Code Coverage with pytest

Measuring and Improving Test Coverage

A comprehensive guide to implementing code coverage in Python projects

Overview

To get a code coverage report with pytest, you must install the pytest-cov plugin and the coverage.py library.

After installation, you can run pytest with specific flags to measure and report on your code coverage.

Step 1: Install the Necessary Packages

Run the following command to install required dependencies:

```
pip install pytest-cov
```

This installs both **pytest-cov** plugin and **coverage.py** library

View a Summary in the Terminal

Get a quick overview of your project's coverage directly in the terminal:

```
pytest --cov=your_module
```

`--cov=your_module`: Specifies the module, package, or directory to measure

Replace `your_module` with your source code directory

Detailed Terminal Report

To see which specific lines were missed, use the flag:

```
pytest --cov=your_module --cov-report=term-missing
```

This command displays a line-by-line breakdown of coverage, highlighting uncovered lines in your code.

Generate Comprehensive HTML Report

For a detailed, interactive report viewable in a web browser:

```
pytest --cov=your_module --cov-report=html
```

Creates a new directory **htmlcov/** in your project root with comprehensive analysis and interactive file exploration.

Step 3: View the HTML Report

Open the **index.html** file in the **htmlcov/** directory

Use any web browser to view the interactive report

See a summary of all files in your project

Click into each file for line-by-line breakdown

Covered and uncovered lines are highlighted for easy identification

Bonus: Enforce Minimum Coverage

In CI/CD environments, enforce a minimum coverage percentage:

```
pytest --cov=your_module --cov-fail-under=80
```

- **--cov-fail-under** causes the test run to fail if coverage is below the threshold
- Example above enforces a minimum of **80%** coverage
- Promotes code quality and testing standards across your team

Summary

- **Install:** pytest-cov and coverage.py
- **Terminal Report:** Quick coverage overview with optional line details
- **HTML Report:** Interactive, detailed analysis with file-level insights
- **Enforcement:** Set minimum coverage thresholds in CI/CD pipelines
- **Result:** Better test coverage and code quality

Python pytest Code Coverage Integration

Integrating pytest Coverage with SonarQube in Jenkins Pipeline

Key Benefits:

- Automated code quality analysis
- Real-time coverage metrics
- Continuous integration best practices
- Quality gate enforcement

Integration Overview

This integration enables seamless code quality and coverage analysis through a modern CI/CD pipeline.

Three Key Components:

- **Jenkins:** Orchestrates the pipeline and coordinates all steps
- **pytest-cov:** Generates detailed code coverage reports
- **SonarQube:** Analyses code quality and visualizes metrics

Together, these tools provide comprehensive insights into code health and test coverage.

Jenkins Configuration

Step 1: Install SonarQube Scanner Plugin

- Navigate to **Manage Jenkins** → **Manage Plugins**
- Search for "SonarQube Scanner"
- Install and restart Jenkins

Step 2: Configure SonarQube Server

- Go to **Manage Jenkins** → **Configure System**
- Add your SonarQube server details
- Include server URL, name, and authentication token
- Enable injection of configurations as environment variables

Jenkins Configuration (continued)

Step 3: Install SonarScanner

- Navigate to **Manage Jenkins** → **Global Tool Configuration**
- Locate the SonarScanner section
- Choose one of two options:

Option A: Enable automatic installation

Option B: Specify an existing SonarScanner installation path

These configurations are foundational for all subsequent pipeline stages.

Project Setup: Python & pytest-cov

Generate Coverage Report

Use pytest-cov to produce a Cobertura XML coverage report:

```
pytest --cov=./your_project_directory --cov-report=xml:coverage.xml
```

Create sonar-project.properties File

Create this file in your project's root directory to define analysis parameters:

- **sonar.projectKey:** Unique identifier for your project
- **sonar.sources:** Path to your source code
- **sonar.python.coverage.reportPaths:** Point to coverage.xml file

Project Setup: Python & pytest-cov

Generate Coverage Report

Use pytest-cov to produce a Cobertura XML coverage report:

```
pytest --cov=./your_project_directory --cov-report=xml:coverage.xml
```

Create sonar-project.properties File

Create this file in your project's root directory to define analysis parameters:

- **sonar.projectKey:** Unique identifier for your project
- **sonar.sources:** Path to your source code
- **sonar.python.coverage.reportPaths:** Point to coverage.xml file

Jenkinsfile: Run Tests & Generate Coverage

Stage 3: Execute Tests and Generate Coverage Report

Run pytest with coverage reporting:

```
stage('Run Tests and Generate Coverage') {
    steps {
        script {
            sh 'pip install pytest pytest-cov'
            sh 'pytest --cov=./your_project_directory --cov-report=xml:coverage.xml'
        }
    }
}
```

Output: Generates coverage.xml in Cobertura format for SonarQube analysis

Jenkinsfile: SonarQube Analysis

Stage 4: SonarQube Analysis

Use withSonarQubeEnv to access server configurations and run the scanner:

```
stage('SonarQube Analysis') {
    steps {
        withSonarQubeEnv('Your SonarQube Server Name') { // Use the name configured in Jenkins
            sh 'sonar-scanner' // If using sonar-project.properties
            // OR specify properties directly:
            // sh 'sonar-scanner -Dsonar.projectKey=my_python_project -Dsonar.sources=./your_project_directory -
Dsonar.python.coverage.reportPaths=coverage.xml'
        }
    }
}
```

Two Approaches:

- Use sonar-project.properties file (recommended)
- Specify properties directly in the scanner command

Jenkinsfile: Quality Gate Check

Stage 5: Quality Gate Validation

Add this stage to enforce quality standards and fail builds that don't meet criteria:

```
stage('Quality Gate Check') {
    steps {
        script {
            def qualityGate = waitForQualityGate()
            if (qualityGate.status != 'OK') {
                error "SonarQube Quality Gate failed with status: ${qualityGate.status}"
            }
        }
    }
}
```

Impact: Ensures code quality standards are met before deployment

Results & Analysis

After Successful Pipeline Execution

Your code coverage and analysis results are available in SonarQube dashboard:

- **Code Coverage Metrics:** Line and branch coverage percentages
- **Quality Issues:** Bugs, vulnerabilities, and code smells identified
- **Trend Analysis:** Historical tracking of code quality
- **Quality Gates:** Pass/fail status against defined criteria

Next Steps: Review metrics regularly and iterate on code improvements based on SonarQube feedback.

Key Takeaways

- Three essential Jenkins configurations set up the foundation
- pytest-cov generates standardized Cobertura coverage reports
- sonar-project.properties centralizes SonarQube configuration
- Five-stage Jenkinsfile orchestrates the entire workflow
- Quality Gates enforce code standards automatically
- SonarQube dashboard provides actionable insights

Result: A fully automated, reproducible code quality pipeline