PHASE 4-TESTING

PROJECT TITLE :- Setting up a CI/CD pipeline for automated deployment

COLLEGE NAME:-Dr. SMCE

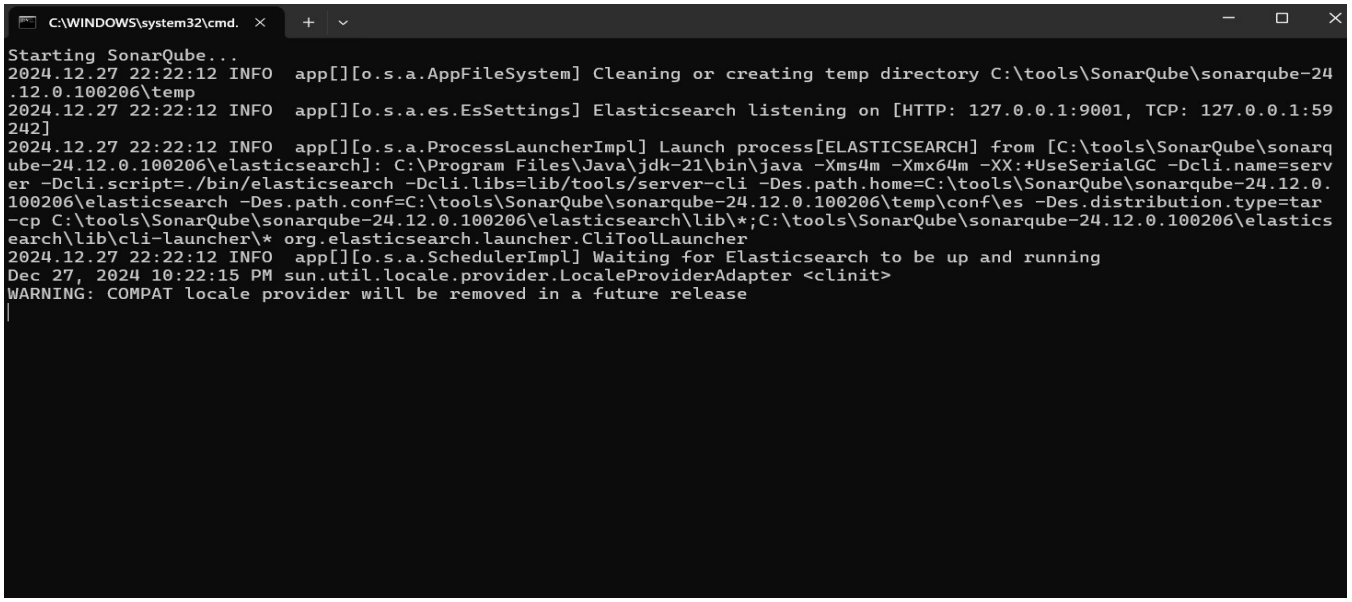
GROUP MEMBERS:-

1. Amith C Y [USN:- 1CC21CS004]
2. Charan K M [USN:- 1CC21CS017]
3. Nitin Teja N[USN:-1CC21CS040]
4. Sunil C M [USN:-1CC21CS057]

Phase 4: TESTING

Step 1: Static Code Analysis (SAST)

- Run **SonarQube** to detect security vulnerabilities in the source code.
- Fix issues before progressing to the build stage.



SAMPLE JENKINS FILE

```
pipeline {
    agent any
    environment {
        IMAGE_NAME = "my-python-app"
        DOCKER_HUB_REPO = "docker.io/myusername/my-python-app"
    }
```

```
        stages {
            stage('Checkout') {
                steps {
                    git 'https://github.com/myusername/my-python-app.git'
                }
            }

            stage('Code Quality Check') {
                steps {
                    withSonarQubeEnv('SonarQube') {
                        sh 'sonar-scanner'
                    }
                }
            }

            stage('Build Docker Image') {
                steps {
                    sh 'docker build -t $IMAGE_NAME .'
                }
            }

            stage('Push to DockerHub') {
                steps {
                    withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                        sh """
                            echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin
                            docker tag $IMAGE_NAME $DOCKER_HUB_REPO
                            docker push $DOCKER_HUB_REPO
                        """
                    }
                }
            }

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

        post {
            always {
                mail to: 'devteam@example.com',
                     subject: "Build ${currentBuild.fullDisplayName}",
                     body: "Pipeline ${currentBuild.fullDisplayName} finished with status:
${currentBuild.currentResult}"
            }
        }
    }
```

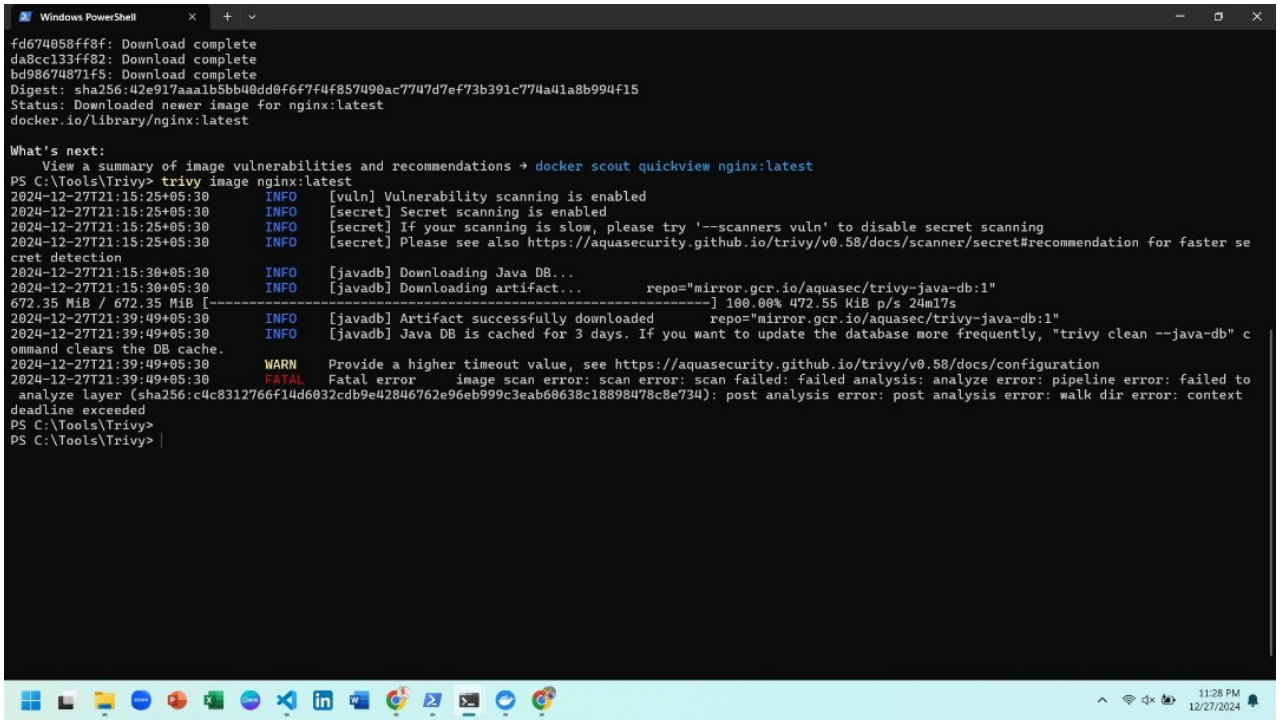
Testing and Validation

The pipeline was rigorously tested across several dimensions:

- **Static Code Analysis**: Validated the integration of SonarQube and verified the accuracy of reported metrics.
- **Build Verification**: Ensured Docker images were successfully built and tagged with the correct version identifiers.
- **Functional Testing**: Unit tests were executed automatically as part of the pipeline using pytest.
- **Deployment Validation**: The containerized Python app was tested post-deployment in a Minikube cluster.
- **Rollback Strategy**: Kubernetes deployment configuration supported rolling updates with rollback on failure.

Step 3: Container Security Testing

- Scan container images using **Trivy** or **Snyk**.
- Enforce least privilege access policies.



Step 4: Infrastructure and Compliance Testing

- Scan Kubernetes configurations with **Checkov** to detect misconfigurations.
- Verify compliance with **GDPR, PCI-DSS, SOC2** security policies.

Step 5: CI/CD Pipeline Security Validation

- Ensure security policies are enforced in the **Jenkins pipeline**.
- Monitor deployments for unauthorized changes using **ELK Stack**.

FUTURE IMPROVEMENTS

1. **Enhance Security Automation**: Implement real-time security monitoring and automated patch management.
2. **Adopt Zero Trust Architecture**: Introduce strict authentication and continuous security verification.
3. **Implement Advanced Threat Detection**: Use AI-powered threat detection tools for anomaly monitoring.
4. **Optimize Multi-Cloud Security**: Extend security controls across **AWS, Azure, and Google Cloud**.
5. **Automate Security Audits**: Set up automated penetration testing and compliance validation.