Project Report: Deploying a Maven Web App using Jenkins, Docker, and Kubernetes on AWS EKS

Project Overview

This project automates the deployment of a Maven-based Java web application using a CI/CD pipeline. The pipeline includes:

- Fetching the source code from GitHub.
- Building the project using Maven.
- Creating a Docker image and pushing it to Docker Hub.
- Deploying the containerized application to an AWS EKS cluster using Kubernetes manifests.
- Infrastructure as Code (IaC) using Terraform for setting up AWS resources.

---

1. Infrastructure Setup using Terraform

The Terraform scripts provision an AWS EKS cluster along with required networking and compute resources.

Terraform Files:

- **provider.tf** - Configures the AWS provider.
- **main.tf** - Defines the EKS cluster, node groups, and networking.
- **input-vars.tf** - Specifies input variables for Terraform configuration.
- **output-vars.tf** - Defines the output variables.
- **vars.tf** - Stores variable values.

---

2. Jenkins Server Setup

Jenkins is installed and configured to automate the CI/CD pipeline. The following script provisions an Ubuntu-based Jenkins server.

Jenkins Setup Script:

- **script.groovy** - Contains the Jenkins pipeline script written in Groovy.

---

3. Application Deployment Workflow

Jenkins Pipeline Configuration (Groovy Script)

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = "darin04/mavenwebapp"
        DOCKER_TAG = "latest"
        EKS_CLUSTER_NAME = "darin-cluster"
        KUBE_CONFIG = "/var/lib/jenkins/.kube/config"
    }

    stages {
        stage('Checkout Code Only') {
            steps {
                git branch: 'master', url:
'https://github.com/suffixscope/maven-web-app.git'
                sh 'rm -f k8s-deploy.yml'
            }
        }

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

        stage('Build Docker Image (Using Local Dockerfile)') {
            steps {
                sh 'docker build -t $DOCKER_IMAGE:$DOCKER_TAG -f
/var/lib/jenkins/workspace/k8s/Docker/Dockerfile .'
            }
        }

        stage('Push Docker Image') {
            steps {
                withDockerRegistry([credentialsId: 'docker-hub-
credentials', url: '']) {
                    sh 'docker push $DOCKER_IMAGE:$DOCKER_TAG'
                }
            }
        }

        stage('Deploy to Kubernetes (Using Local Deployment YAML)') {
            steps {
                sh "kubectl apply -f
/var/lib/jenkins/workspace/k8s/deployment.yaml --kubeconfig=$KUBE_CONFIG"
            }
        }
    }

    post {
        success {
            echo "Deployment Successful!"
        }
        failure {
            echo "Deployment Failed!"
        }
    }
}
```

4. Dockerfile for Containerization

The application is packaged inside a Docker container using the following Dockerfile:

Dockerfile:

```
FROM tomcat:latest
MAINTAINER Vinod <vinod@scopeindia.org>
EXPOSE 8080
COPY target/maven-web-app.war /usr/local/tomcat/webapps/maven-web-app.war
```

---

5. Kubernetes Deployment Configuration

Kubernetes is used to deploy and manage the application inside an AWS EKS cluster.

Deployment and Service YAML Files:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mavenwebappdeployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: mavenwebapp
  template:
    metadata:
      labels:
        app: mavenwebapp
    spec:
      containers:
      - name: mavenwebappcontainer
        image: darin04/mavenwebapp:latest
        ports:
        - containerPort: 8080
        imagePullPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: mavenwebappsvc
spec:
  selector:
    app: mavenwebapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: LoadBalancer
```

---

6. Project Workflow Summary

1. **Infrastructure Setup:** Terraform provisions AWS resources.
2. **Jenkins CI/CD Pipeline:** Automates code retrieval, build, containerization, and deployment.
3. **Docker & Kubernetes:** Manages application packaging and orchestration on AWS EKS.

This project ensures a streamlined and automated deployment process for a Java web application using modern DevOps practices.