Docker Compose

Ce qu'on a vu jusqu'ici peut être long à retenir. Surtout si on doit refaire cette configuration sur plusieurs ordinateurs. C'est là qu'entre en jeu docker compose.

Il permet de décrire les actions à réaliser pour mettre en place un projet docker et avec une seule commande mettre en place le projet dans sa totalité.

Créer le fichier compose.yaml

Pour cela à la racine de notre projet nous allons créer un fichier "**compose.yaml**".

Configuration de apache

Nous pouvons ajouter des images via des fichiers Dockerfile créés précédemment. Dans ce cas nous allons écrire ceci :

```
services:
    web:
        build: .
        container_name: serverApache2
        restart: always
        depends_on:
            - mysql
        ports:
            - "8080:80"
        volumes:
            - .:/var/www/html
        environment:
            - SUPER_ENVIRONEMENT_VARIABLE=Mon Super Site
```

Soyez prudent, en **yaml** l'indentation est importante, la moindre indentation en moins ou en trop peut provoquer des erreurs.

détaillons ce qu'on trouve dans notre fichier "*compose.yaml*" :

- **services** : Cette propriétés va lister les différents services (images) utilisés par notre projet.
- **web**: Chaque service doit avoir une propriété nommée, j'ai choisi d'appeler mon serveur apache "*web*".
- **build**: l'endroit où se trouve le dockerfile à utiliser. (ici . indique qu'il se trouve au même endroit que le compose.yaml)
- **container_name**: Optionnel, il permet d'indiquer un nom pour notre conteneur, chaque conteneur doit avoir un nom unique.
- **restart**: indique dans quel cas si le conteneur s'arrête, il doit se relancer.
- **depends_on**: permet d'indiquer dans quel ordre les services doivent se lancer. (ici on indique qu'il doit se lancer après un service nommé "mysql" que l'on va déclarer après.)
- **ports**: indique quels sont les ports qui doivent être ouvert sur la VM et le conteneur.
- **volumes**: permet de lier le conteneur à des volumes ou de faire des bind mount.

- **environment**: Permet de faire passer des variables d'environnement à notre conteneur ou notre application. (celle ici est un exemple sans utilité)

Configuration de MySQL

Nous pouvons aussi ajouter des services via des images préexistante. Toujours dans les services, ajoutons ceci :

```
mysql:
    image: mariadb
    container_name: serverMySQL2
    restart: always
    environment:
        - MARIADB_ROOT_PASSWORD=root
    ports:
        - "3308:3306"
    volumes:
        - ./mysql.sql:/docker-entrypoint-initdb.d/mysql.sql
```

Nous avons déjà vu certaines de ces propriétés mais penchons nous sur les autres :

- **mysql** : Ici aussi, c'est juste un nom que j'ai décidé de donner.
- **image** : Le nom et les possibles tags de l'image que l'on souhaite utiliser.
- **environment**: cette fois ci la variable donnée aura un véritable rôle, on l'a déjà vu lors de notre première installation de MySQL.
- **/docker-entrypoint-initdb.d/**: Ce dossier à un rôle particulier, tous les fichiers SQL qui y seront lors de la création du conteneur seront automatiquement importés dans la BDD.

Configuration de PHPMyAdmin

Ajoutons une nouvelle image pour compléter notre installation toujours à la suite de "**services**".

```
phpmyadmin:
    image: phpmyadmin
    container_name: serverPHPMyAdmin2
    restart: always
    depends_on:
        - mysql
    environment:
        PMA_ARBITRARY: 1
        PMA_HOST: mysql
    ports:
        - "8081:80"
```

Ici peu de nouveauté si ce n'est le nom que j'ai donnée à la propriété et les variables d'environnement :

- **PMA_ARBITRARY**: si "*true*" un champ permettant la selection du serveur mysql sera affiché.
- **PMA_HOST**: le nom de la BDD à laquelle se connecter.

Il y a bien sûr beaucoup d'autres possibilités mais cela je vous laisse vous renseigner par vous-même.

Les commandes de Docker Compose

Maintenant que notre fichier est créé, il ne nous reste plus qu'à lancer l'installation, pour cela plaçons notre invite de commande à la racine du projet où se trouve le "**compose.yaml**".

```
docker compose up
```

Cette commande va créer un réseau contenant nos 3 images. C'est le minimum qu'il faut écrire mais j'aimerais ajouter des options et libérer mon terminal qui est occupé avec les logs du compose. Pour cela il me faut stopper les conteneurs et les supprimer :

```
docker compose stop
docker compose rm
```

Je peux maintenant relancer ma commande:

```
docker compose -p cours-docker up -d
```

- **-p**: permet de donner un nom à notre compose, sinon il utilisera par défaut le nom du dossier dans lequel on se trouve.
- **-d**: n'a pas changé, il permet de libérer le terminal de l'emprise du compose.

À partir du moment où j'ai donné un nom à mes composes, il faudra aussi que j'utilise ces noms pour arrêter ou supprimer mes composes.

On peut stopper et supprimer un compose avec une seule commande :

```
docker compose down
```