Table of Contents

[AZURE CONTAINER SERVICE 2](#_Toc150546385)

[WHAT AND WHY CONTAINERIZATION? 2](#_Toc150546386)

[VMs VERSUS CONTAINERS 3](#_Toc150546387)

[DOCKER 4](#_Toc150546388)

[KEY CONCEPTS RELATED TO DOCKER 4](#_Toc150546389)

[INSTALLING DOCKER RUNTIME IN LINUX VM 5](#_Toc150546390)

[EXAMPLE 5](#_Toc150546391)

[RUNNING NGINX CONTAINER IN LINUX VM 7](#_Toc150546392)

[AZURE CONTAINER REGISTRY(ACR) 7](#_Toc150546393)

[CREATING AN ACR 8](#_Toc150546394)

[ENABLE ADMIN USER 9](#_Toc150546395)

[PUBLISH A DOCKER IMAGE IN ACR 10](#_Toc150546396)

[PUBLISH DOCKER IMAGE TO ACR FROM LINUX VM USING AZURE CLI 10](#_Toc150546397)

[CONTAINER INSTANCE 11](#_Toc150546398)

[CREATING A CONTAINER INSTANCE 12](#_Toc150546399)

[EXAMPLE 14](#_Toc150546400)

[EXAMPLE 2 18](#_Toc150546401)

[MULTISTAGE BUILDS 21](#_Toc150546402)

[AZURE CONTAINER GROUP 21](#_Toc150546403)

[AZURE KUBERNETES SERVICES 21](#_Toc150546404)

[KUBERNETES FEATURES 21](#_Toc150546405)

[KUBERNETES ARCHITECTURE 21](#_Toc150546406)

[NODES 21](#_Toc150546407)

[CLUSTER 22](#_Toc150546408)

[MASTER 22](#_Toc150546409)

[COMPONENTS OF KUBERNETES 23](#_Toc150546410)

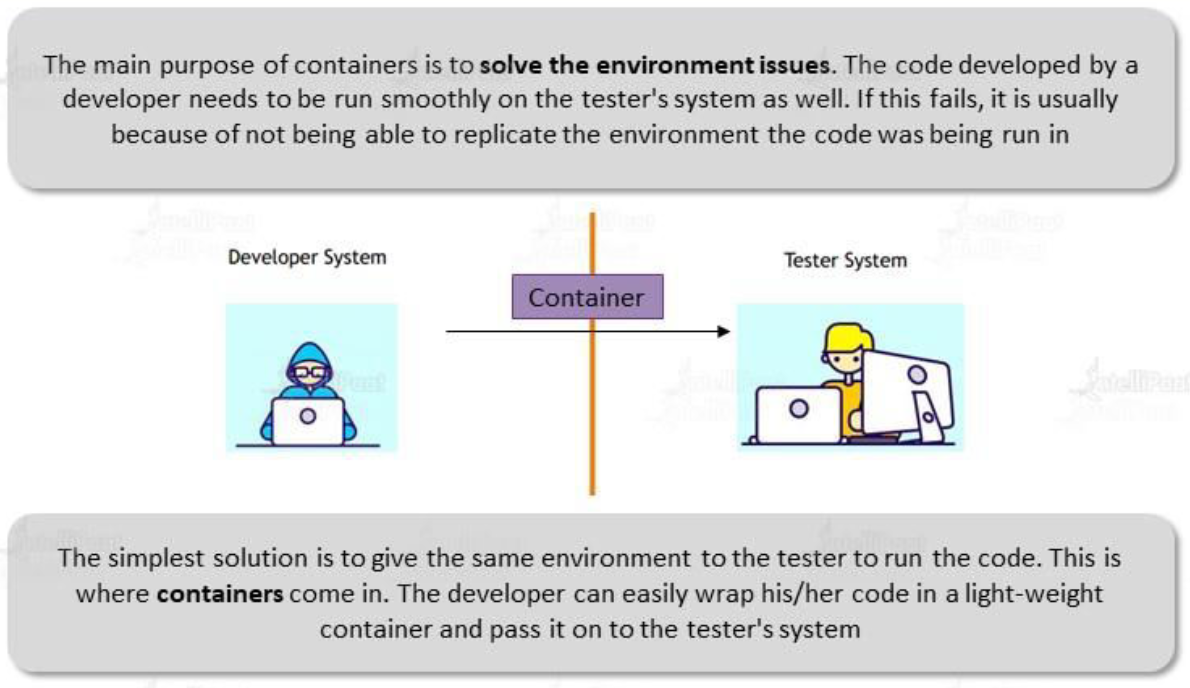
[KUBERNETES IN AZURE 24](#_Toc150546411)

[AKS: A USE CASE 25](#_Toc150546412)

[CREATING THE KUBERNETES SERVICE 25](#_Toc150546413)

# AZURE CONTAINER SERVICE

## WHAT AND WHY CONTAINERIZATION?

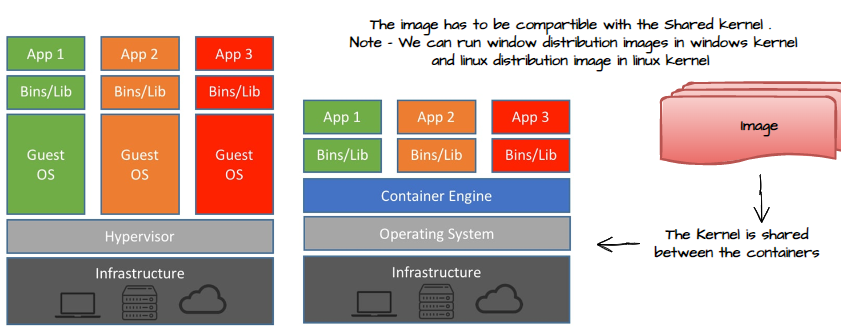


There are several advantages of using containers in software development and deployment:

1. **PORTABILITY**
   1. Containers provide a lightweight and portable way to package applications and their dependencies.
   2. They encapsulate the application and its dependencies into a single unit, making it easy to run consistently across different environments, such as development, testing, and production.
   3. This portability allows for smoother deployments and reduces the chances of compatibility issues.
2. **ISOLATION**
   1. Containers provide a high level of isolation between applications and their environments. Each container runs independently, with its own file system, network stack, and process space.
   2. This isolation ensures that applications and their dependencies do not interfere with each other, improving security and stability.
3. **SCALABILITY**
   1. Containers enable easy scalability of applications. They can be quickly replicated and deployed across different nodes or servers, allowing for horizontal scaling.
   2. This means that as the demand for an application increase, more containers can be spun up to handle the increased load, providing better performance and responsiveness.
4. **EFFICIENCY**

|  |  |
| --- | --- |
|  | * Containers are lightweight and have minimal overhead. They share the host system's operating system kernel, which reduces resource consumption compared to running multiple virtual machines. * This efficiency translates to faster startup times, lower memory usage, and improved resource utilization. |

### VMs VERSUS CONTAINERS



Sure! Here's a comparison table highlighting the key differences between Virtual Machines (VMs) and containers:

|  |  |  |
| --- | --- | --- |
| **ASPECT** | **VIRTUAL MACHINES (VMS)** | **CONTAINERS** |
| **VIRTUALIZATION LEVEL** | Full hardware virtualization | Operating system-level virtualization |
| **RESOURCE OVERHEAD** | Higher resource overhead | Lower resource overhead |
| **OPERATING SYSTEM** | Each VM has its own complete OS | Containers share the host OS |
| **ISOLATION** | Strong isolation between VMs | Containers share the host OS, less isolation |
| **STARTUP TIME** | Longer startup time | Faster startup time |
| **DENSITY** | Lower density, fewer instances per host | Higher density, more instances per host |
| **PORTABILITY** | Less portable, more dependent on the host OS | Highly portable, easily moved between hosts |
| **SECURITY** | Stronger security boundaries | Less security boundaries, shared host OS |
| **DEPLOYMENT SPEED** | Slower deployment due to OS booting | Faster deployment due to lightweight nature |
| **USE CASES** | Running different OSes, legacy applications | Microservices, cloud-native, CI/CD workflows |

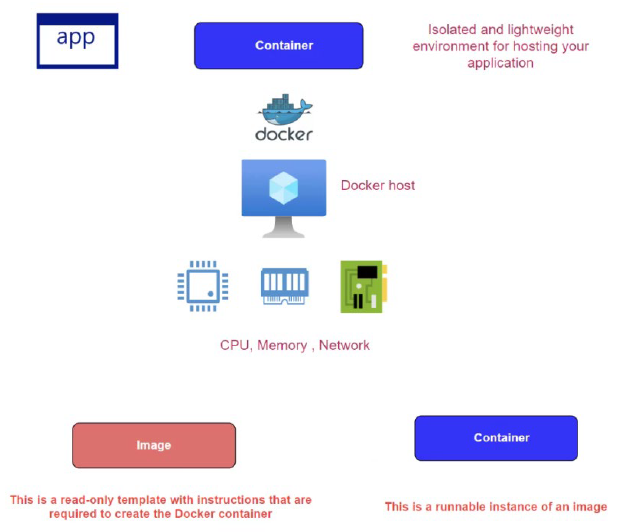
## DOCKER

* Docker is an open-source platform that **enables developers to automate the deployment, scaling, and management of applications using containerization**. It provides an ecosystem of tools and services that simplify the process of creating, distributing, and running containers.
* Docker allows applications to be packaged into containers, which are lightweight and isolated environments that encapsulate an application and its dependencies. **These containers can run consistently across different environments, such as development, testing, and production, without worrying about compatibility issues.**

### KEY CONCEPTS RELATED TO DOCKER

1. DOCKER IMAGE
   1. A Docker image is a read-only template that contains all the necessary files, libraries, and dependencies required to run an application.
   2. Images are built using a declarative Dockerfile, which specifies the steps to create the image.
2. CONTAINER
   1. A container is an instance of a Docker image. It is a runnable environment that isolates the application and its dependencies.
   2. Containers are lightweight, portable, and can be easily started, stopped, or moved between different hosts.
3. DOCKER REGISTRY
   1. Docker images can be stored and shared in a Docker registry, such as **Docker Hub or private registries**.
   2. Registries allow users to download and distribute images, making it easier to collaborate and share applications.
4. DOCKER ENGINE
   1. Docker Engine is the runtime that runs and manages Docker containers. It provides the necessary infrastructure and tools to create, start, stop, and manage containers.
   2. Docker Engine can run on various operating systems, including Linux, Windows, and macOS.
5. DOCKER COMPOSE
   1. Docker Compose is a tool used for defining and managing multi-container applications.
   2. It allows developers to define a multi-container application using a YAML file and then spin up all the required containers with a single command.
6. DOCKER SWARM
   1. Docker Swarm is a native clustering and orchestration solution provided by Docker.
   2. It allows users to create and manage a swarm of Docker nodes, enabling the deployment and scaling of containers across multiple hosts.

As mentioned above - to deploy an application has a container on underlying VM. We need to install the Docker runtime. This Docker runtime 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.



### INSTALLING DOCKER RUNTIME IN LINUX VM

* STEP 1:Provision a Linux VM
* STEP 2: Connect with VM using Putty tool or Cloud Shell.
* 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 OM VM.
2. PULL HSHAR/WEBAPP (https://hub.docker.com/r/hshar/webapp ) REPOSITORY
3. CREATE NEW FILE IN THIS REPOSITORY

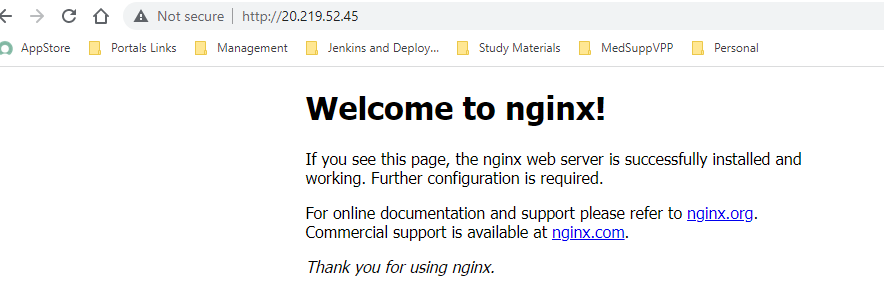
To install Docker Runtime on a Linux VM- Run the following command

|  |  |
| --- | --- |
| 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 | 1. Navigate to webroot directory – **cd /var/www/html** 2. Open the Editor to add the file **- nano index.html** 3. **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***



## IMAGE REGISTRY

* An image registry is a centralized repository for storing and sharing Docker images.
* It serves as a reliable and scalable storage solution, allowing developers to upload, download, and distribute container images.
* The most well-known image registry is Docker Hub, but there are also other options **like Amazon Elastic Container Registry (ECR), Google Container Registry, and private registries that can be set up within organizations**.

### NEED AN IMAGE REGISTRY:

1. IMAGE DISTRIBUTION
   1. An image registry acts as a central hub for storing and distributing Docker images.
   2. It provides a platform where developers can upload their images, making them accessible to other team members or external users.
2. VERSION CONTROL
   1. Image registries allow for version control of Docker images.
   2. Developers can tag and manage different versions of their images, ensuring traceability and enabling easy rollback to previous versions if needed.
3. SECURITY AND POLICY MANAGEMENT
   1. Image registries provide security features like access control and image scanning.
   2. They allow administrators to define access policies, ensuring that only authorized users can access and download images.
4. PERFORMANCE AND CACHING
   1. Once an image is downloaded from the registry, it can be cached locally, reducing the time and network bandwidth required for subsequent downloads. This is particularly useful when deploying containers on multiple hosts or in resource-constrained environments.
5. INTEGRATION WITH CI/CD PIPELINES
   1. Image registries integrate seamlessly with continuous integration/continuous deployment (CI/CD) pipelines.
   2. Developers can configure their CI/CD workflows to automatically build, test, and push Docker images to the registry.
   3. This enables streamlined and automated software delivery, ensuring that the latest versions of applications are readily available for deployment.

# AZURE CONTAINER REGISTRY(ACR)



## KEY FEATURES AND BENEFITS OF ACR

* PRIVATE DOCKER REGISTRY
  + ACR provides a private and secure registry for storing Docker container images.
  + It enables users to control access to their images and restrict visibility to only authorized users or specific Azure resources.
* INTEGRATION WITH AZURE SERVICES
  + ACR seamlessly integrates with other Azure services, such as Azure Kubernetes Service (AKS), Azure Container Instances (ACI), and Azure Functions.
  + This allows users to easily deploy and manage their containerized applications using these services.
  + ACR integrates well with Azure DevOps, Azure Pipelines, and other CI/CD tools. This allows for seamless integration of container image building, testing, and deployment processes within the Azure ecosystem
* SCALABILITY AND AVAILABILITY
  + ACR offers high scalability and availability, allowing users to store and distribute large numbers of container images.
  + It leverages Azure's global infrastructure to ensure reliable and fast access to images from different regions.
* SECURITY AND COMPLIANCE
  + ACR supports features like role-based access control (RBAC), Azure Active Directory (AAD) integration, and network restrictions to enhance security.
  + It also integrates with Azure Security Center to provide vulnerability scanning and image signing capabilities.
* GEO-REPLICATION
  + ACR provides geo-replication options to replicate container images across different Azure regions.
  + This ensures redundancy and enables faster image access for users in different geographic locations.
* MONITORING AND INSIGHTS
  + ACR offers monitoring capabilities, including container image pull and push metrics, which can be integrated with Azure Monitor for deeper insights and analytics.

|  |  |
| --- | --- |
|  | * 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** |

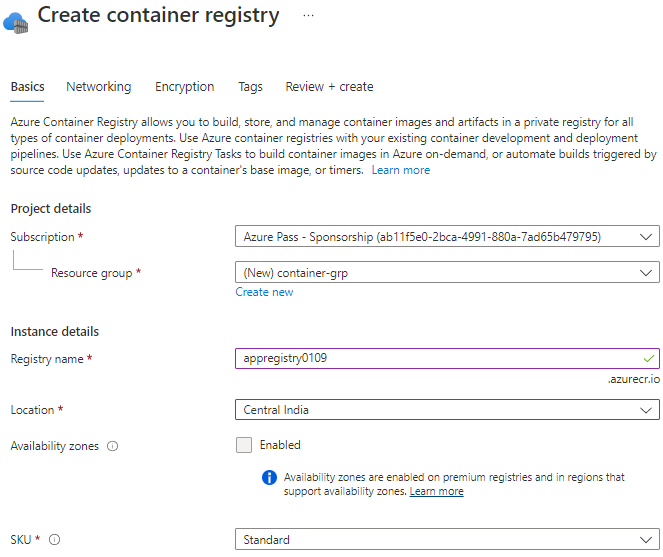


To create a container based out of the images pulled from ACR we need to go through following steps

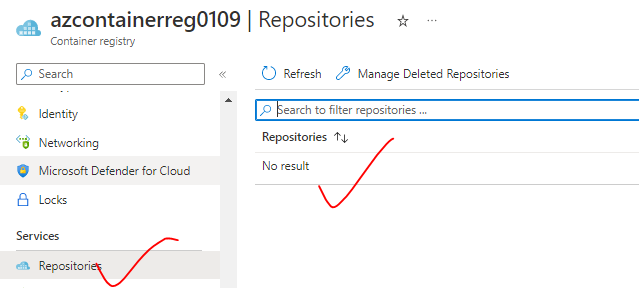
1. CREATE ACR USING AZURE’S ACR SERVICE
2. PUBLISH AN IMAGE TO THE ACR
3. CREATE A CONTAINER INSTANCE FROM AZURE’S CONTAINER INSTANCE SERVICE, THEN CREATE A CONTAINER BASED ON THE IMAGE IN ACR.

## CREATING AN ACR

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

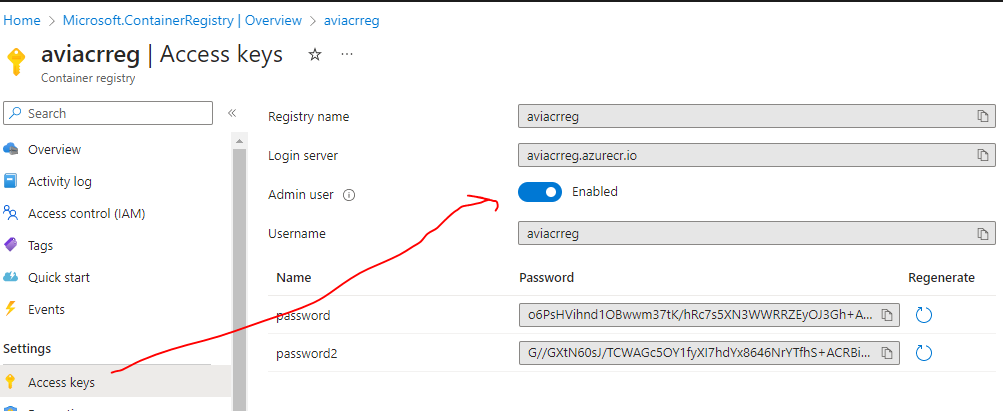


* Initially there will be no image present in the registry



### ENABLE ADMIN USER

* When we enable the admin user, it provide us the admin credential to login in ACR and publish images



## 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>

Note

* The docker images can be built locally or on a VM
* Images can be pushed to ACR using Powershell or Azure CLI command

### STEP 1: CREATING A DOCKER IMAGE

|  |  |
| --- | --- |
| SYNTAX | d |
| EXAMPLE | docker build -t node-web-app . |
| TO VIEW DOCKER IMAGES | docker images |

BREAK DOWN OF docker build COMMAND

* **-t <image\_name>:<tag>:**  specifies the name and tag for the image. Replace <image\_name> with the desired name for your image, and <tag> with a version or label for the image.
* **<path\_to\_dockerfile\_directory>**
  + Indicates the directory where the Dockerfile is located.
* For example, if Dockerfile is in the current directory and we want to create an image named "myapp" with the tag "v1.0", the command would look like this: **docker build -t myapp:v1.0** .

Note that the dot (.) at the end of the command represents the current directory.

|  |  |
| --- | --- |
| 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 TO ACR

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

#### AZ COMMAND TO PUSH AN IMAGE TO ACR

To push an image to an Azure Container Registry (ACR) using the Azure CLI (az command), We can follow these steps:

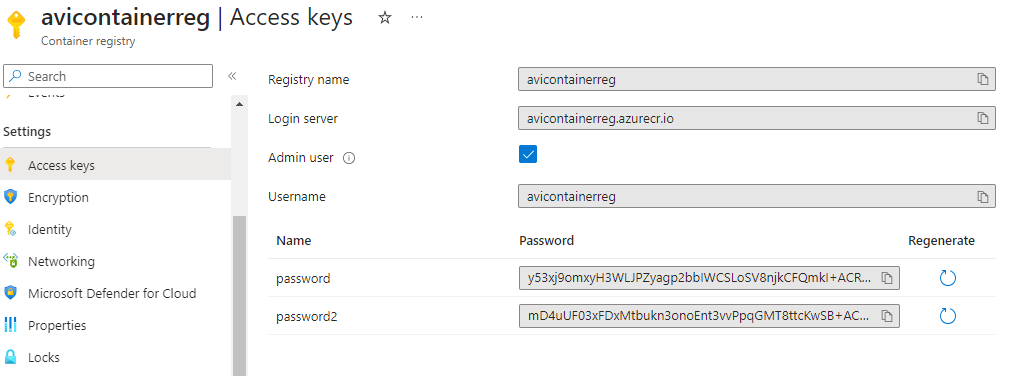
* Make sure you have the Azure CLI installed and logged in to your Azure account (**az login**)

|  |  |
| --- | --- |
| * Build Docker image using the docker build command | **docker build -t <image-name> .** |
| * Tag the Docker image with the ACR login server address. Replace <acr-name> with the name of the ACR and <image-name> with the name of the Docker image | **docker tag <image-name> <acr-name>.azurecr.io/<image-name>:<tag>** |
| * Log in to Azure Container Registry using the Azure CLI: * For The username and password 🡪 ACR 🡪 Access Keys🡪 Enable Admin User | **az acr login --name <acr-name> -u <username> -p <password>**  **Example : az acr login --name avirepo -u avirepo -p 8JxJqOU0o4kw2N7SDqfextZRUEehOJVqVGxNkZKXHU+ACRAplsBg** |
| * Push the Docker image to the ACR using the docker push command: | **docker push <acr-name>.azurecr.io/<image-name>:<tag>** |

*Note: Make sure to replace <acr-name> with the name of the Azure Container Registry, <image-name> with the name of the Docker image, and <tag> with the desired tag for the image.*

### PUBLISH DOCKER IMAGE TO ACR FROM LINUX VM USING AZURE CLI

1. Step 1: Install Azure CLI on Linux VM: <https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=apt>
   1. Command: **sudo apt install azure-cli**
2. Step 2: Login to ACR using admin credential:



|  |  |
| --- | --- |
| **FROM LOCAL MACHINE** | **az acr login --name <name\_of\_acr> -u <username>-p <password>**  **Example** : az acr login --name avicontainerreg -u avicontainerreg -p y53xj9omxyH3WLJPZyagp2bbIWCSLoSV8njkCFQmkI+ACRCwd7iQ |
| **LINUX MACHINE** | **sudo az acr login --name aviacrreg -u <username>-p <password>** |

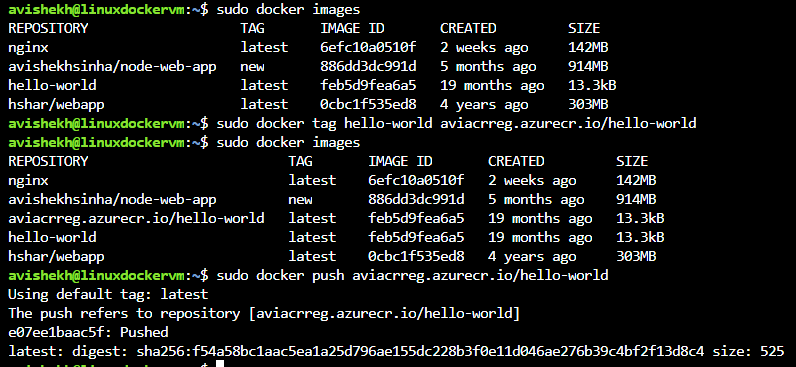


1. Step 3: Tag the image (optional):
   1. **sudo docker tag hello-world aviacrreg.azurecr.io/hello-world**
   2. **sudo docker tag avishekhsinha/node-web-app:new aviacrreg.azurecr.io/node-web-app**

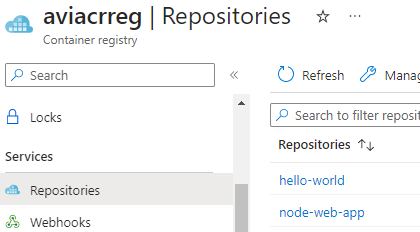
|  |
| --- |
| The docker tag command is used to create a new tag for an existing Docker image. This command allows you to assign a different name or version to an image, making it easier to manage and reference the image. Here's the syntax for the docker tag command:  docker tag <source\_image>[:<source\_tag>] <target\_image>[:<target\_tag>]   Let's break down the command:   * <source\_image>: This is the name or ID of the source image you want to create a new tag for. * <source\_tag>: (Optional) This is the tag associated with the source image. If not specified, it defaults to "latest". * <target\_image>: This is the name or ID of the new image with the desired tag you want to create. * <target\_tag>: This is the tag you want to assign to the target image. It can be any desired name or version.   Here's an example to illustrate how to use the docker tag command:  docker tag myapp:latest myapp:v1.0  In this example, we are creating a new tag for the "myapp" image. The original image has the tag "latest", and we are assigning the new tag "v1.0" to it. After running this command, you will have a new image named "myapp" with the tag "v1.0". |

1. Step 4: Push the image to ACR
   1. **sudo docker push aviacrreg.azurecr.io/hello-world**

b . **sudo docker push aviacrreg.azurecr.io/node-web-app**

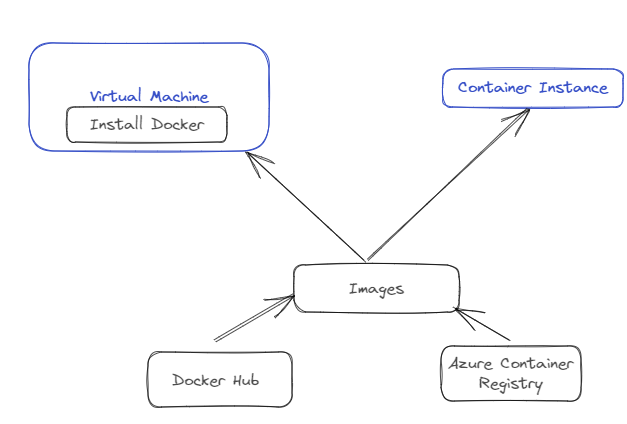


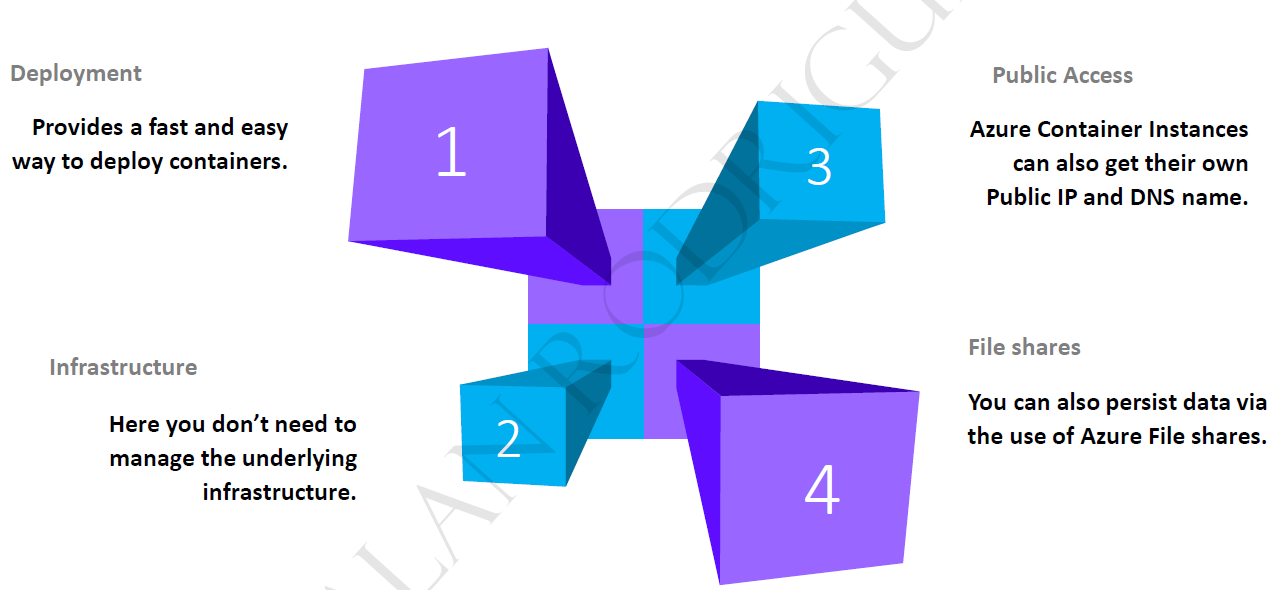
DOCKER IMAGES IN ACR



# AZURE CONTAINER INSTANCE

* Azure Container Instances (ACI) is a serverless offering from Microsoft Azure that allows us to run containers without managing the underlying infrastructure.
* In a typically set up – we need to install docker on a VM and then run a container in the docker runtime. But in case of container instance, we can pull out the image from the Azure container registry other registries like docker-hub and deploy a container. ACI provides a fast and simple way to deploy and run containers in Azure, without the need to provision or manage virtual machines.
* Note: **ACI is suitable for running individual containers or small-scale container workloads. If we require more advanced orchestration features, scalability, and lifecycle management, we must consider using Azure Kubernetes Service (AKS) instead**.





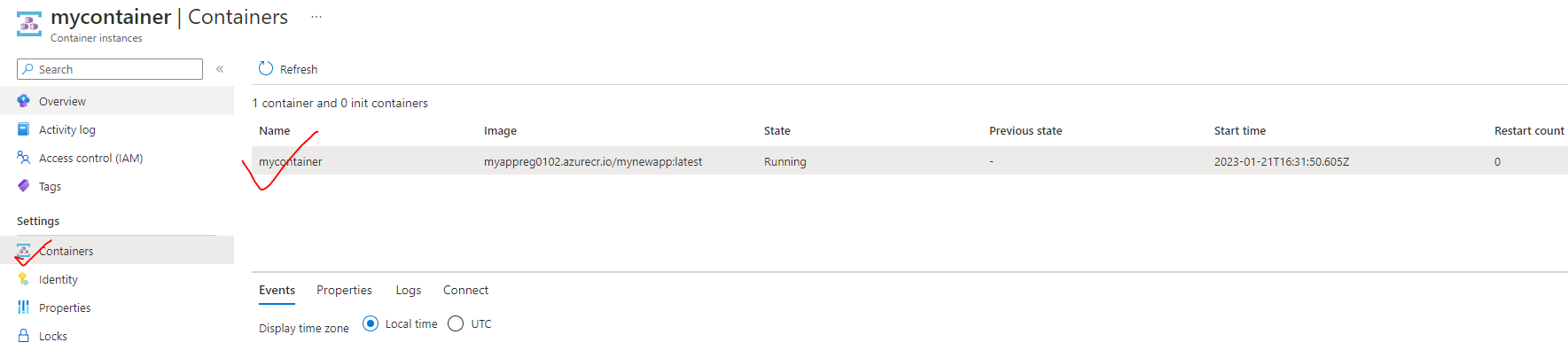
## CREATING A CONTAINER INSTANCE

|  |  |
| --- | --- |
|  | * Container are the instance of image. The container can be created using “Container Instance” Service * The container 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. |

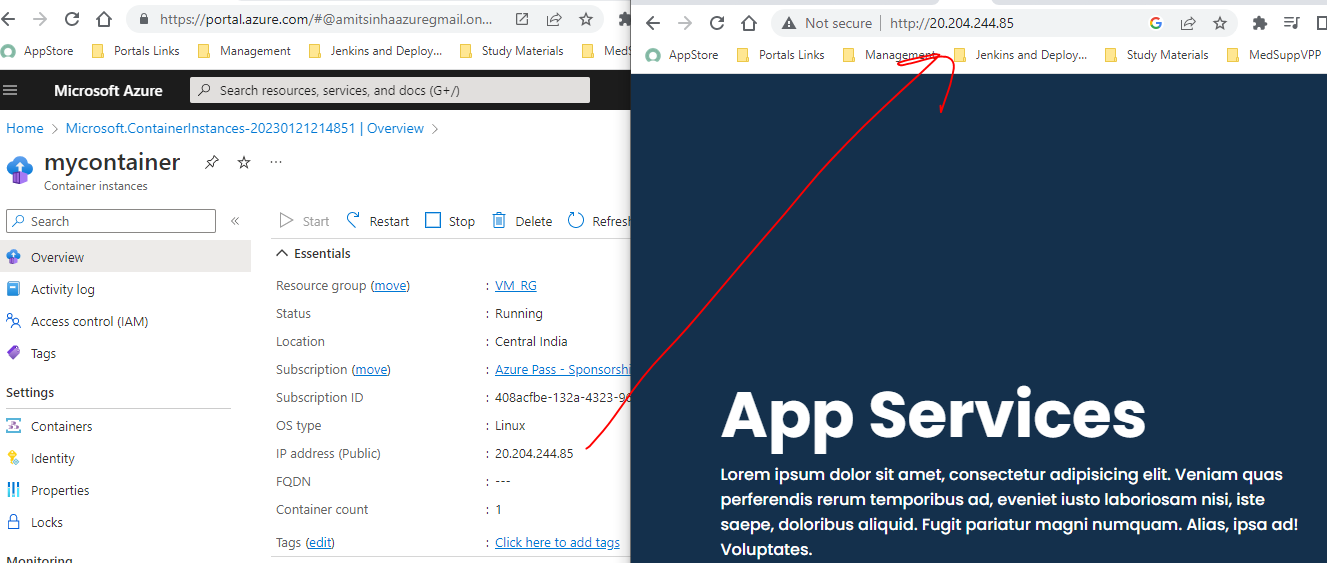
**CONTAINER CAN BE CREATED FROM MULTIPLE SOURCE OF IMAGE**

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

CONTAINER INSTANCE HAS A CONTAINER RUNNING FROM A GIVEN IMAGE

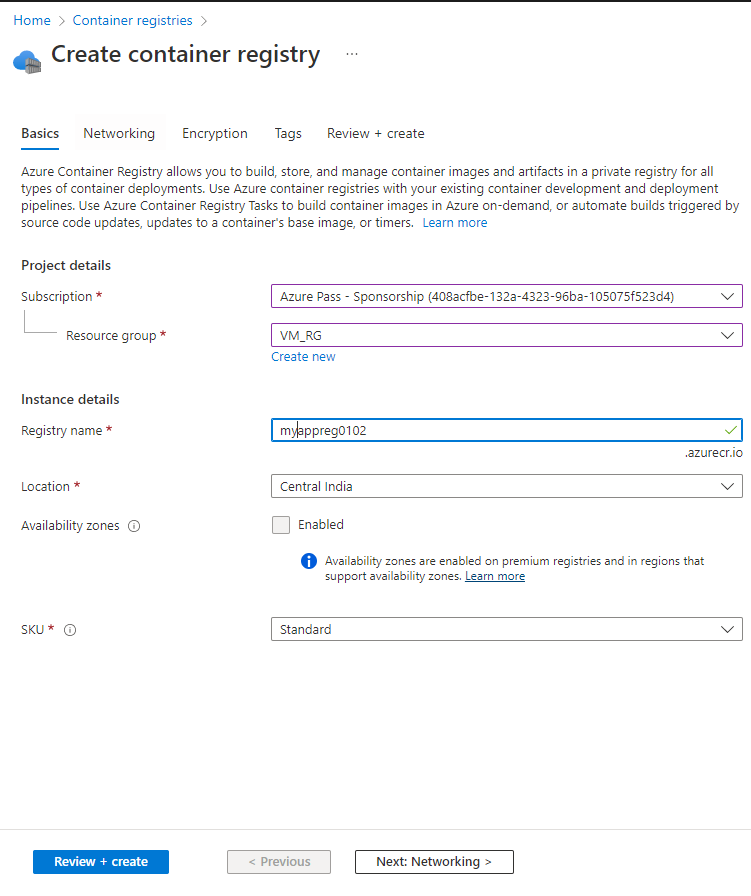


TO ACCESS THE APPLICATION – We use public IP address of the container intance.

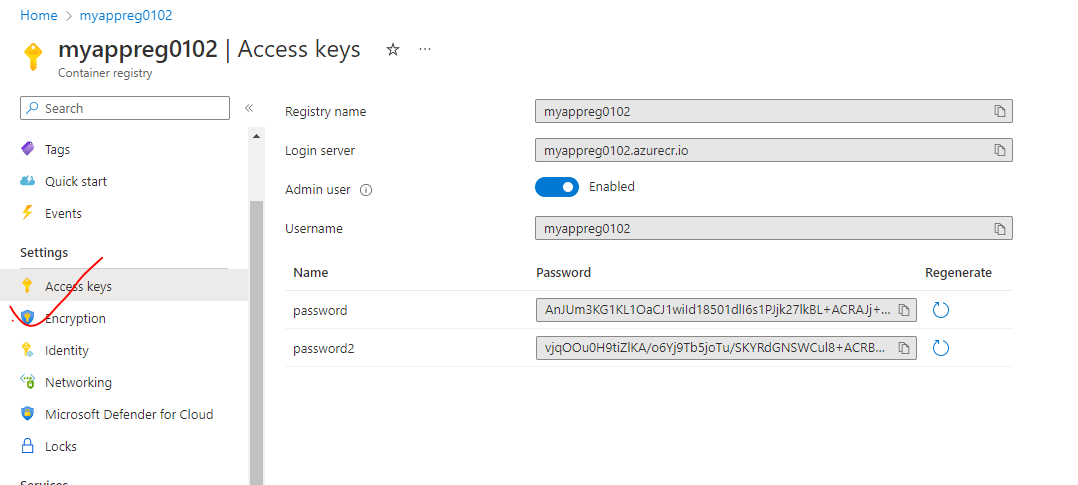


## EXAMPLE

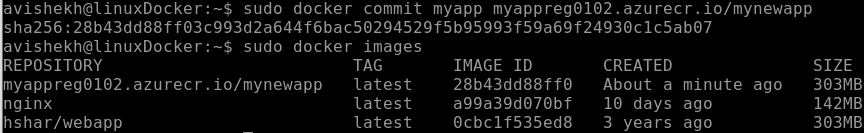
1. **CREATE AZURE CONTAINER REGISTRY AND CONNECT IT TO DOCKER RUNNING IN VM.**
2. **UPLOAD THE IMAGE IN THIS AZURE TO CONTAINER REGISTRY**
3. **CREATE AN APP SERVICE TO THE DEPLOY THE SAME IMAGE**



ENABLE ADMIN USER



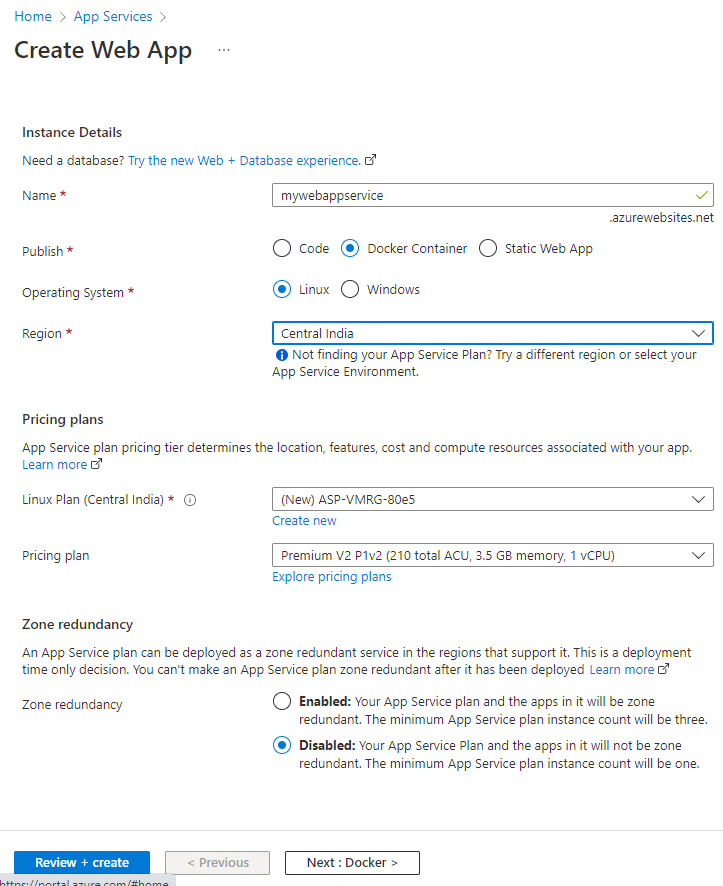
1. COMMITING THE CHANGES (Changes done in Example -1)

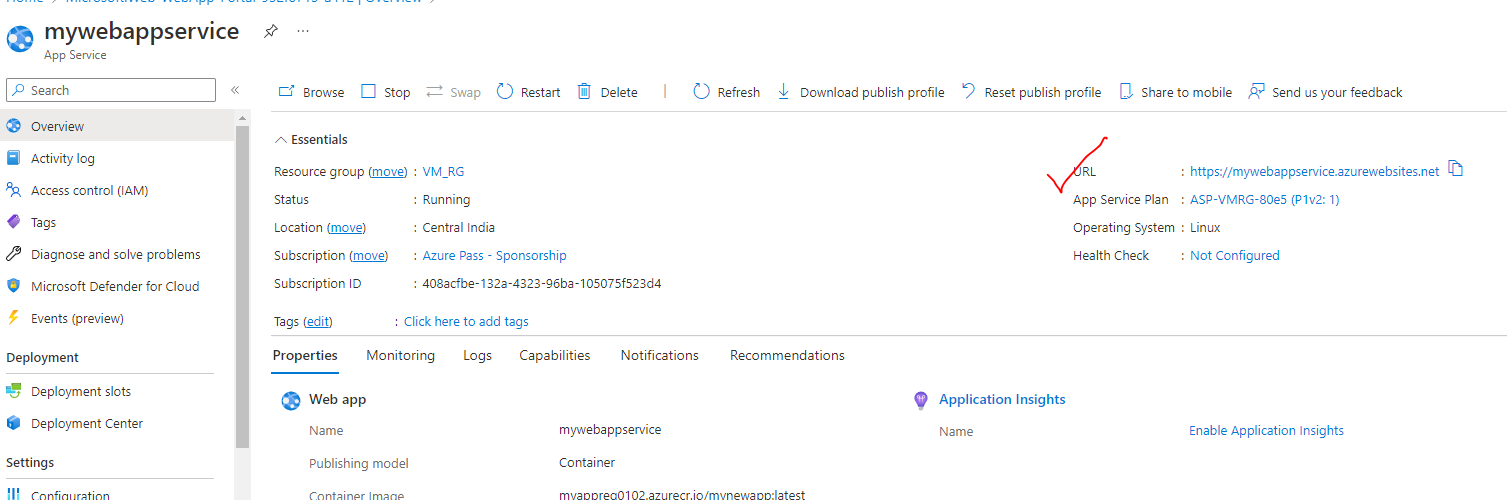
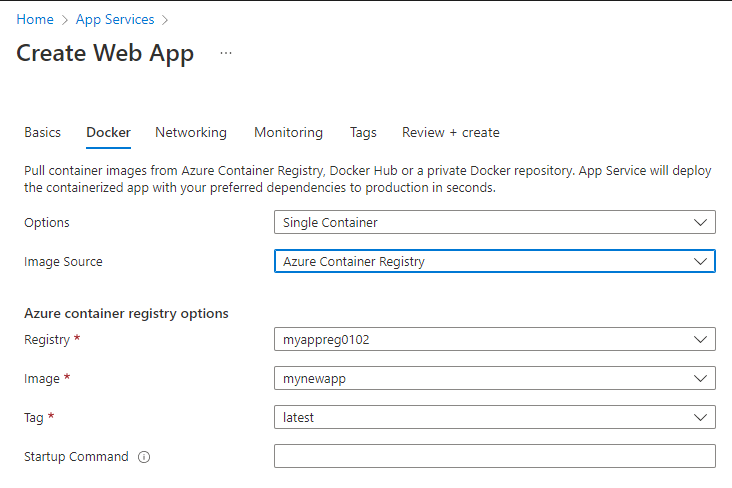


UPLOAD THE IMAGE YOU CREATED IN THIS AZURE TO CONTAINER REGISTRY

|  |  |
| --- | --- |
| **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

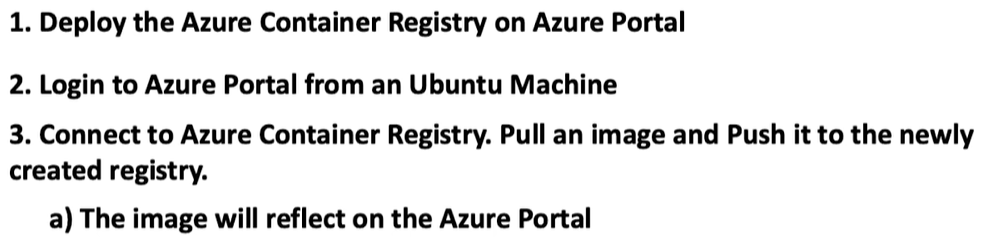




ACCESSING WEB APP

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

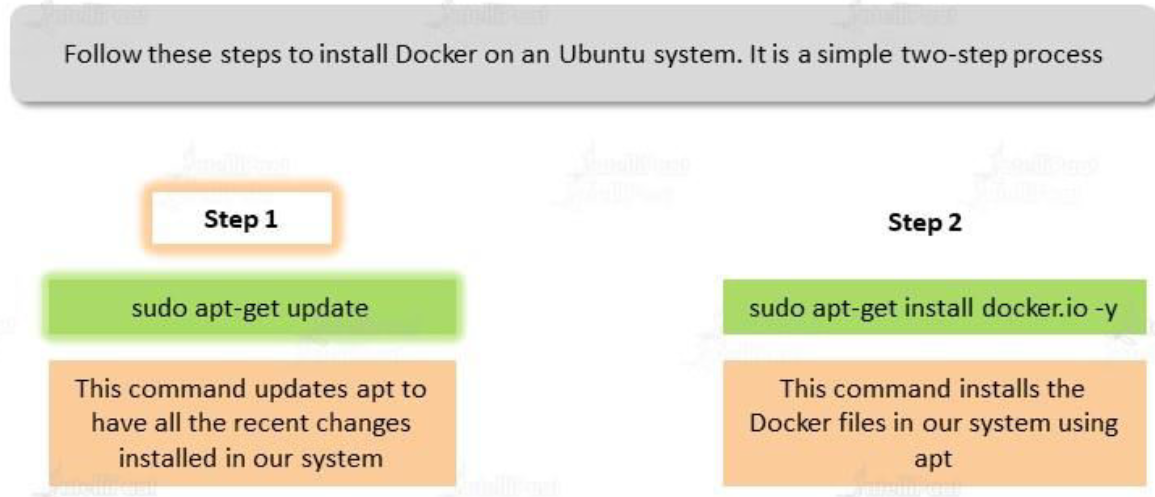
## EXAMPLE 2



STEP 1: SET UP AN UBUNTU VM

STEP 2: INSTALL AZURE CLI (<https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=apt>)

STEP 3: INSTALL DOCKER CLI IN THE VM:



STEP 4: SET UP AZURE CONTAINER REGISTRY

|  |  |
| --- | --- |
|  |  |

STEP 5: LOGIN TO AZURE USING AZURE CLI

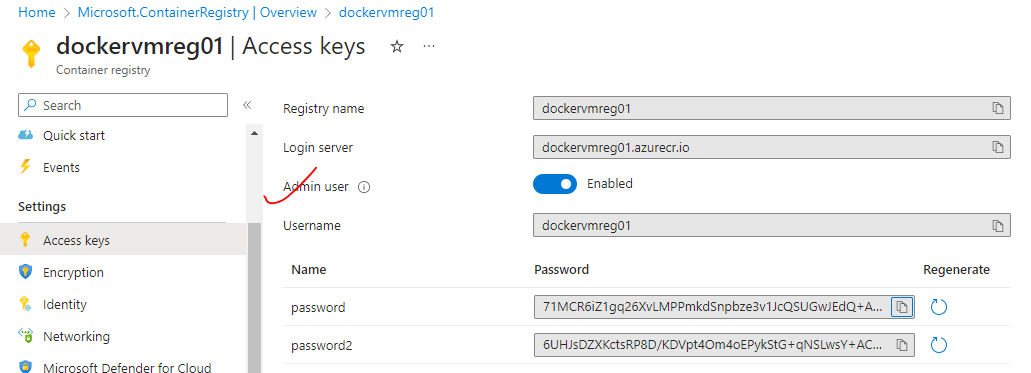
* 1. Open the below URL and enter the below code to authenticate in the Azure Portal.



|  |  |
| --- | --- |
| After authentication it will show up the following message in the terminal |  |

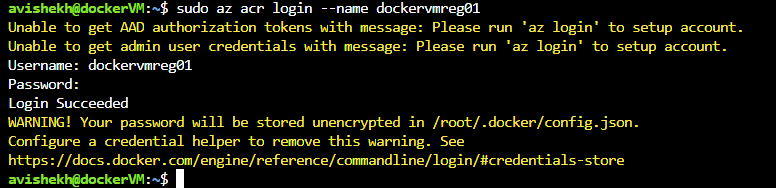
STEP 6: AFTER AUTHENTICATION 🡺CONNECT TO ACR

**ENABLE THE ADMIN USER**



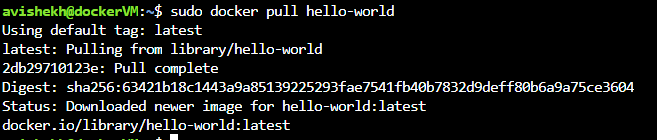
**LOGIN TO ACR THE ADMIN USER**

* **COMMAND**: sudo az acr login --name <acr\_name>
* **EXAMPLE**: sudo az acr login --name dockervmreg01



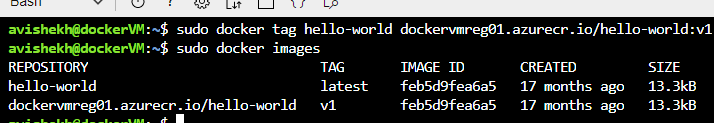
STEP 7: DOCKER OPERATION

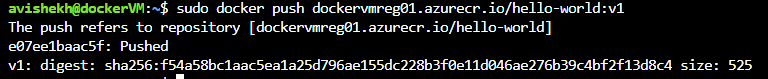
**PULL THE IMAGE FROM DOCKER HUB(ex hello-world)**

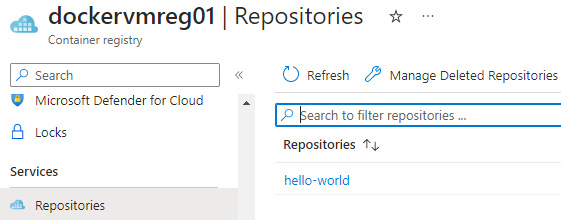


**PUSH THE IMAGE TO ACR**

1. We can tag the image as well before pushing it to ACR and then push it to ACR







## AZURE CONTAINER GROUP

* Azure Container Group is a service provided by Microsoft Azure that **allows us to run multiple containers together as a group, enabling us to deploy and manage multi-container applications easily**.
* Azure Container Group builds on top of Azure Container Instances (ACI) and provides additional features for orchestrating and managing containerized applications.

## KEY FEATURES AND BENEFITS OF AZURE CONTAINER GROUP:

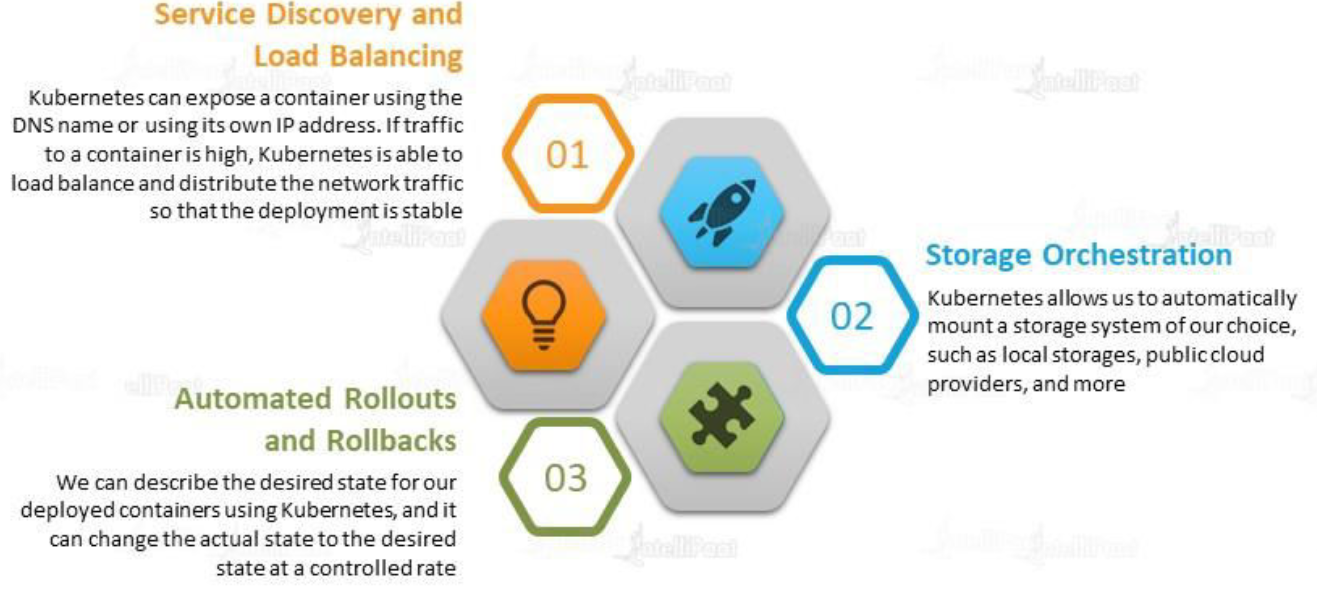
* MULTI-CONTAINER DEPLOYMENT
  + With Azure Container Group, you can deploy multiple containers together as a single unit. which makes it easier to manage and maintain complex applications that consist of multiple interconnected containers.
* APPLICATION-LEVEL SCALING
  + Azure Container Group allows us to scale the entire group of containers together, ensuring that all containers within the group scale up or down based on demand which This simplifies the scaling process and helps maintain consistency across the application.
* SHARED NETWORKING AND STORAGE
  + Containers within an Azure Container Group share the same network and storage resources, making it easier to establish communication between containers and access shared data volumes.
* CONTAINER GROUP LIFECYCLE MANAGEMENT
  + Azure Container Group provides lifecycle management capabilities, allowing you to start, stop, and restart the entire group of containers as a single entity which simplifies the management and maintenance of multi-container applications.

# AZURE CONTAINER APPS

# AZURE KUBERNETES SERVICES

* Kubernetes is an open-source orchestration software for deploying, managing, and scaling containerized applications. Azure Kubernetes Service (AKS) is the managed version of Kubernetes tool itself.

## KUBERNETES FEATURES



* MANAGED KUBERNETES
  + AKS offers a fully managed Kubernetes service, eliminating the need of we are managing the underlying Kubernetes infrastructure.
  + Azure takes care of **managing the master nodes, scaling, and upgrades**, allowing us to focus on deploying and managing the applications.
* EASY DEPLOYMENT
  + AKS makes it straightforward to deploy the applications using **Kubernetes manifests or Helm charts**.
  + We can define your desired state in YAML files and use Kubernetes native commands to deploy and manage your applications.
* SCALABILITY AND RELIABILITY
  + AKS provides automatic scaling capabilities, **allowing us to scale the application horizontally by adding or removing worker nodes based on demand.**
  + It ensures high availability by distributing the application across multiple nodes in the Kubernetes cluster.
* INTEGRATION WITH AZURE SERVICES
  + AKS integrates seamlessly with other Azure services, such as *Azure Container Registry, Azure Monitor, Azure Active Directory, Azure DevOps*, and more.
* SECURITY AND COMPLIANCE
  + AKS provides built-in security features, including role-based access control (RBAC), Azure Active Directory integration, network policies, and Azure Monitor for container security monitoring.
  + It also supports Azure Policy for enforcing compliance requirements.
* MONITORING AND DIAGNOSTICS
  + AKS integrates with Azure Monitor and Azure Log Analytics, allowing you to collect and analyze logs, metrics, and events from your Kubernetes cluster.
* CONTINUOUS INTEGRATION AND DEPLOYMENT (CI/CD) INTEGRATION
  + AKS integrates with Azure DevOps, Azure Pipelines, and other CI/CD tools, enabling us to automate the build, test, and deployment of your containerized applications.
* HYBRID AND MULTI-CLOUD SUPPORT
  + AKS supports hybrid cloud scenarios by allowing us to connect with on-premises resources to the AKS cluster.
  + It also enables multi-cloud deployments, allowing us to manage and deploy applications across Azure and other cloud providers using Azure Arc.

## KUBERNETES ARCHITECTURE

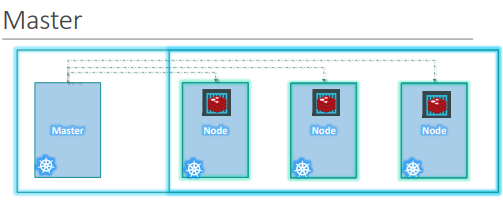
### NODES

|  |  |
| --- | --- |
|  | * 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

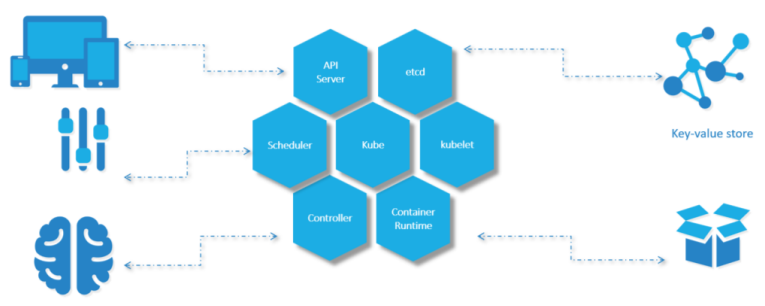


Master node is Kubernetes are

* 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



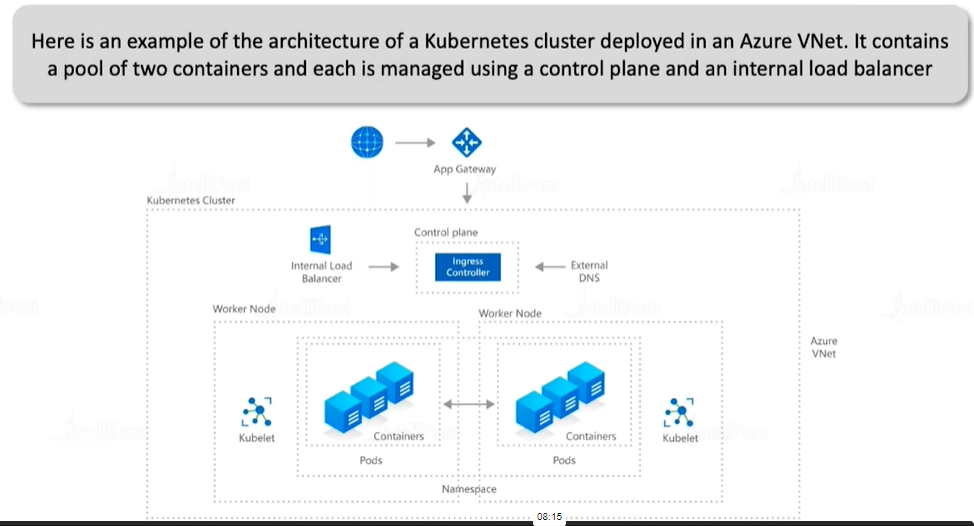
* Among the different components of Kubernetes – Some are Azure managed, and some are customer managed.
  + **CONTROL PLANE IS AZURE MANAGED**
  + **NODES” ARE CUSTOMER MANAGED**

KUBE PROXY

* This allows 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 |

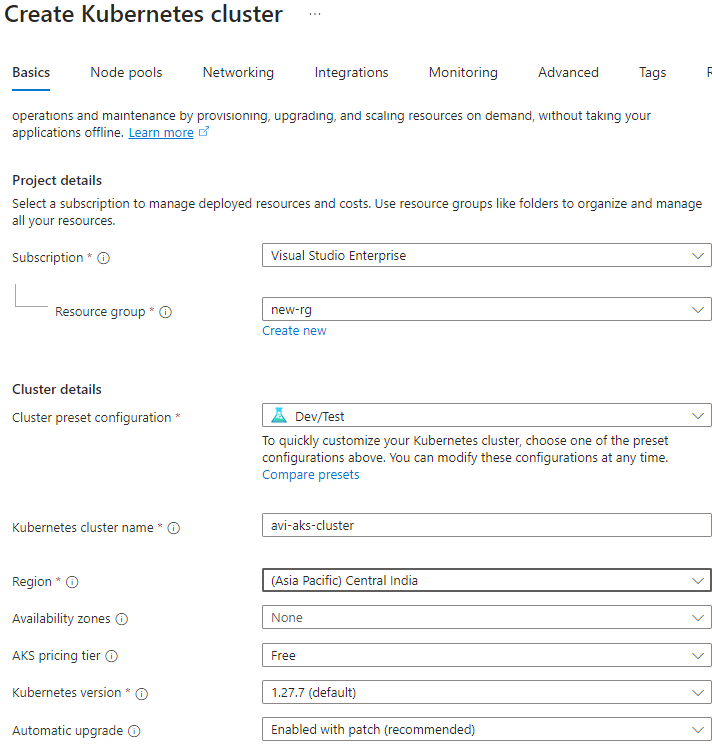
#### AKS: A USE CASE



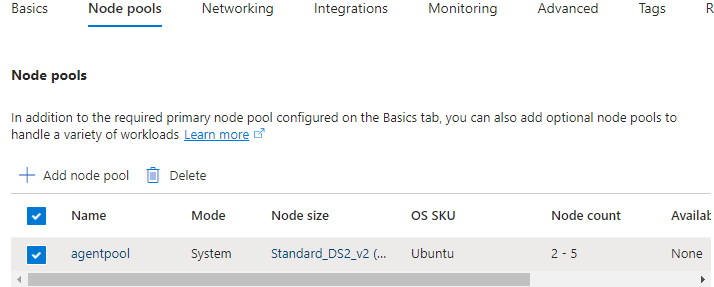
### CREATING THE KUBERNETES SERVICE

* We are creating AKS assuming that we have already have ACR (***avirepo***)
* Search for Kubernetes service in the Azure Portals 🡪 Create a Kubernetes cluster.`

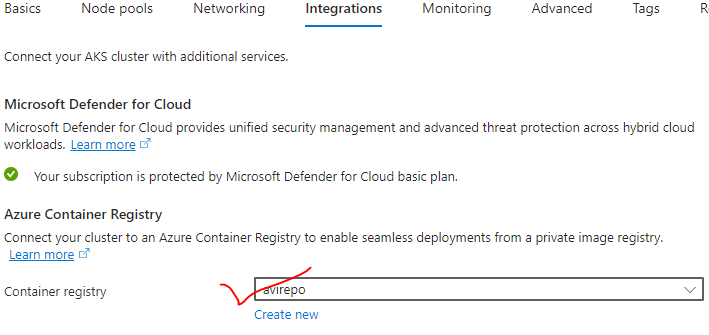
|  |  |
| --- | --- |
|  | * To deploy a container in Kubernetes cluster we can use YAML file * **The source of the image can be the ACR or Docker Hub** |



* Configure Node Pools:  *Min : 2 Nodes ; Max: 5 nodes*



* **INTEGRATE WITH EXISTING ACR**



#### NETWORKING IN AKS

In Azure Kubernetes Service (AKS), there are two different networking models available:

* **KUBENET**
* **AZURE CONTAINER NETWORKING INTERFACE (CNI).**

**Note:** The choice of networking model is made at the time of AKS cluster creation and cannot be changed later.

KUBENET

* Default networking model in AKS.
* Each node in the AKS cluster gets an IP address from an overlay network.
* Pods within the cluster are assigned IP addresses from a separate range.
* Network address translation (NAT) is used to route traffic between nodes and pods.
* kubenet is simpler to set up and manage, as it doesn't require any additional configuration or integration with Azure networking resources.
* However, it may have slightly higher latency compared to Azure CNI, as traffic needs to go through NAT for inter-node communication.

AZURE CNI:

* Azure CNI integrates with Azure networking resources, such as virtual networks and subnets.
* Each pod in the AKS cluster gets an IP address directly from the subnet associated with the node.
* Pods can communicate with each other using the underlying Azure network fabric, without the need for NAT.
* Azure CNI provides better performance and lower latency compared to kubenet, as traffic between pods does not require NAT.
* It also allows more advanced networking configurations, such as using network security groups (NSGs) for pod-level firewall rules.

WHEN CHOOSING BETWEEN KUBENET AND AZURE CNI, CONSIDER THE FOLLOWING FACTORS

* PERFORMANCE REQUIREMENTS
  + If you have high-performance workloads or require low latency inter-pod communication, Azure CNI is recommended.
* NETWORKING CAPABILITIES
  + If you need advanced networking features like pod-level firewall rules using NSGs, Azure CNI provides more flexibility.
* SIMPLICITY
  + If you prefer a simpler networking setup without any additional configuration or integration, kubenet is a good choice.

#### OVERVIEW OF KUBERNETES LOAD BALANCER SERVICE

|  |  |
| --- | --- |
|  | * In Kubernetes, a LoadBalancer service is a type of service that allows external traffic to be load-balanced across multiple pods within a cluster. * It provides a stable IP address and routes traffic to backend pods, distributing the load evenly and ensuring high availability. * LoadBalancer services expose the applications externally and distributing traffic efficiently across pods. |

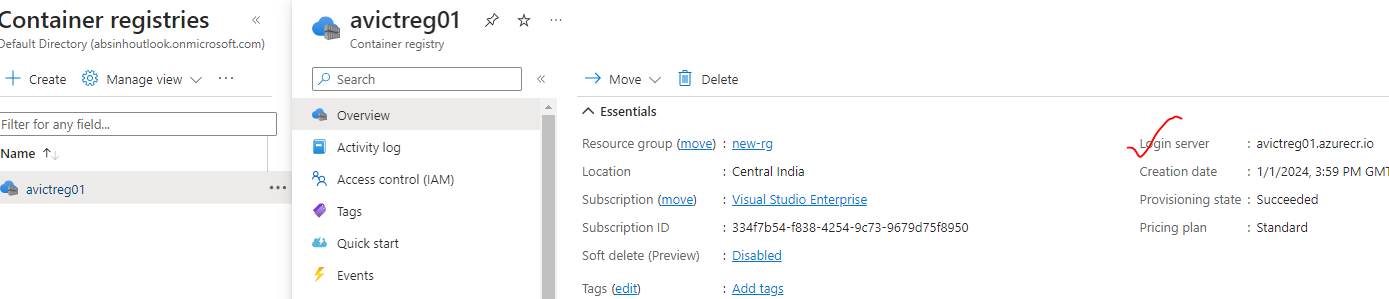
* LOADBALANCER TYPE:
  1. In Kubernetes, services can be of different types, such as **ClusterIP, NodePort, and LoadBalancer**.
  2. To expose a service externally and distribute traffic, we use the LoadBalancer type.
* EXTERNAL LOAD BALANCER:
  1. When we create a LoadBalancer service, Kubernetes automatically provisions an external load balancer (such as a cloud provider's load balancer) to route traffic to the service.
  2. This external load balancer handles the distribution of incoming traffic to the pods.
* SERVICE CONFIGURATION:

|  |  |
| --- | --- |
| * When defining a LoadBalancer service in Kubernetes, we specify the target port (the port on which the service is exposed) and the port(s) on which the pods are listening. * We can also configure additional options such as session affinity and health checks. | apiVersion: v1  kind: Service  metadata:  name: azure-vote-front  spec:  type: LoadBalancer  ports:  **- port: 80**  selector:  app: azure-vote-front |

* EXTERNAL IP:
  1. The LoadBalancer service is assigned an external IP address, which clients can use to access the service.
  2. This IP remains stable even if the underlying pods or nodes change.
  3. The external load balancer forwards traffic to this IP, which is then distributed to the pods.
* SCALING AND HIGH AVAILABILITY:
  1. LoadBalancer services in Kubernetes enable horizontal scaling and high availability.
  2. As we add or remove pods within the service, the load balancer automatically adjusts to distribute traffic evenly across the available pods.
* DNS-BASED SERVICE DISCOVERY
  1. Kubernetes provides DNS-based service discovery, allowing clients to access the LoadBalancer service using its DNS name. This enables easy communication between services within the cluster.

#### DEPLOYMENT IN AKS CLUSTER

* Step 1: Set an application and create a Docker Image
* Step 2: Push the docker image to an image registry like ACR or docker hub.
* Step 3: ACR Details:
  + ***Login Server: avictreg01.azurecr.io***
  + ***Name: avictreg01***



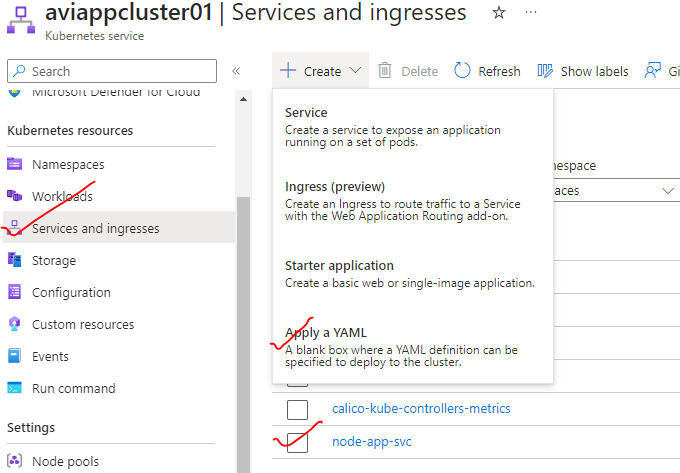
* Step 4: Create Kubernetes deployment and services(Load Balancer) configuration.

|  |  |
| --- | --- |
| **deployment.yaml** | **service.yaml** |
| **apiVersion : apps/v1**  **kind: Deployment**  **metadata:**  **name: node-app-dpl**  **spec:**  **replicas: 1**  **selector:**  **matchLabels:**  **app: web-app**  **template:**  **metadata:**  **labels:**  **app: web-app**  **spec:**  **containers:**  **- name: web-app**  **image: avictreg01.azurecr.io/nodejs-app:latest**  **ports:**  **- containerPort: 80** | **apiVersion: v1**  **kind: Service**  **metadata:**  **name: node-app-svc**  **spec:**  **selector:**  **app: web-app**  **ports:**  **- protocol: TCP**  **port: 8080**  **targetPort: 80**  **type: LoadBalancer** |

DEPLOY THE deploymemt



DEPLOY THE SERVICE



##### PERSISTENCE STORAGE IN AKS CLUSTER – DISK

In Azure Kubernetes Service (AKS), Azure Disks can be used as persistent storage for the applications running in AKS pods. STEPS TO USE AZURE DISK STORAGE IN AKS:

1. PROVISION AZURE DISKS:
   1. Before using Azure Disks in AKS, we need to provision the disks. You can create Azure Disks using the Azure portal, Azure CLI, or Azure Resource Manager templates. Make sure to select the appropriate disk type (Standard or Premium) and size based on your requirements.
2. Create a PersistentVolume (PV):
   1. In AKS, you can define a PersistentVolume (PV) object that represents the Azure Disk storage you want to use. The PV specifies the details of the Azure Disk, such as the disk name, disk size, and disk type. You can create the PV using a YAML or JSON manifest file.
3. Create a PersistentVolumeClaim (PVC):
   1. Next, you need to create a PersistentVolumeClaim (PVC) object to request the Azure Disk storage for your application. The PVC specifies the desired storage capacity and other properties. You can create the PVC using a YAML or JSON manifest file.
4. MOUNT THE PVC IN YOUR APPLICATION:
   1. In your application's deployment or pod configuration, you need to specify the PVC as a volume and mount it to a specific path in your container. This allows your application to read and write data to the Azure Disk.
5. DEPLOY YOUR APPLICATION:
   1. Once you have created the PV, PVC, and configured your application to use the PVC, you can deploy your application to AKS. AKS will automatically bind the PVC to the available PV that matches the PVC's storage requirements.
6. ACCESS AND MANAGE THE DATA:
   1. Your application can now read and write data to the mounted Azure Disk. You can also take advantage of features like disk snapshots or Azure Backup to ensure data protection and availability.

##### PERSISTENCE STORAGE IN AKS CLUSTER – FILE SHARE