**CI/CD Pipeline Implementation Report**

**Project:** Containerized Web App Deployment to Kubernetes via Jenkins

**🔧 Objective:**

To design and implement a CI/CD pipeline that automatically builds, tests, and deploys a Dockerized web application to a Kubernetes cluster when code is pushed to the develop branch on GitHub.

**Tools Used:**

* **GitHub** – Source code hosting and version control
* **Jenkins** – CI/CD automation server
* **Docker & DockerHub** – Containerization and image registry
* **Kubernetes (self-managed)** – Deployment platform
* **kubectl** – CLI tool for Kubernetes
* **EC2 Instances** – Jenkins, Docker build, Kubernetes cluster

**Infrastructure Setup:**

* **EC2-1:** Jenkins Server
* **EC2-2:** Docker Build Node (optional if Jenkins runs Docker)
* **EC2-3+:** Kubernetes Master & Worker Nodes (self-hosted)

**Project Structure:**

* Dockerfile: Defines image using nginx
* Jenkinsfile: Contains pipeline stages
* deployment.yaml: Kubernetes deployment definition

**Jenkins Credentials Added:**

* GitHub Personal Access Token
* DockerHub Username & Password
* kubeconfig file for Kubernetes access (added as a Jenkins secret file)

**Pipeline Flow (Jenkinsfile):**

1. **Clone Repo** – Jenkins pulls code from GitHub on develop branch push
2. **Build Docker Image** – Image is created using Dockerfile
3. **Push to DockerHub** – Image pushed using Docker credentials
4. **Deploy to Kubernetes** – Jenkins uses kubectl and kubeconfig to update the image in the deployment

**Testing & Verification:**

* Jenkins pipeline tested with a manual trigger first
* Kubernetes pods validated using kubectl get pods and kubectl rollout status
* Automatic GitHub webhook confirmed working on code push to develop

**Key Commands Used:**

bash

CopyEdit

git init

git remote add origin <repo-url>

git push -u origin develop

kubectl set image ...

kubectl rollout status ...

**Outcomes:**

* Pipeline is fully functional and auto-deploys new builds to Kubernetes
* Docker image builds and pushes correctly to DockerHub
* No errors encountered in deployment phase with kubeconfig and container naming correctly configured
* GitHub → Jenkins → DockerHub → Kubernetes chain verified end-to-end

**Troubleshooting strategies:**

| **Issue** | **Strategy** |
| --- | --- |
| Build fails | Check npm ci or docker build logs. Validate Dockerfile. |
| Secrets not working | Ensure secrets are set in GitHub repo → Settings → Secrets |
| Deployment fails | Check Kubernetes/ECS logs (kubectl logs, kubectl describe) |
| Rollback required | Use kubectl rollout undo deployment/webapp -n staging |
| CI/CD stuck | Look for stuck GitHub Actions runners. Re-run the job manually. |
| Image not pulling | Ensure registry auth is correct and image is public or accessible |

**Conclusion:**

This architecture ensures faster iterations, repeatable deployments, and simplified Kubernetes operations. The use of Jenkins with DockerHub and custom kubeconfig proved cost-effective and flexible for a non-EKS setup. Future improvements could include adding Helm, Blue/Green deployments, or automated testing stages. Although end result was unachieved due to pricing on AWS instance types t2.medium, which requires better config for kubernetes. The CI/CD pipeline setup successfully automates the containerized app deployment process using open-source tools