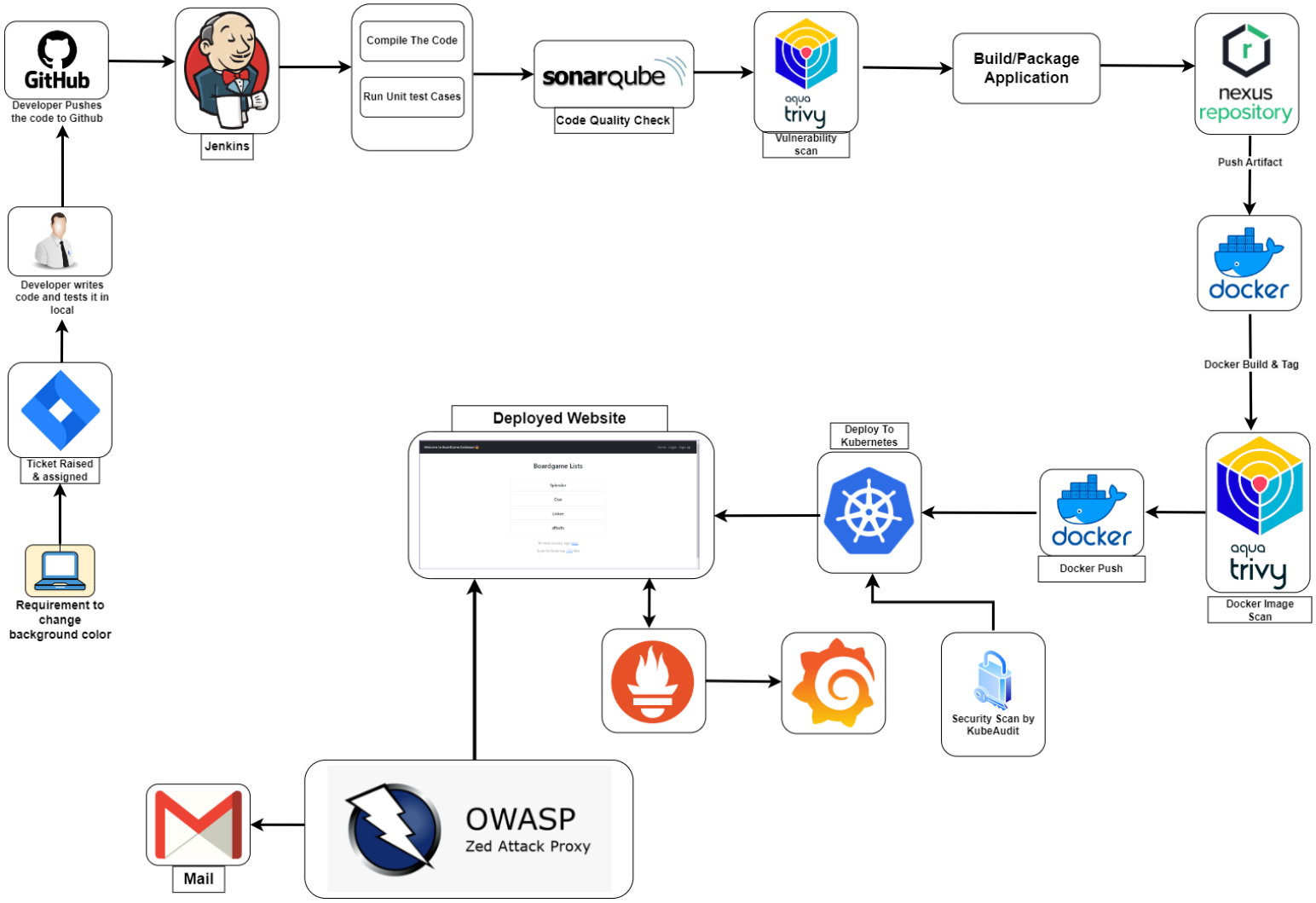
DevOps Shack

Real-Time Corporate CICD Pipeline Stages

[Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps]



1. Git Checkout

This is the first step in the CI/CD pipeline where the source code is fetched from a Git repository. This stage ensures that the latest codebase is used for each build.

2. Install Dependencies

The build environment sets up all the necessary dependencies required by the application. This may include libraries, frameworks, and other tools specified in the project's configuration files (like `package.json` for Node.js or `pom.xml` for Maven projects).

3. Run Test Cases

In this stage, automated tests are executed to ensure that the new changes do not break the application. This can include unit tests, integration tests, and other automated testing frameworks.

4. Sonar Analysis

SonarQube analysis is performed to assess the quality of the code. It checks for bugs, vulnerabilities, and code smells, and ensures the code adheres to predefined quality standards.

5. Trivy FS Scan

Trivy file system scan is used to detect vulnerabilities in project dependencies, ensuring there are no known security flaws in the libraries and packages used by the application.

6. Build Application

The application is compiled or packaged into a deployable format. For example, Java applications might be packaged into WAR or JAR files.

7. Push Artifact to Nexus

The built artifact (like a JAR or WAR file) is then uploaded to a repository manager like Nexus, which serves as a storage point for all the versioned artifacts.

8. Build & Tag Docker Image

A Docker image of the application is created and tagged appropriately. This image includes the application and its environment, ensuring consistency across different deployment environments.

9. Scan Docker Image Using Trivy

The Docker image is scanned for vulnerabilities using Trivy, a tool that detects vulnerabilities in container images. This ensures the Docker image is secure before it is deployed.

10. Push Docker Image

The secure Docker image is then pushed to a Docker registry, such as Docker Hub or an internal registry, making it available for deployment.

11. Deploy Application to Kubernetes (K8s)

The Docker image is deployed to a Kubernetes cluster. Kubernetes orchestrates the containerized applications, managing their lifecycle, scaling, and failover.

12. Perform Penetration Testing Using OWASP ZAP

OWASP ZAP is used to perform penetration testing on the deployed application. This helps identify security vulnerabilities that might be exploited by attackers.

13. Send Mail Notifications

Finally, email notifications are sent to the team or stakeholders to inform them of the build and deployment status, along with any issues or test failures.

Each of these stages is critical for maintaining a robust, efficient, and secure CI/CD pipeline. Proper documentation of these stages aids in troubleshooting, onboarding new developers, and ensuring compliance with development best practices.

Pipeline

```
pipeline {
    agent any

    environment {
```

```
        NEXUS_VERSION = '1.0.0'
        DOCKER_IMAGE = 'myapp:latest'
        DOCKER_REGISTRY = 'dockerhub_username/docker_repository'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git 'https://github.com/your-username/your-repo.git'
            }
        }

        stage('Install Dependencies') {
            steps {
                script {
                    // Commands to install dependencies based on your
project type
                    sh 'npm install' // For Node.js projects
                }
            }
        }

        stage('Run Test Cases') {
            steps {
                script {
                    sh 'npm test' // Run tests, assuming a Node.js project
                }
            }
        }

        stage('Sonar Analysis') {
            steps {
                script {
                    // Make sure SonarQube scanner is configured in Jenkins
                    sh 'sonar-scanner'
                }
            }
        }

        stage('Trivy FS Scan') {
            steps {
                script {
                    sh 'trivy filesystem ./'
                }
            }
        }

        stage('Build Application') {
            steps {
                script {
                    sh 'npm build' // Assuming a build script is defined in
package.json
                }
            }
        }

        stage('Push Artifact to Nexus') {
            steps {
                script {
                    // Make sure Nexus credentials and repository are
configured in Jenkins
```

```
                    sh "curl -u nexus-username:nexus-password -T
./build/app.jar http://nexus_repository_url/repository/maven-
releases/${NEXUS_VERSION}/app-${NEXUS_VERSION}.jar"
                }
            }
        }

        stage('Build & Tag Docker Image') {
            steps {
                script {
                    sh "docker build -t ${DOCKER_IMAGE} ."
                    sh "docker tag ${DOCKER_IMAGE}
${DOCKER_REGISTRY}:${NEXUS_VERSION}"
                }
            }
        }

        stage('Scan Docker Image Using Trivy') {
            steps {
                script {
                    sh "trivy image ${DOCKER_IMAGE}"
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry('https://index.docker.io/v1/',
'docker-credentials-id') {
                        sh "docker push
${DOCKER_REGISTRY}:${NEXUS_VERSION}"
                    }
                }
            }
        }

        stage('Deploy Application to Kubernetes') {
            steps {
                script {
                    sh "kubectl apply -f k8s/deployment.yaml"
                }
            }
        }

        stage('Perform Penetration Testing Using OWASP ZAP') {
            steps {
                script {
                    sh "zap-cli start"
                    sh "zap-cli open-url http://your-application-url"
                    sh "zap-cli active-scan --recursive http://your-
application-url"
                    sh "zap-cli report -o zap-report.html -f html"
                }
            }
        }

        stage('Send Mail Notifications') {
            steps {
```

```
                mail bcc: '', body: "Pipeline execution is complete. Please
check the build and deployment status.", cc: '', from: '', replyTo: '',
subject: 'Build and Deployment Notification', to: 'your-email@example.com'
            }
        }
    }
}
```

Key Points:

- **Environment Variables**: These store values like Docker image names, Nexus version numbers, etc., to be reused across the pipeline.
- **Script Blocks**: These are used to execute shell commands and scripts.
- **docker.withRegistry**: This is used to handle Docker registry authentication.
- **SonarQube, Trivy, and OWASP ZAP**: Ensure that Jenkins has appropriate plugins or configuration for these tools.

This Jenkinsfile should be customized according to your specific project requirements, including the correct repository URLs, Docker image names, and other configuration details. Be sure to have all the necessary plugins installed in Jenkins and credentials (like Nexus, Docker, and SonarQube) configured properly in Jenkins credentials store.