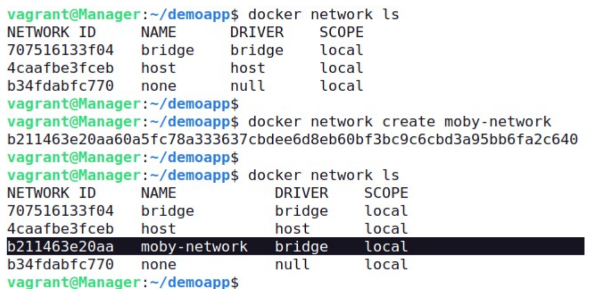
TP5 – Les réseaux Docker
Brahim Hamdi

<u>Objectifs :</u>

- Gérer les réseaux de Docker

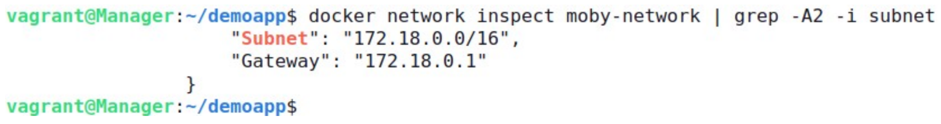
Nous allons lancer une petite application *nodejs* qui fonctionne avec une base de données *redis* qui stocke des paires clé/valeur simples.

1. Créez le nouveau réseau *mobi-network* de type *Bridge* qui va interconnecter les conteneurs de l'application.



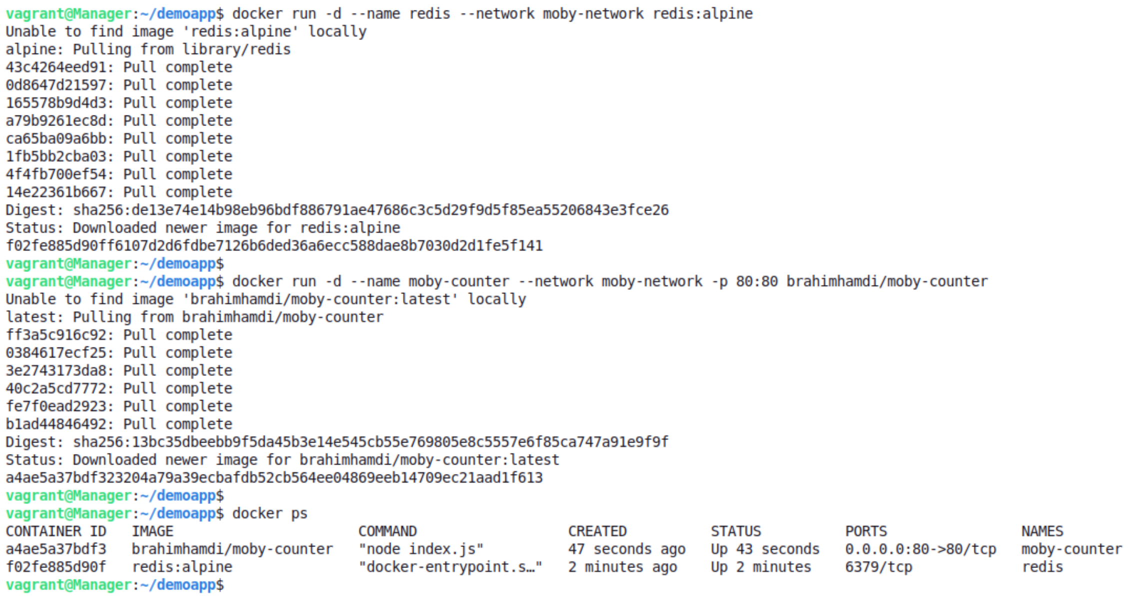
```
vagrant@Manager:~/demoapp$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
707516133f04   bridge    bridge    local
4caafbe3fceb   host      host      local
b34fdabfc770   none      null      local
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network create moby-network
b211463e20aa60a5fc78a333637cbdee6d8eb60bf3bc9c6cbd3a95bb6fa2c640
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network ls
NETWORK ID     NAME           DRIVER    SCOPE
707516133f04   bridge         bridge    local
4caafbe3fceb   host           host      local
b211463e20aa   moby-network   bridge    local
b34fdabfc770   none           null      local
vagrant@Manager:~/demoapp$
```

   ○ Quel est le sous-réseau et la passerelle de *moby-network* ?



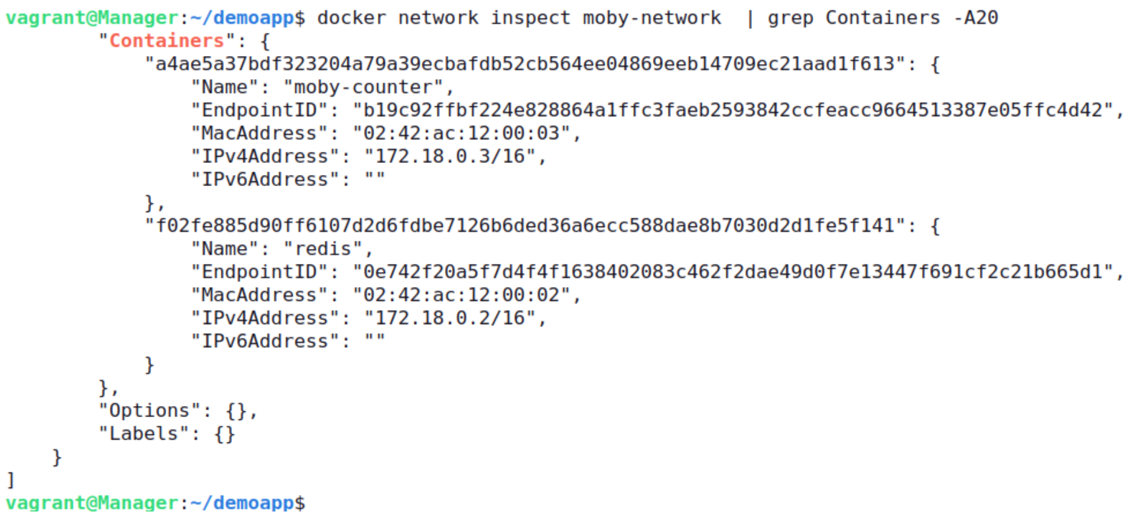
```
vagrant@Manager:~/demoapp$ docker network inspect moby-network | grep -A2 -i subnet
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
vagrant@Manager:~/demoapp$
```

2. Lancez deux nouveaux conteneurs moby-counter et redis qui exécutent, respectivement, les images *brahimhamdi/moby-counter* et *redis*. Les deux conteneurs doivent être connectés au nouveau réseau *moby-network*.



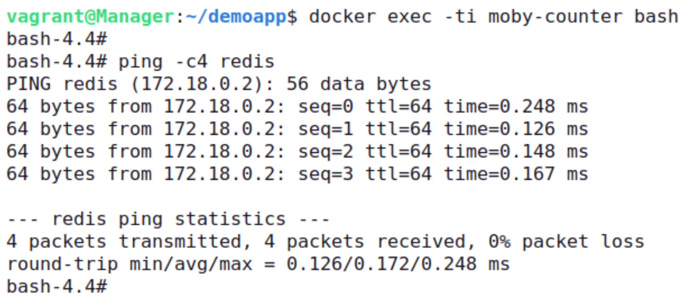
```
vagrant@Manager:~/demoapp$ docker run -d --name redis --network moby-network redis:alpine
Unable to find image 'redis:alpine' locally
alpine: Pulling from library/redis
43c4264eed91: Pull complete
0d8647d21597: Pull complete
165578b9d4d3: Pull complete
a79b9261ec8d: Pull complete
ca65ba09a6bb: Pull complete
1fb5bb2cba03: Pull complete
4f4fb700ef54: Pull complete
14e22361b667: Pull complete
Digest: sha256:de13e74e14b98eb96bdf886791ae47686c3c5d29f9d5f85ea55206843e3fce26
Status: Downloaded newer image for redis:alpine
f02fe885d90ff6107d2d6fdbe7126b6ded36a6ecc588dae8b7030d2d1fe5f141
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker run -d --name moby-counter --network moby-network -p 80:80 brahimhamdi/moby-counter
Unable to find image 'brahimhamdi/moby-counter:latest' locally
latest: Pulling from brahimhamdi/moby-counter
ff3a5c916c92: Pull complete
0384617ecf25: Pull complete
3e2743173da8: Pull complete
40c2a5cd7772: Pull complete
fe7f0ead2923: Pull complete
b1ad44846492: Pull complete
Digest: sha256:13bc35dbeebb9f5da45b3e14e545cb55e769805e8c5557e6f85ca747a91e9f9f
Status: Downloaded newer image for brahimhamdi/moby-counter:latest
a4ae5a37bdf323204a79a39ecbafdb52cb564ee04869eeb14709ec21aad1f613
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS          PORTS                NAMES
a4ae5a37bdf3   brahimhamdi/moby-counter   "node index.js"          47 seconds ago   Up 43 seconds   0.0.0.0:80->80/tcp   moby-counter
f02fe885d90f   redis:alpine               "docker-entrypoint.s…"   2 minutes ago    Up 2 minutes    6379/tcp             redis
vagrant@Manager:~/demoapp$
```

3. Vérifiez que les deux nouveaux conteneurs sont interconnectés sur le réseaux *moby-network*. Quel est l'IP et le MAC de chaque conteneur ?



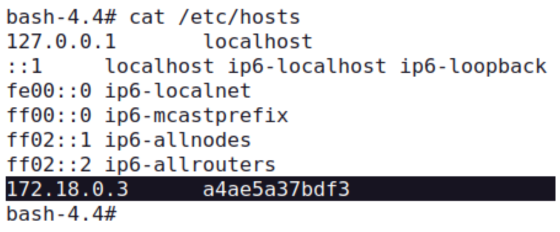
```
vagrant@Manager:~/demoapp$ docker network inspect moby-network  | grep Containers -A20
        "Containers": {
            "a4ae5a37bdf323204a79a39ecbafdb52cb564ee04869eeb14709ec21aad1f613": {
                "Name": "moby-counter",
                "EndpointID": "b19c92ffbf224e828864a1ffc3faeb2593842ccfeacc9664513387e05ffc4d42",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "f02fe885d90ff6107d2d6fdbe7126b6ded36a6ecc588dae8b7030d2d1fe5f141": {
                "Name": "redis",
                "EndpointID": "0e742f20a5f7d4f4f1638402083c462f2dae49d0f7e13447f691cf2c21b665d1",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
vagrant@Manager:~/demoapp$
```

4. Connectez-vous au conteneur *moby-counter* et tester avec ping la connectivité au conteneur *redis*.



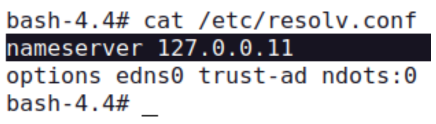
```
vagrant@Manager:~/demoapp$ docker exec -ti moby-counter bash
bash-4.4#
bash-4.4# ping -c4 redis
PING redis (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.248 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.126 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.148 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.167 ms

--- redis ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.126/0.172/0.248 ms
bash-4.4#
```

- Affichez le contenu des fichiers */etc/hosts* du conteneur *moby-counter*. Que constatez-vous ?



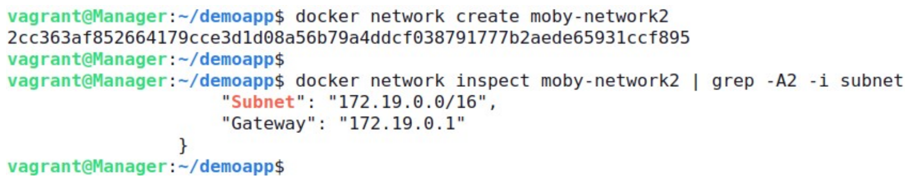
```
bash-4.4# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      a4ae5a37bdf3
bash-4.4#
```

- De même, affichez le contenu de */etc/resolv.conf* du même conteneur ? Expliquez comment a t-il connu l'IP du conteneur *redis*.



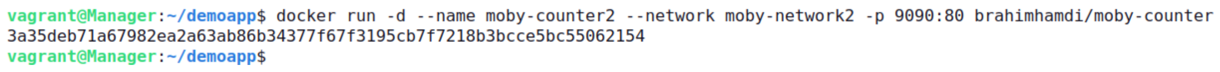
```
bash-4.4# cat /etc/resolv.conf
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
bash-4.4# _
```

5. Créez un deuxième réseau *moby-network2*. Quel est son subnet ?



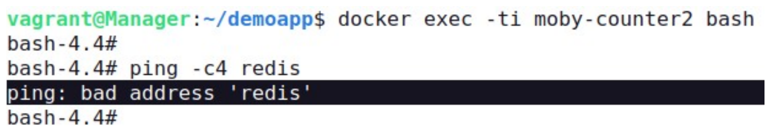
```
vagrant@Manager:~/demoapp$ docker network create moby-network2
2cc363af852664179cce3d1d08a56b79a4ddcf038791777b2aede65931ccf895
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network inspect moby-network2 | grep -A2 -i subnet
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
vagrant@Manager:~/demoapp$
```

- Créez une deuxième instance de l'application dans ce réseau



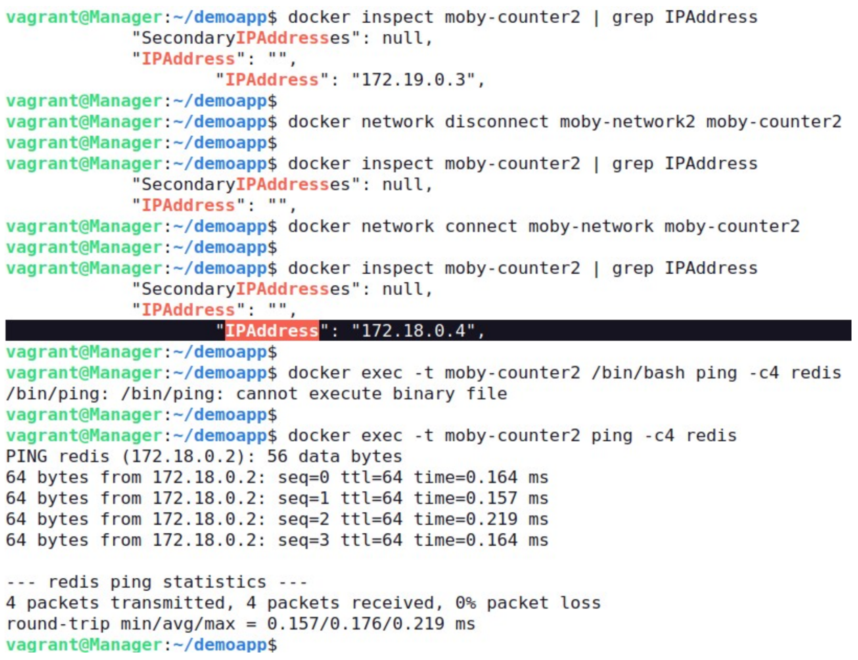
```
vagrant@Manager:~/demoapp$ docker run -d --name moby-counter2 --network moby-network2 -p 9090:80 brahimhamdi/moby-counter
3a35deb71a67982ea2a63ab86b34377f67f3195cb7f7218b3bcce5bc55062154
vagrant@Manager:~/demoapp$
```

- Lorsque vous « pingez » *redis* depuis cette nouvelle instance *moby-counter2*, qu'obtenez-vous ? Pourquoi ?



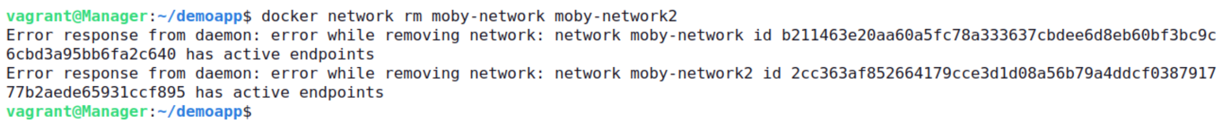
```
vagrant@Manager:~/demoapp$ docker exec -ti moby-counter2 bash
bash-4.4#
bash-4.4# ping -c4 redis
ping: bad address 'redis'
bash-4.4#
```

6. Déconnectez le conteneur *moby-counter2* du réseau *moby-network2*, puis connectez-le au réseau *moby-network* et « repingez » *redis*. Que remarquez-vous ? Quelle est son IP ?



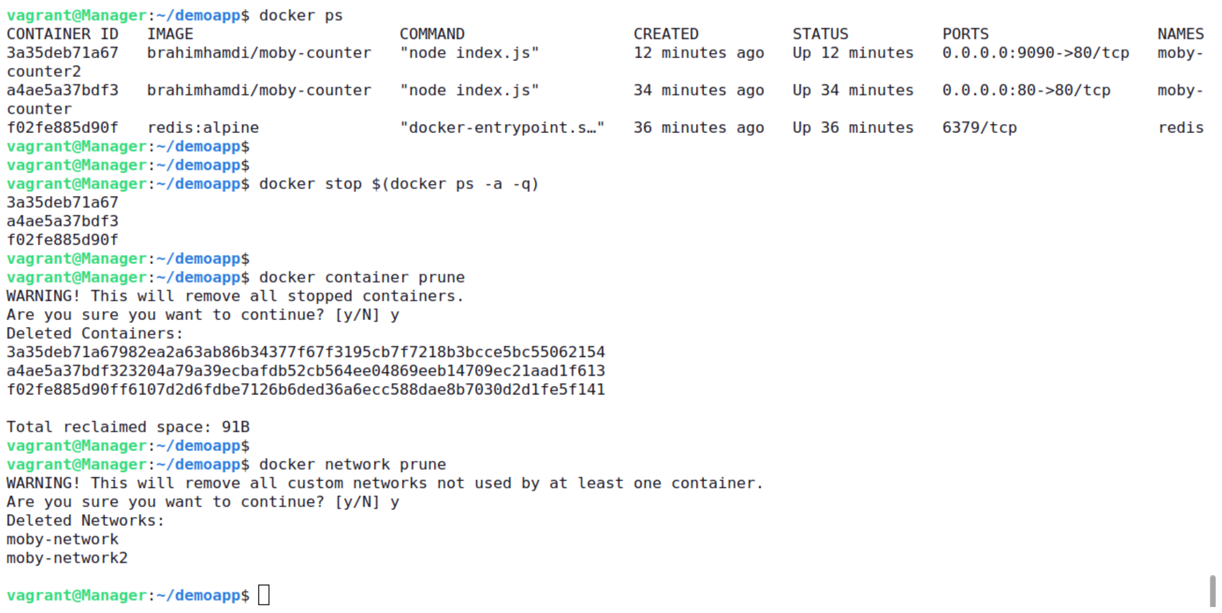
```
vagrant@Manager:~/demoapp$ docker inspect moby-counter2 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.19.0.3",
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network disconnect moby-network2 moby-counter2
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker inspect moby-counter2 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
vagrant@Manager:~/demoapp$ docker network connect moby-network moby-counter2
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker inspect moby-counter2 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.18.0.4",
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker exec -t moby-counter2 /bin/bash ping -c4 redis
/bin/ping: /bin/ping: cannot execute binary file
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker exec -t moby-counter2 ping -c4 redis
PING redis (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.164 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.157 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.219 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.164 ms

--- redis ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.157/0.176/0.219 ms
vagrant@Manager:~/demoapp$
```

7. Supprimez les deux réseaux crées dans ce TP. Que remarquez-vous ?

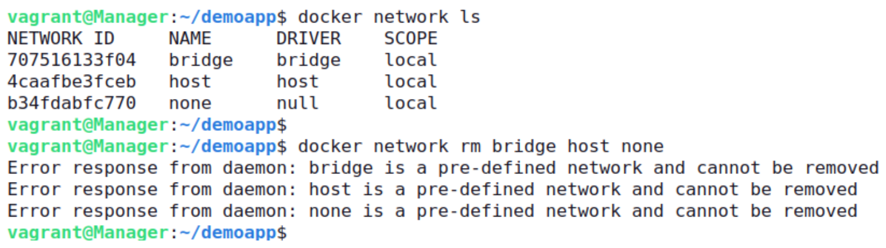


```
vagrant@Manager:~/demoapp$ docker network rm moby-network moby-network2
Error response from daemon: error while removing network: network moby-network id b211463e20aa60a5fc78a333637cbdee6d8eb60bf3bc9c
6cbd3a95bb6fa2c640 has active endpoints
Error response from daemon: error while removing network: network moby-network2 id 2cc363af852664179cce3d1d08a56b79a4ddcf0387917
77b2aede65931ccf895 has active endpoints
vagrant@Manager:~/demoapp$
```

8. Supprimez tous les conteneurs puis tous les réseaux. Les réseaux pré-définis sont-ils supprimés ?



```
vagrant@Manager:~/demoapp$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED          STATUS          PORTS                  NAMES
3a35deb71a67   brahimhamdi/moby-counter "node index.js"          12 minutes ago   Up 12 minutes   0.0.0.0:9090->80/tcp   moby-
counter2
a4ae5a37bdf3   brahimhamdi/moby-counter "node index.js"          34 minutes ago   Up 34 minutes   0.0.0.0:80->80/tcp     moby-
counter
f02fe885d90f   redis:alpine             "docker-entrypoint.s…"   36 minutes ago   Up 36 minutes   6379/tcp               redis
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker stop $(docker ps -a -q)
3a35deb71a67
a4ae5a37bdf3
f02fe885d90f
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
3a35deb71a67982ea2a63ab86b34377f67f3195cb7f7218b3bcce5bc55062154
a4ae5a37bdf323204a79a39ecbafdb52cb564ee04869eeb14709ec21aad1f613
f02fe885d90ff6107d2d6fdbe7126b6ded36a6ecc588dae8b7030d2d1fe5f141

Total reclaimed space: 91B
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
moby-network
moby-network2

vagrant@Manager:~/demoapp$
```



```
vagrant@Manager:~/demoapp$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
707516133f04   bridge    bridge    local
4caafbe3fceb   host      host      local
b34fdabfc770   none      null      local
vagrant@Manager:~/demoapp$
vagrant@Manager:~/demoapp$ docker network rm bridge host none
Error response from daemon: bridge is a pre-defined network and cannot be removed
Error response from daemon: host is a pre-defined network and cannot be removed
Error response from daemon: none is a pre-defined network and cannot be removed
vagrant@Manager:~/demoapp$
```