



Programação III

Introdução ao Docker

SUMÁRIO

01

Introdução

03

Comandos

02

Docker

04

Dockerfile



“**Containerization** is the process of encapsulating software code along with all of its dependencies **inside** a **single package** so that it can be **run consistently anywhere.**”

– IBM (n.d.)

DOCKER

O Docker é uma plataforma de "containerização" de código aberto (*open-source*) e baseado no standard *Open Container Initiative (OCI)*. Este fornece a capacidade de **executar aplicações** num **ambiente isolado** conhecido como **'container'**.

Existem outras plataformas do género como é o caso do Podman, rkt (CoreOs), CRI-O (Google), LXD (LXC), entre outras.





CONTAINERS

Os containers são como máquinas virtuais **muito leves** que podem ser executadas diretamente no kernel do nosso **sistema operativo** (SO) sem a necessidade de um 'hypervisor' - *software que cria e executa máquinas virtuais*.

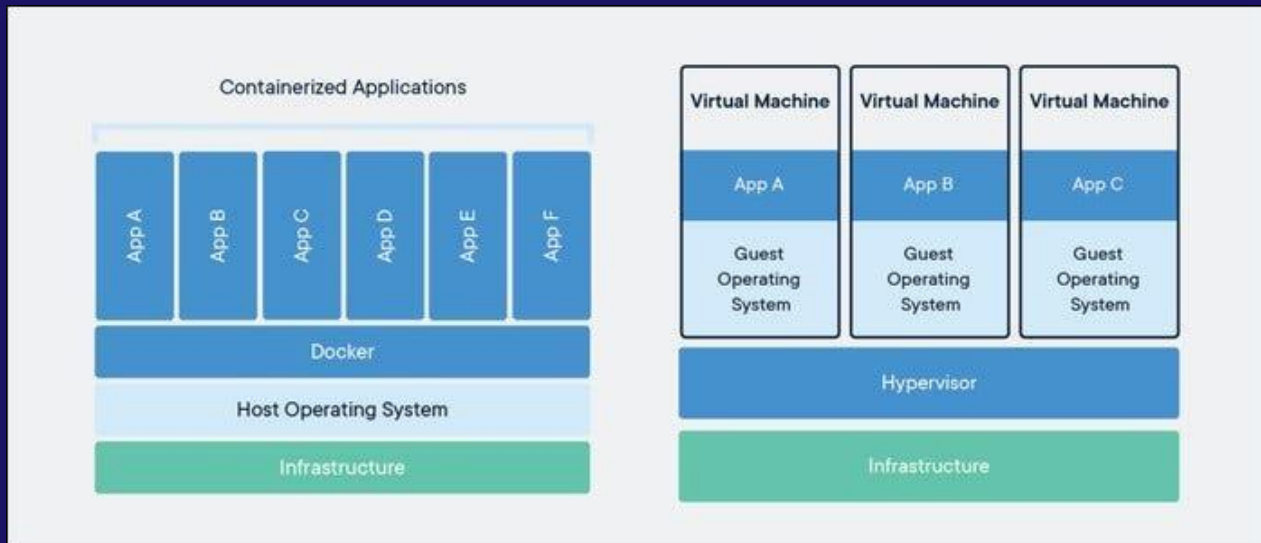
Como resultado, podemos executar **vários containers** em **simultâneo**.



“A **virtual machine** is the emulated equivalent of a **physical computer system** with their **virtual CPU**, memory, storage and operating system.”

– IBM (n.d.)

Containerização vs Virtualização



Cada container contém uma aplicação junto com todas as suas dependências e é isolado dos outros. Os programadores também podem trocar os **containers** como **imagem** por meio de um **registo**, como também podem fazer *deploy* diretamente nos servidores.



“A **container** is an abstraction at the **application layer** that **packages** code and dependencies together. Instead of virtualizing the entire physical machine, containers **virtualize the host operating system only.**”

– Docker (n.d.)



02

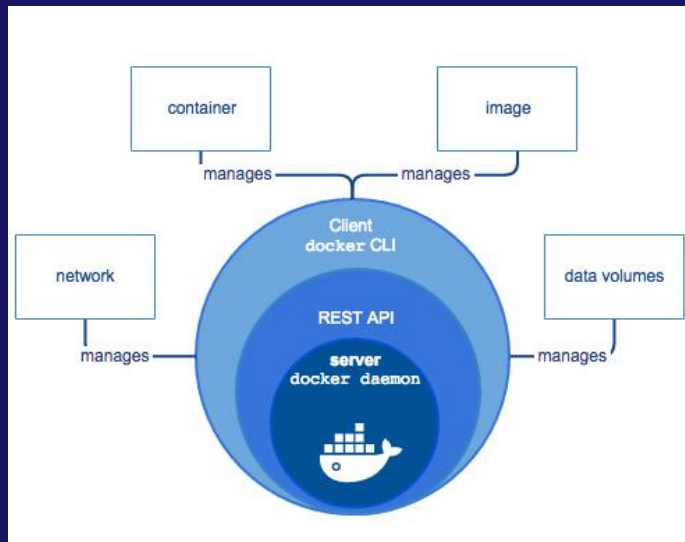
Docker

COMO INSTALAR O DOCKER?

É necessário ir ao site oficial do [Docker](#) e fazer download da aplicação 'Docker for Desktop'.

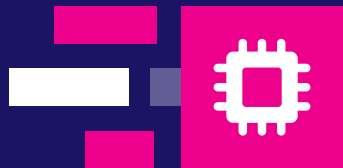
ATENÇÃO: é obrigatório utilizar o Windows 10 Pro/Enterprise (15063+) ou Windows 10 Home (build 2004+, lançada em Maio de 2020). No caso de não ser esta a versão instalada, utilizem esta [ferramenta](#) para forçar a atualização do Windows.

ARQUITETURA DO DOCKER



O Docker usa uma arquitetura **cliente-servidor** e funciona tendo por base três componentes principais - docker client, API e o docker daemon.

ARQUITETURA DO DOCKER



Docker Daemon

É um processo que se encontra em execução em segundo plano, que recebe comandos vindos do *docker client*. O *daemon* é capaz de gerir vários objetos docker (imagens, containers, volumes, etc.).

Docker Client

É a interface com o utilizador, neste caso, um CLI, que é responsável por executar os comandos.



Docker API

Ponte entre o *daemon* e o client. Qualquer comando executado usando o *docker client* passa pela API para finalmente alcançar o *daemon*.



IMAGENS vs CONTAINERS

IMAGENS



As **imagens** (*images*) são ficheiros autocontidos em várias camadas que atuam como **modelos** para a **criação de containers**.

Podemos usar imagens construídas por terceiros ou também criar as nossas próprias imagens (containers personalizados).

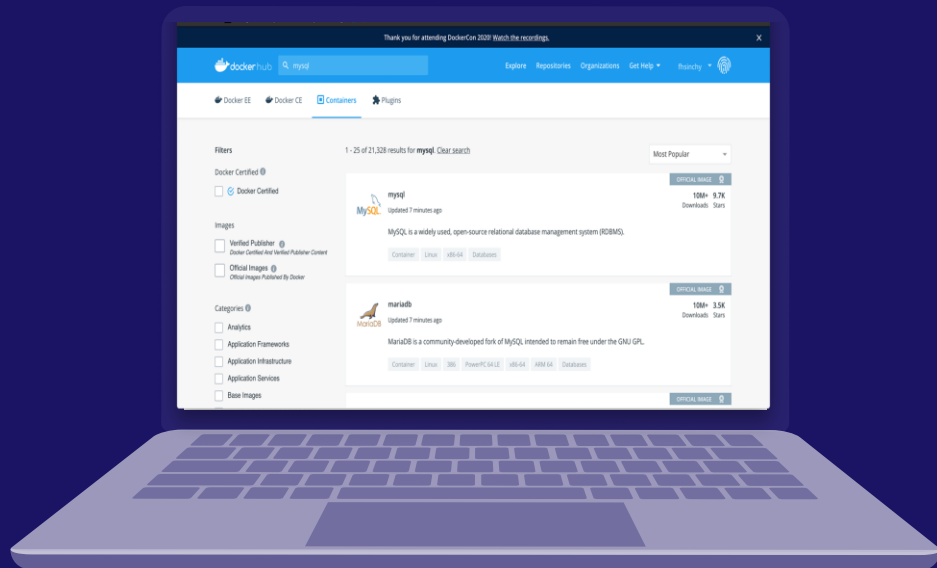
CONTAINERS



Quando **executamos** uma **imagem** este cria um ambiente isolado adequado para a execução do programa incluído na imagem. Este ambiente isolado é um **container**.

Os **containers** são então **imagens** em estado de **execução**.

hub.docker.com



REGISTOS

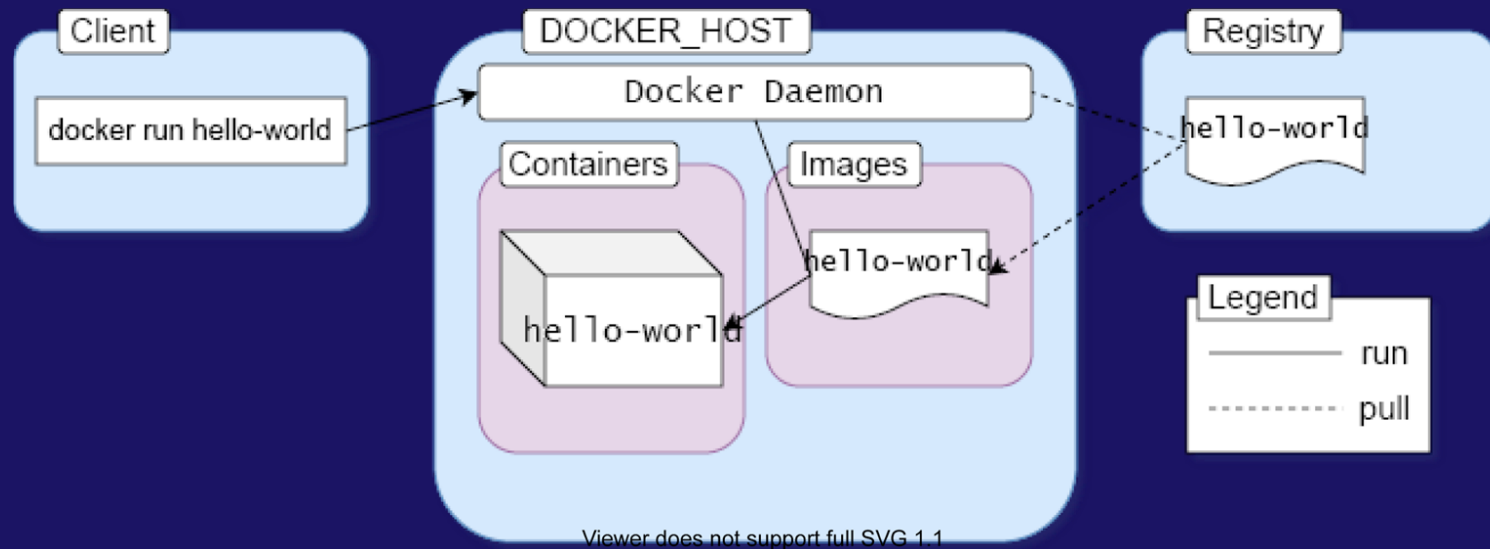
É o sistema de armazenamento de imagens do Docker.

O Docker Hub é um repositório público de imagens. Sempre que executamos comandos como o `docker run` ou `docker pull`, o docker procura primeiro a imagem nos registos locais e de seguida no hub (online).

Como é executado um *container* no *Docker*?



EXECUÇÃO DE UM CONTAINER



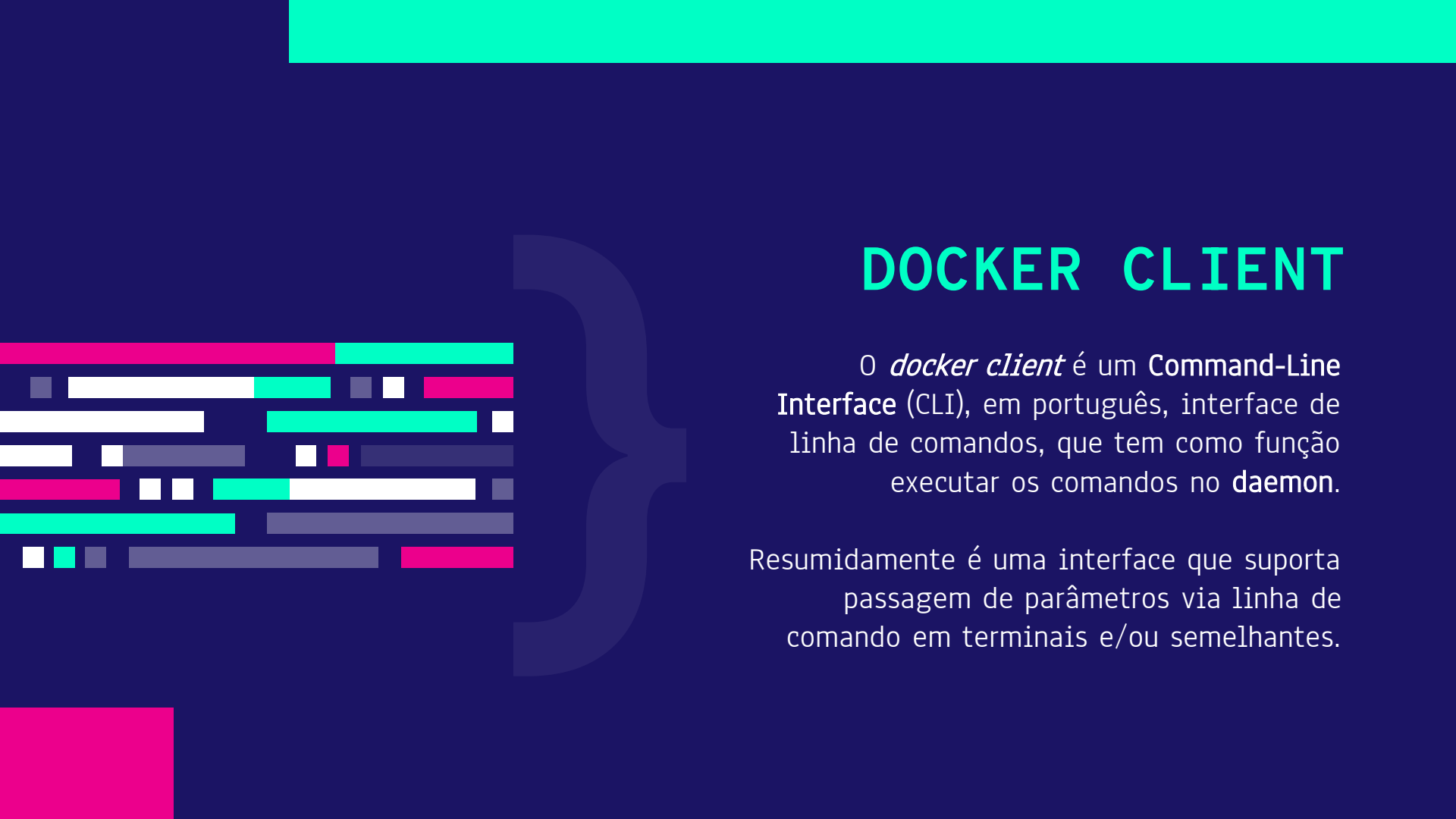
Todo o processo acontece em 5 etapas:

- 1) o utilizador executa o comando **docker run**;
- 2) o *docker client* informa o *docker daemon* que deve executar um container usando a imagem 'hello-world';
- 3) o *docker daemon* extrai a versão mais recente da imagem dos seus **registos**;
- 4) cria um **container** a partir da imagem;
- 5) **executa** o **container** recém-criado.



03

Comandos



DOCKER CLIENT

O *docker client* é um Command-Line Interface (CLI), em português, interface de linha de comandos, que tem como função executar os comandos no **daemon**.

Resumidamente é uma interface que suporta passagem de parâmetros via linha de comando em terminais e/ou semelhantes.

PRINCIPAIS COMANDOS DO DOCKER



ATENÇÃO: o comando `docker run` cria e inicia um novo container todas as vezes que é executado. Deste modo, lembre-se sempre de iniciar os containers criados anteriormente usando o comando `docker start` (e não o `run`).

Executar

O comando `docker run` permite criar e executar um container tendo por base uma **imagem**.

A sintaxe para este comando é: `docker run <image name>`.

Iniciar

O comando `docker start` permite iniciar um container, tendo por base o seu **ID** ou **nome**. Este ID é único e identifica o container.

A sintaxe para este comando é: `docker start <container id/name>`.



Na criação do container é gerado um **ID** único no formato *long string*. Para iniciar o container necessitamos apenas de utilizar os primeiros **12 caracteres** do ID.

PRINCIPAIS COMANDOS DO DOCKER



É possível utilizar o parâmetro `--name` para criar um container com um nome, embora o seu ID continue a existir. Atenção que não é possível dar nomes repetidos ou iguais às imagens.

Criar

O comando `docker create` permite criar um container a partir de uma determinada **imagem**. No final retorna ao CLI o ID do **container** criado.

A sintaxe para este comando é: `docker create <image name>`.

Reiniciar

O comando `docker restart` permite reiniciar um container que se encontre em execução.

A sintaxe para este comando é: `docker restart <container id/name>`.



Atenção que o comando `start` inicia os containers que não estão em execução, enquanto o `restart` termina um container em execução e o inicia novamente.

PRINCIPAIS COMANDOS DO DOCKER



Parar

O comando `docker stop` ou `docker kill` permite parar um container em execução. Também é possível parar um container com o CTRL+C.

A sintaxe para este comando é: `docker stop <container id/name>` ou `docker kill <container id/name>`.

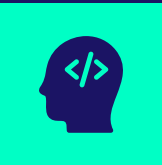
Executar

O comando `docker exec` permite executar comandos nos containers.

A sintaxe para este comando é: `docker exec <container id/name> <command>` ou `docker exec -it <container id/name> <command>`.



PRINCIPAIS COMANDOS DO DOCKER



Remover

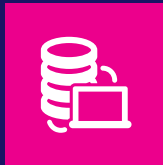
O comando `docker rm` permite **remover** um **container** dos registos.

A sintaxe para este comando é: `docker rm <container id/name>`.

Eliminar Tudo

O comando `docker system prune` permite **eliminar** todos os objetos do Docker (imagens, containers, volumes, etc.).

O Docker pedirá confirmação da operação. Podemos usar o parâmetro `'-f'` ou `'--force'` para saltar a confirmação.



Como executar *containers* no modo interativo?



DESCRIÇÃO DO CENÁRIO

É possível criar um container que inclua um sistema operativo (SO) dentro dele - por exemplo, distribuições Linux como o Ubuntu, Fedora, Debian, etc.

Se executarmos um container que inclua um destes SO com o comando `docker run`, ficamos com a sensação que não acontece nada. Estávamos à espera de ver o bash do sistema para executar comandos linux.


```
docker run -it <image>
```

Para que tal seja possível no *docker* é necessário indicar explicitamente que queremos uma sessão interativa, utilizando para tal o parâmetro **-it**.

ATENÇÃO

Nas nossas aulas iremos necessitar deste parâmetro sempre que quisermos aceder ao **bash** do **Node.js**. Para tal devemos utilizar o comando `docker run -it node.`

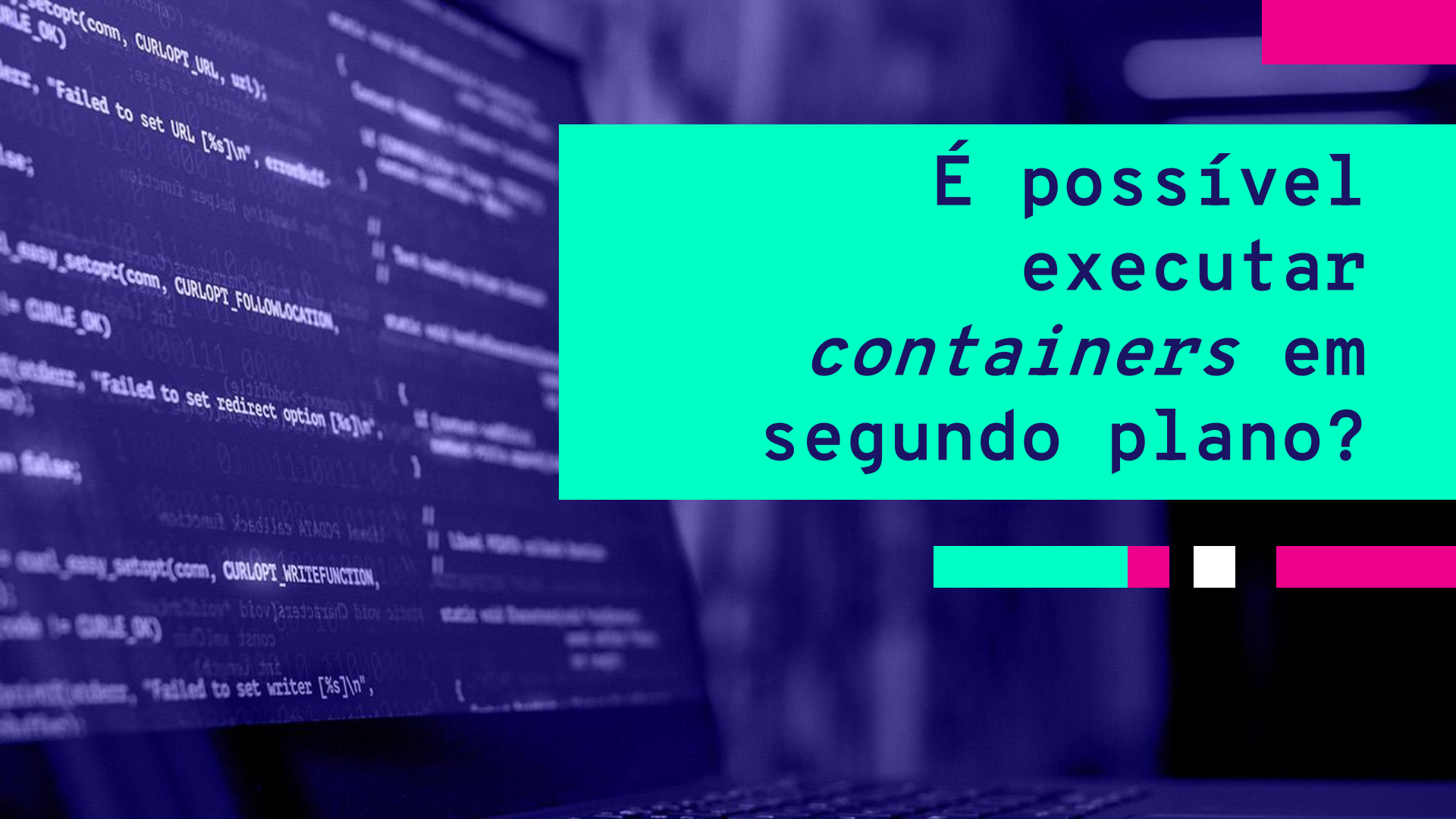
ARGUMENTOS

Utilizando o modo interativo também é possível passar **argumentos** diretamente no próprio comando *docker*. A sintaxe é `docker run <image> <args>`.

Exemplo:

O comando `docker run -it node -v` irá executar um container com a imagem do *node*, em modo interativo, e executar o comando que permite visualizar a versão atual do *node* instalada no sistema.





É possível
executar
containers em
segundo plano?



CENÁRIO ESPECÍFICO

Imaginemos que pretendemos executar um container com um servidor MySQL que possa ser partilhado por diversas aplicações. O MySQL é um Sistema de Gestão de Bases de Dados Relacional (SGBDR) muito utilizado.

O ideal é executar este container em modo 'detached', um género de serviço do Windows, sendo este executado em segundo plano. Para tal utiliza-se o argumento `-d` ou `--detach`.

CONTAINER EM MODO 'DETACHED'

```
docker run -d
  --name mysql
-e MYSQL_ROOT_PASSWORD=123
  mysql:latest
```

-d

Executa o container em modo *detached*, isto é, em segundo plano.

--name

Atribui o nome 'mysql' ao container.

-e

Define a variável de ambiente 'MYSQL_ROOT_PASSWORD' com o valor '123'.

mysql:latest

Tag que garante que o container vai executar a última versão do MySQL.

Mapeamento de Portas





AMBIENTE ISOLADO

Os *containers* são **ambientes isolados** do seu *host* (a nossa máquina) - como também dos outros *containers*, logo o *host* não sabe nada sobre o que está acontecer dentro dos *containers*.

Para que o acesso seja possível é necessário fazer o **mapeamento das portas** entre o **host** e o **container**. Necessitamos de utilizar o argumento `-p` ou `--port` no comando `docker run`.

CONTAINER COM MAPEAMENTO DE PORTAS

A sintaxe é `docker run -p <host port:container port> <image>`.

```
docker run -d
  --name mysql
  -p 3306:3306
-e MYSQL_ROOT_PASSWORD=123
mysql:latest
```

-p

Executa o mapeamento da porta 3306 do host (nosso computador) com a porta 3306 do **container**. É necessário garantir que a porta está disponível para ser compartilhada.

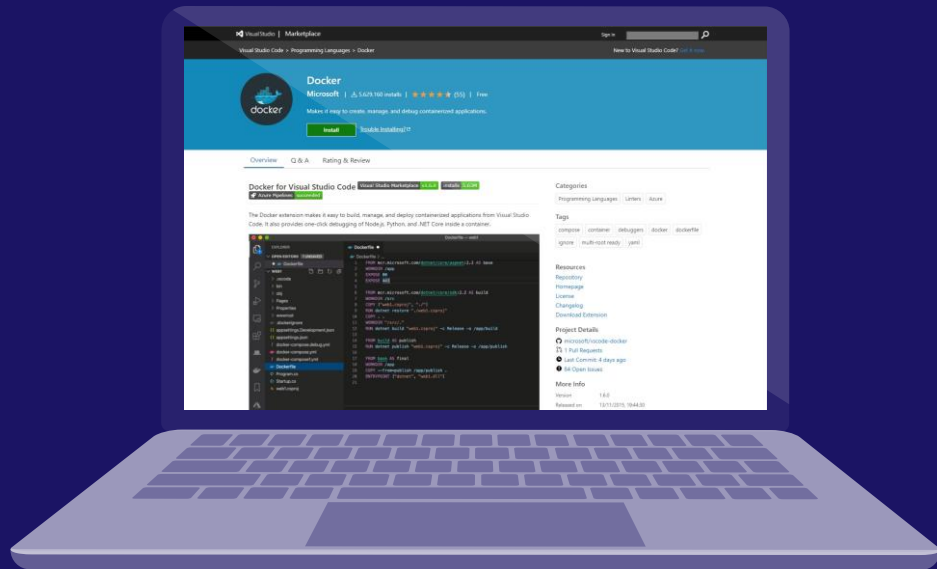




CONTAINERS EM EXECUÇÃO

É possível visualizar os container em execução utilizando o comando `docker ps` e também é possível inspecionar um container com o comando `docker inspect <container id/name>`.

VISUAL STUDIO CODE + DOCKER



Instalar no VSCode o plugin 'Docker' (aqui) disponível no marketplace da Microsoft. Este plugin permite gerir todos os objetos do Docker através da UI do VSC.

Atenção: é necessário ter o *Docker* instalado no computador.

Exercício 1

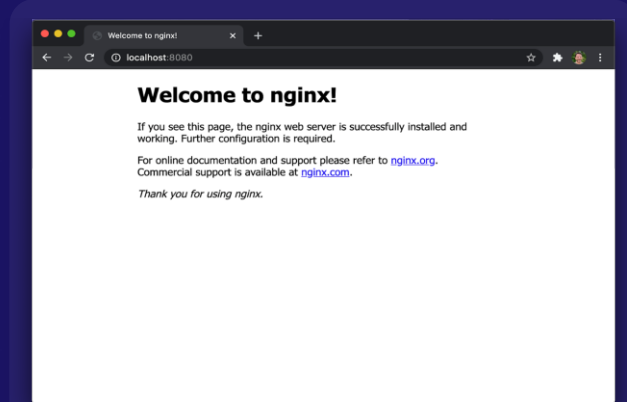


EXERCÍCIO

Recorrendo ao CLI do *docker*, execute os comandos necessários para **criar e executar** um **container** que inclua o servidor Web 'Nginx'. Garanta que podemos aceder ao servidor através da porta 80.

www.nginx.com

<http://localhost:80>



SOLUÇÃO DO EXERCÍCIO

```
docker run  
--name nginx  
-p 80:80  
nginx:latest
```

run

Cria e executa o container de acordo com os parâmetros e imagem definidos.

--name

Atribui o nome 'nginx' ao container.

-p 80:80

Faz o mapeamento da porta 80 do *host* com a porta 80 do *container*. O container fica acessível do exterior.

nginx

Nome da imagem a utilizar no container.

Exercício 2

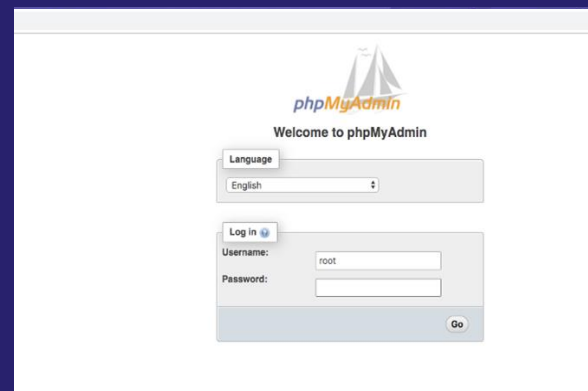


EXERCÍCIO

Recorrendo ao *Docker Hub*, execute os comandos necessários para **criar e executar** a **imagem 'phpMyAdmin'**. Garanta que podemos aceder à ferramenta através da porta 8081. Este registo já inclui o servidor web '**Nginx**', pelo que recomenda-se apagar o container criado anteriormente.

<http://localhost:8081>

phpmyadmin.net



SOLUÇÃO DO EXERCÍCIO

```
docker run -d  
--name phpmyadmin  
-p 8081:80  
phpmyadmin:latest
```

--name

Atribui o nome 'phpmyadmin' ao container.

-p 8081:80

Faz o mapeamento da porta 8081 do *host* com a porta 80 do *container*. O container fica acessível do exterior.

phpmyadmin:latest

Nome da imagem a utilizar no container.



04

Dockerfile

DESCRIÇÃO DO CENÁRIO

Com o intuito de permitir criar **imagens** específicas para projetos específicos (web, mobile, etc.), o Docker dispõe de um ficheiro - chamado **Dockerfile**, que permite criar imagens **personalizadas** através da definição de um conjunto de instruções que são executados no momento da criação do *container*.

ATENÇÃO

No exemplo que analisaremos de seguida, iremos criar uma **imagem** personalizada para a construção de containers que incluam o **Node.js**. Esta imagem é útil para projetos na área do desenvolvimento *back-end*.

No seu **workspace** de projetos 'Node' crie uma diretoria chamada '**docker_example**' e dentro dessa diretoria crie o ficheiro '**Dockerfile**' (sem extensão).

CÓDIGO DO DOCKERFILE

```
FROM ubuntu
RUN apt-get update
RUN apt-get install nodejs -y
CMD ["node"]
```

FROM

Toda a *dockerfile* começa com o comando *FROM*. Neste definimos a imagem base a utilizar no *container*.

RUN

Utilizamos o gestor de packages (apt-get) do Ubuntu para instalar o Node.

CMD

Este comando indica qual a aplicação a executar no arranque do container. Atenção que só pode haver um CMD por *dockerfile* e não se pode utilizar apóstrofes em detrimento das aspas.

Como se executa um Dockerfile?



```
docker build <build context>
```

Este comando requer um dockerfile e um contexto, isto é, um caminho/*path* (localização absoluta) de onde se encontra armazenado o conteúdo e o ficheiro. O Docker procurará o 'dockerfile' no contexto e o usará para construir a imagem.

CÓDIGO DO DOCKERFILE

```
docker build .
```

build

• Cria imagens a partir de um *dockerfile* e um "contexto". O contexto é o conjunto de diretorias e ficheiros localizados no *PATH* ou especificado na *URL*.

•
O "." (ponto) representa a diretoria atual. Se o ficheiro estivesse dentro de um outro diretório, por exemplo, 'src', então era necessário escrever './src'.



No final deste processo, se tudo correr bem, devemos ver algo como “Successfully built <id>”. Este ID é da imagem, e não do container.

Para executar um container tendo por base esta imagem devemos executar o comando `docker run -it <id>` (é necessário o parâmetro `-it` porque o Node é REPL).

Atenção que o ID da imagem varia a cada criação (*build*).

Como incluir um projeto numa imagem personalizada?



DESCRIÇÃO DO CENÁRIO

Tendo por base o *Node*, vamos criar uma imagem que já **inclua** um determinado **projeto** que se encontra na nossa máquina (*host*) é necessário seguir as seguintes etapas: 1) instalar as dependências necessárias do projeto - executando o comando `npm install`; 2) Iniciar a aplicação;



Usando as instruções do *dockerfile*, implementaremos as seguintes etapas:

1. Usar uma imagem que permita executar aplicações em *Node*;
2. Copiar para o *container* o ficheiro '*package.json*' e instalar as respetivas dependências;
3. Copiar todos os ficheiros do projeto para o *container*;
4. Iniciar a aplicação.

CÓDIGO DO DOCKERFILE

```
FROM node
WORKDIR /usr/app
COPY ./package.json ./
RUN npm install
COPY . .
CMD ["npm", "run", "start"]
```

workdir

Define a diretoria onde irá ser instalado o projeto como "app". Esta será armazenada na pasta 'usr', que no caso do Linux é a diretoria onde normalmente se instalam os programas criados a partir do código fonte.

copy

Este comando copia as diretorias e ficheiros de um local para o outro, isto é, do *host* para o *container*. O "." (ponto) representa a raiz do projeto.

EXECUÇÃO DO CONTAINER

Para fazer o *build* à imagem é necessário executar o comando `docker build ..`.

Para criar um container de acordo com o projeto é necessário executar o comando `docker run -it -p 3000:3000 <id da imagem>` - em que o ID da imagem é retornado no momento do build da imagem.

BUILD DA IMAGEM

```
docker build . -t  
<nome_da_imagem_a_criar>
```

- Define a diretoria atual como o conteúdo adicionar à imagem a criar.

- **-t <nome_da_imagem>**

Permite definir o nome pretendido para a imagem a criar.

```
docker build . -t projeto
```

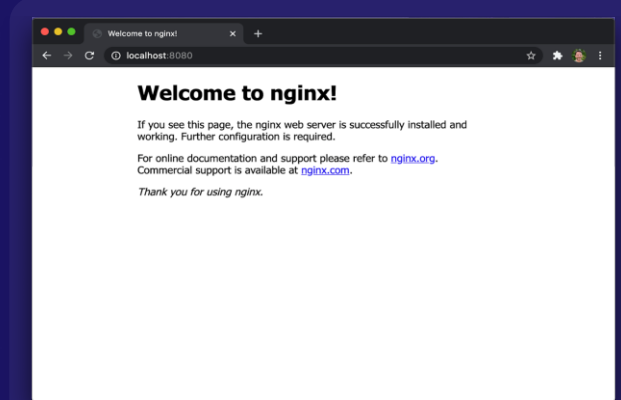
Exercício



EXERCÍCIO

Recorrendo ao **terminal** do **VSCode**, execute os comandos necessários para **criar e executar** um **container** que inclua uma **imagem personalizada** (projeto) de um projeto Node.js criado por si.

http://localhost:80



SOLUÇÃO DO EXERCÍCIO

```
docker run  
--name exemplo  
-p 80:80  
projeto
```

run

Cria e executa o container de acordo com os parâmetros e imagem definidos.

--name

Atribui o nome 'nginx' ao container.

-p 80:80

Faz o mapeamento da porta 80 do *host* com a porta 80 do *container*. O container fica acessível do exterior.

projeto

Nome da imagem a utilizar no container.

Docker Compose



CÓDIGO DO FICHEIRO

(docker-compose.yml)

```
services:
  webserver:
    build:
      context: .
    image: projeto
    container_name: 'app'
    links:
      - mysql
    depends_on:
      - mysql
    volumes:
      - ../home/node/app
    ports:
      - '5000:5000'
    environment:
      - MYSQL_PASS=123
    networks:
      - app-web-net
```

(...)

• context

Define a diretoria a criar o serviço NodeJS. Neste caso será na raiz da pasta da aplicação.

• image

O nome da imagem a utilizar no container.

• depends_on

Dependências do container (outros containers).

• volumes

Caminho (path) para os ficheiros. O "." representa a diretoria atual no nosso computador.

CÓDIGO DO FICHEIRO

(docker-compose.yml)

```
services:
```

```
(...)
```

```
mysql:
```

```
  image: mysql:latest
```

```
  container_name: 'mysql'
```

```
  restart: 'always'
```

```
  ports:
```

```
    - '3306:3306'
```

```
  environment:
```

```
    - MYSQL_ROOT_PASSWORD=123
```

```
  networks:
```

```
    - app-web-net
```

```
(...)
```

ports

Define a diretoria a criar o serviço NodeJS. Neste caso será na raiz da pasta da aplicação.

environment

O nome da imagem a utilizar no container.

container_name

Dependências do container (outros containers).

CÓDIGO DO FICHEIRO

(docker-compose.yml)

```
services:
  (...)
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: 'phpmyadmin'
    links:
      - mysql
    depends_on:
      - mysql
    ports:
      - '8081:80'
    volumes:
      - /sessions
    environment:
      - PMA_HOST=mysql
      - PMA_PORT=3306
    networks:
      - app-web-net
```

• context

Define a diretoria a criar o serviço NodeJS. Neste caso será na raiz da pasta da aplicação.

• image

O nome da imagem a utilizar no container.

• depends_on

Dependências do container (outros containers).

• volumes

Caminho (path) para os ficheiros. O "." representa a diretoria atual no nosso computador.

CÓDIGO DO FICHEIRO

(docker-compose.yml)

```
networks:  
  app-web-net:
```

networks

O nome personalizado da rede a utilizar pelos containers.



EXECUÇÃO DO DOCKER COMPOSE

```
docker compose -p  
<nome_do_grupo_container> up
```

-p

Comando que permite definir um *parent container* onde serão criados os restantes *containers* definidos no ficheiro *docker*.

up

Comando para criar e executar os *containers*. Também existe o comando inverso, o *down* (parar e remover os *containers*).

```
docker compose -p projeto-aulas up
```