TP #3

Etape 0

Stopper et supprimer tous les containers existants
Créer un répertoire **`docker-tp3`**
Chaque étape doit être dans un sous-répertoire nommé “etapeX” (X étant le numéro de l’étape)
Il peut être utile au fur et à mesure des étapes d’organiser les fichiers dans des sous-répertoires (config, src, … par exemple).

Initialiser un repository git dans ce répertoire.
Le TP sera à rendre sous la forme d’un repository github dont tu me donneras le lien (et les droits de consultation ;)
Si tu n’es pas très à l’aise avec git, voici un tuto assez simple pour apprendre ou avoir un rappel des bases : https://www.ionos.fr/digitalguide/sites-internet/developpement-web/tutoriel-git/

Les questions 1 à 2 sont à réaliser sans Docker Compose
Il est interdit d’utiliser Docker Compose avant l’étape 3

Etape 1

2 containers nommés comme suit :
- HTTP : 1 container avec un serveur HTTP qui écoute sur le port 8080
- SCRIPT : 1 container avec un interpréteur PHP (plus le protocole FPM pour NGINX)

Une page index.php qui lorsqu'elle est appelée exécute la fonction `phpinfo()` et qui sera situé dans les containers dans le répertoire `/app`.

Test de validité de l'exercice : avec un navigateur voir le résultat de l'exécution du `php_info()`

Pour réaliser cette étape, tu auras besoin :
- d’organiser dans un répertoire de ton hôte Docker les différents fichiers dont tu auras besoin. Les sources du site (1 fichier php : `index.php`) sont fournies dans une archive qui accompagne ce TP. Les sources du site seront à ajouter aux 2 containers grâce à des volumes bind mount.
- de lancer des containers avec `docker container run`, un `nginx` et un `php`
- de reconfigurer le container `nginx` pour aller communiquer avec le container `php` comme dans les TP précédents avec des volumes bind mount sur le fichier de configuration de `nginx`. Les modifications à effectuer sur le fichier default.conf sont plus bas.

- de mettre les containers en communication via un réseau commun (pas le réseau par défaut) avec `docker network create` et aussi l'option `--network` de `docker container run`

Décommenter et remplacer les lignes 30 à 36 du fichier `default.conf` de Nginx par les suivantes :

```
        location ~ \.php$ {
                root           /app;
                fastcgi_pass   script:9000;
                fastcgi_index  index.php;
                fastcgi_param  SCRIPT_FILENAME
$document_root$fastcgi_script_name;
                include        fastcgi_params;
        }
```

Etape 2

3 containers nommés comme suit :
- HTTP : 1 container avec un serveur HTTP qui écoute sur le port 8080
- SCRIPT : 1 container avec un interpréteur PHP (plus le protocole FPM pour NGINX)
- DATA : 1 container avec un serveur de base données SQL (MariaDB)

Une page `test.php` qui lorsqu'elle est appelée va exécuter 2 requêtes CRUD (Request : lecture, Create Update Delete : écriture) sur le serveur SQL : 1 écriture et 1 lecture.

Test de validité de l'exercice : avec un navigateur voir le résultat de l'exécution de la page en retournant un résultat différent et dépendant du contenu de la base de données à chaque refresh de la page

Pour réaliser cette étape, tu auras besoin :
- d'avoir fait l'étape 1
- du fichier `test.php` et du fichier `create.sql` qui sont fournis dans une archive qui accompagne ce TP.
- de lancer un container supplémentaire basé sur l'image `mariadb`.
- d'initialiser ce container de base de données grâce au répertoire `/docker-entrypoint-initdb.d` (voir le paragraphe "Initializing the database contents" sur la page de l'image MariaDB)
- définir une variable d'environnement pour l'image MariaDB : `MARIADB_RANDOM_ROOT_PASSWORD`
- d'ajouter l'extension `mysqli` dans le container `php` (voir le paragraphe "How to install more PHP extensions" de la page de l'image PHP) grâce à un `Dockerfile`.

Etape 3

Convertir la configuration de l'étape 2 en Docker Compose

Test de validité de l'exercice : identique à l'étape 2