Table of Contents

[AZURE CONTAINER SERVICE 2](#_Toc125204405)

[DOCKER 2](#_Toc125204406)

[INSTALLING DOCKER RUNTIME IN LINUX VM 2](#_Toc125204407)

[EXAMPLE 3](#_Toc125204408)

[RUNNING NGINX CONTAINER IN LINUX VM 4](#_Toc125204409)

[AZURE CONTAINER REGISTRY(ACR) 5](#_Toc125204410)

[CREATING AN ACR 6](#_Toc125204411)

[PUBLISH A DOCKER IMAGE IN ACR 7](#_Toc125204412)

[CREATING A CONTAINER INSTANCE 8](#_Toc125204413)

[EXAMPLE 8](#_Toc125204414)

[AZURE KUBERNETES SERVICES 9](#_Toc125204415)

[KUBERNETES FEATURES 9](#_Toc125204416)

[KUBERNETES ARCHITECTURE 9](#_Toc125204417)

[NODES 9](#_Toc125204418)

[CLUSTER 10](#_Toc125204419)

[MASTER 10](#_Toc125204420)

[COMPONENTS OF KUBERNETES 11](#_Toc125204421)

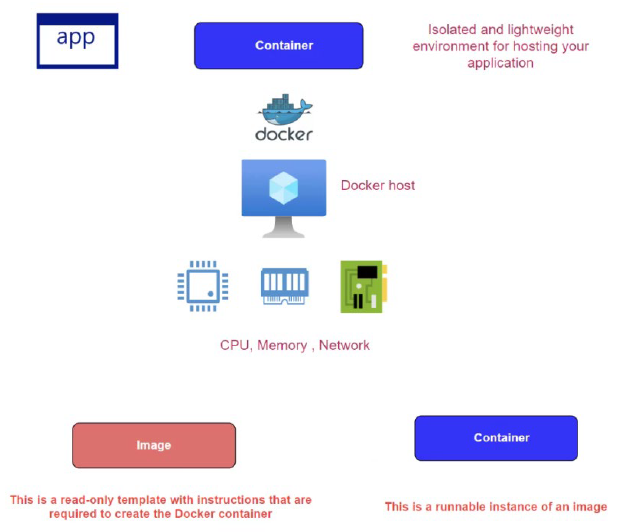
[KUBERNETES IN AZURE 11](#_Toc125204422)

[CREATING THE KUBERNETES SERVICE 12](#_Toc125204423)

# AZURE CONTAINER SERVICE

## DOCKER

* Docker is an open platform that is used for developing, shipping, and running your applications. It has the ability to package and run an application in a loosely isolated environment called a container.



* To deploy an application has a container on underlying VM. We need to install the Docker runtime. This Docker tool is responsible for creating and managing the container. It also helps a container to make use of the underlying hardware resources of the VM e.g., memory, CPU and Networking devices

|  |  |
| --- | --- |
| IMAGE | * Container is a runnable instance of something known has an image. * The image is the read only template which has instructions that required to create the Docker container |

### INSTALLING DOCKER RUNTIME IN LINUX VM

* Step 1 – Provision a Linux VM
* Step 2: Connect with VM using Putty tool
* Step 3: Execute the following command [ <https://docs.docker.com/engine/install/ubuntu/> ]

You can use the following commands to work with Docker on an Ubuntu Linux virtual machine OR You can also refer to Docker documentation - <https://docs.docker.com/engine/install/ubuntu/>

**// Update the package index**

sudo apt-get update

**// Install packages to allow apt to use the repository over HTTPS**

1. sudo apt-get install \
2. ca-certificates \
3. curl \
4. gnupg \
5. lsb-release

**// Add Docker's official GPG key**

1. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

**// Setup a stable repository**

1. echo \
2. "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
3. $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

**// Update the package index**

sudo apt-get update

**// Install docker, containerd**

1. sudo apt-get install docker-ce docker-ce-cli containerd.io

**// Launching a container**

sudo docker run --name mynginx -p 80:80 -d nginx

### EXAMPLE

1. INSTALL A DOCKER USING VM.
2. PULL HSHAR/WEBAPP (https://hub.docker.com/r/hshar/webapp ) REPOSITORY
3. CREATE NEW FILE IN THIS REPOSITORY

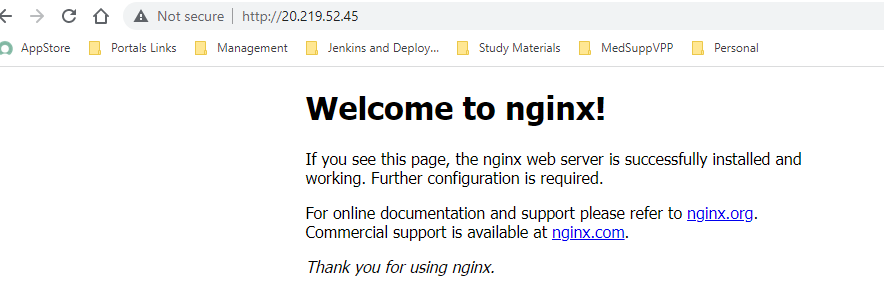
To install Docker Runtime on a Linux VM- Run the following command

|  |  |
| --- | --- |
| UPDATE PACKAGE | sudo apt-get update |
| INSTALL DOCKER | sudo apt-get install docker.io |
| CHECK DOCKER STATUS | sudo service docker status |
| TO PULL THE REPO | sudo docker pull hshar/webapp |
| TO CHECK THE IMAGES | sudo docker images |
| TO CREATE THE DOCKER CONTAINER | sudo docker run -itd --name myapp hshar/webapp |
| TO RUN THE DOCKER CONTAINER | sudo docker exec -it myapp bash |
| CREATING NEW FILE IN THIS REPOSITORY | * Navigate to webroot directory – **cd /var/www/html** * Open the Editor to add the file **- nano index.html** * **Copy a given HTML in index.html**   Note   * 1. To save file - ctrl+s   2. To exit the editor – ctrl +x |

### RUNNING NGINX CONTAINER IN LINUX VM

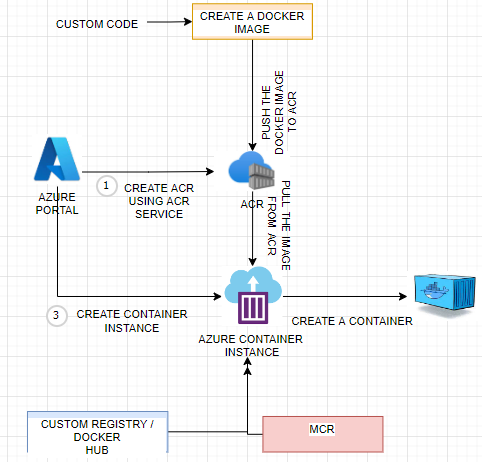
|  |
| --- |
| Command - **sudo docker run --name mynginx -p 80:80 -d nginx** |
| 1. Mynginx: Name of the container 2. -d : Run the container in detached mode. This means the container will continue to run until stopped 3. – p : this tells to map the port number of the container the docker host |
|  |
| RUNNING CONTAINER IN BACKGROUND – **sudo docker ps -a** |

***To access the nginx webserver make sure PORT 80 is allowed in the NSG Inbound rules***



# AZURE CONTAINER REGISTRY(ACR)

|  |  |
| --- | --- |
|  | * When we want to launch a container in physical / VM machine based on some image. The image gets downloaded from Docker hub, so Docker hub is a repository of images. * On the other hand, if we create own custom application. We too can create a Docker image out of that application. The custom image can also be uploaded in Docker hub * This docker image can be pulled to create containers. * **Like Docker hub, Azure has managed service called ACR. It's a private Docker registry** |

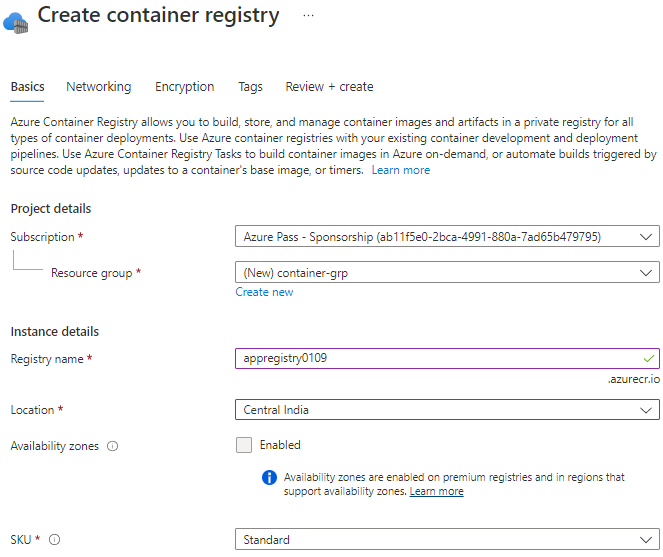


To create a container based out of the images pulled from ACR we need to go through following steps

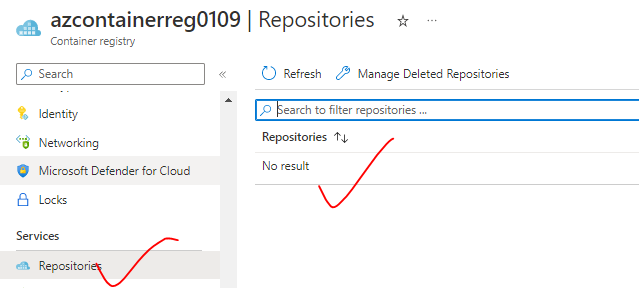
1. Create ACR using Azure’s ACR Service
2. Publish an image to the ACR
3. Create a Container Instance from Azure’s Container instance Service, then create a container based on the image in ACR.

## CREATING AN ACR

* Search for Container Registry 🡪 Create 🡪 Enter the detail as below🡪 Create



* Initially there will be no image present in the registry



## PUBLISH A DOCKER IMAGE IN ACR

1. Create a Node app and create a docker image for the same -
2. Sample Node Project - <https://github.com/avishekhsinhaRepo/Node-JS/tree/master/docker_web_app>

|  |  |
| --- | --- |
|  |  |
| TO BUILD THE APPLICATION | docker build . -t node-web-app |
| TO RUN THE APP IN LOCAL MACHINE | docker run -p 49160:8080 -d node-web-app |
| ACCESSING THE APP |  |

POWERSHELLL COMMAND TO PUBLISH THE IMAGE

|  |  |
| --- | --- |
| CONNECT TO AZURE | Connect-AzAccount -TenantId cf4d6b26-7de4-4fe3-8419-aab6f5713a30 |
|  | docker login azcontainerreg0109.azurecr.io |
|  | docker push azcontainerreg0109.azurecr.io/node-web-app |

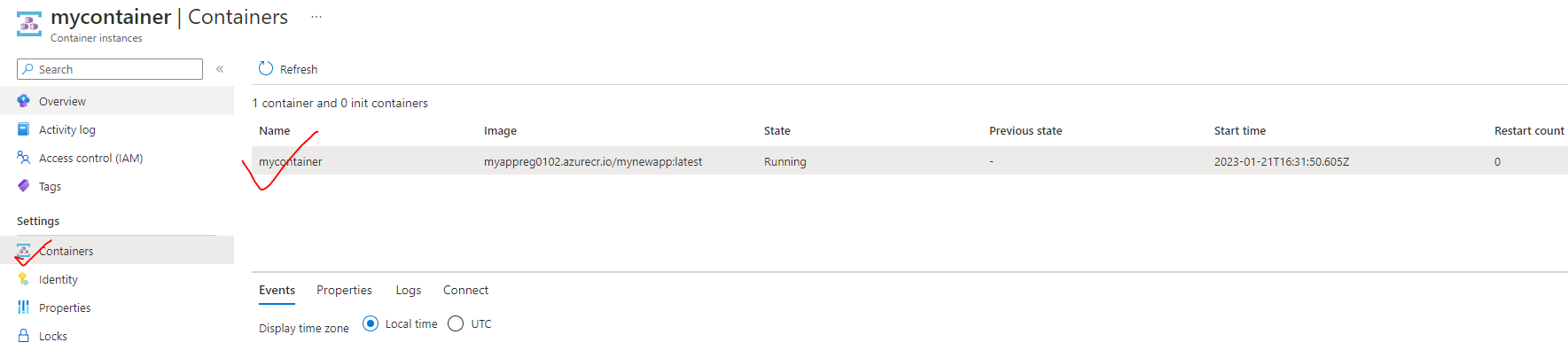
## CREATING A CONTAINER INSTANCE

|  |  |
| --- | --- |
|  | * Container are the instance of image. The container can be created using “Container Instance” Service * The containeer size cannot be changed once created. * The container instance, lightweight, managed service of Azure to deploy * containers. * It can pick up the image from Docker hub or from ACR. Then it will have a Linux environment, a compute environment that will be used for running your containers. |

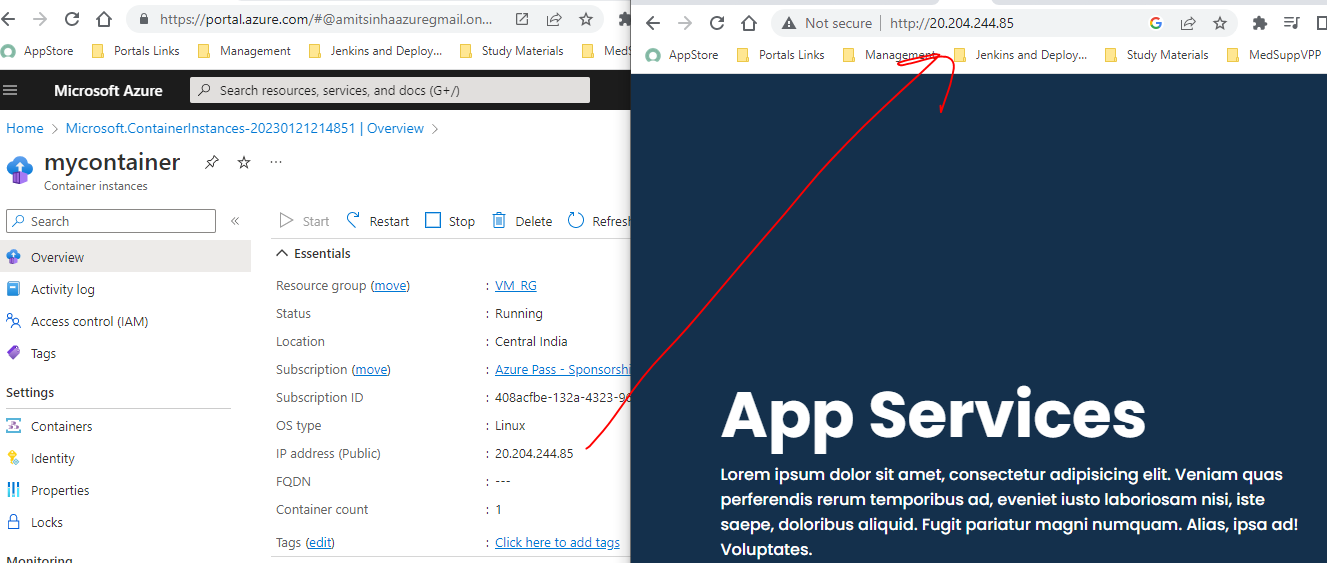
**CONTAINER CAN BE CREATED FROM MULTIPLE SOURCE OF IMAGE**

|  |  |
| --- | --- |
| IMAGE SOURCE | Option |
| MICROSOFT ACR |  |
| ACR |  |
| Note: To make use ACR – we need to enable the admin user  Steps : Container Registry 🡪 Access keys 🡪 Admin user(Enabled) | |
| DOCKER HUB |  |

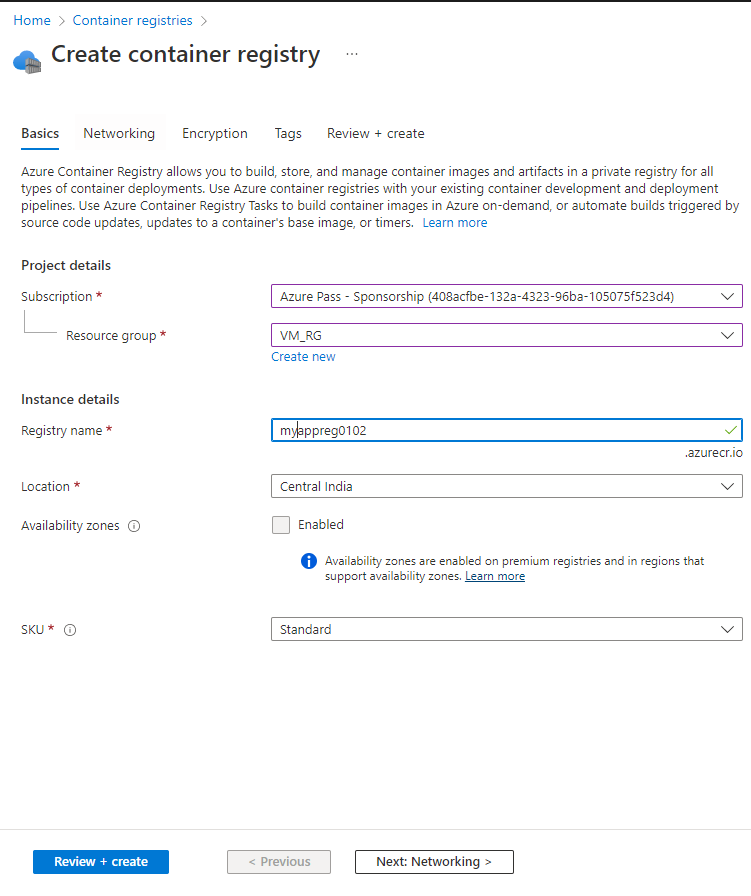
CONTAINER INSTANCE HAS A CONTAINER RUNNING FROM A GIVEN IMAGE



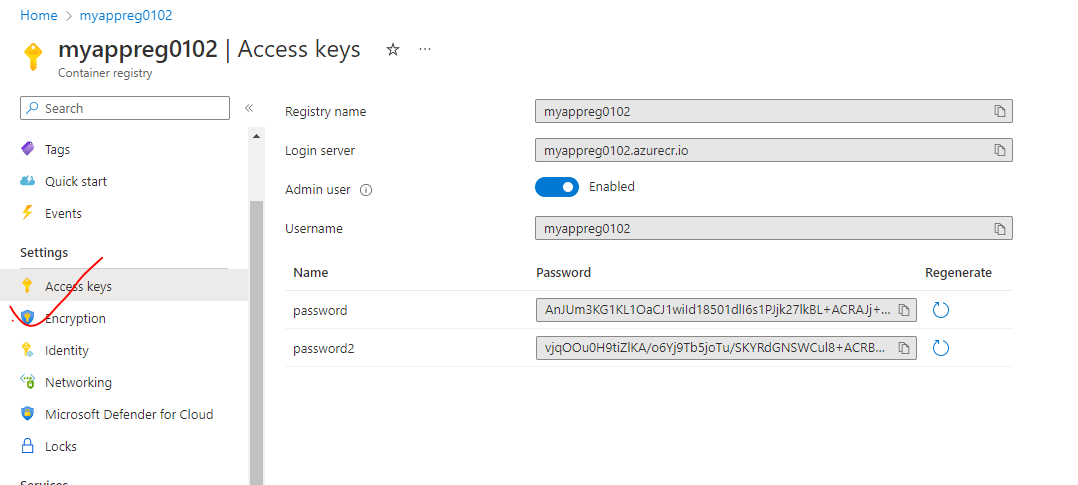
TO ACCESS THE APPLICATION – We use public IP address of the container intance.



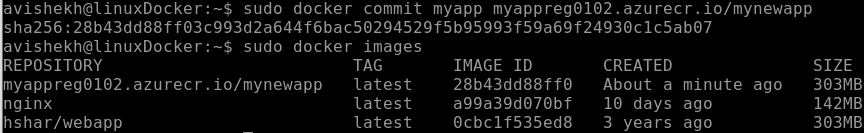
## EXAMPLE

1. **CREATE AZURE CONTAINER REGISTRY AND CONNECT IT TO DOCKER RUNNING IN VM.**
2. **UPLOAD THE IMAGE YOU CREATED IN THIS AZURE TO CONTAINER REGISTRY**
3. **CREATE AN APP SERVICE TO THE DEPLOY THE SAME IMAGE**
4. 

ENABLE ADMIN USER



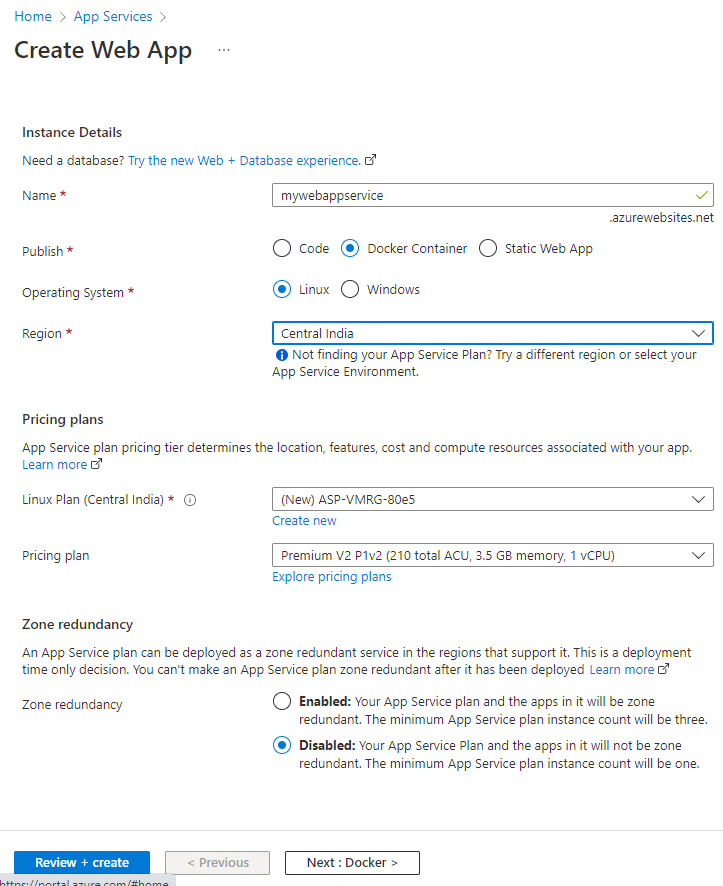
1. COMMITING THE CHANGES (Changes done in Example -1)

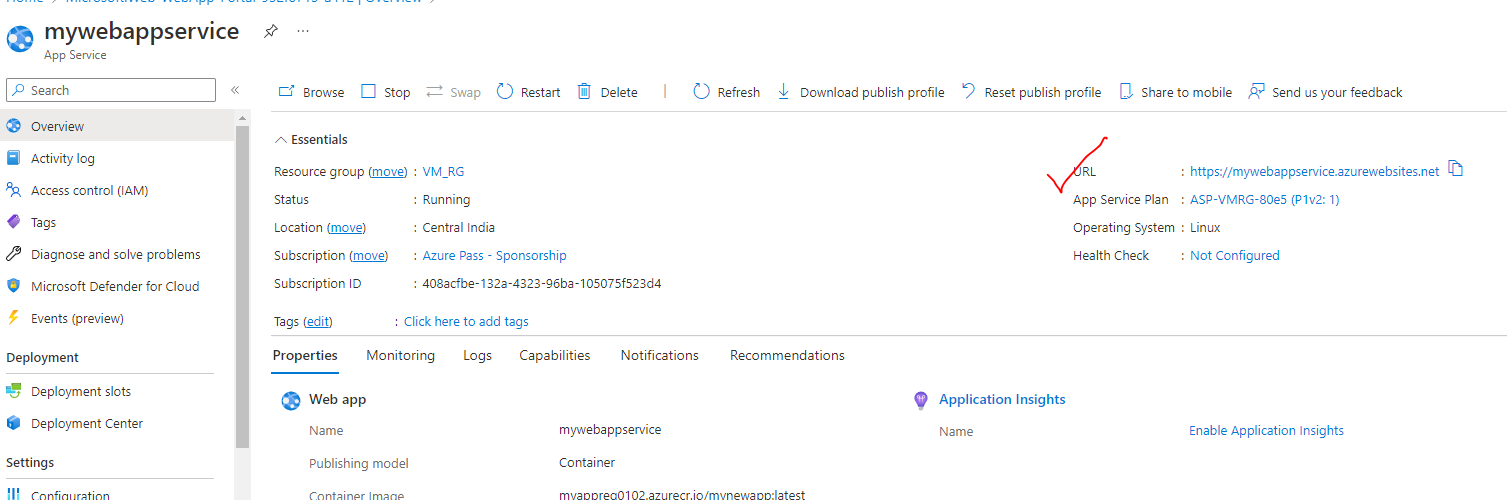
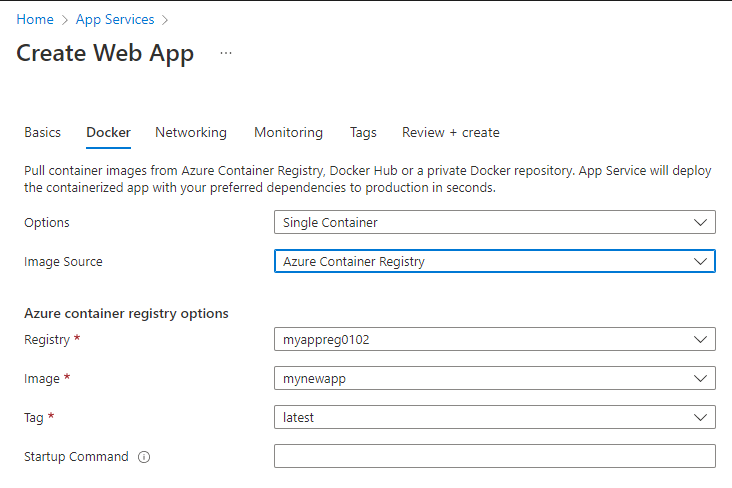


UPLOAD THE IMAGE YOU CREATED IN THIS AZURE TO CONTAINER REGISTRY

|  |  |
| --- | --- |
| LOGIN TO ACR | sudo docker login <acr\_login\_server>  **sudo docker push myappreg0102.azurecr.io/mynewapp** |
| PUSH IMAGE TO ACR | **sudo docker push myappreg0102.azurecr.io/mynewapp** |
| IMAGE IN ACR |  |

1. CREATE AN APP SERVICE TO THE DEPLOY THE SAME IMAGE





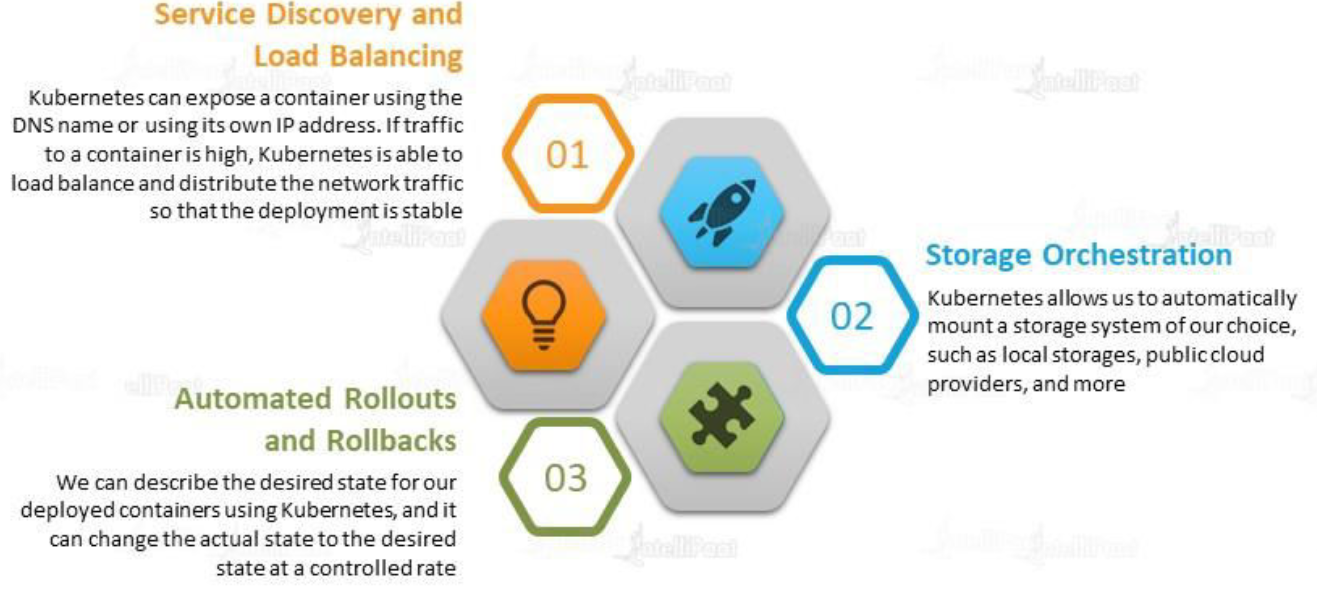
ACCESSING WEB APP

|  |  |
| --- | --- |
| URL – <https://mywebappservice.azurewebsites.net/> |  |

# AZURE KUBERNETES SERVICES

* Kubernetes is an open-source orchestration software for deploying, managing, and scaling containers
* Kubernetes manage the containers that run out application and ensure there is no downtime
* It provides us a framework to run the distributed systems resiliently.
* Azure Kubernetes is the managed version of Kubernetes tool itself

## KUBERNETES FEATURES



## KUBERNETES ARCHITECTURE

### NODES

|  |  |
| --- | --- |
|  | * A node is a machine – physical or virtual – on which Kubernetes is installed. * A node is a worker machine, and this is where containers will be launched by Kubernetes) It was also known as Minions in the past). * But what if the node on which our application is running fails? Well, obviously our application goes down. So, you need to have more than one node. Then comes the concept of Cluster |

### CLUSTER

|  |  |
| --- | --- |
|  | * A cluster is a set of nodes grouped together. * This way even if one node fails the application still accessible from the other nodes. * Having multiple nodes helps in sharing load as well. |

### MASTER



Master node is Kubernetes are

* Is responsible for managing the cluster
* Master has the information about the members of the cluster stored
* Monitoring the Nodes – For example - when a node fails it moves the workload of the failed node to another worker node

**The master is another node with Kubernetes installed in it and is configured as a Master. The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes.**

### COMPONENTS OF KUBERNETES



When we install Kubernetes on a System, following components get installed

1. AN API SERVER.

* The API server acts as the front-end for Kubernetes. **The users, management devices, Command line interfaces (kubectl)all talk to the API server to interact with the Kubernetes cluster.**

1. AN ETCD SERVICE.

* ETCD key store is a distributed reliable key-value store used by Kubernetes to store all data used to manage the cluster. For example - when we have multiple nodes and multiple masters in our cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner.
* ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters.
* It stores the status information of the cluster

1. A KUBELET SERVICE.

* Kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.
* To manage the node - Kubelet is an agent through with the nodes are managed

1. A CONTAINER RUNTIME

* The container runtime is the underlying software that is used to run containers. For example - Docker

1. CONTROLLERS

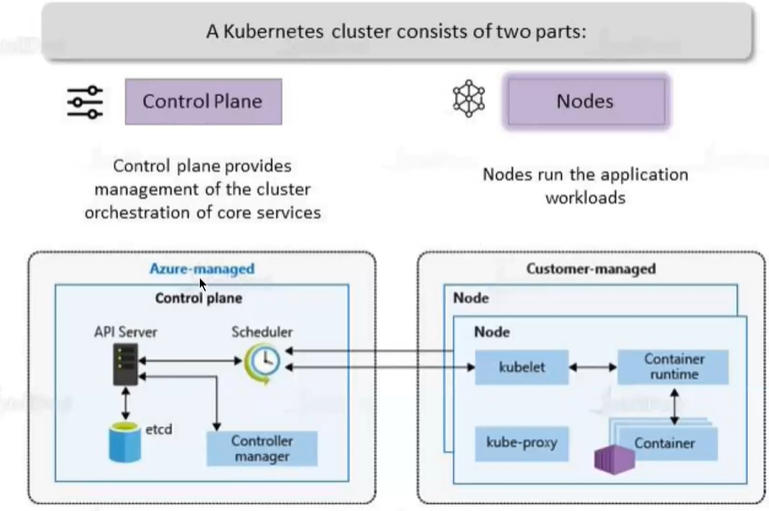
* The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers make decisions to bring up new containers in such cases.

1. SCHEDULERS.

* The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.
* For example – If we fire a “kubectl” command to create new containers – it will intercepted API server and then sent over to the scheduler to schedule the task of container creation.

### KUBERNETES IN AZURE

* Among the different components of Kubernetes – Some are Azure managed, and some are customer managed.
  + Azure Managed Kubernetes components are called “*Control Plane*”
  + Customer Managed Kubernetes components are called “Nodes”



KUBE PROXY

* This allow us to configure networking like load balancing, traffic management etc..

|  |  |
| --- | --- |
|  | * The master / worker nodes are basically VM behind the scene. * All the VMs are managed by Azure itself |

### CREATING THE KUBERNETES SERVICE

* Search for Kubernetes service in the Azure Portals 🡪 Create

|  |  |
| --- | --- |
| NODE SIZE | Size in Underlying VM |
| NODE COUNT RANGE | Minimum and Maximum number of nodes |
|  |  |

* To deploy a container in Kubernetes cluster we can use YAML file
* The source of the image can be the ACR or Docker Hub

