



**Universidade do Minho**  
Escola de Engenharia

**Gestão e Virtualização de Redes**  
ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA  
**2021/2022**

(Docente: Joaquim Melo Henriques Macedo)

16 de dezembro de 2021

# **Trabalho Prático 1**

## **Docker**

Rui Filipe Ribeiro Freitas - [pg47639@alunos.uminho.pt](mailto:pg47639@alunos.uminho.pt)

Tiago João Pereira Ferreira – [pg47692@alunos.uminho.pt](mailto:pg47692@alunos.uminho.pt)

# Índice

Lista de figuras.....	3
Introdução .....	4
Questão 1.....	5
Questão 2.....	6
Questão 3.....	7
Questão 4.....	10
Questão 5.....	13
Questão 6.....	15
Questão 7.....	18
Conclusão.....	21

## Lista de figuras

Figura 1 - Comando questão 1. ....	5
Figura 2 - Diretório do host. ....	5
Figura 3 - Diretório do container. ....	5
Figura 4 - Comando questão 2. ....	6
Figura 5 - Comando para obter informações de um volume. ....	6
Figura 6 - Comando questão 3. ....	7
Figura 7 - Comando para ver as redes ligadas ao docker. ....	7
Figura 8 - Informação relativa à bridge network. ....	8
Figura 9 - Comando ping entre os containers. ....	9
Figura 10 - Comando questão 4. ....	10
Figura 11 - Criação de containers na rede my-network-1. ....	10
Figura 12 - Network com os containers criados. ....	11
Figura 13 - Ping entre os containers. ....	12
Figura 14 - Comando docker network ls. ....	13
Figura 15 - Esquema da rede no docker. ....	13
Figura 16 - Comando docker network inspect. ....	14
Figura 17 - Comando docker volume ls. ....	14
Figura 18 - Esquema dos volumes no docker. ....	14
Figura 19 - Docker compose. ....	15
Figura 20 - Funcionamento de ambos os serviços. ....	16
Figura 21 - Volume partilhado por ambos os containers. ....	16
Figura 22 - Rede partilhada por ambos os containers. ....	17
Figura 23 - Dockerfile da base de dados. ....	18
Figura 24 - Comando para criar a mysql-image. ....	18
Figura 25 - Comando para executar o mysql container. ....	18
Figura 26 - Dockerfile do node. ....	19
Figura 27 - Comando para executar o node container. ....	19
Figura 28 - Código SQL para a base de dados. ....	19
Figura 29 - Código Javascript. ....	19
Figura 30 - Endpoint. ....	20
Figura 31 - Comando docker ps. ....	20

# Introdução

No âmbito da unidade curricular de Gestão e Virtualização de Redes no módulo de Virtualização de redes foi apresentado pelo docente o trabalho prático número 1 ao qual este relatório diz respeito.

O principal objetivo deste relatório é aprender um pouco mais sobre o que são dockers e como podemos tirar partido destes através da sua aprendizagem pois estes são muito importantes quando falamos de virtualização na área de software informático.

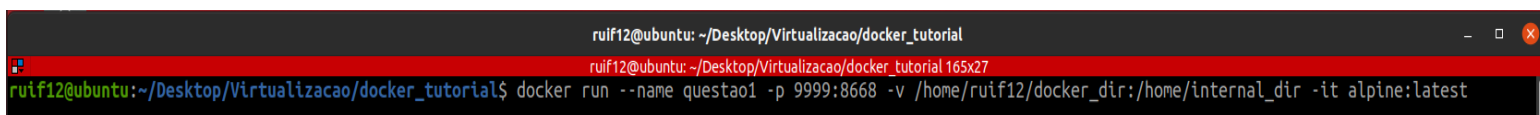
Este relatório está dividido em secções que correspondem às perguntas apresentadas no enunciado.

De modo a sermos capazes de cumprir com o que é pedido na realização deste trabalho prático foi necessário algum investimento de tempo na aprendizagem sobre tudo o que envolve dockers, assim como alguma leitura da sua documentação pois sem isso não seria possível a realização este trabalho.

## Questão 1

**Q:** How to create a container from a Linux alpine that has the container 9999 port mapped in the host 8668. It should also have a bind mount, mounting the container /home/internal\_dir in the hosts /home/user/docker\_dir/.

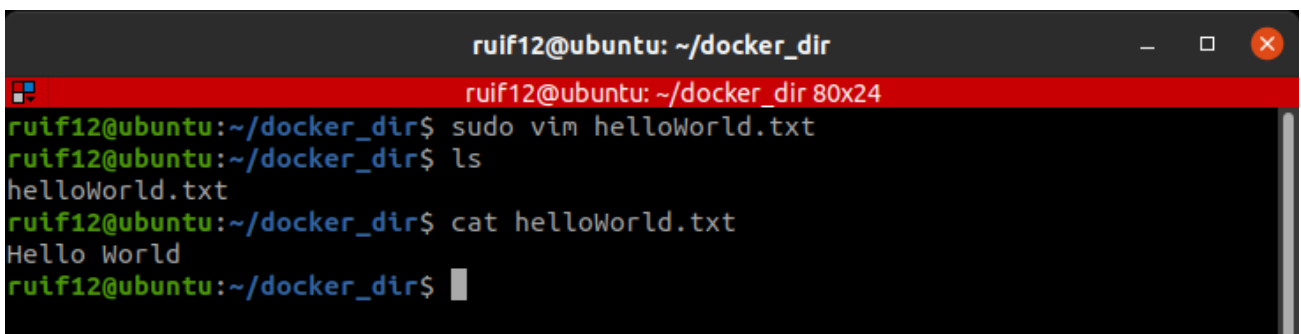
**A:** Para resolver a questão 1 utilizamos o comando presente na figura 1, este comando cria um *container alpine* que tem uma porta 9999 mapeada na porta 8668 do host que serve para estabelecer as portas entre o *host* e o *container*. Para além disso faz diretamente o *mount* ao diretório do *host* no *container*.



```
ruif12@ubuntu: ~/Desktop/Virtualizacao/docker_tutorial
ruif12@ubuntu: ~/Desktop/Virtualizacao/docker_tutorial 165x27
ruif12@ubuntu:~/Desktop/Virtualizacao/docker_tutorial$ docker run --name questao1 -p 9999:8668 -v /home/ruif12/docker_dir:/home/internal_dir -it alpine:latest
```

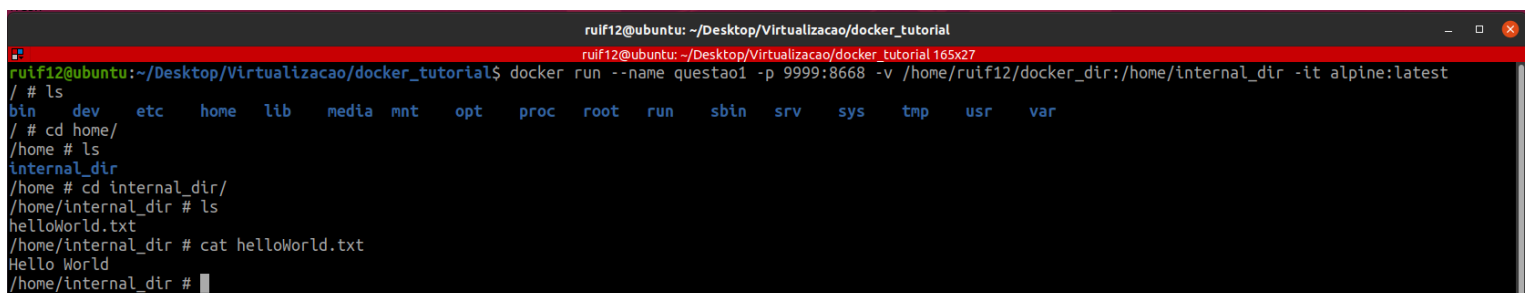
Figura 1 - Comando questão 1.

De modo a testar que o comando funcionou eficientemente decidimos criar um ficheiro no *host* e ver se este ficheiro era acessível no *container*. Na figura 2 apresentamos o processo na parte do *host* através da criação do ficheiro 'helloWorld.txt' na pasta /home/ruif12/docker\_dir. Já na figura 3 conseguimos observar a presença do ficheiro no container na pasta /home/internal\_dir.



```
ruif12@ubuntu: ~/docker_dir
ruif12@ubuntu: ~/docker_dir 80x24
ruif12@ubuntu:~/docker_dir$ sudo vim helloWorld.txt
ruif12@ubuntu:~/docker_dir$ ls
helloWorld.txt
ruif12@ubuntu:~/docker_dir$ cat helloWorld.txt
Hello World
ruif12@ubuntu:~/docker_dir$
```

Figura 2 - Diretório do host.



```
ruif12@ubuntu: ~/Desktop/Virtualizacao/docker_tutorial
ruif12@ubuntu: ~/Desktop/Virtualizacao/docker_tutorial 165x27
ruif12@ubuntu:~/Desktop/Virtualizacao/docker_tutorial$ docker run --name questao1 -p 9999:8668 -v /home/ruif12/docker_dir:/home/internal_dir -it alpine:latest
/ # ls
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
/ # cd home/
/home # ls
internal_dir
/home # cd internal_dir/
/home/internal_dir # ls
helloWorld.txt
/home/internal_dir # cat helloWorld.txt
Hello World
/home/internal_dir #
```

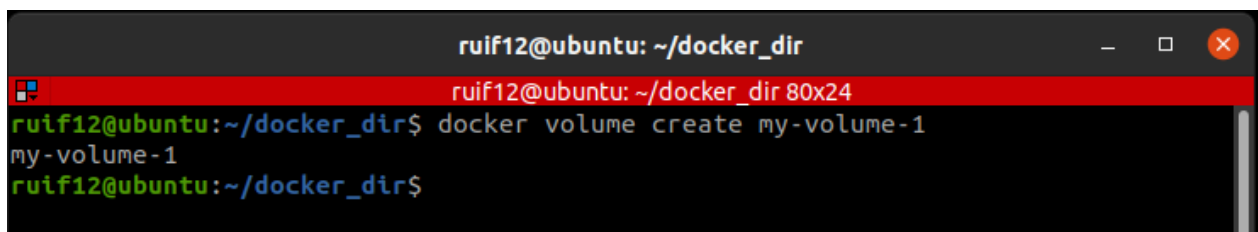
Figura 3 - Diretório do container.

## Questão 2

Q: How to create the volume “my-volume-1”?

- (a) What is the volume mountpoint?
- (b) Which driver is being used?


A: De modo a criar um volume foi necessário introduzir o comando demonstrado na figura 4. Como forma de confirmação da criação do volume o comando devolve o nome do volume criado.



```
ruif12@ubuntu: ~/docker_dir
ruif12@ubuntu: ~/docker_dir 80x24
ruif12@ubuntu:~/docker_dir$ docker volume create my-volume-1
my-volume-1
ruif12@ubuntu:~/docker_dir$
```

Figura 4 - Comando questão 2.

De modo a obtermos mais informações sobre um volume é necessário a utilização de um comando próprio (volume *inspect*) que é representado na figura seguinte.



```
ruif12@ubuntu: ~/docker_dir
ruif12@ubuntu: ~/docker_dir 80x24
ruif12@ubuntu:~/docker_dir$ docker volume create my-volume-1
my-volume-1
ruif12@ubuntu:~/docker_dir$ docker volume inspect my-volume-1
[
  {
    "CreatedAt": "2021-12-15T22:19:41Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/my-volume-1/_data",
    "Name": "my-volume-1",
    "Options": {},
    "Scope": "local"
  }
]
ruif12@ubuntu:~/docker_dir$
```

Figura 5 - Comando para obter informações de um volume.

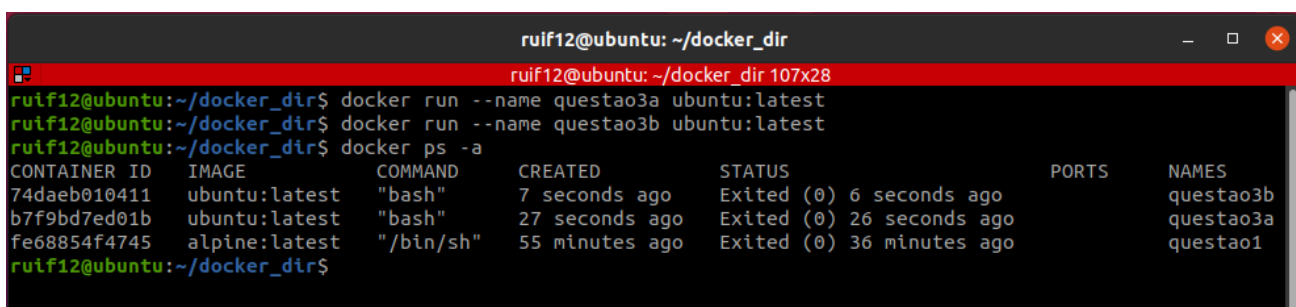
Com esta informação é possível responder às 2 alíneas restantes da questão 2 em que pergunta qual é o *mountpoint* do volume criado e que *driver* está a ser usada. Quanto ao *mountpoint* este está presente na diretoria “/var/lib/docker/volumes/my-volume-1/\_data”. Já o *driver* a ser usado podemos concluir que se trata de um driver local.

## Questão 3

**Q:** Create two basic containers (They should not be attached to any manually created network).

- Is it possible to inspect the bridge network and find their IP addresses?
- Is it possible for the containers to communicate among themselves using their names?
- And with their IPs?

**A:** De modo a criar 2 container básicos foram utilizados os comandos presentes na figura 6 dando o nome *questao3a* a um dos containers e *questao3b* ao outro. Para além disso realizamos o comando *"docker ps -a"* de modo a observar que os containers foram efetivamente criados.

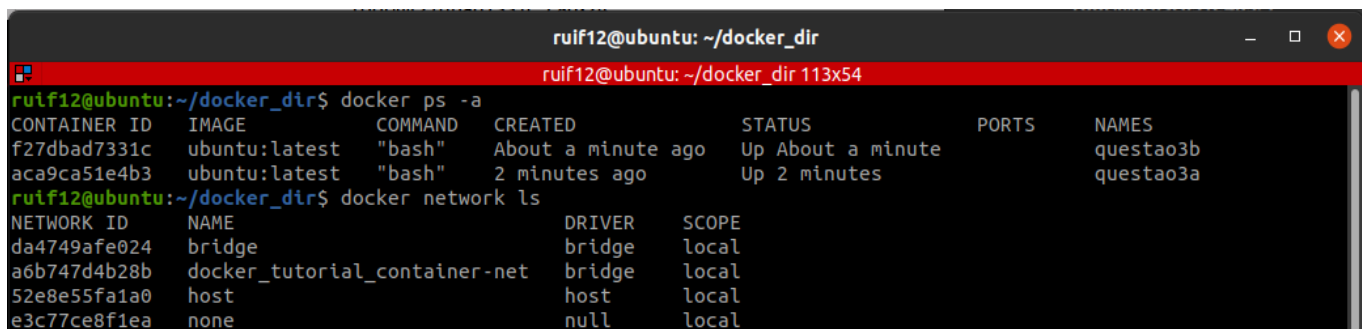


```

ruif12@ubuntu: ~/docker_dir
ruif12@ubuntu: ~/docker_dir 107x28
ruif12@ubuntu:~/docker_dir$ docker run --name questao3a ubuntu:latest
ruif12@ubuntu:~/docker_dir$ docker run --name questao3b ubuntu:latest
ruif12@ubuntu:~/docker_dir$ docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
74daeb010411   ubuntu:latest  "bash"          7 seconds ago    Exited (0) 6 seconds ago
b7f9bd7ed01b   ubuntu:latest  "bash"          27 seconds ago   Exited (0) 26 seconds ago
fe68854f4745   alpine:latest  "/bin/sh"       55 minutes ago   Exited (0) 36 minutes ago
ruif12@ubuntu:~/docker_dir$
  
```

Figura 6 - Comando questão 3.

Através da figura 7 podemos observar a utilização do comando *"docker network ls"* para podermos observar as redes ligadas ao docker onde podemos observar a rede *bridge* que iremos obter mais informação através do uso do comando *"docker network inspect bridge"* na figura 8.



```

ruif12@ubuntu: ~/docker_dir
ruif12@ubuntu: ~/docker_dir 113x54
ruif12@ubuntu:~/docker_dir$ docker ps -a
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
f27dbad7331c   ubuntu:latest  "bash"          About a minute ago    Up About a minute
aca9ca51e4b3   ubuntu:latest  "bash"          2 minutes ago        Up 2 minutes
ruif12@ubuntu:~/docker_dir$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
da4749afe024   bridge              bridge    local
a6b747d4b28b   docker_tutorial_container-net  bridge    local
52e8e55fa1a0   host                host      local
e3c77ce8f1ea   none                null      local
  
```

Figura 7 - Comando para ver as redes ligadas ao docker.

Como dito anteriormente na figura 8 está presente a obtenção de mais informação relativa à rede *bridge*. A partir desta informação é possível a visualização dos IPs pedidos na alínea a) desta questão. Estes IPs são então o '172.17.0.3/16' para o primeiro container e '172.17.0.2/16' para o segundo.

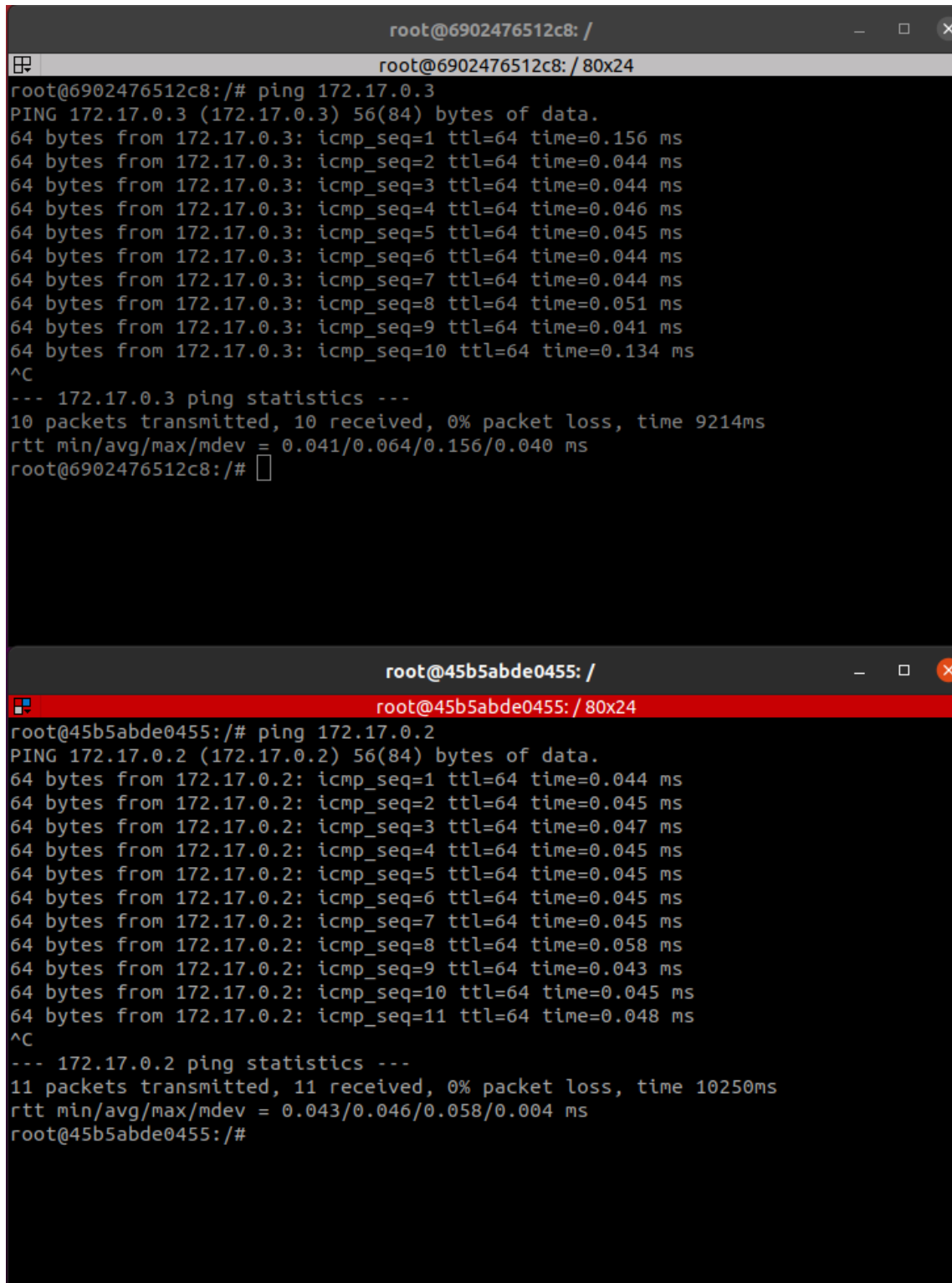
```
ruif12@ubuntu:~$ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "94c33915a16db19d0b14930099c192172bb11909621eb424d23cbdee49c2dad",
    "Created": "2021-12-15T23:55:59.51390719Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "45b5abde045530cef5f6ff0b43896edbcd4a3a6119ce2a2401b0277d68078294": {
        "Name": "questao3b",
        "EndpointID": "5500062c3d35c5e2f794012646233b4e5db7a6c3af2e90b03eec2dc4c11d6a1d",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      },
      "6902476512c88468376c89e65241272e5d4686edc312f97ed8f3f2463b4ffcb2": {
        "Name": "questao3a",
        "EndpointID": "8118d404086ca00a95b03dd02d1eb7ab46cb512c78cbb44a716082fd66c451e3",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

Figura 8 - Informação relativa à bridge network.

Quanto à alínea b) da questão 3 esta pergunta se é possível os containers comunicarem entre eles usando os seus nomes. Isto é verdade se ambos estiverem na mesma rede, no entanto como no enunciado da questão diz que os containers não devem estar ligados a nenhuma rede manual criada então a resposta deixa de ser positiva.



No que toca à última alínea desta questão como conseguimos obter os IPs dos containers através do método *'inspect bridge'* é possível realizar *ping* entre os containers como comprovado pela figura seguinte.



```
root@6902476512c8: /  
root@6902476512c8: / 80x24  
root@6902476512c8: /# ping 172.17.0.3  
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.  
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.156 ms  
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.044 ms  
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.044 ms  
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.046 ms  
64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.044 ms  
64 bytes from 172.17.0.3: icmp_seq=7 ttl=64 time=0.044 ms  
64 bytes from 172.17.0.3: icmp_seq=8 ttl=64 time=0.051 ms  
64 bytes from 172.17.0.3: icmp_seq=9 ttl=64 time=0.041 ms  
64 bytes from 172.17.0.3: icmp_seq=10 ttl=64 time=0.134 ms  
^C  
--- 172.17.0.3 ping statistics ---  
10 packets transmitted, 10 received, 0% packet loss, time 9214ms  
rtt min/avg/max/mdev = 0.041/0.064/0.156/0.040 ms  
root@6902476512c8: /#  
  
root@45b5abde0455: /  
root@45b5abde0455: / 80x24  
root@45b5abde0455: /# ping 172.17.0.2  
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.  
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.044 ms  
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.047 ms  
64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=5 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=6 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=7 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=8 ttl=64 time=0.058 ms  
64 bytes from 172.17.0.2: icmp_seq=9 ttl=64 time=0.043 ms  
64 bytes from 172.17.0.2: icmp_seq=10 ttl=64 time=0.045 ms  
64 bytes from 172.17.0.2: icmp_seq=11 ttl=64 time=0.048 ms  
^C  
--- 172.17.0.2 ping statistics ---  
11 packets transmitted, 11 received, 0% packet loss, time 10250ms  
rtt min/avg/max/mdev = 0.043/0.046/0.058/0.004 ms  
root@45b5abde0455: /#
```

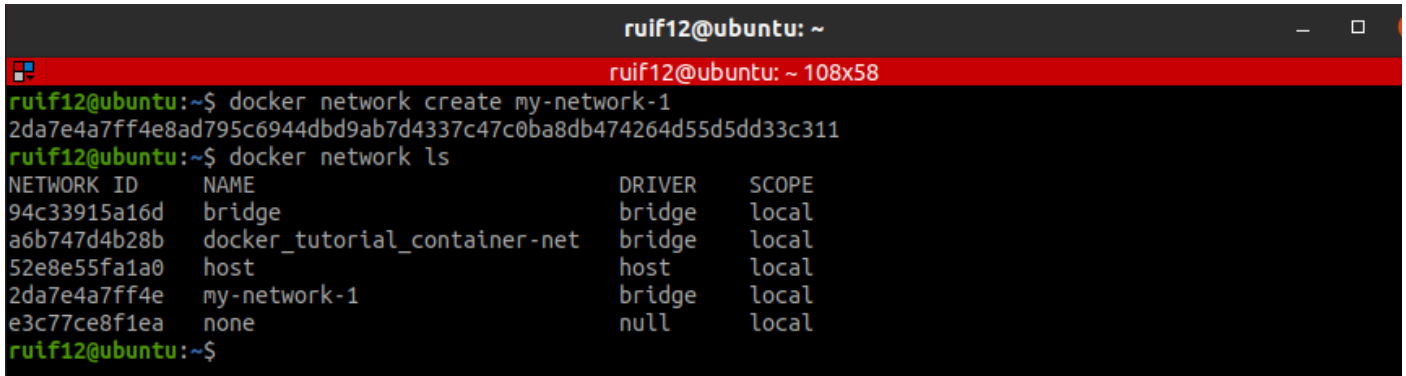
Figura 9 - Comando ping entre os containers.

## Questão 4

Q: What is the command to create the network “my-network-1”?

- (a) How to attach two containers to that network?
- (b) Which relevant parameter is possible to found about that network?
- (c) Can the containers ping one another using their name?

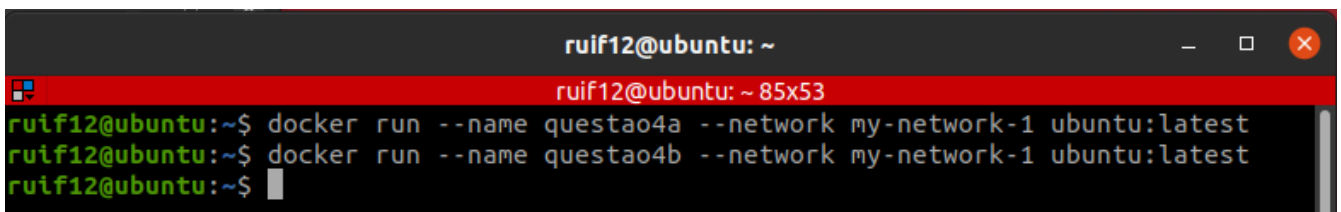
A: De modo a responder à questão 4 realizamos o comando demonstrado na figura 10 que cria uma rede no docker. Para comprovar a sua criação realizamos também o comando ‘*docker network ls*’ que apresenta as várias redes conectadas ao docker. Com a sua visualização podemos observar que na linha 4 encontra-se a rede por nós criada.



```
ruif12@ubuntu: ~  
ruif12@ubuntu: ~ 108x58  
ruif12@ubuntu:~$ docker network create my-network-1  
2da7e4a7ff4e8ad795c6944dbd9ab7d4337c47c0ba8db474264d55d5dd33c311  
ruif12@ubuntu:~$ docker network ls  
NETWORK ID          NAME                                DRIVER  SCOPE  
94c33915a16d        bridge                            bridge  local  
a6b747d4b28b        docker_tutorial_container-net     bridge  local  
52e8e55fa1a0        host                              host     local  
2da7e4a7ff4e        my-network-1                      bridge  local  
e3c77ce8f1ea        none                              null     local  
ruif12@ubuntu:~$
```

Figura 10 - Comando questão 4.

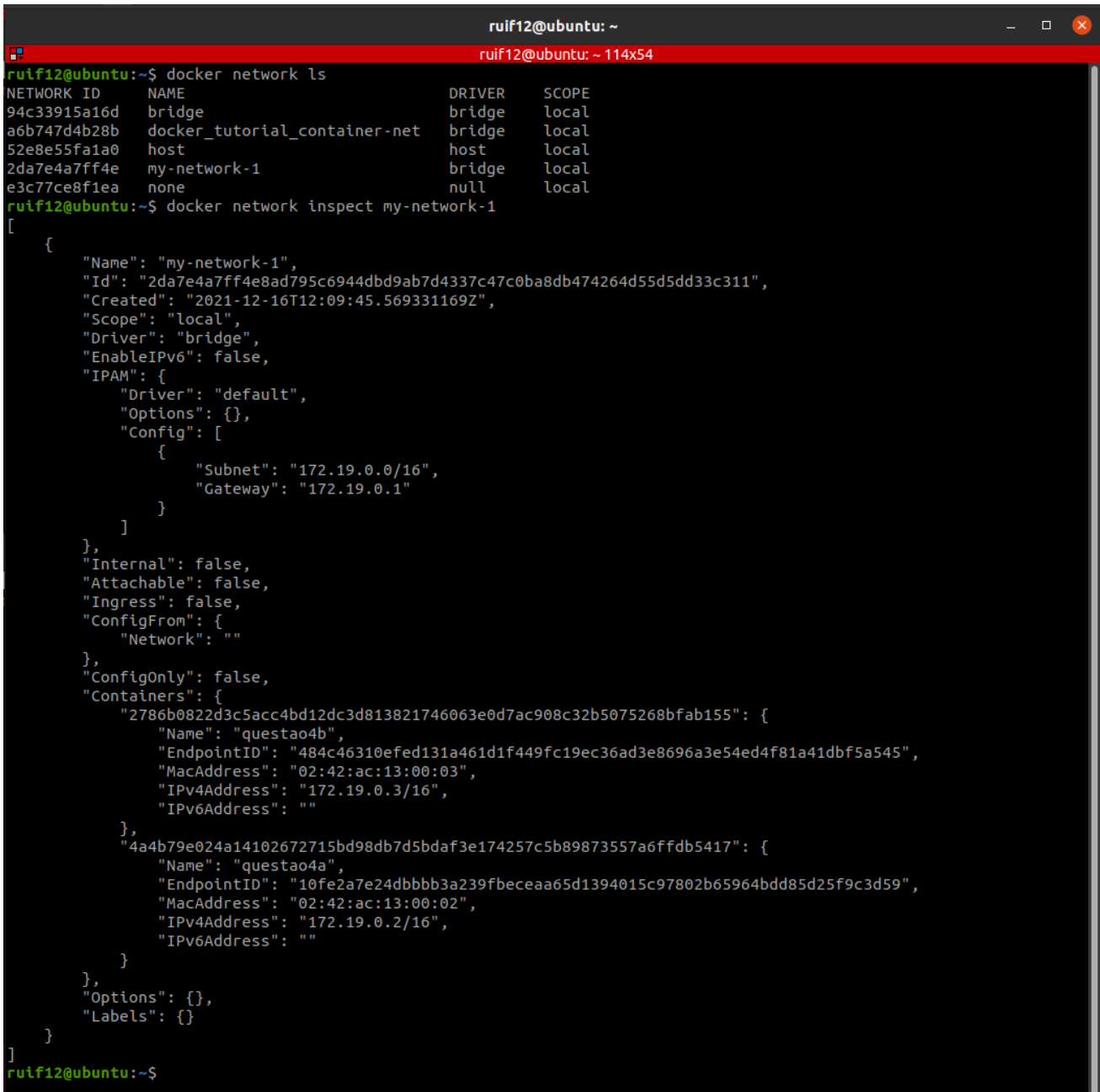
Quanto à questão 4a) que pergunta como se adiciona 2 containers à rede por nós criada é necessário realizar os comandos demonstrados na seguinte figura.



```
ruif12@ubuntu: ~  
ruif12@ubuntu: ~ 85x53  
ruif12@ubuntu:~$ docker run --name questao4a --network my-network-1 ubuntu:latest  
ruif12@ubuntu:~$ docker run --name questao4b --network my-network-1 ubuntu:latest  
ruif12@ubuntu:~$
```

Figura 11 - Criação de containers na rede my-network-1.

Com o objetivo de comprovar que os containers criados pertencem de facto à rede por nós criadas decidimos realizar o seguinte comando onde podemos comprovar a existência de 2 containers com os nomes “questao4a” e “questao4b”.

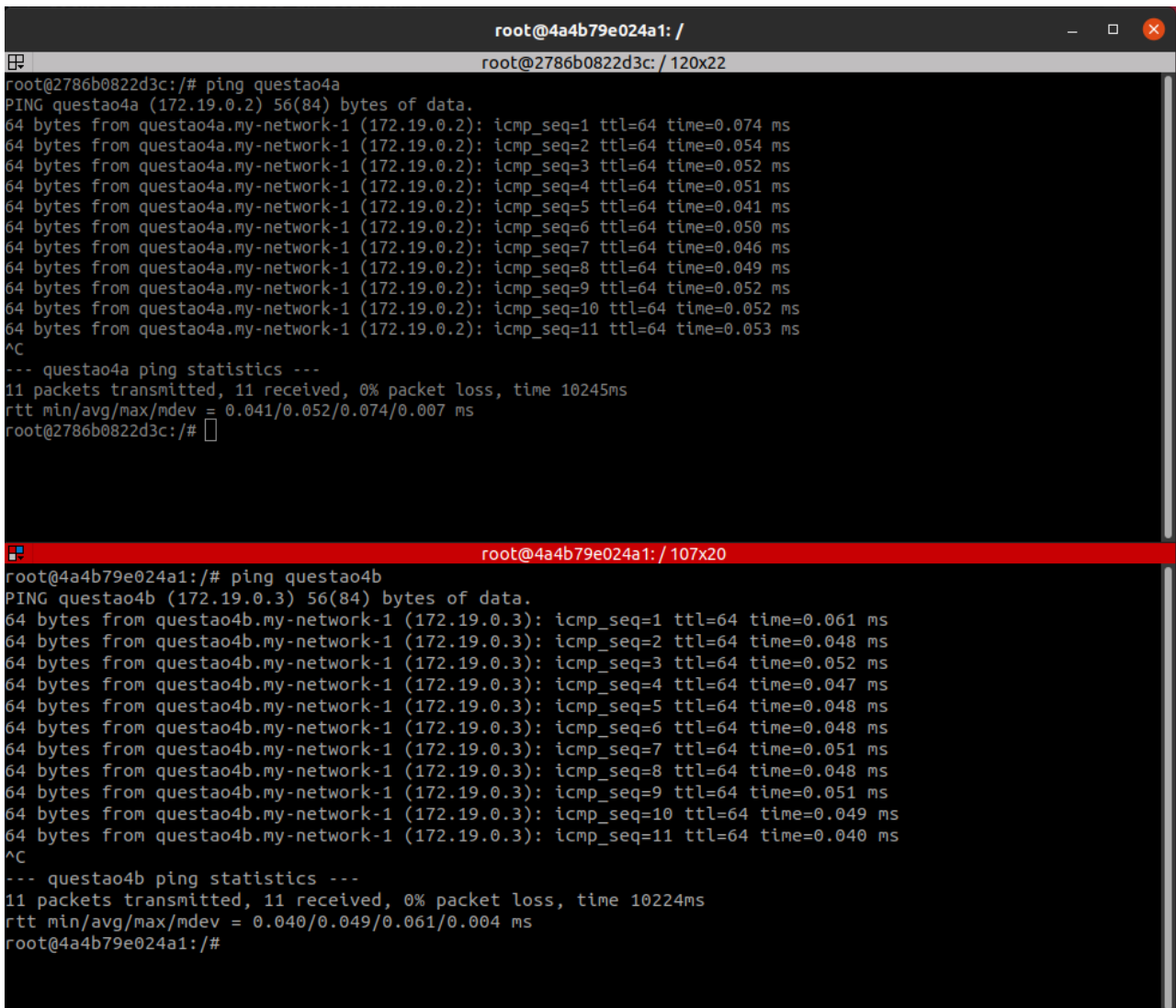


```
ruif12@ubuntu: ~  
ruif12@ubuntu: ~ 114x54  
ruif12@ubuntu:~$ docker network ls  
NETWORK ID          NAME                                DRIVER  SCOPE  
94c33915a16d        bridge                            bridge  local  
a6b747d4b28b        docker_tutorial_container-net     bridge  local  
52e8e55fa1a0        host                             host    local  
2da7e4a7ff4e        my-network-1                     bridge  local  
e3c77ce8f1ea        none                             null    local  
ruif12@ubuntu:~$ docker network inspect my-network-1  
[  
  {  
    "Name": "my-network-1",  
    "Id": "2da7e4a7ff4e8ad795c6944dbd9ab7d4337c47c0ba8db474264d55d5dd33c311",  
    "Created": "2021-12-16T12:09:45.569331169Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.19.0.0/16",  
          "Gateway": "172.19.0.1"  
        }  
      ]  
    },  
    "Internal": false,  
    "Attachable": false,  
    "Ingress": false,  
    "ConfigFrom": {  
      "Network": ""  
    },  
    "ConfigOnly": false,  
    "Containers": {  
      "2786b0822d3c5acc4bd12dc3d813821746063e0d7ac908c32b5075268bfab155": {  
        "Name": "questao4b",  
        "EndpointID": "484c46310efed131a461d1f449fc19ec36ad3e8696a3e54ed4f81a41dbf5a545",  
        "MacAddress": "02:42:ac:13:00:03",  
        "IPv4Address": "172.19.0.3/16",  
        "IPv6Address": ""  
      },  
      "4a4b79e024a14102672715bd98db7d5bdaf3e174257c5b89873557a6ffdb5417": {  
        "Name": "questao4a",  
        "EndpointID": "10fe2a7e24dbbb3a239fbeeaa65d1394015c97802b65964bdd85d25f9c3d59",  
        "MacAddress": "02:42:ac:13:00:02",  
        "IPv4Address": "172.19.0.2/16",  
        "IPv6Address": ""  
      }  
    },  
    "Options": {},  
    "Labels": {}  
  }  
]  
ruif12@ubuntu:~$
```

Figura 12 - Network com os containers criados.

No que toca à questão b) que pergunta quais os parâmetros relevantes que podemos descobrir sobre essa rede é necessário realizar o comando demonstrado na figura 13 onde podemos retirar bastante informação como por exemplo os containers e os seus respetivos IPs e nomes. Para além disso podemos retirar a data de quando foi criada a respetiva rede ou por exemplo o seu ID. Quanto ao parâmetro que pensamos ser o mais relevante é o IPAM onde podemos retirar informação sobre o *Subnet* e o *Gateway* que dão indicações de quais a gama de valores de IPs dos containers.

Relativamente à alínea c da questão 4 podemos responder de forma positiva visto que como ambos os containers estão na mesma rede estes têm informação suficiente para realizarem *ping* entre estes apenas utilizando os seus nomes. Isto pode ser comprovado observando a figura seguinte.



```
root@4a4b79e024a1: /  
root@2786b0822d3c: / 120x22  
root@2786b0822d3c:/# ping questao4a  
PING questao4a (172.19.0.2) 56(84) bytes of data.  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=1 ttl=64 time=0.074 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=2 ttl=64 time=0.054 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=3 ttl=64 time=0.052 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=4 ttl=64 time=0.051 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=5 ttl=64 time=0.041 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=6 ttl=64 time=0.050 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=7 ttl=64 time=0.046 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=8 ttl=64 time=0.049 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=9 ttl=64 time=0.052 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=10 ttl=64 time=0.052 ms  
64 bytes from questao4a.my-network-1 (172.19.0.2): icmp_seq=11 ttl=64 time=0.053 ms  
^C  
--- questao4a ping statistics ---  
11 packets transmitted, 11 received, 0% packet loss, time 10245ms  
rtt min/avg/max/mdev = 0.041/0.052/0.074/0.007 ms  
root@2786b0822d3c:/#  
  
root@4a4b79e024a1: / 107x20  
root@4a4b79e024a1:/# ping questao4b  
PING questao4b (172.19.0.3) 56(84) bytes of data.  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=1 ttl=64 time=0.061 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=2 ttl=64 time=0.048 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=3 ttl=64 time=0.052 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=4 ttl=64 time=0.047 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=5 ttl=64 time=0.048 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=6 ttl=64 time=0.048 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=7 ttl=64 time=0.051 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=8 ttl=64 time=0.048 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=9 ttl=64 time=0.051 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=10 ttl=64 time=0.049 ms  
64 bytes from questao4b.my-network-1 (172.19.0.3): icmp_seq=11 ttl=64 time=0.040 ms  
^C  
--- questao4b ping statistics ---  
11 packets transmitted, 11 received, 0% packet loss, time 10224ms  
rtt min/avg/max/mdev = 0.040/0.049/0.061/0.004 ms  
root@4a4b79e024a1:/#
```

Figura 13 - Ping entre os containers.

## Questão 5

**Q:** Which were the results obtained from the commands `docker network ls`, `docker network inspect` and `docker volume ls` in section 1 of this document? What conclusions is possible to take from the result of these commands?

**A:** Na realização da secção 1 do documento realizamos vários comandos para nos adaptarmos às funcionalidades do docker. Em seguida apresentaremos esses comandos assim como uma breve explicação da utilização e propósito de cada um.

Começando pelo comando que nos permite visualizar as redes que o docker tem conhecimento. Através do comando *“docker network ls”* podemos observar que estão 5 redes conectadas ao docker incluindo a criada na questão anterior, a *‘my-network-1’*.

```
ruif12@ubuntu: ~  
ruif12@ubuntu: ~ 114x54  
ruif12@ubuntu:~$ docker network ls  
NETWORK ID          NAME                                DRIVER  SCOPE  
94c33915a16d        bridge                             bridge  local  
a6b747d4b28b        docker_tutorial_container-net     bridge  local  
52e8e55fa1a0        host                               host    local  
2da7e4a7ff4e        my-network-1                      bridge  local  
e3c77ce8f1ea        none                               null    local  
ruif12@ubuntu:~$
```

Figura 14 - Comando *docker network ls*.

Na figura anterior podemos então verificar as redes que estão efetivamente conectadas ao docker. Estas redes são o que permite que os container e volumes consigam realizar conexões uns com os outros. No que toca à rede *bridge* esta é uma rede especial em que caso não seja especificado o contrário é utilizado por *default* por parte do docker. Esta rede permite a conexão entre os vários containers e pode ser observada a sua informação através do comando *“docker network inspect bridge”* demonstrado na figura 16 na página seguinte. A seguir apresentamos uma imagem ilustrativa da maneira de como a rede *bridge* funciona no docker permitindo a conexão entre os vários containers a este ligado.

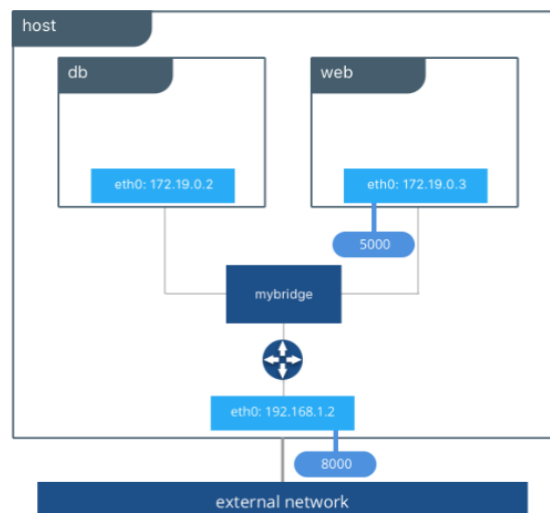


Figura 15 - Esquema da rede no docker.

Quanto ao comando *'docker network inspect'* este necessita de mais um argumento de modo a indicar qual rede ligada ao *docker* pretendemos obter informações. De seguida apresentamos um exemplo do que era apresentado ao utilizador após inserir o comando *'docker network inspect bridge'* que retorna informações sobre a rede *bridge* do *docker*.

```

rui12@ubuntu:~$ docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "105573bf8418885dc4c2a41b8ca3a4157e341aa13859ccc2fc9e7f367bf7b987",
    "Created": "2021-12-10T14:00:29.69161757Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "748f7cbff8138254d2dd5a4f59581b4e4ee2de8966ada5c837d2d19ee79184ca": {
        "Name": "questao3b",
        "EndpointID": "e9401eeef091a2f83d12704d43c298f8dacc5e5d7b3d14950850fca3f894b56",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      },
      "aa2be7eeb86e2378e7a02bd06b49c8414f8f66f06e24ba3721a4e88eb3fee29": {
        "Name": "questao3a",
        "EndpointID": "aebee5c7838bb8398d48875799f5c5cf4deb8c754936a44dea484d7885581f91cc",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]

```

Figura 16 - Comando *docker network inspect*.

Já o comando *'docker volume ls'* retorna todos os volumes conhecidos pelo *docker*. Os volumes são necessários para preservar dados caso um container seja destruído pois quando este é destruído o seu sistema de ficheiros é destruído também. Na figura seguinte é apresentado os vários volumes conhecidos pelo *docker* e na figura 18 uma representação do esquema do funcionamento dos volumes no *docker*.

```

rui12@ubuntu:~$ docker volume ls
DRIVER      VOLUME NAME
local       docker_tutorial_httpd-vol
local       my-volume-1
local       new-test-vol

```

Figura 17 - Comando *docker volume ls*.

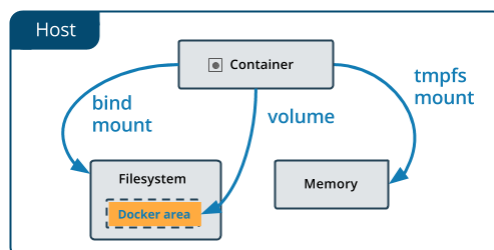


Figura 18 - Esquema dos volumes no *docker*.

Quanto às conclusões que podemos retirar dos resultados destes comandos é que estes servem como uma forma de *debug* para ver as informações que forem necessárias sobre as redes e/ou volumes dos *dockers*.

## Questão 6

**Q:** Create a docker-compose that starts at least two services:

- (a) These should be in the same docker network;
- (b) These should share the same docker volume;
- (c) One of the services should also have one bind mount;
- (d) One of the services should have their 9999 port exposed in the 8888 host port;
- (e) Using the inspect and ls commands, what conclusions can you take about their results?

**A:** Para a resolução desta questão foi necessário o conhecimento aprofundado de *composes* no docker que muito resumidamente tratam de executar várias aplicações presentes em containers ao mesmo tempo e após isso criamos um que iniciasse 2 serviços como demonstrado na figura seguinte.

```
docker-compose.yml
1  version: '3.2'
2  services:
3    service1:
4      image: httpd:latest
5      ports:
6        - "8080:80"
7      volumes:
8        - tp1_vol:/usr/local/apache2
9      networks:
10       - tp1_net
11    service2:
12      image: httpd:latest
13      expose:
14        - "9999"
15      ports:
16        - "8888:80"
17      volumes:
18        - type: bind
19          source: ../usr/local/apache2
20          target: /home/tp1
21      networks:
22        - tp1_net
23  volumes:
24    tp1_vol:
25  networks:
26    tp1_net:
27
```

Figura 19 - Docker compose.

Na figura anterior podemos então observar o ficheiro "*docker-compose.yml*" onde é realizada a configuração do nosso *compose* em que no nosso caso criamos 1 volume e 1 uma rede partilhada por ambos os serviços em que estes têm portas diferentes uma com a 9999 exposta e ambos contêm a imagem *httpd:latest*.

De modo a observar o correto funcionamento do nosso *compose* podemos ver na figura seguinte uma captura do *browser* com os *localhosts* abertos nas portas 8888 e 8080 em que podemos observar ambos a funcionarem, pois em ambos conseguimos ver a mensagem “It works!”.

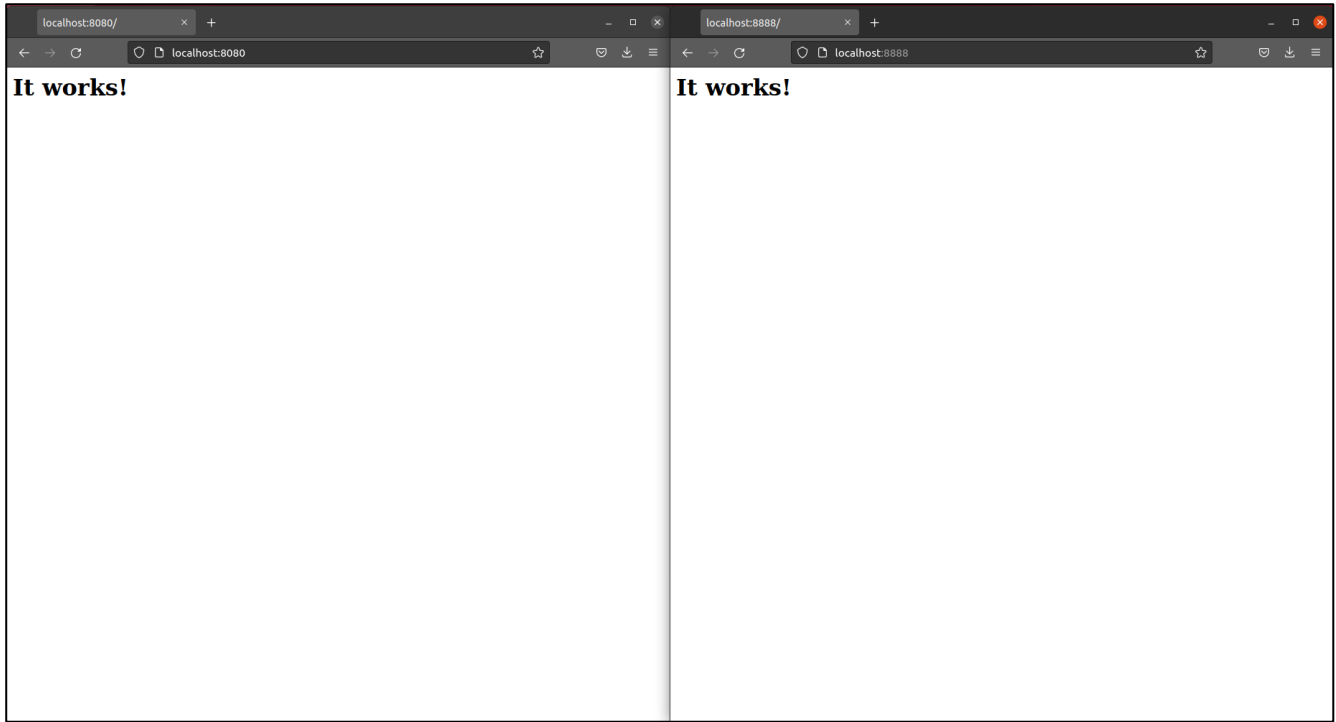


Figura 20 - Funcionamento de ambos os serviços.

Para além disso ao realizarmos o comando “*docker network inspect compose\_tp1\_vol*” podemos observar o volume criado para os dois containers, no entanto não conseguimos obter a informação de que ambos pertencem a este.

```
ruif12@ubuntu: ~/Desktop/Virtualizacao/compose
ruif12@ubuntu: ~/Desktop/Virtualizacao/compose 111x51
ruif12@ubuntu:~/Desktop/Virtualizacao/compose$ docker volume inspect compose_tp1_vol
[
  {
    "CreatedAt": "2021-12-18T17:41:33Z",
    "Driver": "local",
    "Labels": {
      "com.docker.compose.project": "compose",
      "com.docker.compose.version": "1.29.2",
      "com.docker.compose.volume": "tp1_vol"
    },
    "Mountpoint": "/var/lib/docker/volumes/compose_tp1_vol/_data",
    "Name": "compose_tp1_vol",
    "Options": null,
    "Scope": "local"
  }
]
ruif12@ubuntu:~/Desktop/Virtualizacao/compose$
```

Figura 21 - Volume partilhado por ambos os containers.



Já quando observamos a rede criada dentro do *compose* com auxílio do comando “*docker network inspect compose\_tp1\_net*” podemos ver que ambos os containers estão presentes nesta rede como demonstrado na figura seguinte.

```
ruif12@ubuntu: ~/Desktop/Virtualizacao/compose
ruif12@ubuntu: ~/Desktop/Virtualizacao/compose 111x51
ruif12@ubuntu:~/Desktop/Virtualizacao/compose$ docker network inspect compose_tp1_net
[
  {
    "Name": "compose_tp1_net",
    "Id": "55b24656bc6923b7e7439e814e77496c10b5c8f23e993fa8c23e923c992dd7c7",
    "Created": "2021-12-18T17:22:01.173504599Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.25.0.0/16",
          "Gateway": "172.25.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "a217c67dd3e454348c141f0bd83d9486e4c56fab70ff2e0848dd03c3da0d83bb": {
        "Name": "compose_service1_1",
        "EndpointID": "85df9994b7ed29da16319775b45d5a32c9895b6fc71169e291a6999de50c79c6",
        "MacAddress": "02:42:ac:19:00:02",
        "IPv4Address": "172.25.0.2/16",
        "IPv6Address": ""
      },
      "ed7929c8560254865ce5da6bf6515b18c4975671d21a4445d8a9dfbb4c0d1a23": {
        "Name": "compose_service2_1",
        "EndpointID": "d212c852b701262d3cd043f80c0c4295bb8d525fa421039b9c8bbbf69f756bd30",
        "MacAddress": "02:42:ac:19:00:03",
        "IPv4Address": "172.25.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {
      "com.docker.compose.network": "tp1_net",
      "com.docker.compose.project": "compose",
      "com.docker.compose.version": "1.29.2"
    }
  }
]
```

Figura 22 - Rede partilhada por ambos os containers.

## Questão 7

**Q:** Create a Docker file that:

- (a) Have some tool installed. Any that is chosen by the student. Extra points for any application that is listening from outside requests (web server, for example);
- (b) Has a volume, for later persistence;

**A:** Depois de pensar em várias opções para resolver esta questão, optamos por criar uma base de dados que contem informações de vários alunos e uma API que vai buscar a base de dados de todos os alunos.

Em primeiro lugar criamos um Dockerfile representado na figura 23 onde é configurada a imagem que criamos, neste caso trata-se de uma imagem MYSQL, em seguida construímos a imagem do *mysql* através do comando representado na figura 24.

```
api > db > Dockerfile
1 FROM mysql
2 ENV MYSQL_ROOT_PASSWORD 12345
3
```

Figura 23 - Dockerfile da base de dados.

```
→ ex7 docker build -t msyql-image -f api/db/Dockerfile .
Sending build context to Docker daemon 6.508MB
Step 1/2 : FROM mysql
---> bb6f571db497
Step 2/2 : ENV MYSQL_ROOT_PASSWORD 12345
---> Using cache
---> ac772f8fe231
Successfully built ac772f8fe231
Successfully tagged msyql-image:latest
→ ex7
```

Figura 24 - Comando para criar a *mysql-image*.

Para resolver a aliena b) acrescentamos ao comando para executar o container do mysql (figura 25) a *flag* '-v', esta permite que o *host* e o container partilhem uma pasta.

```
→ ex7 docker run -d -v $(pwd)/api/db/data:/var/lib/mysql --rm --name mysql-cointainer mysql-image
7fd9077336a2125e5393295dc2041c007b10f9e2b7375444f7856114081add55
→ ex7
```

Figura 25 - Comando para executar o *mysql* container.

O próximo passo foi criar o container onde iria correr o node, para isto foi criado um ficheiro Dockerfile onde estão as instruções mais importantes como demonstrado na figura 26. Na figura 27 está exibido o comando para executar o container do node.

```
api > Dockerfile
1 FROM node:10-slim
2 WORKDIR /home/node/app
3 CMD npm start
4
5
6
```

Figura 26 - Dockerfile do node.

```
→ ex7 docker run -d -v $(pwd)/api:/home/node/app -p 9001:9001 --rm --name node-container node-image
ca51eb29e13a45789daaf2c1ce900e5cfa972a3752485682546fe45119b1dd86
→ ex7
```

Figura 27 - Comando para executar o node container.

Relativamente ao código realizado para a base de dados e para a apresentação no browser são apresentados nas próximas figuras sendo que a figura 23 corresponde à base de dados no MySQL onde adicionamos 2 alunos.

```
api > db > script.sql
1 CREATE DATABASE IF NOT EXISTS
2   alunos;
3 USE alunos;
4
5 CREATE TABLE IF NOT EXISTS identificacao (
6   name VARCHAR(255),
7   number VARCHAR(255),
8   email VARCHAR(255),
9   PRIMARY KEY(number)
10 );
11
12 INSERT INTO identificacao VALUE('Rui Filipe Ribeiro Freitas','PG47639','pg47639@alunos.uminho.pt');
13 INSERT INTO identificacao VALUE('Tiago Joao Pereira Ferreira','PG47692','pg47692@alunos.uminho.pt')
14
```

Figura 28 - Código SQL para a base de dados.

Quanto ao código realizado em JavaScript para apresentação dos resultados provenientes da base de dados no browser para melhor observação é apresentado na figura 24 sendo que apenas foi realizado um simples método em que a *response* é proveniente da base de dados. Como o foco deste trabalho não é a apresentação do browser não foi tido como crucial a sua execução pelo que este é bastante simples.

```
api > src > index.js > ...
1 const express = require('express');
2 const mysql = require('mysql');
3
4 const app = express();
5
6 const connection = mysql.createConnection({
7   host: '172.17.0.2',
8   user: 'root',
9   password: '12345',
10  database: 'alunos'
11 });
12
13 connection.connect();
14
15 app.get('/identificacao', function(req, res){
16   connection.query('SELECT * FROM identificacao', function(error, results){
17     if(error){
18       throw error
19     };
20     res.send(results.map(item => ({name: item.name, number: item.number, email: item.email})));
21   });
22 });
23
24 app.listen(9001, '0.0.0.0', function(){
25   console.log('Listening on port 9001')
26 })
```

Figura 29 - Código Javascript.

Com os dois containers a correr podemos aceder então aos dados dos alunos através do *endpoint* “localhost:9001/identificacao” como representado na figura seguinte.

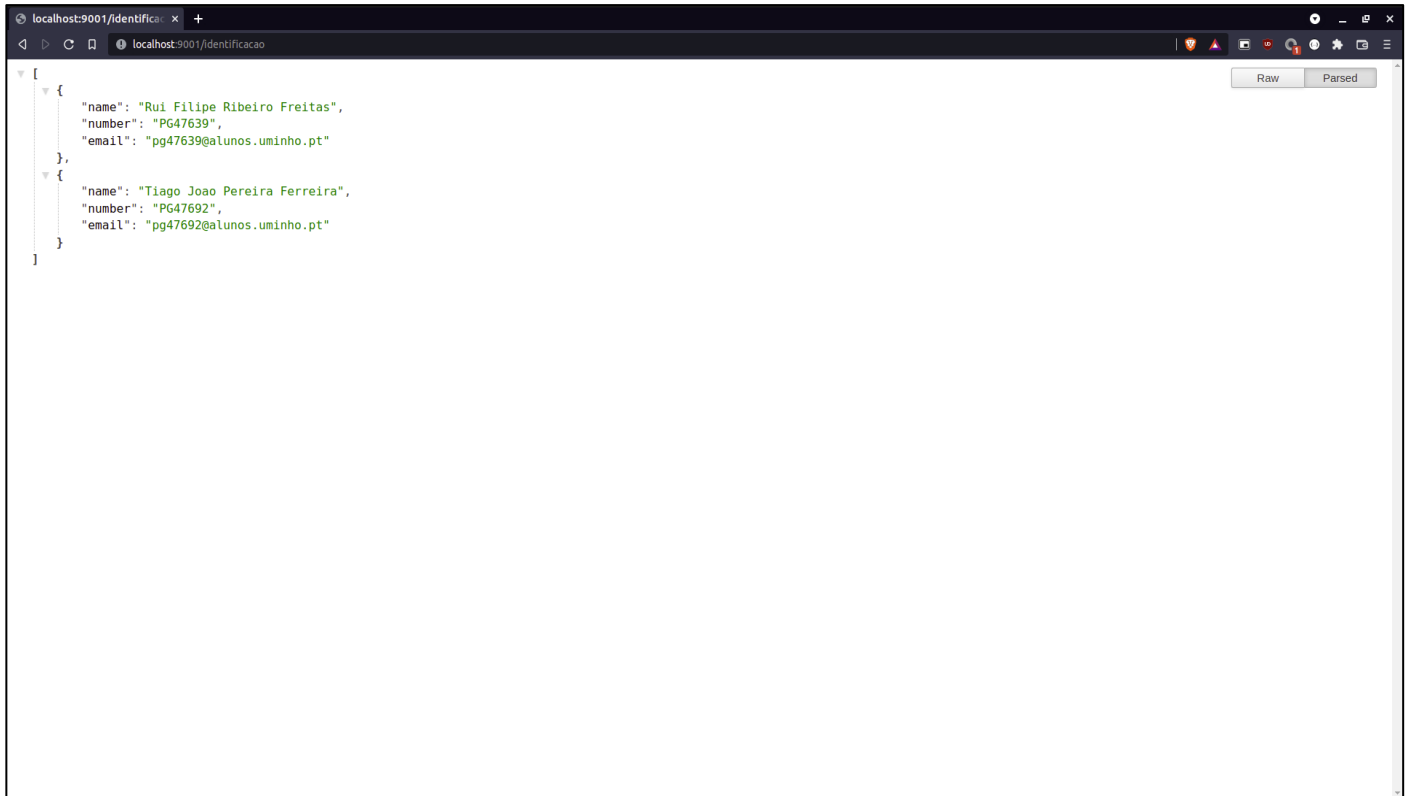


Figura 30 - Endpoint.

Para além disso usufruindo do comando ‘docker ps’ podemos observar que ambos os containers se encontram ativos verificando assim que tudo funciona como pretendido. Isto é demonstrado na figura seguinte.

```
➔ ~ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                               NAMES
2cef01e687fc   node-image "docker-entrypoint.s..." 4 hours ago   Up 4 hours   0.0.0.0:9001->9001/tcp, :::9001->9001/tcp   node-container
60b69b546b15   mysql-image "docker-entrypoint.s..." 4 hours ago   Up 4 hours   3306/tcp, 33060/tcp                  mysql-cointainer
```

Figura 31 - Comando docker ps.

## Conclusão

Com a realização deste trabalho prático ganhamos bastante conhecimento relativo a tudo o que engloba dockers e como estes funcionam. Achamos que foi um projeto enriquecedor visto tratar-se de uma tecnologia que tem vindo a aumentar substancialmente na área de virtualização.

Damos concluído este trabalho com o reconhecimento que conseguimos realizar tudo o que foi pedido menos a última questão visto ser necessário realizar um pagamento no Docker Hub pois só tendo a versão Pro é que seria possível realizar uma build automática do nosso docker.

Concluindo terminamos este trabalho com expetativas altas na sua apreciação visto termos realizado tudo o que era pedido dentro das nossas capacidades sabendo que conseguimos melhorar substancialmente as nossas capacidades no que diz respeito a dockers e seus atributos.

Uma última nota passa por oferecer uma alternativa à visualização do código visto que este não irá completo nos ficheiros enviados em conjunto com o relatório pois apenas é pedido os ficheiros Dockerfile e dos composes. Esta alternativa é através do GitHub que o grupo utilizou para guardar e partilhar código e que pode ser visto em <https://github.com/RuiFilipe2802/Virtualizacao> .