Ecole Normale Supérieure
de l'Enseignement
Technique Mohammedia

Université Hassan II de
Casablanca

ENS
ET

ا لمدرسة العليا لأساتذة التعليم
التقني المحمدية

جامعة الحسن الثاني بالدار البيضاء

DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Compte Rendu
Micros Services

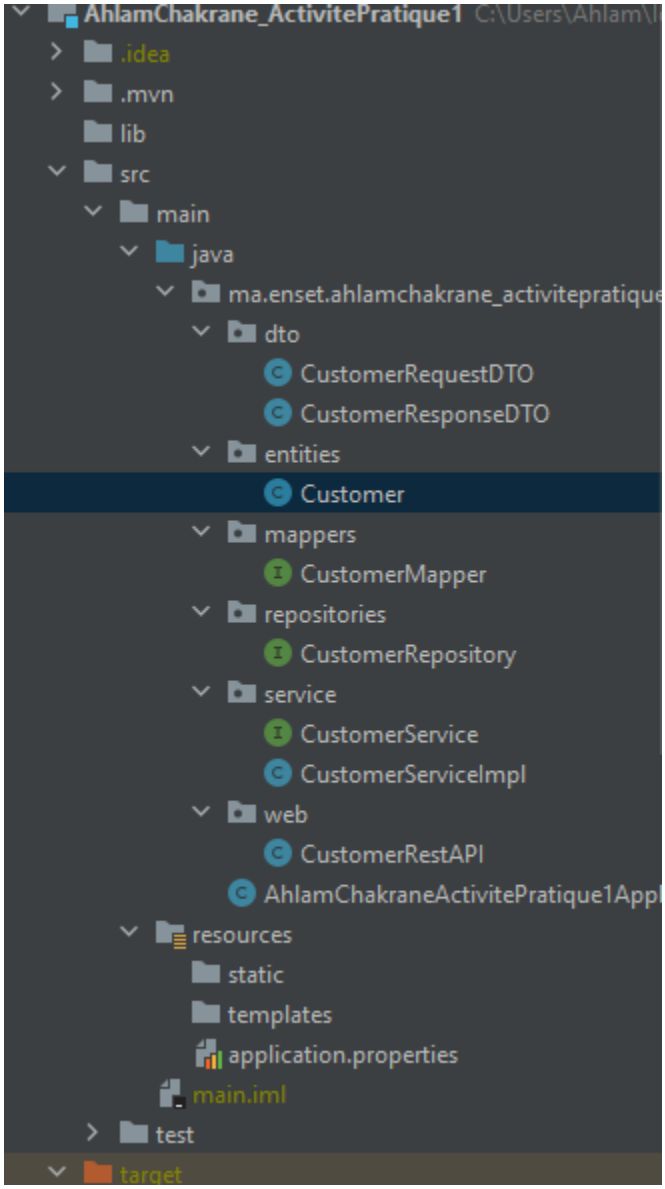
Filière :
« Génie du Logiciel et des Systèmes Informatiques Distribués »
GLSID

le 26/10/2022

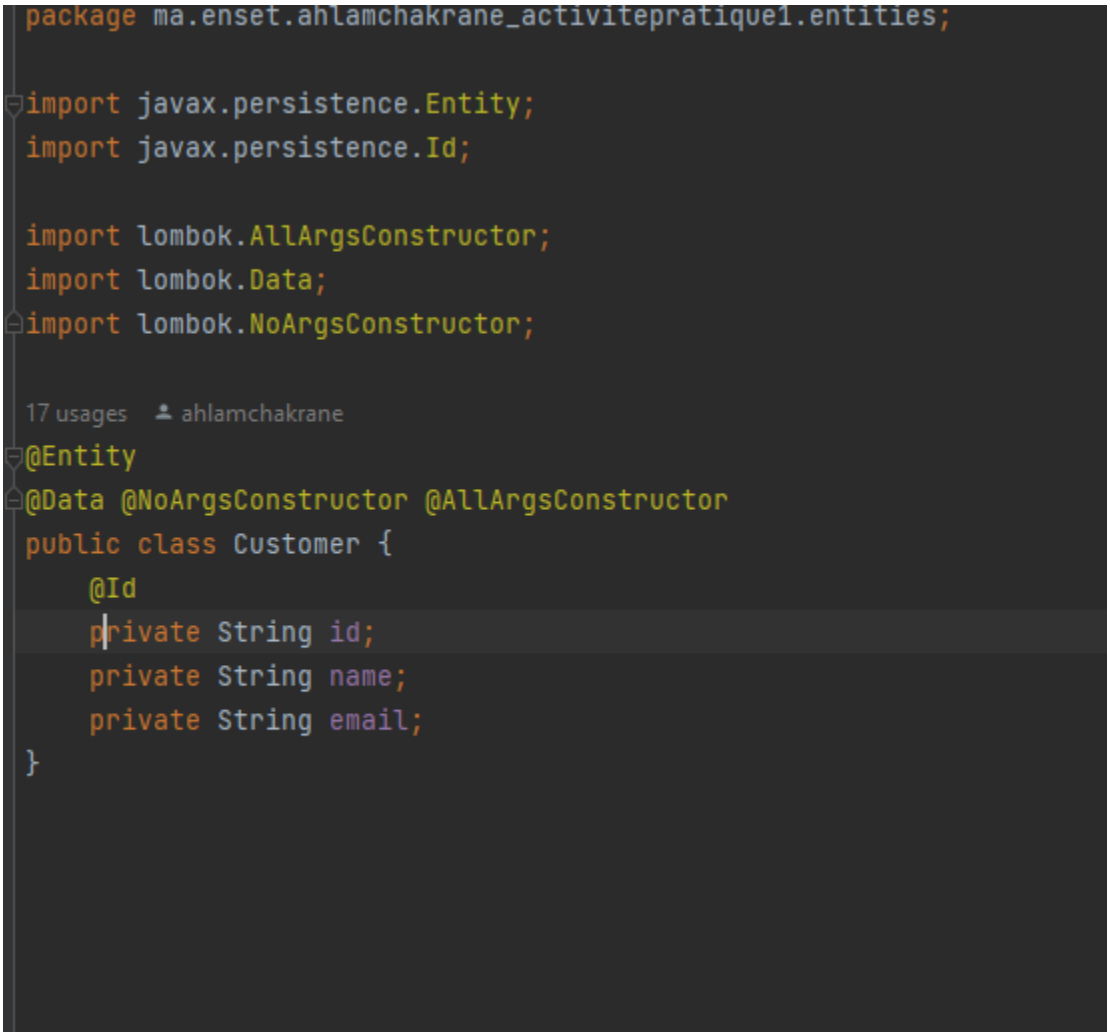
Préparé par : Ahlam CHAKRANE

Année Universitaire : 2022-2023

Customer-MicroService



Les entités :



```java
package ma.enset.ahlamchakrane_activitepratique1.entities;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

17 usages  ahlamchakrane
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Customer {
    @Id
    private String id;
    private String name;
    private String email;
}
```

L'annotation @Entity pour définir que cette classe correspond à une table dans la base de données.

L'annotation Id pour définir que c'est un identifiant de la classe.

@Data pour gerer les getters et setters. (lombok)

@NoArgsConstructor pour définir un constructeur sans parametres

@AllArgsConstructor pour définir un constructeur avec paramétre

Les DTOs :

```java
package ma.enset.ahlamchakrane_activitepratique1.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

16 usages
@Data @NoArgsConstructor @AllArgsConstructor
public class CustomerRequestDTO {
    private String id;
    private String name;
    private String email;
}
```

```java
package ma.enset.ahlamchakrane_activitepratique1.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

22 usages  ahlamchakrane
@Data @NoArgsConstructor @AllArgsConstructor
public class CustomerResponseDTO {
    private String id;
    private String name;
    private String email;
}
```

Puisque les dtos ne correspondent pas à une classe de la base de données alors on ne mis pas l'annotation @Entity et @Id.

Pour fiare comminiquer les entités et les DTOS on est besoin des mappers. Donc on vas utiliser MappeStract

```java
package ma.enset.ahlamchakrane_activitepratique1.mappers;

import org.mapstruct.Mapper;

import ma.enset.ahlamchakrane_activitepratique1.dto.CustomerRequestDTO;
import ma.enset.ahlamchakrane_activitepratique1.dto.CustomerResponseDTO;
import ma.enset.ahlamchakrane_activitepratique1.entities.Customer;

4 usages  1 implementation  ahlamchakrane
@Mapper(componentModel = "spring")
public interface CustomerMapper{
    4 usages  1 implementation  ahlamchakrane
    CustomerResponseDTO customerToCustomerResponseDTO(Customer customer);
    2 usages  1 implementation  ahlamchakrane
    Customer customerRequestDTOCustomer(CustomerRequestDTO customerRequestDTO);
}
```

Le contrôleur pour gérer les requêtes http :

```java
@RestController
@RequestMapping(path = "/api")
public class CustomerRestAPI {
    4 usages
    private CustomerService customerService;
    ahlamchakrane
    public CustomerRestAPI(CustomerService customerService) { this.customerService = customerSe
    ahlamchakrane
    @GetMapping(path = "/customers")
    public List<CustomerResponseDTO> allCustomers() { return customerService.listCustomers(); }
    ahlamchakrane
    @PostMapping(path = "/customers")
    public CustomerResponseDTO save(@RequestBody CustomerRequestDTO customerRequestDTO) {
        customerRequestDTO.setId(UUID.randomUUID().toString());
        return customerService.save(customerRequestDTO);
    }
    ahlamchakrane
    @GetMapping(path = "/customers/{id}")
    public CustomerResponseDTO getCustomer(@PathVariable String id) { return customerService.ge
```

Puisqu'on utilise une architecture REST alors on utilise le RestController. Dans le Controller on définit un ensemble de méthodes notamment Get, Post, Delete…

L’annotation @RequestMapping permet de définir comment nous allons accéder au contrôleur. On est obligé alors de saisir dans l’URL /api/customers pour qu’on puisse avoir des réponses http.

Le repository pour gérer les relations entre les entités et la base de données

```java
package ma.enset.ahlamchakrane_activitepratique1.repositories;

import org.springframework.data.jpa.repository.JpaRepository;

import ma.enset.ahlamchakrane_activitepratique1.entities.Customer;

3 usages  ahlamchakrane
public interface CustomerRepository extends JpaRepository<Customer, String> {

}
```

Le service pour exploiter des méthodes qui vont être utiliser pour traiter les requête http

```java
package ma.enset.ahlamchakrane_activitepratique1.service;
//metier

import java.util.List;

import ma.enset.ahlamchakrane_activitepratique1.dto.CustomerRequestDTO;
import ma.enset.ahlamchakrane_activitepratique1.dto.CustomerResponseDTO;

6 usages  1 implementation  ahlamchakrane
public interface CustomerService {
    4 usages  1 implementation  ahlamchakrane
    CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO);
    1 usage  1 implementation  ahlamchakrane
    CustomerResponseDTO getCustomer(String id);
    1 implementation  ahlamchakrane
    CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO);
    1 usage  1 implementation  ahlamchakrane
    List<CustomerResponseDTO>  listCustomers();

}
```

Sevice implémentation pour implémenter les méthodes de l'interface

```java
@Service
@Transactional
public class CustomerServiceImpl implements CustomerService {

    private CustomerRepository customerRepository;

    private CustomerMapper customerMapper;

    public CustomerServiceImpl(CustomerRepository customerRepository, CustomerMapper customerM
        this.customerRepository = customerRepository;
        this.customerMapper = customerMapper;
    }

    @Override
    public CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO) {
        /* code lourd
        //maping objet objet
        Customer customer = new Customer();
        customer.setId(customerRequestDTO.getId());
        customer.setName(customerRequestDTO.getName());
        customer.setEmail(customerRequestDTO.getEmail());
        */
        Customer customer = customerMapper.customerRequestDTOCustomer(customerRequestDTO);
        Customer saveCustomer = customerRepository.save(customer);
        //maping objet objet
```


@Service est une annotation qu'on peut attribués qu'aux classes, et dire que cette classe vas jouer le rôle d'un service provider, il va alors servir d'autres classes.

Pour le fichier application.properties nous avons le configurer comme cela :

```
server.port = 8882
spring.application.name = CUSTOMER-SERVICE
spring.h2.console.enabled = true
spring.cloud.discovery.enabled= true
eureka.instance.prefer-ip-address=true
spring.datasource.url=jdbc:h2:mem:customer-db
```

Le fichier Docker est comme suit :

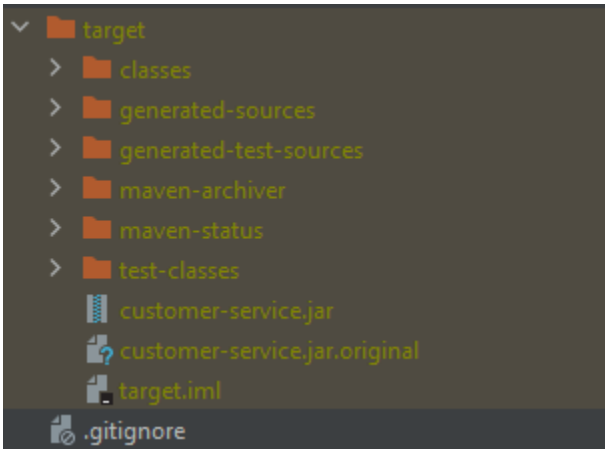
```
FROM openjdk:17-alpine
COPY ./target/*.jar app.jar
CMD ["java","-jar","app.jar"]
EXPOSE 8882
```

Le port mentionner dans Docker doit être le même dans application.properties

Apres nous avons définit comment nous voudrons le nom du jar qui soir générer. Ceci a été fait dans pom.xml :

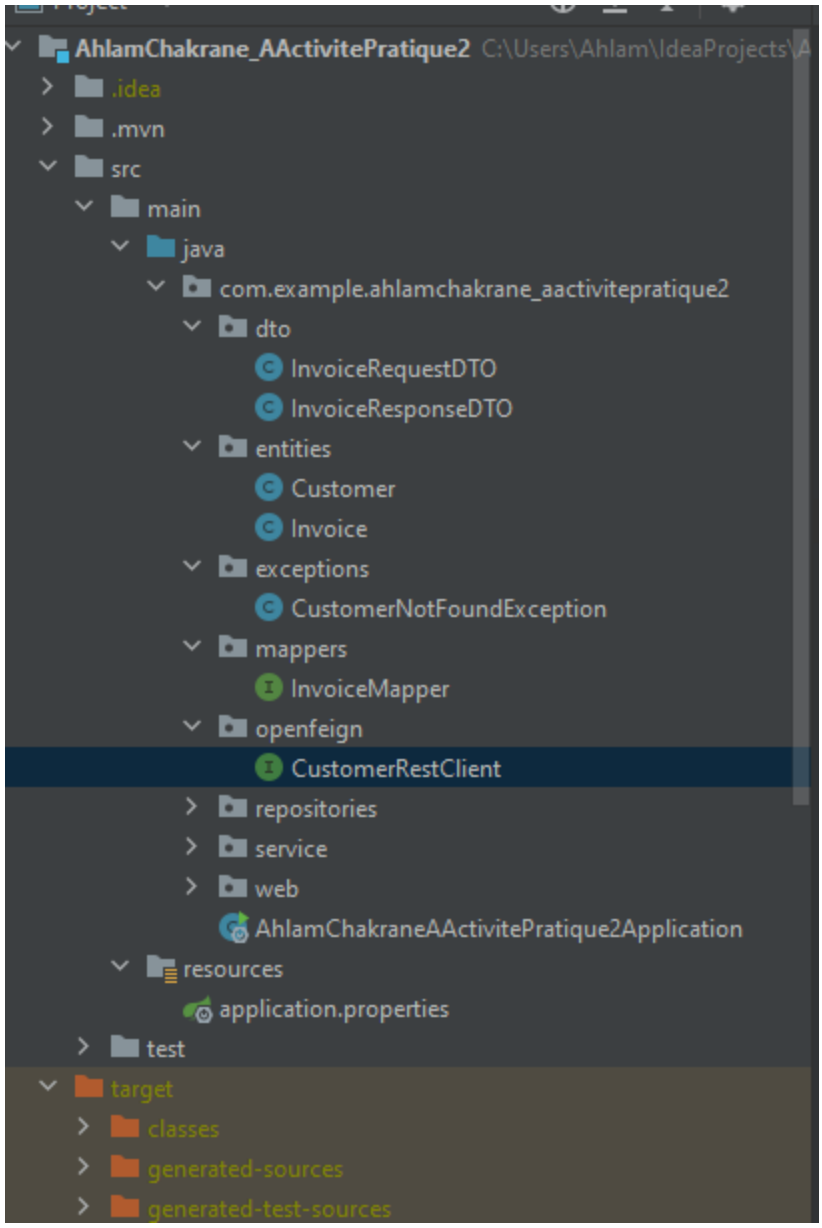
```
                    </path>
                </annotationProcessorPaths>
            </configuration>
        </plugin>
    </plugins>
    <finalName>customer-service</finalName>
</build>

oject>
```

Par la suit nous avons tourner la commande Maven install. Et le jar a été générer :



Ces étapes sont presque les mêmes pour les autres micros services. Nous allons par la suite monter que les fichiers ou il y a une différence.

Belling Service :


AhlamChakrane_AActivitePratique2 C:\Users\Ahlam\IdeaProjects\A
.idea
.mvn
src
main
java
com.example.ahlamchakrane_aactivitepratique2
dto
InvoiceRequestDTO
InvoiceResponseDTO
entities
Customer
Invoice
exceptions
CustomerNotFoundException
mappers
InvoiceMapper
openfeign
CustomerRestClient
repositories
service
web
AhlamChakraneAActivitePratique2Application
resources
application.properties
test
target
classes
generated-sources
generated-test-sources

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Invoice {
    @Id
    private String id;
    private Date date;
    private BigDecimal amount;
    private String customerId;
    @Transient
    private Customer customer;
}
```

La classe Invoice. Le customer est une classe et non pas une entité JPA.

```java
ahlamchakrane
@Data @NoArgsConstructor @AllArgsConstructor
public class Customer {
    private String id;
    private String name;
    private String email;
}
```

De plus pour communiquer avec le micro service customer on est besoin d'un outil de communication. C'est pour cela on va utiliser open feign. En donnant au paramètres le nom du micro service que nous avons déjà mis dans application.properties de customer.

```
3 usages  ahlamchakrane
@FeignClient(name = "CUSTOMER-SERVICE")
public interface CustomerRestClient {

    4 usages  ahlamchakrane
    @GetMapping(path = "/api/customers/{id}")
    Customer getCustomer(@PathVariable(name = "id") String id);

    ahlamchakrane
    @GetMapping(path = "/api/customers")
    List<Customer> allCustomers();
}
```

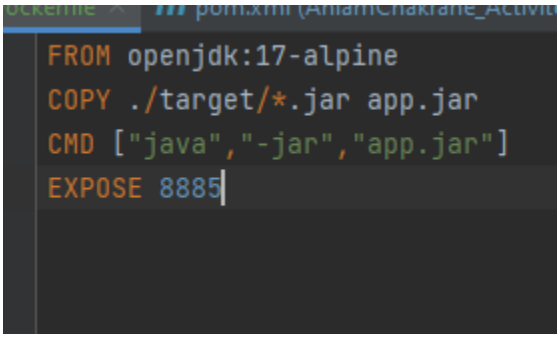
Il faut l'autorisé dans la classe main :

```
@SpringBootApplication
@EnableFeignClients
public class AhlamChakraneAActivitePratique2Application {

    ahlamchakrane
    public static void main(String[] args) {
        SpringApplication.run(AhlamChakraneAActivitePratique2Appl
    }
   /*@Bean
```

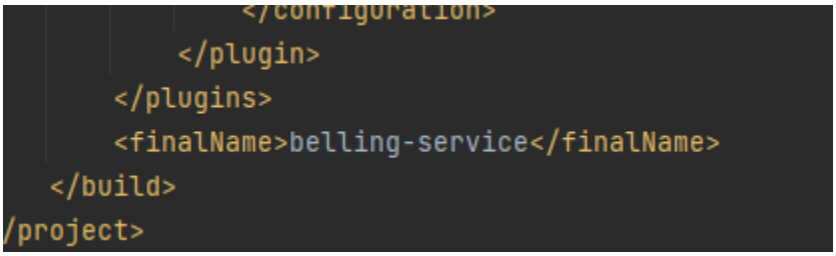
Le fichier aplication.properties :

```
server.port = 8885
spring.application.name = BELLING-SERVICE
spring.h2.console.enabled = true
spring.cloud.discovery.enabled= true
eureka.instance.prefer-ip-address=true
spring.datasource.url=jdbc:h2:mem:belling-db
```

Le fichier docker :

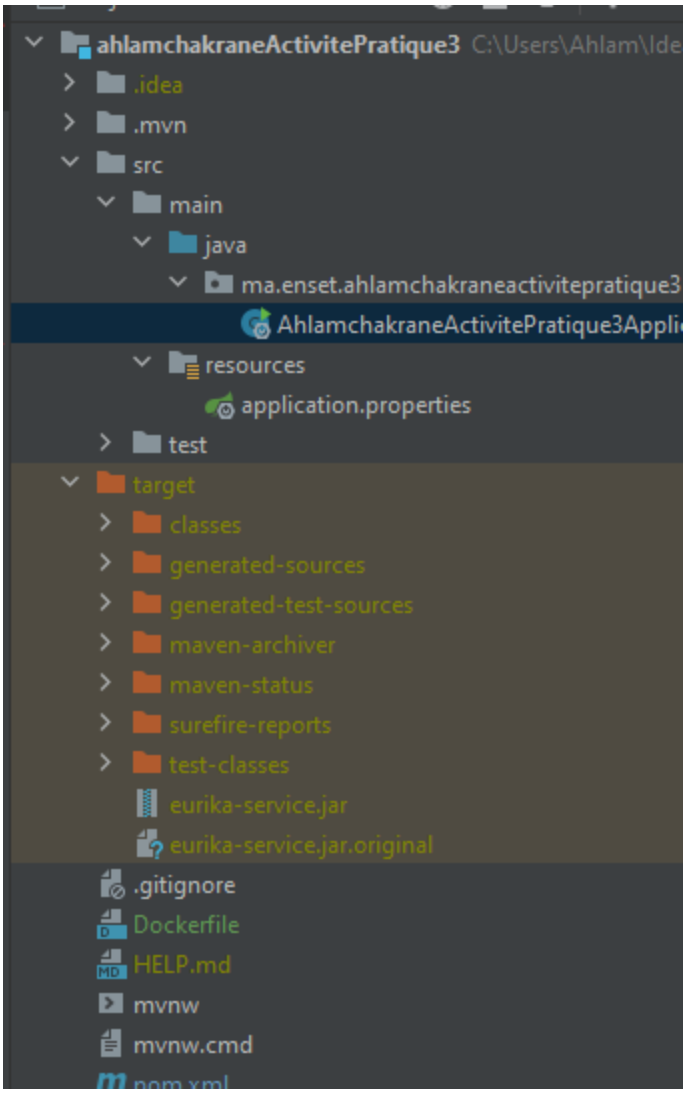


Le nom que nous voudrons comme jar :

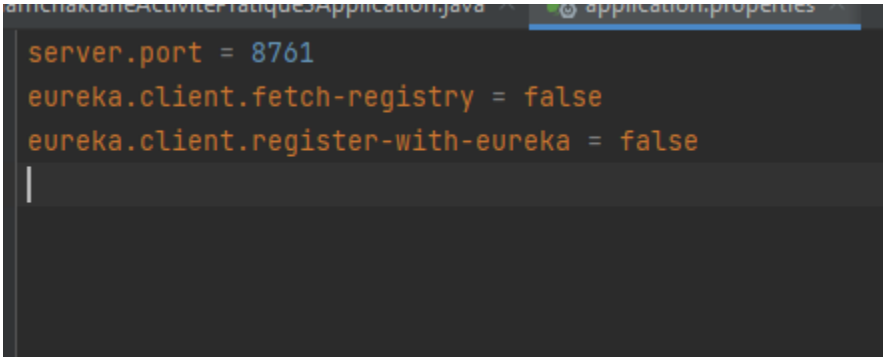


Les jars générés :

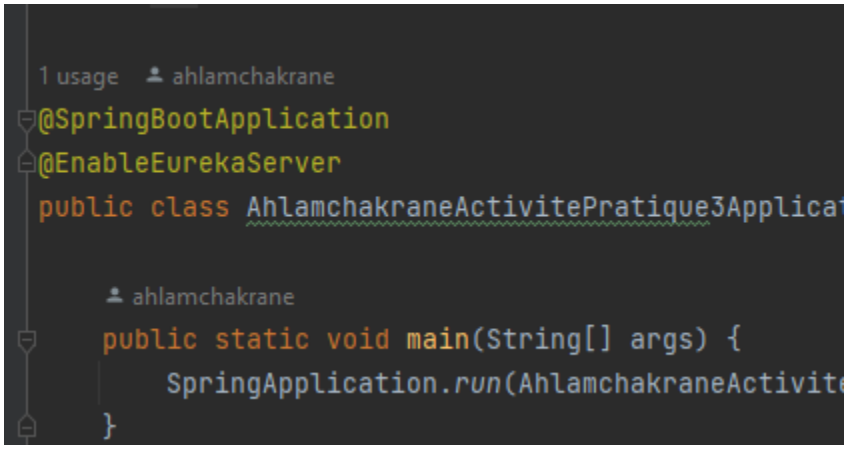
Pour que les micros services puissent publier leurs ports, adresse ip et nom, nous utilisons spring cloud et plus précisément eurika.



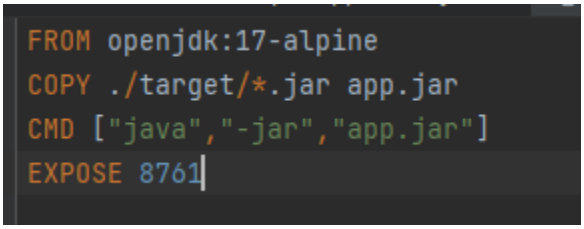
Dans application.properties :



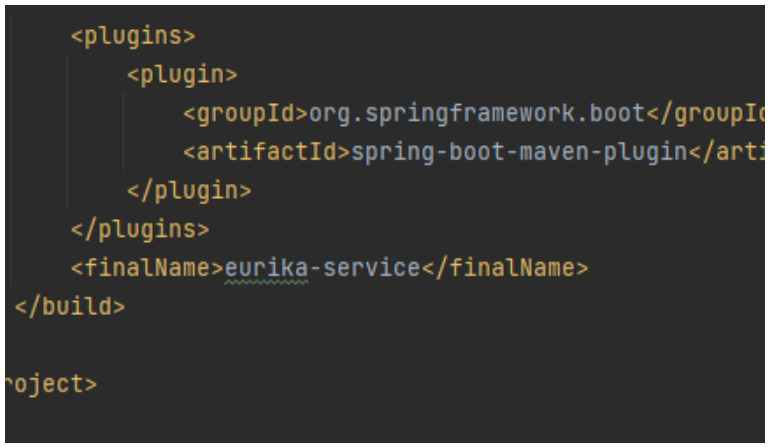
Il faut autoriser eurika dans la classe main :



Le fichier Docker :



Dans pm.xml :



Les jars générés :

Pour qu'on puisse contrôler et diriger les requêtes http vers le micro service le plus adéquat, on est besoin de la Gateway :

La classe main :

```
1 usage  ahlamchakrane *
@SpringBootApplication
public class AhlamchakraneActivitePratique4Application {

    ahlamchakrane
    public static void main(String[] args) {
        SpringApplication.run(AhlamchakraneActivitePratique4Application.class, args);
    }
    ahlamchakrane *
    @Bean
    DiscoveryClientRouteDefinitionLocator discoveryClientRouteDefinitionLocator(
            ReactiveDiscoveryClient reactiveDiscoveryClient,
            DiscoveryLocatorProperties discoveryLocatorProperties) {
        return new DiscoveryClientRouteDefinitionLocator(reactiveDiscoveryClient,
                discoveryLocatorProperties);
    }
```

Le fichier application.properties :

```
server.port = 9999
spring.application.name = GATEWAY
spring.cloud.discovery.enabled= true
eureka.instance.prefer-ip-address= true
```

Le fichier Docker :

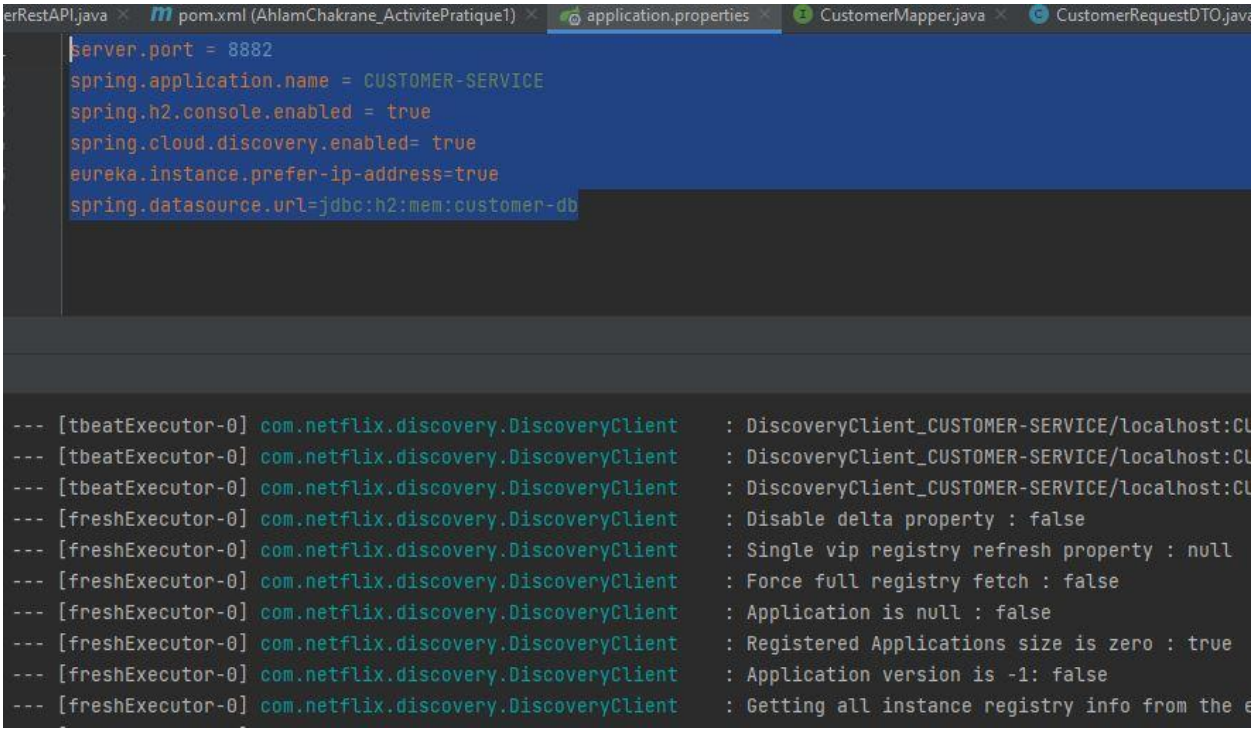
```
FROM openjdk:17-alpine
COPY ./target/*.jar app.jar
CMD ["java","-jar","app.jar"]
EXPOSE 9999
```

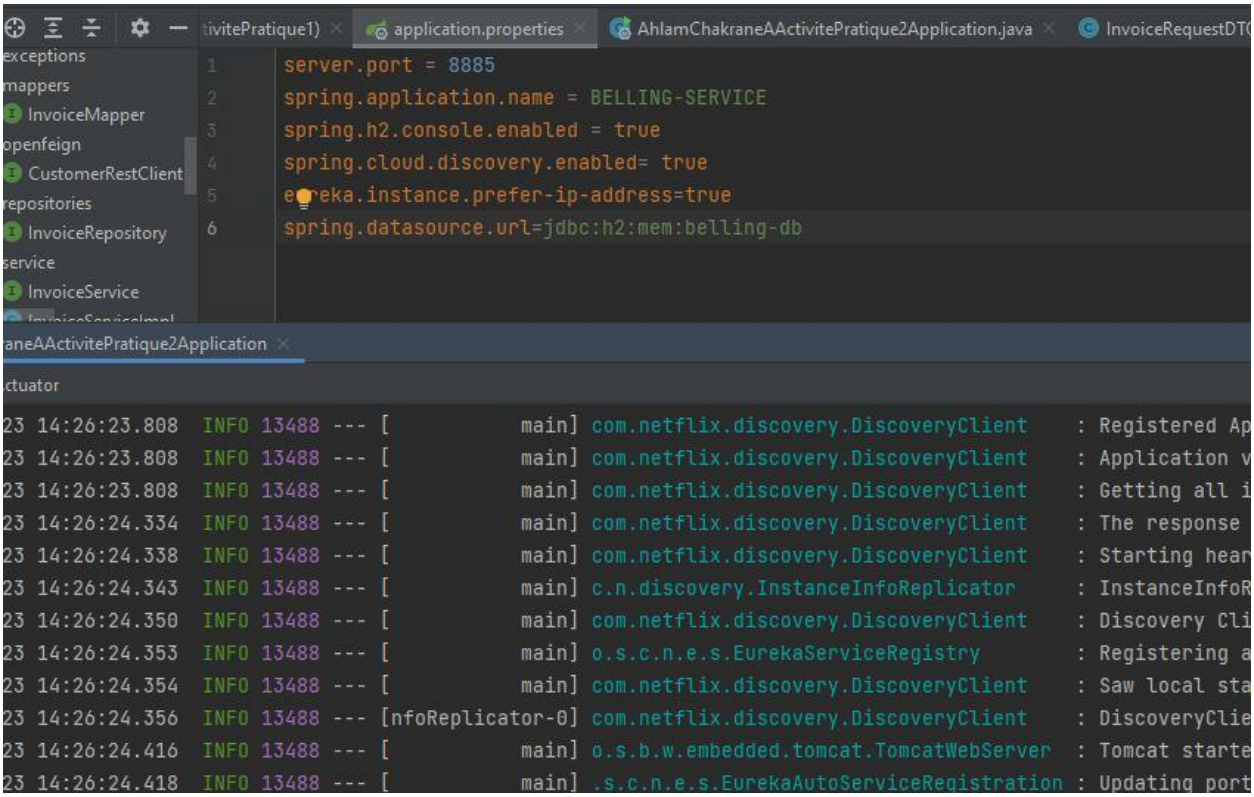
Dans le fihcier pom.xml

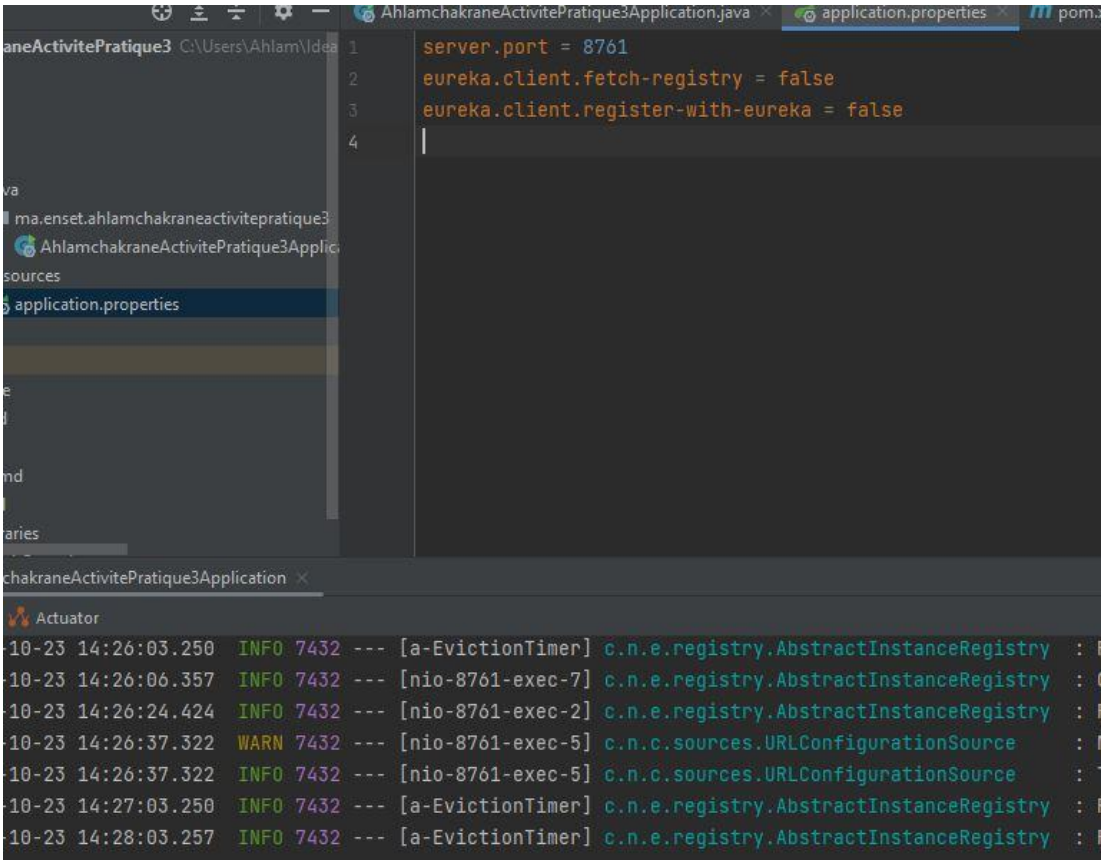
```
        </plugin>
    </plugins>
    <finalName>gateway-service</finalName>
</build>

oject>
```

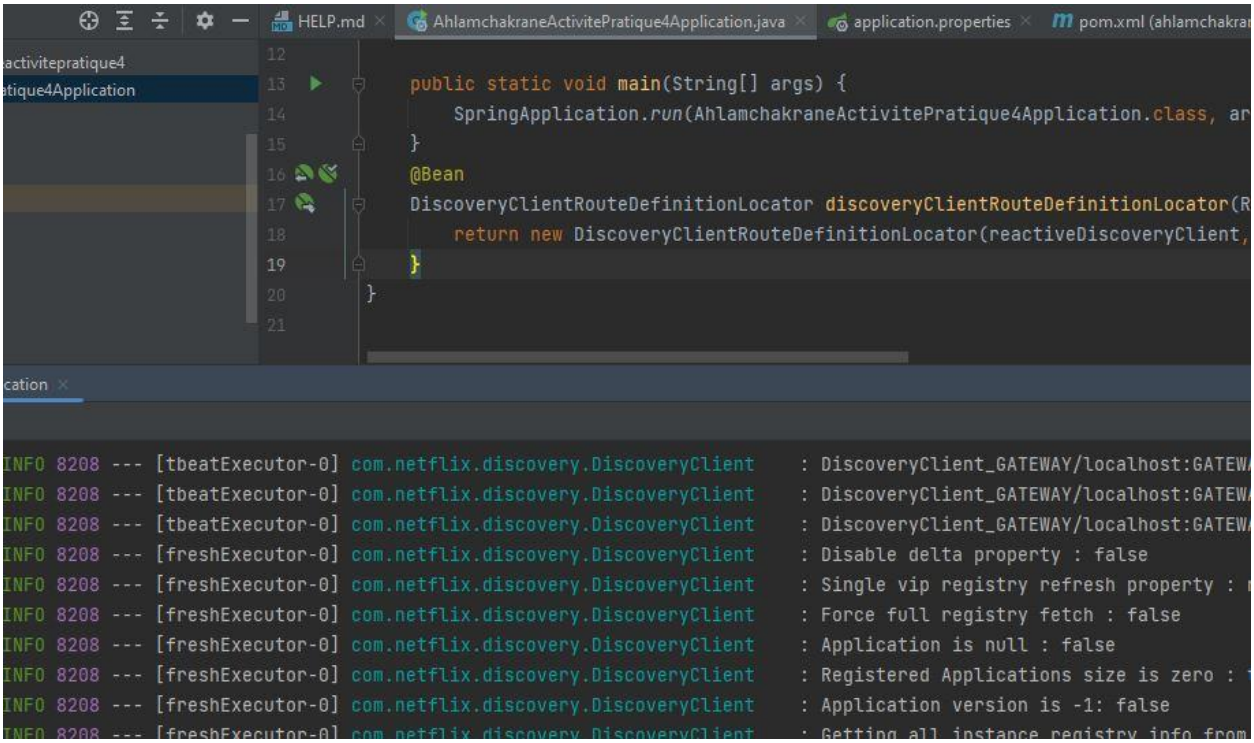
Les jars générés :

Les résultats :

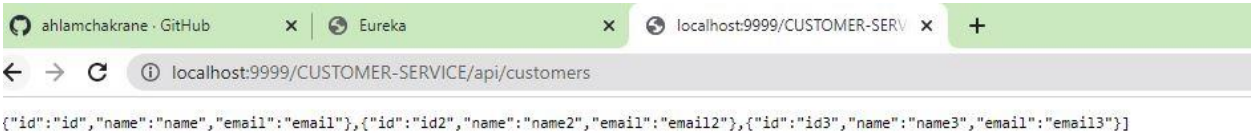



application.properties
AhlamChakraneAActivitePratique2Application.java
exceptions
mappers
InvoiceMapper
openfeign
CustomerRestClient
repositories
InvoiceRepository
service
InvoiceService
server.port = 8885
spring.application.name = BELLING-SERVICE
spring.h2.console.enabled = true
spring.cloud.discovery.enabled= true
eureka.instance.prefer-ip-address=true
spring.datasource.url=jdbc:h2:mem:belling-db
Actuator
23 14:26:23.808 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Registered Ap
23 14:26:23.808 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Application v
23 14:26:23.808 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Getting all i
23 14:26:24.334 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : The response
23 14:26:24.338 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Starting hear
23 14:26:24.343 INFO 13488 --- [main] c.n.discovery.InstanceInfoReplicator : InstanceInfoR
23 14:26:24.350 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Discovery Cli
23 14:26:24.353 INFO 13488 --- [main] o.s.c.n.e.s.EurekaServiceRegistry : Registering a
23 14:26:24.354 INFO 13488 --- [main] com.netflix.discovery.DiscoveryClient : Saw local sta
23 14:26:24.356 INFO 13488 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClie
23 14:26:24.416 INFO 13488 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat starte
23 14:26:24.418 INFO 13488 --- [main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port


AhlamchakraneActivitePratique3Application.java
application.properties
aneActivitePratique3 C:\Users\Ahlam\Idea
ma.enset.ahlamchakraneactivitepratique3
AhlamchakraneActivitePratique3Applic
sources
application.properties
server.port = 8761
eureka.client.fetch-registry = false
eureka.client.register-with-eureka = false
Actuator
-10-23 14:26:03.250 INFO 7432 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry
-10-23 14:26:06.357 INFO 7432 --- [nio-8761-exec-7] c.n.e.registry.AbstractInstanceRegistry
-10-23 14:26:24.424 INFO 7432 --- [nio-8761-exec-2] c.n.e.registry.AbstractInstanceRegistry
-10-23 14:26:37.322 WARN 7432 --- [nio-8761-exec-5] c.n.c.sources.URLConfigurationSource
-10-23 14:26:37.322 INFO 7432 --- [nio-8761-exec-5] c.n.c.sources.URLConfigurationSource
-10-23 14:27:03.250 INFO 7432 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry
-10-23 14:28:03.257 INFO 7432 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry



Listes des customers :



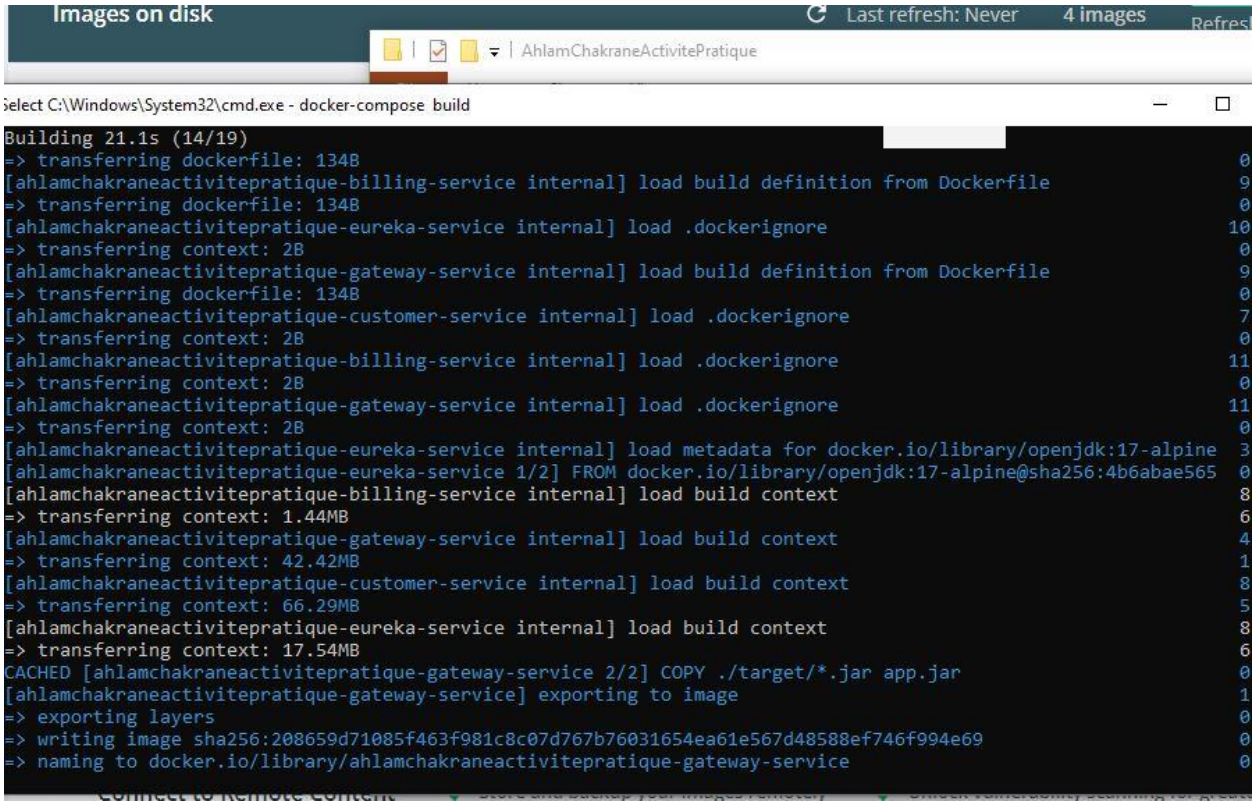
pour exploiter Docker, on est besoin de docker-compose.yml :

```yaml
version: '3'
services:
  eureka-service:
    build: ./ahlamchakraneActivitePratique3/
    hostname: eureka-service
    ports:
      - "8761:8761"
    networks:
      - default-network

  customer-service:
    build: ./AhlamChakrane_ActivitePratique1/
    hostname: customer-service
    ports:
      - "8082:8082"
    depends_on:
      - eureka-service
    environment:
      - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
    networks:
      - default-network

  billing-service:
    build: ./AhlamChakrane_AActivitePratique2/
    hostname: billing-service
    ports:
      - "8083:8083"
    restart: on-failure
    depends_on:
      - customer-service
      - eureka-service
```

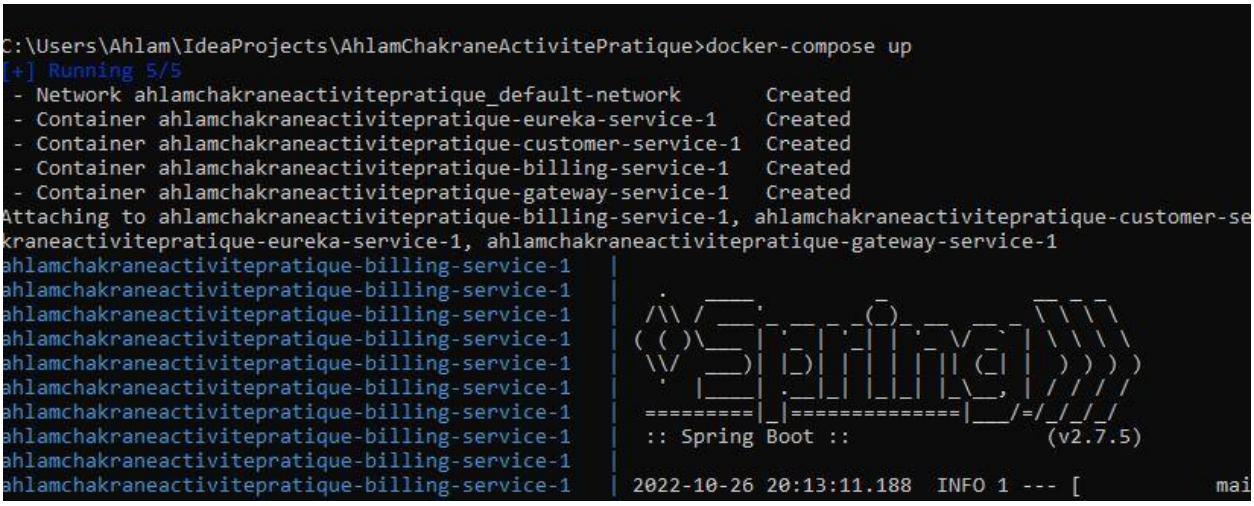
```
    hostname: billing-service
    ports:
      - "8083:8083"
    restart: on-failure
    depends_on:
      - customer-service
      - eureka-service
    environment:
      - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
    networks:
      - default-network

  gateway-service:
    build: ./ahlamchakraneActivitePratique4/
    hostname: gateway-service
    ports:
      - "9999:9999"
    depends_on:
      - customer-service
      - billing-service
      - eureka-service
    environment:
      - eureka.client.service-url.defaultZone=http://eureka-service:8761/eureka
    networks:
      - default-network

networks:
  default-network:
    driver: bridge
```

Il faut installer docker. Puis utiliser :

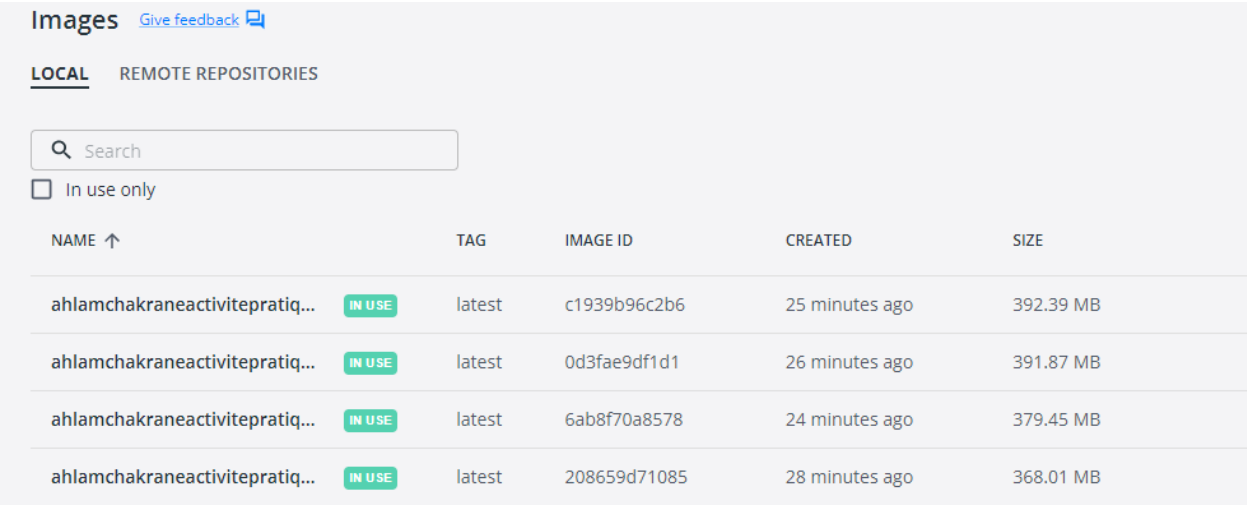
docker-compose build



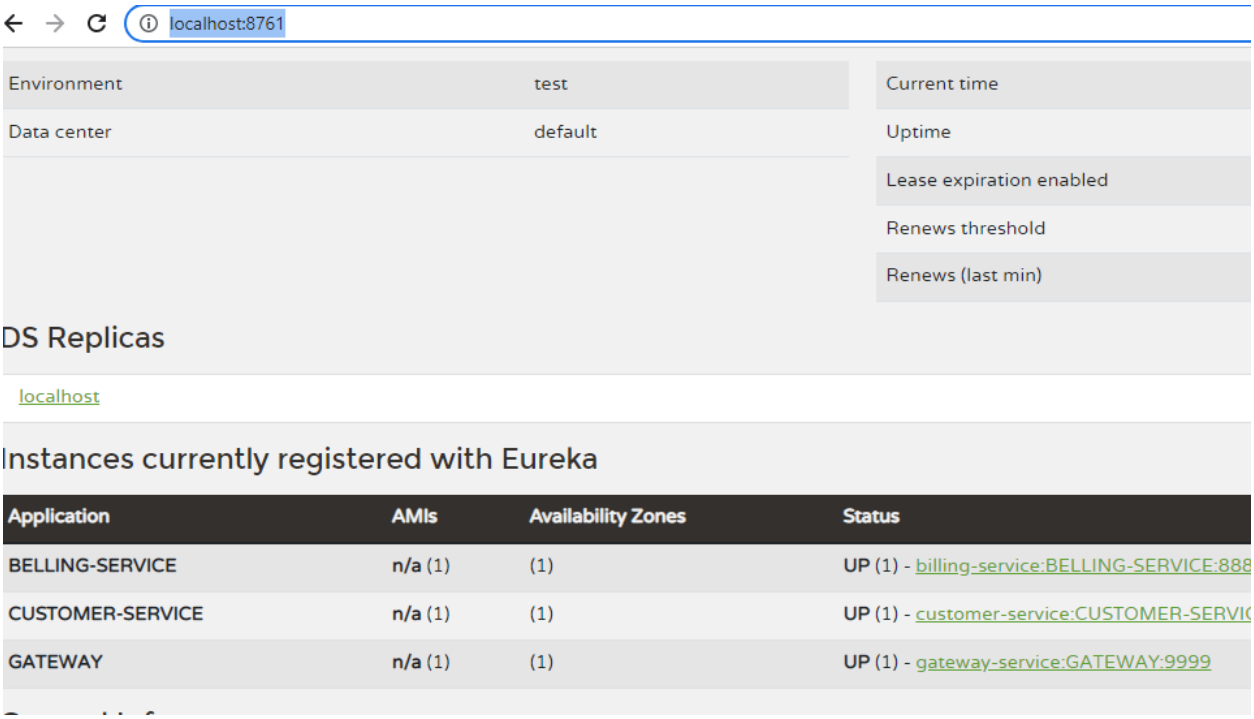
Puis docker-compose up :

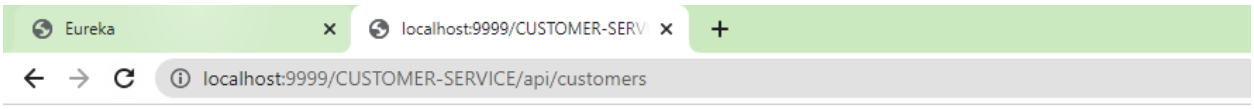


On peut voir dans docker desktop les images :



MAINTENANT sans avoir démarré mon wamp server, je peux consulter mes services :





[{"id":"id","name":"name","email":"email"},{"id":"id2","name":"name2","email":"email2"},{"id":"id3","name":"name3","email":"email3"}]