TP-Conteneurisation-1ere partie

Master IAO, Dept. Informatique

1ere partie : installation de docker sur la machine virtuelle UBUNTU

Comme première étape on se connecte par le compte root pour faciliter la tâche de l'installation, ainsi on execute les trois fichiers d'installation : container, client, serveur.

Pour installer Docker CE, on change le chemin pour se positionner sur le dossier ou on a telechargé le package Docker.

Pour telecharger les fichiers on utilise la commande suivante :

apt – get install docker

root@ubuntuserver #curl -O https://download.docker.com/linux/ubuntu/dists/bionic/pool/stable/amd64/Nom_Fichier

ensuite on dezippe la package par la commande dpkg :

root@ubuntuserver # sudo dpkg -i /path/to/package.deb

ainsi on execute les trois packages d'installation:

pour le container on a l'instruction :

root@ubuntuserver # containerd.io_1.2.4-1_amd64.deb

puis on installle la partie client par :

root@ubuntuserver # docker-ce-cli_18.09.3_3-0_ubuntu-bionic_amd64.deb

et on ajoute le serveur par:

root@ubuntuserver# docker-ce_18.09.3_3-0_ubuntu-bionic_amd64.deb

un fois le DockerEngine installé, on lance le Docker et on le met dans le boot du systeme :

root@ubuntuserver# systemctl start docker && systemctl enable docker

pour verifier le statut du service de docker :

root@ubuntuserver# systemctl status docker

on verifie l'installation de docker par la commande:

root@ubuntuserver# sudo docker run hello-world

(sudo en cas de compte utilisateur)

Cette commande télécharge l'image hello-world et l'execute locallement dans notre cas on va charger cette image par la commande load avec l'option –input pour le fichier tar :

root@ubuntuserver#docker load --input hello-world.tar

une autre option de chargement d'archivage est le symbole < :

root@ubuntuserver#docker load < hello-world.tar.gz

dans l'autre sens, on peut sauvegarder les images, afin des les charges ulterieurement par la commande load :

root@ubuntuserver# docker save hello-world > hello-world.tar

root@ubuntuserver# ls -sh hello-world.tar

xenon on utilize la commande save avec l'option --output :

root@ubuntuserver# docker save --output hello-world.tar hellow-world

et tester l'existence de l'archivage par :

root@ubuntuserver# ls -sh hello-world.tar

une option equivalente a output est l'option -o :

root@ubuntuserver# docker save -o hello-world.tar hellow-world

root@ubuntuserver# docker save -o hellow-world.tar hello-world:latest

(une autre alternative au save –o est le symbole >)

root@ubuntuserver# docker hello-world:latest > hellow-world.tar

Afin de lister tous les images installes dans notre environnement, on utilise la commande :

root@ubuntuserver# docker images

de la meme façon on charge busybox et debian dans notre machine virtuelle, et on liste les images.

Une fois les images en place, on demarre le container debian et busybox :

root@ubuntuserver# docker run debian

Un container ne reste en vie que si un processus est actif. On peut lister les containers actifs avec la commande docker ps. On peut aussi lister tous les containers, actifs ou inactifs avec docker ps –a.

root@ubuntuserver# docker ps

root@ubuntuserver# docker ps -a

Nous allons maintenant rediriger l'entrée standard du container avec l'option -i et ouvrir un pseudo-terminal avec -t, le tout en exécutant le processus /bin/bash.

root@ubuntuserver:~# docker run -ti --name=debian debian /bin/bash

par cette commande le prompt commande change pour refleter qu'on travaille sur un container

root@d9b100f2f636:/#

le container créé possede un identifiant `d9b100f2f636`. Cet identifiant est employé quon veut detruire un container.

Une fois qu'on est à l'interieur du container on peut executer n'importe quel commande.

A titre d'exemple on peut mettre à jour le package de la base de donnees du contenaire.

root@d9b100f2f636:/# apt update
comme on peut instlaller n'importe quel application, à titre d'exemple on installe python dans le container d9b100f2f636::

root@d9b100f2f636:/# apt install python

on arrete le container debian par la commande :

root@ubuntuserver:~# docker stop debian

comme on peut sortir du container par la commande exit.

on liste les containers actifs et inactifs par :

root@ubuntuserver:~# docker ps –a (qu'est ce que vous remarquez ?)

root@ubuntuserver:~#docker start debian

et on liste les containers une nouvelle fois :

root@ubuntuserver:~# docker ps –a (qu'est ce que vous remarquez ?)

root@ubuntuserver:~# docker attach debian #

(parfois il faut appuyer sur une touche du clavier pour avoir le prompt)

Pour tuer notre container :

root@ubuntuserver:~# docker rm debian

root@ubuntuserver:~# docker run -ti --name=jilali debian /bin/bash

On peut inspecter ce qui se passe dans le container depuis la machine hôte avec la commande, essayer sur un autre terminal de taper :

root@ubuntuserver:~# docker stats jilali (sur un autre terminal)

root@ubuntuserver:~# docker logs –f jilali (sur un autre terminal)

root@ubuntuserver:~# docker stats jilali (sur un autre terminal)

Pour supprimer une image locale afin de libérer de l'espace disque, on a besoin de supprimer tous les conteneurs qui référence l'image en utilisant la commande docker rm, suivie par soit l'ID du conteneur soit par le nom du conteneur.

L'id ou le nom du conteneur on peur l'extraire par la commande docker ps –a.

//**********//

2éme partie : installation de hadoop dans le container docker

De la meme façon qu'en premier paragraphe, on va charger l'image de Hadoop sous format fichier tar et ensuite on va installer son container.

Tout d'abord on copie le fichier hadoop-docker.tar dans le dossier local

root@ubuntuserver:~# mv /mnt/hgfs/Shared\ Files\hadoop-docker.tar .

root@ubuntuserver:~# docker load   < hadoop-docker.tar

on verifie si l'image est chargé correctement :

root@ubuntuserver:~# docker images

Apres avoir charge le fichier on remarque qu'il y a ete attribué un TAG qu'on va utiliser pour l'execution, ce qui va créer une container Docker ou hadoop2.7.0 va s'executer :

root@ubuntuserver:~# docker run -it sequenceiq/hadoop-docker:2.7.1 /etc/bootstrap.sh   --bash

Maintenant que le contenaire docker a demarré , executer jps pour voir si les services hadoop sont installés et executes.

bash-4.1# jps

on ouvre un nouveau terminal pour voir les listes des containers qui s'executent

root@ubuntuserver:~#   docker ps

Retournons à notre termina du contenaire docker, et on execute les commandes dessous pour avoir une adresse ip du contenaire docker.

bash-4.1# ifconfig

apres avoir executé la commande jps, on va verifier le namenode dans le browser par l'adresse 172.17.0.2 :500701, et ainsi on va trouver le namenode du cluster hadoop qui tourne dans contenaire docker.

Pour completer le cluster hadoop on va mettre en œuvre un exemple d'execution avec MapReduce :

bash-4.1# cd $HADOOP_PREFIX

bash-4.1# bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input output 'dfs[a-z.]+'

//**********//

3eme partie : installation de Docker-Compose

Docker-Compose est utilisé pour exécuter plusieurs contenaires comme un seul service. Par exemple supposons qu'on a une application qui nécessite NGNIX et MySQL, ainsi on peut créer un seul fichier qui lance les deux contenaires comme service sans avoir à demarrer chacun separement.

Dans cette 3eme partie du TP on verra comment lancer Docker-Compose, ensuite on va verra coment demarrer service simple avec MySql et NGNIX et l'executer en utilisant Docker Compose.

Pour installer docker-compose on a la commande suivante :

root@ubuntuserver:~# sudo apt install docker-compose

sinon on telecharge le package de docker-compose avec une version:

root@ubuntuserver:~# curl -L "https://github.com/docker/compose/releases/download/N°_VERSION/dockercompose -$(uname -s) -$(uname -m)" -o /usr/local/bin/docker-compose
ensuite on donnera les permissions:

root@ubuntuserver:~# sudo chmod +x /usr/local/bin/docker-compose

verifier l'installation de docker –compose

root@ubuntuserver:~# docker-compose --version

//**********//

4 éme partie: Execution d'un contenaire de hello-world avec docker-compose

Premierement on cree le fichier de configuration yaml dans un repertoire :

root@ubuntuserver:~# mkdir hello-world

root@ubuntuserver:~# cd hello-world

root@ubuntuserver:~# nano docker-compose.yml

inserer le nom de l'image dans le ficher yml:

```
version : '2'

services:

  my-test:

     image:  hello-world
```

La première ligne du fichier YAML est utilisé comme une partie du nom du conteneur. La 2eme ligne specifie quel image est utilisé pour créer le conteneur. Quand on exécute la commande docker-compose up il va apparaître à l'image local par le nom spécifié hello-world. Dans le répertoire créé hello-world on lance docker-compose.

```
root@ubuntuserver:~#    docker-compose up
```

La première fois on exécute la commande, si il n'ya pas d'image locale qui a pour nom hello-world, alors Docker-Compose va le charger du dépôt Docker Hub. Apres le chargement de l'image, docker-compose crée le conteneur, attache et exécute le programme hello, qui confirme que l'installation marche.

Le conteneur Docker s'exécute seulement si la commande est active, une fois hello a fini son exécution le conteneur est arête. Par conséquent quand nous affichons les processus actives, le conteneur hello-world ne sera pas listé parce qu'il ne tourne pas.

```
root@ubuntuserver:~# docker ps
```

Par ailleurs on peut afficher les informations sur le conteneur, en utilisant l'option –a qui affiche tous les conteneurs, non seulement les conteneurs actives.

```
root@ubuntuserver:~# docker ps -a
```

//**********//

5éme partie : installation du serveur nginx et la base de donnees mysql avec docker-compose

Dans cette partie on va créer un serveur web nginx et un systeme de gestion de base de données **mysql**. La base de données et le web sont utilisés pour définir deux services en utilisant les mots clés **database** et **web**. Le premier va executer le contenaier qui comporte le SGBD **mysql** et le 2eme contenaire notre serveur web **nginx.**

Le mot clé **image** est utilisé pour specifier l'image du **dockerhub** pour nos conteneurs **mysql et nginx.**

Pour la base de données, on utilise les mots-clés ports pour mentionner les ports qui ont besoin d'être exposé pour mysql. On spécifie ensuite les variables d'environnement nécessaires pour exécuter mysql.

La configuration des contenaires est decrite dans le fichier `docker-compose.yml` dans le repertoire local.

root@ubuntuserver:~# vim docker-compose.yml

```
version: "2"
services:
    databases:
      image: mysql
      ports:
      - "3306:3306"
      environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=demodb
    web:
      image: nginx
```

On execute la commande pour construire les conteneurs dans le repertoire local.

root@ubuntuserver:~# Docker-compose up

une fois execute, toutes les images vont etre telechargé et les coneteneurs vont demarrere automatiquement .

on peut verifier l'execution par les commandes docker-ps ou docker ps –a dans un autre terminal.

Pour se connecter à mysql dans le conteneur on utilise la commande :

root@ubuntuserver:~# docker exec -it nginxmysql_databases_1 mysql -h localhost -P 3306 -u root –ppassword

- le localhost fait référence au serveur de base de données dans ce cas c'est un serveur local.

- Le port par défaut du serveur est 3306 et on le retrouve par le drapeau –P, et le mot de passe par -p et nom de l'utilisateur est root par -u.

on retrouve le prompt de mysql:

mysql> show databases ;

ainsi à l'interieur du conteneur on peut manipuler le SGBD mysql avec tous les objets(base de donnees, tables, triggers etc….)

//**********//

Exercice : installation de Nginx et phpmyadmin et connexion avec la base de donnees mysql par l'outil Docker-Compose.