Rapport devoir Docker Victor Bechet

ARCHITECTURE DU PROJET

Base de données :

Pour la base de données, j'ai utilisé une image postgres (postgres:16-alpine). Celle-ci va être remplie par le script init.sql (ajout des items).

J'ai attaché un volume à la base de données pour assurer la persistance des données par la suite, même si le script les initialise au démarrage pour le moment.

Backend /API :

J'ai fait le backend en node.js avec Express puisque j'ai l'habitude de faire des serveurs de cette manière. J'ai exposé deux routes /status et /items qui vont être par la suite appelées par le front.

La connexion à la base de données se fait par le biais de pg-promise.

Pour le back j'ai choisi d'exposer le port 4000.

Frontend :

Pour le frontend j'ai utilisé un Nginx et j'ai fait le code en HTML et javascript très basique.

Il permet d'appeler le backend avec un reverse proxy configuré dans nginx.

Le port exposé est le 8080.

COMMANDES CLÉS UTILISÉES

Commandes pour la construction des images :

docker build -t app-api:latest ./backend -> Pour build le backend

docker build -t app-frontend:latest ./frontend -> Pour build le frontend

docker compose build -> Pour déclencher le docker compose qui attache les volumes et start la base de données

Scripts d'automatisations :

deploy.sh : Déploiement complet

security-scan.sh : Scan de sécurité

sign-images.sh : Signature des images

J'ai rajouté un makefile qui me permet de faire "make deploy" rapidement.

BONNES PRATIQUES

1. Utilisation d'images docker légères grâce aux images alpines.

Je les ai utilisés partout comme vu en cours, pour node, nginx et postgres.

2. Build multi-étapes

J'ai utilisé le build multi étape en séparant bien les fichier dockerfile ainsi que le docker compose qui me permet de lancer mes conteneurs les uns après les autres.

3. Dockerignore

J'ai mis mon dossier node_modules dans le dockerignore comme on le ferait pour un gitignore. On pourrait y mettre les variables d'environnement via le .env si on souhaitait les cacher, ainsi que les fichiers de configuration de l'IDE.

4. Variables d'environnement

L'utilisation des variables d'environnement consignés dans le .env est primordiale.

Dans le repository se trouve un .env.example qui indique comment remplir correctement le .env avec les bons champs.

Rien n'est en dur dans le code, on a uniquement des variables qui gère les authentifications.

5. Healthchecks

L'ajout d'un healthcheck entre le démarrage des services permet d'être sûr que le déploiement s'est bien passé et que les différents services enfants s'appuient bien sur des services parents healthy.

SÉCURITÉ

J'ai créé puis utiliser des utilisateurs non-root (nodejs / nginx-user). Ceux-ci exécutent les processus avec les bons droits et permettent de séparer mon user de celui utilisé

uniquement pour lancer le service. Il assure en plus de ça une sécurité, puisqu'il ne dispose pas de tous les droits.

J'ai aussi utilisé le scan des images via Docker Scout. Cela me génère des rapports de sécurité avec les failles de mes images. Les rapports sont disponibles sous le dossier /scripts/security-reports et régénérables avec le script associé 'security-scan'.

J'ai essayé d'utiliser Docker Content Trust pour signer mes images sur le Docker Hub mais je n'ai pas réussi puisque j'ai rencontré un bug insolvable.

DIFFICULTÉS RENCONTRÉES

Durant le processus de création de ce projet j'ai trouvé assez facile la base, en répliquant ce que nous avions vu en cours. Mais au contraire, j'ai trouvé très dur l'intégration des différents points demandés.

Les différents services de scan de sécurité, signature des images, utilisation du Healthcheck, création d'users limitées était des points tout nouveaux pour moi et j'ai appris beaucoup de chose en les explorant, bien que je n'aie fais que suivre la documentation et m'aider de l'IA pour savoir à quoi servaient certaines commandes ou lignes bien spécifiques.

Je n'ai pas réussi à signer mes images via Docker Content Trust à cause d'un bug que je n'ai pas pu résoudre. J'ai quand même push mes images sur Docker Hub non signées.

J'ai bien compris que le but de cet exercice était de nous faire explorer les bonnes pratiques et automatisations du code et de docker, et j'ai eu personnellement du mal (mais du plaisir) à sortir la tête du code pour aller voir des concepts tel que la signature des images docker, ou encore le reverse proxy via nginx en frontend.

CONCLUSION

Ce projet crée assemble et monte 3 services qui communiquent entre eux via l'environnement Docker pour former un ensemble BDD – API – FRONTEND.

Les services sont basés sur des images légères alpine, le tout orchestré par un docker-compose qui attache les volumes aux images crées.

Des scripts simples permettent de rassembler les commandes de déploiement pour fluidifier la mise en route des différents services. Le HealthCheck mit en place nous assure que les services soient correctement créés.

Les bonnes pratiques concernant la sécurité et l'analyse des images se fait via des services Docker tel que Docker Scout ou Docker Trust Sign. En ce qui concerne l'utilisation d'user non-root, on a un user par service qui possède des droits limités.

J’ai pu explorer des technologies principalement liées à Docker, mais aussi des bonnes pratiques que je ne connaissais pas. J’ai pu aussi apprendre comment faire des scripts et me familiariser avec la mise en place de Healthcheck.

J’ai mieux compris les principes de dockerfile et docker-compose qui sont essentiel à l’environnement moderne de Docker. J’ai pu aussi mieux saisir l’utilité de séparer des services via Docker et plus largement via les Machines virtuelles.