



MATHÉMATIQUES ET INFORMATIQUE

**Sciences**

Université de Paris

# **Master 1 Informatique Rapport de projet**

---

## **Programmation distribué**

---

**Chatri CHANSUK**

**Année universitaire : 2019 – 2020**

---

# TABLE DES MATIÈRES

---

<b>1. ARCHITECTURE MICROSERVICE .....</b>	<b>4</b>
1.1 DEFINITION/SEMANTIQUE .....	4
1.1.1 PHILOSOPHIE .....	4
1.1.2 STRUCTURE.....	5
1.2 CONCLUSIONS.....	7
<b>2. FRONT SERVICE.....</b>	<b>8</b>
2.1 PRESENTATION.....	8
<b>3. SERVICES .....</b>	<b>11</b>
3.1 PRÉSENTATION.....	11
<b>4. RÉSULTATS SOUS DOCKER .....</b>	<b>14</b>

---

# LISTE DES FIGURES

---

FIGURE 1 - DIAGRAMME DE CLASSE.....	6
FIGURE 2 SCHÉMA DE L'APPLICATION.....	6
FIGURE 3 - ACCUEIL DE L'APPLICATION.....	8
FIGURE 4 LISTE DES PRODUITS/MÉDIAS.....	9
FIGURE 5 - INSERTION D'UN RATING.....	10
FIGURE 6 - JSON CRÉE .....	10
FIGURE 7 - EUREKA SERVER LOCAL.....	11
FIGURE 8 BASE DE DONNÉES H2 .....	12
FIGURE 9 AFFICHE DES DONNÉES SOUS FORMAT JSON .....	12
FIGURE 10 - H2 RATING.....	13
FIGURE 11 H2 APRÈS UNE REQUÊTE POST .....	13
FIGURE 12 - PRODUCT-SERVICE DOCKER.....	14
FIGURE 13 EUREKA DOCKER.....	15
FIGURE 14 IMAGES DOCKER .....	15
FIGURE 15 EUREKA SOUS LE PORT DE DOCKER .....	16
FIGURE 16 RATING SERVICE SOUS LE PORT DE DOCKER.....	16
FIGURE 17 CONSOLE H2 DU PRODUCT-SERVICE SOUS DOCKER.....	17

---

# ARCHITECTURE MICROSERVICE

---

## 1. Architecture microservice

### 1.1 Definition/Sémantique

Cette application a été réalisée dans le cadre de l'UE programmation web et distribuée. La thématique est la suivante : architecture microservice.

Des API [REST](#) sont souvent employées pour relier chaque microservice les uns aux autres. Un avantage avancé est que lors d'un besoin critique de mise à jour d'une ressource, seul le microservice contenant cette ressource sera mis à jour, l'ensemble de l'application restant compatible avec la modification, contrairement à la totalité de l'application dans une architecture classique.

#### 1.1.1 Philosophie

L'architecture microservice est une approche minimaliste du développement de logiciels modulaires. La modularité est définie comme le degré auquel les composants d'un système peuvent être séparés et recombinaison.

Avantages des microservices Maintenance simplifiée.

- Scalabilité facilitée en tant que composant individuel.
- Détection, isolation et récupération des défauts (FDIR).
- Une défaillance de processus dans un composant n'affectera pas les composants séparés.
- Déploiement indépendant. Déploiement rapide.
- Base de code plus petite. Support et communication renforcés pour les équipes plus petites et parallèles.
- Variété technologique.
- Avec les microservices, vous pouvez mélanger des bibliothèques, des bases de données, des frameworks, etc.

## 1.1.2 Structure

Avant tout, présentons l'application dans son ensemble.

Un cas d'utilisation simple : l'utilisateur se rend sur le site web, interagit avec les fonctionnalités implémentées. Il est en contact avec la vue, servant d'interface entre homme/machine.

Un lien est cependant nécessaire entre les deux, c'est le rôle du contrôleur, les requêtes post/get → CRUD sont à sa charge.

À la suite de ces requêtes des ressources seront attendues, ainsi après l'acheminement par le contrôleur, le modèle se chargera de traiter ces commandes en envoyant, ajoutant, modifiant, supprimant des données.

Voici les classes présentes dans l'architecture :

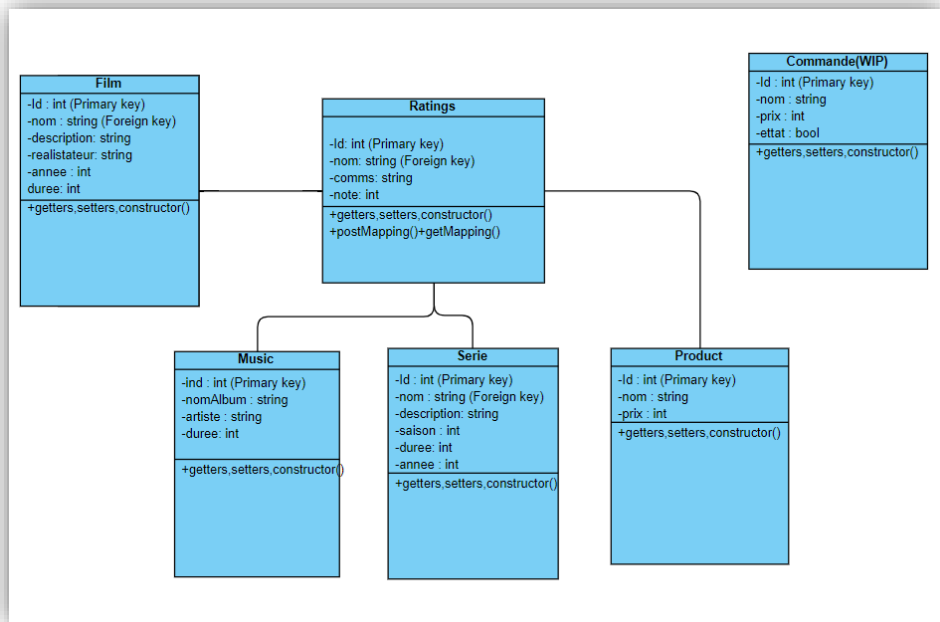


Figure 1 - Diagramme de classe

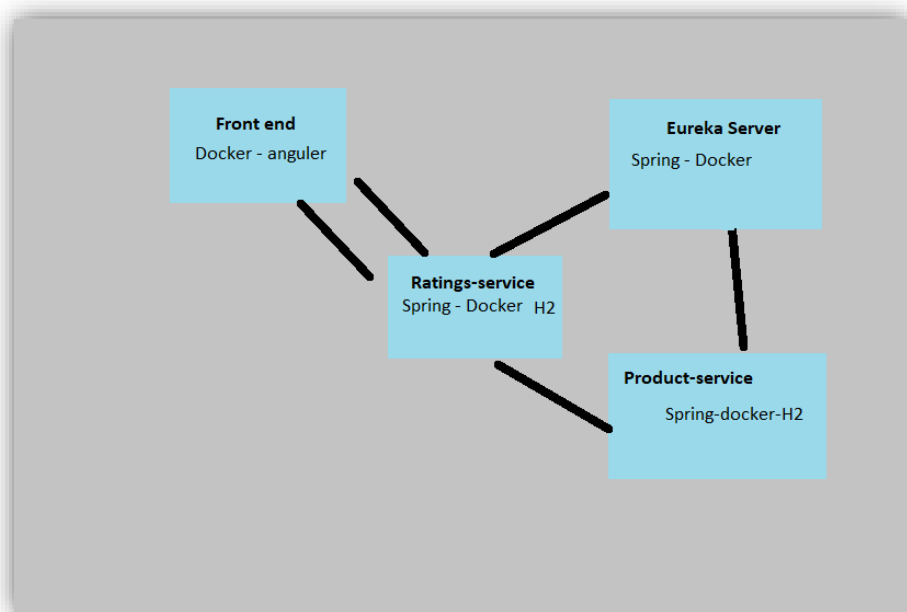


Figure 2 Schéma de l'application

Le cas d'Eureka Server est atypique car ici il sert d'application qui contient les informations sur toutes les applications de service client. Chaque service Micro s'enregistrera sur le serveur Eureka et le serveur Eureka connaît toutes les applications clientes exécutées sur chaque port et adresse IP.

## 1.2 Conclusions

Les technologies ayant été utilisées sont les suivantes :

- Stack front :
  - Angular
  - NodeJs
  - Bootstrap
- Stack back :
  - SpringBoot
  - Eureka
  - JPA
  - H2
- Containers :
  - Docker Toolbox
  - Oracle VM

---

# FRONT SERVICE

---

## 2. Front service

### 2.1 Présentation

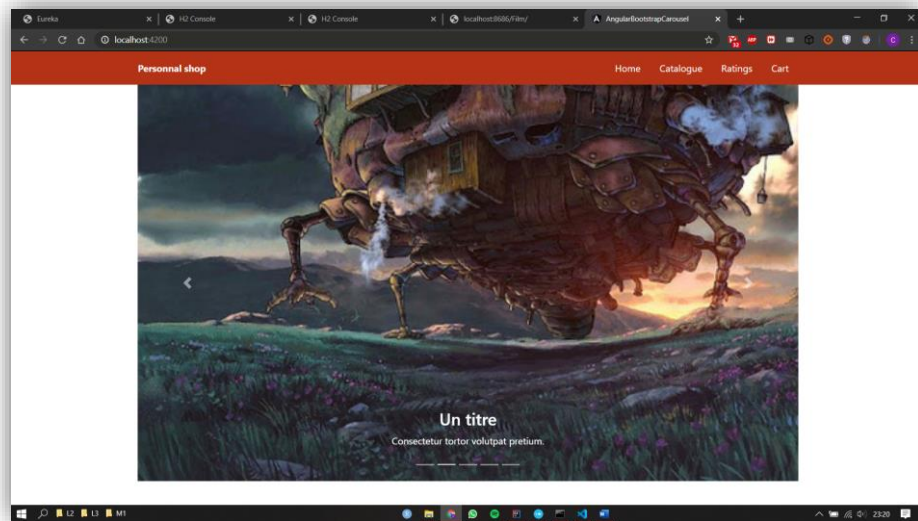


Figure 3 - Accueil de l'application



Première page de l'application, on peut remarquer son URL : localhost :4200, le port de base attribué par le client.

Une fois arrivé l'utilisateur pourra naviguer selon les options présente :

Personnal shop/Home : permet de retourner sur la première page, c'est-à-dire celle-ci.

Catalogue : Les classes avec leurs caractéristiques présentées précédemment sous forme d'article/produits.

L'affichage des articles dès l'initialisation de la page, et on rajoute à cela la réactivité des composants.

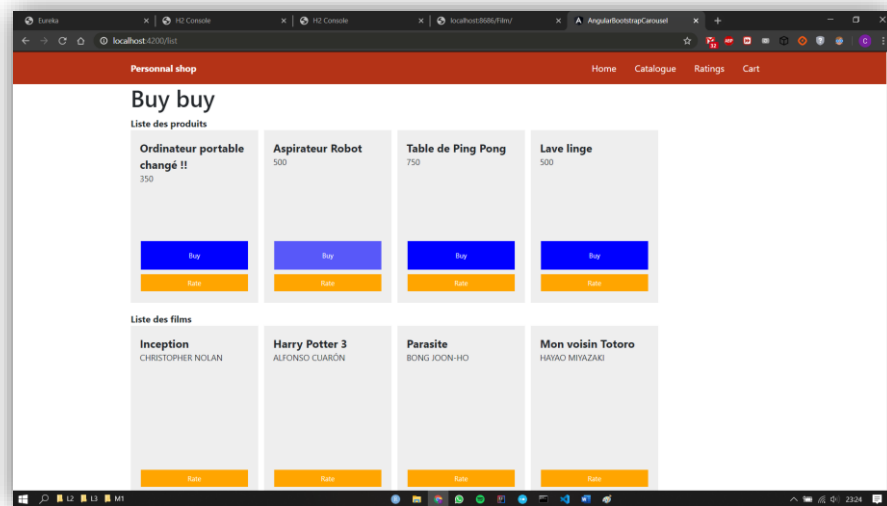


Figure 4 Liste des produits/médias

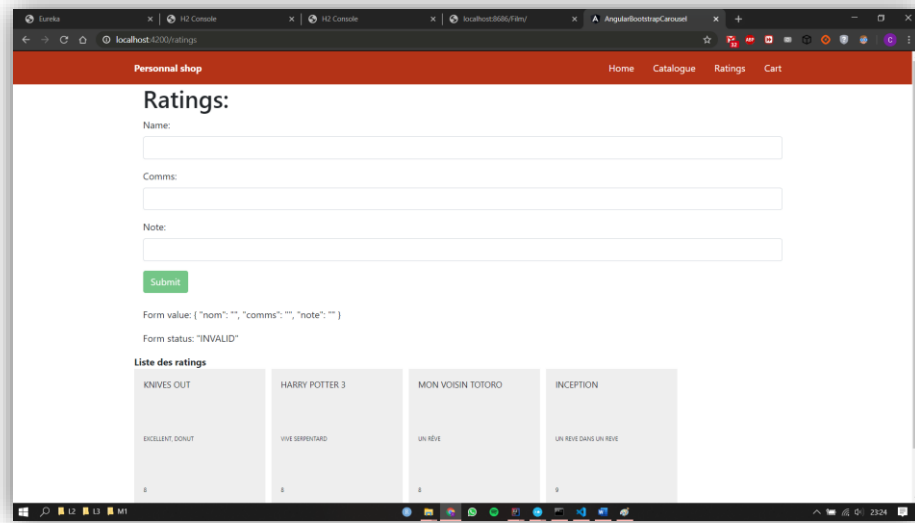


Figure 5 - Insertion d'un rating

Ici la création d'un rating est orchestrée par la fonction `formGroup` d'Angular, la présence du `Json value` nous présente les valeurs envoyées par validation du formulaire.

Des valeurs non valables sont surlignées à l'utilisateur (cf : valeur trop petite ou type non attendus).

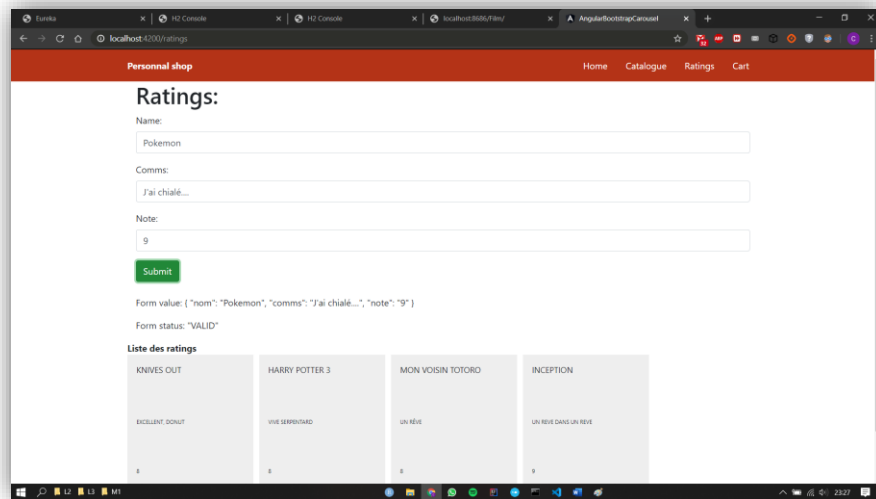


Figure 6 - Json crée

---

# SERVICES

---

## 3. Services

### 3.1 Présentation

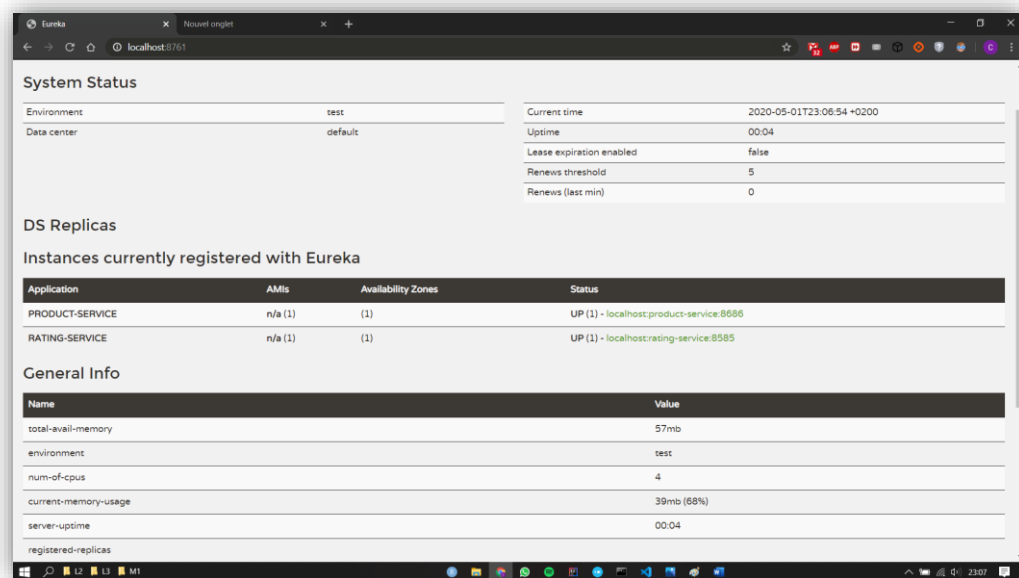


Figure 7 - Eureka server local

Eureka Server, qui est à initialisés avant tout lancement de microservices nous permet de servir d'annuaire dans notre cas présent, elle permet notamment au service ratings d'accéder à un modèle du service Product.

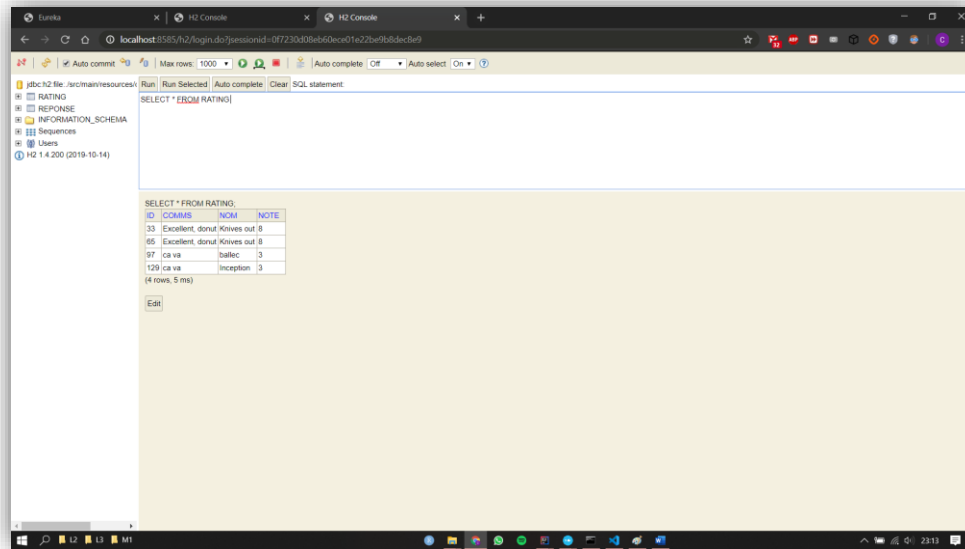


Figure 8 Base de données H2

Dans cette console-ci, nous pouvons voir à gauche la classe utilisé dans le rating-service. On y enregistre les informations nécessaires à sa construction/consultation. Elles sont structurés d'une certaine manière défini au sein du code. On obtient ainsi un fichier json, texte écrit en notation javascript.



Figure 9 Affiche des données sous format Json

The screenshot shows the H2 database interface. On the left, a tree view displays the database structure: RATING (with columns ID, COMMS, NOM, NOTE), REPONSE, INFORMATION\_SCHEMA, Sequences, and Users. The main area shows a SQL statement: `SELECT * FROM RATING`. Below the statement, a table displays the results of the query.

Action	ID	COMMS	NOM	NOTE
	33	Excellent, donut	Knives out	8
	65	Vive serpentard	Harry potter 3	8
	97	Un rêve	Mon voisin totoro	8
	129	Un reve dans un reve	Inception	9

Figure 10 - H2 rating

The screenshot shows the H2 database interface after a post request. The SQL statement remains `SELECT * FROM RATING`. The results table now includes an additional row with ID 66.

Action	ID	COMMS	NOM	NOTE
	33	Excellent, donut	Knives out	8
	65	Vive serpentard	Harry potter 3	8
	66	J'ai chialé...	Pokemon	9
	97	Un rêve	Mon voisin totoro	8
	129	Un reve dans un reve	Inception	9

Figure 11 H2 après une requête post

En revenant sur le précédent exemple d'utilisation sur notre navigateur, nous avons passés en paramètres les valeurs nécessaires pour créer un rating. Cependant dans le cas du front, le passage des paramètres dans les requêtes post entraînerent un conflit, le résultat obtenu est à la suite d'une utilisation postman.

## DOCKER

## 4. Résultats sous docker

Après paramétrage des fichiers docker, nous le lançons à partir de docker Toolbox (pour système Windows), à noter : le lancement des urls s'effectue à partir de l'ip obtenu par la machine local (en tapant docker-machine ip). Voici le déroulement :

```

com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server
    at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(RetryableEurekaHttpClient.java:112) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.shared.transport.decorator.FurekaHttpClientDecorator.register(FurekaHttpClientDecorator.java:56) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.shared.transport.decorator.FurekaHttpClientDecorator$1.execute(FurekaHttpClientDecorator.java:59) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(RetryableEurekaHttpClient.java:72) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.shared.transport.decorator.FurekaHttpClientDecorator.register(FurekaHttpClientDecorator.java:56) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.DiscoveryClient.register(DiscoveryClient.java:850) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.InstanceInfoReplicator.run(InstanceInfoReplicator.java:121) ~[eureka-client-1.9.17.jar/!1.9.17]
    at com.netflix.discovery.InstanceInfoReplicator$1.run(InstanceInfoReplicator.java:181) ~[eureka-client-1.9.17.jar/!1.9.17]
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) ~[na:1.8.0_212]
    at java.util.concurrent.FutureTask.run(FutureTask.java:266) ~[na:1.8.0_212]
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$280(ScheduledThreadPoolExecutor.java:180) ~[na:1.8.0_212]
    at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:239) ~[na:1.8.0_212]
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) ~[na:1.8.0_212]
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:626) ~[na:1.8.0_212]
    at java.lang.Thread.run(Thread.java:748) ~[na:1.8.0_212]

2020-05-01 21:41:00.370 INFO --- [main] o.s.b.o.embedded.Tomcat.TomcatableServer : Tomcat started on port(s): 8086 (http) with context path ""
2020-05-01 21:41:00.387 INFO --- [main] s.c.b.o.s.furekaAutoserviceRegistration : Updating port to 8086
2020-05-01 21:41:00.413 INFO --- [main] s.c.p.s.ProductServiceApplication : Started ProductServiceApplication in 24.581 seconds (JVM running for 26.693)
```

Figure 12 - Product-service docker

```

2020-05-01 21:38:14.698 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Registering application provided with eureka with status of UP
2020-05-01 21:38:14.711 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Setting the eureka configuration..
2020-05-01 21:38:14.721 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Eureka data center value eureka.datacenter is not set, defaulting to default
2020-05-01 21:38:14.739 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Eureka environment value eureka.environment is not set, defaulting to test
2020-05-01 21:38:14.759 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : isAwes returned false
2020-05-01 21:38:14.802 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Initialized server context
2020-05-01 21:38:14.807 INFO 1 --- [Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 instances from neighboring DS node
2020-05-01 21:38:14.809 INFO 1 --- [Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew threshold is: 1
2020-05-01 21:38:14.814 INFO 1 --- [Thread-10] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing status to UP
2020-05-01 21:38:14.897 INFO 1 --- [Thread-10] o.s.c.n.e.server.EurekaServerBootstrap : Started Eureka Server
2020-05-01 21:38:15.801 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8761 (http) with context path ''
2020-05-01 21:38:15.804 INFO 1 --- [main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
2020-05-01 21:38:15.814 INFO 1 --- [main] c.c.discovery.DiscoveryApplication : Started DiscoveryApplication in 22.244 seconds (JVM running for 24.177)

```

Figure 13 Eureka Docker

```

$ docker build -t movieApp-discovery .
Invalid argument "movieApp-discovery" for "-t, --tag" flag: invalid reference format: repository name must be lowercase
see 'docker build --help'.

loj@PC-TAM MINGW64 ~/Desktop/MovieApp/2-discovery
$ docker build -t movie-app-discovery .
Sending build context to Docker daemon  58.53MB
Step 1/5 : FROM openjdk:8-jdk-alpine
--> a3562aa0b991
Step 2/5 : VOLUME /tmp
--> Using cache
--> d700440c4ba7
Step 3/5 : EXPOSE 8761
--> Using cache
--> cb42f73a0956
Step 4/5 : ADD ./build/libs/discovery-0.0.1-SNAPSHOT.jar app3.jar
--> f4cb2238fea8
Step 5/5 : ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app3.jar"]
--> Running in a23cf2f3f42f
Removing intermediate container a23cf2f3f42f
--> bfcf2213bb71
Successfully built bfcf2213bb71
Successfully tagged movie-app-discovery:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

loj@PC-TAM MINGW64 ~/Desktop/MovieApp/2-discovery
$ docker run -p 4000:8761 -t movie-app-discovery

```

```
loj@PC-TAM MINGW64 ~/Desktop/MovieApp/1-angular-bootstrap-carousel (master)
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
movie-app-ang	latest	afcab1fdf469	2 minutes ago	45.4MB
<none>	<none>	92adf9867cf6	2 minutes ago	562MB
movie-app-ps	latest	569adba7d39d	13 minutes ago	167MB
movie-app-fs	latest	62425dad9a44	13 minutes ago	169MB
movie-app-discovery	latest	bfcf2213bb71	16 minutes ago	163MB
	latest	61e5c47f5eb5	2 hours ago	45.4MB

Figure 14 Images docker

```

com.netflix.discovery.shared.transport.TransportException: Cannot execute request on any known server
at com.netflix.discovery.shared.transport.decorator.RetryableEurekaHttpClient.execute(RetryableEurekaHttpClient.java:112) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator$1.execute(EurekaHttpClientDecorator.java:59) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.shared.transport.decorator.SessionedEurekaHttpClient.execute(SessionedEurekaHttpClient.java:77) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.shared.transport.decorator.EurekaHttpClientDecorator.register(EurekaHttpClientDecorator.java:56) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.DiscoveryClient.register(DiscoveryClient.java:850) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.InstanceInfoReplicator.run(InstanceInfoReplicator.java:121) ~[eureka-client-1.9.17.jar!/:1.9.17]
at com.netflix.discovery.InstanceInfoReplicator$1.run(InstanceInfoReplicator.java:101) [eureka-client-1.9.17.jar!/:1.9.17]
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) [na:1.8.0_212]
at java.util.concurrent.FutureTask.run(FutureTask.java:266) [na:1.8.0_212]
at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$201(ScheduledThreadPoolExecutor.java:180) [na:1.8.0_212]
at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:293) [na:1.8.0_212]
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149) [na:1.8.0_212]
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624) [na:1.8.0_212]
at java.lang.Thread.run(Thread.java:748) [na:1.8.0_212]
2020-05-01 21:41:35.603 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8585 (http) with context path ''
2020-05-01 21:41:35.611 INFO 1 --- [main] s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8585
2020-05-01 21:41:35.626 INFO 1 --- [main] c.e.F.W.S.FrontWebServiceApplication : Started FrontWebServiceApplication in 32.781 seconds (JVM running for 36.751)

```

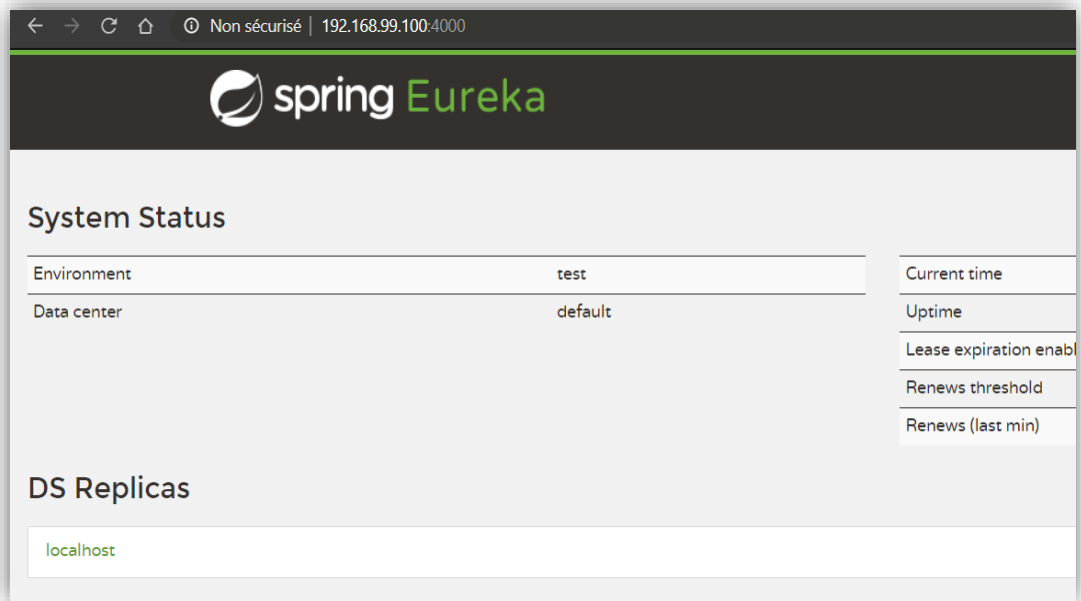


Figure 15 Eureka sous le port de docker



Figure 16 Rating service sous le port de Docker



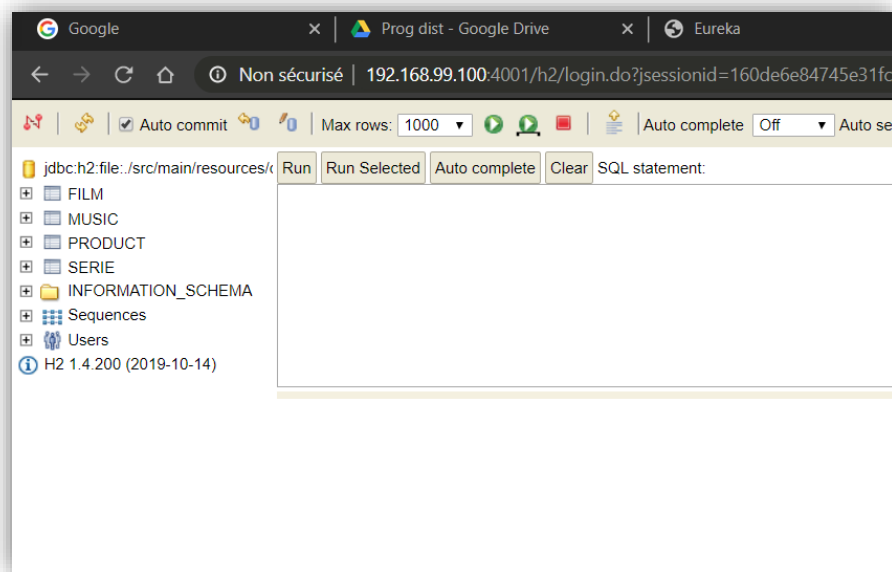


Figure 17 Console H2 du product-service sous Docker