# Kubernetes GitOps Deployment with CI/CD Automation – Report

## Abstract:

Modern software systems demand automated, reliable, and observable deployment pipelines. Manual deployments are often slow, error-prone, and risky in production environments. This project demonstrates a **GitOps-powered CI/CD pipeline** for a Notes Application using **GitHub Actions, Jenkins, ArgoCD, Kubernetes, Docker, Prometheus, and Grafana**.

The pipeline enables **zero-touch deployment**, **security-first CI**, and **continuous monitoring**, ensuring updates are delivered safely, efficiently, and in a production-grade manner.

## Introduction:

Deploying applications manually or without a robust pipeline increases risk and slows delivery. **GitOps**, combined with CI/CD automation, ensures that the **cluster state always reflects the Git repository**, enabling safe, predictable, and observable deployments.

In this project:

- **Frontend (React)** and **Backend (Node.js)** services are containerized.
- **CI pipelines** perform security scans, version checks, and Docker builds.
- **CD pipelines** deploy updated manifests to the Kubernetes cluster using **Jenkins and ArgoCD**, maintaining desired state and enabling rollback if necessary.

This setup simulates **real-world production workflows** for cloud-native applications, demonstrating automation, reliability, and observability.

## Tools Used:

- **Kubernetes (MicroK8s / K3s / Production Cluster):** Cluster environment for deploying and managing workloads
- **Docker:** Containerization of frontend and backend applications

- **GitHub Actions:** CI automation, including build, test, and security checks
- **Trivy & CodeQL:** Security and static code analysis
- **Jenkins:** Automates deployment tasks and updates Kubernetes manifests
- **ArgoCD:** GitOps operator for continuous deployment, manifest syncing, and rollback support
- **Prometheus & Grafana:** Observability, metrics collection, and dashboard visualization

## Steps Involved in Building the Project:

**1. Application Development**

- Developed a **Notes Application** with React (frontend) and Node.js (backend)
- Maintained **version.txt** in each module to track Docker image updates

**2. CI Pipeline (GitHub Actions)**

- Triggered on code push to frontend/ or backend/
- Performed **security scans**: Trivy (container vulnerabilities) and CodeQL (static analysis)
- Checked version.txt to determine if Docker images required rebuilding
- Built and pushed Docker images to the registry with semantic tagging

**3. CD Pipeline (Jenkins)**

- Triggered via GitHub Actions webhook
- Pulled the latest Docker images
- Updated Kubernetes manifests and deployed changes to the cluster

**4. GitOps Deployment (ArgoCD)**

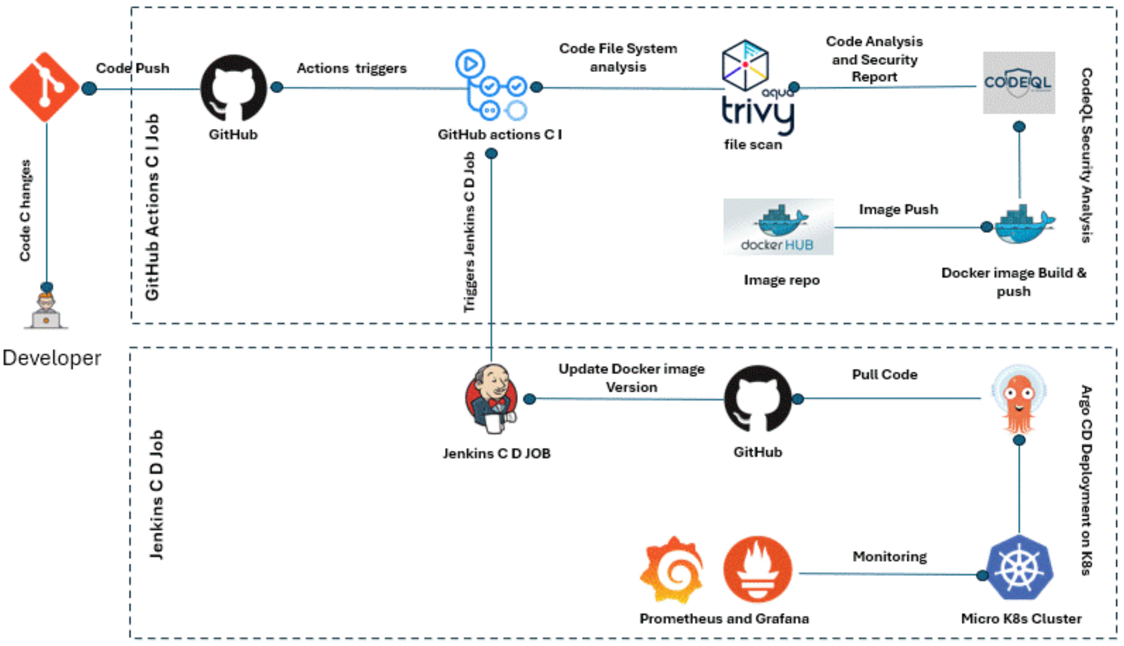
- Continuously monitored the Git repository for manifest changes
- Auto-synced updates to the Kubernetes cluster
- Maintained desired state and supported rollbacks when required

**5. Observability & Monitoring**

- **Prometheus** collected metrics from the applications and cluster

- **Grafana dashboards** visualized CI/CD workflow, app health, and metrics in real-time

## Workflow Diagram:



## Conclusion:

The **Kubernetes GitOps Deployment with CI/CD Automation** project demonstrates a **production-ready pipeline** for deploying applications with automation, security, and observability.

By leveraging **GitOps, CI/CD, and containerization**, updates are applied safely, can be rolled back if necessary, and are fully observable. This approach ensures **faster, reliable, and secure deployments**, reflecting modern **cloud-native best practices**.