

Laboratório de Desenvolvimento de Software

Célio Carvalho
cdf@estg.ipp.pt

P. PORTO

CONTEXTUALIZAÇÃO

O que é o Docker?

- Software de virtualização.
- Facilita o processo de desenvolvimento e publicação de aplicações.
- Permite fazer o *packaging* de uma aplicação em conjunto com as suas dependências, configurações, e *runtime* e ferramentas de sistema.
- Basicamente, permite transportar num pacote todo o ambiente que uma aplicação precisa para executar sem problemas.
- Pode também ser visto como uma forma de uniformizar instalações (e.g. entre ambientes de desenvolvimento de uma equipa de programadores).
- O pacote produzido é facilmente distribuído (ocupa alguns MB) e partilhado (e.g. Docker hub).

CONTEXTUALIZAÇÃO

Quais os problemas que ajuda a resolver na **fase de desenvolvimento?**

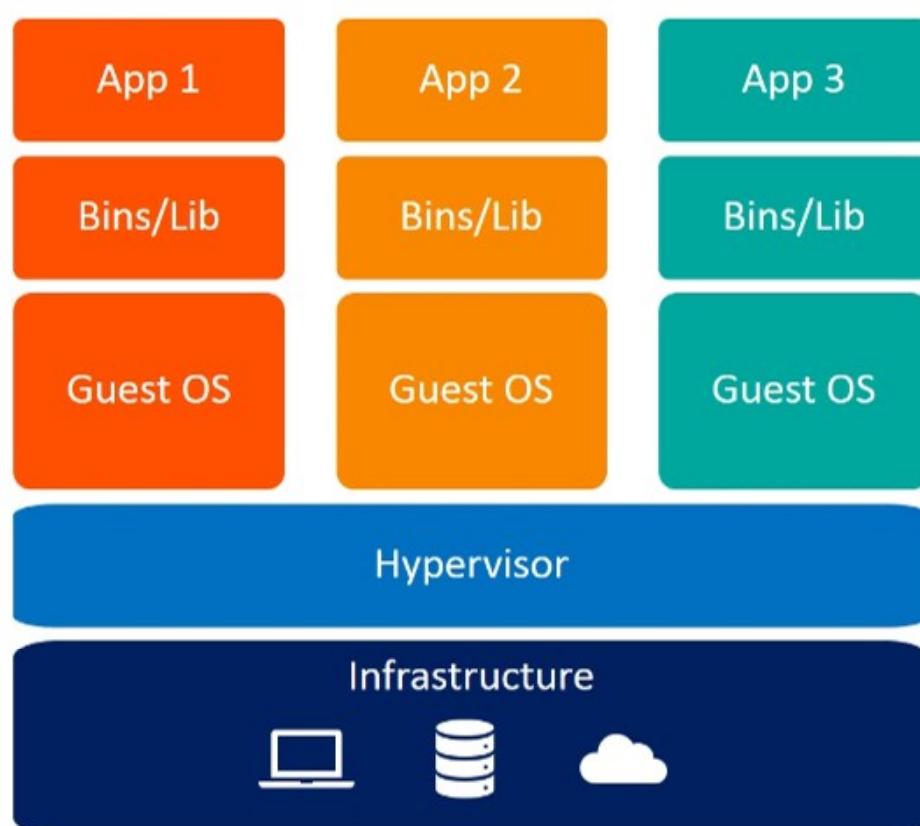
- **Antes**, as equipas de desenvolvimento tinham que instalar no seu sistema todo o ambiente necessário para conseguirem desenvolver e testar o seu trabalho (e.g. MySQL, Cassandra, Redis, PostgreSQL, *Caching*, *Messaging*).
 - Cada elemento da equipa poderia ter diferentes Sistemas Operativos (SO) (e.g. Windows, MacOS, Linux) e diferentes versões (e.g. Win11, Win8.1, Ubuntu, Fedora, CentOS).
 - O processo de instalação implica executar várias passos / configurações, que podem até diferir entre SO. Durante a instalação podem ser aplicadas configurações diferentes, tornando os ambientes de desenvolvimento diferentes entre si.
-
- ✓ **Agora**, basta que se usem imagens com os componentes necessários. Cada *container* será executado num ambiente isolado, onde todas as dependências estão configuradas.
 - ✓ Se os programadores usarem todas as mesmas imagens, terão todos um ambiente de desenvolvimento muito parecido.
 - ✓ Uma imagem pode ser executada rapidamente com um comando Docker (passa a chamar-se *container*).
 - ✓ Pode executar-se várias versões de uma mesma aplicação sem qualquer conflito (e.g. várias versões PostgreSQL podem estar a ser executadas em simultâneo, cada uma, em containers diferentes).

CONTEXTUALIZAÇÃO

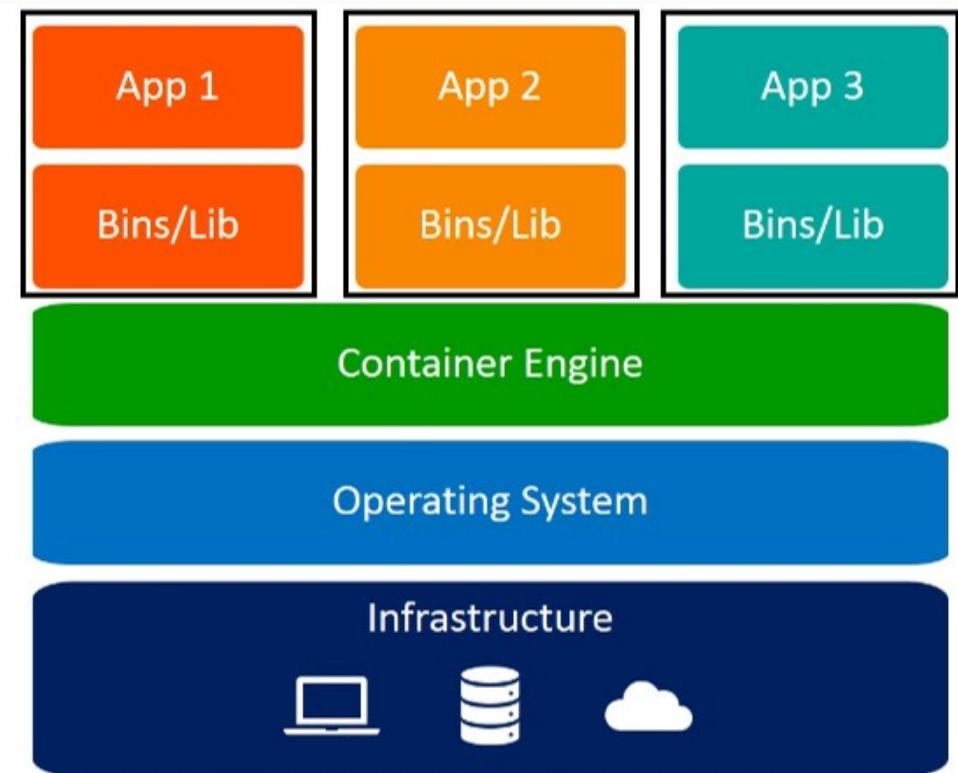
Quais os problemas que ajuda a resolver na **fase de publicação (entrada em produção)**?

- **Antes**, os programadores entregavam a aplicação funcional (e.g. ficheiros executáveis) ao pessoal das operações (infraestrutura), para que estes procedessem à publicação dos serviços, aplicações, bases de dados, etc., etc.
 - Além destes artefactos compilados, tinham também que entregar um conjunto de instruções e/ou scripts, para que a equipa de operações tivesse todos os detalhes necessários para instalar, configurar, e colocar em funcionamento todos os serviços, aplicações, bases de dados, etc. etc., e as respetivas dependências.
 - O problema é que, como tinham que instalar tudo num ambiente novo (sistemas operativos diferentes do dos programadores), muitas coisas podiam e corriam mal.
 - Também acontecia dos programadores não darem todas as instruções necessárias, ou com o detalhe necessário.
-
- ✓ **Agora**, com containers, o processo fica todo mais simples. Os programadores criam imagens que incluem todo o código necessário em conjunto com as respetivas dependências.
 - ✓ Os artefactos que dão à equipa de operações já é funcional, basta publicar as imagens e fazer algumas configurações.
 - ✓ Tudo fica encapsulado dentro de um ambiente, fazendo com que não haja tanta possibilidade de problemas, fazendo com que o processo de entrada em produção seja muito mais simples e rápido.

COMPARAÇÃO COM MÁQUINAS VIRTUAIS (VIRTUAL MACHINES)



Virtual Machines



Containers

COMPARAÇÃO COM MÁQUINAS VIRTUAIS (VIRTUAL MACHINES)

	Docker	Virtual Machines (VMs)
Boot-Time	Boots in a few seconds.	It takes a few minutes for VMs to boot.
Runs on	Dockers make use of the execution engine.	VMs make use of the hypervisor.
Memory Efficiency	No space is needed to virtualize, hence less memory.	Requires entire OS to be loaded before starting the surface, so less efficient.
Isolation	Prone to adversities as no provisions for isolation systems.	Interference possibility is minimum because of the efficient isolation mechanism.
Deployment	Deploying is easy as only a single image, containerized can be used across all platforms.	Deployment is comparatively lengthy as separate instances are responsible for execution.
Usage	Docker has a complex usage mechanism consisting of both third party and docker managed tools.	Tools are easy to use and simpler to work with.

INSTALAÇÃO

- Instalar o **Docker Desktop**.
- Download a partir do site oficial do Docker:
 - ✓ <https://www.docker.com/>
- Seguir as instruções corretas para o SO.
- A instalação inclui:
 - **Docker Engine** – que executa o serviço de servidor, e permite a gestão de imagens / *containers*.
 - **Docker CLI** – *Command Line Interface* que permite enviar comandos para o Docker Engine (iniciar ou parar um *container* Docker).
 - **Docker CLI GUI** – Permite gerir o Docker Engine através de um ambiente gráfico, em alternativa ao CLI.
- Executar as instruções abaixo para testar o funcionamento:
 - `docker --version`
 - `docker run -d -p 80:80 docker/getting-started`
 - ✓ Depois de colocar o container em execução, navegar <http://localhost> ou <http://127.0.0.1>

DOCKER IMAGES VS. DOCKER CONTAINERS

- Uma **Docker Image** é um *package* que contém uma aplicação. Este *package* contém todo o código, bibliotecas, dependências e configurações, necessárias à execução da aplicação. Tudo o que é necessário para a aplicação executar, fica empacotado e configurado dentro da *Image*.
 - ✓ Pode ser facilmente partilhada e movida entre sistemas.
 - ✓ Pode ser disponibilizada num *registry* (e.g. Docker Hub), para ser consumido por um ou mais consumidores.
- Um **Docker Container** é uma instância executável de uma aplicação existente numa Docker Image. Um Docker Container é basicamente uma *Image* em execução.
 - ✓ Permite a execução isolada.
 - ✓ Podem existir vários *Containers* da mesma *Image* em execução no mesmo Docker Engine (mesmo servidor Docker) (*1).

(*1) Bastante útil, por exemplo, para lançar várias instâncias da mesma aplicação, com o objetivo de escalar aplicações (cf. *Scale-up* e *Scale-down* de microsserviços).

EXERCÍCIO

- A instrução abaixo apresenta as Docker Images disponíveis no Docker:

➤ docker images

C:\>docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker/getting-started	latest	3e4394f6b72f	9 months ago	47MB

- A instrução abaixo apresenta os containers atualmente em execução do Docker:

➤ docker ps

C:\>docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b50296341367	docker/getting-started	"/docker-entrypoint...."	55 minutes ago	Up 53 minutes	0.0.0.0:80->80/tcp	thirsty_pike

DOCKER REGISTRIES

- São sites (serviços na internet) onde estão e podem ser publicadas imagens Docker. São serviços que permitem armazenar, distribuir e compartilhar Docker Images.
- Existem vários Docker Registries públicos e privados:
 - ✓ **Docker Hub** | <https://hub.Docker.com> – será o *registry* mais conhecido e mais utilizado. Contém imagens públicas que podem ser utilizadas pela comunidade. Contém imagens oficiais (e.g. Mongo, Postgres) que são mantidas pelos fabricantes de software, mas também pode conter imagens não oficiais disponibilizadas pela comunidade.
 - **Google Container Registry (GCR)** | <https://cloud.google.com/container-registry>
 - **Amazon Elastic Container Registry (ECR)** | <https://aws.amazon.com/ecr>
 - **Azure Container Registry (ACR)** | <https://azure.com/containerregistry>
 - **Harbor** | <https://goharbor.io>
 - ...

DOCKER REGISTRIES | DOCKER HUB | Registries, Repositories, e tags

- Os conceitos de *registry* e de *repository* do Docker estão relacionados mas são distintos.
- **Registry** – é um serviço / servidor onde podem ser guardadas, distribuídas e recuperadas imagens. Existem *registries* públicos e privados. Um Registry pode ser visto como uma conjunto de Repositories.
- **Repository** – é um conjunto de imagens com o mesmo nome, que funcionam como uma forma de melhor organizar e localizar o conteúdo dentro de um Registry. Um *Repository* pode ser visto como um conjunto de *images* relacionadas (mesmo nome mas com diferentes versões).
- **Tag** – é uma forma de diferenciar / identificar cada imagem dentro de um *Repository*. Cada repositório pode conter várias versões da mesma imagem, e cada imagem / versão é identificada através de etiquetas (*tags*). A versão mais atual é etiquetada com a tag **latest**, e é a versão assumida por defeito, caso não seja indicada qualquer versão no *pull* de uma imagem.



Supported tags and respective Dockerfile links

- 11.1.2-jammy , 11.1-jammy , 11-jammy , jammy , 11.1.2 , 11.1 , 11 , latest
- 11.0.3-jammy , 11.0-jammy , 11.0.3 , 11.0
- 10.11.5-iammv , 10.11-iammv , 10-iammv , lts-iammv , 10.11.5 , 10.11 , 10 , lts



DOCKER REGISTRIES | DOCKER HUB | Demonstração Images (1/3)



- Aceder a <https://hub.docker.com>
 - ✓ Para aceder ao *registry* público de imagens não é necessário registo.
 - ✓ Para aceder ao *registry* privado é necessário registo.
- Nesta demonstração vai ser utilizada uma imagem oficial do nginx, porque é um servidor web que já tem uma página de *landing* por defeito (para testar funcionamento do serviço dentro do *container*):
 - Primeiro localizar o *Repository*.



- Depois escolher a versão (*tag*) pretendida, deve fazer-se o *pull* da imagem.

nginx Docker Official Image · 1B+ · 10K+
Official build of Nginx.

Overview Tags

Quick reference

- Maintained by:
the NGINX Docker Maintainers
- Where to get help:
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

- 1.25.2, mainline, 1, 1.25, latest, 1.25.2-bookworm, mainline-bookworm, 1-bookworm, 1.25-bookworm, bookworm
- 1.25.2-perl, mainline-perl, 1-perl, 1.25-perl, perl, 1.25.2-bookworm-perl, mainline-bookworm-perl, 1-bookworm-perl, 1.25-bookworm-perl, bookworm-perl
- 1.25.2-alpine, mainline-alpine, 1-alpine, 1.25-alpine, alpine, 1.25.2-alpine3.18, mainline-alpine3.18, 1-alpine3.18, 1.25-alpine3.18, alpine3.18
- 1.25.2-alpine-perl, mainline-alpine-perl, 1-alpine-perl, 1.25-alpine-perl,

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Images (2/3)



- Para esta demonstração, escolheu-se uma imagem com a versão já mais antiga: 1.25.2-alpine

- docker pull nginx:1.25.2-alpine
- ✓ Não é necessário indicar qual o Registry porque, por defeito, é o Docker Hub.

```
C:\>docker pull nginx:1.25.2-alpine
1.25.2-alpine: Pulling from library/nginx
7264a8db6415: Already exists
518c62654cf0: Pull complete
d8c801465ddf: Pull complete
ac28ec6b1e86: Pull complete
eb8fb38efa48: Pull complete
e92e38a9a0eb: Pull complete
58663ac43ae7: Pull complete
2f545e207252: Downloading [==>
] 792.9kB/12.62MB
```

- docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	1.25.2-alpine	433dbc17191a	4 weeks ago	42.6MB
docker/getting-started	latest	3e4394f6b72f	9 months ago	47MB

nginx Docker Official Image · 1B+ · 10K+
Official build of Nginx.

Overview Tags

Quick reference

- Maintained by:
the NGINX Docker Maintainers
- Where to get help:
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

- 1.25.2, mainline, 1, 1.25, latest, 1.25.2-bookworm, mainline-bookworm, 1-bookworm, 1.25-bookworm, bookworm
- 1.25.2-perl, mainline-perl, 1-perl, 1.25-perl, perl, 1.25.2-bookworm-perl, mainline-bookworm-perl, 1-bookworm-perl, 1.25-bookworm-perl, bookworm-perl
- 1.25.2-alpine, mainline-alpine, 1-alpine, 1.25-alpine, alpine, 1.25.2-alpine3.18, mainline-alpine3.18, 1-alpine3.18, 1.25-alpine3.18, alpine3.18
- 1.25.2-alpine-perl, mainline-alpine-perl, 1-alpine-perl, 1.25-alpine-perl,

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Images (3/3)



- Se não for escolhida nenhuma versão (*tag*) será utilizada a *latest*.

➤ docker pull nginx

```
C:\>docker pull nginx
Using default tag: latest ←
latest: Pulling from library/nginx
a803e7c4b030: Downloading 593.1kB/29.12MB
8b625c47d697: Downloading 426kB/41.34MB
4d3239651a63: Download complete
0f816efa513d: Waiting
01d159b8db2f: Waiting
5fb9a81470f3: Waiting
9b1e1e7164db: Waiting
```

➤ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest →	61395b4c586d	5 days ago	187MB
nginx	1.25.2-alpine	433dbc17191a	4 weeks ago	42.6MB
docker/getting-started	latest	3e4394f6b72f	9 months ago	47MB

nginx Docker Official Image · 1B+ · 10K+

Official build of Nginx.

Overview Tags

Quick reference

- Maintained by: the NGINX Docker Maintainers
- Where to get help: the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

- 1.25.2, mainline, 1, 1.25, latest, 1.25.2-bookworm, mainline-bookworm, 1-bookworm, 1.25-bookworm, bookworm
- 1.25.2-perl, mainline-perl, 1-perl, 1.25-perl, perl, 1.25.2-bookworm-perl, mainline-bookworm-perl, 1-bookworm-perl, 1.25-bookworm-perl, bookworm-perl
- 1.25.2-alpine, mainline-alpine, 1-alpine, 1.25-alpine, alpine, 1.25.2-alpine3.18, mainline-alpine3.18, 1-alpine3.18, 1.25-alpine3.18, alpine3.18
- 1.25.2-alpine-perl, mainline-alpine-perl, 1-alpine-perl, 1.25-alpine-perl, alpine-perl, 1.25.2-alpine3.18-perl, mainline-alpine3.18-perl, 1-alpine3.18-perl, 1.25-alpine3.18-alpine-perl

docker pull nginx

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (1/7)

- Para se poder utilizar a imagem, tem primeiro que se executar, criando uma instância dessa imagem, i.e., criar um *container*.

➤ `docker run nginx:1.25.2-alpine`

C:\>docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	nginx	latest	61395b4c586d	5 days ago	187MB
	nginx	1.25.2-alpine	433dbc17191a	4 weeks ago	42.6MB
	docker/getting-started	latest	3e4394f6b72f	9 months ago	47MB

- O *container* é criado, mas a consola fica bloqueada com a execução do processo. Para se desbloquear a consola, utiliza-se o `<ctrl><c>`, para terminar o processo, e o *container* fica parado.
- Para se executar um *container* de forma a não bloquear a consola, utiliza-se o argumento `-d` ou `--detach`. Desta forma, o *container* será executada em segundo plano.

➤ `docker run -d nginx:1.25.2-alpine`

C:\>docker run -d nginx:1.25.2-alpine
6e13533c62ac5a4344a4c761957fa39000984720a1b124e4a3f3bbe145bee794

- Para ver os *containers* em execução, utiliza-se o argumento `ps`.

➤ `docker ps`

C:\>docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint...."	2 minutes ago	Up 2 minutes	80/tcp	loving_mclean

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (2/7)

- Quando se executa um *container* sem opção *detach*, a consola fica bloqueada mas os *logs* de arranque do container são apresentados (útil, por exemplo, para *debug*). O mesmo não acontece com a opção **-d** em que apenas é apresentado o id do *container* instanciado.
- Se for necessário aceder aos *logs* gerados durante o carregamento de um *container*, pode utilizar-se o comando abaixo (necessita do *container id*).

➤ **docker logs 6e13533c62ac**

C:\>docker ps	CONTAINER ID	IMAGE	COMMAND	CREAT
	6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint..."	12 mi

```
C:\>docker logs 6e13533c62ac
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/09/26 08:27:44 [notice] 1#1: using the "epoll" event method
2023/09/26 08:27:44 [notice] 1#1: nginx/1.25.2
2023/09/26 08:27:44 [notice] 1#1: built by gcc 12.2.1 20230924 (Alpine 12.2.1_git20230924_r10)
```

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (3/7)

O comando docker ps dá várias informações acerca dos *containers* em execução:

C:\>docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint...."	23 minutes ago	Up 23 minutes	80/tcp	loving_mclean

Informação	Descrição
CONTAINER ID	Identificador único atribuído a cada <i>container</i> . Este valor é importante para referenciar o container em comandos do Docker (e.g. docker logs 6e13533c62ac, docker stop 6e13533c62ac).
IMAGE	Imagen e versão (<i>tag</i>) a partir do qual o <i>container</i> foi criado.
COMMAND	Comando que está em execução dentro do <i>container</i> (processo / aplicação) em execução.
CREATED	Quando é que o <i>container</i> foi criado.
STATUS	Estado do <i>container</i> . ✓ Este tema vai ser abordado nos slides seguintes.
PORTS	Portas mapeadas do <i>host</i> para o <i>container</i> (portas do <i>container</i> que estão acessíveis ao sistema <i>host</i>). ✓ Este tema vai ser abordado nos slides seguintes.
NAMES	Nomes que podem ser utilizados para referenciar o <i>container</i> (atribuído um aleatório se não indicado). ✓ Este tema vai ser abordado nos slides seguintes.

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (4/7)

- Se o comando docker run for executado para uma imagem inexistente no Docker local, o próprio sistema faz o docker pull para carregar a imagem respetiva e depois executa essa imagem para criar um *container*.
- Neste teste utiliza-se a imagem do nginx correspondente à versão 1.24

➤ **docker run nginx:1.24**

```
C:\>docker run nginx:1.24
Unable to find image 'nginx:1.24' locally
1.24: Pulling from library/nginx
7dbc1adf280e: Downloading [=====] 15.37MB/31.42MB
a7184f3665ed: Downloading [=====] 17.57MB/25.6MB
f144d5d97503: Download complete
9097eea98b48: Download complete
356d4b647b64: Download complete
608e661a622a: Download complete
```

- Abrir outra janela da consola e executar

➤ **docker ps**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f13957d2f595	nginx:1.24	"./docker-entrypoint..."	About a minute ago	Up About a minute	80/tcp	agitated_jang
6e13533c62ac	nginx:1.25.2-alpine	"./docker-entrypoint..."	About an hour ago	Up About an hour	80/tcp	loving_mclean

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (5/7)

- Para aceder ao conteúdo do container, tem que se fazer o ***binding*** das portas disponibilizadas pelo *container*.
- As aplicações em execução dentro do *container* executam de forma isolada numa rede interna do Docker. Para se poder comunicar com os serviços internos do *container*, tem que se especificar um mapeamento válido, da porta interna do *container* para a porta externa do sistema *host*.
- O mapeamento de portas internas para o sistema *host*, permite que várias *containers* da mesma imagem possam ser executados no mesmo *host* em simultâneo (e.g.):
 - nginx(c1, 80) → host: 8081
 - nginx(c2, 80) → host: 8082
 - nginx(c?, 80) → host: ?
- A partir do momento que se faz estes mapeamentos, os serviços associados dentro dos *containers*, ficam acessíveis ao *host*.

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (6/7)

- As portas expostas pelo *container*, podem ser consultadas através do docker ps.

C:\>docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	f13957d2f595	nginx:1.24	"/docker-entrypoint...."	14 minutes ago	Up 14 minutes	80/tcp	agitated_jang
	6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint...."	About an hour ago	Up About an hour	80/tcp	loving_mclean

- Para fazer o *binding* da porta interna 80 do *container* para a porta externa do *host* 8081, aquando da criação do container, utiliza-se o argumento adicional –p ou --publish. O exemplo abaixo termina o *container* em execução, e executa-o novamente mapeamento a porta interna 80 para a porta externa 8081. Por sim, apresenta novamente os containers em execução.

```
➤ docker stop f13957d2f595
➤ docker run -d -p 8081:80 nginx:1.24
➤ docker ps
```

C:\>docker ps	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	d25d49db2a92	nginx:1.24	"/docker-entrypoint...."	2 minutes ago	Up 2 minutes	0.0.0.0:8081->80/tcp	festive_mestorf
	6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint...."	2 hours ago	Up 2 hours	80/tcp	loving_mclean

continua no slide seguinte...

DOCKER REGISTRIES | DOCKER HUB | Demonstração Containers (7/7)

- Para testar o funcionamento do serviço nginx dentro do *container*, aceder ao endereço <http://localhost:8081>
- Os *logs* do serviço podem ser consultados com o comando docker logs.

➤ docker logs d25d49db2a92

```
C:\>docker logs d25d49db2a92
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/09/26 09:57:33 [notice] 1#1: using the "epoll" event method
2023/09/26 09:57:33 [notice] 1#1: nginx/1.24.0
2023/09/26 09:57:33 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/09/26 09:57:33 [notice] 1#1: OS: Linux 5.15.90.1-microsoft-standard-WSL2
2023/09/26 09:57:33 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/09/26 09:57:33 [notice] 1#1: start worker processes
2023/09/26 09:57:33 [notice] 1#1: start worker process 28
2023/09/26 09:57:33 [notice] 1#1: start worker process 29
2023/09/26 09:57:33 [notice] 1#1: start worker process 30
2023/09/26 09:57:33 [notice] 1#1: start worker process 31
2023/09/26 09:57:33 [notice] 1#1: start worker process 32
2023/09/26 09:57:33 [notice] 1#1: start worker process 33
2023/09/26 09:57:33 [notice] 1#1: start worker process 34
2023/09/26 09:57:33 [notice] 1#1: start worker process 35
172.17.0.1 - - [26/Sep/2023:10:00:54 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/117.0" "-"
2023/09/26 10:00:54 [error] 28#28: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 172.17.0.1, server: localhost, request: "GET /favicon.ico HTTP/1.1", host: "localhost:8081", referrer: "http://localhost:8081/" ←
172.17.0.1 - - [26/Sep/2023:10:00:54 +0000] "GET /favicon.ico HTTP/1.1" 404 153 "http://localhost:8081/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/117.0" "-"
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

DOCKER HUB | Alguns comandos adicionais (1/2)

- Quando se faz docker run o sistema cria sempre um *container* novo. Quando se faz docker stop o *container* é parado, mas não desaparece do sistema.
- Se o comando docker ps for utilizado com o argumento -a ou --all, todos os containers serão apresentados, estando ou não em execução.

➤ **docker ps -a**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d25d49db2a92	nginx:1.24	"/docker-entrypoint..."	About an hour ago	Exited (0) 4 seconds ago		festive_mestorf
f13957d2f595	nginx:1.24	"/docker-entrypoint..."	About an hour ago	Exited (0) About an hour ago		agitated_jang
6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Up 3 hours	80/tcp	loving_mclean
c1a4689fba50	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Exited (0) 3 hours ago		quirky_mirzakhan
b50296341367	docker/getting-started	"/docker-entrypoint..."	18 hours ago	Exited (255) 3 hours ago	0.0.0.0:80->80/tcp	thirsty_pike

- Para voltar a iniciar um *container*, pode utilizar-se o comando docker start em vez do docker run. O comando abaixo inicia dois *containers* em simultâneo.

➤ **docker start d25d49db2a92 f13957d2f595**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d25d49db2a92	nginx:1.24	"/docker-entrypoint..."	About an hour ago	Up 9 seconds	0.0.0.0:8081->80/tcp	festive_mestorf
f13957d2f595	nginx:1.24	"/docker-entrypoint..."	2 hours ago	Up 8 seconds	80/tcp	agitated_jang
6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Up 3 hours	80/tcp	loving_mclean
c1a4689fba50	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Exited (0) 3 hours ago		quirky_mirzakhan
b50296341367	docker/getting-started	"/docker-entrypoint..."	18 hours ago	Exited (255) 3 hours ago	0.0.0.0:80->80/tcp	thirsty_pike

continua no slide seguinte...

DOCKER HUB | Alguns comandos adicionais (2/2)

- Os *containers* podem ser referenciados de forma mais amigável através do seu nome. Para isso, quando se executa o `run`, utiliza-se o argumento `--name` seguido do nome pretendido para o *container*. O comando seguinte para os containers indicados (notar que um deles é referenciado pelo nome atribuído pelo sistema).

C:\>docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d25d49db2a92	nginx:1.24	"/docker-entrypoint..."	About an hour ago	Up 7 minutes	0.0.0.0:8081->80/tcp	festive_mestorf
f13957d2f595	nginx:1.24	"/docker-entrypoint..."	2 hours ago	Up 7 minutes	80/tcp	agitated_jang
6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Up 3 hours	80/tcp	loving_mclean

- `docker stop d25d49db2a92 agitated_jang`
- De seguida executa-se novamente um container mas, desta vez, indicando o argumento `--name`.
- `docker run --name MyApp -d -p 8081:80 nginx:1.24`

C:\>docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6872a575cdd3	nginx:1.24	"/docker-entrypoint..."	7 seconds ago	Up 5 seconds	0.0.0.0:8081->80/tcp	MyApp
6e13533c62ac	nginx:1.25.2-alpine	"/docker-entrypoint..."	3 hours ago	Up 3 hours	80/tcp	loving_mclean

- O container `MyApp` pode ser agora referenciado através do nome (exemplo abaixo).
- `docker logs MyApp`

```
C:\>docker logs MyApp
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
```

DOCKER | Custom Images

- Uma das vantagens em utilizar containers (e.g. Docker) é ser possível fazer um *package* com o código, dependências, e configurações, necessárias à execução de uma aplicação.
- Quando se termina o desenvolvimento de uma aplicação, todo o ambiente pode ser empacotado dentro de um Docker *container*, para facilitar o *shipping* da solução para o ambiente de testes ou produção.
- O programador cria uma imagem Docker (similar à utilizada anteriormente) e disponibiliza-a (num repositório público ou privado), para que possa ser instalado num servidor Docker Engine em execução.
- Para criar uma imagem personalizada, tem que se criar um ficheiro de configurações chamado *Dockerfile*. Posteriormente, com base neste ficheiro de configurações, pede-se ao Docker para gerar uma *Image* com base nessas configurações. A imagem resultante pode ser instanciada no *container* imediatamente no ambiente local, ou pode ser disponibilizada num repositório público ou privado para ser consumido por outros utilizadores / sistemas.
- O *Dockerfile* é um ficheiro de texto simples, que contém os comandos necessários, para que o Docker saiba criar uma imagem automaticamente.

DOCKER | CUSTOM IMAGES | Demonstração Dockerfile (1/5)

- Neste exemplo vai ser criada uma pequena aplicação em Node.js para ser demonstrada a criação de uma Docker Image personalizada.
- Depois de convenientemente testada, a aplicação vai ser empacotada dentro de um Docker Image, para tornar mais simples o seu *shipping* para outros sistemas (e.g. ambiente de testes e/ou produção). A imagem que vai ser criada, terá que conter as dependências necessárias à execução autónoma dentro do *container*.
- Depois da imagem criada, pode ser submetida para um repositório público ou privado para ser reutilizada por outros utilizadores, ou então pode ser instanciada localmente para ser testada.
- Para esta demonstração, é necessário ter o Node.js e npm instalado e a funcionar na máquina local. Os comandos seguintes validam se ambos os componentes estão prontos para serem utilizados na máquina de desenvolvimento.

➤ `node --version` D:\>node --version
v18.18.0

➤ `npm --version` D:\>npm --version
9.8.1

continua no slide seguinte...

DOCKER | CUSTOM IMAGES | Demonstração Dockerfile (2/5)

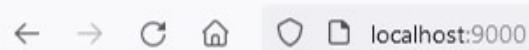
- Crie uma pasta para a nova solução Node.js e crie lá dentro os ficheiros: **package.json** e **app1.js**.
- Para que o Node.js consiga executar esta aplicação, as dependências têm que estar disponíveis no sistema. O comando seguinte garante que as dependências do **package.json** ficam disponíveis no SO local.

➤ **npm install**

- O comando seguinte inicia aplicação, carregando o Node.js como servidor da aplicação.

➤ **node app1.js**

- Para testar o funcionamento do serviço no computador local bastará aceder ao endereço <http://localhost:9000>



continua no slide seguinte...

```
package.json > ...
1  {
2    "name": "app1",
3    "version": "1.0",
4    "dependencies": {
5      "express": "4.18.2"
6    }
7 }
```

```
app1.js > ...
1  const express = require('express')
2  const app = express()
3  const port = 9000
4
5  app.get('/', (req, res) => {
6    res.send('app1 is working fine...')
7  })
8
9  app.listen(port, () => {
10    console.log(`app1 is on port ${port}`)
11 })
```

DOCKER | CUSTOM IMAGES | Demonstração Dockerfile (3/5)

- A lógica apresentada no slide anterior, deve ser reutilizada na definição do Dockerfile. Cada linha neste ficheiro de configurações, deve seguir a mesma lógica para configuração do ambiente de execução.
- Adicione o Dockerfile dentro da solução anterior, e configure o conteúdo seguinte (ver imagem):
 - FROM node:20.7-slim – imagem base a usar.
 - COPY . /app1/ – copiar ficheiros para a pasta app1 da imagem.
 - WORKDIR /app1 – mudar para a pasta app1 (dentro da imagem).
 - RUN npm install – instala as dependências necessárias.
 - CMD ["node", "app1.js"] – executa o programa.
- Por fim, pedir a criação da imagem tendo por base as definições anteriores (executar dentro da pasta onde está o Dockerfile).
 - docker build -t app1:1.0 .

```
Dockerfile > ...
1 # what is the base image
2 FROM node:20.7-slim
3
4 # copy content to the image
5 COPY package.json /app1/
6 COPY app1.js /app1/
7 #COPY . /app1/
8
9 # change current folder to app1
10 WORKDIR /app1
11
12 # download dependencies
13 RUN npm install
14
15 # programa execution
16 CMD ["node", "app1.js"]
17
```

continua no slide seguinte...

DOCKER | CUSTOM IMAGES | Demonstração Dockerfile (4/5)

- Por fim, basta pedir a criação da imagem (executar dentro da pasta onde está o Dockerfile). O comando abaixo cria uma imagem chamada app1 com a tag 1.0. O último ponto (.) indica que os ficheiros base à criação da *Image* estão na pasta atual.

➤ docker build -t app1:1.0 .

```
D:\Pendentes\IPP\LDS\Datas\Nodejs_app1>docker build -t app1:1.0 .
[+] Building 18.0s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 328B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:20.7-slim
=> [auth] library/node:pull token for registry-1.docker.io
`--> sha256:6fa8bba1fd18f87ea37f2588ha92214ea98a5531770af06h72r48a0212846h17 18.0s
`--> sha256:5d16d23ed3962045a3579h76h351f0a2af18d9c27861ddd17eed69d648he665d 0.0s
=> [internal] load build context
=> => transferring context: 555B
=> [2/5] COPY package.json /app1/
=> [3/5] COPY app1.js /app1/
=> [4/5] WORKDIR /app1
=> [5/5] RUN npm install
=> exporting to image
=> => exporting layers
=> => writing image sha256:aa269ff1fb4169a0737546ca005b6eb2a437d498b8f09e87a7933cb74bd42c1d
=> => naming to docker.io/library/app1:1.0
```

➤ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
app1	1.0	aa269ff1fb41	10 minutes ago	253MB
nginx	1.24	22c2ef579d56	5 days ago	142MB
nginx	latest	61395b4c586d	5 days ago	187MB
nginx	1.25.2-alpine	433dbc17191a	4 weeks ago	42.6MB
docker/getting-started	latest	3e4394f6b72f	9 months ago	47MB

continua no slide seguinte...

DOCKER | CUSTOM IMAGES | Demonstração Dockerfile (5/5)

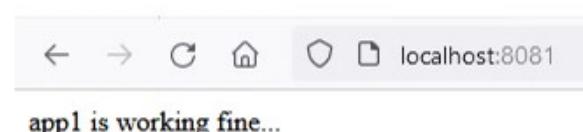
- Falta apenas testar se a aplicação está a funcionar corretamente. Os comandos abaixo criam um *container* da imagem, fazem o *bindind* da porta 9000 e testar num *browser*.

➤ `docker run -d -p 8081:9000 app1:1.0` D:\>docker run -d -p 8081:9000 app1:1.0
431551d698f27a29288205c845634adbdfc9eb4f80d66cd6551346f0044aee60

➤ `docker ps` D:\>docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
431551d698f2	app1:1.0	"docker-entrypoint.s..."	43 seconds ago	Up 42 seconds	0.0.0.0:8081->9000/tcp	practical_galileo

➤ <http://localhost:8081>



app1 is working fine...

➤ `docker logs 431551d698f2` D:\>docker logs 431551d698f2
app1 is on port 9000

COMANDOS DO DOCKER CLI

Comando	Descrição
<code>docker --version</code>	Versão atualmente instalada do Docker.
<code>docker images</code>	Lista todas as Imagens disponíveis no Docker.
<code>docker ps</code>	Lista todos os <i>containers</i> em execução.
<code>docker ps -a</code>	Lista todos os <i>containers</i> , incluindo os que estão desligados.
<code>docker pull nginx</code>	Download da imagem <code>latest</code> do nginx.
<code>docker pull nginx:1.25.2-alpine</code>	Download da <i>image</i> <code>1.25.2-alpine</code> do nginx.
<code>docker run -d nginx:1.25.2-alpine</code>	Executa o <i>container</i> , o <code>-d</code> executa em segundo plano.
<code>docker create --name alp nginx:1.25.2-alpine</code>	Cria o <i>container</i> com o nome <code>alp</code> mas não o executa (de seguida tem que se executar o <code>docker start</code>).
<code>docker logs nginx:1.25.2-alpine</code>	Apresenta os <i>logs</i> de arranque do <i>container</i> .
<code>docker stop f13957d2f595</code>	Termina a execução do container com id <code>f13957d2f595</code> .
<code>docker start d25d49db2a92 f13957d2f595</code>	Inicia os dois containers indicados em simultâneo.
<code>docker build -t app1:1.0 .</code>	Cria uma imagem com o nome <code>app1</code> e <i>tag</i> <code>1.0</code> , tendo por base a diretoria atual (significado do último ponto).

Os comandos / instruções possíveis de executar com o Docker são as mesmas entre todos os SO.

COMANDOS PERMITIDOS NO FICHEIRO Dockerfile (1/2)

Comando	Descrição
FROM	Define a imagem base que será usada como ponto de partida para a construção da nova imagem. Exemplo: FROM node:20.7-slim
COPY	Copia um ou mais ficheiros do <i>host</i> para o <i>container</i> . Exemplo: COPY app.jar /app/
ADD	Também copia ficheiros, mas consegue também descompactar, copiar ficheiros a partir de URL, etc. Exemplo: ADD https://example.com/file.tar.gz /
WORKDIR	Muda a diretoria de trabalho para a indicada. Exemplo: WORKDIR /app1
RUN	Executa comandos no terminal durante a construção da imagem para instalar pacotes, configurar o ambiente, etc. Exemplo: RUN npm install
CMD	Define o comando padrão a ser executado quando o <i>container</i> é iniciado. Exemplo: CMD ["node", "app1.js"]
ENTRYPOINT	Define o comando que será executado quando o <i>container</i> é iniciado. Exemplo: ENTRYPOINT ["java", "-jar", "app.jar"]

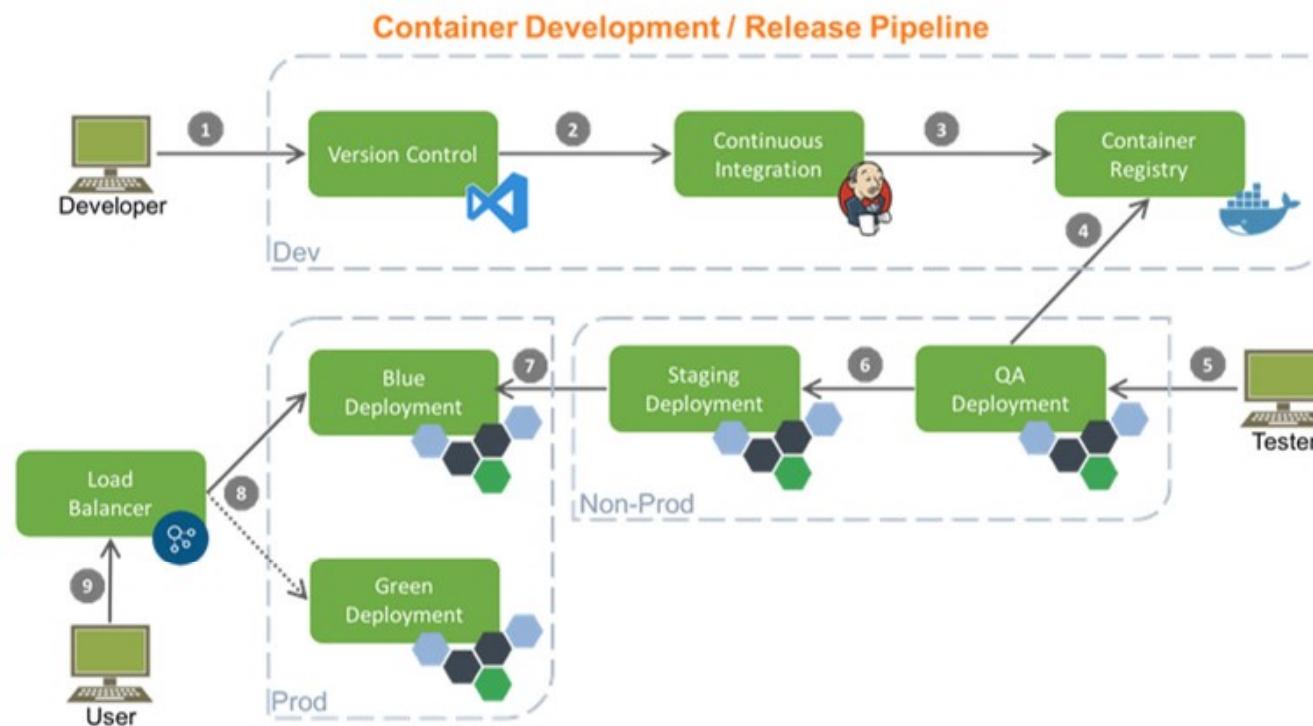
COMANDOS PERMITIDOS NO FICHEIRO Dockerfile (2/2)

Comando	Descrição
EXPOSE	Indica ao Docker quais as portas a expor dentro do <i>container</i> . A porta não fica disponível no <i>host</i> . Exemplo: EXPOSE 8080
ENV	Define variáveis de ambiente no <i>container</i> . Exemplo: ENV MY_VAR=value
VOLUME	Cria um local de armazenamento persistente ou para partilha de dados entre o <i>host</i> e o <i>container</i> . Exemplo: VOLUME /container_directory Posteriormente, se o <i>container</i> for criado com o argumento -v , a diretoria do <i>container</i> pode ser mapeada para um caminho físico no sistema operativo <i>host</i> . Exemplo: docker run -v /host_directory:/container_directory app1:1.0

PAPEL DO DOCKER NO CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE

- O Docker pode ser uma peça importante no ciclo de vida de desenvolvimento do software. Pode ser útil em várias situações, desde uniformizar os ambientes de desenvolvimento, até facilitar a entrada em produção.
- No ciclo de desenvolvimento, tipicamente os programadores fazem *commit* do seu trabalho para um repositório de código que, por sua vez, está integrado com um sistema CICD (e.g. Gitlab).
- Durante a fase de integração (CI) é criada automaticamente uma imagem Docker com a aplicação em execução, e é feito um push para um repositório (normalmente privado).
- Posteriormente o servidor em produção faz o pull dessa imagem e instancia-a criando *containers* tendo por base essa imagem.
- A utilização de sistemas de “*containerização*” como o Docker, facilita a entrada em produção, tornando-a mais isenta de problemas e menos propensa a erros.

PAPEL DO DOCKER NO CICLO DE VIDA DO DESENVOLVIMENTO DE SOFTWARE



EXERCÍCIOS

- <https://learn.microsoft.com/en-us/dotnet/core/docker/build-container?tabs=windows&pivots=dotnet-7-0>
- <https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-7.0>

RECURSOS ÚTEIS

- <https://docs.docker.com/>
- <https://www.docker.com/101-tutorial/>
- <https://docker-curriculum.com>
- <https://robertgreiner.com/example-docker-deployment-pipeline/>

P. PORTO