SHCHERBAKOVA Kateryna
TKACHENKO Oleksii
M2 IL, TLC
TP1 Projet Docker

Étape 0. Test de votre installation

This command opened a shell within the Ubuntu container, distinctly different from the host machine's distribution.

```
kate@Stitch:~/Kate/Rennes1/M2/TLC$ docker run -t -i ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
a48641193673: Pull complete
Digest: sha256:6042500cf4b44023ea1894effe7890666b0c5c7871ed83a97c36c76ae560bb9b
Status: Downloaded newer image for ubuntu:latest
```

The container operates with its unique network interface, different from the host machine's one.

```
root@e7935ff0732c:/# /sbin/ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 6874  bytes 29504787 (29.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5033  bytes 276529 (276.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Étape 1. Jouons avec docker: mise en place d'un load balancer et d'un reverse proxy avec docker et nginx

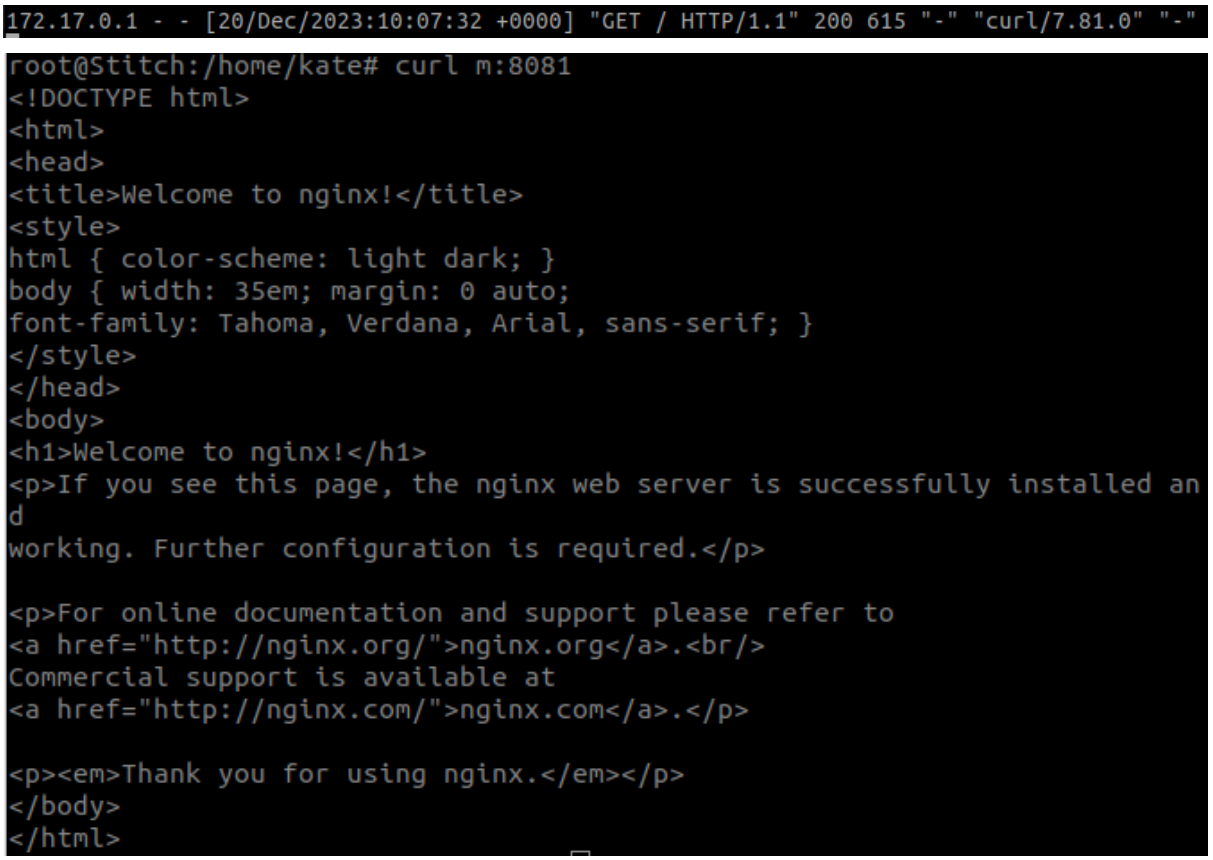
Running Nginx as Reverse Proxy using Docker

```
kate@Stitch:~$ docker run -d -p 8080:80 -v /var/run/docker.sock:/tmp/docker.sock -t jwilder/nginx-proxy
2d21d9121d5bfd32104cccf989469529fb2151644f63b926ec4cf2de90b53432
```

The `apt-get install terminator` command is used to install the `Terminator` terminal. Once installed, Terminator is started in root mode using the `sudo terminator` command.

Set up a mapping between the hostname "m" and the local address 127.0.0.1 (localhost). This allows "m" to be used as the host to address the local machine, and requests addressed to "m" will be redirected to the IP address 127.0.0.1 (the current machine).

After that, a new window is created in Terminator, where the Nginx container is started with the VIRTUAL_HOST environment variable set to "m". I started it on port `8081`, since the default port `8080` is being used on my machine.
```docker run -e VIRTUAL_HOST=m -t -i -p 8081:80 nginx```
```curl m:8081```



The `docker exec -it 1c65b553cb0f bash` command is used to run an interactive shell (in this case bash) inside a Docker container.
Finally, to stop all running Nginx containers, use the `docker kill` command with the container ID.

Etape 2: Utilisation de docker compose

Utilisez docker compose pour déployer vos 4 services nginx et votre loadbalancer.

When using jwilder/nginx-proxy together with Docker Compose to automatically configure a load balancer (Reverse Proxy) for Nginx services, there is no direct link between the services and the load balancer in the docker-compose.yml file. However:

- The VIRTUAL_HOST environment variable defined for each Nginx service specifies which hostname each service should listen to.
- The jwilder/nginx-proxy image is used as a service load-balancer that automatically determines the VIRTUAL_HOST environment variable and routes requests to the appropriate Nginx services based on that variable.

This `docker-compose.yml` file describes several Nginx services (nginx1, nginx2, nginx3, nginx4) with the `VIRTUAL_HOST` environment variable set to "m" and opening port 80 to access each service. The loadbalancer service is also defined using the `jwilder/nginx-proxy` image. Thus, this file allows deployment of four Nginx services and a load balancer, with a reverse proxy configuration to handle requests for hostname "m" and load balancing between running Nginx services.

```yaml
version: '3' # version of docker compose

services:
  nginx1:
     image: nginx
     environment:
        VIRTUAL_HOST: "m"
     expose:
        - "80" # open a port
  nginx2:
     image: nginx
     environment:
        VIRTUAL_HOST: "m"
     expose:
        - "80" # open a port
  nginx3:
     image: nginx
     environment:
        VIRTUAL_HOST: "m"
     expose:
        - "80" # open a port
  nginx4:
     image: nginx
     environment:
        VIRTUAL_HOST: "m"
     expose:
        - "80" # open a port
```

```
  loadbalancer:
     image: jwilder/nginx-proxy
     ports:
        - "8081:80"
     volumes:
        - /var/run/docker.sock:/tmp/docker.sock
```

```
kate@Stitch:~/Kate/Rennes1/M2/TLC$ docker-compose up
Creating network "tlc_default" with the default driver
Creating tlc_nginx1_1       ... done
Creating tlc_nginx4_1       ... done
Creating tlc_nginx2_1       ... done
Creating tlc_loadbalancer_1 ... done
Creating tlc_nginx3_1       ... done
```

```curl m:8081``` several times ->

```
loadbalancer_1  | nginx.1     | 2023/12/20 11:28:45 [notice] 21#21: signal 29 (SIGIO) received
nginx2_1        | 172.21.0.6 - - [20/Dec/2023:11:29:18 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:18 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.2:80"
nginx2_1        | 172.21.0.6 - - [20/Dec/2023:11:29:30 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:30 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.2:80"
nginx2_1        | 172.21.0.6 - - [20/Dec/2023:11:29:32 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:32 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.2:80"
nginx2_1        | 172.21.0.6 - - [20/Dec/2023:11:29:35 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:35 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.2:80"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:39 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.4:80"
nginx3_1        | 172.21.0.6 - - [20/Dec/2023:11:29:39 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
loadbalancer_1  | nginx.1     | m 172.21.0.1 - - [20/Dec/2023:11:29:52 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.3:80"
nginx1_1        | 172.21.0.6 - - [20/Dec/2023:11:29:52 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "172.21.0.1"
```

Different services are responsible for requests. Services are the SAME.

Optimization:

```
version: '3' # version of docker compose

services:
  nginx1:
     image: nginx
     environment:
        VIRTUAL_HOST: "m"
     expose:
        - "80" # open a port
     deploy:
        replicas: 4

  loadbalancer:
     image: jwilder/nginx-proxy
     ports:
        - "8081:80"
     volumes:
        - /var/run/docker.sock:/tmp/docker.sock
```
```

This updated `docker-compose.yml` file will deploy four replicas of the Nginx service. It will do essentially the same thing as the previous example, but without duplicating code.

`expose` - open port - services
`-p` - bind port - load balancer

Etape 3: Dockeriser une application existante

1. I started by cloning the `TPDockerSampleApp` repository. I then installed the necessary dependencies using the commands:

```
sudo apt-get update
sudo apt-get install -y openjdk-8-jdk
sudo apt-get install -y maven
sudo apt-get install -f libpng16-16
sudo apt-get install -f libjasper1
sudo apt-get install -f libdc1394-22
```

2. And launch the application:

```
cd TPDockerSampleApp
    mvn install:install-file -Dfile=./lib/opencv-3410.jar \
    -DgroupId=org.opencv  -DartifactId=opencv -Dversion=3.4.10 -Dpackaging=jar

mvn package
java -Djava.library.path=lib/ubuntuupperthan18/ -jar target/fatjar-0.0.1-SNAPSHOT.jar
```

In the line `Server server = new Server(8081);`, I changed the port to 8081, since 8080 is busy.

A **Dockerfile** was created in the root folder of the project:

```
# specify the base image
FROM ubuntu
```

```
# set the working directory
WORKDIR /app
# copy application files to the container
COPY . /app

# specify dependencies
RUN apt-get update && apt-get install -y openjdk-8-jdk && apt-get install
-y maven && apt-get install -f libpng16-16

RUN       mvn       install:install-file       -Dfile=./lib/opencv-3410.jar
-DgroupId=org.opencv -DartifactId=opencv -Dversion=3.4.10 -Dpackaging=jar
RUN mvn package

# specify default executable
ENTRYPOINT    ["java","-Djava.library.path=lib/ubuntuupperthan18/","-jar",
"target/fatjar-0.0.1-SNAPSHOT.jar"]
```

To compile a Dockerfile it is necessary to run the command:

```
`docker build -t docker-face:v2 .`
```

After that, to start the container the command needs to be executed:

```
`docker run -p 8081:8081 docker-face:v1`
```