

# 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
<b>Version Control</b>	Git, GitHub
<b>CI/CD Pipeline</b>	Jenkins
<b>Containerization</b>	Docker
<b>Orchestration</b>	Kubernetes
<b>Infrastructure as Code</b>	Terraform
<b>Configuration Management</b>	Ansible
<b>Application Framework</b>	Java, Spring Boot, Maven
<b>Testing</b>	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

The screenshot shows the AWS CloudWatch Metrics Insights interface. At the top, there's a search bar with the query: "AWS Lambda". Below the search bar, there are two tabs: "Metrics" and "Logs". Under the "Metrics" tab, there are two sections: "Metrics from CloudWatch Metrics" and "Metrics from CloudWatch Metrics Insights". The "Metrics from CloudWatch Metrics Insights" section is expanded, showing a table with columns: Metric Name, Metric Type, and Last Value. The table contains several rows related to AWS Lambda metrics like "Lambda Function Invocations" and "Lambda Function Errors". At the bottom of the interface, there's a "Run" button.

Bank | Bank | Train | Train | Insta | Chat | Phas | Dash | SA-C | Stag | SA-C | SA-C | Build | +

New Tab StarAgile IAM-USER-LGIN Udemy Gmail YouTube ChatGPT StarAgile-AWS StarAgile-DevOps Vikram GITHUB Email - Vikram Waghmare AWS Student Details

# Jenkins

Search (CTRL+K) ? 🔔 1 🌐 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

#	Build Number	Start Time
3	#3	13:44
2	#2	12:47
1	#1	12:15

Filter /

Today

32°C Partly cloudy

Search

7:18 PM 2/16/2025

Bank | Bank | Train | Train | Insta | Chat | Phas | Dash | SA-C | Stag | SA-C | SA-C | Build | +

New Tab StarAgile IAM-USER-LGIN Udemy Gmail YouTube ChatGPT StarAgile-AWS StarAgile-DevOps Vikram GITHUB Email - Vikram Waghmare AWS Student Details

# Jenkins

Search (CTRL+K) ? 🔔 1 🌐 Vikram Sadashiv Waghmare log out

Dashboard > SA-CICD-Project-Java > Stages

## Build SA-CICD-Project-Java

Build #3: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

Build #2: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

Build #1: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

Build #3: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

Build #2: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

Build #1: Start → SCM\_Checkout → Application Build → Build Docker Im... → Login2DockerHub → Publish\_to\_Dock... → Deploy to Kuber... → End

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

The screenshot shows a Jenkins pipeline console output for build #3. The left sidebar lists various build-related options like Status, Changes, Console Output (which is selected), Edit Build Information, Delete build, Timings, Git Build Data, Pipeline Overview, Pipeline Console, Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main pane displays the 'Console Output' with the following log entries:

```
Started by user Vikram Sadashiv Waghmare
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Java-SlaveNode1 in /home/devopsadmin/workspace/SA-CI-CD-Project-Java
[Pipeline] {
[Pipeline] withCredentials
Masking supported pattern matches of $DOCKERHUB_CREDENTIALS or $DOCKERHUB_CREDENTIALS_PSW
[Pipeline] {
[Pipeline] stage
[Pipeline] { (SCM_Checkout)
[Pipeline] echo
Perform SCM Checkout
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Fetching changes from the remote Git repository
> git rev-parse --resolve-git-dir /home/devopsadmin/workspace/SA-CI-CD-Project-Java/.git # timeout=10
Checking out Revision f924523b20c6c4928c5533d3dec05d308bb1c77d (refs/remotes/origin/master)
> git config remote.origin.url https://github.com/Vikram9890066071/BankingApp.git # timeout=10
Fetching upstream changes from https://github.com/Vikram9890066071/BankingApp.git
> git --version # timeout=10
```

## **Microservices Endpoints:**

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

The screenshot shows a Jenkins pipeline steps page. On the left, there is a sidebar with various options: Status, Changes, Console Output, Edit Build Information, Timings, Git Build Data, Pipeline Overview, Pipeline Console, Thread Dump, Pause/resume, Replay, Pipeline Steps (which is selected and highlighted in grey), Workspaces, and Previous Build. The main area displays a table of pipeline steps:

Step	Arguments	Status
Start of Pipeline - (47 sec in block)		✓
node - (44 sec in block)	slave1	✓
node block - (44 sec in block)		✓
withCredentials - (43 sec in block)		✓
withCredentials block - (43 sec in block)		✓
stage - (5.4 sec in block)	SCM_Checkout	✓
stage block (SCM_Checkout) - (5.3 sec in block)		✓
echo - (34 ms in self)	Perform SCM Checkout	✓
git - (5.1 sec in self)	https://github.com/Vikram9890066071/BankingApp.git	✓
stage - (24 sec in block)	Application Build	✓

At the bottom of the screen, there is a Windows taskbar with icons for File Explorer, Microsoft Edge, and other applications. The system tray shows the date and time as 2/16/2025 at 7:18 PM.

## ◆ 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

In progress 21 sec ago in 21 sec and counting

Stage 'Deploy to Kubernetes Cluster'  
⌚ Started 22 sec ago  
🕒 Queued 0 ms  
⌚ Took 3.8 sec  
ℹ Success  
➡ View as plain text

⌚ Send build artifacts over SSH 3.2 sec

```
0 SSH: Connecting from host [ip-172-31-84-112]
1 SSH: Connecting with configuration [KMaster_Project] ...
2 SSH: EXEC: completed after 801 ms
3 SSH: Disconnecting configuration [KMaster_Project] ...
4 SSH: Transferred 1 file(s)
```

Jenkins 2.479.2

## 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

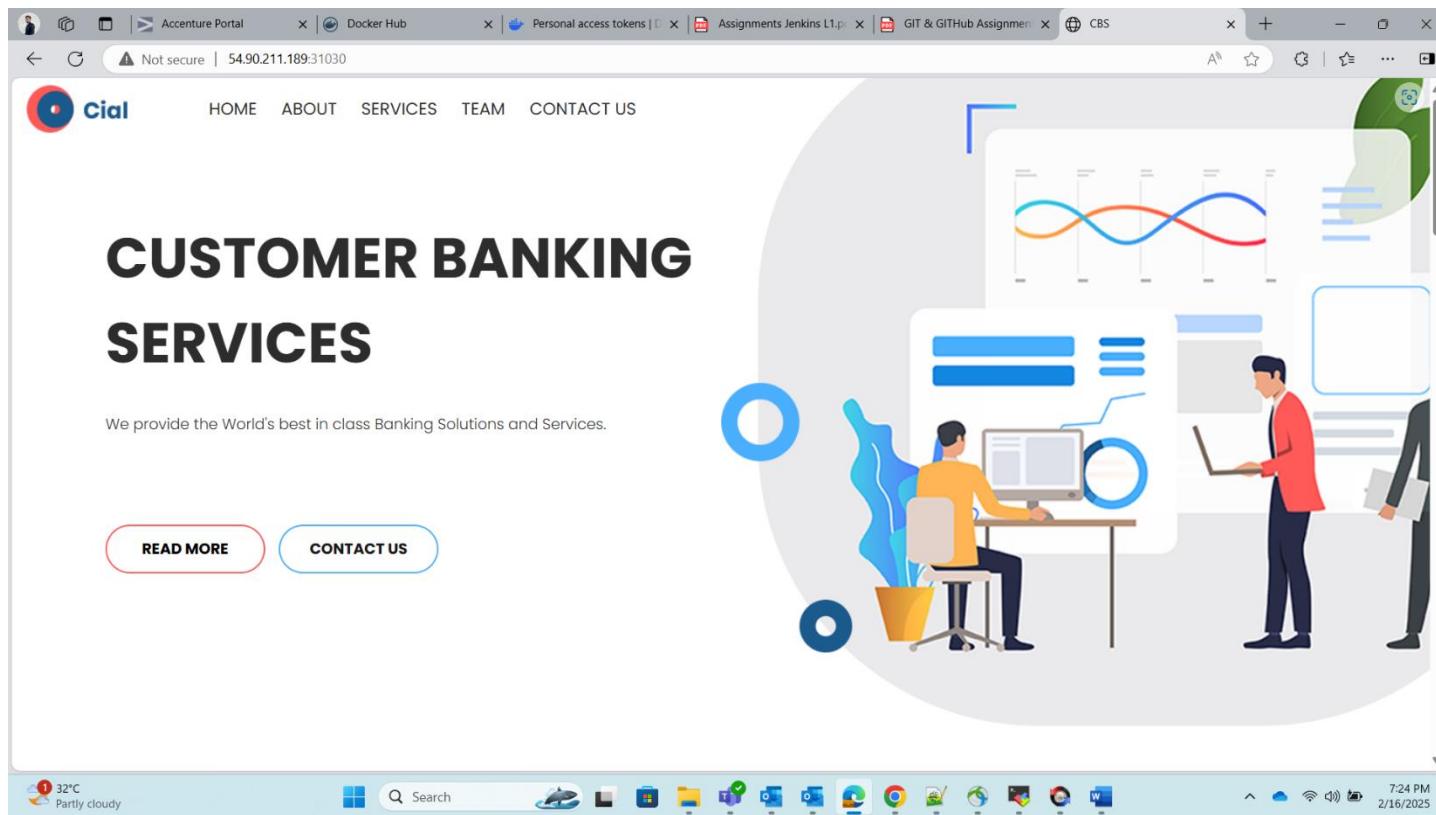
The screenshot shows the Docker Hub interface for the repository `vsw210/bankapp-eta-app`. The page displays the `latest` tag. Key details include:

- Image Layers:** Shows the build history with 17 layers. Layer 1 is an `ADD file ... in /` command (52.45 MB). Layer 2 is a `CMD ["bash"]` command (0 B). Layer 3 is a `/bin/sh -c set -eux; apt-get` command (4.92 MB). Layer 4 is a `/bin/sh -c set -ex; if` command (10.37 MB). Layer 5 is a `/bin/sh -c apt-get update &&` command (52.06 MB). Layer 6 is a `/bin/sh -c set -eux; apt-get` command (5.17 MB).
- Command:** The final command shown is `ADD file:d0f758e50c654c225f6c7f03e8a579efc9106437051573bdbae5e63b1c4b2c1f in /`.
- Manifest Digest:** sha256:58722a337b72e3579da6e43ec42f17e67baeb569876a0fc85b343f2b2fe11a32
- Last Pushed:** 5 minutes ago by `vsw210`
- Type:** Image
- OS/Arch:** linux/amd64
- Compressed Size:** 353.3 MB

This screenshot is identical to the one above, showing the Docker Hub interface for the `vsw210/bankapp-eta-app` repository and the `latest` tag. It displays the same build history, command details, and metadata as the first 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"
```

---