COMMENT UTILISER DOCKER COMPOSE

Exercice 22 – Intro à docker compose

Jusqu’à présent, nous avons utilisé des commandes pour lancer des applications mono-conteneurs, soit avec un seul script Python, un seul script PHP, ou un binaire Rust.

Hors, ce n’est pas la pratique courante, souvent votre application (web ou autre) aura plusieurs conteneurs (un pour le back, un pour la BDD, un pour le front, etc.). Mais alors, on ne va construire les images une par une et lancer les conteneurs. On va tout orchestrer grâce à **Docker compose** !

Exercice 22 – Intro à docker compose

Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs en utilisant un fichier au format YAML pour spécifier les services, les réseaux et les volumes. Il vous permet de créer, démarrer, arrêter et gérer plusieurs conteneurs Docker en une seule commande !

Exercice 22 – Intro à docker compose

Que dois-je installer ? Rien, en théorie docker compose est déjà installé au moment de l’installation de Docker Desktop !

Comment vérifier ? Lancez la commande :

```
docker compose --help
```

Si vous n’avez pas docker compose, il faudra l’installer sur votre poste de travail !

Exercice 22 – Intro à docker compose

Pour utiliser docker compose dans cet exercice, suivez ces étapes :

1. Créez un dossier dans lequel placer 2 sous-dossiers (PHP et PYTHON), dans lesquels vous allez ajouter les scripts des exercices précédents (exo 18 et 19), chacun dans son sous-dossier. Placez-y également les fichier Dockerfile que vous avez fabriqué, posés à côté des scripts
2. Créez un fichier « docker-compose.yaml » à la racine du dossier
3. Modifiez le fichier dans votre IDE (voir diapo suivante) :

Exercice 22 – Intro à docker compose

```yaml
version: '3.9'

services:
  php:
    build:
      context: ./PHP
      dockerfile: Dockerfile
    command: tail -f /dev/null
  python:
    build:
      context: ./PYTHON
      dockerfile: Dockerfile
    command: tail -f /dev/null
```

Exercice 22 – Intro à docker compose

Regardez bien comment est rédigé le fichier « docker-compose.yaml ». Celui-ci respecte le format YAML, qui est basé sur les indentations (comme en Python), et permet de définir plusieurs catégories et sous-catégories d’éléments.

Mon conseil : installez une extension permettant d’ajouter de la gestion de fichier YAML si votre IDE n’en a pas encore pour vous faciliter la vie !

Exercice 22 – Intro à docker compose

Une fois l’architecture respectée et les fichiers modifiés, lancez la commande depuis un terminal là ou est placé votre fichier « docker-compose.yaml » :

```
docker compose up
```

Vous allez voir tout l’intérêt, vos 2 conteneurs vont se lancer tout seul, en mode attaché au terminal !

Exercice 22 – Intro à docker compose

Si vous souhaitez couper tous les conteneurs, lancez la commande (depuis un autre terminal ouvert sur le même dossier) :

```
docker compose down
```

Vous allez voir les conteneurs se stopper tous ensembles. Vous venez d’orchestrer 2 conteneurs, chacun basé sur une image différente et faisant 2 tâches séparées.

Exercice 22 – Intro à docker compose

Beaucoup de commandes existent pour gérer vos conteneurs avec docker compose. Premièrement, à partir de maintenant, nous ne parlons plus de conteneur, mais de **service**, car vous déclarez des services dans le fichier de configuration YAML.

Ensuite, pour chaque service, vous pouvez déclarer un grand nombre d'arguments. Nous en avons utilisé 2 pour le moment, **build** afin de pointer vers le Dockerfile, et **command** pour ajouter une commande lancée par défaut au lancement du conteneur. Nous en utiliserons plus très bientôt.

Exercice 22 – Commandes docker compose

Voilà la liste des commandes courantes pour votre terminal :

- docker compose up : Démarre les services.
- docker compose up -d : Démarre les services en mode silencieux.
- docker compose down : Arrête et supprime les services.
- docker compose ps : Liste les services en cours d'exécution.
- docker compose exec <nom du service> <commande> : Exécute une commande dans un conteneur spécifique, comme nos /bin/bash ou bien des lancements de scripts !

Exercice 22 – Commandes docker compose

1. En regardant la liste des commandes disponibles, relancez vos conteneurs (vous pouvez ajouter le paramètre « -d » pour qu’ils soient lancés en arrière-plan)
2. Exécutez à nouveau des tentatives d’utilisation de vos scripts, en appelant avec la nouvelle commande exec vos services un à un et en déclenchant soit la génération d’un MDP, soit un calcul de distance entre 2 points dans le monde.
3. Pensez bien à stopper vos conteneurs une fois terminé, sinon ils vont tourner à l’infini

Exercice 23 – Gérer un volume

Les volumes Docker sont des espaces de stockage persistants qui sont partagés entre un conteneur et l'hôte sur lequel il s'exécute. Ils sont utilisés pour stocker des données qui doivent être conservées entre les redémarrages du conteneur ou les mises à jour de l'image.

On parle par exemple de la persistance de données pour un service lié à une base de données.

Exercice 23 – Gérer un volume

Docker Compose vous permet d'utiliser des volumes Docker dans vos applications. Pour ce faire, vous devez définir un volume dans votre fichier « docker-compose.yaml ». Par exemple, pour définir un volume local nommé my-database, vous pouvez utiliser la configuration suivante tout à la fin du fichier :

```
volumes:
  my-database:
```

Exercice 23 – Gérer un volume

1. Dans un nouveau dossier, créez un sous-dossier « docker » dans lequel vous allez placer un Dockerfile et un fichier « library_dump.sql » fourni par mes soins.
2. Dans votre Dockerfile, basez-vous sur l’image publique « mysql:5.7 », et copiez le « library_dump.sql » dans le dossier « /docker-entrypoint-initdb.d » de votre conteneur
3. Créez votre fichier « docker-compose.yaml », ajouter un service « mysql » qui fera appel à votre Dockerfile lors de son build

Exercice 23 – Gérer un volume

4. Ajoutez quelques éléments à votre service « mysql » :

```
environment:
  MYSQL_ROOT_PASSWORD: admin
  MYSQL_DATABASE: library
  MYSQL_USER: admin
  MYSQL_PASSWORD: admin
command: --init-file /docker-entrypoint-initdb.d/library_dump.sql
```

Vous donnez ainsi des variables comme les users, mdp, et la commande d’initialisation automatique de votre BDD au lancement du service

Exercice 23 – Gérer un volume

4. Ajoutez ensuite le volume pour la persistance, d’abord dans la définition du service « mysql » :

```
    volumes:
      - db-data:/var/lib/mysql
```

Ajoutez enfin tout en bas du fichier, au même niveau d’indentation que les services :

```
volumes:
  db-data:
```

Exercice 23 – Gérer un volume

Si tout s’est bien passé, vous devriez avoir un service « mysql » hébergeant une BDD MySQL, avec déjà une base existante « library » et quelques données à l’intérieur via le fichier « library_dump.sql »

Pour vérifier cela, allez dans votre service via la commande exec et lancez la commande :

```
mysql -u admin -padmin library
```

Vous entrerez alors dans le terminal MySQL dans votre terminal de conteneur (Inception)

Exercice 23 – Gérer un volume

Faites quelques requêtes SQL pour vérifier que votre BDD contient bien des données (SELECT par exemple).

Vous pouvez également tester d’ajouter, supprimer, modifier ces données (toujours via des requêtes SQL). Comme celles-ci sont associées à un volume docker, elles seront persistantes !

Exercice 24 – Créez un network

Orchestrer plusieurs conteneurs via Docker Compose est une démarche fascinante, mais il est essentiel de comprendre que ces conteneurs ne sont pas isolés les uns des autres lorsqu'ils sont déclarés dans le même fichier Docker Compose, ils ont un réseau par défaut. C’est un grand avantage par rapport à l’utilisation de docker seul (sans docker compose), qui était très complexe pour ajouter des communications inter-conteneurs.

Exercice 24 – Créez un network

Dans de nombreuses situations, il est toutefois nécessaire de faciliter la communication entre ces conteneurs. Prenons l'exemple d'un service "back-end" qui doit être capable de dialoguer avec une base de données exécutée sur un autre service, même si ces services partagent le même réseau par défaut.

C'est là qu'interviennent les réseaux Docker. Ils agissent comme des ponts de communication, permettant de configurer et d'optimiser les connexions entre vos services pour garantir un fonctionnement harmonieux. Testons ça !

Exercice 24 – Créez un network

Nous allons reprendre une partie de l’exercice *23*, *cad* reprendre un service basé sur l’image mysql:5.7. Il n’y a plus besoin de Dockerfile, utilisez directement la propriété « image » dans la définition de votre service mysql !

Nous allons ajouter un second service « phpmyadmin », basé sur l’image publique « phpmyadmin/phpmyadmin ». C’est ce service qui va administrer notre BDD, et les 2 doivent communiquer via un network.

Exercice 24 – Créez un network

Pour cet exercice, vous n’avez besoin que d’un seul fichier « docker-compose.yaml », puisque vous faites appel à 2 images publiques. Plusieurs choses importantes à suivre :

1. Ajoutez bien le network à la racine du fichier, avec la même indentation que les services et les volumes :

```
networks:
  mynetwork:
```

Exercice 24 – Créez un network

2. Ajoutez ensuite le network dans chaque service qui doit pouvoir s’y connecter, donc dans vos 2 services :

```
networks:
  - mynetwork
```

3. Dans le service « phpmyadmin », mappez le port de votre machine à celui du port 80 du conteneur :

```
ports:
  - "8080:80"
```

Exercice 24 – Créez un network

4. Toujours dans le service « phpmyadmin », ajoutez certaines variables permettant à ce conteneur de savoir quel est le type de BDD et comment s’y connecter pour l’administrer :

```
environment:
  PMA_HOST: mysql
  PMA_USER: admin
  PMA_PASSWORD: admin
```

Exercice 24 – Créez un network

C’est tout ! En lançant vos services, vous pourrez profiter de l’interface graphique PhpMyAdmin en vous rendant dans un navigateur web sur :

http://localhost:8080

Cerise sur le gâteau, les données sont persistées puisque vous avez un volume. Donc toutes les modifications faites depuis PhPMyAdmin seront conservées, tant que le volume reste disponible !

Exercice 25 – Un exemple à suivre

J’ai trouvé pour vous un bon exemple d’application de docker compose sur un projet hebergé par github. Celui-ci administre plusieurs services, un pour le back en PHP, un pour un front en React et un autre pour un front en Angular (pourquoi pas) :

https://github.com/kaidoj/simple-spa-reactjs-php-docker

Clonez/téléchargez ce repository, et lancez-le en local. Observez bien comment l’auteur a construit ses fichiers de configuration docker, il y a beaucoup à apprendre !

Récapitulatif

Prenez maintenant le temps de noter toutes les commandes nouvellement apprises, ce qu'elles font voire les options que vous pouvez passer en paramètre. Cette prise de notes vous sera utile !