



Neste artigo vou apresentar os conceitos básicos relativos ao Docker sob o ponto de vista de um desenvolvedor .NET.



Até o momento já sabemos criar contêineres a partir de imagens, compartilhar recursos de contêineres e baixar e tratar imagens. ([artigo anterior](#))



Sabemos também que existem repositórios contendo imagens prontas que podemos baixar e usar conforme a nossa necessidade. Assim, o [Docker Hub](#) contém uma ampla variedade de imagens prontas.

Ocorre que nem sempre a imagem que desejamos usar esta pronta do jeitinho que queremos, e, muitas vezes, teremos que ter uma imagem customizada. Nestes casos não tem jeito: vamos ter que criar as nossas próprias imagens.

Veremos a seguir como podemos criar imagens.

Criando imagens usando Dockerfile

O Docker constrói imagens automaticamente lendo as instruções a partir de um arquivo texto chamado Dockerfile (*esse processo é conhecido como processo de build*) que contém todos os comandos em ordem necessários para criar uma imagem.

Assim, podemos criar a nossa própria imagem através do arquivo [Dockerfile](#); ele é o responsável por criar as imagens no Docker, é nele que definimos todas as regras, informações e instruções que estarão contidas nas imagens.

Mas o que é esse tal de [Dockerfile](#) ?

O [Dockerfile](#) é um arquivo responsável por realizar a criação e construção de imagens no Docker, dentro dele são definidas instruções que o Docker vai seguir para conseguir realizar a criação de uma imagem. Essas instruções são interpretadas linha a linha pelo engine do Docker.

Para definir as regras e instruções que a imagem terá, devemos utilizar comandos que o Dockerfile entende. Pense no [Dockerfile](#) como um arquivo de lote que contém [instruções/comandos](#) com uma sintaxe definida ([INSTRUÇÃO argumento](#)) que devemos seguir para que uma imagem seja criada.

Assim, uma imagem Docker consiste de camadas somente-leitura onde cada camada representa uma instrução do [Dockerfile](#).

Para poder mostrar os comandos que o [Dockerfile](#) usa e como ele funciona, vamos usar o Dockerfile para criar uma imagem, e, depois, a partir dessa imagem, vamos criar um contêiner que, a partir do [debian 8](#), instale e inicie o servidor web [nginx](#).

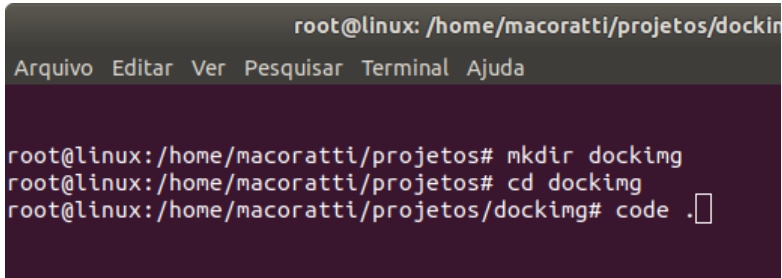
A primeira tarefa que temos que fazer é definir os passos que vamos usar para criar a imagem.

1. Definir uma imagem base
2. Definir informações para a imagem
3. Definir o nome de quem criou o arquivo
4. Executar comandos para instalar o nginx na imagem e iniciar o nginx
5. Expor a porta do contêiner e definir em qual porta o servidor vai atender
6. Definir o ponto de entrada a aplicação
7. Definir a execução de um comando para inicializar o servidor nginx

Crie uma pasta chamada [docking](#) no seu host, entre na pasta, e, crie um arquivo com o nome [Dockerfile](#) sem extensão: (*Eu vou usar o VS Code para criar o arquivo [Dockerfile](#)*)

Nota: O nome do arquivo tem que ser exatamente [Dockerfile](#)

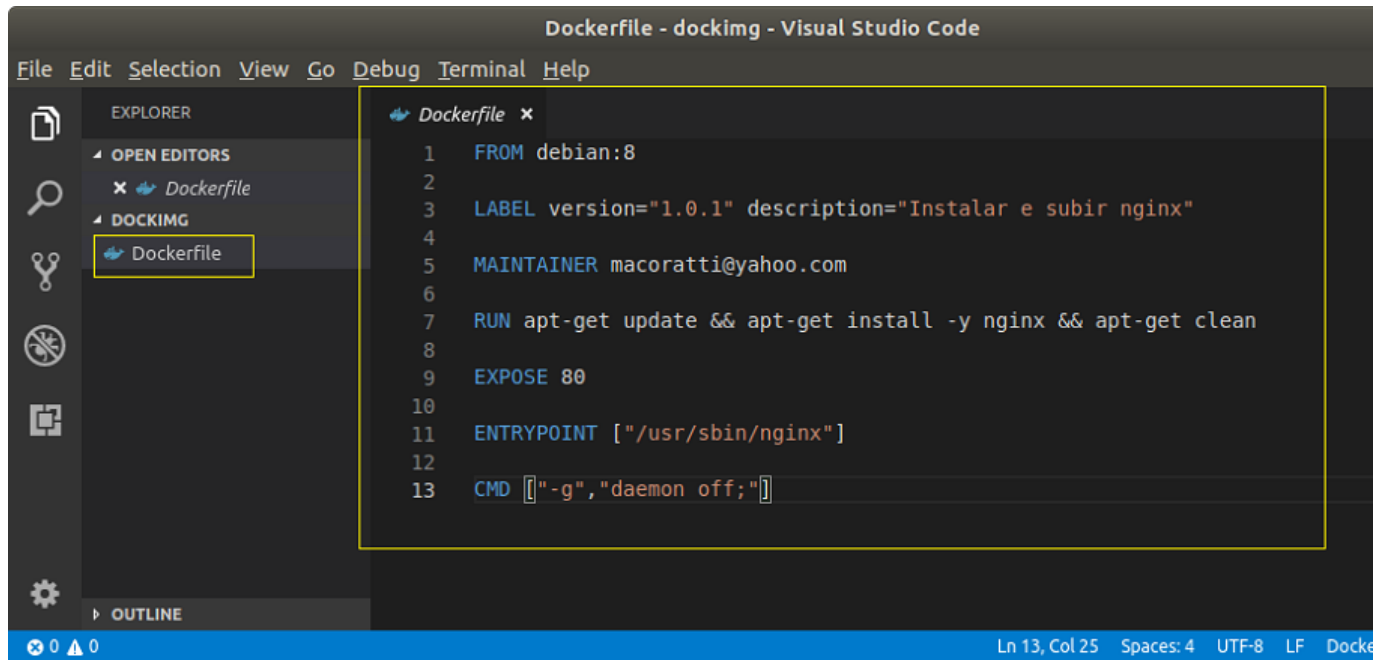
```
mkdir dockimg
cd dockimg
code .
```



```
root@linux: /home/macoratti/projetos/dockimg
Arquivo Editar Ver Pesquisar Terminal Ajuda

root@linux:/home/macoratti/projetos# mkdir dockimg
root@linux:/home/macoratti/projetos# cd dockimg
root@linux:/home/macoratti/projetos/dockimg# code .
```

Com o [Visual Studio Code](#) aberto crie o arquivo **Dockerfile** e inclua o código abaixo:



```
Dockerfile - dockimg - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
├ OPEN EDITORS
├ Dockerfile
└ DOCKIMG
  └ Dockerfile

Dockerfile
1 FROM debian:8
2
3 LABEL version="1.0.1" description="Instalar e subir nginx"
4
5 MAINTAINER macoratti@yahoo.com
6
7 RUN apt-get update && apt-get install -y nginx && apt-get clean
8
9 EXPOSE 80
10
11 ENTRYPOINT ["/usr/sbin/nginx"]
12
13 CMD ["-g", "daemon off;"]

Ln 13, Col 25 Spaces: 4 UTF-8 LF Dockerfile
```

Os comandos definidos no arquivo **Dockerfile** foram:

```
FROM debian:8
LABEL version="1.0.1" description="Instalar e subir nginx"
MAINTAINER macoratti@yahoo.com
RUN apt-get update && apt-get install -y nginx && apt-get clean
EXPOSE 80
ENTRYPOINT ["/usr/sbin/nginx"]
CMD ["-g", "daemon off;"]
```

Vamos entender cada uma das instruções usadas:

FROM - Podemos definir uma [imagem base](#) através da instrução **FROM** seguida pelo nome da imagem que queremos utilizar;

LABEL - Permite definir informações sobre a imagem como a versão, descrição, o criador, etc.;

MAINTAINER - Permite informar quem criou o arquivo;

RUN - Permite executar comandos conhecidos pelo [shell ou linux](#). Em nosso exemplo definimos um comando para instalar o [nginx](#) e informamos o parâmetro **-y** para aceitar a instalação;

EXPOSE - Expõe a porta do contêiner. A instrução não faz o mapeamento de porta, apenas deixa explícito que uma determinada porta pode ser mapeada durante a criação do container que utilizar essa imagem. (Por padrão o [nginx](#) sobe na porta 80)

ENTRYPOINT - Define um caminho onde o comando definido em **CMD** deve ser executado;

CMD - Define um ou mais comandos que serão executados. Necessita da informação do **ENTRYPOINT**. Assim as duas instruções se complementam. Para o exemplo, a execução do comando ficaria assim:

```
/usr/sbin/nginx -g daemon off;
```

Para o nosso exemplo estamos executando o comando para iniciar o **nginx** na imagem no modo iterativo.

Nota: A instrução **MAINTAINER** esta sendo depreciada e a informação do autor pode ser definida na instrução **LABEL** com a palavra **maintainer**.

Dessa forma já temos o arquivo **Dockerfile** criado para gerar a imagem. Para fazer isso usamos o comando **build** e informamos o **nome da imagem**, a **tag** e um **ponto(.)**. O comando fica assim:

```
docker build -t macoratti/nginx:1.0 .
```

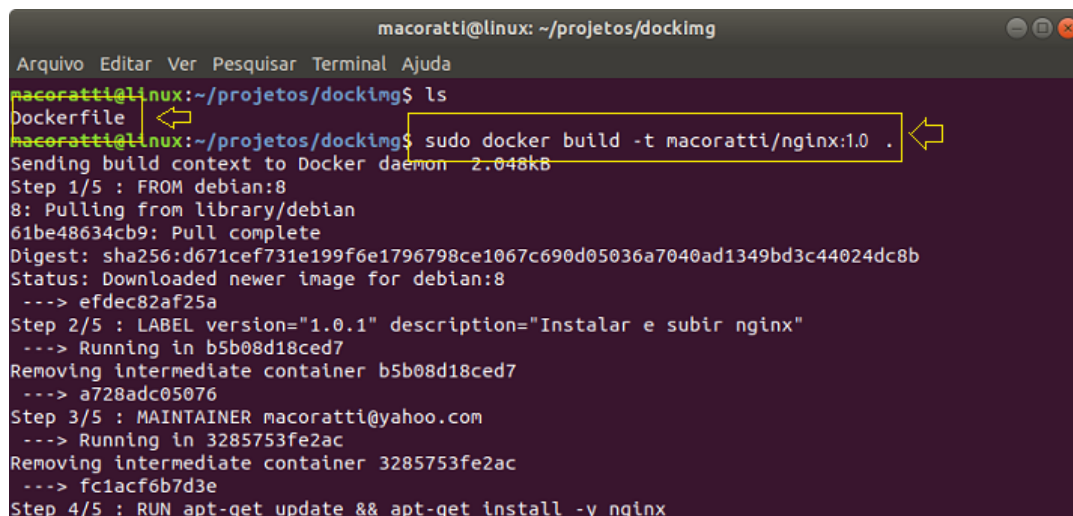
onde:

```
docker build -> O comando
-t          -> Parâmetro usado para informar que a imagem pertence ao meu usuário
macoratti/nginx:1.0 -> O nome da imagem e a tag atribuída à imagem
.          -> significa o diretório atual (pois dei o build dentro da pasta do Dockerfile)
```

Nota: O **build** não trabalha com o caminho do arquivo, apenas com o seu diretório, então é preciso informar o caminho do diretório ou, no nosso caso, como estamos no diretório onde se localiza o **Dockerfile**, apenas um **ponto(.)** para identificar que o **Dockerfile** está no diretório atual.

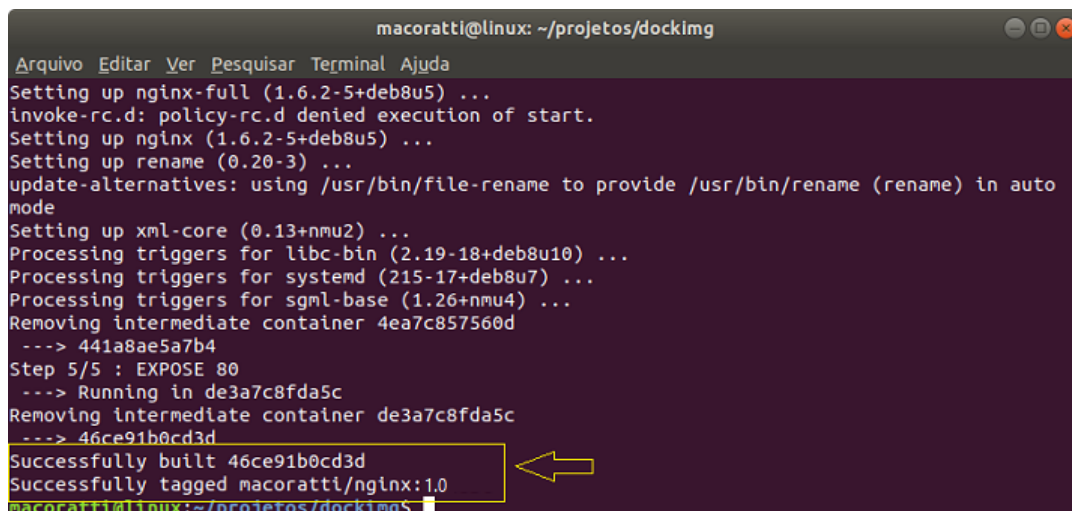
O comando **docker build** constrói uma imagem a partir de um **Dockerfile** e de um **contexto**. O **contexto** do build é o **conjunto de arquivos** na localização especificada **PATH** ou **URL**. O **PATH** é o diretório no seu sistema de arquivos local e a **URL** é a localização do repositório **GIT**.

Obs: Um contexto é processado recursivamente, assim **PATH** inclui os subdiretórios e a **URL** inclui o repositório e seus submódulos.



```
macoratti@linux: ~/projetos/dockimg
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/dockimg$ ls
Dockerfile
macoratti@linux:~/projetos/dockimg$ sudo docker build -t macoratti/nginx:1.0 .
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM debian:8
8: Pulling from library/debian
61be48634cb9: Pull complete
Digest: sha256:d671cef731e199f6e1796798ce1067c690d05036a7040ad1349bd3c44024dc8b
Status: Downloaded newer image for debian:8
--> efdec82af25a
Step 2/5 : LABEL version="1.0.1" description="Instalar e subir nginx"
--> Running in b5b08d18ced7
Removing intermediate container b5b08d18ced7
--> a728adc05076
Step 3/5 : MAINTAINER macoratti@yahoo.com
--> Running in 3285753fe2ac
Removing intermediate container 3285753fe2ac
--> fc1acf6b7d3e
Step 4/5 : RUN apt-get update && apt-get install -y nginx
```

Voltando ao **host** na pasta **dockimg** e executando o comando vemos o início do processamento acima, e, abaixo a conclusão indicando que nossa imagem foi criada com sucesso:



```
macoratti@linux: ~/projetos/dockimg
Arquivo Editar Ver Pesquisar Terminal Ajuda
Setting up nginx-full (1.6.2-5+deb8u5) ...
invoke-rc.d: policy-rc.d denied execution of start.
Setting up nginx (1.6.2-5+deb8u5) ...
Setting up rename (0.20-3) ...
update-alternatives: using /usr/bin/file-rename to provide /usr/bin/rename (rename) in auto mode
Setting up xml-core (0.13+nmv2) ...
Processing triggers for libc-bin (2.19-18+deb8u10) ...
Processing triggers for systemd (215-17+deb8u7) ...
Processing triggers for sgml-base (1.26+nmv4) ...
Removing intermediate container 4ea7c857560d
--> 441a8ae5a7b4
Step 5/5 : EXPOSE 80
--> Running in de3a7c8fda5c
Removing intermediate container de3a7c8fda5c
--> 46ce91b0cd3d
Successfully built 46ce91b0cd3d
Successfully tagged macoratti/nginx:1.0
macoratti@linux:~/projetos/dockimg$
```

Exibindo as imagens: [docker images](#)

```
macoratti@linux: ~/projetos/dockimg
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/dockimg$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
macoratti/nginx     1.0                1be6668aea57       8 minutes ago      202MB
microsoft/dotnet    2.1-aspnetcore-runtime 1fe6774e5e9e       2 weeks ago        255MB
nginx               latest             dbfc48660aeb       2 weeks ago        109MB
debian              latest             be2868bebaba       2 weeks ago        101MB
debian              8                  efdec82af25a       2 weeks ago        127MB
hello-world         latest             4ab4c602aa5e       8 weeks ago        1.84kB
macoratti@linux:~/projetos/dockimg$
```

Vemos a imagem **macoratti/nginx:1.0** criada com a **TAG** igual a **1.0**, o **ID**, data de criação e tamanho.

Agora para concluir, podemos criar um contêiner com a nossa imagem usando o comando:

docker container create -p 8080:80 macoratti/nginx:1.0

Neste comando estamos criando o contêiner usando parâmetro **-d** de forma que ele será executando em segundo plano(*daemon*).

Para conferir basta abrir o navegador em <http://localhost:8080>.

Para exibir o contêiner criado em execução : [docker container ps](#)

```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
c2e1b22cfba3   macoratti/nginx:1.0   "/usr/sbin/nginx -g ..."   29 seconds ago   Up 26 seconds   0.0.0.0:8080->80/tcp
macoratti@linux:~$
```

Vemos assim o nosso contêiner criado a partir de uma imagem que geramos localmente contendo o **debian** e o **nginx**.

Depois de criar a imagem podemos subir a imagem para o [Docker Hub](#) mas para isso é preciso criar uma conta no site e fazer, login e usar o comando **push** para enviar a imagem.

[docker login](#)

[docker push <nome_imagem>](#)

A seguir os comandos que podemos usar no [Dockerfile](#):

- **MAINTAINER**: Campo opcional, que informa o nome do mantenedor da nova imagem; Indica quem escreve o dockerfile; (*em depreciação*)
Ex: MAINTAINER macoratti@yahoo.com
- **RUN**: Executa os comandos dentro do container;
Ex: `apt-get update && apt-get install -y nginx`
- **ADD**: Adiciona arquivos, diretórios do host ou de uma url para dentro de um diretório do container, mas não funciona com arquivos compactados;
Ex: `ADD arquivo.txt /home/macoratti/`
- **CMD**: Define um comando a ser executado quando um container baseado nessa imagem for iniciado; Informa os parâmetros que serão usados no ENTRYPOINT. Também pode ser utilizado para informar qual comando será executado após a criação do container, mas pode ser sobrescrito caso seja passada algum parâmetro de execução de algum comando na hora de executar o container;
Ex: `CMD ["nginx", "-g", "daemon off;"]`
- **LABEL**: Coloca informações de metadados no container;
Ex: `LABEL description="Container Nginx", version="1.0.0"`
- **COPY**: Copia arquivos e diretórios do host para dentro de um diretório do container. Funciona com arquivos compactados;
Ex: `COPY arquivo.txt /home/macoratti/`
- **ENTRYPOINT**: Informa qual será a aplicação principal do container, sendo executada após a inicialização do container. Diferente do CMD, não será sobrescrita. Caso a seja finalizada a execução do programa definido no ENTRYPOINT, o container será encerrado;
Ex: `ENTRYPOINT ["nginx", "-D", "daemon off;"]`

- **ENV**: Instrução que cria e atribui um valor para uma variável dentro da imagem; útil para realizar a instalação de alguma aplicação ou configurar um ambiente inteiro. Permite definir variáveis de ambiente;
Ex: `NOME_AUTOR="Macoratti"`
- **EXPOSE**: Expõe uma ou mais portas, isso quer dizer que o container quando iniciado poderá ser acessível através dessas portas, ou seja, expõe a porta informada do container;
Ex: `EXPOSE 80`
- **USER**: Define qual será o usuário padrão para o container. Caso não seja definido, o padrão é o usuário root;
Ex: `USER macoratti`
- **WORKDIR**: Determina qual diretório será o diretório de trabalho;
Ex: `WORKDIR /home/macoratti/downloads`
- **VOLUME**: Mapeia um diretório do host para ser acessível pelo container; Permite assim a criação de um diretório no host onde ficam armazenados os dados do container;
Ex: `VOLUME /home/docker/nginx:/usr/share/nginx/html`

Na [próxima aula](#) vamos veremos como cria uma imagem para uma aplicação ASP .NET Core MVC.