MATHÉMATIQUES ET INFORMATIQUE
Sciences
Université de Paris

Master 1 Informatique
Rapport de projet

Programmation distribué

Chatri CHANSUK

Année universitaire : 2019 – 2020

TABLE DES MATIÈRES

LISTE DES FIGURES

ARCHITECTURE MICROSERVICE

1. Architecture microservice

1.1 Definition/Sémantique

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

Des API REST sont souvent employées pour relier chaque microservice les un 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 recombinés.

Avantages des microservices Maintenance simplifiée.

- Scalabilité facilité 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 attendus, ainsi après l'acheminement par le contrôleur, le modèle se chargera de traités ces commandes en envoyant, ajoutant, modifiant, supprimant des données.

Voici les classes présentes dans l'architecture :

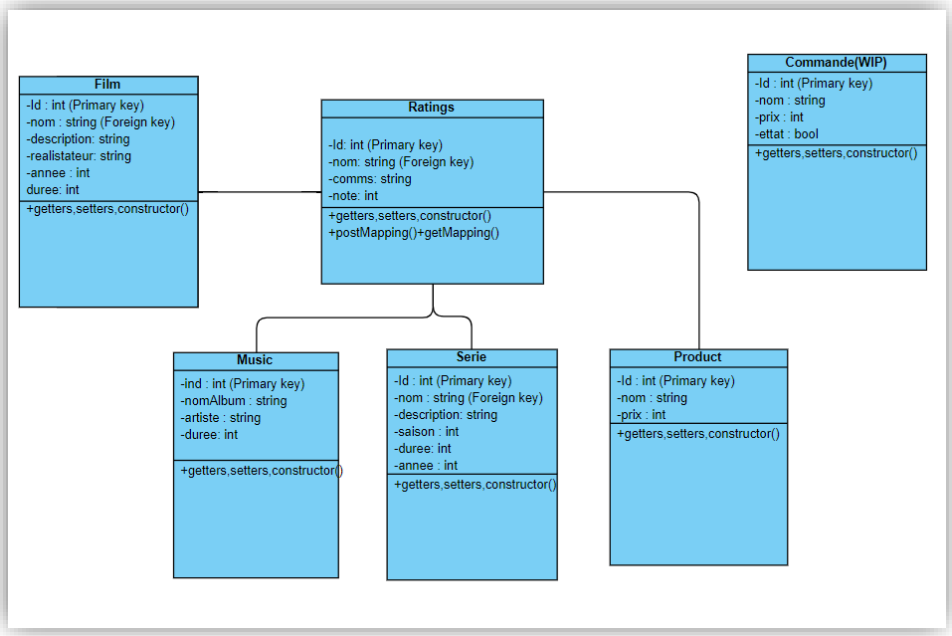


Figure 1 - Diagramme de classe

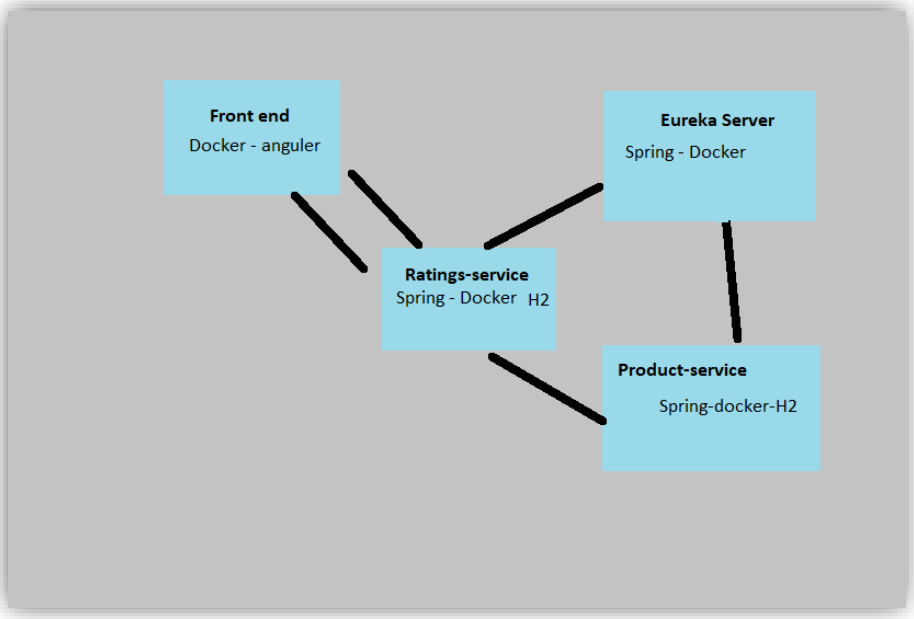


Figure 2 Schéma de l'application

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

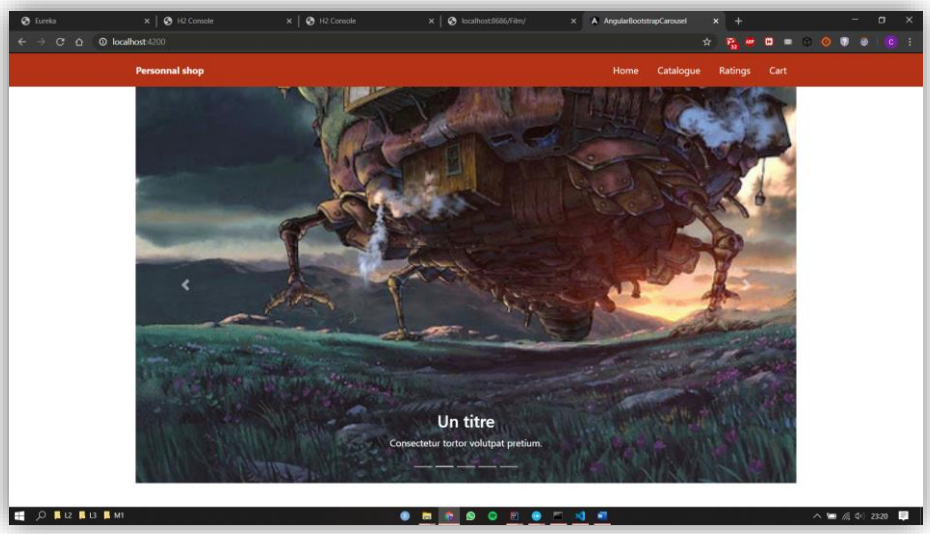


Figure 3 - Accueil de l'application

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

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

Personnal shop/Home : permets 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 réactivités des composants.

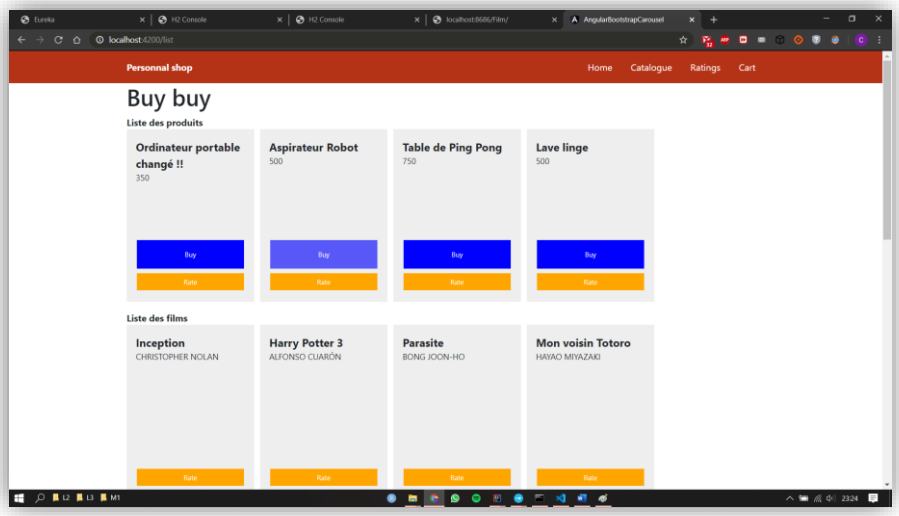


Figure 4 Liste des produits/médias

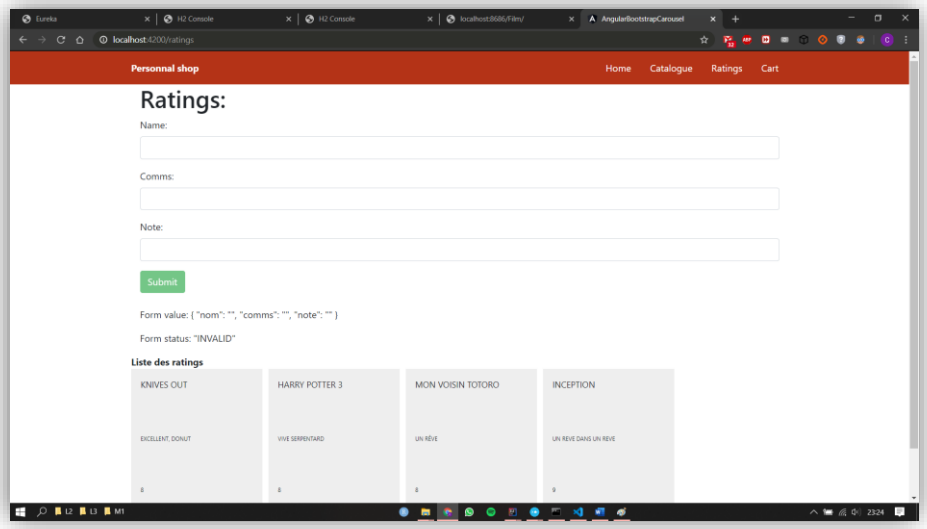


Figure 5 - Insertion d'un rating

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).

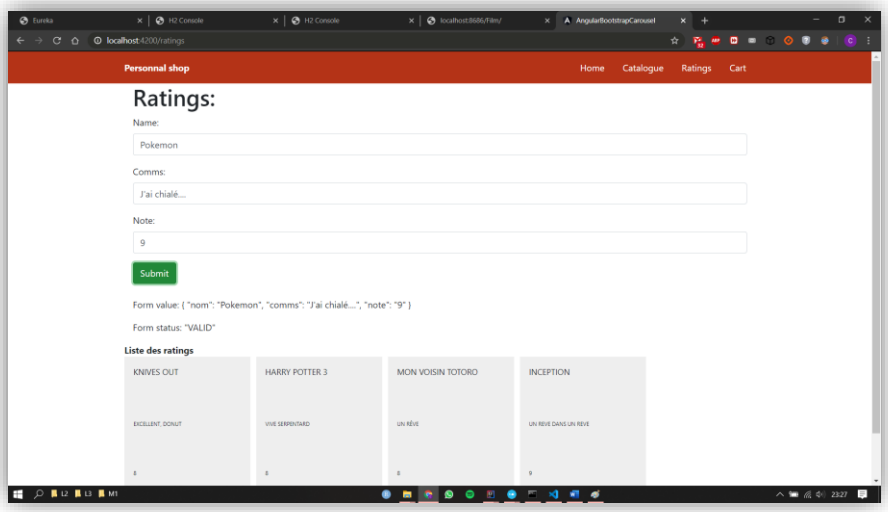


Figure 6 - Json crée

SERVICES

3. Services

3.1 Présentation

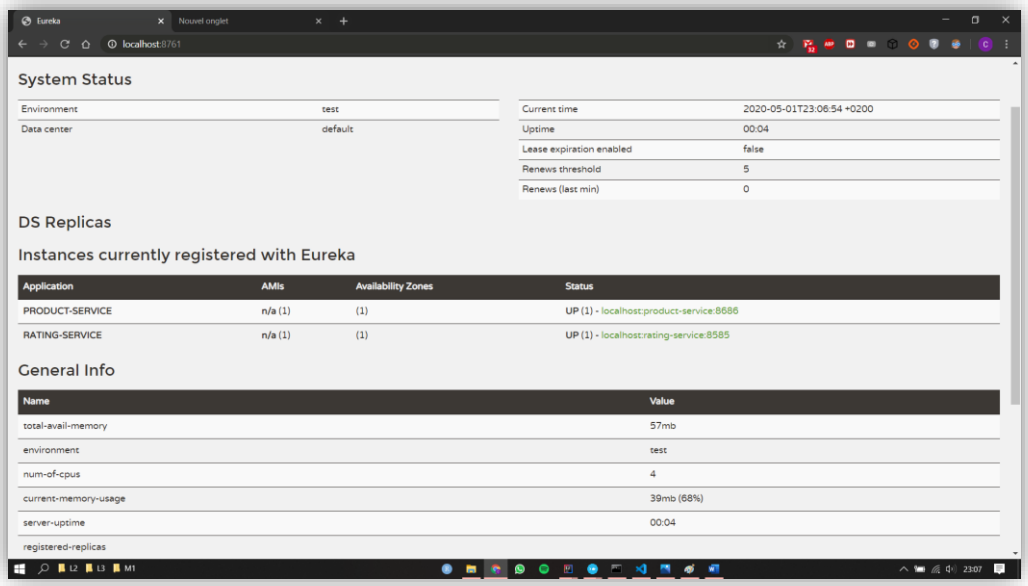


Figure 7 - Eureka server local

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

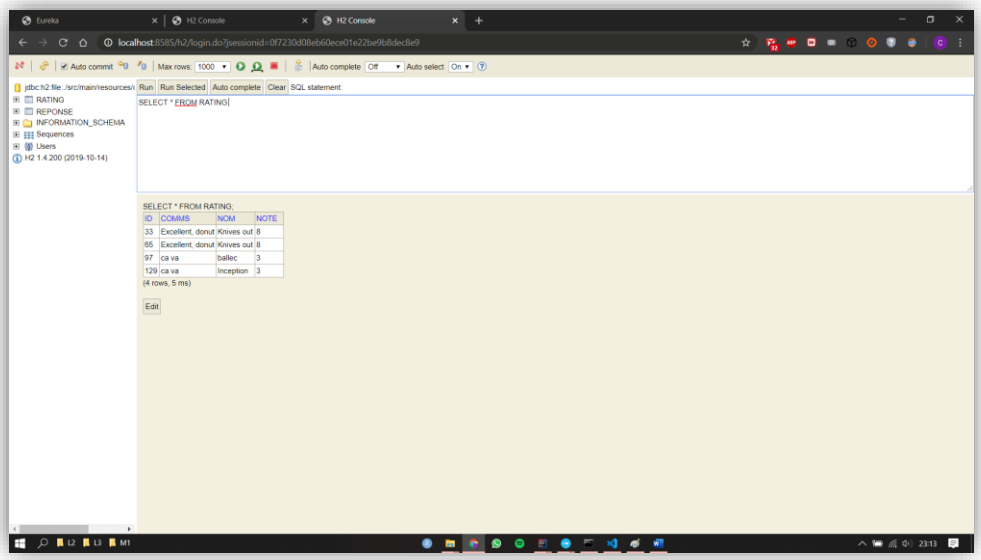


Figure 8 Base de données H2

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.

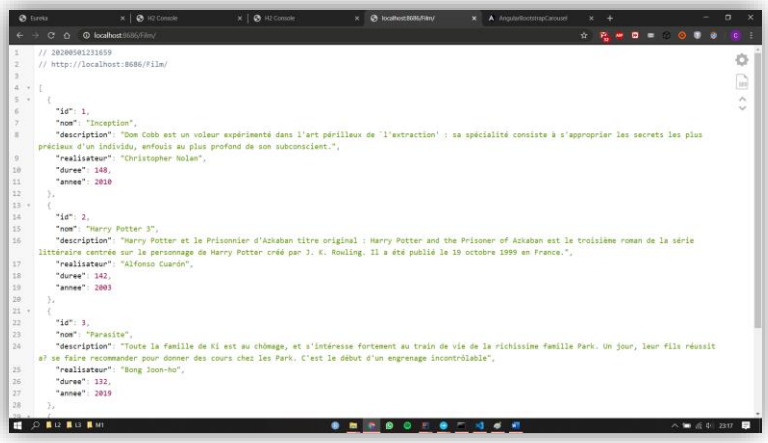


Figure 9 Affiche des données sous format Json

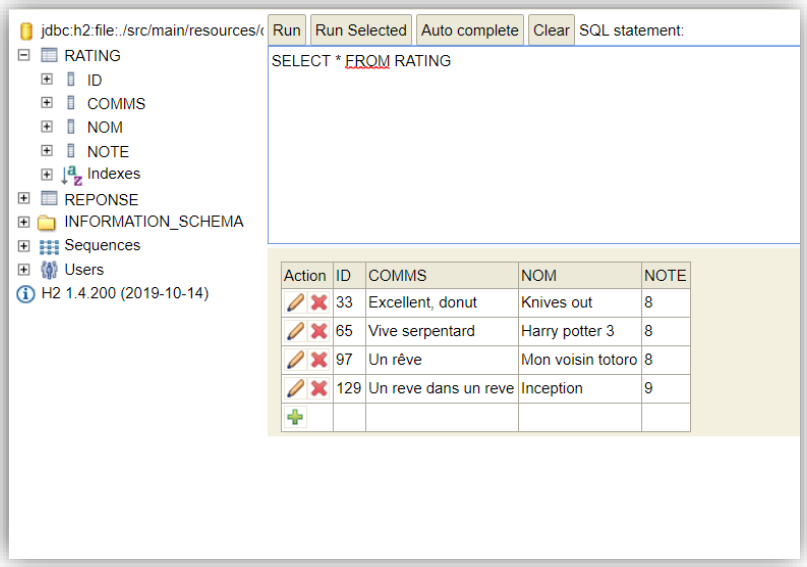


Figure 10 - H2 rating

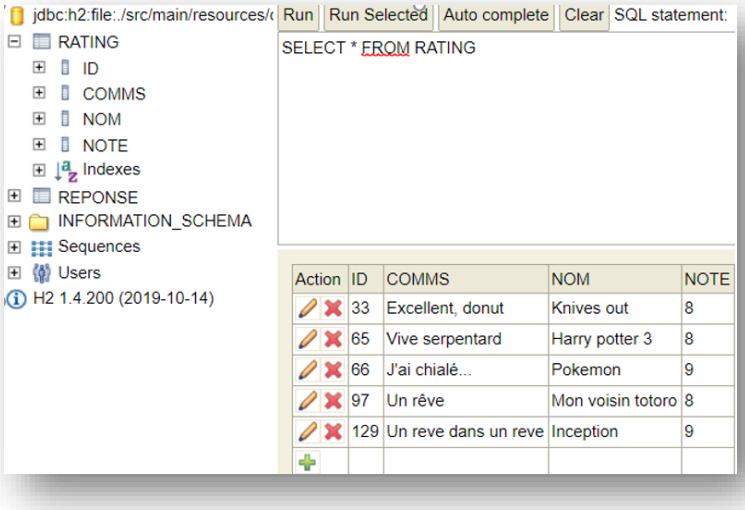


Figure 11 H2 après une requête post

En revenant sur le précédant exemple d'utilisation sur notre navigateur, nous avions passés en paramètres les valeurs nécessaires pour créer un rating. Cependant dans le cas du front, le passage des paramètres dans les requêtes post entrainèrent 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 :

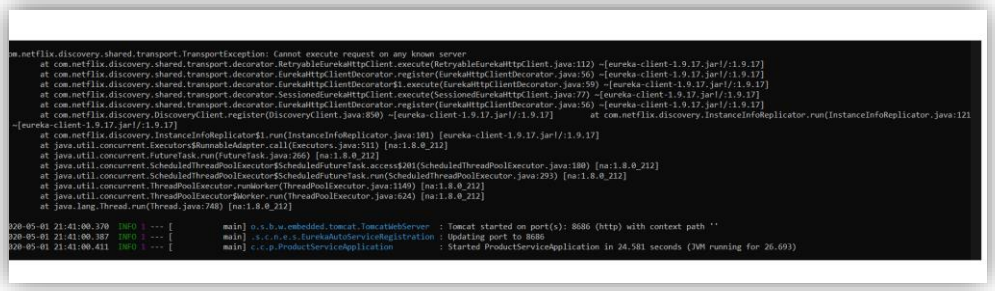


Figure 12 - Product-service docker

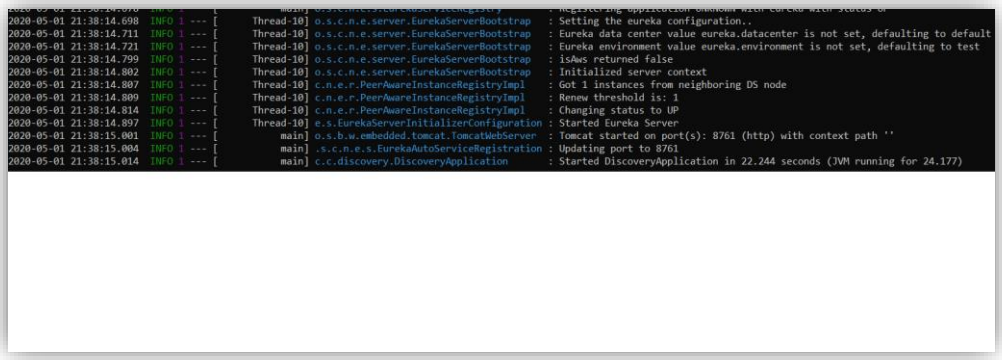
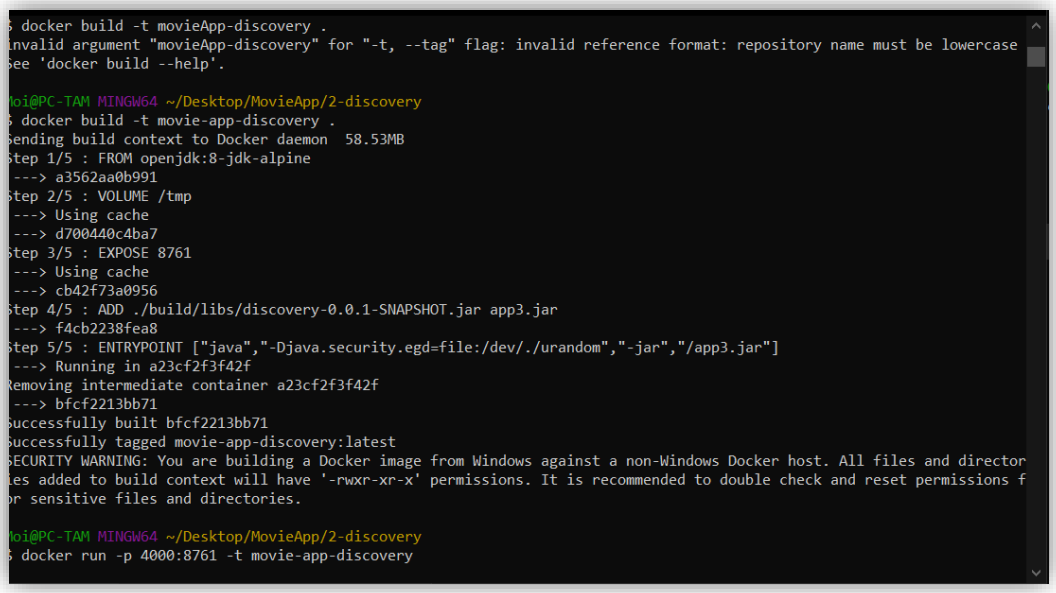


Figure 13 Eureka Docker



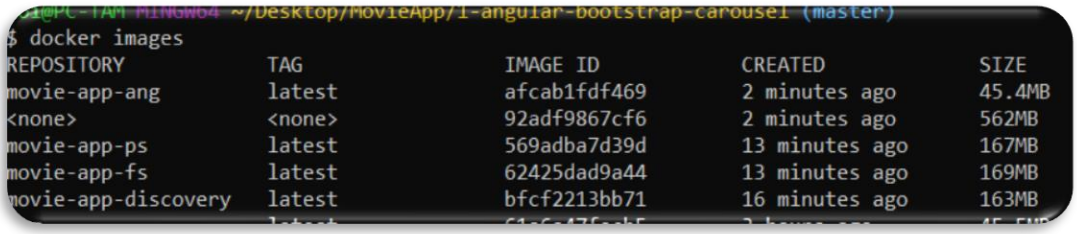
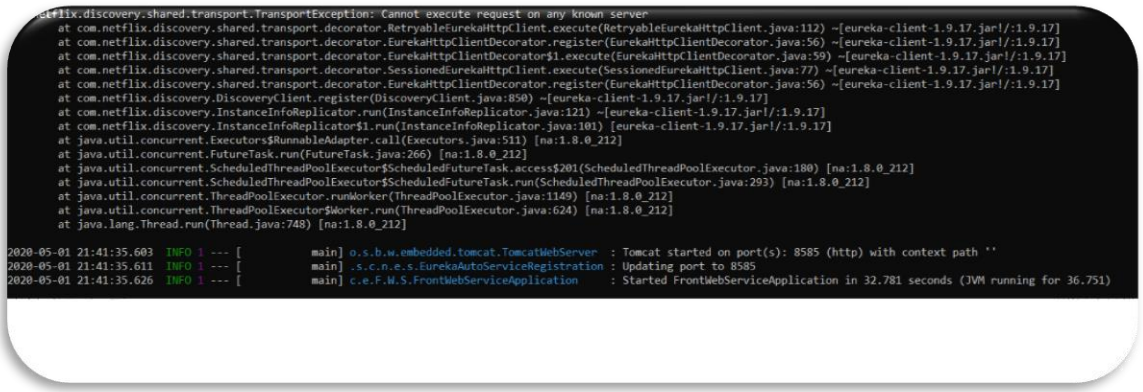


Figure 14 Images docker



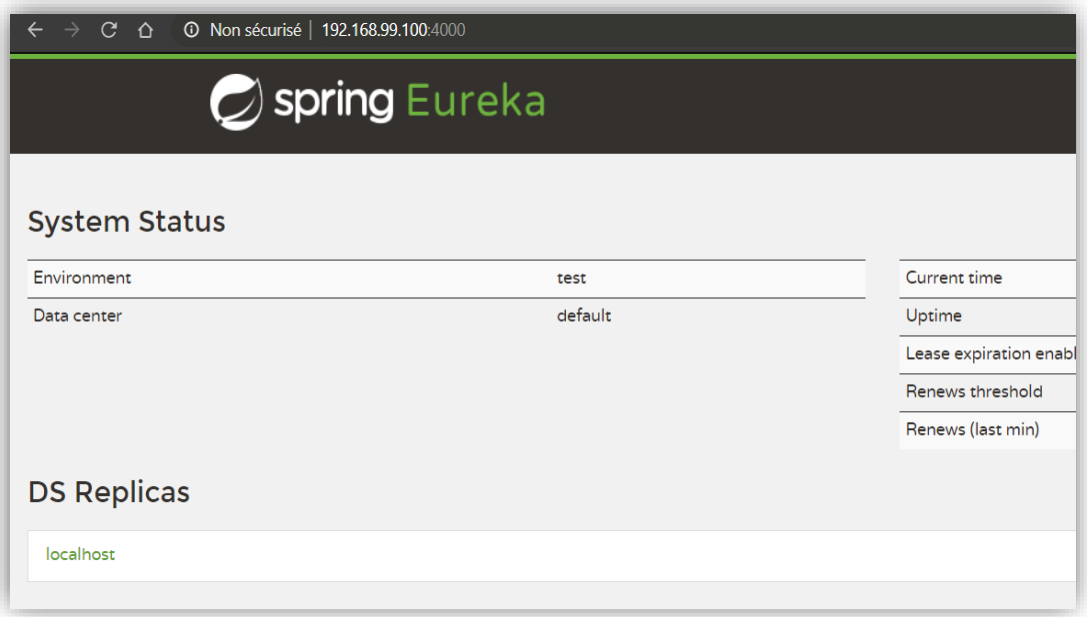


Figure 15 Eureka sous le port de docker

Figure 16 Rating service sous le port de Docker

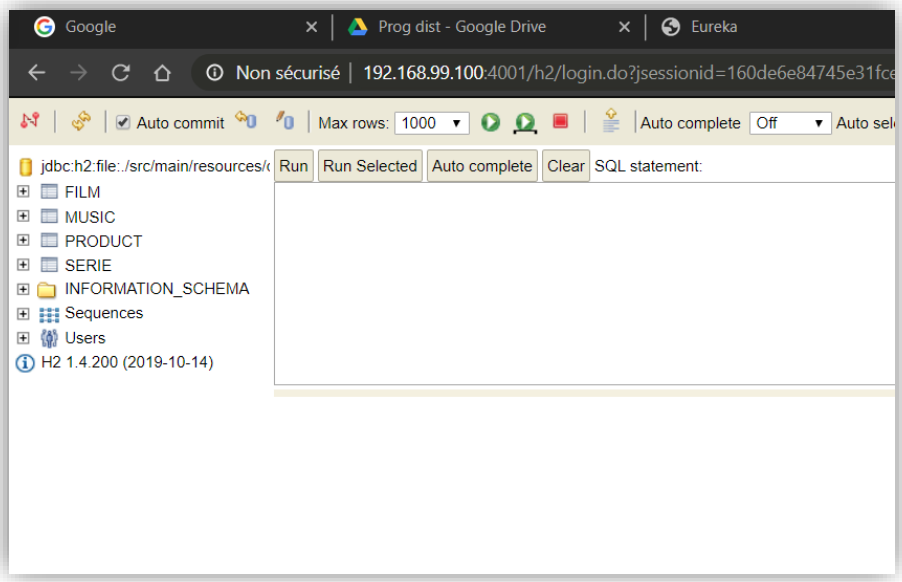


Figure 17 Console H2 du product-service sous Docker