



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



No [artigo anterior](#), vimos como [combinar contêineres](#), volumes e redes em uma aplicação mais complexa. O problema com a abordagem usada é que cada passo foi executado manualmente, o que pode ser um processo sujeito a erros.

Não apenas cada comando deve ser inserido corretamente, mas as etapas devem ser executadas na ordem correta, porque um contêiner tem que estar habilitado para enviar requisições para outro contêiner que tem que ser criado antes dele. Se você pular uma etapa ou executar uma etapa fora de ordem, então sua aplicação vai falhar.



É justamente para evitar esses problemas que serve o [Docker Compose](#), o orquestrador de contêineres do Docker.

Neste artigo veremos como usar o [Docker Compose](#) que é uma ferramenta usada para descrever aplicações complexas e gerenciar os contêineres, as redes e os volumes que essas aplicações exigem para funcionar. Ele simplifica o processo de configuração e execução de aplicativos para que você não precise digitar comandos complexos, o que pode levar a erros de configuração.

O [Docker Compose](#) é usado para descrever aplicações de forma consistente e previsível usando um arquivo de composição que contém detalhes de todos os volumes, redes e contêineres que compõem um aplicativo e os relacionamentos entre eles. Para processar o arquivo de composição usamos o comando : **docker-compose**

Você não é obrigado a usar o [Docker Compose](#), podendo gerenciar seus contêineres manualmente ou usar outra alternativa como o [Crowdr](https://github.com/polonskiy/crowdr) (<https://github.com/polonskiy/crowdr>).

A primeira coisa a fazer é instalar o Docker Compose em nosso ambiente.

Instalando o Docker Compose no Linux

As instruções de instalação estão disponíveis neste link: <https://docs.docker.com/compose/install/>

Para instalar o [Docker Compose](#) digite o comando abaixo e depois aplique as permissões aos binários:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
macoratti@linux: ~  
Arquivo Editar Ver Pesquisar Terminal Ajuda  
macoratti@linux:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker  
-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
[sudo] senha para macoratti:  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 617 0 617 0 0 552 0 --:--:-- 0:00:01 --:--:-- 552  
100 11.1M 100 11.1M 0 0 2347k 0 0:00:04 0:00:04 --:--:-- 3778k  
macoratti@linux:~$ sudo chmod +x /usr/local/bin/docker-compose  
macoratti@linux:~$ docker-compose --version  
docker-compose version 1.23.1, build b02f1306  
macoratti@linux:~$ clear
```

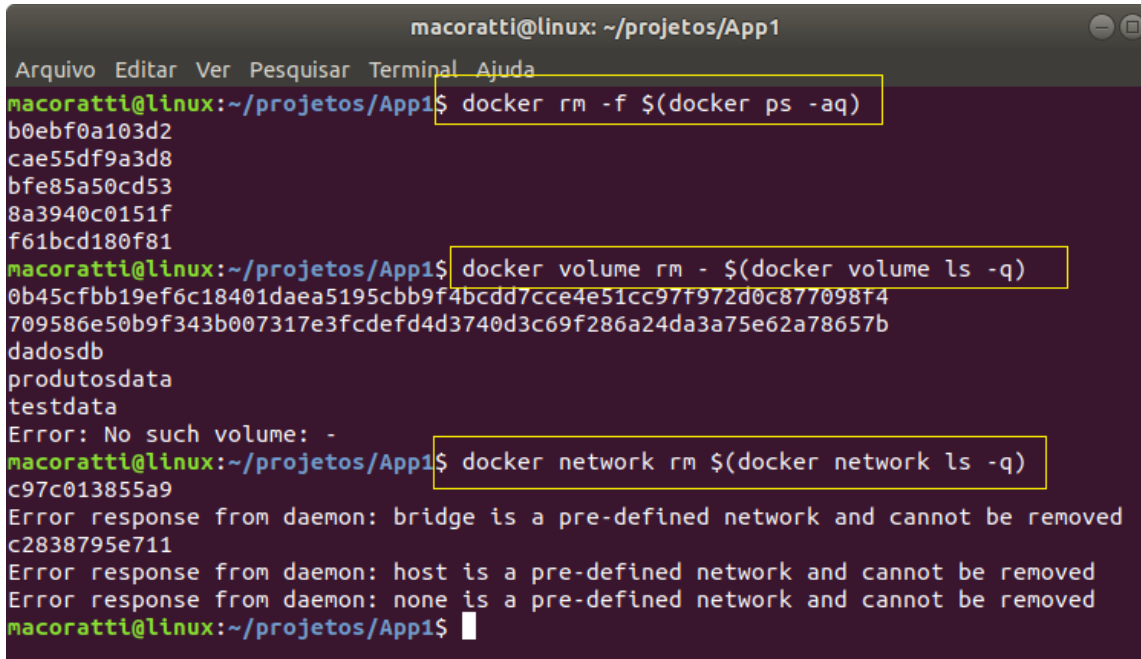
Para verificar digite o comando : **docker-compose --version**

Você deverá ver exibida a versão instalada do [docker-compose](#).

Preparando o ambiente

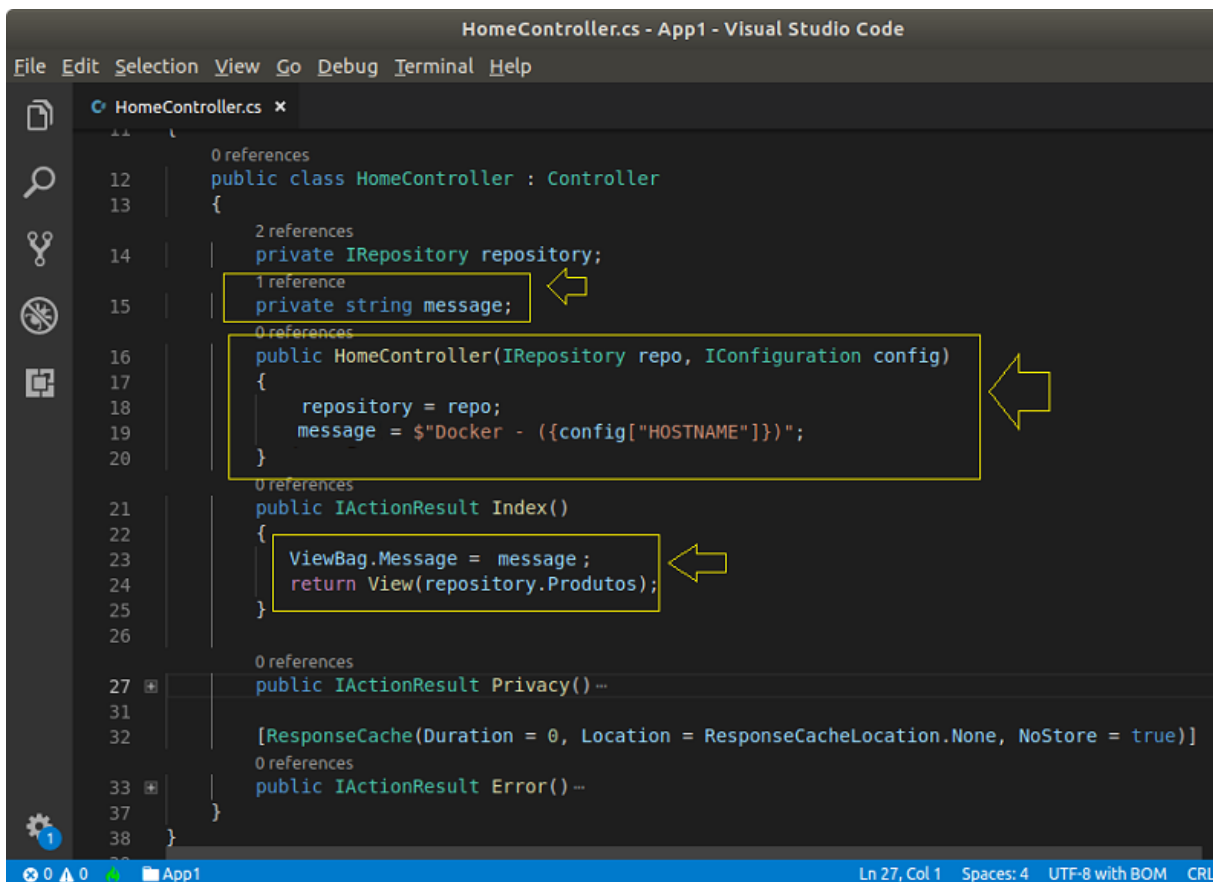
Antes de prosseguir vamos limpar todos os [contêineres](#), [volumes](#) e [redes](#) criadas em nosso ambiente digitando os comandos a seguir:

```
docker container rm -f $(docker ps -aq)
docker volume rm $(docker volume ls -q)
docker network rm $(docker network ls -q)
```



```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker rm -f $(docker ps -aq)
b0ebf0a103d2
cae55df9a3d8
bfe85a50cd53
8a3940c0151f
f61bcd180f81
macoratti@linux:~/projetos/App1$ docker volume rm - $(docker volume ls -q)
0b45cfbb19ef6c18401daea5195cbb9f4bcd7c7ce4e51cc97f972d0c877098f4
709586e50b9f343b007317e3fcdefd4d3740d3c69f286a24da3a75e62a78657b
dadosdb
produtosdata
testdata
Error: No such volume: -
macoratti@linux:~/projetos/App1$ docker network rm $(docker network ls -q)
c97c013855a9
Error response from daemon: bridge is a pre-defined network and cannot be removed
c2838795e711
Error response from daemon: host is a pre-defined network and cannot be removed
Error response from daemon: none is a pre-defined network and cannot be removed
macoratti@linux:~/projetos/App1$
```

Vamos agora ajustar a aplicação que criamos no [artigo 7](#) alterando o código do controlador [HomeController](#) conforme abaixo:



```
HomeController.cs - App1 - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
HomeController.cs x
0 references
12 public class HomeController : Controller
13 {
14     2 references
15     private IRepository repository;
16     1 reference
17     private string message;
18     0 references
19     public HomeController(IRepository repo, IConfiguration config)
20     {
21         repository = repo;
22         message = $"Docker - ({config["HOSTNAME"]}");
23     }
24     0 references
25     public IActionResult Index()
26     {
27         ViewBag.Message = message;
28         return View(repository.Produtos);
29     }
30     0 references
31     public IActionResult Privacy() ...
32     [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
33     0 references
34     public IActionResult Error() ...
35 }
Ln 27, Col 1 Spaces: 4 UTF-8 with BOM CRL
```

Estamos incluindo uma variável [message](#) e definindo um item de configuração chamado [HOSTNAME](#) para exibir o nome do [Hostna](#) view.

Na classe [Startup](#), no método [ConfigureServices\(\)](#) inclua a linha de código abaixo que registra um serviço para [IConfiguration](#):

```
...
```

```
services.AddSingleton<IConfiguration>(Configuration);
services.AddTransient<IRepository, ProdutoRepository>();
...
```

Iremos aplicar este recurso para usar a variável de ambiente do Docker chamada `HOSTNAME` dentro dos contêineres que será atribuída para o [ID do contêiner](#) e exibida na [View](#) para identificar o contêiner.

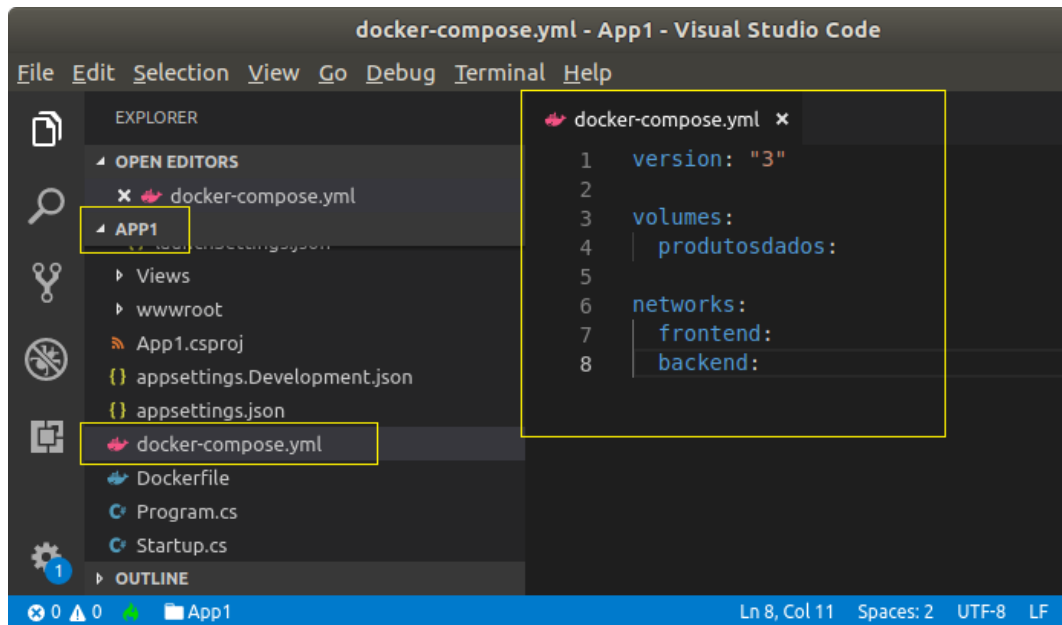
Criando o arquivo de composição

Vamos agora criar um arquivo de composição para configurar corretamente os componentes de uma aplicação que o Docker compose vai usar para gerenciar os contêineres, volumes e redes.

Este arquivo de composição possui a extensão `.yaml` e o formato [YAML](#) que possui um formato específico onde a indentação com espaços é usada para definir a estrutura do arquivo.

Vamos criar o arquivo `docker-compose.yml` na pasta `App1` da aplicação de exemplo criada no [artigo 7](#).

A seguir vamos definir a seguinte configuração neste arquivo:



Neste arquivo temos as seguinte seções:

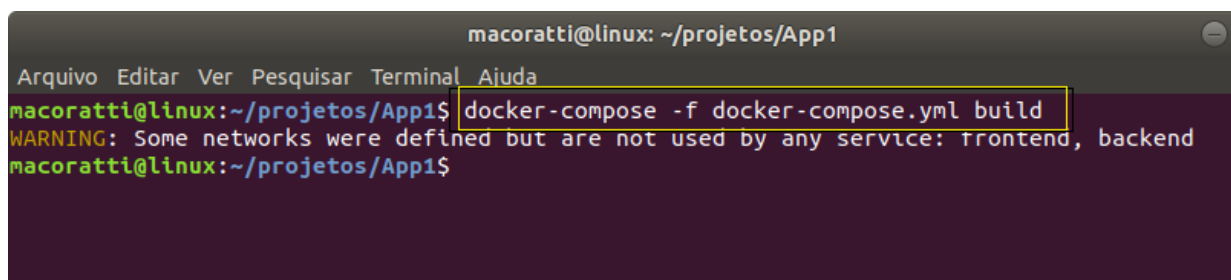
Nome	Descrição
version	Essa configuração especifica a versão do esquema Docker Compose que o arquivo usa. Atualmente, a versão mais recente é 3.
volumes	Esta configuração é usada para configurar os volumes que serão usados pelos contêineres definidos para compor arquivo. Este exemplo define um volume chamado <code>produtosdados</code> .
networks	Esta configuração é usada para configurar as redes que serão usadas pelos contêineres definidos no arquivo de composição. Este exemplo define redes chamadas <code>frontend</code> e <code>backend</code> .

A seção `version` informa ao Docker qual versão do esquema do arquivo de composição está sendo usada. A versão mais recente é 3, que inclui suporte para os recursos mais recentes do Docker.

Os arquivos de composição são processados usando uma ferramenta de linha de comando chamada `docker-compose` (observe o hífen), que constrói e gerencia o aplicativo.

Nosso arquivo de composição ainda esta no início tendo pouca configuração útil mas mesmo assim vamos executar o arquivo digitando o comando abaixo na pasta `App1` apenas para testar:

`docker-compose -f docker-compose.yml build`



O argumento **-f** é usado para especificar o nome do arquivo de composição, apesar de o Docker usar como padrão o nome `docker-compose.yml` ou `docker-compose.yaml` se um arquivo não for especificado.

O argumento **build** informa ao Docker para processar o arquivo e criar as imagens do Docker que ele contém. Não há imagens definidas no arquivo no momento, e veremos uma mensagem de alerta(**WARNING**) indicando qual as redes definidas não estão sendo usadas.

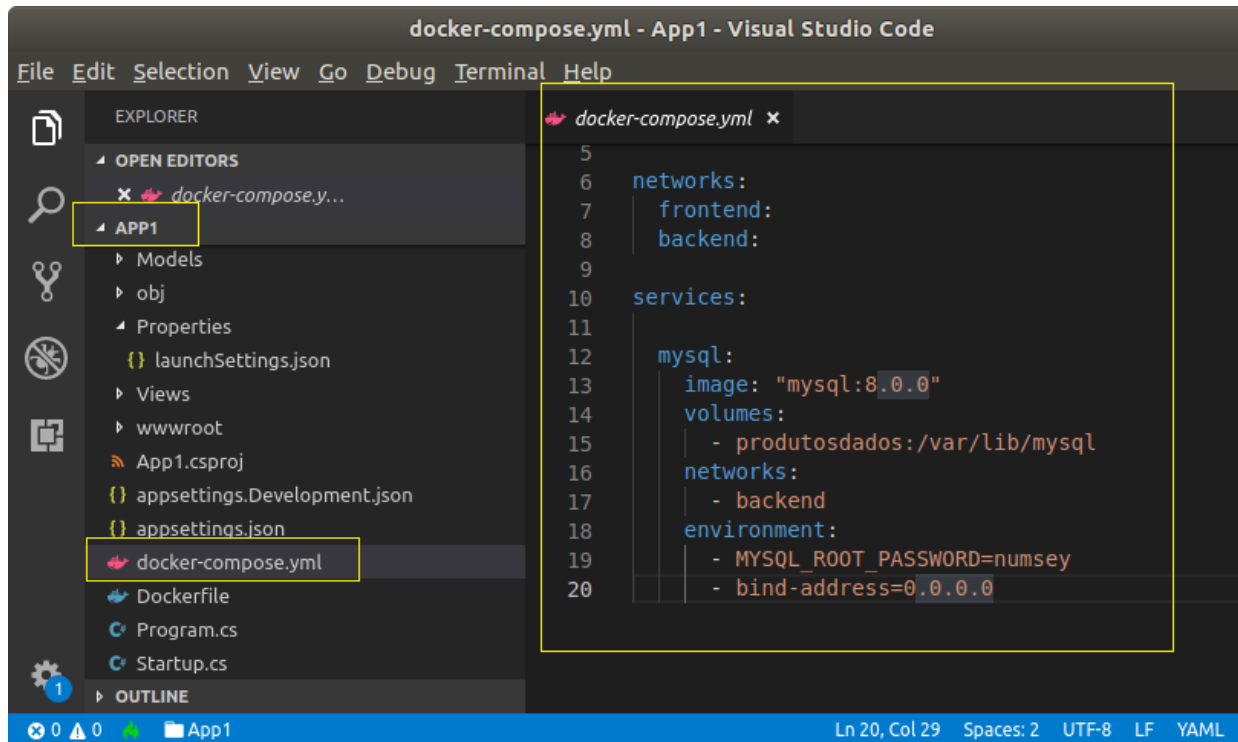
Esse aviso indica que o arquivo de composição informa ao Docker para criar as redes, mas elas não estão sendo usadas em nenhum outro lugar do aplicativo e, portanto, não foram criadas.

Compondo o banco de dados

Vamos continuar a definir o nosso arquivo de composição configurando o banco de dados.

O processo para descrever o aplicativo usando o **Docker Compose** segue o mesmo caminho da criação dos contêineres manualmente e a próxima etapa é configurar o contêiner do banco de dados.

Vamos incluir no arquivo de composição criado a configuração do contêiner para o MySQL conforme mostrado na figura a seguir:



A palavra-chave **services** é usada para indicar a seção do arquivo que contém as descrições que serão usadas para criar contêineres. O termo serviço é usado porque a descrição pode ser usada para criar mais de um contêiner. Cada serviço recebe sua própria seção.

O serviço descrito no arquivo é chamado de **mysql** e descreve como os contêineres de banco de dados devem ser criados usando as propriedades de configuração descritas a seguir:

Nome	Descrição
services	Indica o início da seção de serviços do arquivo de composição, que descreve os serviços que serão usados para criar contêineres
mysql	Esta propriedade indica o início de uma descrição de serviço chamada mysql .
image	Essa propriedade especifica a imagem do Docker que será usada para criar contêineres. Neste exemplo, a imagem oficial do MySQL será usada
volumes	Esta propriedade especifica os volumes que serão usados pelos contêineres e pelos diretórios eles serão usados. Neste exemplo, o volume produtosdados será usado para fornecer o conteúdo do diretório /var/lib/mysql .
networks	Esta propriedade especifica as redes às quais os contêineres serão conectados. Neste exemplo, os contêineres serão conectados à rede backend .
environment	Esta propriedade é usada para definir as variáveis de ambiente que serão usadas quando um recipiente for criado. Neste exemplo, definimos as variáveis MYSQL_ROOT_PASSWORD e bind-address .

Vamos agora executar o comando para verificar se as alterações feitas podem ser processadas.

Digite o comando : **docker-compose build**

Nota: Eu estou omitindo o *nome do arquivo* pois estou usando o nome do *arquivo padrão* que o Docker vai procurar.

```

macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker-compose build
WARNING: Some networks were defined but are not used by any service: frontend
mysql uses an image, skipping
macoratti@linux:~/projetos/App1$

```

O aviso sobre as redes não utilizadas foi alterado porque o contêiner do banco de dados será conectado à rede backend. A outra parte da saída indica que nenhuma ação é necessária para o serviço `mysql` no momento porque ele é baseado em uma imagem existente, que será usada para criar e configurar um contêiner quando o arquivo de composição for usado para iniciar o aplicativo.

Testando a configuração da aplicação

Já temos configuração suficiente no arquivo de composição para realizar um teste.

Vamos executar, na pasta do projeto `App1`, o comando : **docker-compose up**

```

macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker-compose up
WARNING: Some networks were defined but are not used by any service: frontend
Creating network "app1_backend" with the default driver
Creating volume "app1_produtosdados" with default driver
Creating app1_mysql_1_9c50ada8467c ... done
Attaching to app1_mysql_1_6b51910bbf9a
mysql_1_6b51910bbf9a | Initializing database
mysql_1_6b51910bbf9a | 2018-11-13T19:53:10.143404Z 0 [Warning] TIMESTAMP with implicit
mysql_1_6b51910bbf9a | use --explicit_defaults_for_timestamp server option (see documentation for
mysql_1_6b51910bbf9a | mbind: Operation not permitted
mysql_1_6b51910bbf9a | mbind: Operation not permitted
mysql_1_6b51910bbf9a | mbind: Operation not permitted
mysql_1_6b51910bbf9a | mbind: Operation not permitted
mysql_1_6b51910bbf9a | 2018-11-13T19:53:12.207414Z 1 [Warning] InnoDB: New log fi
mysql_1_6b51910bbf9a | 2018-11-13T19:53:12.737879Z 1 [Warning] InnoDB: Creating f

```

O comando **docker-compose up** informa ao Docker para processar o conteúdo do arquivo de composição e configurar os volumes, redes e contêineres especificados. O Docker vai baixar todas as imagens necessárias do Docker Hub para que elas possam ser usadas para criar um contêiner.

Os detalhes do processo de configuração são mostrados no prompt de comando, junto com a saída dos contêineres criados.

O nome da rede, o volume e o container que o Docker cria são prefixados com `app1_`. A rede é nomeada como `app1_backend`, o volume é chamado de `app1_produtosdados` e o contêiner é chamado de `app1_mysql_1_<aleatorio>`. (A parte `_1` do nome do container MySQL refere-se a como o Docker escala os aplicativos descritos usando arquivos de composição).

O prefixo é retirado do **nome do diretório** que **contém o arquivo de composição** e garante que diferentes arquivos de composição podem usar os mesmos nomes para redes, volumes e contêineres sem haver conflitos quando o aplicativo for iniciado. (Você pode alterar o prefixo usado com o argumento `-p` para o comando `docker-compose up` command).

Quando o banco de dados tiver concluído seu processo de inicialização, digite **Control + C** para finalizar o comando `dockercompose` e pare os contêineres descritos no arquivo de composição.

Você pode explorar o resultado do processamento do arquivo de composição usando os comandos Docker.

1- exibir os contêineres criados : **docker container ps -a**

```

macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker container ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
e4d085bb849e        mysql:8.0.0        "docker-entrypoint.s..." 14 minutes ago     Exited (0) 3 minutes ago
macoratti@linux:~/projetos/App1$

```

2- exibir as redes criadas : **docker network ls**

```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
0523f5b41f15        app1_backend        bridge              local
27f81ed2c98b        bridge              bridge              local
a83847d4c15e        host                host                local
853d3af4b37f        none                null                local
macoratti@linux:~/projetos/App1$
```

Apenas uma foi criada pois nenhum contêiner se conecta à rede [front_end](#)

3- exibir os volumes : **docker volume ls**

```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker volume ls
DRIVER              VOLUME NAME
local               app1_produtosdados
macoratti@linux:~/projetos/App1$
```

Antes de prosseguir vamos executar o comando abaixo na pasta [App1](#) para remover os contêineres e as redes que estão descritos no arquivo de composição. Não vamos remover os volumes (*para isso use argumento -v*).

docker-compose down

```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ docker-compose down
WARNING: Some networks were defined but are not used by any service: frontend
Removing app1_mysql_1_6b51910bbf9a ... done
Removing network app1_backend
macoratti@linux:~/projetos/App1$
```

Na próxima parte do artigo vamos continuar a definição do arquivo de composição para a aplicação ASP .NET Core MVC.