



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



Hoje vamos incluir um banco de dados na aplicação ASP .NET Core MVC, mas antes vamos criar uma imagem, definir um volume e criar um contêiner para o MySQL. ([artigo anterior](#)).



No artigo - [Docker uma introdução básica - VII](#) - criamos uma imagem para uma aplicação ASP .NET Core MVC que foi criada neste artigo : [Criando uma aplicação Web no Linux](#)

A aplicação web que criamos usava um [repositório com dados estáticos](#) e não usava um banco de dados para persistência.

Vamos agora incluir um banco de dados para ser usado pela nossa aplicação ASP .NET Core MVC, e, isso significa que vão existir arquivos de banco de dados que deverão ser armazenados de forma que eles não serão deletados quando um contêiner da nossa aplicação for destruído. Para isso vamos podemos criar um volume.

💡 Não fique tentado a criar uma imagem Docker que contenha seu aplicativo Asp.Net Core MVC e o banco de dados para que eles possam ser executados em um único contêiner. A convenção para o Docker é usar um contêiner separado para cada componente em um aplicativo, o que facilita a atualização ou a substituição de partes do aplicativo e permite uma abordagem mais flexível para expandir o aplicativo depois de implantado. Você não se beneficiará dos recursos mais úteis do Docker se você criar um contêiner monolítico que inclua todos os seus componentes de aplicativo.

No entanto, como o conteúdo de um volume não é incluído nas imagens, mesmo quando o comando `docker commit` é usado, algumas medidas especiais são necessárias para garantir que o esquema do modelo de dados da aplicação seja criado e que qualquer dado inicial seja aplicado quando o banco de dados for iniciado pela primeira vez.

Vamos então adicionar um banco de dados à nossa aplicação ASP .NET Core MVC, criar um volume e a seguir usar o volume para conter os arquivos de dados.

A primeira coisa a fazer é decidir qual banco de dados usar, e, no caso de aplicações Web em ambiente Linux o mais recomendado é usar o [MySQL](#) que também tem o suporte para o [Entity Framework Core](#).

Antes de iniciar vamos remover todos os contêineres criados em nosso ambiente digitando o comando:

```
docker rm -f $(docker ps -aq)
```

```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker container rm -f $(sudo docker ps -aq)
[sudo] senha para macoratti:
9228b2144c7c
8ad0c24e9c1f
1778a50c2747
macoratti@linux:~$
```

## Baixando e inspecionando a imagem do MySQL

Vamos baixar uma [imagem base](#) do [MySQL](#) a partir do repositório, e, quando vamos adicionar um novo componente a um aplicativo em um contêiner, é importante começar inspecionando a imagem para saber como ela sua [volumes](#) de forma a poder configurar os contêineres criados.

Primeiro vamos baixar a imagem do MySQL usando o comando:

```
docker pull mysql:8.0.0
```

```
macoratti@linux: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
macoratti@linux:~$ sudo docker pull mysql:8.0.0  
8.0.0: Pulling from library/mysql  
6d827a3ef358: Pull complete  
ed0929eb7dfe: Pull complete  
03f348dc3b9d: Pull complete  
fd337761ca76: Pull complete  
7e6cc16d464a: Pull complete  
ca3d380bc018: Pull complete  
809061e6dec3: Pull complete  
b3b4e9bf814c: Pull complete  
569cd556d913: Pull complete  
7d0518604156: Pull complete  
cc158c0bcedf: Pull complete  
Digest: sha256:449cb894b00a490e830bb9b2d774bf99c5af75e8cb10bfbe74db1c43eb12b211  
Status: Downloaded newer image for mysql:8.0.0  
macoratti@linux:~$
```

Conferindo a imagem baixada localmente: [docker images](#)

```
macoratti@linux: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
macoratti@linux:~$ sudo docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
macvol/app2         1.0                85bed7379968       2 hours ago        4.82MB  
aspnetcoremvc/app1 1.0                a90c99ff854c       22 hours ago       257MB  
macoratti/nginx     1.0                1be6668aea57       41 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       8 weeks ago        4.82MB  
hello-world         latest             4ab4c602aa5e       2 months ago       1.84kB  
mysql                8.0.0             228d71078f8c       19 months ago      433MB  
macoratti@linux:~$
```

A seguir vamos inspecionar essa imagem baixada para saber como ela trata volumes:

[docker inspect mysql:8.0.0](#)

```
macoratti@linux: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
[...]  
  "Cmd": [  
    "mysqld"  
  ],  
  "ArgsEscaped": true,  
  "Image": "sha256:02e78cc7aaa1eb2a5d8097bddc7faf23ea8985f22c38c4",  
  "Volumes": {  
    "/var/lib/mysql": {}  
  },  
  "WorkingDir": "",  
  "Entrypoint": [  
    "docker-entrypoint.sh"  
  ],  
  "OnBuild": [],  
  "Labels": {}  
}
```

Examinando a saída do comando vemos a seção **Volumes** que indica que esta imagem usa o volume para o diretório **/var/lib/mysql** que é o local onde o MySQL armazena os seus arquivos de dados.

## Criando o Volume e o Contêiner

Para preparar o contêiner do banco de dados, vamos primeiro criar o volume usando o comando:

[docker volume create --name produtosdata](#)

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker volume create --name produtosdata
produtosdata
macoratti@linux:~$ sudo docker volume ls
DRIVER          VOLUME NAME
local           0b45cfbb19ef6c18401daea5195cbb9f4bcdd7cce4e51cc97f972d0c877098f4
local           709586e50b9f343b007317e3fcdefd4d3740d3c69f286a24da3a75e62a78657b
local           produtosdata ←
local           testdata
macoratti@linux:~$

```

Estamos criando um volume chamado **produtosdata** que será usado para armazenar os arquivos de dados do banco de dados.

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.

Vamos agora criar e iniciar um novo contêiner **MySQL** que usa este volume para fornecer ao contêiner o conteúdo do diretório **/var/lib/mysql** :

**docker container run -d --name mysql -v produtosdata:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=numsey -e bind-address=0.0.0.0 mysql:8.0.0**

**Nota:** Este comando deve ser informado em uma única linha

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker container run -d --name mysql -v produtosdata:/var/lib/mysql -e MYSQL_ROOT
PASSWORD=numsey -e bind-address=0.0.0 mysql:8.0.0
8a05e496d3984bb9c8fdb0457311454c9db62412cce3a97a3e52752da4344f4b
macoratti@linux:~$

```

Verificando o contêiner cujo processo esta em execução : **docker container ps**

```

macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker container ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
8a05e496d398   mysql:8.0.0  "docker-entrypoint.s..."  2 minutes ago Up 2 minutes   3306/tcp     mysql
macoratti@linux:~$

```

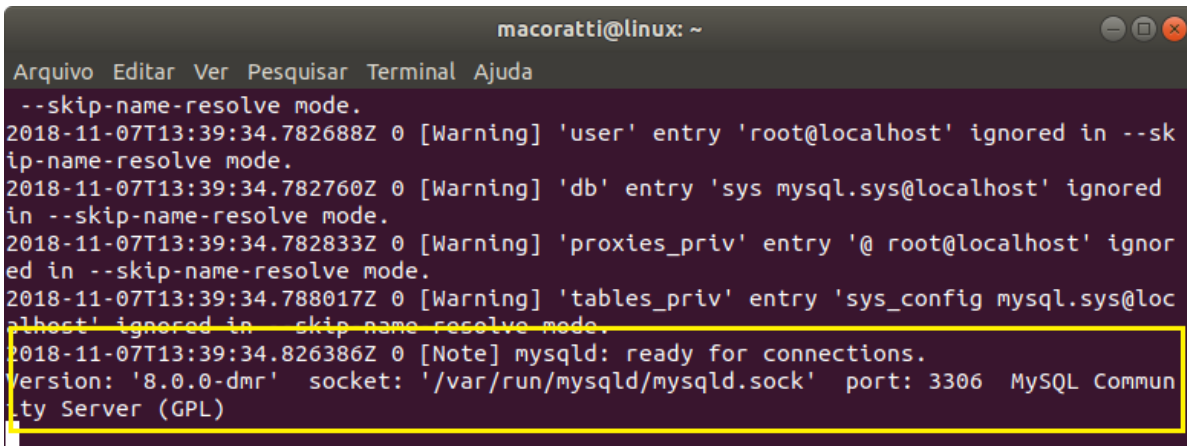
Vamos entender melhor o comando usado:

**docker container run** - Este comando crie e inicia um contêiner a partir de uma imagem

<b>-d</b>	Esse argumento informa ao Docker para executar o contêiner em segundo plano
<b>--name mysql</b>	Este argumento é usado para atribuir o nome <b>mysql</b> ao container
<b>-e MYSQL_ROOT_PASSWORD</b>	Este argumento define uma variável de ambiente. Neste caso, o contêiner MySQL usa a variável de ambiente <b>MYSQL_ROOT_PASSWORD</b> para definir a senha necessária para se conectar ao banco de dados.
<b>-e bind-address</b>	Define uma variável de ambiente que assegura que o MySQL aceita requisições de todas as interfaces de rede
<b>-v produtosdata:/var/lib/mysql</b>	Este argumento diz ao Docker para usar um volume chamado <b>produtosdata</b> para fornecer o conteúdo do diretório <b>/var/lib/mysql</b> do contêiner.

Agora vamos examinar a informação gerada pela execução do contêiner do banco de dados na inicialização usando o comando :

**docker logs -f mysql**



```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
--skip-name-resolve mode.
2018-11-07T13:39:34.782688Z 0 [Warning] 'user' entry 'root@localhost' ignored in --skip-name-resolve mode.
2018-11-07T13:39:34.782760Z 0 [Warning] 'db' entry 'sys mysql.sys@localhost' ignored in --skip-name-resolve mode.
2018-11-07T13:39:34.782833Z 0 [Warning] 'proxies_priv' entry '@ root@localhost' ignored in --skip-name-resolve mode.
2018-11-07T13:39:34.788017Z 0 [Warning] 'tables_priv' entry 'sys_config mysql.sys@localhost' ignored in --skip-name-resolve mode.
2018-11-07T13:39:34.826386Z 0 [Note] mysqld: ready for connections.
Version: '8.0.0-dmr' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server (GPL)
```

Durante a inicialização do MySQL serão gravadas mensagens de log, dentre elas a mais importante é que informa que o banco de dados esta pronto para aceitar conexões de rede como mostrado na imagem acima : **ready for connections**.

As inicializações subsequentes serão muito mais rápidas porque poderão usar os arquivos que foram criados no volume **produtosdata**. Quando o banco de dados estiver em execução, digite **Control+C** para parar de monitorar saída e deixar o banco de dados em execução em seu contêiner em segundo plano.

Dessa forma já temos uma imagem do MySQL e um contêiner com um volume preparado.

Na [próxima aula](#) vamos ajustar a aplicação ASP .NET Core MVC para usar o MySQL.