Rapport de dockerisation de l'application Marvels

Objectifs

L'objectif de ce projet était de :

1. Intégrer l'API REST Marvels pour récupérer des données sur les personnages Marvel.
2. Créer une interface web dynamique affichant les personnages à l'aide de Fastify et Handlebars.
3. Dockeriser l'application pour faciliter son déploiement et rendre l'environnement de développement reproductible.

Étapes du Projet

Étape 1 – Accès à l'API Marvels

1. **Création du compte sur Marvel Developer** :
    - Nous avons commencé par créer un compte sur le site developer.marvels.com, où nous avons obtenu une clé d'API qui permet d'effectuer jusqu'à 3000 requêtes par jour.
    - Nous avons testé l'API via l'Interactive Documentation, en accédant à l'endpoint `/v1/public/characters` pour récupérer des informations sur les personnages.
    - Nous avons observé la structure de la réponse de l'API, qui inclut des informations comme le nom, la description, et les images miniatures des personnages.

Étape 2 – Authentification avec Postman

2. **Authentification API avec clé publique et privée** :
    - L'accès à l'API nécessite une authentification via un calcul de hachage utilisant la clé publique et privée, ainsi qu'un timestamp.
    - Nous avons configuré Postman pour effectuer ce calcul avant chaque requête en utilisant un script de pré-requête (pre-request script).
    - Nous avons créé une collection Postman, où chaque requête vers l'API Marvels utilise ce mécanisme d'authentification automatique.

Étape 3 – Récupération et Affichage des Données avec Node.js

3. **Développement du backend avec Node.js** :
    - Nous avons écrit le code de l'application en Node.js pour interagir avec l'API Marvels en utilisant la méthode `fetch()` du module `node-fetch`.

- La fonction `getHash()` a été utilisée pour générer le hachage nécessaire à l'authentification.
- Nous avons filtré les résultats pour n'inclure que les personnages avec une image valide (et non `image_not_available`), puis constitué un tableau de personnages avec leurs images et descriptions.

Étape 4 – Affichage des Données avec Fastify et Handlebars

4. **Création du frontend avec Fastify et Handlebars** :
   - Nous avons utilisé Fastify, un framework rapide pour Node.js, pour créer un serveur web.
   - Le moteur de template Handlebars a été intégré pour afficher dynamiquement les informations des personnages Marvel sur la page web.
   - Des fichiers `header.hbs` et `footer.hbs` ont été utilisés comme partials pour inclure un layout Bootstrap cohérent sur toutes les pages.
   - Nous avons configuré l'affichage des personnages sous forme de cartes Bootstrap ou de listes, avec leurs images et descriptions.

Étape 5 – Dockerisation de l'Application

5. **Création du Dockerfile** :
   - Nous avons créé un fichier `Dockerfile` pour conteneuriser l'application. L'image Docker utilise la version `node:lts-bullseye-slim` comme base.
   - L'application est installée dans un répertoire `/home/node/app`, et les dépendances sont installées à l'aide de `npm install`.
   - Nous avons défini le fichier Docker pour démarrer l'application avec la commande `CMD ["node", "server.js"]`.
6. **Exclusion des fichiers inutiles avec .dockerignore** :
   - Nous avons ajouté un fichier `.dockerignore` pour exclure les répertoires `node_modules` et les fichiers `npm-debug.log`, afin de réduire la taille de l'image Docker.
7. **Sécurisation des Identifiants avec .env** :
   - Afin de ne pas exposer nos identifiants API dans le code source, nous avons ajouté un fichier `.env` contenant les clés d'API et d'autres variables sensibles.
   - Nous avons utilisé le module `dotenv` pour charger ces variables d'environnement au moment de l'exécution de l'application.
8. **Construction et Test du Conteneur Docker** :
   - Nous avons construit l'image Docker avec la commande `docker build . -t nomDeLImage`.
   - L'application a été lancée avec `docker run`, et le comportement du conteneur a été testé pour vérifier que l'application fonctionnait correctement.

Conclusion

La dockerisation de l’application Marvels a permis de simplifier son déploiement sur différentes machines sans avoir à configurer à nouveau l’environnement Node.js. En plus de sécuriser les clés API et de rationaliser les dépendances via un fichier `.env`, la solution Docker garantit que l’application peut être exécutée de manière uniforme sur toutes les plateformes. Cette approche améliore également la portabilité du projet et facilite sa mise en production