Step-by-step Guide to Dockerizing an ASPNet Core Application

# Starting-up

## Preview

With the arrival for .Net core 1.0.0, Microsoft has provided very first official Docker container for .Net Framework and hence comes the opportunity to fully utilize the Docker’s exceptional features for easing the whole deployment paradigm. With increased support for windows, Docker itself has become more obvious choice towards simplifying enhancing the deployment capability.

Said that, there is no official support as of now for utilizing Docker for .net framework earlier than Core 1.0.0.

## Pre-requisites

Fair knowledge of [Docker containers](https://www.docker.com/what-docker) and concept behind why to use them, is primary requisite before you delve into this tutorial.

VS .net Core: It comes with VS 2015, though having VS2015 IDE is not required for this tutorial. You can go [here](https://msdn.microsoft.com/en-us/library/dn878908(v=vs.110).aspx) to learn more about .Net core

Docker Client tool: For Win7 and Windows 10.x, Docker has different versions. Both the Docker Tool-box as well as Docker Engine is supported by this tutorial. As a disclaimer, at the time of writing, I have not tested the steps for win 10.x yet (I do not see any reasons for not working, though)

Let’s begin with step 1 of tutorial, on your favorite windows machine.

## Install the Docker:

Go to following URL and follow the instructions to download and install Docker appropriate to your version of OS:

<https://docs.docker.com/toolbox/toolbox_install_windows/>

Verify the version: in Docker Command line client (CLI), run the following command to check the version:

$ Docker --version

Pay attention, it is a double hyphen before *version* in above command

The version shall be 1.12.0 or higher

# Create a Sample ASP.net core application

(Ref: [***https://stormpath.com/blog/tutorial-deploy-asp-net-core-on-linux-with-docker***](https://stormpath.com/blog/tutorial-deploy-asp-net-core-on-linux-with-docker))

## Pre-requisites

Make sure you have downloaded .Net Core tooling from official site <https://www.microsoft.com/net/core#windowsvs2015>

## Create Sample .Net Core Application

1. Open the Windows Command Prompt (as administrator) and change to your preferred directory (in my case D:\)
2. Make the new directory *aspnetCoreHelloWorld* for the application

mkdir AspNetCoreHelloWorld

1. Change directory to new folder

cd AspNetCoreHelloWorld

1. Now scaffold the new ASP .net project

dotnet new -t web

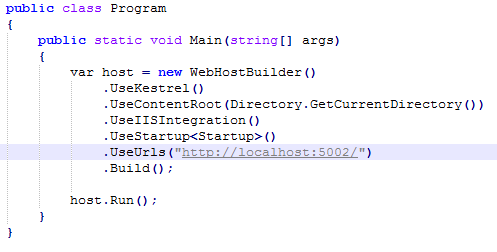
## Verify the Project locally: run the following commands

dotnet restore --- It will bring down all the libraries and dependencies required

dotnet run

By default, application shall be listening on port 5000. Try browsing with URL <http://localhost:5000>

*If you wish to change the port, please edit the program.cs by adding .UseUrls(<new URL>) followed by dotnet build and dotnet run command.*

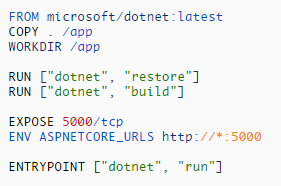


# Build a Docker Container

**(Ref:** [*https://stormpath.com/blog/tutorial-deploy-asp-net-core-on-linux-with-docker*](https://stormpath.com/blog/tutorial-deploy-asp-net-core-on-linux-with-docker))

## Define Dockerfile

1. In the folder *aspnetCoreHelloWorld*, create a new file -> save as *Dockerfile* (make sure this file has no extensions, no .txt).
2. Open the Dockerfile for edit and put the following content:



Here’s what each of these instructions does:

* **FROM:** tells Docker that you want to base your image on the existing image called *microsoft/dotnet:latest*. This image already contains all the dependencies for running .NET Core on Linux, so you don’t have to worry about setting those up.
* **COPY** and **WORKDIR** copy the current directory’s contents into a new directory inside the container called /app, and set that to the working directory for the subsequent instructions.
* **RUN** executes dotnet restore and dotnet build, which restores the packages needed to run the ASP.NET Core application and compiles the project.
* **EXPOSE** tells Docker to expose port 5000 (the default port for ASP.NET) on the container.
* **ENV** sets the environment variable ASPNETCORE\_URLS in the container. This will ensure that ASP.NET Core binds to the correct port and address.
* **ENTRYPOINT** specifies the command to execute when the container starts up. In this case, *dotnet run*.

## Create the Docker image

1. In Docker CLI, change the directory to your *Dockerfile* location as following:





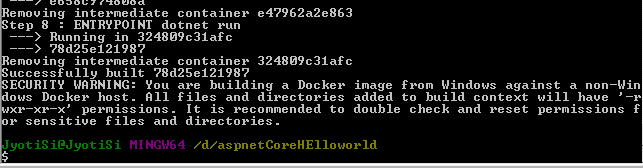
1. Now, build the Docker image as following:

$ docker build –t aspnetcore-backup .

Do not miss the trailing dot in the screenshot above. That trailing dot (.) means to look in current folder for *dockerfile*

**-t**: specifies the tag for docker image (which is expected in format <repositoryname>:<tag>). If you do not specify any tag, docker will use *latest* as *tag*

Docker build will be a multi-step process and output shall look something similar to following:



Look at the SECURITY WARNING at the end of success message. We’ll consider that when we deploy the container on Ubuntu host.

## Publish to Docker Hub

(Ref: <https://docs.docker.com/engine/getstarted/step_six/>)

1. Visit <https://hub.docker.com>
2. Sign-in to hub using your credentials
3. Create a new Repository labelled as aspnetcore-backup
4. In Docker CLI, see the images created:

$ docker images

It shall list down the image you created in previous *build* step:





1. Tag the image: using your Dockerhub userid, repository name and tag, complete the tagging as following:

$ docker tag <imageid> <userid>/<repository name>:latest

Take a look at docker images listing to confirm the successful tagging

1. Now login

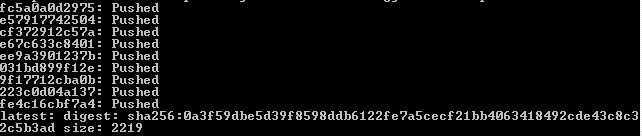
$ docker login

Provide your docker hub credentials

1. Push to docker hub:

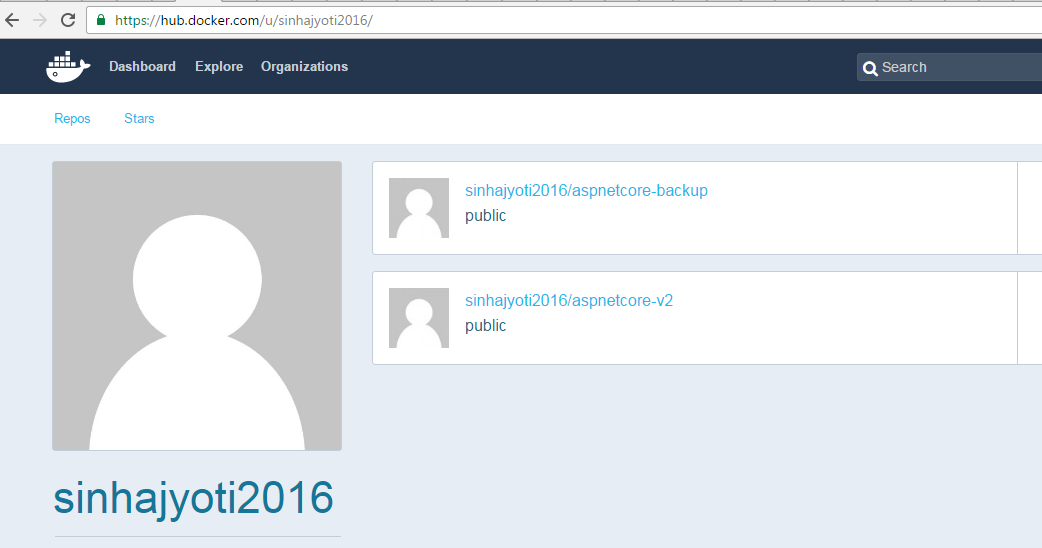
$ docker push <userid>/<repository name>

It shall output as following:



1. Verify the Push: login to <https://www.hub.docker.com>

You shall see the new repository with 1 Pull.



Your container has now been pushed to hub successfully and is available for distribution.

# Deploy the Docker Container

## On Ubuntu Box

1. Install Docker:



1. Start the Docker service:



Check the Docker version to make sure Docker is all good:



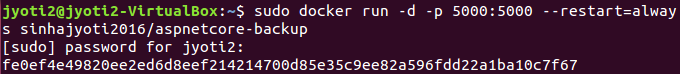
1. Make sure the user has full privileges on Docker: [when docker image is built on Windows host, by default, ‘Full permission’ is NOT granted](#SecurityWarningWin). Please use following command line to grant the minimum required permissions.
2. Now, run the Docker container (*sinhajyoti2016/aspnetcore-backup*) that’s been pushed to Docker hub



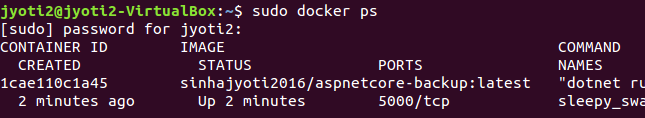
In the above example –p 5000:5000 actually maps the port (containerPort:hostPort). *Remember, the original docker container had port 5000 specified in it for running the application.*

It might ask you for password (since using *sudo*)

You should see the final output screen similar to following:



1. Get the IP address of Docker container:
2. Open another new Terminal window
3. Get the container ID using command $ docker ps



1. Now, open the *bash* console with # prompt:



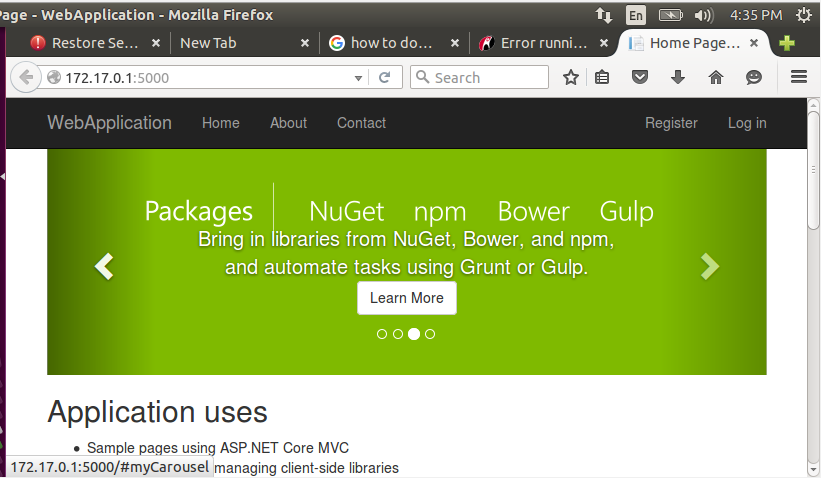
1. Now get the IP of the Docker container:



That shows that container has IP addresses in the range of 172.17.0.1 through 16. You might have different ranges available depending on Docker Daemon’s capacity to allocate IPs to each container. In current example, since, Docker container listens at port 5000 from outside, the typical endpoint to access the site should be following:

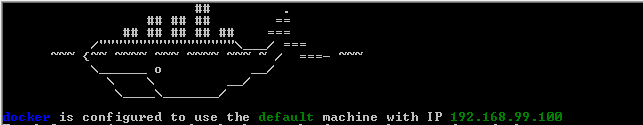
<http://172.17.0.1:5000>

1. Verify in browser:



## On Windows Box

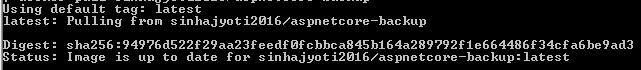
* 1. open the Docker CLI
  2. note down the default IP address as Docker CLI boots up



* 1. Pull the Docker Container to get the latest version from Docker hub



It should show up the result as following:



* 1. Now RUN the docker container



Make sure, the port mapping is done correctly (containerPort:hostPort)

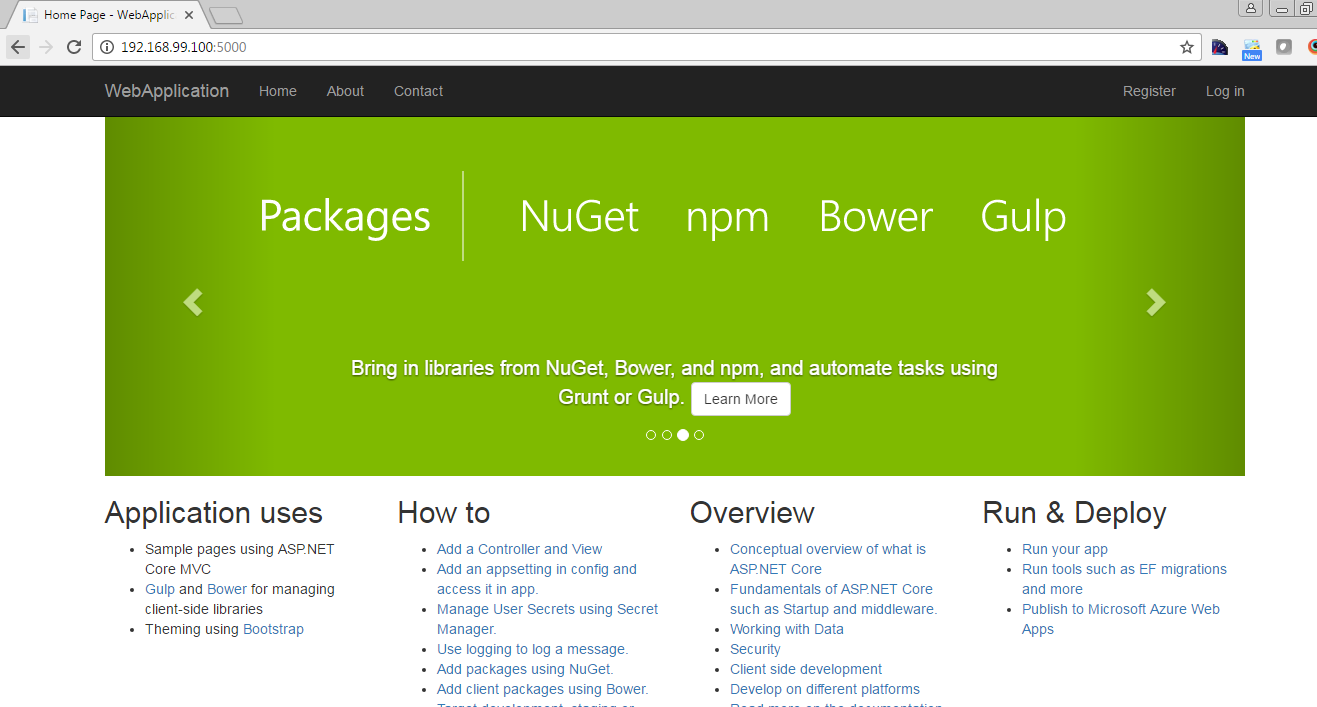
It shall show following result:



* 1. Verify in browser:

Open the browser and use the URL based on [IP address you collected above](#IPAddressWin)

Per this example, URL is <http://192.168.99.100:5000>



That’s it.