

## Virtual Lab Experiment – 6

### CI/CD Pipeline to Deploy Dockerized Application on Kubernetes using Jenkins, GitHub, Maven, Terraform, and Ansible

#### **Objective:**

To build and deploy a Dockerized Java web application to a Kubernetes cluster on AWS using a fully automated CI/CD pipeline with Jenkins, Maven, GitHub, Terraform, and Ansible.

---

#### **Tools & Technologies Used**

- Terraform (IaC)
  - Ansible (Configuration Management)
  - Docker (Containerization)
  - Jenkins (CI/CD)
  - GitHub (Source Control)
  - Maven (Build Automation)
  - Kubernetes (Orchestration)
  - AWS (Cloud Infrastructure)
- 

#### **Step-by-Step Procedure**

---

##### ◆ **STEP 1: Clone Java Application from GitHub**

1. Use a sample Maven-based Java web app (e.g., Spring Boot).
2. Fork or clone it from GitHub:

```
git clone https://github.com/your-username/sample-java-app.git
```

---

##### ◆ **STEP 2: Provision Infrastructure Using Terraform**

1. **Install Terraform:**

```
sudo apt-get install terraform
```

2. **Create Terraform scripts to:**

- Provision an EC2 instance for Jenkins
- Create an EKS (Elastic Kubernetes Service) cluster
- Set up IAM roles, security groups, and networking (VPC, Subnets)

### 3. Initialize & Apply Terraform:

```
terraform init
```

```
terraform plan
```

```
terraform apply
```

```
ubuntu@ip-172-31-37-221:~/sample-java-app/terraform-cicd$ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.15.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
ubuntu@ip-172-31-37-221:~/sample-java-app/terraform-cicd$ terraform plan
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_eks_cluster.cicd_cluster will be created
+ resource "aws_eks_cluster" "cicd_cluster" {
    + arn          = (known after apply)
    + bootstrap_self_managed_addons = true
    + certificate_authority      = (known after apply)
    + cluster_id                = (known after apply)
    + created_at                = (known after apply)
    + deletion_protection       = (known after apply)
    + endpoint                 = (known after apply)
    + id                      = (known after apply)
    + identity                 = (known after apply)
    + name                    = "cicd-cluster"
    + platform_version         = (known after apply)
    + region                  = "eu-north-1"
    + role_arn                = (known after apply)
    + status                  = (known after apply)
    + tags_all                = (known after apply)
    + version                 = (known after apply)

    + access_config (known after apply)
    + compute_config (known after apply)
    + kubernetes_network_config (known after apply)
}
```

```
ubuntu@ip-172-31-37-221:~/sample-java-app/terraform-cicd$ terraform apply -auto-approve
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_eks_cluster.cicd_cluster will be created
+ resource "aws_eks_cluster" "cicd_cluster" {
    + arn          = (known after apply)
    + bootstrap_self_managed_addons = true
    + certificate_authority      = (known after apply)
    + cluster_id                = (known after apply)
    + created_at                = (known after apply)
    + deletion_protection       = (known after apply)
    + endpoint                 = (known after apply)
    + id                      = (known after apply)
    + identity                 = (known after apply)
    + name                    = "cicd-cluster"
    + platform_version         = (known after apply)
    + region                  = "eu-north-1"
    + role_arn                = (known after apply)
    + status                  = (known after apply)
    + tags_all                = (known after apply)
    + version                 = (known after apply)

    + access_config (known after apply)
}
```

---

### ◆ STEP 3: Configure Jenkins Server Using Ansible

#### 1. Install Ansible:

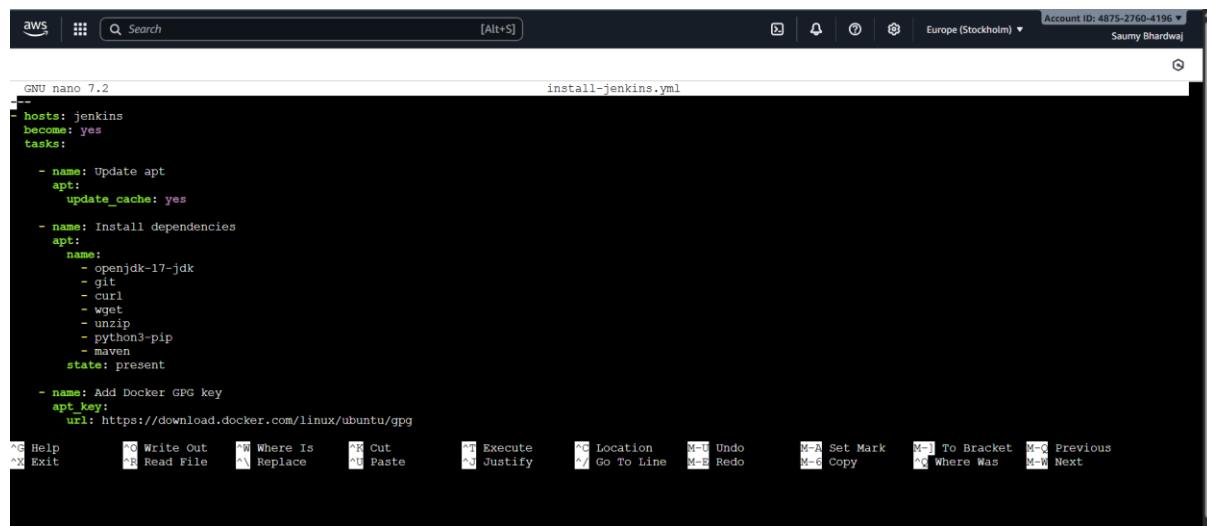
```
sudo apt install ansible
```

## 2. Create Inventory File with Jenkins EC2 IP.

```
ubuntu@ip-172-31-37-221:/sample-java-app/ansible$ ansible -i inventory.ini jenkins -m ping
[WARNING]: Platform linux on host 51.20.132.251 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python
interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.18/reference_appendices/interpreter_discovery.html for more
information.
51.20.132.251 | SUCCESS => [
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.10"
    },
    "changed": false,
    "ping": "pong"
```

## 3. Write Ansible Playbook to:

- Install Java, Jenkins, Docker, kubectl, and Maven
- Start and enable Jenkins service
- Add Jenkins user to Docker group

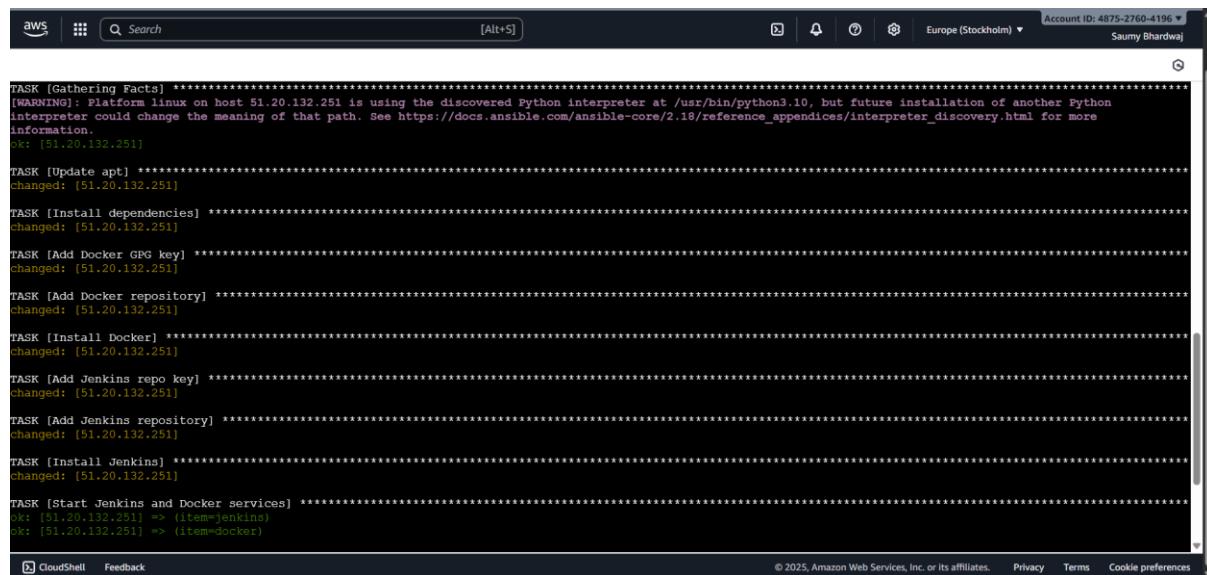


```
GNU nano 7.2                               install-jenkins.yml

hosts: jenkins
become: yes
tasks:
- name: Update apt
  apt:
    update_cache: yes
- name: Install dependencies
  apt:
    name:
      - openjdk-17-jdk
      - git
      - curl
      - wget
      - unzip
      - python3-pip
      - maven
    state: present
- name: Add Docker GPG key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
```

## 4. Run Playbook:

```
ansible-playbook install-jenkins.yml -i inventory
```



```
TASK [Gathering Facts] ****
[WARNING]: Platform linux on host 51.20.132.251 is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python
interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-core/2.18/reference_appendices/interpreter_discovery.html for more
information.
ok: [51.20.132.251]

TASK [Update apt] ****
changed: [51.20.132.251]

TASK [Install dependencies] ****
changed: [51.20.132.251]

TASK [Add Docker GPG key] ****
changed: [51.20.132.251]

TASK [Add Docker repository] ****
changed: [51.20.132.251]

TASK [Install Docker] ****
changed: [51.20.132.251]

TASK [Add Jenkins repo key] ****
changed: [51.20.132.251]

TASK [Add Jenkins repository] ****
changed: [51.20.132.251]

TASK [Install Jenkins] ****
changed: [51.20.132.251]

TASK [Start Jenkins and Docker services] ****
ok: [51.20.132.251] => (item=jenkins)
ok: [51.20.132.251] => (item=docker)
```

```

Learn more about enabling ESM Apps service at https://ubuntu.com/esm
New release '24.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

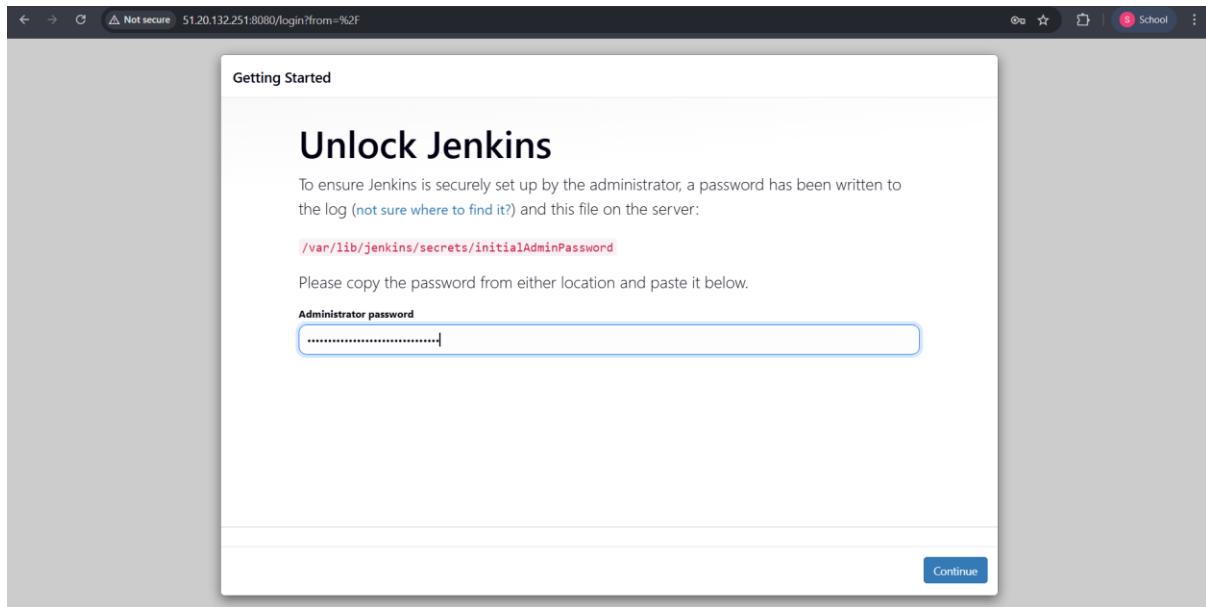
Last login: Mon Oct  6 15:01:33 2025 from 13.60.233.169
ubuntu@ip-10-0-1-88:~$ sudo systemctl status jenkins
sudo docker ps
java -version
mvn -v
kubectl version --client
aws --version
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2025-10-06 15:01:17 UTC; 57s ago
       PID: 9798 (java)
      Tasks: 43 (limit: 1073)
     Memory: 303.1M
        CPU: 20.134s
      CGroup: /system.slice/jenkins.service
              └─9798 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Oct 06 15:01:11 ip-10-0-1-88 jenkins[9798]: [I/F] > This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Oct 06 15:01:11 ip-10-0-1-88 jenkins[9798]: [I/F] ****
Oct 06 15:01:17 ip-10-0-1-88 jenkins[9798]: 2025-10-06 15:01:17.522+0000 [id=30]      INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
Oct 06 15:01:17 ip-10-0-1-88 jenkins[9798]: 2025-10-06 15:01:17.555+0000 [id=23]      INFO    hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running
Oct 06 15:01:18 ip-10-0-1-88 jenkins[9798]: 2025-10-06 15:01:18.635+0000 [id=49]      INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file
Oct 06 15:01:18 ip-10-0-1-88 jenkins[9798]: 2025-10-06 15:01:18.636+0000 [id=49]      INFO    hudson.util.Retriger#start: Performed the action check updates serv
lines 1-20/20 (END)

```

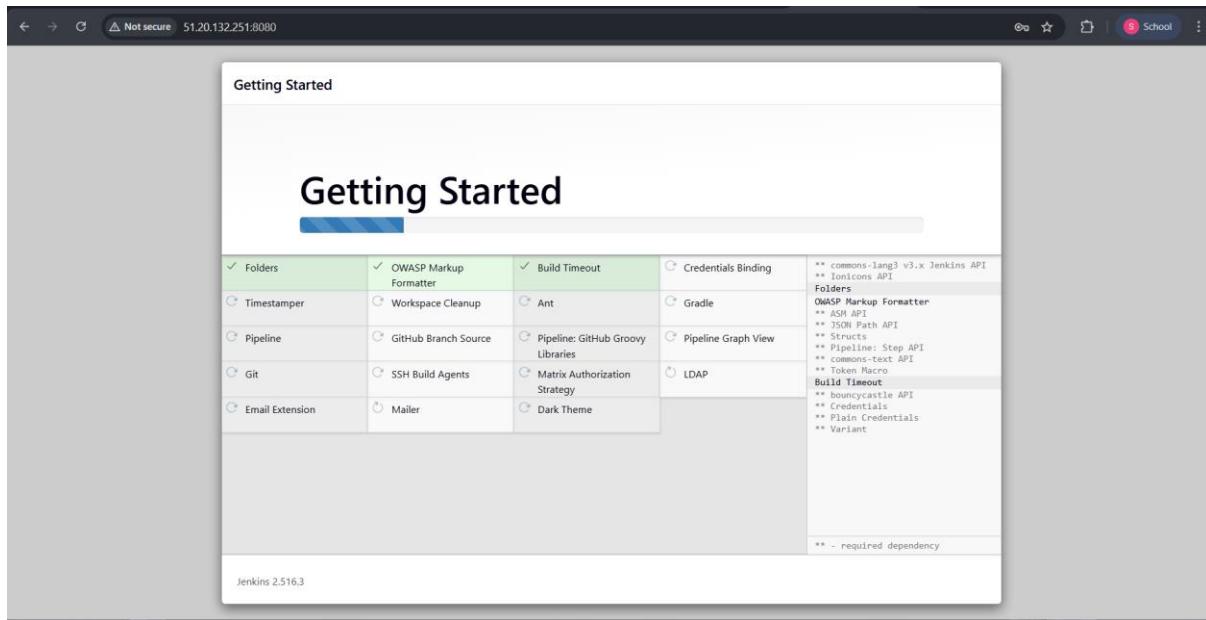
## STEP 4: Configure Jenkins for CI/CD

- Access Jenkins via browser: <http://<jenkins-ec2-ip>:8080>



- Install required plugins:

- GitHub, Maven Integration, Docker, Kubernetes CLI



### 3. Create Jenkins Credentials for:

- GitHub
- Docker Hub
- AWS IAM Access

### 4. Create a **new Jenkins Pipeline Job**.

#### ◆ STEP 5: Write Jenkinsfile for CI/CD Pipeline

**Jenkinsfile example:**

groovy

```

pipeline {
    agent any

    tools {
        maven 'Maven 3.8.1'
    }

    environment {
        DOCKER_IMAGE = "yourdockerhubusername/sample-java-app:${BUILD_NUMBER}"
    }

    stages {
        stage('Checkout Code') {
            steps {
                git 'https://github.com/your-username/sample-java-app.git'
            }
        }
    }
}

```

```

    }
    stage('Build with Maven') {
        steps {
            sh 'mvn clean package'
        }
    }
    stage('Build Docker Image') {
        steps {
            sh 'docker build -t $DOCKER_IMAGE .'
        }
    }
    stage('Push to Docker Hub') {
        steps {
            withDockerRegistry([credentialsId: 'docker-hub-creds']) {
                sh 'docker push $DOCKER_IMAGE'
            }
        }
    }
    stage('Deploy to Kubernetes') {
        steps {
            sh 'kubectl set image deployment/sample-app-deployment sample-
container=$DOCKER_IMAGE'
        }
    }
}

```

```

GNU nano 6.2
pipeline {
agent any
environment {
    AWS_REGION = "eu-north-1"
    AWS_ACCOUNT_ID = sh(returnStdout: true, script: "aws sts get-caller-identity --query Account --output text").trim()
    ECR_REPO = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/sample-java-app"
}
stages {
    stage('Checkout') {
        steps {
            git url: 'https://github.com/your-username/sample-java-app.git'
        }
    }
    stage('Build with Maven') {
        steps {
            sh 'mvn -B clean package'
        }
    }
    stage('Build & Push Docker Image') {
        steps {
            script {
                def tag = "${ECR_REPO}:${BUILD_NUMBER}"
            }
        }
    }
}

```

The terminal window also shows a standard nano editor status bar at the bottom with various keyboard shortcuts.

## ◆ STEP 6: Create Kubernetes Deployment Files

1. Write YAML files for:
  - o Deployment
  - o Service (NodePort/LoadBalancer)

```

deployment.yaml *
GNU nano 7.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
        - name: sample-container
          image: <AWS_ACCOUNT_ID>.dkr.ecr.eu-north-1.amazonaws.com/sample-java-app:latest
          ports:
            - containerPort: 8080

```

```

service.yaml
GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: sample-app-service
spec:
  selector:
    app: sample-app
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080

```

## 2. Deploy manually once:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

```
ubuntu@ip-10-0-1-88:~/sample-java-app$ kubectl apply -f k8s/deployment.yaml
deployment.apps/sample-app-deployment created
ubuntu@ip-10-0-1-88:~/sample-java-app$ kubectl apply -f k8s/service.yaml
service/sample-app-service created
ubuntu@ip-10-0-1-88:~/sample-java-app$ kubectl get svc sample-app-service -w
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
sample-app-service   LoadBalancer   172.20.85.47   a7f74207c45e64ea7b2526bd7a8d3668-479627857.eu-north-1.elb.amazonaws.com   PORT(S)        AGE
                                                               80:31734/TCP   8s
```

## ◆ STEP 7: Verify CI/CD Pipeline Execution

1. Make a code change in GitHub (e.g., update a string in the Java app).
2. Jenkins will:
  - Pull code
  - Build with Maven
  - Build and push Docker image
  - Deploy new version to Kubernetes
3. Access the app via:

<http://<LoadBalancer-DNS or NodePort-IP>>

---

 **Expected Outcome:**

- Fully functional CI/CD pipeline
- Automatic code build, Docker packaging, and deployment to Kubernetes on AWS

**Rubric (Total: 6 Marks)**

Criteria	Excellent (1 Marks)	Good (0.5 Mark)	Needs Improvement (0 Mark)
<b>1. Infrastructure Provisioning (Terraform)</b>	Fully functional AWS infra with EC2 & EKS	Partially functional infra, minor issues	Infrastructure not provisioned or fails entirely
<b>2. Configuration Management (Ansible)</b>	Jenkins and dependencies correctly installed	Partial setup; minor misconfigurations	Jenkins not installed or major errors
<b>3. CI/CD Pipeline in Jenkins</b>	Pipeline works end-to-end with all stages	Pipeline executes partially; minor stage issues	Pipeline fails or is not configured
<b>4. Dockerization &amp; Image Management</b>	Docker image built and pushed to Docker Hub	Image built but push or tag missing	Docker image not built or pushed
<b>5. Kubernetes Deployment</b>	Application deployed and accessible via service	App deployed but not accessible properly	Deployment failed or not attempted
<b>6. Version Control Integration (GitHub)</b>	GitHub repo connected and automated builds work	GitHub integrated but manual triggers used	No GitHub integration or broken webhook