Docker Compose : installation et pratique

1. Introduction et présentation

Docker Compose est une application Docker permettant de mettre en place des multi conteneurs sous forme de service. Jusqu'à présent, nous avons créé des conteneurs à l'aide de la commande docker run. Lorsqu'on veut définir un service faisant appel à plusieurs conteneurs, nous utiliserons docker compose. On définit ces différents conteneurs en créant un fichier YAML qui va contenir une configuration particulière, avec une seule commande, on créé et démarre tous les services à partir de la configuration : `docker-compose up`
Compose fournit des commandes pour gérer tout le cycle de vie des applications : démarrer, arrêter, reconstruire des services, télécharger les images, les mettre à jours, supprimer, …
Il permet aussi d'afficher l'état des services en exécution ainsi que fournir les journaux des services en cours d'exécution

2. Installation

Docker compose est déjà installé avec la version graphique que nous avons déployée sur notre Windows.
Pour Linux, il faudra l'installer manuellement.

- A partir du git de docker-compose, installez-le sur votre machine debian

3. Premier service

Pour activer un service, nous utiliserons la commande `docker-compose up`
Elle utilise par défaut un fichier nommé docker-compose.yml que nous stockerons dans un répertoire dédié

- Créez un dossier /app

- Rendez vous à l'intérieur et créez un fichier nommé docker-compose.yml

La syntaxe du fichier contient des mots clés suivis de ":"
Nous allons, dans ce premier exemple, créer un service web que nous nommerons serveur-web, basé sur la dernière image de nginx.

- Pour cela, remplissez le fichier ainsi :

```
services:
 web:
  container_name: serveur-web
  image: nginx:latest
```

- Sauvegardez, et vérifier la syntaxe de votre fichier avec la commande `docker-compose config`

- Que constatez vous ?

- Lancez votre conteneur avec la commande `docker-compose up -d`
- Vérifiez le lancement de votre conteneur avec la commande `docker-compose ps`

- Vérifiez à partir d'un navigateur si l'accès au nginx est possible. Expliquez le résultat
- Arrêtez le service avec `docker-compose stop`

- Puis supprimez le service avec `docker-compose rm`
- Vérifiez la suppression

- Dans votre fichier yml, modifier les lignes ainsi :

```
version: '2'

services:
  web:
    container_name: serveurweb
    image: nginx:latest
    ports:
      - '8080:80'
```

- Enregistrez et vérifiez la syntaxe
- Corrigez les éventuelles erreurs

- Revérifiez la syntaxe.

- Relancez votre conteneur. Que remarquez-vous concernant le réseau ?

```
docker-compose up -d

docker inspect serveurweb
```

- Vérifiez le lancement

- Vérifiez le lancement de votre serveur web dans le navigateur

- Vérifiez les réseaux docker, puis inspectez celui correspondant à votre serveur web.

```
docker network inspect app_default
```

- Identifiez l'adresse IP de votre conteneur puis tentez de le pinguer.

```
ping 172.20.0.2
```

- Stoppez et supprimer le conteneur

```
docker stop serveurweb && docker rm serveurweb
```

- Vérifiez la présence de votre réseau

```
docker network ls
```

- Modifiez votre fichier compose en y ajoutant un serveur de base de données (service db) mysql nommé serveur-db dont le port sera NAT sur le 3306

```
version: '2'

services:
  web:
    container_name: serveurweb
    image: nginx:latest
    ports:
      - '8080:80'

  db:
    container_name: serveurdb
    image: mysql:latest
    ports:
      - '3306:3306'
```

- Vérifiez votre config et lancez votre conteneur

- Effectuez un `docker-compose ps`, que remarquez vous concernant votre serveur-db ?

- Afin de trouver l'origine du problème, nous allons observer les logs avec la commande `docker-compose logs`

- Pourquoi votre service de base de données est-il arrêté ?

- Ajoutez la ligne suivante à votre fichier :

```
environment:
  -  MYSQL_ROOT_PASSWORD=root
```

- Comme votre serveur web est toujours en fonctionnement, nous pouvons uniquement relancer le service db en tapant la commande : `docker-compose up -d –no-deps db`

- Vérifiez avec un ps le fonctionnement de vos 2 services

- Inspectez le réseau, que remarquez-vous concernant les services web et db au niveau IP ?

- Exécutez un shell bash dans le conteneur web afin d'y installer le paquet iputils-ping (non présent dans l'image nginx de base)
  `docker-compose exec web /bin/bash`

- Depuis votre serveur web, tentez de ping le serveur db

- Sortez de votre conteneur (CTRL+P+Q)

- Effectuez la commande `docker-compose down`

- Quel est sa fonction ?

4. Build et docker compose

Il est possible de créez un service à partir d'un fichier yml qui appelle un dockerfile.

Pour cela, nous allons reprendre notre fichier concernant le dockerfile ping du TP précédent que nous allons stocker dans un dossier nommé /ping

```
FROM ubuntu:16.04
RUN apt-get update -y && apt-get install -y iputils-ping
ENTRYPOINT ["ping"]
CMD ["8.8.8.8"]
```

Puis nous allons créer un fichier docker-compose.yml ainsi :

```
version: '2'
services:
 ping:
  build: .
```

- Construisez votre image en lançant `docker-compose up`

5. Les volumes

Nous allons reprendre l'expérience tentée sur docker.

- Pour cela, lancez votre docker compose contenant le serveur web nginx, vérifiez la page d'accueil dans un navigateur, puis en se connectant par un bash sur le conteneur serveur-web, modifiez le fichier index.html et testez.

- Arrêtez et supprimez votre service

- Relancez votre docker compose, puis vérifiez votre navigateur

- Que pouvez vous en conclure ?

Tout comme pour docker classique, nous allons pouvoir effectuer de la persistance de données en stockant certains dossiers de nos services dans le disque de notre hôte. Nous allons pour cela créer des volumes.

- Créez dans votre dossier app, un sous dossier html

- Créez à l'intérieur un fichier index.html contenant le message d'accueil de votre choix

- Afin de mettre en place une synchronisation, ajoutez les lignes suivantes à votre service web :

```
volumes:
  -   ./html/:/usr/share/nginx/html
```

- Relancez votre service et vérifiez le bon fonctionnement dans votre navigateur

- Modifiez en direct votre fichier index.html

6. La gestion des réseaux

Lorsque la directive services est appelée dans un fichier docker-compose.yml, un réseau par défaut est créé qui portera le nom du dossier dans lequel se trouve le fichier .yml suivi de _default

Un DNS sera effectif sur le réseau permettant de nommer les machine non pas par leur IP, mais par leur nom de service (web, db, …)

- Créez un fichier yml ainsi :

```
version: '2'
services:
 web:
  image: nginx
  ports:
 -  8080:80
volumes:
 -  web-file:/usr/share/nginx/html
```

```
 db:
  image: mysql
  ports:
 -   3306:3306
  volumes:
 -   db-file:/var/lib/mysql
environment:
 -   MYSQL_ROOT_PASSWORD=root
 ping:
  build: .
volumes:
 web-file: {}
 db-file: {}
```

- Créez par la même occasion un dockerfile comme ceci :

```
FROM ubuntu
RUN apt update && apt install -y iputils-ping
COPY ./entrypoint.sh /usr/bin
RUN chmod +x /usr/bin/entrypoint.sh
CMD /usr/bin/entrypoint.sh
```

- Enfin créez le script entrypoint.sh

```
while :
do
 ping -c 5 db && ping -c 5 web
done
```

- Testez et concluez
- Détruisez vos services et vos réseaux
- Vérifiez
- Dans votre fichier yml, créez un réseau spécifique pour les services web et db uniquement (pas pour le ping). Le mot clé est `networks` :

Nommez le webnet, faites en sorte qu'il ait un driver de type bridge et une ip 172.20.20.0/24

- Lancez votre service
- Quels sont les réseaux créés ? pourquoi ?
- Vérifiez le ping dans la log
- Qu'en concluez vous ?
- Inspectez les réseaux et validez

- Ajoutez ce réseau au ping et retestez le bon fonctionnement

- Créez un nouveau réseau nommé dbnet que vous affecterez au service db. Il aura pour ip 172.21.0.0/24

- Affectez les 2 réseau webnet et dbnet au service ping

- Testez

- Exécutez un bash dans le conteneur ping, installez les net-tools, et vérifiez ses adresses IP