ASP.NET 5/Core + Docker + Variaveis de ambientes

#Docker #.NET Core #.NET 5

Gabriel Faraday

24/03/2021 10:46

Este artigo é sequência do anterior que escrevi aqui na DIO, onde escrevi sobre "*Trabalhando com variáveis de ambiente e arquivos de configuração no ASP.NET 5/Core*" (https://digitalinnovation.one/artigos/trabalhando-com-variaveis-de-ambiente-e-arquivos-de-configuracao-no-aspnet-5core).

Eu também vou utilizar os conceitos apresentados em outro artigo que escrevi aqui sobre Docker (https://digitalinnovation.one/artigos/conhecendo-o-docker).

Se você ainda não leu estes artigos, sugiro que os leia antes para fazer mais sentido. Irei usar a mesma api do artigo anterior.

Então recapitulando, nós temos uma web api que possui 3 arquivos de configuração, um para ambiente de dev, outro para ambiente de qa e outro para ambiente de prod. Em cada arquivo de configuração é referenciada uma respectiva "connection string" diferente para acesso a um banco de dados fictício.

No inicializar da aplicação , dependendo do valor que eu passo na variável de ambiente "ASPNETCORE_ENVIRONMENT", um desses arquivos é utilizado como configuração da api e isso determina por exemplo qual base de dados a aplicação utilizaria, através da connection string.

Porém, no artigo anterior, vimos como eu executo a api em modo debug, mas e como seria para executar a api dentro de um container Docker, diferenciando os ambientes?

Veremos a seguir por partes.

ASP.NET 5/Core + Docker

Para rodar nossa aplicação num container Docker precisamos criar nosso Dockerfile na raiz do nosso projeto, com o conteúdo a seguir:

```
FROM mcr.microsoft.com/dotnet/sdk:5.0-focal AS build-env

WORKDIR /app

COPY api-dio.csproj ./

RUN dotnet restore

COPY . ./

RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:5.0

WORKDIR /app

COPY --from=build-env /app/out .

EXPOSE 80

ENTRYPOINT [ "dotnet", "api-dio.dll" ]
```

Vamos entender cada linha aqui:

- FROM mcr.microsoft.com/dotnet/sdk:5.0-focal AS build-env

Iremos começar a construção de nossa imagem tendo como base a imagem de sdk do .NET 5 e vamos dar um alias (apelido) para essa etapa de "build-env"

- WORKDIR /app

Estamos definindo que a pasta que estarei trabalhando é a "app" (mesmo que não tenha, ele irá criar!)

- COPY api-dio.csproj ./

Copiamos o arquivo csproj para dentro da imagem, exatamente na pasta que estamos trabalhando, definida no step anterior (app)

- RUN dotnet restore

Como estamos usando uma imagem base do sdk do .NET, temos acesso ao dotnet CLI, e então executamos o restore dos pacotes da aplicação, no caso do arquivo csproj copiado no step anterior

- COPY . ./

Após o restore ser concluído, então copiamos todos os demais arquivos da nossa aplicação para dentro da imagem, na nossa workdir (./), que é a "app"

- RUN dotnet publish -c Release -o out

Então executamos o comando de publish com configuração (-c) de Release e indicamos que os arquivos compilados (DLLs) devem ser gerados numa pasta (-o) "out".

Até aqui temos a primeira fase (stage) do nosso build, que é no fim das contas a aplicação compilada em modo Release, pronta para "deploy". Então seguimos para a segunda etapa (stage) do build. Essa estratégia de dividir o build da imagem em várias etapas é chamada de multistage build e ajuda a obter melhor aproveitamento do cache interno do docker no momento de criação das imagens.

Seguindo então no nosso build temos:

- FROM mcr.microsoft.com/dotnet/aspnet:5.0

Para nosso segundo stage partiremos de uma nova imagem base, a imagem "aspnet" na versão 5, esta é uma imagem otimizada para deploy de uma aplicação aspnet

- WORKDIR /app

Dentro deste novo stage também definimos nossa pasta de trabalho nomeada "app" (não é a mesma app do stage anterior)

- COPY --from=build-env /app/out .

Aqui vem a mágica do multistage build: eu pego o resultado final do meu publish em modo Release do stage anterior "build-env" (que foi publicado na pasta app/out/) e copio ele para a minha pasta de trabalho atual (.) neste segundo stage (no caso uma pasta app)

- EXPOSE 80

Indico qual a porta que o container desta imagem irá esperar requests, para conseguir vincular uma porta do meu host no momento de execução

- ENTRYPOINT ["dotnet", "api-dio.dll"]

Indico qual comando será executado quando um container desta imagem subir, que no nosso caso é a execução da api (já compilada)

Para criarmos uma imagem Docker com base no nosso Dockerfile basta executar o seguinte comando na raiz do projeto:

```
docker build . -t apidio
```

Estamos aqui dizendo ao dockerd: construa uma imagem utilizando o Dockerfile da pasta atual (.) e insira o nome (-t) nela de "apidio".

Após isso podemos confirmar se a imagem foi gerada através do comando

```
docker images

REPOSITORY               TAG               IMAGE ID    CREATED     SIZE
apidio                  latest            1523f362f979  2 minutes ago  210MB
```

Docker + Variaveis de ambiente

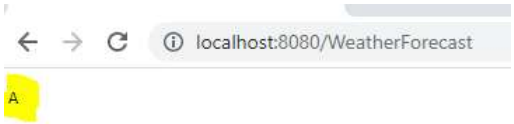
Após termos criado a imagem docker, vamos executá-la, mas lembre-se: temos que passar para o container o valor desejado para a variável de ambiente "ASPNETCORE_ENVIRONMENT", para que a aplicação ao iniciar possa saber qual config deve utilizar.

Desta forma podemos executar o seguinte comando:

```
docker run -p 8080:80 -e ASPNETCORE_ENVIRONMENT=Development apidio
```

Aqui estamos dizendo ao dockerd: rode um container da imagem "apidio" e ligue a porta 80 do container à porta 8080 da máquina física. E insira no container uma variável de ambiente (-e) ASPNETCORE_ENVIRONMENT com valor Development.

O resultado pode ser visto no log de inicialização da aplicação e ao acessar a nossa api no navegador em http://localhost:8080/WeatherForecast:



Agora se executarmos o comando a seguir conseguiremos ver dados do container em execução:

```
docker ps

CONTAINER ID  IMAGE   COMMAND         CREATED     STATUS     PORTS          NAMES
bb4bb94457a9  apidio  "dotnet api-dio.dll"  17 minutes ago  Up 17 minutes  0.0.0.0:8080-
>80/tcp  condescending_grothendieck
```

Para encerrar a execução do container, basta executar o seguinte comando, trocando <container_id> pelo CONTAINER ID obtido no comando anterior (docker ps):

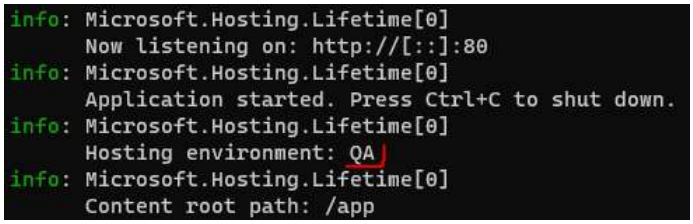
```
docker stop <container_id>
```

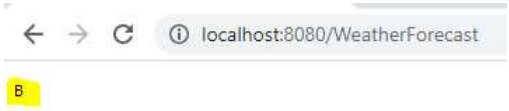
Agora podemos executar o seguinte comando para subir a api considerando como se fosse ambiente de QA:

```
docker run -p 8080:80 -e ASPNETCORE_ENVIRONMENT=QA apidio
```

Aqui estamos dizendo ao dockerd: rode um container da imagem "apidio" e ligue a porta 80 do container à porta 8080 da máquina física. E insira no container uma variável de ambiente (-e) ASPNETCORE_ENVIRONMENT com valor QA.

O resultado pode ser visto no log de inicialização da aplicação e ao acessar a nossa api no navegador em http://localhost:8080/WeatherForecast:





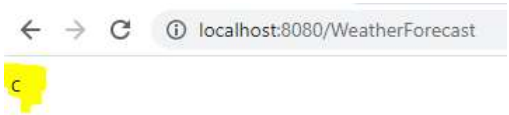
E por fim, podemos parar o container (conforme explicado anteriormente) e executar o seguinte comando para subir a api considerando como se fosse ambiente de Prod:

```
docker run -p 8080:80 -e ASPNETCORE_ENVIRONMENT=Prod apidio
```

Aqui estamos dizendo ao dockerd: rode um container da imagem "apidio" e ligue a porta 80 do container à porta 8080 da máquina física. E insira no container uma variável de ambiente (-e) ASPNETCORE_ENVIRONMENT com valor Prod.

O resultado pode ser visto no log de inicialização da aplicação e ao acessar a nossa api no navegador em http://localhost:8080/WeatherForecast:

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Prod
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```



Recapitulando

Complementando o artigo anterior (já citado no início deste texto), subimos nossa api em um container docker, diferenciando o ambiente através da variável de ambiente ASPNETCORE_ENVIRONMENT.

O fato é que podemos passar qualquer variável de ambiente da mesma forma para um container, ou seja, o céu é o limite! Em um ambiente de produção, por exemplo, a connection string pode ser passada para dentro do container pelo orquestrador de containers em forma de variável de ambiente, desta forma o arquivo de config não precisa ter dados sensíveis expostos!

COMENTAR

Comentários (2)

0

Alexsander Rossi
24/03/2021 11:18

Essa é pra imprimir e encadernar e manter ao lado do PC, excelente conteúdo 👏👏

0

Rosemeire Deconti
24/03/2021 11:14

Adorei! Informação precisa! Com certeza vou utilizar durante os meus estudos! Grata!

anterior 1 próximo

Gabriel Faraday

None

SEGUIR