

# **RAPPORT DE CONFIGURATION DU SERVEUR ET DU SERVICE DOCKER**

## **..SOMMAIRE**

### **I-INTRODUCTION**

### **II-INFRASTRUCTURES**

### **III-CONFIGURATION DES MICROSERVICES/CONSTRUCTIONS D'IMAGES**

#### **III-1USER-SERVICE**

#### **III-2.PRODUCT-SERVICE**

#### **III-3 App**

#### **III-4 FICHER .yaml (Création des containers)**

### **IV-Commandes Utiles**

### **V-SECURITE**

### **VI-GESTION DES ACCES**

### **VII-CONCLUSION(point d'amélioration)**

### **I-INTRODUCTION**

Notre projet consistait à la réalisation d'un serveur utilisant docker permettant le l'hébergement et le déploiement notre application Node.js/express à travers l'usage de containers et d'image approprié. Ce document fournit des instructions sur la configuration et l'utilisation du projet `e-maillot`. Ce projet est divisé en plusieurs services conteneurisés à l'aide de Docker et Docker Compose.

## **II-INFRASTRUCTURES**

. POWERSHELL/CMD(L'invite de commandes windows) : windows 11

.DOCKER Desktop: pour windows

.MYSQL

## **III-CONFIGURATION DES MICROSERVICES**

### **III-1USER-SERVICE**

.DOCKERFILE :

```
1  # Utiliser une image de base appropriée pour Node.js
2  FROM node:18
3
4  # Définir le répertoire de travail
5  WORKDIR /app
6
7  # Copier les fichiers package.json et package-lock.json
8  COPY package*.json ./
9
10 # Installer les dépendances
11 RUN rm -rf node_modules
12 RUN npm install
13 RUN npm install bcryptjs
14
15 # Copier tout le reste du code de l'application
16 COPY . .
17
18 # Exposer le port sur lequel l'application va tourner
19 EXPOSE 5000
20
21 # Démarrer l'application
22 CMD ["node", "userApi.js"]
23 |
```

.Package.json file:

```
{
  "name": "userdata",
  "version": "1.0.0",
  "description": "",
  "main": "userApi.js",
  ▶ Debug
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "mysql2": "^3.10.1",
    "rxjs": "^7.8.1"
  }
}
```

### III-2 PRODUCT-SERVICE

.DOCKERFILE :

```
# Utiliser une image de base appropriée pour Node.js
FROM node:18

# Définir le répertoire de travail
WORKDIR /app

# Copier les fichiers package.json et package-lock.json
COPY package*.json ./

# Installer les dépendances
RUN rm -rf node_modules
RUN npm install

# Copier tout le reste du code de l'application
COPY . .

# Exposer le port sur lequel l'application va tourner
EXPOSE 5002

# Démarrer l'application
CMD ["node", "product.js"]
```

Package.json file:

```
{
  "name": "productdata",
  "version": "1.0.0",
  "description": "",
  "main": "product.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.19.2",
    "mysql2": "^3.10.1",
    "rxjs": "^7.8.1"
  }
}
```

### III-3 App (Frontend)

.DOCKERFILE :

```
1 # Etape de build
2 FROM node:18 as build
3
4 # Définir le répertoire de travail
5 WORKDIR /app
6
7 # Copier les fichiers package.json et package-lock.json
8 COPY package*.json ./
9
10 # Installer les dépendances
11 RUN npm install
12
13 # Copier tout le reste du code de l'application
14 COPY . .
15
16 # Construire l'application pour la production
17 RUN npm run build --configuration=production
18
19 # Étape de production
20 FROM nginx:alpine
21
22 # Copier les fichiers de build Angular vers Nginx
23 COPY --from=build /app/dist/app /usr/share/nginx/html
24
25 # Exposer le port sur lequel Nginx s'exécute
26 EXPOSE 80
27
28 # Commande par défaut pour démarrer Nginx
29 CMD ["nginx", "-g", "daemon off;"]
30
```

### III-4 FICHIER .yaml (Création des containers)

```
1 version: '3'
2 services:
3   frontend:
4     build:
5       context: ../frontend/app
6       dockerfile: Dockerfile
7     ports:
8       - "4200:80"
9     volumes:
10      - ../frontend/app:/app
11     environment:
12       - NODE_ENV=development
13     depends_on:
14       - userdata
15       - productdata
16
17   userdata:
18     build:
19       context: ../userdata
20     ports:
21       - "5000:5000"
22     environment:
23       - MYSQL_HOST=userdata-db
24       - MYSQL_USER=root
25       - MYSQL_PASSWORD=rootpassword
26       - MYSQL_DATABASE=userdb
27     depends_on:
28       - userdata-db
29
30   userdata-db:
31     image: mysql:latest
32     environment:
33       MYSQL_ROOT_PASSWORD: rootpassword
34       MYSQL_DATABASE: userdb
35     volumes:
36       - userdata-db:/var/lib/mysql
37       - ../sql/userdb.sql:/docker-entrypoint-initdb.d/userdb.sql
38     ports:
39       - "3307:3306"
40
41   productdata:
42     build:
```



```
40
41 productdata:
42   build:
43     context: ./productdata
44   ports:
45     - "5002:5002"
46   environment:
47     - MYSQL_HOST=productdata-db
48     - MYSQL_USER=root
49     - MYSQL_PASSWORD=rootpassword
50     - MYSQL_DATABASE=productdb
51   depends_on:
52     - productdata-db
53
54 productdata-db:
55   image: mysql:latest
56   environment:
57     MYSQL_ROOT_PASSWORD: rootpassword
58     MYSQL_DATABASE: productdb
59   volumes:
60     - productdata-db:/var/lib/mysql/
61     - ./sql/productdb.sql:/docker-entrypoint-initdb.d/productdb.sql
62   ports:
63     - "3308:3306"
64
65 volumes:
66   userdata-db:
67   productdata-db:
```

## Explications :

### Frontend

```
frontend:
  build:
    context: ./frontend/app
    dockerfile: Dockerfile
  ports:
    - "4200:80"
  volumes:
    - ./frontend/app:/app
  environment:
    - NODE_ENV=development
  depends_on:
    - userdata
    - productdata
```

- Construit l'image Docker à partir du répertoire `./frontend/app` en utilisant le `Dockerfile` spécifique.
- Mappe le port 4200 de l'hôte au port 80 du conteneur.
- Monte le répertoire `./frontend/app` sur `/app` dans le conteneur pour le développement.
- Définit la variable d'environnement `NODE_ENV` à `development`.
- Dépend des services `userdata` et `productdata` pour s'assurer qu'ils démarrent avant.

## Userdata

```
userdata:
  build:
    context: ./userdata
  ports:
    - "5000:5000"
  environment:
    - MYSQL_HOST=userdata-db
    - MYSQL_USER=root
    - MYSQL_PASSWORD=rootpassword
    - MYSQL_DATABASE=userdb
  depends_on:
    - userdata-db
```

- Construit l'image Docker à partir du répertoire `./userdata`.
- Mappe le port 5000 de l'hôte au port 5000 du conteneur.
- Définit les variables d'environnement pour la connexion à la base de données MySQL.
- Dépend du service `userdata-db` pour s'assurer qu'il démarre avant.

## userdata-db

```
userdata-db:
  image: mysql:latest
  environment:
    MYSQL_ROOT_PASSWORD: rootpassword
    MYSQL_DATABASE: userdb
  volumes:
    - userdata-db:/var/lib/mysql
    - ./sql/userdb.sql:/docker-entrypoint-initdb.d/userdb.sql
  ports:
    - "3307:3306"
```

- Utilise l'image MySQL la plus récente.
- Configure le mot de passe root et crée la base de données `userdb`.
- Monte un volume Docker pour persister les données et initialise la base de données avec le script `userdb.sql`.
- Mappe le port 3307 de l'hôte au port 3306 du conteneur.

## Productdata

```
productdata:
  build:
    context: ./productdata
  ports:
    - "5002:5002"
  environment:
    - MYSQL_HOST=productdata-db
    - MYSQL_USER=root
    - MYSQL_PASSWORD=rootpassword
    - MYSQL_DATABASE=productdb
  depends_on:
    - productdata-db
```

- Construit l'image Docker à partir du répertoire `./productdata`.
- Mappe le port 5002 de l'hôte au port 5002 du conteneur.
- Définit les variables d'environnement pour la connexion à la base de données MySQL.
- Dépend du service `productdata-db` pour s'assurer qu'il démarre avant.

## productdata-db

```
productdata-db:
  image: mysql:latest
  environment:
    MYSQL_ROOT_PASSWORD: rootpassword
    MYSQL_DATABASE: productdb
  volumes:
    - productdata-db:/var/lib/mysql/
    - ./sql/productdb.sql:/docker-
      entrypoint-initdb.d/productdb.sql
  ports:
    - "3308:3306"
```

- Utilise l'image MySQL la plus récente.
- Configure le mot de passe root et crée la base de données `productdb`.
- Monte un volume Docker pour persister les données et initialise la base de données avec le script `productdb.sql`.
- Mappe le port 3308 de l'hôte au port 3306 du conteneur.

## Volumes

```
volumes:
  userdata-db:
  productdata-db:
```

Définit des volumes Docker nommés pour persister les données des bases de données `userdata-db` et `productdata-db`.

#### **IV-Commandes Utiles :**

- **docker-compose up --build // Démarrer tous les services**
- **docker-compose down // Arrêter tous les services**
- **docker-compose logs -f // Consulter les logs**
- **docker exec -it <nom\_du\_container> /bin/bash // Accéder à un conteneur**

#### **V-SECURITE**

On a essayé de l'hébergement gratuit sur heroku, mais en vain, connexion local depuis la machine et

#### **VI-GESTION DES ACCES à l'application et aux services**

Accédez à l'application principale via <http://localhost:4200/home>,

au microservice User via <http://localhost:5000/>, et

au microservice Products via <http://localhost:5002>.

#### **VII-CONCLUSION**

Bien que il reste bien des points d'amélioration comme le côté sécurité de l'application ce projet nous a permis de nouveaux acquis à travers la gestion de d'une application via docker et de comprendre le concept de micro-services qui nous paraissait plutôt abstrait.

Cette documentation devrait vous fournir les informations nécessaires pour configurer et exécuter le projet `e-maillot`.