- Rapport Conteneurisation :

- ***Nom et Prénom :*** **BOUKHRIS** Mohamed Lyazid
- ***Encadré par :*** **QUAFAFOU** Mohamed

TABLE DES MATIERES

1. INTRODUCTION :

Ce rapport présente le processus complet de conteneurisation et de déploiement d'une application Web à l'aide de Docker et Docker Compose. L'application est composée de trois éléments principaux :

- Un back-end développé en Python avec le framework Flask, chargé de gérer la logique métier et les interactions avec la base de données ;
- Un front-end statique réalisé en HTML, CSS et JavaScript, qui sert d'interface utilisateur
- Une base de données MySQL utilisée pour stocker les informations des utilisateurs.

L'objectif de cette conteneurisation est de rendre l'application portable, modulaire et facilement déployable sur n'importe quelle machine, sans avoir à reconfigurer chaque environnement. Grâce à Docker, chaque composant est isolé dans un conteneur indépendant, ce qui facilite le déploiement, la maintenance et la montée en charge de l'application.

2. INSTALLATION DE DOCKER

L'environnement utilisé pour ce projet est un système Windows, avec l'installation de Docker Desktop, un outil qui permet d'exécuter des conteneurs localement. L'installation ne nécessitait ni machine virtuelle personnalisée, ni activation de WSL, ce qui a permis une mise en place rapide.

Pour la gestion des conteneurs, seule l'interface en ligne de commande (CLI) a été utilisée, notamment pour exécuter des commandes telles que docker build, docker run, docker-compose up.

L'installation a été vérifiée avec les commandes suivantes :

docker –version docker info

3. ARCHITECTURE DE L'APPLICATION

L'application est composée de trois services principaux, chacun étant isolé dans un conteneur Docker distinct afin d'assurer une architecture modulaire et facilement déployable :

- ***Back-end*** (Métier) : une API développée en Python avec le framework Flask. Elle est responsable de la logique métier de l'application, notamment la vérification des utilisateurs et l'accès aux données stockées en base de données.
- ***Front-end*** (Client) : une interface utilisateur statique développée en HTML, CSS et JavaScript, servie via un serveur web Nginx. Elle permet aux utilisateurs d'interagir avec l'application via leur navigateur.
- ***Base de données*** (Data) : un serveur MySQL qui stocke les noms des utilisateurs. C'est ce service que le back-end interroge pour vérifier si un utilisateur est autorisé ou non.

L'ensemble de ces services est déployé dans un réseau Docker dédié, ce qui permet une communication interne fluide entre les conteneurs sans exposition inutile à l'extérieur.

4. CREATION DES CONTENEURS

4.1 CONTENEURISATION DU BACKEND

Le back-end de l'application a été développé en Python à l'aide du micro-framework Flask. Afin de le rendre portable et exécutable dans un environnement isolé, un conteneur Docker a été mis en place à l'aide d'un fichier Dockerfile personnalisé.

Voici les principales étapes réalisées dans ce Dockerfile :

- Utilisation de l'image officielle Python 3.11 comme base.
- Définition du répertoire de travail /app dans lequel le code de l'application sera copié et exécuté.
- Installation des dépendances spécifiées dans le fichier requirements.txt (flask, flask-cors, mysql-connector-python).
- Copie de l'ensemble du code source de l'application dans le conteneur.
- Exposition du port 5000, utilisé par défaut par le serveur Flask.
- Désactivation du cache Python avec PYTHONUNBUFFERED=1 pour faciliter le débogage.
- Lancement de l'application avec la commande python connexionBD.py qui contient le point d'entrée principal du back-end.

Ce conteneur a ensuite été testé localement à l'aide des commandes suivantes :

docker build -t flask-backend .

docker run -p 5000:5000 --env-file .env flask-backend

- ***Test*** : L'API a pu être testée avec succès depuis un navigateur ou des outils comme Postman via l'URL http://localhost:5000

4.2 CONTENEURISATION DU FRONTEND

Le front-end de l'application est une interface statique développée en HTML CSS et JavaScript. Afin de la rendre accessible via un navigateur elle a été servie à l'aide d'un serveur web Nginx exécuté dans un conteneur Docker.

Un Dockerfile a été rédigé pour préparer l'image personnalisée du front-end. Celui-ci exécute les étapes suivantes :

- Utilisation de l'image officielle nginx:alpine légère et optimisée pour le déploiement de contenu statique.
- Suppression de la configuration par défaut de Nginx, pour ne garder que les fichiers personnalisés de l'application.
- Copie des fichiers statiques (HTML, CSS, JS) dans le répertoire /usr/share/nginx/html qui est le dossier par défaut utilisé par Nginx pour servir du contenu.
- Exposition du port 80 pour permettre l'accès à l'interface depuis un navigateur ;
- Lancement du serveur Nginx via la commande CMD.

L'image du front-end a été construite à l'aide de la commande suivante :

docker build -t frontend-nginx .

Test : Une fois lancée l'interface était disponible à l'adresse : http://localhost:80

4.3 CONTENEURISATION DE LA BASE DE DONNEES MYSQL

La base de données utilisée par l'application est un serveur MySQL chargé de stocker les noms des utilisateurs. Contrairement aux services front-end et back-end ce composant n'a pas nécessité de Dockerfile personnalisé. Il a été directement

configuré et lancé via le fichier docker-compose.yml à l'aide de l'image officielle mysql :8.0.

5. DEPLOIEMENT AVEC DOCKER COMPOSE

Afin de faciliter l'orchestration des différents services de l'application un fichier docker-compose.yml a été mis en place. Celui-ci permet de définir construire configurer et exécuter les conteneurs correspondants au front-end au back-end et à la base de données.

Le fichier comporte les éléments suivants :

- Trois services :
  - frontend : construit à partir du dossier ./frontend exposé sur le port 8080 (redirigé vers le port 80 de Nginx).
  - backend : construit à partir du dossier ./backend exposé sur le port 5000
  - db : basé sur l'image officielle mysql:8.0 contenant la base de données.

- La directive depends_on garantit que le service backend ne démarre qu'après le service db et que le frontend attend le backend.

- La configuration des variables d'environnement nécessaires à la connexion à la base de données est incluse directement dans la définition du backend.

- La persistance des données MySQL est assurée par un volume nommé mysql_data monté sur le répertoire /var/lib/mysql à l'intérieur du conteneur.

- Un script SQL d'initialisation init.sql est automatiquement exécuté à la création du conteneur MySQL grâce au montage dans docker-entrypoint-initdb.d

- Un réseau Docker personnalisé (mynetwork) permet à tous les services de communiquer entre eux de manière isolée et sécurisée.

Le déploiement complet de l'application s'effectue ensuite en une seule commande :

```
docker-compose up --build
```

Cette commande télécharge les images si elles n'existent pas, construit les images locales, crée les volumes, le réseau, et démarre tous les conteneurs.

6. PUBLICATION ET DEPLOIEMENT SUR UNE AUTRE MACHINE

6.1. PUBLICATION DES IMAGES SUR DOCKER HUB

Pour permettre le déploiement de l'application sur une autre machine sans avoir à reconstruire les images localement les images du frontend et du backend ont été publiées sur Docker Hub un registre public d'images Docker.

Voici les étapes suivies pour publier chaque image :

1. docker login : Connexion à Docker Hub
2. Taguer les images localement avec le nom du dépôt distant :

   docker tag frontend-nginx nom_user/frontend-nginx:latest

   docker tag flask-backend nom_user/flask-backend:latest
3. Pousser les images vers Docker Hub :

   docker push nom_user/frontend-nginx:latest

   docker push nom_user/flask-backend:latest

Une fois publiées, ces images peuvent être téléchargées depuis n'importe quelle machine connectée à Docker Hub, en modifiant le docker-compose.yml pour remplacer build: par image:.

6.2 DEPLOIEMENT SUR UNE AUTRE MACHINE

Une fois les images Docker du front-end et du back-end publiées sur Docker Hub, le déploiement de l'application sur une nouvelle machine devient simple et rapide.

Voici les étapes réalisées sur l'autre ordinateur :

1. Se connecter à Docker Hub : docker login
2. Copier le fichier docker-compose.yml sur la nouvelle machine et le fichier init.sql pour initialiser la base de données.
3. Modifier docker-compose.yml pour utiliser les images publiées, en remplacent les blocs build: par image:

4. Lancer le déploiement de l'application : docker-compose up -d

- ***Test :*** L'application est alors accessible localement sur la nouvelle machine via les ports exposés :

  http://localhost:8080 pour le front-end

  http://localhost:5000 pour l'API Flask

7. GESTION DE LA PERSISTANCE DES DONNEES

Dans le cadre de la conteneurisation de la base de données MySQL, il était essentiel de s'assurer que les données ne soient pas perdues lors de l'arrêt ou du redémarrage du conteneur. Pour cela un volume Docker nommé mysql_data a été utilisé.

Ce volume est défini dans le fichier docker-compose.yml et monté dans le répertoire interne /var/lib/mysql du conteneur MySQL qui correspond à l'endroit où MySQL stocke physiquement ses fichiers de données.

```
volumes:
  - mysql_data:/var/lib/mysql
```

Grâce à cette configuration, les données sont conservées même si le conteneur est supprimé ou redémarré. Ce mécanisme de persistance est essentiel pour garantir l'intégrité et la continuité des données utilisateur sans avoir à les réinsérer après chaque déploiement.

8. CONCLUSION

L'application a été entièrement conteneurisée à l'aide de Docker et orchestrée efficacement avec Docker Compose. Chaque composant le front-end, le back-end, et la base de données fonctionne de manière isolée dans un conteneur dédié tout en étant interconnecté dans un réseau Docker privé.

Grâce à cette architecture, l'application est désormais portable, reproductible et facilement déployable sur n'importe quelle machine en une seule commande. La persistance des données MySQL a également été assurée à l'aide de volumes Docker garantissant ainsi la conservation des informations utilisateur.