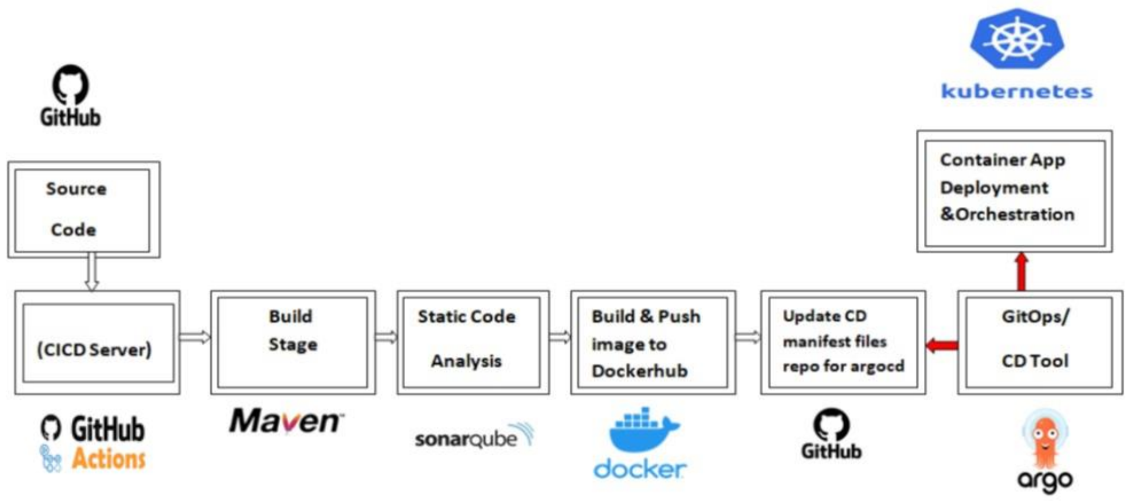
CI/CD Pipeline using GitHub Actions, Docker, and ArgoCD for Kubernetes

Overview

This project implements a **complete CI/CD pipeline** to automate application build, quality analysis, containerization, and deployment on Kubernetes using **GitHub Actions**, **SonarQube**, **DockerHub**, and **ArgoCD**.

The entire flow ensures faster delivery, automated testing, and continuous deployment aligned with **GitOps principles**.



Architecture Diagram Explanation

Source Code (GitHub Repository)

- The application source code resides in **GitHub**.
- The repository also contains:
  - Dockerfile
  - Kubernetes manifest files (YAMLs)
  - GitHub Actions workflow file (.github/workflows/ci-cd.yml)
- Any **code push or PR merge** triggers the CI/CD pipeline.

CI/CD Server (GitHub Actions)

- Acts as the automation server for Continuous Integration (CI) and Continuous Deployment (CD).
- Workflow YAML defines multiple stages: build, test, analysis, dockerize, and deploy.

Example trigger:

```
on:
  push:
    branches:
      - main
```

Build Stage (Maven)

- Maven compiles and builds the Java application.
- Generates the .jar or .war artifact used later for Docker image creation.

Sample command:

```
- name: Build with Maven
  run: mvn clean package -DskipTests
```

Static Code Analysis (SonarQube)

- Ensures code quality and detects vulnerabilities.
- GitHub Actions connects to a **SonarQube server** using authentication tokens.

Example snippet:

```
- name: SonarQube Scan
  run: mvn sonar:sonar -Dsonar.projectKey=demo-app -Dsonar.host.url=http://<sonar-url> -Dsonar.login=${{ secrets.SONAR_TOKEN }}
```

Build & Push Docker Image (DockerHub)

- Docker builds the container image of the application using the Dockerfile.
- The image is tagged and pushed to **DockerHub**.

Example:

```
- name: Build Docker image
  run: docker build -t ${{ secrets.DOCKER_USERNAME }}/demo-app:${{ github.run_number }} .

- name: Push Docker image
  run: |
    echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin
    docker push ${{ secrets.DOCKER_USERNAME }}/demo-app:${{ github.run_number }}
```

Update CD Manifest Files (GitHub Repo for ArgoCD)

- After pushing the new Docker image, the workflow updates the **Kubernetes deployment YAML** file with the **new image tag**.
- Commits this change to the **GitOps repository**, which ArgoCD watches.

Example:

```
- name: Update manifest
  run: |
    sed -i "s|image:.*|image: ${{ secrets.DOCKER_USERNAME }}/demo-app:${{ github.run_number }}|" k8s/deployment.yaml
    git config user.name "github-actions"
    git config user.email "actions@github.com"
    git commit -am "Updated image to new version"
    git push
```

GitOps / CD Tool (ArgoCD)

- ArgoCD continuously monitors the **CD manifest repository**.
- When the YAML file changes (new image tag), ArgoCD automatically deploys the new version to the Kubernetes cluster.

ArgoCD Workflow:

1. Watch for changes in the repo.
2. Sync Kubernetes cluster state to match the desired state (updated manifests).
3. Trigger rolling updates.

Complete Project Repo: https://github.com/jadalaramani/argocd_github_actions_project.git

Container App Deployment & Orchestration (Kubernetes)

- Kubernetes deploys and manages the containerized application.
- Handles rolling updates, load balancing, and auto-healing.
- Users can access the app via the **service endpoint** or **Ingress**.

Implementation Steps for Your Repo

Step 1: Clone Repository

```
git clone https://github.com/jadalaramani/argocd_github_actions_project.git
```

cd argocd_github_actions_project

Step 2: Configure GitHub Secrets

In your GitHub repository → **Settings** → **Secrets** → **Actions**, add:

- DOCKER_USERNAME
- DOCKER_PASSWORD
- SONAR_TOKEN
- K8S_MANIFEST_REPO (if using a separate repo for manifests)

Step 3: SonarQube Setup

- Run SonarQube locally or use a hosted instance.
- Generate a token under *My Account → Security*.
- Store the token as SONAR_TOKEN in GitHub secrets.

Step 4: DockerHub Setup

- Create a DockerHub account and repository.
- Add credentials in GitHub secrets for automation.

Step 5: ArgoCD Setup

- Install ArgoCD on Kubernetes:
- kubectl create namespace argocd
- kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
- Login and connect your **manifest repo**:
- argocd repo add https://github.com/jadalaramani/argocd_github_actions_project.git --username <your-username> --password <token>
- Create application:
- argocd app create demo-app \
-   --repo https://github.com/jadalaramani/argocd_github_actions_project.git \
-   --path k8s \
-   --dest-server https://kubernetes.default.svc \
-   --dest-namespace default

Step 6: GitHub Actions Workflow

- The CI/CD workflow file .github/workflows/ci-cd.yml runs on every push.
- It automates:
    - Build → SonarQube → Docker Push → Manifest Update → ArgoCD Sync

Step 7: Access Application

- Once deployed, check pod and service status:
- kubectl get pods
- kubectl get svc
- Access app via LoadBalancer or NodePort.

Summary of Tools Used

Stage	**Tool**	**Purpose**
Source Control	GitHub	Version Control & CI/CD trigger
Build	Maven	Compile & package app
Code Quality	SonarQube	Static analysis
Containerization	Docker	Build & push images
CI/CD	GitHub Actions	Pipeline orchestration
Deployment	ArgoCD	GitOps-based CD
Orchestration	Kubernetes	App deployment & management

Conclusion

This project demonstrates a **modern DevOps CI/CD workflow** integrating:

- **Continuous Integration** (GitHub Actions, Maven, SonarQube)
- **Continuous Delivery** (ArgoCD, Kubernetes)
- **GitOps principles** ensuring version-controlled deployments.

It is a production-ready architecture suitable for microservice-based cloud applications.