Exercice 2

L'objectif de cet exercice est de créer une stack de développement pour le service dev de votre entreprise.

Il s'agit d'une petite application permettant d'afficher et d'analyser les livres disponibles dans un catalogue. L'application doit exposer 3 endpoints principaux :

— La liste complète des livres disponibles
— Le chiffre d'affaires par éditeur
— Les variations de ventes entre 2015 et 2016

Le développeur en charge vous confirme que " ça marche sur sa machine"

Vous disposez de :

— Un répertoire api/ contenant le code source de l'API Flask
— Un répertoire sql/ contenant le script pour créer une base de données MySQL pré-remplie avec des données de livres

Votre mission Vous devez réaliser un fichier docker-compose.yml qui permette de créer et de lancer les conteneurs suivants :

— Un conteneur MySQL contenant la base de données des livres
— Un conteneur Flask hébergeant l'API qui expose les endpoints mentionnés ci-dessus.

Attendus

— Vous devez créer le Dockerfile nécessaire à la création de l'image de vote application.
— Le docker compose doit construire l'images à partir du Dockerfile

Les aides, en Annexe, vous aideront à dockeriser cette application !

Pensez à bien les lire, en complément de la documentation des images !

API

— Le fichier **requirements.txt** contient les dépendances python nécessaires à l'API
— Il devra être copié dans le conteneur et installé avec la commande **pip install -r requirements.txt**
— L'API Flask écoute sur le port 8000 dans le conteneur. Elle est accessible sur le port 5000 depuis la machine hôte.
— Vous devez utiliser l'image **python :3.11-slim** comme image de base

BDD

— Vous pouvez utiliser l'image **mysql :8.0** disponible sur docker hub.
— Vous devez copier le fichier **book.sql** dans l'entrypoint du conteneur MySQL
— Le nom de la base de données doit être **books**
— Le mot de passe root doit être **root**

Avant de vous diriger vers les indices, essayez de réaliser l'exercice sans ;).

Annexe

Les responsabilités des différents fichiers :

— Le **Dockefile** permet de construire une images. Il copie les contenus et installe les dépendances.

— Le **compose.yml** permet de créer les conteneurs à partir des images construites précédemment. Il définit les ports à exposer et les variables d'environnement, ainsi que les dépendances entre les conteneurs.

Dockerfile

Vous devrez constituer un **Dockerfile**, pour le service (API). Vous placerez ce fichier dans le répertoires **api**.

— Vous déclarerez l'image de base à utiliser pour construire l'image Docker, par exemple : **python :3.11-slim**.

— Vous définirez le répertoire de travail dans le conteneur à l'aide de l'instruction `WORKDIR` (par exemple `/srv`).

— Vous installerez les dépendances système nécessaires à l'utilisation de `mysqlclient`. Pour cela, vous utiliserez une commande similaire à :

```
RUN apt-get update && apt-get install -y --no-install-recommends \
      build-essential \
      default-libmysqlclient-dev \
      pkg-config \
    && rm -rf /var/lib/apt/lists/*
```

— Vous copierez le fichier `requirements.txt` dans l'image, puis installerez les dépendances Python avec `pip install -r requirements.txt`.

— Vous copierez le code source de l'application (le dossier `app/`) dans l'image (dans le répertoire du travail).

— Vous définirez les variables d'environnement nécessaires pour exécuter Flask (`FLASK_APP`, `FLASK_RUN_HOST`, `FLASK_RUN_PORT`).

— Vous exposerez le port `8000` à l'aide de l'instruction `EXPOSE`.

— Enfin, vous démarrerez le serveur Flask en utilisant la commande :
`flask run` (NB : La commande CMD utilise un tableau !)

compse.yml

— Vous créerez un fichier `compose.yml` à la racine du projet afin d'orchestrer l'exécution de plusieurs conteneurs.

— Vous définirez deux services principaux :
  — **db** : un conteneur MySQL qui héberge la base de données.
  — **api** : un conteneur Flask qui exécute l'API Python.

— Pour le service **db** :
  — Vous utiliserez l'image officielle **mysql :8.0**.
  — Vous définirez les variables d'environnement nécessaires :
    — `MYSQL_ROOT_PASSWORD` : mot de passe root.
    — `MYSQL_DATABASE` : nom de la base à créer (`books`).
  — Vous monterez un volume afin d'exécuter automatiquement le script d'initialisation SQL présent dans le répertoire `sql/` au démarrage du conteneur MySQL. *Indice : MySQL exécute automatiquement tous les fichiers `.sql` placés dans le*

répertoire spécial `/docker-entrypoint-initdb.d/` à la première création de la base de données.

- Vous mettrez en place un `healthcheck` pour attendre que MySQL soit prêt avant de lancer l'API :

```
healthcheck:
  test: ["CMD-SHELL", "mysqladmin ping -h 127.0.0.1 -proot || exit 1"]
  interval: 5s
  timeout: 3s
  retries: 20
```

Explication du bloc :

- `test` : définit la commande exécutée régulièrement pour vérifier l'état du conteneur. Ici, la commande utilise `mysqladmin ping` pour interroger le serveur MySQL et s'assurer qu'il répond.
- `interval` : spécifie la fréquence d'exécution de la commande de test (par exemple toutes les `5s`).
- `timeout` : indique le temps maximum d'attente pour une réponse avant de considérer le test comme échoué.
- `retries` : précise le nombre de tentatives consécutives échouées avant de marquer le conteneur comme *unhealthy*.

— Pour le service **api** :

- Vous utiliserez l'instruction `build` pour construire l'image à partir du répertoire `api/`.
- Vous définirez les variables d'environnement nécessaires à la connexion MySQL :

```
environment:
  DB_HOST: db
  DB_USER: root
  DB_PASSWORD: root
  DB_DB: books
```

- Vous exposerez le port externe `5000` de la machine hôte vers le port interne `8000` utilisé par Flask :
- Vous ajouterez une dépendance pour que le conteneur `api` ne démarre qu'une fois MySQL prêt :

```
depends_on:
  db:
    condition: service_healthy
```

— Enfin, vous lancerez les conteneurs avec la commande :

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

— Vous pourrez ensuite accéder à l'API via l'URL : `http://localhost:5000`