IIT INSTITUT INTERNATIONAL TECHNOLOGIE Université Nord Américaine privée	Matière : Virtualisation et cloud computing Travaux pratiques	**SE :** Ubuntu 18.04 LTS
	TP 4. Stockage persistant, Dockerfile et Docker-compose	**AU.** 2021-2022 Classe : 3ème GLSI

I. Objectifs

- Utiliser un stockage persistant sur des conteneurs Docker
- Créer votre propre image docker à l'aide d'un fichier Dockerfile
- Installer la dernière version de Docker Compose sur Ubuntu 18.04 et explorer les concepts et commandes de base de Docker Compose

II. Mode opératoire

- Se connecter en tant qu'utilisateur avec les privilèges root.
- Avoir Docker installé

III. Ressources

- Paramètres indiqués par le formateur

IV. Persistance des données

Les conteneurs Docker ne stockent pas de données persistantes. Les données écrites dans la couche inscriptible d'un conteneur ne seront plus disponibles une fois que le conteneur aura cessé de fonctionner. En outre, il peut être difficile d'obtenir des données écrites dans un conteneur pour un autre processus. Pour résoudre le problème de la persistance des données d'un conteneur, Docker a deux options :

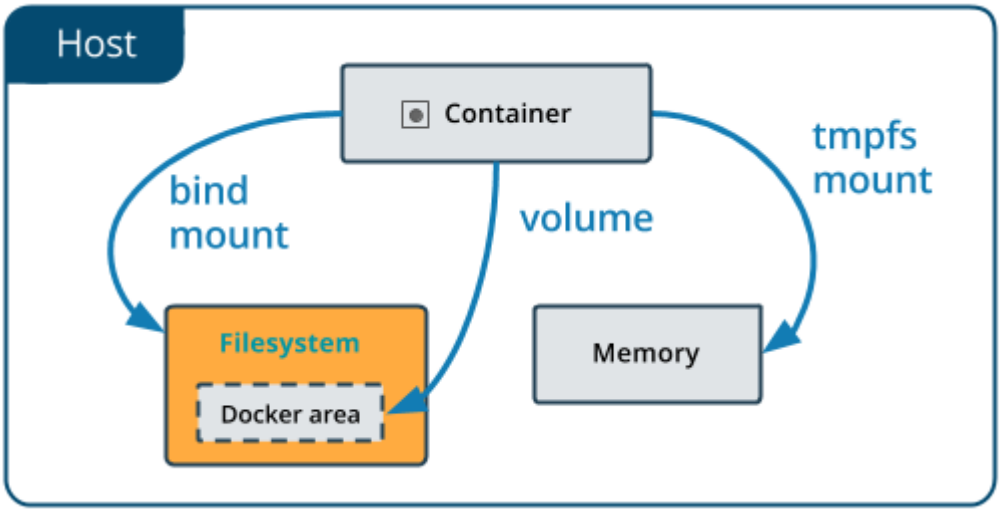
✓ **Montages Bind (bind mounts) :** un montage bind est un fichier ou un dossier stocké n'importe où sur le système de fichiers hôte du conteneur, monté dans un conteneur en cours d'exécution.

La principale différence entre un montage bind et un volume est que, puisqu'il peut exister n'importe où sur le système de fichiers hôte, les processus en dehors de Docker peuvent également le modifier. En revanche, lorsque vous utilisez un volume, un nouveau répertoire est créé dans le répertoire de stockage de Docker sur la machine hôte et Docker gère le contenu de ce répertoire.

Il n'est pas nécessaire que le fichier ou le répertoire existe déjà sur l'hôte Docker. Il est créé à la demande s'il n'existe pas encore.

Les montages bind sont très performants, mais ils reposent sur le système de fichiers de la machine hôte ayant une structure de répertoire spécifique disponible. Si vous développez de nouvelles applications Docker, envisagez plutôt d'utiliser des volumes nommés. Vous ne pouvez pas utiliser les commandes Docker CLI pour gérer directement les montages bind.

Si votre conteneur génère des données d'état non persistantes, envisagez d'utiliser un montage tmpfs pour éviter de stocker les données n'importe où de façon permanente et pour augmenter les performances du conteneur en évitant d'écrire dans la couche inscriptible du conteneur. Un montage tmpfs est donc temporaire et ne persiste que dans la mémoire de l'hôte. Lorsque le conteneur s'arrête, le montage tmpfs est supprimé et les fichiers qui y sont écrits ne seront pas conservés.




```
iset@ubuntu:~$ mkdir $HOME/webapp
iset@ubuntu:~$ docker run -itd --name nginx1 --mount type=bind,source=$HOME/webapp,target=/usr/share/nginx/html
 -p 8080:80 nginx:latest
fade4b5343386acb9c6785010774c6d5832444d74e7c64d365d9371a630fae97
```

Remarque 1 : Si vous effectuez un montage « bind » dans un répertoire non vide du conteneur, le contenu existant du répertoire est masqué par le montage bind. Cela peut être bénéfique, par exemple lorsque vous souhaitez tester une nouvelle version de votre application sans créer une nouvelle image. Cependant, cela peut également être surprenant et ce comportement diffère de celui des volumes docker.

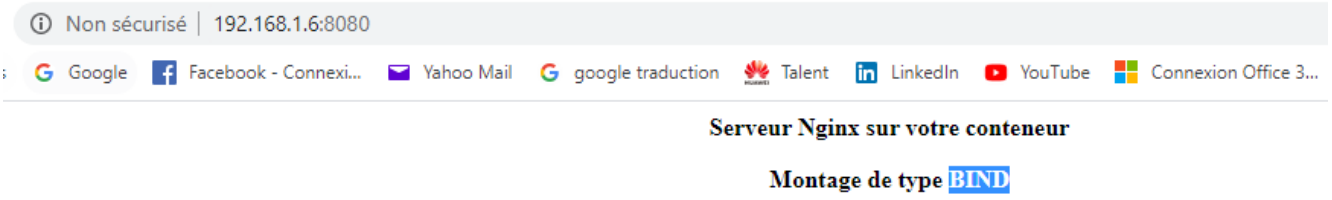
Remarque 2 : Le type du montage peut être « bind », « volume » ou « tmpfs ».

Pour tester ce montage, créer et modifiez le fichier index.html dans le répertoire webapp (vous allez donc modifier la page d'accueil de nginx située sous le répertoire « /usr/share/nginx/html » dans le conteneur):

```
iset@ubuntu:~$ nano webapp/index.html
```

```
  GNU nano 2.9.3                                webapp/index.html

<html>
<title>Test du montage de type BIND </title>
<body><center><b>
<p>Serveur Nginx sur votre conteneur</p>
<p>Montage de type BIND</p>
</b></center></body>
</html>
```

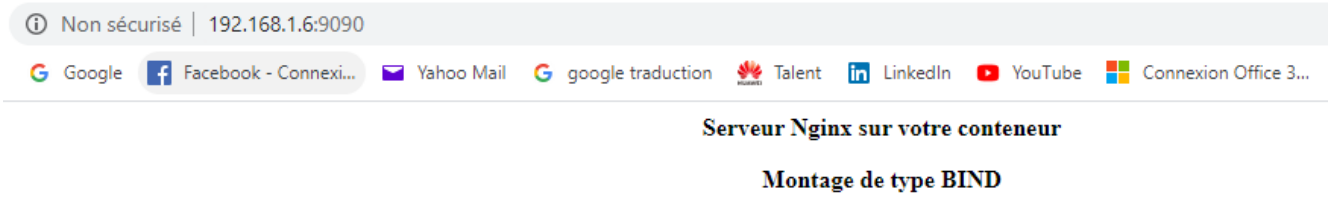
Testez la modification en se connectant sur le serveur nginx à partir de l'adresse IP de la machine hôte :



Maintenant, arrêtez et supprimer ce conteneur « nginx1 » et créer un nouveau conteneur nginx « nginx2 », effectuer le même type de montage sur le répertoire webapp et modifier le port d'écoute sur la machine hôte par 9090. (Notez que vous pouvez utiliser l'option -v (--volume) à la place de l'option --mount)

```
iset@ubuntu:~$ docker stop nginx1
nginx1
iset@ubuntu:~$ docker rm nginx1
nginx1
iset@ubuntu:~$ docker run -itd --name nginx2 -v $HOME/webapp:/usr/share/nginx/html -p 9090:80 nginx:latest
712512f7d2db9b0953668302afad823fc0a43cc027242a47104fd23963148415
iset@ubuntu:~$
```

Puis, testez la persistance des données en se connectant sur ce nouveau serveur nginx :



Utilisez la commande « docker inspect nginx2 » pour vérifier que le montage de type bind a été créé correctement.

```
iset@ubuntu:~$ docker inspect nginx2
```

Recherchez la section « Mounts » :

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/iset/webapp",
        "Destination": "/usr/share/nginx/html",
        "Mode": "",
        "RW": true,
        "Propagation": "rprivate"
    }
],
```

✓ **Volumes :** les volumes sont le moyen préféré pour stocker ou utiliser des données persistantes dans des conteneurs Docker. Les volumes sont entièrement gérés par Docker. Les volumes présentent plusieurs avantages par rapport aux montages bind :

- ❖ Les volumes sont plus faciles à sauvegarder ou à migrer que les montages bind.
- ❖ Vous pouvez gérer les volumes à l'aide des commandes CLI Docker ou de l'API Docker.
- ❖ Les volumes fonctionnent sur les conteneurs Linux et Windows.
- ❖ Les volumes peuvent être partagés de manière plus sûre entre plusieurs conteneurs.
- ❖ Les pilotes de volume vous permettent de stocker des volumes sur des hôtes distants ou des fournisseurs de cloud, pour crypter le contenu des volumes ou ajouter d'autres fonctionnalités.
- ❖ Les nouveaux volumes peuvent avoir leur contenu prérempli par un conteneur.

Contrairement à un montage bind, vous pouvez créer et gérer des volumes en dehors de la portée de n'importe quel conteneur.

Créez un volume :

```
iset@ubuntu:~$ docker volume create elyesvolume
elyesvolume
iset@ubuntu:~$
```

Liste des volumes :

```
iset@ubuntu:~$ docker volume ls
DRIVER              VOLUME NAME
local               2e98edc41feaad97dd01eac98bfd26e7378a268c46a50e988e094c4e46ab412e
local               9e6007c3f096c60527ead619ea6457d8741aac38515b449ba4eb216c8288639a
local               elyesvolume
local               f1b69ed346e4b121af176aa5551290c16fe8b5b4b4cb35e85d15bb6d0b376312
local               f1c89373d70229ea82b34a7acd3431a8670cdd8d5fd1003ff6f84ad9191271da
local               faff62ea5ade57a5e6051441cf1eba55eeb24dee6f1e1d17e9ad99ad6ddc3b70
iset@ubuntu:~$
```

Inspectez un volume :

```
iset@ubuntu:~$ docker volume inspect elyesvolume
[
    {
        "CreatedAt": "2020-04-10T09:51:26-07:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/elyesvolume/_data",
        "Name": "elyesvolume",
        "Options": {},
        "Scope": "local"
    }
]
iset@ubuntu:~$
```

Démarrer un conteneur avec un volume :

```
iset@ubuntu:~$ docker run -itd --name nginx3 --mount type=volume,source=elyesvolume,target=/usr/share/nginx/html -p 7070:80 nginx:latest
79254bdb164f618fe54d9f07887db5c29307e17d3ecc8641d2c1a802fd28205b
iset@ubuntu:~$
```

Pensez à consulter le section « Mounts » en inspectant ce conteneur avec la commande « docker inspect nginx3 ».

Si le conteneur a des fichiers ou des répertoires dans le répertoire à monter, le contenu du répertoire est copié dans le volume. Le conteneur monte ensuite et utilise le volume, et d'autres conteneurs qui utilisent le volume ont également accès au contenu prérempli.

```
iset@ubuntu:~$ sudo ls /var/lib/docker/volumes/elyesvolume/_data
[sudo] password for iset:
50x.html  index.html
iset@ubuntu:~$
```

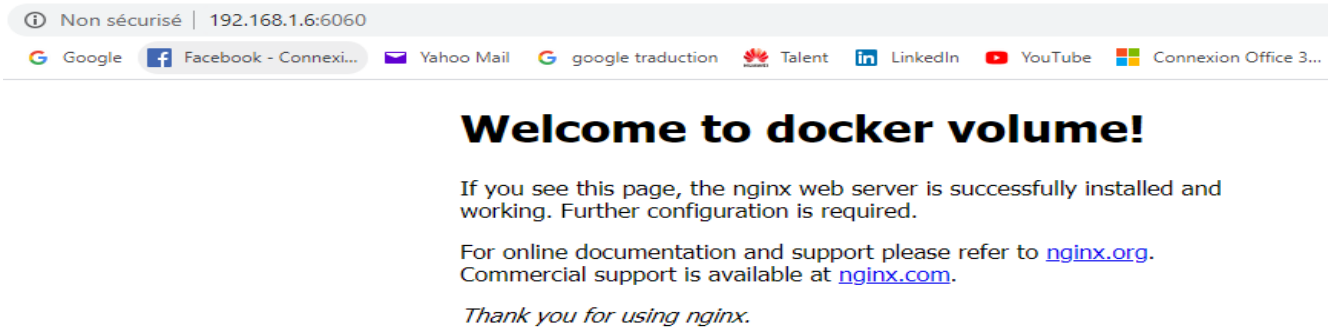
Vous pouvez modifier la page « index.html » dans le volume, supprimer le conteneur « nginx3 » et créer un autre « nginx4 » en montant le volume précédent.

```
iset@ubuntu:~$ sudo nano /var/lib/docker/volumes/elyesvolume/_data/index.html
```

```
 GNU nano 2.9.3                  /var/lib/docker/volumes/elyesvolume/_data/index.html

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to docker volume!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

```
iset@ubuntu:~$ sudo rm -f nginx3
iset@ubuntu:~$ docker run -itd --name nginx4 --mount type=volume,source=elyesvolume,target=/usr/share/nginx/htm
l -p 6060:80 nginx:latest
a80199c673ce63c0d5e9a83ad59197e1fe1cfccf12329b6a555905877218e14e
iset@ubuntu:~$
```

Vérifiez que la nouvelle page est hébergée par notre conteneur :



Remarque 3 : Si vous créer un nouveau conteneur en le montant sur un volume inexistant, ce volume sera créé automatiquement.

```
iset@ubuntu:~$ docker run -itd --name nginx5 --mount type=volume,source=elyesvolume1,target=/usr/share/nginx/ht
ml -p 5050:80 nginx:latest
```

```
iset@ubuntu:~$ docker volume ls
DRIVER              VOLUME NAME
local               2e98edc41feaad97dd01eac98bfd26e7378a268c46a50e988e094c4e46ab412e
local               9e6007c3f096c60527ead619ea6457d8741aac38515b449ba4eb216c8288639a
local               elyesvolume
local               elyesvolume1
local               f1b69ed346e4b121af176aa5551290c16fe8b5b4b4cb35e85d15bb6d0b376312
local               f1c89373d70229ea82b34a7acd3431a8670cdd8d5fd1003ff6f84ad9191271da
local               faff62ea5ade57a5e6051441cf1eba55eeb24dee6f1e1d17e9ad99ad6ddc3b70
```

Remarque 3 : L'option « readonly » (lecture seule), si elle existe, entraîne le montage dans le conteneur en lecture seule.

```
iset@ubuntu:~$ docker run -it --name elyes --mount type=volume,source=vol1,target=/bin,readonly ubuntu
root@05172751626f:/# ls /bin
bash      bzip2recover  dd            findmnt   lsblk         ps          sleep     uncompress   zforce
bunzip2   bzless        df            grep      mkdir         pwd         stty      vdir         zgrep
bzcat     bzmore        dir           gunzip    mknod         rbash       su        wdctl        zless
bzcmp     cat           dmesg         gzexe     mktemp        readlink    sync      which        zmore
bzdiff    chgrp         dnsdomainname gzip      more          rm          tar       ypdomainname znew
bzegrep   chmod         domainname    hostname  mount         rmdir       tempfile  zcat
bzexe     chown         echo          kill      mountpoint    run-parts   touch     zcmp
bzfgrep   cp            egrep         ln        mv            sed         true      zdiff
bzgrep    dash          false         login     nisdomainname sh          umount    zegrep
bzip2     date          fgrep         ls        pidof         sh.distrib  uname     zfgrep
root@05172751626f:/# mkdir /bin/ok
mkdir: cannot create directory '/bin/ok': Read-only file system
root@05172751626f:/#
```

V. Créer votre propre image à l'aide d'un fichier Dockerfile

Jusqu'à présent, je vous ai présenté le fonctionnement de base de Docker. Mais cela vous limitait à l'usage des images que vous pouviez trouver sur le Docker Hub. Afin de vraiment pouvoir utiliser Docker au maximum, il serait appréciable de pouvoir créer des images adaptées à nos projets et c'est là l'utilité des Dockerfiles.

Exemple 1 :

Les Dockerfiles sont des fichiers qui permettent de construire une image Docker adaptée à nos besoins, étape par étape. Pour commencer, créez un nouveau fichier Dockerfile à la racine de votre projet. La première chose à faire dans un Dockerfile est de définir de quelle image vous héritez. Pour cet exemple, je vous propose d'utiliser une image ubuntu comme base.

1. J'indique la distribution de départ avec la ligne

```
FROM ubuntu
```

2. Je nomme la personne qui a écrit ce Dockerfile, ici c'est moi. Grâce à la ligne

```
MAINTAINER Elyes Gassara <elyes.gassara@yahoo.fr>
```

3. Je recherche les paquets disponibles et j'installe Nginx.

```
RUN apt-get update \
   && apt-get install -y \
      nginx
```

4. Je copie successivement les configurations et scripts de mon système hôte vers mon image

```
COPY default /etc/nginx/sites-enabled

COPY index.html /var/www/html

COPY service_start.sh /home/docker/script/service_start.sh
```

5. J'applique les droits pour exécuter mon script

```
RUN chmod 744 /home/docker/script/service_start.sh
```

6. Je définie un point d'entrée : le premier script qui va se lancer au démarrage du container

```
ENTRYPOINT /home/docker/script/service_start.sh
```

7. Le dossier dans lequel je serai quand j'exécuterai un nouveau container sera WORKDIR

```
WORKDIR /home/docker
```

Nous avons toutes les lignes pour faire notre Dockerfile. Le voilà en version complète :

```
FROM ubuntu

MAINTAINER Elyes Gassara <elyes.gassara@yahoo.fr>

RUN apt-get update \
    && apt-get install -y \
        nginx

COPY index.html /var/www/html
COPY default /etc/nginx/sites-enabled
COPY service_start.sh /home/docker/script/service_start.sh

RUN chmod 744 /home/docker/script/service_start.sh

ENTRYPOINT /home/docker/script/service_start.sh
WORKDIR /home/docker
```

Contenu du fichier default :

```
 GNU nano 4.8                                default
server {
        listen 2080 default_server;
        listen [::]:2080 default_server;

        root /var/www/html;

        index index.html index.htm;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Contenu du fichier index.html :

```
 GNU nano 2.9.3                          index.html

<html><title>Test Dockerfile</title>
<body><center><b>Test Dockerfile</b></center></body>
</html>
```

Contenu du script service_start.sh :

```
 GNU nano 2.9.3                                          service_start.sh

#!/bin/bash
echo Bonjour mes amis
service nginx restart
/bin/bash
```

8. Construire une image depuis un Dockerfile

Maintenant que nous avons un Dockerfile, nous voulons créer notre image. Et la commande est : « docker build ».

```
iset@ubuntu:~/exemple1$ docker build -t elyesnginx .
Sending build context to Docker daemon   5.12kB
Step 1/9 : FROM ubuntu
 ---> 4e5021d210f6
Step 2/9 : MAINTAINER Elyes Gassara <elyes.gassara@yahoo.fr>
 ---> Running in 5416f5b51a8c
Removing intermediate container 5416f5b51a8c
 ---> fe67d3fd2776
Step 3/9 : RUN apt-get update     && apt-get install -y         nginx
 ---> Running in 6bbfca6915ee
Get:1 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
```

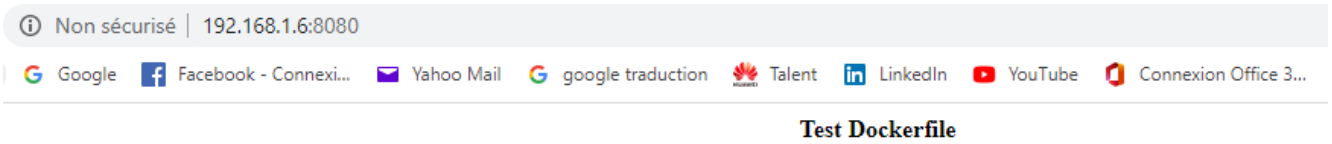
Lorsque vous exécutez « docker build », vous devez spécifier le chemin du Dockerfile (d'où le point à la fin de la commande si vous lancez la commande depuis le même endroit).

Durant la construction vous allez voir défiler toutes les étapes s'exécuter les unes après les autres. Le premier lancement est long, car aucune des étapes n'a été déjà appelée. Lors d'un second build, vous verrez que les étapes sont expédiées à grande vitesse. Tout ça à condition que les étapes précédentes n'aient pas été modifiées dans votre Dockerfile bien sûr.

Lancez un conteneur à partir de cette image comme suit :

```
iset@ubuntu:~/exemple1$ docker run -it --name elyesnginx1 -p 8080:2080 elyesnginx
Bonjour mes amis
 * Restarting nginx nginx                                                                                    [ OK ]
root@58cedd6c35d6:/home/docker# ss -anltp
State       Recv-Q       Send-Q              Local Address:Port              Peer Address:Port
LISTEN      0            128                       0.0.0.0:2080                   0.0.0.0:*          users:(("nginx",pid=29,fd=6))
LISTEN      0            128                          [::]:2080                      [::]:*          users:(("nginx",pid=29,fd=7))
root@58cedd6c35d6:/home/docker#
```

Effectuer une requête sur votre serveur web depuis la machine hôte :



Test Dockerfile

Exemple 2 :

Dans cet exemple, nous allons créer ensemble une image Docker, dans laquelle nous allons installer Node.js, ainsi que les différentes dépendances de notre application.

Fichier **package.json** :

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "Elyes Gassara <elyes.gassara@yahoo.fr>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

Fichier **server.js** :

```
'use strict';
const express = require('express');
// Constantes
const PORT = 8080;
const HOST = '0.0.0.0';
var os = require("os");

// App
const app = express();
app.get('/', (req, res) => {
  res.send('<center>Formation présentée par Elyes Gassara<br>Application web <font color="0000FF"><b>version 2</b> </font><br> Conteneur : <b>' + String(os.hostname()));
});
app.listen(PORT, HOST);
console.log(String(os.hostname()));
```

Fichier **Dockerfile** :

```
FROM node:14

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

Maintenant on utilise la commande « docker build » pour créer une image « nodejs-http-server ».

```
$ docker build -t nodejs-http-server .
```

Après, je vais lancer un conteneur que je vais nommer « http » à partir de cette image :

```
$ docker run -d --name http -p 80:8080 nodejs-http-server
```

Test de notre serveur web :

Formation présentée par Elyes Gassara
Application web **version 2**
Conteneur : **30b7a1d2d924**

VI. Docker Compose

Docker Compose est un outil qui vous permet de définir et de gérer des applications Docker contenant plusieurs conteneurs. Il utilise un fichier YAML pour configurer les services, les réseaux et les volumes de l'application. Les déploiements d'applications à hôte unique, les tests automatisés et le développement local sont les cas d'utilisation les plus courants de Docker Compose.

Le package d'installation de Docker Compose est disponible dans les référentiels officiels d'Ubuntu 18.04, mais il ne s'agit peut-être pas toujours de la version la plus récente. L'approche recommandée consiste à installer Docker Compose à partir du référentiel GitHub de Docker.

Avant de télécharger le fichier binaire, visitez la page de publication du référentiel Compose sur GitHub et vérifiez si une nouvelle version est disponible au téléchargement.

Pour installer Docker Compose sur Ubuntu 18.04, procédez comme suit :

- Téléchargez le fichier binaire Docker Compose dans le répertoire /usr/local/bin à l'aide de la commande curl suivante :

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.0.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

- Une fois le téléchargement terminé, appliquez des autorisations exécutables au binaire Compose :

```
elyes@tunisie:~$ sudo chmod +x /usr/local/bin/docker-compose
```

- Vérifiez l'installation en exécutant la commande suivante qui affichera la version de Compose :

```
elyes@ubuntu:~/exemple2/my_app$ docker-compose --version
Docker Compose version v2.0.1
```

VII. Démarrer avec Docker-compose

Dans cette section, nous montrerons comment utiliser Docker Compose pour configurer une application WordPress contenant plusieurs conteneurs sur Ubuntu 18.04.

Commencez par créer un répertoire de projet et naviguez dans celui-ci :

```
elyes@tunisie:~$ mkdir my_app
elyes@tunisie:~$ cd my_app/
elyes@tunisie:~/my_app$
```

Lancez votre éditeur de texte et créez un fichier nommé docker-compose.yml dans le répertoire du projet :

```
elyes@tunisie:~/my_app$ gedit docker-compose.yml
```

Mettez le contenu suivant :

```
version: "3.9"

services:
  db:
    image: mysql
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: elyespasswd
      MYSQL_DATABASE: wordpress
      MYSQL_USER: elyes
      MYSQL_PASSWORD: elyespass

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
```

```
    - "8000:80"
  restart: always
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_USER: elyes
    WORDPRESS_DB_PASSWORD: elyespass
    WORDPRESS_DB_NAME: wordpress
volumes:
 db_data: {}
 wordpress_data: {}
```

Analysons le code ligne par ligne :

Dans la première ligne, nous spécifions la version du fichier Compose. Il existe plusieurs versions du format de fichier Compose prenant en charge des versions spécifiques de Docker.

Ensuite, nous définissons deux services, db et wordpress. Chaque service exécute une image et crée un conteneur distinct lorsque docker-compose est exécuté.

Le service db :

- Utilise l'image mysql:latest. Si l'image n'est pas présente sur le système, elle sera extraite du référentiel public de Docker Hub.
- Utilise la stratégie de redémarrage qui demandera au conteneur de toujours redémarrer.
- Crée un volume nommé db_data pour rendre la base de données persistante.
- Définit les variables d'environnement pour l'image mysql:latest.

Le service wordpress :

- Utilise l'image wordpress. Si l'image n'est pas présente sur votre système, Compose l'extraira du référentiel public de Docker Hub.
- Utilise la stratégie de redémarrage qui demandera au conteneur de toujours redémarrer.
- Monte le volume wordpress_data sur l'hôte au répertoire /var/www/html à l'intérieur du conteneur.
- Transfère le port exposé 80 du conteneur au port 8000 de la machine hôte.
- Définit les variables d'environnement pour l'image wordpress.
- L'instruction depends_on définit la dépendance entre les deux services. Dans cet exemple, db sera démarré avant wordpress.

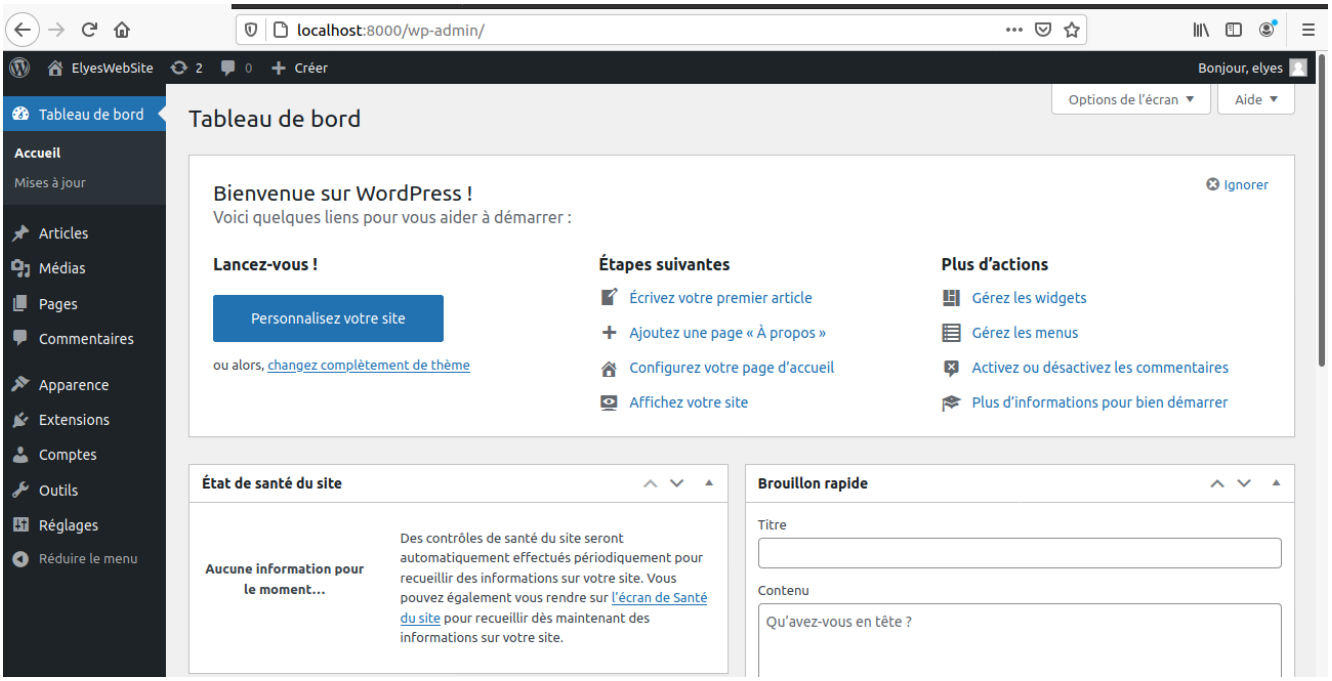
À partir du répertoire du projet, démarrez l'application WordPress en exécutant la commande suivante :

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

Compose extrait les deux images et démarre deux conteneurs.

Entrez http://0.0.0.0:8000/ dans votre navigateur et vous verrez l'écran d'installation de Wordpress.

À ce stade, l'application Wordpress est opérationnelle et vous pouvez commencer à travailler sur votre thème ou votre plugin.



Pour vérifier les services en cours, utilisez l'option ps :

```
elyes@ubuntu:~/my_app$ docker-compose ps
NAME                COMMAND                  SERVICE             STATUS              PORTS
my_app-db-1         "docker-entrypoint.s…"   db                  running             33060/tcp
my_app-wordpress-1  "docker-entrypoint.s…"   wordpress           running             0.0.0.0:8000->80/tcp, :::8000->80/tcp
```

Pour arrêter l'utilisation des services lorsque Compose est exécuté en mode détaché :

```
elyes@ubuntu:~/my_app$ docker-compose stop
[+] Running 2/2
 ⠿ Container my_app-wordpress-1  Stopped                                    1.4s
 ⠿ Container my_app-db-1         Stopped                                    1.9s
```

Si vous souhaitez supprimer entièrement les conteneurs, utilisez l'option down :

```
elyes@ubuntu:~/my_app$ docker-compose down
[+] Running 3/3
 ⠿ Container my_app-wordpress-1  Removed                                    0.0s
 ⠿ Container my_app-db-1         Removed                                    0.0s
 ⠿ Network my_app_default        Removed                                    0.1s
```

Passer l'option -v supprimera également les volumes de données :

```
elyes@ubuntu:~/my_app$ docker-compose down -v
[+] Running 2/2
 ⠿ Volume my_app_db_data         Removed
 ⠿ Volume my_app_wordpress_data  Removed
```

Si, pour une raison quelconque, vous souhaitez désinstaller Docker Compose, vous pouvez simplement supprimer le fichier binaire en tapant :

```
elyes@tunisie:~/my_app$ sudo rm /usr/local/bin/docker-compose
```

VIII. APPLICATION:

On va maintenant améliorer notre serveur web crée précédemment avec nodejs en utilisant une base de données pour stocker le nombre de visite.

Fichier **package.json** :

```
{
  "name": "sample_nodejs_with_redis",
  "version": "1.0.0",
  "description": "Node.js and redis on Docker",
  "author": "Elyes Gassara <elyes.gassara@yahoo.fr>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.13.3",
    "redis": "^2.2.5"
  }
}
```

Fichier **server.js** :

```
'use strict';

const express = require('express'),
      http = require('http'),
      redis = require('redis'),
      os = require('os');

const client = redis.createClient(6379, 'redis');
const app = express();

app.get('/', function(req, res, next) {
  client.incr('visits', function(err, visits) {
    if(err) return next(err);
    const response = '<center><br> <font color="0000FF">Formation animée par Elyes Gassara
</font><br>Cette requête a été traitée par le conteneur <b>' + os.hostname() + ' </b><br> Vous avez
vu cette page <b> <font color="0F000F">' + visits + ' fois.</font></b> <br>';

    res.send(response);
  });
});

const appPort = 8080;
http.createServer(app).listen(appPort, function() {
  console.log('Listening on port ' + appPort);
});
```

Fichier **Dockerfile** :

```
FROM node:5.10-slim
MAINTAINER Elyes Gassara <elyes.gassara@yahoo.fr>

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
```

```
RUN npm install

COPY server.js /usr/src/app/

EXPOSE 8080
CMD [ "npm", "start" ]
```

Fichier **docker-compose.yml** :

```
version: "3"
services:
  http:
   image: nodejs-http-server
   ports:
     - "80:8080"
   depends_on:
     - redis
   networks:
     - frontend
     - backend

  redis:
   image: redis:alpine
   networks:
     - backend
   volumes:
     - elyesvolume:/data
networks:
 frontend:
 backend:
volumes:
 elyesvolume:
```

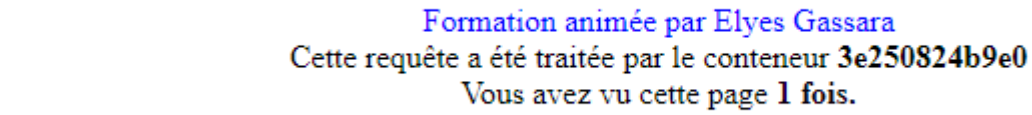
Commençons par créer l'image nodejs-http-server :

```
$ docker build -t nodejs-http-server .
```

Lançons maintenant l'application avec docker-compose :

```
elyes@ubuntu:~/application$ docker-compose up -d
[+] Running 7/7
 ⠿ redis Pulled
   ⠿ a0d0a0d46f8b Pull complete
   ⠿ a04b0375051e Pull complete
   ⠿ cdc2bb0f9590 Pull complete
   ⠿ 0aa2a8e7bd65 Pull complete
   ⠿ f64034a16b58 Pull complete
   ⠿ 7b9178a22893 Pull complete
[+] Running 5/5
 ⠿ Network application_frontend       Created
 ⠿ Network application_backend        Created
 ⠿ Volume "application_elyesvolume"   Created
 ⠿ Container application-redis-1      Started
 ⠿ Container application-http-1       Started
elyes@ubuntu:~/application$
```

Test de l'application :

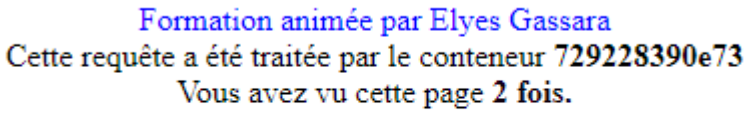


Supprimer puis lancer les conteneurs pour tester la persistance des données :

```
elyes@ubuntu:~/application$ docker-compose down
[+] Running 4/4
 ⠿ Container application-http-1    Removed
 ⠿ Container application-redis-1   Removed
 ⠿ Network application_backend     Removed
 ⠿ Network application_frontend    Removed
elyes@ubuntu:~/application$
```

```
elyes@ubuntu:~/application$ docker-compose up -d
[+] Running 4/4
 ⠿ Network application_frontend    Created
 ⠿ Network application_backend     Created
 ⠿ Container application-redis-1   Started
 ⠿ Container application-http-1    Started
elyes@ubuntu:~/application$
```

Notez que le nombre de visite est maintenant égale à deux car la première visite a été stockée dans le volume elyesvolume :



IX. Conclusion

Vous avez appris à utiliser un stockage persistant pour les conteneurs docker, créer votre propre image docker avec l'utilisation du fichier Dockerfile et enfin installer et utiliser Docker Compose sur Ubuntu 18.04. L'utilisation de Docker Compose peut considérablement améliorer votre flux de travail et votre productivité. Vous pouvez définir votre environnement de développement avec Docker Compose et le partager avec les collaborateurs du projet.

Bibliographie

https://linuxize.com/post/how-to-install-and-use-docker-compose-on-ubuntu-18-04/

https://docs.docker.com/storage/bind-mounts/

https://docs.docker.com/storage/volumes/