



Mexendo com arquivos/pastas e volumes

Vimos que é possível subirmos um container à partir de uma imagem já pronta com o que precisamos, mas agora vem outra pergunta: e se além da aplicação que estamos subindo com o Docker, também houver a necessidade de incluirmos arquivos nossos específicos para serem utilizados pela imagem, sem precisar criar uma imagem nova para cada arquivo que atualizemos?

A resposta é: volumes! Um volume é literalmente uma pasta onde colocamos estes arquivos e dizemos ao container para usar essa pasta como local onde a aplicação contida no container precisará ler estes arquivos.

Tá complicado? Vamos descomplicar com um exemplo bem simples!

Vamos subir um container novo, partindo de uma imagem do nginx, que é um servidor web bastante enxuto. Certifique-se de que o Docker esteja ativo e veja o comando que podemos executar para subir um container novo:

```
docker run --name NginxSemVolume -p 9080:80 -d nginx:latest
```

```
C:\Windows\System32\cmd.exe
C:\Users\event\Documents\Gitlab\descomplica\docker\02-Volumes>docker run --name NginxSemVolume -p 9080:80 -d nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
42c077c10790: Already exists
62c70f376f6a: Pull complete
915cc9bd79c2: Pull complete
75a963e94de0: Pull complete
7b1fab684d70: Pull complete
db24d06d5af4: Pull complete
Digest: sha256:2bcabc23b45489fb0885d69a06ba1d648aeda973fae7bb981bafbb884165e514
Status: Downloaded newer image for nginx:latest
cd385f195482a0bb11c00a3f0913d882e34a844d3169cb8a83313f0294c30ef9
C:\Users\event\Documents\Gitlab\descomplica\docker\02-Volumes>
```

Vamos testar nosso container abrindo ele no Google Chrome, só digitar
localhost:9080~



Essa é a página padrão de uma imagem do Nginx, mas e se quiséssemos usar uma página nossa, customizada?

Eu criei um arquivo super simples chamado index.html com o conteúdo abaixo:

```
<html>
  <body>
    <h1>Olá mundo! Este é um teste com Nginx usando volume para customizar
uma página!</h1>
  </body>
</html>
```

Eu coloquei este arquivo em um lugar muito especial em meu computador, ele está localizado aqui: D:\temp\docker.

Agora eu vou criar mais um container, desta vez indicando um volume para uso desta pasta D:\temp\docker que possui meu index.html que será usado para ser exibido no Google Chrome ao invés daquela página padrão. O comando que usei foi este aqui:

```
docker run --name NginxComVolume -v /D/temp/docker:/usr/share/nginx/html -p
9180:80 -d nginx:latest
```

```
C:\Windows\System32\cmd.exe
C:\Users\event\Documents\Gitlab\descomplica\docker\02-Volumes>docker run --name NginxComVolume -v /D/temp/docker:/usr/share/nginx/html -p 9180:80 -d nginx:latest
683ecef233328856ee9c21d8e8226ee4d16268df63ebb547c1bed5d45df8c47c
C:\Users\event\Documents\Gitlab\descomplica\docker\02-Volumes>
```

Aqui tem algumas diferenças:

“--name NginxComVolume”: é o nome desse meu container novo, para diferenciar do outro container que já havíamos criado antes.

“-v /D/temp/docker:/usr/share/nginx/html”: é aqui que indicamos que vai ter um volume para o container. Esse comando faz a “montagem” da pasta D:\temp\docker (do Windows) para a pasta interna do container chamado /usr/share/nginx/html. O Nginx internamente usa este local (/usr/share/nginx/html) para exibir as páginas no servidor web, o que estou fazendo é indicar que ao ler este local o que está efetivamente acontecendo é que o container olha para o conteúdo do meu computador local, em D:\temp\docker!

“-p 9180:80”: eu diferencio a porta pois a porta 9080 já está sendo usada pelo outro container, se eu não diferenciar, ao digitar no Google Chrome localhost:9080, não teria como saber qual dos 2 containers é para o Google Chrome chamar.

Essa forma é o que chamamos de “bind mount”, nós montamos o volume do container apontando para uma pasta do “host”. O “host” é nosso computador local, fora do container.

Vamos olhar o resultado chamando localhost:9180 no Google Chrome:



E se eu quiser mudar o conteúdo do arquivo index.html, o que preciso fazer?

A resposta é: mude o arquivo index.html direto! Não precisa mexer em nada no container para este exemplo!. Vou alterar o arquivo index.html diretamente na pasta D:\temp\docker, vai ficar assim:

```
<html>
  <body>
    <h1>Olá mundo! Mudei aqui sem recriar o container!!!</h1>
  </body>
</html>
```

Não precisamos nem dar stop e start no container, vou lá no Google Chrome e apertar a tecla F5, que faz a atualização (refresh) da página, olha o resultado:



Há uma outra forma de criar um container com um volume, você pode usar o parâmetro “--mount”, o resultado é o mesmo mas este parâmetro é o recomendável (pelo site da própria Docker):

```
docker          run          --name          NginxComVolume          --mount
type=bind,src=/D/temp/docker,dst=/usr/share/nginx/html -p 9180:80 -d nginx:latest
```

A diferença do “--mount” é que você indica o parâmetro tudo junto separado por vírgula, sendo:

- type=bind: é o tipo de montagem, bind é para você referenciar um local do “host” à um local do container
- src=/D/temp/docker: é o local do “host” que você está referenciando

- `dst=/usr/share/nginx/html:` é o local do container que será referenciado em relação ao `src`

Agora vamos um pouco além! Vamos ver a importância de volumes sob um outro aspecto. Até agora brincamos com volume alterando um arquivo direto no “host” em `D:\Temp\Docke\index.html`, mas e se tivermos um banco de dados onde as informações são alteradas de dentro do container?

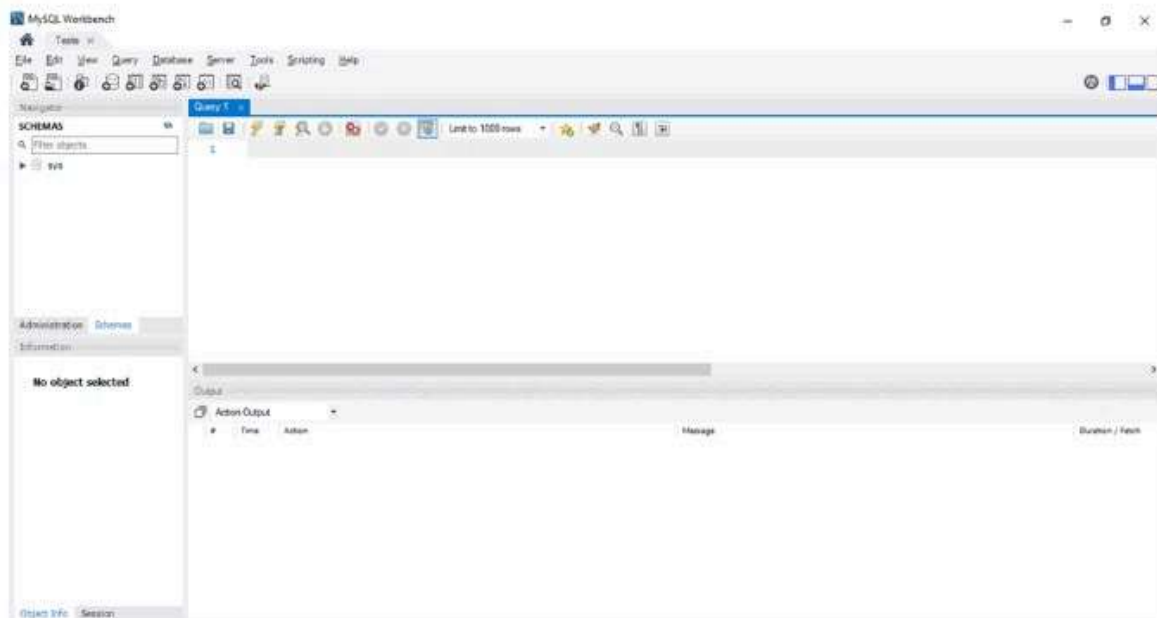
Vamos fazer uma experiência com um banco de dados MySQL sem volume nenhum, iremos então criar um schema, uma tabela e inserir dados, depois vamos destruir o container e recriá-la para ver o que acontece, me acompanhem:

Vamos criar um container com o MySQL, através desse comando:

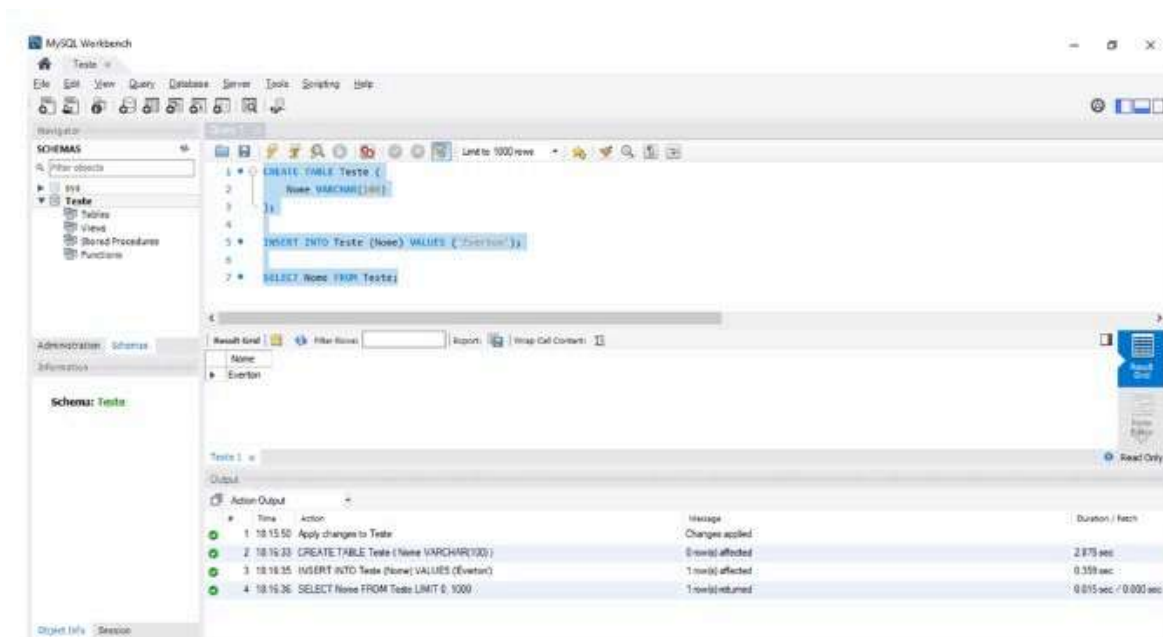
```
docker run --name MySQLsemVolume -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=MinhaSenha -e MYSQL_USER=MeuUsuario -e
MYSQL_PASSWORD=MinhaSenha -d mysql:latest
```

Neste momento eu vou pular toda a explicação dos parâmetros do MySQL pois veremos isso mais pra frente, então apenas vejamos o que acontecerá quando eu abrir o MySQL Workbench, conectar e criar o schema, tabela e dados:

O MySQL que acabamos de subir só tem o schema `sys`:



Criei um schema chamado “Teste”, uma tabela também chamada “Teste” dentro do schema e inseri 1 linha, depois consultei essa linha:



Muito bem, agora o que acontece se eu destruir o container e recriá-lo? Os dados ainda estarão lá uma vez que eu inseri conectando no container pelo MySQL Workbench? Vamos testar:

Vou primeiro parar o container e removê-lo:

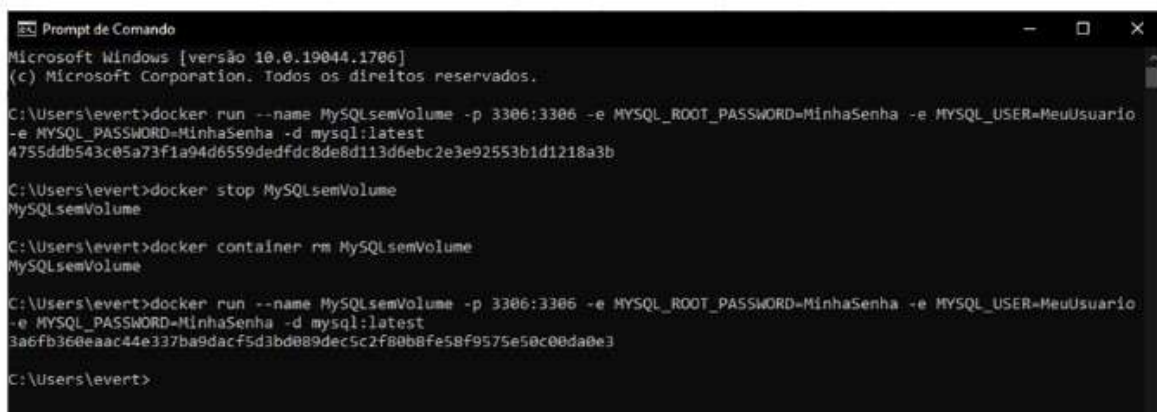
`docker stop MySQLsemVolume`

Depois vou excluir o container:

`docker container rm MySQLsemVolume`

E depois vou simplesmente recriá-lo, com o mesmo comando de antes:

```
docker run --name MySQLsemVolume -p 3306:3306 -e MYSQL_ROOT_PASSWORD=MinhaSenha -e MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
```



```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.1706]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\event>docker run --name MySQLsemVolume -p 3306:3306 -e MYSQL_ROOT_PASSWORD=MinhaSenha -e MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
4755ddb543c05a73f1a94d6559dedfdc8de8d113d0ebc2e3e92553b1d1218a3b

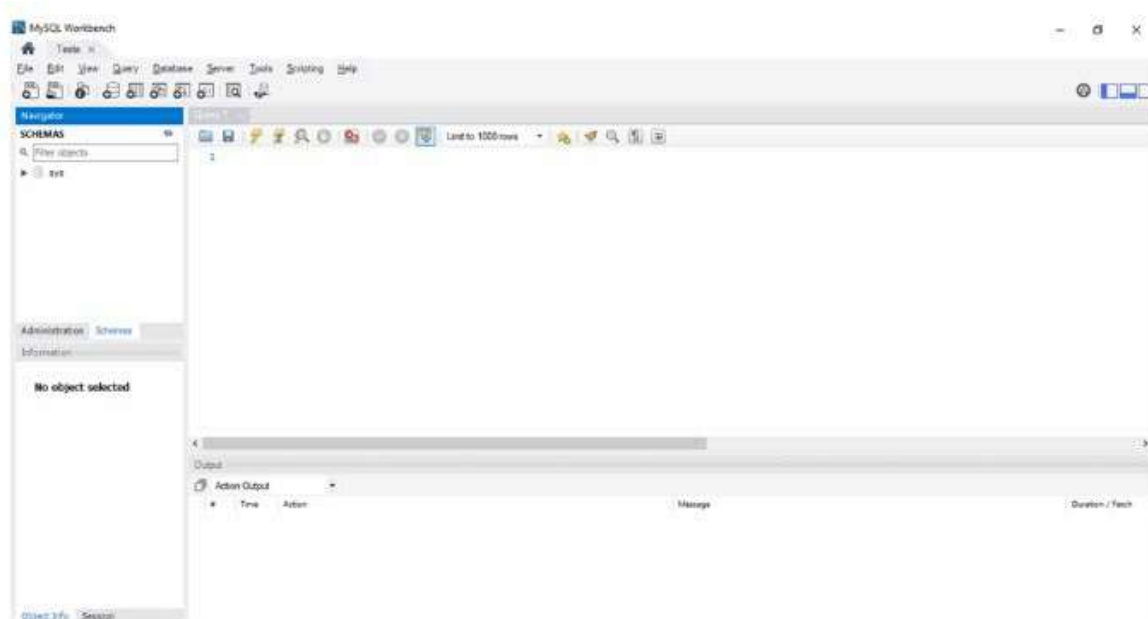
C:\Users\event>docker stop MySQLsemVolume
MySQLsemVolume

C:\Users\event>docker container rm MySQLsemVolume
MySQLsemVolume

C:\Users\event>docker run --name MySQLsemVolume -p 3306:3306 -e MYSQL_ROOT_PASSWORD=MinhaSenha -e MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
3a6fb360eaaac44e337ba9dacf5d3bd089dec5c2f80b8fe58f9575e50c00da0e3

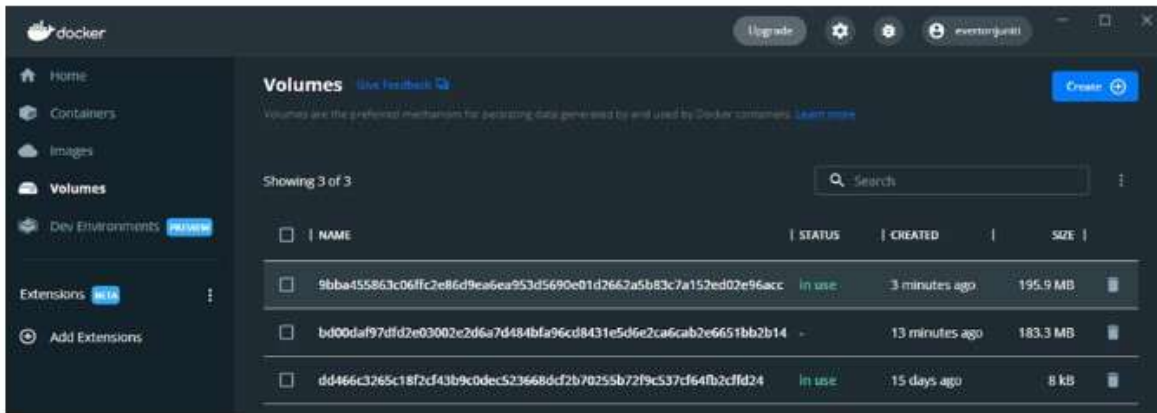
C:\Users\event>
```

Vou abrir o MySQL Workbench novamente:



O que aconteceu com os dados?

Vamos ver uma coisa interessante no Docker Desktop, na guia “Volumes”:



A explicação é simples: se eu não indico um volume para onde o MySQL possa guardar os dados, a cada vez que eu crio um container da imagem MySQL é criado automaticamente um volume novo, logo os dados estarão “zerados”!

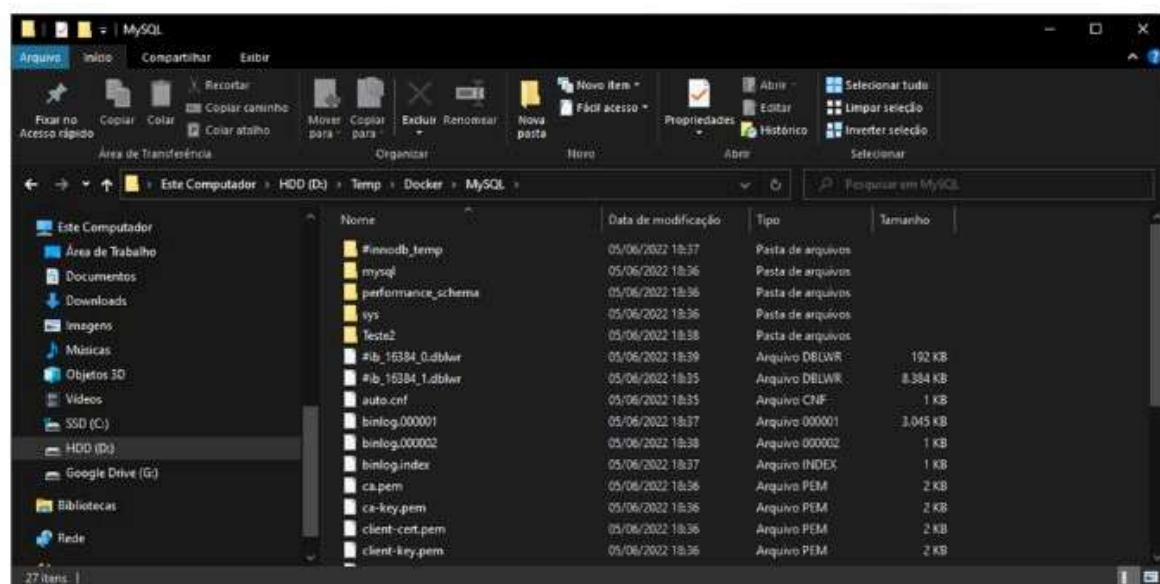
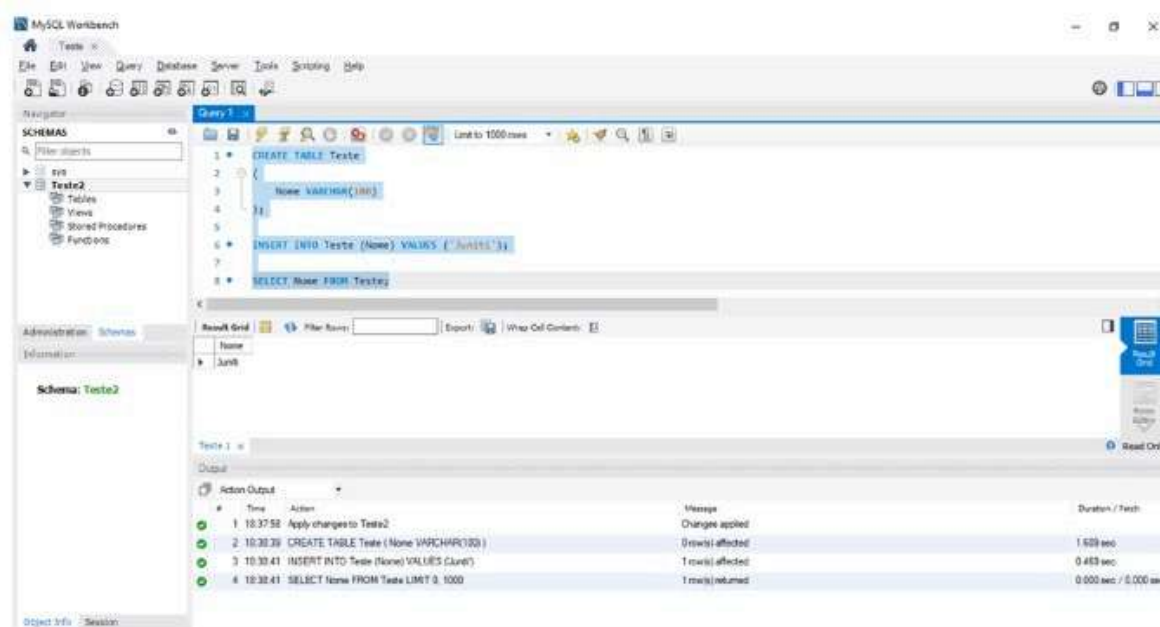
Note as datas dos volumes, um foi criado há 13 minutos e outro há 3. Eu propositalmente oculte o tempo de inicialização do MySQL para que vocês não soubessem que o MySQL estava inicializando tudo do zero nas 2 vezes em que o container foi criado.

Agora vamos criar um outro container, desta vez indicando um volume para onde os dados serão salvos, vou fazer através do seguinte comando:

```
docker run --name MySQLcomVolume -v /D/Temp/Docker/MySQL:/var/lib/mysql  
-p 3306:3306 -e MYSQL_ROOT_PASSWORD=MinhaSenha -e  
MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
```

Um detalhe muito importante é que eu criei uma pasta chamada D:\Temp\Docker\MySQL antes de executar este comando!

Desta vez temos 2 coisas interessantes para olhar:



Bom, criei um schema diferente chamado Teste2, note que a pasta onde eu indiquei o volume (D:\Temp\Docker\MySQL) criou vários arquivos, isso foi a inicialização do MySQL quem criou. Já o schema chamado “Teste2” foi criada uma pasta só para ele.

Agora eu vou efetuar o mesmo procedimento: parar este container, excluí-lo e recriá-lo, só que haverá uma pequena diferença na recriação do container, observem:

Vou primeiro parar o container e removê-lo:

`docker stop MySQLcomVolume`

Depois vou excluir o container:

`docker container rm MySQLcomVolume`

E depois vou simplesmente recriá-lo, com o mesmo comando de antes:

```
docker run --name MySQLcomVolume -v /D/Temp/Docker/MySQL:/var/lib/mysql  
-p 3306:3306 -d mysql:latest
```

Note que o comando diminuiu! O comando original tinha a senha do usuário root do MySQL, um usuário customizado e a senha para este usuário. Na inicialização do MySQL foram criados e configurados esses usuários e as informações guardadas na pasta do host (D:\Temp\Docker\MySQL). Como estamos subindo um outro container só que apontando para o mesmo volume, não precisamos passar estas informações que já foram configuradas! O MySQL irá ler o conteúdo da pasta já existente e simplesmente carregará as configurações prontas! A subida é até mais rápido do que a primeira vez:



```
Prompt de Comando
Microsoft Windows [versão 10.0.19044.1706]
(c) Microsoft Corporation. Todos os direitos reservados.

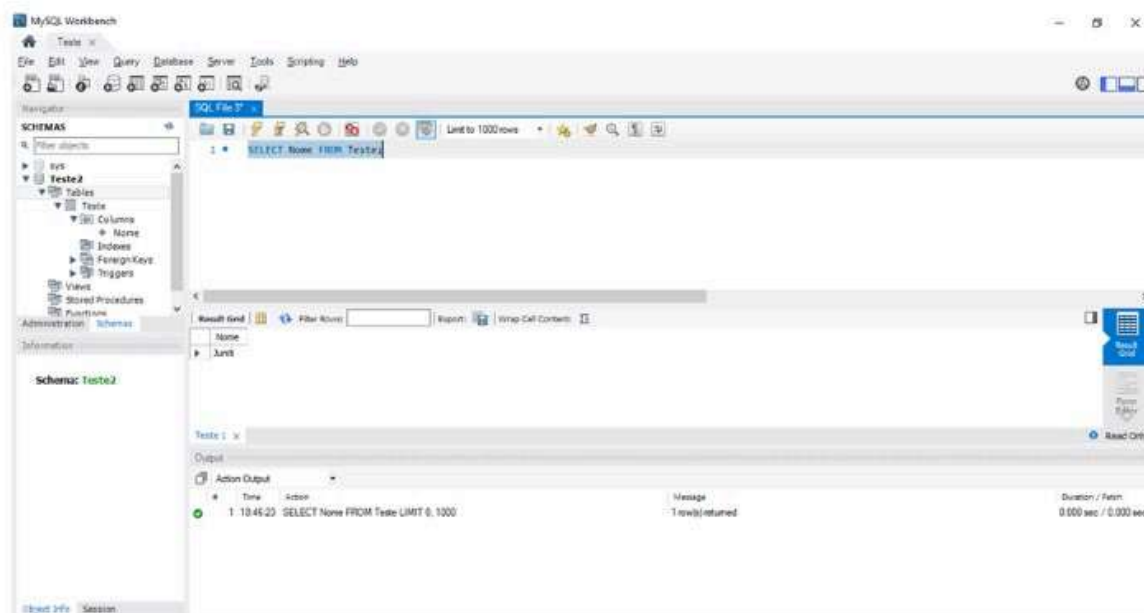
C:\Users\event>docker run --name MySQLcomVolume -v /D/Temp/Docker/MySQL:/var/lib/mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=MinhaSenha -e MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
f41ba321afa4a47b8a70c1a8c49beee863091f2495df0282fcd0db73cbdc1054

C:\Users\event>docker stop MySQLcomVolume
MySQLcomVolume

C:\Users\event>docker container rm MySQLcomVolume
MySQLcomVolume

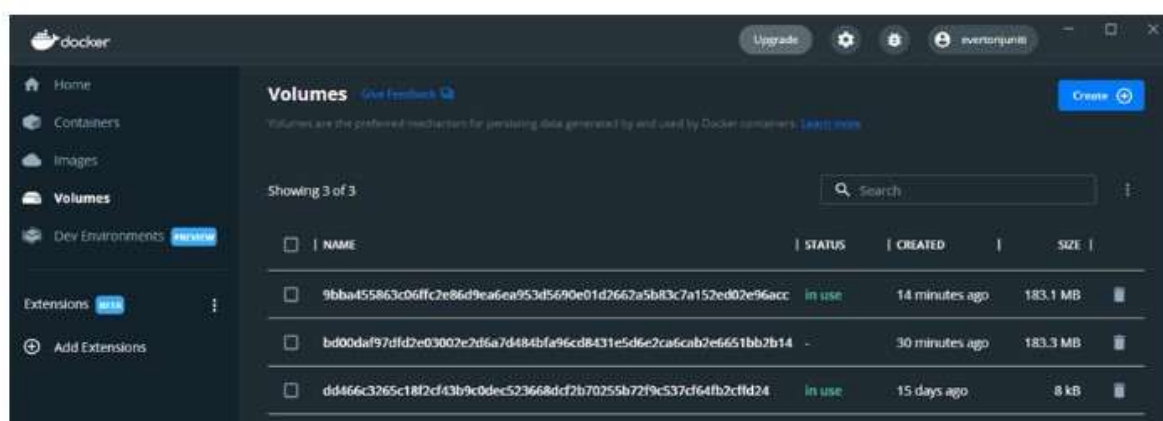
C:\Users\event>docker run --name MySQLcomVolume -v /D/Temp/Docker/MySQL:/var/lib/mysql -p 3306:3306 -d mysql:latest
0bf5c2a4ac8da611d90558513be574625a8c203def9f3e9cac873879d7dc18f4

C:\Users\event>
```



Note que os dados estão lá, não perdemos nada, mesmo que haja a necessidade de destruir e recriar o container os dados continuarão lá!

Note também no Docker Desktop na guia Volumes que nenhum volume adicional interno foi criado, isso porque fizemos o que chamamos de “bind mount” apontando para uma pasta do “host” (D:\Temp\Docker\MySQL), então não foi preciso criar um volume interno (que fica em uma pasta gerenciada pelo Docker):



Vimos a enorme vantagem de uso de volumes, principalmente para containers que guardam dados e que permitem alteração desses dados!

É claro que tudo depende da necessidade, em certos casos não há necessidade nenhuma de criar um container com volumes, sempre verifique a necessidade!

Atividade Extra

Para se aprofundar no assunto desta aula leia o capítulo 7 da bibliografia de referência: VITALINO, J. F. N.; CASTRO, M. A. N. Descomplicando o docker. 2.ed. Brasport: 2018

Como bibliografia complementar com maior detalhamento sobre volumes, leia o capítulo 13 da seguinte referência: POULTON, Nigel. Docker Deep Dive: Zero to Docker in a single book (English Edition). May 2020 ed. Nigel Poulton: 2020;

Referência Bibliográfica

- VITALINO, J. F. N.; CASTRO, M. A. N. Descomplicando o docker. 2.ed. Brasport: 2018.

Ir para exercício