Banking and Finance Domain Project

Project Overview

- **Project Name:** Finance Me - Banking and Finance Domain Project
- **Objective:** Automate CI/CD pipeline using Jenkins, Docker, Kubernetes, Terraform, and Ansible for a microservices-based banking application.
- **Cloud Provider:** AWS
- **GitHub Repository:** https://github.com/Vikram9890066071/BankingApp.git

---

📌 1. Problem Statement

FinanceMe is a **global banking and financial services provider** based in **Germany**. The company was facing challenges with its **monolithic application architecture** and **manual deployments**. They have decided to migrate to a **microservices-based architecture** and automate deployments using **DevOps methodologies**.

Challenges faced by FinanceMe:

✅ **Complex monolithic application architecture** was difficult to maintain.

✅ **Manual testing and deployments** caused delays.

✅ **Scaling issues** when traffic load increased.

✅ **Manual infrastructure provisioning** was time-consuming.

✅ **Monitoring the application manually** was difficult.

Solution:

👉 Implement **Microservices** using **Spring Boot & MySQL**.

👉 Use **DevOps tools** to enable **Continuous Integration & Continuous Deployment (CI/CD)**.

👉 Deploy the application on **AWS** using **Docker, Kubernetes, and Terraform**.

👉 Monitor application performance with **Prometheus & Grafana**.

📌 2. Technologies Used

Component	Technology
Version Control	Git, GitHub
CI/CD Pipeline	Jenkins
Containerization	Docker
Orchestration	Kubernetes
Infrastructure as Code	Terraform
Configuration Management	Ansible
Application Framework	Java, Spring Boot, Maven
Testing	JUnit, Selenium, TestNG

📌 3. Implementation Steps

🔹 Phase 1: Infrastructure Setup

1. **Provision AWS EC2 Instances using Terraform**

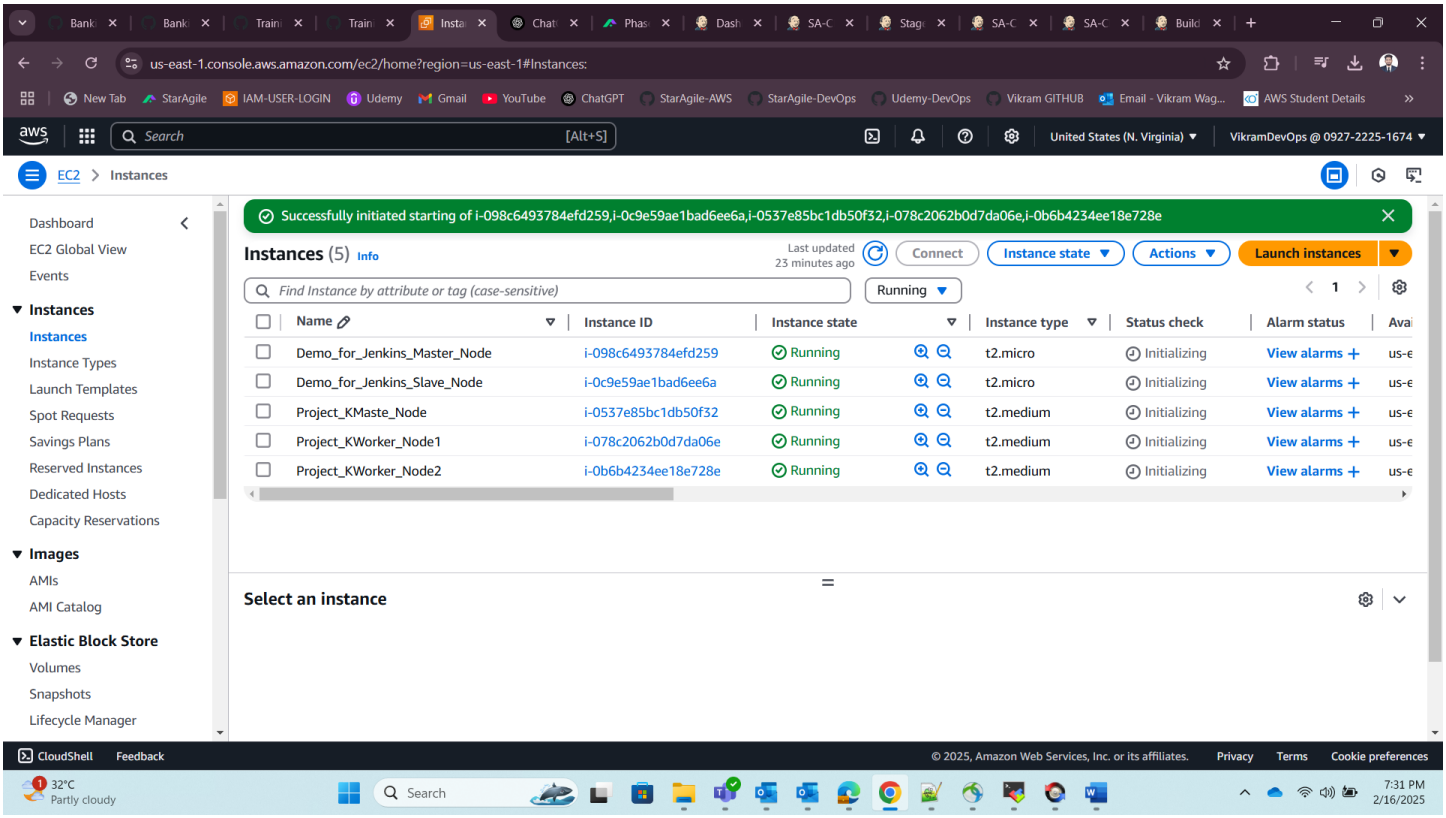
   terraform init

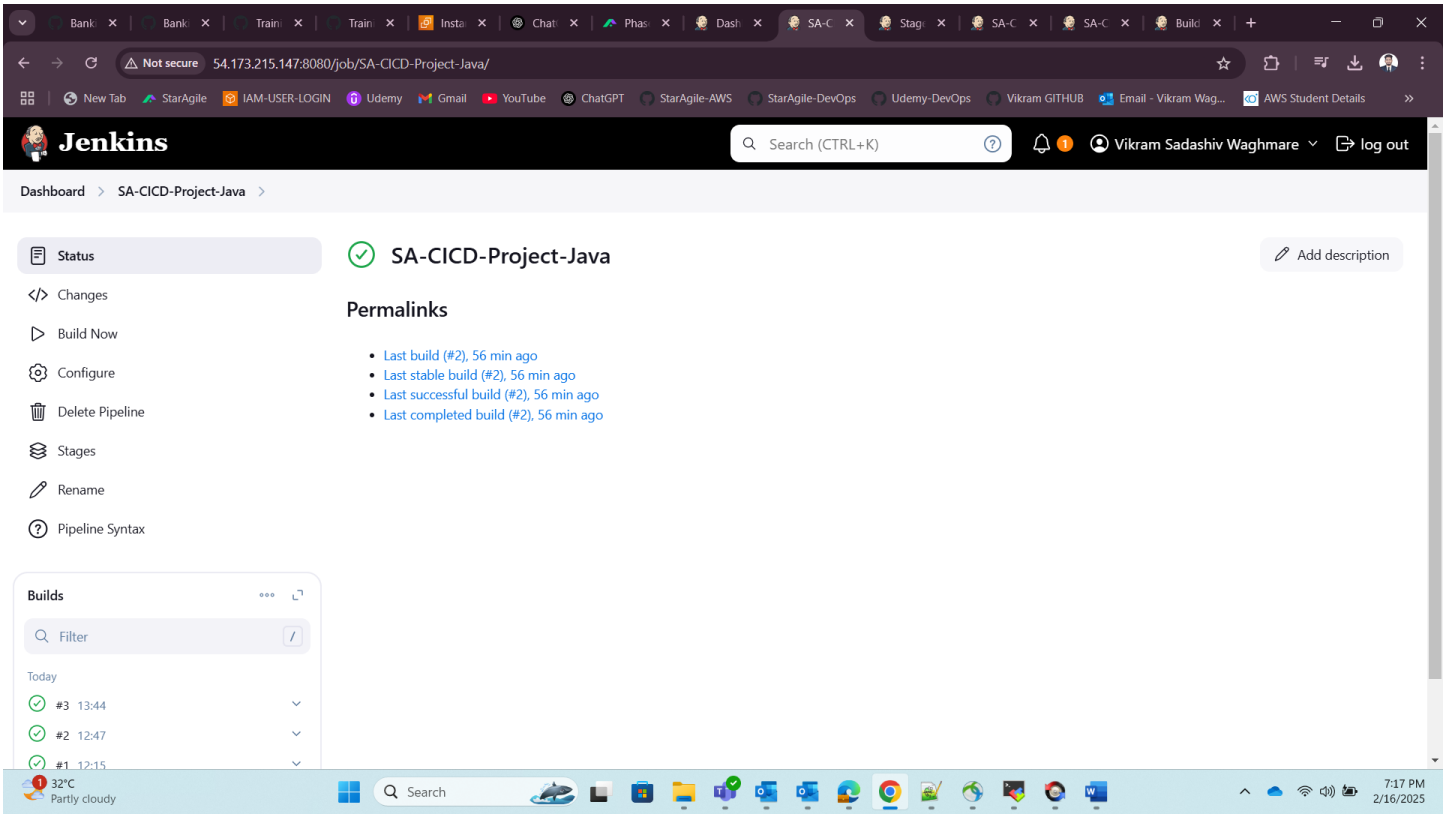
   terraform apply

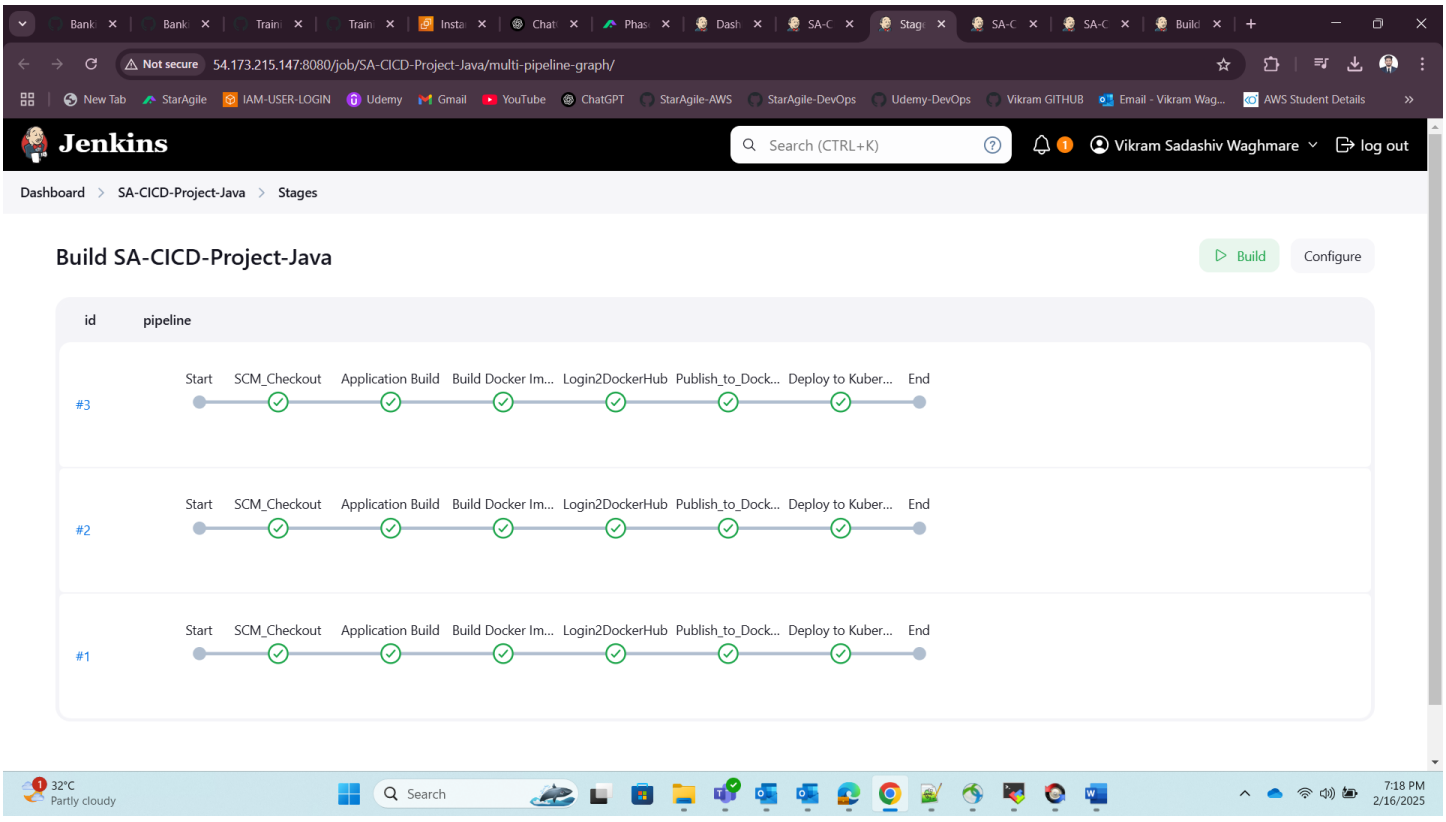
2. **Configure Security Groups**
   - Open ports: **22, 8080, 8082, 31030, 3000, 9090, 6443**
3. **Use Ansible for Configuration Management**

   ansible-playbook setup-environment.yml

📸 Refer to Screenshot




Not secure 54.173.215.147:8080/job/SA-CICD-Project-Java/
Jenkins
Search (CTRL+K)
Vikram Sadashiv Waghmare
log out
Dashboard > SA-CICD-Project-Java >
Status
Changes
Build Now
Configure
Delete Pipeline
Stages
Rename
Pipeline Syntax
SA-CICD-Project-Java
Add description
Permalinks
Last build (#2), 56 min ago
Last stable build (#2), 56 min ago
Last successful build (#2), 56 min ago
Last completed build (#2), 56 min ago
Builds
Filter
Today
#3 13:44
#2 12:47
#1 12:15
32°C
Partly cloudy
Search
7:17 PM
2/16/2025


Not secure 54.173.215.147:8080/job/SA-CICD-Project-Java/multi-pipeline-graph/
Jenkins
Search (CTRL+K)
Vikram Sadashiv Waghmare
log out
Dashboard > SA-CICD-Project-Java > Stages
Build SA-CICD-Project-Java
Build
Configure
id
pipeline
Start SCM_Checkout Application Build Build Docker Im... Login2DockerHub Publish_to_Dock... Deploy to Kuber... End
#3
#2
#1
32°C
Partly cloudy
Search
7:18 PM
2/16/2025

Phase 2: Banking Microservice Development

1. **Clone the Repository**

   git clone https://github.com/Vikram9890066071/BankingApp.git

   cd BankingApp

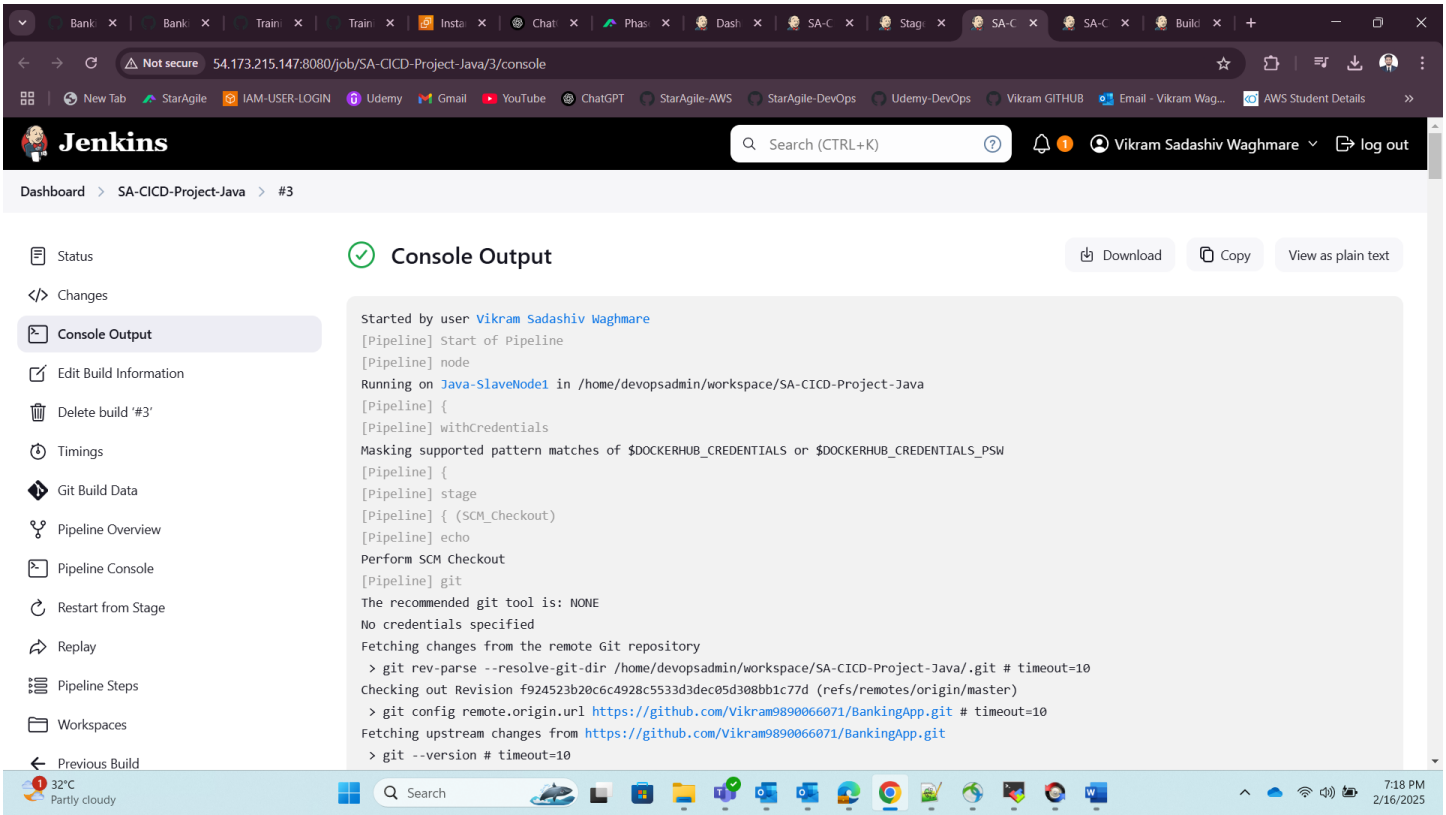
2. **Build the Application**

   mvn clean package

3. **Run Unit Tests**

   mvn test

Refer to Screenshot



Microservices Endpoints:

API Endpoint	HTTP Method	Description
/createAccount	**POST**	Creates a new bank account
/updateAccount/{account_no}	**PUT**	Updates account details
/viewPolicy/{account_no}	**GET**	Fetches account policy
/deletePolicy/{account_no}	**DELETE**	Deletes account policy

- **Phase 3: Dockerization**

1. **Create a Dockerfile**

```
FROM openjdk:11
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
EXPOSE 8082
```

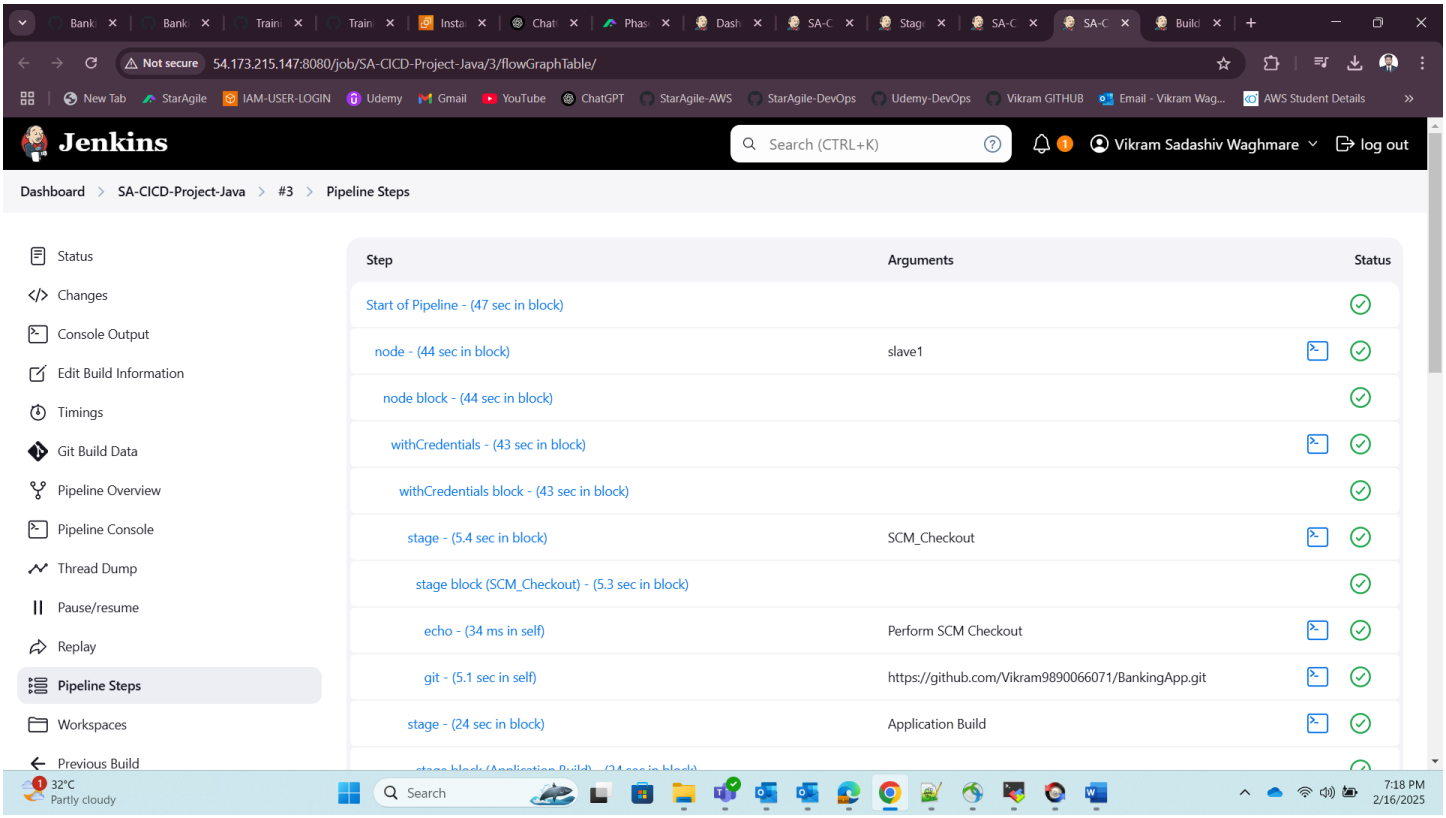
2. **Build Docker Image**

```
docker build -t vsw210/bankapp-eta-app:latest .
```

3. **Push Image to Docker Hub**

```
docker login -u vsw210
docker push vsw210/bankapp-eta-app:latest
```

Refer to Screenshot



🔹 Phase 4: Kubernetes Deployment

1. Apply Kubernetes Deployment & Service

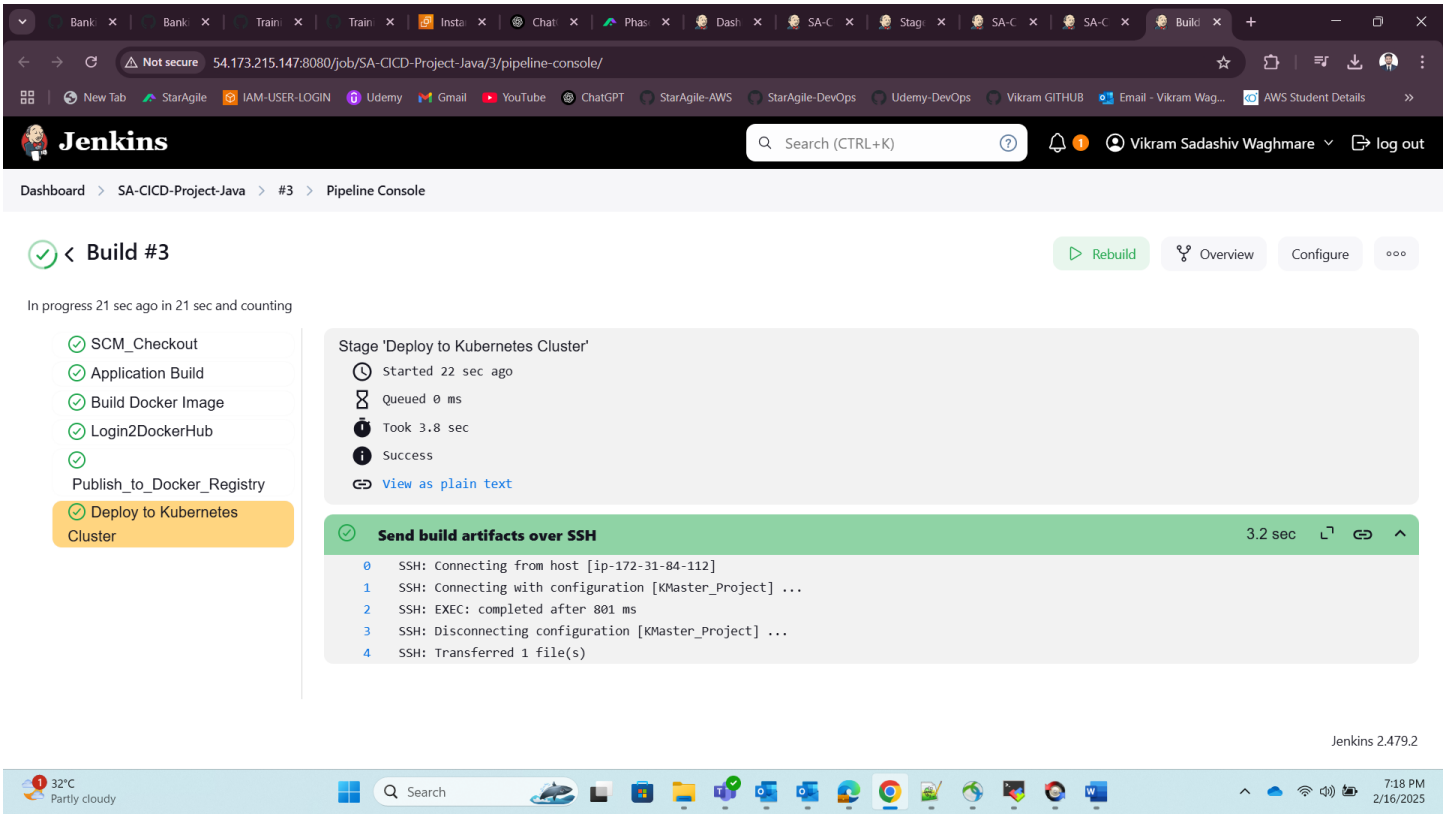
kubectl apply -f kubernetesdeploy.yaml

2. Verify Deployment

kubectl get pods

kubectl get services

📸 Refer to Screenshot



Kubernetes Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: bankapp-eta-deploy
 labels:
  app: bankapp-eta-app
spec:
 replicas: 3
 selector:
  matchLabels:
   app: bankapp-eta-app
 template:
  metadata:
   labels:
    app: bankapp-eta-app
  spec:
   containers:
   - name: bankapp-eta-container
     image: vsw210/bankapp-eta-app:latest
     ports:
     - containerPort: 8082
---
apiVersion: v1
kind: Service
metadata:
 name: bankapp-eta-np-service
 labels:
  app: bankapp-eta-app
spec:
```

```
  selector:
    app: bankapp-eta-app

  type: NodePort
  ports:
  - nodePort: 31030
    port: 8082
    targetPort: 8082
```

- **Phase 5: Jenkins CI/CD Pipeline**

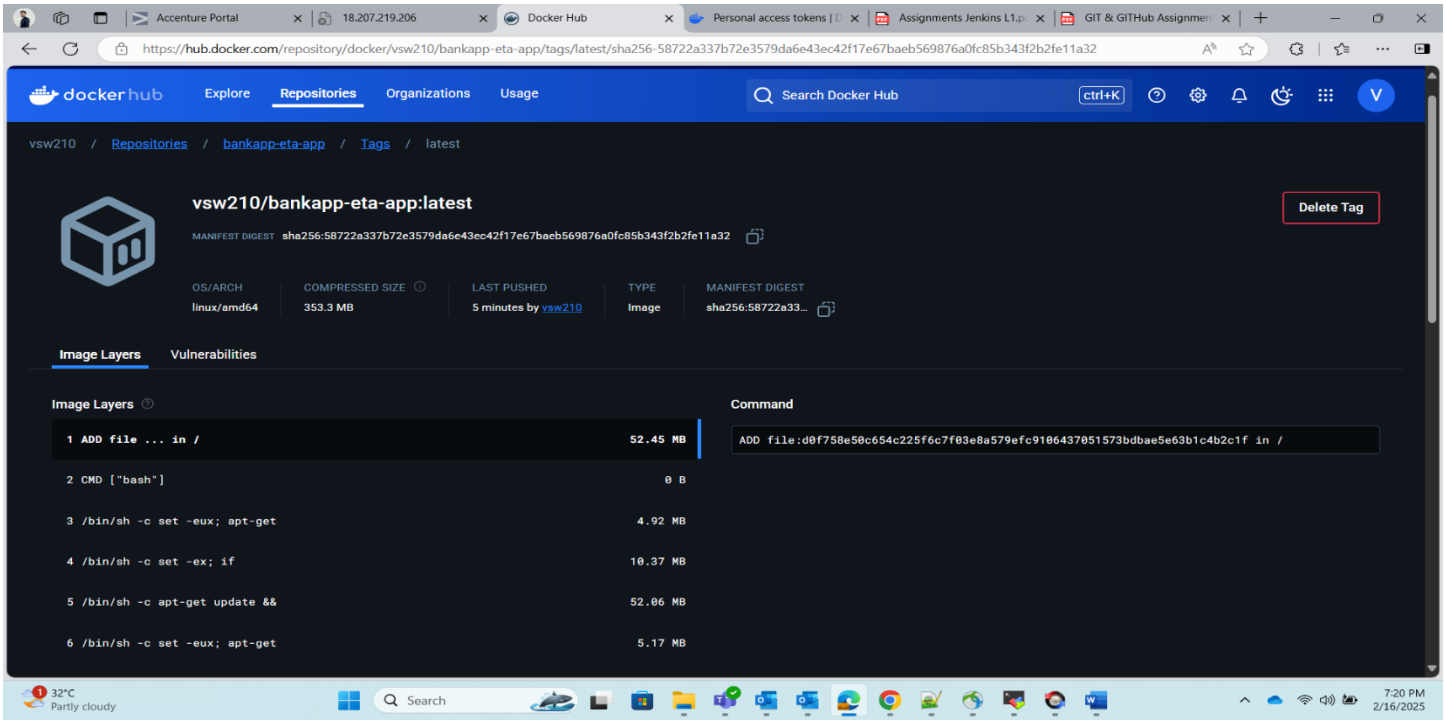
1. **Create Jenkins Pipeline Job**
2. **Use the Following Jenkinsfile**

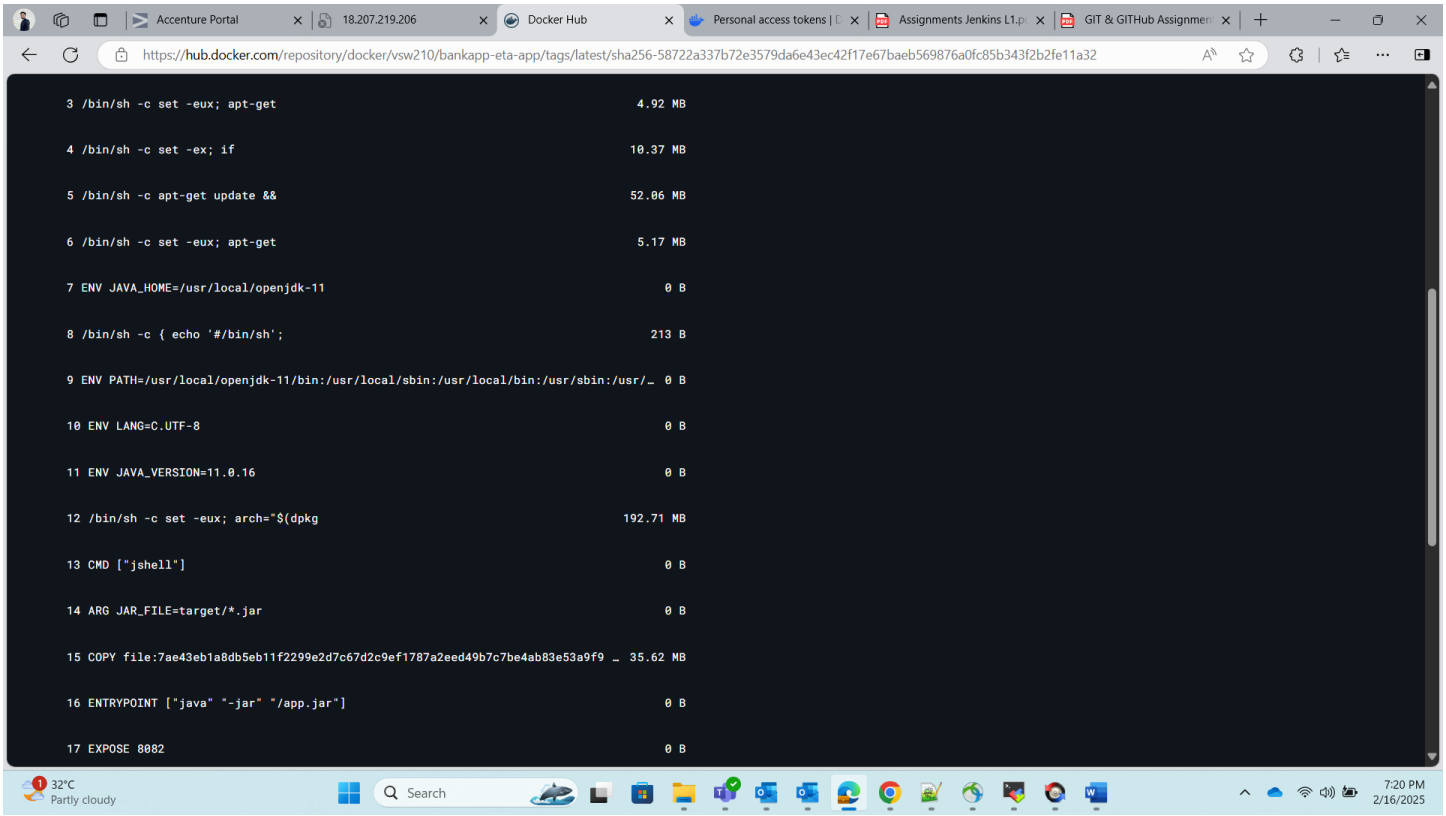
```
pipeline {
    agent { label 'slave1' }

    environment {
                DOCKERHUB_CREDENTIALS=credentials('Project_Docker_Login')
    }

    stages {
        stage('SCM_Checkout') {
            steps {
                echo "Perform SCM Checkout"
                                git 'https://github.com/Vikram9890066071/BankingApp.git'
            }
        }
```

```
    stage('Application Build') {
        steps {
            echo "Perform Application Build"
                                sh 'mvn clean package'
        }
    }
    stage('Build Docker Image') {
        steps {
                                sh 'docker version'
                                sh "docker build -t vsw210/bankapp-eta-app:${BUILD_NUMBER} ."
                                sh 'docker image list'
                                sh "docker tag vsw210/bankapp-eta-app:${BUILD_NUMBER}
vsw210/bankapp-eta-app:latest"
        }
    }
            stage('Login2DockerHub') {

                    steps {
                                sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin'
                    }
            }
            stage('Publish_to_Docker_Registry') {
                    steps {
                                sh "docker push vsw210/bankapp-eta-app:latest"
                    }
            }
    stage('Deploy to Kubernetes Cluster') {
        steps {
            script{
```

```
                sshPublisher(publishers: [sshPublisherDesc(configName: 'KMaster_Project',
transfers: [sshTransfer(cleanRemote: false, excludes: '', execCommand: 'kubectl apply -f
kubernetesdeploy.yaml', execTimeout: 120000, flatten: false, makeEmptyDirs: false,
noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '.', remoteDirectorySDF:
false, removePrefix: '', sourceFiles: '*.yaml')], usePromotionTimestamp: false,
useWorkspaceInPromotion: false, verbose: false)])
            }
        }
    }
  }
}
```

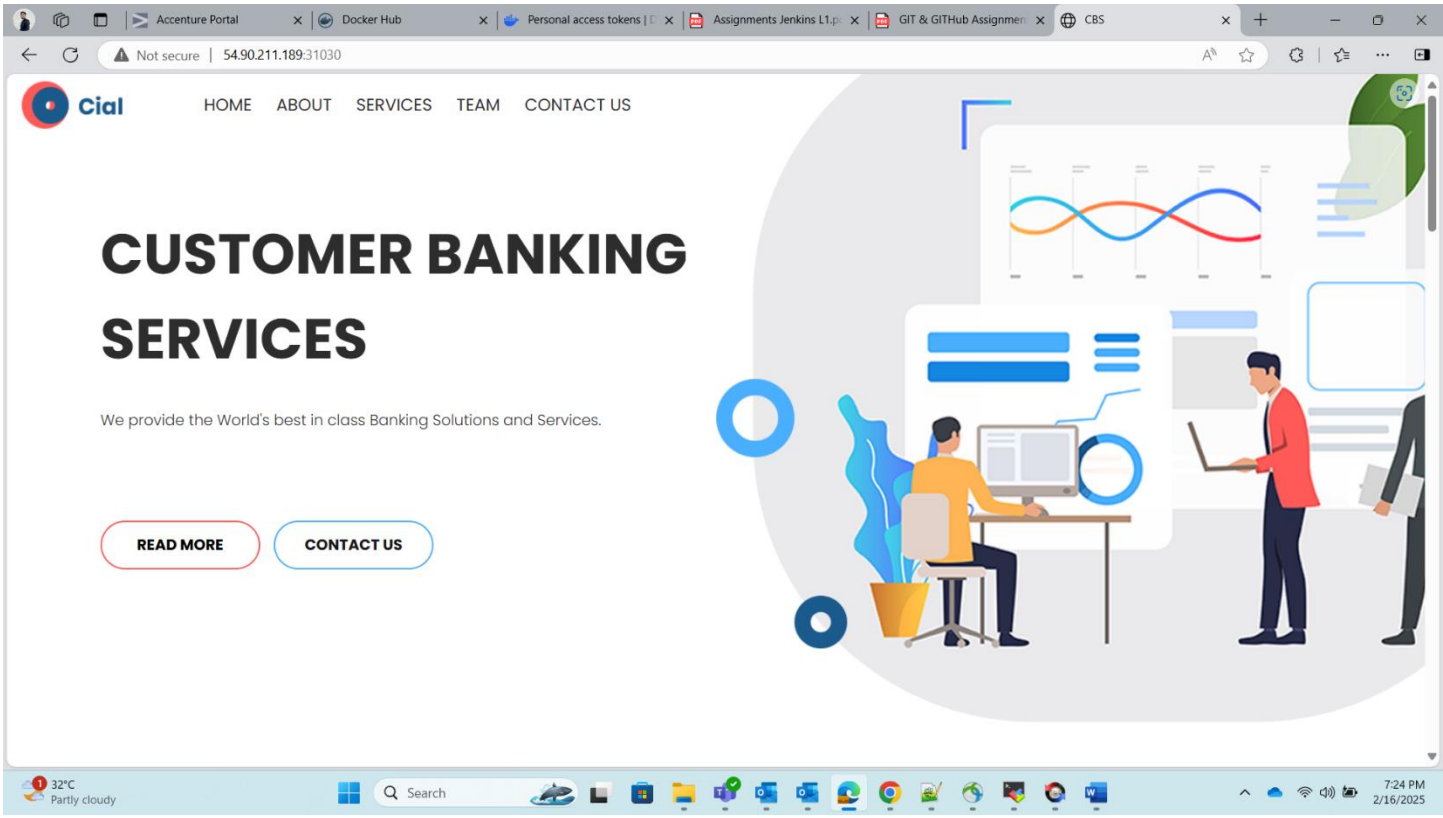
Refer to Screenshot





🔹 **Phase 6: Final Running Web Application**

📸 **Refer to Screenshot**



Phase 7: Monitoring Setup

Installing & Configuring Prometheus & Grafana

Installed Prometheus to collect metrics from Kubernetes clusters.

Configured Grafana dashboards to visualize application and infrastructure metrics.

Monitored key metrics such as:

- CPU Utilization
- Memory Usage
- Disk Space Utilization
- Network Traffic

Monitoring Setup

Step 6: Installing & Configuring Prometheus & Grafana

In this step, we will install **Prometheus** and **Grafana** on our Kubernetes cluster to monitor the infrastructure and application metrics.

---

1. Install Prometheus on Kubernetes

Step 1: Create a Prometheus Namespace

```
kubectl create namespace monitoring
```

Step 2: Deploy Prometheus using Helm

1. Add the Helm repository for Prometheus:

   ```
   helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

   helm repo update
   ```

2. Install Prometheus:

   ```
   helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring
   ```

3. Verify the installation:

   ```
   kubectl get pods -n monitoring
   ```

Output should show running Prometheus components:

NAME	READY	STATUS	RESTARTS	AGE
prometheus-operator-xyz123	1/1	Running	0	2m
prometheus-prometheus-node-exporter-0	1/1	Running	0	2m
prometheus-kube-state-metrics-0	1/1	Running	0	2m
prometheus-prometheus-server-0	2/2	Running	0	2m

---

2. Install Grafana on Kubernetes

Step 1: Deploy Grafana using Helm

```
helm install grafana prometheus-community/grafana --namespace monitoring
```

Step 2: Get Grafana Admin Password

```
kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

Step 3: Port-forward Grafana to Access the UI

```
kubectl port-forward svc/grafana 3000:80 -n monitoring
```

- Open **http://localhost:3000** in your browser.
- Login with:
  - **Username**: admin
  - **Password**: (output from Step 2)

---

3. Configure Prometheus as a Data Source in Grafana

1. **Login to Grafana UI** at http://localhost:3000
2. Navigate to **Configuration → Data Sources**
3. Click **Add Data Source**
4. Select **Prometheus**
5. Enter Prometheus URL:

   ```
   http://prometheus-server.monitoring.svc.cluster.local
   ```

6. Click **Save & Test**

---

4. Create Grafana Dashboards for Monitoring

Step 1: Import a Prebuilt Dashboard

1. Go to **Grafana → Dashboards → Import**
2. Enter dashboard ID **3119** (Kubernetes Cluster Monitoring)
3. Select **Prometheus** as the data source
4. Click **Import**

Step 2: View Metrics

You will see:

- **CPU Utilization**
- **Memory Usage**
- **Disk Space Utilization**

- **Network Traffic**
- **Pod Status**

---

5. Validate Monitoring Setup

- Run the following command to check if Prometheus is scraping metrics:

```
kubectl logs -l app=prometheus -n monitoring
```

- Open **Grafana Dashboards** and verify that data is being visualized.

Verifying Prometheus Metrics

After setting up Prometheus, you can check the available metrics using:

```
kubectl port-forward svc/prometheus-server 9090:80 -n monitoring
```

- Open **http://localhost:9090**
- Run PromQL queries like:

```
container_cpu_usage_seconds_total

node_memory_Active_bytes

container_network_receive_bytes_total
```

- Sample **Prometheus UI Output**:

Time | CPU Usage | Memory Usage | Network Traffic
12:00 | 0.3 cores | 500 MB | 2.5 MB/s
12:05 | 0.4 cores | 600 MB | 3.0 MB/s

---

2. Grafana Dashboard Sample

After integrating **Prometheus as a data source**, importing the **Kubernetes Cluster Dashboard (ID: 3119)** in Grafana will generate real-time metrics.

---

3. Kubernetes Metrics Sample

```
kubectl top nodes
```

Sample Output:

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
kworker-node1	120m	15%	300Mi	30%
kworker-node2	200m	25%	400Mi	40%

Run:

kubectl top pods -n monitoring

Sample Output:

NAME	CPU(cores)	MEMORY(bytes)
prometheus-server-0	110m	250Mi
grafana-xyz123	90m	200Mi

4. Alerting in Grafana (Example)

You can set **alerts** for high CPU usage using **Prometheus AlertManager**.

Example Alert Rule:

```
groups:
 - name: HighCPUUsage
   rules:
     - alert: HighCPUUsage
       expr: node_cpu_seconds_total > 80
       for: 1m
       labels:
         severity: critical
       annotations:
         description: "CPU usage is above 80% for 1 min"
```
