



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



Neste artigo veremos como resolver a perda de dados quando um contêiner é excluído criando [volumes](#) no Docker. ([artigo anterior](#))



Os [volumes](#) do Docker resolvem o problema dos arquivos de dados mantendo-os fora do contêiner e tornando-os ainda acessíveis ao aplicativo que é executado dentro do contêiner. Vejamos então como podemos implementar [volumes](#).

No exemplo do artigo anterior vimos que ao excluir o contêiner os dados foram perdidos, pois faziam parte do sistema de arquivos do contêiner. O que vamos fazer é criar uma cópia dos dados que estão no contêiner para a nossa máquina, e, assim, caso o contêiner seja removido, podemos informar onde os dados estão, e, nossos dados serão preservados independente do que acontecer com o contêiner.

Retornando ao nosso exemplo usado no artigo anterior quando criamos o arquivo [Dockerfile](#) com o seguinte conteúdo:

```
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 abrir o arquivo [Dockerfile](#) e incluir o seguinte comando no arquivo:

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

O comando [VOLUME](#) informa ao Docker que quaisquer arquivos armazenados em [/data](#) devem ser armazenados em um [volume](#), colocando-os fora do sistema de arquivos dos contêineres regulares.

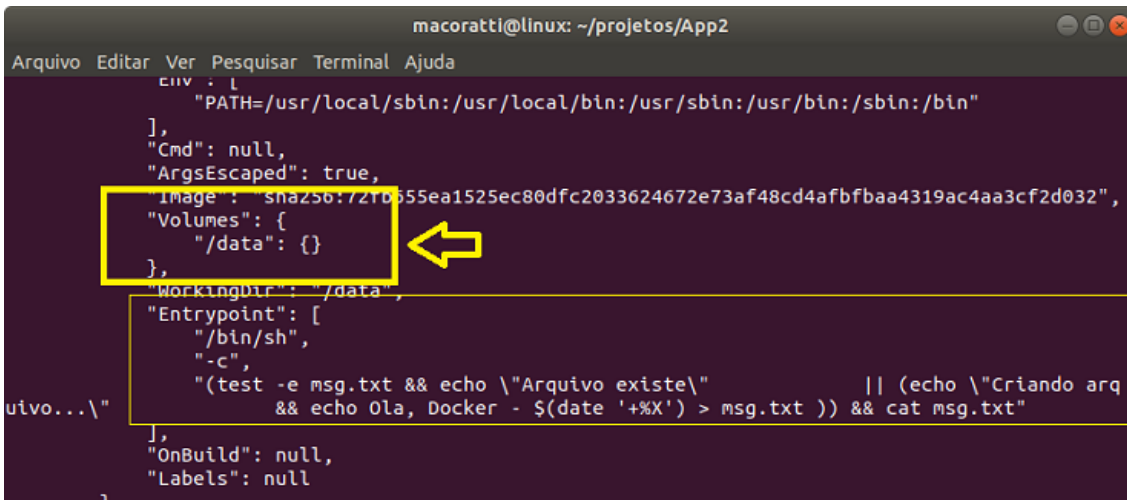
O ponto importante a observar é que o aplicativo em execução no contêiner não saberá que os arquivos no diretório [/data](#) são especiais: eles serão lidos e gravados apenas como qualquer outro arquivo no sistema de arquivos do contêiner.

Vamos agora recriar a [imagem](#) definida pelo arquivo [Dockerfile](#) digitando:

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

Após recriar a imagem podemos inspecioná-la para verificar se ela contém o volume criado digitando:

```
docker image inspect macvol/app2:1.0
```



```

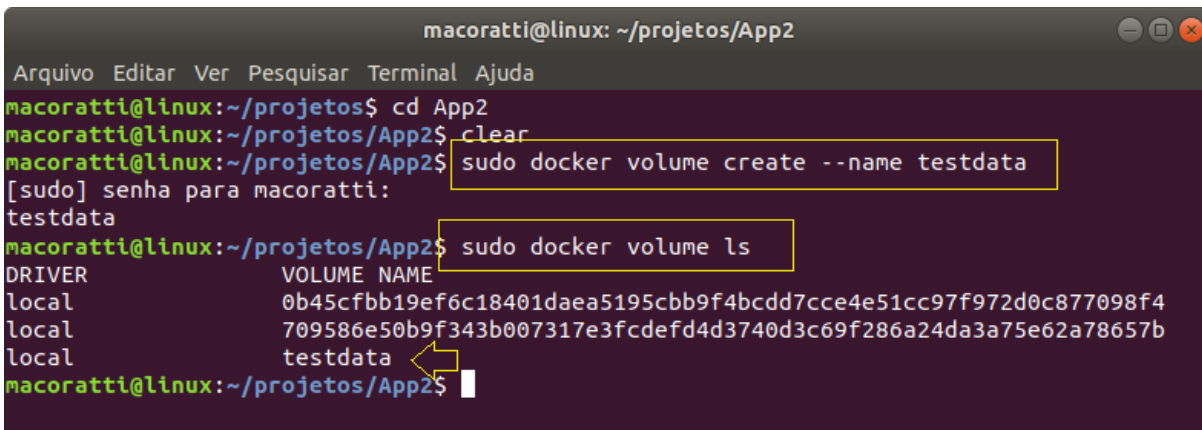
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
ENV : [
  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
],
"Cmd": null,
"ArgsEscaped": true,
"Image": "sha256:721b55ea1525ec80dfc2033624672e73af48cd4afbfbbaa4319ac4aa3cf2d032",
"Volumes": {
  "/data": {}
},
"Workingdir": "/data",
"Entrypoint": [
  "/bin/sh",
  "-c",
  "(test -e msg.txt && echo \"Arquivo existe\" || (echo \"Criando arquivo...\" && echo Ola, Docker - $(date +%X) > msg.txt )) && cat msg.txt"
],
"OnBuild": null,
"Labels": null

```

Vemos a informação `"Volumes" : { "/data" : {} }`, indicando a informação do volume.

A seguir vamos criar o **volume** no host que vai armazenar os arquivos de dados da aplicação usando o comando:

docker volume create --name testdata



```

macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos$ cd App2
macoratti@linux:~/projetos/App2$ clear
macoratti@linux:~/projetos/App2$ sudo docker volume create --name testdata
[sudo] senha para macoratti:
testdata
macoratti@linux:~/projetos/App2$ sudo docker volume ls
DRIVER          VOLUME NAME
local           0b45cfbb19ef6c18401daea5195cbb9f4bcdd7cce4e51cc97f972d0c877098f4
local           709586e50b9f343b007317e3fcdefd4d3740d3c69f286a24da3a75e62a78657b
local           testdata
macoratti@linux:~/projetos/App2$

```

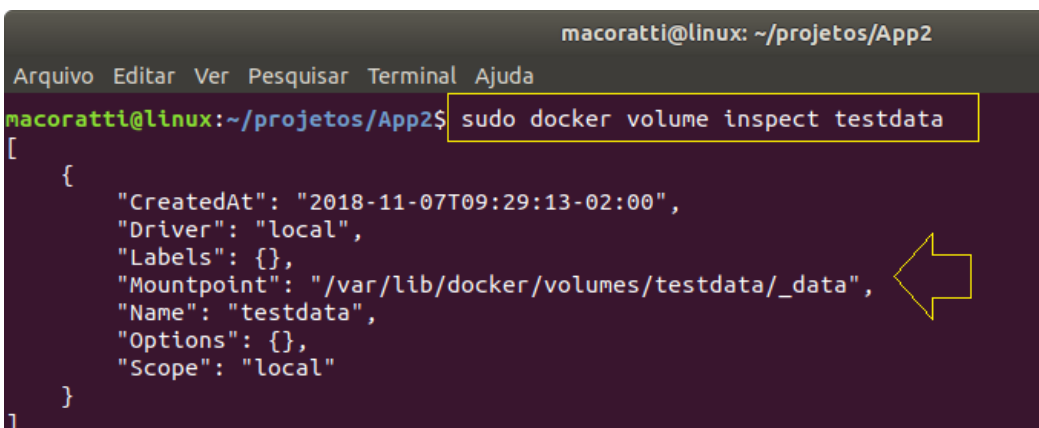
O comando **docker volume create** é usado para criar um novo volume e designar um nome, que para o nosso exemplo é **testdata**. (Com isso criamos um repositório de dados, ou seja, um volume para o contêiner)

O volume é um sistema de arquivos independente que fornecerá o conteúdo para um diretório no sistema de arquivos do contêiner. Como o volume não faz parte do contêiner, os arquivos nele contidos não serão excluídos se por ventura o contêiner for destruído.

Para exibir o volume criado usamos o comando : **docker volume ls**

Vemos o nome do volume e o seu driver, ou seja, qual é a forma que ele deve montar o volume. Para este exemplo usamos o *driver local*, o *driver padrão (built-in) do Docker*.

Para inspecionar o volume podemos usar o comando: **docker volume inspect testdata**



```

macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker volume inspect testdata
[
  {
    "CreatedAt": "2018-11-07T09:29:13-02:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/testdata/_data",
    "Name": "testdata",
    "Options": {},
    "Scope": "local"
  }
]

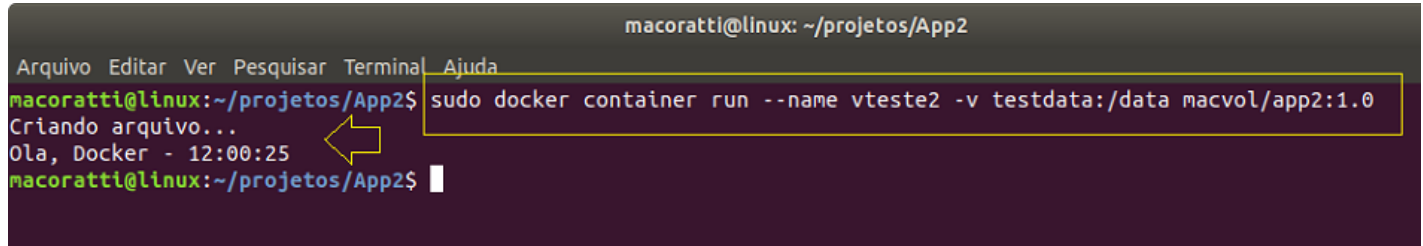
```

Pronto temos o volume criado com sucesso.

Para concluir vamos informar ao Docker qual o volume deve ser usado pelo contêiner executando o comando abaixo na pasta **App2** para criar um contêiner que usa o volume **testdata** que vai fornecer o conteúdo do diretório **/data** que configuramos :

docker container run --name vteste2 -v testdata:/data macvol/app2:1.0

O argumento **-v** informa ao Docker que quaisquer dados que o contêiner criar no diretório **/data** devem ser armazenados no volume **testdata**. O volume aparece como um diretório regular para a aplicação que cria o arquivo de dados como antes.



```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container run --name vteste2 -v testdata:/data macvol/app2:1.0
Criando arquivo...
Ola, Docker - 12:00:25
macoratti@linux:~/projetos/App2$
```

O resultado exibido na execução do contêiner mostra que o arquivo foi criado e com uma data. Como os volumes estão vazios quando são criados a aplicação não encontrou nenhum arquivo e assim criou o arquivo conforme mostrado acima.

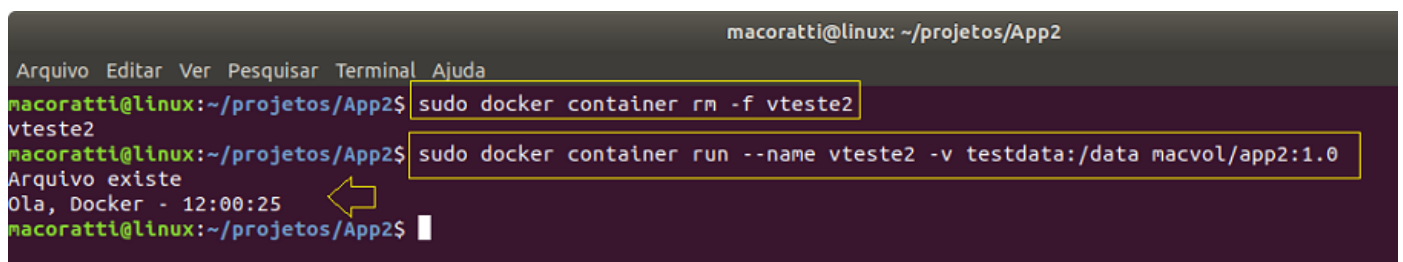
Então vamos fazer a prova dos nove. Vamos destruir este contêiner criado digitando o comando no terminal:

docker container rm -f vteste2

Ao executar este comando o contêiner **vteste2** será excluído.

Vamos agora recriar o contêiner com o comando:

docker container run --name vteste2 -v testdata:/data macvol/app2:1.0



```
macoratti@linux: ~/projetos/App2
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App2$ sudo docker container rm -f vteste2
vteste2
macoratti@linux:~/projetos/App2$ sudo docker container run --name vteste2 -v testdata:/data macvol/app2:1.0
Arquivo existe
Ola, Docker - 12:00:25
macoratti@linux:~/projetos/App2$
```

Percebemos pelo resultado da execução do contêiner que o arquivo ainda existe no volume mesmo após o contêiner ser removido.

Desta vez, quando o script **ENTRYPOINT** procurar pelo arquivo **/data/msg.txt**, vai descobrir o arquivo criado pelo contêiner anterior, que sobreviveu porque o volume não foi afetado quando o contêiner foi destruído.

Pronto, resolvemos a perda de dados do contêiner criando um volume.

Na [próxima aula vamos](#) incluir um banco de dados na aplicação ASP .NET Core MVC.