PROJECT- III

Name: T. VISHNUVARDHAN

Batch no: 129

Phn no: 8520081327

Gmail: thappetlavishnuvardhan9@gmail.com

TOPIC: Automate the Process of Building and Pushing Docker Images into Elastic Container Registry using Jenkins Pipeline.

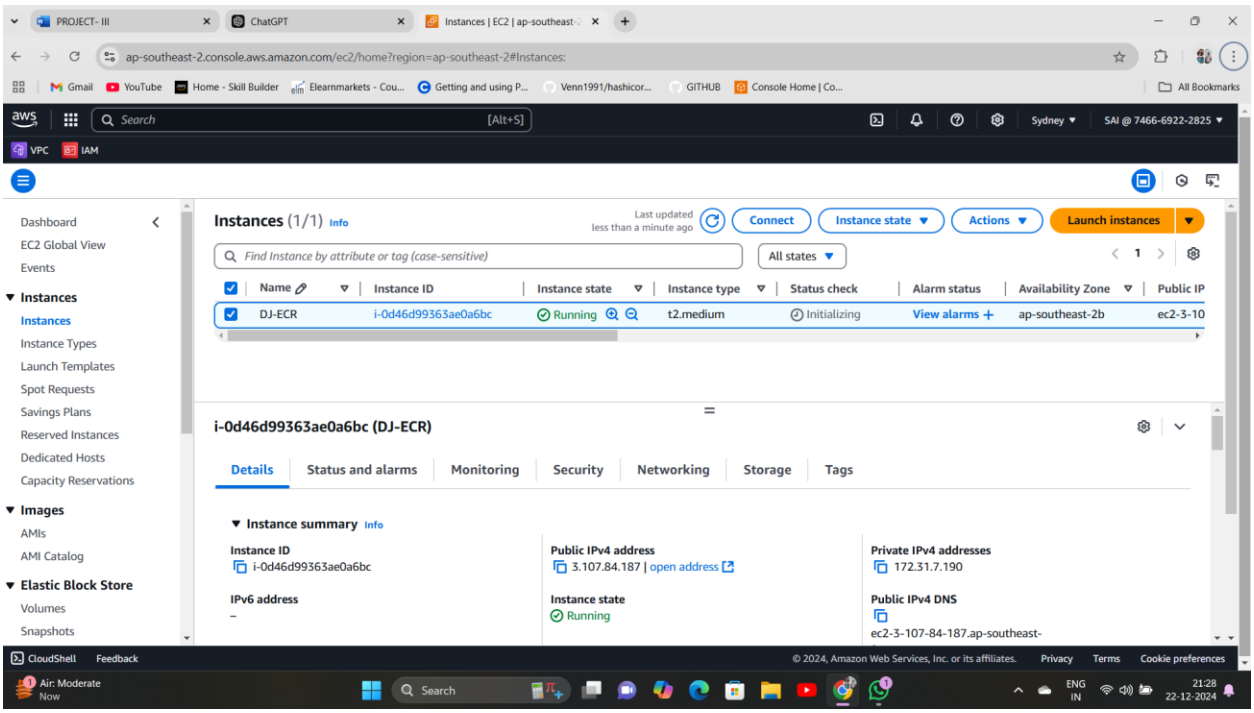
Step 1: Launch an Ubuntu Instance

Configurations: 1) Instance Type At least t2. Medium
2) Security Group Allow all Traffic
3) User data

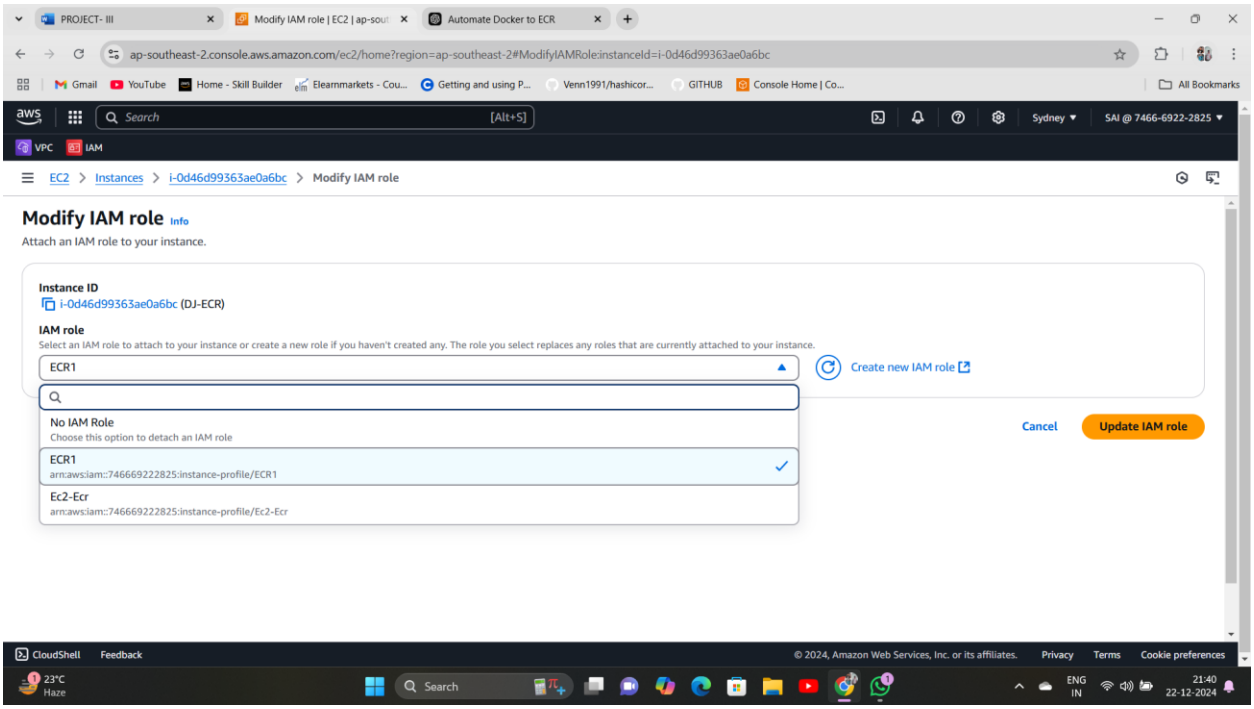
```
#! /bin/bash
apt update -y
apt install unzip -y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
apt install docker.io -y
apt install default-jdk -y
apt install maven -y
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Above Script is installing Unzip, AWS CLI, Java, Maven, Docker, Jenkins.

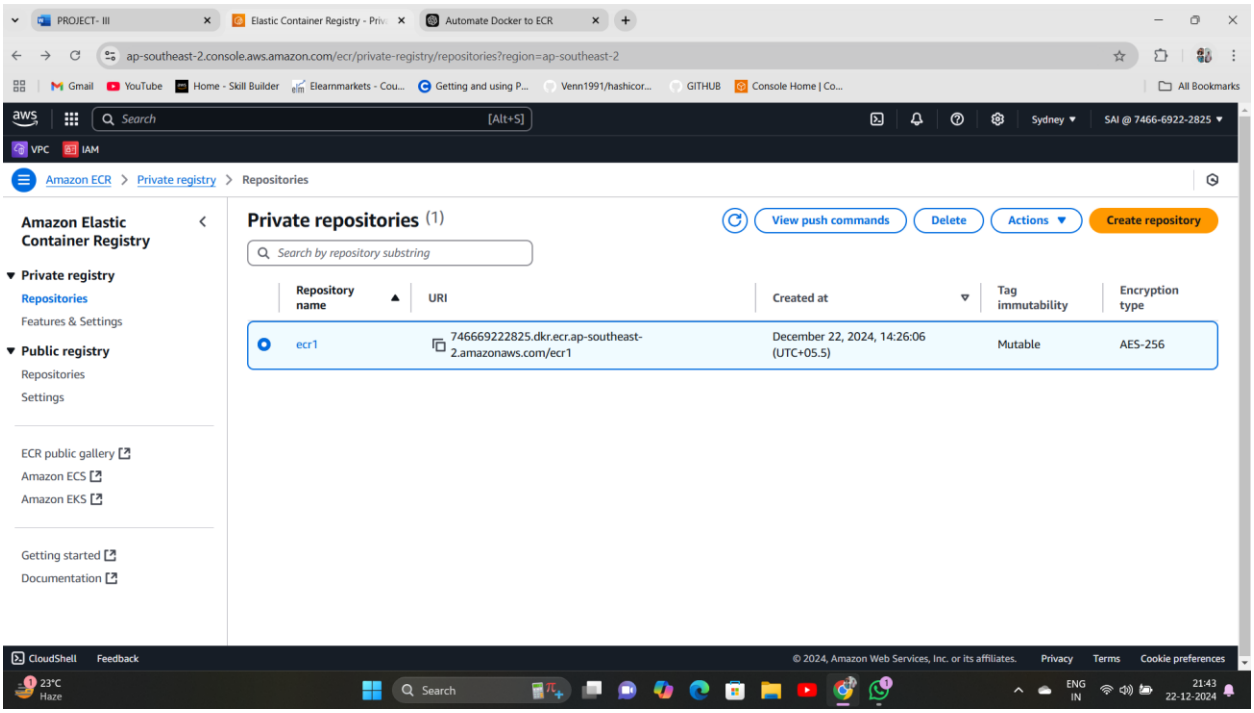
Configure AWS CLI with aws configure command, To enable AWS permissions enter Access Key and Secret Access Key of your AWS IAM User



Step 2: Create IAM Role with Attaching ECR policy [AmazonEC2ContainerRegistryFullAccess]. Attach the IAM role to Created Instance



Step 3: Create ECR Repository



Step 4: Create a Dockerfile and Index.html file in given Path /var/lib/Jenkins/workspace/JOB_NAME

Dockerfile:

```
FROM nginx:latest

LABEL maintainer="VISHNU" version="1.0.0"

RUN mkdir mhbd

COPY ./index.html /usr/share/nginx/html

WORKDIR /usr/share

EXPOSE 80
```

After Creating Dockerfile Use This Command To Add The Jenkins User To The Docker Group. This Allows The Jenkins User To Execute Docker Commands Without Requiring Sudo Privileges.

--> sudo usermod -aG docker jenkins

Step 5: Connect to Jenkins through Public IP Address with 8080 Port Number.

After Login into Jenkins Install Some PLUGINS

1) Pipeline Plugin
2) Docker
3) AWS Credentials
4) Elastic Container Registry

After Installing RESTART, the Jenkins Server

Step 6: Create AWS Credentials in Jenkins

1. Navigate to Manage Jenkins > Manage Credentials.
2. Select a credentials domain (e.g., `global`).
3. Click Add Credentials.
4. Choose AWS Credentials as the kind.
5. Enter the Access Key ID and Secret Access Key of your AWS IAM user.
6. Provide an ID (e.g., aws-ecr-credentials) and description.
7. Save the credentials.

Step 7: Configure ECR Login in Jenkins Pipeline

Update the Jenkins Pipeline to include steps for logging into ECR, building the Docker image, tagging it, and pushing it to ECR.

EXAMPLE PIPELINE:

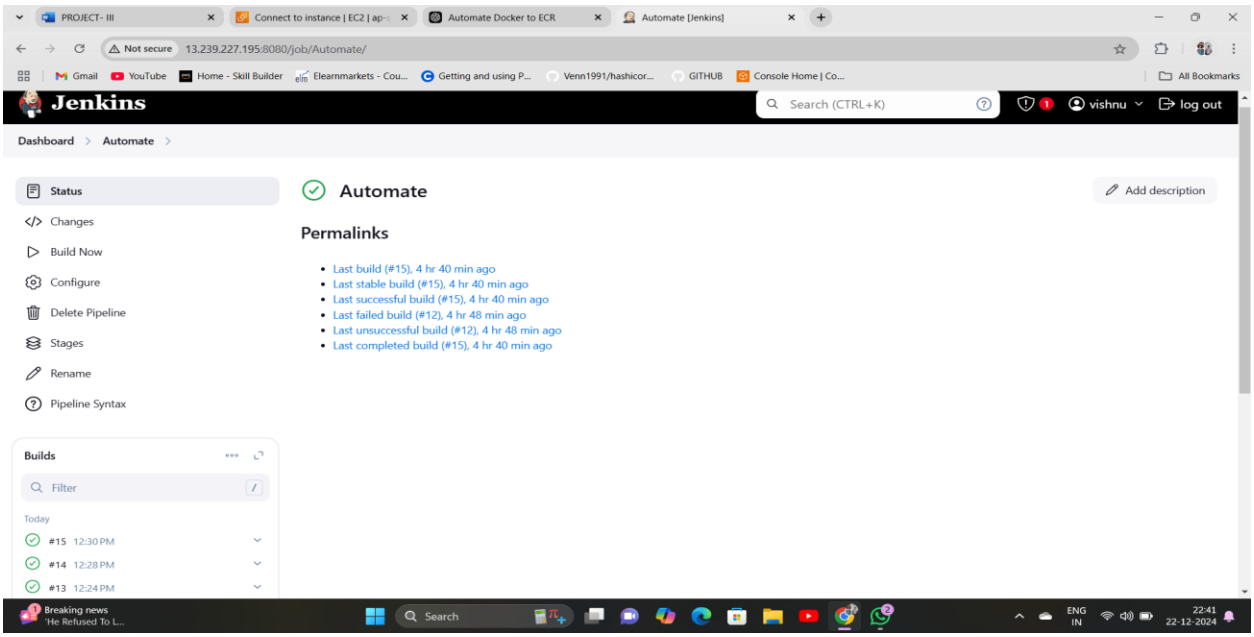
```
pipeline {

   agent any

   environment {

      AWS_ACCOUNT_ID = '746669222825 '          // Account ID

      AWS_REGION = 'ap-east-1'       // AWS Region

      ECR_REPOSITORY = 'ecr1'        // Repository Name
```

```
        IMAGE_TAG = "v1"      // Image Tag

        AWS_CREDENTIALS = credentials('vishnu') // Reference to Jenkins credentials ID

      REPOSITORY_URL = '746669222825.dkr.ecr.ap-east-1.amazonaws.com/ecr1' // ECR URL

    }

    stages {

        stage ('Login to AWS ECR') {

            steps {

                sh """

  aws configure set aws_access_key_id $AWS_CREDENTIALS_USR

  aws configure set aws_secret_access_key  $AWS_CREDENTIALS_PSW

  aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --password-stdin $REPOSITORY_URL """

            }

        }

        stage('Build Docker Image') {

            steps {

           sh """ docker build -t $ECR_REPOSITORY:$IMAGE_TAG . """
         }
}
 stage('Tag Docker Image')
 {
  steps {

                sh """ docker tag $ECR_REPOSITORY:$IMAGE_TAG $REPOSITORY_URL:$IMAGE_TAG """
```

```
            }
        }
        stage('Push Docker Image to ECR') {
            steps {
                sh """ docker push $REPOSITORY_URL:$IMAGE_TAG  """
            }
        }
    }
}
```

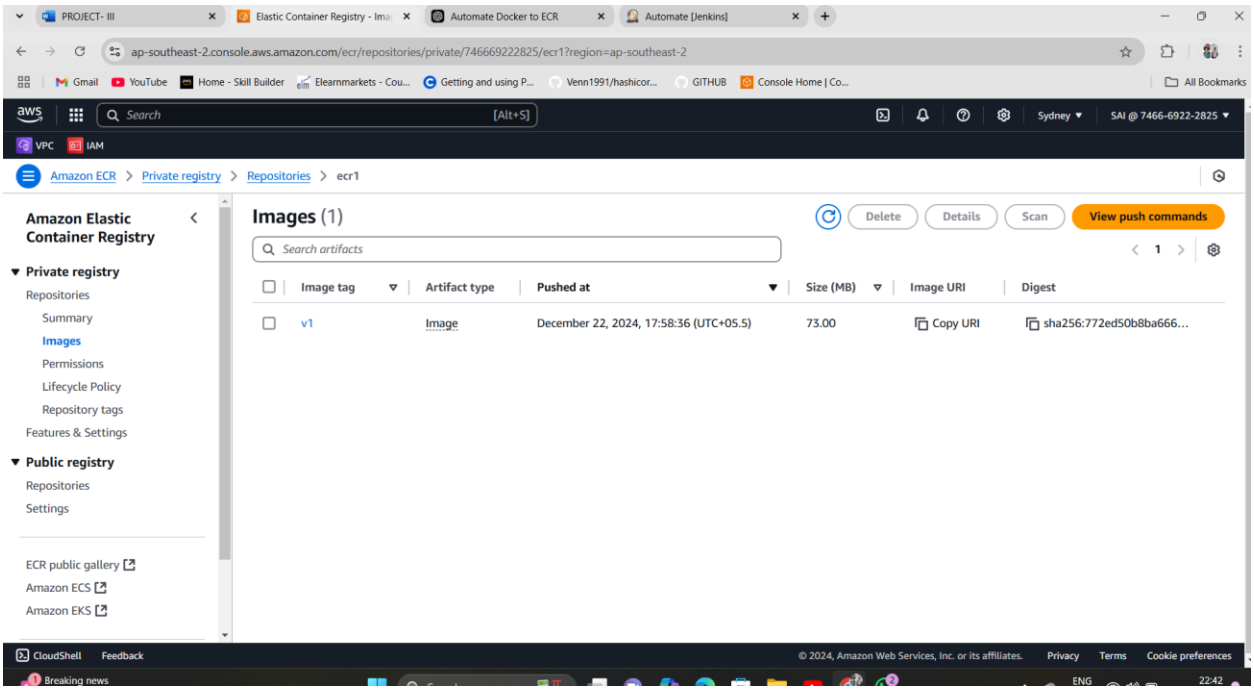
Step 8: Test the Jenkins Pipeline

1. Create a new Jenkins Pipeline job.
2. Configure the Pipeline to use the script provided above.
3. Replace placeholders like AWS_ACCOUNT_ID, AWS_REGION, IMAGE_TAG, AWS_CREDENTIALS_ID and ECR_REPO_NAME with actual values.
4. Run the Pipeline.



Step 9: Jenkins will build the Docker image.

The image will be tagged and pushed to your specified ECR repository.



CONCLUSION:

The Jenkins pipeline script automates the process of building, tagging, and pushing Docker images to Amazon ECR (Elastic Container Registry).

This pipeline provides a robust and secure foundation for integrating Docker image management into your CI/CD process. With this setup, each Jenkins job run will build and push a new Docker image to ECR, making it ready for deployment.

By integrating Jenkins with ECR, you can streamline your CI/CD workflows and ensure efficient deployment of your containerized applications.