



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



Neste artigo vamos tratar de **volumes** no Docker.([artigo anterior](#))

O recurso **volumes** separa os arquivos de dados que são gerados por um aplicativo ou banco de dados do restante do armazenamento do contêiner, o que facilita a substituição ou atualização de um contêiner.



Os **volumes** permitem que dados importantes existam fora do contêiner, o que significa que você pode substituir um contêiner sem perder os dados que ele criou.

Os **volumes** tornam possível excluir um contêiner sem também excluir o dados que contém, o que permite que os contêineres sejam alterados ou atualizados sem perder dados do usuário.

Para usar esse recurso usamos o comando **Volume**, e, ele deve ser preparado antes que os contêineres que usam os dados sejam criados.

Existem dois tipos de arquivos associados a um aplicativo:

- 1- Os arquivos necessários para executar o aplicativo;
- 2- Os **arquivos de dados** que o aplicativo gera enquanto é executado, que são tipicamente produzidos como resultado de ações do usuário;

No mundo Docker, esses dois tipos de arquivos são tratados de maneira diferente. Os arquivos necessários para executar o aplicativo fazem parte do **contêiner Docker** de um aplicativo. Quando o Docker processa as instruções em um **Dockerfile**, ele cria a **imagem** que forma o modelo para criar contêineres.

Para uma aplicação ASP.NET Core MVC, isso significa que os contêineres incluem o **.NET Core runtime**, **os pacotes ASP.NET Core**, **as classes C# personalizadas**, **a folha de estilo CSS do Bootstrap**, **as views Razor**, e **todos os arquivos de configuração**. Sem esses arquivos, a aplicação MVC em um contêiner não seria capaz de ser executado com sucesso.

Os **arquivos de dados** não são incluídos em contêineres. Um dos principais benefícios do uso de contêineres é que eles são fáceis de criar e destruir. Quando um contêiner é destruído, os arquivos em seu sistema de arquivos também são excluídos. Excluir um contêiner seria desastroso para os arquivos de dados, porque eles seriam perdidos para sempre.

O Docker fornece um recurso chamado **volumes** para gerenciar dados do aplicativo e veremos como os volumes funcionam, e como usar ferramentas disponíveis para trabalhar com eles e mostrar um tipo comum de aplicativo que usa volumes: **um banco de dados**.

Entendendo Volumes : demonstrando o problema

Creio que a melhor maneira de entender **Volumes** é mostrar o que acontece quando não usamos volumes em uma aplicação que gera dados.

Para isso vamos criar uma imagem usando a distribuição **Alpine** do Linux e gerar um **arquivo de texto** que contém uma mensagem e uma data que serão os dados da aplicação.

Vamos então criar o arquivo **Dockerfile** para gerar a imagem.

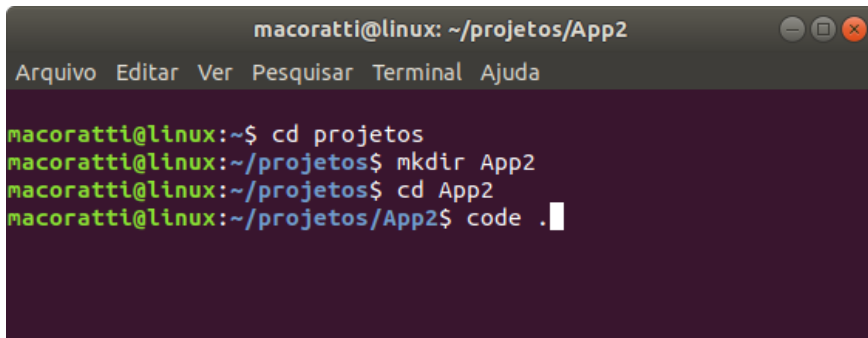
Criando o arquivo Dockerfile

A primeira coisa que temos que fazer é definir o passos que vamos usar para criar a imagem.

1. Definir uma imagem base;
2. Definir a pasta de trabalho;
3. Definir o ponto de entrada da aplicação e criar um arquivo texto com uma mensagem e data;

Vamos criar uma pasta **App2** dentro da pasta **projetos** e criar o arquivo **DockerFile** nesta pasta. Vamos entrar na pasta e abrir o Visual Studio Code digitando o comando code.

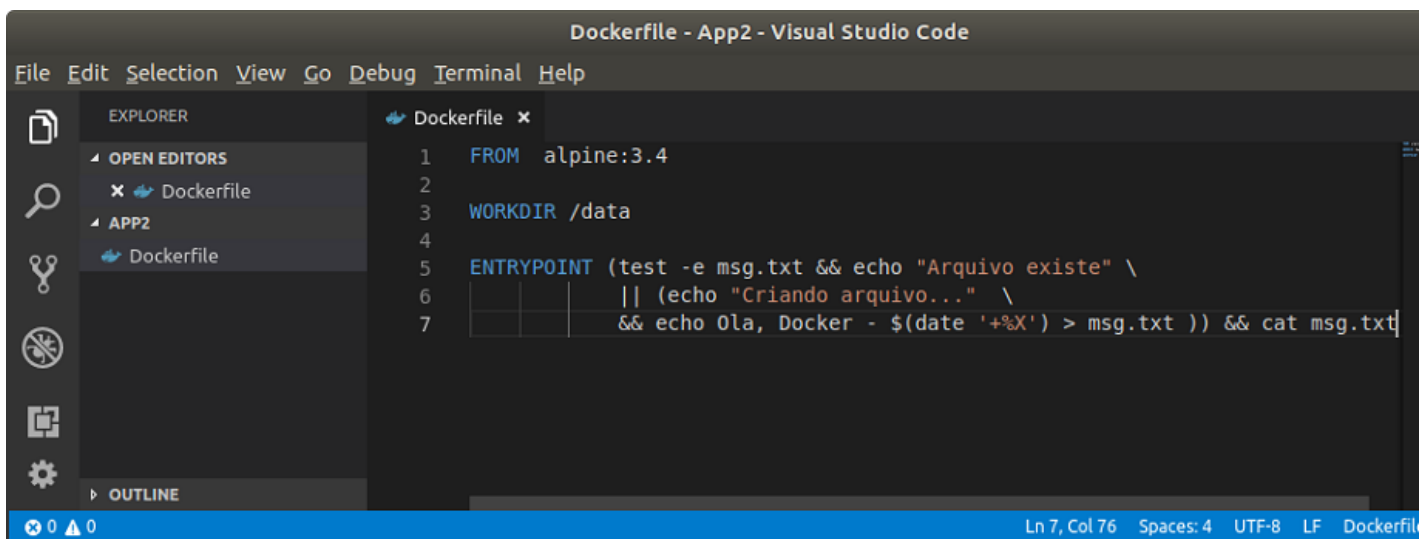
```
cd projetos
md App2
cd App2
code .
```



```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda

macoratti@linux:~$ cd projetos
macoratti@linux:~/projetos$ mkdir App2
macoratti@linux:~/projetos$ cd App2
macoratti@linux:~/projetos/App2$ code .
```

Com o Visual Studio Code aberto crie o arquivo **Dockerfile** e inclua o código abaixo:



```
Dockerfile - App2 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    Dockerfile
  APP2
    Dockerfile
  OUTLINE

Dockerfile x
1 FROM alpine:3.4
2
3 WORKDIR /data
4
5 ENTRYPOINT (test -e msg.txt && echo "Arquivo existe" \
6             || (echo "Criando arquivo..." \
7                 && echo Ola, Docker - $(date '+%X') > msg.txt )) && cat msg.txt
```

Nota: O nome do arquivo tem que ser exatamente **Dockerfile**

Os comandos definidos para o nosso arquivo **Dockerfile** foram:

```
FROM alpine:3.4
WORKDIR /data
ENTRYPOINT (test -e msg.txt && echo "Arquivo existe" \
            || (echo "Criando arquivo..." \
                && echo Ola, Docker $(date '+%X') > msg.txt)) && cat msg.txt
```

Vamos entender cada uma das instruções usadas:

FROM alpine:3.4
Definimos a **imagem base** como: **alpine:3.4**. Esta é uma distribuição mínima do Linux.

WORKDIR /data
Definimos a pasta **data** como diretório de trabalho

ENTRYPOINT (test -e msg.txt && echo "Arquivo existe" \
Para simular um aplicativo que gera dados, o comando **ENTRYPOINT** cria um arquivo de dados chamado **arquivo/data/msg.txt** que contém uma mensagem e um timestamp. O arquivo de dados não será criado até que o contêiner seja iniciado e não será parte da imagem criada do arquivo Docker, semelhante ao conteúdo de um

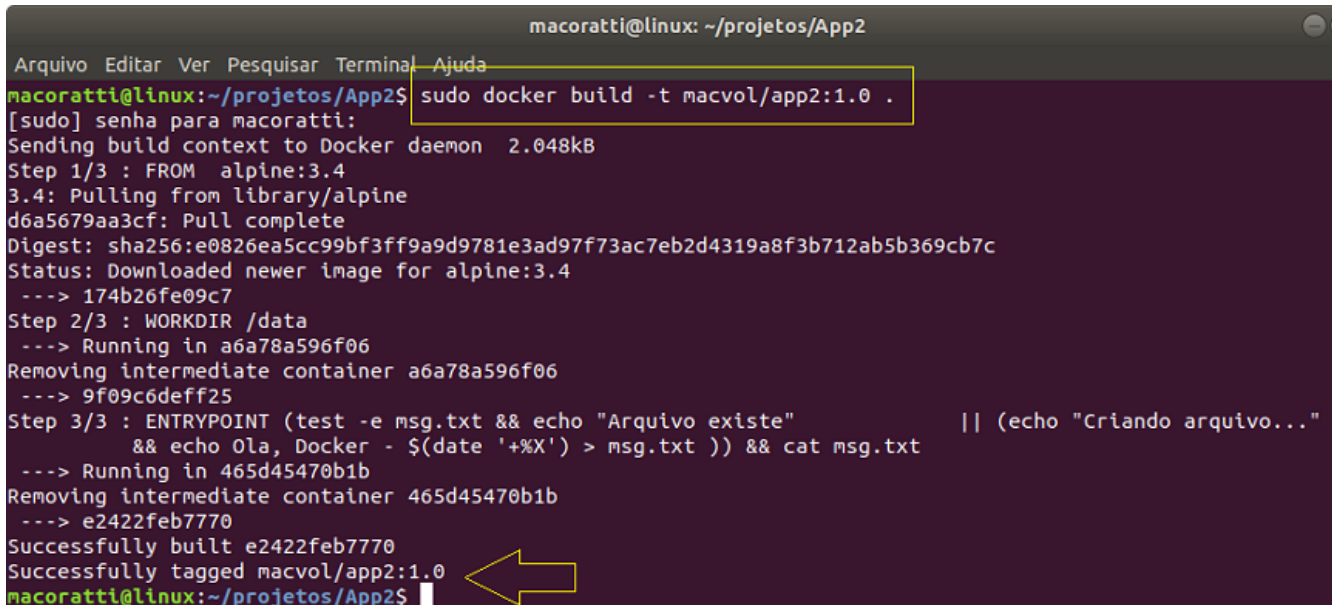
banco de dados em um aplicativo real.

Dessa forma já temos o arquivo [Dockerfile](#) criado para gerar a imagem. Vamos executar o [Dockerfile](#):

`docker build -t macvol/app2:1.0 .`

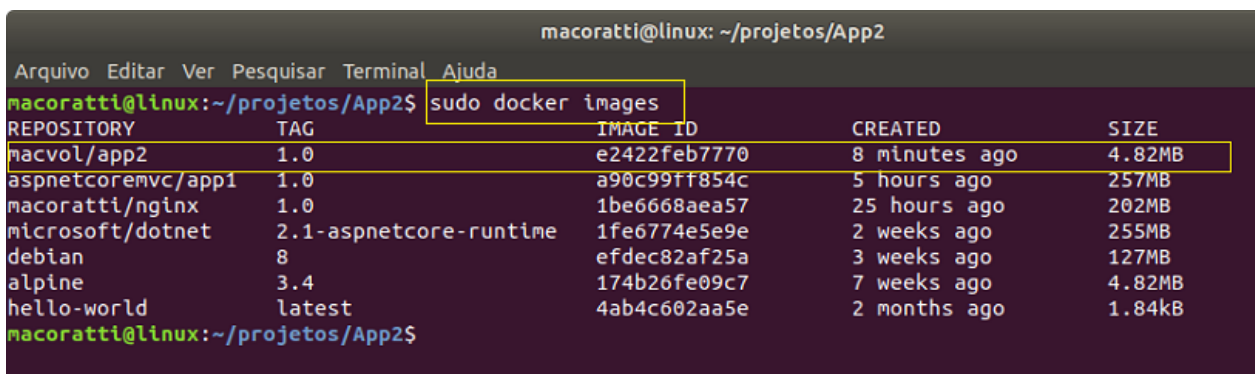
onde:

- `docker build`** -> O comando
- `-t`** -> Parâmetro usado para informar que a imagem pertence ao meu usuário
- `macvol/app2:1.0`** -> O nome da imagem e a tag atribuída à imagem
- `.`** -> significa o diretório atual (*pois dei o build dentro da pasta do DockerFile*)



```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker build -t macvol/app2:1.0 .
[sudo] senha para macoratti:
Sending build context to Docker daemon 2.048kB
Step 1/3 : FROM alpine:3.4
3.4: Pulling from library/alpine
d6a5679aa3cf: Pull complete
Digest: sha256:e0826ea5cc99bf3ff9a9d9781e3ad97f73ac7eb2d4319a8f3b712ab5b369cb7c
Status: Downloaded newer image for alpine:3.4
--> 174b26fe09c7
Step 2/3 : WORKDIR /data
--> Running in a6a78a596f06
Removing intermediate container a6a78a596f06
--> 9f09c6deff25
Step 3/3 : ENTRYPOINT (test -e msg.txt && echo "Arquivo existe" || (echo "Criando arquivo..."
&& echo Ola, Docker - $(date +%X') > msg.txt )) && cat msg.txt
--> Running in 465d45470b1b
Removing intermediate container 465d45470b1b
--> e2422feb7770
Successfully built e2422feb7770
Successfully tagged macvol/app2:1.0
macoratti@linux:~/projetos/App2$
```

Exibindo as imagens: **`docker images`**



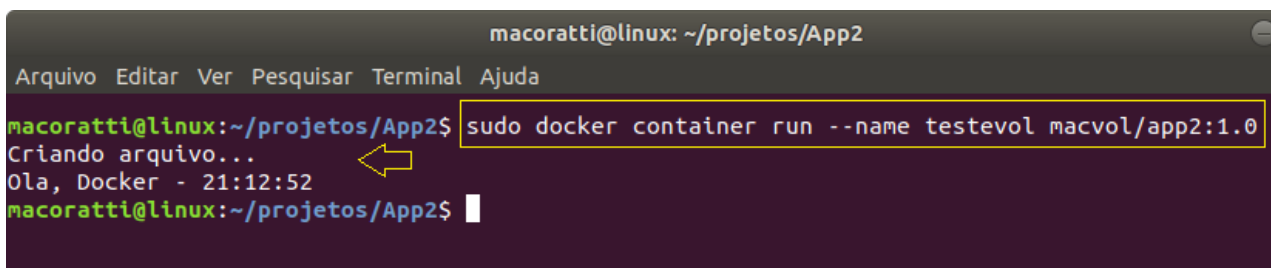
```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
macvol/app2          1.0                 e2422feb7770       8 minutes ago      4.82MB
aspnetcoremvc/app1  1.0                 a90c99ff854c       5 hours ago        257MB
macoratti/nginx     1.0                 1be6668aea57       25 hours ago       202MB
microsoft/dotnet    2.1-aspnetcore-runtime 1fe6774e5e9e       2 weeks ago        255MB
debian              8                   efdec82af25a       3 weeks ago        127MB
alpine              3.4                 174b26fe09c7       7 weeks ago        4.82MB
hello-world         latest              4ab4c602aa5e       2 months ago       1.84kB
macoratti@linux:~/projetos/App2$
```

Vemos a imagem **`macvol/app2:1.0`** criada com a **TAG** igual a **`1.0`**, o ID, data de criação e tamanho.

Agora vamos criar e executar um contêiner com base nesta imagem:

`docker container run --name testevol macvol/app2:1.0`

Aqui usamos o parâmetro **`--name`** para dar um novo ao contêiner criado.



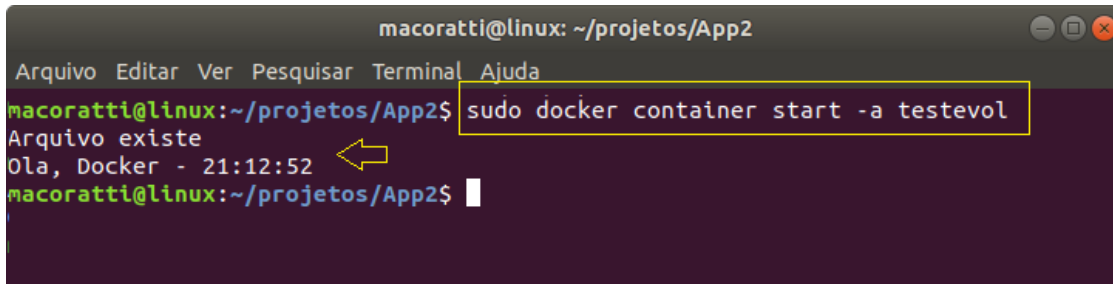
```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container run --name testevol macvol/app2:1.0
Criando arquivo...
Ola, Docker - 21:12:52
macoratti@linux:~/projetos/App2$
```

Vemos na figura acima a saída produzida pela execução do contêiner.

O contêiner é executado e sai depois de ter escrito a mensagem, o que mostra que os dados no arquivo `/data/msg.txt` foram criados e que foi definido um timestamp às `21:12:52`.

Como não configuramos um volume para o arquivo de dados, ele se tornou parte do sistema de arquivos do contêiner. O sistema de arquivos é persistente, e, podemos provar isso iniciando novamente o contêiner com o comando:

`docker container start -a testevol`



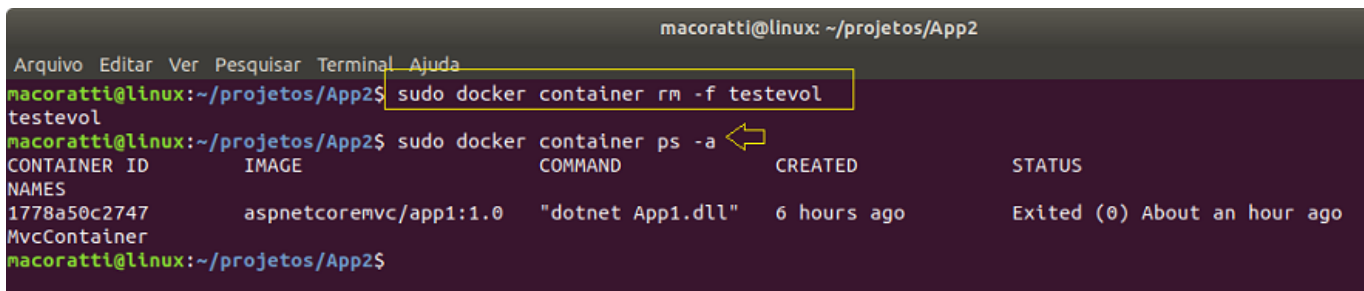
```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container start -a testevol
Arquivo existe
Ola, Docker - 21:12:52
macoratti@linux:~/projetos/App2$
```

A saída indica que o arquivo `/data/msg.txt` já existe e possui o mesmo registro de data e hora.

Tudo bem, esse é o comportamento esperado. Vamos agora excluir o contêiner para mostrar o que vai acontecer, digitando os comandos:

`docker container rm -f testevol`

`docker container ps -a`



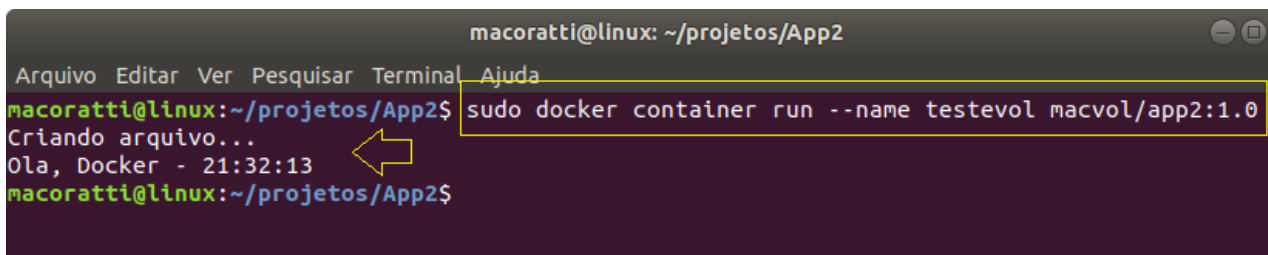
```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container rm -f testevol
testevol
macoratti@linux:~/projetos/App2$ sudo docker container ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS
1778a50c2747   aspnetcoremvc/app1:1.0 "dotnet App1.dll"        6 hours ago    Exited (0) About an hour ago
MvcContainer
macoratti@linux:~/projetos/App2$
```

O primeiro comando exclui o contêiner identificado por `testevol` e o segundo exibe todos os contêineres.

Como vemos não temos mais o contêiner que criamos pois o mesmo foi excluído. Assim o Docker excluiu o contêiner e o arquivo de dados `/data/msg.txt` foi perdido. Perdemos os dados...

Para confirmar isso vamos criar outro contêiner a partir da mesma imagem:

`docker container run --name testevol macvol/app2:1.0`



```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container run --name testevol macvol/app2:1.0
Criando arquivo...
Ola, Docker - 21:32:13
macoratti@linux:~/projetos/App2$
```

O resultado exibido na execução do contêiner mostra que o arquivo foi criado novamente com uma nova data e que os dados anteriores foram perdidos.

Como resolver esse problema ?

Na [próxima aula](#) vamos veremos como criar volumes para evitar a perda de dados.