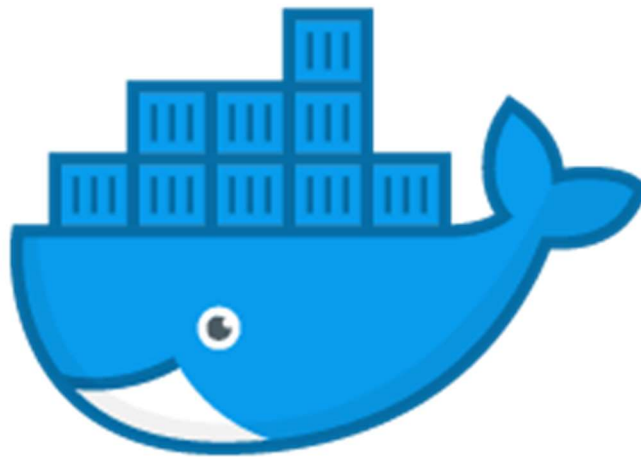


Docker

Guia para iniciantes



docker

Elaborado por:
Nelson Henriques – 2190514

Orientado por:
Carlos Neves
Luís Conde

Índice

1.	INTRODUÇÃO.....	1
1.1.	OBJETIVOS.....	1
2.	TERMINOLOGIA DOCKER.....	2
2.1.	IMAGES.....	2
2.2.	CONTAINERS	2
2.3.	VOLUMES	2
2.4.	NETWORKS.....	2
3.	INSTALAR O SOFTWARE DOCKER E DOCKER-COMPOSE	3
3.1.	DOCKER	3
3.2.	DOCKER-COMPOSE	4
4.	COMANDOS BÁSICOS DOCKER.....	5
5.	COMANDOS BÁSICOS DOCKER-COMPOSE.....	11

1. Introdução

O Docker é uma ferramenta utilizada para tornar mais fácil criar, implementar e executar aplicações utilizando *containers*. Os *containers* permitem que o desenvolvedor empacote a aplicação bem como todas as partes de que precisa, como bibliotecas e outras dependências, e implemente-o como um *package*. Assim o desenvolvedor pode ter certeza de que a aplicação será executada em qualquer outra máquina Linux, independentemente de quaisquer configurações personalizadas que a máquina possa ter, que podem ser diferentes da máquina usada para escrever e testar o código.

De certa forma, o Docker é idêntico a uma máquina virtual. Mas, ao contrário de uma máquina virtual, em vez de criar um sistema operativo virtual completo, o Docker permite que as aplicações usem o mesmo *kernel* Linux do sistema em que estão a ser executadas, requer apenas que as aplicações sejam enviadas com coisas que ainda não estão a ser executadas no host. Isto possibilita um aumento significativo no desempenho e reduz o tamanho da aplicação.

Compose é uma ferramenta utilizada para definir e executar aplicações Docker compostas por vários containers. O Compose utiliza arquivos YAML para configurar os serviços da aplicação. Então, com um único comando cria e inicia todos os serviços da configuração.

1.1. Objetivos

Pretende-se com este tutorial fornecer os conhecimentos básicos para a utilização do *software Docker e Docker-Compose*.

Ao longo do tutorial vai-se abordar os seguintes temas:

- Terminologia Docker;
- Instalar o software Docker e Docker-Compose;
- Comandos básicos Docker.
- Comandos básicos Docker-Compose.

(Este tutorial é destinado ao Docker e Docker-Compose em um sistema Linux 20.04)

2. Terminologia Docker

2.1.Images

As *images* do docker são a "receita" para um *container*. As *images* contém todas as definições de como inicializar o ambiente Linux. Normalmente, uma imagem Docker existe para cumprir especificamente uma tarefa. Por exemplo, uma imagem definiria o seu servidor web e outra imagem definiria a database subjacente.

2.2.Containers

Os *containers* são instanciações de imagens. Eles são uma forma de imagem. Compare-a com a programação orientada a objetos, então sua classe seria uma imagem e uma instância da classe, um *container*. Outra comparação seria com contentores reais num cargueiro. Estes tem uma aparência idêntica por fora, mas por dentro são/podem ser completamente diferentes. Alguns factos interessantes sobre *containers*:

- Os *containers* não são persistentes. Assim que não estiverem a ser utilizados eles são desligados, ou até mesmo eliminados.
- Um *container* só vive enquanto um processo é executado dentro dele.
- Os *containers* podem incluir dados, mas estes também não são persistentes. Se quiser torná-los persistentes, pode-se utilizar *volumes*.
- Para comunicar entre *containers*, usa-se os mesmos protocolos que se usariam para comunicar entre computadores (por exemplo, TCP / IP).

2.3.Volumes

Os *volumes* podem ser usados como camada de dados subjacente. Estes podem ser utilizados por vários *containers*. Todos os dados guardados num volume podem ser acedidos por qualquer *container* conectado a ele.

2.4.Networks

O Docker vem com as suas próprias capacidades de *networking*. O nome de um *container* é seu nome de host. A maneira mais fácil de experimentar isso é ter dois *containers* em execução e executar o comando para verificar o ping em um *container* de dentro de outro *container*. O Docker Compose irá gerar sua própria sub-rede.

3. Instalar o software Docker e Docker-Compose

3.1.Docker

Iremos instalar utilizando o repositório oficial do Docker.

1. Atualize as *packages* do apt e adicione novas para o mesmo poder utilizar o repositório por HTTPS:

```
sudo apt-get update  
  
sudo apt-get install \  
    apt-transport-https \  
    ca-certificates \  
    curl \  
    gnupg \  
    lsb-release
```

2. Adicione a chave GPG do Docker.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor  
-o /usr/share/keyrings/docker-archive-keyring.gpg
```

3. Usa o seguinte comando para criar o repositório local.

```
echo \  
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/ubuntu \  
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >  
/dev/null
```

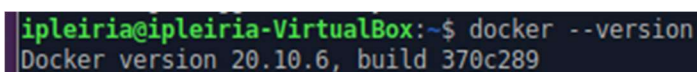
4. Instale o Docker.

```
sudo apt-get update  
  
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

5. Verifique se o Docker foi instalado corretamente..

```
docker --version
```

deverá obter algo semelhante á Figura 1



```
ipleiria@ipleiria-VirtualBox:~$ docker --version  
Docker version 20.10.6, build 370c289
```

Figura 1 - Versão do Docker.

3.2.Docker-Compose

1. Descarregue a versão mais recente e estável do Docker-Compose

```
sudo curl -L  
"https://github.com/docker/compose/releases/download/1.29.1/docker-compose-  
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

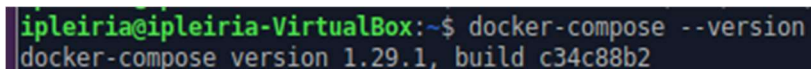
2. Dê permissão de execução ao binário descarregado.

```
sudo chmod +x /usr/local/bin/docker-compose
```

3. Verifique se o Docker-Compose foi instalado corretamente.

```
docker-compose --version
```

Deverá obter algo semelhante á Figura 1



```
ipleiria@ipleiria-VirtualBox:~$ docker-compose --version  
docker-compose version 1.29.1, build c34c88b2
```

Figura 2 - Versão do Docker-Compose.

4. Comandos básicos Docker

1. Iremos começar correndo o clássico “Hello World”.

```
docker run hello-world
```

Deverás obter a seguinte resposta (Figura 3).

```
ipleiria@ipleiria-VirtualBox:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:f2266cbfcl27c960fd30e76b7c792dc23b588c0db76233517e1891a4e357d519
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
ipleiria@ipleiria-VirtualBox:~$
```

Figura 3 - Output de "Hello World"

A *output* já descreve muito bem o que acontece ao executar este comando. Iniciou-se um *container*, cujo a imagem foi descarregada do Docker hub, de seguida executou-o e enviou a resposta para o terminal.

2. Vê todos os containers que estão em execução ou foram executados.

```
docker ps -a
```

```
ipleiria@ipleiria-VirtualBox:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
bf640b7c9a27   hello-world    "/hello"                 11 minutes ago Exited (0)    11 minutes ago          relaxed_dewdney
ipleiria@ipleiria-VirtualBox:~$
```

Figura 4 - containers que estão em execução ou foram executados

Os “Names” são gerados aleatoriamente, podemos referir-nos aos *containers* pelos “Names” ou pelo “container id” (Figura 4)

3. Execute novamente o “Hello World” mas desta vez iremos atribuir um nome, assim não dependemos de um nome gerado aleatoriamente.

```
docker run --name My-HW hello-world
```

4. Execute novamente o comando do passo nº2 e verifique que o nome é igual ao que foi atribuído no comando anterior (Figura 5).

```
ipleiria@ipleiria-VirtualBox:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d6476a89c254	hello-world	"/hello"	4 seconds ago	Exited (0) 3 seconds ago		My-HW
95ce9959bc10	hello-world	"/hello"	14 minutes ago	Exited (0) 14 minutes ago		relaxed_dewdney

```
ipleiria@ipleiria-VirtualBox:~$
```

Figura 5 - container com o nome customizado

5. Agora vamos executar um container que contem o ubuntu (como este container não está localmente o Docker irá descarrega-lo do Docker Hub) e ao mesmo tempo iremos atribuir um nome customizado

```
docker run -it --name my-linux-container ubuntu bash
```

```
ipleiria@ipleiria-VirtualBox:~$ docker run -it --name my-linux-container ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
a70d879fa598: Pull complete
c4394a92d1f8: Pull complete
10e6159c56c0: Pull complete
Digest: sha256:3c9c713e0979e9bd6061ed52ac1e9elf246c9495aa063619d9d695fb8039aa1f
Status: Downloaded newer image for ubuntu:latest
root@bc96b85e5169:/#
```

Figura 6 - ubuntu container

Agora está dentro do container ubuntu (Figura 6) onde pode executar qualquer comando Linux isto é bastante útil quando estamos num sistema windows ou macOS, para sair basta escrever “exit”.

6. Mais uma vez ao executar o comando do passo nº2 para ver o estado de todos os containers executados ou em execução ate ao momento (Figura 7).

```
ipleiria@ipleiria-VirtualBox:~$ docker ps -a
```

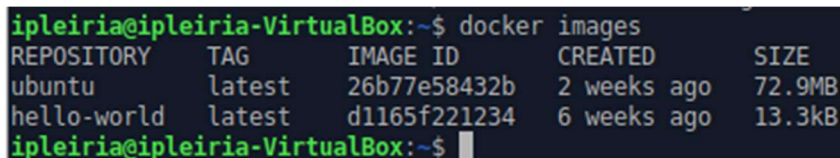
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bc96b85e5169	ubuntu	"bash"	4 minutes ago	Exited (0) 4 seconds ago		my-linux-container
d6476a89c254	hello-world	"/hello"	14 minutes ago	Exited (0) 14 minutes ago		My-HW
95ce9959bc10	hello-world	"/hello"	29 minutes ago	Exited (0) 29 minutes ago		relaxed_dewdney

```
ipleiria@ipleiria-VirtualBox:~$
```

Figura 7 - Containers executados ou em execução

7. Verifique quais as imagens que tem guardados localmente.

```
docker images
```



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	26b77e58432b	2 weeks ago	72.9MB
hello-world	latest	d1165f221234	6 weeks ago	13.3kB

Figura 8 - docker images

Este comando pode ser bastante útil pois permite saber todas as informações principais das “images” (Figura 8), como estão guardados localmente a sua execução da próxima vez será instantânea.

8. Uma vez iniciado um container com um nome pré-definido pelo utilizador o mesmo já não pode ser iniciado com o mesmo nome, por isso podemos executar o seguinte comando de modo a limpar o “histórico” de containers executados (Figura 9).

```
docker rm $(docker ps -a -f status=exited -q)
```

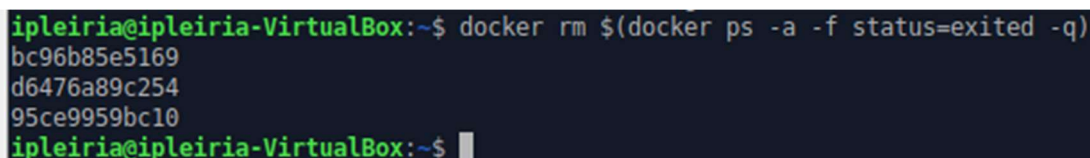


Figura 9 - output

9. Verifique que a lista está limpa correndo o comando do passo nº2.
10. Inicie um container e ao mesmo tempo conecte-o a uma pasta local, para isso crie uma pasta no seu sistema operativo local e guarde o seu caminho, substituindo-o no comando abaixo.

```
docker run -it --name my-linux-container --rm -v [diretoria local] ubuntu  
bash
```

- -it (interactive container) – entra no mesmo assim que inicia.
- --rm – assim que termina elimina-o da lista.
- -v – comando para conectar a diretoria local sendo que a pasta a partilhar tem que anteceder “:”. Ex:

```
docker run -it --name my-linux-container --rm -v  
/home/ipleiria/Desktop:/meus_dados ubuntu bash
```

11. Verifique que a pasta foi partilhada executando o comando `ls` dentro do *container*.

```
ipleiria@ipleiria-VirtualBox:~$ docker run -it --name my-linux-container --rm -v /home/ipleiria/Desktop:/meus_dados ubuntu bash
root@fc7952ad7869:/# ls
bin  dev  home  lib32  libx32  meus_dados  opt  root  sbin  sys  usr
boot  etc  lib  lib64  media  mnt  proc  run  srv  tmp  var
root@fc7952ad7869:/#
```

Figura 10 - pasta partilhada

Como pode observar a pasta foi partilhada corretamente (Figura 10).

12. Execute um comando no *container* ubuntu, para isso, abra um novo terminal (não feche o atual), e utilize o seguinte código.

```
docker exec -it my-linux-container echo "Hello World"
```

Com este código irá imprimir um Hello World (Figura 11), sendo que depois do nome do *container* vem o comando a executar no mesmo.

```
ipleiria@ipleiria-VirtualBox:~$ docker exec -it my-linux-container echo "Hello World"
Hello World
ipleiria@ipleiria-VirtualBox:~$
```

Figura 11 - comando executado

13. Crie um Docker image, para isso crie uma nova pasta e atribua-lhe o nome “My-Docker-Image”, dentro dessa pasta crie um ficheiro com o nome “Dockerfile” (certifique-se que não tem um tipo de ficheiro ex: .txt), abra-o com um editor de texto, neste momento deverá ter algo semelhante á Figura 12.

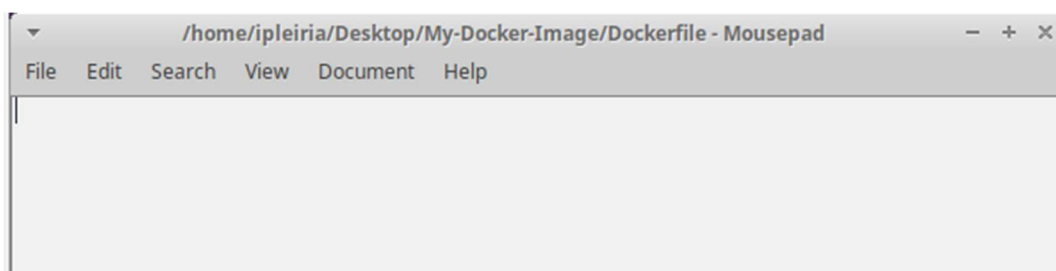


Figura 12 - ficheiro Dockerfile

14. No ficheiro escreva as seguintes linhas de código (Figura 13), guarda e fecha o ficheiro.

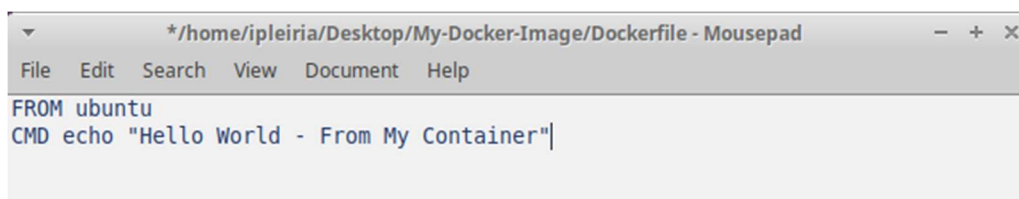


Figura 13 - código a escrever

Existe uma grande variedade de “Keywords” para criar imagens Docker estas podem ser encontradas aqui: <https://docs.docker.com/engine/reference/builder/>.

15. No terminal vá á diretoria criada (esta diretoria varia dependendo de onde criou a pasta).

```
cd /home/ipleiria/Desktop/My-Docker-Image
```

16. Construa a imagem.

```
docker build -t my-ubuntu-image .
```

- -t – atribui um nome á imagem.

Deverá obter algo semelhante á Figura 14.

```
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$ docker build -t my-ubuntu-image .
Sending build context to Docker daemon 2.048kB
Step 1/2 : FROM ubuntu
--> 26b77e58432b
Step 2/2 : CMD echo "Hello World - From My Container"
--> Running in b8e9573dbbcd
Removing intermediate container b8e9573dbbcd
--> da02e6e1bf42
Successfully built da02e6e1bf42
Successfully tagged my-ubuntu-image:latest
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$
```

Figura 14 - imagem criada

17. Verifique que a imagem foi criada (Figura 15).


```
docker images
```

```
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
my-ubuntu-image     latest         da02e6e1bf42   2 minutes ago  72.9MB
ubuntu              latest         26b77e58432b   2 weeks ago    72.9MB
hello-world         latest         d1165f221234   6 weeks ago    13.3kB
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$
```

Figura 15 - lista de imagens locais

18. Corra a imagem criada.

```
docker run my-ubuntu-image
```



```
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$ docker run my-ubuntu-image
Hello World - From My Container
ipleiria@ipleiria-VirtualBox:~/Desktop/My-Docker-Image$
```

Figura 16 - output do meu container

Como pode observar ao executar o container ele escreveu no terminal o que foi escrito no Dockerfile (Figura 16).

Ex:

Pretende iniciar um ubuntu Docker já com o python3 instalado, para isso no ficheiro criado anteriormente basta trocar o “CMD echo "Hello World - From My Container"” por “RUN apt-get upgrade && apt-get update && apt-get install -y python3” e contruir novamente a imagem, assim terá uma imagem ubuntu com o python3 já instalado.

Pode também guardar o estado do seu container em uma nova imagem através do seguinte comando:

```
docker commit $CONTAINER_ID image name
```

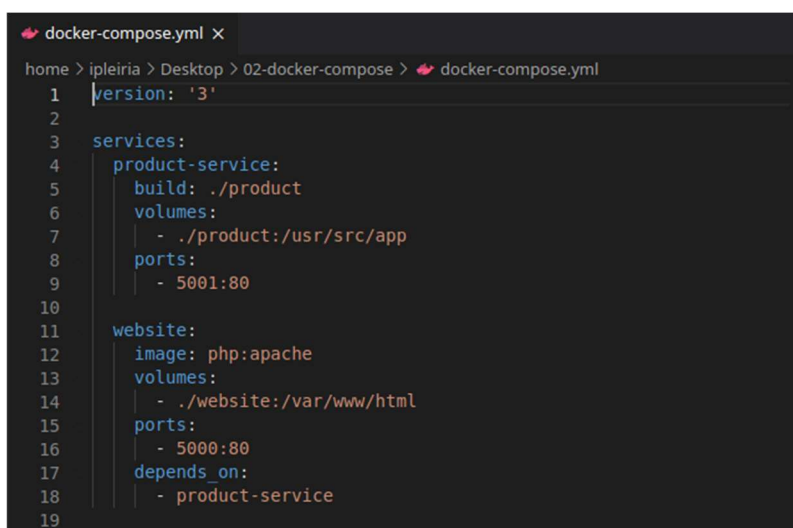
5. Comandos básicos Docker-Compose

1. Descarregue o conteúdo do repositório, este será o exemplo utilizado para o tutorial.
(<https://github.com/jakewright/tutorials/tree/master/docker/02-docker-compose>)

Nessa pasta irás encontrar 4 ficheiros, “product”, “Website”, “readme.md” e “Docker-compose.yml”, o ficheiro “Docker-compose.yml” é o ficheiro onde estão todas as definições da aplicação, que neste caso é um web site muito simples.

Este exemplo está pronto a ser executado mas antes disso iremos entender o que irá o ficheiro “Docker-compose.yml” fazer.

2. Abra o ficheiro “Docker-compose.yml” com o seu editor preferido (Figura 17).



```

1 version: '3'
2
3 services:
4   product-service:
5     build: ./product
6     volumes:
7       - ./product:/usr/src/app
8     ports:
9       - 5001:80
10
11   website:
12     image: php:apache
13     volumes:
14       - ./website:/var/www/html
15     ports:
16       - 5000:80
17     depends_on:
18       - product-service
19

```

Figura 17- Docker-Compose.yml

Version: – Versão em que está a ser escrita o ficheiro (opcional a partir da versão 1.27).

services: – serviços a iniciar.

Product-service e website – nome dos serviços.

O restante são as dependências e definições dos serviços sendo que tem que ser adaptadas ao que pretende realizar, para mais informação consulte este link:

<https://docs.docker.com/compose/compose-file/>

3. Abra o terminal e vá até á diretoria descarregada.

```
cd /home/ipleiria/Desktop/02-docker-compose/
```

4. Agora irá “construir” a aplicação com base no ficheiro criado (Figura 18).

```
docker-compose build
```

```
Successfully built cd6a014cc511
Successfully tagged 02-docker-compose_product-service:latest
ipleiria@ipleiria-VirtualBox:~/Desktop/02-docker-compose$
```

Figura 18 - construção terminada

5. Agora irá iniciar a aplicação criada (Figura 19).

```
docker-compose up
```

```
ipleiria@ipleiria-VirtualBox:~/Desktop/02-docker-compose$ docker-compose up
Creating network "02-docker-compose_default" with the default driver
Pulling website (php:apache)...
apache: Pulling from library/php
f7ec5a41d630: Pull complete
941223b59841: Pull complete
a5f2415e5a0c: Pull complete
b9844b87f0e3: Pull complete
5a07de50525b: Pull complete
caeca1337a66: Pull complete
5dbe0d7f8481: Pull complete
7bc44096b360: Pull complete
dc87c3ea8feb: Pull complete
c02fa4183668: Pull complete
a3bdae68ae09: Pull complete
757f28c05a2d: Pull complete
0831ef5c7c6f: Pull complete
Digest: sha256:52bed532a4cd1d08b1fb12f375d252db268bf1510ca9ac9d8317b5835139b356
Status: Downloaded newer image for php:apache
Creating 02-docker-compose_product-service_1 ... done
Creating 02-docker-compose_website_1 ... done
Attaching to 02-docker-compose_product-service_1, 02-docker-compose_website_1
product-service_1 | * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
product-service_1 | * Restarting with stat
product-service_1 | * Debugger is active!
product-service_1 | * Debugger PIN: 127-286-934
website_1 | AH00558: apache2: Could not reliably determine the server's
website_1 | fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive g
website_1 | globally to suppress this message
website_1 | AH00558: apache2: Could not reliably determine the server's
website_1 | fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive g
website_1 | globally to suppress this message
website_1 | [Mon Apr 26 15:04:05.485067 2021] [mpm_prefork:notice] [pid
website_1 | 1] AH00163: Apache/2.4.38 (Debian) PHP/8.0.3 configured -- resuming normal oper
website_1 | ations
website_1 | [Mon Apr 26 15:04:05.485147 2021] [core:notice] [pid 1] AH0
website_1 | 0094: Command line: 'apache2 -D FOREGROUND'
```

Figura 19 - aplicação iniciada

6. Aceda a localhost:5000 no seu browser, deverá aparecer algo semelhante á Figura 20.

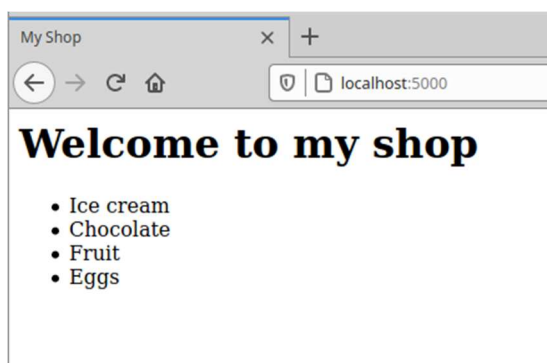


Figura 20 - website exemplo

Caso dê um erro semelhante ao da Figura 21, corra o seguinte comando:

```
sudo chmod 755 website
```

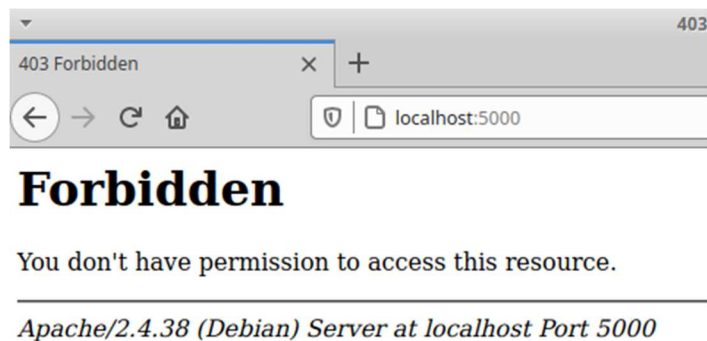


Figura 21 - possível erro