TP 2 : Docker et Docker Compose

Prérequis

Installer Docker sur la machine.

- Sur Ubuntu

```
sudo apt install -y docker.io
sudo apt install -y docker-compose-v2
```

- Sur MacOS, avec Brew :

```
brew install docker
```

- Sur Windows ? Utilisez une VM...

Télécharger Docker Desktop (`https://www.docker.com/products/docker-desktop/`)

1 Docker in a nutshell

1.1 Premier conteneur

Après avoir démarré Docker Desktop, ouvrez un terminal pour exécuter les commandes du TP.

- Lancez votre premier conteneur avec l'image officielle `hello-world` :

  `docker run hello-world`

  Observez le message affiché et expliquez ce qui se passe.

- Listez les conteneurs existants avec `docker ps -a`.

2 Explorer l'image Alpine

1) Téléchargez l'image Alpine :

```
docker pull alpine
```

2) Lancez un conteneur interactif basé sur Alpine :

```
docker run -it alpine /bin/sh
```

3) Explorez le système de fichiers (commande `ls`, `cat`, etc.).

4) Essayez de trouver un noyau Linux dans le conteneur. Que remarquez-vous ?

Note : vous pouvez quitter l'environnement interactif d'Alpine en utilisant `exit`.

2.1 Manipulation d'images

1) Listez les images disponibles localement avec `docker images`

2) Téléchargez l'image `busybox` avec la commande `docker pull busybox`

3) Inspectez cette nouvelle image avec `docker image inspect busybox`.

4) Supprimez une image inutilisée avec `docker image rm busybox`

2.2 Construire une image avec un Dockerfile

1) Créez un fichier `Dockerfile` contenant les instructions suivantes :

```
FROM alpine:latest
WORKDIR /app
COPY hello.sh .
RUN chmod +x hello.sh
CMD ["./hello.sh"]
```

2) Créez un script `hello.sh` qui affiche `"Hello Docker!"`.

3) Construisez l'image :

```
docker build -t monapp:latest .
```

4) Lancez l'image et vérifiez le résultat.

2.3 Bonnes pratiques

1) Modifiez votre Dockerfile pour installer `curl`, puis supprimez les fichiers temporaires afin de réduire la taille de l'image.

2) Comparez la taille de l'image avant et après l'optimisation.

2.4 Déploiement avec Docker Compose

1) Créez un fichier `compose.yaml` décrivant deux services :

- `web` basé sur l'image `nginx:latest`, exposant le port 8080.
- `db` basé sur l'image `mariadb:latest`, avec un mot de passe administrateur défini par une variable d'environnement.

2) Lancez la pile avec :

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

3) Vérifiez que le serveur Nginx est accessible sur `http://localhost:8080`.

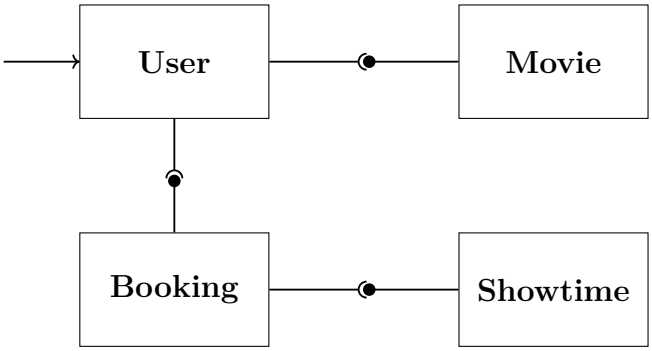
4) Arrêtez et supprimez les conteneurs avec :

```
docker-compose down
```

3 Deployer une stack logicielle avec Docker

Pour cet exercice, nous allons déployer une application écrite en Python. Il s'agit d'une application jouet et peu réaliste pour gérer les films et les réservations d'utilisateurs dans un cinéma. Le code source de l'application est disponible en ligne. Cette application est composée de 4 services :

- **Movie** est le service responsable de la gestion des films du cinéma. Il contient et gère une petite base de données json contenant la liste des films disponibles avec quelques informations sur les films.
- **Showtime** est le service responsable des jours de passage des films dans le cinéma. Il contient et gère une petite base de données json contenant la liste des dates avec l'ensemble des films disponibles à cette date.
- **Booking** est le service responsable de la réservation des films par les utilisateurs. Il contient et gère une petite base de données json contenant une entrée par utilisateur avec la liste des dates et films réservés. `Booking` fait appel à `Showtime` pour connaître et vérifier que les créneaux réservés existent bien puisqu'il ne connaît pas lui-même les créneaux des films.
- **User** est le service qui sert de point d'entrée à tout utilisateur et qui permet ensuite de récupérer des informations sur les films, sur les créneaux disponibles et de réserver. Il contient et gère une petite base de données json avec la liste des utilisateurs. `User` fait appel à `Booking` et `Movie` pour respectivement permettre aux utilisateurs de réserver un film ou d'obtenir des informations sur les films.



3.1 Démarrer à la main les services

Chaque service est codé en Python et possède son propre dossier (`booking`, `movie`, `showtime`, `user`). À la racine de chaque dossier, vous trouverez un fichier `requirements.txt` listant les dépendances nécessaires.

1) Créez un environnement virtuel dans chaque dossier :

```
python3 -m venv .venv
```

2) Activez l'environnement virtuel :

```
source .venv/bin/activate
```

3) Installez les dépendances nécessaires :

```
pip install -r requirements.txt
```

4) Lancez chaque service (par exemple pour `user`) :

```
python user.py
```

Une fois les quatre services démarrés, ouvrez votre navigateur et accédez à l'URL suivante : `http://localhost:3203/`
Essayez les routes suivantes (via le navigateur ou un client HTTP) :

- Accès aux utilisateurs : `http://localhost:3203/users`
- Accès à un utilisateur particulier : `http://localhost:3203/users/chris_rivers`
- Accès aux films : `http://localhost:3203/movies`
- Accès aux séances : `http://localhost:3203/showtime`

Regardez le code source des services pour comprendre comment ils communiquent entre eux.

3.2 Utiliser Docker et Docker Compose

Chaque service doit maintenant être exécuté dans un conteneur Docker.

1. Écrire un `Dockerfile`, à partir de l'image `python:3.8-alpine`, pour chaque service.

2. Écrire un fichier `docker-compose.yml` pour lancer les services.

```
version: "3.9"
services:
  movie:
    build: ./movie/
    ports:
      - "3200:3200"
  showtime:
    build: ./showtime/
    ports:
      - "3202:3202"
```

```
booking:
  build: ./booking/
  ports:
    - "3201:3201"
user:
  build: ./user/
  ports:
    - "3203:3203"
```

3. Adapter le code.

- Dans le code du service `user`, remplacez les appels locaux (`http://localhost:PORT/...`) par les noms des services définis dans `docker-compose.yml` (`http://booking:3201/...`, `http://movie:3200/...`, etc.).
- Cela permet à Docker d'utiliser son DNS interne pour la communication entre conteneurs.

4. Lancer l'ensemble de la stack

3.3 Exercice subsidiaire

En étudiant le code Python des services et les spécifications REST, ajoutez de nouveaux points d'entrée au service `user` :

- Créer une réservation pour un utilisateur (appel au service `booking`).
- Consulter les séances réservées par un utilisateur.
- (Optionnel) Ajouter un endpoint pour annuler une réservation.

Pour chaque nouveau point d'entrée, décrivez :

- La requête HTTP d'entrée (méthode, chemin, corps éventuel).
- Le format de la réponse attendue.
- Les cas de test possibles (exemple : utilisateur inexistant, réservation sur un film non disponible, etc.).