JENKINS CI/CD PIPELINE WITH DOCKER, EKS, AND AWS RDS

(JENKINS MASTER-SLAVE + MAVEN + DOCKER + KUBERNETES (EKS) + RDS)

Project Documentation

Prepared By: Jenifa R

Date: 08 february 2026

❖ PROJECT DETAILS

Item	Details
Project Title	End to End CI/CD Pipeline (Jenkins Master-Slave + Maven + Docker + Kubernetes EKS + AWS RDS)
Domains	DevOps / CI-CD Automation / Cloud-Native Database Integration
Application Type	Java Maven Application (.jar → Docker Image → Kubernetes Deployment)
Source Code Repository	https://github.com/suffixscope/01_products_api.git
Deployment Target	Kubernetes Deployment on Amazon EKS with AWS RDS as backend database
Cloud Platform	AWS (EC2 for Jenkins Master & Slave, EKS for deployment, RDS for database)
Pipeline Tool	Jenkins (Master-Slave setup)
Build Tool	Apache Maven
Containerization	Docker
Orchestration	Kubernetes (Amazon EKS)
Artifact Repository	Docker Hub

Item	Details
Database	AWS RDS (MySQL)

Tools	Version
Jenkins	2.541.1
Java (OpenJDK)	11.0.29
ekstcl	0.221.0
Maven	3.8.4
Docker	25.0.14

❖ OBJECTIVE

Implement a complete CI/CD workflow for a Maven-based Java web application with Jenkins Master-Slave, Docker, Kubernetes (EKS), and AWS RDS integration to:

- Automatically pull source code from GitHub whenever developers push changes.
- Build and package the project using Apache Maven to generate a .jar file.
- Containerize the application by creating a Docker image from the .jar file.
- Push the Docker image to Docker Hub for centralized storage and versioning.
- Deploy the containerized application seamlessly into a Kubernetes cluster (Amazon EKS).
- Integrate AWS RDS (MySQL) as the persistent backend database for storing application data.

- **Verify successful deployment by checking running pods, services, and accessing the application via the AWS Load Balancer URL.**
- **Ensure distributed build execution using Jenkins Master-Slave setup for scalability and reliability.**

❖ HIGH-LEVEL ARCHITECTURE

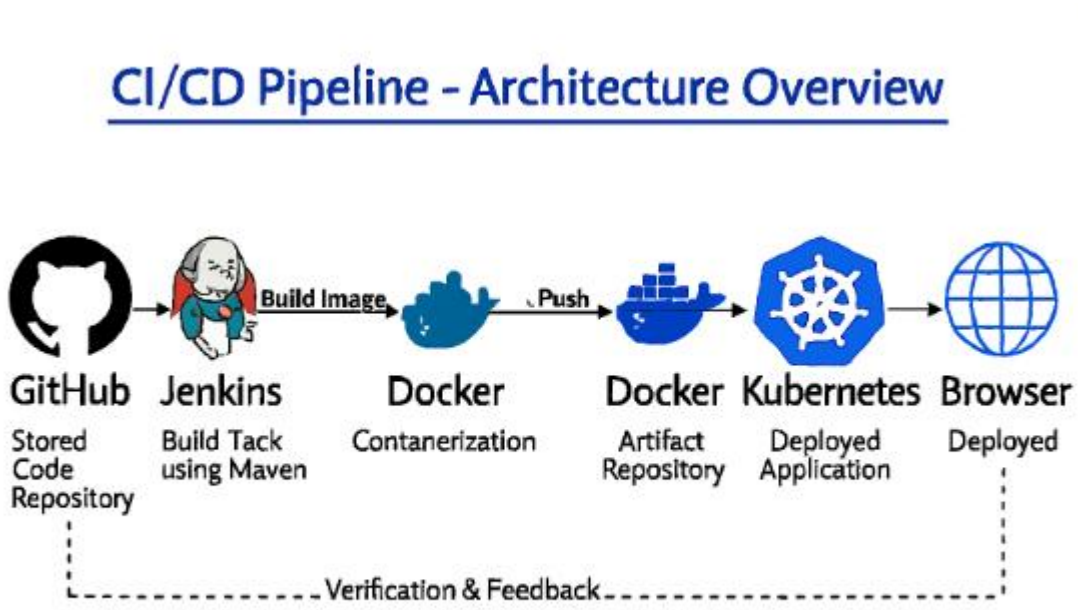
Flow: GitHub → Jenkins Master-Slave Pipeline → (Build/Test with Maven) → Docker Image Build → Docker Hub Upload → Kubernetes (EKS) Deployment → AWS RDS Database Integration → Verification

Components:

- **GitHub: Stores and manages the source code repository.**
- **Jenkins (Master-Slave): Orchestrates the CI/CD pipeline, distributes build/test stages across master and slave nodes.**
- **Maven: Builds and packages the Java application into a .jar file.**
- **Docker: Containerizes the application by creating Docker images.**
- **Docker Hub: Stores and versions the Docker images for deployment.**
- **Kubernetes (EKS): Orchestrates and manages containerized application deployment in AWS.**
- **AWS RDS (MySQL): Provides persistent database storage for application data.**
- **AWS Load Balancer: Exposes the application to external users with a public endpoint.**

- **Browser: Used to validate successful deployment by accessing the application via the Load Balancer URL.**

Architecture Diagram



❖ INFRASTRUCTURE SETUP – AWS EC2 & RDS ARCHITECTURE

Operating System

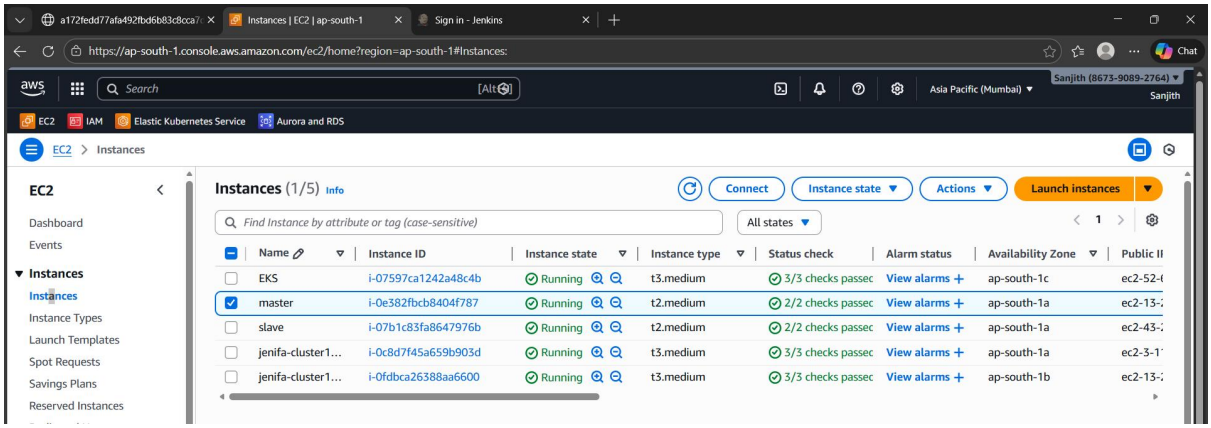
Amazon Linux AMI

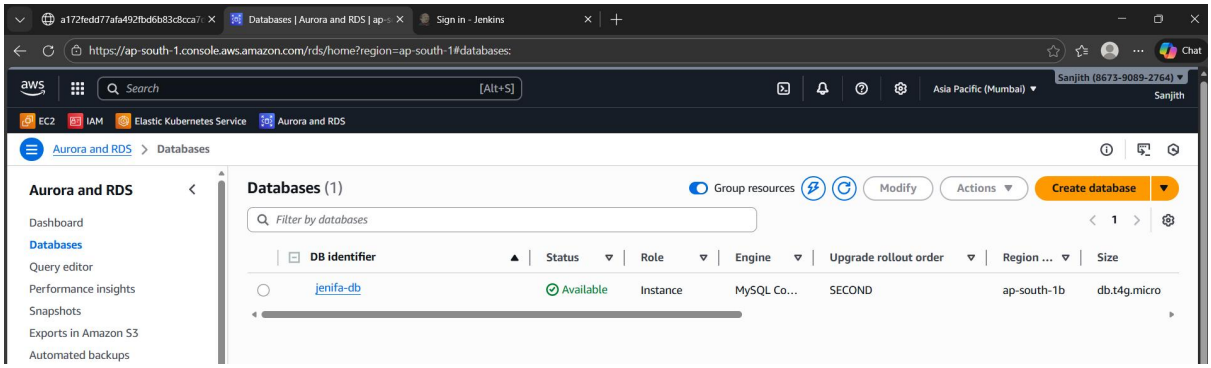
Service	EC2 Instance Name	Purpose	Default Port

Service	EC2 Instance Name	Purpose	Default Port
Jenkins Master	jenkins-master	CI/CD pipeline orchestration & job scheduling	8080
Jenkins Slave	jenkins-slave	Distributed build execution (runs jobs assigned by master)	22 (SSH)
EKS Cluster	eks-cluster	Kubernetes cluster creation & management (kubectl, eksctl, AWS CLI)	22 (SSH)
Docker Hub	Cloud Service	Artifact repository for container images	–
AWS RDS (MySQL)	jenifa-db	Persistent database for application data	3306

Security Group – Inbound Ports

- 22 (SSH) → Restrict to your IP only (for EC2 & RDS admin access)
- 8080 → Jenkins Master web UI
- 3306 → MySQL (RDS database access from Jenkins/EKS pods)

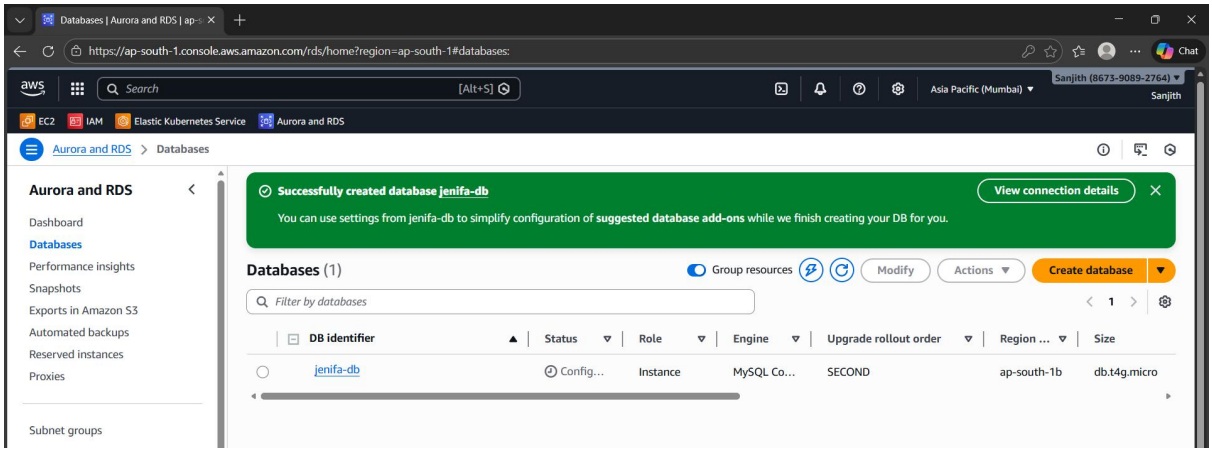




❖ RDS SETUP – AWS MySQL Database

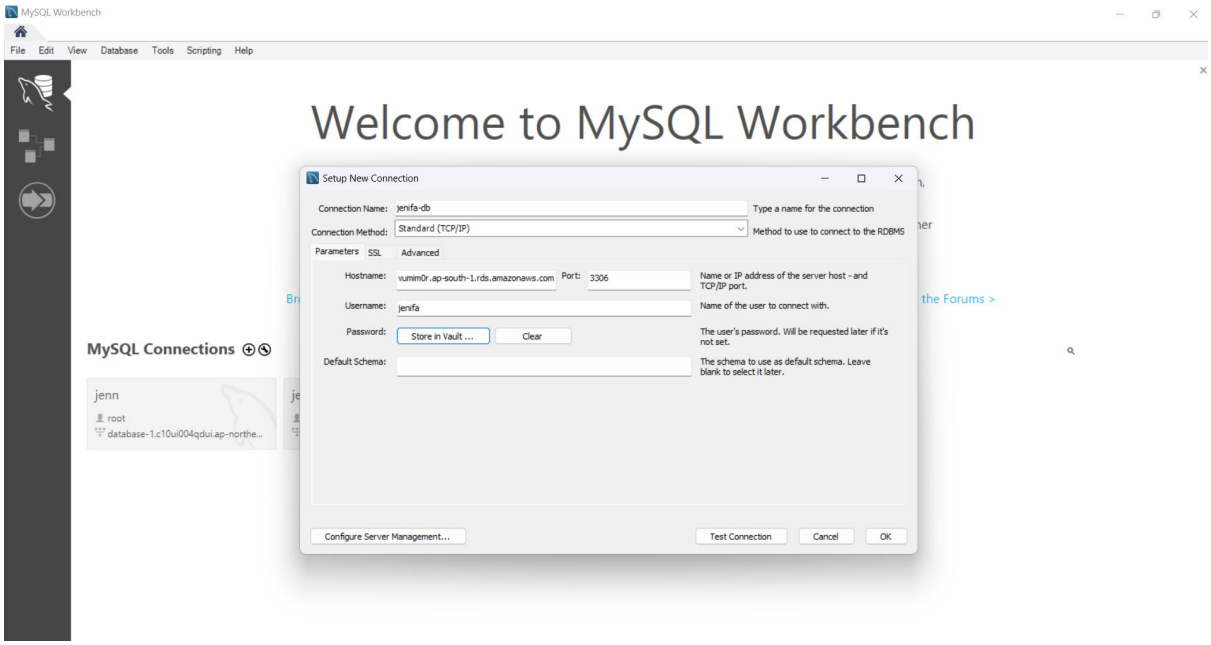
Database Creation

- **Created a new AWS RDS MySQL instance with identifier: jenifa-db.**
- **Engine: MySQL Community Edition.**
- **Instance type: db.t4g.micro (suitable for testing and small workloads).**
- **Region: ap-south-1 (Mumbai).**
- **Port: 3306 (default MySQL port).**
- **Publicly accessible: Yes (to allow Jenkins/EKS pods to connect).**

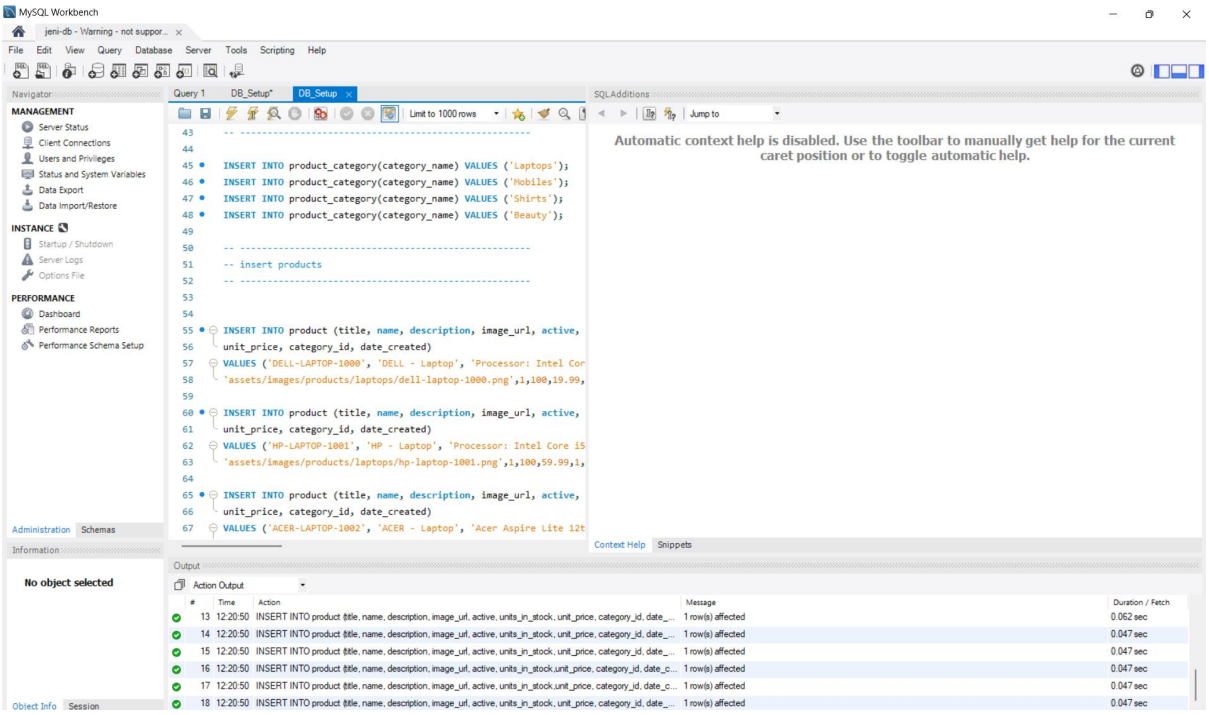


Connectivity

- **Endpoint: jenifa-db.cjqsqwumim0r.ap-south-1.rds.amazonaws.com**
- **Username: jenifa**
- **Database: jenifa_db**
- **Verified connection using MySQL Workbench → *Test Connection successful.***

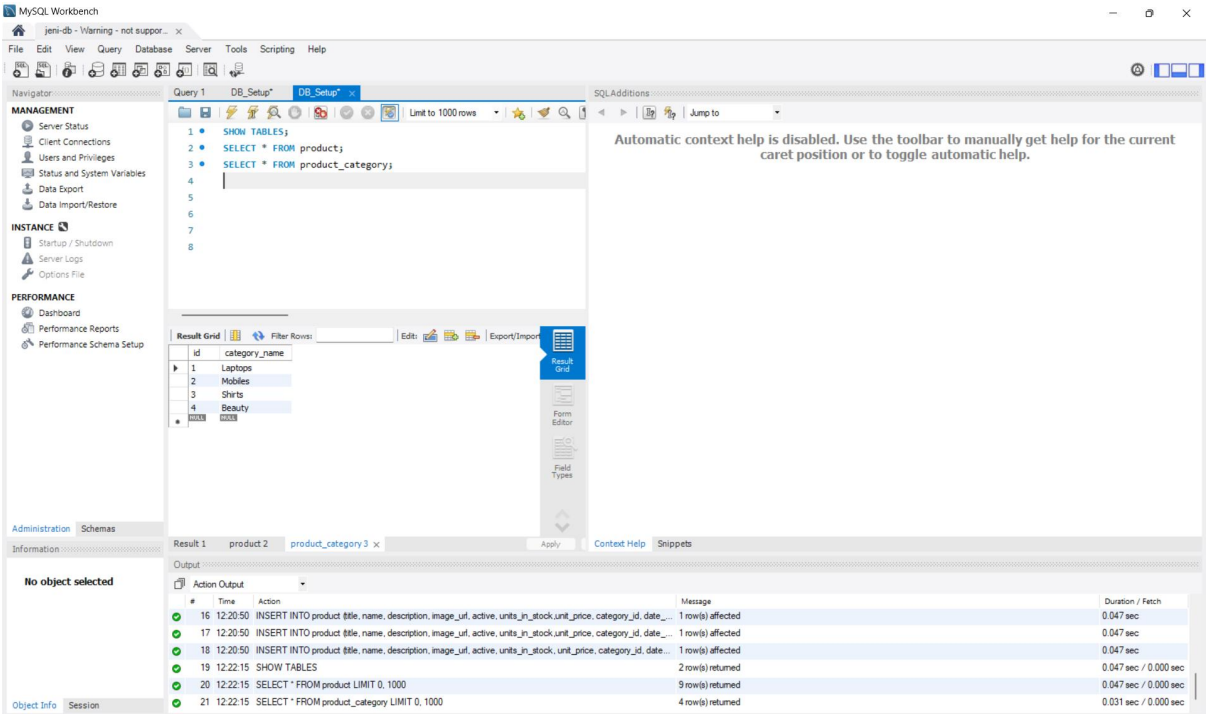


Database Initialization



Verification

Ran queries in MySQL Workbench:



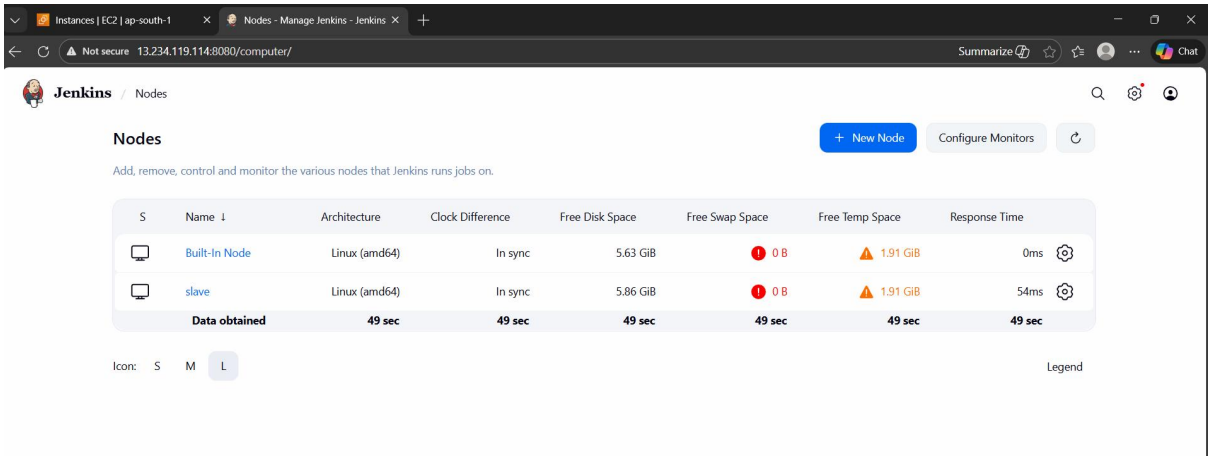
Purpose

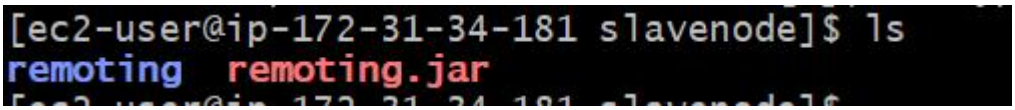
- Provides persistent storage for the Maven web application.
- Ensures data survives container restarts in Kubernetes.
- Allows multiple pods to connect to the same database for scalability.
- Credentials will be managed via Jenkins credentials or Kubernetes secrets for security.

❖ JENKINS SETUP – MASTER & SLAVE CONFIGURATION

- Jenkins Master installed on EC2 instance jenkins-master.
- Jenkins Slave configured and connected via SSH from the master dashboard.
- Port 8080 opened for Jenkins web UI access.

- Slave node added under Manage Jenkins → Nodes → New Node with proper labels and credentials.



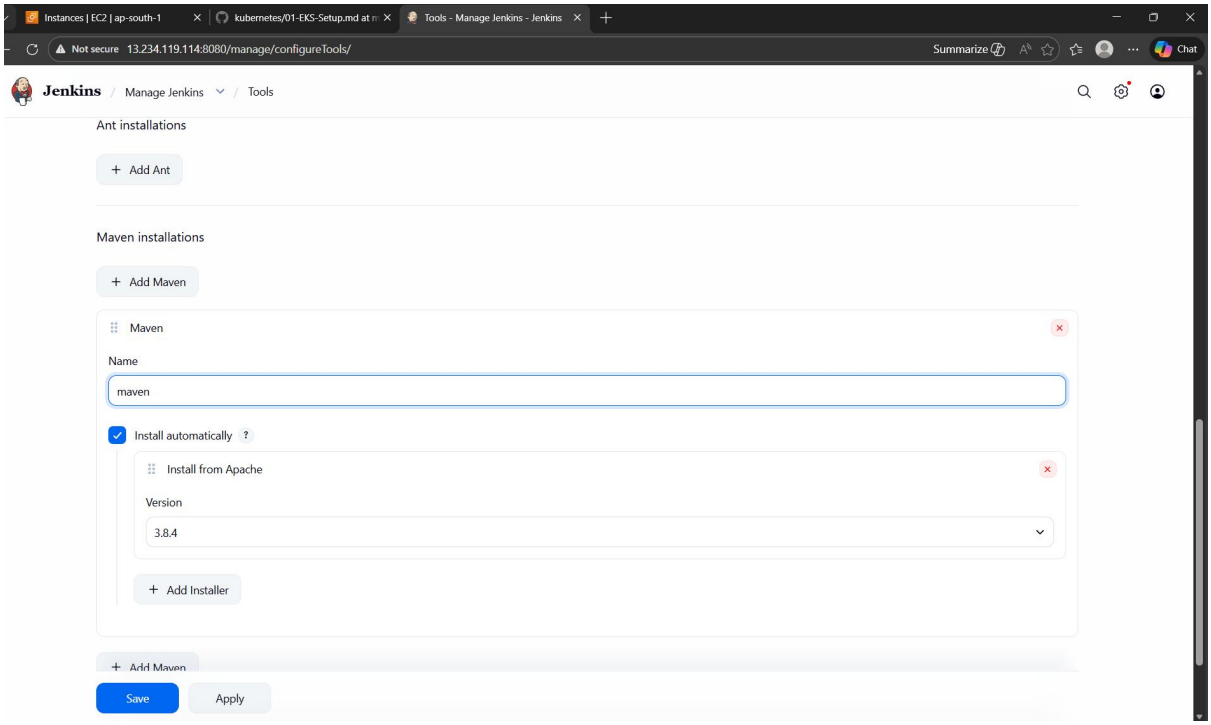


Global Tool Configuration

- Navigate to: Manage Jenkins → Global Tool Configuration
- Configure

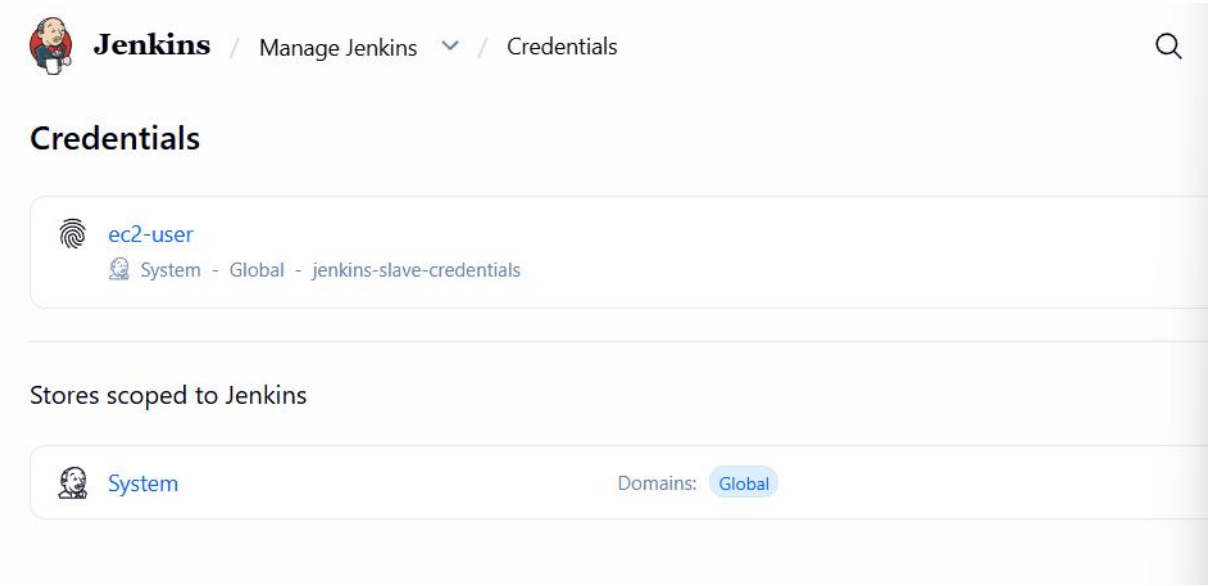
  Maven → Name: maven

  Version: 3.8.4



Credentials Setup

- **Added SSH credentials (ec2-user) under Manage Jenkins → Credentials.**
- **Scope: System → Global.**
- **Credential ID: jenkins-slave-credentials.**
- **Used for connecting Jenkins Master to Slave securely.**



❖ EKS HOST VM SETUP

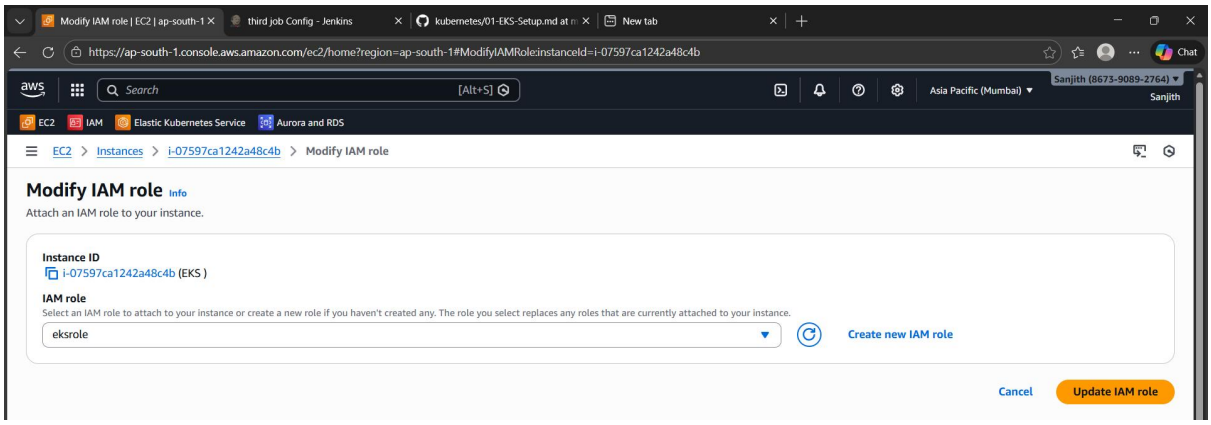
The EKS Host VM was launched and configured with AWS CLI, kubectl, and eksctl to manage the Kubernetes cluster.

⇒ IAM Role setup:

An IAM role was created and the following permissions were attached to the EKS Host VM to enable secure management of AWS resources and Kubernetes cluster operations.

Attach Permissions

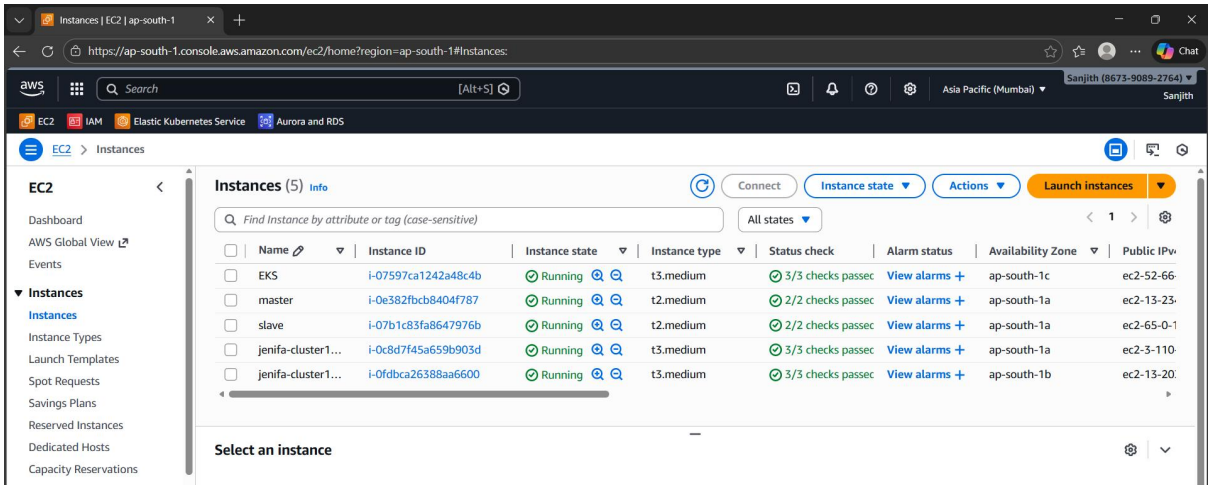
- **IAM Full Access**
- **VPC Full Access**
- **EC2 Full Access**
- **CloudFormation Full Access**
- **Administrator Access**



⇒ Cluster Creation

create Kubernetes cluster using eksctl

```
eksctl create cluster --name jenifa-cluster1 --region ap-south-1 --node-type t3.medium  --zones ap-south-1a,ap-south-1b
```



❖ KUBERNETES DEPLOYMENT SETUP (JENKINS CLUSTER CONNECTION)

1. Install Required Tools on Jenkins Server

- **AWS CLI → to connect Jenkins with AWS EKS.**
- **kubectl → to interact with the Kubernetes cluster.**
- **Ensure both are installed and available in Jenkins environment.**

2. Copy Kubeconfig File to Jenkins

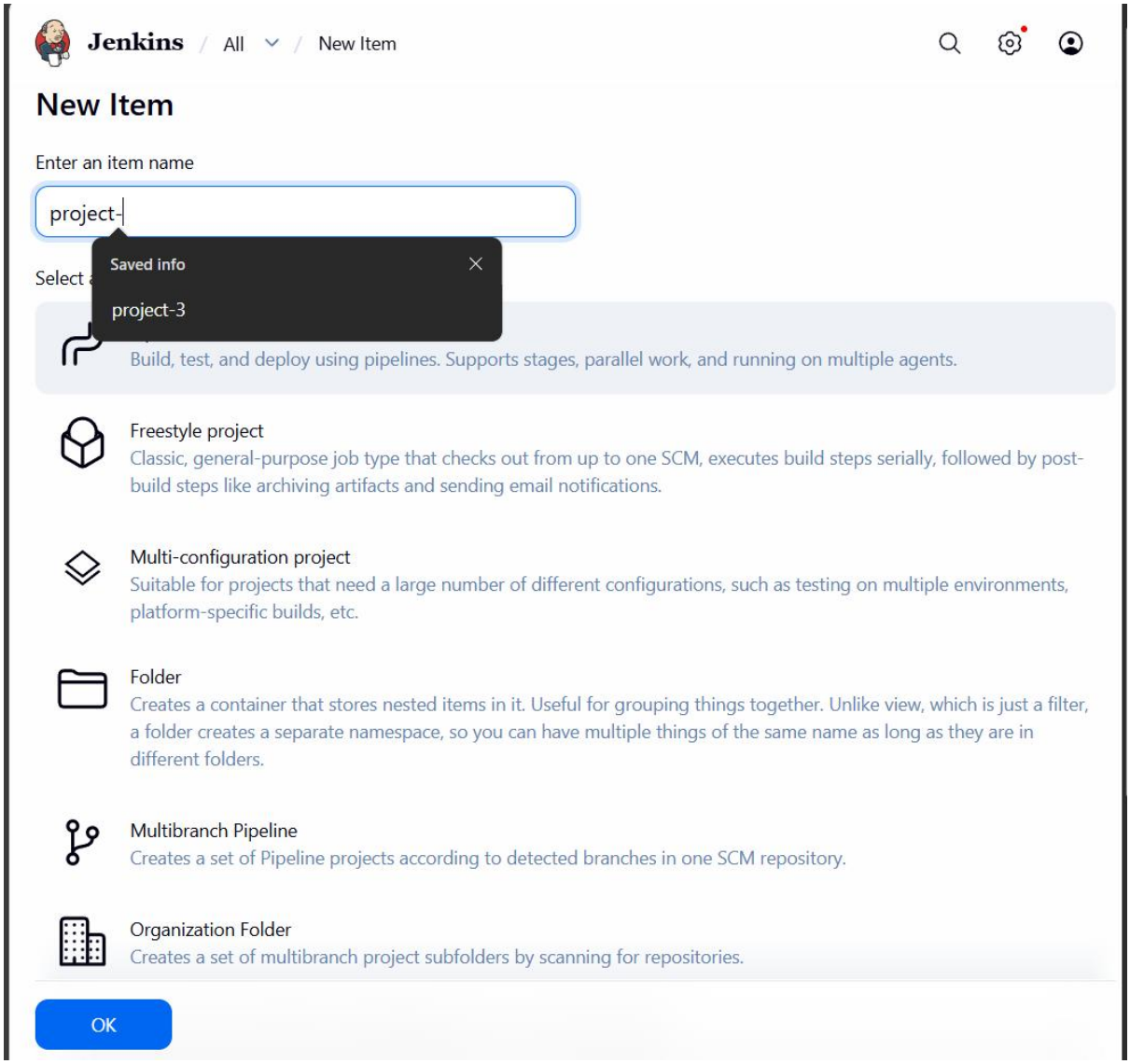
- **When you create the cluster in EKS VM, a kubeconfig file**
- **Copy this file into the Jenkins server.**

3. Verify Cluster Nodes

```
[ec2-user@jenkins-master ~]$ kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
ip-192-168-32-217.ap-south-1.compute.internal   Ready    <none>   23h   v1.32.9-eks-ecaa3a6
ip-192-168-7-167.ap-south-1.compute.internal    Ready    <none>   23h   v1.32.9-eks-ecaa3a6
[ec2-user@jenkins-master ~]$
```

```
[ec2-user@jenkins-slave ~]$ kubectl get nodes
NAME                                        STATUS   ROLES    AGE   VERSION
ip-192-168-32-217.ap-south-1.compute.internal   Ready    <none>   23h   v1.32.9-eks-ecaa3a6
ip-192-168-7-167.ap-south-1.compute.internal    Ready    <none>   23h   v1.32.9-eks-ecaa3a6
[ec2-user@jenkins-slave ~]$
```

4.Job Creation



5.Application Configuration (spring.datasource)

- **Add RDS endpoint, username, and password to application.properties.**
- **This ensures the app knows how to connect to the database when deployed.**

```
ec2-user@slave:~/slavenode/workspace/project-3
spring.datasource.url=jdbc:mysql://jenifa-db.cjqsgwumim0r.ap-south-1.rds.amazonaws.com:3306/jenifa_db
spring.datasource.username=jenifa
spring.datasource.password=jenifa!1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
~
~
```

6. Create deployment.yaml (slave vm)

This file defines how your app runs in Kubernetes.

- **On yourJenkins server (or local machine), go to your project workspace**
- **Create the file: deployment.yaml**
- **Paste this content:**

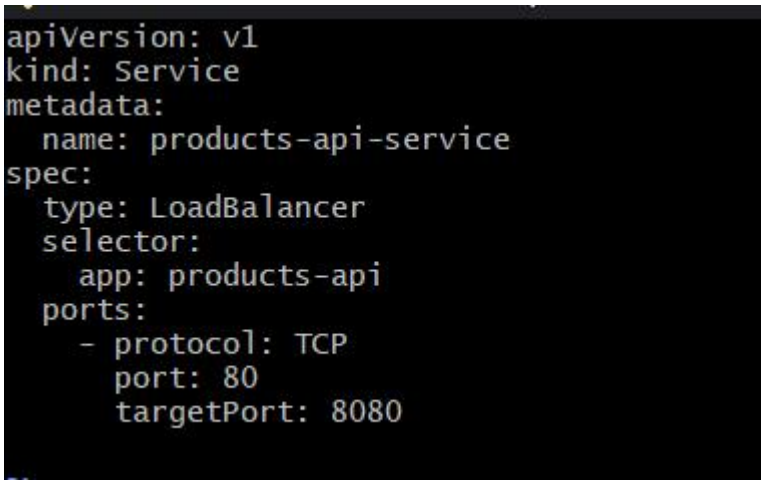
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products-api-deployment
  labels:
    app: products-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: products-api
  template:
    metadata:
      labels:
        app: products-api
    spec:
      containers:
        - name: products-api
          image: <your-dockerhub-username>/01_products_api:latest
          ports:
            - containerPort: 8080
          env:
            - name: SPRING_DATASOURCE_URL
              value: jdbc:mysql://jenifa-db.cjqsgwumim0r.ap-south-1.rds.amazonaws.com:3306/jenifa_db
            - name: SPRING_DATASOURCE_USERNAME
              value: jenifa
            - name: SPRING_DATASOURCE_PASSWORD
              value: jenifa!1234
            - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
              value: com.mysql.cj.jdbc.Driver
```

- **Save and exit.**

7. Create service.yaml (slave vm)

This file exposes your app via AWS LoadBalancer

- **On yourJenkins server (or local machine), go to our project workspace.**
- **Create the file: service.yaml**
- **Paste this content:**



```
apiVersion: v1
kind: Service
metadata:
  name: products-api-service
spec:
  type: LoadBalancer
  selector:
    app: products-api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

- **Save and exit.**

8. Pipeline Execution for Deployment

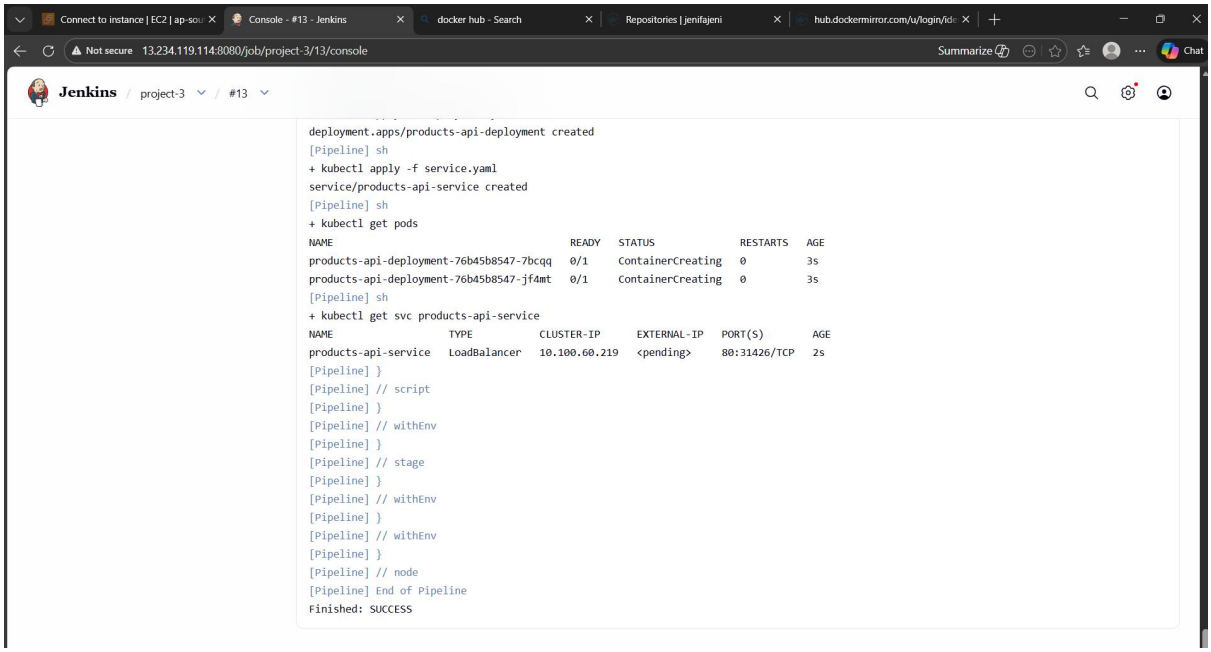
```
pipeline {
    agent { label 'slave' }

    tools {
        maven 'maven'
    }

    environment {
        DOCKER_HUB_USER = 'janifajani'
        DOCKER_HUB_PASS = 'janijanis'
        IMAGE_NAME = '01_products_api'
    }

    stages {
        stage('Clone') {
            steps {
                git branch: 'main', url: 'https://github.com/suffixscope/01_products_api.git'
            }
        }

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

        stage('Docker Build & Push') {
            steps {
                script {
                    writeFile file: 'Dockerfile', text: '''
                    FROM eclipse-temurin:17
                    COPY target/products_api.jar /usr/app/
                    WORKDIR /usr/app
                    CMD ["java", "-jar", "products_api.jar"]
                    '''

                    sh "docker build -t ${IMAGE_NAME} ."
                    sh "docker login -u ${DOCKER_HUB_USER} -p ${DOCKER_HUB_PASS}"
                    sh "docker tag ${IMAGE_NAME} ${DOCKER_HUB_USER}/${IMAGE_NAME}:latest"
                    sh "docker push ${DOCKER_HUB_USER}/${IMAGE_NAME}:latest"
                }
            }
        }

        stage('Deploy to EKS') {
            steps {
                script {
                    writeFile file: 'deployment.yaml', text: '''
                    apiVersion: apps/v1
                    kind: Deployment
                    metadata:
                      name: products-api-deployment
                      labels:
                        app: products-api
                    spec:
                      replicas: 2
                      selector:
                        matchLabels:
                          app: products-api
                      template:
                        metadata:
                          labels:
                            app: products-api
                        spec:
                          containers:
                            - name: products-api
                              image: ${DOCKER_HUB_USER}/${IMAGE_NAME}:latest
                              ports:
                                - containerPort: 8080
                              env:
                                - name: SPRING_DATASOURCE_URL
                                  value: jdbc:mysql://janifa-db.cjqzgwunin0r.ap-south-1.rds.amazonaws.com:3306/janifa_db
                                - name: SPRING_DATASOURCE_USERNAME
                                  value: janifa
                                - name: SPRING_DATASOURCE_PASSWORD
                                  value: janifa11234
                                - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
                                  value: com.mysql.cj.jdbc.Driver
                    ---

                    writeFile file: 'service.yaml', text: '''
                    apiVersion: v1
                    kind: Service
                    metadata:
                      name: products-api-service
                    spec:
                      type: LoadBalancer
                      selector:
                        app: products-api
                      ports:
                        - protocol: TCP
                          port: 80
                          targetPort: 8080
                    ---

                    sh "kubectl apply -f deployment.yaml"
                    sh "kubectl apply -f service.yaml"
                    sh "kubectl get pods"
                    sh "kubectl get svc products-api-service"
                }
            }
        }
    }
}
```

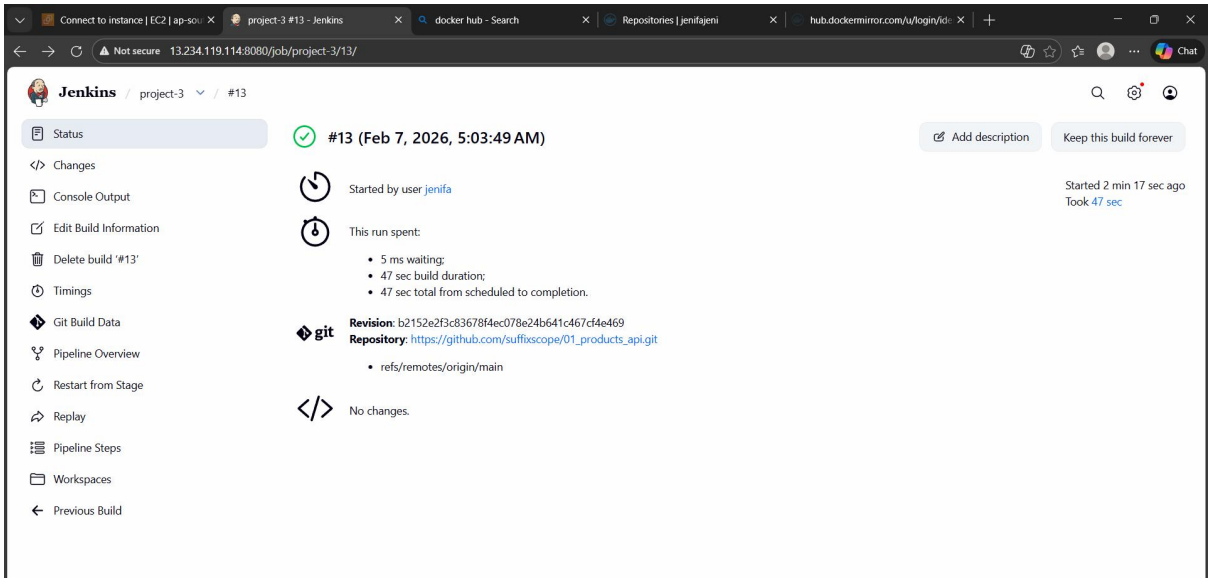
❖ CI/CD Pipeline Implementation (Jenkins)

Pipeline Stages

Stage	What it does	Output / Result
Clone Repo	Clones the master branch from GitHub into Jenkins workspace	Source code available in Jenkins Master/Slave workspace
Maven Build	Runs mvn -B clean package using Maven3 on Jenkins Slave	JAR/WAR generated: target/maven-web-app.jar
Docker Build	Builds Docker image from the JAR/WAR using Dockerfile	Docker image created locally
Docker Push	Pushes the Docker image to Docker Hub using stored credentials	Image versioned and stored in Docker Hub
Kubernetes Deploy (EKS)	Applies deployment.yaml and service.yaml via kubectl from Jenkins Master	Application deployed to EKS cluster
AWS RDS Integration	Application connects to RDS using application.properties datasource config	Persistent database connection established
Load Balancer Exposure	AWS Load Balancer exposes the Kubernetes service to the internet	Public endpoint generated (DNS URL)
Verification	Access application via Load Balancer URL in browser	Application accessible, data retrieved from RDS

❖TESTING AND VERIFICATION





```
[ec2-user@slave ~]$ kubectl get nodes
NAME                                           STATUS   ROLES    AGE   VERSION
ip-192-168-32-217.ap-south-1.compute.internal  Ready    <none>   3d    v1.32.9-eks-ecaa3a6
ip-192-168-7-167.ap-south-1.compute.internal   Ready    <none>   3d    v1.32.9-eks-ecaa3a6
[ec2-user@slave ~]$
```

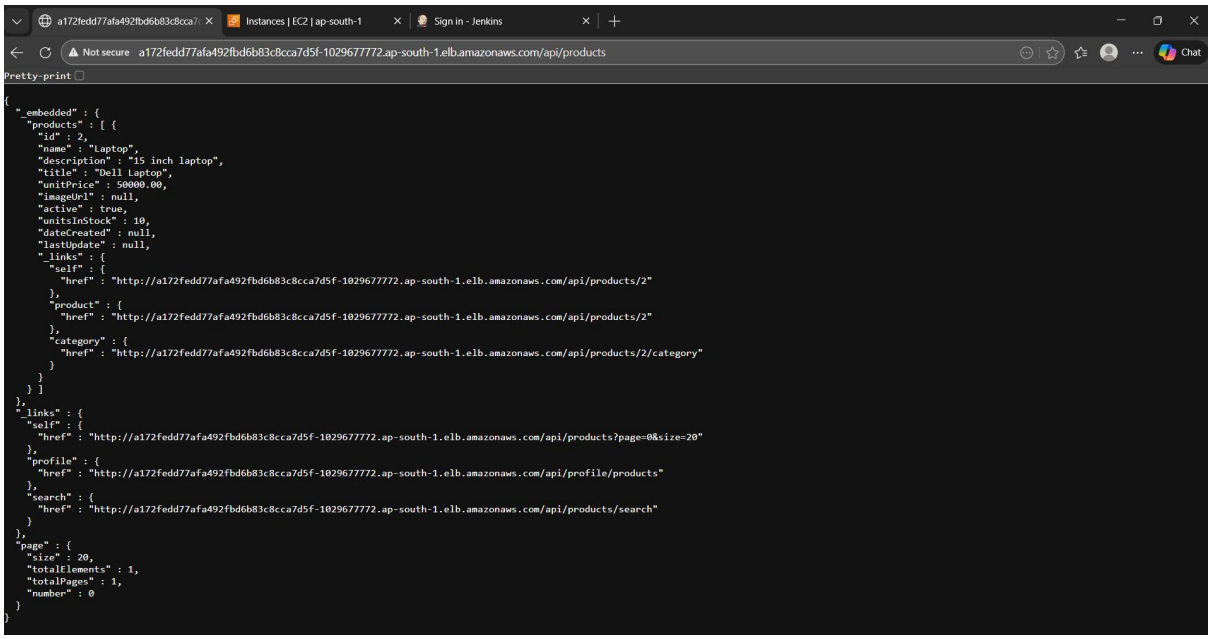
```
[ec2-user@slave ~]$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
products-api-deployment-76b45b8547-7bcqq   1/1     Running   0          26h
products-api-deployment-76b45b8547-jf4mt   1/1     Running   0          26h
```

```
[ec2-user@slave ~]$ kubectl get svc
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
kubernetes          ClusterIP      10.100.0.1      <none>                                                                     443/TCP        3d
products-api-service   LoadBalancer   10.100.60.219   a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com   80:31426/TCP   26h
[ec2-user@slave ~]$
```

Application Access:

Copy the LoadBalancer DNS and paste in browser as :

a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products




```
{
  "_embedded" : {
    "products" : [ {
      "id" : 2,
      "name" : "Laptop",
      "description" : "15 inch laptop",
      "title" : "Dell Laptop",
      "unitPrice" : 50000.00,
      "imageUrl" : null,
      "active" : true,
      "unitsInStock" : 10,
      "dateCreated" : null,
      "lastUpdate" : null,
      "_links" : {
        "self" : {
          "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products/2"
        },
        "product" : {
          "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products/2"
        },
        "category" : {
          "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products/2/category"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products?page=0&size=20"
    },
    "profile" : {
      "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/profile/products"
    },
    "search" : {
      "href" : "http://a172fedd77afa492fbd6b83c8cca7d5f-1029677772.ap-south-1.elb.amazonaws.com/api/products/search"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 1,
    "totalPages" : 1,
    "number" : 0
  }
}
```


❖ CONCLUSION

This project successfully demonstrates a modern, cloud- native CI/CD pipeline that integrates Jenkins Master-Slave, Docker, Kubernetes (EKS), and AWS RDS. By following a structured approach:

- **RDS Setup ensured persistent database storage with schema and seed data ready before deployment.**
- **Jenkins Master-Slave Configuration distributed build workloads securely and efficiently.**
- **Pipeline Implementation automated the flow from GitHub source → Maven build → Docker image → Docker Hub → Kubernetes deployment.**
- **Application Configuration connected the deployed app to AWS RDS, enabling real-time data persistence.**
- **AWS Load Balancer exposed the application publicly, making it accessible via browser endpoints.**

This pipeline highlights your progression from a traditional CI/CD workflow (Jenkins + SonarQube + Nexus + Tomcat) to a cloud-native architecture (Jenkins + Docker + Kubernetes + RDS). It showcases not only technical depth but also industry-standard practices in automation, scalability, and reliability.

❖Final Outcomes

By completing all stages, your third project achieves a fully automated, cloud-native CI/CD workflow. The outcomes are clear and measurable:

- **Source Control (GitHub)**
  - **Codebase securely stored and versioned.**
  - **Jenkins jobs always pull the latest commit from the master branch.**
- **Build & Packaging (Maven on Jenkins Slave)**
  - **Application compiled and packaged into JAR/WAR.**

    - **Build logs and artifacts verified for consistency.**
- **Containerization (Docker)**
    - **Application wrapped into a Docker image.**
    - **Image tagged with version numbers for traceability.**
- **Artifact Management (Docker Hub)**
    - **Docker images pushed to Docker Hub repository.**
    - **Centralized storage ensures reproducibility across environments.**
- **Deployment (Kubernetes EKS)**
    - **Application deployed to AWS EKS cluster using deployment.yaml and service.yaml.**
    - **Cluster nodes verified in Ready state.**
- **Database Integration (AWS RDS)**
    - **Application connected to jenifa_db via Spring Boot datasource configuration.**
    - **Persistent storage ensures data survives pod restarts.**
- **Exposure (AWS Load Balancer)**
    - **Application publicly accessible via Load Balancer DNS endpoint.**
    - **API endpoints (e.g., /api/products) return live data from RDS.**
- **Verification & Feedback**
    - **Browser access confirms application is live.**
    - **Queries to /api/products validate DB integration.**
    - **Logs and monitoring confirm pipeline stability.**