

# Docker

**The Docker Azure Integration enables developers to use native Docker commands to run applications in Azure Container Instances (ACI) when building cloud-native applications.**

Docker is **an open platform for developing, shipping, and running applications**. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications

- We can run multiple containers on one sever. That container can host individual applications
- All applications will be isolated in container.
- There won't be any communication between the containers.
- We can install docker on VM's or hypervisor (VM ware) or own laptop also.
- Docker engine has to install on top of VM
- Container will run on top of Docker Engineer

## Installing Docker on VM

- Install one Ubuntu VM (Open all 3 ports)
- Login to it
- Go to docker.docs to guidance to install
- Go to <https://docs.docker.com/desktop/>
- Click on install docker desktop
- Install on Linux
- Select ubuntu
- Click on Setup Dockers package repository under Install Docker Desktop
- Follow below documentation

## Install using the repository

Before you install Docker Engine for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

## Set up the repository

1. Update the `apt` package index and install packages to allow `apt` to use a repository over HTTPS:

```
2. $ sudo apt-get update
3.
4. $ sudo apt-get install \
5.     ca-certificates \
6.     curl \
7.     gnupg \
8.     lsb-release
```

9. Add Docker's official GPG key:

```
10. $ sudo mkdir -p /etc/apt/keyrings
11. $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
    -o /etc/apt/keyrings/docker.gpg
```

12. Use the following command to set up the repository:

```
13. $ echo \
14.     "deb [arch=$(dpkg --print-architecture) signed-
    by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
15.     $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
    /dev/null
```

## Install Docker Engine

1. Update the `apt` package index, and install the *latest version* of Docker Engine, containerd, and Docker Compose, or go to the next step to install a specific version:
2. `$ sudo apt-get update`
3. `$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin`

### Receiving a GPG error when running `apt-get update`?

Your default umask may not be set correctly, causing the public key file for the repo to not be detected. Run the following command and then try to update your repo again: `sudo chmod a+r /etc/apt/keyrings/docker.gpg`.

- The above-mentioned commands will install the docker
- To check docker installed or not run below command  
CMD: `docker version`

- To work as administrator in Linux we need to change demo user to root user use below command
  - CMD: `sudo -i`
- Note: when we are working as root user we don't need to enter sudo. The commands which required sudo, no need to give that.
- To check what are the Images are in the docker run below command
  - CMD: `docker images ls`

## Installing NGINX Image

- Go to <https://hub.docker.com/>
- Click on explore
- Click on NGINX
- Copy the command and paste in putty, it will install latest nginx version
  - CMD: `docker pull nginx`
- To check the image pulled or not run below command NGINX will show
  - CMD: `docker images`
- Note: now we have pulled latest version of nginx, if you don't want to pull latest version and you want a particular version from docker hub we need to run below command, before that check what we need by going docker hub
  - Click on tags under nginx
  - As below image we can see perl version



- CMD: `docker pull nginx:perl`
- Now it will install perl version of the nginx
- To check that again run below command
  - CMD: `docker images`
- Now as per below image it will show below 2 images

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	perl	1e9483bb852b	9 days ago	188MB
nginx	latest	2d389e545974	9 days ago	142MB
root@docker-vm:~#				

- If you want to remove any Image, for now we need to delete the perl version, run below command
  - CMD: `docker image rm nginx:perl`
- Now check whether it has deleted or not by below command it will only show latest version of nginx as per below image
  - CMD: `docker images`

```

REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    2d389e545974   9 days ago    142MB
root@docker-vm:~#

```

- Now check whether any container running or not by any one command or below
  - CMD: `docker ps`
  - Or
  - CMD: `docker container ls`
- Now run below command
  - CMS: `docker run nginx`
- The above command will start the worker process as below those are running in container

```

root@docker-vm:~# docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2022/09/22 10:18:54 [notice] 1#1: using the "epoll" event method
2022/09/22 10:18:54 [notice] 1#1: nginx/1.23.1
2022/09/22 10:18:54 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2022/09/22 10:18:54 [notice] 1#1: OS: Linux 5.15.0-1020-azure
2022/09/22 10:18:54 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2022/09/22 10:18:54 [notice] 1#1: start worker processes
2022/09/22 10:18:54 [notice] 1#1: start worker process 32
2022/09/22 10:18:54 [notice] 1#1: start worker process 33

```

- To exit form that give
  - CMD: `ctr+C`
- The above command will kill the container. Because its killed the container is running on terminal.
- To run the container in backend run below command
  - CMD: `docker run -d nginx`
- The above command will run the container in detached mode
- Stop container by using below command, we can use first 4 digits of container ID or container name.

- Look at below screenshot we have removed both containers, in 2 different ways.

```
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
02ec192ca237   nginx    "/docker-entrypoint..." 2 minutes ago  Up 2 minutes  80/tcp       reverent_benz
237f24c74fed   nginx    "/docker-entrypoint..." 2 hours ago    Up 2 hours    80/tcp       amazing_hofstadter
root@docker-vm:~# docker stop 02ec
02ec
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
237f24c74fed   nginx    "/docker-entrypoint..." 2 hours ago    Up 2 hours    80/tcp       amazing_hofstadter
root@docker-vm:~# docker stop amazing_hofstadter
amazing_hofstadter
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
root@docker-vm:~#
```

- As we can see no container running above. To check what are the containers stopped. Any one of below 2 commands will work

```
root@docker-vm:~# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
root@docker-vm:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
02ec192ca237   nginx    "/docker-entrypoint..." 11 minutes ago Exited (0) 4 minutes ago  80/tcp       reverent_benz
e3589e0437e0   nginx    "/docker-entrypoint..." 13 minutes ago Exited (0) 13 minutes ago  80/tcp       clever_blackwell
c98547b7f0e6   nginx    "/docker-entrypoint..." 16 minutes ago Exited (0) 16 minutes ago  80/tcp       keen_solomon
237f24c74fed   nginx    "/docker-entrypoint..." 2 hours ago    Exited (0) 3 minutes ago  80/tcp       amazing_hofstadter
root@docker-vm:~# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
02ec192ca237   nginx    "/docker-entrypoint..." 12 minutes ago Exited (0) 6 minutes ago  80/tcp       reverent_benz
e3589e0437e0   nginx    "/docker-entrypoint..." 15 minutes ago Exited (0) 15 minutes ago  80/tcp       clever_blackwell
c98547b7f0e6   nginx    "/docker-entrypoint..." 18 minutes ago Exited (0) 18 minutes ago  80/tcp       keen_solomon
237f24c74fed   nginx    "/docker-entrypoint..." 2 hours ago    Exited (0) 5 minutes ago  80/tcp       amazing_hofstadter
root@docker-vm:~#
```

- The above commands will show stopped containers list
- No I want to remove all the stopped containers we need to give command like "docker rm (id of container 1 2 3) like below screenshot
- Now we can see no container is running in the background

```
root@docker-vm:~# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
02ec192ca237   nginx    "/docker-entrypoint..." 12 minutes ago Exited (0) 6 minutes ago  80/tcp       reverent_benz
e3589e0437e0   nginx    "/docker-entrypoint..." 15 minutes ago Exited (0) 15 minutes ago  80/tcp       clever_blackwell
c98547b7f0e6   nginx    "/docker-entrypoint..." 18 minutes ago Exited (0) 18 minutes ago  80/tcp       keen_solomon
237f24c74fed   nginx    "/docker-entrypoint..." 2 hours ago    Exited (0) 5 minutes ago  80/tcp       amazing_hofstadter
root@docker-vm:~# docker rm 02ec192ca237 e3589e0437e0 c98547b7f0e6 237f24c74fed
02ec192ca237
e3589e0437e0
c98547b7f0e6
237f24c74fed
root@docker-vm:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
root@docker-vm:~#
```

- Now I want to run container and give it to name by below command
  - CMD: docker run -d --name nginxcontainer nginx

```
root@docker-vm:~# docker run -d --name nginxcontainer nginx
ab59fcdde00fb510249b324c735a2964c420fac452cd083b144a83cb75e2fffa
root@docker-vm:~#
```

- Now check our container name by below command it will give name as nginxcontainer
  - CMD: docker container ls

```
root@docker-vm:~# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
ab59fcdde00f   nginx    "/docker-entrypoint..." 4 minutes ago  Up 4 minutes  80/tcp       nginxcontainer
root@docker-vm:~#
```

- To inspect how our container running, give below command it will give json file it shows when it has started and all
  - CMD: docker container inspect <container ID>

```
root@docker-vm:~# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
ab59fcdde00f   nginx    "/docker-entrypoint..." 4 minutes ago  Up 4 minutes  80/tcp       nginxcontainer
root@docker-vm:~# docker container inspect ab59fcdde00f
[
```

- In that json file we can see that docker has assigned the private IP or internal IP to communicate. As below screenshot

```
"Gateway": "172.17.0.1",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "172.17.0.2",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"MacAddress": "02:42:ac:11:00:02",
"Networks": {
  "bridge": {
```

- Now if we copy paste the IP in the browser, we can't able to reach out to nginx we page, because the IP is private.
- To reach nginx page we need divert traffic which coming on host (VM) public IP port 80 it should divert to container which is running the nginx
- To do that we need to remove our running container, run below command
  - CMD: docker rm -f <container ID>

```
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND
ab59fcdde00f   nginx    "/docker-entrypoint
root@docker-vm:~# docker rm -f ab59
ab59
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED
root@docker-vm:~#
```

- Now we can see container is removed
- Now run below command to divert port 80 to container by below command
  - CMD: docker run -d -p 80:80 --name nginxcontainer nginx

- We can see it has created the container again

```
root@docker-vm:~# docker run -d -p 80:80 --name nginxcontainer nginx
589b2d9237ff73d5dccc25de9a0c9d35bb45083bc5886c3cd84284ac4786bc732
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                               NAMES
589b2d9237ff   nginx     "/docker-entrypoint..." About a minute ago Up About a minute 0.0.0.0:80->80/tcp, :::80->80/tcp   nginxcontainer
root@docker-vm:~#
```

- Now go to browser and search with <VM public IP:80>
- We can reach to our nginx container

## Running multiple container and port diverting

- Earlier we have one container running the nginx Image
- Now we can install one more Image of "httpd"
- Below is the command to pull the "httpd" image
  - CMD: docker pull httpd
- Now we can see below httpd image installed

```
root@docker-vm:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
httpd         latest   f2789344c573   9 days ago    145MB
nginx         latest   2d389e545974   9 days ago    142MB
root@docker-vm:~#
```

- We have used port 80 to nginx and now we can't use the same port to httpd. So we need to map httpd to other port is 8080 by below command
  - CMD: docker run -d -p 8080:80 --name httpdcontainer httpd

```
root@docker-vm:~# docker run -d -p 8080:80 --name httpdcontainer httpd
c9c74a51da75c4857f0220bde1778dc9alcaef1510f74b7afc38c06823935e8
root@docker-vm:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                               NAMES
c9c74a51da75   httpd     "httpd-foreground"      57 seconds ago Up 55 seconds 0.0.0.0:8080->80/tcp, :::8080->80/tcp
589b2d9237ff   nginx     "/docker-entrypoint..." 24 minutes ago Up 24 minutes 0.0.0.0:80->80/tcp, :::80->80/tcp
root@docker-vm:~#
```

- Now add one more in bound rule in VM in portal port:8080
- Destination port select 8080 and add
- Now go to portal and search VM IP:8080, then we can reach to httpd server
- If you want to run one more container we need to open port 9090
- Like that we can map multiple containers like that
- Now if you want to stop nginx run below command
  - CMD: docker stop <container id>

```

CONTAINER ID   IMAGE     COMMAND
c9c74a51da75   httpd     "httpd-foreground"
589b2d9237ff   nginx     "/docker-entrypoint...."
root@docker-vm:~# docker stop 589b
589b
root@docker-vm:~# █

```

- Now go and search VM IP:80 we can't able to reach nginx. Because it's stopped. But httpd still works.
- Now to start nginx container run below command
  - CMD: docker start <container id>

```

root@docker-vm:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND
ES
c9c74a51da75   httpd     "httpd-foreground"
pdcontainer
589b2d9237ff   nginx     "/docker-entrypoint...."
nxcontainer
root@docker-vm:~# docker start 589b2d9237ff
589b2d9237ff
root@docker-vm:~# █

```

- Now search nginx in portal by VM IP:80 we can reach nginx web page
- Note: we can install same image in multiple containers
- Note: we can't use same port to multiple containers

Now If we want to go to NGINX folder in container as like root folder in IIS server. We need to run below commands to go to directory.

- Below command will go to container
  - CMD: docker container exec -it nginxcontainer bash

```

root@docker-vm:~# docker container exec -it nginxcontainer bash
root@cd89964e9469:/# █

```

- Now we are in container

Note: in above command we have used bash command, bash already pre-installed in linux.

- To go to folder of nginx run below command
  - CMD: cd /usr/share/nginx/html

```

root@cd89964e9469:/# cd /usr/share/nginx/html
root@cd89964e9469:/usr/share/nginx/html# █

```

- As per above screenshot we are in nginx user folder
- To check what are the files saved in that folder run below command
  - CMD: ls



```
root@cd89964e9469:/usr/share/nginx/html# ls
50x.html  index.html
```

- As per above pic we can see index.html file
- To open and see the content in the "index.html"
  - CMD: cat index.html
- As per below snapshot html page content will show

```
root@cd89964e9469:/usr/share/nginx/html# cat index.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@cd89964e9469:/usr/share/nginx/html#
```

- If you want to exit from container, go to root run below command
  - CMD: exit
- We can see that we are now in root.

```
root@cd89964e9469:/usr/share/nginx/html# exit
exit
root@docker-vm:~#
```

- If you exit from root it will go to demo user like below

```
root@docker-vm:~# exit
logout
demouser@docker-vm:~$
```

- If you exit from here, we will get logged out from VM

- While your in demo user if you want to check container, we need to ensure to give sudo
  - CMD: sudo docker ps



```
demouser@docker-vm:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
cd89964e9469   nginx    "/docker-entrypoint...."  2 hours ago   Up 2 hours
demouser@docker-vm:~$
```

## Creating Customized Image

- Now download html template from internet and extract the file
- Copy only html file and name it as "myhtml"
- Create one more text file and write below code. While saving give name as it is <"Dockerfile">
- The content in the Docker file would be as per below.

```
FROM nginx:latest
Add myhtml /usr/share/nginx/html/
```

- Files must be below

Name	Date modified	Type	Size
 myhtml	8/16/2019 9:41 PM	File folder	
 Dockerfile	9/27/2022 1:37 PM	File	1 KB

- Now to copy the both files in our docker VM we need to use FTP protocol by WINSCP on local machine
- Now open WINSCP
  - Select file protocol as : SCP
  - Host name : VM public IP
  - Give user name and password
  - Click on login
- Now copy paste like below and using drag and drop
- Once copy task done go to linux vm and check files using below command, ensure to be in demouser
  - CMD: ls

```
demouser@docker-vm:~$ ls
Dockerfile  myhtml
demouser@docker-vm:~$
```

- We can see both files in that
- Now run below command

- CMD: `sudo docker image build -t customimage .`

```
demouser@docker-vm:~$ sudo docker image build -t customimage .
Sending build context to Docker daemon 7.149MB
Step 1/2 : FROM nginx:latest
--> 2d389e545974
Step 2/2 : Add myhtml /usr/share/nginx/html/
--> 61d600cecl4e
Successfully built 61d600cecl4e
Successfully tagged customimage:latest
demouser@docker-vm:~$
```

- Now we have created custom image
- Now we can see below created custom image

```
demouser@docker-vm:~$ sudo docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
customimage     latest      61d600cecl4e  2 minutes ago  149MB
nginx           latest      2d389e545974  2 weeks ago   142MB
demouser@docker-vm:~$
```

- To run the custom image run below command
  - CMD: `sudo docker run --name customcontainer -d -p 9090:80 customimage`

```
demouser@docker-vm:~$ sudo docker run --name customcontainer -d -p 9090:80 customimage
72b828e35655578ee39517832c12bb07e432ab185027808e1b41b45090475446
demouser@docker-vm:~$
```

- Now go to Azure portal and docker-vm
- Create new inbound rule for port : 9090
- Now go to browser and search with <vm ip:9090>
- Then we can see the custom image portal

## Stopping, starting and creating the docker containers

- To stop running container run below command
  - CMD: `sudo docker stop <container name or id>`

```
demouser@docker-vm:~$ sudo docker stop 40dcc5010ad7
40dcc5010ad7
demouser@docker-vm:~$ sudo docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
40dcc5010ad7   nginx     "/docker-entrypoint...." About a minute ago Exited (0) 13 seconds ago
72b828e35655   customimage "/docker-entrypoint...." 33 minutes ago   Up 33 minutes
0->80/tcp, :::9090->80/tcp customcontainer
demouser@docker-vm:~$ sudo docker start 40dcc5010ad7
40dcc5010ad7
```

- To start exited or stopped container
  - CMD: `sudo docker start <container name or id>`

```
demouser@docker-vm:~$ sudo docker start 40dcc5010ad7
40dcc5010ad7
demouser@docker-vm:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
40dcc5010ad7	nginx	"/docker-entrypoint..."	2 minutes ago	Up 7 seconds
72b828e35655	customimage	"/docker-entrypoint..."	34 minutes ago	Up 34 minutes
90->80/tcp	customcontainer			

```
demouser@docker-vm:~$
```

- To create new container
  - CMD: `sudo docker run -d --name <container name> nginx`
- If you want to add port mapping on the container run below command
  - CMD: `sudo docker run -d --name newnginx -p 8080:80 nginx`

# Creating demo Webapp for docker

- We need to create new ASP.Net core webapp in Visual studio 2022.
- Give name as “demodockerwebapp”
- Go to project directory in file manager and type “CMD” in address bar and hit enter.
- Now we need to publish the code in command prompt, using below command
  - CMD: dotnet publish
- The above command will create the dll files in project  
“dockerwebapp\dockerwebapp\bin\Debug\netcoreapp3.1\publish”
- Now open browser and search as “docker file for dotnet core”
- Select the same as below

## Dockerize an ASP.NET Core application

- Now we need to follow the Method 2
- Now copy below script and open notepad and paste the code there.

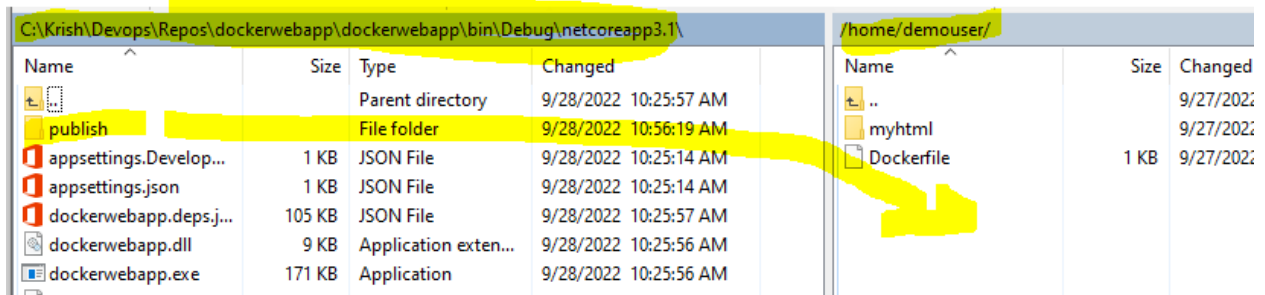
```
CMD: # syntax=docker/dockerfile:1
FROM mcr.microsoft.com/dotnet/aspnet:5.0
COPY bin/Release/netcoreapp3.1/publish/ App/
WORKDIR /App
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

- Edit the code as below

```
# syntax=docker/dockerfile:1
FROM mcr.microsoft.com/dotnet/core/sdk:3.1
COPY . App/
WORKDIR /App
ENTRYPOINT ["dotnet", "dockerwebapp.dll"]
```

- Save as the code save in our publish folder in project and name it as <“Dockerfile”> and document type must be all files.
- Now open WINSCP application and login to our VM which hosting docker using scp protocol.
- Now in the winscp go to below path  
“C:\Krish\Devops\Repos\dockerwebapp\dockerwebapp\bin\Debug\netcoreapp3.1”

- Drag and drop publish folder in vm as below.



- It will copy the code in VM.
- Now go to VM and check whether the file copied or not using below command
  - CMD: `ls`

```
demouser@docker-vm:~$ ls
Dockerfile  myhtml  publish
demouser@docker-vm:~$
```

- Check the Dockerfile using below commands

```
demouser@docker-vm:~$ ls
Dockerfile  myhtml  publish
demouser@docker-vm:~$ cd publish
demouser@docker-vm:~/publish$ ls
Dockerfile          dockerwebapp.Views.dll  dockerwebapp.dll  dockerweba
appsettings.Development.json  dockerwebapp.Views.pdb  dockerwebapp.exe  web.config
appsettings.json             dockerwebapp.deps.json  dockerwebapp.pdb  wwwroot
demouser@docker-vm:~/publish$ cat Dockerfile
# syntax=docker/dockerfile:1
FROM mcr.microsoft.com/dotnet/aspnet:5.0
COPY . App/
WORKDIR /App
ENTRYPOINT ["dotnet", "dockerwebapp.dll"]
demouser@docker-vm:~/publish$
```

- Now run below command
  - CMD: `sudo docker image build -t netimagecustom .`

- The above command will pull the code as below

```
demouser@docker-vm:~/publish$ sudo docker image build -t netimagecustom .
Sending build context to Docker daemon 4.773MB
Step 1/4 : FROM mcr.microsoft.com/dotnet/aspnet:5.0
5.0: Pulling from dotnet/aspnet
clad9731b2c7: Pull complete
169ba0027942: Pull complete
c4c86b92f556: Pull complete
e76245086e24: Pull complete
0bf07af7e5b6: Pull complete
Digest: sha256:1a7d811242f001673d5d25283b3af03da526delee8d3bb5aa295f480b7844d44
Status: Downloaded newer image for mcr.microsoft.com/dotnet/aspnet:5.0
---> 29delb9e96c0
Step 2/4 : COPY . App/
---> 58fc0ff61db5
Step 3/4 : WORKDIR /App
---> Running in cadf38595d94
Removing intermediate container cadf38595d94
---> fc3c7e88027f
Step 4/4 : ENTRYPOINT ["dotnet", "dockerwebapp.dll"]
---> Running in 01bc246783ad
Removing intermediate container 01bc246783ad
---> 4e987d0a8e5c
Successfully built 4e987d0a8e5c
Successfully tagged netimagecustom:latest
demouser@docker-vm:~/publish$
```

- Now run below command
  - CMD: sudo docker images
- We can see below our custom image has created

```
demouser@docker-vm:~/publish$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
netimagecustom	latest	4e987d0a8e5c	3 minutes ago	210MB
customimage	latest	61d600cecl4e	21 hours ago	149MB
nginx	latest	2d389e545974	2 weeks ago	142MB
mcr.microsoft.com/dotnet/aspnet	5.0	29delb9e96c0	4 months ago	205MB

```
demouser@docker-vm:~/publish$
```

- Now run command and map port using below command
  - CMD: sudo docker run --name netcontainer -p 9091:80 -d netimagecustom

```
demouser@docker-vm:~/publish$ sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
netimagecustom	latest	e15deb07055d	18 seconds ago	719MB
customimage	latest	61d600cecl4e	21 hours ago	149MB
mcr.microsoft.com/dotnet/core/sdk	3.1	0c5d2de892f8	2 weeks ago	714MB
nginx	latest	2d389e545974	2 weeks ago	142MB
mcr.microsoft.com/dotnet/aspnet	5.0	29delb9e96c0	4 months ago	205MB

```
demouser@docker-vm:~/publish$ sudo docker run --name netcontainer -p 9091:80 -d netimagecustom
6a9b3f8d6a09ba39d6ebd6e93e4f8fc5bb1afa8143ab396b26cflc4a485d6f31
demouser@docker-vm:~/publish$
```

- The above command will create the container and run the container and map it to port:9090
- Now add inbound rule for port 9091 in VM

- Now open browser and search with VM IP:9091 , we can able to see our custom website


### Storing our Custom image:

We have 2 ways to save the images

1. Docker hub (register and store)
2. Container registry in Azure

### Container Registry:

- Now go to portal and search as “container registry”



## Create container registry ...

#### Project details

Subscription \*

Free Trial

Resource group \*

(New) container-rg

Create new

#### Instance details

Registry name \*

demoregistry322


.azurecr.io

Location \*

Central India

Availability zones ⓘ

☐ Enabled


 Availability zones are enabled on premium registries and in regions that support availability zones. [Learn more](#)

SKU \* ⓘ

Basic

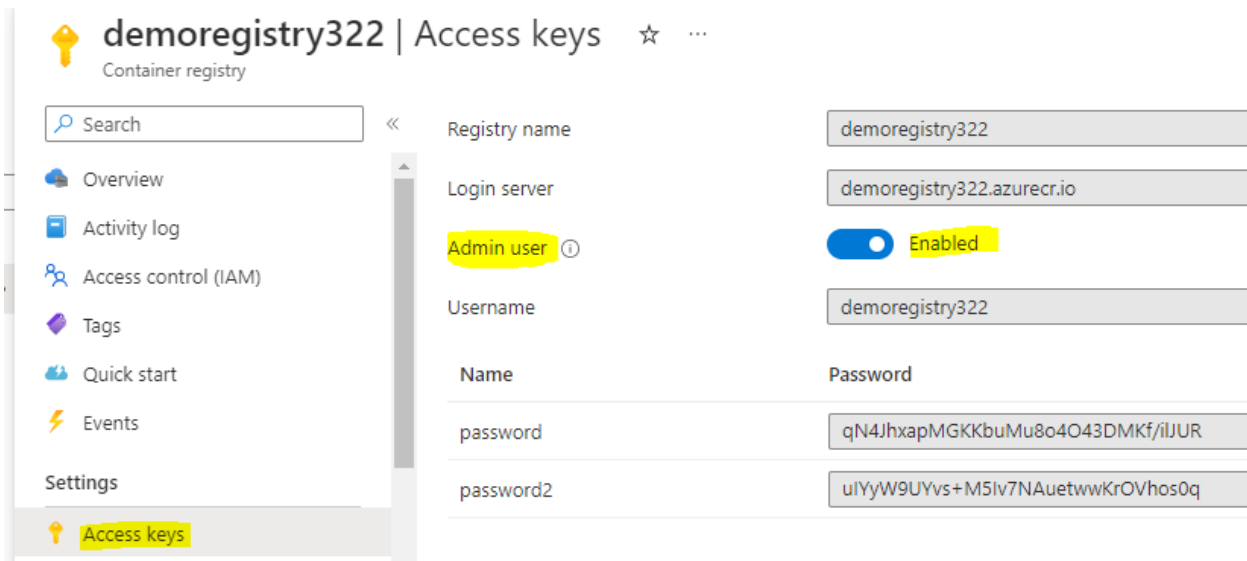
Review + create

< Previous

Next: Networking >

- Click on review and create.
- Now go to container registry and
  - Access keys from blade
  - Enable admin user





- Before pushing our image to Azure container registry we need to install Azure CLI in Linux VM
- Go to browser and search as “install azure cli on ubuntu”

<https://learn.microsoft.com> > Learn > Azure > Azure CLI

## Install the Azure CLI on Linux - Microsoft Learn

[About featured snippets](#)

- Click on the top
- Scroll down and copy below command
  - CMD: `curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash`

## Option 1: Install with one command

The Azure CLI team maintains a script to run all installation commands in one step. This script is downloaded via `curl` and piped directly to `bash` to install the CLI.

If you wish to inspect the contents of the script yourself before executing, simply download the script first using `curl` and inspect it in your favorite text editor.

```
Bash Copy  
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

- The above command will install the azure cli on ubuntu.
- Now to login to azure run below command
  - CMD: `sudo az login`
- Once we run above command it will show below

```
demouser@docker-vm:~/publish$ sudo az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code
HET4F9XGS to authenticate.
```

- Now go to our local machine browser and search as <https://microsoft.com/devicelogin>
- Now we need to copy paste the code from linux to portal
- Copy above code : HET4F9XGS
- Once we paste and login it will login azure in linux as below

```
demouser@docker-vm:~/publish$ sudo az login
To sign in, use a web browser to open the page https://microsoft
HET4F9XGS to authenticate.
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "b9a3b4ac-e307-4940-9f37-8a6990d429cf",
    "id": "b36294ff-b022-4ff3-9fcd-39716852e936",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "b9a3b4ac-e307-4940-9f37-8a6990d429cf",
    "user": {
      "name": "gantasai400@gmail.com",
      "type": "user"
    }
  }
]
demouser@docker-vm:~/publish$
```

- Now we need to login ACR using below command
  - CMD: `sudo az acr login --name demoregistry322`
  - Note: demoregistry322 is the container registry name what we have created.
- Once we run above command it will login to ACR as below

```
demouser@docker-vm:~/publish$ sudo az acr login --name demoregistry322
Login Succeeded
demouser@docker-vm:~/publish$
```

- Show docker images
  - CMD: `sudo docker images`

```
demouser@docker-vm:~/publish$ sudo docker images
REPOSITORY          TAG          IMAGE ID
netimagecustom       latest       e15deb07055d
customimage          latest       61d600cecl4e
mcr.microsoft.com/dotnet/core/sdk 3.1          0c5d2de892f8
nginx                latest       2d389e545974
mcr.microsoft.com/dotnet/aspnet   5.0          29delb9e96c0
demouser@docker-vm:~/publish$
```

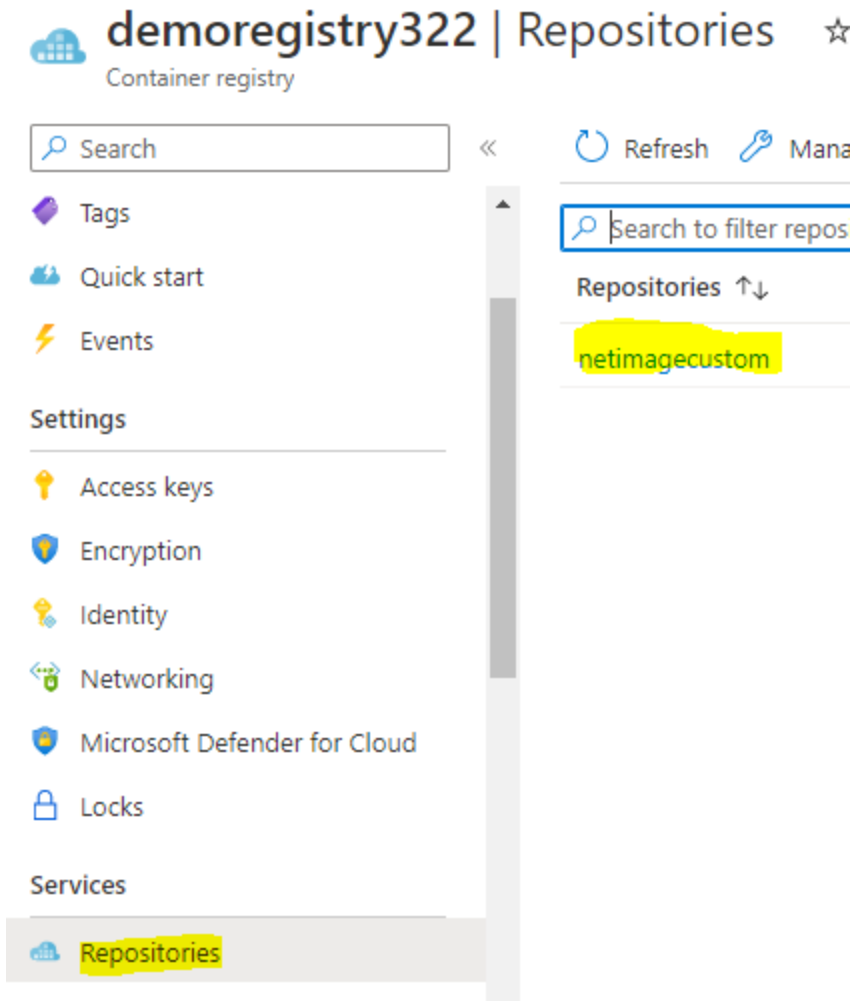
- To push the image to container registry run below command
- CMD: `sudo docker image tag netimagecustom demoregistry322.azurecr.io/netimagecustom`
- Note: `demoregistry322.azurecr.io` url copied from the container registry
- The above command will create one more images as below, it has tagged to image as our repository name

```
demouser@docker-vm:~/publish$ sudo docker image tag netimagecustom demoregistry322.azurecr.io/netimagecustom
demouser@docker-vm:~/publish$ sudo docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
demoregistry322.azurecr.io/netimagecustom latest       e15deb07055d     About an hour ago 719MB
netimagecustom       latest       e15deb07055d     About an hour ago 719MB
customimage          latest       61d600cecl4e     23 hours ago     149MB
mcr.microsoft.com/dotnet/core/sdk 3.1          0c5d2de892f8     2 weeks ago      714MB
nginx                latest       2d389e545974     2 weeks ago      142MB
mcr.microsoft.com/dotnet/aspnet   5.0          29delb9e96c0     4 months ago     205MB
demouser@docker-vm:~/publish$
```

- Now we can push the freshly created image using below command
  - CMD: `sudo docker push demoregistry322.azurecr.io/netimagecustom`
- The above command will successfully pushed the image to repository of container registry

```
demouser@docker-vm:~/publish$ sudo docker push demoregistry322.azurecr.io/netimagecustom
Using default tag: latest
The push refers to repository [demoregistry322.azurecr.io/netimagecustom]
4d757a23910b: Pushed
af80835f20cf: Pushed
0cfee092b3b2: Pushed
43aa896cb5d9: Pushed
fd44bbdl64c0: Pushed
690ebc895d9b: Pushed
34de7d7bf30e: Pushed
7dba48d63ddl: Pushed
latest: digest: sha256:b0d60cda04bac71fcc65720679274596fe3534d36fcdab23a35a6731b0647b4 size: 2011
demouser@docker-vm:~/publish$
```

- Now we can check the pushed image in azure portal
  - Go to container registry
  - Repositories on blade
- We can able to see our image as below



**Note:**

- We can use this stored image whenever we want.
- If our running image or VM or crashed we can use it.
- We can secure this image by applying VNet.
- We can use these images while using devops pipelines

- Below screen shot will show what are the steps involved in manual process vs automation using DevOps.

## ## Manual approach

### ## Secondly - DevOps

- Created a VM which is ubuntu
- Installed docker
- Dotnet publish -- dlls of my application (executable version)
- Dockerfile
- docker build -t imagename .
- docker run -d -p 80:80 --name customcontainer imagename
- Running on my host
- Save -- ACR

### ### Automate the above using Azure DevOps

- I will use the agent service (build and create images)
- docker comes preinstalled (already taken care)
- Build, restore and publish using Azure DevOps
- Dockerfile
- Create the image using Azure DevOps pipeline
- Save the image in an ACR which is again created using Infra as code

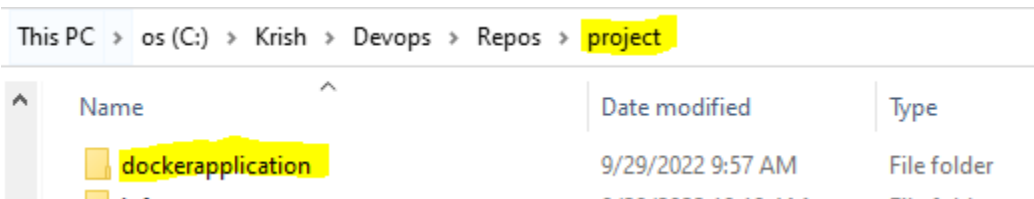
## Approach

Infra as code : Using ARM create an ACR

Using Azure Pipeline to build the code and create an image of the code

Finally, push the image into ACR

- Now we need to create new .Net application in Visual studio code.
- Name the project name as "dockerapplication"
- Go to index.cshtml change the line as "This is an .Net application running from a container"
- Save this application code in project folder.



Now we need to create ARM template in VS code. Follow below.

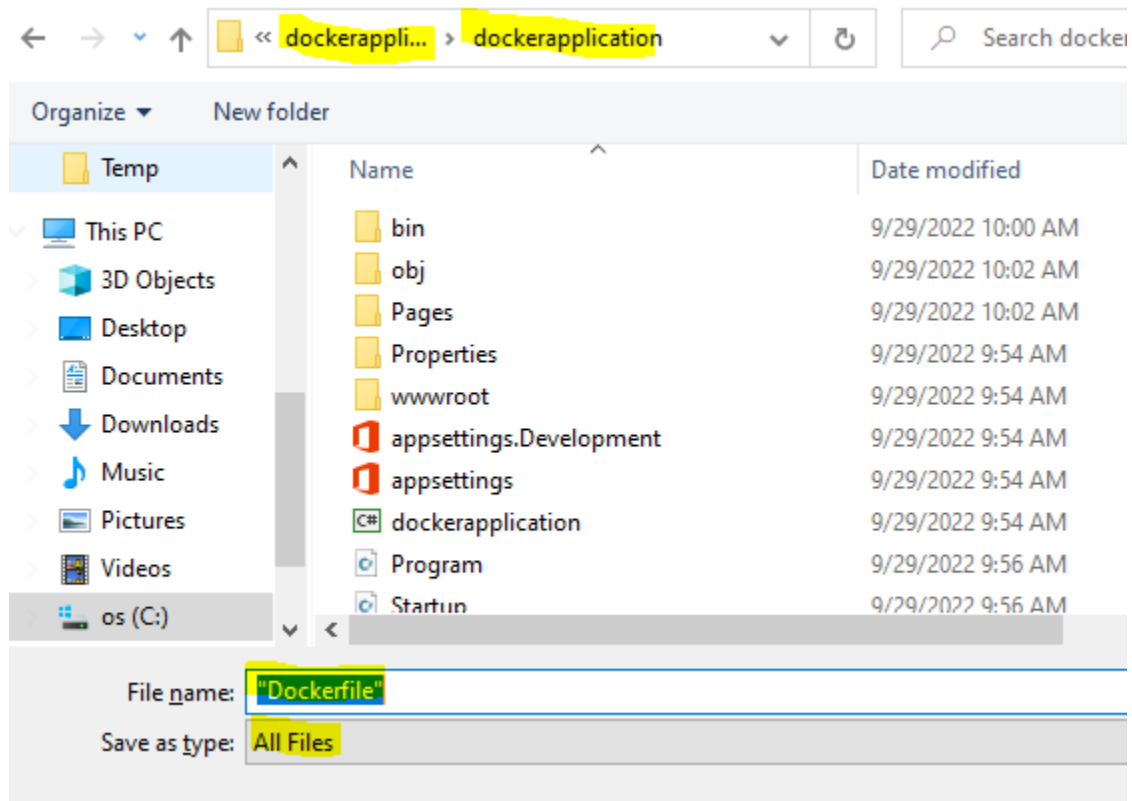
- ARM code must be

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "registryname": {
      "type": "string",
      "metadata": {
        "description": "description"
      }
    }
  },
  "functions": [],
  "variables": {},
  "resources": [
    {
      "name": "[parameters('registryname')]",
      "type": "Microsoft.ContainerRegistry/registries",
      "apiVersion": "2019-05-01",
      "location": "[resourceGroup().location]",
      "sku": {
        "name": "Standard"
      },
      "properties": {
        "adminUserEnabled": true
      }
    }
  ],
  "outputs": {}
}
```

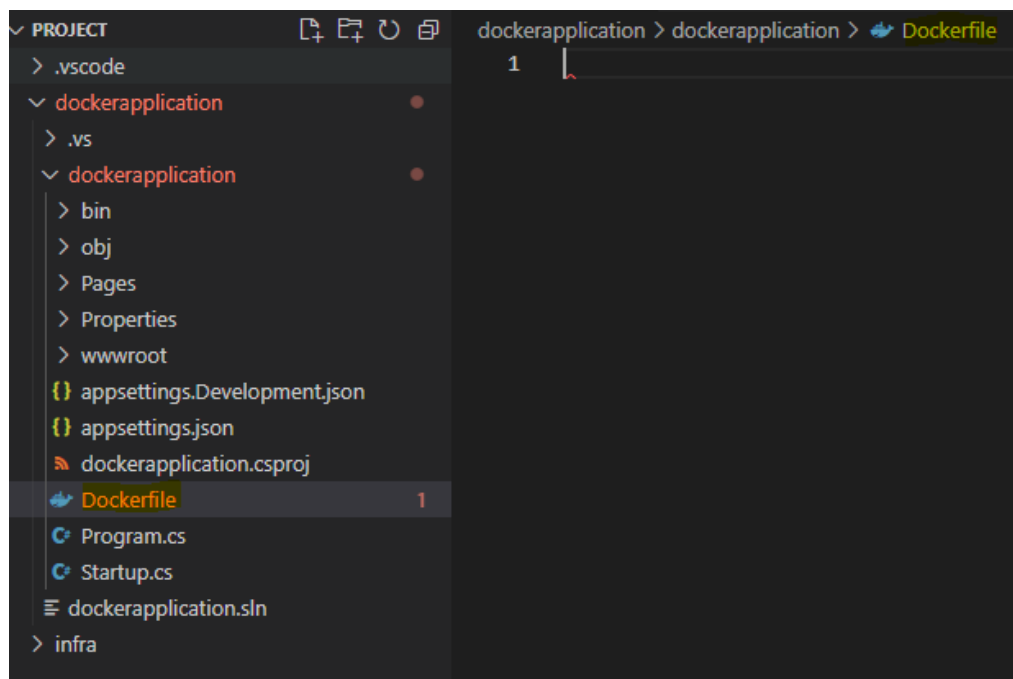
- Save this ARM file as “acr.json” save this in infra folder. Infra folder is sub folder in project.
- Now both folders like below

This PC > os (C:) > Krish > Devops > Repos > project		
Name	Date modified	Type
dockerapplication	9/29/2022 9:57 AM	File folder
infra	9/29/2022 10:19 AM	File folder

- Now we need to create <"Dockerfile"> in dockerapplication folder. The file must be empty.



- Save the file
- Now go to VS code open complete our project folder
- Open Docker file in that as below



- Now open browser and search as “docker file for dotnet core”
- Select the same as below

## Dockerize an ASP.NET Core application

- Now we need to follow the Method 1
- Now copy below script and paste the in VS Code Docker file

```

• # syntax=docker/dockerfile:1
• FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
• WORKDIR /app
•
• # Copy csproj and restore as distinct layers
• COPY *.csproj ./
• RUN dotnet restore
•
• # Copy everything else and build
• COPY ../engine/examples ./
• RUN dotnet publish -c Release -o out
•
• # Build runtime image
• FROM mcr.microsoft.com/dotnet/aspnet:6.0
• WORKDIR /app
• COPY --from=build-env /app/out .
• ENTRYPOINT ["dotnet", "aspnetapp.dll"]

```



Now change the above code as below.

```
# syntax=docker/dockerfile:1
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /app

# Copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:3.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "dockerapplication.dll"]
```

- Save the code
- Now go to DevOps portal create new project as “docker”
- Go to files and create new repository and name it as “docker-repo”
- Clone the project to vs code
- Save that file in Repo folder
- Now go to project folder and copy below folders

‣ This PC ‣ os (C:) ‣ Krish ‣ Devops ‣ Repos ‣ project

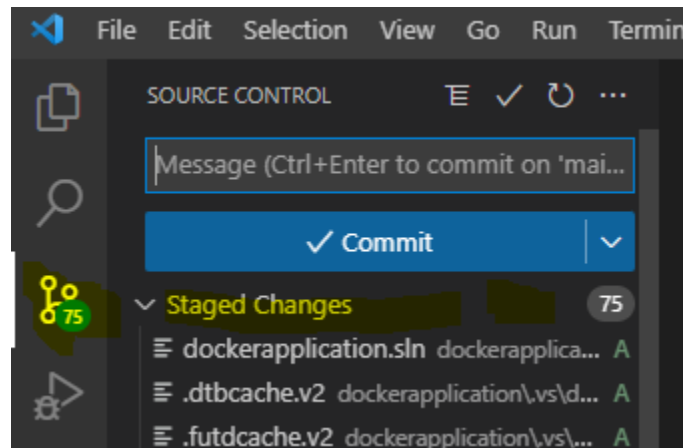
Name	Date modified	Type
.vscode	9/29/2022 10:45 AM	File folder
dockerapplication	9/29/2022 9:57 AM	File folder
infra	9/29/2022 10:19 AM	File folder

- Paste in docker-repo folder.

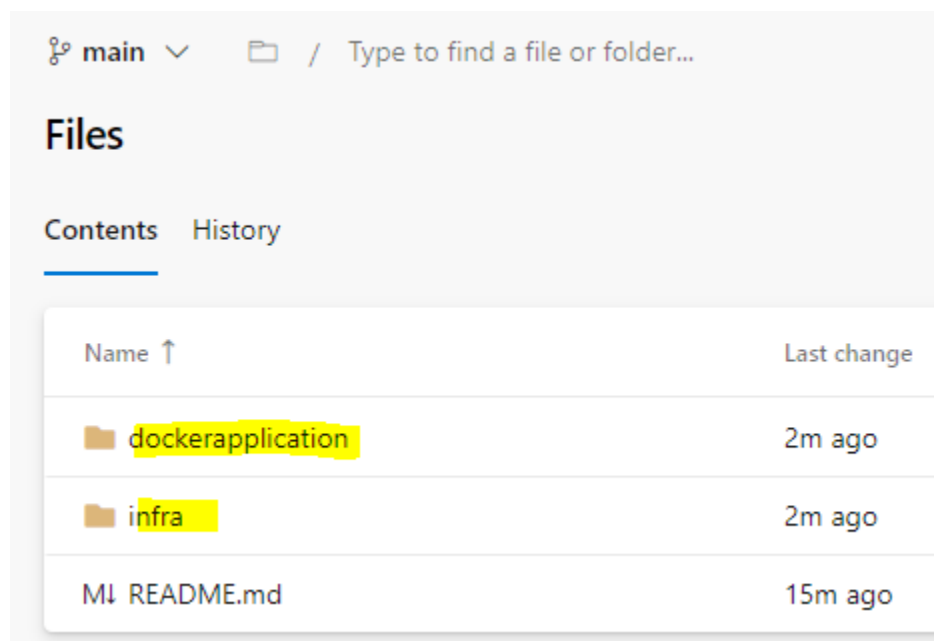
This PC ‣ os (C:) ‣ Krish ‣ Devops ‣ Repos ‣ docker-repo

Name	Date modified
.git	9/29/2022 11:58 AM
dockerapplication	9/29/2022 12:06 PM
infra	9/29/2022 12:06 PM
README	9/29/2022 11:57 AM

- Now go to VS code click on repo symbol and click on staged changes +




- Commit as ARM push
- Click on sync changes
- Now go to portal and check the files are pulled from the local machine.



- Now go to pipe lines and create classic pipe line
- Search as “docker container” and click on apply

## Select a template

Or start with an  Empty job

docker container



## Configuration as code



### YAML

Looking for a better experience to configure your pipelines using YAML files? Try the new YAML pipeline creation experience. [Learn more](#)

## Featured



### Docker container

Build a Docker image and push it to a container registry.

- Give pipeline name as “dockerimagebuilde”
- Click on pipe line
  - Select Agent specification as “Ubuntu-latest”
- Now add one more task as “ARM template deployment”
  - Display name as: Create ACR
  - Select connection, subscription
  - Resource group name as : \$(rgname)
  - Location : select west us 3
  - Template : select ACR.json file form ifra
  - Override template parameters : as below

## Override template parameters

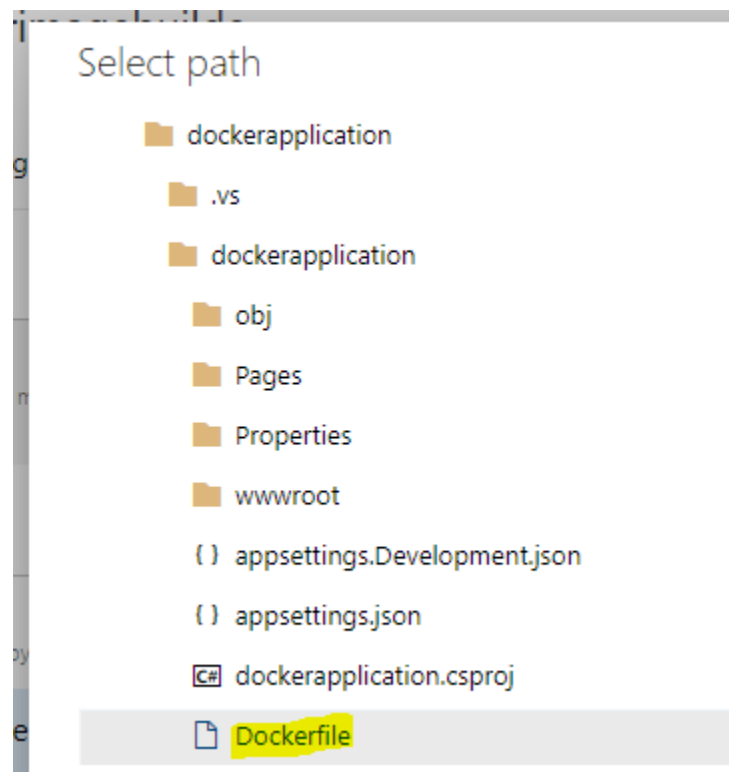
Name	Value
registryname	\$(registryname)

- Now go to Variables and then pipe line variables

- We need to add below 3 highlighted variables

Name ↑	Value
system.collectionId	f2df251b-bfe9-4b06-a18a-7330841b379e
system.debug	false
system.definitionId	< No pipeline ID yet >
system.teamProject	docker
rgname	acr-rg
registryname	devopskrishacr00
imagename	devopscustomimage

- Now go to Task and Build an Image
  - Azure container registry : \$(registryname).azurecr.io
  - Docker file: brows and select as below docker file



- Image name : \$(imagename)
- Check the include source tags

- Now go to Push Image
  - Select subscription as our connection
  - Image name and azure container registry copy from Build an Image and paste here
  - Select check box for include source tag
- Now save the pipeline
- Now run the pipe line it will create resource group in that it will create container registry then it will push the custom image
- We can go to container registry then repositories there we can see our image.

Now create new webapp

- Creating webapp
  - Resource group: select acr-rg
  - Name: krishwebapp
  - Publish: docker container
  - Operating system: Linux
  - Select location
  - click on next to docker
  - Image source: Azure container registry
  - Select registry and image
  - Now review and create
- Now go to web app link and brows we can see our .Net application which we deployed.

Command	Used
<code>docker ps --filter status=paused</code>	To show all paused containers
<code>docker ps --filter status=exited</code>	To show all exited or stopped containers
<code>docker rm \$(docker ps --filter status=exited) -f</code>	To removed all exited containers
<code>docker rm \$(docker ps --filter status=paused) -f</code>	To removed all paused containers