R4.08 - TD 4
Mini Projet

Introduction

La ville de Trifouilly-lès-bigorneaux fait appel à votre expertise afin de concevoir, pour sa bibliothèque municipale, un outil de gestion des prêts de livres.

Pour ce TD, on s'intéressera uniquement à l'aspect entrées et sorties des livres.

Un livre sort de la bibliothèque dans 2 situations :

- Prêt à une personne, c'est une sortie temporaire
- Suppression, c'est une sortie définitive

Un livre entre dans la bibliothèque dans 2 situations aussi :

- Retour de prêt d'une personne
- Ajout d'un nouveau livre

Chaque livre a les caractéristiques suivantes :

- **`id`** : Numéro unique (entier)
- **`nom`** : Nom (chaîne de 80 caractères)
- **`auteur`** : Auteur (chaîne de 40 caractères)
- **`total`** : Quantité possédée par la bibliothèque (entier ≥ 0)
- **`pret`** : Quantité prêtée (entier ≥ 0)

Vous devez utiliser **`PDO`**.

Vous devez produire un **`docker-compose.yml`**.

Vous devez produire les **`Dockerfile`** utiles en fonction des images disponibles et des adaptations que vous jugez nécessaires.

Exercice 1 - Mise en place

Il s'agit ici d'une reproduction du **TD 3**, mais lisez bien cet **Exercice 1** jusqu'au bout avant de vous lancer.

Comme pour le **TD 3**, mettez en place une stack **PHP_Apache + MySQL + PHPMyAdmin**

Attention, vous devez configurer cette stack à l'aide d'un **`docker-compose.yml`** et non pas manuellement !

Rappel des noms des images à l'IUT :

- PHP_Apache : **`r408-php:8.2-apache`**
  N'oubliez pas qu'il manque sans doute le driver **`PDO`** de **`MySQL`** dans cette image.
- MySQL : **`r408-mysql:5.7`**
- PHPMyAdmin : **`r408-phpmyadmin:5.2`**

Votre serveur Apache doit écouter le port **`5555`** sur l'hôte. Le port dans le conteneur est évidemment **`80`**, puisqu'il s'agit d'un serveur Web.

Votre base de données doit s'appeler **`biblio`**.

Préservation des données

Compte tenu du fait que vous souhaitez sans doute persister vos données de BDD en cas de suppression du conteneur MySQL, vous devrez prévoir un volume pour stocker vos fichiers de BDD.

Rappel sur l'utilisation d'un volume managé dans votre **`docker-compose.yml`** (les **`...`** indiquent que vous avez d'autres lignes, à vous de voir où insérer ce code)

```
services:
...
    db:
        ...
        volumes:
            - mysql_vol:/var/lib/mysql
...
volumes:
    mysql_vol:
```

Exercice 2 - Création de la BDD

Dans votre BDD **`biblio`**, créez une table **`livres`** pour stocker des livres structurés tels que décrit en **Introduction**. Vous aurez donc certainement besoin d'un client MySQL pour faire ça. Référez-vous au **TD 3** si votre mémoire fait défaut.

Remplissez-là avec les données du fichier **`biblio.txt`**, récupéré sur Moodle. Vous avez un module d'import de données dans l'interface de PHPMyAdmin. Ce fichier de données doit, en principe, être importable sans aucun changement, sous réserve d'avoir respecté la structure de données imposée (liste et position des champs, types).

Exercice 3 - Création automatique de la BDD

Vous savez déjà qu'à la création d'un conteneur MySQL avec l'image officielle **`r408-mysql:5.7`** (celle de l'IUT est une copie exacte de l'officielle), une base de données vierge est créée, dont le nom est donné par la variable d'environnement **`MYSQL_DATABASE`**. Si vous avez suivi les instructions, la vôtre doit s'appeler **`biblio`**.

C'est d'ailleurs dans cette base que vous avez créé la table **`livres`**.

Si vous transférez votre **`docker-compose.yml`** sur une autre machine et que vous y démarrez cet environnement dockerisé, vous aurez bien une base de données **`biblio`**, mais elle sera totalement vierge. C'est normal car vos fichiers de BDD sont restés sur l'autre machine.

Il peut être intéressant de livrer votre environnement dockerisé (le **`docker-compose.yml`**) avec une base de données de départ. Ici on aurait besoin de la table **`livres`**, même vide.

L'image MySQL officielle prévoit cette situation.

Scripts d'initialisation à la création

Dans l'image (et donc dans tout conteneur créé à partir de cette image) il y a un dossier **`/docker-entrypoint-initdb.d/`** dans lequel on peut placer des scripts SQL (ayant une extension **`.sql`**) qui seront exécutés à la création de la BDD et plus après.

Ces scripts sont exécutés par ordre alphabétique et ne doivent contenir que des requêtes SQL, et rien d'autre.

Par défaut et sans instruction contraire, ces scripts sont exécutés sur la base de données indiquée dans **`MYSQL_DATABASE`** (la base de données créée au démarrage)

Création automatique de la table `livres`

Créez un script **`init_biblio.sql`** et placez-y le code SQL nécessaire à la création de votre table **`livres`**. Vous devez trouver ce code SQL dans les informations de la table, dans l'interface PHPMyAdmin.

Placez ce script au bon endroit. Pour ce faire, vous avez le choix entre deux méthodes :

- **Méthode 1** : Créez aussi une image personnalisée pour votre conteneur de BDD. Placez-y une étape de copie du script dans l'image.

  Comme vous avez déjà un **`Dockerfile`** pour l'image **`Apache-PDO`**, vous allez devoir soit nommer autrement votre second **`Dockerfile`**, soit le placer ailleurs quand dans le dossier courant. Peu importe votre choix, vous devrez préciser les

sous-attributs **`context`** et **`dockerfile`** dans l'attribut **`build`** de votre **`docker-compose.yml`**. Pour plus de détail, consultez la documentation (https://docs.docker.com/compose/compose-file/compose-file-v3/#build)

Dans le **`Dockerfile`** (ou le nom que vous lui aurez choisi), pour ajouter un fichier de l'hôte dans l'image, utilisez l'action **`COPY`** (voir la documentation sur https://docs.docker.com > **Reference** > **Dockerfile reference** > **COPY**). Rappel : les scripts d'initialisation doivent être placés dans le dossier **`/docker-entrypoint-initdb.d/`** du conteneur de BDD.

- **Méthode 2** : Placez votre script dans un autre volume que vous devez mapper sur **`/docker-entrypoint-initdb.d/`** dans le conteneur.

La **Méthode 2** n'est pratique que dans le cadre de la mise au point car elle évite la reconstruction d'une image, mais elle nécessite de fournir aussi le script si vous passez votre environnement à quelqu'un d'autre et impose la création d'un volume dédié et utile uniquement au 1er lancement. La première méthode est donc à privilégier si vous ne fournissez qu'une image qui pourra alors être publiée sur un Hub.

Pour ce projet, vous devez mettre en œuvre la **Méthode 1**.

Test

Testez votre nouvel environnement dockerisé (à base d'image avec la **Méthode 1**) en suivant ces étapes :

- Arrêtez votre stack.
- Dans votre **`docker-compose.yml`**, mettez en commentaire la section de mappage du volume stockant les données de la BDD. Ainsi, au prochain lancement, le conteneur sera démarré sans BDD passant alors par la phase d'initialisation, c'est-à-dire l'exécution de tous les scripts placés dans le dossier **`/docker-entrypoint-initdb.d/`**.
- Redémarrez votre stack.

Votre test est validé si vous trouvez une base de données **`biblio`** avec ses tables vides.

Lors de la phase de mise au point, entre chaque test, n'oubliez pas d'arrêter votre stack.

Une fois le test terminé et validé, vous pouvez supprimer les commentaires concernant la section mappage du volume et redémarrer une dernière fois votre stack.

Conseil d'entretien de la base initiale

Pour conserver son intérêt, ce mécanisme de scripts d'initialisation placés dans **`/docker-entrypoint-initdb.d/`** nécessitera que chaque évolution d'une BDD soit répercutée dans le script d'initialisation (ici **`init_biblio.sql`**), et donc re-testé comme ci-dessus.

Exercice 4 - Affichage des données

Créez une page Web PHP, nommée **livres.php**, qui affiche la liste des livres de la table **livres**, dans une **<TABLE>...</TABLE>**. Vous devez afficher tous les attributs de chaque livre. Faites simple, pas besoin de pagination.

Exercice 5 - Traitement par lot

Un traitement par lot (**batch** en anglais) permet de faire des séries d'actions sans intervention humaine.

Vous allez mettre en œuvre un batch permettant ces actions :

- **OUT** : Sortie d'un livre (prêt à un usager de la bibliothèque), quantité **= 1**
- **IN** : Retour d'un livré (retour de prêt), quantité **= 1**, seulement si **pret > 0**
- **ADD** : Ajout d'un livre (achat, don, etc), quantité **> 0**
- **DEL** : Suppression d'un livre (perte, destruction, don), quantité **> 0** mais ne peut pas dépasser **total**

Sur Moodle, vous avez un fichier **mvt** contenant des mouvements sur le stock de livres.

Voici ce qu'il contient :

```
id_livre:action:nom:auteur:qté
```

Certains champs sont vides car inutiles pour l'action en question :

- **nom** et **auteur** ne servent que pour l'action **ADD**
- **id** ne sert pas pour **ADD** car il sera créé automatiquement

Créez un script **do_batch.php** qui traite le contenu d'un fichier texte (comme le fichier **mvt**) et effectue les actions trouvées dans le fichier.

Ce script doit s'exécuter dans un conteneur éphémère PHP. C'est un conteneur séparé du conteneur **PHP_Apache** mis en place précédemment. Cependant, par soucis de simplicité et comme vous avez déjà créé une image PHP avec **PDO** intégré, vous utiliserez la même image (l'image pas le conteneur) que pour le conteneur **PHP_Apache**. Vous devez brancher ce conteneur éphémère sur un réseau qui lui est dédié et qui doit être séparé de celui du conteneur **PHP_Apache**. Vous devez prévoir que ce conteneur doit communiquer avec la BDD, donc ils doivent être tous deux sur ce même réseau dédié à ce besoin.

Ce conteneur éphémère est lancé manuellement pour exécuter le script **do_batch.php**. Pour ce faire prévoyez un script shell **go_batch**

Exercice 6 - Dépôt FTP

Il est décidé que les fichiers de données servant au batch de l'**Exercice 5** vont être déposés dans un dépôt FTP dans lequel des processus externes pourront venir déposer des fichiers d'actions et sur lequel **`do_batch.php`** viendra s'alimenter.

Ajoutez un conteneur FTP à votre environnement (dans le **`docker-compose.yml`**), avec les caractéristiques suivantes :

- Image : **`r408-pure-ftpd:1.0`** (vous penserez à faire un **`docker image pull`** avant)
- Mappage de ports :
  - **`9021`** de l'hôte vers **`21`** du conteneur
  - **`30000-30009`** (c'est un lot, à saisir ainsi) de l'hôte vers le même lot de ports **`30000-30009`** du conteneur
- Variable d'environnement (mot clé **`environment:`** dans un service)
  - **`FTP_USER_NAME=bibliotekr`**
  - **`FTP_USER_PASS=poipoi`**
  - **`FTP_USER_HOME=/home/bibliotekr`**
- Isolé des autres conteneurs, aucune connexion réseau avec eux

Pour ce TD, il n'est pas nécessaire de définir de volume, mais généralement on souhaitera persister les données du conteneur. Dans ce cas, il vous faudrait mapper un autre volume créé pour ce besoin, vers **`/home`** dans le conteneur.

IMPORTANT : les variables **`FTP_USER_NAME`**, **`FTP_USER_PASS`** et **`FTP_USER_HOME`** sont précisées en dur ici pour une question de simplicité. Dans la vraie vie (d'entreprise) on gérerait les utilisateurs différemment et de façon plus sécurisée et confidentielle. Pour ce TD, ce sera suffisant ainsi, pour ne pas complexifier la configuration. Pour plus d'information, consultez la documentation de l'image originelle (**`stilliard/pure-ftpd`**).

Vous l'aurez donc compris, un seul utilisateur dans notre serveur FTP. Il s'appelle **`bibiotekr`** avec **`poipoi`** comme mot de passe.

Un **`docker-compose up -d`** devrait ajouter et démarrer ce nouveau conteneur.

Pour tester, utiliser le client **FileZilla** qui existe sous Linux, Windows et macOS.

Exercice 7 - Scan FTP

Sur la base du script **`do_batch.php`** de l'**Exercice 5** écrivez un script **`scan_ftp.php`** qui va, à intervalle régulier, regarder s'il y a des fichiers à traiter sur le dépôt **`FTP`**, les récupérer et les passer à la moulinette du **`do_batch.php`**.

L'intervalle est fixé à 30 secondes pour les tests mais serait, par exemple, de 15 minutes voire plus en production et en fonction du temps de réaction jugé suffisant pour votre application.

Vous savez peut-être (sinon vous allez l'apprendre) que les fonctions de lecture de fichiers en PHP (**`fopen()`**, **`file()`**, **`file_get_contents()`**, etc.) savent travailler aussi bien avec des fichiers locaux qu'avec des fichiers accessibles par différents protocoles, tels que **`HTTP`** ou encore **`FTP`** ! Et ça c'est plutôt cool comme fonctionnalité.

Cependant, dans cet exercice nous allons être obligés d'opérer un peu différemment car lire un fichier sur un serveur FTP à l'aide d'une fonction comme **`file_get_contents()`** est possible uniquement si on connaît son URL et donc le nom du fichier ciblé ! Or, ce ne sera pas le cas ici. On peut déposer des fichiers de n'importe quel nom sur le serveur FTP.

Vous allez donc devoir utiliser les fonctions de manipulation FTP fournies par PHP, mais rien de bien compliqué.

Consultez, dans la documentation PHP sur https://www.php.net, la famille des fonctions **`FTP`**. Voici les fonctions qui vont certainement vous être utiles :

- **`ftp_connect`**
- **`ftp_login`**
- **`ftp_nlist`**
- **`ftp_get`**
- **`ftp_delete`**
- **`ftp_close`**

Vous devez, dans une boucle sans fin, traiter les fichiers présents sur le serveur FTP et les supprimer ensuite pour ne pas les traiter à nouveau au passage suivant dans la boucle sans fin.

Faites une pause de 30 secondes avant de re-boucler.

Comme toujours, vous afficherez des logs détaillés de ce qui se passe, ça rassure et ça informe !

Créez un conteneur intégrant ce script **`scan_ftp.php`** et le **`do_batch.php`**. Ce conteneur doit démarrer avec les autres conteneurs, à partir du **`docker-compose.yml`**.
Arrangez-vous pour le placer sur un réseau lui permettant d'accéder aux conteneurs dont il a besoin et uniquement à ceux-là.

Exercice 8 - Backup BDD

Comme tout informaticien·ne qui se respecte, vous avez la sécurité et la prévoyance fermement ancrées en vous. Vous avez donc décidé de faire des sauvegardes régulières

de votre BDD et de les mettre à disposition de votre client, sur le serveur FTP, dans un espace dédié.

Les sauvegardes iront dans un sous-dossier du serveur FTP, nommé **`backups/`**

Vous devez donc commencer par adapter un peu le script **`scan_ftp.php`** pour exclure ce dossier de la liste des traitements qu'il fait.

Ecrivez ensuite un script **`backup.php`** qui :

- Génère un fichier **`backup_YYYYMMDD_hhmmss`** (où **`YYYY`** = année, **`MM`** = mois, **`DD`** = jour, **`hh`** = heure, **`mm`** = minutes et **`ss`** = secondes) contenant la liste des livres de la table livres au même format que le fichier **`mvt`** de l'Exercice 5.
- Dépose ce fichier dans le dossier **`backups/`** du serveur FTP.

Ce script **`backup.php`** doit s'exécuter dans un conteneur éphémère de type **Apache_PHP** aussi, car c'est la partie PHP avec **`PDO`** qui nous intéresse et pas Apache.

Pour cet exercice, la fonction **`ftp_put`** viendra certainement compléter votre panoplie de fonctions FTP utiles.

Le conteneur sera lancé manuellement et ponctuellement. Pour ce faire prévoyez un script shell **`go_backup`**

Vérifiez avec **FileZilla** le bon dépôt du fichier de backup. Vérifiez aussi son contenu !