



**Université Abdelmalek Essaadi**  
**Faculté des Sciences et techniques de Tanger**  
**Département Génie Informatique**



# Mini-Projet : Docker

SHELL ET FILTRE LINUX

**Encadré par :**

**Mr. Jadouli ayoub**

**Réalisé par:**

**- IKRAM AIT-KADDI**

# Remerciement

Au terme de ce travail, je tiens à exprimer mon profonde gratitude à mon professeur **Mr. Jadouli ayoub** pour votre suivi et pour votre énorme soutien, qui n'a cessé de me prodiguer et de me guidé tout au long de ce semestre, ainsi pour votre générosité en matière de formation et d'encadrement.

Je tiens à vous remercier monsieur de m' avoir incités à travailler en mettant à ma disposition vos expériences et votre compétence.

# Plan

1- Introduction .....	4
1-1 Présentation de l'application .....	5
1-2 Objectif de ce Projet.....	5
1-3 l'architecture de projet.....	6
2- Standalone Container .....	7
2-1 definition :.....	7
2-2 Configuration des fichiers.....	8
2-3 Création et lancement de l'image.....	14
3- Docker Compose.....	17
3-1 definition :.....	17
3-2 Configuration des fichiers.....	18
3-3 Exécution et les Testes.....	28
4- Conclusion.....	33

## 1 - INTRODUCTION :

**Docker** est une plateforme permettant de lancer certaines applications dans des conteneurs logiciels. Selon la firme de recherche sur l'industrie 451 Research, « Docker est un outil qui peut emballer une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur ». Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

Cette approche permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, une machine nue, etc

Utiliser Docker pour créer et gérer des conteneurs peut simplifier la mise en œuvre de systèmes distribués en permettant à de multiples applications, tâches de fond et autres processus de s'exécuter de façon autonome sur une seule machine physique ou à travers un éventail de machines isolées. Ceci permet de déployer des nœuds en tant que ressources sur besoin, fournissant ainsi une plateforme de déploiement de style PAAS et l'extensibilité de systèmes tels Apache Cassandra, MongoDB ou Riak, ainsi que la simplification de la création et maintenance de queues de tâches ou autres systèmes distribués.

## 1-1 Présentation de l'application :

Une application avec une architecture décentralisée basée sur micro-services consommable depuis une application web front end de type SPA/WPA. L'application gère le suivi des détails de la propriété dans l'immobilier à travers des smart contrats. Les smart contrats permettent une alternative plus transparente et moins chère à la gestion des titres de propriété. Les vices de titre peuvent entraver les transferts, ce qui entraîne des frais juridiques. Cependant, les smart contrats gardent une trace de l'historique, de l'emplacement et de tous les autres détails importants d'une propriété qui seront nécessaires pour l'évaluation du titre. Ils aident à éviter la fraude grâce à des codes cryptés qui sont infalsifiables et sécurisés.

Outils : Ethereum, Ganache, Truffle, spring boot, jwt, angular.

## 1-2 Objectif de ce Projet :

Mettre mon application micro services dans Docker avec deux approches:

- **Un container standalone** : un seul container exécutant toute les processus nécessaires pour le fonctionnement de votre application (utiliser un entrypoint Bash avec l'outil supervisord).

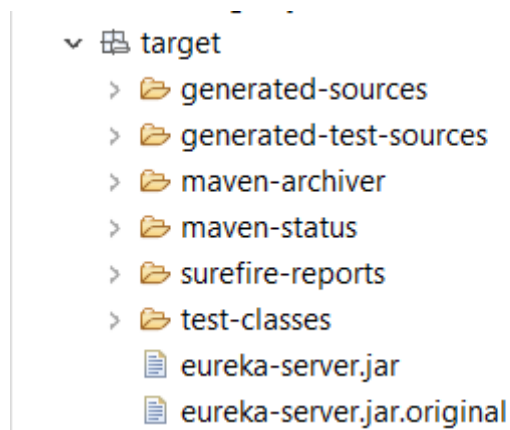
- **Plusieurs containers** : dont un pour chaque service utilisé par votre application (**multi-container Docker applications**). Pour ça utiliser l'outil docker-compose

## 1-3 l'architecture de projet :

Dans la partie backend on a quatre microservices :

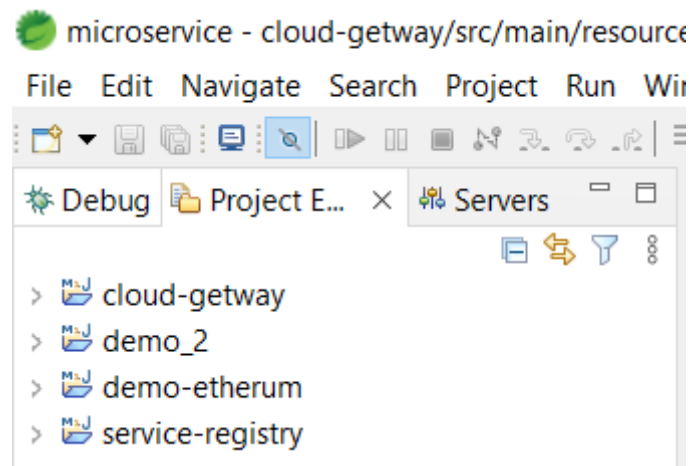
- service-registry : celui de eureka
- service-getway
- demo2 : pour l'authentification
- demo-ethereum : pour la gestion des contractes

Avant de travailler avec ces microservices j'ai généré des jar



Dans la partie front-end on a deux applications angular :

- angular-10-client : pour la partie admin
- angular-10-client2 : pour la partie client



## 2- Standalone Container :

### 2-1 definition :

Traditionnellement, un conteneur Docker exécute un seul processus lorsqu'il est lancé, par exemple dans notre cas on a plusieurs services tel que quatre services spring boot, un service de base données et angular, vous pouvez utiliser supervisord. pour utiliser cette approche on va réaliser les étapes suivants:

- Créez votre supervisord.conf fichier de configuration.
- Créez ensuite un Dockerfile.
- après créez un script pour chaque service
- Ensuite, vous pouvez créer votre image.
- Ensuite, vous pouvez l'exécuter.

### 2-2 Configuration des fichiers :

**Les fichiers : Dockerfile et Supervisord.conf se trouvent dans la racine de dossier qui contient toutes les projets.**

microservice -Supervisord	
Nom	
.metadata	1
angular-10-client	2
angular-10-client2	2
cloud-getway	1
demo_2	1
demo-ethereum	1
import	1
service-registry	1
Dockerfile	2
mongod.conf	1
supervisord.conf	2

### → Création des EntryPoint :

Ces fichiers se trouvent à l'intérieur de chaque projet :

microservice -Supervisord > cloud-getway	
Nom	Modifié le
.mvn	15/05/2022 22:57
.settings	15/05/2022 22:57
src	15/05/2022 22:57
target	16/05/2022 17:24
.classpath	27/12/2021 22:35
.gitignore	27/12/2021 22:35
.project	12/05/2022 15:01
ENTRYPOINT_getway.sh	14/05/2022 13:15
HELP.md	27/12/2021 22:35
mvnw	27/12/2021 22:35
mvnw.cmd	27/12/2021 22:35
pom.xml	22/04/2022 00:41

**ENTRYPOINT\_getway.sh**



```
#!/bin/bash
java -jar target/cloud-getway.jar
```

ENTRYPOINT\_demo.sh

```
#!/bin/bash
java -jar target/demo_2.jar
```

ENTRYPOINT\_ethereum.sh

```
#!/bin/bash
java -jar target/demo-ethereum.jar
```

ENTRYPOINT\_registry.sh

```
#!/bin/bash
java -jar target/eureka-server.jar
```

ang.sh

```
#!/bin/bash
exec ng serve --host 0.0.0.0
```

**Pour la base de données :**

**➔ L'emplacement de import.sh :**

← → ▾ ↑ 📁 > microservice -Supervisord > import			
Nom	Modifié le	Type	Taille
📁 mongo-imp	16/05/2022 07:23	Dossier de fichiers	
📄 import.sh	16/05/2022 07:45	Shell Script	1 Ko

Pour les fichier Json sont importé dans la base de données pour créer une autre base de donnée dans l'image de mongodb avec le script : import.sh

```
#!/bin/bash

mongoimport -d bezkoderdb -c users --jsonArray --file
/opt/mongo/users.json
mongoimport -d bezkoderdb -c roles --jsonArray --file
/opt/mongo/roles.json
```

```
mongoimport -d bezkoderdb -c bienlm --jsonArray --file
/opt/mongo/bienlm.json
mongoimport -d bezkoderdb -c acteurs --jsonArray --file
/opt/mongo/acteurs.json
```

## → Front end

← → ▾ ↑ > Ikram Ait kaddi > Bureau > microservice -Supervisord > angular-10-client			
Nom	Modifié le	Type	Taille
e2e	15/05/2022 13:23	Dossier de fichiers	
node_modules	15/05/2022 13:30	Dossier de fichiers	
src	15/05/2022 13:30	Dossier de fichiers	
.browserslistrc	12/01/2022 11:08	Fichier BROWSERS...	1 Ko
.editorconfig	12/01/2022 11:08	Fichier EDITORCO...	1 Ko
.gitignore	12/01/2022 11:08	Document texte	1 Ko
ang.sh	16/05/2022 13:56	Shell Script	1 Ko
angular.json	12/01/2022 11:08	Fichier JSON	4 Ko
Dockerfile	15/05/2022 10:23	Fichier	1 Ko
karma.conf.js	12/01/2022 11:08	Fichier de JavaScri...	2 Ko
package.json	12/01/2022 11:08	Fichier JSON	2 Ko
package-lock.json	12/01/2022 11:16	Fichier JSON	1 114 Ko
README.md	12/01/2022 11:08	Fichier MD	4 Ko
tsconfig.app.json	12/01/2022 11:08	Fichier JSON	1 Ko
tsconfig.base.json	12/01/2022 11:08	Fichier JSON	1 Ko
tsconfig.json	12/01/2022 11:08	Fichier JSON	1 Ko
tsconfig.spec.json	12/01/2022 11:08	Fichier JSON	1 Ko
tslint.json	12/01/2022 11:08	Fichier JSON	4 Ko

## → Ang.sh

```
#!/bin/bash
exec ng serve --host 0.0.0.0
```

## → Le contenu de supervisord.conf :

Le supervisord.conf fichier de configuration contient des directives qui configurent Supervisor et les processus qu'il gère. Le premier bloc [supervisord] fournit la configuration pour le superviseur lui-même. La nodaemon directive est utilisée, ce qui indique à Supervisor de s'exécuter de manière interactive plutôt que de démonisé.

Les blocs suivants gèrent les services que nous souhaitons contrôler. Chaque bloc contrôle un processus séparé. Les blocs contiennent une seule directive, `command`, qui spécifie la commande à exécuter pour démarrer chaque processus.

```
[supervisord]
nodaemon=true

[program:mongo]
command=/usr/bin/mongod --config /etc/mongod.conf
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true
user=mongodb
priority=10

[program:import]
directory=/usr/api
command=/bin/bash -c "/usr/api/import.sh"
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autostart=true
autorestart=true

[program:service-registry]

directory=/usr/api
command=/bin/bash -c "/usr/api/ENTRYPOINT_registry.sh"
autostart=true
autorestart=true

[program:cloud-getway]

directory=/usr/api
command=/bin/bash -c "/usr/api/ENTRYPOINT_getway.sh"
autostart=true
autorestart=true

[program:demo_2]

directory=/usr/api
command=/bin/bash -c "/usr/api/ENTRYPOINT_demo.sh"
autostart=true
autorestart=true

[program:demo-ethereum]
```

```

directory=/usr/api
command=/bin/bash -c "/usr/api/ENTRYPOINT_etherum.sh"
autostart=true
autorestart=true

[program:angular]

directory=/usr/api/angular-10-client
command=/bin/bash -c "/usr/api/angular-10-client/ang.sh"
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autostart=true
autorestart=true

```

## ➔ Le contenu de Dockerfile :

Le Dockerfile permet de configurer et de créer rapidement une image pour la rendre partageable plus facilement.

- **FROM** permet de définir depuis quelle base votre image va être créé ici on l'image de ubuntu focale
- **RUN** permet de lancer une commande, mais aura aussi pour effet de créer une image intermédiaire, par exemple : l'installation de mongoDB ou angular...
- **ADD** permet de copier un fichier depuis la machine hôte ou depuis une URL, j'ai placé toutes les fichiers dans le dossier /usr/api
- **EXPOSE** permet d'exposer un port du container vers l'extérieur
- **CMD** détermine la commande qui sera exécutée lorsque le container démarrera, le /usr/bin/supervisord binaire doit être exécuté au lancement du conteneur.
- **WORKDIR** permet de définir le dossier de travail ( /usr/api ) pour toutes les autres commandes (par exemple RUN, CMD et ADD)

```
FROM ubuntu:focal
```

```
# Update Ubuntu
```

```
RUN apt-get update --fix-missing && apt-get -y upgrade
```

```
# Installer supervisor
```

```
RUN apt-get -y install software-properties-common supervisor
```

```
RUN apt-get -y update
```

**RUN apt install wget**

**#stage installation node js and angular**

**RUN apt-get -y install curl**

**RUN curl -sL https://deb.nodesource.com/setup\_16.x | bash -**

**RUN apt-get -y install nodejs**

**RUN npm install -g @angular/cli --registry=https://registry.npmjs.org**

**#install mongo**

**RUN apt-get install gnupg**

**RUN wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | apt-key add -**

**RUN echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | tee /etc/apt/sources.list.d/mongodb-org-4.2.list**

**RUN apt-get update**

**RUN apt-get install -y mongodb-org**

**RUN echo "mongodb-org hold" | dpkg --set-selections \**

**RUN echo "mongodb-org-server hold" | dpkg --set-selections \**

**RUN echo "mongodb-org-shell hold" | dpkg --set-selections**

**RUN echo "mongodb-org-mongos hold" | dpkg --set-selections**

**RUN echo "mongodb-org-tools hold" | dpkg --set-selections**

**# le fichier de configuration de mongo**

**COPY mongod.conf /etc/mongod.conf**

**VOLUME ["/data/db"]**

**# Définer mountable directories.**

**RUN mkdir /usr/api**

**WORKDIR /usr/api**

**# installer java 17**

**RUN apt-get install -y openjdk-17-jdk-headless**

**#copier les dossier dans /usr/api**

**COPY import /usr/api**

**ADD service-registry /usr/api**

**ADD cloud-getway /usr/api**

**ADD demo\_2 /usr/api**

**ADD demo-ethereum /usr/api**

**COPY angular-10-client /usr/api/angular-10-client**

**#ajouter le droit d'exécution aux script bash**

**#RUN chmod +x /usr/api/\*.sh**



## Installation réussi de mongodb :

```
PS C:\Users\Dell\Desktop\microservice -Supervisord> docker exec -it helo1 /bin/bash
root@a92d05dde331:/usr/api# mongo
MongoDB shell version v4.2.20
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("162a72e3-c4de-465b-b8ed-95fe44b44b7b") }
MongoDB server version: 4.2.20
Server has startup warnings:
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** Remote systems will be unable to connect to this server.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** addresses it should serve responses from, or with --bind_ip_all to
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** bind to all interfaces. If this behavior is desired, start the
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** server with --bind_ip 127.0.0.1 to disable this warning.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten] ** We suggest setting it to 'never'
2022-05-16T06:05:08.920+0000 I CONTROL [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

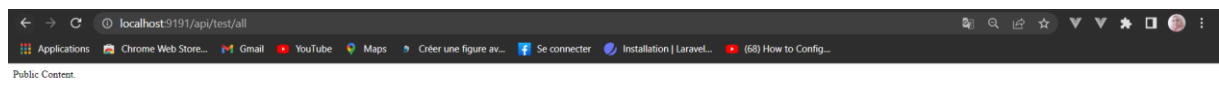
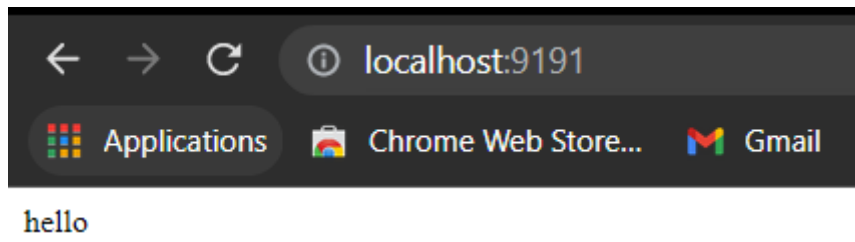
## Tous les services sont bien existés dans eureka :

The screenshot shows the Spring Eureka dashboard in a web browser. The dashboard has a dark header with the 'spring Eureka' logo and navigation links: HOME, LAST 1000 SINCE STARTUP. Below the header, there are three main sections:

- System Status:** A table showing environment (N/A), data center (N/A), current time (2022-05-20T16:32:02+0000), uptime (00:00), lease expiration enabled (false), renew threshold (6), and renew (last min) (0).
- DS Replicas:** A table showing the local host (localhost).
- Instances currently registered with Eureka:** A table with columns: Application, AMIs, Availability Zones, and Status. It lists three instances: API-GATWAY, DEMO-ETHERUM, and DEMO\_2, all with status 'UP'.

Below these sections is a 'General Info' section with a table showing the name and value of various metrics: total-avail-memory (473mb) and num-of-cpus (4).

## Quelque vérification dans le navigateur :





## 3- Docker Compose :

### 3-1 Définition :

Docker Compose est un outil développé par Docker pour créer les architectures logicielles containerisées. Dans cette logique, chaque brique de l'application (code, base de données, serveur web...) sera hébergée par un container. Cet outil repose sur le langage YAML (pour Yet Another Markup Language) pour décrire l'architecture. Une fois celle-ci codée dans un fichier YAML, l'ensemble des services applicatifs seront générés via une commande unique.












### 3-2 Configuration des fichiers :

#### ➔ Génération des Jar :

Dans pom.xml :



















```
.
├─ <build>
├─ │   <plugins>
├─ │   │   <plugin>
├─ │   │   │   <groupId>org.springframework.boot</groupId>
├─ │   │   │   <artifactId>spring-boot-maven-plugin</artifactId>
├─ │   │   </plugin>
├─ │   </plugins>
├─ │   <finalName>cloud-getway</finalName>
├─ </build>
└─ </project>
```

Avec maven build :


















- >  src
- ▼  target
  - >  generated-sources
  - >  generated-test-sources
  - >  maven-archiver
  - >  maven-status
  - >  surefire-reports
  -  cloud-getway.jar
  -  cloud-getway.jar.original
  -  docker-compose.yml
  -  Dockerfile

## ➔ Emplacement des Dockerfiles dans les projets :





















Service de eureka :

- ▼  service-registry [boot]
  - ▼  src/main/java
    - ▼  ma.fstt.demo
      - >  ServiceRegistryApplication.java
      - >  TestControllerr.java
  - ▼  src/main/resources
    -  application.yml
  - >  src/test/java
  - >  JRE System Library [JavaSE-11]
  - >  Maven Dependencies
  - >  src
  -  target
  -  Dockerfile
  -  ENTRYPOINT\_registry.sh
  -  HELP.md
  -  mvnw
  -  mvnw.cmd
  -  pom.xml

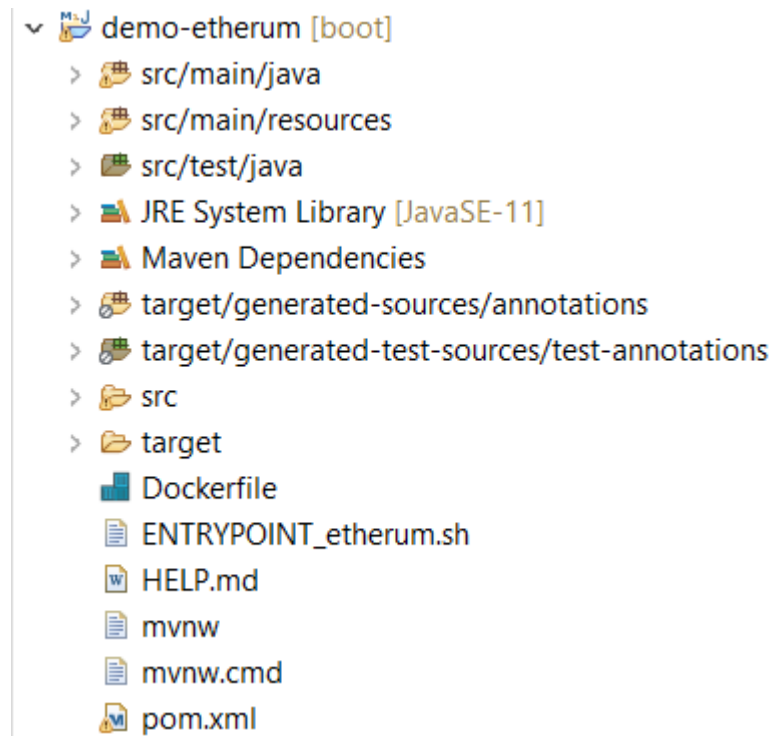
Service de gateway :

- ▼  cloud-getway [boot]
  - >  src/main/java
  - >  src/main/resources
  - >  src/test/java
  - >  JRE System Library [JavaSE-11]
  - >  Maven Dependencies
  - >  target/generated-sources/annotations
  - >  target/generated-test-sources/test-annotations
  - >  src
  - >  target
    -  docker-compose.yml
    -  Dockerfile
    -  ENTRYPOINT\_getway.sh
    -  HELP.md
    -  mvnw
    -  mvnw.cmd
    -  pom.xml

## Service de demo2 :

- ▼  demo\_2 [boot]
  - >  src/main/java
  - ▼  src/main/resources
    -  static
    -  templates
    -  application.yml
    -  docker-compose.yml
  - >  src/test/java
  - >  JRE System Library [JavaSE-11]
  - >  Maven Dependencies
  - >  target/generated-sources/annotations
  - >  target/generated-test-sources/test-annotations
  - >  src
  - >  target
    -  Dockerfile
    -  ENTRYPOINT\_demo.sh
    -  HELP.md
    -  mvnw
    -  mvnw.cmd
    -  pom.xml

## Service de ethereum :



Pour le frontend :

← → ▾ ▴		microservice > angular-10-client	
Nom		Modifié le	
📁	e2e	12/01/2022	
📁	node_modules	12/01/2022	
📁	src	12/01/2022	
📄	.browserslistrc	12/01/2022	
📄	.editorconfig	12/01/2022	
📄	.gitignore	12/01/2022	
📄	angular.json	12/01/2022	
📄	Dockerfile	17/05/2022	
📄	karma.conf.js	12/01/2022	
📄	package.json	12/01/2022	
📄	package-lock.json	12/01/2022	
📄	README.md	12/01/2022	
📄	tsconfig.app.json	12/01/2022	
📄	tsconfig.base.json	12/01/2022	
📄	tsconfig.json	12/01/2022	
📄	tsconfig.spec.json	12/01/2022	
📄	tslint.json	12/01/2022	

Dockerfile des projets Backend par exemple service-getway:

```
FROM adoptopenjdk/openjdk11:latest
ARG JAR_FILE=target/*.jar
RUN mkdir opt/cloud-getway
COPY ${JAR_FILE} /opt/cloud-getway/app.jar
ENTRYPOINT ["java", "-jar", "/opt/cloud-getway/app.jar"]
```

Pour le front end le Dockerfile est:

```
FROM node:16.13.0

RUN mkdir /usr/src/app
WORKDIR /usr/src/app

RUN npm install -g @angular/cli --
registry=https://registry.npmjs.org
COPY . /usr/src/app

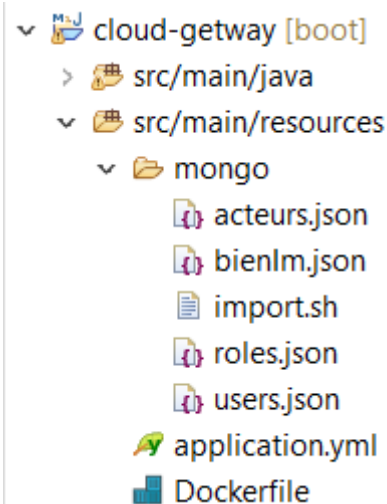
CMD ng serve --host 0.0.0.0 --port 4500
```

Test pour le front end:

```
C:\Users\Dell\Desktop\microservice>docker-compose up
Building angular-service
[+] Building 899.3s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> == transferring dockerfile: 210B
=> [internal] load .dockerignore
=> == transferring context: 2B
=> [internal] load metadata for docker.io/library/node:16.13.0
```

```
Creating angular-service ... done
Attaching to angular-service
angular-service | WARNING: This is a simple server for use in testing or debugging Angular applications
angular-service | locally. It hasn't been reviewed for security issues.
angular-service |
angular-service | Binding this server to an open connection can result in compromising your application or
angular-service | computer. Using a different host than the one passed to the "--host" flag might result in
angular-service | websocket connection issues. You might need to use "--disableHostCheck" if that's the
angular-service | case.
angular-service | (node:9) [DEP0111] DeprecationWarning: Access to process.binding('http_parser') is deprecated.
angular-service | (Use `node --trace-deprecation ...` to show where the warning was created)
angular-service |
angular-service | chunk {main} main.js, main.js.map (main) 171 kB [initial] [rendered]
angular-service | chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
angular-service | chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
angular-service | chunk {styles} styles.js, styles.js.map (styles) 23.9 kB [initial] [rendered]
angular-service | chunk {vendor} vendor.js, vendor.js.map (vendor) 3.01 MB [initial] [rendered]
angular-service | Date: 2022-05-15T08:46:16.063Z - Hash: 7c54f850c49b9d7a0d31 - Time: 22025ms
angular-service | ** Angular Live Development Server is listening on 0.0.0.0:4500, open your browser on http://localhost:4500/ **
angular-service | : Compiled successfully.
```

Emplacement de Dockerfile pour la base de donnée :



Pour les fichier Json sont importés dans la base de donnée pour créer une autre base de donnée dans l'image de mongodb avec le script :  
import.sh

```
#!/bin/bash
```

```
mongoimport -d bezkoderdb -c users --jsonArray --file  
/opt/mongo/users.json  
mongoimport -d bezkoderdb -c roles --jsonArray --file  
/opt/mongo/roles.json  
mongoimport -d bezkoderdb -c bienIm --jsonArray --file  
/opt/mongo/bienIm.json  
mongoimport -d bezkoderdb -c acteurs --jsonArray --file  
/opt/mongo/acteurs.json
```

Dockerfile de mongodb :

```
#stage installation mongodb
```

```
FROM mongo:latest
```

```
#copier les collections dans le répertoire mongo-seed dans l'image
```

```
COPY mongo .
```

```
#ajouter le droit d'exécution aux script bash
```

```
RUN chmod +x import.sh
```

```
RUN ./import.sh
```

## Docker compose:

Le docker-compose contient les éléments suivants :

- **Les services** : mes projets backend et front end et la base de données.
- **Les variables d'environnement** : Les variables d'environnement permettent de découpler la configuration de l'exécutable de l'application.
- **les volumes** : Il existe trois types de volumes : anonymes , nommés et hôtes . Docker gère à la fois les volumes anonymes et nommés, en les montants automatiquement dans des répertoires auto-générés sur l'hôte. Alors que les volumes anonymes étaient utiles avec les anciennes versions de Docker, les volumes nommés sont la voie suggérée de nos jours. Les volumes hôtes nous permettent également de spécifier un dossier existant dans l'hôte. On peut configurer des volumes hôtes au niveau service et des volumes nommés au niveau externe de la configuration, afin de rendre ces derniers visibles aux autres conteneurs et pas seulement à celui auquel ils appartiennent.
- **Déclarer les dépendances** : Souvent, nous devons créer une chaîne de dépendance entre nos services, de sorte que certains services soient chargés avant (et déchargés après) les autres. Nous pouvons obtenir ce résultat grâce au mot clé `depends on`. Dans notre cas toutes les services sont dépend de eureka et les bases données.
- **healthcheck** : Elle indique à Docker comment tester votre container pour vérifier qu'il fonctionne toujours correctement, on l'utilise toujours pour les dépendances entre les services



pour informer un service qu'il doit attendre le d'démarrage e de l'autre.

```
version: '3.8'
```

```
services:
```

```
  javatechiemongodb:
```

```
    container_name: "javatechiemongodb"
```

```
    restart: unless-stopped
```

```
    build:
```

```
      context: ./cloud-getway/src/main/resources
```

```
      dockerfile: Dockerfile
```

```
    hostname: javatechiemongodb
```

```
    ports:
```

```
      - 27017:27017
```

```
    command: mongod --port 27017
```

```
server:
```

```
  container_name: server
```

```
  build:
```

```
    context: ./service-registry
```

```
    dockerfile: Dockerfile
```

```
  ports:
```

```
    - "8761:8761"
```

```
  #healthcheck:
```

```
    #test: ["CMD", "curl", "-f", "http://localhost:8761"]
```

```
    #interval: 30s
```

```
    #timeout: 10s
```

```
    #retries: 5
```

```

cloud-getway:
  container_name: cloud-getway
  build:
    context: ./cloud-getway
    dockerfile: Dockerfile
  environment:
    - eureka.client.service-url.default-
zone=http://server:8761/eureka
  depends_on:
    - server

  ports:
    - "9191:9191"
  #healthcheck:
    #test: ["CMD", "curl", "-f", "http://localhost:9191"]
    # interval: 30s
    #timeout: 10s
    #retries: 5

demo_2:
  container_name: demo_2
  build:
    context: ./demo_2
    dockerfile: Dockerfile
  environment:
    - eureka.client.service-url.default-
zone=http://server:8761/eureka

  depends_on:
    - server
    - cloud-getway
    - javatechiemongodb

```

ports:

- "8050:8050"

demo-ethereum:

container\_name: demo-ethereum

build:

context: ./demo-ethereum

dockerfile: Dockerfile

environment:

- eureka.client.service-url.default-zone=http://server:8761/eureka

depends\_on:

- server
- cloud-getway
- javatechiemongodb

ports:

- "8055:8055"

angular-service :

container\_name : angular-service

build : ./angular-10-client

volumes:

- './angular-10-client:/usr/src/app'

ports :

- "4500:4500"

angular-client :

container\_name : angular-client

build : ./angular-10-client2

### volumes:

- './angular-10-client2:/usr/src/app'

### ports :

- "4200:4200"

## 3-3 Exécution et les Testes:

En execute la commande : docker-compose up:

```
C:\Users\Dell\Desktop\microservice>docker-compose up
Creating network "microservice_default" with the default driver
Building javatechiemongodb
[+] Building 12.7s (5/8)
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 270B                                              0.0s
=> [internal] load .dockerignore                                                 0.1s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/mongo:latest                 4.6s
=> [auth] library/mongo:pull token for registry-1.docker.io                    0.0s
```

En attente que

```
server | 2022-05-20 17:17:38.033 INFO 1 --- [a-Eviction
onTime 0ms
javatechiemongodb | {"t":{"$date":"2022-05-20T17:18:07.965+00:00"},
'attr":{"message":"[1653067087:965709][1:0x7fccb0c44700], WT_SESSION.c
x: 51 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint ti
server | 2022-05-20 17:18:38.034 INFO 1 --- [a-Eviction
onTime 0ms
javatechiemongodb | {"t":{"$date":"2022-05-20T17:19:08.014+00:00"},
'attr":{"message":"[1653067148:14854][1:0x7fccb0c44700], WT_SESSION.ch
: 54 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint tim
server | 2022-05-20 17:19:38.034 INFO 1 --- [a-Eviction
onTime 0ms
```

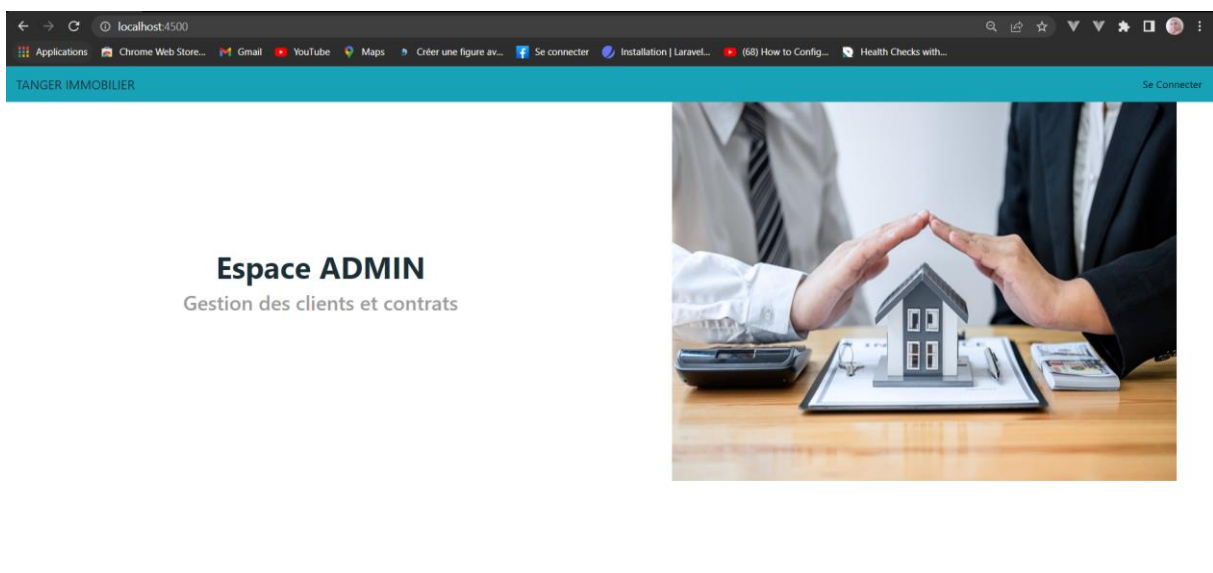
```
Sélection Windows PowerShell

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
WARNING: Image for service angular-service was built because it did not already exist. To rebuild this image you must use
'docker-compose build' or 'docker-compose up --build'.
Building angular-client
[+] Building 8.9s (11/11) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 12B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 1B 0.0s
=> [internal] load metadata for docker.io/library/node:16.13.0 3.3s
[auth] library/node:pull token for registry-1.docker.io 0.0s
[1/5] FROM docker.io/library/node:16.13.0sha256:580a0850049c59a48f06090edd48c9f966c1e6572bbbab369ba3ecbc485 0.0s
=> [internal] load build context 5.0s
=> => transferring context: 3.05MB 4.0s
=> CACHED [2/5] RUN mkdir /usr/src/app 0.0s
=> CACHED [3/5] WORKDIR /usr/src/app 0.0s
=> CACHED [4/5] RUN npm install -g @angular/cli --registry=https://registry.npmjs.org 0.0s
=> CACHED [5/5] COPY . /usr/src/app 0.0s
=> exporting to image 0.3s
=> exporting layers 0.0s
=> writing image sha256:496285a0935d427e2f14f86a0e1ce24509347268e53fa828b9068f677eb948aa 0.0s
=> naming to docker.io/library/microservice-angular-client 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
WARNING: Image for service angular-client was built because it did not already exist. To rebuild this image you must use
'docker-compose build' or 'docker-compose up --build'.
Creating javatechmongodb ... done
Creating angular-client ... done
Creating angular-service ... done
Creating server ... done
Creating cloud-getway ... done
Creating demo-ethereum ... done
Creating demo_2 ... done
Attaching to angular-client, javatechmongodb, server, angular-service, cloud-getway, demo_2, demo-ethereum
angular-client | Node packages may not be installed. Try installing with 'npm install'
angular-client | Could not find the '@angular-devkit/build-angular:dev-server' builder's node package.
javatechmongodb | {"t":{"$date":"2022-05-20T18:22:17.943+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"-",
"msg":"Initialized wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},
"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion":0,"maxWireVersion":13},
"isInternalClient":true}}}
javatechmongodb | {"t":{"$date":"2022-05-20T18:22:17.946+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main",
"msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
javatechmongodb | {"t":{"$date":"2022-05-20T18:22:17.947+00:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":"main",
"msg":"No TransportLayer configured during NetworkInterface startup"}
javatechmongodb | {"t":{"$date":"2022-05-20T18:22:17.947+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main",
"msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and
n","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and
```

Pour se connecter à l'admin : utilisez admin1 comme login

Et Ikram 999 comme mot de passe



## Teste dans eureka :

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

**System Status**

Environment	N/A	Current time	2022-05-20T16:32:02 +0000
Data center	N/A	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	0

**DS Replicas**

localhost

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
API-GATWAY	n/a (1)	(1)	UP (1) - a1f3c7e5d290:API-GATWAY:9191
DEMO-ETHERUM	n/a (1)	(1)	UP (1) - 3325e8168cab:demo-ethereum:8055
DEMO_2	n/a (1)	(1)	UP (1) - 99f09857d944:demo_2:8050

**General Info**

Name	Value
total-avail-memory	473mb
num-of-cpus	4

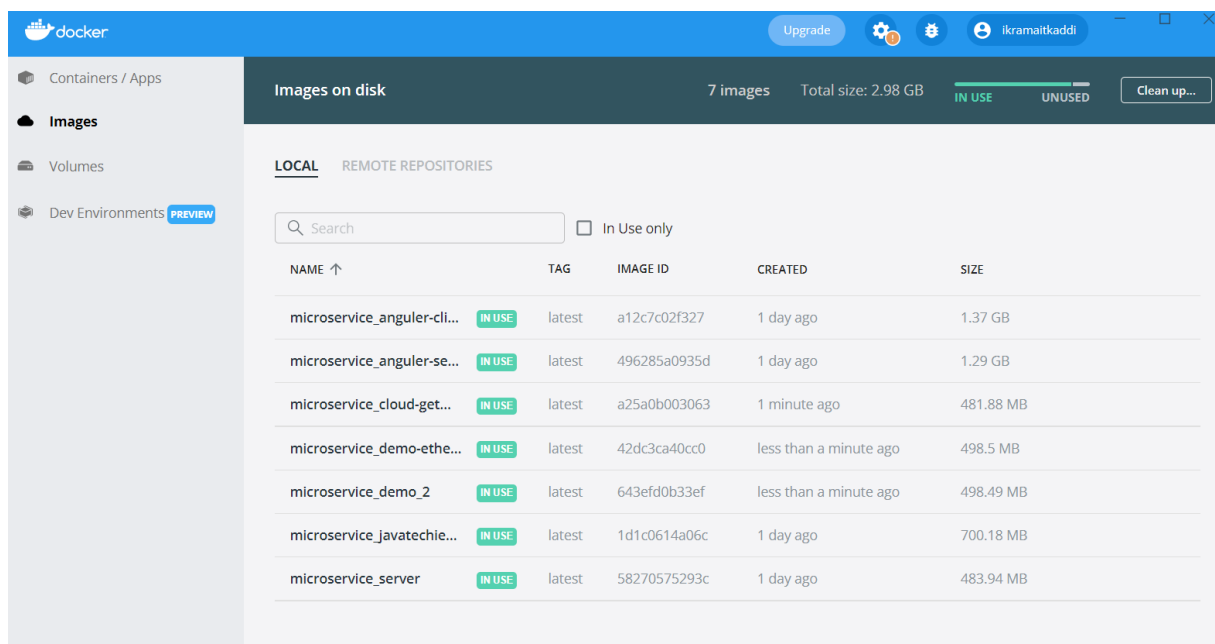
## Vérifier dans Docker desktop :

The screenshot shows the Docker Desktop interface. The left sidebar has a menu with 'Containers / Apps' selected, and other options like 'Images', 'Volumes', and 'Dev Environments' (with a PREVIEW tag). The main area shows a list of running containers under the 'microservice' namespace.

**microservice RUNNING**

- anguler-client microservice\_an...  
RUNNING PORT: 4200
- anguler-service microservice\_an...  
RUNNING PORT: 4500
- server microservice\_se...  
RUNNING PORT: 8761
- javatechiemongodb microservice\_ja...  
RUNNING
- cloud-getway microservice\_cl...  
RUNNING PORT: 9191
- demo-etherum microservice\_de...  
RUNNING PORT: 8055
- demo\_2 microservice\_de...  
RUNNING PORT: 8050

At the bottom right of the container list, there are icons for actions like refresh, restart, stop, and delete, along with a 'DELETE' button.



J'ai exécuté ce script à l'intérieur de container car j'ai recevais une erreur de refus de connexion à mongodb l'ors de son exécution dans docker compose

```
PS C:\Users\bell\Desktop\microservice> docker exec -it javatechiemongodb /bin/bash
root@javatechiemongodb:/# ./import.sh
2022-05-24T20:56:56.456+0000    connected to: mongodb://localhost/
2022-05-24T20:56:56.984+0000    13 document(s) imported successfully. 0 document(s) failed to import.
2022-05-24T20:56:57.013+0000    connected to: mongodb://localhost/
2022-05-24T20:56:57.135+0000    3 document(s) imported successfully. 0 document(s) failed to import.
2022-05-24T20:56:57.150+0000    connected to: mongodb://localhost/
2022-05-24T20:56:57.225+0000    3 document(s) imported successfully. 0 document(s) failed to import.
2022-05-24T20:56:57.238+0000    connected to: mongodb://localhost/
2022-05-24T20:56:57.359+0000    20 document(s) imported successfully. 0 document(s) failed to import.
root@javatechiemongodb:/#
```

```
---
> show dbs
admin            0.000GB
bezkoderdb      0.000GB
config          0.000GB
local           0.000GB
>
```

http://localhost:8050/api/auth/signup

POST

http://localhost:8050/api/auth/signup

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1 { "username" : "kldssdddch",
2   "email" : "addhnhnm@gmail.com",
3   "password" : "ijkram-999",
4   "roles" : [ "admin" ]
5 }
```

Body

Cookies

Headers (14)

Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "message": "User registered successfully!"
3 }
```

root@javatechiemongodb: /

Copyright (C) Microsoft Corporation. Tous droits réservés.

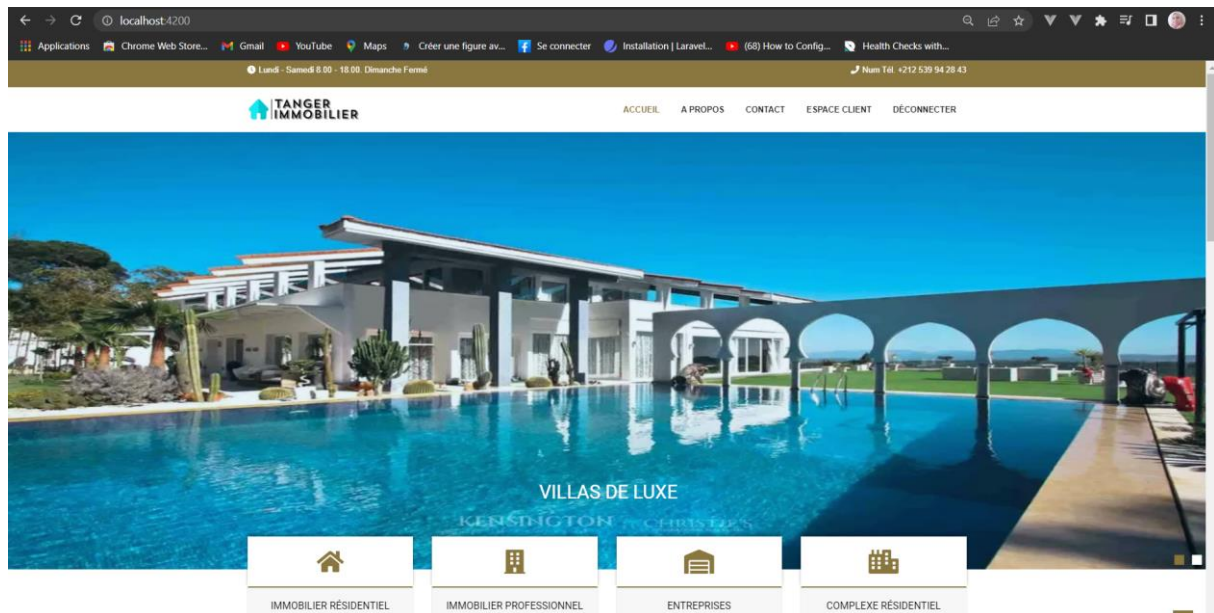
Testez le nouveau système multiplateforme PowerShell <https://aka.ms/pscore6>

```
PS C:\Users\Dell\Desktop\microservice> docker exec -it javatechiemongodb /bin/bash
root@javatechiemongodb:/# mongoimport -d bezkoderdb -c users --jsonArray --file uroot@
root@javatechiemongodb:/# mongoimport -d bezkoderdb -c roles --jsonArray --file roles.
json
```

```
2022-05-21T22:08:09.441+0000    connected to: mongod://localhost/
2022-05-21T22:08:09.779+0000    3 document(s) imported successfully. 0 document(s) failed to import.
```

```
root@javatechiemongodb:/#
```





### Remarque :

- pour docker compose il y a d'autres variables d'environnement mais j'ai placé dans les fichiers application.yml de projet.
- Dans ce projet j'ai travaillé avec docker dans windows car j'ai un problème d'espace mémoire dans mon PC.

Voilà le lien github de projet :

[https://github.com/ikramaitkaddi/Tanger\\_Immobilier\\_JEE\\_Angular](https://github.com/ikramaitkaddi/Tanger_Immobilier_JEE_Angular)

## 4- Conclusion :

Docker est une technologie très efficace pour la gestion de conteneurs uniques. Cependant, à mesure qu'augmente le nombre de conteneurs et d'applications conteneurisées (tous décomposés en centaines de composants), la gestion et l'orchestration se complexifient. Au final, vous devez prendre du recul et regrouper plusieurs conteneurs pour assurer la distribution des services (réseau,

sécurité, télémétrie, etc.) vers tous vos conteneurs. C'est précisément à ce niveau qu'intervient la technologie Kubernetes.