DevOps Shack

Real-Time Pipeline Stages for CICD

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

Stage 1: Install Required Tools

Description:

Ensure that all necessary tools and dependencies are installed on the CI/CD server.

Steps:

1. Install Java Development Kit (JDK)
2. Install Node.js
3. Install Apache Maven
4. Install other required tools as per project specifications.

Stage 2: Install Project Dependencies

Description:

Install project-specific dependencies using package managers like npm (for Node.js projects) or Maven (for Java projects).

Steps:

1. Install Node.js dependencies using npm install.
2. Install Java dependencies using Maven.

Stage 3: Execute Test Cases

Description:

Run automated test cases to ensure code functionality and integrity.

Steps:

1. Execute unit tests.
2. Execute integration tests.
3. Execute end-to-end tests if applicable.

Stage 4: Perform File System Scan (Trivy)

Description:

Scan project files for vulnerabilities and security issues using Trivy.

Steps:

1. Run Trivy scan on project files.
2. Generate report highlighting vulnerabilities found.

Stage 5: Perform SonarQube Analysis

Description:

Perform static code analysis and assess code quality using SonarQube.

Steps:

1. Run SonarQube analysis on project codebase.
2. Generate report with code quality metrics and code coverage.

Stage 6: Perform Quality Gate Check

Description:

Evaluate code quality against predefined quality gates to ensure adherence to quality standards.

Steps:

1. Check SonarQube analysis results against quality gate criteria.
2. Proceed if code meets quality gate requirements; otherwise, halt pipeline and alert stakeholders.

Stage 7: Build Application Artifact

Description:

Compile source code and package application into an executable artifact.

Steps:

1. Compile Java source code using Maven.
2. Bundle Node.js applications into distributable packages.
3. Generate JAR files or WAR files for Java applications.

Stage 8: Publish Artifact to Nexus

Description:

Upload the built artifact to a repository manager like Nexus for version control and distribution.

Steps:

1. Authenticate with Nexus repository.
2. Publish artifact to designated repository.

Stage 9: Build Docker Image

Description:

Package the application artifact into a Docker image for containerized deployment.

Steps:

1. Create Dockerfile specifying image configuration.
2. Build Docker image using Docker build command.

Stage 10: Scan Docker Image (Trivy)

Description:

Scan Docker image for vulnerabilities and security issues using Trivy.

Steps:

1. Pull Docker image locally.
2. Run Trivy scan on Docker image.
3. Generate report highlighting vulnerabilities found.

Stage 11: Push Docker Image to Docker Hub Repo

Description:

Push the built Docker image to a Docker Hub repository for centralized storage and distribution.

Steps:

1. Authenticate with Docker Hub.
2. Push Docker image to designated repository.

Stage 12: Update Image in Manifests Files

Description:

Update Kubernetes manifests files to reference the newly built Docker image.

Steps:

1. Update deployment YAML files with the latest image tag.
2. Commit changes to version control.

Stage 13: Deploy Application to Kubernetes Cluster

Description:

Deploy the application to a Kubernetes cluster for container orchestration.

Steps:

1. Authenticate with Kubernetes cluster.
2. Apply updated manifests files to deploy new version of the application.

Stage 14: Verify Deployment

Description:

Verify successful deployment of the application to the Kubernetes cluster.

Steps:

1. Check deployment status in Kubernetes dashboard or CLI.
2. Perform smoke tests to ensure application functionality.

Stage 15: OWASP ZAP Analysis

Description:

Conduct security testing using OWASP ZAP to identify and mitigate potential security vulnerabilities.

Steps:

1. Configure OWASP ZAP to scan the deployed application.
2. Analyze scan results for security vulnerabilities.

Stage 16: Send Mail Notifications

Description:

Notify stakeholders about pipeline status and results via email.

Steps:

1. Configure email server settings.
2. Send email notifications with pipeline status and relevant reports.

Each stage in the pipeline plays a crucial role in ensuring the reliability, security, and quality of the deployed application. Integrating these stages into a seamless CI/CD pipeline facilitates efficient and automated software delivery.

Here's a declarative Jenkins pipeline script for the described stages:

```
pipeline {
    agent any

    environment {
        // Define environment variables
        JAVA_HOME = "/path/to/java/home"
        MAVEN_HOME = "/path/to/maven/home"
        NODE_HOME = "/path/to/node/home"
        DOCKER_HUB_CREDENTIALS = 'docker-hub-credentials'
        NEXUS_REPO_URL = 'https://nexus.example.com/repository/maven-
releases/'
        KUBECONFIG = "/path/to/kubeconfig"
    }

    stages {
        stage('Install Tools') {
            steps {
                // Install required tools
                sh 'sudo apt-get install -y nodejs npm'
                sh 'curl -sL https://deb.nodesource.com/setup_14.x | sudo -
E bash -'
                sh 'sudo apt-get install -y nodejs'
                sh 'sudo apt-get install -y default-jdk'
                sh 'sudo apt-get install -y maven'
                sh 'sudo apt-get install -y trivy'
                sh 'sudo apt-get install -y sonarqube'
                sh 'sudo apt-get install -y docker.io'
            }
        }

        stage('Install Dependencies') {
            steps {
                // Install project dependencies
                sh 'npm install'
                sh 'mvn install'
            }
        }

        stage('Execute Test Cases') {
            steps {
                // Execute test cases
                sh 'npm test'
                sh 'mvn test'
            }
        }

        stage('File System Scan (Trivy)') {
            steps {
                // Perform Trivy filesystem scan
                sh 'trivy filesystem scan'
            }
        }

        stage('SonarQube Analysis') {
            steps {
                // Perform SonarQube analysis
                sh 'sonar-scanner'
            }
        }
        stage('Quality Gate Check') {
            steps {
                // Perform quality gate check
                // Implement quality gate check based on SonarQube results
            }
        }

        stage('Build Application Artifact') {
            steps {
                // Build application artifact
                sh 'mvn package'
            }
        }

        stage('Publish Artifact to Nexus') {
            steps {
                // Publish artifact to Nexus repository
                sh "mvn deploy -Drepository.url=${NEXUS_REPO_URL}"
            }
        }

        stage('Build Docker Image') {
            steps {
                // Build Docker image
                sh 'docker build -t myapp .'
            }
        }

        stage('Scan Docker Image (Trivy)') {
            steps {
                // Scan Docker image using Trivy
                sh 'trivy image scan myapp'
            }
        }

        stage('Push Docker Image to Docker Hub Repo') {
            steps {
                // Push Docker image to Docker Hub repository
                withCredentials([usernamePassword(credentialsId: 
DOCKER_HUB_CREDENTIALS, passwordVariable: 'DOCKER_PASSWORD', 
usernameVariable: 'DOCKER_USERNAME')]) {
                    sh "docker login -u ${DOCKER_USERNAME} -p 
${DOCKER_PASSWORD}"
                    sh 'docker push myapp'
                }
            }
        }

        stage('Update Image in Manifests Files') {
            steps {
                // Update image tag in Kubernetes manifests files
                // Implement updating Kubernetes deployment YAML files
            }
        }

        stage('Deploy Application to Kubernetes Cluster') {
            steps {
                // Deploy application to Kubernetes cluster
                sh 'kubectl apply -f manifests/'
            }
        }

        stage('Verify Deployment') {
            steps {
                // Verify deployment
                // Implement verification steps
            }
        }

        stage('OWASP ZAP Analysis') {
            steps {
                // Perform OWASP ZAP analysis
                // Implement OWASP ZAP analysis steps
            }
        }

        stage('Send Mail Notifications') {
            steps {
                // Send mail notifications
                // Implement email notification steps
            }
        }
    }
}
```