CI/CD Pipeline Deployment with Jenkins & Kubernetes using Kubeadm

CI/CD Pipeline Project with Jenkins and Kubernetes

Project Author: Prasad Pawar

GitHub: https://github.com/prasadpawar252/SprintBootService-1

Docker Image: prasadpawar2522/spring:1.0

1. Project Overview

In this project, we are creating a CI/CD pipeline using Jenkins and Kubernetes. The pipeline will build a Spring Boot project, create a Docker image, push it to Docker Hub, and finally deploy it to a Kubernetes cluster. We use Jenkins for automation, Docker for containerization, and Kubernetes (via kubeadm) for orchestration.

2. Infrastructure

- Jenkins Master: t2.micro EC2 instance (for Jenkins UI and control)

- Jenkins Slave + Kubernetes Master: t2.medium (for running builds and managing Kubernetes)

- Kubernetes Worker Node: t2.medium (for running deployed containers)

3. Jenkinsfile Pipeline Summary

The Jenkinsfile has the following stages:

1. Checkout: Pulls code from GitHub

2. Build Maven: Builds JAR using Maven

3. Build Docker Image: Creates Docker image with the app

4. Push Docker Image: Pushes image to Docker Hub

5. Deploy to Kubernetes: Deploys the app using deploy.yaml

4. How to Set Up

Step 1: Install Jenkins on Jenkins Master instance.

Step 2: Install Docker, Java, Maven on Jenkins Slave (also acts as Kubernetes Master).

Step 3: Initialize Kubernetes using kubeadm.

Step 4: Join Worker Node to the cluster.

Step 5: Set up Jenkins credentials and configure a pipeline job with your Jenkinsfile.

5. Final Output

- The application gets deployed in Kubernetes.

- You can check pods with: kubectl get pods

- Check services: kubectl get svc

- Open your app: http://<Worker_Node_IP>:<NodePort>/getData

6. Summary

This project demonstrates how to set up a simple yet powerful CI/CD pipeline. It covers integration of Jenkins, GitHub, Docker, and Kubernetes, making it a great foundation for real-world DevOps workflows.