Emma PAVELAK

TP1

4. Initialiser un nouveau repository git qui vous permettra de sauvegarder les fichiers créés pendant le TP.

```
root@Ubuntu:/home/emma/docker-TP1# git init .
Reinitialized existing Git repository in /home/emma/docker-TP1/.git/
```

5. Exécuter un serveur web (apache, nginx, …) dans un conteneur docker

a. Récupérer l'image sur le Docker Hub

```
root@Ubuntu:/home/emma/docker-TP1# docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
3f9582a2cbe7: Already exists
9423d69c3be7: Already exists
d1f584c02b5d: Already exists
758a20a64707: Pull complete
08507f82f391: Pull complete
Digest: sha256:76618ddd53f315a1436a56dc84ad57032e1b2123f2f6489ce9c575c4b280c4f4
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

b. Vérifier que cette image est présente en local

```
root@Ubuntu:/home/emma/docker-TP1# docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
httpd         latest    daab1fa13f86   17 hours ago    145MB
httpd         <none>    b304753f3b6e   7 days ago      145MB
hello-world   latest    feb5d9fea6a5   17 months ago   13.3kB
```

c. Créer un fichier index.html simple

```
root@Ubuntu:/home/emma/docker-TP1# touch index.html
root@Ubuntu:/home/emma/docker-TP1# nano index.html
```

d. Démarrer un conteneur et servir la page html créée précédemment à l'aide d'un volume (option -v de docker run)

```
root@Ubuntu:/home/emma/docker-TP1# docker run -d -p 8080:80 -v $(pwd)/index.html
:/usr/local/apache2/htdocs/index.html --name mon-serveur-web httpd
de29a034ffe9f010995d8faf883d1d892f4206412d45f419aedf68e13dc25c06
```

e. Supprimer le conteneur précédent et arriver au même résultat que précédemment à l'aide de la commande docker cp

```
root@Ubuntu:/home/emma/docker-TP1# docker stop mon-serveur-web
docker rm mon-serveur-web
mon-serveur-web
mon-serveur-web
```

```
root@Ubuntu:/home/emma/docker-TP1# docker run -p 80:80 -d httpd
78b4707a594b750def5f5297fd27c439eee4d1d1898d9184b30edfa51bbe448c
```

On copie l'index.html dans le conteneur :

```
root@Ubuntu:/home/emma/docker-TP1# docker cp index.html 78b4707a594b:/usr/local/
apache2/htdocs/
Preparing to copy...
Copying to container - 2.048kB
Successfully copied 2.048kB to 78b4707a594b:/usr/local/apache2/htdocs/
```

6. Builder une image

a. A l'aide d'un Dockerfile, créer une image (commande docker build)

```
root@Ubuntu:/home/emma# touch dockerfile
```

Contenu du Dockerfile :

```
FROM nginx:latest
COPY index.html /usr/share/nginx/html
```

```
root@Ubuntu:/home/emma/docker-TP1# docker build -t monimage:latest .
[+] Building 4.2s (7/7) FINISHED
 => [internal] load build definition from dockerfile                       0.0s
 => => transferring dockerfile: 94B                                        0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load metadata for docker.io/library/nginx:latest            1.6s
 => [internal] load build context                                          0.0s
 => => transferring context: 199B                                          0.0s
 => [1/2] FROM docker.io/library/nginx:latest@sha256:aa0afebbb3cfa473099a  2.1s
 => => resolve docker.io/library/nginx:latest@sha256:aa0afebbb3cfa473099a   0.0s
 => => sha256:942ae2dfd73088b54d7151a3c3fd5af038a51c50029 1.57kB / 1.57kB   0.0s
 => => sha256:9a8c6f28671867118799d40ec6748ffbd9eab3747 25.47MB / 25.47MB   1.0s
 => => sha256:e81b85700bc2530b1298885f74210138872f60cd139920b 626B / 626B  0.2s
 => => sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a7 1.86kB / 1.86kB   0.0s
 => => sha256:904b8cb13b932e23230836850610fa45dce9eb0650d 7.66kB / 7.66kB   0.0s
 => => sha256:73ae4d4511202a823260a789f97afb3a46b8c9172ae8d99 956B / 956B   0.2s
 => => sha256:3a1b8f20135650110763d223f82a25007e00b28b268 1.40kB / 1.40kB   0.3s
 => => sha256:6058e3569a68e825f1085592ab0bb04c29345726119ed05 772B / 772B   0.4s
 => => extracting sha256:9a8c6f28671867118799d40ec6748ffbd9eab3747503ca47   0.5s
 => => extracting sha256:e81b85700bc2530b1298885f74210138872f60cd139920b5   0.0s
 => => extracting sha256:73ae4d4511202a823260a789f97afb3a46b8c9172ae8d99d   0.0s
 => => extracting sha256:6058e3569a68e825f1085592ab0bb04c29345726119ed059   0.0s
 => => extracting sha256:3a1b8f20135650110763d223f82a25007e00b28b268e6a87   0.0s
 => [2/2] COPY index.html /usr/share/nginx/html                            0.5s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:32ce2f4903abcd1359d6787ac27fd21e4a59e01d7af6d  0.0s
 => => naming to docker.io/library/monimage:latest                         0.0s
```

b. Exécuter cette nouvelle image de manière à servir la page html (commande docker run)

```
root@Ubuntu:/home/emma/docker-TP1# docker run -d -p 90:8090 monimage:latest
e8461a426605378e6c629752a3dd6bc6b2dd1206713267300d6dc0d07b49d1a9
```

c. Quelles différences observez-vous entre les procédures 5. et 6. ?

La procédure 5 utilise un conteneur existant à partir de dockerHub alors que la 6 créer une nouvelle image à partir d'un docker file.

Avantages et inconvénients de l'une et de l'autre méthode ? (Mettre en relation ce qui est observé avec ce qui a été présenté pendant le cours)

Pour la procédure 5 :
Avantages :

Les images de serveur web sont facilement accessibles sur le Docker Hub.

L'utilisation d'un volume permet de séparer le code de l'application de l'image de l'application, ce qui facilite la maintenance et la mise à jour de l'application.

Inconvénients :

L'utilisation de volumes peut rendre la configuration de l'application plus complexe, en particulier si l'on utilise des volumes pour plusieurs applications.

Pour la procédure 6 :

Avantages :

La méthode est plus flexible que l'utilisation d'une image existante, car elle permet de personnaliser l'image selon les besoins de l'application.

Les images peuvent être créées de manière reproductible, ce qui facilite la mise à jour et la maintenance de l'application.

La méthode est portable, car l'image de l'application peut être déplacée facilement entre différents environnements.

Inconvénients :

La création d'une image peut être plus complexe et prendre plus de temps que l'utilisation d'une image existante. La maintenance est aussi plus complexe, car il est nécessaire de mettre à jour régulièrement le Dockerfile.

7. Utiliser une base de données dans un conteneur docker

a. Récupérer les images mysql:5.7 et phpmyadmin/phpmyadmin depuis le Docker Hub

```
root@Ubuntu:/home/emma/docker-TP1# docker pull mysql:5.7
5.7: Pulling from library/mysql
7b659169cb92: Pull complete
e47c3f06a3f5: Pull complete
e0656a27f56e: Pull complete
b9f33f34ef42: Pull complete
9fc5cbaf8704: Pull complete
5fb8407bef93: Pull complete
f8be4a2d031b: Pull complete
b6cb2bff25e3: Pull complete
056cc4a5c89a: Pull complete
842896973144: Pull complete
e55b6e95b292: Pull complete
Digest: sha256:9202fc6bc8fa63615e6bfc0049fc660f712d164220c5c54d86519870c305ea48
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

```
root@Ubuntu:/home/emma/docker-TP1# docker pull phpmyadmin/phpmyadmin
Using default tag: latest
latest: Pulling from phpmyadmin/phpmyadmin
01b5b2efb836: Pull complete
45244a9928d1: Pull complete
139d4815e950: Pull complete
9a420fd884ad: Pull complete
1de46a46cfcd: Pull complete
9cc46e699e97: Pull complete
9a8d67ebc9db: Pull complete
f9464a3489e8: Pull complete
5fececf0d356: Pull complete
6dd28e697cc8: Pull complete
b38a56ab5f21: Pull complete
ca91e5a7ae8b: Pull complete
185f934955ec: Pull complete
a077202948ae: Pull complete
1f511389296f: Pull complete
335d8ddb8d9f: Pull complete
68f3e41ea3a9: Pull complete
a28d69dfa6d4: Pull complete
Digest: sha256:0d951ee3bec76c5d7083122f5db509ebfa6c209efc5e70f1d47af2e13a34f543
Status: Downloaded newer image for phpmyadmin/phpmyadmin:latest
docker.io/phpmyadmin/phpmyadmin:latest
```

b. Exécuter deux conteneurs à partir des images et ajouter une table ainsi que quelques enregistrements dans la base de données à l'aide de phpmyadmin

```
root@Ubuntu:/home/emma/docker-TP1# docker run -d --name mysql-db -e MYSQL_ROOT_PASSW
ORD=password mysql:5.7
6eebba2fb1445a21d6c3d3404440f878bee1b750e57f143501aed408746d69f3
```

```
root@Ubuntu:/home/emma/docker-TP1# docker run -d --name phpmyadmin-container --link
mysql-db:db -p 8080:80 phpmyadmin/phpmyadmin
5b4d22546b0390430e162ac6320d54bd10ba084e3b04f0cb8a8d00ca9f2b317f
```

8. Faire la même chose que précédemment en utilisant un fichier docker-compose.yml

```
root@Ubuntu:/home/emma/docker-TP1# touch docker-compose.yml
```

Contenu du docker-compose.yml :

```yaml
version: '3'

services:
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
      - "8080:80"
    links:
      - db
```

```
root@Ubuntu:/home/emma/docker-TP1# docker-compose up -d
docker-tp1_db_1 is up-to-date
Starting docker-tp1_phpmyadmin_1 ... done
```

a. **Qu'apporte le fichier docker-compose par rapport aux commandes docker run ? Pourquoi est-il intéressant ? (cf. ce qui a été présenté pendant le cours)**
   Le fichier docker compose permet de décrire les configurations de plusieurs conteneurs dans un seul fichier YML. Ce qui simplifie la gestion, la création et le déploiement dans le cas où il y a plusieurs conteneurs. 'docker run' lui ne permet de lancer qu'un seul fichier à la fois.

b. **Quel moyen permet de configurer (premier utilisateur, première base de données, mot de passe root, …) facilement le conteneur mysql au lancement ?**
c. **En utilisant l'option -e ici,on donne le user, le mot de passe et la database.**

```
root@Ubuntu:/home/emma/docker-TP1# docker run --name mysql -e MYSQL_ROOT_PASSWORD=passwo
L_DATABASE=database1 -d mysql:5.7
0c964040752db923bab19b51a70516b15fa826fbcb0cf4afa1e82c86edaafa85
```

   On peut aussi utiliser un fichier de configuration mysql, dans ce cas on lance la commande docker run de la façon suivante :
   docker run –name mysql -v /path/to/my.cnf:/etc/mysql/my.cnf -d mysql :5.7

9. Observation de l'isolation réseau entre 3 conteneurs

a. A l'aide de docker-compose et de l'image praqma/network-multitool disponible sur le Docker Hub créer 3 services (web, app et db) et 2 réseaux (frontend et backend). Les services web et db ne devront pas pouvoir effectuer de ping de l'un vers l'autre

Contenu du docker-compose :

```yaml
version: '3'
services:
  web:
    image: praqma/network-multitool
    networks:
      - frontend
    command: tail -f /dev/null
  app:
    image: praqma/network-multitool
    networks:
      - frontend
      - backend
    command: tail -f /dev/null
  db:
    image: praqma/network-multitool
    networks:
      - backend
    command: tail -f /dev/null
networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
```

```
root@Ubuntu:/home/emma/docker-TP1# docker-compose exec app ping db
root@Ubuntu:/home/emma/docker-TP1# docker-compose exec app ping web
root@Ubuntu:/home/emma/docker-TP1# docker-compose exec db ping web
OCI runtime exec failed: exec failed: unable to start container process: exec: "ping": executable file not found in $PATH: u
root@Ubuntu:/home/emma/docker-TP1# docker-compose exec web ping db
```

b. Quelles lignes du résultat de la commande docker inspect justifient ce comportement ?

docker inspect permet de récupérer des informations sur le conteneur tel que les réseaux auquels le conteneurs est connecté

```
root@Ubuntu:/home/emma/docker-TP1# docker-compose up -d
```

```
root@Ubuntu:/home/emma/docker-TP1# docker inspect docker-tp1_app_1
```

```
"Networks": {
    "docker-tp1_backend": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "90c5430965a0",
            "app"
        ],
        "NetworkID": "cfdfb37d79e3e94c61590cdcfe40d81431f8ea3d6f47c20899a3a5d767ab6cf1",
        "EndpointID": "472b60f8ea24845b0adb1f7ca8b459dba9e22f9042955232bddbbf48ea50bb03",
        "Gateway": "172.20.0.1",
        "IPAddress": "172.20.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:14:00:02",
        "DriverOpts": null
    },
```

```
"docker-tp1_frontend": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
        "90c5430965a0",
        "app"
    ],
    "NetworkID": "e3cf093b0fd9559ff0261fbbda328172f25615771565945e127802db6d1e6f4f",
    "EndpointID": "e0dbf04ecf3efa4ab701f14adac3518185ed92705ba1db55869e4d066d557614",
    "Gateway": "172.19.0.1",
    "IPAddress": "172.19.0.3",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "02:42:ac:13:00:03",
    "DriverOpts": null
}
```


d. Dans quelle situation réelles (avec quelles images) pourrait-on avoir cette configuration réseau ? Dans quel but ?

Dans le cas ou un service de base données serait connecté à un réseau privé mais également à un réseau public. De cette façon, avoir des services isolés permette aux utilisateurs d'accéder aux services nécessaires en toute sécurité.

Un autre cas serait un service web connecté à un réseau frontend pour communiquer avec un load balancer ainsi qu'un à un réseau backend pour communiquer avec une base de données.