## Activité 3 : Lancement d'une application avec Docker Compose

Dans la précédente activité, le lancement de l'application Web « geststages » a nécessité l'utilisation et le démarrage de deux conteneurs.
Dans une architecture « micro-services », cette situation est très courante avec beaucoup plus de conteneurs et des commandes pour les lancer parfois très longues et compliquées.

La technologie Docker Compose va nous permettre de nous consacrer uniquement à la définition de l'architecture, et de laisser la gestion des scripts à l'outil. Un fichier docker-compose.yml va contenir la liste des conteneurs avec leurs liens, leur image, la définition des ports, etc. La commande docker-compose nous permettra de piloter en une seule ligne de commande le lancement de tous les conteneurs décrits dans ce fichier, leur arrêt, etc.

**Docker Compose** est un outil qui permet (entre autres) de décrire, gérer et démarrer plusieurs conteneurs (et les liens entre eux) qui composent une seule et même application dans un environnement isolé et sécurisé.

**Typiquement, Docker Compose permet la définition de l'architecture de l'application et l'orchestration très simple des conteneurs en :**

- définissant l'ensemble des services qui composent une application dans un fichier YAML « docker-compose.yml » afin qu'ils puissent être exécutés ensemble dans un environnement isolé ;
- permettant d'obtenir une application en cours d'exécution en une seule commande (docker-compose up).

Cela va donc nous donner la possibilité, depuis une commande unique, de créer et démarrer tous les services nécessaires au fonctionnement de notre application « geststages »..

De nombreuses applications connues (comme OCSinventory et Graylog) sont fournies sous forme de « stack » via Docker Compose. Il est donc intéressant d'en comprendre le fonctionnement.

À noter que Docker Compose est fourni par les développeurs de Docker mais n'est pas inclus dans l'installation de base des paquets.

**Vous disposez de la documentation suivante** :

- **document 1** : Installation de docker compose ;
- **document 2** : Exemple d'une configuration « docker compose » ;
- **document 3 :** Commandes utiles.

## Travail à faire

Installez Docker-compose.

Avant de continuer, il est conseillé de réaliser toutes les manipulations détaillées dans le document 2. *Vous pourrez éventuellement ensuite supprimer tous les conteneurs créés et toutes les bases de données (en supprimant les volumes correspondants) ainsi que les images..*

Vous trouverez l'application « geststages » à l'adresse suivante : https://gitlab.com/reseaucerta
La récupération du code, via la commande « git clone », créera l'arborescence suivante :
.

un dossier « docker-stack-geststages » qui accueillera :

- le fichier « docker-compose.yml » ;
- un dossier « initSQL » qui contient le script sql « gestages.sql » ;
- un dossier « codePHP » qui contient les fichiers sources de l'application web.

Récupérez (via la commande « git clone ») le code de l'application.

**2 services doivent être créés : « web » et « servbd ».**

Au niveau de l'application, il est rappelé que l'accès à la base de données (qui se fait sur l'hôte « servbd » d'où le nom de service obligatoire « servbd » sinon il est nécessaire d'ajouter un alias) est configuré dans « include/fonction/f_bdd.php » dont l'extrait est fourni ci-après :

```
...
        $bdd = new PDO('mysql:host=servbd;port=3306;dbname=bdd_geststages;charset=utf8', 'usergs', 'mdpGS', array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
...
```

Ainsi, l'application web a accès à la base de données « bdd_geststages » avec l'utilisateur « usergs » qui a pour mot de passe « mdpGS ». Ces éléments sont créés lors de l'importation du fichier SQL.

**En ce qui concerne le service « web » :**

- il sera basé sur l'image « vos_initiales/debian:stretch-apache2-php7 » créée dans l'activité 2 ;
- le dossier « codePHP » sera mappé sur « /var/www/html » ;
- aucune variable d'environnement n'est nécessaire ;
- le service « apache » doit être démarré (/usr/sbin/apache2ctl -DFOREGROUND).

**En ce qui concerne le service « servbd » :**

- il sera basé sur l'image « mariadb » ;
- le dossier « initSQL » sera mappé sur /docker-entrypoint-initdb.d. L'image « mariadb » prévoit, à la création du conteneur, le lancement du fichier « entrypoint.sh » (présent à la racine du conteneur) qui, lui-même, charge tous les scripts du répertoire /docker-entrypoint-initdb.d ;
- un volume « mysqldata » sera prévu pour persister les données de la base ;
- l'accès au serveur de base de données doit être accessible via le port 3306 uniquement par l'application. Ce port est exposé au niveau de l'image, il est donc inutile de faire une redirection sur l'hôte ;
- seul le mot de passe de l'utilisateur root de mysql a besoin d'être défini.

La base de données ainsi que l'utilisateur associé sont créés dans le script SQL ; il est possible (comme dans l'exemple du document 2) de les créer via les variables d'environnement mais il faut alors supprimer les lignes correspondantes dans le script.

- En vous aidant du document 2, créez et complétez le fichier « docker-compose.yml » présent dans le répertoire « docker-stack-geststages ».

- Lancez les conteneurs et vérifiez leurs exécutions.

- Vérifiez l'opérationnalité de l'application.

# Document 1 - Installation de docker-compose

Docker Compose est un outil permettant de définir des applications composées de plusieurs conteneurs, de les *packager*, de les exécuter et de les déployer.

Le docker-compose disponible dans les dépôts officiels Debian est obsolète, nous installerons la dernière version en manuel. Pour connaître la dernière version : https://github.com/docker/compose/releases

**Installation des dépendances**

```
apt install python-backports.ssl-match-hostname python-cached-property python-docker python-dockerpty python-docopt python-functools32 python-jsonschema python-texttable python-websocket python-yaml
```

**Téléchargement de la dernière version du logiciel docker-compose.**Le numéro de version doit être adapté en visitant le lien https://github.com/docker/compose/releases

```
curl -L https://github.com/docker/compose/releases/download/1.23.2/docker-compose-`uname -s`-`uname -m` -o /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
```

Pour vérifier que tout fonctionne :

```
docker-compose version
```

```
docker-compose version 1.23.2, build 1110ad01
docker-py version: 3.6.0
CPython version: 3.6.7
OpenSSL version: OpenSSL 1.1.0f  25 May 2017
```

# Document 2 - Exemple d'une configuration « docker compose »

## D2.1 : le fichier docker-compose.yml du docker ocs-inventory-ng officiel

Vous pouvez télécharger le « stack » docker « OCSinventory-ng » via la commande « git clone https://github.com/OCSInventory-NG/OCSInventory-Docker-Stack ». L'arborescence créée contient notamment le fichier « docker-compose.yml » présenté et expliqué ci-dessous et le script *sql* d'initialisation de la base de données « ocsweb ». La syntaxe est bien sûr conforme à celle d'un fichier YAML (https://fr.wikipedia.org/wiki/YAML)

Comme nous allons le voir, le fichier « docker-compose.yml » va permettre de créer 2 conteneurs. Il contient tous les paramètres de connexion à la base de données, de l'application web (avec le *mapping* des ports) ainsi que les éléments permettant de créer les volumes persistants. Les images se basent sur des images disponibles sur le docker hub. Il est bien évidemment possible de modifier les valeurs pour les faire correspondre à l'infrastructure voulue.

```
1   version: '3'
2   services :
3     web :
4       image : ocsinventory/ocsinventory-docker-image:latest
5       container_name : ocsinventory-server
6       environment :
7           OCS_DBNAME : ocsweb
8           OCS_DBSERVER_READ : ocsinventory-db
9           OCS_DBSERVER_WRITE : ocsinventory-db
10          OCS_DBUSER : ocs
11          OCS_DBPASS : ocs
12      volumes :
13          - ocsdownload:/usr/share/ocsinventory-reports/ocsreports/download/
14          - ocssrv:/etc/ocsinventory-reports/
15          - ocslib:/var/lib/ocsinventory-reports/
16      links :
17          - db
18      ports :
19          - 80:80
20          - 443:443
21
22    db :
23      image : mysql:5.7
24      container_name : ocsinventory-db
25      restart: always
26      environment:
27          MYSQL_ROOT_PASSWORD : mdp
28          MYSQL_USER : ocs
29          MYSQL_PASSWORD : ocs
30          MYSQL_DATABASE : ocsweb
31      volumes :
32          - ./sql/:/docker-entrypoint-initdb.d/
33          - ocsdata:/var/lib/mysql
34      ports :
35          - 3306:3306
36
37  volumes:
38      ocsdata:
39          driver: local
40      ocssrv:
41          driver: local
42      ocslib:
43          driver: local
44      ocsdownload:
45          driver: local
```

**version** : il s'agit de la version de docker-compose utilisée (ici la version 3).

**services** : il est nécessaire de définir les services que l'on souhaite déployer et exécuter quand la commande ***docker-compose up*** est exécutée : ici 2 conteneurs **web** et **db. Ces noms seront automatiquement les noms de l'hôte du conteneur et sont utilisés pour les liens entre eux.**

**Description du conteneur « web » :**

- **utilisation de la dernière image officielle** présente sur le « docker hub » avec un nom de conteneur « ocsinventory-server » ;
- **utilisation de 5 variables d'environnement** qui vont permettre à l'application de se connecter à la base de données « ocsweb » pour pouvoir s'y connecter par la suite ;
- **création de 3 volumes** permettant l'utilisation de données persistances (équivalent à l'option -v de la commande « docker »). Ils sont créés (lorsqu'ils sont prévus en chemin relatif) dans « /var/lib/docker/volumes/ ». Par défaut, le nom prévu dans le fichier est préfixé par le nom du dossier dans lequel se trouve le fichier « yml » . Dans notre cas, 3 dossiers seront créés sur l'hôte « OCSInventory-Docker-Stack-master_ocsdownload », « OCSInventory-Docker-Stack-master _ocslib » et « OCSInventory-Docker-Stack-master _ocssrv » ;
- **liaison vers le service « db »** décrit plus loin (équivalent à l'option *link* permettant de lier les conteneurs entre eux) ;
- **mappage des ports** (équivalent à l'option -p de la commande « docker » qui permet une correspondance des ports entre le conteneur et la machine hôte. Les ports 80 (http) et 443 (https) sont utilisés à l'intérieur du conteneur Docker et sont également exposés (comportement à modifier si ces ports sont déjà utilisés sur l'hôte).Ici, les ports 80 et 443 doivent donc être utilisés pour se connecter à l'application Web.

**Description du conteneur « db » :**

- **utilisation de l'image** « mysql:5.7 » présente sur le « docker hub » avec un nom de conteneur « ocsinventory-db » ;
- **restart: always** : ce conteneur redémarrera automatiquement si pour une raison ou une autre il s'arrête ;
- **utilisation de 4 variables d'environnement** nécessaires à l'initialisation et l'utilisation des bases de données MariaDB. Le mot de passe de l'administrateur de MariaDB est défini à « mdp », la base « ocsweb » sera créée, de même que l'utilisateur « ocs » avec le mot de passe « ocs ».
- **création de 2 volumes :** un dossier « OCSInventory-Docker-Stack-master_ocsdata » sera créé dans « /var/lib/docker/volumes/ et « sql » dans le dossier où la commande est lancée (on peut constater le chemin absolu de la variable). Ce dernier volume accueille en fait le script d'initialisation de la base de données d'OCS. Le dossier « sql » est déjà présent dans l'archive avec le script *sql* « init.db.sql »)**.** L'image mysql:5.7 » prévoit, à la création du conteneur, le lancement du fichier « entrypoint.sh » (présent à la racine du conteneur) qui, lui-même, charge tous les scripts du répertoire /docker-entrypoint-initdb.d (mappé sur le dossier local « sql »).
- **mappage des ports** (équivalent à l'option -p de la commande « docker » qui permet une correspondance des ports entre le container et la machine hôte ) ; le port classique de MySQL 3306 de l'hôte sera redirigé vers le port 3306 du conteneur.

**Les volumes** créés dans « /var/lib/docker/volumes/doivent être déclarés dans une section à part.

De nombreux autres mots clés peuvent être utilisés dans un fichier yml. Pour en avoir le détail : https://docs.docker.com/compose/compose-file/

Lors d'une description d'un service, un autre mot clé utile est « **command** », il spécifie une ou plusieurs commandes qui seront exécutées lors du démarrage du conteneur.

### D2.2 : lancement des conteneurs

À partir du répertoire contenant le fichier « docker-compose.yml », il faut lancer la commande :
**docker-compose up -d**

L'option « -d » permet de prendre la main sur la console.

Ici, il est nécessaire de se déplacer dans le dossier « OCSInventory-Docker-Stack-master » qui contient le fichier docker-compose.yml.
Une fois ce dernier adapté aux besoins, il suffit donc de le lancer.

```
Creating network "OCSInventory-Docker-Stack-master_default" with the default driver
Creating volume "OCSInventory-Docker-Stack-master_ocsdata" with local driver
Creating volume "OCSInventory-Docker-Stack-master_ocssrv" with local driver
Creating volume "OCSInventory-Docker-Stack-master_ocslib" with local driver
Creating volume "OCSInventory-Docker-Stack-master_ocsdownload" with local driver
Pulling db (mysql:5.7)...
5.7: Pulling from library/mysql
177e7ef0df69: Extracting [>
cac25352c4c8: Download complete
...
Creating ocsinventory-db …
Starting ocsinventory-db … done
Creating ocsinventory-server …
Starting ocsinventory-server ... done
```

Au premier lancement, Il faut attendre un peu (ou beaucoup selon le débit de la connexion Internet) que les images soient téléchargées.

**root@servDocker:~# docker ps**

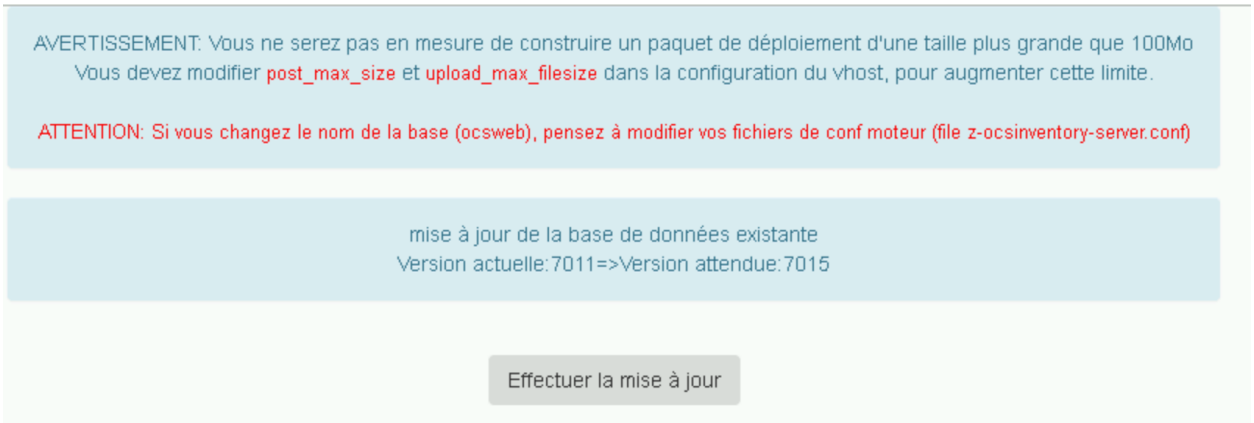
| CONTAINER ID | IMAGE | … PORTS | ...NAMES |
|---|---|---|---|
| eea3ff83560c | ...docker-image:latest | 0.0.0.0:80→80/tcp,0.0.0.0:443→443/tcp | ocsinventory-server |
| 052d047b2dfb | mysql:5.7 | 0.0.0.0:3306->3306/tcp, 33060/tcp | ocsinventory-db |

docker-compose accepte de nombreuses commandes (voir document 3) permettant de gérer les conteneurs (redémarrer les services, stopper/supprimer les conteneurs, etc.). Ces commandes doivent être exécutées dans le dossier contenant le fichier « yml ».

**root@servDocker:~# docker-compose ps**

| Name | Command | State | Ports |
|---|---|---|---|
| ocsinventory-db | docker-entrypoint.sh mysqld | Up | 0.0.0.0:3306->3306/tcp, 33060/tcp |
| ocsinventory-server | /bin/bash /root/run.sh | Up | 0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp |

On accède à ocsinventory-ng avec l'URL : http://adresseIP_serveurDocker/ocsreports/ (une petite mise à jour est proposée et nécessaire).



**Nous constatons également que :**

- les bases de données ne seront pas supprimées si les conteneurs sont eux-mêmes supprimés (tant que l'on ne supprime pas les volumes, bien évidemment). Elles se trouvent bien dans /var/lib/docker/volumes/root_ocsdata/_data/ :
  **ls /var/lib/docker/volumes/OCSInventory-Docker-Stack-master_ocsdata/_data/**
  auto.cnf ca.pem client-key.pem ibdata1 ib_logfile1 **mysql** **performance_schema** public_key.pem server-key.pem ca-key.pem client-cert.pem ib_buffer_pool ib_logfile0 ibtmp1 **ocsweb** private_key.pem server-cert.pem sys
- deux commandes sont lancées à la création du conteneur « ocsinventory-server » : ***/bin/bash*** et **/root/run.sh**. Ce dernier script exécute, entre autre, le service « apache ». Ce lancement n'est pas programmé dans « docker-compose.yml » (via la mot clé « command ») mais directement dans l'image source (voir le dockerfile : https://hub.docker.com/r/ocsinventory/ocsinventory-docker-image/dockerfile ou la commande docker inspect ocsinventory/ocsinventory-docker-image:latest).

La commande docker-compose fonctionne de la même manière que la commande docker, sauf qu'elle réalise l'opération sur tous les conteneurs spécifiés dans le fichier docker-compose.yml

Les commandes « docker » classiques peuvent bien évidemment être utilisées. Par exemple :
**docker exec -it ocsinventory-server bash** ou **docker exec -it ocsinventory-db bash** pour accéder à une console du conteneur.
**docker logs ocsinventory-server** ou **docker logs ocsinventory-db** pour accéder aux logs.

Le fichier « docker-compose.yml » de base définit la structure de l'application. Il est possible de spécifier d'autres fichiers pour le surcharger en modifiant ou ajoutant des propriétés sur des services existants et aussi d'ajouter des services supplémentaires. Par défaut, Docker Compose lit les deux fichiers suivants :

- docker-compose.yml ;
- docker-compose.override.yml (optionnel).

Il est également possible d'utiliser des noms de fichiers différents avec l'option `-f, par exemple :
**docker-compose -f docker-compose.ocs.yml up**

## Document 3 - Commandes utiles

**docker-compose config** vérifie la configuration du fichier docker-compose.yml

**docker-compose up** crée les conteneurs et démarre les services décrits dans le fichier docker-compose.yml et ne rend pas la main.

**docker-compose up -d** démarre les services décrits dans le fichier docker-compose.yml et rend la main une fois que les services sont démarrés.

**docker-compose build** reconstruit les services avant de les lancer. Elle n'est utile que si l'on souhaite qu'un container utilise une configuration définie dans un Dockerfile. Pour utiliser une image existante (comme dans notre cas ici), il suffit d'utiliser la syntaxe image suivie du nom de l'image.

**docker-compose logs** retourne l'ensemble des logs des services depuis le dernier démarrage et rend la main.
**docker-compose logs <nom_d'un_service>** retourne les logs du service correspondant.

**docker-compose ps** affiche des informations sur les conteneurs créés.

**docker-compose logs -f** affiche les logs des services et continue à les « écouter » sans rendre la main.
**docker-compose logs -f <nom_d'un_service>** retourne les logs du service correspondant et continue à les « écouter » sans rendre la main.

**docker-compose stop** arrête l'ensemble des conteneurs.
**docker-compose stop** <nom_d'un_service> arrête le conteneur correspondant.

**docker-compose start** démarre l'ensemble des conteneurs.
**docker-compose start** <nom_d'un_service> démarre le conteneur correspondant.

**docker-compose restart** redémarre l'ensemble des services.
**docker-compose restart** <nom_d'un_service> redémarre le service.

**docker-compose down** stoppe et supprime l'ensemble des conteneurs.
**docker-compose down** <nom_d'un_service> stoppe le service et supprime le conteneur correspondant.

**docker-compose rm** supprime l'ensemble des conteneurs (le ou les conteneurs doivent avoir été arrêtés au préalable).
**docker-compose rm** <nom_d'un_service> supprime le conteneur correspondant (le conteneur doit avoir été arrêté au préalable).

**docker-compose exec <nom_d'un_service> bash** fournit une console bash au sein du conteneur correspondant.

**docker-compose run** exécute une commande dans un conteneur.