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 FICHIER .yml (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

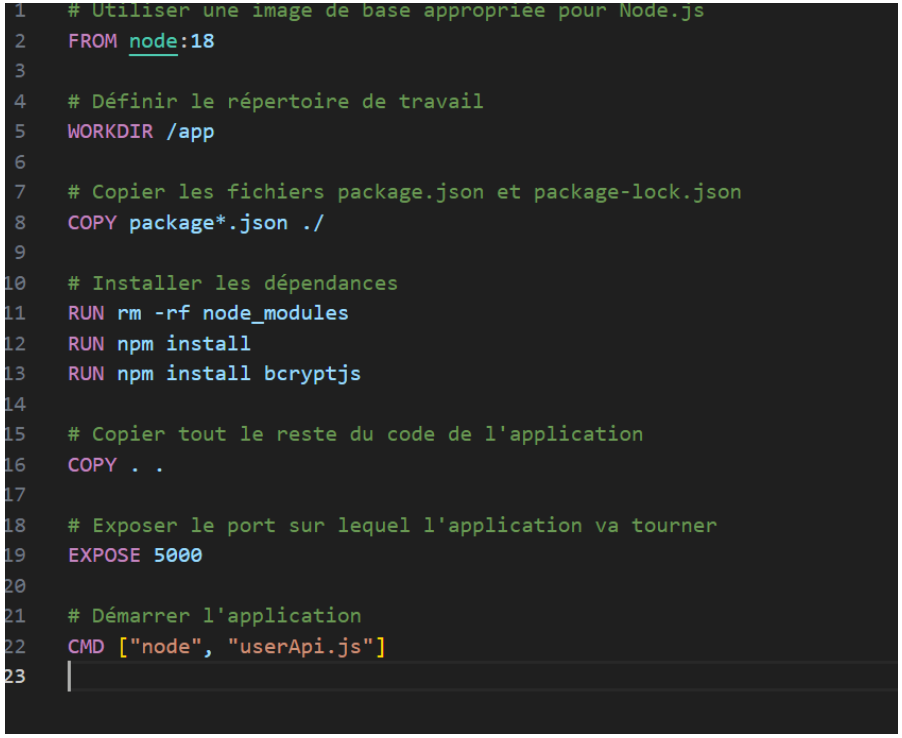
.DOCCKER Desktop: pour windows

.MYSQL

III-CONFIGURATION DES MICROSERVICES

III-1USER-SERVICE

.DOCKERFILE :

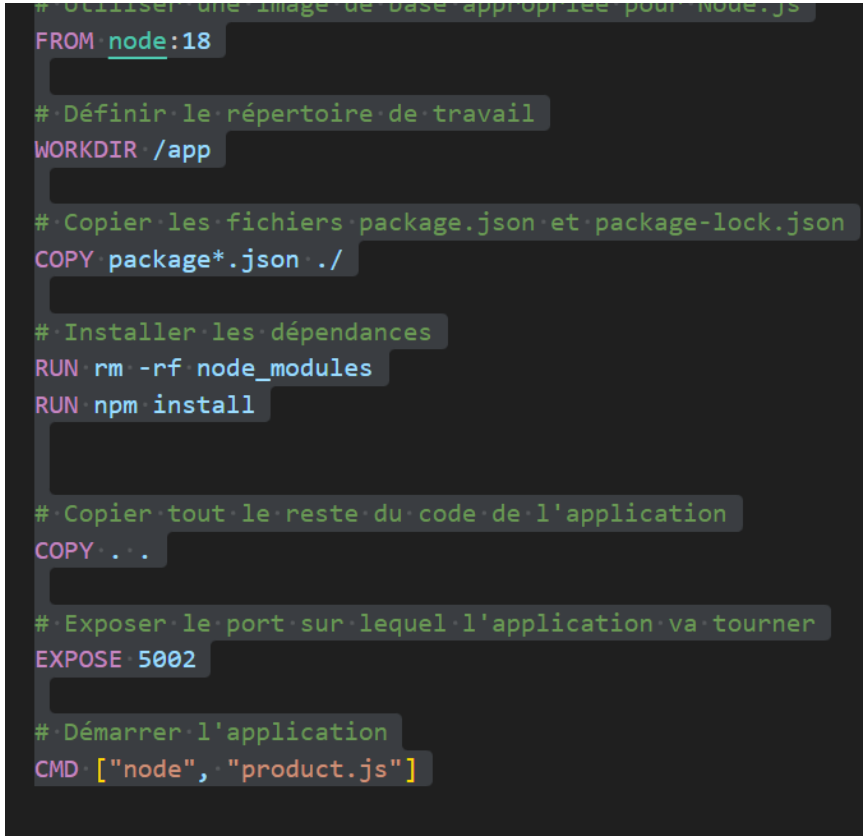


```dockerfile
# Utiliser une image de base appropriée pour Node.js
FROM node:18

# 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
RUN npm install bcryptjs

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

# Exposer le port sur lequel l'application va tourner
EXPOSE 5000

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

.Package.json file:

III-2 PRODUCT-SERVICE

.DOCKERFILE :

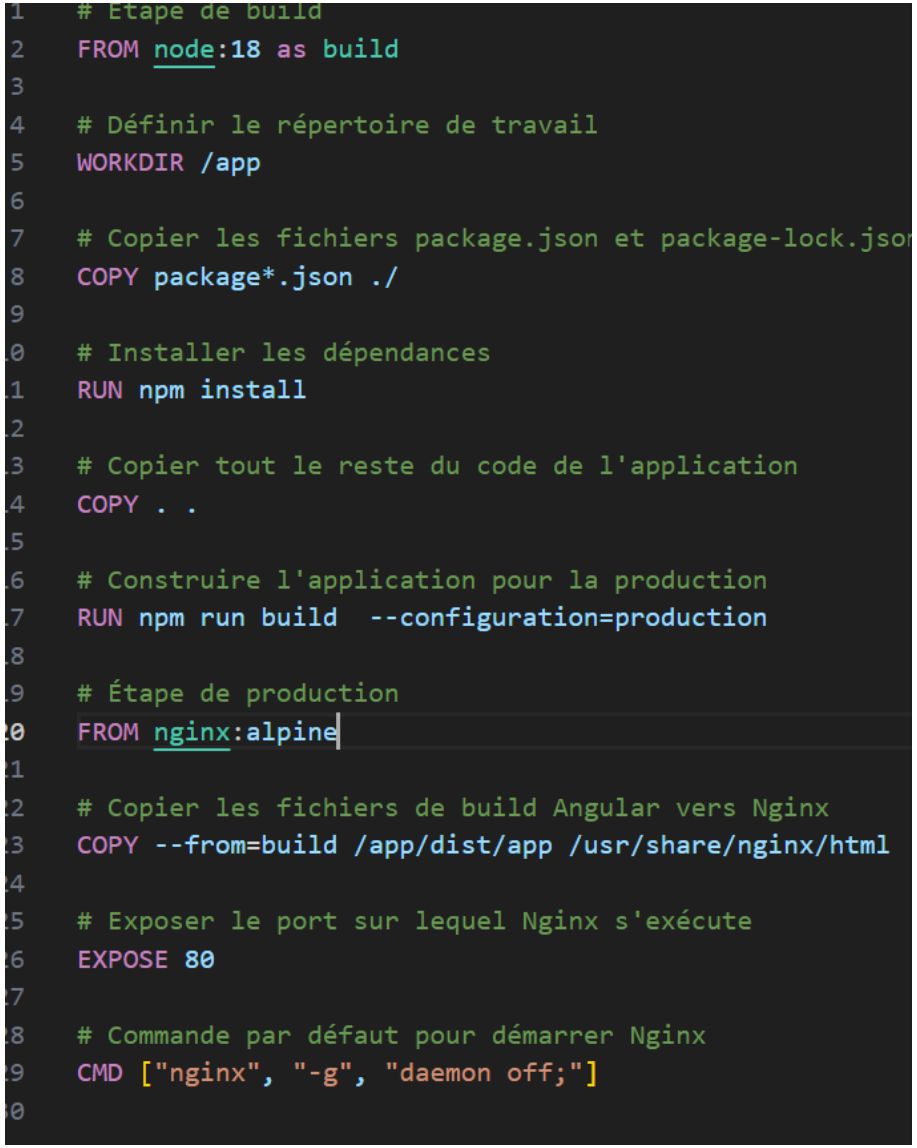


```
# Utiliser une image de base appropriée pour Node.js
FROM node:18

# 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:

III-3 App (Frontend)

.DOCKERFILE :



```dockerfile
# Étape de build
FROM node:18 as build

# Définir le répertoire de travail
WORKDIR /app

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

# Installer les dépendances
RUN npm install

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

# Construire l'application pour la production
RUN npm run build  --configuration=production

# Étape de production
FROM nginx:alpine

# Copier les fichiers de build Angular vers Nginx
COPY --from=build /app/dist/app /usr/share/nginx/html

# Exposer le port sur lequel Nginx s'exécute
EXPOSE 80

# Commande par défaut pour démarrer Nginx
CMD ["nginx", "-g", "daemon off;"]
```

III-4 FICHIER .yml (Création des containers)

```yaml
version: '3'
services:
  frontend:
    build:
      context: ./frontend/app
      dockerfile: Dockerfile
    ports:
      - "4200:80"
    volumes:
      - ./frontend/app:/app
    environment:
      - NODE_ENV=development
    depends_on:
      - userdata
      - productdata

  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

  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"

  productdata:
    build:
```

```yaml

  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

  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"

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

Explications :

Frontend

```yaml
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

```yaml
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

```yaml
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 essayer 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 côte 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`.