Final Usecase Document

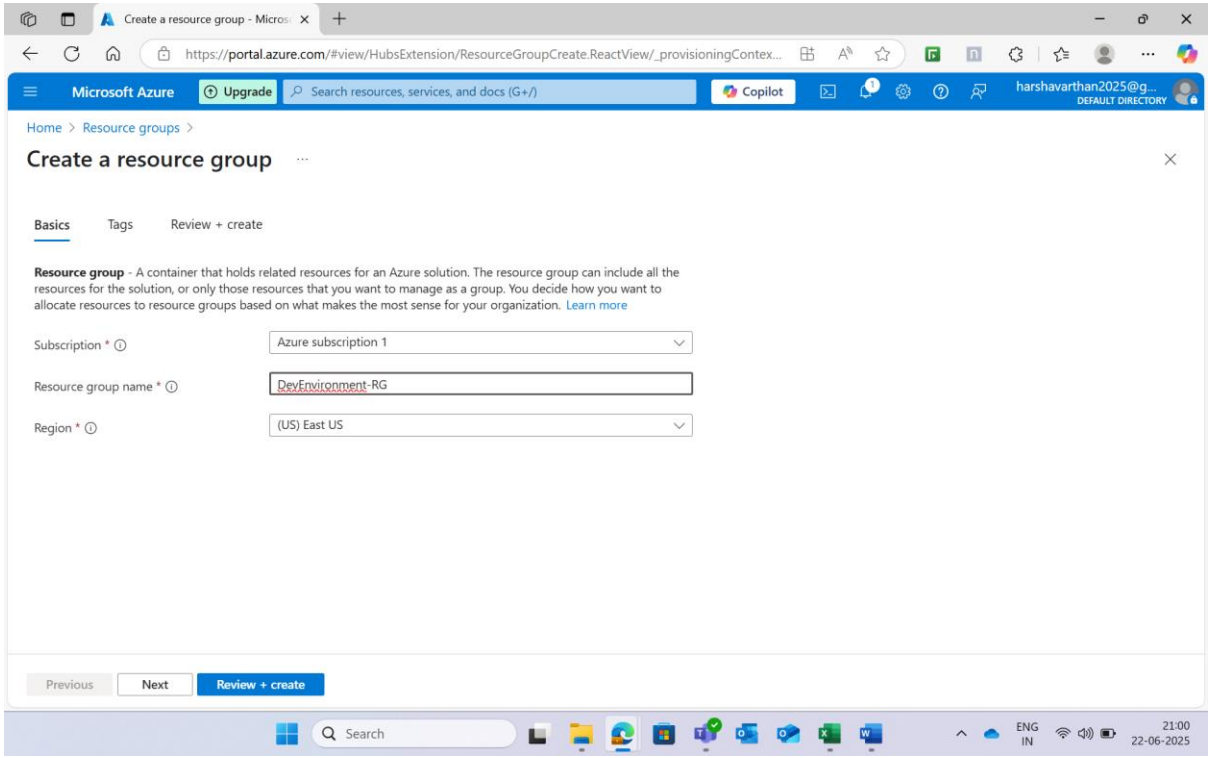
Objective: To familiarize the associate with cloud infrastructure, containerization, and automation. By deploying a "Hello World" web app using HTML and CSS on Azure Free Tier, participants will learn to set up resources like VMs, AKS, and ACR, containerize applications using Docker, and automate deployment with Azure DevOps pipelines. The goal is to provide hands-on experience in DevOps practices, CI/CD workflows, and resource optimization within free-tier limits.

Phase 1 : Infrastructure Setup

Step1 :

Created a resource group named “DevEnvironment-RG”.

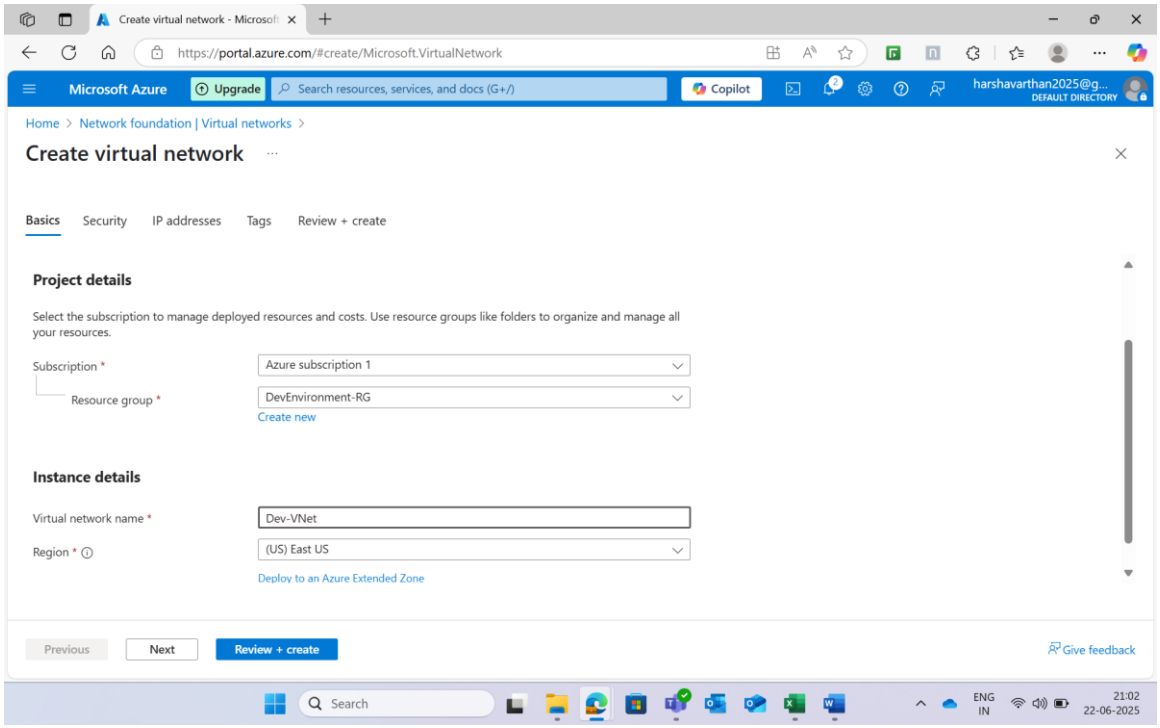
- **Name:** DevEnvironment-RG
- **Region:** East US
- **Purpose:** Group all resources related to this project.

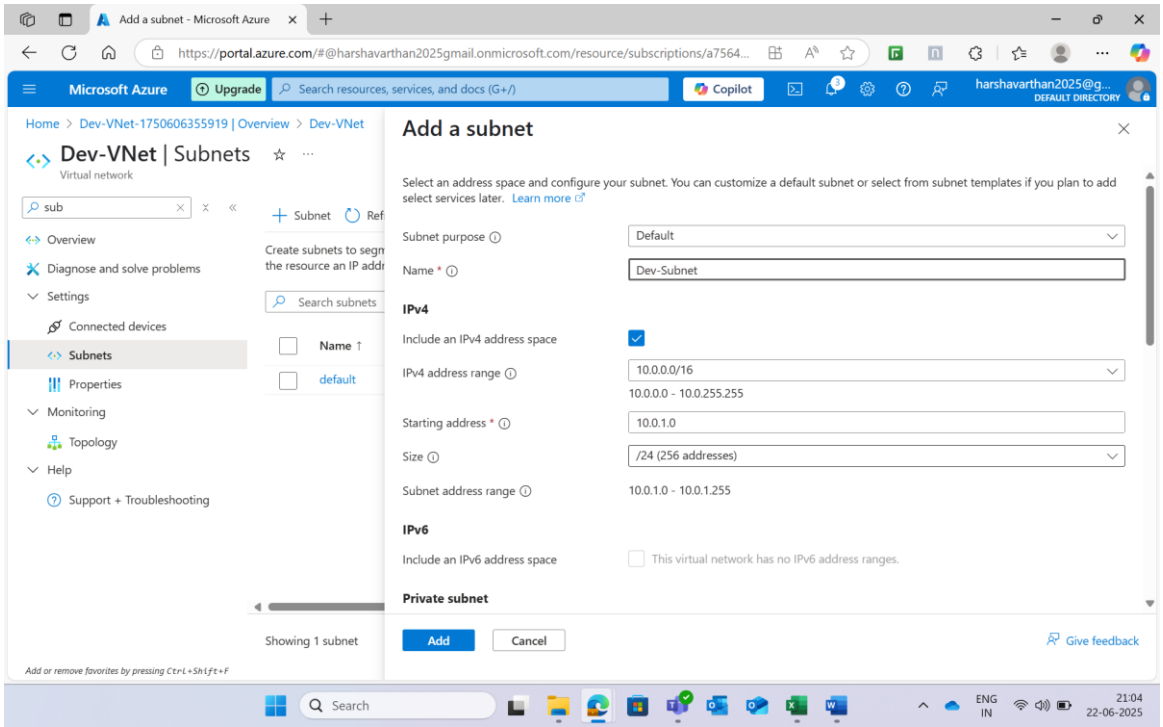


Step2:

Created a virtual network with the address space 10.0.0.0/16 and created a subnet with the address space 10.0.1.0/24.

- **Name:** Dev-VNet
- **Address Space:** 10.0.0.0/16
- **Purpose:** Enable communication between resources in the Azure environment.

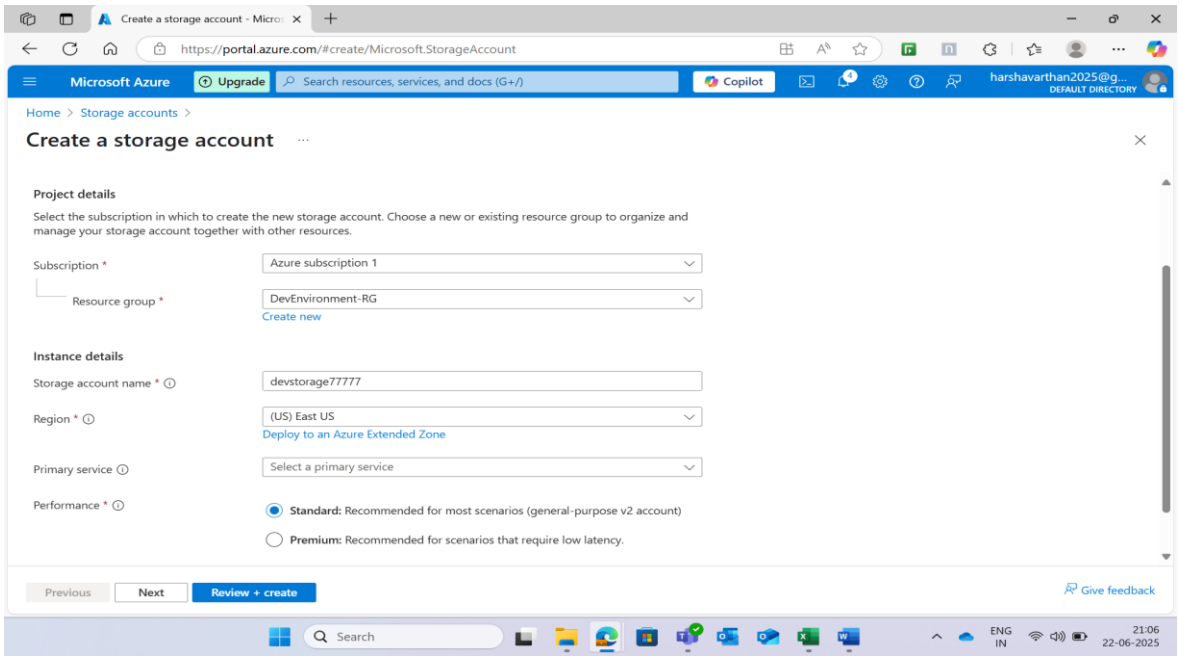


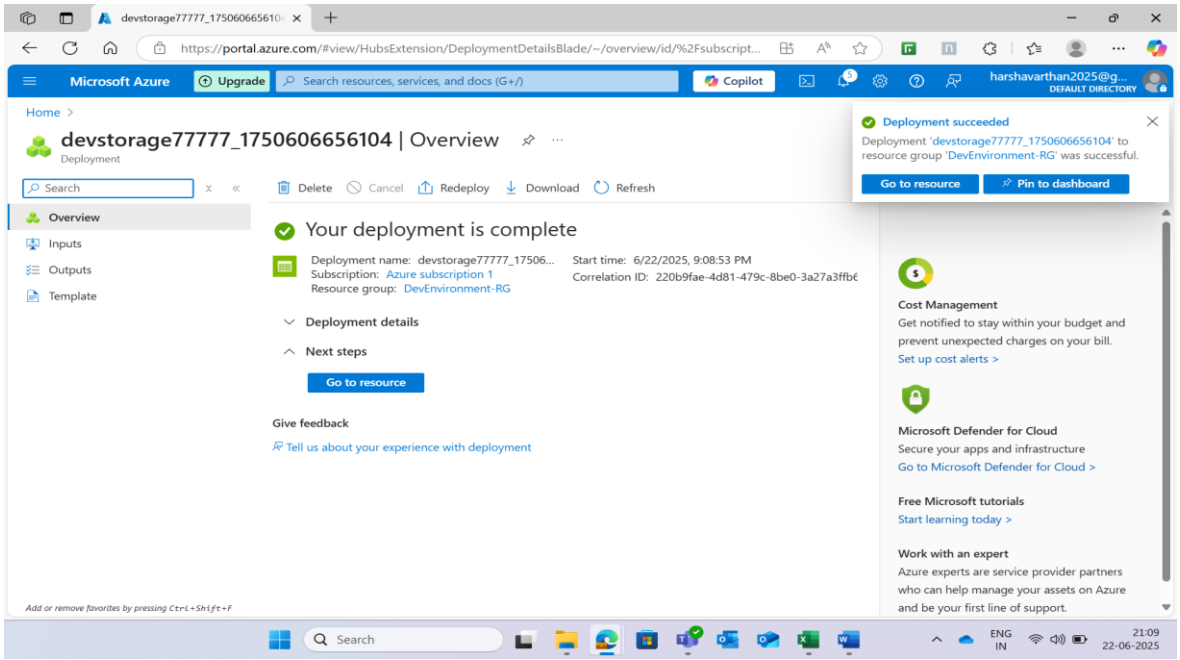


Step 3:

Created a Storage account named devstorage7777 in the same resource group.

- **Name:** devstorage7777
- **Location:** East US
- **SKU:** Standard_LRS
- **Purpose:** Store logs and application-related data.

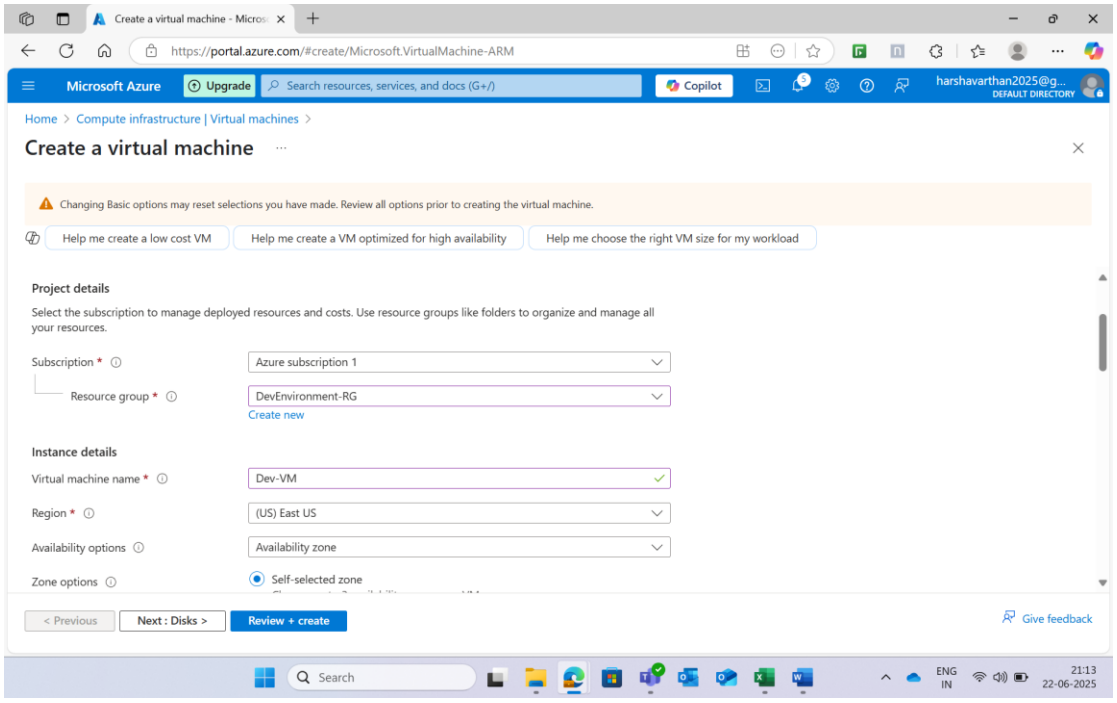


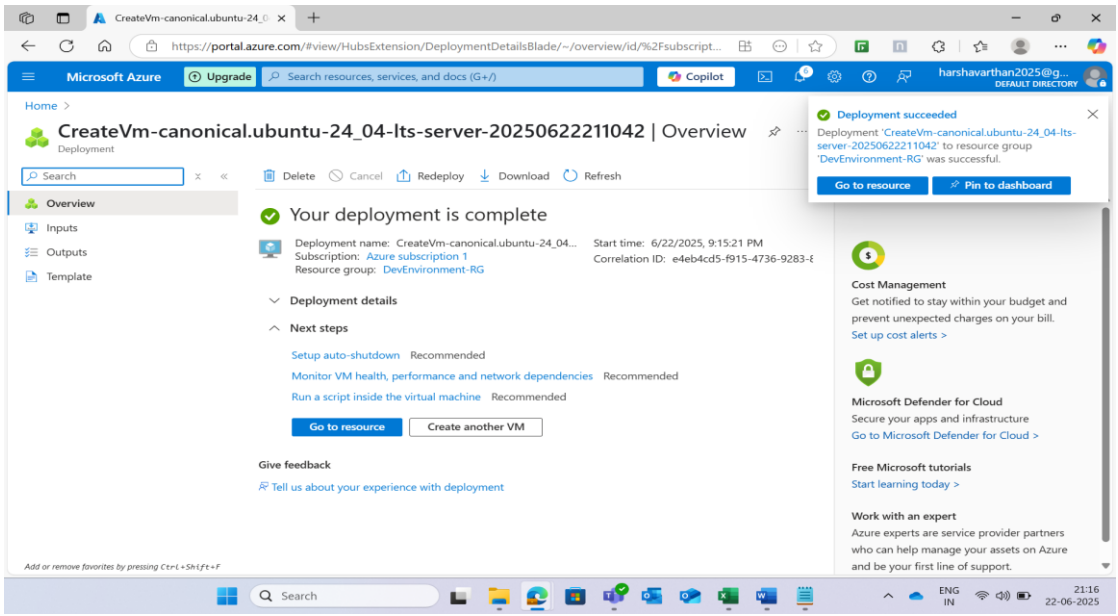


Step 4:

Created a Virtual machine of ubuntu image with the size Standard_B1s.

- **Name:** Dev-VM
- **Operating System:** Ubuntu 20.04 LTS
- **Size:** Standard_B1s (1 vCPU, 1 GB RAM, Free Tier eligible)
- **Admin Username:** azure-dev-user
- **Tags:** Project=DevEnvironment, Owner=azure-dev-user

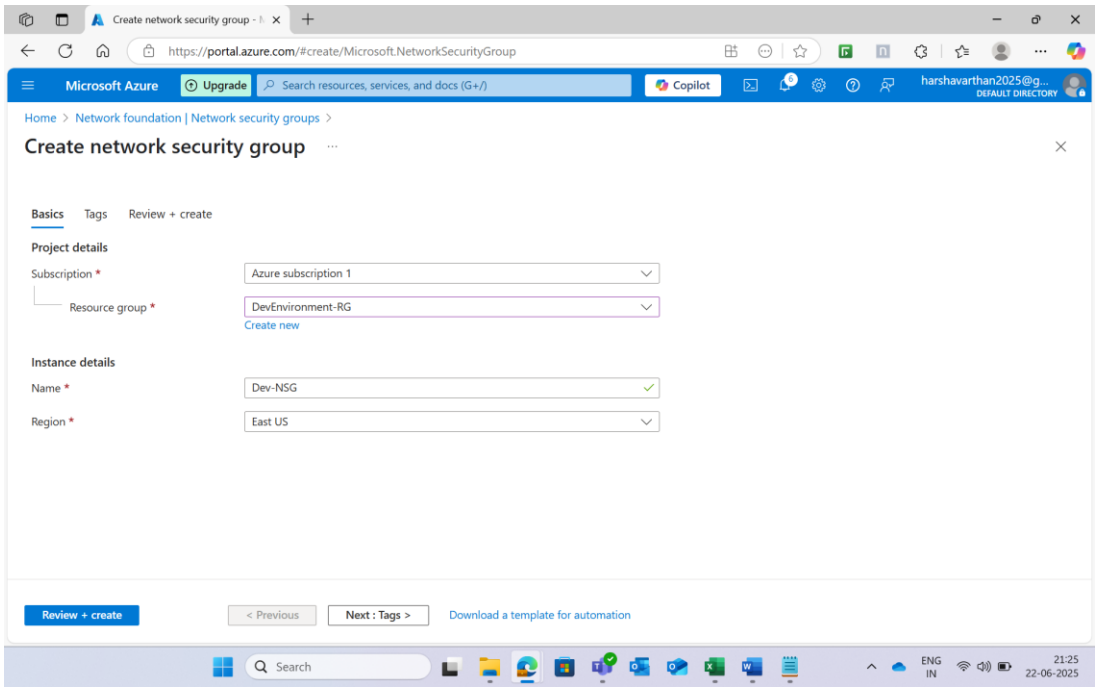




Step 5:

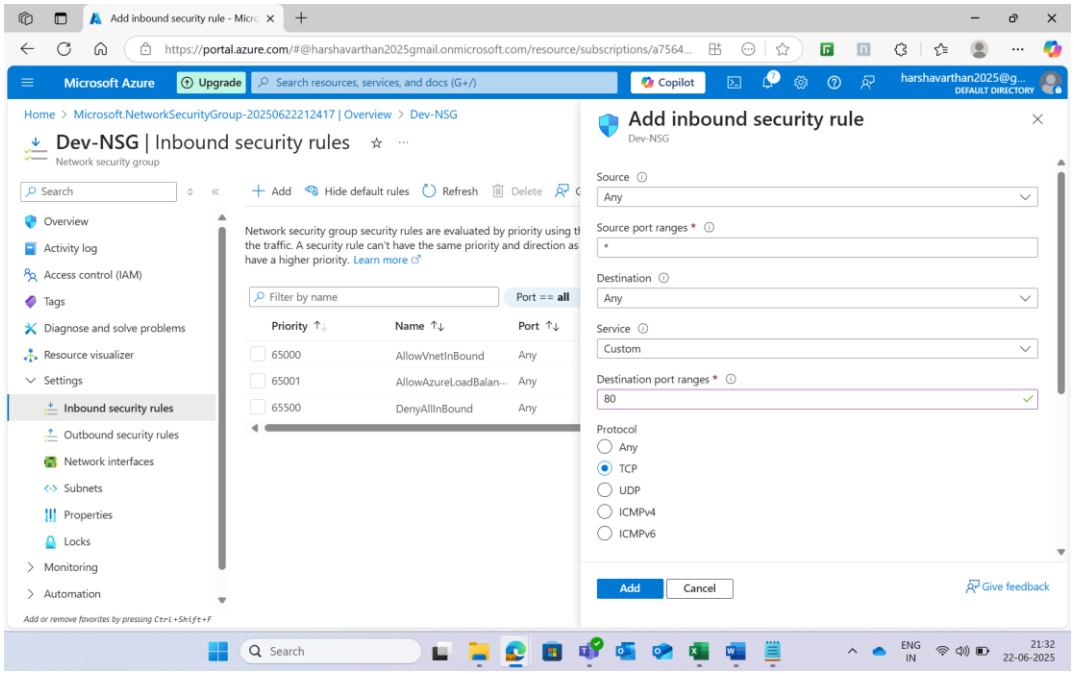
Create a network security group in the virtual machine .

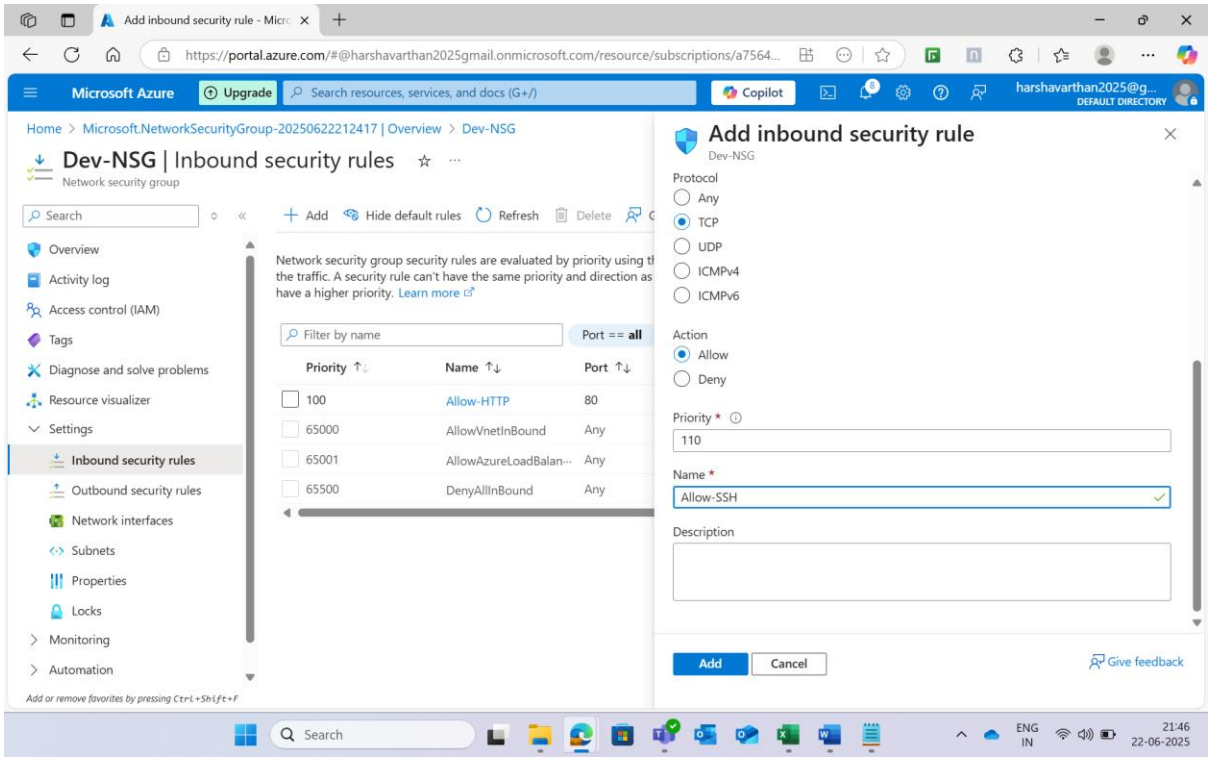
- **Name:** Dev-NSG
- **Rules:** Allow HTTP traffic on port 80 and SSH traffic on port 22.



Step 6:

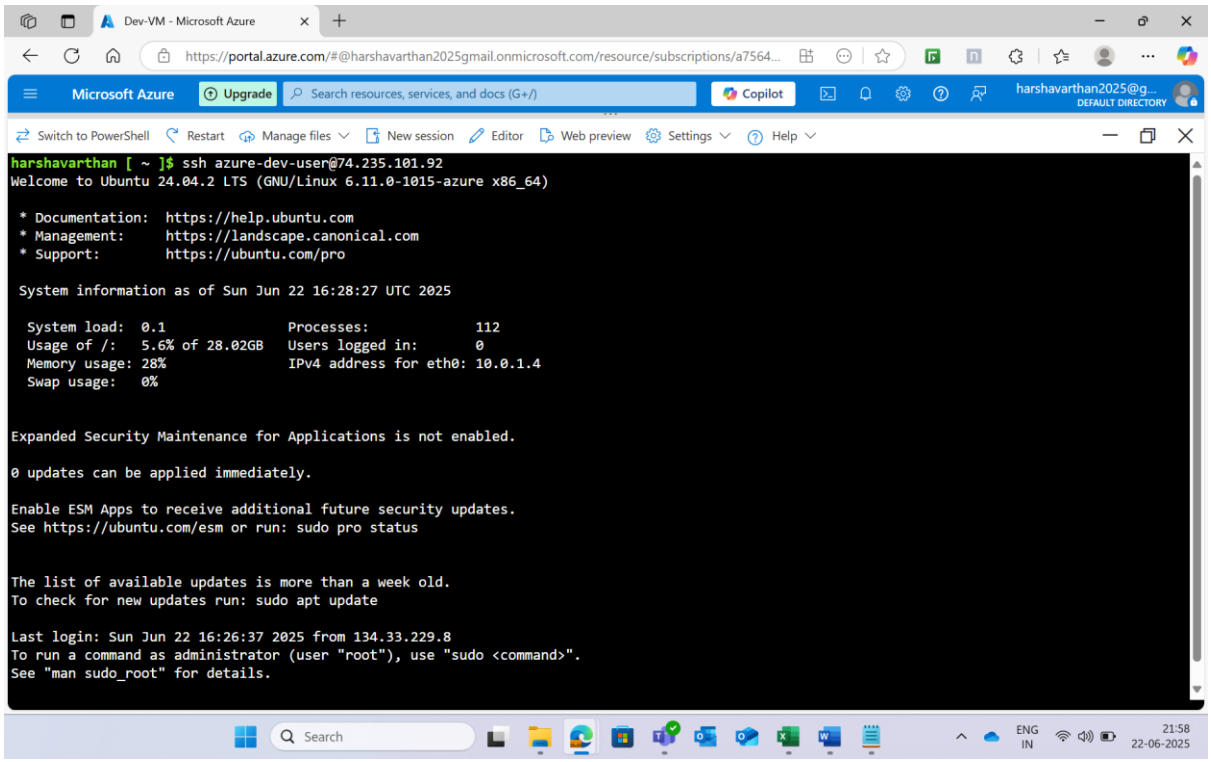
Add the inbound rules in the Nsg with the ports 80 and 22.





Step 7:

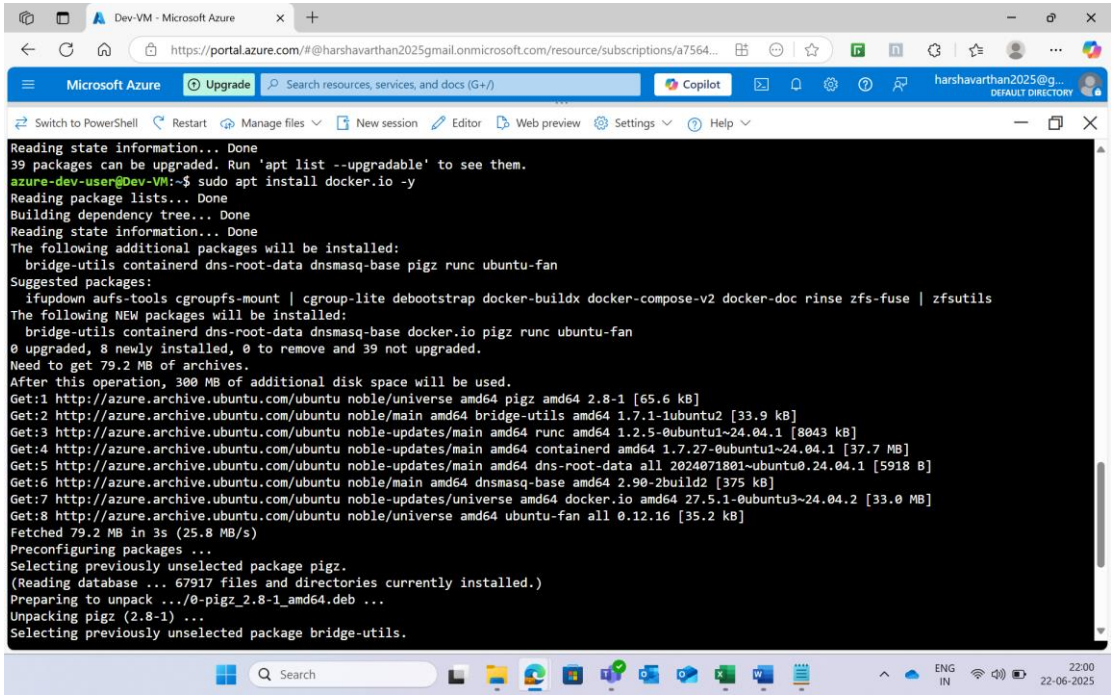
Connected the virtual machine.

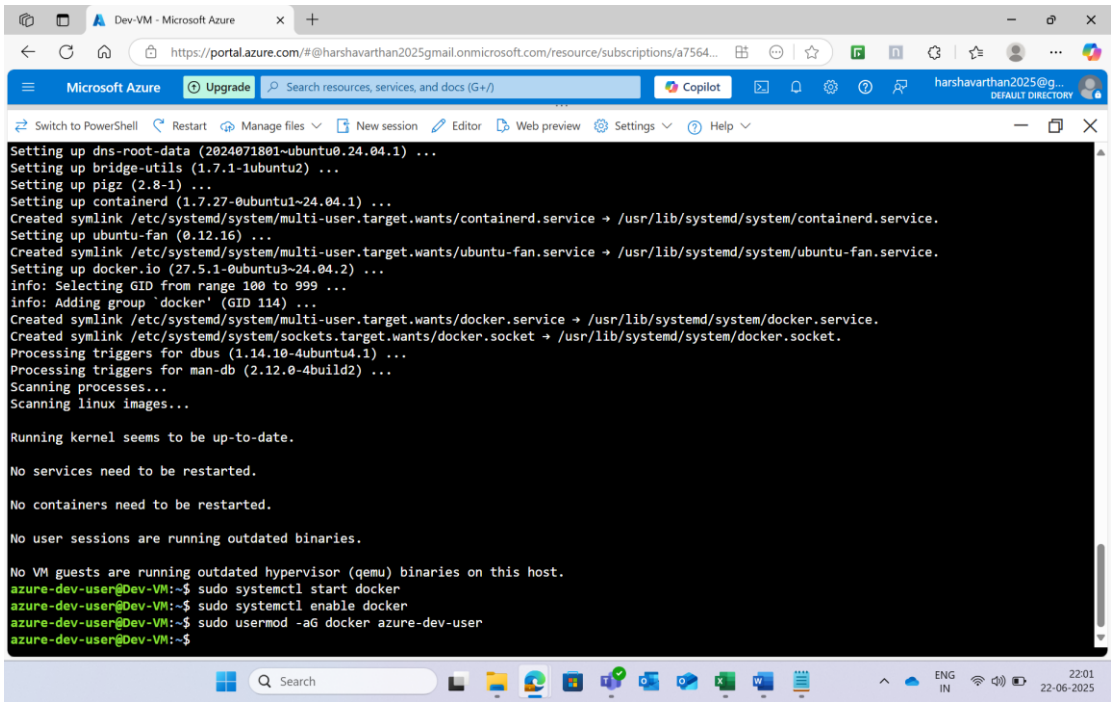


Step 8:

Install the docker in the VM.

- sudo systemctl start docker
- sudo systemctl enable docker
- sudo usermod -aG docker azure-dev-user

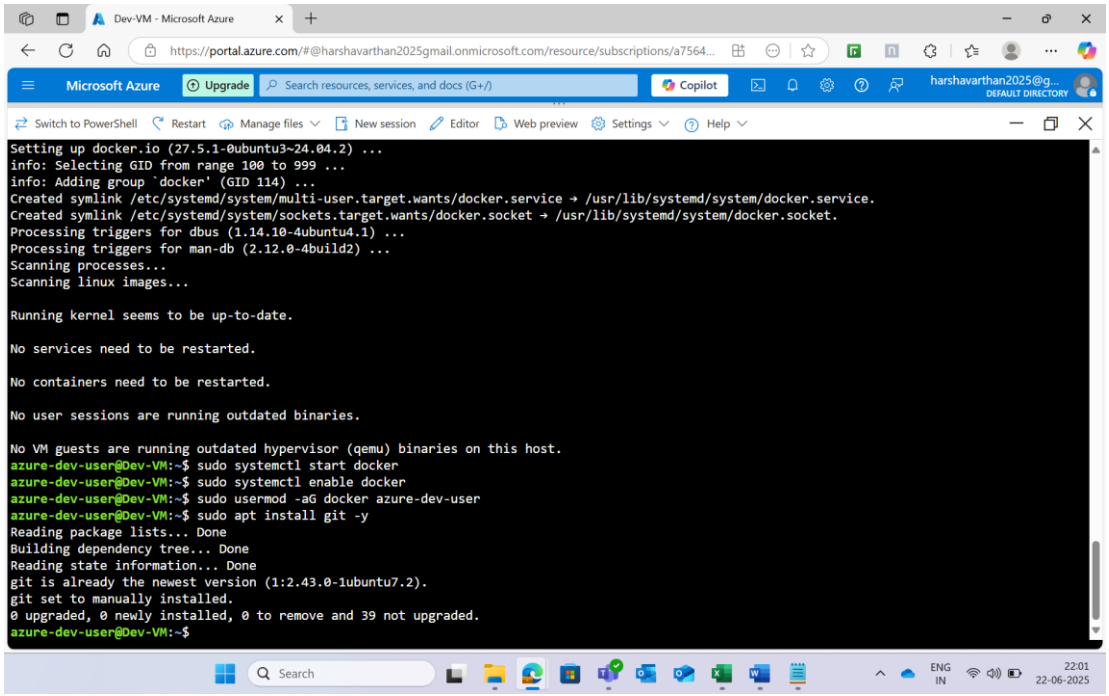




Step 9:

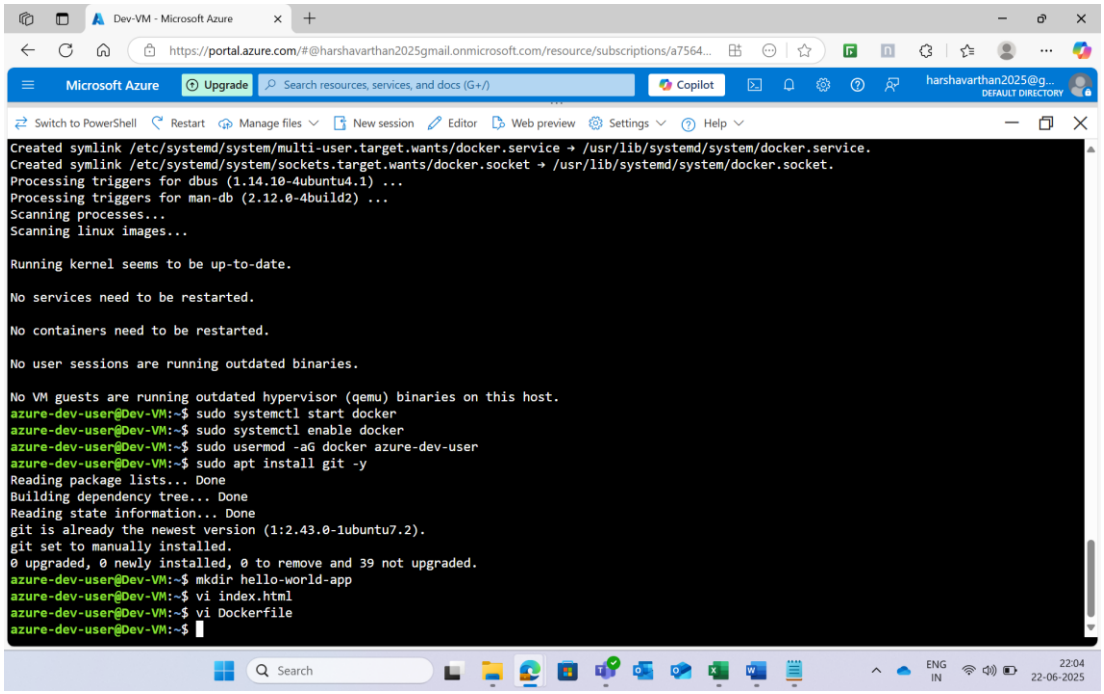
Install the git in the virtual machine.

- sudo apt install -y git

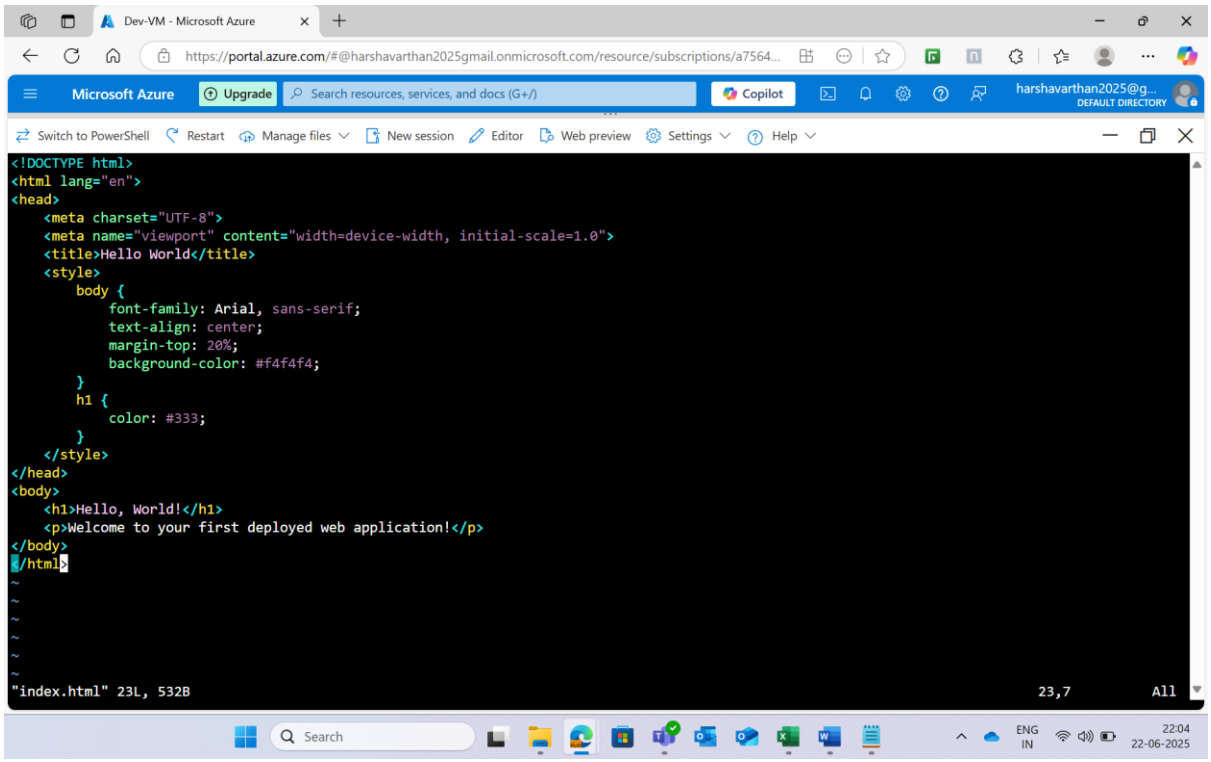


Step 10:

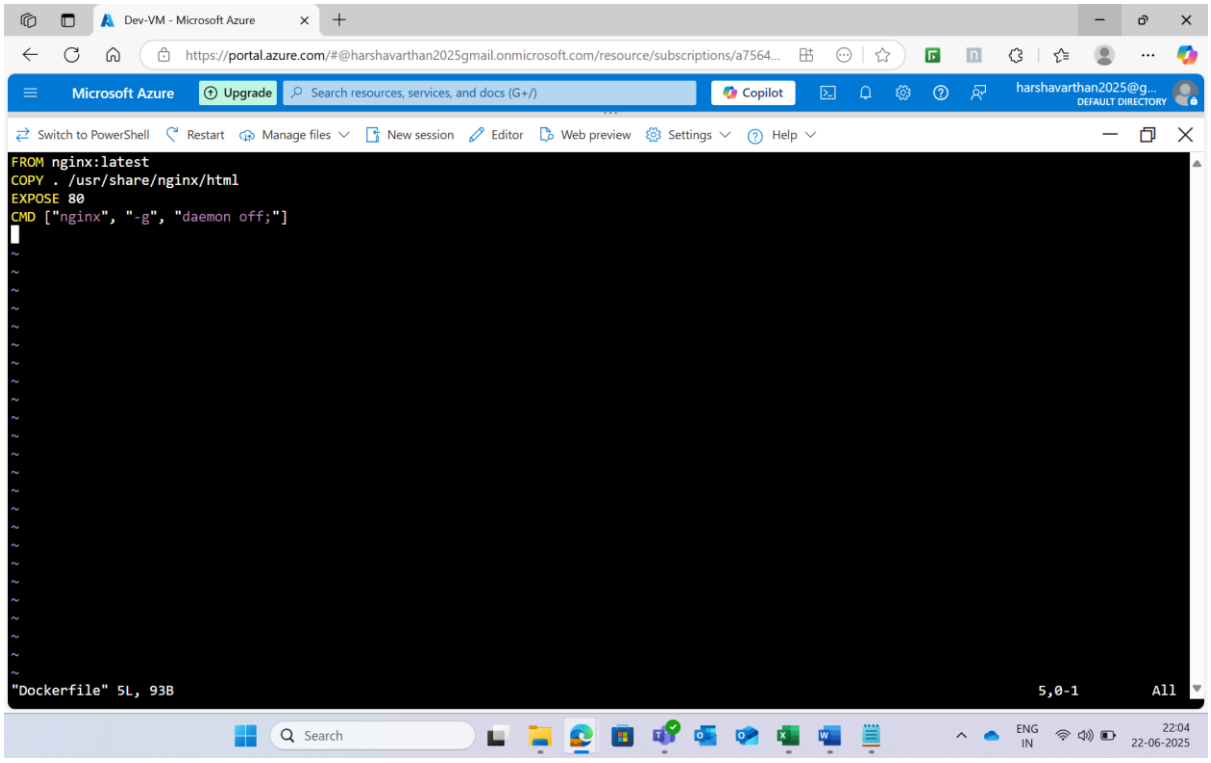
Create a Directory named “Hello-world-app” and make a file named “index.html” and “Dockerfile”.



- Insert the html file in this index.html.



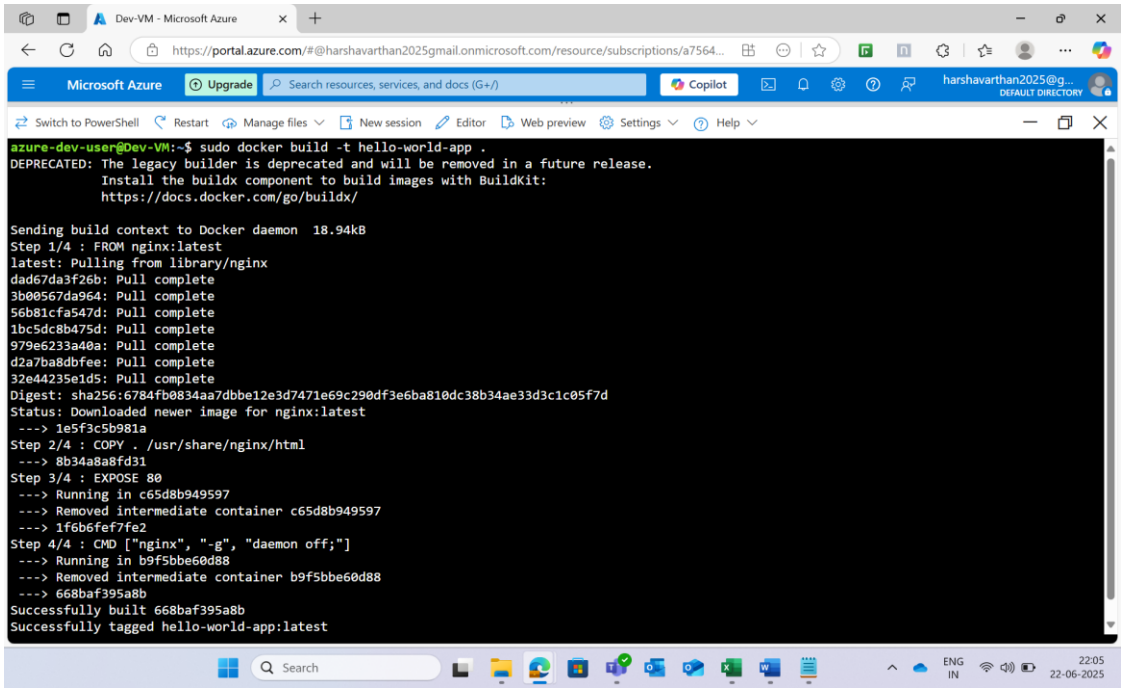
- Insert the dockerfile.



Step 11:

Build the image

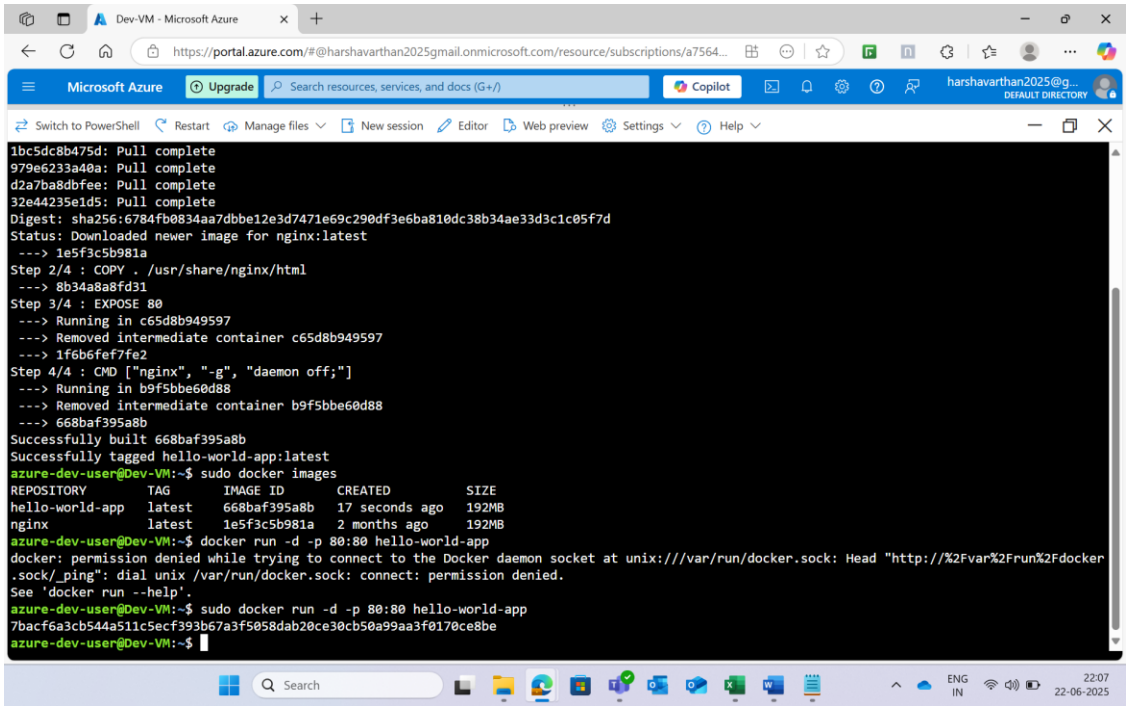
- docker build -t hello-world-app .

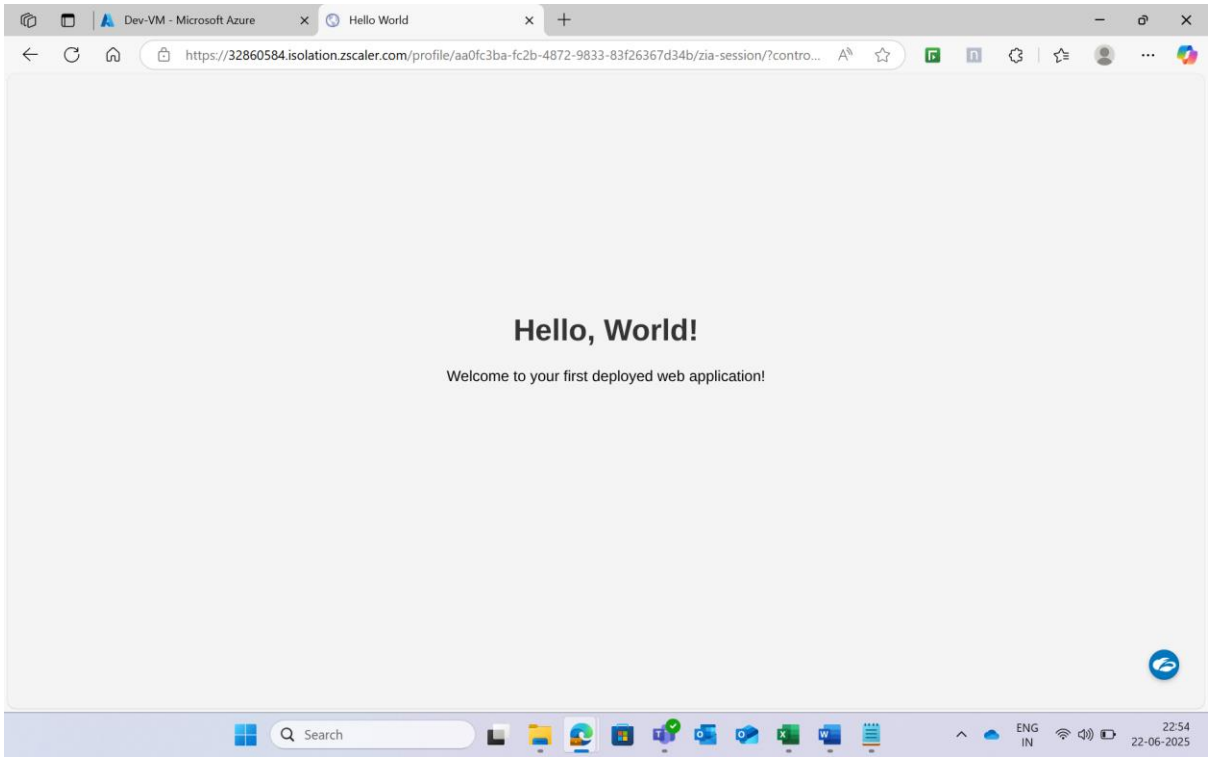


Step 12:

Run the image.

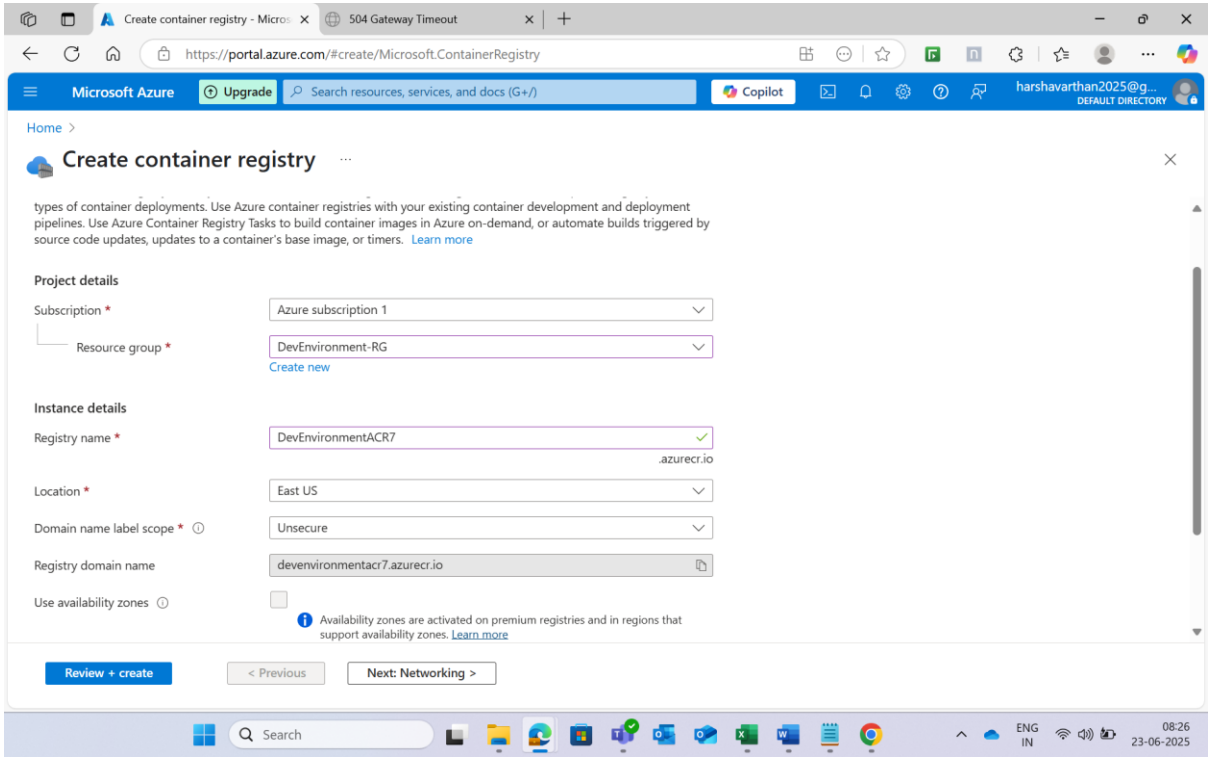
- docker run -p 80:80 hello-world-app

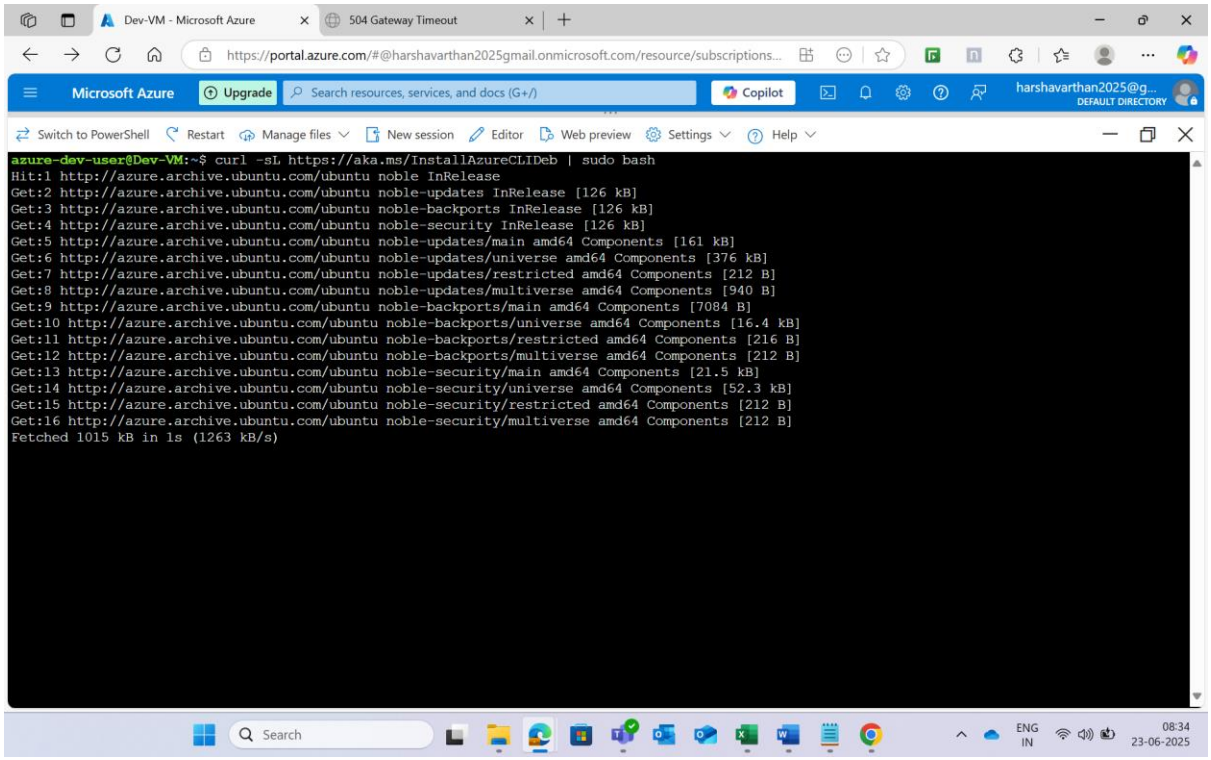




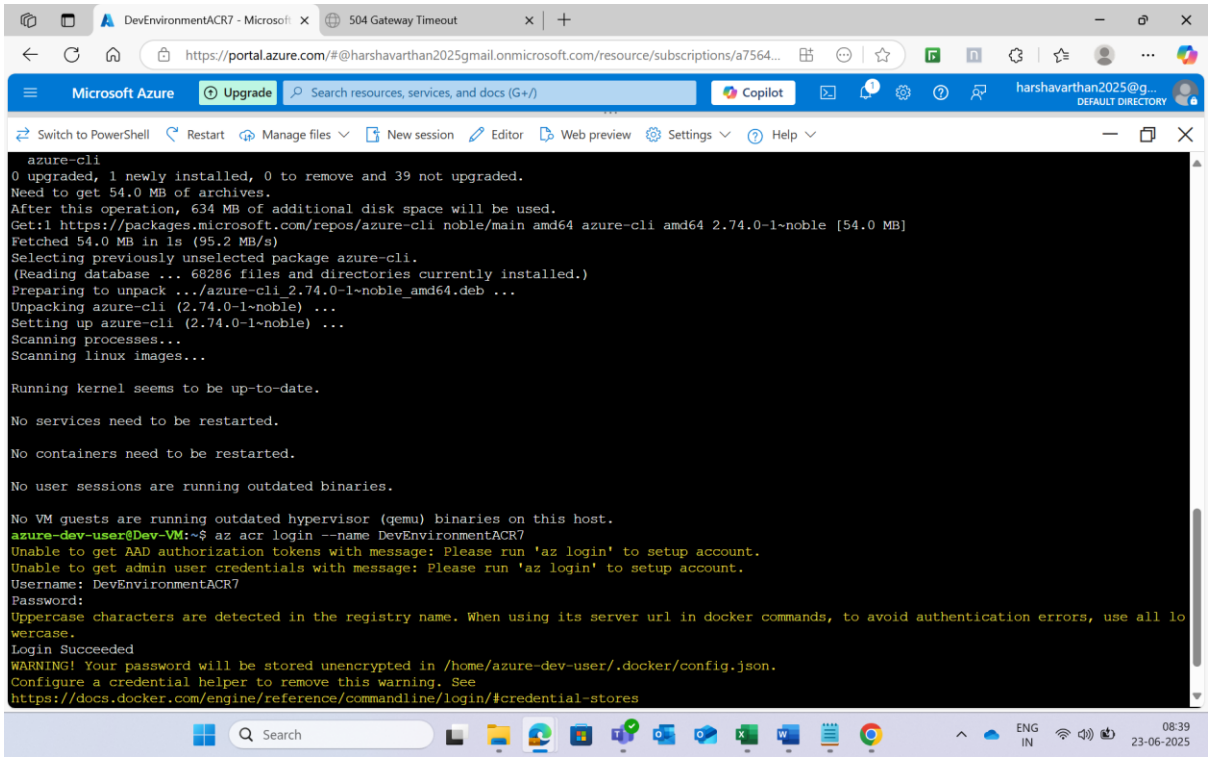
Step 13:

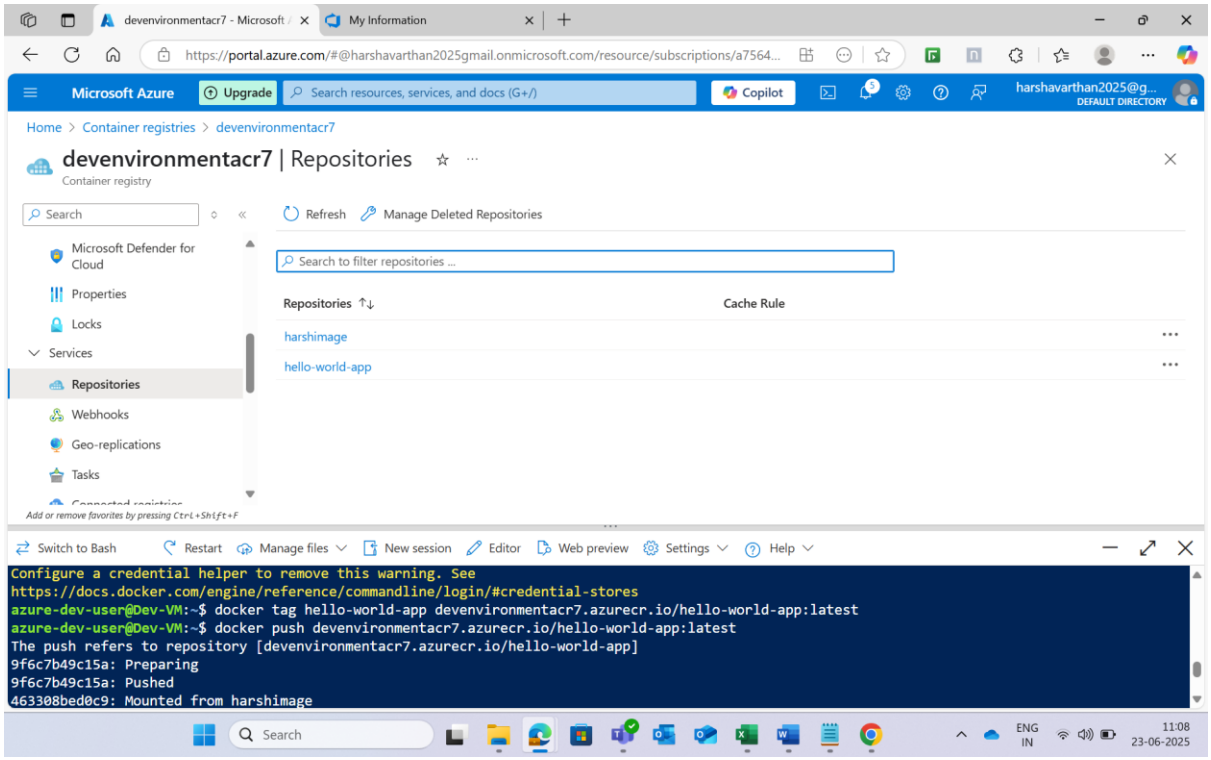
Crete a Azure Container registry to push the image.





- az acr login --name DevEnvironmentACR
- docker tag hello-world-app DevEnvironmentACR.azurecr.io/hello-world-app:latest
- docker push DevEnvironmentACR.azurecr.io/hello-world-app:latest

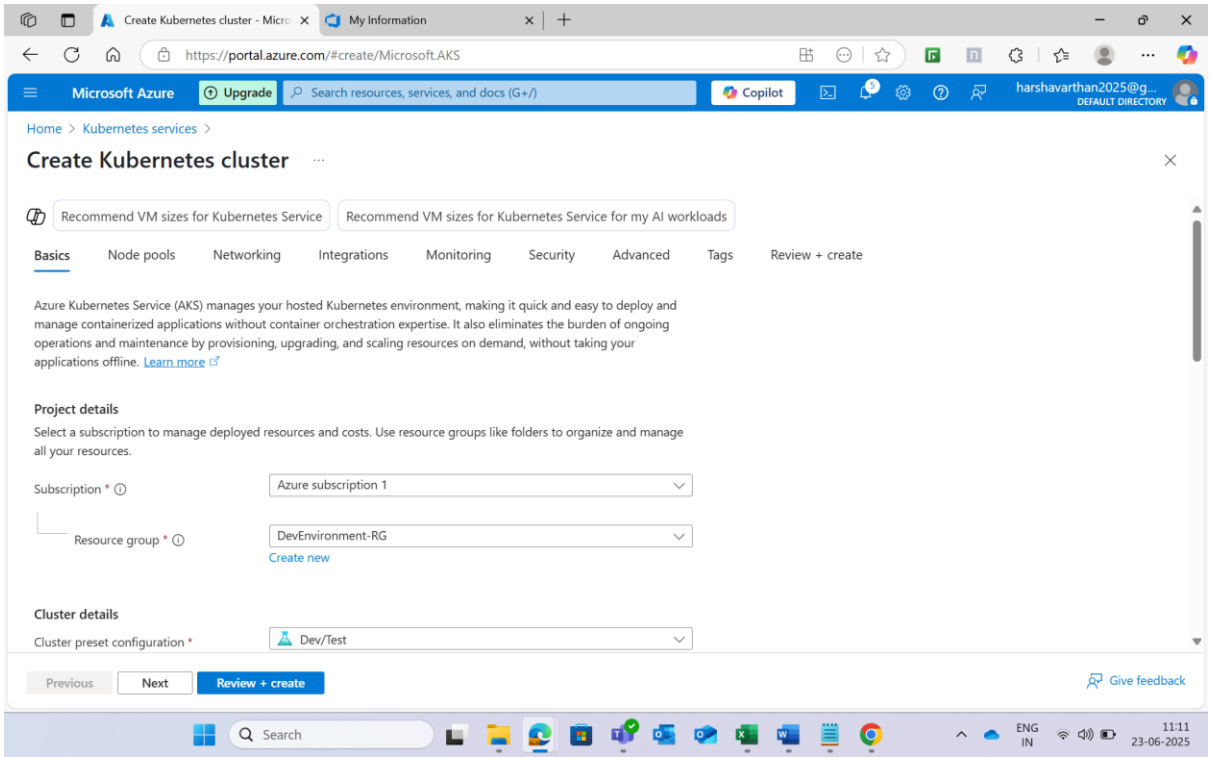


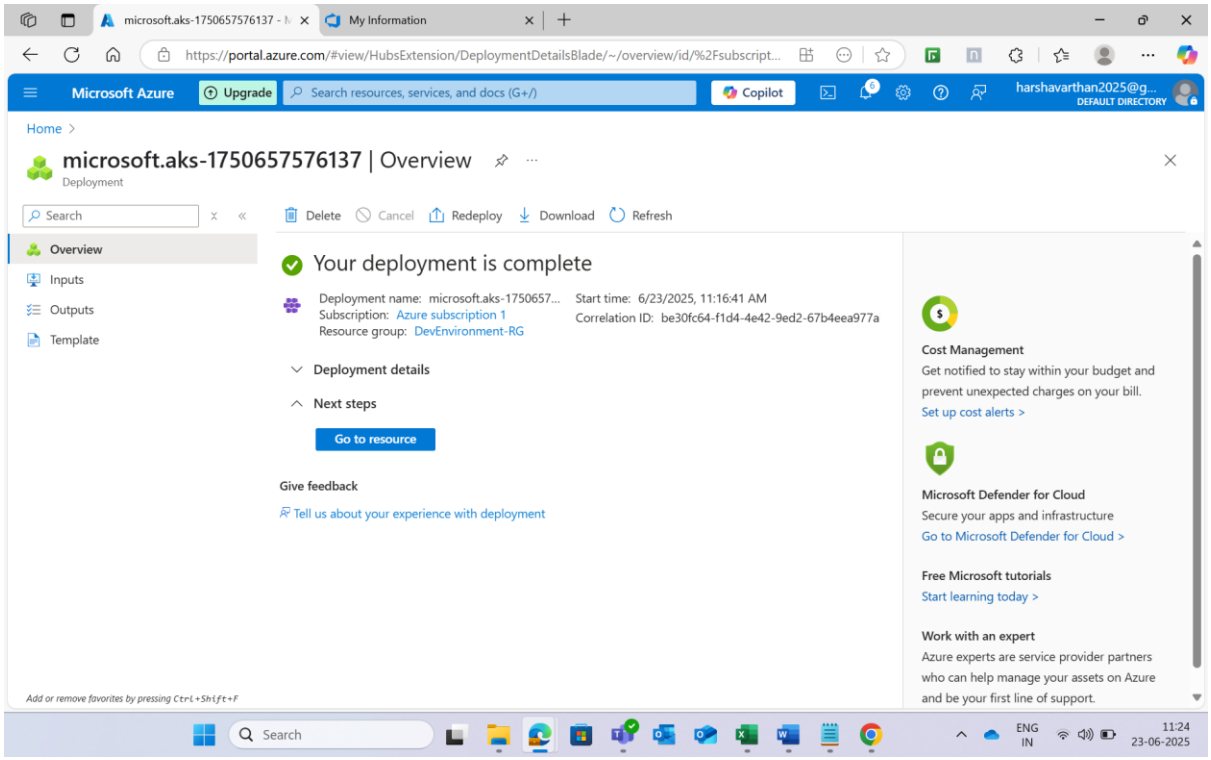


Step 14:

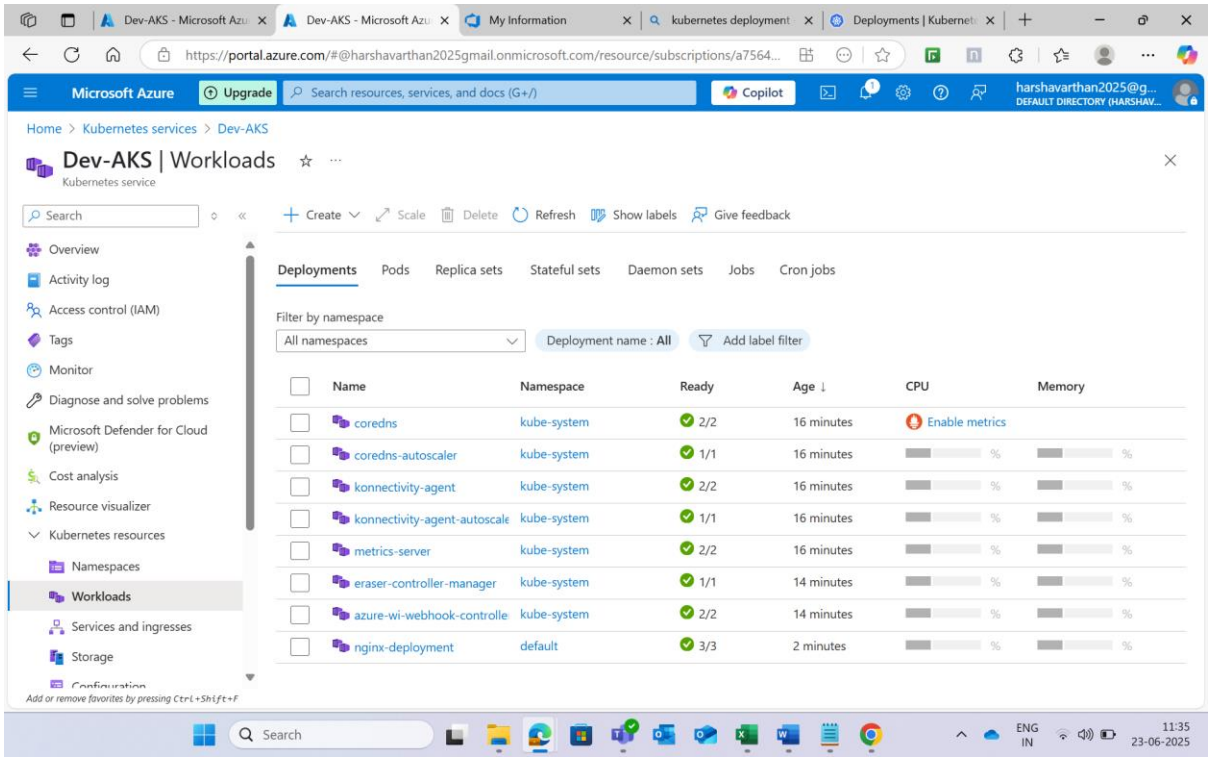
Create a Azure Kubernetes Cluster to Deploy the images.

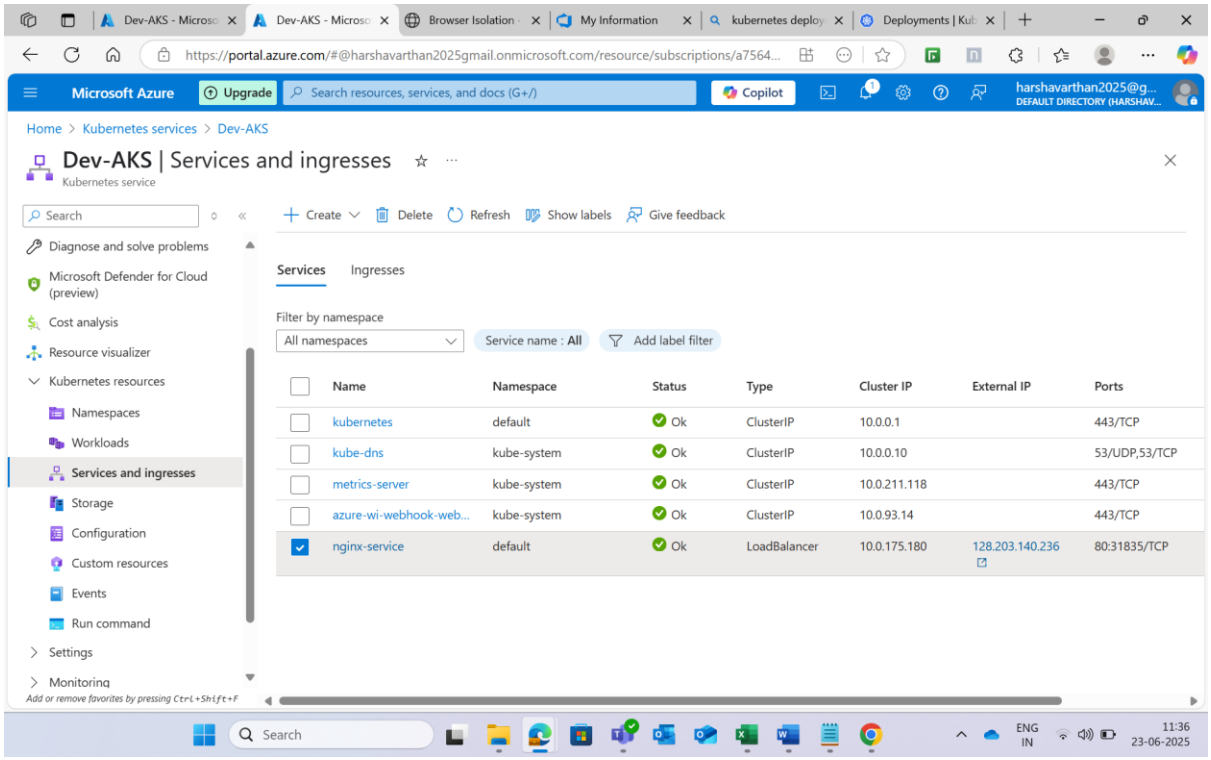
- **Cluster Name:** Dev-AKS
- **Node Count:** 1
- **Node Size:** Standard B2s (Free Tier eligible)





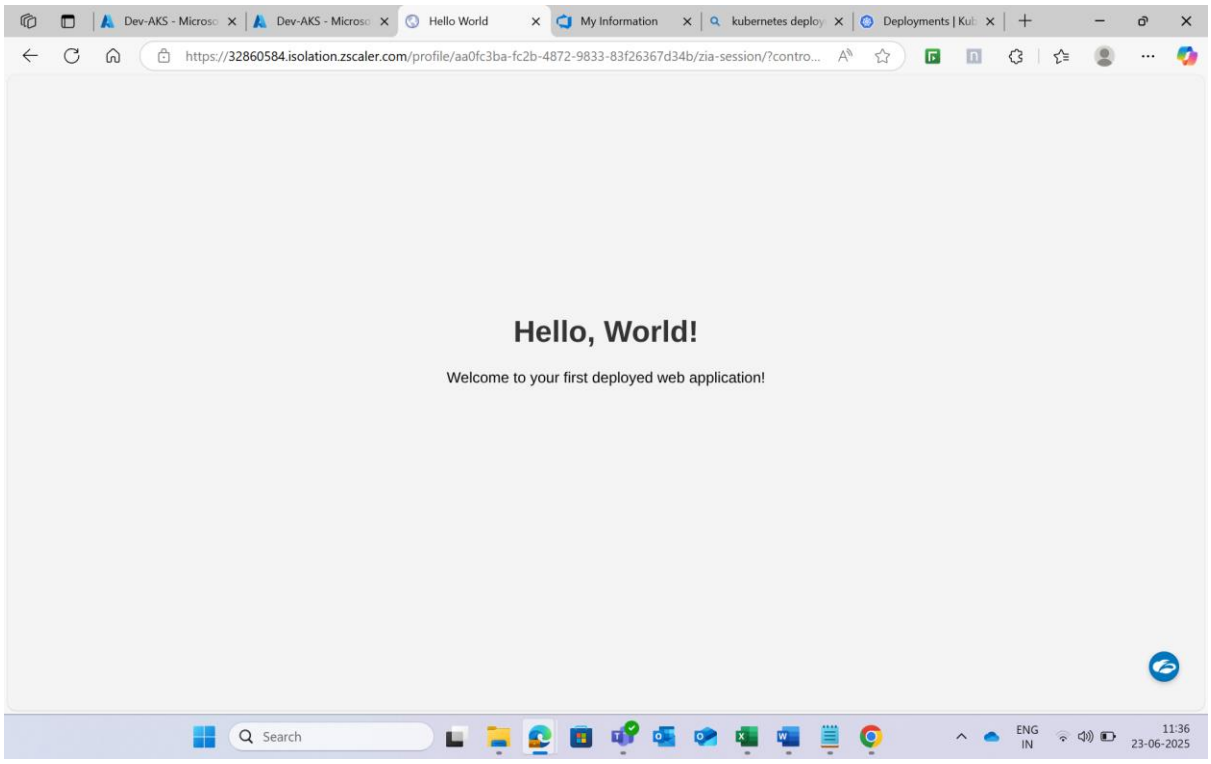
- Create a Deployment.yaml file and deploy it.
- Create a Service.yaml.





Step 15:

After Deploying open the ip address of the service.



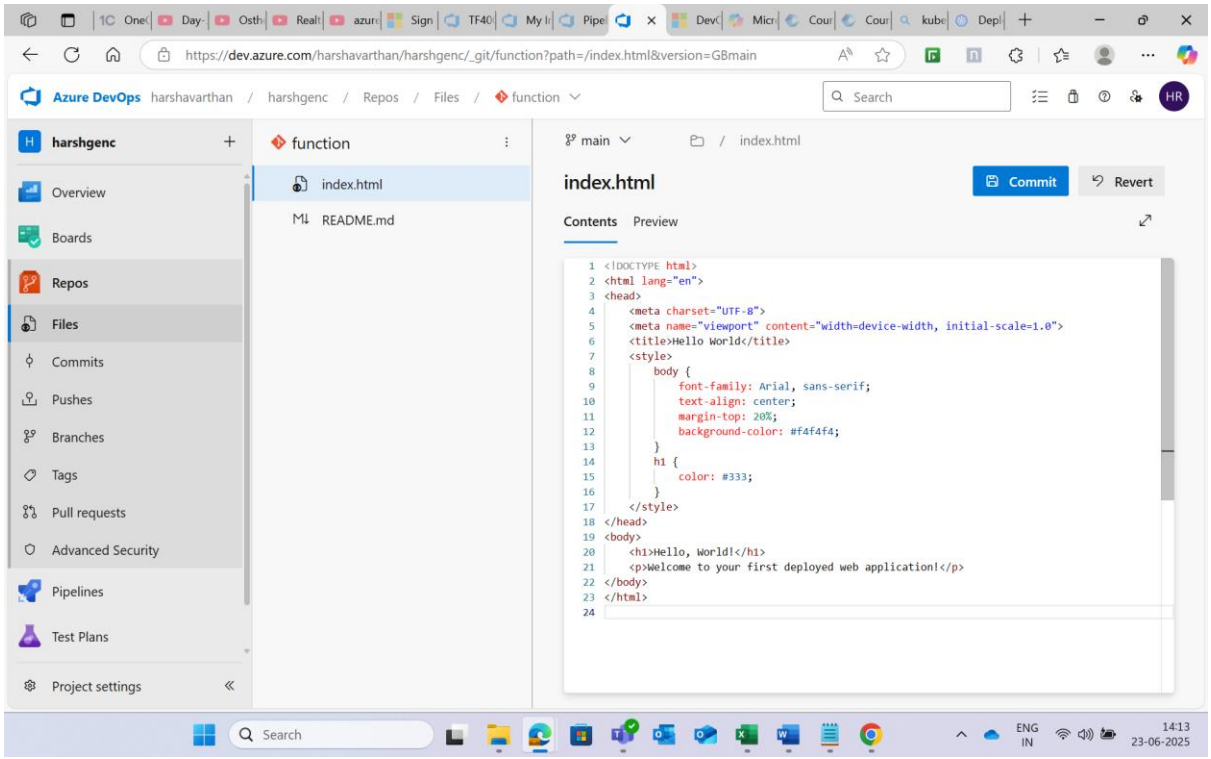
PHASE2: AZURE DEVOPS CI/CD PIPELINES

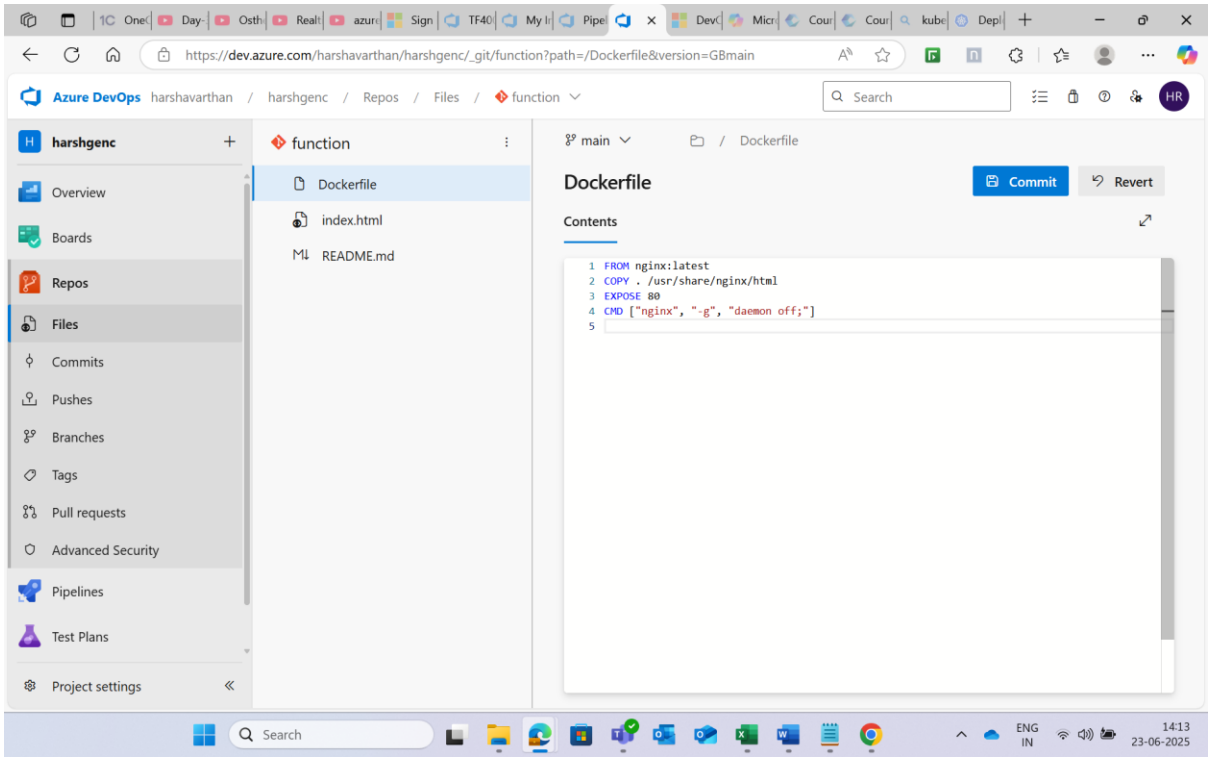
Using Azure Devops we would use build , push an image to the azure container registry and deploy it to the Azure Kubernetes service.

- **CI (Build Pipeline):**
  - Build the Docker image from the application code.
  - Push the image to Azure Container Registry (ACR).
- **CD (Release Pipeline):**
  - Pull the Docker image from ACR.
  - Deploy the application to Azure Kubernetes Service (AKS).

Step 1:

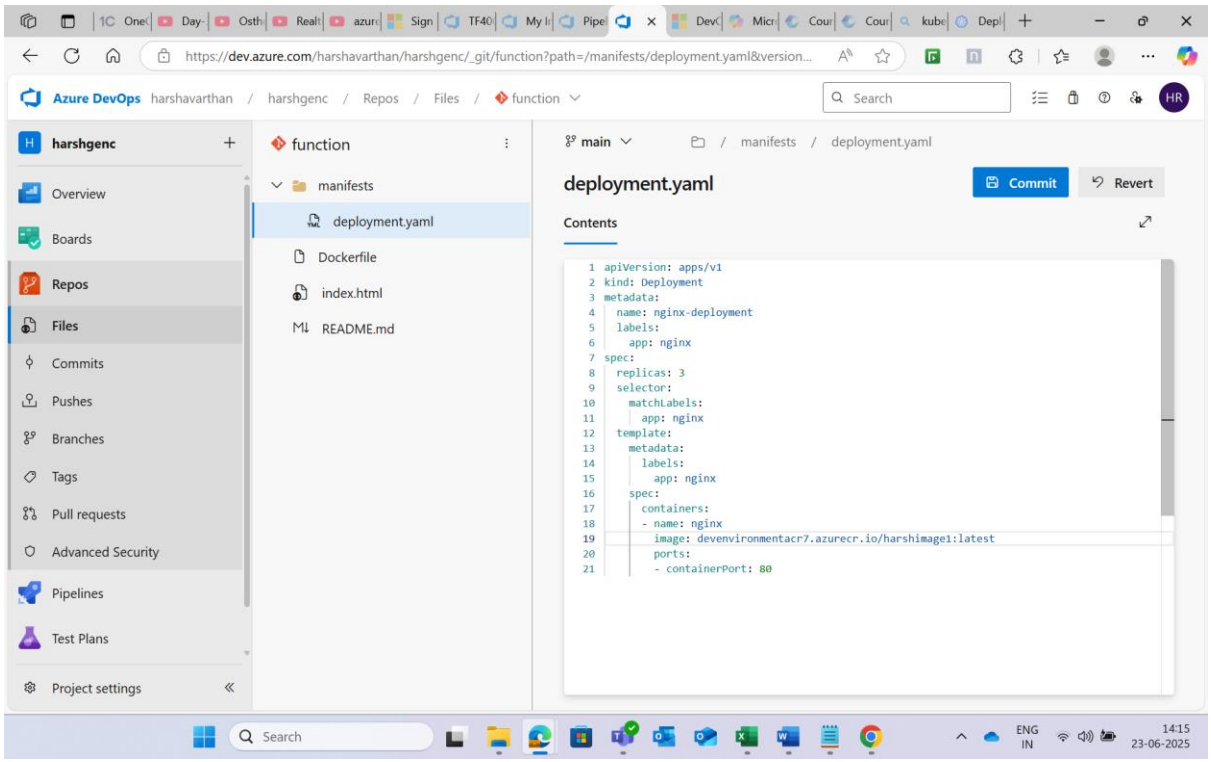
In azure repos create files named as “index.html” and “Dockerfile”.

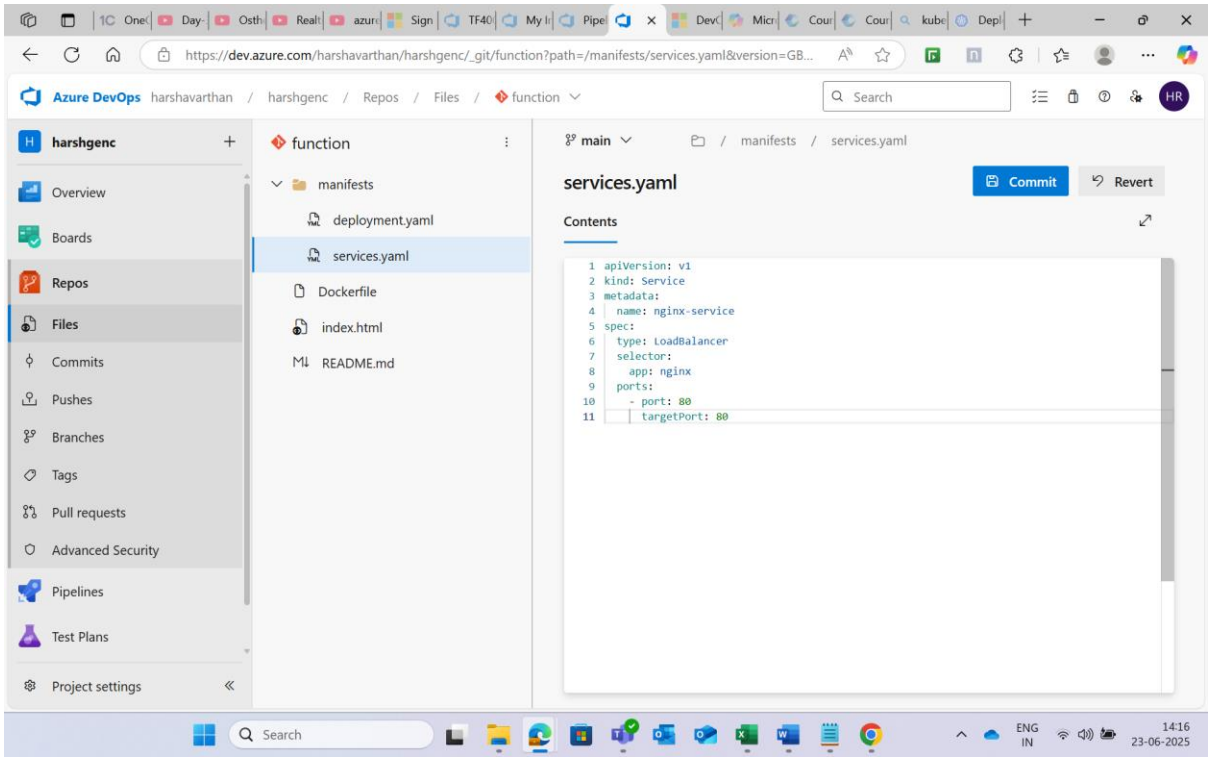




Step 2: Create a folder and add a “deployment.yaml “ file

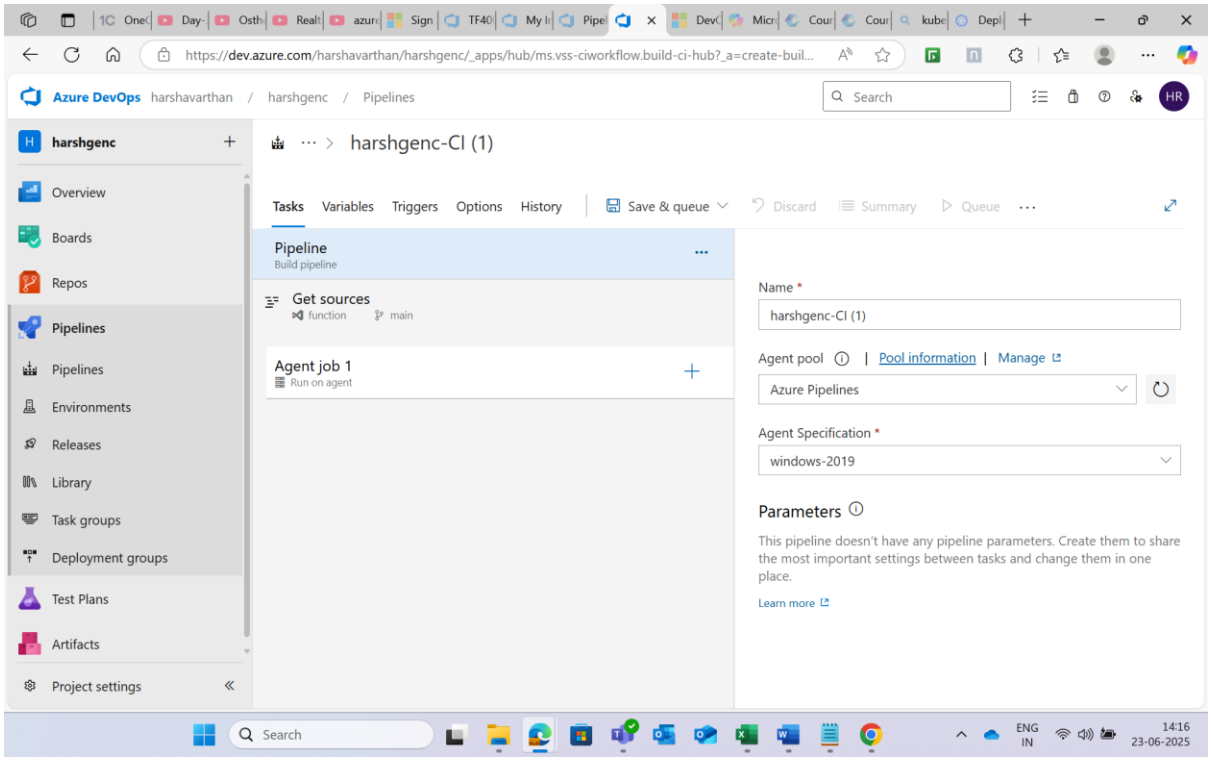
Create another file “Service.yaml”





Step 3: After the repos setup , I was not able to run the pipeline because of the **parallelism error**. So I filled the form to get access but I did’nt get the access.

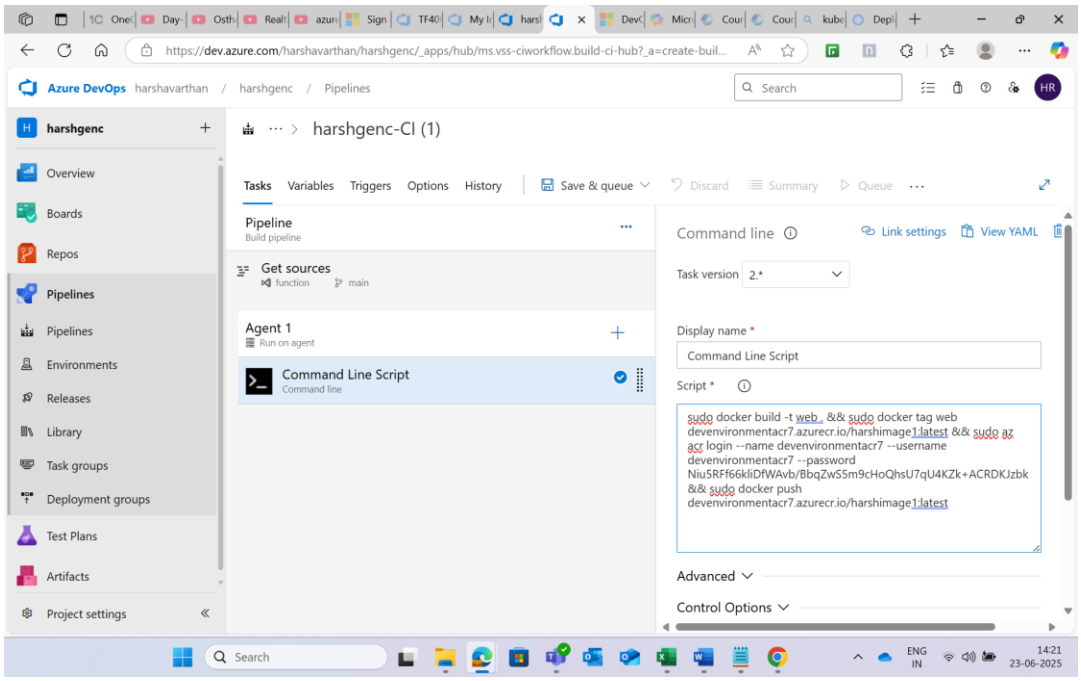
- So I used self hosted agent (created a virtual machine in azure and used that as agent to run pipeline)



Step 4:

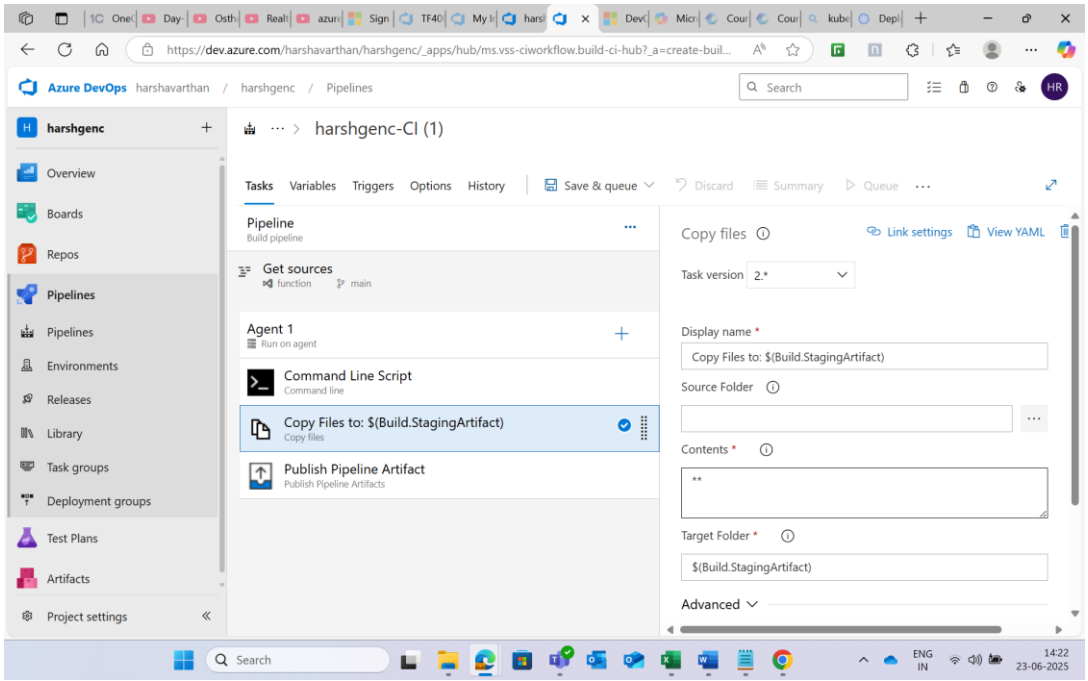
The first task was command line where I used the commands to :

- Build a docker image
- Login to the azure container registry
- Push the image to Azure Container registry



Step 5:

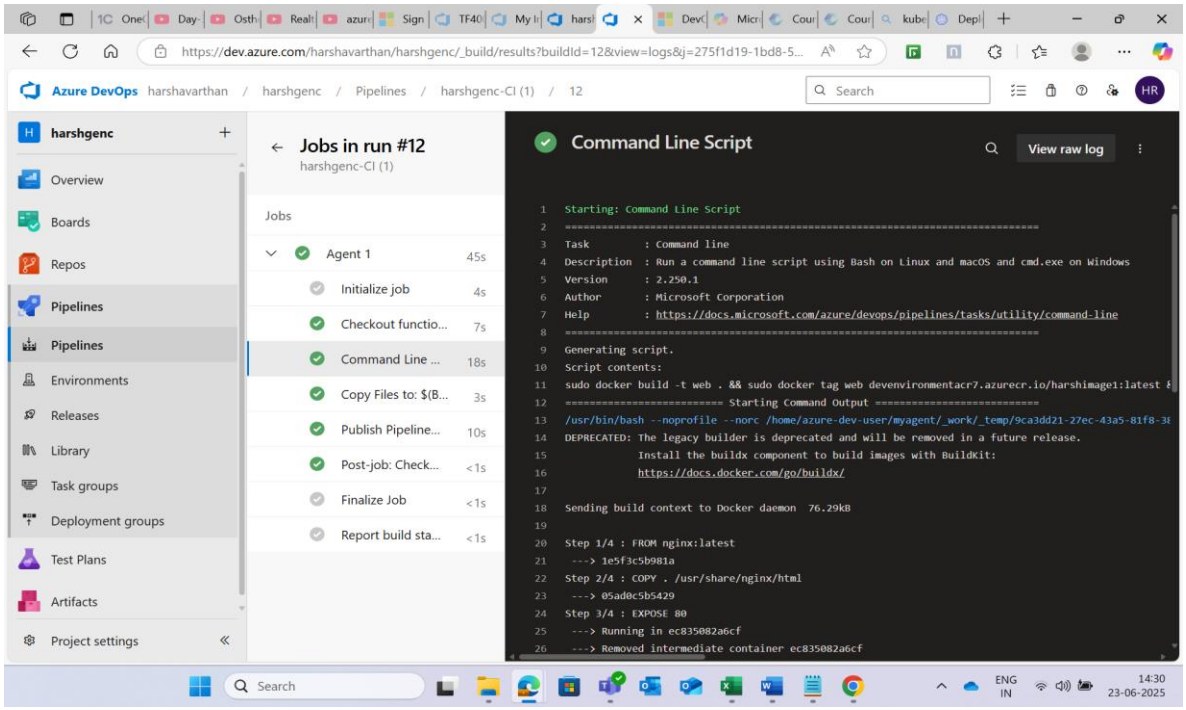
The next task is to add the Copy files task where I mentioned the target folder path to store the build.



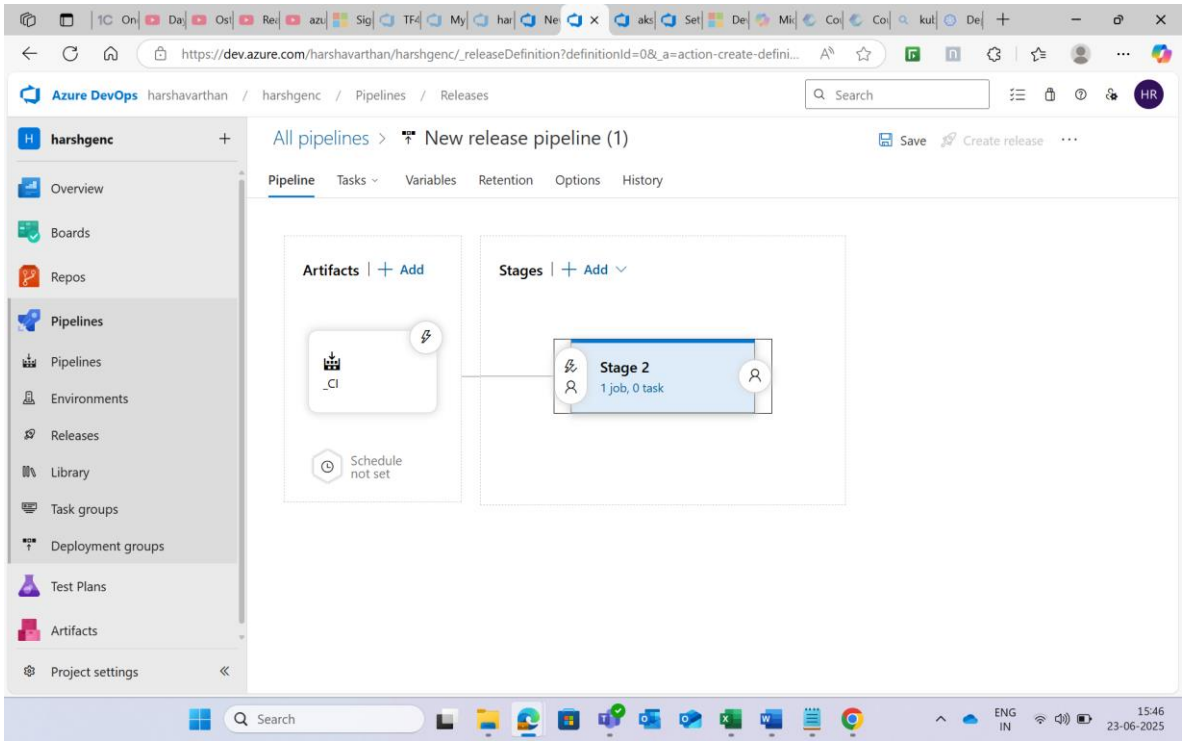
Step 6: Added publish artifact task.

Step 7: Saved and Created run the pipeline.

The pipeline executed correctly and image has been pushed to the azure container registry.



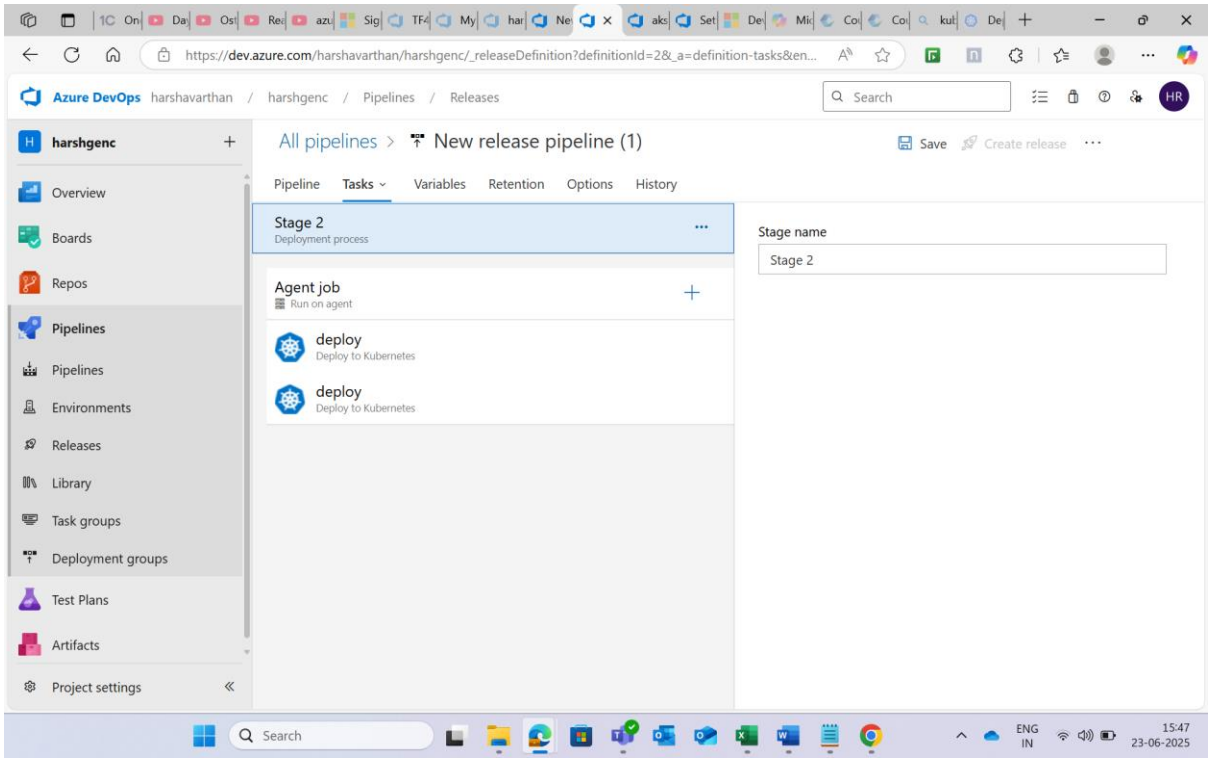
The next phase is to create the release pipeline to deploy the image in azure Kubernetes.



Step 8:

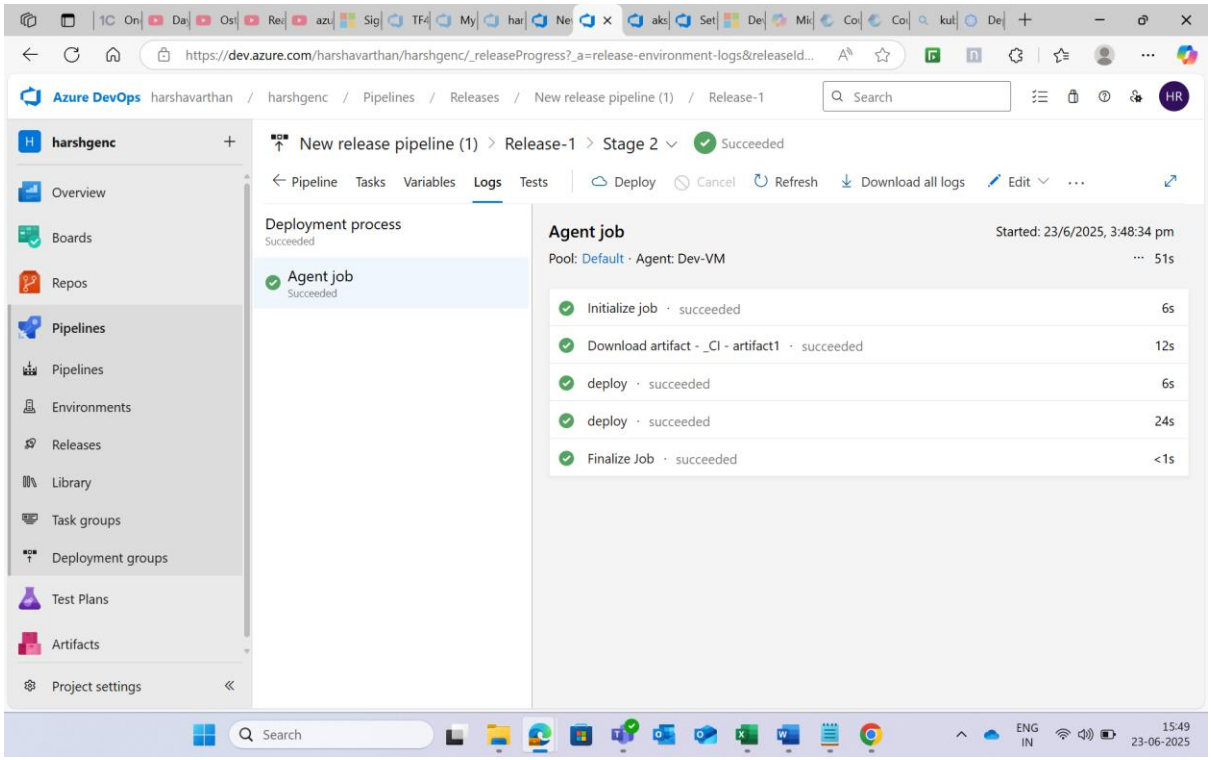
create a new release pipeline with empty job and add agent pool as “Default”

- Add task “ Deploy to Kubernetes” two times for separate deployment and service
- Give the deployment.yaml path to the first task.
- Give the service.yaml path to the next task.
- Save and Run the Pipeline.



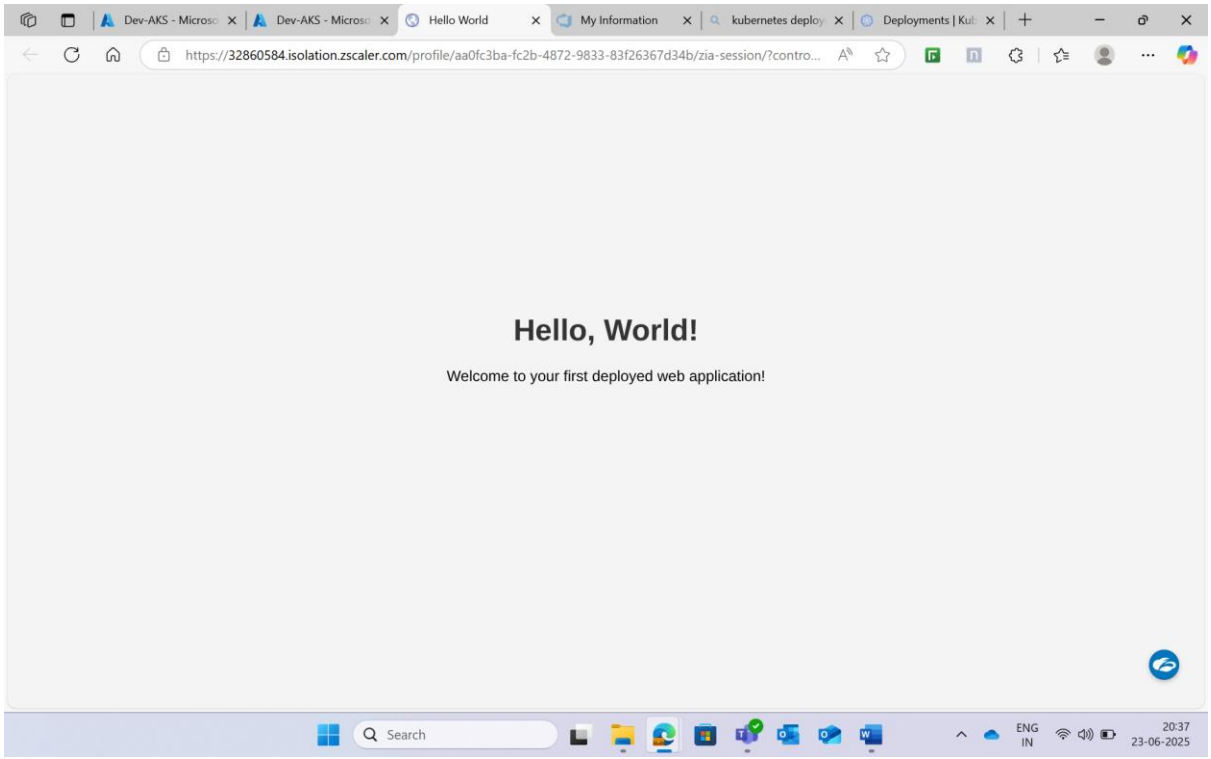
Step 9:

The release pipeline executed successfully and the image has been deployed in the azure Kubernetes services.



Step 10:

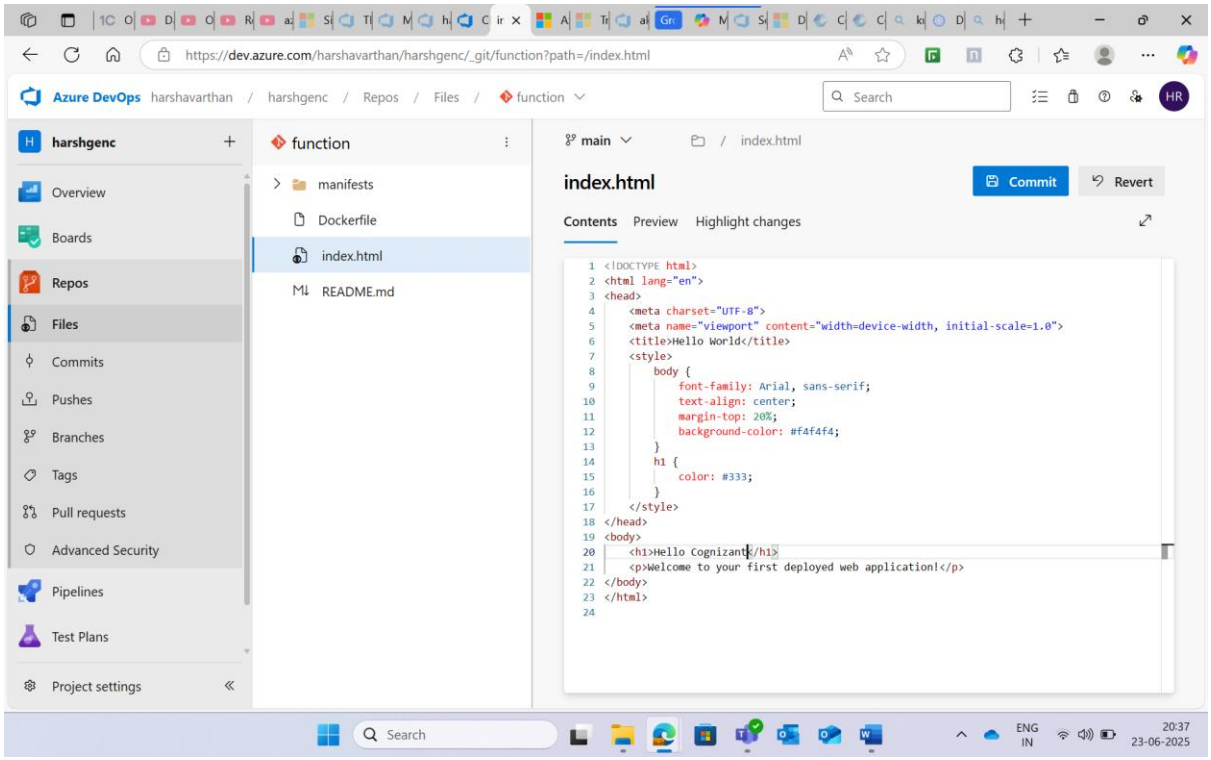
Opened the external ip address in the services to open the deployed image.



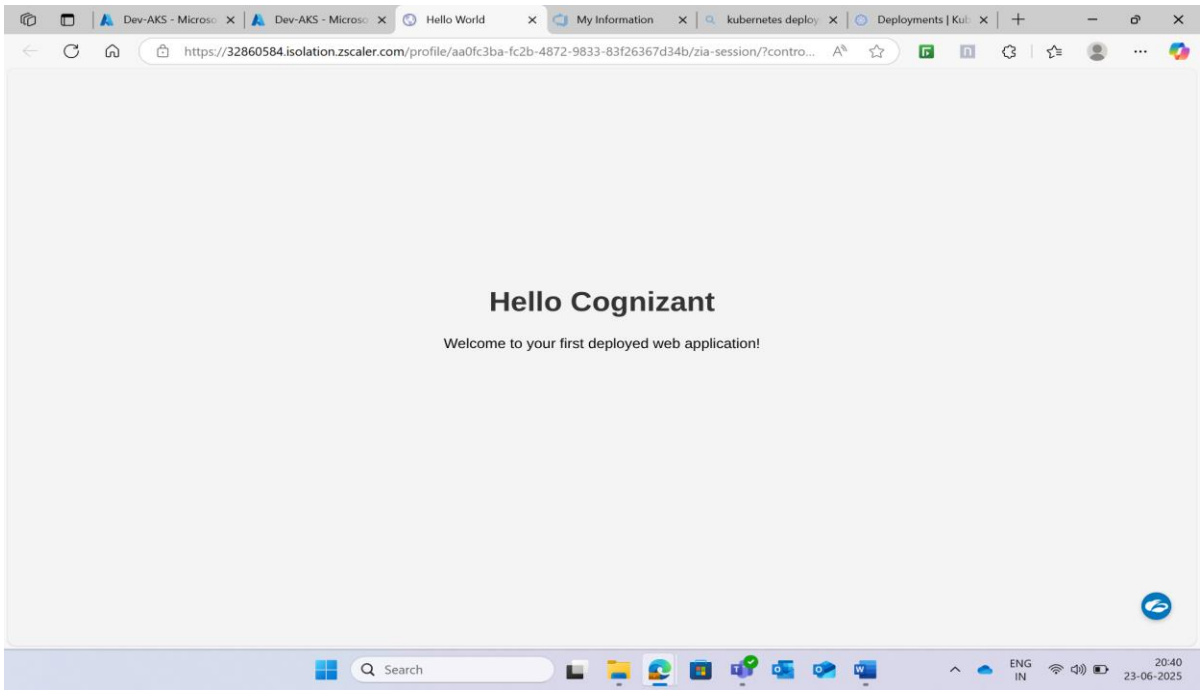
PHASE 3

The next phase is to make changes in the source file and check whether it triggered automatically and got deployed again.

- I made a change in “index.html” from “Hello world” to “hello Cognizant”.
- It automatically triggered the pipeline and release pipeline.



- When I checked the deployed imaged it got changed.



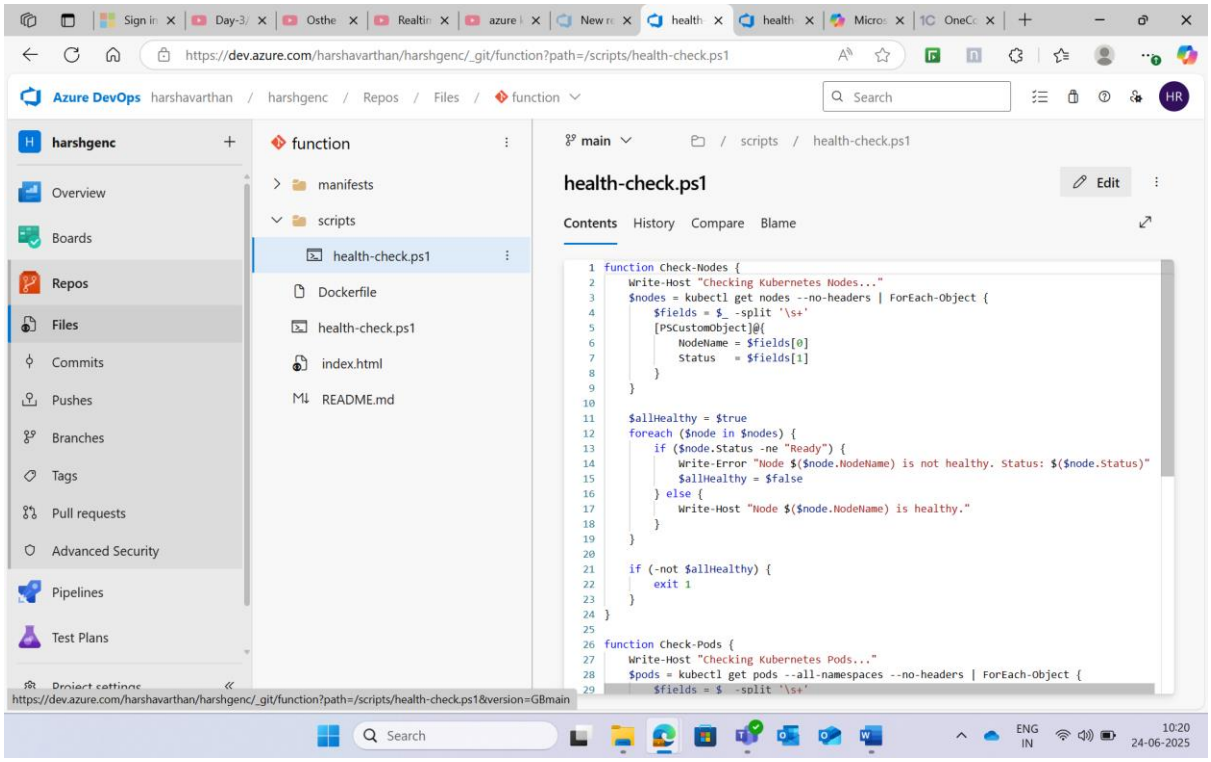
PHASE 4

This phase has the health check using powershell script to check the podes and nodes in the azure container service.

- If the pods and nodes are all online and running then the script will run successfully.
- Else the script will stop with errors.

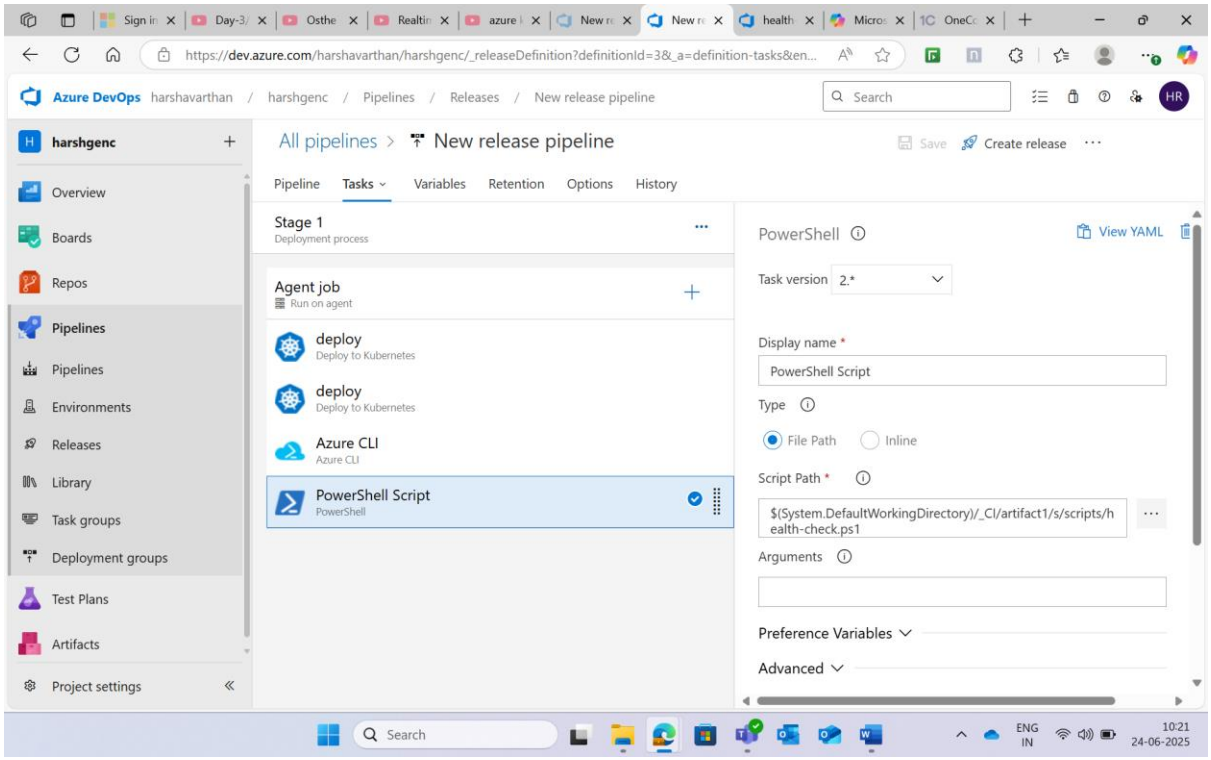
Step 1:

Add the health-check.ps1 script in the azure repos.

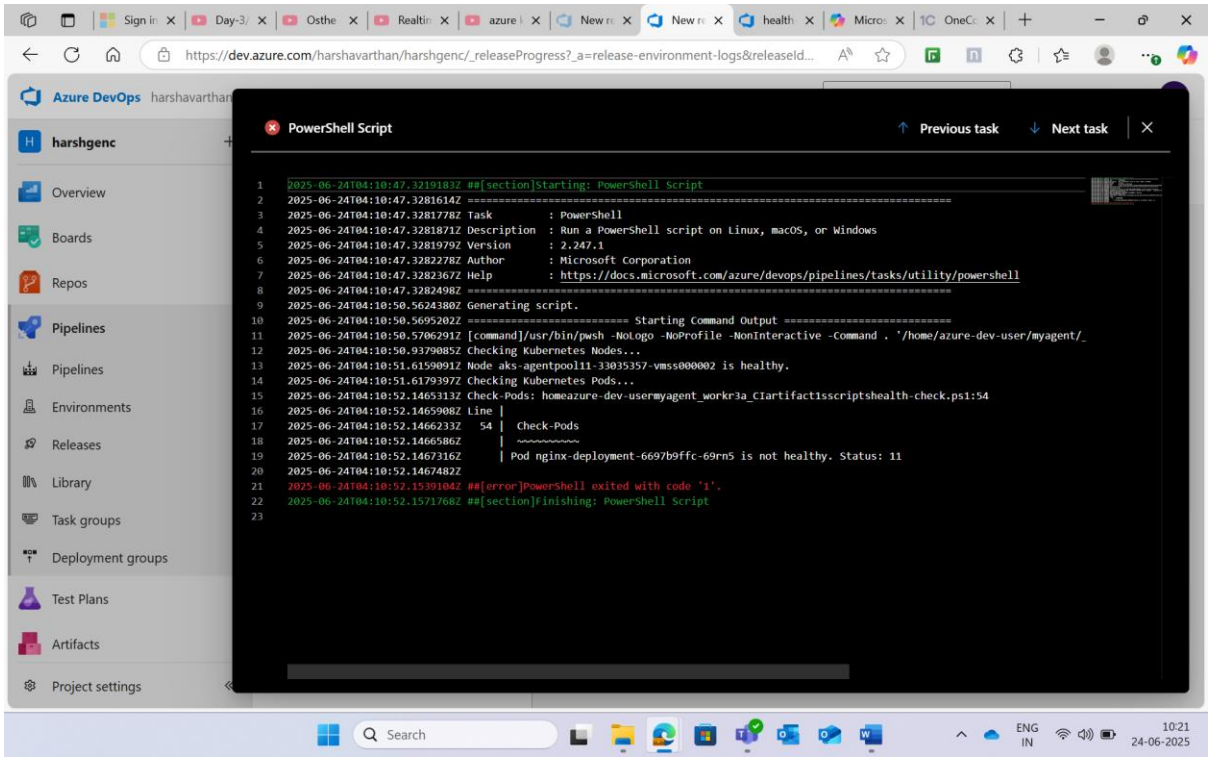


Step 2:

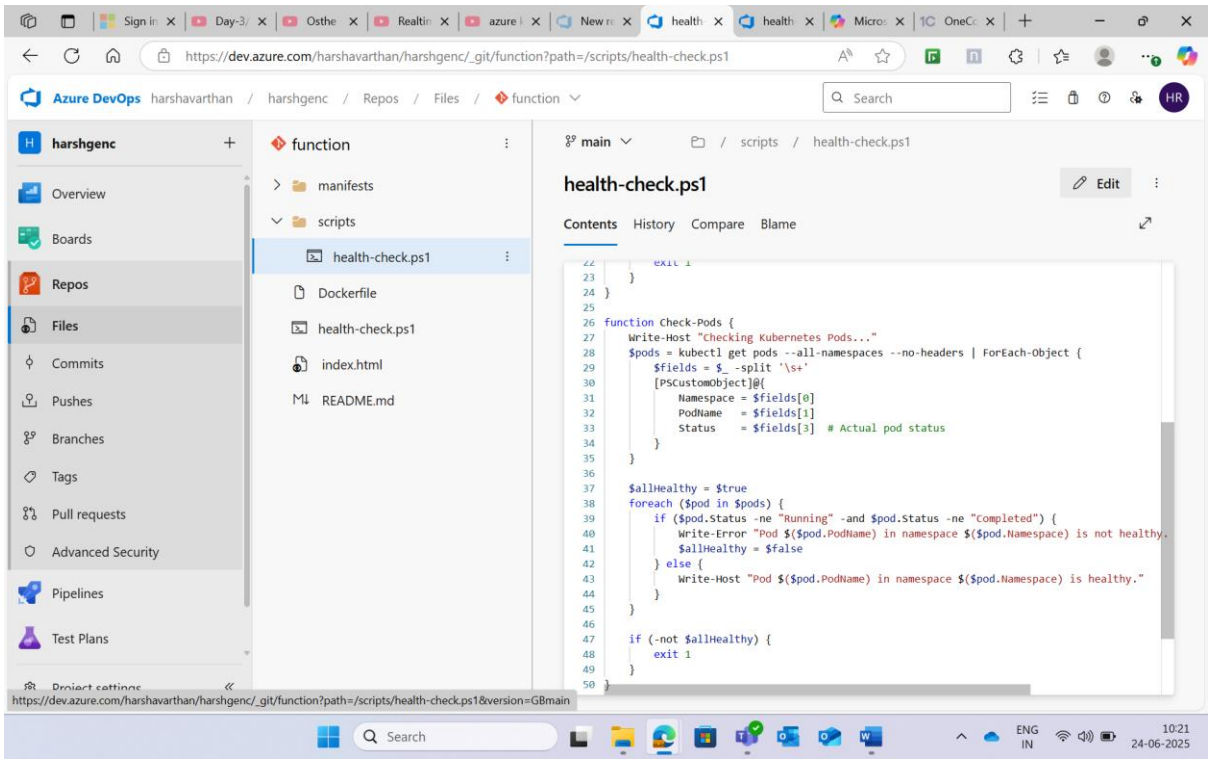
In the release pipeline add a task as “ powershell Script” and Give the path of the health-check.ps1 file the the script path.



- The script exited with the error as pods not healthy.
- But the pod was online and running.
- Then I figured out that the script given in the usecase has some mistakes .



- Then I changed the script with correct script and gave run pipeline.



- It executed correctly and gave the message “All Kubernetes components are healthy”