Docker

Paulo Bento A81139, Carlos Alves PG41840 and Bruno Rodrigues PG41066

Departamento de Informática, Universidade do Minho

Resumo O pequeno trabalho pretende responder a uma série de questões com o objetivo de aprofundar o conhecimento sobre a ferramenta Docker. Esta ferramenta permite a execução de sistemas em containers. Através desse mecanismo permite-se o deployment rápido de vários serviços em diversas máquinas.
Os exercícios realizados permite explorar as capacidades básicas da ferramenta, isto é, fazer pull de Imagens, criar volúmes, ligar containers através de networks, utilizar o Docker-compose e de Docker files.

Palavras-chave: Docker · Dockerfile · Dockercompose · volumes · bind · network

1 Container Linux Alpine

No primeiro exercicio pretende-se criar um container que tenha a port 9999 do container mapeada para a port 8668 do host. Também pretende-se criar um bind mount que mapeia uma diretoria do container numa diretoria do host.

Para tal executou-se o comando seguinte:

```
docker run -it --name alpine-container \
-v /home/apocas/docker_dir:/home/internal_dir \
-p 8668:9999 alpine:latest /bin/ash
```

O comando dá o nome de alpine-container ao container que foi criado, mapeando a diretoria (¨v¨) *home/apocas/docker_dir* do host para a diretoria */home/internal_dir* dentro do conatiner. Os ports foram mapeados com a opção (¨p¨). Finalmente, vai-se buscar a última versão da imagem do Alpine correndo o programa ash que corresponde ao intrepretador de comandos default do Sistema Operativo.

2 Volumes

Nesta questão pretende-se criar um volume com o nome ¨myvolume1¨para tal executou-se o comando:

```
docker volume create my-volume-1
```

Também se pretende saber qual é o mountpoint (local onde o volume faz as persistências) e o driver(Qual o host onde se vai montar) do mesmo. Para tal, executou-se o comando:

```
docker volume inspect test-vol
```

O resultado obtido do mesmo foi

```
[
        {
                "CreatedAt": "2020-02-20T11:02:31Z",
                "Driver": "local",
                "Labels": {},
                "Mountpoint": "/var/lib/docker/volumes
                /my-volume-1/_data",
                "Name": "my-volume-1",
                "Options": {},
                "Scope": "local"
        }
        ]
```

A partir do resultado podemos retirar que o mountpoint é */var/lib/docker/volumes/my-volume-1/_data* e que o Driver é *"local"*.

3 Bridge Network

A partir de dois containers criados manualmente, pretende-se inspecionar a Bridge Network para verificar os IPs dos mesmos e se é possível comunicar-se entre eles.

Antes de proceder a inspeção da Bridge Network criada por defeito, criou-se dois containers através dos comandos:

```
docker run -dit --name container1 ubuntu:latest /bin/bash

docker run -dit --name container2 ubuntu:latest /bin/bash
```

Sabendo que os IDs devolvidos para cada um deles são,

```
"01fb17f6ffd6a4ba02395719351d8a89b4d3d81047d854beddc987dcf904bbfb"

"2f2005b7c527e5c4656a6c023b8b5b4731eedf97fd8d9da5bfa181f1c757aa72"
```

Para inspecionar recorreu-se ao comando:

```
docker network inspect bridge
```

Obtendo-se o resultado que está na próxima figura:

Figura 1. Bridge Network

Pode-se constatar os IPs dos container associados as redes locais *172.17.0.4/16* e *172.17.0.3/16*.

Respondendo a questão da comunicação entre eles pode-se se constatar que , como dito no próprio, não se pode comunicar entre eles por nome visto que não estão na mesma network. No entanto, é possível comunicar usando os IPs do comando anterior como está representado na figura seguinte.

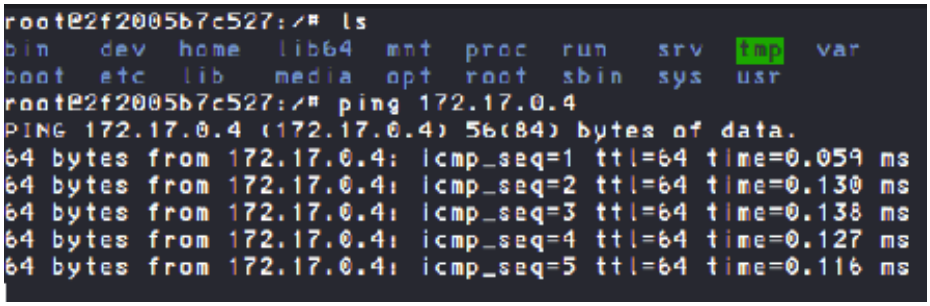


Figura 2. Comunição por IP através da Bridge Network

4 Manual Network

Neste exercicio pretende-se criar uma Network para juntar os dois containers e, verificar se já é possível pingar-se entre eles usando os nomes.

Para criar a rede com nome m̈y-network-1ütilizou-se o comando:

```
docker network create my-network-1
```

No caso de se quiser começar un novo container pode-se fazer:

```
docker run -dit --name container3 --network my-network-1 ubuntu:latest
/bin/bash
```

Assim cria-se un container a correr o Ubuntu em modo Daemon usando a bash como comando default ligando-o a network criada anteriormente.

Caso estivesse já correr os containers pode-se fazer:

```
docker network connect my-network-1 container1
```

Para inspecionar e encontrar mais informações sobre a network criada utiliza-se o comando:

```
docker network inspect my-network-1
```

Pode descobrir-se que qual o IP da subrede criada, como tambem os IPs dos container ligados a mesma. Tambem é possivel descobrir qual o Driver que esta network está a usar, quando foi criada e o seu nome.

Figura 3. Resultado do Inspect da Network

A comunicação dos containers ligados a network criada manualmente, pode ser feita através dos nomes dos mesmos como se pode ver na figura seguinte. Aqui o Container com ID Ïf...ëomunica com o Container de nome ëontainer1:¨

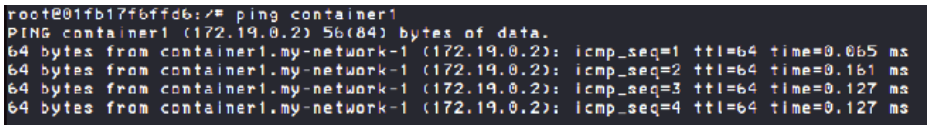
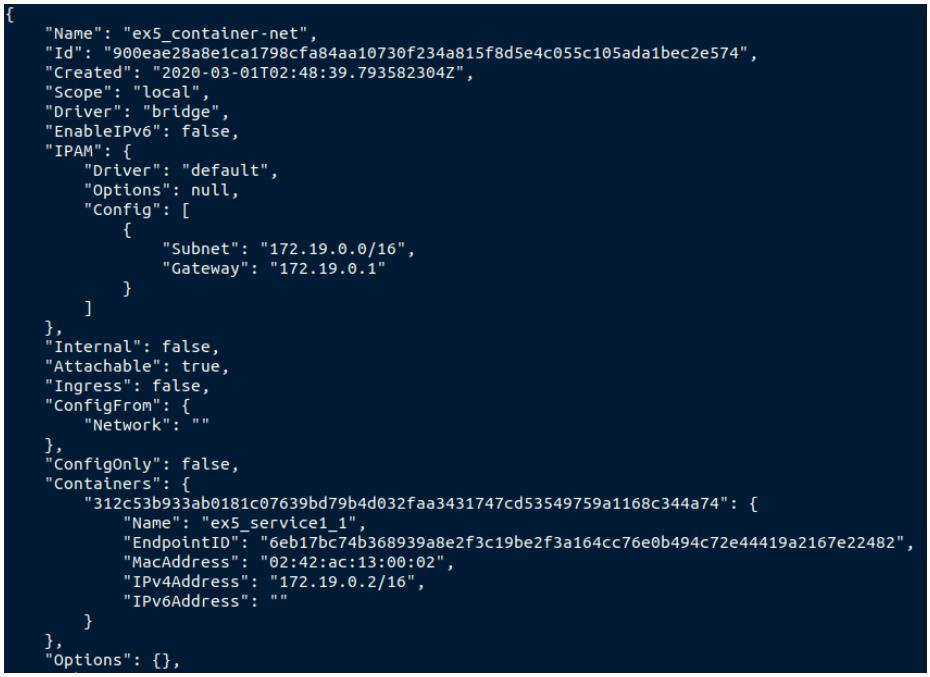


Figura 4. Ping aos Containers por Nome

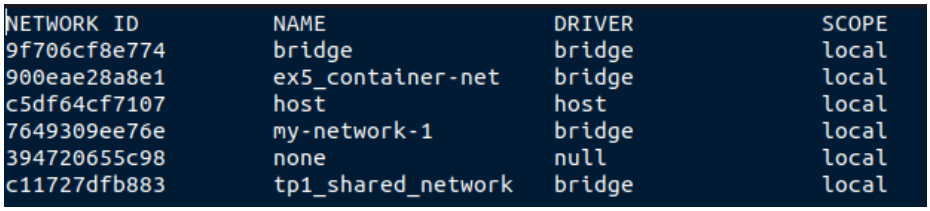
5 Analise de Comandos após Docker-Compose

Nesta seção pretende-se após executar o docker-compose(docker-compose -d) que se encontra na seção 2 do enunciado analisar o que este fez. Após a execução pode ver-se que foi criada a network ëx5_container-netë o volume ëx5_httpd-vol:¨ Os valores do volume e da network criada foram os default que normalmente são postos quando criados normalmente.



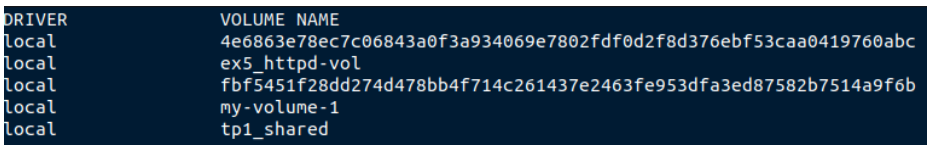
```
{
    "Name": "ex5_container-net",
    "Id": "900eae28a8e1ca1798cfa84aa10730f234a815f8d5e4c055c105ada1bec2e574",
    "Created": "2020-03-01T02:48:39.793582304Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
        "Driver": "default",
        "Options": null,
        "Config": [
            {
                "Subnet": "172.19.0.0/16",
                "Gateway": "172.19.0.1"
            }
        ]
    },
    "Internal": false,
    "Attachable": true,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "312c53b933ab0181c07639bd79b4d032faa3431747cd53549759a1168c344a74": {
            "Name": "ex5_service1_1",
            "EndpointID": "6eb17bc74b368939a8e2f3c19be2f3a164cc76e0b494c72e44419a2167e22482",
            "MacAddress": "02:42:ac:13:00:02",
            "IPv4Address": "172.19.0.2/16",
            "IPv6Address": ""
        }
    },
    "Options": {},
```


Figura 5. Resultado do Inspect da Network após o Docker-Compose



```
NETWORK ID          NAME                DRIVER              SCOPE
9f706cf8e774        bridge              bridge              local
900eae28a8e1        ex5_container-net   bridge              local
c5df64cf7107        host                host                local
7649309ee76e        my-network-1        bridge              local
394720655c98        none                null                local
c11727dfb883        tp1_shared_network  bridge              local
```


Figura 6. Resultado do LS das Networks após o Docker-Compose



```
DRIVER              VOLUME NAME
local               4e6863e78ec7c06843a0f3a934069e7802fdf0d2f8d376ebf53caa0419760abc
local               ex5_httpd-vol
local               fbf5451f28dd274d478bb4f714c261437e2463fe953dfa3ed87582b7514a9f6b
local               my-volume-1
local               tp1_shared
```


Figura 7. Resultado do LS dos Volumes após o Docker-Compose

```
[
    {
        "CreatedAt": "2020-03-01T02:48:44Z",
        "Driver": "local",
        "Labels": {
            "com.docker.compose.project": "ex5",
            "com.docker.compose.version": "1.25.3",
            "com.docker.compose.volume": "httpd-vol"
        },
        "Mountpoint": "/var/lib/docker/volumes/ex5_httpd-vol/_data",
        "Name": "ex5_httpd-vol",
        "Options": null,
        "Scope": "local"
    }
]
```


Figura 8. Resultado do Inspect do Volume após o DockerCompose

6 Docker-Compose criado

No exercicio pretende-se criar um ficheiro Docker-Compose que comece pelo menos dois serviços que estão na mesma Network, partilham um volume, um dos serviços tambem têm um bind mount, um dos serviços tem que ter a porta 9999 exposta na 8888 do host.

O seguinte Docker-Compose foi criado:

```
version: "3"

services:
    mongo:
        image: mongo
        container_name: Mongo
        volumes:
            - shared:/home
        networks:
            - shared_network
        ports:
            - "8888:9999"
    httpd:
        image: httpd
        container_name: Httpd
        volumes:
            - shared:/home
            - .:/home/exemplo
        networks:
            - shared_network
volumes:
    shared:
networks:
    shared_network:
```


Figura 9. Docker-Compose

Este Docker-Compose inicia dois containers:

- Uma imagem Mongo num container com nome Mongo que tem persistência num volume shared que vai ser criado pelo Docker-Compose.Também coneta-se a uma network que vai ser criada nesse mesmo ficheiro.Finalmente, mapeia-se a porta do host 8888 para a porta do container 9999.

- Uma imagem HTTPD com nome Httpd utilizando para persistência o mesmo volume que o Mongo e mais um bind mount da pasta atual para uma pasta /home/exemplo e está conetado a mesma Network do Mongo.

7 DockerFile

No Docker file foi definido primeiramente a imagem base, ubuntu. De seguida foi definido o maintainer. Com o RUN atualizamos os pacotes da imagem e instalamos o nginx que é um server HTTP e proxy reverso. Com o Expose expomos a porta do container e por fim com o CMD informamos qual o comando que será executado depois de ser feita a criaçao do container. Alem disto se usasse-mos o ENTRYPOINT podemos fazer uso do comando anterior para definir parâmetros que este usar.

```
FROM ubuntu:latest
LABEL maintainer = "TPVR"
RUN apt-get update && apt-get upgrade -y
RUN apt-get install nginx -y
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
```

Figura 10. DockerFile

Apos criado o DockerFile foi necessario fazer build do mesmo. Usamos o comando: docker build -t tpvr/nginx:1.0 .

```
Sending build context to Docker daemon  2.048kB
Step 1/6 : FROM ubuntu:latest
 ---> ccc6e87d482b
Step 2/6 : LABEL maintainer = "TPVR"
 ---> Using cache
 ---> 779cd721c6fa
Step 3/6 : RUN apt-get update && apt-get upgrade -y
 ---> Using cache
 ---> 6f408a37821e
Step 4/6 : RUN apt-get install nginx -y
 ---> Using cache
 ---> 8be698d9db23
Step 5/6 : EXPOSE 80
 ---> Using cache
 ---> c10e15964c94
Step 6/6 : CMD ["nginx","-g","daemon off;"]
 ---> Using cache
 ---> 7f5982bc2c30
Successfully built 7f5982bc2c30
Successfully tagged tpvr/nginx:1.0
```

Figura 11. Steps do Docker File

Como é possível observar na imagem acima, foi criada uma imagem "espelhada" do ubuntu e com o nginx instalado, entre outros pacotes. Com o comando docker imagens provamos uma vez mais que esta foi criada.

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
tpvr/nginx          1.0                 7f5982bc2c30        36 minutes ago      164MB
```

Figura 12. Image Criada

De seguida, foi necessario testar e para isso criamos um container

```
docker container run --name teste -p 80:80 tpvr/nginx:1.0
```

Figura 13. Comando para correr a Imagem

E por fim podemos observar que ao abrir localhost:80 este direciona-nos para a pagina inicial do nginx, como mostra a figura abaixo.

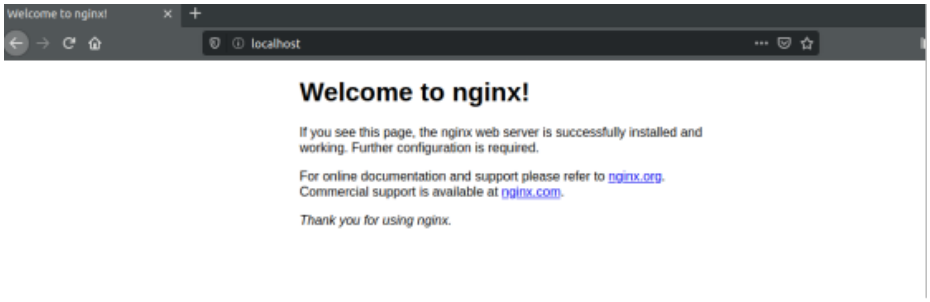


Figura 14. Serviço NGINX

8 Build Automática

A build automática pode ser feita seguindo o link seguinte Imagem

https://hub.docker.com/r/kaiserfrost/tp1
?fbclid=IwAR2Lz_-7bJ2MdjYj38VvFP2TxxP3n-LMpGkPUi5sURwsbloGhXOCwPnaxqQ