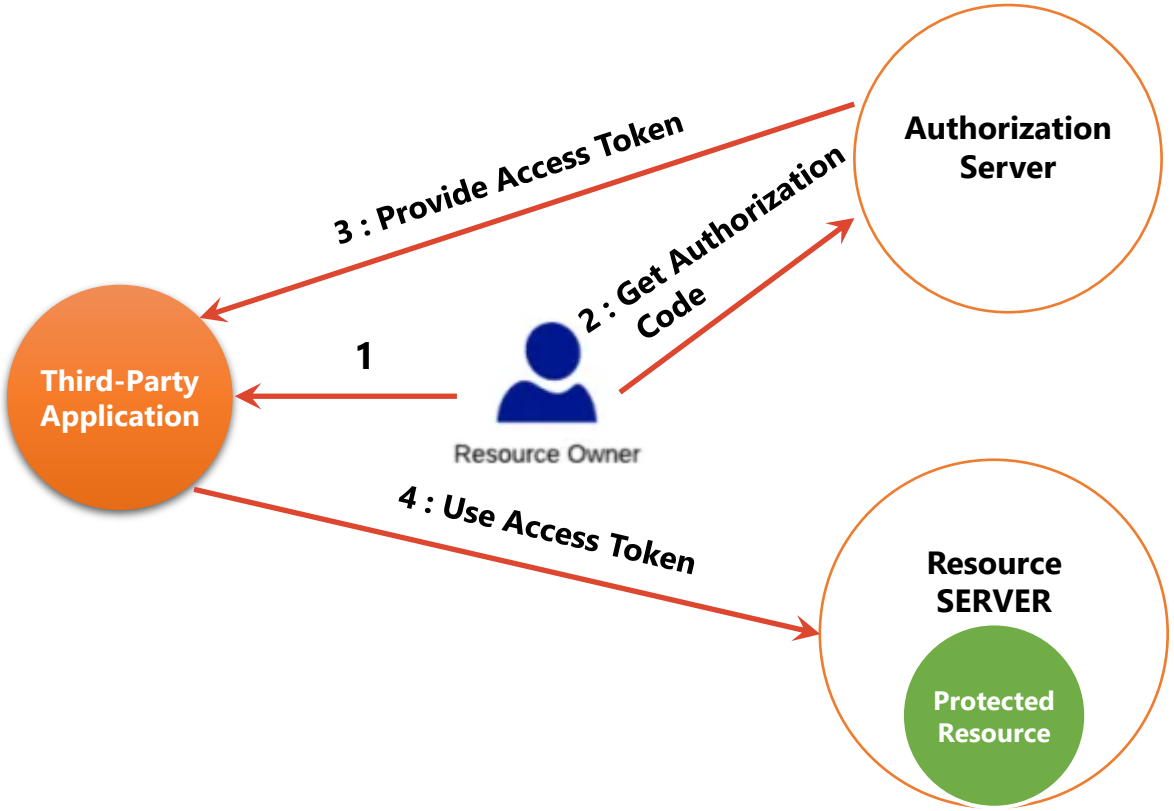




Sécurité des Micro-services avec OAuth2

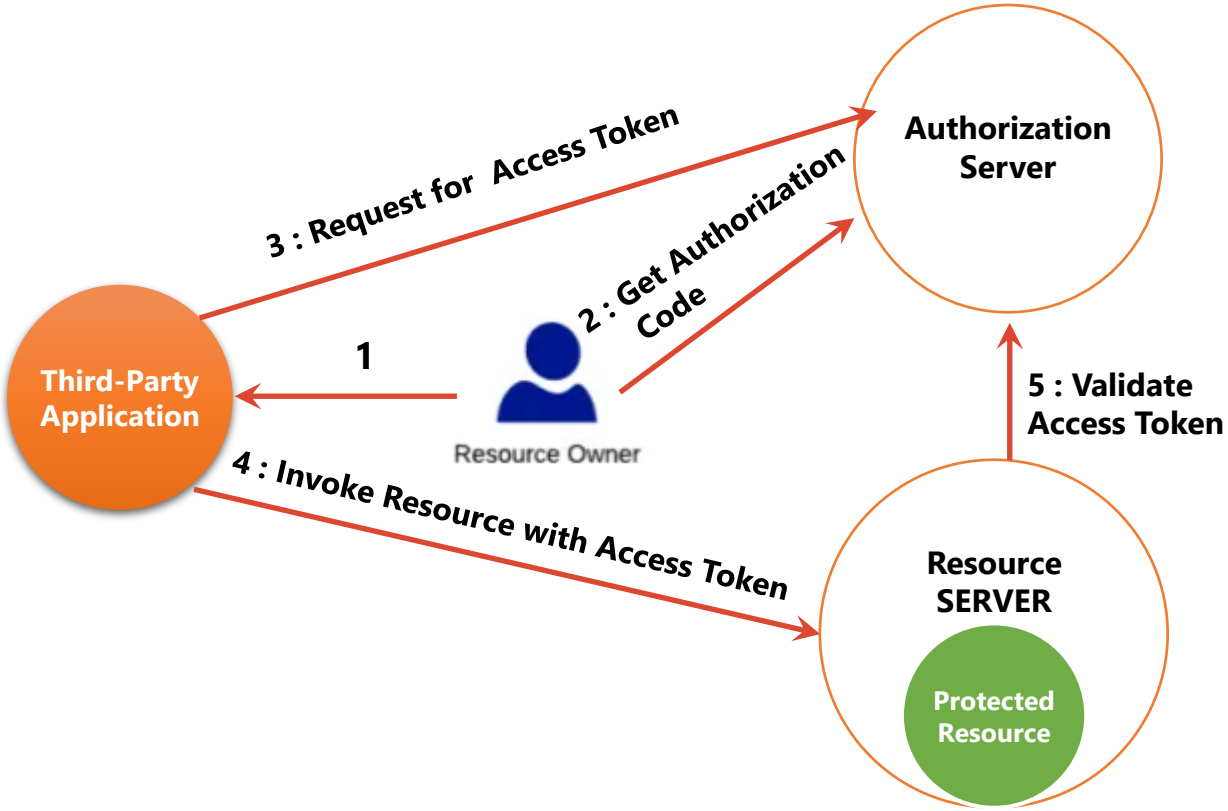
OAuth2

- Protocole de délégation d'autorisations
- Architecture à trois parties
  - Un utilisateur qui possède des ressources exposées par un service en ligne et veut gérer les autorisations d'accès à ces ressources à d'autres applications tierces,
  - Exemple : Autorisation d'une application quelconque X d'accéder aux données de mon compte GitHub, sans obliger l'utilisateur à saisir son mot de passe dans les applications tierce X. Le mot de passe ne doit pas être donné qu'à GitHub.
- Protocole très populaire utilisé par les géants du Web : Facebook, Google, Twiter, etc…
- OAuth2 n'est pas un SSO (Single Sign On et Single Sign Out)
- OAuth2 n'est pas un système de gestion des identités



OpenID Connect 1.0

- Différent de OpenID (1 et 2)
- Développé par OpenID Fundation
- OpenID Connect Core : Novembre 2014
- Construit pour s'adapter aux nouvelles tendances applicatives :
  - Web Server Side
  - Web Client Side
  - Signe Page Application
  - Mobile Applications
  - Systèmes Distribués
  - Web services
  - Micro Services
- Couche au dessus de OAuth2 pour la gestion des identités.
- Introduit le concept de ID Token qui utilise JWT (Json Web Token)



Json Web Token (JWT)

- JSON Web Token (JWT) est un standard (RFC 7519) qui définit une solution **compacte** et **autonome** pour transmettre de manière sécurisée des informations entre les applications en tant qu'objet structuré au format JSON (Java Script Object Notation).
- Cette information peut être vérifiée et fiable car elle est signée numériquement.
- JWT est constitué de trois parties séparées par un point « . » :
  - Header
  - Payload
  - Signature

La forme d'un JWT est donc :

- xxx.yyy.zzz

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL2F1dGgiLCJhdWQiOlsiV2ViIEZyb250IEVuZCIsIk1vYmlsZSBBcHAiXSwiZXhwIjo1NDc4OTAwNSwibmJmIjpudWxsLCJpYXQiOjQ5ODY1NDMyLCJqdGkiOiJpZHI1NjU0M2Z0dTg5MDk4NzYiLCJuYW1lIjoibWVkIiwicm9sZXMiOlsiYWRtaW4iLCJhdXRob3IiXX0.-V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0
```

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Encodage Base 64 URL

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

```
{
  "sub": "1234567890",
  "iss":"http://localhost:8080/auth",
  "aud":["Web Front End","Mobile App"],
  "exp":54789005,
  "nbf":null,
  "iat":49865432,
  "jti":"idr56543ftu8909876",
  "name": "med",
  "roles":["admin","author"]
}
```

Encodage Base 64 URL eyJzdWIiOiIxMjM0NTY3ODkwIiwiaXNzIjoiQmFja2VuZCIsImF1ZCI6WyJXZWIg .... vciJdfQ

Signature : RSA (2 clès publiques et privée) ou HMAC (1 clé privée)

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

V4FXAPpIBx1HqONU6qp7ptqzq32RrolDlhj4cVUQV0

Oauth 2 Authorization Server

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>org.sid</groupId>
<artifactId>oauth2-server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>oauth2-server</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>2020.0.0-M5</spring-cloud.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-security</artifactId>
    </dependency>
```

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

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

<repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
    </repository>
</repositories>
```

Oauth 2 Authorization Server

```
server.port=8082
#spring.security.user.name=med
#spring.security.user.password=1234
#spring.security.user.roles=USER,ADMIN
#security.oauth2.client.client-id=mobile
#security.oauth2.client.client-secret=pin
#security.oauth2.client.access-token-validity-seconds=3600
#security.oauth2.client.authorized-grant-
types=refresh_token,authorization_code,password,client_credentials
#security.oauth2.client.scope=READ,WRITE
#security.oauth2.authorization.check-token-access=permitAll
```

```
package org.sid.oauth2server;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

@SpringBootApplication
@EnableAuthorizationServer
public class Oauth2ServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(Oauth2ServerApplication.class, args);
    }
    @Bean
    BCryptPasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

}
```

Oauth 2 Authorization Server

```
package org.sid.oauth2server.config;
import org.springframework.beans.factory.annotation.Autowired;import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.DelegatingPasswordEncoder;import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;
@Configuration
public class OAuth2ServerConfig  extends WebSecurityConfigurerAdapter implements AuthorizationServerConfigurer {
    @Autowired
    BCryptPasswordEncoder passwordEncoder;
    @Override
    public void configure(AuthorizationServerSecurityConfigurer securityConfigurer) throws Exception {
        securityConfigurer.checkTokenAccess("permitAll()");
    }
    @Override
    public void configure(ClientDetailsServiceConfigurer client) throws Exception {
        client
                .inMemory()
                .withClient("mobile").secret(passwordEncoder.encode("1234"))
                .authorizedGrantTypes("password","authorization_code")
                .scopes("READ","WRITE");
    }
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.authenticationManager(authenticationManagerBean());
    }
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

Oauth 2 Authorization Server

```
package org.sid.oauth2server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.authentication.configuration.GlobalAuthenticationConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class UserConfiguration extends GlobalAuthenticationConfigurerAdapter {
    @Autowired
    PasswordEncoder passwordEncoder;
    @Override
    public void init(AuthenticationManagerBuilder auth) throws Exception {
      auth.inMemoryAuthentication()
              .withUser("admin").password(passwordEncoder.encode("1234"))
              .roles("USER","ADMIN","MANAGER")
               .authorities("CAN_READ","CAN_WRITE","CAN_DELETE","CAN_UPDATE").and()
            .withUser("user").password(passwordEncoder.encode("1234"))
              .roles("USER")
              .authorities("CAN_READ");
    }
}
```

Authentification et validation du token

Authentification

```
curl -d "grant_type=password&username=user&password=1234" -X POST http://localhost:8082/oauth/token -H "Content-Type: application/x-www-form-urlencoded" -u mobile:1234
```


```
{
 "access_token":"6ab5a186-294b-4046-8242-1af0134540db",
 "token_type":"bearer",
 "expires_in":40903,
 "scope":"READ WRITE"
}
```


Validation du token

curl http://localhost:8082/oauth/check_token?token=6ab5a186-294b-4046-8242-1af0134540db

{"active":true,"exp":1606023818,"user_name":"user","authorities":["CAN_READ"],"client_id":"mobile","scope":["READ","WRITE"]}

curl http://localhost:8082/oauth/check_token?token=6ab5a186-294b-4046-8242-1af0134540db | json_pp -json_opt pretty,canonical


```
{
   "active" : true,
   "authorities" : [
      "CAN_READ"
   ],
   "client_id" : "mobile",
   "exp" : 1606023818,
   "scope" : [
      "READ",
      "WRITE"
   ],
   "user_name" : "user"
}
```

Oauth 2 Resource Server

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-security</artifactId>
    </dependency>
```

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Oauth 2 Resource Server

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
class Compte{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY )
    private Long id;
    private String type;
    private double solde;
}
```

```java
@RepositoryRestResource
interface CompteRepository extends JpaRepository<Compte,Long> {
}
```

```java
@SpringBootApplication
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class DemoComptesApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoComptesApplication.class, args);
    }
@Bean
    CommandLineRunner start(CompteRepository compteRepository){
        String[] types={"courant","epargne"};
        return args -> {
            for (int i = 0; i <10 ; i++) {
                String type=types[new Random().nextInt(1)];
                compteRepository.save(new
Compte(null,type,Math.random()*9000));
            }
        };
    }
}
```

```
server.port=8081
spring.datasource.url=jdbc:h2:mem:comptes-db
security.oauth2.resource.token-info-uri=http://localhost:8082/oauth/check_token
security.oauth2.client.client-id=mobile
security.oauth2.client.client-secret=1234
```

Oauth 2 Resource Server

```java
@RestController
@RequestMapping("/api")
class BanqueRestController{
    @Autowired
    private CompteRepository compteRepository;
    @GetMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_READ')")
    public Compte getCompte(@PathVariable Long id){
        return compteRepository.findById(id).get();
    }
    @PostMapping(path = "/comptes")
    @PreAuthorize("hasAuthority('CAN_WRITE')")
    public Compte save(@RequestBody Compte cp){
        return compteRepository.save(cp);
    }
    @PutMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_UPDATE')")
    public Compte update(@PathVariable Long id,@RequestBody Compte cp){
        cp.setId(id);
        return compteRepository.save(cp);
    }
    @DeleteMapping(path = "/comptes/{id}")
    @PreAuthorize("hasAuthority('CAN_DELETE')")
    public void delete(@PathVariable Long id){
        compteRepository.deleteById(id);
    }
}
```

Oauth 2 Resource Server

```java
package org.sid.democomptes;

import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

@Configuration
public class HasAutorityConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers(HttpMethod.POST,"/comptes/**").hasAuthority("CAN_WRITE")
                .anyRequest().authenticated();
    }
}
```

Oauth 2 Resource Server

```java
package org.sid.democomptes;

import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

@Configuration
public class HasAutorityConfig extends ResourceServerConfigurerAdapter {
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers(HttpMethod.POST,"/comptes/**").hasAuthority("CAN_WRITE")
                .anyRequest().authenticated();
    }
}
```

Accès à la ressource

```
curl -d "grant_type=password&username=user&password=1234" -X POST
http://localhost:8082/oauth/token -H "Content-Type: application/x-www-form-urlencoded" -
u mobile:1234
```


```
{
 "access_token":"6ab5a186-294b-4046-8242-1af0134540db",
 "token_type":"bearer",
 "expires_in":40903,
 "scope":"READ WRITE"
}
```


```
curl http://localhost:8081/api/comptes/1 -H "Content-Type: application/x-www-
form-urlencoded" -H "Authorization:bearer 6ab5a186-294b-4046-8242-
1af0134540db"
```


```
{"id":1,"type":"courant","solde":6016.382751181522}
```