Jenkins Pipeline Implementation Report

By Charbel Toumieh

Introduction

This report details the implementation of a Jenkins pipeline for Python application testing, security scanning, code coverage analysis, and deployment. The pipeline was enhanced to include additional stages for comprehensive quality assurance and deployment automation.

Pipeline Stages Implementation

1. Security Scanning Stage

- Implemented using Bandit security scanner

- Generates both HTML and JSON reports

- Features:

- Recursive scanning of Python code

- HTML report for human readability

- JSON report for potential automated analysis

- Reports archived as build artifacts

- HTML report published in Jenkins UI

2. Coverage Analysis Stage

- Implemented using coverage.py tool

- Comprehensive coverage reporting setup:

- Console report for immediate feedback

- HTML report for detailed analysis

- XML report for CI tool integration

- Coverage badge generation for repository

- Post-build actions:

- Archives all coverage reports

- Publishes HTML report in Jenkins UI

- Preserves coverage badge

3. Deployment Stage

- Implements local deployment simulation

- Deployment process:

1. Creates isolated deployment directory

2. Copies application files

3. Creates version tracking file

4. Sets up virtual environment

5. Installs dependencies

6. Performs deployment test

- Post-deployment:

- Archives deployment directory

- Provides deployment verification

Pipeline Execution Results

Security Scan Results

- Successfully scans Python code for security vulnerabilities

- Generates comprehensive security reports

- Results accessible through Jenkins UI

Coverage Analysis Results

- Achieves comprehensive code coverage

- Generates detailed reports showing:

- Line coverage

- Branch coverage

- Missing coverage areas

- Results visualized through HTML reports

Deployment Results

- Successfully creates isolated deployment environment

- Verifies application functionality post-deployment

- Maintains deployment history through artifacts

Additional Enhancements

1. Environment Variables

- VIRTUAL\_ENV for Python virtual environment

- DEPLOY\_DIR for deployment location

2. Post-Build Actions

- Workspace cleanup

- Build status notifications

- Artifact archiving

3. Error Handling

- Comprehensive post-build cleanup

- Failure notifications

- Artifact preservation on failure

Conclusion

The enhanced pipeline successfully implements all required functionality:

- Security scanning with Bandit

- Code coverage analysis with coverage.py

- Automated deployment process

- Comprehensive reporting and artifact management

The pipeline ensures code quality, security, and reliable deployment while maintaining complete visibility into the build process through detailed reports and artifacts.

Setup Instructions

1. Install required Jenkins plugins:

- HTML Publisher Plugin

- Pipeline Plugin

- Workspace Cleanup Plugin

2. Configure Jenkins pipeline:

- Create new pipeline job

- Configure Git repository

- Set Jenkinsfile path to "Jenkinsfile\_Charbel\_Toumieh"

3. Required Python packages:

- bandit

- coverage

- pytest

- coverage-badge