Containerize .NET 8 App with Docker Compose Launch Profiles

Bohdan Belei

.

Follow

4 min read

.

Dec 21, 2023

--

Listen

Share

Hey there, fellow developers! ■■ Are you diving into the world of Docker with your shiny new .NET 8 application and wondering how to leverage Docker Compose and Launch Settings profiles to streamline your deployment process? Well, you're in the right place! In this guide, we'll take a journey together to Containerize your .NET 8 app using Docker Compose profile which looks at your Compose file to determine the group of services to run. ■

Why Docker?

But first, let's recap why Docker is a game-changer. Docker simplifies application deployment by encapsulating your app and its dependencies into containers, ensuring consistency across various environments. And that's just scratching the surface of its awesomeness! ■

What's Docker Compose, anyway?

Docker Compose is like a conductor orchestrating a symphony of containers. It allows us to define and run multi-container Docker applications. It's like having your own little ecosystem where containers chat and collaborate. ■■

Getting Started

Assuming you have a sweet .NET 8 app ready, and Docker installed on your machine, the first step is to set up your **Dockerfile**. ■ Let's say we're building an **ASP.NET Web API** combined with a **PostgreSQL** database example. Here's a snippet to get you started:

```
# Dockerfile
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["DockerizeWithDockerComposeLaunchSettings/DockerizeWithDockerComposeLaunchSettings.csproj", "DockerizeWithDockerComposeL
RUN dotnet restore "./DockerizeWithDockerComposeLaunchSettings/./DockerizeWithDockerComposeLaunchSettings.csproj"
COPY . .
WORKDIR "/src/DockerizeWithDockerComposeLaunchSettings"
RUN dotnet build "./DockerizeWithDockerComposeLaunchSettings.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./DockerizeWithDockerComposeLaunchSettings.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=f

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

> Make sure not to miss the crucial step of replacing the existing '**DockerizeWithDockerComposeLaunchSettings**' value with the actual name of your project.

Docker Compose to the Rescue

Now, let's introduce Docker Compose to orchestrate multiple containers. Utilizing launch profiles in Docker Compose allows you to manage different configurations effortlessly. ■

1. In Visual Studio Solution Explorer right-click on your project, choose ***Add*** > ***Container Orchestrator Support***. The Docker Support Options dialog appears.
2. Choose ***Docker Compose*** and, for instance, select ***Linux*** as your Target OS.

Upon your selection, Visual Studio swiftly ■ works its magic, generating a ***docker-compose.yml*** file and a ***.dockerignore*** file within the docker-compose node in the solution. ■

Ensure to update both your ***docker-compose.yml*** and ***docker-compose.override.yml*** files following the example below, keeping in mind that in this instance, we are composing a Web API with a PostgreSQL database.

```
# docker-compose.yml
version: '3.9'

services:
 web-api:
   container_name: web-api
   image: web-api
   build:
     context: DockerizeWithDockerComposeLaunchSettings # replace with the actual name of your project
     dockerfile: Dockerfile
   depends_on:
     - db

 db:
   container_name: postgres-db
   image: postgres:16


# docker-compose.override.yml
version: '3.9'

services:
 web-api:
   environment:
     - ASPNETCORE_ENVIRONMENT=Development
     - ASPNETCORE_HTTP_PORTS=8080
     - ASPNETCORE_HTTP_PORTS=8081
   ports:
     - 8080:8080
     - 8081:8081
   volumes:
     - ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:ro
     - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

 db:
   environment:
     POSTGRES_DB: dev-db
     POSTGRES_USER: postgres
     POSTGRES_PASSWORD: postgres
   ports:
     - 5432:5432
```

Leveraging Compose Profiles

Profiles allow you to customize the Compose application model for different purposes and environments by selectively starting services. Each service can be assigned to one or more profiles. When unassigned, the service always starts, but if it's assigned to a profile, it starts only when that profile is selected. ■ This feature helps you to specify extra services within a single compose file, intended to start exclusively for particular scenarios like debugging or development tasks.

Managing Docker Compose Launch Settings

To configure your Docker Compose Launch settings in Visual Studio, begin by navigating to ***Debug*** > ***Manage Docker Compose Launch Settings:***

From here, you have the flexibility to decide ■ whether to run specific services by choosing **predefined profiles** as outlined in our '***docker-compose.yml***' or by selecting **actions for individual services**. Additionally, this launch profile triggers the automatic opening of a browser to the Swagger page upon application

startup.

After clicking ***Save***, the following ***launchSettings.json*** file should be generated:

```
// launchSettings.json
{
 "profiles": {
   "Docker Compose": {
     "commandName": "DockerCompose",
     "commandVersion": "1.0",
     "composeLaunchAction": "LaunchBrowser",
     "composeLaunchServiceName": "web-api",
     "composeLaunchUrl": "{Scheme}://localhost:{ServicePort}/swagger",
     "composeProfile": {
       "includes": [
         "backend"
       ]
     }
   }
 }
}
```

Facing the Music: Checking Configuration

Let’s put our work to the test, shall we? Click on the ***Debug Docker Compose*** button, and if you’ve nailed every step correctly, prepare for lift-off! ■ Your app should smoothly initiate within the Docker environment, highlighting your awesome configuration skills. ■■

> You can also build and run the app from the Terminal by executing commands from the root of your solution. Show that command line who’s boss! ■■
> > docker build -t web-api -f YOUR_PROJECT/Dockerfile .
> > docker compose --profile backend up

Conclusion

Congratulations, you’ve successfully Containerized your .NET 8 app using Docker Compose and launch settings profiles! ■ Embrace the power of containerization to simplify deployment and enhance scalability across various environments. ■

Stay tuned for more tech tips and tricks! If you found this guide helpful, don’t forget to clap, share, and spread the knowledge. Until next time, happy containerizing! ■■