



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



Neste artigo vamos realizar as seguintes tarefas: ([artigo anterior](#))

1. Ajustar a aplicação ASP .NET Core MVC para usar o banco de dados MySQL e fazer o acesso aos dados usando o EF Core;
2. Criar uma imagem da aplicação da aplicação ajustada;
3. Criar um contêiner da imagem da aplicação;
4. Executar o contêiner da aplicação e realizar a comunicação do seu contêiner com o contêiner do banco de dados MySql



A primeira tarefa você pode acompanhar neste artigo: [ASP .NET Core MVC - Ajustando o projeto para usar o MySQL - II](#)

Após acompanhar o artigo citado acima retorne a este artigo e prossiga.

Criando a imagem para a aplicação MVC

Chegou a hora de atualizar a imagem para a nossa aplicação MVC de forma que ela inclua o código que ajustamos para dar suporte ao banco de dados **MySQL**, aos pacotes do **EF Core** e a migração inicial que irá criar o banco de dados e a tabela.

Antes de criar a imagem temos que refazer o deploy da aplicação para a pasta dist para atualizar os arquivos da aplicação. Para isso digite o comando abaixo:

dotnet publish --configuration Release --output dist

```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ sudo dotnet publish --configuration Release --output dist
[sudo] senha para macoratti:
Microsoft(R) Build Engine versão 15.8.169+g1ccb72aefa para .NET Core
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Restauração concluída em 63,12 ms para /home/macoratti/projetos/App1/App1.csproj.
App1 -> /home/macoratti/projetos/App1/bin/Release/netcoreapp2.1/App1.dll
App1 -> /home/macoratti/projetos/App1/bin/Release/netcoreapp2.1/App1.Views.dll
App1 -> /home/macoratti/projetos/App1/dist/
macoratti@linux:~/projetos/App1$
```

Agora vamos recriar a imagem da aplicação **ASP .NET Core MVC**:

docker build -t aspnetcoremvc/app:2.0 .

```

macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ sudo docker build -t aspnetcoremvc/app:2.0 .
Sending build context to Docker daemon 11.72MB
Step 1/6 : FROM microsoft/dotnet:2.1-aspnetcore-runtime
--> 1fe6774e5e9e
Step 2/6 : LABEL version="1.0.0" description="Aplicacao ASP .NET Core MVC"
--> Using cache
--> 4d6a7042f935
Step 3/6 : COPY dist /app
--> e4ccb69ca757
Step 4/6 : WORKDIR /app
--> Running in b1a6b70d9d84
Removing intermediate container b1a6b70d9d84
--> c436d134e350
Step 5/6 : EXPOSE 80/tcp
--> Running in d8f2e2ed8118
Removing intermediate container d8f2e2ed8118
--> 3b351dc05587
Step 6/6 : ENTRYPOINT ["dotnet", "App1.dll"]
--> Running in 33ec620e3b93
Removing intermediate container 33ec620e3b93
--> ce203fde8694
Successfully built ce203fde8694
Successfully tagged aspnetcoremvc/app:2.0
macoratti@linux:~/projetos/App1$

```

Visualizando as imagens locais : [docker images](#)

```

macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~/projetos/App1$ sudo docker images

```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|--------------------|------------------------|--------------|---------------|--------|
| aspnetcoremvc/app | 2.0 | ce203fde8694 | 2 minutes ago | 261MB |
| macvol/app2 | 1.0 | 85bed7379968 | 9 hours ago | 4.82MB |
| aspnetcoremvc/app1 | 1.0 | a90c99ff854c | 29 hours ago | 257MB |
| macoratti/nginx | 1.0 | 1be6668aea57 | 2 days ago | 202MB |
| microsoft/dotnet | 2.1-aspnetcore-runtime | 1fe6774e5e9e | 3 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:~/projetos/App1$

```

Vemos a imagem [aspnetcoremvc/app](#) com tag [2.0](#) definida e pronta para ser usada.

Assim já temos a imagem da aplicação MVC e a imagem do MySQL.

Resta fazer o teste final criando um novo contêiner com a imagem da aplicação MVC e fazer com que ele se comunique com o banco de dados MySQL que já deve estar rodando no contêiner [mysql](#) criado no artigo [anterior](#).

Para isso vamos entender mais um pouco sobre como o Docker realiza a comunicação entre contêineres.

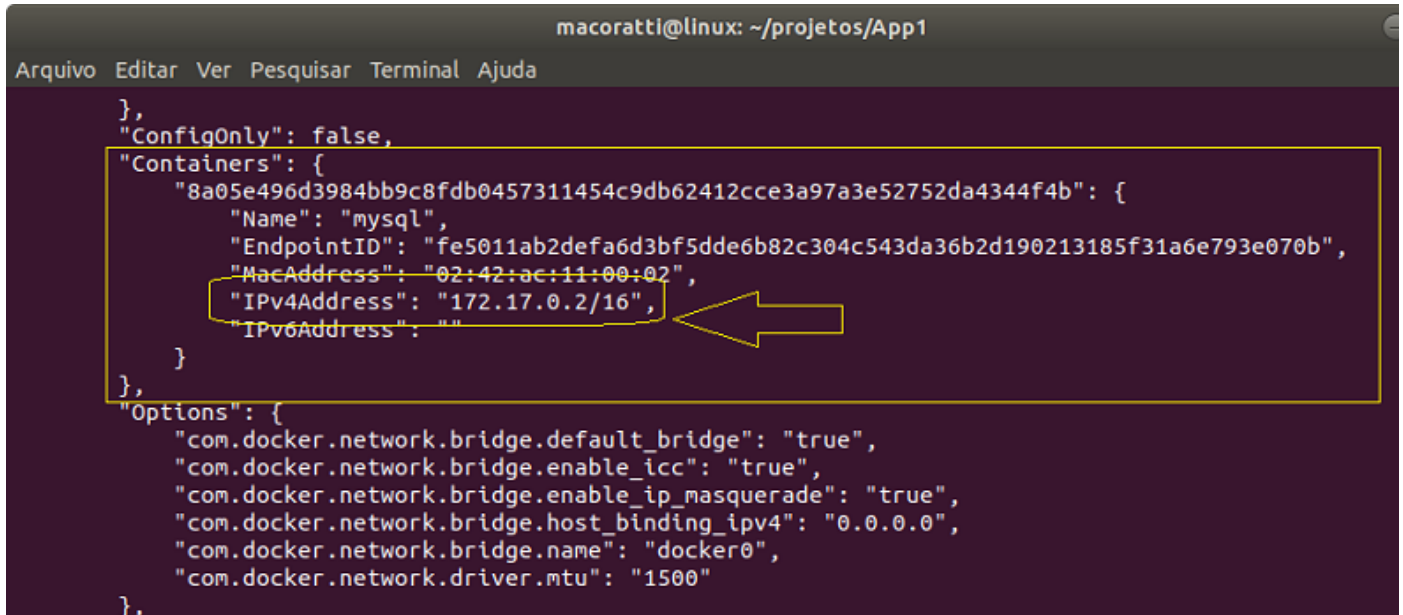
Quando você inicia um contêiner, o Docker conecta-o a uma rede virtual interna e atribui a ele um [Internet Protocolo \(IP\)](#) para que ele possa se comunicar com o servidor [host](#) e com outros contêineres na mesma rede. Este é o ponto de entrada para um recurso chave do Docker que veremos no próximo artigo: [as redes](#).

Para que o contêiner MVC converse com o banco de dados Mysql, preciso saber o [endereço IP](#) que o Docker atribuiu ao contêiner do MySQL que criamos.

Para isso vamos executar o comando abaixo e examinar a configuração do Docker para [rede virtual](#).

docker network inspect bridge

A resposta desse comando mostrará como o Docker configurou a [rede virtual](#) e vai exibir uma seção [Contêineres](#) que mostra os contêineres conectados à rede e os endereços IP que são atribuídos a eles. Deve haver apenas um contêiner e seu campo [Name](#) será [mysql](#).



```
macoratti@linux: ~/projetos/App1
Arquivo Editar Ver Pesquisar Terminal Ajuda
{
  "ConfigOnly": false,
  "Containers": {
    "8a05e496d3984bb9c8fdb0457311454c9db62412cce3a97a3e52752da4344f4b": {
      "Name": "mysql",
      "EndpointID": "fe5011ab2defa6d3bf5dde6b82c304c543da36b2d190213185f31a6e793e070b",
      "MacAddress": "02:42:ac:11:00:02",
      "IPv4Address": "172.17.0.2/16",
      "IPv6Address": ""
    }
  },
  "Options": {
    "com.docker.network.bridge.default_bridge": "true",
    "com.docker.network.bridge.enable_icc": "true",
    "com.docker.network.bridge.enable_ip_masquerade": "true",
    "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
    "com.docker.network.bridge.name": "docker0",
    "com.docker.network.driver.mtu": "1500"
  }
}
```

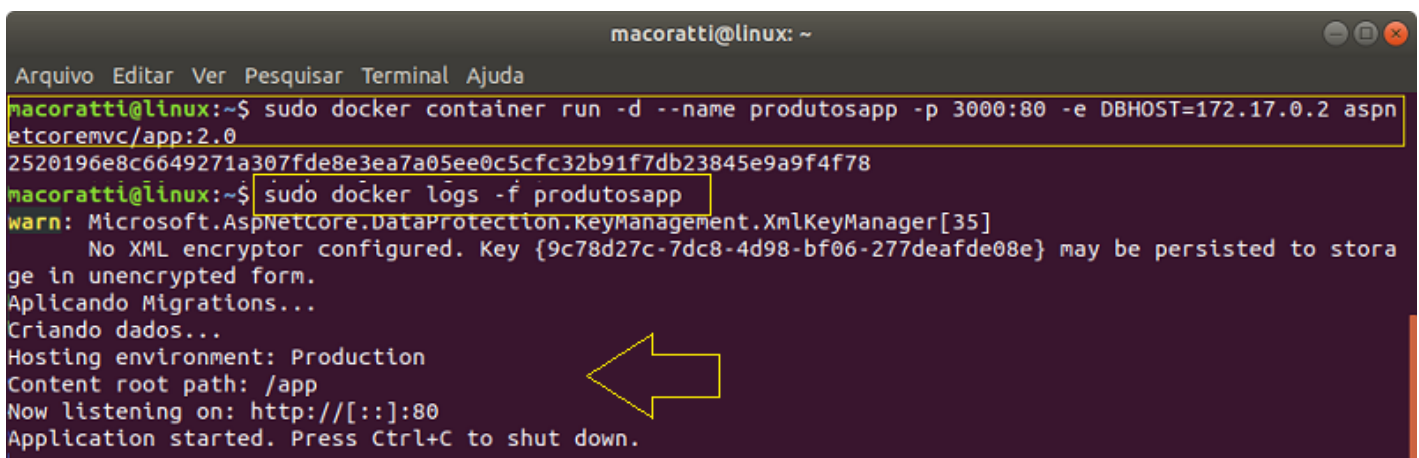
Anote o endereço do campo: `"IPv4Address" : "172.17.0.2/16"`

Esse é o endereço IP que o Docker atribuiu ao contêiner. Para mim, o endereço é [172.17.0.2](#), mas você pode ver um endereço diferente. Este é o endereço IP que o aplicativo MVC deve usar para se comunicar com o banco de dados MySQL.

Esse endereço pode ser fornecido ao aplicativo por meio da variável de ambiente [DBHOST](#), que foi definida na classe [Startup](#) do aplicativo MVC no qual fizemos os ajustes.

Vamos passar esse valor quando formos criar o contêiner. Assim digite o comando abaixo para criar o contêiner da aplicação MVC:

`docker container run -d --name produtosapp -p 3000:80 -e DBHOST=172.17.0.2 aspnetcoremvc/app:2.0`



```
macoratti@linux: ~
Arquivo Editar Ver Pesquisar Terminal Ajuda
macoratti@linux:~$ sudo docker container run -d --name produtosapp -p 3000:80 -e DBHOST=172.17.0.2 aspnetcoremvc/app:2.0
2520196e8c6649271a307fde8e3ea7a05ee0c5cfc32b91f7db23845e9a9f4f78
macoratti@linux:~$ sudo docker logs -f produtosapp
warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
      No XML encryptor configured. Key {9c78d27c-7dc8-4d98-bf06-277deafde08e} may be persisted to storage in unencrypted form.
Aplicando Migrations...
Criando dados...
Hosting environment: Production
Content root path: /app
Now listening on: http://[::]:80
Application started. Press Ctrl+C to shut down.
```

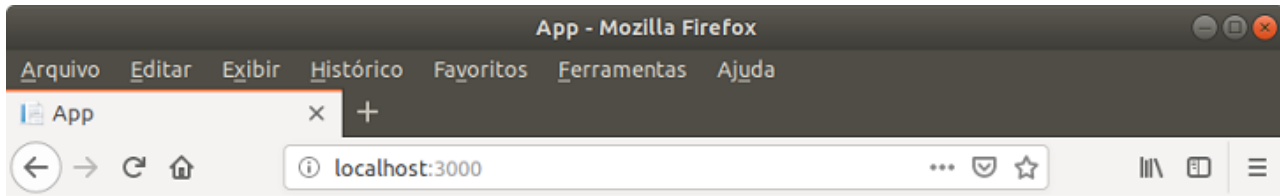
Criamos um contêiner com o nome [produtosapp](#) rodando em background (`-d`) passando o IP([172.17.0.2](#)) para que este contêiner se comunique com o contêiner [mysql](#).

No comando estamos mapeando a [porta 3000](#) no host para a porta 80 no [contêiner](#), que é a porta que o Kestrel está usando para receber solicitações HTTP para o runtime do ASP.NET Core.

A seguir exibimos o log gerado usando o comando: [docker container logs -f produtosapp](#)

Vemos no log as mensagens aplicando a migração e a criação dos dados na tabela [Produtos](#).

Vamos agora abrir um navegador no endereço: <http://localhost:3000>



The screenshot shows a Mozilla Firefox browser window with the title 'App - Mozilla Firefox'. The address bar displays 'localhost:3000'. The main content area shows a table with four columns: ID, Nome, Categoria, and Preço. The table contains seven rows of data representing sports equipment.

| ID | Nome | Categoria | Preço |
|----|---------------------|-----------|---------|
| 1 | Luvas de goleiro | Futebol | \$25.00 |
| 2 | Bola de basquete | Basquete | \$48.95 |
| 3 | Bola de Futebol | Futebol | \$19.50 |
| 4 | Óculos para natação | Aquáticos | \$34.95 |
| 5 | Meias Grandes | Futebol | \$50.00 |
| 6 | Calção de banho | Aquáticos | \$16.00 |
| 7 | Cesta para quadra | Basquete | \$29.95 |

Note que os dados exibidos são os dados que definimos no método [IncluiDadosDB](#) da classe [Populadb](#).

Conclusão: Criamos um contêiner para uma aplicação ASP .NET Core MVC que acessa dados de um banco de dados MySQL que esta em outro contêiner.

Na [próxima aula](#) vamos entender o que é a rede do Docker.