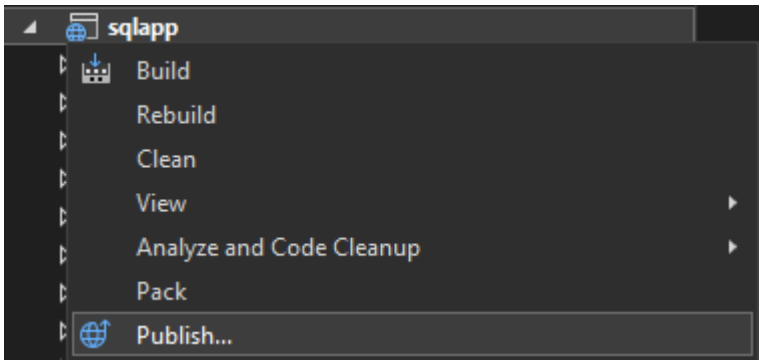
😊 LAB: CONTAINERIZE AN APPLICATION

Now, we have an application place based on .NET, which you created in the previous labs. I now want to go ahead and use the Docker tool set to Dockerize this application. For this I need the build of my application on a Docker host. I need to run some Docker based commands in order to tokenize or containerized my application. At the same time, I need also something known as a docker file. Again, all of these resources are available once you download the Docker file. This file will be available on **GitHub.**

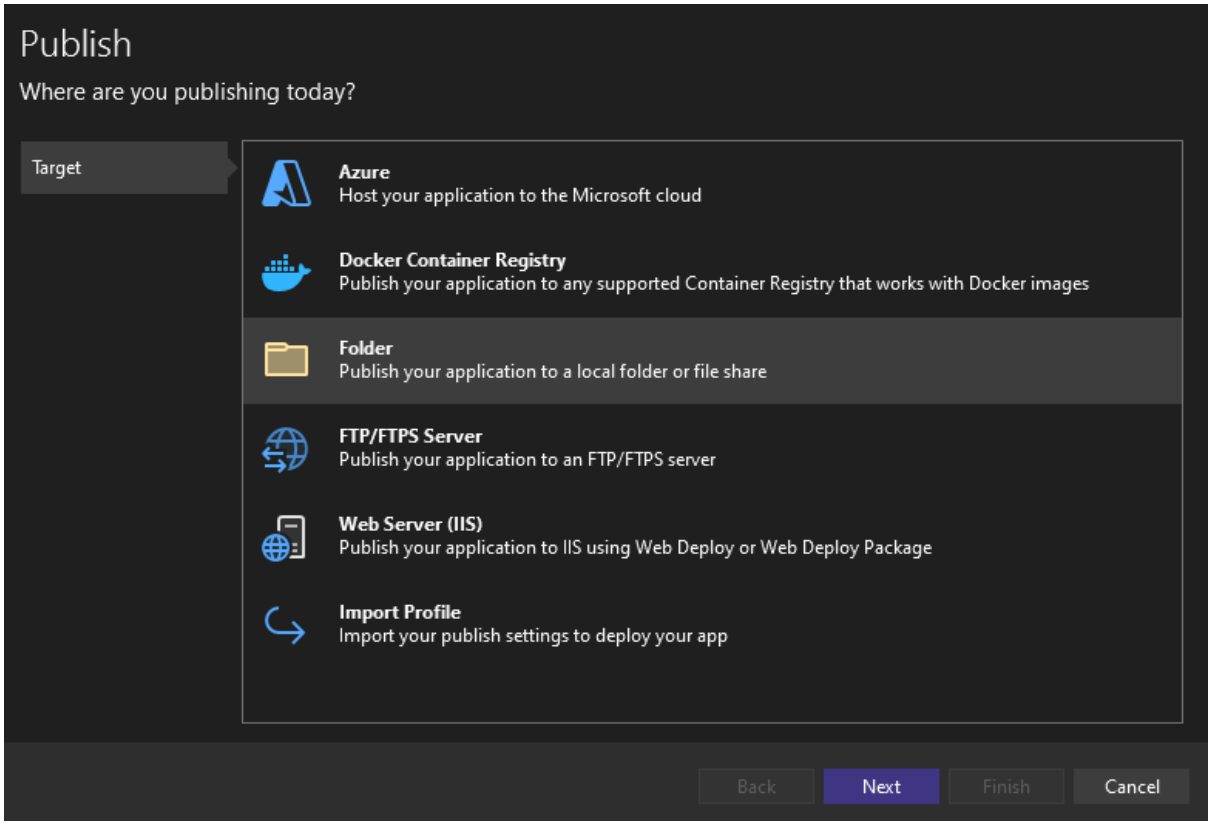
These are the commands in the file. You can download it and use it.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /app
COPY  . .
EXPOSE 80
ENTRYPOINT ["dotnet", "sqlapp.dll"]
```

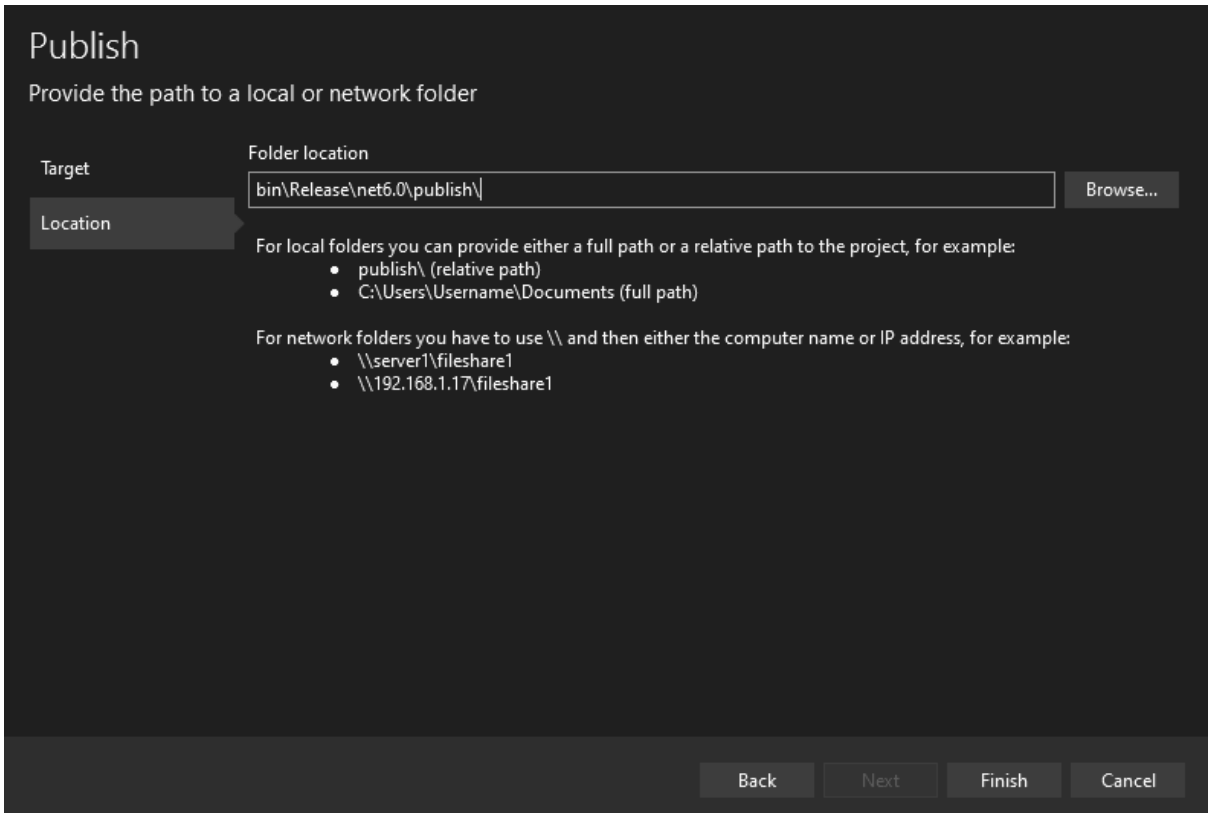
- The commands here are very simple. There is a from command which is used for the base image. So, normally for building your application image, you start from some sort of base image that is available. Here I'm using the base image, which is available from Microsoft when it comes on to hosting ASP.NET applications.
- I'm creating a working directory, copying all of my build files as my project files, exposing Port 80. And finally running my dot net application.
- We now need to build our application on our local machine. Copy that build onto our Linux VM, which we still have running, as a Docker host. We also have to copy this Docker file onto the VM as well.
- Lastly, in Visual Studio, let me do a build of my entire solution. For this what I am doing, I am publishing my entire project onto my local machine.
- To do that first you need to open Visual Studio. Then you need to right click on your application then click on publish.



- After clicking on publish, then you need to select where you want to publish it, which is in our case, it is our own local machine.
- Here you need to select folder then moves to next page.



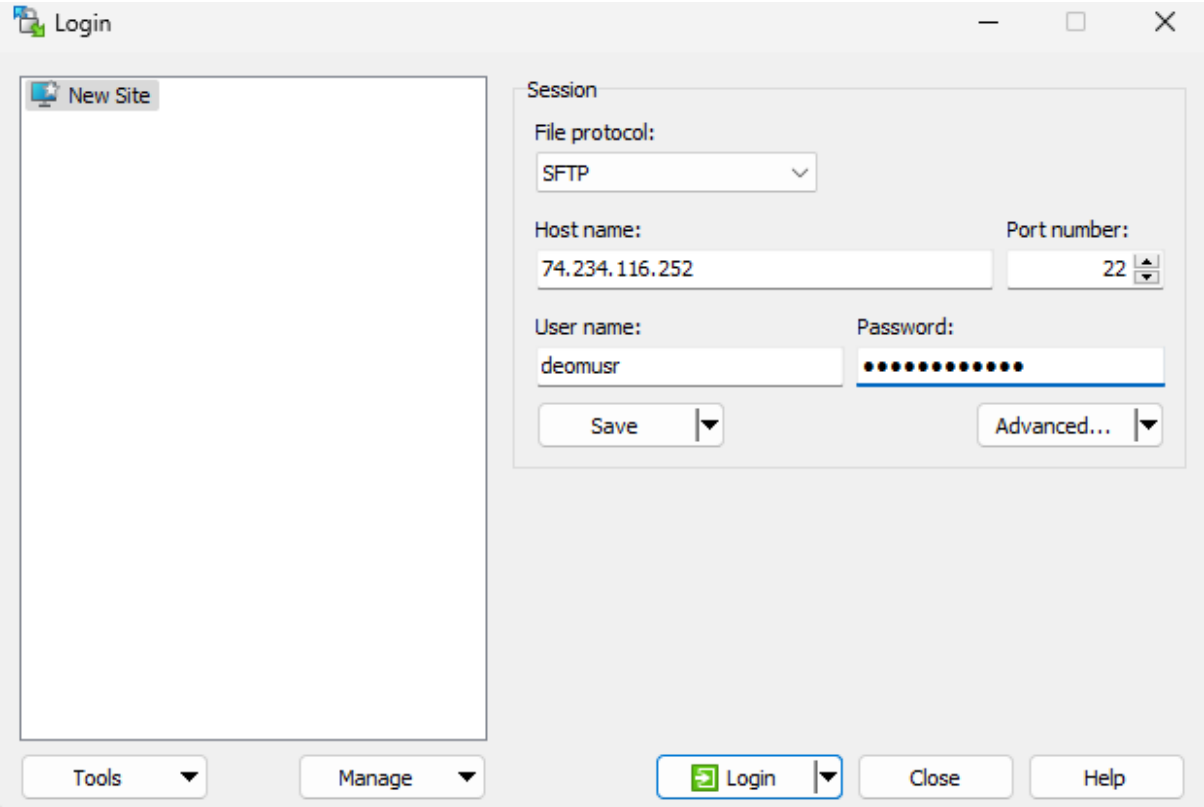
- Then it will give you a location use this location and click on finish.



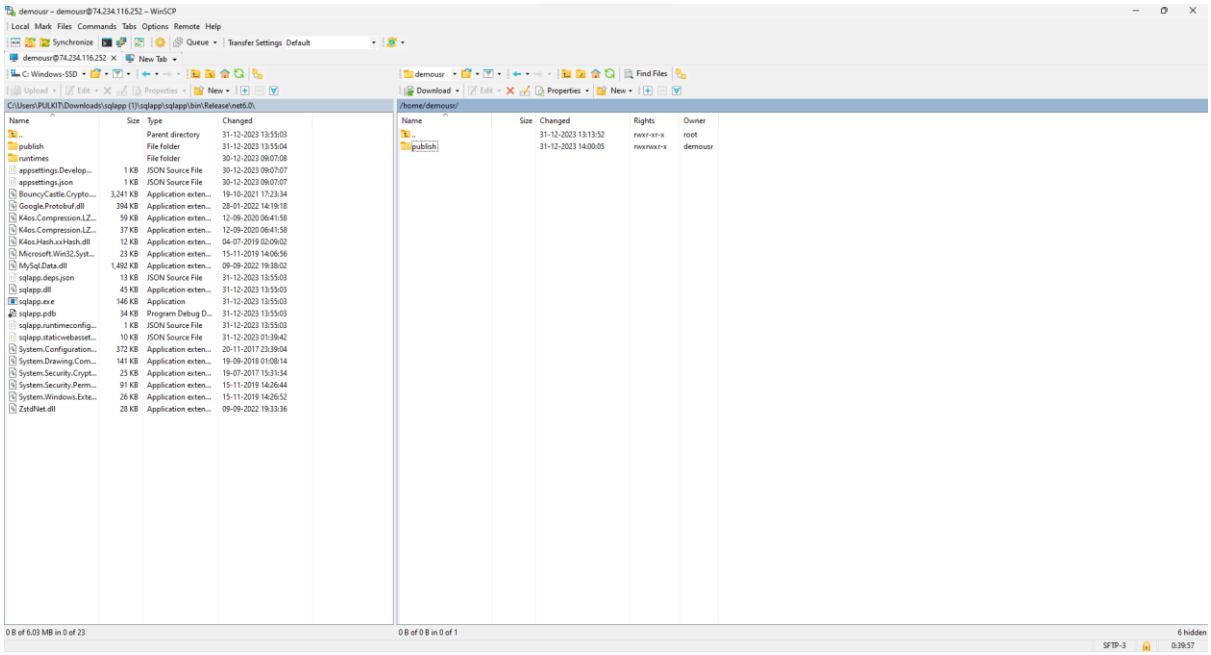
- Then you'll see that a folder has appeared, you need to click on publish then you build will be successful.


```
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 13:55 and took 01.745 seconds ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
========== Publish completed at 13:55 and took 01.745 seconds ==========
```

- Now, we need to copy this publish folder onto our Azure Virtual Machine. For that I am going to use a tool called Win SCP. This is a free tool; you can get it from internet. The link is down below use it to download Win SCP
- https://winscp.net/eng/index.php
- Once you have downloaded Win SCP tool, now open it, to log into this tool you will need your Public IP address of your Virtual Machine.
- Copy your public IP address and paste it on Win SCP. Then give your user ID and password. Now click on log in.



- Once you are in Win SCP, you will need to copy that publish folder to your VM. Now in Win SCP you will have two sides on the left-hand side you can see your local machine files from where you are going to copy the files.
- On the right-hand side, you can see that it is empty side because it your Virtual Machine Portion.
- Now you need to go to that publish folder, then drag it to right side it will start copying all the files from your machine to Virtual machine.



- Once the publish folder is copied now you need to copy the docker file to your Virtual Machine in the publish folder.
- The process is same finding the docker file and drag it to right side in the publish folder.

/home/demousr/publish/

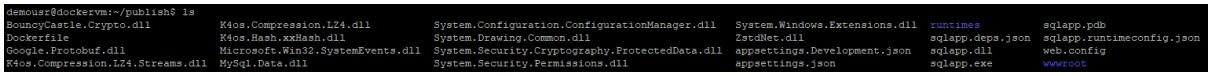
Name	Size	Changed	Rights	Owner
..		31-12-2023 13:59:03	rwxr-x---	demousr
runtimes		31-12-2023 13:59:13	rwxrwxr-x	demousr
wwwroot		31-12-2023 14:00:05	rwxrwxr-x	demousr
appsettings.Develop...	1 KB	30-12-2023 09:07:05	rw-rw-r--	demousr
appsettings.json	1 KB	30-12-2023 09:07:05	rw-rw-r--	demousr
BouncyCastle.Crypto....	3,241 KB	19-10-2021 17:23:34	rw-rw-r--	demousr
Dockerfile	1 KB	31-12-2023 01:46:36	rw-rw-r--	demousr
Google.Protobuf.dll	394 KB	28-01-2022 14:19:18	rw-rw-r--	demousr
K4os.Compression.LZ...	59 KB	12-09-2020 06:41:58	rw-rw-r--	demousr
K4os.Compression.LZ...	37 KB	12-09-2020 06:41:58	rw-rw-r--	demousr
K4os.Hash.xxHash.dll	12 KB	04-07-2019 02:09:02	rw-rw-r--	demousr
Microsoft.Win32.Syst...	23 KB	15-11-2019 14:06:56	rw-rw-r--	demousr
MySql.Data.dll	1,492 KB	09-09-2022 19:38:02	rw-rw-r--	demousr
sqlapp.deps.json	13 KB	31-12-2023 13:55:03	rw-rw-r--	demousr
sqlapp.dll	45 KB	31-12-2023 13:55:03	rw-rw-r--	demousr
sqlapp.exe	146 KB	31-12-2023 13:55:03	rw-rw-r--	demousr
sqlapp.pdb	34 KB	31-12-2023 13:55:03	rw-rw-r--	demousr
sqlapp.runtimeconfig...	1 KB	31-12-2023 13:55:03	rw-rw-r--	demousr
System.Configuration...	372 KB	20-11-2017 23:39:04	rw-rw-r--	demousr
System.Drawing.Com...	141 KB	19-09-2018 01:08:14	rw-rw-r--	demousr
System.Security.Crypt...	25 KB	19-07-2017 15:31:34	rw-rw-r--	demousr
System.Security.Perm...	91 KB	15-11-2019 14:26:44	rw-rw-r--	demousr
System.Windows.Exte...	26 KB	15-11-2019 14:26:52	rw-rw-r--	demousr
web.config	1 KB	31-12-2023 13:55:04	rw-rw-r--	demousr
ZstdNet.dll	28 KB	09-09-2022 19:33:36	rw-rw-r--	demousr

]

- Now go to your putty session where you have your virtual machine running.
- There now go to publish folder as you have copied that folder to your VM.
- For that use command **cd publish**

```
demousr@dockervm:~$ cd publish
```

- Once you are in the publish folder then list your file by writing **ls,** now you can see that multiple folders and files, here you will also see your docker file.



- I'm going to now execute a command known as sudo Docker build, this Docker Build Command will take now the Docker file and create an image with my application. It will tag my image with this particular name.
- The command its **sudo docker build -t sqlapp .**

```
demousr@dockervm:~/publish$ sudo docker build -t sqlapp .
[+] Building 10.9s (8/8) FINISHED                                                              docker:default
 => [internal] load build definition from Dockerfile                                                     0.1s
 => => transferring dockerfile: 150B                                                                     0.0s
 => [internal] load .dockerignore                                                                        0.1s
 => => transferring context: 2B                                                                          0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0                                     0.3s
 => [1/3] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:64dc2a4bc6e20d759ad80c00e2f9864d4f11c56ae66f398284cd  9.0s
 => => resolve mcr.microsoft.com/dotnet/aspnet:6.0@sha256:64dc2a4bc6e20d759ad80c00e2f9864d4f11c56ae66f398284cd  0.1s
 => => sha256:4ece0626219de4407033ldafleff6932a03a31333a6f7f2d7b8b592a2e80d5b0 14.97MB / 14.97MB          1.5s
 => => sha256:e28e0d9753cflcc188e5f3641c9d3b67c0973b081ddd8d7d06ea4b452f227d0e 31.66MB / 31.66MB          1.7s
 => => sha256:64dc2a4bc6e20d759ad80c00e2f9864d4f11c56ae66f398284cdf7c321ab31c5 1.79kB / 1.79kB            0.0s
 => => sha256:dec4f930383a2e6a17dcc6490ff03acc990739e21d5198c1b05979c6cf0f7c92 1.37kB / 1.37kB            0.0s
 => => sha256:2b988280b1e499008d1528cfb9028ad7f7fc7da65c8d16a6d116548012270153 2.36kB / 2.36kB            0.0s
 => => sha256:b5a0d5c14ba9ece1eecd5137c468d9a123372b0af2ed2c8c4446137730c90e5b 31.42MB / 31.42MB          1.4s
 => => extracting sha256:b5a0d5c14ba9ece1eecd5137c468d9a123372b0af2ed2c8c4446137730c90e5b                 2.4s
 => => sha256:7d4032af8a3c8c63bc16af843e52e9074b151a579d2387cde43f3ed454945907 9.47MB / 9.47MB            3.7s
 => => sha256:ba36662ecfbe29d327a890ff1483655267c531337f9ab6b01e492992ee12be0e 156B / 156B                1.8s
 => => extracting sha256:4ece0626219de4407033ldafleff6932a03a31333a6f7f2d7b8b592a2e80d5b0                 1.0s
 => => extracting sha256:e28e0d9753cflcc188e5f3641c9d3b67c0973b081ddd8d7d06ea4b452f227d0e                 1.5s
 => => extracting sha256:ba36662ecfbe29d327a890ff1483655267c531337f9ab6b01e492992ee12be0e                 0.0s
 => => extracting sha256:7d4032af8a3c8c63bc16af843e52e9074b151a579d2387cde43f3ed454945907                 0.4s
 => [internal] load build context                                                                        1.6s
 => => transferring context: 15.38MB                                                                     1.3s
 => [2/3] WORKDIR /app                                                                                   0.1s
 => [3/3] COPY  . .                                                                                      0.4s
 => exporting to image                                                                                   0.7s
 => => exporting layers                                                                                  0.7s
 => => writing image sha256:bcb7653f47e5424ed3955c0ce24ae84f92452a746c4d6bf18f96a6915626b1ad             0.0s
 => => naming to docker.io/library/sqlapp                                                                0.0s
```

- I'm going to stop this nginX container so that I can run my application that could listen for traffic on Port 80. Please note that there are a lot of options available when it comes onto running multiple applications via the use of Docker containers. They could be running on different ports or listening on different ports accordingly. But to make things simple, I will first issue a command to stop the nginX container from running.
- But to make things simple, I will first issue a command to stop the engine X container from running.
- **sudo docker ps**

```
demousr@dockervm:~/publish$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
  NAMES
1cff33cd7eb7   nginx     "/docker-entrypoint.…"   48 minutes ago   Up 48 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp
  mynginx
```

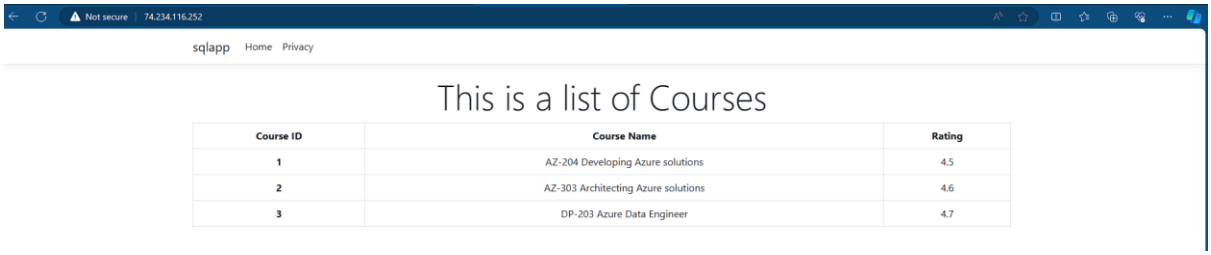
- Now to stop nginx you can use **sudo docker stop** command, following with first 4-5 digits or alphabets of your ID.

```
demousr@dockervm:~/publish$ sudo docker stop 1cff33cd7eb7
1cff33cd7eb7
demousr@dockervm:~/publish$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

- I want to now make my application run as a container that can listen for request hence on the VM, I'm issuing again the docker run command.
- **sudo docker run --name sqlapp-1 -p 80:80 -d sqlapp**

```
demousr@dockervm:~/publish$ sudo docker run --name sqlapp-1 -p 80:80 -d sqlapp
379e4f2ed310924500af0dbb511889aae95cd5abd3f74ffc50805d3cf0fa9739
```

- Once it is done then go back to your virtual machine and copy your Public IP address and paste it in a new tab.



- **You will see your application running. This application, remember, is connecting onto my SQL database. We have now looked at the process of containerizing, a dot net application. We built an image and now we're running a container out of that image**.