TP/TD Docker

Jean-Marc Pouchoulon

Avril 2024

1 Compétences à acquérir lors du TP.

Compétence principale:

— "Containériser" des logiciels pour la mise en œuvre de projets au cours de votre apprentissage.

Savoir:

— Connaître les différences et l'articulation entre la containérisation et la virtualisation.
— Distinguer les containers systèmes des containers applicatifs.
— Comprendre la philosophie de Docker (Continuous delivery, infrastructure as code...).
— Comprendre les briques de bases de la containérisation.
— Décrire une architecture en microservices s'appuyant sur des clusters de containers.

La validation sera faite à l'aide d'un QCM.

Savoir-faire:

— Instanciez un container et le manipuler.
— Rendre persistent des données au travers d'un volume.
— Encapsuler une commande dans un container.
— Utiliser un registry Docker de production.
— Créer un registry pour sa production personnelle.
— Dockériser une application.
— Lier des containers entre eux à l'aide de docker-compose et scaler une application.
— Mettre en réseau un container Docker de différentes façons.
— Débugguer a container Docker.

2 Pré-requis, recommandations et notation du TP.

Les pré-requis sont les suivants:

— Avoir un PC sous Linux.

— Avoir installé Docker ou utiliser une OVA prête à l'emploi. Merci **de ne pas** utiliser les packages fournis par les distributions qui sont souvent moins récents que les packages fournis par Docker.

— Avoir installé docker-compose, il est présent sur les OVA à votre disposition.

— Vous devez avoir un compte sur le site Docker: `https://hub.docker.com/`.

Vous travaillerez individuellement. Il vous explicitement demandé de faire valider votre travail par l'enseignant. Ces "checks" permettront de vous noter. Un compte rendu succinct (fichiers de configuration , copie d'écran montrant la réussite de votre construction ...) est demandé et à rendre sur Moodle.

2.1 Installation de Docker et obtenir de l'aide.

2.1.1 Changements à réaliser sur votre VM

Relancez docker:

```
systemctl reload docker
```

2.1.2 Rappel: Installation de Docker sous Linux.

Vous travaillerez avec une VM en utilisant l'OVA Debian sur `http://store.iutbeziers.fr`

2.1.3 Aide sur Docker.

```
man docker-run
man docker-create
```

Accéder à la Documentation Docker:
`https://docs.docker.com/`
Documentation sur les commandes Docker:
`https://docs.docker.com/engine/reference/commandline/`
La complétion avec la touche tab fonctionne aussi.

3 Docker sous Linux.

3.1 Installation de Docker sous Linux

L'installation est faite sur la machine virtuelle mais vous pouvez retrouver la procédure d'installation sous `https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/`

1. Quelle la version de Docker installée? Retrouvez des informations sur le daemon docker.

2. Vérifiez que votre installation fonctionne bien avec la commande :

   ```
   docker run hello-world
   ```

   a) Que vous explique le retour de cette commande (au delà de "tout s'est bien passé" reformulez en Français..) ?

   b) Retrouvez sur `https://hub.docker.com/` l'image hello-world.

   c) Expliquez les mécanismes en jeux pour la création du container helloworld. Quel est le fichier sur DockerHub qui permet de créer le container ?

3. Recherchez les images officielles Debian à l'aide de docker search. Récupérez-les ainsi que les images officielles busybox (une distribution légère).

4. Créez votre premier container à partir de l'image Debian officielle et en utilisant la commande "docker run -d debian" sans argument.

5. En utilisant la commande "docker ps" vérifiez que le container est "vivant" ? expliquez ?
6. Relancez le "docker run" en lui donnant comme argument bash -c "while :; do echo "coucou" ; sleep 1; done".
7. Stoppez et redémarrez le container.
8. Supprimez le container.
9. Utilisez les options "-it" afin d'être dans le container après son lancement.
10. Même opération mais nommant le container et son hostname DebianOne.
11. Détachez-vous du container DebianOne puis rattachez-vous à lui de nouveau.
12. Lancez un processus bash supplémentaire dans le container DebianOne. Pour celà utilisez la commande docker exec.
13. Listez le container restant. Ne listez ensuite que le dernier ContainerId.
14. Utilisez un volume pour donner à votre container l'accès à un répertoire de l'hôte. Quels sont les avantages de l'utilsation d'un volume ? un inconvénient ? A l'aide de la commande docker volume affichez les volumes présents sur votre hôte.
15. Supprimez le container et son image.
16. Supprimez tous les containers avec un oneliner sous bash. Idem pour les images.
17. Supprimez les images et les containers non utilisés avec la commande "docker system prune".

4 Création d'images Docker

Dans cette partie nous allons apprendre à créer une image Docker.

Récupérez les fichiers pour cet exercice via git:

```
git clone https:///github.com/pushou/tpdocker.git
```

4.1 Build d'une image Docker Debian

1. Construisez l'image "debian:vosinitiales" à partir du Dockerfile du repository et de la commande "docker build..."
2. Expliquez ce que font les différentes commandes "RUN, ENV, FROM" de ce Dockerfile.
3. Quel est l'intérêt de faire tous les apt-get en une seule fois pour la taille de l'image Docker. (indice: voir AUFS et Docker).
4. A partir de l'image "debian:vosinitiales" générez une image "pingfour" qui permettra de lancer un container de type ping se limitant à 4 envois ICMP vers www.iutbeziers.fr par défaut. Vous utiliserez les commandes "ENTRYPOINT" et "CMD" dans le Dockerfile.
5. Lancez un container issu de cette image au travers de la commande "docker run –rm -it ...". A quoi sert le –rm ?
6. Peux-t-on changer la destination du ping ? Le nombre de ping ?
7. Utilisez l'option entrypoint de "docker run" pour changer la commande ping par traceroute.
8. Transformez votre container en image en le "comittant" via la commande "docker commit...".
9. Créez un projet dont le nom sera de la forme "prenom.nom" sur registry.iutbeziers.fr)
10. Utilisez la commande "docker tag" pour générer une image registry.iutbeziers.fr/votre-prenom.votre-nom/adet-ping à partir de l'image comittée précedemment. Poussez cette image sur votre namespace généré précédemment vers le registry de l'IUT de Béziers.
11. Récupérez l'image de votre voisin via un docker pull sur le registry mis en place par votre enseignant. Instanciez-la afin de vérifier qu'elle fonctionne.

4.2 Installation d'un "insecure registry" sur votre poste de travail

En suivant `https://docs.docker.com/registry/insecure` installez un registry sur votre VM et testez-le. L'installation d'un certificat n'est pas demandée.

4.3 Création d'un Dockerfile afin de générer une image debian ssh

Créez un Dockerfile afin de générer un container fournissant un serveur SSH. Vous utiliserez l'image registry.iutbeziers.fr/debianiut comme image de base.

Vous instancierez cette image sous forme d'un container accessible en ssh sur le port 2222. Le container permettra l'authentification sur le compte root.

Indication: Utilisez chpasswd pour saisir le mot de passe root du container lors de son build.

4.4 Dockérisation d'une application Python

4.4.1 Sans le container lancez l'appliquette suivante fonctionnant avec Python3

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return "Le Python c'est bon mangez en"

if __name__ == '__main__':
        app.run(debug=True,host='0.0.0.0')

```

Lancez l'appli via:

```
pip3 install flask
export ENV FLASK_APP=app.py
export ENV FLASK_ENV=development
FLASK_APP=app.py  flask run
```

4.4.2 "Dockérisez" cette application

Dockérisez cette application en créant un Dockerfile et en utilisant une image python 3 (base Debian) [1]. L'application doit être accessible sur le port 9999 de l'hôte.

4.4.3 Création d'un docker-compose pour une application en "micro-services"

Il est possible de démarrer plusieurs containers afin de former un ensemble applicatif cohérent.

Pour cela il faut utiliser un fichier docker-compose.yml qui sera utilisé par la commande docker-compose afin de démarrer cet ensemble cohérent de containers.

1. Modifiez l'application flask précédente de façon à afficher l'IP du container sur l'uri /whoami. Recréez une image.

```
from flask import Flask,jsonify, request
import os

app = Flask(__name__)

```

1. `https://hub.docker.com/_/python`

```
@app.route('/')
def hello_world():
    return "Le Python c'est bon mangez en\n"

@app.route('/whoami')
def get_tasks():
    ipv4 = os.popen('ip addr show eth0').read().split("inet ")[1].split("/")[0]
    return jsonify({'ip_hote': ipv4}), 200

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=5000)
```

2. Créez un fichier docker-compose.yml qui générera un ensemble contenant un container traefik (version 2+) et un container issus de notre application. Aidez-vous de `https://docs.traefik.io/`.
   Vous pouvez utilisez le domaine ni.io qui renvoie une ip privée lorsqu'on fait une requête (mon ip privée est ici 192.168.1.95):

```
>host 192.168.1.95.nip.io
192.168.1.95.nip.io has address 192.168.1.95

>host whoami.192.168.1.95.nip.io
whoami.192.168.1.95.nip.io has address 192.168.1.95
```

   Vous pouvez générer un certificat auto-signé avec mkcert [2]:

```
mkcert 192.168.1.95.nip.io "whoami.192.168.1.95.nip.io" "*.192.168.1.95.nip.io"
```

   Le fichier statique traefik.toml est le suivant:

```
[entryPoints]
  [entryPoints.web]
    address = ":80"
    [entryPoints.web.http.redirections.entryPoint]
      to = "websecure"
      scheme = "https"

  [entryPoints.websecure]
    address = ":443"

[api]
  dashboard = true

[providers.docker]
  watch = true
  network = "web"

[providers.file]
  filename = "traefik_dynamic.toml"
```

   Le fichier dynamique traefik_dynamic.toml est le suivant [3]

2. `https://github.com/FiloSottile/mkcert`
3. user admin, mot de pass admin encodé en base 64 avec htpasswd

```
[http.middlewares.simpleAuth.basicAuth]
  users = [
    "admin:$apr1$..Ql.Lsl$MWLpkfS026zCDTU6NOB9x0"
  ]

[http.routers]
[http.routers.api]
  rule = "Host(`192.168.55.134.nip.io`)"
  entrypoints = ["websecure"]
  middlewares = ["simpleAuth"]
  service = "api@internal"
  [http.routers.api.tls]

[[tls.certificates]]
        certFile = "/certs/192.168.1.95.nip.io+2.pem"
        keyFile = "/certs/192.168.1.95.nip.io+2-key.pem"
```

L'application whoami sera testée avec la commande suivante:

```
curl -H Host:whoami.192.168.1.95.nip.io http://whoami.192.168.1.95.nip.io
"Le Python c'est bon mangez en"
```

Il vous reste les certificats à générer et le fichier docker-compose à créer.

3. Utilisez l'option scale de docker-compose pour générer plusieurs containers afin d'équilbrer la charge. docker-compose up -d –scale monappy=3

5 Réseaux Docker

1. Listez les réseaux Docker présents sur votre hôte.
2. Créez un réseau bridge supplémentaire.
3. Créez deux nouveaux containers en les rattachant à ce nouveau réseau NAT.Expliquez comment le container accède au réseau de la salle.
4. Créez un nouveau container en le rattachant à un réseau macvlan.
5. Créez un nouveau container en le rattachant à un réseau ipvlan.
6. Expliquez l'utilité des différents types de réseaux.
7. Quelle est la chaine de résolution DNS utilisée par le container ? Comment changer le DNS au run ?

6 Tips & Tricks

6.1 Connexion à distance au daemon Docker

Docker est bâtie sur une architecture modulaire qui lui permet de se connecter à un daemon Docker sur une machine distante en TLS ou SSH.

1. Créez un contexte pour vous connectez à distance au daemon Docker de votre VM (Vous pouvez utiliser une autre VM Linux ou installer Docker)

```
docker context create ubuntuvm --docker "host=ssh://student@VOTRE_IP"
```

2. Vérifier le bon fonctionnement du contexte en créant un container sur la VM.

6.2 Débugguer un container

Les containers sont optimisés pour être les plus légers possibles et n'embarquent pas en général pas d'outils facilitant l'analyse. Voilà une façon de les débugguer (sur une idée de J. Petazzoni) [4]:

1. Générer une instance Apache. Cette instance ne contient pas d'éditeur ni de d'outils réseaux (iproute2/ifconfig)

```
docker run  -d --rm  httpd
```

2. Télécharger un binaire statique busybox [5]
3. Récupérez l'Id du container afin de copier le binaire busybox.

```
docker cp ./busybox id_du_container:/
```

4. Vous pouvez maintenant utiliser les commandes de busybox pour analyser votre container.

```
docker exec -it 1edd81a917315bf  /busybox ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
84: eth0@if85: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fd00::242:ac11:4/80 scope global flags 02
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:4/64 scope link
       valid_lft forever preferred_lft forever
```

5. Installez nsenter [6].
6. Trouvez le pid d'un process dans le container :

```
docker inspect id-du-container|grep -i pid
```

7. Utilisez nsenter pour retrouver les NameSpaces du container et lancez les commandes de votre hôte dans le container.

```
nsenter --target PID_trouvé_précedemment -p -u -n -i
```

4. `https://www.youtube.com/watch?v=-DsegUFcENc`
5. voir https://www.busybox.net/downloads/binaries
6. `https://github.com/jpetazzo/nsenter`