

Course-End Project

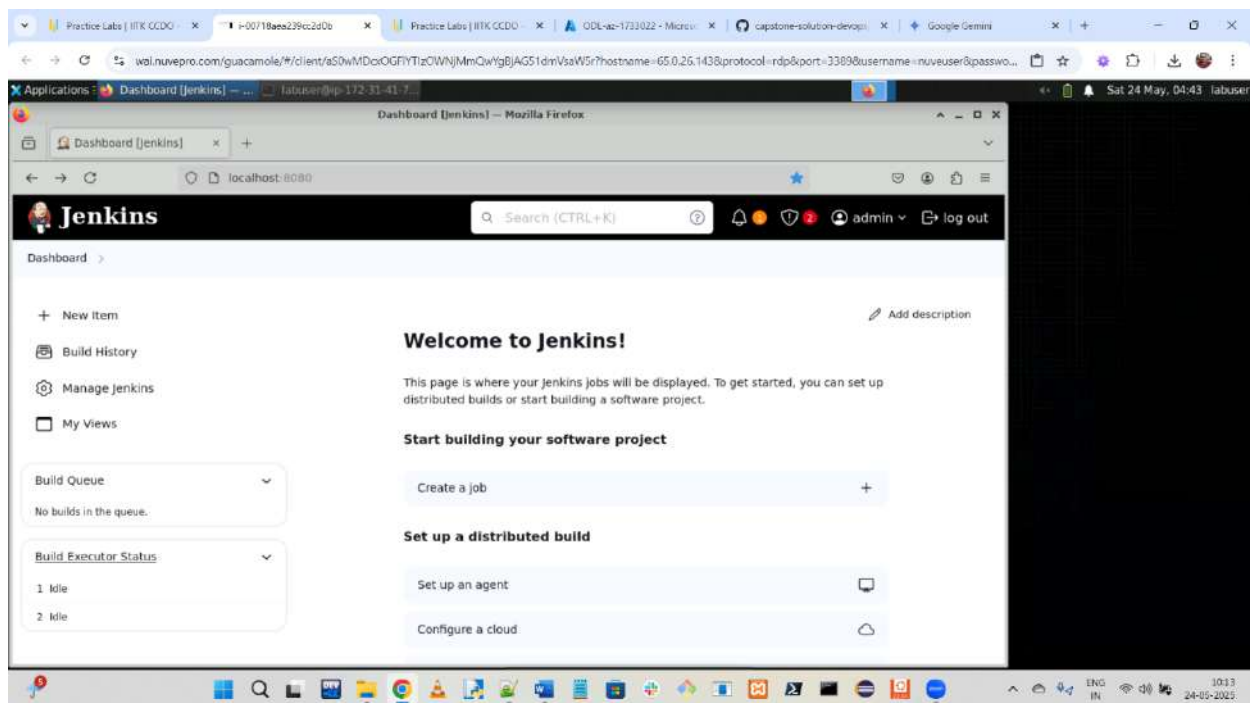
Application Deployment to Multi-Cloud

Steps to be followed:

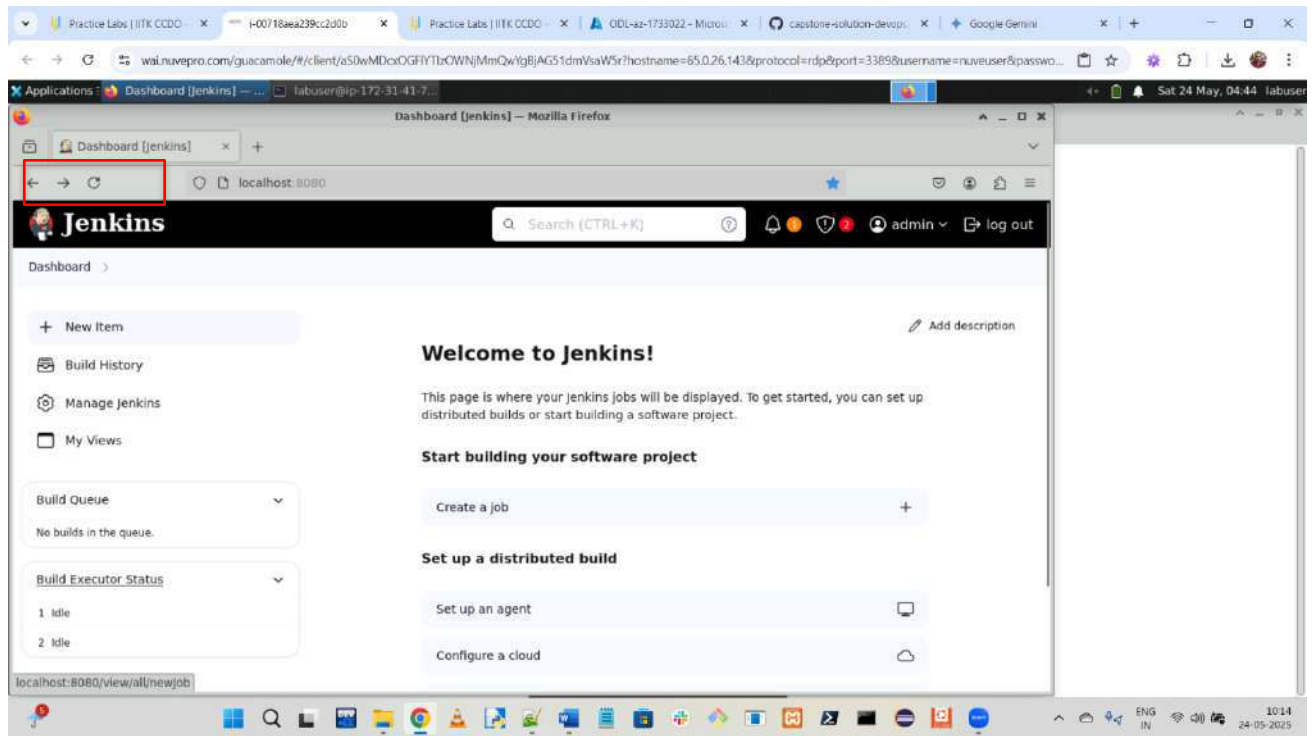
1. Create Jenkins pipeline for CI of Java application
2. Configure Ansible to connect AWS and Azure VMs
3. Configure Docker in Jenkins for CI/CD
4. Configure Ansible on Simplilearn lab
5. Validate Deployment

Step 1: Create Jenkins pipeline for CI of Java application

- 1.1 Navigate to the DevOps lab where Jenkins is preinstalled, Open a browser and access Jenkins at <http://localhost:8080/>



- 1.2 Open Jenkins and click on **New Item** to create a new Jenkins job



Before proceeding further open terminal and then validate if ansible, docker and maven runtimes are installed on VM

- 1.3 Execute the following command to check the Docker version and confirm that Docker is correctly installed on your system

docker version

The screenshot shows a terminal window with the following output:

```
(base) labuser@ip-172-31-41-71:~$ [GFX1-]: RenderCompositorSMGL failed mapping default framebuffer, no dt
^C
(base) labuser@ip-172-31-41-71:~$ az logout
(base) labuser@ip-172-31-41-71:~$ docker version
Client: Docker Engine - Community
Version: 27.1.1
API version: 1.46
Go version: go1.21.12
Git commit: 6312585
Built: Tue Jul 23 19:59:47 2024
OS/Arch: linux/arm64
Context: default

Server: Docker Engine - Community
Engine:
Version: 27.1.1
API version: 1.46 (minimum version 1.24)
Go version: go1.21.12
Git commit: ccl3f95
Built: Tue Jul 23 19:59:47 2024
OS/Arch: linux/arm64
Experimental: false
containerd:
Version: 1.7.19
GitCommit: 2bf793ef6dc9a18e0cb12efb64355c2c9d5eb41
runc:
Version: 1.7.19
GitCommit: v1.1.13-0-g58aa920
docker-init:
Version: 0.19.0
GitCommit: de40ad0
(base) labuser@ip-172-31-41-71:~$ ansible --version
ansible [core 2.16.9]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/labuser/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
```

1.4 Execute the following command to check the Ansible version and configuration details
ansible version

The screenshot shows a terminal window with the following output:

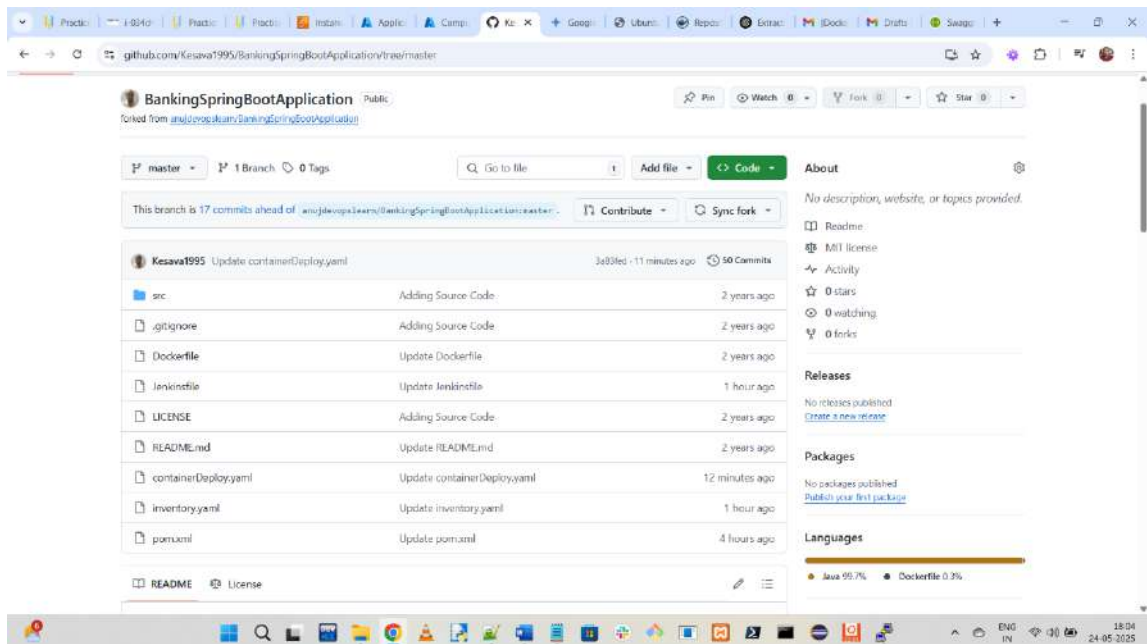
```
(base) labuser@ip-172-31-41-71:~$ ansible --version
ansible [core 2.16.9]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/labuser/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/labuser/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Jul 29 2024, 16:56:48) [GCC 11.4.0] (/usr/bin/python3)
  jinja2 version = 3.0.3
  libyaml = True
(base) labuser@ip-172-31-41-71:~$
```

Step 2: Configure Ansible to connect AWS and Azure VMs

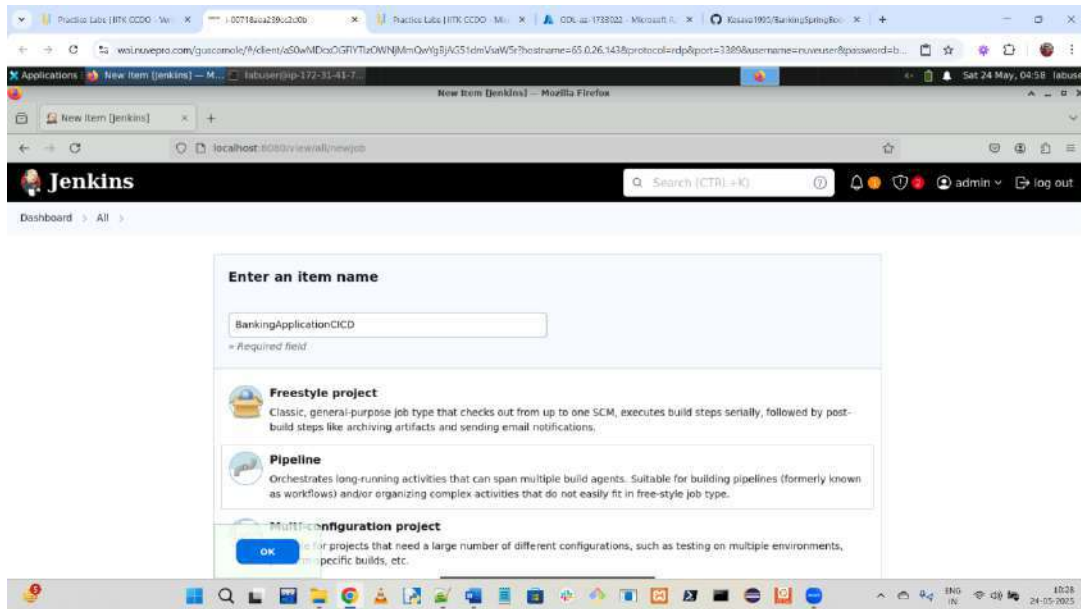
2.1 Fork the GitHub repository below to execute the Jenkins pipeline, and validate that Maven scripts are present in the code repository for source code validation

<https://github.com/Kesava1995/BankingSpringBootApplication>

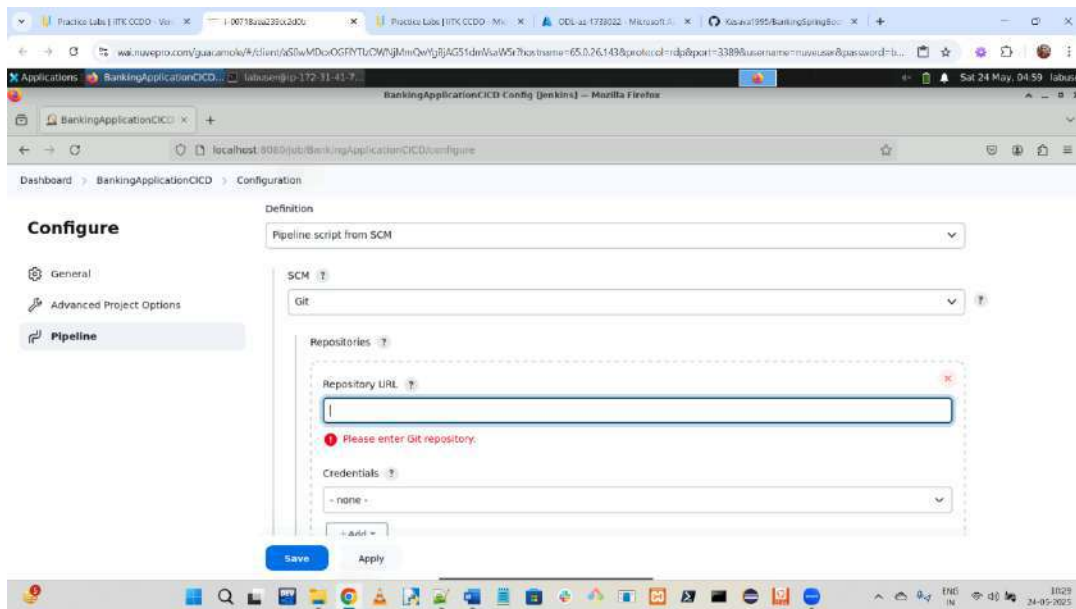
2.2 Access Jenkins application and click on **New Item** to create a new Jenkins job



2.3 Select desired Jenkins pipeline job type and fill in job name as per project requirement

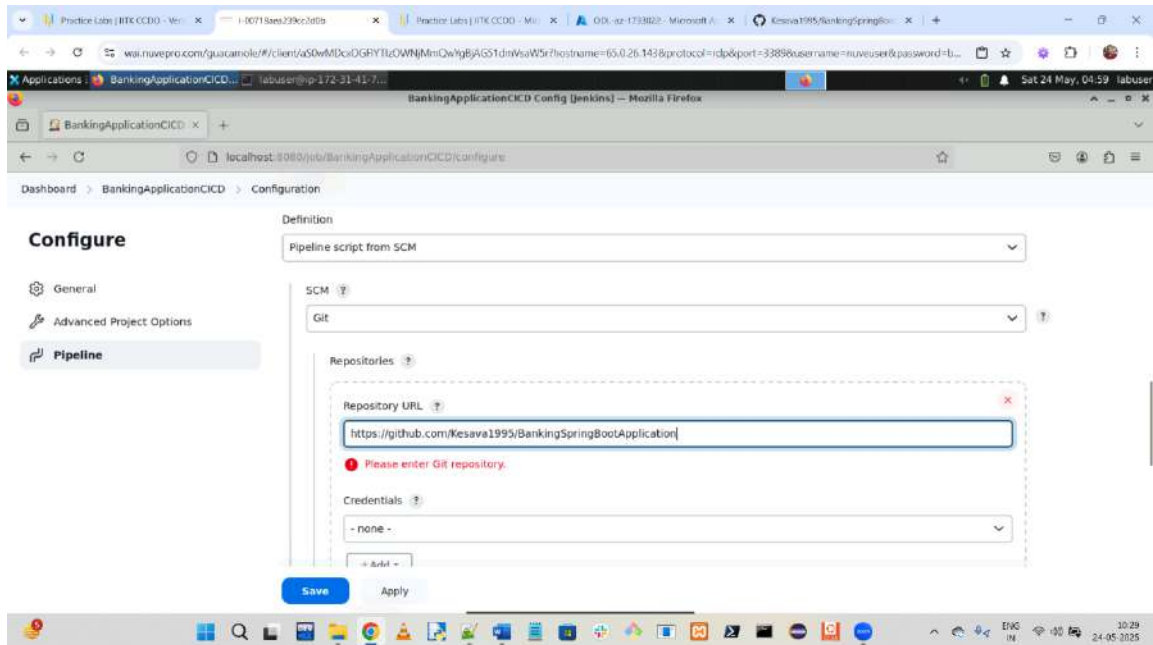


- 2.4 In your Jenkins project configuration, set the Pipeline Definition to **Pipeline script from SCM**, select SCM as **Git**, and enter the Repository URL where the pipeline script is stored. Once configured, click **Save** to apply the changes, enabling Jenkins to fetch the pipeline script directly from the specified Git repository



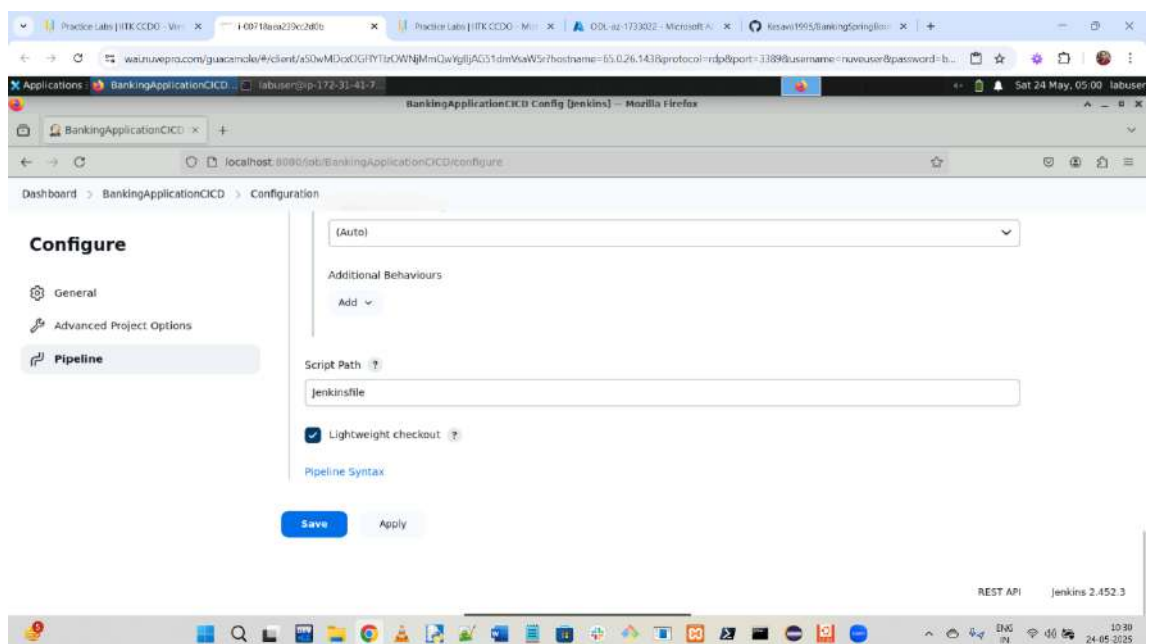
- 2.5 Configure below code repository or forked repository in Jenkins pipeline to checkout Pipeline script from Git version control system

<https://github.com/Kesava1995/BankingSpringBootApplication>



Note: In above code repository we are using default Jenkins pipeline name, but we can modify in case we are using a different pipeline name

2.6 Set the Script Path to **Jenkinsfile** and enable Lightweight checkout in your Jenkins pipeline configuration, then click **Save** to apply these settings and ensure Jenkins can locate and load the pipeline script efficiently



2.7 Create a **Jenkinsfile** in your code repository and add Code Checkout stage in the pipeline script

```
node{

    def tag, dockerHubUser, containerName, httpPort = ""

    stage('Prepare Environment'){
        echo 'Initialize Environment'
        tag="3.0"
        withCredentials([usernamePassword(credentialsId: 'dockerHubAccount', usernameVariable:
'dockerUser', passwordVariable: 'dockerPassword'))] {
            dockerHubUser="$dockerUser"
        }
        containerName="bankingapp"
        httpPort="8989"
    }

    stage('Code Checkout'){
        try{
            checkout scm
        }
        catch(Exception e){
            echo 'Exception occured in Git Code Checkout Stage'
            currentBuild.result = "FAILURE"
        }
    }

    stage('Maven Build'){
        echo '--- Diagnosing Maven Build ---'
        echo '1. Current Working Directory:'
        sh 'pwd'
        echo '2. Contents of pom.xml:'
        sh 'cat pom.xml' // CRITICAL: This will show us what pom.xml Jenkins is actually using
        echo '3. Java Version on Jenkins Agent:'
        sh 'java -version' // CRITICAL: Confirm the JDK version
        echo '4. Running Maven with debug logging:'
        sh 'mvn clean package -X' // CRITICAL: This will provide verbose output
        echo '--- End Diagnosis ---'
    }
}
```

```

stage('Docker Image Build'){
    echo 'Creating Docker image'
    sh "docker buildx build --platform linux/amd64 -t $dockerHubUser/$containerName:$tag --load ."
}

stage('Publishing Image to DockerHub'){
    echo 'Pushing the docker image to DockerHub'
    withCredentials([usernamePassword(credentialsId: 'dockerHubAccount', usernameVariable:
'dockerUser', passwordVariable: 'dockerPassword'))] {
        sh "docker login -u $dockerUser -p $dockerPassword"
        sh "docker push $dockerUser/$containerName:$tag"
        echo "Image push complete"
    }
}

stage('Ansible Playbook Execution'){
    withCredentials([string(credentialsId: 'ssh_password', variable: 'password')]) {
        sh "export ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i
inventory.yaml containerDeploy.yaml -e httpPort=$httpPort -e containerName=$containerName -e
dockerImageTag=$dockerHubUser/$containerName:$tag -e ansible_password=$password -e
key_pair_path=/var/lib/jenkins/server.pem --become"
    }
}
}

```

Note: The Jenkins file and all the other files are present in the GitHub repository

2.8 Below is the build automation stage which will invoke maven command to perform compilation, test execution and packaging

```

stage('Maven Build'){
    sh "mvn clean package"
}

```

Note: We need to integrate Docker image build process within your Jenkins pipeline, so we must create a Dockerfile build script which will help in building custom Docker image

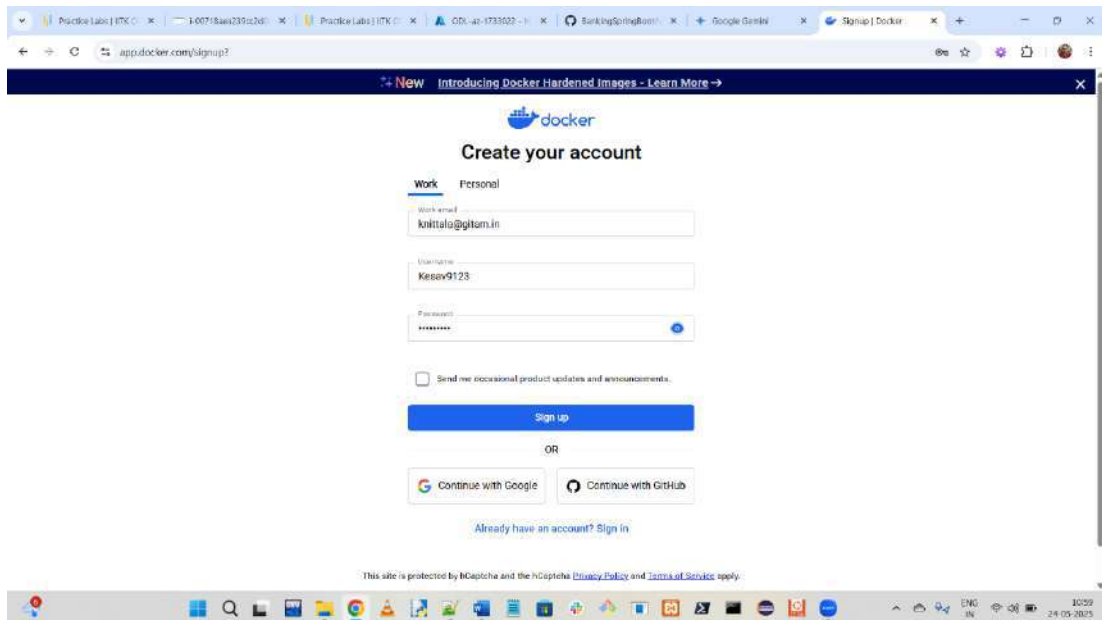
2.9 Create Dockerfile script for Backend:

FROM openjdk:11


```
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8989
ENTRYPOINT ["java","-jar","/app.jar"]
```

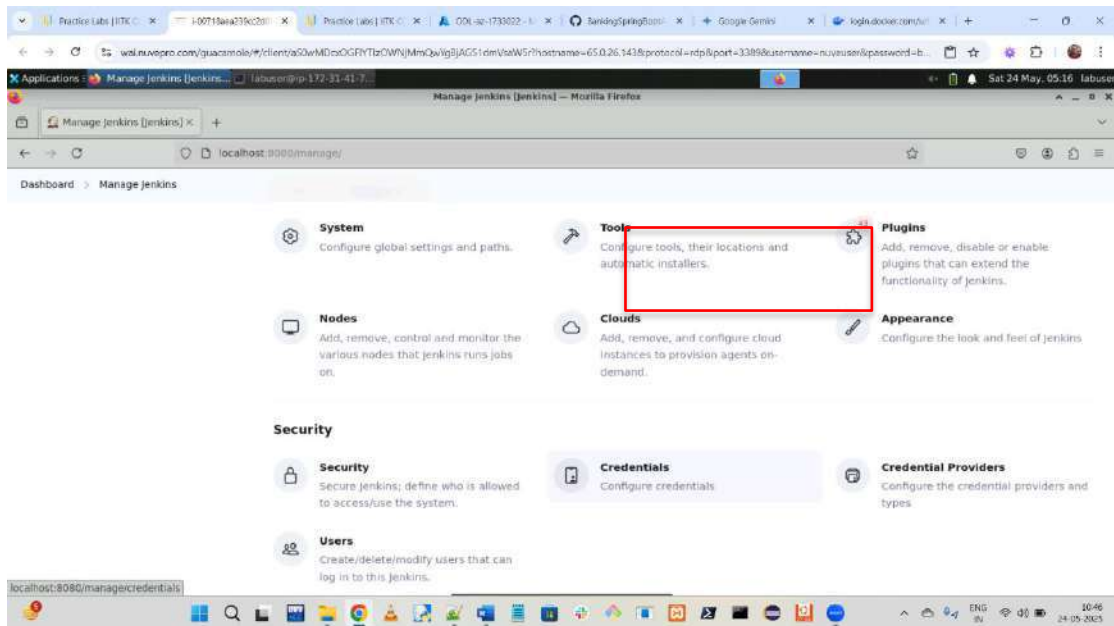
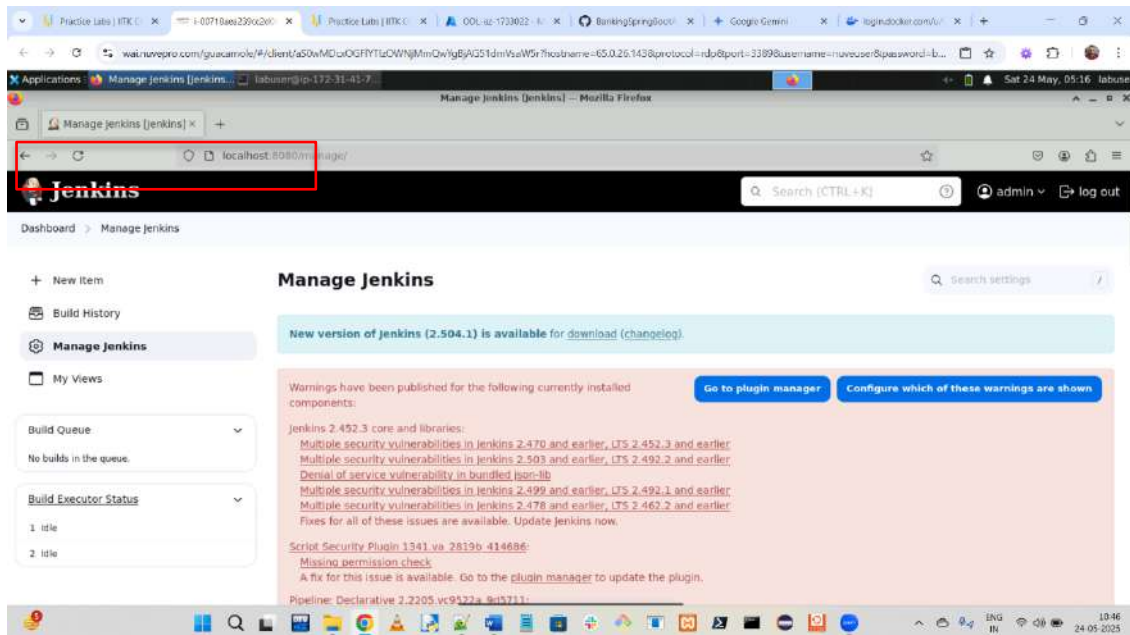
Note: Once Jenkins job configuration is done, we must create a Docker credential which will be used to push Docker image to Docker hub

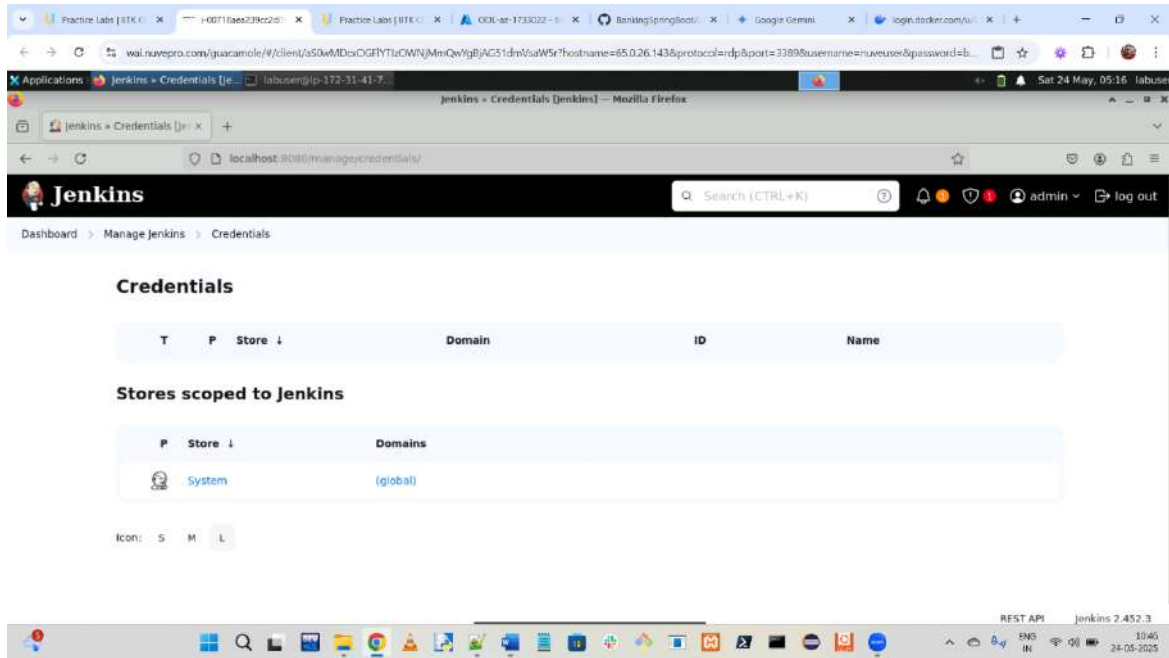
- 2.10 Navigate to Docker hub and create a new free account and keep Docker hub username and password with you before configuring Jenkins credential
<https://hub.docker.com/>



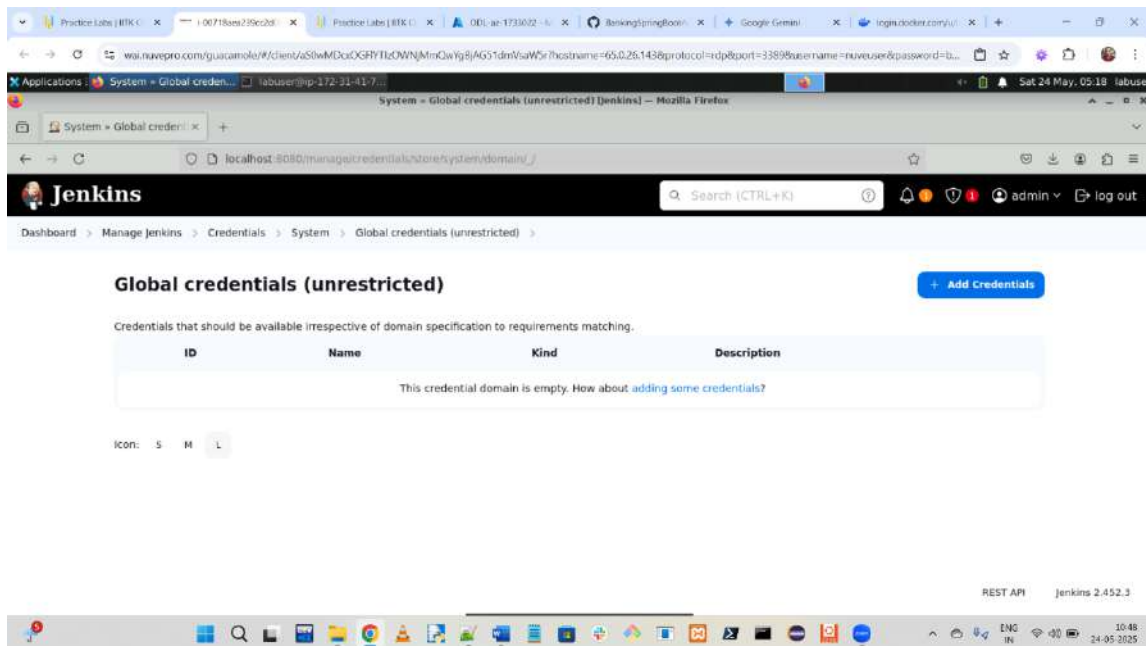
Step 3: Configure Docker in Jenkins for CI/CD

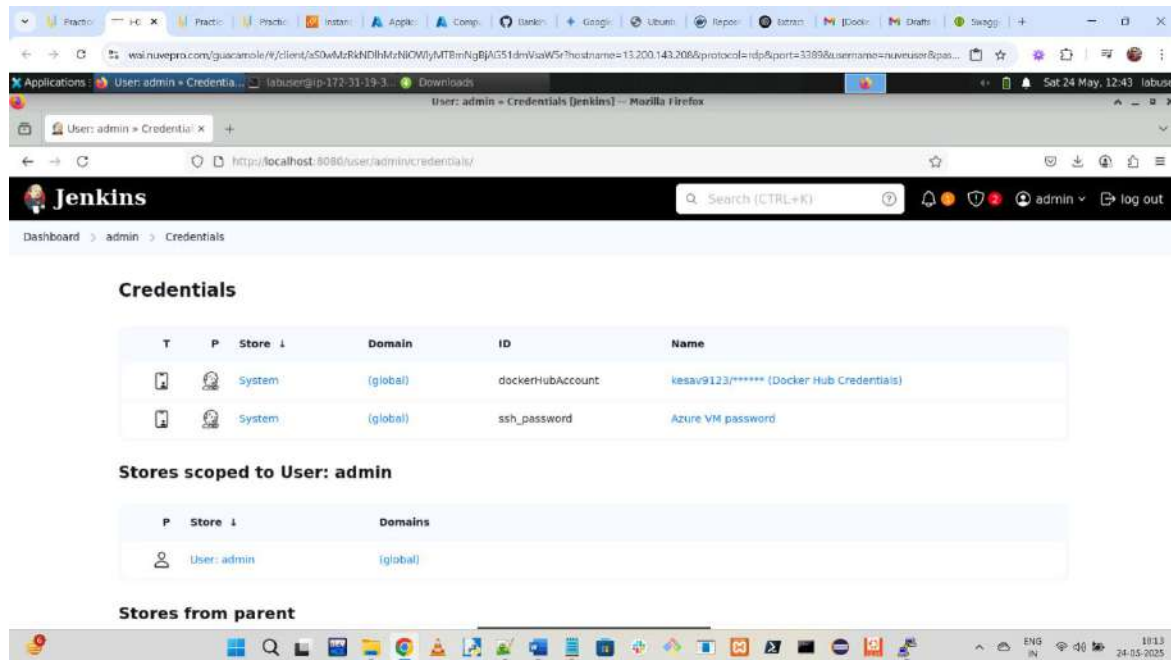
3.1 Once Docker hub account is created, navigate to **Manage Jenkins** and then **Credentials** and then select **global** domain





3.2 Click on **Add Credentials** to create new Docker Hub credentials with the details provided below and click on **Create**





Note: Credential id in below screenshot should match what you have mentioned in Jenkins pipeline

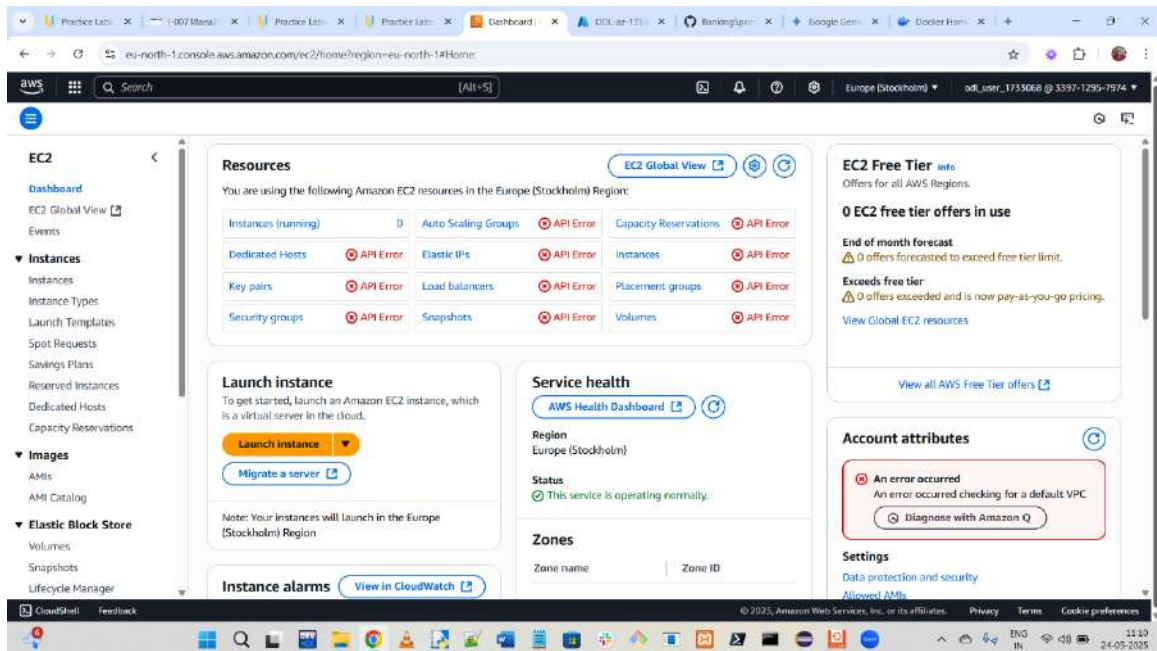
- 3.3 Once Docker image is built, we can integrate Docker image publish stage to publish Docker image to Docker hub registry

```
stage('Publishing Image to DockerHub'){
    echo 'Pushing the docker image to DockerHub'
    withCredentials([usernamePassword(credentialsId: 'dockerHubAccount', usernameVariable:
    'dockerUser', passwordVariable: 'dockerPassword')) {
        sh "docker login -u $dockerUser -p $dockerPassword"
        sh "docker push $dockerUser/$containerName:$tag"
        echo "Image push complete"
    }
}
```

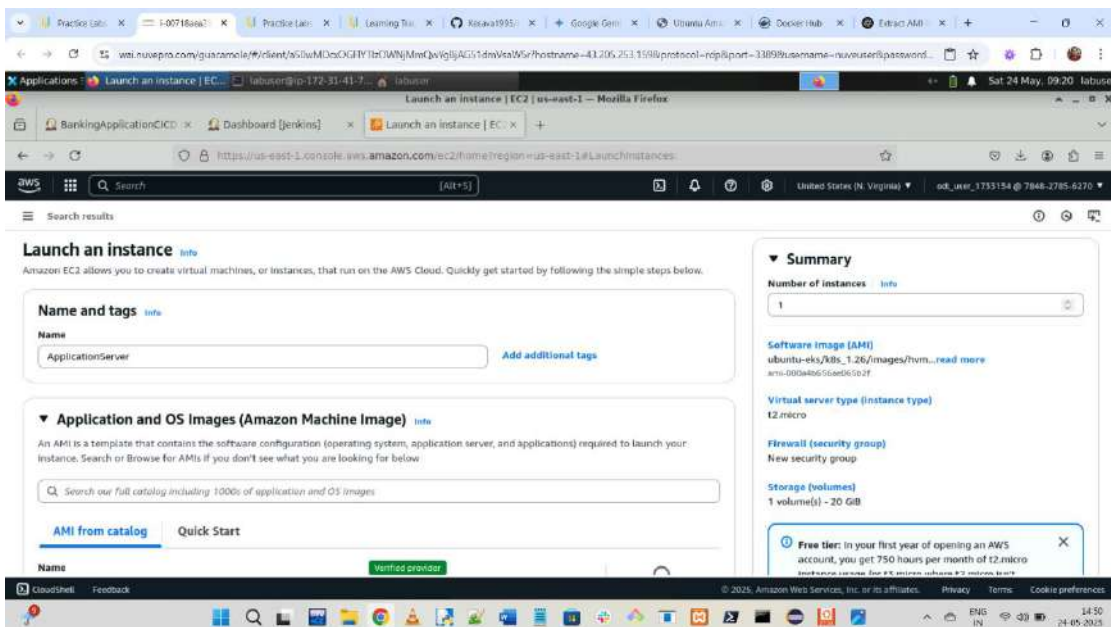
Step 4: Configure Ansible on Simplilearn lab

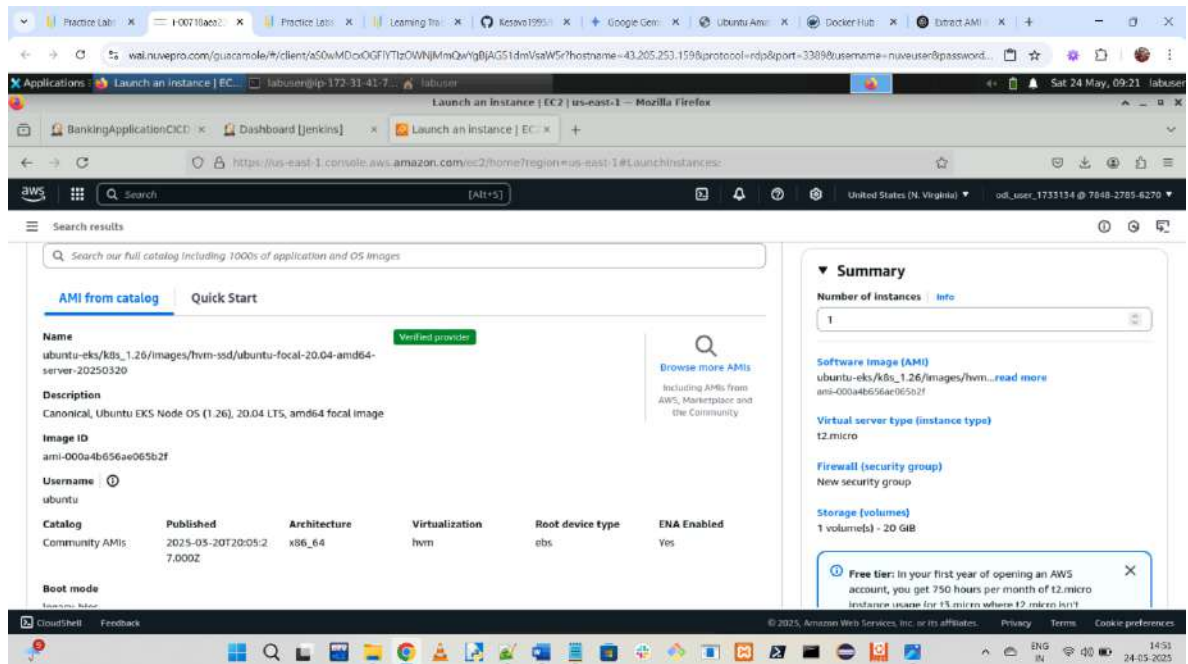
Note: we need to configure Ansible Controller on Simplilearn Lab from where it would connect to both AWS and Azure VM

- 4.1 Navigate to AWS Console and EC2 service to launch VM using below steps

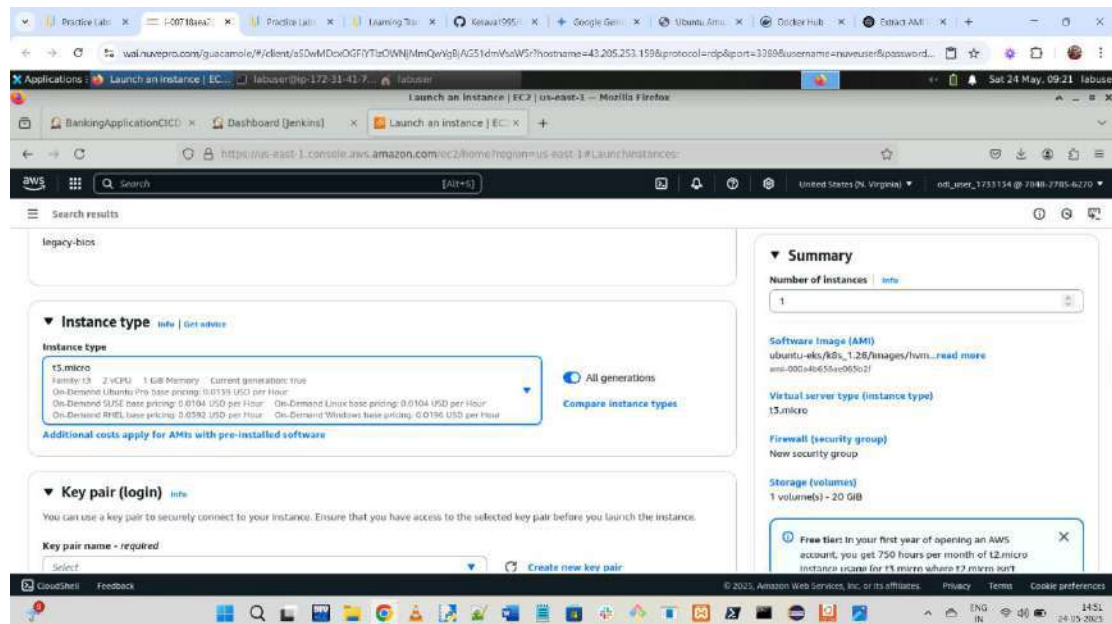


4.2 Provide name as **ApplicationServer** and Ubuntu 24.04 which will be used to launch VM



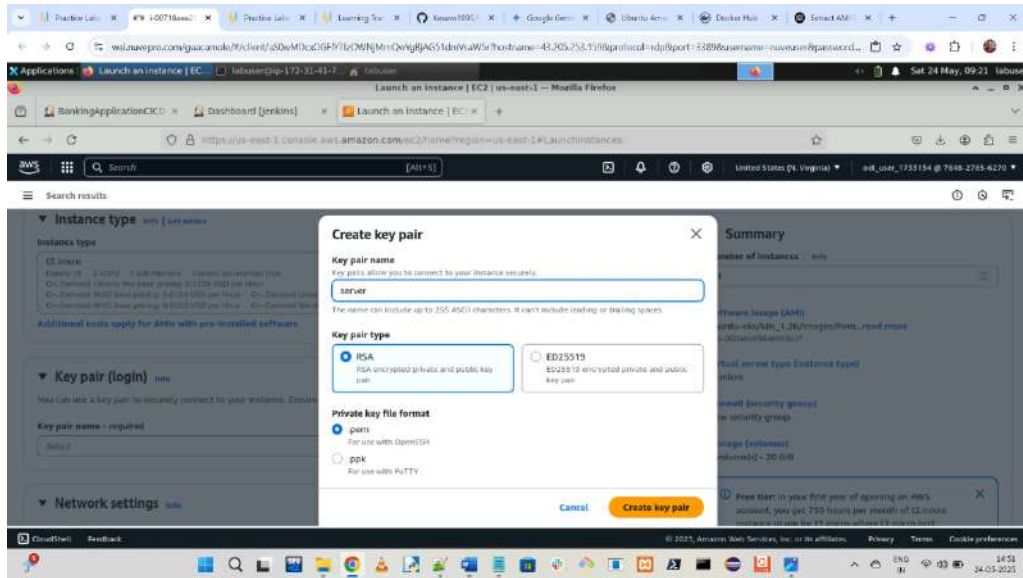


4.3 Select Instance type as **t3.micro**



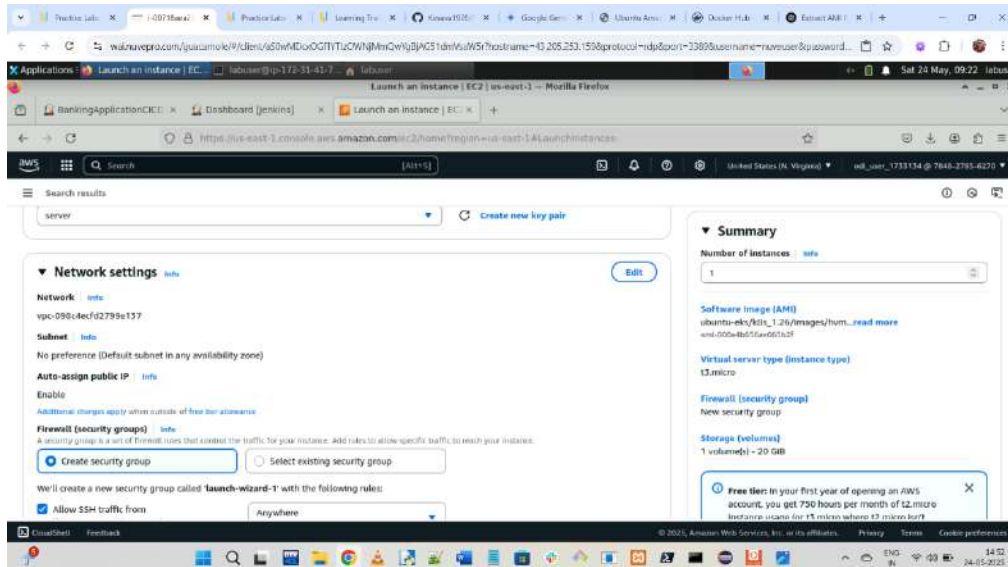
Next, we need to create key pair which will be used to perform SSH connectivity with server

4.4 Enter a key pair name as **server**, select the key pair type, choose the private key file format (.pem or .ppk), and click **Create key pair**. Save the private key securely for future instance connections.

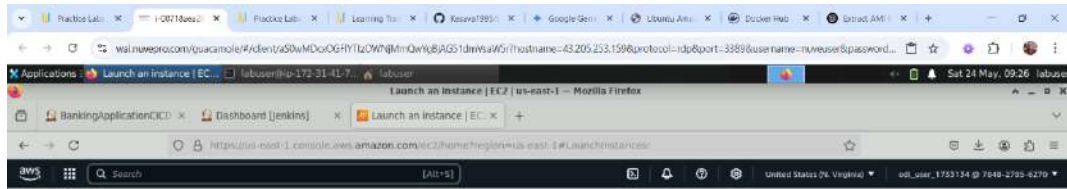


Note: This will download pem file locally, which we need to preserve since we would be using this key pair to establish connectivity from Ansible. Select this key pair once its created

4.5 Click on Edit under network settings and add Application port in security group to allow that port.



4.6 Create a security group named **sg_application**, set the description to **Application Security Group**, and add inbound rules for **SSH** on port 22 from Anywhere and Custom TCP on port **8989** from **0.0.0.0/0**. Finally, click **Launch instance** to start the instance with these settings.



Search results

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) Remove

Type info	Protocol info	Port range info
ssh	TCP	22
Source type info	Source info	Description - optional info
Anywhere	<input type="text" value="Add CIDR, prefix list or security group"/> 0.0.0.0/0	e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 8989) Remove

Type info	Protocol info	Port range info
Custom TCP	TCP	8989
Source type info	Source info	Description - optional info
Custom	<input type="text" value="Add CIDR, prefix list or security group"/> 0.0.0.0/0	e.g. SSH for admin desktop

Summary

Number of instances [info](#)
1

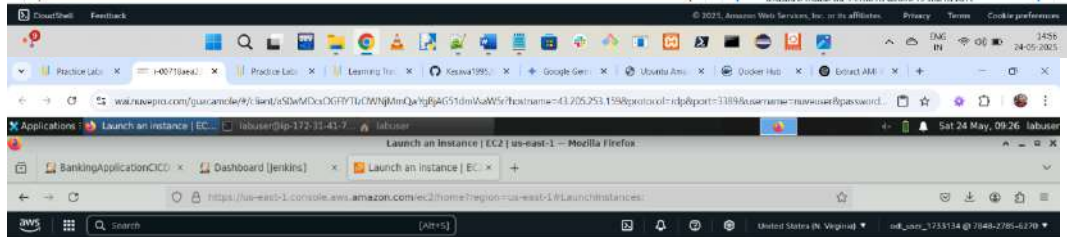
Software image (AMI)
ubuntu-eks/sbs_1.26/images/hvm...read more
ami-005b1653ae0650a2f

Virtual server type (instance type)
t3.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 20 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t3.micro instance usage for t3.micro where t3.micro has 1



Search results

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0) Remove

Type info	Protocol info	Port range info
ssh	TCP	22
Source type info	Source info	Description - optional info
Anywhere	<input type="text" value="Add CIDR, prefix list or security group"/> 0.0.0.0/0	e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 8989, 0.0.0.0/0) Remove

Type info	Protocol info	Port range info
Custom TCP	TCP	8989
Source type info	Source info	Description - optional info
Custom	<input type="text" value="Add CIDR, prefix list or security group"/> 0.0.0.0/0	e.g. SSH for admin desktop

Summary

Number of instances [info](#)
1

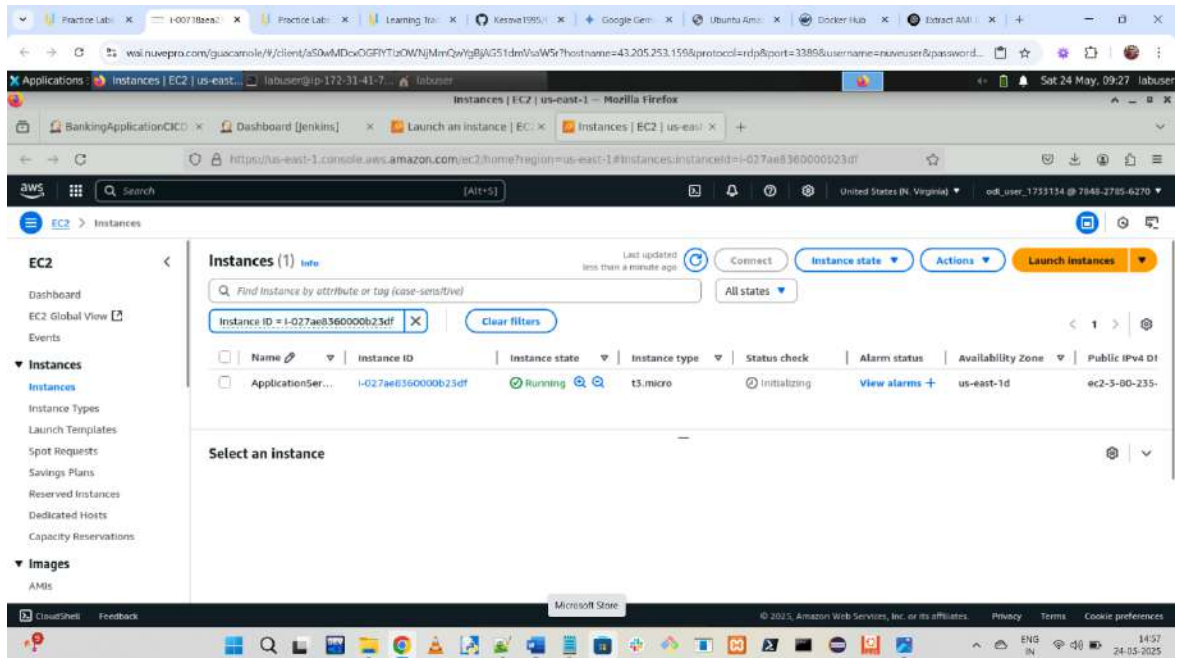
Software image (AMI)
ubuntu-eks/sbs_1.26/images/hvm...read more
ami-005b1653ae0650a2f

Virtual server type (instance type)
t3.micro

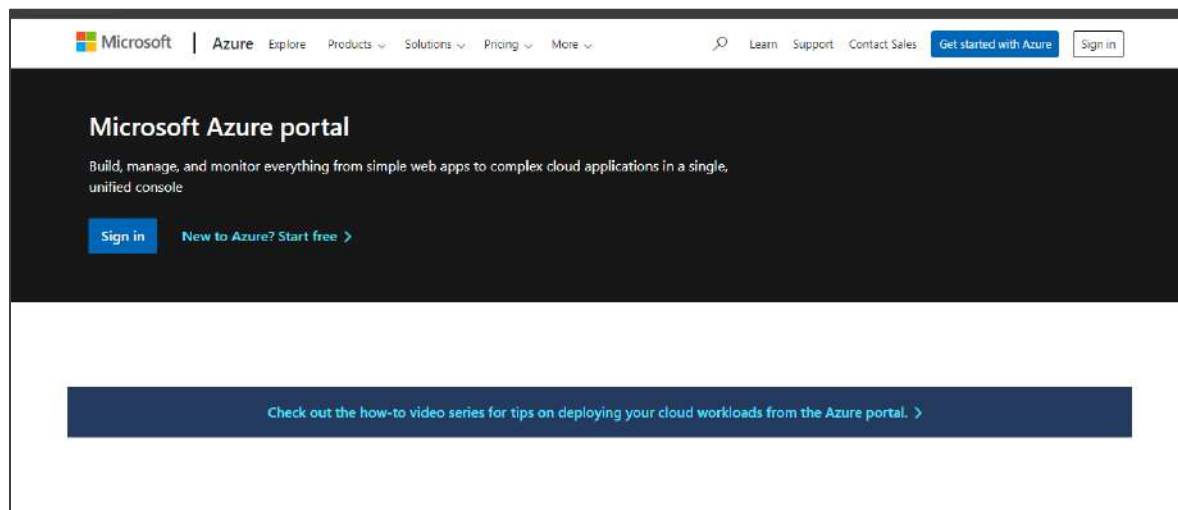
Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 20 GiB

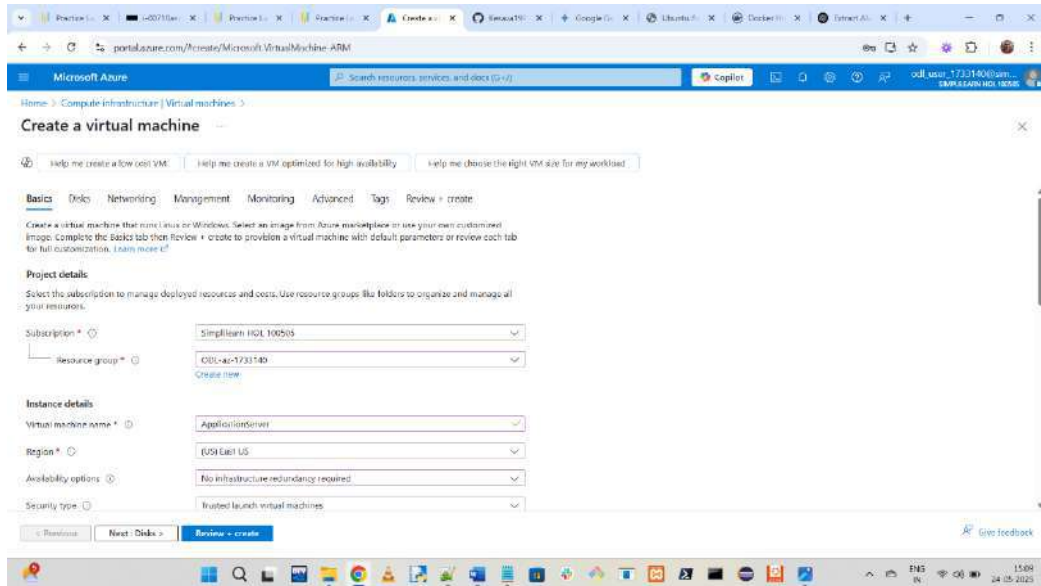
Free tier: In your first year of opening an AWS account, you get 750 hours per month of t3.micro instance usage for t3.micro where t3.micro has 1



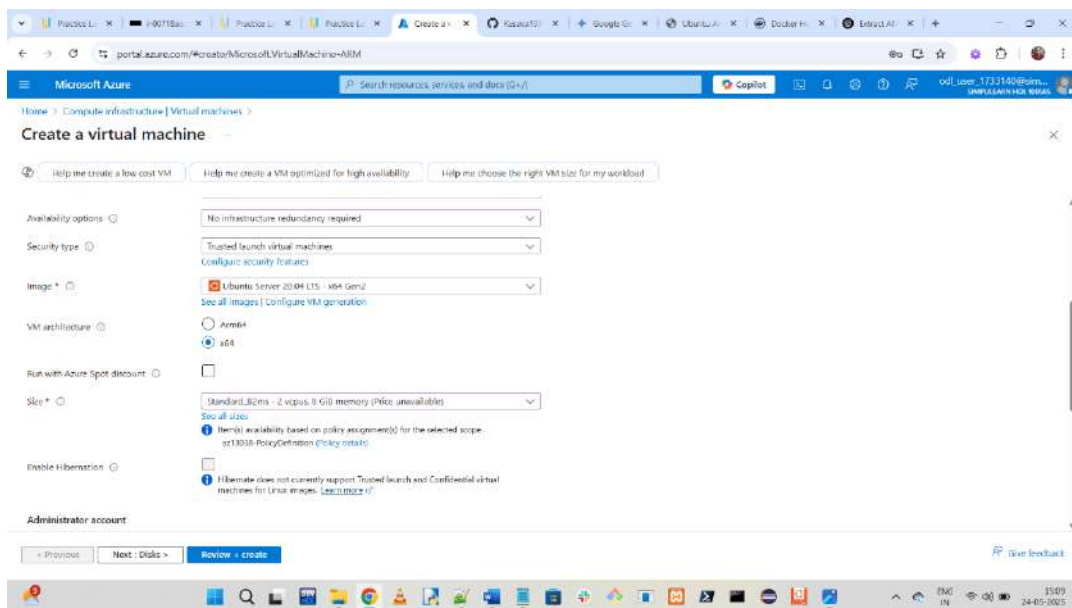
4.7 Navigate to Azure Console and launch a new VM which we will be connecting using Ansible Controller



4.8 Navigate to Virtual Machine service in Azure and proceed with launching new ubuntu 24.04 VM. Click on Create virtual machine and provide details as per below screenshot



4.9 Select Ubuntu 24.04 image, VM size



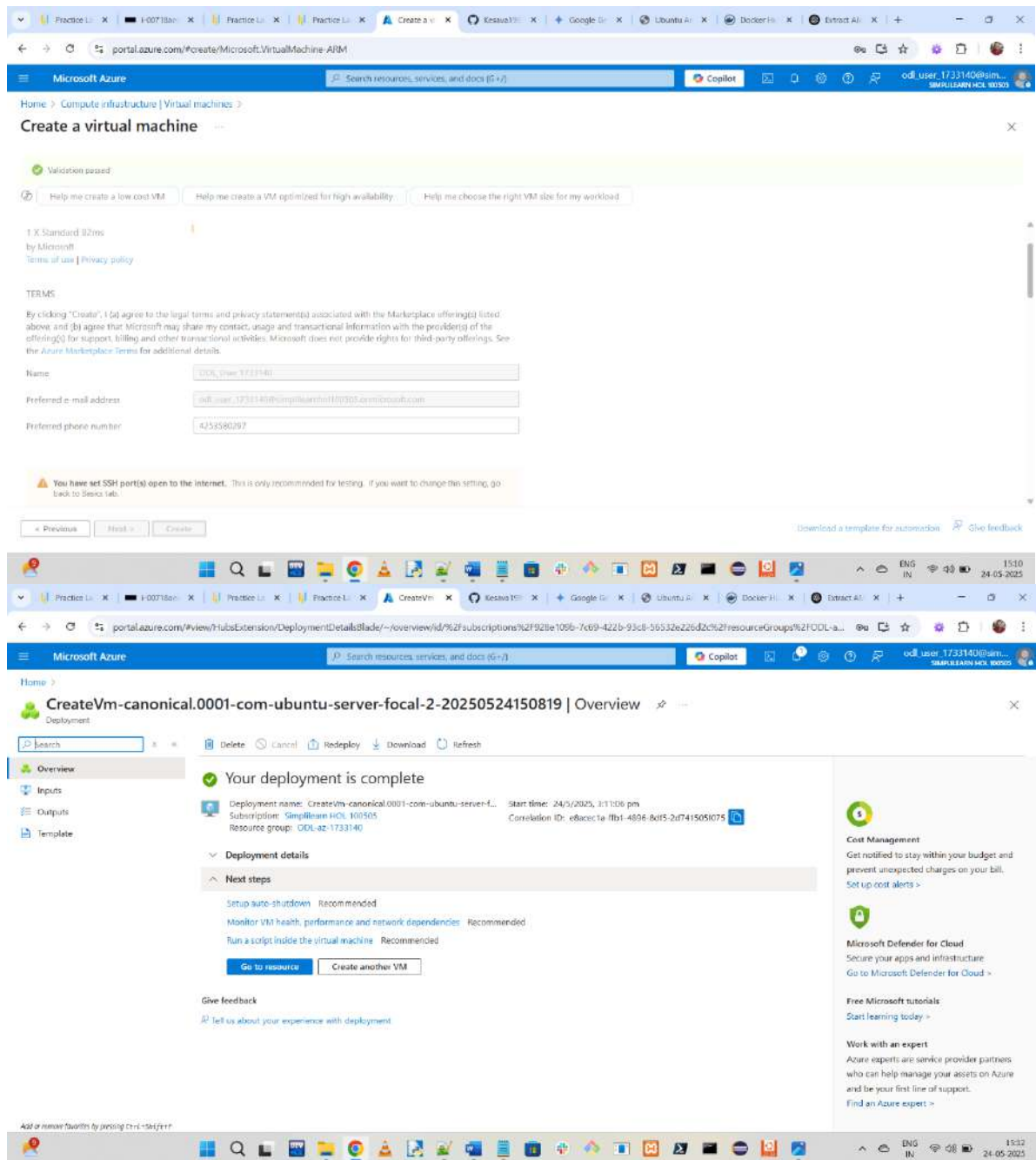
4.10 For authentication provide username and password which we will be using with Anisble to authenticate

Microsoft Azure portal - Create a virtual machine page. The page shows the configuration for a new VM. The authentication type is set to Password. The username is Az001. The password is masked with asterisks. The confirm password is also masked. The inbound port rules are set to Allow selected ports, and the selected inbound ports are SSH (22). A warning message indicates that this configuration will allow all IP addresses to access the VM, which is only recommended for testing. The page includes navigation buttons: Previous, Next: Disks, Review > create, and Give feedback.

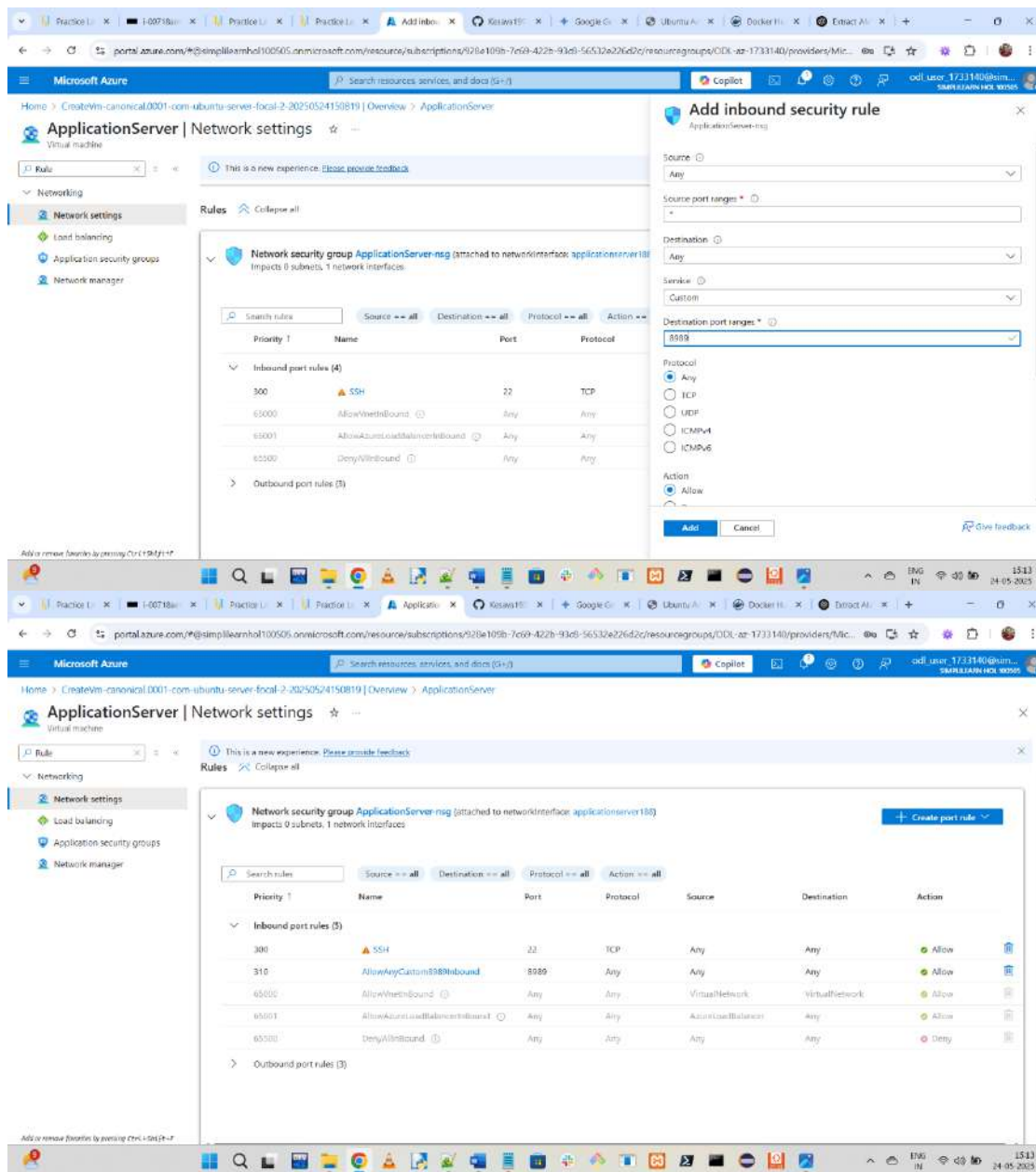
4.11 Click on Review and create to launch VM. Once VM is launched we must open **8989** ports from firewall rules to access application

Microsoft Azure portal - Create a virtual machine page. The page shows the configuration for a new VM. The authentication type is set to Password. The username is Az001. The password is masked with asterisks. The confirm password is also masked. The inbound port rules are set to Allow selected ports, and the selected inbound ports are SSH (22). A warning message indicates that this configuration will allow all IP addresses to access the VM, which is only recommended for testing. The page includes navigation buttons: Previous, Next: Disks, Review > create, and Give feedback.

4.12 Navigate to Virtual Machine service, select desired VM and modify networking configuration as per below screenshot



4.13 Create inbound port rule for 8989 as shown in the screenshot below:



Step 5: Validate Deployment

Note: Once both AWS and Azure VM is launched, proceed with configuring Ansible inventory file to perform connectivity between Ansible and VMs

5.1 Create **inventory.yaml** in GitHub repository with below content

[all:children]

aws

azure

[aws]

54.198.248.15

[aws:vars]

ansible_ssh_user=ubuntu

ansible_ssh_private_key_file="{{ key_pair_path }}"

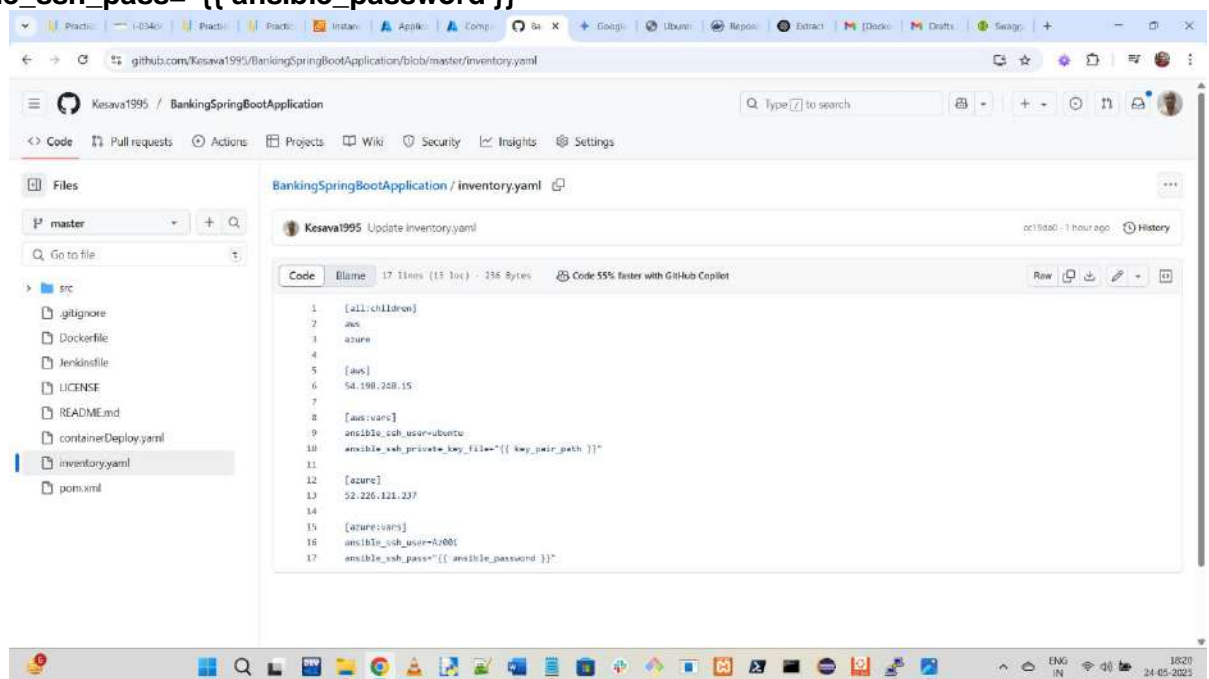
[azure]

52.226.121.237

[azure:vars]

ansible_ssh_user=Az001

ansible_ssh_pass="{{ ansible_password }}"

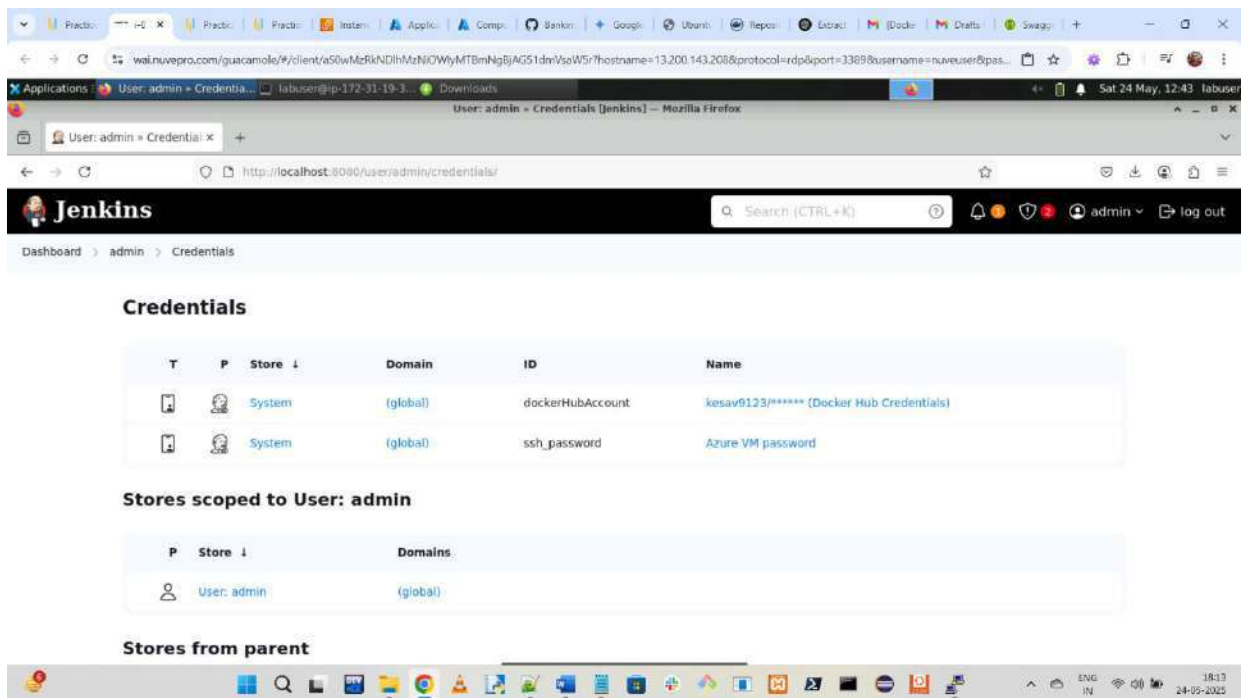


We need to create key pair on DevOps lab which was downloaded while launching EC2 instance

- 5.2 Navigate to **/var/lib/jenkins**, open the private key file **server.pem** using **vi**, verify its permissions with **ls -lart**, update the permissions to read-only using **chmod 400 server.pem**, and confirm the change with **ls -lart**, ensuring the file is secure with read-only access for the Jenkins user

```
labuser@ip-172-31-19-37: /var/lib/jenkins$
API version: 1.46 (minimum version 1.24)
Go version: go1.21.12
Git commit: cc13f95
Built: Tue Jul 23 19:59:47 2024
OS/Arch: linux/arm64
Experimental: false
containerd:
  Version: 1.7.19
  GitCommit: 2bf793ef6dc9a18e00cb12efb64355c2c9d5eb41
runc:
  Version: 1.7.19
  GitCommit: v1.1.13-0-g58aa920
docker-init:
  Version: 0.19.0
  GitCommit: de40ad0
(base) labuser@ip-172-31-19-37:~$ ansible --version
ansible [core 2.16.9]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/labuser/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/labuser/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Jul 29 2024, 16:56:48) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
(base) labuser@ip-172-31-19-37:~$ sudo cp /home/labuser/Downloads/server.pem /var/lib/jenkins
(base) labuser@ip-172-31-19-37:~$ cd /var/lib/jenkins
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$ ls server.pem
server.pem
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$ sudo chmod 400 /var/lib/jenkins/server.pem
sudo chown jenkins:jenkins /var/lib/jenkins/server.pem
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$ sudo usermod -aG docker Jenkins
usermod: user 'Jenkins' does not exist
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$ sudo usermod -aG docker jenkins
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$ sudo systemctl restart jenkins
(base) labuser@ip-172-31-19-37:/var/lib/jenkins$
```

We need to create a Jenkins credential with Azure password for VM connectivity



5.3 Now, we need to create Ansible Playbook file as per below content

- name : Deploy Pods on Docker Container
- hosts : all

tasks :

- name: Apt Update

apt:

update_cache: yes

- name: Install Docker

apt: name=docker.io state=latest

- name: Stop existing container if running

command: "docker stop {{ containerName }}"

ignore_errors: true

- name: Remove Existing Container

command: "docker rm {{ containerName }}"

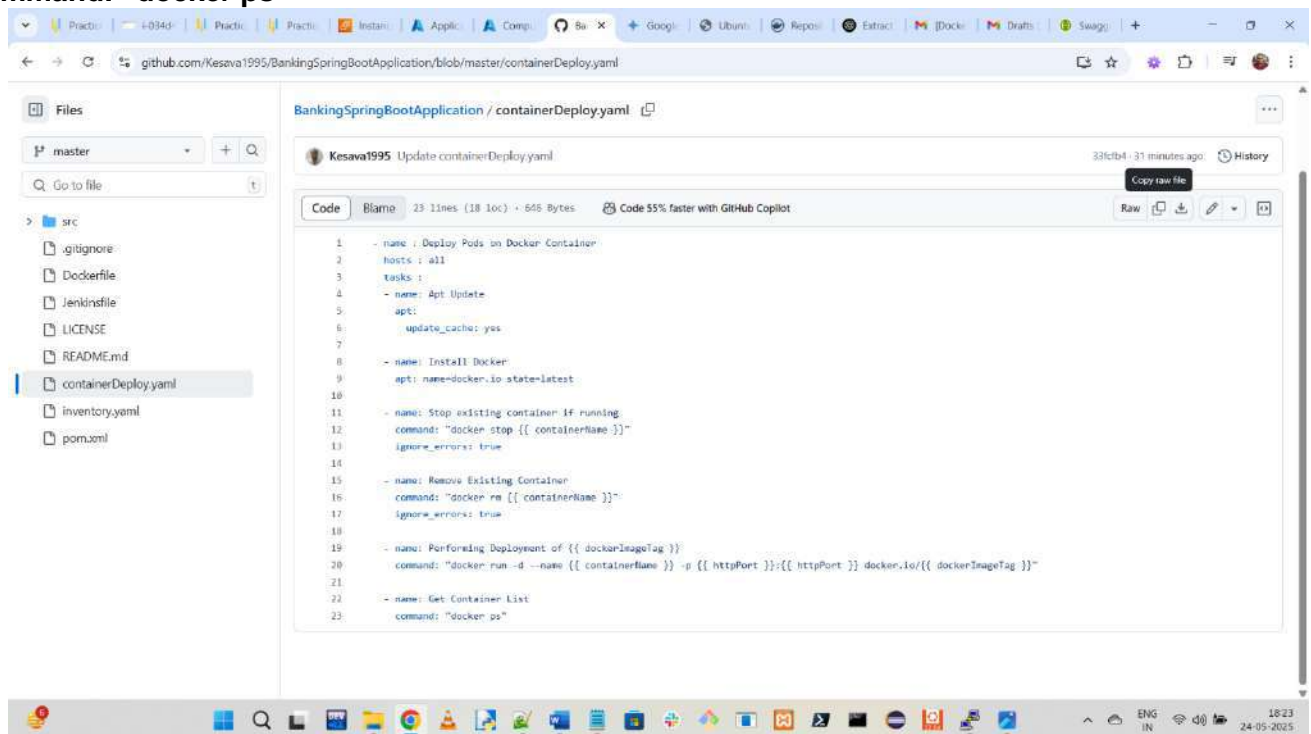
ignore_errors: true

- name: Performing Deployment of {{ dockerImageTag }}

command: "docker run -d --name {{ containerName }} -p {{ httpPort }}:{{ httpPort }} docker.io/{{ dockerImageTag }}"

- name: Get Container List

command: "docker ps"



5.4 Navigate to Jenkins job created and click on Build now to start running build for Jenkins job created

The screenshot displays the Jenkins web interface for a build named 'BankingApplicationCICD #8'. The 'Console Output' tab is selected, showing a log of file copying operations. The build is marked as successful with a green checkmark. Below the console output, a 'PLAY RECAP' section shows the build status for two IP addresses: 52.226.121.237 and 54.198.248.15. Both show 'ok=7' and 'changed=5', with all other status metrics (unreachable, failed, skipped, rescued, ignored) set to 0. The build concludes with 'Finished: SUCCESS'. The bottom of the image shows the Windows taskbar and system tray, indicating the time is 17:59 on 24-05-2025.

Dashboard > BankingApplicationCICD > #8

Status
Changes
Console Output
View as plain text
Edit Build Information
Delete build '#8'
Timings
Git Build Data
Pipeline Overview
Pipeline Console

Console Output

Skipping 139 KB.. [Full Log](#)

```
[[1;36mDEBUG[m] file AddCustomer_1001.json has a filtered file extension
[[1;36mDEBUG[m] copy /var/lib/jenkins/workspace/BankingApplicationCICD/src/test/resources/AddCustomer_1001.json to /var/lib/jenkins/
workspace/BankingApplicationCICD/target/test-classes/AddCustomer_1001.json
[[1;36mDEBUG[m] Copying file addAccountInformation5001_Customer1000.json
[[1;36mDEBUG[m] file addAccountInformation5001_Customer1000.json has a filtered file extension
[[1;36mDEBUG[m] copy /var/lib/jenkins/workspace/BankingApplicationCICD/src/test/resources/addAccountInformation5001_Customer1000.json to /
var/lib/jenkins/workspace/BankingApplicationCICD/target/test-classes/addAccountInformation5001_Customer1000.json
[[1;36mDEBUG[m] Copying file addAccountInformation5000_Customer1000.json
[[1;36mDEBUG[m] file addAccountInformation5000_Customer1000.json has a filtered file extension
[[1;36mDEBUG[m] copy /var/lib/jenkins/workspace/BankingApplicationCICD/src/test/resources/addAccountInformation5000_Customer1000.json to /
var/lib/jenkins/workspace/BankingApplicationCICD/target/test-classes/addAccountInformation5000_Customer1000.json
[[1;36mDEBUG[m] Copying file addAccountInformation3001_Customer1001.json
[[1;36mDEBUG[m] file addAccountInformation3001_Customer1001.json has a filtered file extension
[[1;36mDEBUG[m] copy /var/lib/jenkins/workspace/BankingApplicationCICD/src/test/resources/addAccountInformation3001_Customer1001.json to /
var/lib/jenkins/workspace/BankingApplicationCICD/target/test-classes/addAccountInformation3001_Customer1001.json
[[1;36mDEBUG[m] Copying file AddCustomer_1000.json
[[1;36mDEBUG[m] file AddCustomer_1000.json has a filtered file extension
```

changed: [52.226.121.237]
changed: [54.198.248.15]

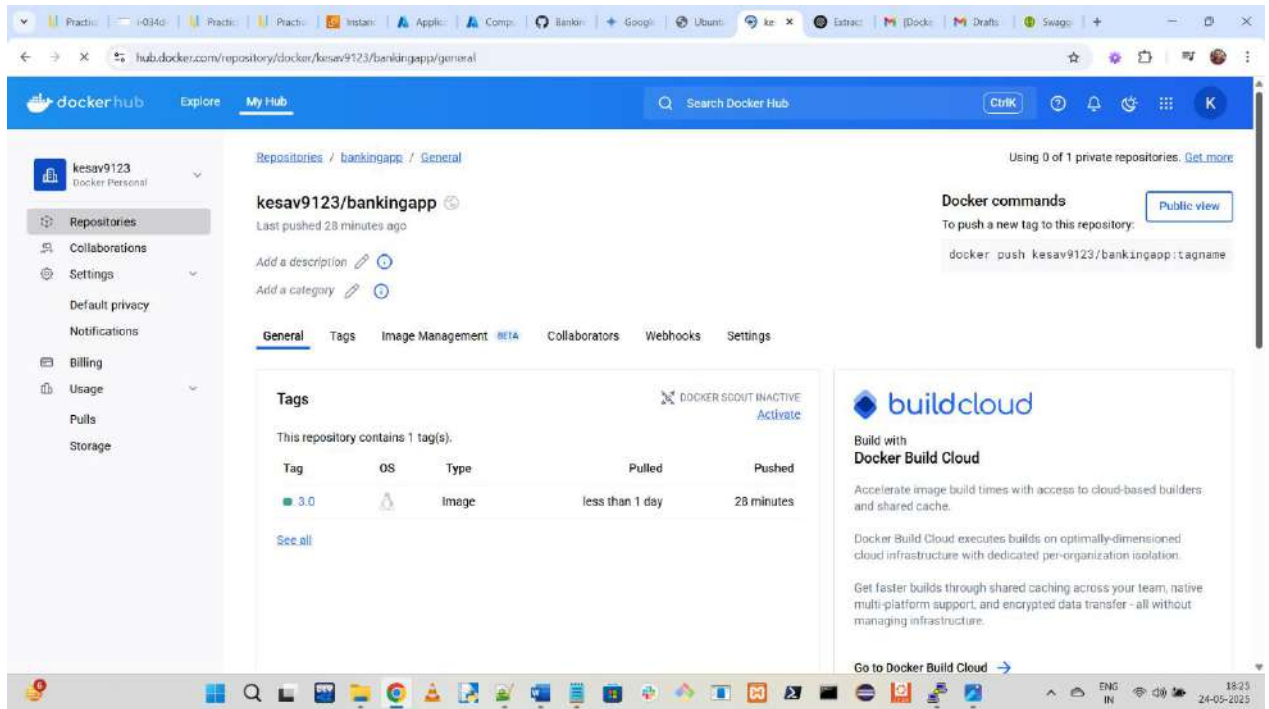
TASK [Get Container List] *****
changed: [52.226.121.237]
changed: [54.198.248.15]

PLAY RECAP *****
52.226.121.237 : ok=7 changed=5 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
54.198.248.15 : ok=7 changed=5 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

REST API Jenkins 2.452.3

5.5 Navigate to Docker hub to validate if Docker image gets published or not



Note: First time build may take more time than expected since lot of maven dependencies will be downloaded and cache locally first time. Subsequent executions will not take much time later.

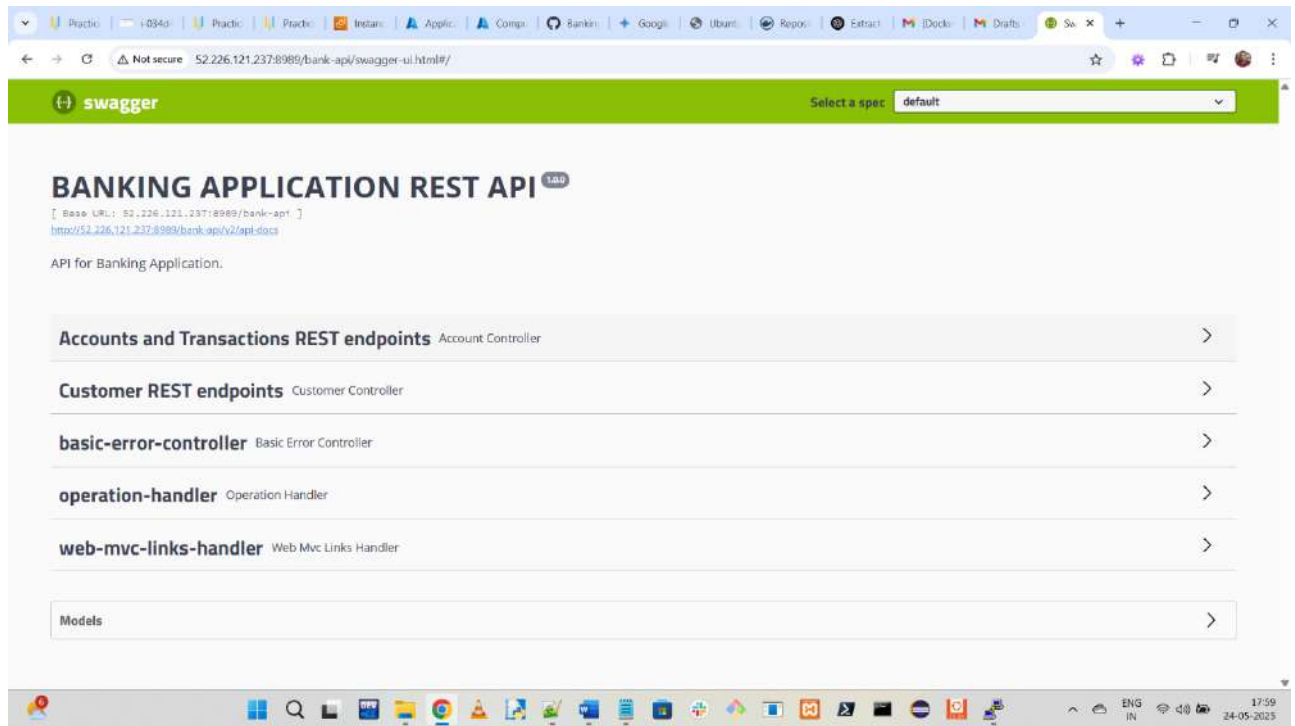
5.6 Once Application container is deployed on Docker host you can manually validate if containers are really running or not, connect to AWS VM and running below command to validate

```
Az001@ApplicationServer: ~
Az001@ApplicationServer:~$ sudo docker rmi ec2edbaaa897
Untagged: kesav9123/bankingapp@sha256:58c0d3ca414bfcae0ffbbda5adbc2d377fee31562
fe3d8d9c9a303433063f1ac
Deleted: sha256:ec2edbaaa897ea4bc4c6d5adb04afee14a734f27748e9c43139c3c7f003a05a5
Deleted: sha256:36a4d80c9e9ac53e57a11a28d9f8cc9cdffbaad13cc38caa72ef6892993ff16d8
Deleted: sha256:43ed8227c64e3230f6e1168c6f1194789760d2378a2c3839e-b5a57fde1184
Deleted: sha256:39cb397c4b795a7988592fc3c0c7f59afae8864a8c42184c8a17524e8de12
Deleted: sha256:29f3916944cb6055cd2fde8c07924676d22b1bd12bbea87f57c4e5a70de264
Deleted: sha256:f89f632b42a8028aa1b630ea4992d811c05bfbcb60456f6360f236ade1c05e2a
Deleted: sha256:61af91db1df2ca75c42d06da1b3b59f78508c9352e8d914bf783c4db1d48051
Deleted: sha256:63712c7cd94a320b271a1394604ba959a29753228946a02fcb3e600f56a0b93
Deleted: sha256:b024f5ec45e5cd979b20ca4efdc32385afca92f5a64faef6769c6a56543150a
Az001@ApplicationServer:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED
STATUS        PORTS      NAMES
db317c2cb61b  kesav9123/bankingapp:3.0   "java -jar /app.jar"    3 minutes ago
Up 3 minutes   0.0.0.0:8989->8989/tcp, :::8989->8989/tcp   bankingapp
Az001@ApplicationServer:~$ sudo docker rmi db317c2cb61b
Error response from daemon: No such image: db317c2cb61b:latest
Az001@ApplicationServer:~$ ^C
Az001@ApplicationServer:~$ sudo docker stop bankingapp
bankingapp
Az001@ApplicationServer:~$ sudo docker rm bankingapp
bankingapp
Az001@ApplicationServer:~$ sudo docker images
REPOSITORY    TAG      IMAGE ID      CREATED      SIZE
kesav9123/bankingapp   3.0      fa5979100739   6 minutes ago   702MB
Az001@ApplicationServer:~$ sudo docker rmi fa5979100739
Untagged: kesav9123/bankingapp:3.0
Untagged: kesav9123/bankingapp@sha256:e7ff1001fe356ae1bf773fa5ee7ba3853c49fa3691e95ff1957730490b85c4ee
Deleted: sha256:fa5979100739cb0a6e394ccaaed993b6d4991cae00ecdd474d14c9a5f00a1344
Deleted: sha256:be4dc8a7406d8bb1337c28561c61b0470bf0e394d8f7016a37e702021ae
Deleted: sha256:899b70952e3322d865da8c89f2f5b2b703cc2f8aa38a372b3ee402b061e42b78
Deleted: sha256:f751ef4f650736778778291061112642b84b3d01ac8003d7c13f8ad7d5aee3d
Deleted: sha256:7dd29f3ffa7289d8d9cc98f1507bb35fe14749d1f5a7c1a4b33ea62a113a1b
Deleted: sha256:1376fe23491c7bd8a29c2469f6489e5ef8b2311f78191e87c47acd67e846372
Deleted: sha256:935ab298b59cf495c8a62f40960786cc0de432f3e87f22371d57b67e05d0a
Deleted: sha256:6fa04b26fe15e4fb447dd372945f0ce70e3bdf6f6904081537ae18d10
Deleted: sha256:9c742cd6c7a5752ea38ba8ecb14be43c005e10e6d34f26a9aa1eb095c5d492
Az001@ApplicationServer:~$ sudo docker rmi kesav9123/bankingapp:3.0
Error response from daemon: No such image: kesav9123/bankingapp:3.0
Az001@ApplicationServer:~$ sudo docker rmi kesav9123/bankingapp
Error response from daemon: No such image: kesav9123/bankingapp:latest
Az001@ApplicationServer:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  STATUS        PORTS      NAMES
Az001@ApplicationServer:~$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED      STATUS        PORTS      NAMES
fa250710f69   kesav9123/bankingapp:3.0   "java -jar /app.jar"    28 minutes ago   Up 28 minutes   0.0.0.0:8989->8989/tcp, :::8989->8989/tcp   bankingapp
Az001@ApplicationServer:~$
```

5.7 Navigate to Azure VM and validate in case Docker container is deployed there or not, use system default browser to access application deployed on Docker Host. We will be getting Swagger Interface using which you can validate your Rest APIs.

<http://<SERVER-IP>:8989/bank-api/swagger-ui.html>

<http://52.226.121.237:8989/bank-api/swagger-ui.html>



By following the above steps, you have successfully implemented a multi-cloud deployment pipeline using Jenkins, Ansible, and Docker, enabling seamless integration and automation for the Java application across AWS and Azure environments. This setup ensures efficient CI/CD processes, robust containerized deployments, and enhanced application availability, validated through Swagger API testing.