**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 Enginer

**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. $ sudo apt-get install \
4. ca-certificates \
5. curl \
6. gnupg \
7. lsb-release
8. Add Docker’s official GPG key:
9. $ sudo mkdir -p /etc/apt/keyrings
10. $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
11. Use the following command to set up the repository:
12. $ echo \
13. "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
14. $(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.

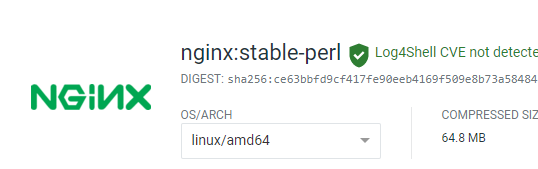
* The above-mentioned commands will install the docker
* To check docker installed or not run below command

CMD: docker version

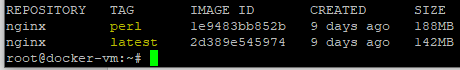
* To work as administrator in Linux we need to change demo user to root user use below command
  + CMD: sudo -i
* Note: when we are working as root user we don’t need to enter sudo. The commands which required sudo, no need to give that.
* To check what are the Images are in the docker run below command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



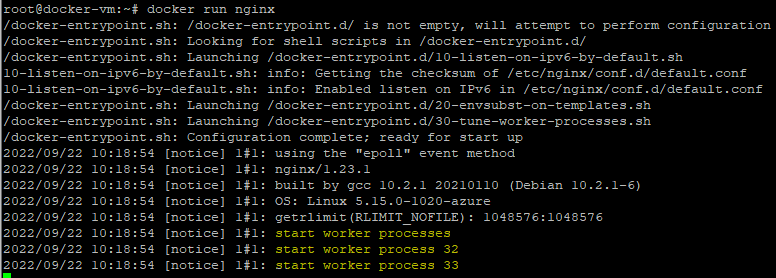
* + CMD: docker pull nginx:perl
* Now it will install perl version of the nginx
* To check that again run below command
  + CMD: docker images
* Now as per below image it will show below 2 images



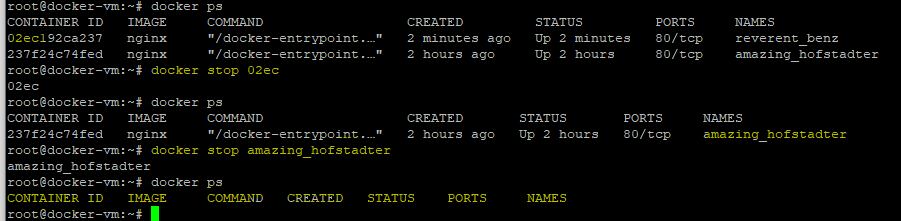
* If you want to remove any Image, for now we need to delete the perl version, run below command
  + CMD: docker image rm nginx:perl
* Now check whether it has deleted or not by below command it will only show latest version of nginx as per below image
  + CMD: docker images



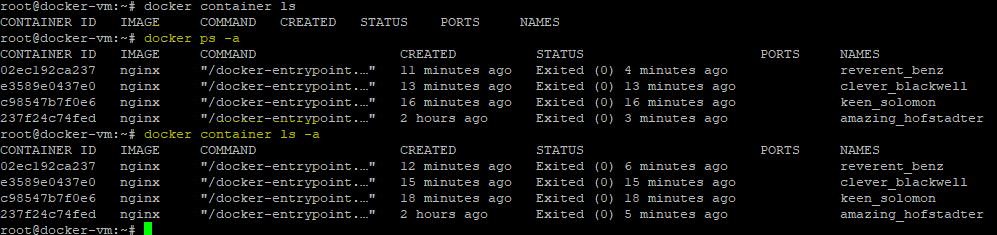
* Now check whether any container running or not by any one command or below
  + CMD: docker ps
  + Or
  + CMD: docker container ls
* Now run below command
  + CMS: docker run nginx
* The above command will start the worker process as below those are running in container



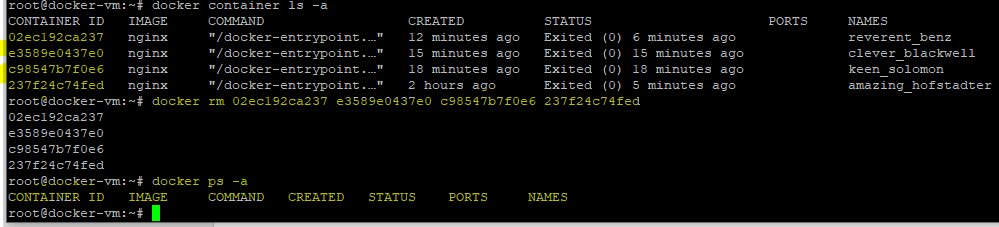
* To exit form that give
  + CMD: ctr+C
* The above command will kill the container. Because its killed the container is running on terminal.
* To run the container in backend run below command
  + CMD: docker run -d nginx
* The above command will run the container in detached mode
* Stop container by 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.



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



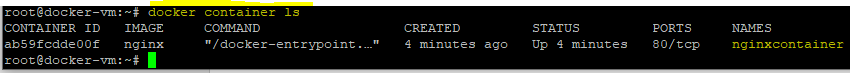
* The above commands will show stopped containers list
* No I want to remove all the stopped containers we need to give command like “ docker rm (id of continer1 2 3) like below screenshot
* Now we can see no container is running in the background



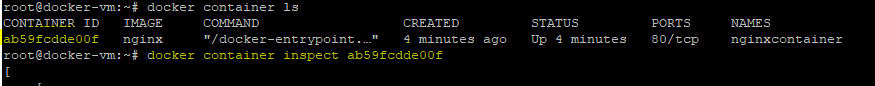
* Now I want to run container and give it to name by below command
  + CMD: docker run -d --name nginxcontainer nginx



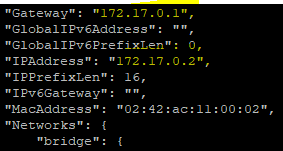
* Now check our container name by below command it will give name as nginxcontainer
  + CMD: docker container ls



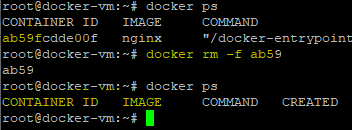
* To inspect how our container running, give below command it will give json file it shows when it has started and all
  + CMD: docker container inspect <container ID>



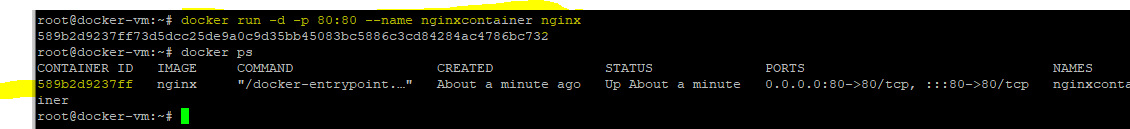
* In that json file we can see that docker has assigned the private IP or internal IP to communicate. As below screenshot



* 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>



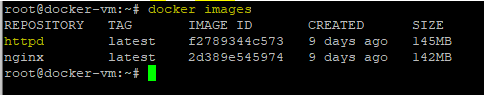
* Now we can see container is removed
* Now run below command to divert port 80 to container by below command
  + CMD: docker run -d -p 80:80 --name nginxcontainer nginx
* We can see it has created the container again



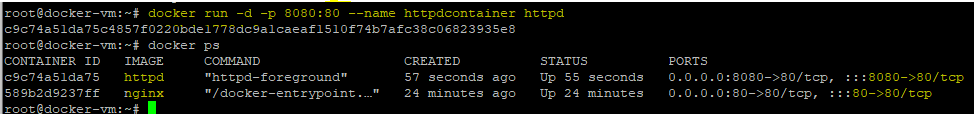
* Now go to browser and search with <VM public IP:80>
* We can reach to our nginx container

**Running multiple container and port diverting**

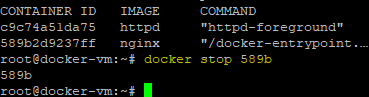
* Earlier we have one container running the nginx Image
* Now we can install one more Image of “httpd”
* Below is the command to pull the “httpd” image
  + CMD: docker pull httpd
* Now we can see below httpd image installed



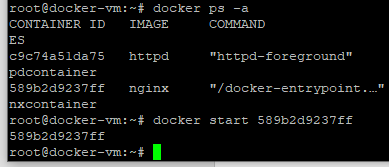
* We have used port 80 to nginx and now we can’t use the same port to httpd. So we need to map httpd to other port is 8080 by below command
  + CMD: docker run -d -p 8080:80 --name httpdcontainer httpd



* 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>



* 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 commd
  + CMD: docker start <container id>



* 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



* Now we are in container

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

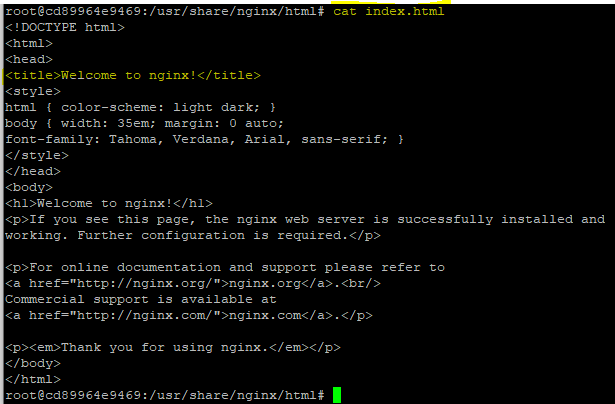
* To go to folder of nginx run below command
  + CMD: cd /usr/share/nginx/html



* As per above screenshot we are in ngnix user foler
* To check what are the files saved in that folder run below command
  + CMD: ls



* 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



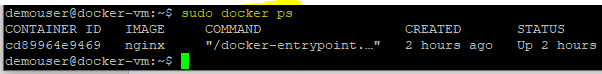
* If you want to exit from container, go to root run below command
  + CMD: exit
* We can see that we are now in root.



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



* 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

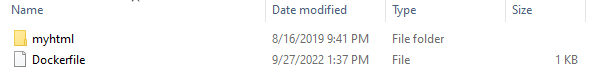


**Creating Customized Image**

* 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.



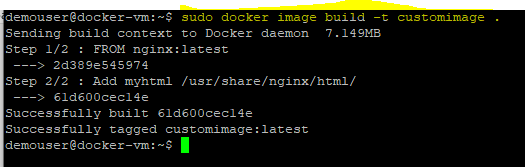
* Files must be below



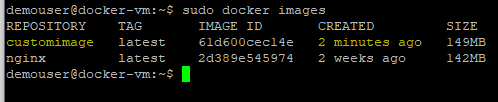
* Now to copy the both files in our docker VM we need to use FTP protocol by WINSCP on local machine
* Now open WINSCP
  + Select file protocol as : SCP
  + Host name : VM public IP
  + Give user name and password
  + Click on login
* Now copy paste like below and using drag and drop
* Once copy task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



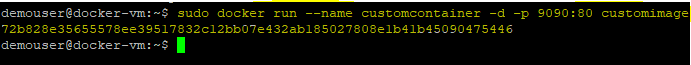
* We can see both files in that
* Now run below command
  + CMD: sudo docker image build -t customimage .



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



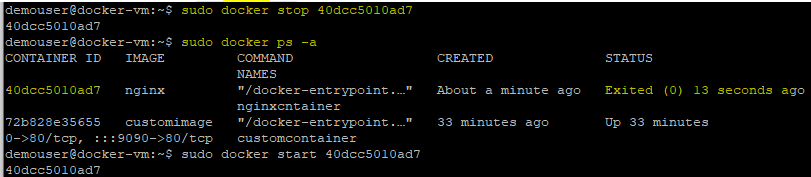
* To run the custom image run below command
  + CMD: sudo docker run --name customcontainer -d -p 9090:80 customimage



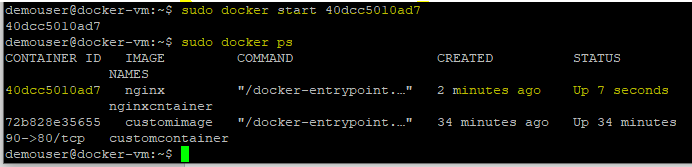
* Now go to Azure portal and docker-vm
* Create new inbound rule for port : 9090
* Now go to browser and search with <vm ip:9090>
* Then we can see the custom image portal

**Stopping, starting and creating the docker containers**

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



* To start exited or stopped container
  + CMD: sudo docker start <container name or id>



* 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](https://docs.docker.com/samples/dotnetcore/)

* 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

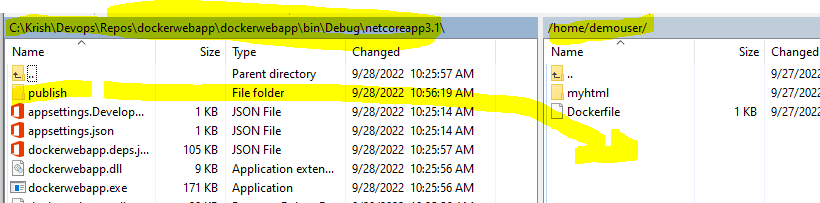
FROM mcr.microsoft.com/dotnet/aspnet:5.0

COPY bin/Release/netcoreapp3.1/publish/ App/

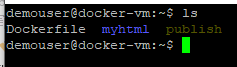
WORKDIR /App

ENTRYPOINT ["dotnet", "aspnetapp.dll"]

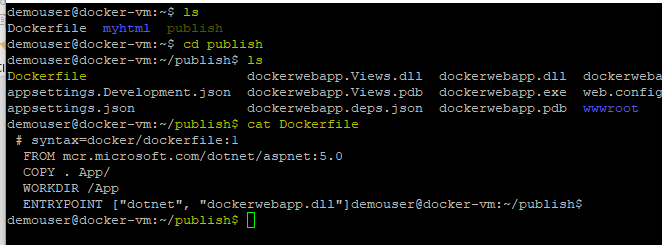
* Edit the code as below
* # syntax=docker/dockerfile:1
* FROM mcr.microsoft.com/dotnet/core/sdk:3.1
* COPY . App/
* WORKDIR /App
* ENTRYPOINT ["dotnet", "dockerwebapp.dll"]
* Save as the code save in our publish folder in project and name it as <”Dockerfile”> and document type must be all files.
* Now 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.



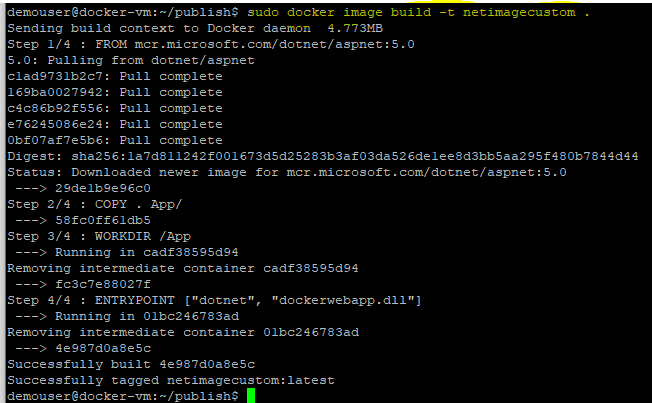
* It will copy the code in VM.
* Now go to VM and check whether the file copied or not using below commad
  + CMD: ls



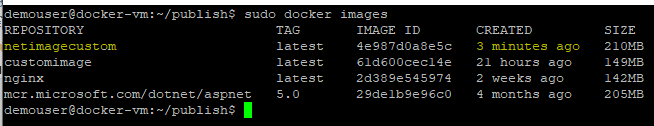
* Check the Dockerfile using below commands



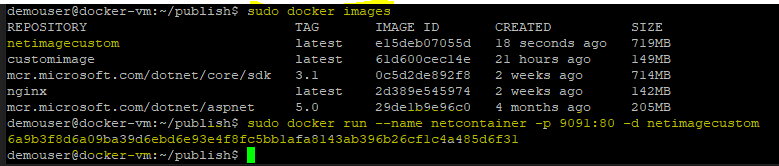
* Now run below command
  + CMD: sudo docker image build -t netimagecustom .
* The above command will pull the code as below



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



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



* 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

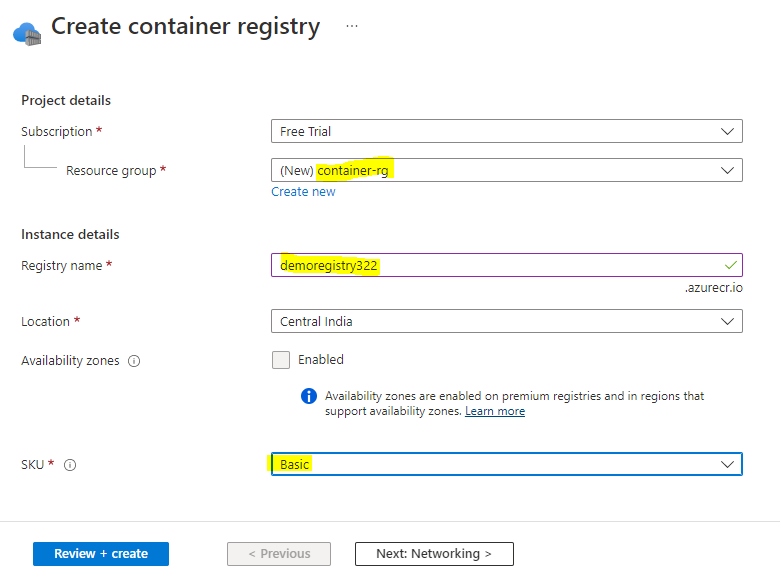
**Storing our Custom image:**

We have 2 ways to save the images

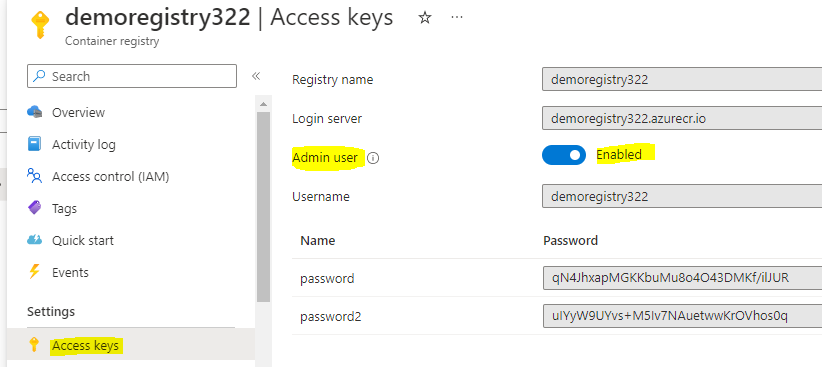
* 1. Docker hub (register and store)
  2. Container registry in Azure

**Container Registry:**

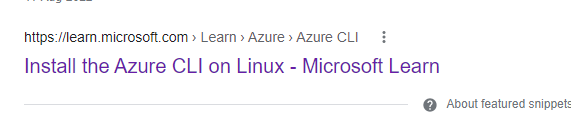
* Now go to portal and search as “container registry”



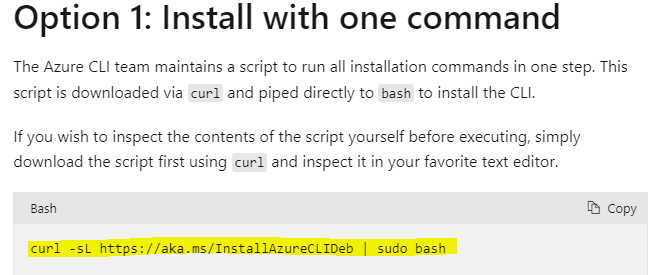
* Click on review and create.
* Now go to container registry and
  + Access keys from blade
  + Enable admin user



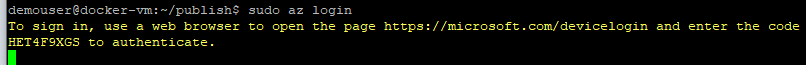
* Before pushing our image to Azure container registry we need to install Azure CLI in Linux VM
* Go to browser and search as “ install azure cli on ubuntu”



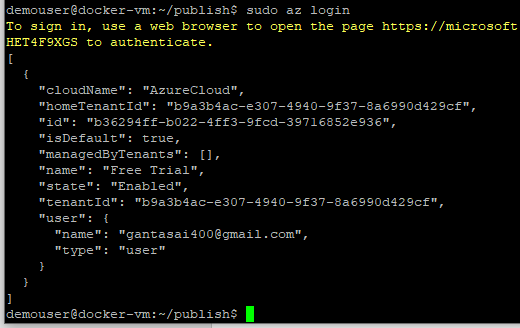
* Click on the top
* Scroll down and copy below command
  + CMD: curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash



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



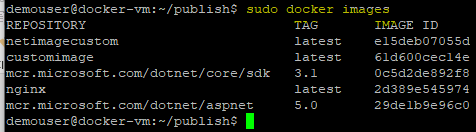
* Now go to our local machine browser and search as <https://microsoft.com/devicelogin>
* Now we need to copy paste the code from linux to portal
* Copy above code : HET4F9XGS
* Once we paste and login it will login azure in linux as below



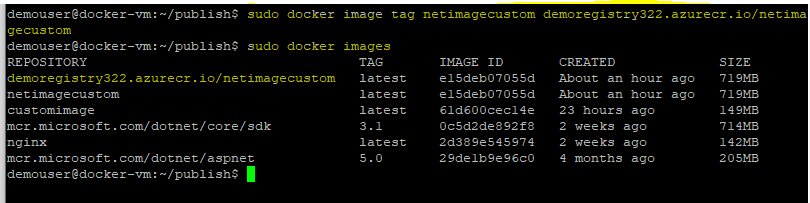
* Now we need to login ACR using below command
  + CMD: sudo az acr login --name demoregistry322
  + Note: demoregistry322 is the container registry name what we have created.
* Once we run above command it will login to ACR as below



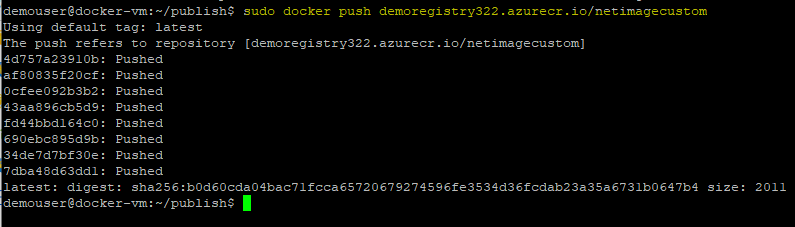
* Show docker images
  + CMD: sudo docker images



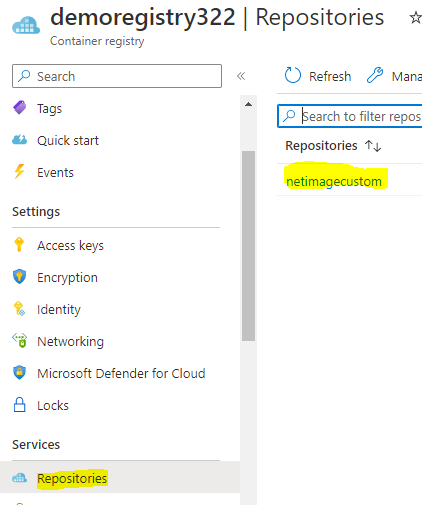
* To push the image to container registry run below command
* CMD: sudo docker image tag netimagecustom demoregistry322.azurecr.io/netimagecustom
* Note: demoregistry322.azurecr.io url copied from the container regestry
* The above command will create one more images as below, it has tagged to image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



* 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

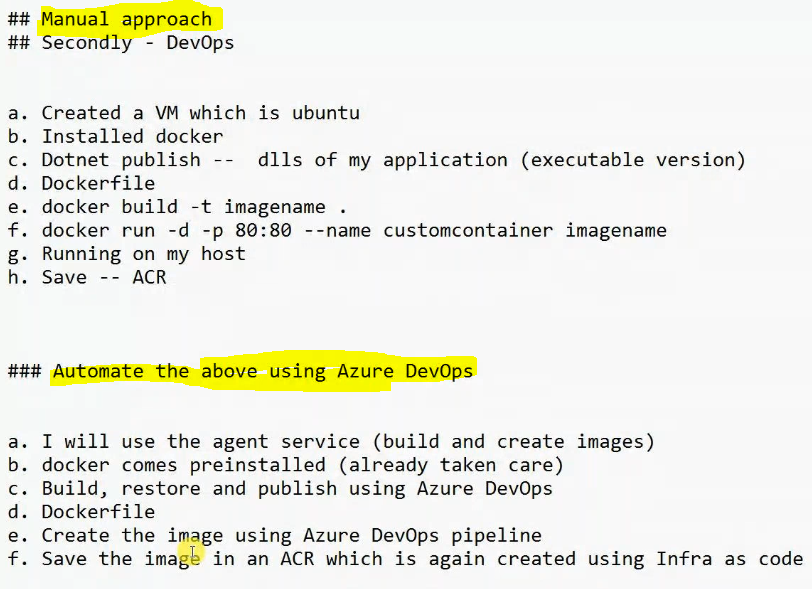


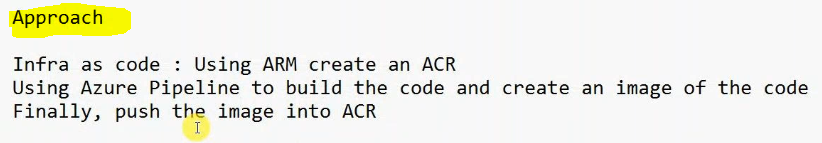
* Now we can check the pushed image in azure portal
  + Go to container registry
  + Repositories on blade
* We can able to see our image as below



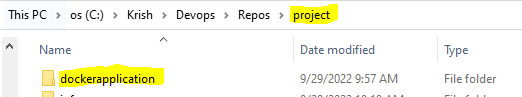
**Note:**

* 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.



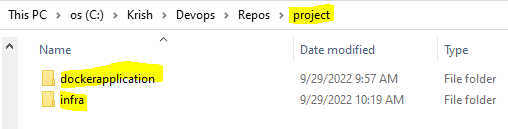


* Now we need to create new .Net application in Visual studio code.
* Name the project name as “dockerapplication”
* Go to index.cshtml change the line as “This is an .Net application running from a container”
* Save this application code in project folder.

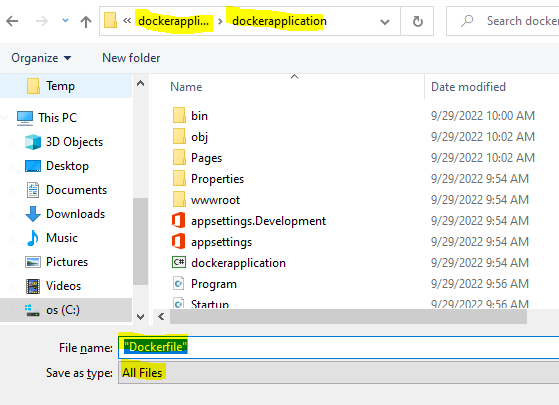


Now we need to create ARM template in VS code. Follow below.

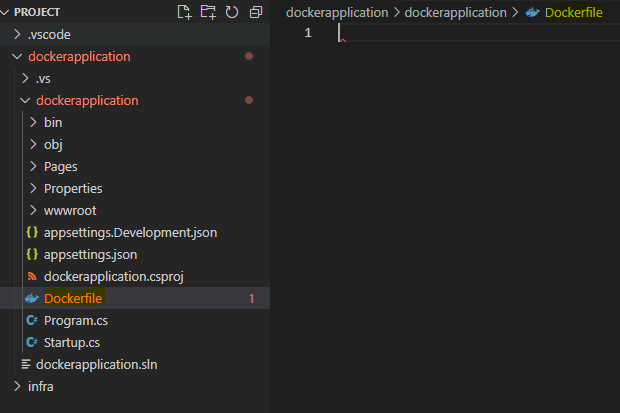
* ARM code must be
* {
* "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
* "contentVersion": "1.0.0.0",
* "parameters": {
* "registryname": {
* "type": "string",
* "metadata": {
* "description": "description"
* }
* }
* },
* "functions": [],
* "variables": {},
* "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



* 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](https://docs.docker.com/samples/dotnetcore/)

* Now we need to follow the Method 1
* Now copy below script and 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

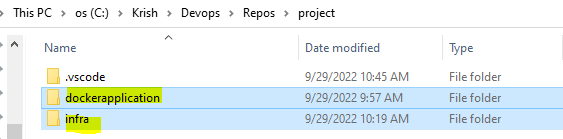
FROM mcr.microsoft.com/dotnet/aspnet:3.1

WORKDIR /app

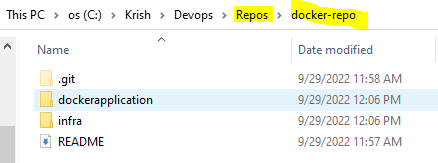
COPY --from=build-env /app/out .

ENTRYPOINT ["dotnet", "dockerapplication.dll"]

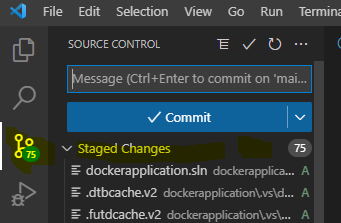
* Save the code
* Now go to Devops portal create new project as “docker”
* Go to files and create new repository and name it as “docker-repo”
* Clone the project to vs code
* Save that file in Repo folder
* Now go to to project folder and copy below folders



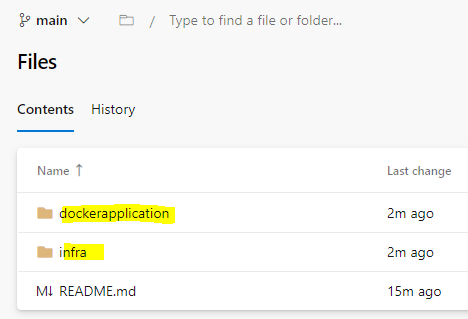
* Paste in docker-repo folder.



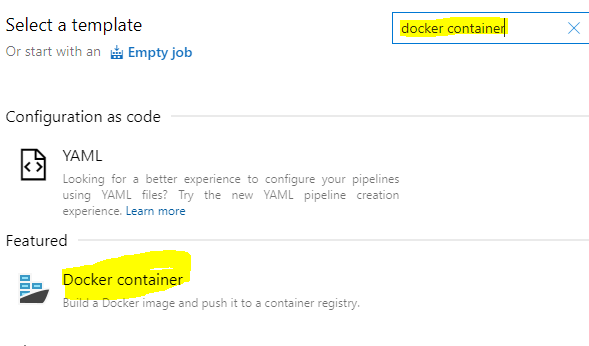
* Now go to VS code click on repo symbol and click on staged changes +



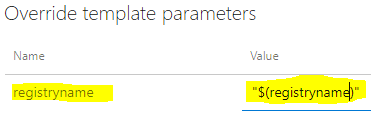
* Commit as ARM push
* Click on sync changes
* Now go to portal and check the files are pulled from the local machine.



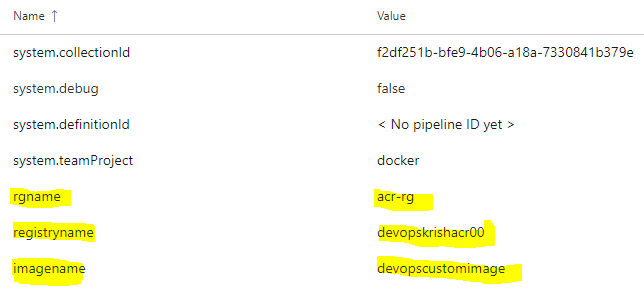
* Now go to pipe lines and create classic pipe line
* Search as “docker container” and click on apply



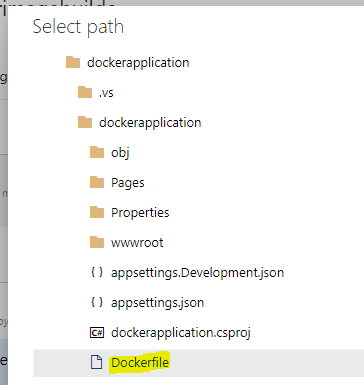
* Give pipeline name as “dockerimagebuilde”
* Click on pipe line
  + Select Agent specification as “Ubuntu-latest”
* Now add one more task as “ARM template deployment”
  + Display name as: Create ACR
  + Select connection, subscription
  + Resource group name as : $(rgname)
  + Location : select west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



* Now go to Variables and then pipe line variables
* We need to add below 3 highlighted variables



* 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** |
| docker ps --filter status=paused | To show all paused containers |
| docker ps --filter status=exited | To show all exited or stopped containers |
| docker rm $(docker ps --filter status=exited) -f | To removed all exited containers |
| docker rm $(docker ps --filter status=paused) -f | To removed all paused containers |
|  |  |