End-to-End Corporate CI/CD Pipeline with Kubernetes Deployment

Project Report

---

1. Introduction

This project implements a robust, end-to-end CI/CD pipeline designed to emulate corporate DevOps practices. The objective was to automate the entire software delivery lifecycle—from code commit to production deployment—incorporating industry-standard tools for code quality, security, and container orchestration. The pipeline builds, tests, scans, and deploys a containerized web application to a multi-node Kubernetes cluster, ensuring reliability, security, and efficiency.

2. Abstract

The pipeline was built to demonstrate mastery of modern DevOps principles. A sample web application was containerized using Docker. The core automation was achieved by integrating multiple tools: SonarQube for static code analysis, Trivy for vulnerability scanning of the Docker image, and Nexus Repository Manager for artifact storage. The final stage involved deploying the application to a highly available Kubernetes cluster (comprising one master and two worker nodes) provisioned on virtual machines. The pipeline concludes with an email notification, providing visibility into the deployment status. This project showcases a practical implementation of Infrastructure as Code (IaC), Continuous Integration, Continuous Deployment, and DevSecOps.

3. Tools & Technologies Used

Category	Tools

Version Control	GitHub
CI/CD Server	Jenkins
Containerization	Docker, Docker Hub
Code Quality Analysis	SonarQube
Vulnerability Scanner	Trivy
Artifact Repository	Sonatype Nexus
Container Orchestration	Kubernetes (kubeadm)
Infrastructure	VirtualBox/VMware (4 VMs: Jenkins, Master, Worker-1, Worker-2)
Notification	SMTP (Email)
Scripting	Shell, Groovy (Jenkinsfile)

4. Steps Involved in Building the Project

4.1. Infrastructure & Environment Setup

1. VM Provisioning: Four Ubuntu virtual machines were provisioned using VirtualBox.
2. Kubernetes Cluster Setup: Initialized a Kubernetes cluster using `kubeadm` on the master node. The two worker nodes were joined to the cluster to create a distributed environment.
3. Networking: Configured the Weave Net pod network add-on for internal cluster communication.
4. Jenkins Setup: Installed and configured Jenkins on a dedicated server VM.

4.2. Pipeline Configuration (Jenkinsfile)

The pipeline was defined as a Jenkins declarative pipeline, consisting of the following sequential stages:

Figure 1: Pipeline Architecture Diagram

```
<p align="center"> <img src="Architecture Overview diagram/Screenshot 2025-09-07 163326.png" alt="CI/CD Pipeline Architecture" width="500"/> </p>
```

Page 2:

5. Pipeline Stages Breakdown

1. Stage 1: Code Checkout
    - The pipeline is triggered on a code commit to the main GitHub branch.
    - The source code is checked out from the repository.
2. Stage 2: SonarQube Analysis
    - The code is analyzed by SonarQube for bugs, vulnerabilities, and code smells.
    - The build can be set to fail based on quality gate thresholds, enforcing code quality.
3. Stage 3: Docker Build
    - A Docker image is built from the application's Dockerfile, tagging it with the build ID for traceability.
4. Stage 4: Vulnerability Scan with Trivy
    - The newly built Docker image is scanned by Trivy for known Common Vulnerabilities and Exposures (CVEs).
    - Critical vulnerabilities can be configured to fail the build, preventing insecure images from progressing.
5. Stage 5: Push to Nexus
    - Upon passing all scans, the Docker image is tagged as `RELEASE` and pushed to a private repository in Nexus Repository Manager.
    - This serves as the definitive, versioned artifact store.
6. Stage 6: Deploy to Kubernetes
    - The pipeline authenticates with the Kubernetes cluster and applies the deployment manifests (YAML files) via `kubectl`.

    - These manifests pull the approved image from Nexus and deploy it to the cluster, ensuring a consistent environment.
7. Stage 7: Email Notification
    - A post-build action sends an email to the development and operations teams with the build status (Success/Failure), providing immediate feedback.

6. Challenges & Solutions

- Challenge: Configuring network plugins (e.g., Weave Net) for pod networking in the Kubernetes cluster.
    - Solution: Referenced official Kubernetes documentation and community guides to properly apply the CNI configuration.
- Challenge: Ensuring Jenkins agents had the necessary tools (Docker, kubectl, Trivy) installed.
    - Solution: Used custom Jenkins agent images or installed the tools directly on the VMs for simplicity.
- Challenge: Configuring role-based access control (RBAC) in Kubernetes for the Jenkins service account to allow deployments.
    - Solution: Created a dedicated service account with minimal necessary permissions (get, list, create) in the target namespace.

7. Conclusion

This project successfully demonstrates the implementation of a secure, automated, and corporate-standard CI/CD pipeline. It integrates critical DevSecOps practices by shifting security left, embedding code quality and vulnerability checks early in the development process. The automated deployment to a multi-node Kubernetes cluster highlights the ability to manage containerized applications at scale. The skills honed in this project—including containerization, orchestration, automation, and security scanning—are directly applicable to modern cloud-native development environments and provide a strong foundation for a career in DevOps.