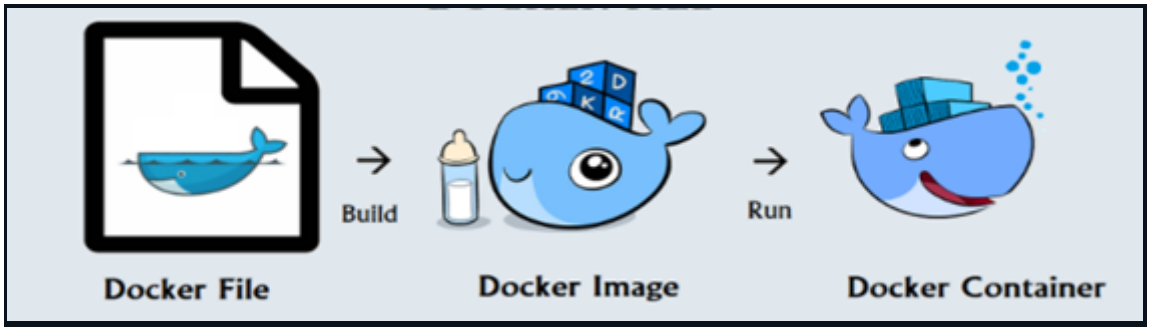
Certified Tech Developer
The Ultimate Degree

Dockerfile

Docker pode construir imagens automaticamente lendo instruções de um **Dockerfile**. Um Dockerfile é um documento de texto que contém todos os comandos que um usuário pode chamar na linha de comando para construir uma imagem. Porém como veremos é muito mais simples eu usar um documento com todas as instruções gravadas do que ficar digitando em um terminal sempre que for necessário. Este documento servirá como uma documentação do que conterá na imagem, além da possibilidade de rodar quantas vezes for necessário sem precisar digitar novamente todas as informações.

Usando o ***docker build*** podemos construir uma imagem baseada nas instruções declaradas no **Dockerfile**.



Dentro de um Dockerfile podemos ter várias instruções, vamos ver as principais e o que elas fazem.

Aproveitamos a explicação sobre **Dockerfile,** para criar uma imagem que pode ser utilizada com qualquer projeto HTML que você tenha, ele pode conter CSS e JS (Javascript que seja executado no navegador como um Client Side).

FROM é uma instrução obrigatória, pois é ela que indica qual imagem será utilizada como base, podemos utilizar uma imagem que contém um Sistema Operacional mais uma aplicação, por exemplo, Python ou apenas colocar uma S.O, exemplo, Ubuntu, no nosso caso iremos utilizar uma imagem base que vem com um S.O mais o Apache:

```
FROM httpd:2.4
```

LABEL esta instrução adiciona algumas etiquetas a imagem de forma a ajudar na organização e informações sobre a imagem, do caso vamos informar o mantenedor da imagem, aquele que criou a imagem:

```
LABEL maintainer="Nidio Dolfini"
```

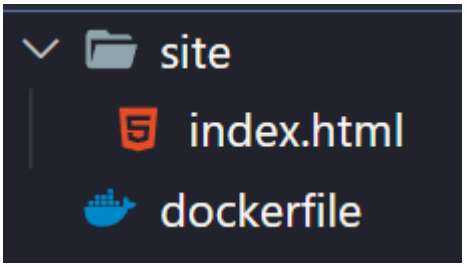
EXPOSE ela serve basicamente para informar, documentar, a porta que será utilizada pela aplicação que está sendo usada dentro do container, caso fosse MySQL deveríamos informar a porta 3306, no nosso caso será a porta HTTP que é a 80:

```
EXPOSE 80
```

WORKDIR Define uma pasta dentro do container onde serão executados os comandos e o diretório que será mostrado ao acessarmos o container via terminal interativo, neste caso como iremos utilizar o Apache como aplicação, iremos definir a pasta onde ficam os arquivos HTMLs do Apache como nosso diretório de trabalho:

```
WORKDIR /usr/local/apache2/htdocs/
```

COPY serve para copiar qualquer tipo de arquivo para dentro da imagem, em nosso caso podemos copiar a pasta site que tem os arquivos HTMLs necessários em nosso site para dentro da pasta do Apache, mas você pode copiar para uma pasta que você queira criar:




```
COPY /site /usr/local/apache2/htdocs/
```

Aqui já temos o suficiente para criar uma imagem baseada no Apache, que também contêm nosso projeto de site, pronto para ser acessado quando criarmos um container baseado nesta imagem:

```
FROM httpd:2.4

LABEL maintainer="Nidio Dolfini"

EXPOSE 80

WORKDIR /usr/local/apache2/htdocs/

COPY /site /usr/local/apache2/htdocs/
```

Para construirmos uma imagem baseada neste **Dockerfile**, precisamos estar no mesmo diretório do **Dockerfile** em um terminal de sua preferência e chamarmos o **docker** e fazer um **image build** como o comando abaixo:

```
docker image build -t nomedaimagem .
```

Neste comando estamos chamando o **build** de imagem do **Docker** para construir uma imagem baseada no **Dockerfile** está no mesmo diretório, dizemos isso fazendo a referência do caminho utilizando o **" . "**, que significa que o arquivo está no diretório atual. Passamos a FLAG **-t** (--TAG) para passar um nome para imagem.

```
➜docker image build -t nomedaimagem .
[+] Building 8.5s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 185B                                                0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load metadata for docker.io/library/httpd:2.4                        1.5s
 => [1/3] FROM docker.io/library/httpd:2.4@sha256:2d1f8839d6127e400ac5f65481d8      3.4s
 => => resolve docker.io/library/httpd:2.4@sha256:2d1f8839d6127e400ac5f65481d8      0.0s
 => => sha256:ea211260dba688134865f1f0f11e2f4b4877f05df1624839 1.36kB / 1.36kB      0.0s
 => => sha256:c58ef9bfbb5789a9882cee610ba778b1368d21b513d6caf3 9.04kB / 9.04kB      0.0s
 => => sha256:214ca5fb90323fe769c63a12af092f2572bf1c6b300263 31.38MB / 31.38MB      1.2s
 => => sha256:7cf31a2eeec6ac0953b123fc95d3f54bd0b08038aa69791279d1 175B / 175B      0.6s
 => => sha256:bf666e57b9f28b78cde9f890d0af95d0a75c0ed4d95c 917.19kB / 917.19kB      0.5s
 => => sha256:2d1f8839d6127e400ac5f65481d8a0f17ac46a3b91de40b0 1.86kB / 1.86kB      0.0s
 => => sha256:c15a4e94ae6b799e14422b659d15e9dc4d84de6fad9257 24.14MB / 24.14MB      2.3s
 => => sha256:dc25474c7f97bb2bfe901e77ab63a2e725a82c11cd0e36eb4e8c 295B / 295B      0.8s
 => => extracting sha256:214ca5fb90323fe769c63a12af092f2572bf1c6b300263e098839      1.2s
 => => extracting sha256:7cf31a2eeec6ac0953b123fc95d3f54bd0b08038aa69791279d18      0.0s
 => => extracting sha256:bf666e57b9f28b78cde9f890d0af95d0a75c0ed4d95c5a3fb186d      0.1s
 => => extracting sha256:c15a4e94ae6b799e14422b659d15e9dc4d84de6fad9257e4d871a      0.7s
 => => extracting sha256:dc25474c7f97bb2bfe901e77ab63a2e725a82c11cd0e36eb4e8c2      0.0s
 => [internal] load build context                                                   0.5s
 => => transferring context: 57.89MB                                                0.5s
 => [2/3] WORKDIR /usr/local/apache2/htdocs/                                        2.8s
 => [3/3] COPY /site /usr/local/apache2/htdocs/                                     0.5s
 => exporting to image                                                              0.2s
 => => exporting layers                                                             0.2s
 => => writing image sha256:a605edaf0b2f5ec5062b342d53f83bf9dcdf3f239f329ced12      0.0s
 => => naming to docker.io/library/nomedaimagem                                     0.0s
```

Após essa saída no terminal de que a imagem foi gerada, podemos verificar utilizando o comando de lista imagens do **Docker.**

```
docker image ls
```

```
➜docker image ls
REPOSITORY                      TAG       IMAGE ID       CREATED              SIZE
nomedaimagem                    latest    a605edaf0b2f   About a minute ago   201MB
```

Sua imagem está pronta para uso, mas temos outros comandos importantes, vamos dar uma olhada neles:

RUN com ele podemos colocar comando que são executados no momento de criação da imagem, podemos utilizar o por exemplo, **RUN apt-get install apache2,** pois ele faria a instalação do Apache como fazemos em um terminal Linux, podemos usar também o **RUN npm install,** para chamarmos o **NPM** caso estejamos usando uma imagem que tenha o **NodeJS,** abaixo um comando para chamar o compilador Java em uma imagem que tenha o OpenJDK instalado:

```
RUN ["javac", "HelloWorld.java"]
```

CMD executa um comando quando o container for criado, ele serve para executar um arquivo do Java mostrando uma saída que tem no arquivo HelloWorld, esta saída será mostrada no terminal do container. Também podemos utilizar o npm start em um container que contém o NodeJS ou iniciar o serviço do TomCat. No caso abaixo estamos chamando a máquina virtual Java para executar o arquivo HelloWorld:

```
CMD ["java", "HelloWorld"]
```