Build an ASP.NET Core Service and App with .NET (Core) 5.0 Two-Day Hands-On Lab

Lab 12 (Optional)

This lab is the last in the series that builds the RESTful service. This lab configures the application to run entirely in Docker. Prior to starting this lab, you must have completed Lab 11.

Part 1: Add Docker Support to AutoLot.Api

To configure the service to run in Docker, start with a `DockerFile`. This will be referenced by the Docker-Compose orchestration later in this lab.

Visual Studio

- Right click on the `AutoLot.Api` project and select Add -> Docker support. This will create a `Dockerfile` in the project and add a new profile in `launchSettings.json` for running in Docker.
  NOTE: Be sure to select Linux containers and not Windows containers
- Add a reference to the `Microsoft.VisualStudio.Azure.Containers.Tools.Targets`

```
dotnet add AutoLot.Api package Microsoft.VisualStudio.Azure.Containers.Tools.Targets
```

- Check the contents of the generated `Dockerfile` against the `DockerFile` in the next section

Visual Studio Mac (Experimental)

- Right click on the `AutoLot.Api` project and select Add -> Docker support. This will create a `Dockerfile` in the project and add a new profile in `launchSettings.json` for running in Docker.
  NOTE: Be sure to select Linux containers and not Windows containers
- PHIL: not sure about the targets ???
- Check the contents of the generated `Dockerfile` against the `DockerFile` in the next section

Visual Studio Code

- Update the `AutoLot.Api` project file for Docker and Docker Compose

```
<PropertyGroup>
  <DockerComposeProjectPath>..\docker-compose.dcproj</DockerComposeProjectPath>
  <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
</PropertyGroup>
```

- Add a Docker profile to the `AutoLot.Api` `launchSettings.json` file (the **bolded** portion):

```
{
...
    "Docker": {
      "commandName": "Docker",
      "launchBrowser": true,
      "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}/swagger",
      "environmentVariables": {
        "ASPNETCORE_URLS": "https://+:443;http://+:80",
        "ASPNETCORE_HTTPS_PORT": "5021"
      },
      "httpPort": 5020,
      "useSSL": true,
      "sslPort": 5021
    }
 }
}
```

Create the Dockerfile

- Create a file named `Dockerfile` in the root of the `AutoLot.Api` project. Update the file to the following:

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0-buster-slim AS build
WORKDIR /src
COPY ["AutoLot.Api/AutoLot.Api.csproj", "AutoLot.Api/"]
COPY ["AutoLot.Api/AutoLot.Api.xml", "app/"]
COPY ["AutoLot.Models/AutoLot.Models.csproj", "AutoLot.Models/"]
COPY ["AutoLot.Services/AutoLot.Services.csproj", "AutoLot.Services/"]
COPY ["AutoLot.Dal/AutoLot.Dal.csproj", "AutoLot.Dal/"]
RUN dotnet restore "AutoLot.Api/AutoLot.Api.csproj"
COPY . .
WORKDIR "/src/AutoLot.Api"
RUN dotnet build "AutoLot.Api.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "AutoLot.Api.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "AutoLot.Api.dll"]
```


Part 2: Create the Docker-Compose Orchestration

Visual Studio

- Right click on the `AutoLot.Api` project and select Add -> Container Orchestration Support. Select Docker-Compose to add a new project named docker-compose into the solution. Update the `docker-compose.yml` and `docker-compose.override.yml` files as shown below.

  NOTE: Be sure to select Linux containers and not Windows containers

- When you update `docker-compose.yml` and `docker-compose.override.yml` in the docker-compose project, you will find the `override.yml` file as a child file of the main `docker-compose.yml` file.

Visual Studio Mac

- When you update `docker-compose.yml` and `docker-compose.override.yml` in the docker-compose project, you'll find the `override.yml` file as a child file of the main `docker-compose.yml` file.

Visual Studio Code

- Create `docker-compose.yml` and `docker-compose.override.yml` files in the solution directory

All IDE's

- Update the `docker-compose.yml` file to the following code. This creates an image for the database (named `db`) and then pulls in the image for the service. The depends on makes sure the database container is up and running before the service starts:

```
version: '3.8'

services:
  db:
    image: "mcr.microsoft.com/mssql/server:2019-latest"
    ports:
      - "7433:1433"
    environment:
      SA_PASSWORD: "P@ssw0rd"
      ACCEPT_EULA: "Y"
      MSSQL_PID: "Express"

  autolot.api:
    image: ${DOCKER_REGISTRY-}autolotapi
    ports:
      - "5020:80"
      - "5021:443"
    build:
      context: .
      dockerfile: AutoLot.Api/Dockerfile
    depends_on:
      - "db"
```


- Update the `docker-compose.override.yml` file to the following code.

```
version: '3.8'

services:
  autolot.api:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
    ports:
      - "80"
      - "443"
    volumes:
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
```

Part 3: Update the Connection String

- Update the connection string in the `appsettings.Development.json` file (`AutoLot.Api`):

```
"ConnectionStrings": {
    //Docker-Compose
    "AutoLot": "Server=db;Database=AutoLot50;User Id=sa;Password=P@ssw0rd; "
  }
```

Part 4: Run under HTTP (not HTTPS)

- To run HTTP in `Docker` comment out the `app.UseHttpsRedirection` in `Configure` (`Startup.cs`):

Part 5: Create Dev Certificate for HTTPS in Docker (Windows)

- To run HTTPS in Docker you must create a dev certificate. This is done in your development environment with the following command:

```
dotnet dev-certs https
```

- To run in Docker with a dev cert, it must be added into the docker container. Open PowerShell, and first clear out your existing certificates:

```
dotnet dev-certs https --clean
```

- Create a new dev cert as a file (replace `MySecretPassword` with a real password):

```
dotnet dev-certs https --trust -ep $env:USERPROFILE\.aspnet\https\aspnetapp.pfx -p
MySecretPassword
```


- Update the `docker-compose.override.yml` with the following changes (in bold):

```
version: '3.8'

services:
  autolot.api:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      - ASPNETCORE_Kestrel__Certificates__Default__Password=MySecretPassword
      - ASPNETCORE_Kestrel__Certificates__Default__Path=/https/aspnetapp.pfx
    ports:
      - "80"
      - "443"
    volumes:
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
      - ~/.aspnet/https:/https:ro
```

Part 5: Run the project in Docker

Visual Studio and Visual Studio Mac

- Select docker-compose from the drop down and click F5.

Visual Studio Code

- Open a command prompt in the location of the `docker-compose.yml` file. To run the project, enter:

```
Docker-compose up
```

- If the API project dies after starting, change `RebuildDataBase` to false in the app settings, or start the container again after waiting a few seconds.

Summary

This lab configured Docker and the Docker-Compose orchestration.

Next steps

In the next part of this tutorial series, you will start building the ASP.NET Core Web Application.