Rapport INFO0803

Abdoulaye Djibril Diallo
LECKOMBA-OWOCKA Jude Luther

Outils de programmation pour l'IA

https://github.com/djibrilIbnSaid/virtualisation-avec-docker

Introduction

Ce projet a pour but de mettre en place un environnement de virtualisation en utilisant Docker. Docker est un outil qui permet de créer, de déployer et de gérer des applications en utilisant des conteneurs. Un conteneur est une unité logicielle qui contient une application et toutes ses dépendances. Docker permet de créer des conteneurs légers et portables qui fonctionnent de la même manière sur n'importe quel environnement.

Ce rapport décrit la configuration et le fonctionnement d'un fichier `docker-compose.yml` utilisé pour orchestrer une application composée de trois services : une base de données MongoDB, une API backend expressjs, et un frontend Angular. Ce fichier définit comment chaque service est configuré, construit et interconnecté. De plus, il inclut les détails des Dockerfiles pour les services API et frontend.

Objectif

Les objectifs de ce projet sont les suivants :

- Créer un conteneur Docker à partir d'une image existante mongo pour la base de données.
- Créer un conteneur Docker à partir d'une image existante node pour le serveur (API). Pour cela, il faut créer un fichier Dockerfile contenant les instructions nécessaires pour la créeation de ce conteneur.
- Créer un conteneur Docker à partir des images existantent nginx et node pour le serveur web. Pour cela, il faut créer un fichier Dockerfile contenant les instructions nécessaires pour la créeation de ce conteneur.

Fonctionnement du fichier `docker-compose.yml`

Service db_api

Fonctionnalité

- **Base de données** : Ce service exécute une instance de MongoDB, qui est une base de données NoSQL utilisée pour stocker et gérer les données de l'application.

Configuration

- **Image** : Utilise l'image officielle `mongo` de Docker Hub, garantissant que la version standard de MongoDB soit utilisée.
- **Redémarrage automatique** : La directive `restart: always` assure que le conteneur redémarre automatiquement en cas de panne ou de redémarrage du système hôte.
- **Nom du conteneur** : Le conteneur est nommé `db_api` pour faciliter la référence dans d'autres parties de la configuration.
- **Ports** : Le port `27017` de MongoDB est mappé entre l'hôte et le conteneur, permettant l'accès à MongoDB depuis l'hôte ou d'autres conteneurs.
- **Volumes** : Un volume est monté de `./data` (répertoire de l'hôte) à `/data/db` (répertoire du conteneur) pour persister les données de MongoDB, même si le conteneur est recréé.

Service api

Fonctionnalité

- **API Backend** : Ce service exécute une API Node.js qui fournit des endpoints pour interagir avec la base de données MongoDB et traiter la logique métier.

Configuration

- **Construction** : Le conteneur est construit à partir du contexte `api` en utilisant le Dockerfile spécifié.
- **Nom du conteneur** : Le conteneur est nommé `api`.
- **Ports** : Le port `3000` est exposé à l'hôte, permettant l'accès à l'API.
- **Dépendances** : Utilise `depends_on` pour garantir que le service `db_api` soit démarré avant que l'API ne se lance.

Dockerfile

- **Base Image** : Utilise l'image `node:latest` pour fournir un environnement Node.js.
- **Configuration** : Définit le répertoire de travail à `/app`, copie les fichiers `package*.json` pour installer les dépendances, puis copie le reste des fichiers de l'application.
- **Exposition de port** : Le port `3000` est exposé pour l'API.
- **Commande de démarrage** : Utilise `CMD ["node", "app.js"]` pour démarrer l'application Node.js.

Service front

Fonctionnalité

- **Frontend** : Ce service exécute une application Angular qui sert l'interface utilisateur de l'application.

Configuration

- **Construction** : Le conteneur est construit à partir du contexte `front` en utilisant le Dockerfile spécifié.
- **Nom du conteneur** : Le conteneur est nommé `front`.
- **Ports** : Le port `4200` est exposé à l'hôte, permettant l'accès au frontend.
- **Volumes** : Le répertoire `./front` de l'hôte est monté dans le conteneur à `/usr/src/app` pour faciliter le développement en reflétant les modifications locales dans le conteneur.
- **Dépendances** : Utilise `depends_on` pour garantir que le service `api` soit démarré avant que le frontend ne se lance.

Dockerfile

- **Étape de construction** :
  - Utilise `node:latest` comme base pour installer les dépendances Angular et construire l'application.
  - Définit le répertoire de travail à `/app`, copie les fichiers `package*.json` pour installer les dépendances, puis installe

Angular CLI et construit l'application avec `npm run build --prod`.

- **Étape de production** :
  - Utilise `nginx:latest` pour servir l'application construite.
  - Copie la configuration Nginx et les fichiers construits de l'application Angular dans le répertoire de Nginx.

Interactions entre les services

1. **Démarrage** : Lorsque vous exécutez `docker-compose up`, Docker Compose démarre tous les services définis dans le fichier `docker-compose.yml`. Les services sont démarrés dans l'ordre des dépendances définies.
2. **Base de données (db_api)** : MongoDB démarre en premier, écoutant sur le port `27017`.
3. **API (api)** : Après que MongoDB est démarré, l'API démarre. Elle se connecte à MongoDB pour exécuter les opérations CRUD (Create, Read, Update, Delete).
4. **Frontend (front)** : Enfin, l'application Angular démarre. Elle se connecte à l'API pour récupérer et envoyer des données. Angular est servi par Nginx, écoutant sur le port `4200`.

Conclusion

Cette configuration Docker Compose permet de déployer facilement une application web complète avec une base de données MongoDB, une API Node.js, et un frontend Angular. Chaque service est isolé dans son propre conteneur, assurant modularité et scalabilité. Les dépendances entre les services sont gérées automatiquement, assurant un démarrage et une exécution coordonnés de l'application entière.