Rapport de TP :

Installation et Configuration de Docker

Objectif du TP

L'objectif de ce TP est d'installer Docker et de mettre en place une architecture composée de trois conteneurs :

- **Cont1 : Client (Frontend)**
- **Cont2 : Métier (Backend)**
- **Cont3 : Data (Base de données PostgreSQL)**

Nous avons également traité le problème de la persistance des données à l'aide des volumes Docker et utilisé Docker Hub pour stocker et récupérer nos images.

1. Installation de Docker

L'installation de Docker a été réalisée en suivant les étapes suivantes :

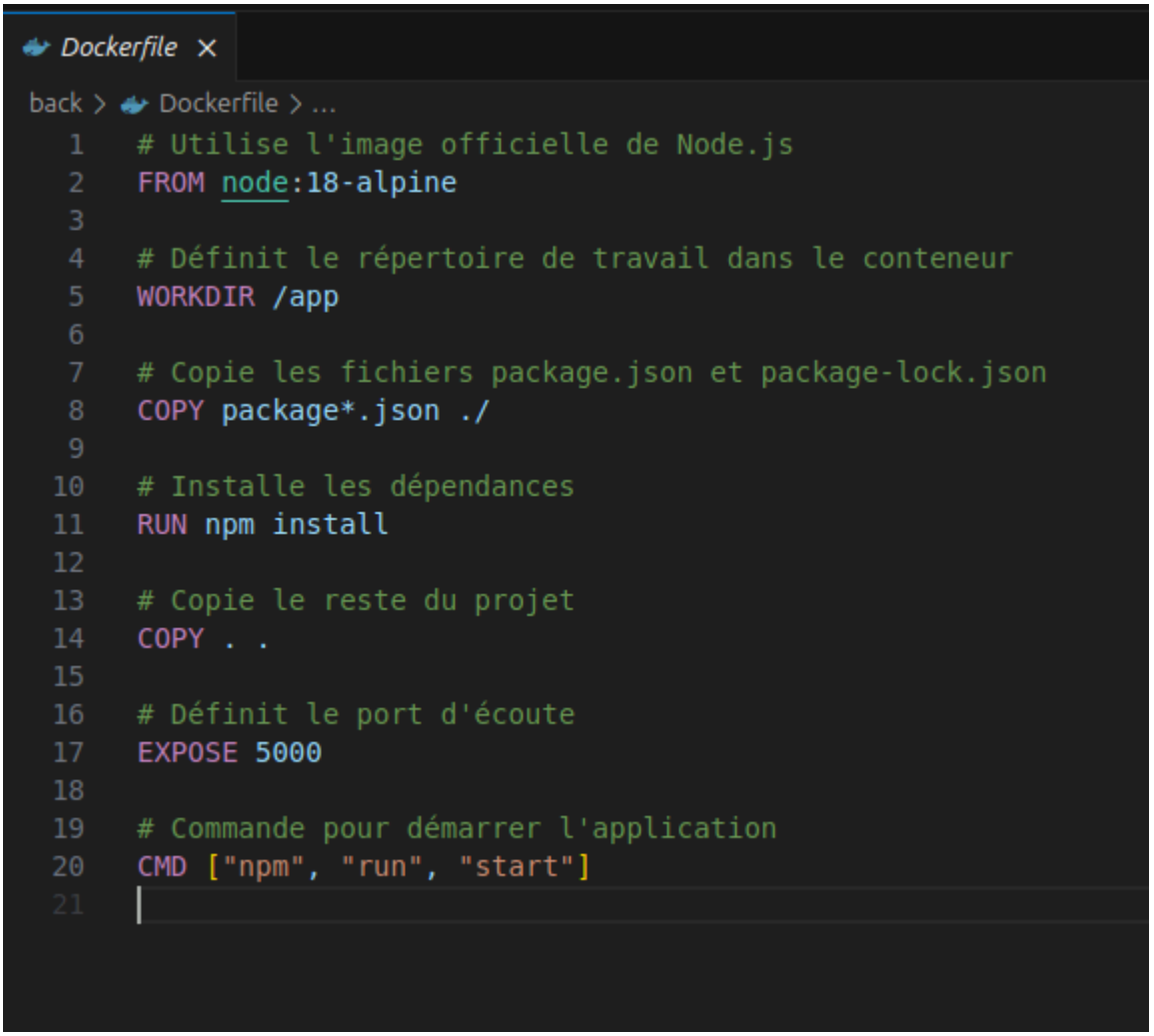
1. **Téléchargement et installation** :Docker Engine (Linux)
2. **Vérification de l'installation** avec la commande : docker --version
3. **Lancement et activation** du service Docker

2. Création des Conteneurs

2.1 Création des Dockerfiles

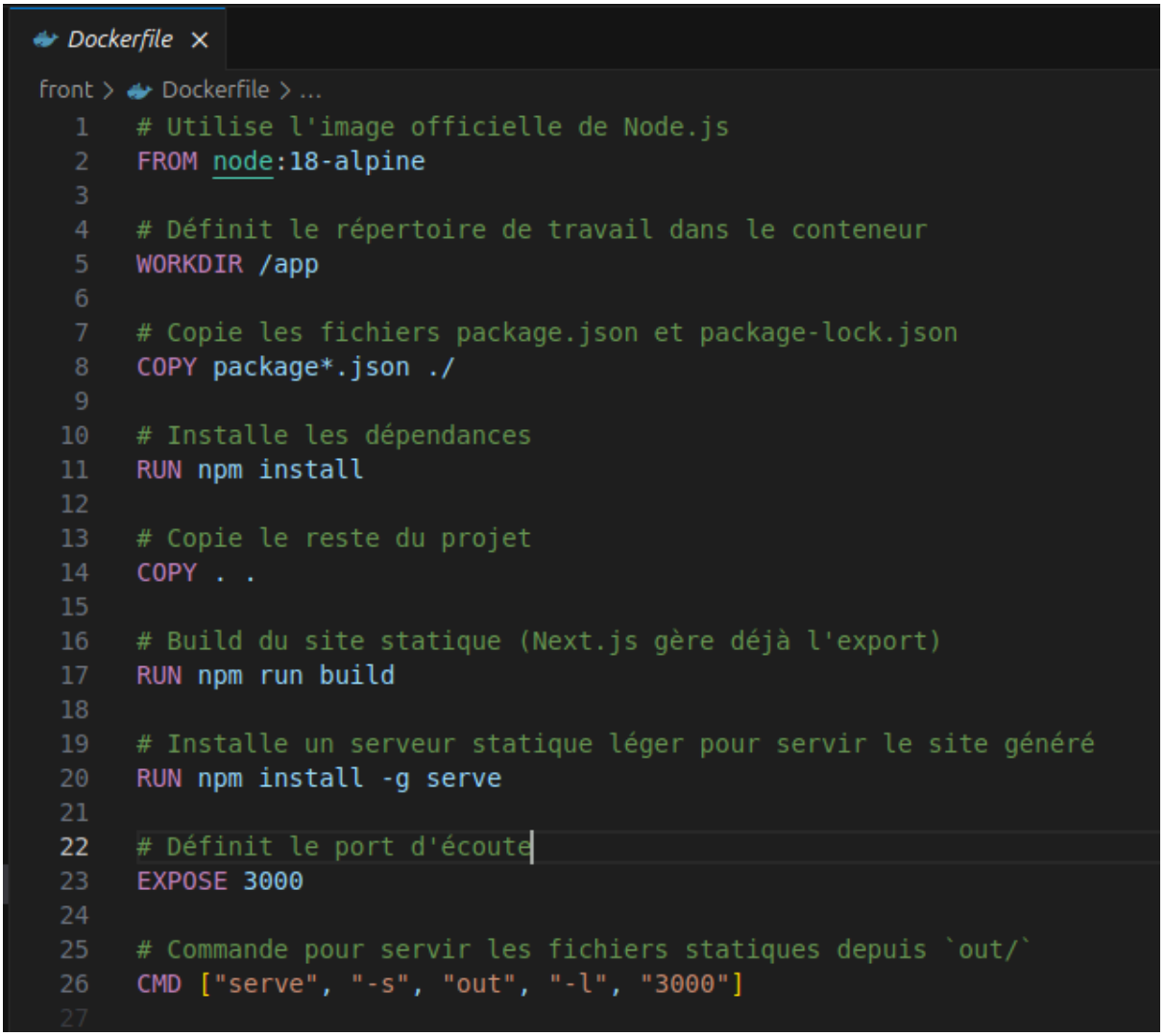
Backend (Node.js + Express)

Un Dockerfile a été créé pour le backend :



Frontend (Next.js)

Un autre Dockerfile a été créé pour le frontend :



2.2 Création du fichier `docker-compose.yml`

Le fichier `docker-compose.yml` permet d'orchestrer les trois services :

```yaml
version: '3.8'

services:
  db:
    image: postgres:16  # Assure-toi que la version est correcte
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data  # Volume pour stocker les données
      - ./data.sql:/docker-entrypoint-initdb.d/data.sql  # Charge `data.sql` au démarrage
    networks:
      - app-network

  backend:
    build: ./back
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      - DB_USER=postgres
      - DB_HOST=db
      - DB_DATABASE=mydatabase
      - DB_PASSWORD=password
      - DB_PORT=5432
    networks:
      - app-network

  frontend:
    build: ./front
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
      - app-network
```

Ce fichier permet de :

- Lancer une base de données PostgreSQL avec persistance des données.
- Lancer le backend qui se connecte à la base de données.
- Lancer le frontend qui communique avec le backend.

2.3 Exécution des conteneurs

Les conteneurs sont démarrés avec la commande : `docker-compose up -d`. Le flag `-d` permet de les exécuter en arrière-plan.

3. Gestion de la persistance des données

Le volume `pgdata` est utilisé pour assurer la persistance des données PostgreSQL :

```
volumes:
  pgdata:
```

Ainsi, même après l'arrêt ou la suppression des conteneurs, les données restent stockées et accessibles.

4. Déploiement des images sur Docker Hub

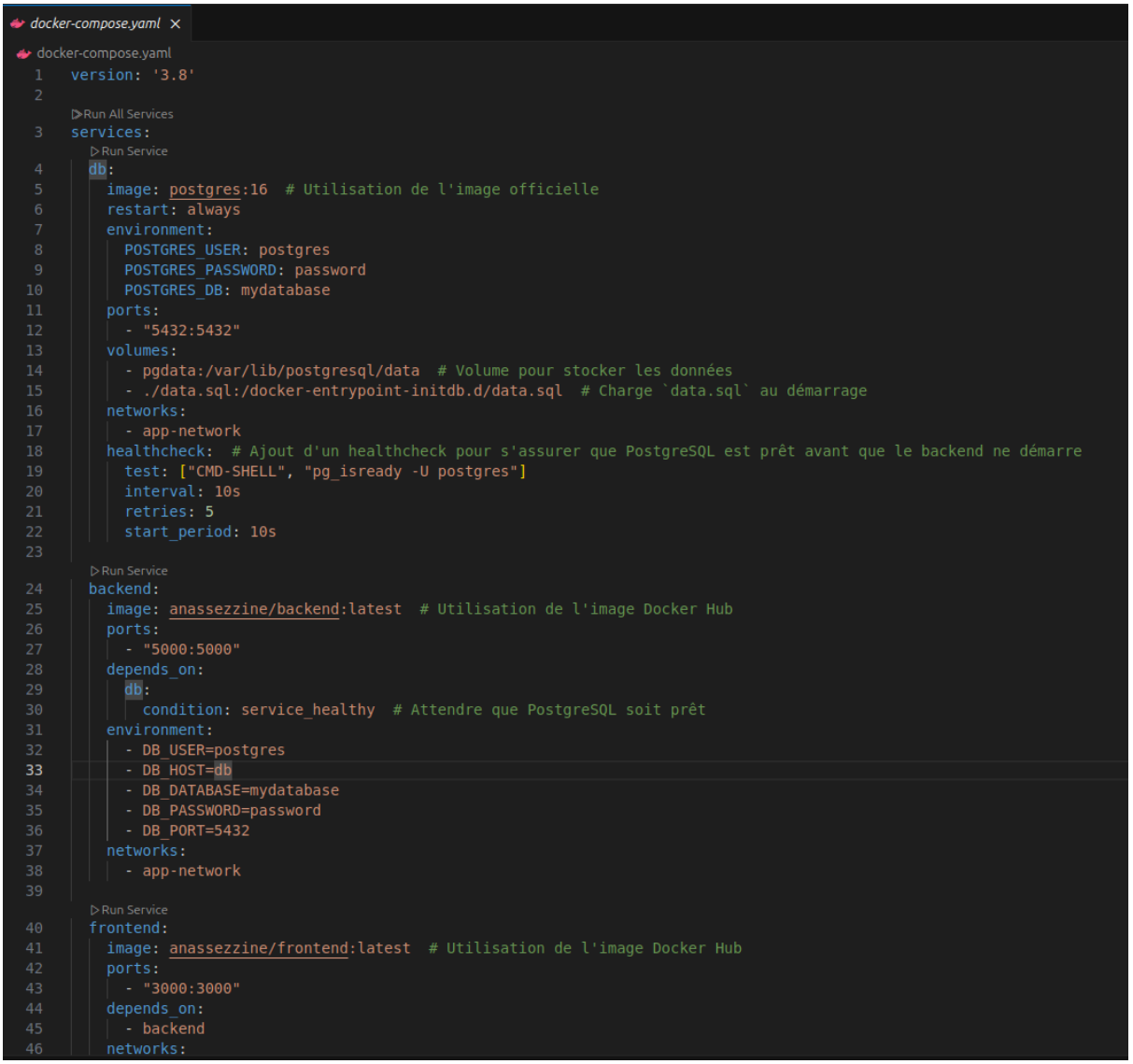
1. **Création d'un compte Docker Hub**

Création des images et push vers Docker Hub

```
docker build -t anassezzine/backend:latest ./back
docker build -t anassezzine/frontend:latest ./front

docker login
docker push anassezzine/backend:latest
docker push anassezzine/frontend:latest
```

2. **Modification du `docker-compose.yml` pour utiliser les images Docker Hub** :



```yaml
version: '3.8'

services:
  db:
    image: postgres:16  # Utilisation de l'image officielle
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data  # Volume pour stocker les données
      - ./data.sql:/docker-entrypoint-initdb.d/data.sql  # Charge `data.sql` au démarrage
    networks:
      - app-network
    healthcheck:  # Ajout d'un healthcheck pour s'assurer que PostgreSQL est prêt avant que le backend ne démarre
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      retries: 5
      start_period: 10s

  backend:
    image: anassezzine/backend:latest  # Utilisation de l'image Docker Hub
    ports:
      - "5000:5000"
    depends_on:
      db:
        condition: service_healthy  # Attendre que PostgreSQL soit prêt
    environment:
      - DB_USER=postgres
      - DB_HOST=db
      - DB_DATABASE=mydatabase
      - DB_PASSWORD=password
      - DB_PORT=5432
    networks:
      - app-network

  frontend:
    image: anassezzine/frontend:latest  # Utilisation de l'image Docker Hub
    ports:
      - "3000:3000"
    depends_on:
      - backend
    networks:
```

Cela permet de récupérer directement les images depuis Docker Hub sans avoir à les reconstruire à chaque fois. Ainsi, toute personne souhaitant utiliser l'application peut simplement récupérer le fichier `docker-compose.yml`, exécuter la commande : docker-compose up -d et le système se lancera automatiquement en récupérant les images depuis Docker Hub.

5. Test de l'Application

Étapes pour tester l'application en tant qu'utilisateur lambda

1. **Installer Docker et Docker Compose** (si ce n'est pas encore fait).
2. **Récupérer le fichier `docker-compose.yml`**.
3. **Ouvrir un terminal et exécuter la commande** : docker-compose up -d
4. Cela télécharge les images depuis Docker Hub et lance tous les services.
5. **Vérifier que les conteneurs sont bien lancés** avec : docker ps
6. **Accéder à l'application** :
    - Frontend : http://localhost:3000

    - Backend (API) : http://localhost:5000
7. **Tester l'interaction entre le frontend et le backend** en naviguant sur l'application.
8. **Arrêter les services** si nécessaire avec : docker-compose down