TP Docker évalué

Task 1 :

Tout d'abord, il faut installer prestashop avec la commande suivante :
docker pull bitnami/prestashop

Ensuite, créons notre réseau :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network create global-network
af1adb819a078ff2716ec0b64315099750f641617253ffb268627d1d6d5e2249
```

Ci-dessous prestashopNetwork.

Créons notre container frontend avec l'image Prestashop, une plateforme eCommerce :

```
moham@DESKTOP-4927EH0 MINGW64 /
$ docker run -ti --privileged --name mysql --network prestashopNetwork -e MYSQL_ROOT_PASSWORD=admin -p 3307:3306 -d mysql
5a4992739a6a37460db6d07de6f038b336dc6a3aa6ea7e2bf381efad7abd3500
```

/!\ Il faut bien préciser le port pour pouvoir ouvrir l'application en localhost.

```
moham@DESKTOP-4927EH0 MINGW64 ~
$ docker run -ti --privileged --name prestashop --network prestashopNetwork -e DB_SERVER=Servermysql -p 8080:80 -d prestashop/prestashop
02ac95ffef7bb3391e65d58562ff1d8d607d8b1fad8242ddb5b62f980d8c2417
```

Créons notre container backend avec l'image PostGres, une base de données :

Voilà le contenu de notre réseau :

On installe les librairies nécessaires avec les commandes ci-dessous :
apt update
apt install iputils-ping
apt install inetutils-traceroute
apt install iproute2
apt install curl telnet dnsutils vim

Après cela, on peut ping l'autre container :

```
root@4e8e8f0cac6f:/# ping 127.19.0.3
PING 127.19.0.3 (127.19.0.3) 56(84) bytes of data.
64 bytes from 127.19.0.3: icmp_seq=1 ttl=64 time=0.173 ms
64 bytes from 127.19.0.3: icmp_seq=2 ttl=64 time=0.036 ms
64 bytes from 127.19.0.3: icmp_seq=3 ttl=64 time=0.024 ms
64 bytes from 127.19.0.3: icmp_seq=4 ttl=64 time=0.024 ms
64 bytes from 127.19.0.3: icmp_seq=5 ttl=64 time=0.027 ms
^C
--- 127.19.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4141ms
rtt min/avg/max/mdev = 0.024/0.056/0.173/0.058 ms
```

Ils communiquent bien !

Task 2 :

On crée les 2 subnets dans lesquels on stockera nos container backend et frontend :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network create --subnet=192.168.1.0/24 frontend-network
f18b3d6a39e67aac0ebf78a9722ad57842f96ed79b183b2cfac8b6ac08e4364b

axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network create --subnet=192.168.2.0/24 backend-network
0d52663713df7b3b1bf9db278b3ec272e1e41e5d55d5d7f43ac0e222f7551b10
```

On recrée les containers back et front :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker run -d --privileged --name ynov-frontend --network frontend-network -e PRESTASHOP_DATABASE_PAS
SWORD=admin prestashop/prestashop
86ebae844d94ec9c771455224e759da3c33af3f6326dd96181c485631a1a020a

axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker run -d --privileged --name ynov-backend --network backend-network -e POSTGRES_PASSWORD=admin p
ostgres
154305aee2832316303ef209238f4d5bd00406002ee9548b5725a6d2e8ae4234
```

On crée le container qui fait office de router :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker run -d --privileged --name ynov-router --network frontend-network nginx
55095e5d0a4adc82931bacee01eea3b8975f34472018650f3ac4c713c9e43751

axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network connect backend-network ynov-router
```

/!\ Il est important de préciser l'option --privileged pour pouvoir procéder au routage par la suite.

Voici nos 3 containers :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS              POR
TS      NAMES
55095e5d0a4a   nginx                  "/docker-entrypoint.…"   2 minutes ago   Up About a minute   80/
tcp     ynov-router
154305aee283   postgres               "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes        543
2/tcp   ynov-backend
86ebae844d94   prestashop/prestashop  "docker-php-entrypoi…"   6 minutes ago   Up 6 minutes        80/
tcp     ynov-frontend
```

Contenu du réseau frontend :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "f18b3d6a39e67aac0ebf78a9722ad57842f96ed79b183b2cfac8b6ac08e4364b",
        "Created": "2023-12-07T13:32:50.724008364Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.1.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "55095e5d0a4adc82931bacee01eea3b8975f34472018650f3ac4c713c9e43751": {
                "Name": "ynov-router",
                "EndpointID": "a817e71b579f177654f494e7633fd9abcfe30ff828c0c102b951c5f49b4988fd",
                "MacAddress": "02:42:c0:a8:01:03",
                "IPv4Address": "192.168.1.3/24",
                "IPv6Address": ""
            },
            "86ebae844d94ec9c771455224e759da3c33af3f6326dd96181c485631a1a020a": {
                "Name": "ynov-frontend",
                "EndpointID": "ee296b3a07d2f8636e228dcc1ce9348baae1c29ce533e03fe4eae19343337081",
                "MacAddress": "02:42:c0:a8:01:02",
                "IPv4Address": "192.168.1.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Contenu du réseau backend :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ docker network inspect backend-network
[
    {
        "Name": "backend-network",
        "Id": "0d52663713df7b3b1bf9db278b3ec272e1e41e5d55d5d7f43ac0e222f7551b10",
        "Created": "2023-12-07T13:33:01.316568945Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.2.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "154305aee2832316303ef209238f4d5bd00406002ee9548b5725a6d2e8ae4234": {
                "Name": "ynov-backend",
                "EndpointID": "907822acc38379304898a8d1de471e209762ca7d5ff8fc26edf2b1855230a328",
                "MacAddress": "02:42:c0:a8:02:03",
                "IPv4Address": "192.168.2.3/24",
                "IPv6Address": ""
            },
            "55095e5d0a4adc82931bacee01eea3b8975f34472018650f3ac4c713c9e43751": {
                "Name": "ynov-router",
                "EndpointID": "5ddaff0c5fb051b71b8b2f4db9549772157915e5b9c444fac2daa7aab98b2c1d",
                "MacAddress": "02:42:c0:a8:02:02",
                "IPv4Address": "192.168.2.2/24",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

On rentre dans le container router et on y installe les mêmes librairies que pour l'étape 1 :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ winpty docker exec -u root -it ynov-router bash
root@55095e5d0a4a:/# apt update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [52.1 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8780 kB]
```

Et on peut procéder au routage :

```
root@55095e5d0a4a:/# ip route add 192.168.1.0/24 via 192.168.1.3
RTNETLINK answers: File exists
root@55095e5d0a4a:/# ip route add 192.168.2.0/24 via 192.168.2.3
RTNETLINK answers: File exists
root@55095e5d0a4a:/# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.923 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.044 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.045 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.049 ms
64 bytes from 192.168.1.2: icmp_seq=6 ttl=64 time=0.047 ms
64 bytes from 192.168.1.2: icmp_seq=7 ttl=64 time=0.044 ms
64 bytes from 192.168.1.2: icmp_seq=8 ttl=64 time=0.043 ms
64 bytes from 192.168.1.2: icmp_seq=9 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=10 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=11 ttl=64 time=0.045 ms
64 bytes from 192.168.1.2: icmp_seq=12 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=13 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=14 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=15 ttl=64 time=0.052 ms
64 bytes from 192.168.1.2: icmp_seq=16 ttl=64 time=0.043 ms
64 bytes from 192.168.1.2: icmp_seq=17 ttl=64 time=0.043 ms
64 bytes from 192.168.1.2: icmp_seq=18 ttl=64 time=0.043 ms
64 bytes from 192.168.1.2: icmp_seq=19 ttl=64 time=0.040 ms
64 bytes from 192.168.1.2: icmp_seq=20 ttl=64 time=0.039 ms
64 bytes from 192.168.1.2: icmp_seq=21 ttl=64 time=0.039 ms
64 bytes from 192.168.1.2: icmp_seq=22 ttl=64 time=0.041 ms
^C
--- 192.168.1.2 ping statistics ---
22 packets transmitted, 22 received, 0% packet loss, time 21805ms
rtt min/avg/max/mdev = 0.039/0.083/0.923/0.183 ms
root@55095e5d0a4a:/# ip route show
default via 192.168.2.1 dev eth1
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.3
192.168.2.0/24 dev eth1 proto kernel scope link src 192.168.2.2
```

On peut ping les 2 containers back et front depuis la passerelle, et on voit qu'on a bien ajouté les 2 routes.

On essaie de ping le backend depuis le container frontend, mais cela ne fonctionne pas :

```
axelt@LAPTOP-JJQ1PBIG MINGW64 ~
$ winpty docker exec -it ynov-frontend bash
root@86ebae844d94:/var/www/html# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
^C
--- 192.168.2.2 ping statistics ---
1266 packets transmitted, 0 received, 100% packet loss, time 1315577ms

root@86ebae844d94:/var/www/html# ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3) 56(84) bytes of data.
^C
--- 192.168.2.3 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11466ms
```

Même en essayant avec traceroute, cela ne fonctionne pas :