Introduction

Contexte et Objectifs du Projet

Ce projet a pour objectif de créer une application complète utilisant Docker Swarm pour déployer une stack front-end et back-end. L'application comprend un service front-end qui communique avec un service back-end. Ce dernier interagit avec une base de données MySQL et renvoie des réponses au front-end. Le tout est orchestré sur Docker Swarm pour une gestion optimale des services.

Stack Utilisée

- **Docker** : Pour la conteneurisation de l'application et la gestion des services via Docker Swarm.
- **Portainer** : Pour faciliter la gestion des services via une interface web.

- **Frontend** : Vue.js, utilisé pour la partie interface utilisateur.
- **Backend** : Node.js, avec Express, pour gérer les requêtes et interagir avec la base de données.
- **Base de données** : MySQL, pour stocker les données de l'application.
- **Client web pour la BDD** : PhpMyAdmin, pour une gestion facile des données MySQL.
- **Uptime Kuma** : Pour la surveillance des services, y compris le frontend, le backend, et la base de données.

Configuration Docker Swarm

Le projet utilise Docker Swarm pour déployer et orchestrer les services. Le fichier compose.yml a été configuré pour faciliter le déploiement de la stack complète, en définissant les services nécessaires et en les interconnectant à travers des réseaux Docker. De plus, les informations sensibles telles que les identifiants de connexion à la base de données sont stockées sous forme de **secrets Docker**, garantissant leur sécurité.

Configuration des Conteneurs

1. Dockerfiles
    - **Frontend** : Le Dockerfile pour le frontend utilise l'image node:18-alpine. Il installe les dépendances nécessaires, crée un utilisateur non-root (*totoUser*) pour des raisons de sécurité, et exécute un script pour générer un fichier settings.json avant de démarrer l'application en mode développement.
    - **Backend** : Le Dockerfile pour le backend utilise également l'image node:18-alpine, crée un utilisateur non-root (*totoUser*), et installe les dépendances avant de lancer le serveur Node.js (node index.js).
2. Healthcheck
    - **Frontend** : Un healthcheck est configuré pour vérifier que l'application frontend fonctionne correctement.
    - **Backend** : Un healthcheck est également mis en place pour garantir que le backend est prêt à recevoir des requêtes.

Lancement des Services avec Docker Swarm

Les services sont lancés en mode Docker Swarm, permettant une gestion efficace des services et leur déploiement sur plusieurs nœuds. Tous les services sont connectés à un réseau Docker Swarm avec des paramètres de sécurité appropriés (par exemple, utilisation des secrets pour la gestion de la base de données).

Gestion des Utilisateurs Non-root

Les conteneurs frontend et backend sont exécutés avec des utilisateurs non-root (totoUser), respectant ainsi les bonnes pratiques de sécurité recommandées pour les applications Docker.

Variables d'Environnement

Les services frontend et backend sont configurés via des **variables d'environnement** pour s'adapter à leurs besoins spécifiques :

- **Frontend** : L'URL de l'API est définie par une variable d'environnement, permettant une configuration dynamique lors du déploiement.
- **Backend** : Le backend utilise des variables d'environnement pour définir des paramètres tels que l'origine autorisée pour CORS et les informations de connexion à la base de données.

Bien que l'utilisation d'un fichier .env soit envisagée, elle n'est pas prise en charge de manière native par Docker Swarm. Cela m'a contraint à configurer manuellement ces variables d'environnement directement dans le fichier compose.yml, tout en permettant leur personnalisation via le terminal, comme décrit dans le fichier README.md.

Réseaux et Sécurité

1. Réseaux Docker
    - **Réseau Public** : Le frontend, le backend, Portainer et Uptime Kuma sont connectés à un réseau public, ce qui permet leur accès depuis l'extérieur du serveur.
    - **Réseau Privé** : Le backend et la base de données, ainsi que Uptime Kuma et PhpMyAdmin, sont isolés dans un réseau privé non exposé à l'extérieur, garantissant la sécurité des communications internes.
2. **Overlay Chiffré (Bonus)**

    Malheureusement, je n'ai pas pu implémenter un overlay chiffré sur Docker Swarm. Bien que cela soit une bonne pratique pour sécuriser les communications entre les nœuds Swarm, la documentation de Docker indique que cette fonctionnalité n'est pas disponible sous Windows (lien : Docker Overlay Encryption).

3. **Secrets Docker**

    Les informations sensibles, comme les identifiants de connexion à la base de données, sont stockées sous forme de **secrets Docker**. Cette approche garantit que

ces informations ne sont pas exposées dans les fichiers de configuration et restent sécurisées au sein de l'environnement Docker Swarm.

Déploiement et Outils

1. **Uptime Kuma et Monitoring**

   Un service **Uptime Kuma** a été déployé pour surveiller en temps réel la disponibilité des services frontend, backend, et base de données via des sondes HTTP. Ce service assure une surveillance constante pour vérifier la disponibilité de l'application.

2. **Portainer**

   Le service **Portainer** a été installé pour administrer facilement le cluster Docker Swarm via une interface web, permettant une gestion simplifiée des conteneurs et une visualisation des services.

3. **Client Web de Base de Données (PhpMyAdmin)**

   Un client web **PhpMyAdmin** a été configuré pour permettre la gestion facile des données de la base de données MySQL. Ce service est configuré pour démarrer en mode scaling (replicas: 0), afin de n'être activé que lorsqu'il est nécessaire.

Problèmes Rencontrés

1. **Support limité des fichiers .env sous Docker Swarm**

   Docker Swarm ne supporte pas directement les fichiers .env dans les configurations de services. J'ai dû contourner cela en définissant les variables d'environnement directement dans le fichier compose.yml. Toutefois, ces variables peuvent être modifiées via le terminal (voir le fichier README.md) pour répondre aux besoins de déploiement.

2. **Overlay chiffré sous Docker Swarm**

   En raison des limitations de Docker sur Windows, je n'ai pas pu implémenter un réseau **overlay chiffré**, une fonctionnalité pourtant utile pour sécuriser les communications entre les nœuds Swarm.

3. **Kubernetes**

   Bien que Kubernetes ait été mentionné comme un bonus, je n'ai pas eu le temps d'intégrer des manifestes Kubernetes. Le processus de transformation du fichier compose.yml en plusieurs fichiers .yml pour Kubernetes a été plus complexe que prévu, et je n'ai pas réussi à lancer Kubernetes avec succès.

4. **Utilisation des Secrets dans Docker Swarm**

   Initialement, j'avais des difficultés à faire fonctionner les secrets Docker, car mon environnement n'était pas configuré en mode Swarm. Une fois Swarm activé, j'ai oublié d'utiliser la méthode appropriée pour lire les secrets dans le backend et le frontend. Pour le backend, par exemple, j'ai utilisé la commande **fs.readFileSync("/run/secrets/mysql_user", "utf8").trim()** pour récupérer les informations sensibles.

5. **Problème de Communication Backend - Frontend**

   J'ai également rencontré un problème au début avec la communication entre le frontend et le backend. En effet, le backend n'avait pas de port exposé, ce qui empêchait le frontend de communiquer avec lui, malgré leur présence sur le même réseau Docker. Après avoir exposé le port du backend dans le fichier compose.yml, le problème a été résolu.

Conclusion

Le projet est pleinement fonctionnel avec une stack complète déployée sur Docker Swarm. Le frontend communique avec le backend, qui interagit avec la base de données et renvoie les réponses appropriées. Les services additionnels tels qu'Uptime Kuma pour la surveillance, Portainer pour la gestion du cluster, et PhpMyAdmin pour la gestion de la base de données ont été implémentés avec succès, de même pour les secrets, le réseaux privé ou public etc...

Cependant, bien que la majeure partie des fonctionnalités aient été intégrées avec succès, certains aspects n'ont pas pu être mis en œuvre :

- **Manifestes Kubernetes** : La configuration des manifestes Kubernetes n'a pas été réalisée en raison de la complexité de l'intégration et du manque de temps.
- **Chiffrement de l'Overlay** : L'overlay chiffré n'a pas pu être mis en place sous Docker Swarm sur Windows.
- **Fichier .env** : Docker Swarm ne prenant pas en charge les fichiers .env de manière native, cette fonctionnalité a dû être contournée par la configuration manuelle des variables d'environnement dans compose.yml.