

## TP n° 8

### MongoDB et volumes docker

préambule : Dans ce TP, nous allons mettre en œuvre un *stack* presque complet avec node.js, Express et mongoDB. Comme précédemment, tous les exercices de cette séance peuvent être réalisés sur le serveur `licinfo2.univ-jfc.fr` ou bien sur votre machine personnelle. Dans les 2 cas, l'infrastructure du *stack* est grandement facilitée par l'utilisation de docker.

### Exercice 1 – Volumes docker

Pour éviter les problèmes d'incompatibilité entre les différents SGF sur lesquels une pile de conteneurs pourraient être déployée, docker met à disposition un système de volumes. Un volume docker est une zone de stockage, munie de son propre système de gestion de fichiers (SGF), qui l'on peut monter dans un conteneur et utiliser pour y stocker les données du conteneur.

 Créer un nouveau volume docker grâce à la commande suivante :

```
docker volume create db_volume
```

Vérifier que le volume apparaît lorsqu'on exécute la commande : `docker volume ls`.


Un volume, bien qu'il soit stocké localement sur la machine, n'est pas explorable directement comme un répertoire classique. Il est nécessaire de le monter dans un conteneur. Exécuter par exemple la commande suivante :

```
docker run -it -v db_volume:/data alpine
```

Cette commande lance (run) en mode interactif (-it) un conteneur Linux alpine dans lequel le volume `db_volume` est monté (-v) au démarrage dans le répertoire `/data`.


 Créer un nouveau fichier dans `/data` en exécutant la commande :

```
echo "Quelques caractères dans un" > fichier
```

 Quitter le conteneur grâce à la commande `exit`. Relancer le conteneur pour vérifier que le fichier est toujours présent dans le volume.


### Exercice 2 – Serveur mongoDB


Dans cet exercice, nous mettons en place le serveur mongoDB de notre *stack*. Pour cela, nous utilisons l'image officielle "mongo" en version 4. MongoDB<sup>1</sup> est un SGBD orienté document dans lequel les données ne sont pas stockées sous forme d'enregistrements dans des tables mais de documents JSON binaires (BSON) dans des collections. MongoDB offre toutes les fonctionnalités nécessaires pour l'insertion, la recherche et la manipulation de données hétérogènes.

 Écrire un fichier `docker-compose.yml` dans lequel le serveur de base de données est configuré comme suit :


```
1 version: "2"
2 services:
3   mongodb:
4     container_name: dpanzoli_mongodb
5     image: "mongo:4.0"
6     volumes:
7       - db_volume:/data/db
8
9 volumes:
10   db_volume:
11     external: true
```


À la ligne 4, le nom du conteneur doit être personnalisé pour le retrouver facilement. Le conteneur monte le volume `db_volume` (ligne 7) qui est déclaré en fin de fichier (lignes 9 à 11). La propriété `external: true` indique à docker que le volume est déjà existant et ne doit pas être (re)créé.

 Démarrer le *stack* avec la commande `docker-compose up`. Utiliser le paramètre `-d` pour lancer en mode détaché, ou bien lancer un nouveau terminal pour les questions suivantes.

 Lancer un interpréteur de commande `bash` sur le serveur de base de données en exécutant la commande suivante :

```
docker exec -it dpanzoli_mongodb bash
```

 Une fois à l'intérieur du conteneur, lancer le *shell* mongoDB en exécutant simplement la commande : `mongo`.

 Exécuter successivement les commandes<sup>2 3</sup> suivantes :


- `use test`  
↪ Utilise (ou crée le cas échéant) la base de données *test*
- `db.createCollection('users')`  
↪ Crée une nouvelle collection *users*.
- `db.users.insertOne({username: 'dpanzoli', admin: true})`  
↪ Insère un nouvel enregistrement dans *users*.
- `db.users.insertMany([{username: 'ngarric'}, {username: 'fpouit'}])`  
↪ Insère deux nouveaux enregistrements.

1. Informations ici : <https://www.mongodb.com/fr>

2. Une introduction à mongoDB : <https://www.tutorialspoint.com/mongodb/index.htm>

3. La documentation officielle de la version 4.0 : <https://docs.mongodb.com/v4.0/reference/>

- `show dbs`  
↪ Liste les bases existantes (*test* apparaît désormais).
- `db.users.find({})`  
↪ Affiche tous les documents de *users*.
- `db.users.find({admin: {$exists: false}})`  
↪ Affiche tous les utilisateurs qui ne sont pas administrateurs.


 Fermer le *shell* mongoDB, terminer l'exécution de l'interpréteur de commande et enfin stopper le conteneur.

 Comme vu dans l'exercice précédent, explorer le volume à l'aide d'un conteneur Linux alpine. Vérifier la présence des fichiers créés par mongoDB dans le volume.

```
/data # ls
WiredTiger                fichier
WiredTiger.lock           index-1-5625045800999026312.wt
WiredTiger.turtle         index-10-5625045800999026312.wt
WiredTiger.wt             index-3-5625045800999026312.wt
WiredTigerLAS.wt          index-5-5625045800999026312.wt
_mdb_catalog.wt           index-6-5625045800999026312.wt
collection-0-5625045800999026312.wt index-8-5625045800999026312.wt
collection-2-5625045800999026312.wt journal
collection-4-5625045800999026312.wt mongod.lock
collection-7-5625045800999026312.wt sizeStorer.wt
collection-9-5625045800999026312.wt storage.bson
diagnostic.data
/data # |
```

## Exercice 3 – (*Almost full-*)stack

Dans cet exercice, on connecte le serveur de base de données avec un serveur Express.

 Modifier le fichier `docker-compose.yml` pour ajouter un conteneur node.js, comme illustré ci-dessous :


```
1 version: "2"
2 services:
3   nodeserver:
4     container_name: dpanzoli_node
5     image: "node:alpine"
6     user: "node"
7     working_dir: /home/node/app
8     volumes:
9       - ./node:/home/node/app
10    ports:
11      - "10042:3000"
12    command: "npm start"
```

```

13     links:
14         - mongodb
15     mongodb:
16         container_name: dpanzoli_mongodb
17         image: "mongo:4.0"
18         volumes:
19             - db_volume:/data/db
20
21 volumes:
22     db_volume:
23         external: true

```

Aux lignes 4, 11 et 16, les noms des conteneurs ainsi que le port exposé par le serveur node.js doivent être personnalisés.

 Créer un sous-répertoire node à la racine du projet, puis démarrer un nouveau projet avec la commande `npm init`. Ajouter ensuite dans votre `package.json` la commande `start` qui permet de lancer le serveur `server.js`.

 Créer le fichier `server.js` à partir du code *boilerplate* ci-dessous :

```

1 const express = require('express')
2 const app = express();
3 const http = require('http').createServer(app);
4 const MongoClient = require('mongodb').MongoClient;
5
6 const url_db = 'mongodb://mongodb:27017';
7
8 app.get('/', (req, res) => {
9     MongoClient.connect(url_db, function(err, db) {
10         var dbo = db.db("test");
11         dbo.collection("users").find({}).toArray(function(err, result) {
12             if (err) throw err;
13             res.send(result);
14             db.close();
15         });
16     });
17 });
18
19 http.listen(3000, () => {
20     console.log('listening on *:3000');
21 });


```


En plus du code nécessaire pour construire un serveur Express simple, on trouve à la ligne 6 la déclaration de l'URL pour accéder au serveur mongoDB en utilisant le protocole `mongodb` : celui-ci est accessible par son nom de service, car dans un *stack* docker, la résolution des serveurs utilise leur alias.

Dans les lignes 9 à 16, le script accède à la base, effectue une transaction (*find*) et ferme l'accès. Une fonction de *callback* récupère le résultat sous forme d'un tableau JSON et le retourne au client.

 Installer les modules `express` et `mongodb` avec la commande `npm install ....` Mon-




ter votre *stack* et tester l'application dans un navigateur.

 Fabriquer un gabarit PUG permettant d'afficher les enregistrements de la collection `users` dans un tableau stylisé comme illustré dans la figure ci-dessous. Mettre en place le moteur de gabarit dans le serveur.

 Enrichir l'application avec la possibilité d'ajouter de nouveaux utilisateurs. Les étapes nécessaires sont :

- Ajout d'un formulaire dans le gabarit (voir illustration ci-dessous).
- Ajout (directement dans le gabarit) d'un script pour gérer l'appel AJAX lors de l'envoi du formulaire et le rechargement de la page après insertion.
- Ajout dans le serveur d'une route `/addUser` (GET) pour gérer l'insertion dans la base de données <sup>4</sup>


Le rendu final de l'application affiche les documents récupérés dans la base dans un tableau stylisé (ici avec *semantic.ui*). Un formulaire permet d'ajouter de nouveaux utilisateurs dans la base. La gestion de la suppression n'est pas demandée.

Username	Administrator	
dpanzoli	✓	
ngarric		
fpouit		
<input type="text" value="tmontaut"/>	<input checked="" type="checkbox"/> administrator	<input type="button" value="Insert"/>

## Exercice 4 – Importation de données

Dans cet exercice, nous apprenons à importer des données depuis un fichier JSON vers un serveur mongoDB. Nous utilisons toujours le volume `db_volume` et la même pile de conteneurs (`node.js` + `mongoDB`) que l'exercice suivant.

 Récupérer le fichier `books.json` contenant une collection de livres d'informatique.

 Depuis l'emplacement du fichier `books.json`, lancer les commandes suivantes pour 1) copier le fichier `books.json` à la racine de votre conteneur mongoDB puis 2) importer le fichier dans votre base de données *test* dans une nouvelle collection *books*.

```
docker cp books.json dpanzoli_mongodb:/
docker exec dpanzoli_mongodb mongoimport --db=test
→ --collection=books --file books.json
```

En cas de succès de l'opération, la console doit afficher que 432 documents ont été importés.

Note : la commande `mongoexport` permet d'effectuer l'opération inverse, qui consiste à sauvegarder une collection sous la forme d'un fichier JSON.

 Lancer ensuite un interpréteur de commande puis le *shell* mongoDB (voir exercice 2) ou bien directement la commande :

4. Documentation : [https://www.w3schools.com/nodejs/nodejs\\_mongodb\\_insert.asp](https://www.w3schools.com/nodejs/nodejs_mongodb_insert.asp).

```
docker exec -it dpanzoli_mongodb mongo
```

Vérifier que tous les documents ont bien été importés, par exemple en écrivant :

```
> use test
switched to db test
> db.books.count()
431
```

✎ Implémenter une mini-application Express permettant d'effectuer des recherches de livres de la collection *books* à partir de leur titre. L'application (illustrée ci-dessous) se présente sous la forme d'une barre de recherche dans laquelle l'utilisateur est invité à entrer une séquence de caractères. Dès que cette dernière est modifiée, et si sa taille supérieure à 2 caractères, l'application met à jour en temps réel la liste mise en forme des ouvrages dont le titre contient la séquence de caractères (on utilisera pour la recherche une expression régulière<sup>5</sup>).

#### Titre de l'ouvrage



#### Flex on Java

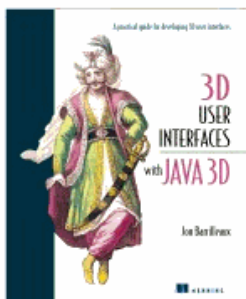
Auteurs

Bernard Allmon, Jeremy Anderson

Description

A beautifully written book that is a must have for every Java Developer. Ashish Kulkarni, Technical Director, E-Business Software Solutions Ltd.

Publié en 2010



#### 3D User Interfaces with Java 3D

Auteurs

Jon Barrilleaux

Publié en 2000



#### Java Persistence with Hibernate

Auteurs

Christian Bauer, Gavin King

Description

"...this book is the ultimate solution. If you are going to use Hibernate in your application, you have no other choice, go rush to the store and get this book." --JavaLobby

Publié en 2006



#### Java Foundation Classes

5. <https://docs.mongodb.com/manual/reference/operator/query/regex/>