TD 1 : Docker

Définition

Le Docker est un conteneur visant à répondre aux propriétés des micro services : il isole les environnements du système. On a par conséquent un Docker par micro service (également appelé processus). Chaque processus interagit directement avec l'OS. L'outil docker a pour rôle la gestion d'un ensemble de conteneurs. Il peut s'agir de multiples conteneurs mais aussi d'images.

Partie 1 : Initialisation

1)

Une première étape consiste tout d'abord à vérifier le bon fonctionnement des commandes suivantes :

```
docker --version
docker-compose --version
docker-machine --version
```

Au début les commandes n'existent pas, il faut les installer. Par exemple :

Sudo apt install docker.io

Sudo apt install docker-compose

On peut à présent vérifier la présence des exécutables ci-dessous :

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN$ docker --version
Docker version 17.12.1-ce, build 7390fc6
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN$ docker-compose --version
docker-compose version 1.17.1, build unknown
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN$ docker-machine --version
docker-machine version 0.15.0, build b48dc28
```

2)

On exécute notre premier conteneur par l'intermédiaire de la commande : *sudo docker run hello-world*

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 1 : Exécution d'un conteneur

Dans la figure 1, on va d'abord regarder en local si l'image 'hello-world' est présente. Elle n'est pas trouvée. On regarde ensuite dans le docker hub. Une fois que l'image est mise en local, on ira la chercher directement en local.

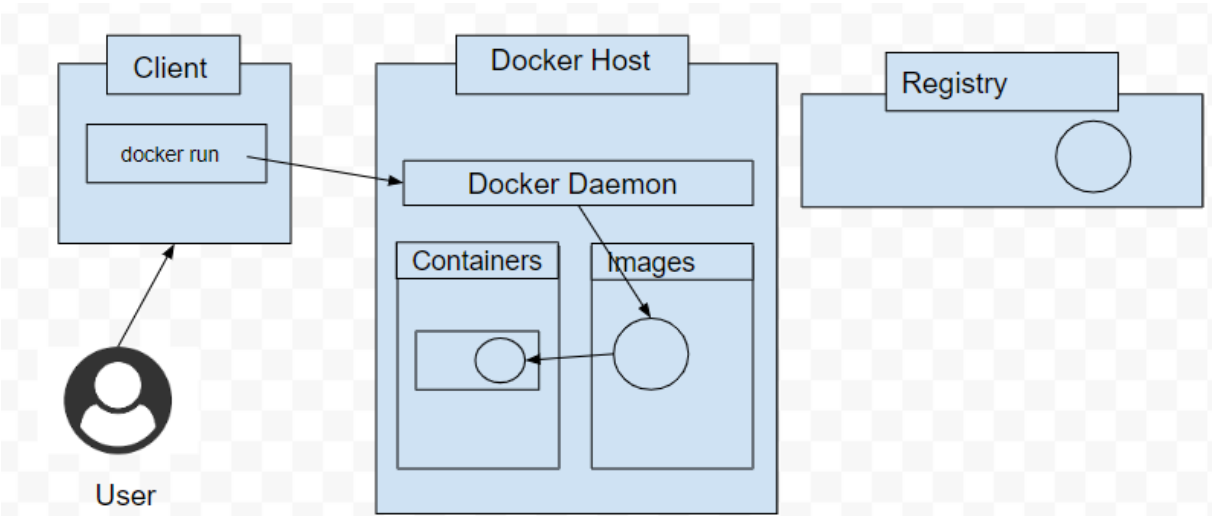


Figure 2 : Architecture Docker

Docker utilise une architecture client-serveur. Dans la figure 2, « Client » correspond au client docker, soit l'interface en ligne de commande. D'autre part, le processus serveur est le « Docker Daemon » en cours d'utilisation sur l'hôte. Le serveur et le client peuvent se trouver sur la même machine.

On va distinguer l'image docker du conteneur docker :

- Une image docker est le composant de construction de Docker (Système d'exploitation, serveur...). On parle de modèle en lecture seule.

- Le conteneur docker est le processus utilisant l'image Docker. Il peut être exécuté, démarré, déplacé, arrêté et supprimé. Chaque conteneur constitue une plateforme applicative sécurisée et isolée des autres.

Le Docker Daemon effectue un gros travail : il construit des images, extrait des images du registre, exécute des conteneurs avec des images. On peut accéder au Docker Daemon par l'intermédiaire du client Docker.

Pour finir, les images peuvent être stockées dans des registres Docker. Docker fournit un accès public que ce soit à Docker Cloud ou à Docker Hub. Ils contiennent des images prédéfinies. On peut ainsi extraire ces images docker et utiliser des conteneurs dans notre hôte docker.

Pour résumer, pour une commande « docker run hello-world », le client Docker contacte le docker daemon. Ce dernier récupère l'image « hello-world » sur le docker hub. Il crée un nouveau conteneur pour cette image. Puis il lance l'exécutable de la sortie qu'on est en train de lire. Le docker daemon envoie cette sortie au client docker qui l'envoie à nouveau vers notre terminal.

3)

On va à présent lancer un serveur web à l'intérieur d'un conteneur :

```
orain@orain-VirtualBox:~$ sudo docker run -d -p 80:80 --name webserver nginx
[sudo] Mot de passe de orain :
```

```
[sudo] Mot de passe de orain :
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f17d81b4b692: Pull complete
d5c237920c39: Pull complete
a381f92f36de: Pull complete
Digest: sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd503e853af5be16597cb2e
Status: Downloaded newer image for nginx:latest
246d6704fcf181e6c22b795889f894c007e35730fb41f4908a1d4ce0f7bb0c74
```

On n'a pas trouvé l'image en local donc on a été la chercher dans le docker hub. Une image peut être divisée en plusieurs couche ce qui explique les 3 pull. Le serveur a bien été lancé dans un conteneur.

4)

On vérifie la présence de nos images grâce à la commande *docker images*

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/td1$ sudo docker
images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              dbfc48660aeb        3 days ago
109MB
hello-world         latest              4ab4c602aa5e        5 weeks ago
1.84kB
```

Ces images nginx et hello-world sont maintenant stockées en local.

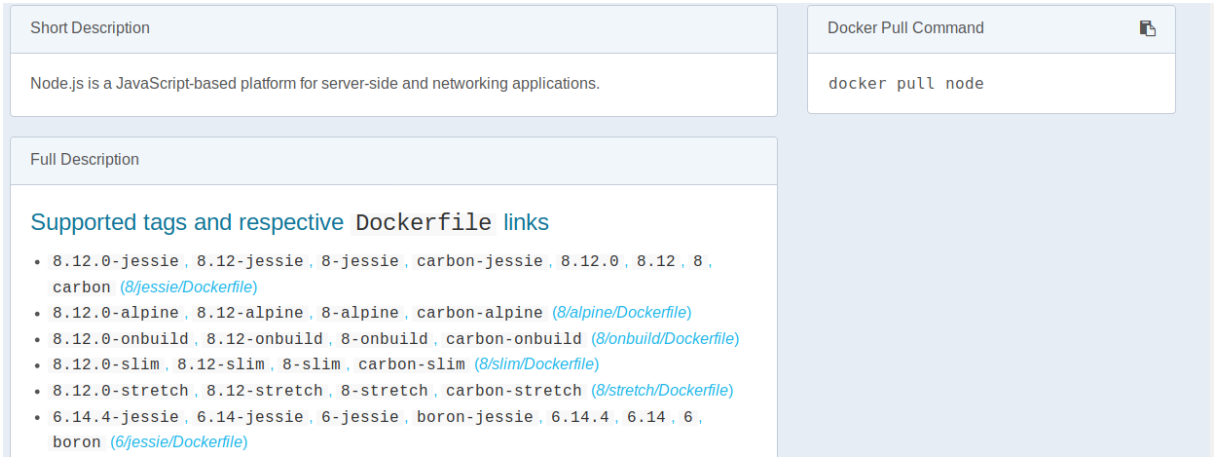
5)

On va utiliser node js à l'intérieur d'un conteneur. Pour cela on va faire une recherche de l'image officielle.

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker search --filter "is-official=true" --filter "stars=3" node.js
[sudo] Mot de passe de orain :
NAME                DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
node                Node.js is a JavaScript-based platform for s…   6403                [OK]
mongo-express       Web-based MongoDB admin interface, written w…   310                 [OK]
iojs                io.js is an npm compatible platform original…   126                 [OK]
```

6)

Depuis le navigateur il est possible de voir le détail de l'image officielle déployée sur Docker Hub à l'adresse suivante : https://hub.docker.com/_/node/. Cela permet de pouvoir choisir la version de l'image.



7)

On récupère la dernière version de l'image officielle de node.js à l'aide de la commande *docker pull*

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker pull node
Using default tag: latest
latest: Pulling from library/node
61be48634cb9: Pull complete
fa696905a590: Pull complete
b6dd2322bbef: Pull complete
32477089adb4: Pull complete
febe7209ec28: Pull complete
4364cbe57162: Pull complete
437859acfd49: Pull complete
d8268e1e433b: Pull complete
Digest: sha256:00a7fb3df8e94ed24f42c2920f132f06e92ea5ed69b1c5e53c4bb3d20e85a3e2
Status: Downloaded newer image for node:latest
```

Lorsque l'on ne spécifie pas le numéro de version, il prend la dernière par défaut.

8)

On vérifie la présence de nos images en local :

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker images
[sudo] Mot de passe de orain :
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mynginximage        latest              65ca1af410ba        17 hours ago        109MB
node                latest              a2b9536415c2        6 days ago          674MB
nginx               1.15.5              dbfc48660aeb        6 days ago          109MB
```

On a bien la dernière version de l'image node.js

On cherche à voir les conteneurs qui s'exécutent en tâche de fond. L'image node.js n'est pas exécutée car elle n'est pas dans un conteneur.

Ayant démarré une nouvelle session, la commande *sudo docker ps* ne donne rien : aucun conteneur n'est en route. On utilise la commande *sudo docker ps -a* pour voir les conteneurs arrêtés.

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
c6bfbbe4e6d5        hello-world         "/hello"                 27 hours ago        Exited (0) 27 hours ago                         compassionate_hodgkin
246d6704fcf1        nginx               "nginx -g 'daemon of…"   27 hours ago        Exited (255) 27 hours ago   0.0.0.0:80->80/tcp  webserver
f042b6f8c58b        hello-world         "/hello"                 2 days ago          Exited (0) 2 days ago                           quizzical_edison
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$
```

En général, il faut dans un premier temps arrêter les conteneurs. Ensuite, on supprime les conteneurs arrêtés (*sudo docker rm $(sudo docker ps -a -q)*). Maintenant que les conteneurs sont supprimés, on peut supprimer les images.

10)

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker stop 857c33c60a6629aee611835878826e67d2a881a2afce1b6c3d3ec95b6b1cadb6
857c33c60a6629aee611835878826e67d2a881a2afce1b6c3d3ec95b6b1cadb6
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker rm $(sudo docker ps -a -q)
857c33c60a66
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              dbfc48660aeb        4 days ago          109MB
```

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd503e853af5be16597cb2e
Deleted: sha256:dbfc48660aeb7ef0ebd74b4a7e0822520aba5416556ee43acb9a6350372e516f
Deleted: sha256:1a34717cf175feab802f74f0edd1c41a811165f6e6af5cddf9b33f9211acde10
Deleted: sha256:df31c4d2dc314417ca1507e7e6ac4e732683a67b5aec725ede170ea7c2ecc99e
Deleted: sha256:237472299760d6726d376385edd9e79c310fe91d794bc9870d038417d448c2d5
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$
```

11)

Pour résumer, *docker rm* permet de supprimer un ou plusieurs conteneurs tandis que *docker rmi* a pour objectif la suppression d'images.

Partie 2 : Personnalisation d'une image

1)

On lance d'abord un conteneur de nginx (que l'on appelle webserver) avec la version 1.15.5 qui est accessible par le port 80.

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker run -d -p 80:80 --name webserver nginx:1.15.5
Unable to find image 'nginx:1.15.5' locally
1.15.5: Pulling from library/nginx
f17d81b4b692: Pull complete
d5c237920c39: Pull complete
a381f92f36de: Pull complete
Digest: sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd503e853af5be16597cb2e
Status: Downloaded newer image for nginx:1.15.5
dd747771f889a1c0c751dc4521b677549214c402ae839894ccd94b1b24e9a4f3
```

2)

On vérifie que le conteneur est lancé par l'intermédiaire de la commande *curl http://localhost:80*

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
3c90e633076b        nginx:1.15.5        "nginx -g 'daemon of…"   10 minutes ago      Up 10 minutes       0.0.0.0:80->80/tcp  webserver
```

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               1.15.5              dbfc48660aeb        4 days ago          109MB
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ curl http://localhost:80
<!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 nginx!</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>
</html>
```

On constate bien que le conteneur est lancé.

3)

On va maintenant créer notre propre contenu. On crée un répertoire docker-nginx/html puis on crée un fichier index.html dans ce dernier.

4)

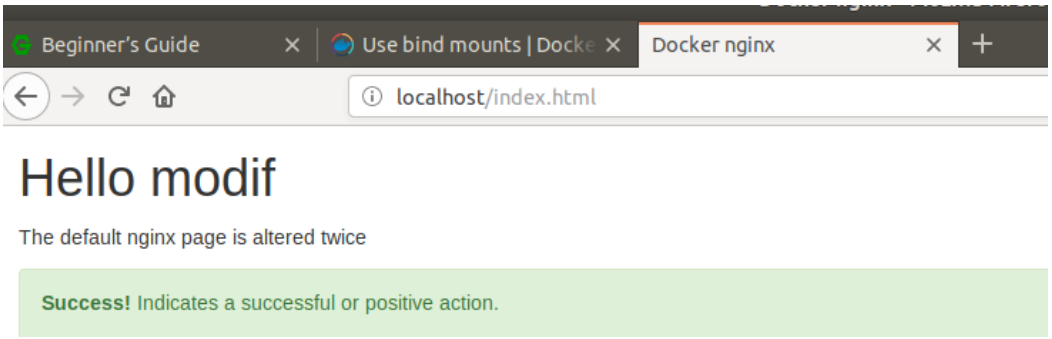
Pour intégrer cette nouvelle page dans le conteneur nginx, on va créer un lien entre l'espace de stockage de l'hôte et celui du conteneur. On supprime tout d'abord le conteneur webserver, puis on lance la commande suivante :

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/docker-nginx/html$ sudo docker run -d -p 80:80
-v "$(pwd)":/usr/share/nginx/html:ro --name test-bind nginx
46a552186d3a3ecdb2d3a63994963321947d14c8d816f31fe14dcb021cb5d13e
```

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/docker-nginx/html$ sudo docker inspect testbind
```

```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/orain/Documents/Cloud/TD1-ORAIN/td1-antho35/docker-nginx/html",
        "Destination": "/usr/share/nginx/html",
        "Mode": "ro",
        "RW": false,
        "Propagation": "rprivate"
    }
],
```

On vérifie le bon fonctionnement du serveur dans le navigateur. On se rend à l'URL suivante : http://localhost:80/index.html



Le nouveau fichier index.html a bien été pris en compte.

7) Pour modifier les fichiers de configuration, il n'est pas possible de se connecter directement au conteneur nginx en ssh car on souhaite garder une certaine sécurité.

Création d'une image

8)

On souhaite visualiser les fichiers de configuration du conteneur nginx. On va créer notre propre image nginx exposant le répertoire qui nous intéresse.

9)

On crée un répertoire dans lequel on crée un fichier Dockerfile.

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35$ cd crea-img/
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/crea-img$ touch Dockerfile
```

Le fichier Dockerfile contient les lignes suivantes :

```
FROM nginx
VOLUME /etc/nginx
```

10)

On crée une nouvelle image que l'on appelle « mynginximage » :

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/crea-img$ sudo docker build - < Dockerfile --tag mynginximage
double free or corruption (out)
SIGABRT: abort
PC=0x7f0f6f560e97 m=0 sigcode=18446744073709551610
signal arrived during cgo execution
```

```
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM nginx
 ---> dbfc48660aeb
Step 2/2 : VOLUME /etc/nginx
 ---> Running in 3b4f4cc7043e
Removing intermediate container 3b4f4cc7043e
 ---> 65ca1af410ba
Successfully built 65ca1af410ba
Successfully tagged mynginximage:latest
```

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/crea-img$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mynginximage        latest              65ca1af410ba        About a minute ago  109MB
nginx               1.15.5              dbfc48660aeb        5 days ago          109MB
nginx               latest              dbfc48660aeb        5 days ago          109MB
```

Dans les figures ci-dessus on remarque que l'image mynginximage a bien été créée.

11)

On crée un conteneur à partir de notre image mynginximage. Le nom du conteneur est « nginx2 ».

```
orain@orain-VirtualBox:~/Documents/Cloud/TD2-ORAIN/td_docker/site2$ sudo docker run -d -p 82:80 --name nginx2 mynginximage
6c62b8b4be1c8b8a9709125374c39ece0eadc8c781750487943841db9d0f292f
```

```
orain@orain-VirtualBox:~/Documents/Cloud/TD2-ORAIN/td_docker/site2$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
6c62b8b4be1c        mynginximage        "nginx -g 'daemon of…"   34 seconds ago      Up 33 seconds       0.0.0.0:82->80/tcp     nginx2
```

Dans l'image ci-dessus, on voit que le conteneur nginx2 a bien été lancé.

12)

On souhaite maintenant créer un conteneur debian partageant le volume /etc/nginx du conteneur nginx2. Il devra disposer comme processus principal la commande bash permettant d'avoir une console.

```
orain@orain-VirtualBox:~/Documents/Cloud/TD2-ORAIN/td_docker/site2$ sudo docker run -i -t --volumes-from nginx2 --name nginxfiles debian /bin/
bash
root@fdd8b5b583d1:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@fdd8b5b583d1:/# cd etc
root@fdd8b5b583d1:/etc# cd nginx/
root@fdd8b5b583d1:/etc/nginx# ls
conf.d  fastcgi_params  koi-utf  koi-win  mime.types  modules  nginx.conf  scgi_params  uwsgi_params  win-utf
root@fdd8b5b583d1:/etc/nginx# cat nginx.conf
```

13)

Dans la figure ci-dessus, on navigue jusqu'au répertoire du conteneur : /etc/nginx.

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

On analyse le contenu du fichier nginx.conf ainsi que le fichier définissant le comportement du serveur : default.conf. On copie le contenu de ce dernier dans un nouveau fichier sur notre hôte : docker-nginx/default.conf

Conteneur personnalisé

16)

Afin de créer un simple proxy, nous devons modifier le fichier de configuration default.conf. Le serveur nginx est capable de délivrer des images à partir d'un répertoire local /usr/share/nginx/image.

On réalise un bind-mount de façon à partager les images du dépôt git sur un répertoire local de notre conteneur :


```
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/orain/Documents/Cloud/TD1-ORAIN/td1-antho35/td1/images",
        "Destination": "/usr/share/nginx/image",
        "Mode": "ro",
        "RW": false,
        "Propagation": "rprivate"
    }
],
```


Dans la figure ci-dessous, on ajoute complète la configuration de notre serveur dans default.conf de façon à délivrer des images à partir d'un répertoire local /usr/share/nginx/image de notre conteneur, ainsi qu'un serveur web écoutant sur le port 8080 pour toutes les autres requêtes.

```
server {
    location / {
        proxy_pass http://localhost:8080/;
    }

    location ~\.(gif|jpg|png)$ {
        root /usr/share/nginx/image;
    }
```

17)

On complète notre fichier Dockerfile situé dans le répertoire crea-img. On copie le contenu et les fichiers de configuration à l'intérieur de l'image. Le répertoire crea-img contient également le fichier default.conf.

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY default.conf /etc/nginx/conf.d/default.conf
VOLUME /etc/nginx
```

On crée une image « mynginximg2 » qui contiendra le fichier de configuration default.conf

```
orain@orain-VirtualBox:~/Documents/Cloud/TD1-ORAIN/td1-antho35/crea-img$ sudo docker build --tag mynginximg2 .
```

```
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM nginx
 ---> dbfc48660aeb
Step 2/4 : RUN rm /etc/nginx/conf.d/default.conf
 ---> Running in 77f3d27baee2
Removing intermediate container 77f3d27baee2
 ---> 5398cfc19ae2
Step 3/4 : COPY default.conf /etc/nginx/conf.d/default.conf
 ---> 0935a7c5fd07
Step 4/4 : VOLUME /etc/nginx
 ---> Running in 1a9bbdfe1e6c
Removing intermediate container 1a9bbdfe1e6c
 ---> 8dac0adf54ca
Successfully built 8dac0adf54ca
Successfully tagged mynginximg2:latest
```

On a bien créé une nouvelle image.

Pour finir, il suffit de créer un conteneur pour cette image grâce à la commande suivante :

Sudo docker run -d -p 83 :80 --name imagecontainer mynginximg2

Pour regarder si les photos sont présentes dans le conteneur, on va naviguer dans ce dernier :

Sudo docker exec -it mynginximg2 bash

On navigue jusqu'au répertoire */usr/share/nginx/image*