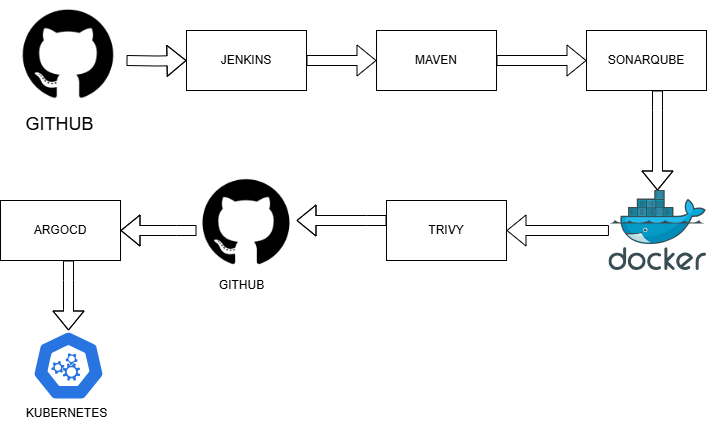
**Case Study-1:**

**Problem Statement:** You have been asked to set up a multi-environment **declarative/yaml based** deployment pipeline for a web application running on Elastic Kubernetes Service (EKS). A company controls the code for an application in **source code repository**. Containerized the web application using Docker and captured vulnerabilities by **image scanning** before pushing to registry. The application needs to be deployed **automatically to multiple EKS** environments (dev, qa, prod etc.) whenever changes are made to the respective branch code. Changes occur frequently every day, so the pipeline must be as responsive as possible. Prod pipeline must wait for **manual approval** before deployment. You are expected to integrate various DevOps/DevSecOps tools to perform given activities - **build management, test coverage, code coverage** etc and execute code and test coverage stages in **parallel manner**. Also trigger **notification** if coverage is less than 80% or any critical observation found and **terminate** the pipeline.

BLOCK DIAGRAM:



STAGES:

1. Clone Repository – Fetch the latest source code from GitHub using secure authentication, to the Jenkins workspace.

2. Build – Compile the application source code, resolve dependencies, and generate artifacts, ensuring proper packaging for execution.

3. Test – Check if the code works properly by running different tests to find errors or security problems.

4. Quality Gate – Perform static code analysis using tools like SonarQube to enforce security and coding standards.

5. Docker Build – The application is containerized by creating a Docker image with defined dependencies and environment configurations.

6. Image Scan – Use vulnerability scanning tools like Trivy to identify security risks within the built Docker image before deployment.

7. Docker Push – Push the validated Docker image to a DockerHub container registry for storage and further use.

8. Update GitHub Repo – Modify Kubernetes configuration files (deployment.yaml) to reflect the updated Docker image, enabling ArgoCD to sync the latest deployment version.

9. Deploy to Kubernetes using ArgoCD

Dev Environment – Perform automated deployment to the Dev cluster for verification, stability testing, and security validation.

QA Environment – Perform automated deployment to the QA cluster for verification, stability testing, and security validation.

Production with Manual Approval – Gate production deployment with an approval workflow to ensure compliance and risk mitigation before releasing updates.

SCREENSHOTS:

