***Containerization:***

*Containerization is the packaging of software code with the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable****—called a container****, containers runs consistently on any infrastructure. More portable and resource-efficient than*[*virtual machines (VMs)*](https://www.ibm.com/in-en/topics/virtual-machines)*, containers have become the de facto compute units of modern cloud-native applications.*

*Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable(able to run across any platform or cloud, free of issues)*

***Docker containers:***

* *Each container has its own process, storage but host O/S is common for all the containers, if anything goes wrong with App1 then their won’t be any impact on other applications (like App2, App3…etc) as there are independent with each other and that’s the beauty of containerization.*
* *Virtual Machines are heavy in weight and containers are light in weight, containers can be created anywhere on top of physical machines, virtual machines.*
* *As we can in above image, we can create a containers which has application/application code and its software’s, Environment variables, agents, Frameworks and dependencies…etc.*

***Containers:*** *Containers will carry our applications and its dependencies. Containers are light in weight, it does not contain the CPU and memory as much as a VM, the container does not have an operating system, containers are portable which can be moved from 1stenv to another environment(env)*

***Docker in Brief:***

**

***Docker container in Azure Cloud Computing:***

**Containers:**

A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly reliably from one computing environment to another.

Graphical user interface, application, Teams

Description automatically generated

* In case of VM’s there’s always a host, the host is created from the infrastructure on top of which there’s a hub hypervisor running and this hypervisor is responsible for hosting virtual machines and managing them, so there’s always a virtual machine on which we always have something called guest operating system, therefore we need to virtualize an entire operating for we to be able to host our application and we need to this again for every single application and when it comes to containers we also have a host and this host again has an infrastructure, but whats difference here is that we have a host operating system and top of which there’s something called run time and this run time is responsible for hosting and managing containers, but what is good here is that from here onwards we can just host containers, we no longer need that guest operating system for each application and each container.
* *There*f*ore* releasing a lot of unnecessary resources and as such allowing us to host more containers in more applications. So the main difference is here are first of all there’s less development overhead and developers don’t need to create container and they don’t need to create those images for virtual machines which takes a lot of time creating containers takes just minutes and if we know and if we have a priority prepare scripts in just couple of seconds and we have seen there is there’s no guest operating system’s, so there’s much less resources consumed and which also means since there’s not guest operating system loaded into the container and the file itself is in much smaller size, its from hundreds of megabytes down to 20 or 30 megabytes of size.
* Since the small size is there and the less consumption is required that means they also are starting up faster and we can get our containers running in just seconds and as we are not running the O/S this against means there’s much less updates that we need to do, therefore there’s a reduced and simplified updating process that we need to implement for managing our applications

**Advantage of containers as compare to Virtual Machines:**

* Less development overhead.
* Less resources(RAM, VCPU’S, Percentage CPU’s, processing, Disk consumption…etc) consumed.
* Smaller size
* Faster start-up
* Reduced and simplified updates

**Azure Container Instances(ACI):**

Azure container instances offer the fastest and simplest way to run a container in Azure, without having to provision any virtual machines and without having to adopt a higher-level service.

**Container Group:** It is just a grouping of multiple containers where we can host multiple applications in it, in containers we host each application on port No’s (like port 80, 5000…etc)to allow public connectivity and which allows us to get a public ip and a public DNS name for our container and hence a public connectivity can be established and in this way we can host our web applications using containers(docker containers) in Azure cloud computing platform.

**Key Benefits of using Azure Container Instances(ACI):**

* It provides fast startup times – in just few seconds ACI can start up containers
* It provides public IP & DNS names
* Custom Sizes
* We can host both Linux and Windows containers
* We can get persistent(continuing to exist ) storage
* It even allows us Virtual Network deployment

**Demo’s on Azure Docker Container Instances (ACI):**

1)Simple web application with Docker container image on ACI on top of Linux O/s

2)Custom image using node js application and will deploy this image to ACI after this will deploy Docker container registry then container instance for hosting application on Docker container.

**Demo 1: Using Custom image docker file to host on top of Docker container instance.**

1. Login to Azure portal>>search for container instances>>create>>and fill the below details:

**Subscription:** Free trial (or existing whatever)

**Resource Group:** any

**Container name:** starbucks-cafe

**Region:** give any region

**Availability zone:** none

**SKU:** Standard

**Image Source:** Other registry

**Image Type:** Public

**Image:** ashfaque9x/my-html-website [ Here ashfaque9x>>Username & my-html-website>>Image name]

**Size: keep as default**

**Click next on Networking tab**

**DNS name label:** cromaservices>>and leave all rest values as is and finally click on Review+Create and provision the first ACI

Now copy and paste the IP address or FQDN of the ACI in another browser then will see Starbucks Cafe Dynamic website is hosted on top of our Docker container Instance.

**Demo 2: Practical Implementation of ACI:**

* Login to Azure portal>>search for container instances>>create>>and fill the below details:

**Subscription:** Free trial (or existing whatever)

**Resource Group:** any

**Container name:** give any name >>this is our container group name actually

**Region:** give any region

**Image Source:** QuickStart images

**Image:** mcr.microsoft.com/azuredocs/aci-helloworld:latest >>so, here we are pasting the URL which is a container containing node.js application with helloworld message in Microsoft content registry with microsoft.com

**Size:** keep as default

**Click next on Networking tab**

**DNS name label:** intelliservices>>and leave all rest values as is and finally click on Review+Create and provision the first ACI

* Now copy and paste the IP address or FQDN of the ACI in another browser then will see **Welcome to Azure Container Instances!** >>Now this indicates our simple web application running on docker container image via ACI.

**Demo 3: Custom image using node js application and will deploy this docker image to ACI**

click on cloud shell>>show advanced settings>>and fill below

**Subscription:** existing subscription

**Cloud Shell Region:** give any.

**Resource Group:** give any.

**Storage account:** give any name.

**File share:** give any name.

And finally click on Create Storage, after some 2-3 minutes of time the Bash with all its background gets ready and now pass the below command and hit enter

**git clone https://github.com/Azure-Samples/aci-helloworld.git**>>this we are cloning the git repository of the container that we deployed in above demo

type **ls** and hit enter>>and here we see we have a folder **aci-helloworld**

type **cd** **aci-helloworld/** and hit enter>>to go inside the aci-helloworld folder

type **code .** and hit enter>>in cloud shell we pass this command to open an editor that looks very similar to visual studio code and now here an editor will get open with .git and app folder with multiple files in it and also some other files

# click on index.html file inside the app folder and remove the complete coding of tag <svg> to </svg> and type in tag <h1> **Welcome to Azure Container Instances Class for Naresh IT Students!**

</h1>

click on top right three dots and click on save to save the changes and close this editor by clicking on close(below)

**Container Registry:**

It is an inbuild service for hosting containers for more than just other container instances, we can also use it with Kubernetes services, fabric services…etc and here this container registry will allow us to build a docker container and host it

Now come to Azure portal and **search for container registries**>>create>>and fill the details

**Subscription:** take any

**Resource Group:** take any

**Registry name:** nareshcr

**Location:** EastUS

**SKU:** Basic

And leave all rest values to default and finally click on Review+Create to provision the container registry and this container registry will allow us to build a docker container and host it.

Now come back to cloud shell and type the below command

**az acr build --resource-group RGName --image demo/custom-image-demo:v1 --registry containerRegistryName --file Dockerfile .**

And with above command we can send the entire docker code, build it, and store it on Azure container registry.

Now come back to Container Registry in Azure portal and click on Access control IAM(left side)>>+Add>>Add role assignment>>privileged administrator roles>>owner>>Next>>+Select members>>choose the user with you have logged in with>>Select>>Next>>click on +Select roles and principals>>click on configure in constrain roles box>>+Add role>>select owner and contributor check box>>select>>save>>save>>Next>>Review+Assign>>Refresh

Now click on repositories(in ACR left side)>>Refresh>> then will see our container image as below

Demo/custom-image-demo

If we click on the container image above, then will find v1 as Version 1 that we passed above.

**Note:** In container registry on left side click on access keys and **Enable** the Admin User, if we don’t make it as enable then we cannot deploy the private container instance by considering this container registry**(very important)**

**Now execute the below commands in a sequence of order in same cloud shell window in Bash**

**az acr repository show-tags --name containerRegistryName --repository repositoryName**

**az acr login -n containerRegistryName --expose-token**

**az acr repository list --name containerRegistryName --output table**

**az acr repository show --name containerRegistryName --repository repositoryName --output table**

# Hence, we have created a docker image in private azure container registry.

**Docker** is one of the most popular tools among all other container tools in the industry while working with Azure Container repository we deal with Docker CLI

**Azure Container Registry(ACR):**

ACR is a service provided by Microsoft Azure that is managed by private Docker registry service based on the open-source Docker registry, using the Azure Container Registry we can store and manage private Docker container images.

The main purpose of Azure container registry is to pull images to various deployment targets and these deployment targets are setup on the receiving end and the deployment targets can be as follows.

**(i)Scalable Orchestration systems:** these orchestration systems manage the containerization of application over a cluster of hosts, which are the services like Kubernetes, DC/OS, Docker swarm...etc.

**(ii)Azure Services:** These are the services deployed by Microsoft Azure such as Azure Kubernetes services, app services...etc which supports building and scaling up running applications.

To install docker in Ubuntu systems/servers we use the below commands:

**sudo apt-get update**>>command to update.

**sudo apt-get install docker.io -y**>>command to install docker in the system.

Docker images

Docker pull <image-name>

Docker login <login server>

Docker tag>>basically this for repository version inside the docker container registry

Docker push <Image name>

Docker rmi <Image name>>to remove the docker image.

Docker run <login server>

**Implementation of Private Docker Container Registry in Ubuntu Linux Servers in Azure cloud computing:**

**Step1:**

Create a new VM with below details.

**Virtual machine name:** VM1

**Region:** West US3

**Availability options:** No infrastructure redundancy required

**Security type:** Standard

**Image:** Ubuntu Server 20.04 LTS - x64 Gen2(free services eligible)

**Size:** 2 VCPu's & 8GB memory

**Authentication type:** Password

**Username:** VM1

**Password:** Shaikpet@123

**Confirm Password:** Shaikpet@123

**Select inbound ports:** HTTP, HTTPS, SSH

Leave rest of the values to default and deploy the Virtual Machine

**Step2:**

Download & install Azure CLI in our local laptop(procedure and details mentioned in the document below)



**Step3:**

Open the command prompt(right click as administrator) in our local laptop to login to the Ubuntu server with below commands.

ssh VMUsername@VM\_ip\_address>>hit enter

yes>>hit enter

give the password of our VM>>hit enter.

and then finally will make a login/connection to our Ubuntu server

**Step4:**

Create an Azure container registry in Azure portal.

**Step5:**

Goto the below link

<https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-linux?pivots=apt>

scroll down little copy the commands from 1.Get packages needed for the installation process:(from above link) and paste it in our local laptop command prompt>>Hit Enter>>Y>>Hit Enter

**Step6:** copy the commands from 2.Download and install the Microsoft signing key:(from above link) and paste it in our local laptop command prompt>>Hit Enter

**Step7:** copy the commands from 3.Add the Azure CLI software repository:(from above link) and paste it in our local laptop command prompt>>Hit Enter

**Step8:** copy the commands from 4.Update repository information and install the azure-cli package:(from above link) and paste it in our local laptop command prompt>>Hit Enter>>Hit Enter

**Step9:** sudo apt-get install docker.io -y>>type this command in command prompt and hit enter(with this command we are installting docker in our cloud ubuntu server)

**Step10:** az login>>with this command will get the ID and CODE to authenticate(as shown in below example)

Ex: https://microsoft.com/devicelogin >>ID

EV8WMG5BU>>CODE/PASSWORD(this code will different every time for every user)

**Step11:** Now open the below ID/link(https://microsoft.com/devicelogin) in a new browser & paste the code/password>>Next>>click on the correct subscription in which you are doing this demo(if you are having multiple subscription)>>Continue>> and finally will get a message as shown in image below

We also see all our subscription details in command prompt(Azure CLI)

**Step12:** sudo az acr login --name containerRegistryName>>type this command in Azure CLI and hit Enter.

Ex: sudo az acr login --name nareshcr1471

Now here it will ask for UserName & Password to get this details goto container registry in Azure portal>>Access keys>>Check Admin user text box>>and here we can see the username & Password pass the creds like userName>>Hit Enter>>pass the password>>Hit Enter>>Finally will get a message like Login succeeded

**Step13:** sudo docker pull hello-world>>copy and paste this command in Azure cli

**Step 14:** sudo docker tag hello-world LoginServer/hello-world:v1

**Note:** Loginserver: this will get from Access keys in Azure Container Registry

**Step 15:** sudo docker push LoginServer/hello-world:v1

**Step 16:** sudo docker run LoginServer/hello-world:v1

**Step 17:** come to container registry in azure portal>>Repositories(left side)>>and here will see the repository created inside our container registry with version v1.

Hence, we have implemented a private docker image and hosted it in azure container registry via Ubuntu Linux server.

**Commands to clear the screens:**

Ctrl+L>>command to clear the screen in command prompt.

**Cls>>**Command to clear the command prompt screen in our local laptop.