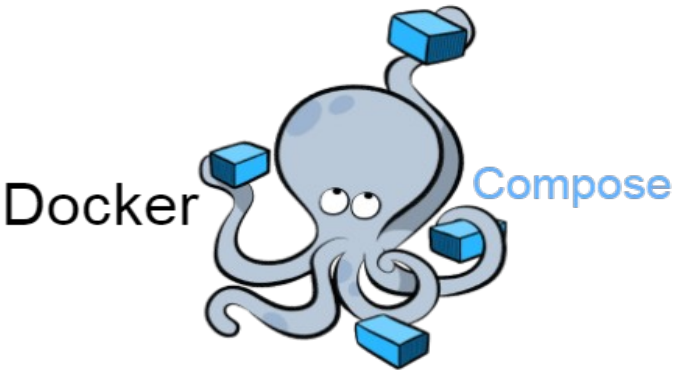
LAB n°4: Exploitation de Docker compose

Description du thème

Propriétés	Description
Intitulé long	Exploitation de Docker compose
Formation concernée	M2-Expert Réseaux
Matière	Cloud Computing
Présentation	Les objectif de ce Labo sont : -définir le comportement de vos conteneurs. -exécuter des applications Docker à conteneurs multiples. Ce Labo comporte 02 activités : -Activité 1 : Mise en place de docker compose. -Activité 2 : Découverte du driver virtualbox et des commandes Docker Machine
Prérequis	Commandes de base d'administration d'un système Linux.
Outils	• Un serveur physique ou virtuel avec une distribution Linux 64 bits (ici Centos 7– version stable actuelle). • Docker 18.03 ou supérieur. • Docker machine • docker compose Sites officiels : • https://www.docker.com/ • https://registry.hub.docker.com • https://docs.docker.com/
Mots-clés	Docker compose - fichier YAML
Auteur	Slim Marghli
Version	V1.0
Date de publication	Septembre 2020



Introduction

Docker Compose est un outil permettant de définir le comportement de vos conteneurs et d'exécuter des applications Docker à conteneurs multiples. La config se fait à partir d'un fichier **YAML**, et ensuite, avec une seule commande, vous créez et démarrez tous vos conteneurs de votre configuration.

Activité 1: Mise en place de docker compose

Docker Compose n'est pas installé par défaut et s'appuie sur le moteur Docker pour fonctionner. Au jour d'aujourd'hui, la dernière version de Docker Compose est la 1.27.3.

Voici la procédure à suivre pour télécharger Docker Compose sous un environnement Linux:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.27.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Vérifiez ensuite votre installation :

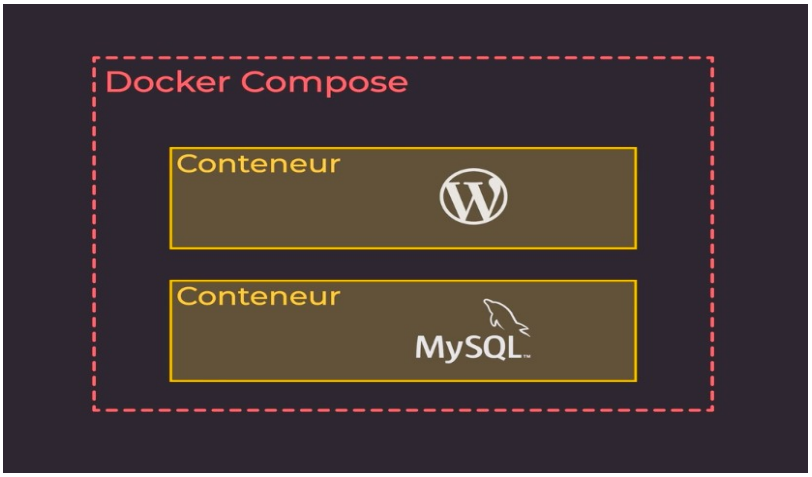
```
docker-compose --version
docker-compose version 1.27.3, build 4092ae5d
```

Activité n°2: exécuter des applications Docker à conteneurs multiples

Présentation de travail à faire

vous avez un nouveau projet de site avec **Wordpress**, et vous souhaitez simplifier la gestion de l'infrastructure. Vous devez maintenant réaliser un déploiement en production, où l'ensemble des composants sont dans des conteneurs Docker. Pour cela, vous allez avoir besoin de plusieurs composants :

- une base de données **MySQL**;
- le système de fichiers**Wordpress**.



Les besoins pour les conteneurs

Les besoins pour le conteneur de la base de données

On va débuter par la récolte des besoins du conteneur de la base de données. Pour celle-ci, il nous faudra un volume pour stocker les données.
En lisant la description de l'image Mysql sur Docker Hub, les informations qui nous captive sont ses variables d'environnements qu'on peut surcharger, notamment :

- **MYSQL_ROOT_PASSWORD**: spécifie le mot de passe qui sera défini pour le compte MySQL root (c'est une variable obligatoire).
- **MYSQL_DATABASE**: spécifie le nom de la base de données à créer au démarrage de l'image.
- **MYSQL_USER** et **MYSQL_PASSWORD**: utilisées conjointement pour créer un nouvel utilisateur avec son mot de passe. Cet utilisateur se verra accorder des autorisations de super-utilisateur pour la base de données MYSQL_DATABASE.

Les besoins pour le conteneur de Wordpress

Concernant le conteneur WordPress, nous aurons besoin d'une communication avec le conteneur de la base de données.

Lancer les conteneurs sans le docker-compose

Options utile de la commande **docker run**:

- -e: définit/surcharge des variables d'environnement
- --link: ajoute un lien à un autre conteneur afin de les faire communiquer entre eux.

Voici à quoi va ressembler la commande **docker run** pour la création du conteneur de la base de données et le conteneur de Wordpress:

```
docker run -d -e MYSQL_ROOT_PASSWORD='somewordpress' \
-e MYSQL_DATABASE='wordpress' \
-e MYSQL_USER='wordpress' \
-e MYSQL_PASSWORD='wordpress' \
--volume db-data:/var/lib/mysql \
--name db mysql:5.7
```

```
docker run -d -e WORDPRESS_DB_PASSWORD='wordpress' \
--link db:mysql -p 80:80
--name wordpress   wordpress
```

Dans cet exemple, on peut vite remarquer que les commande **docker run** sont assez longue. De plus, vous aurez à lancer ces commande pour chaque nouveau démarrage de l'application. Mais vous

aurez aussi à gérer vos différents conteneurs séparément. C'est pour ces raisons, que nous utiliserons le fichier **docker-compose.yml** afin de centraliser la gestion de nos multiples conteneurs d'une application Docker depuis un seul fichier. Dans notre cas il va nous permettre d'exécuter et définir les services, les volumes et la mise en relation des différents conteneurs de notre application.

Contenu du docker-compose

```
version: '3.3'


services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
volumes:
    db_data: {}
```

version: '3'
Un fichier docker-compose.yml commence toujours par l'argument **version** qui permet de spécifier à Docker compose quelle **version** on souhaite utiliser. Dans notre cas, nous utiliserons la version 3, qui est actuellement la version la plus utilisée.

Déclaration du premier service

```
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
```

services:
L'ensemble des conteneurs qui doivent être créés doivent être définis sous l'argument service. Chaque conteneur commence avec un nom qui lui est propre ; dans notre cas, notre premier conteneur se nomme **db**.

Image:
Cet argument nous permet de définir l'image Docker que nous souhaitons utiliser. Nous aurions pu aussi utiliser l'argument **build** en lui spécifiant le chemin vers notre fichier Dockerfile ; ainsi, lors de l'exécution de Docker Compose, il aurait construit le conteneur via le Dockerfile avant de l'exécuter.

Volumes:
Les conteneurs Docker ne sont pas faits pour faire fonctionner des services stateful, et une base de données est par définition un service stateful. Cependant, l'argument **volumes** nous a permis de stocker l'ensemble du contenu du dossier **/var/lib/mysql** dans un disque persistant. Et donc, de pouvoir garder les données en local sur notre host.
db_data est un volume créé par Docker directement, qui permet d'écrire les données sur le disque hôte sans spécifier l'emplacement exact. Vous auriez pu aussi faire un ***/data/*mysql:/var/lib/mysql** qui serait aussi fonctionnel.

restart: always
Les conteneurs étant par définition monoprocessus, si celui-ci rencontre une erreur fatale, il peut être amené à s'arrêter. Dans notre cas, si le serveur MySQL s'arrête, celui-ci redémarrera automatiquement.

environnement:
L'image MySQL fournie dispose de **plusieurs variables d'environnement** que nous pouvons utiliser ; dans votre cas, nous allons donner au conteneur les valeurs des différents mots de passe et utilisateurs qui doivent exister sur cette base.

Déclaration du second service

```
wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
```

depends_on:
Cet argument nous permet de créer une dépendance entre deux conteneurs. Ainsi, Docker démarrera le service db avant de démarrer le service Wordpress. Ce qui est un comportement souhaitable, car Wordpress dépend de la base de données pour fonctionner correctement.

ports:
Cet argument nous permet de dire à Docker Compose qu'on veut exposer un port de notre machine hôte vers notre conteneur, et ainsi le rendre accessible depuis l'extérieur.

```
volumes:
    db_data
```

Enfin, on demande au moteur Docker de nous créer un volume nommé db-data, c'est le volume pour stocker les données de la base de données.

Lancer wordpress

Placez vous au niveau du dossier qui contient le fichier docker-compose.yml. Ensuite lancez la commande suivante pour exécuter les services du docker-compose.yml:

```
docker-compose up -d
```

Ici l'option **-d** permet d'exécuter les conteneur du Docker compose en arrière-plan.
Si vous le souhaitez, vous pouvez vérifier le démarrage des conteneurs issus du docker-compose.yml

Pour seulement lister les conteneurs du docker-compose.yml, il suffit d'exécuter la commande suivante :

```
docker-compose ps
```

Pour avoir la liste des services, il suffit d'exécuter la commande suivante :

```
docker-compose ps --services
```

Si jamais vos conteneurs ne sont pas dans l'état UP, alors vérifiez les logs des services de votre Docker Compose en tapant la commande suivante :

```
docker-compose logs
```

Conclusion

Je pense que vous l'aurez compris, le Docker Compose est un outil permettant de faciliter la gestion des applications Docker à conteneurs multiples, comme :

- Démarrer, arrêter et reconstruire des services
- Afficher le statut des services en cours d'exécution
- Diffuser la sortie des logs des services en cours d'exécution
- Exécuter une commande unique sur un service
- etc …
-

Aide mémoire

```
## Exécuter les services du docker-compose.yml
docker-compose up
   -d : Exécuter les conteneurs en arrière-plan

## Lister des conteneurs du Docker Compose
docker-compose ls
   -a ou --all : afficher aussi les conteneurs stoppés

## Sorties/erreurs des conteneurs du Docker Compose
docker-compose logs
   -f : suivre en permanence les logs du conteneur
   -t : afficher la date et l'heure de la réception de la ligne de log
   --tail=<NOMBRE DE LIGNE> = nombre de lignes à afficher à partir de la fin pour chaque conteneur.
```

```
## Tuer les conteneurs du Docker Compose
docker-compose kill

## Stopper les conteneurs du Docker Compose
docker-compose stop
   -t ou --timeout : spécifier un timeout en seconde avant le stop
(par défaut : 10s)

## Démarrer les conteneurs du Docker Compose
docker-compose start

## Arrêtez les conteneurs et supprimer les conteneurs, réseaux,
volumes, et les images
docker-compose down
   -t ou --timeout : spécifier un timeout en seconde avant la
suppression (par défaut : 10s)

## Supprimer des conteneurs stoppés du Docker Compose
docker-compose rm
   -f ou --force : forcer la suppression

## Lister les images utilisées dans le docker-compose.yml
docker-compose images
```