 Building Umbraco 11 with dotnet 7 and running the web app in the docker container



Since its original release in March 2013, Docker has been around and has truly impacted the structure of web apps and services and how applications are deployed and scaled.

In our world of content management websites, it hasn't nearly had the same effect, and I believe we're missing out on some excellent chances to make the sites we develop faster, more effective, more scalable, and more durable. In this post, I'll walk you through how to achieve this.

**Prerequisites**

In order to run this application you will need the following installed on your machine.

* Visual Studio Code
  + There's a helpful docker extension for Visual studio code: [https://code.visualstudio.com/docs/containers/overview](https://www.blogger.com/u/1/blog/post/edit/6064076155419309550/1458604426471496119)
* WSL2
  + Windows Subsystem for Linux
* .NET SDK version 7
  + This will also work with the earlier version .NET version core, but you need a compatible SDK
* Docker Desktop
  + The best solution if you're using Windows or a Mac is Docker desktop, a free (for individuals) program offered by Docker that simply configures everything for you. It takes some effort to set up Docker on Linux. I won't go through particular instructions for each platform here, but you can find some pretty decent information on how to do it here: [https://docs.docker.com/desktop/windows/install/](https://www.blogger.com/u/1/blog/post/edit/6064076155419309550/1458604426471496119) and [https://docs.docker.com/get-docker/](https://www.blogger.com/u/1/blog/post/edit/6064076155419309550/1458604426471496119)
* Few key concepts of Docker
  + What is Docker, Docker Image, Docker Container, Container Registry, Dockerfile, and Docker Networking are a few fundamental concepts.

**Steps to follow:**

1. Start with a simple Umbraco website.
2. Make an MSSQL database instance running on Docker container.
3. Make the application container and have them communicate with one another.

**Building an Umbraco container application**

Let's start from scratch and build a brand-new standalone Umbraco application on our local machine. This Umbraco 11 site will be.NET 7 by following commands.

### Install the Umbraco templates first and then create a new solution named “MyUmbracoAppOnDocker” and then add the project to the solution, and install a starter kit

dotnet new -i Umbraco.Templates::11

dotnet new sln --name MyUmbracoAppOnDocker

dotnet new umbraco -n MyUmbracoAppOnDocker --friendly-name "Admin User" --email "admin@umbraco.com" --password "1234567890" --connection-string "Data Source=(localdb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\MyUmbracoAppOnDockerData.mdf;Integrated Security=True"

dotnet sln MyUmbracoAppOnDocker.sln add MyUmbracoAppOnDocker  
dotnet add MyUmbracoAppOnDocker package Clean

Now run the site by following command and it will create new localDB

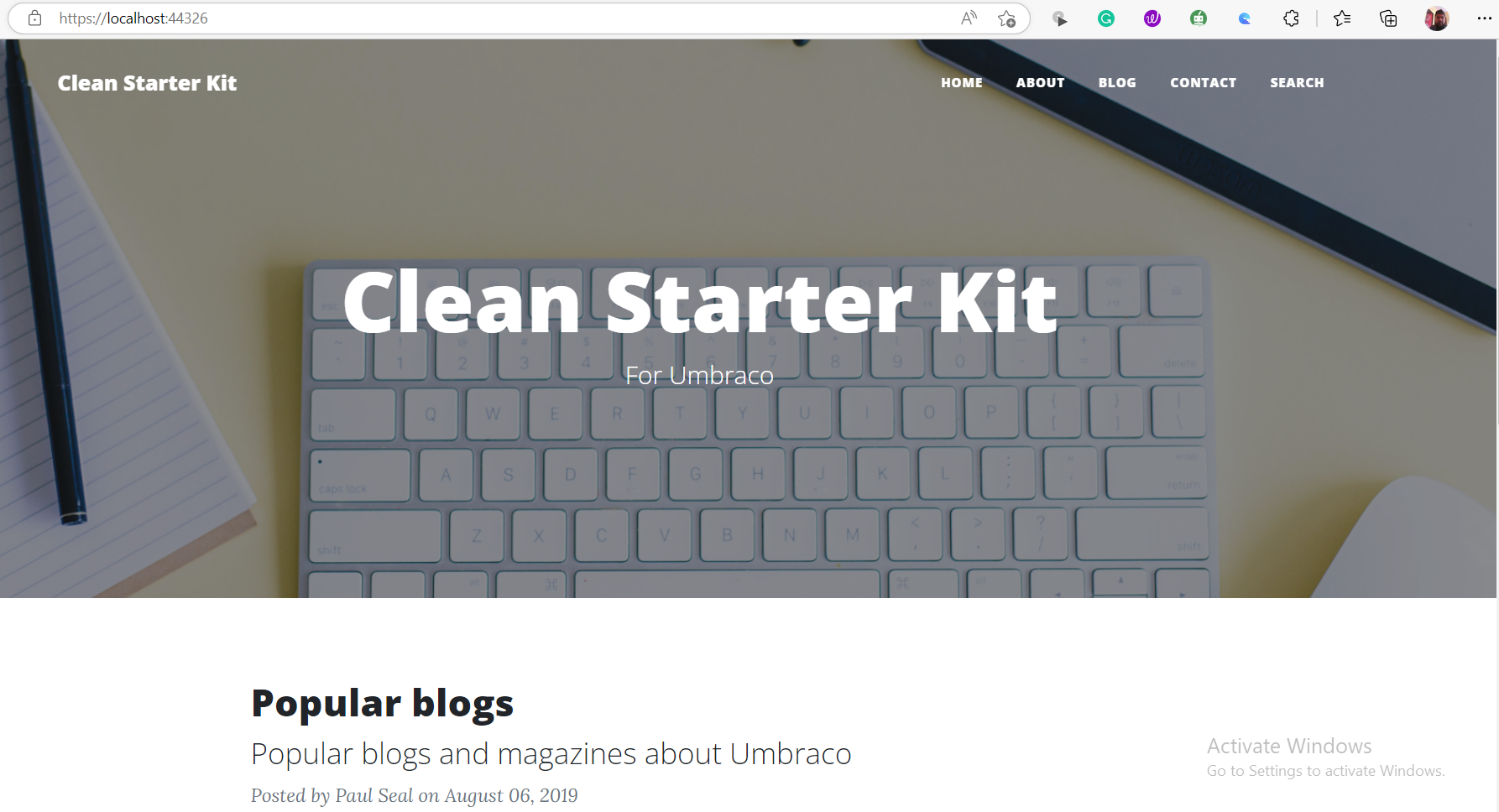
dotnet run --project MyUmbracoAppOnDocker

Text

Description automatically generated

We can see what port the site is using in the output and we can launch the website in any browser of choice. This step must be finished to build the Database.

The project is now running locally on the development workstation as an ordinary Umbraco site.



## Now it’s time to Run the database from a container.

Stop the running application before running the below command.

mkdir UmbracoData;

$currDir = Get-Location; Copy-Item -Path $currDir/MyUmbracoAppOnDocker/umbraco/Data/MyUmbracoAppOnDockerData.mdf -Destination $currDir/UmbracoData -PassThru;

$currDir = Get-Location; Copy-Item -Path $currDir/MyUmbracoAppOnDocker/umbraco/Data/MyUmbracoAppOnDockerData\_log.ldf -Destination $currDir/UmbracoData -PassThru;

**Now we will create three files in newly created folder “UmbracoData”**

1. startup.sh
2. setup.sql
3. Dockerfile

To start the MSSQL database, we will now create a bash program called Startup.sh, which we will call from a Docker file. The main job of startup.sh is to start setup.sql as the server admin (or sa) account and sleep for 15 seconds while the sql server is starting up. We have to define the password here.

#!/bin/bash

set -e

if [ "$1" = '/opt/mssql/bin/sqlservr' ]; then

# If this is the container's first run, initialize the application database

if [ ! -f /tmp/app-initialized ]; then

# Initialize the application database asynchronously in a background process. This allows a) the SQL Server process to be the main process in the container, which allows graceful shutdown and other goodies, and b) us to only start the SQL Server process once, as opposed to starting, stopping, then starting it again.

function initialize\_app\_database() {

# Wait a bit for SQL Server to start. SQL Server's process doesn't provide a clever way to check if it's up or not, and it needs to be up before we can import the application database

sleep 15s

#run the setup script to create the DB and the schema in the DB

/opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Abcd!234 -d master -i setup.sql

# Note that the container has been initialized so future starts won't wipe changes to the data

touch /tmp/app-initialized

}

initialize\_app\_database &

fi

fi

exec "$@"

The setup.sql file looks like:

USE [master]

GO

IF NOT EXISTS (SELECT \* FROM sys.databases WHERE name = 'MyUmbracoAppOnDockerDB')

BEGIN

CREATE DATABASE [MyUmbracoAppOnDockerDB] ON

( FILENAME = N'/var/opt/sqlserver/MyUmbracoAppOnDockerData.mdf' ),

( FILENAME = N'/var/opt/sqlserver/MyUmbracoAppOnDockerData\_log.ldf' )

FOR ATTACH

END;

GO

USE MyUmbracoAppOnDockerDB;

Now we will create a Dockerfile that describes the Database server that will run on Docker. The first line in Dockerfile describes the image that will be used: SQL Server 2019 that running on Ubuntu. After that we will define the password for SA account.

FROM mcr.microsoft.com/mssql/server:2019-GDR1-ubuntu-16.04

ENV ACCEPT\_EULA=Y

ENV SA\_PASSWORD=Abcd!234

ENV MSSQL\_PID=Express

USER root

RUN mkdir /var/opt/sqlserver

RUN chown mssql /var/opt/sqlserver

ENV MSSQL\_BACKUP\_DIR="/var/opt/sqlserver"

ENV MSSQL\_DATA\_DIR="/var/opt/sqlserver"

ENV MSSQL\_LOG\_DIR="/var/opt/sqlserver"

COPY setup.sql /

COPY startup.sh /

COPY MyUmbracoAppOnDockerData.mdf /var/opt/sqlserver

COPY MyUmbracoAppOnDockerData\_log.ldf /var/opt/sqlserver

ENTRYPOINT [ "/bin/bash", "startup.sh" ]

CMD [ "/opt/mssql/bin/sqlservr" ]

Go back to our Umbraco project now and change the connection string to link to this SQL server that are running on Docker. Modify the app settings. According to appsettings.Development.json

"ConnectionStrings": {

"umbracoDbDSN": "Server=localhost,1401;Database= MyUmbracoAppOnDockerDB;User Id=sa;Password=Abcd!234;"

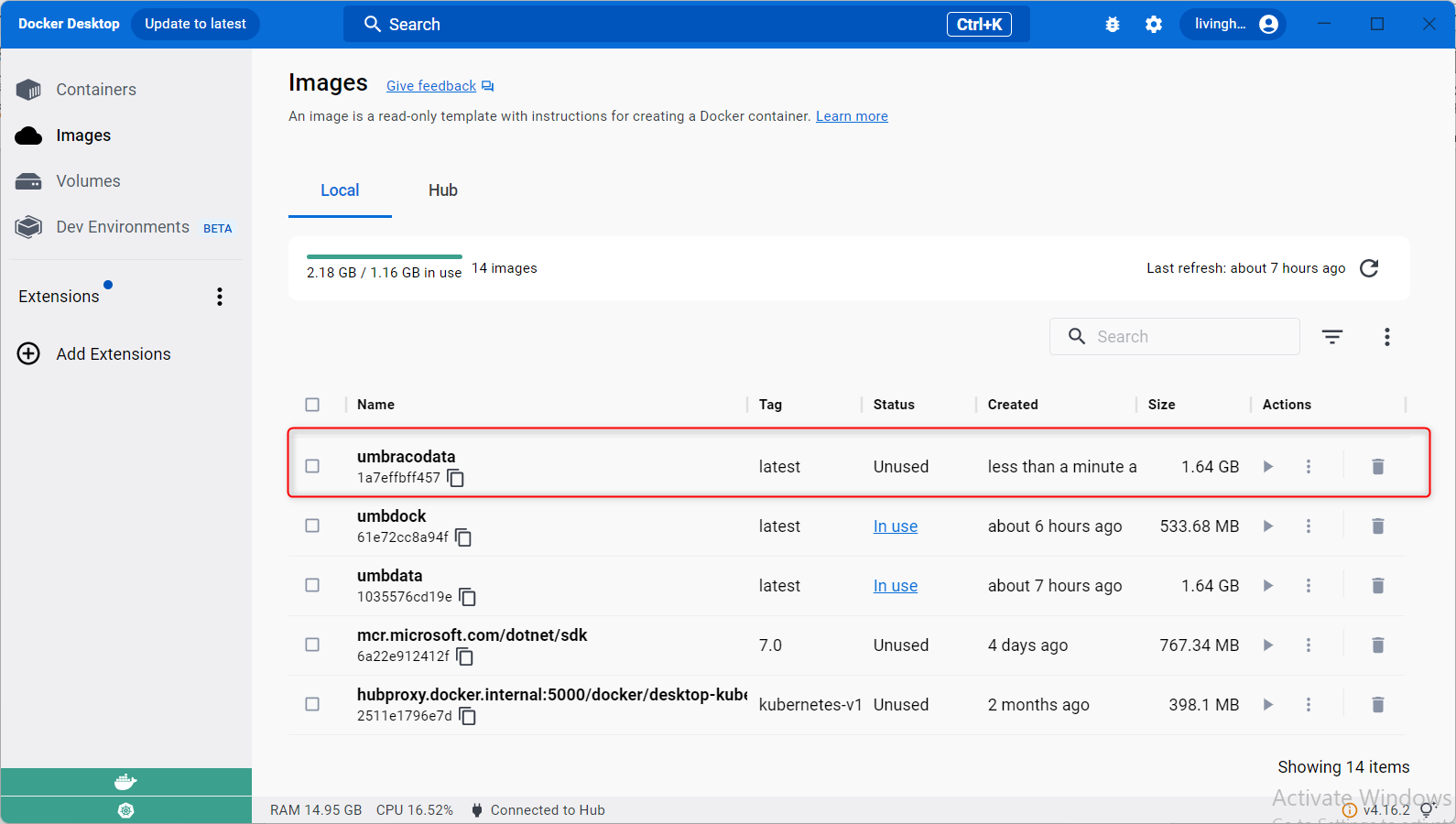
}

***Note: Because it's likely you already have MSSQL Server installed locally and using port 1433 would cause a problem, we're utilizing port 1400 instead of the standard 1433.***

Now we will create the MSSQL Docker image by using the command.

docker build --tag=umbracodata .\UmbracoData

It will create a database image in local docker host.



After creating this docker image, it’s time to run the docker SQL server by executing below command. Again, take notice of the port that is being used, while the Docker image still uses 1433 internally, 1401 is used externally to avoid conflict with any other local SQL servers.

docker run --name umbracodata -p 1401:1433 --volume sqlserver:/var/opt/sqlserver -d umbracodata

It will create a MSSQL container from the image umbracodata

Graphical user interface, text, application, email

Description automatically generated

## Verify that the website is still functional.

### Now we will run the Umbraco application locally by pointing the database serving from Docker. The command window will once again show which port should be used to access the site.

### dotnet run --project MyUmbracoAppOnDocker

## Now it’s time to create another docker file to Run the application in Docker.

### We must add a new file called Dockerfile inside the root of our web project in order to start the project in Docker.

### # Use the SDK image to build and publish the website

### FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

### WORKDIR /src

### COPY ["MyUmbracoAppOnDocker.csproj", "."]

### RUN dotnet restore "MyUmbracoAppOnDocker.csproj"

### COPY . .

### RUN dotnet publish "MyUmbracoAppOnDocker.csproj" -c Release -o /app/publish

### # Copy the published output to the final running image

### FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS final

### WORKDIR /app

### COPY --from=build /app/publish .

### ENTRYPOINT ["dotnet", "MyUmbracoAppOnDocker.dll"]

## This section will describes that most developers struggle with is the Docker Networking.

## We must have a basic understanding of Docker networking before we can run the fully functional website in a container. In Docker, containers will communicate via a variety of networking protocols. One of those is Bridge networks, which come in both default and user-defined Bridge network.

## Any container may connect with any other container on the default bridge network to which all containers are can communicate if none are specified. However, it can only do so using IP addresses, which are assigned dynamically.

## Containers can use the name of the network to interact in a user-defined bridge network, but only containers that have been explicitly added to the network can use the container name to connect with one another.

## We will create a user-defined Bridge network first and then attach the network with the previously created database container by using the container name (umbracodata). Stop the running web application if it's still active by pressing Ctrl + C.

## docker network create -d bridge umbracoNetwork

## docker network connect umbracoNetwork umbracodata

#### **Connectionstring for Application Container:**

## Create an appsettings.Staging.config file by coping the content of current appsettings.Development.json file and replace the connectionstring as below. This is necessary to use nonstandard port. Important part is Server=umbracodata where umbracodata is the name of the database container.

## "umbracoDbDSN": "Server=umbracodata;Database=MyUmbracoAppOnDockerDB;User Id=sa;Password=Abcd!234;"

## This stage we will Create the application container image.

## We can create our application image now by running the below command. Our database container is running and attached to the user-defined network umbracoNetwork.

## docker build --tag=umbracoappondocker .\MyUmbracoAppOnDocker

## This command will create Docker image of the Umbraco application. Observe that the web image is not being used, but you can see that the database image is being used since it is now operating.

## Graphical user interface, text, application, email Description automatically generated

## Now we can create a container from umbracoappondocker image for our Umbraco application on Docker container.

docker run --name umbracoappondocker -p 8000:80 -v media:/app/wwwroot/media -v logs:/app/umbraco/Logs -e ASPNETCORE\_ENVIRONMENT='Staging' --network=umbracoNetwork -d umbracoappondocker

## Graphical user interface, text, application, email Description automatically generated

## Closure

## In this tutorial, we've generated two containers and run them locally on our Docker instance, one for our Umbraco 11 application and the other for the MSSQL database server and they can communicate each other by user defined Bridge network. We've discussed networking, Docker images, and some of the prerequisites for getting things up and running.

## Happy coding!