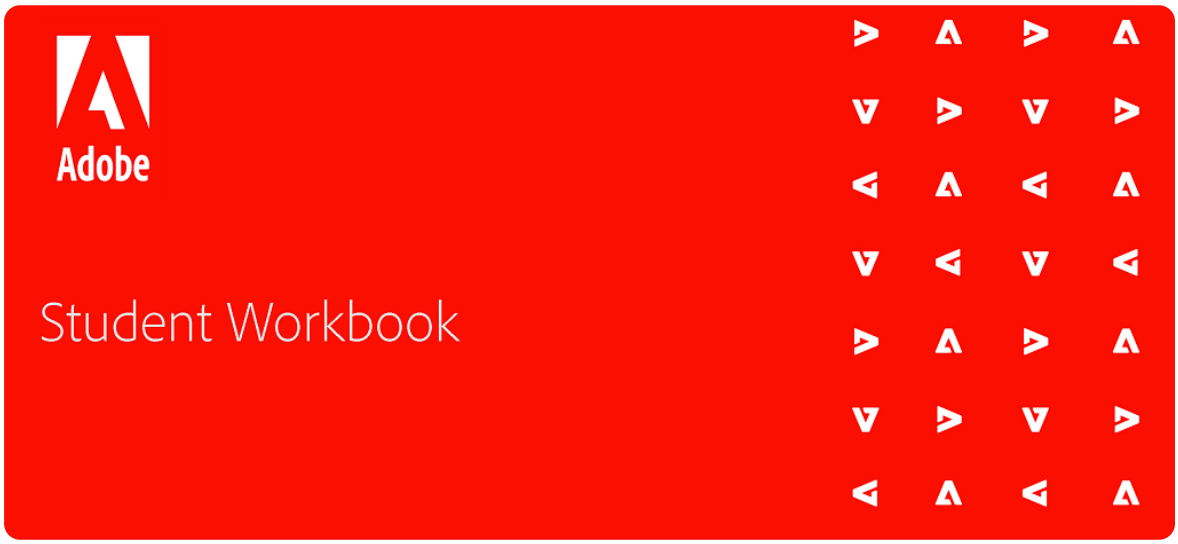
DevOps for AEM as a Cloud Service

Code Testing in Pipelines



Find your journey at learning.adobe.com >

Contents

DevOps for AEM as a Cloud Service

10-14-2020

Introduction

Cloud Manager for Cloud Services pipeline executions will support execution of tests that run against the stage environment. As part of the pipeline the source code is scanned to ensure that deployments meet certain quality criteria. In Adobe Experience Manager (AEM) as a Cloud Service, this is implemented by a combination of SonarQube and content package-level examination using OakPAL.

Objectives

After completing this module, you will be able to:

- Explain Unit Testing
- Run a code quality pipeline of Unit tests failing
- Explain code quality Rules
- Run a code quality pipeline of SonarQube tests failing
- Describe other tests specific to Production pipeline
- Run a successful code quality pipeline

Unit Testing

Maven automatically runs Junit tests with every single build. These unit tests can be triggered locally when installing locally with Maven. You can also run unit tests without building or installing the entire project:

```
# Run all the unit test classes.
$ mvn test

# Run a single test class.
$ mvn -Dtest=TestApp1 test

# Run multiple test classes.
$ mvn -Dtest=TestApp1,TestApp2 test

# Run a single test method from a test class.
$ mvn -Dtest=TestApp1#methodname test

# Run all test methods that match pattern 'testHello*' from a test class.
$ mvn -Dtest=TestApp1#testHello* test

# Run all test methods match pattern 'testHello*' and 'testMagic*' from a
test class.
$ mvn -Dtest=TestApp1#testHello*+testMagic* test
```

In your Maven project, Junit tests are located in `core/src/test/java/`. These java based unit tests are used to test the actual java classes in `core/src/main/java/`. These tests use mocked objects that can come from many different mocking frameworks. Adobe recommends using the wcm.io aem-mocks[1] for testing AEM code.

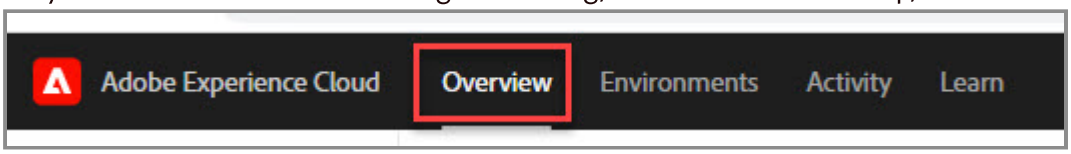
```
@ExtendWith(AemContextExtension.class)
public class ExampleTest {
  private final AemContext context = new AemContext();
  @Test
  public void testSomething() {
    Resource resource =
context.resourceResolver().getResource("/content/sample/en");
    Page page = resource.adaptTo(Page.class);
    // further testing
  }
}
```

The Cloud Manager pipeline also builds with Maven, therefore invoking these tests. These unit tests are near the beginning of the pipeline process, ensuring the remaining pipeline steps can run properly.

Exercise 1: Run a code quality pipeline of Unit tests failing

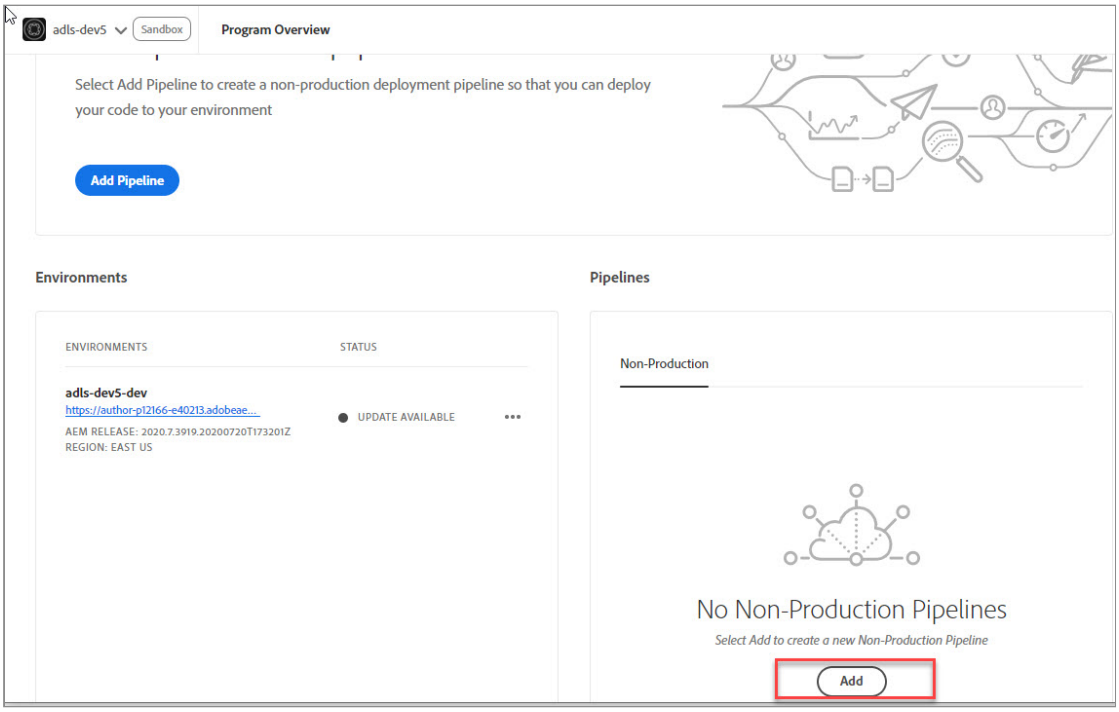
Scenario: Typically pipelines are started by an automated process to validate whether the build passes the basic code quality before it is loaded onto an environment. The Cloud Manager pipeline builds and executes all custom unit tests at the beginning of the pipeline. In this exercise you will learn what happens when a pipeline fails this step.

1. On your browser where Cloud Manager is running, click **Overview** at the top, as shown:



The **Program Overview** page opens.

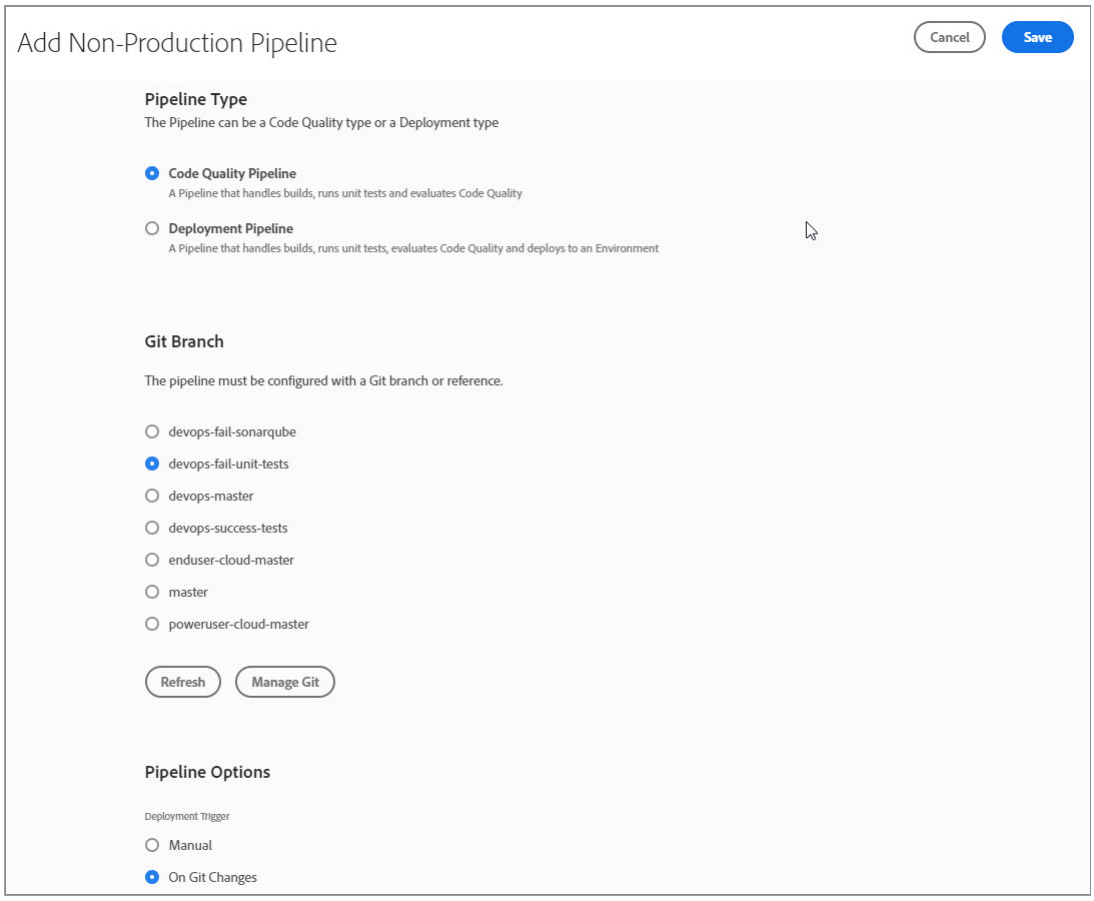
2. On the **Program Overview** page, under **Pipelines** click the **Add** button, as shown:



The **Add Non-Production Pipeline** is displayed.

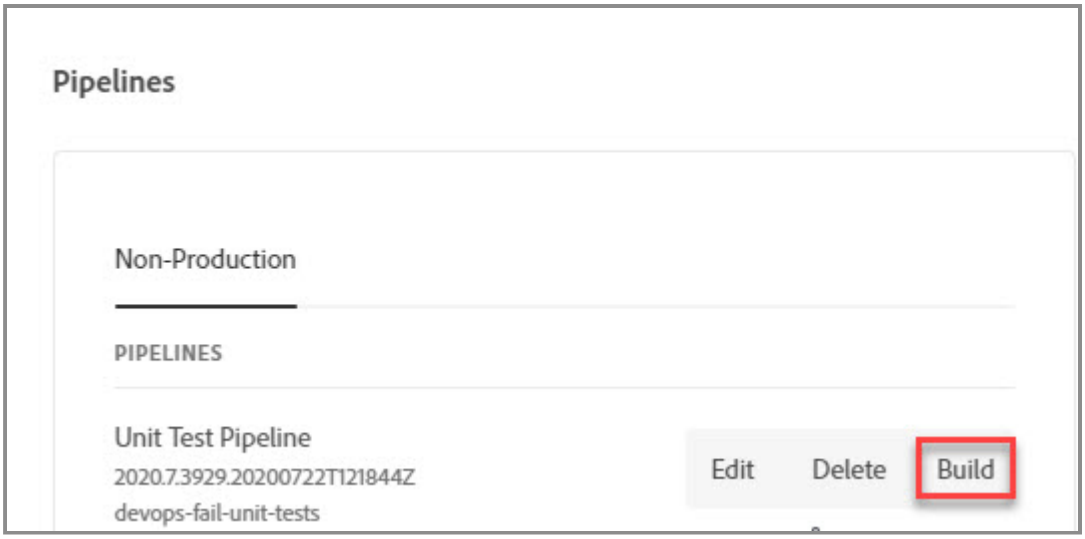
3. On the **Add Non-Production Pipeline** , enter the following details, as shown:

   - Pipeline Name: **Unit Test Pipeline**
   - Pipeline Type: **Code Quality Pipeline**
   - Git Branch: **devops-fail-unit-tests**
   - Pipeline Options: **On Git Changes**

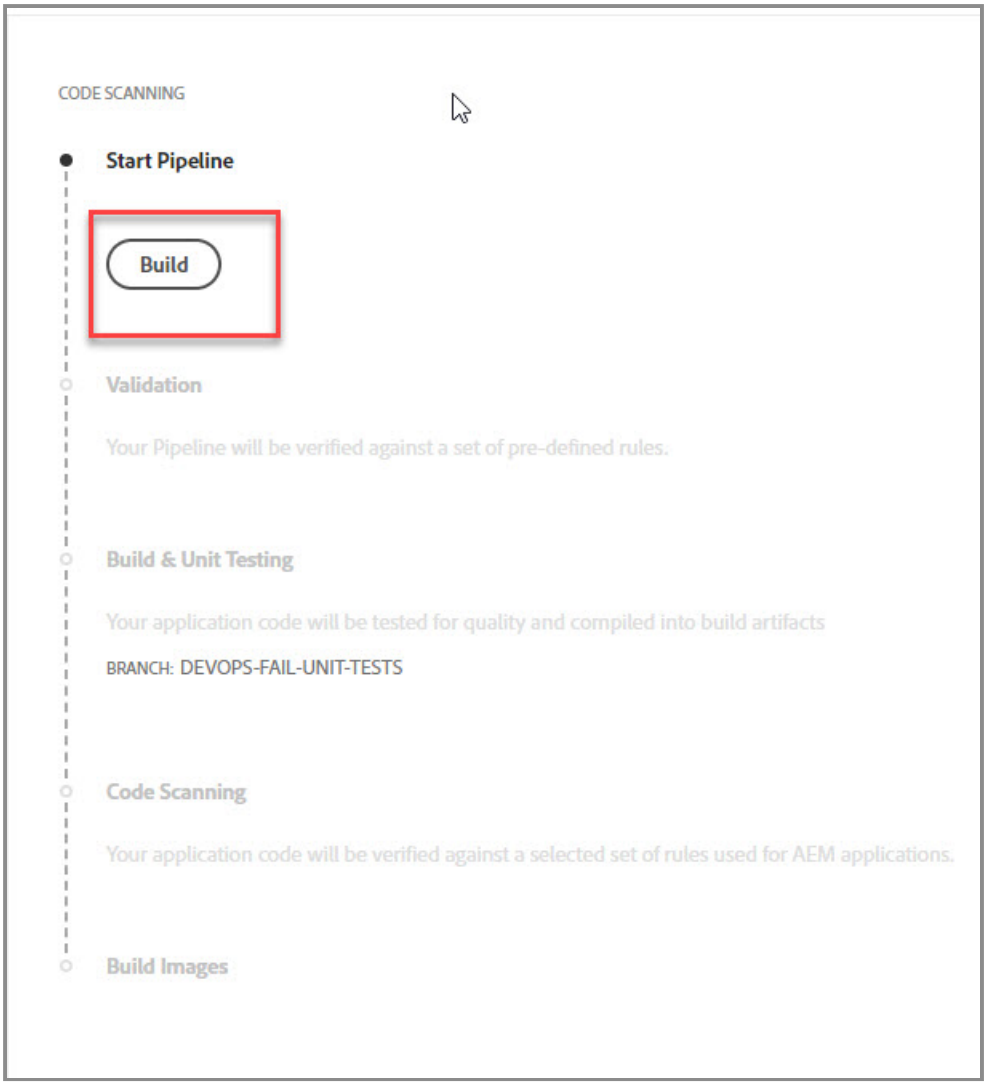


4. Click **Save**. The **Non-Production** page opens.

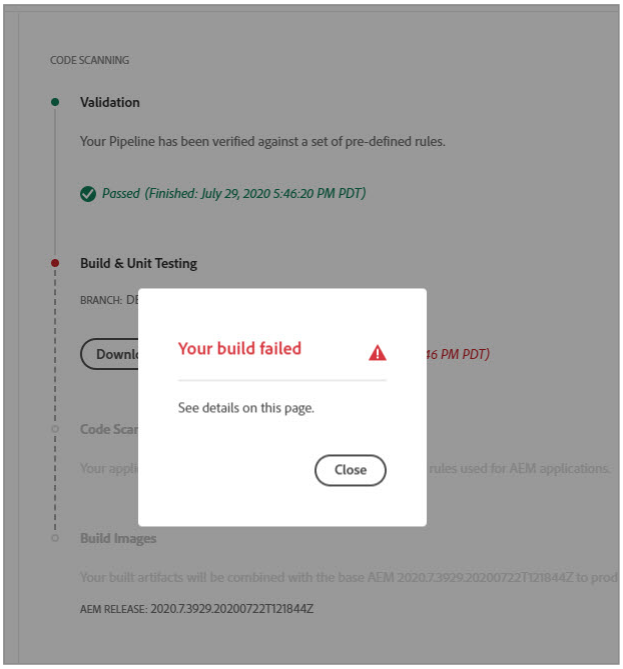
5. Hover over **Unit Test Pipeline** and click **Build**. This opens the **Build and Code Scanning** screen, as shown:



6. Click **Build** to start the pipeline, as shown. The pipeline starts to build.



After about 5 minutes, you will notice the Pipeline fails. Notice the pipeline failed on the **Build & Unit Testing** step, as shown:



Note: If you are unable to run the failed pipeline, a sample log has been provided for you in **training-files/Testing/build_maven_build.log**

7. Click **Close** on **Your build failed** wizard.

8. Click the **Download Log** button. The **build_maven_build.log** file is downloaded.

Note: If you are no longer on the Pipeline page, you can navigate there by selecting **Activity** at the top of the Cloud Manager. Find the failed Pipeline and click **Details**.

9. Open the log and observe the output. You should notice that two unit tests failed, as shown:

```
19:31:00,065 [ThreadedStreamConsumer] [ERROR] doGet{AemContext}  Time
elapsed: 1.592 s  <<< FAILURE!
   org.opentest4j.AssertionFailedError: expected: <Title = resource title>
but was: <Title = null> at
com.adobe.training.devops.core.servlets.SimpleServletTest.doGet(SimpleServle
tTest.java:47)

19:31:00,186 [ThreadedStreamConsumer] [ERROR] testGetMessage  Time elapsed:
0 s  <<< FAILURE!
   org.opentest4j.AssertionFailedError: expected: not <null> at
com.adobe.training.devops.core.models.HelloWorldModelTest.testGetMessage(Hel
loWorldModelTest.java:58)
```

Note: These two errors were intentionally put into the code to show unit tests failing. To view these java issues you will need to switch project branches in your devops-training project.

To observe these failure tests in your local environment/Readytech:

10. Open a command line window and navigate to your **devops-training** project:

```
cd C:/Users/adlsadmin/Desktop/devops-training
```

Switch branches to see the unit test errors. To do:

11. View the branches you have locally

```
git branch -a
```

12. Change to the branch you just tested in the pipeline

```
git checkout fail-unit-tests
```

13. Confirm you are on the correct branch

```
git branch
```

14. Open your Project in **Eclipse** (if it is not open already) and navigate to **core/src/main/java/com/adobe/training/devops/core/**:

a. Open **servlets/SimpleServlet.java** and notice that the `write()` method is writing the **JcrConstants.JCR_DESCRIPTION** instead of **JcrConstants.JCR_TITLE**.

b. Open **models/HelloWorldModel.java** and notice the `getMessage()` method returns null rather than the **message**..

Code Scanning

Code scanning in the pipeline consists of over 100 tests run consisting of java based rules as well as AEM. These tests are implemented by a combination of SonarQube [2] and content package level examination using OakPAL [3]. The following table summarizes the rating for testing criteria:

Metric	Failure Threshold
Security Rating	< B
Reliability Rating	< C
Maintainability Rating	< A
Coverage	< 50%
Skipped Unit tests	> 1
Open issues	> 0
Duplicated Lines	> 1%
Cloud Service Compatibility	> 0

The fully detailed table can be found in Helpx [4]

Code Quality Rules

Code quality rules are currently derived from three different projects:

- Sonarqube [5] for generic Java rules
- CQRules [6] for AEMaaCS specific rules
- AEMRules [7] for general AEM best practice

The output from the code scanning step will identify several details of the failure:

Column Name	Description
File Location	Location of file in maven project
Line Number	Line number in file
Issue	Description of issue
Type	Bug, Vulnerability, Code Smell [8]
Severity	Blocker, Critical, Major, Minor, Info [8]
Effort	Estimated time to fix issue
Rule	Specific rule identified
Tags	Used to categorize the issue
Documentation	Link to online doc for this rule

Exercise 2: Run a Code Quality pipeline of SonarQube tests failing

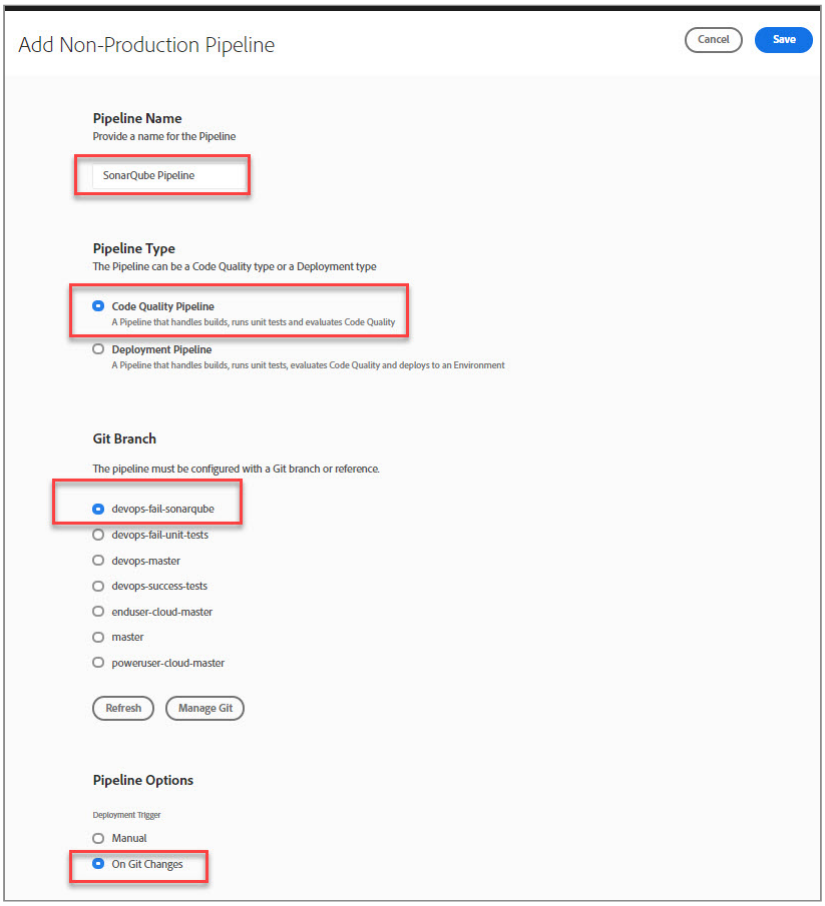
Scenario: Every pipeline run goes through a series of code quality tests to ensure the code passes inspection for production. In this exercise, you will examine failed sonar tests and how to fix them.

1. On your browser where Cloud Manager is running, click **Overview** at the top. The **Program Overview** page opens.

2. On the **Program Overview** page, under **Pipelines** click **Add** button. The **Add Non-Production Pipeline** opens.

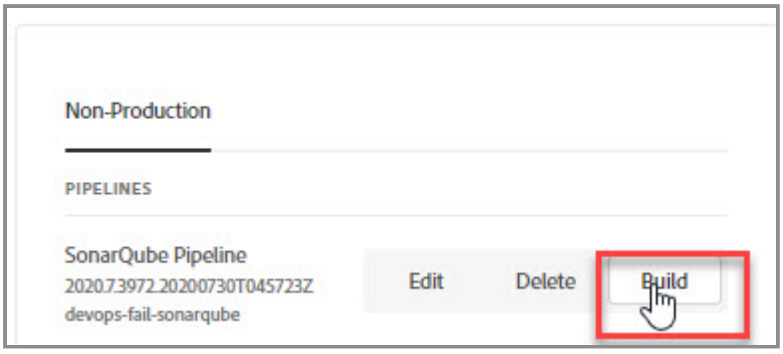
3. Enter the following details, as shown:

    - Name: **SonarQube Pipeline**
    - Pipeline Type: **Code Quality Pipeline**
    - Git Branch: **devops-fail-sonarqube**
    - Pipeline Options: **On Git Changes**



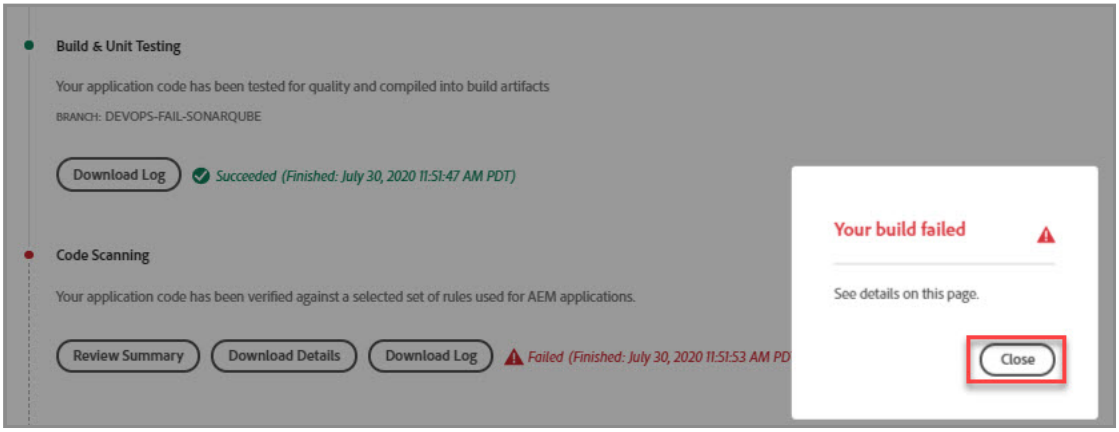
4. Click **Save**. The **Non-Production** screen is displayed.

5. Hover over **SonarQube Pipeline** and click **Build**, as shown. This opens the **Build and Code Scanning** page.



6. Click **Build** to start the Pipeline. The Pipeline starts to build.
   After about seven minutes, you will notice the pipeline fails.

7. Notice the pipeline failed on the **Code Scanning** step, as shown:

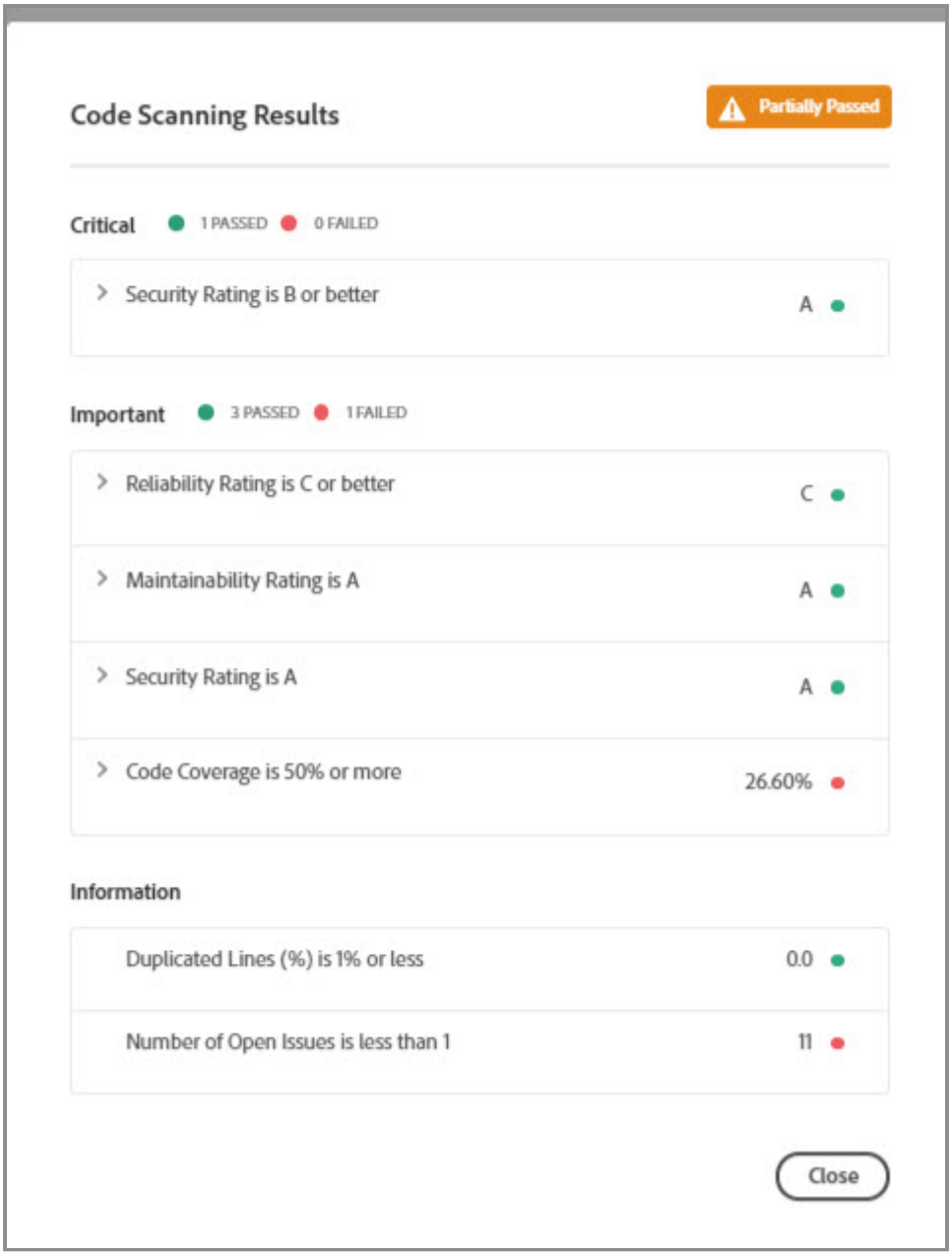


8. Click **Close** on **Your build failed** wizard.

9. Click the **Download Log** button. This downloads the **build_code_quality.log** file.

> Note: If you are no longer on the Pipeline screen, you can navigate there by selecting **Activity** at the top of Cloud Manager. Find the failed Pipeline and click **Details**.

10. Click **Review Summary**. The Code Scanning Results opens, as shown:



11. Notice at the bottom of the window, the **Number of Open Issues** is more than 1, which results in a failure.

12. Click **Close**.

13. Click the **Download Details** button, as shown. This downloads the build_project_issues.csv file.

> Note: If you are unable to run the failed pipeline, a sample log has been provided for you in **training-files/Testing/build_project_issues.csv**

14. Open the csv and observe the output. You should be able to investigate each issue and estimate the time it will take to fix each based on each row in the csv file.

File Locati	Line Num	Issue	Type	Severity	Effort	Rule	Tags	Documentation
com.adob	46	A "NullPointerException" could be throv	Bug	Major	10min	squid:S22	cert,cwe	https://www.adobe.com/go/aem_cmcq_s2259_en
com.adob	34	Make "logger" transient or serializable.	Code Smell	Critical	30min	squid:S19	cwe,serial	https://www.adobe.com/go/aem_cmcq_s1948_en
com.adob	85	Catch a list of specific exception subtype	Code Smell	Major	15min	squid:S22	cwe,error	https://www.adobe.com/go/aem_cmcq_s2221_en
com.adob	33	Method 'getAdministrativeResourceRes	Code Smell	Major	30min	AEM Rule:	aem	https://www.adobe.com/go/aem_cmcq_aem-11_en
com.adob	27	Do not use Sling servlet paths to registe	Code Smell	Major	30min	CQRules:C	cqsoftwar	https://www.adobe.com/go/aem_cmcq_cqbp-75_en
com.adob	87	Do not use Exception.getMessage() as th	Code Smell	Minor	15min	CQRules:C	cqsoftwar	https://www.adobe.com/go/aem_cmcq_cqbp-44---exceptiong_en
com.adob	89	Do not use Exception.printStackTrace() t	Code Smell	Minor	15min	CQRules:C	cqsoftwar	https://www.adobe.com/go/aem_cmcq_cqbp-44---exceptionp_en
com.adob	33	Remove this use of "getAdministrativeR	Code Smell	Minor	15min	squid:Call	cert,cwe,(	https://www.adobe.com/go/aem_cmcq_calltodeprecatedmeth_en
com.adob	38	Use constant JCR_TITLE from interface cc	Code Smell	Minor	5min	AEM Rule:	aem	https://www.adobe.com/go/aem_cmcq_aem-2_en
com.adob	37	Do not hardcode paths using String liter	Code Smell	Minor	10min	CQRules:C	cqsoftwar	https://www.adobe.com/go/aem_cmcq_cqbp-71_en
com.adob	42	Do not log with INFO level in GET or HEA	Code Smell	Minor	15min	CQRules:C	cqsoftwar	https://www.adobe.com/go/aem_cmcq_cqbp-44---loginfoing_en

> **Note**: These errors were intentionally put in the code to show different SonarQube rulesets triggered. To view these java issues you will need to switch project branches in your devops-training project.

To observe these failure tests in your local environment or Readytech:

15. Open a command line window and navigate to your devops-training project:

```
cd C:/adobe/devops-training
```

To see the unit test errors, switch branches:

16. View the branches you have locally:

```
git branch -a
```

17. Change to the branch you just tested in the pipeline

```
git checkout fail-sonarqube
```

18. Confirm you are on the correct branch

```
git branch
```

19. Open your Project in **Eclipse** (if it is not open already) and observe the different rules triggered in each of the following java classes under **core/src/main/java/com/adobe/training/devops/core/:**

20. **listeners/TitlePropertyListener.java**

21. **AutoAssignACL.java**

22. **servlets/TitleSlingServlet.java**

Note: You can use the links in the csv to quickly navigate to the documentation and to check how to fix it for each ruleset. [5] [6] [7]

Exercise 3: Run a successful code quality pipeline

Scenario: In an actual development environment, a developer would take unit test failures and SonarQube test failures and resolve them individually. In this exercise you will observe the resolved issues and then run a pipeline successfully.

This exercise includes the following tasks:

1. Observe Unit Test and SonarQube fixes
2. Run a code quality pipeline

Task 1: Observe Unit Test and SonarQube fixes

To observe these failure tests in your local environment/Readytech:

1. Open a command line window and navigate to your **devops-training** project:

   ```
   cd C:/adobe/devops-training
   ```

   To see the unit test errors, switch branches:

2. View the branches you have locally

   ```
   git branch -a
   ```

3. Change to the branch with the issues fixed

   ```
   git checkout success-tests
   ```

4. Confirm you are on the correct branch

   ```
   git branch
   ```

5. Open your project in **Eclipse** (if it is not already open) and observe the fixed unit tests and correctly implemented SonarQube rules. Under **core/src/main/java/com/adobe/training/devops/core/:**

    a. Fixed Unit Tests:

        i. **servlets/SimpleServlet.java**

        ii. **models/HelloWorldModel.java**

    b. SonarQube Fixed Issues:

        i. **listeners/TitlePropertyListener.java**

        ii. **AutoAssignACL.java**

        iii. **servlets/TitleSlingServlet.java**

Note: In a real scenario, these bug fixes would be resolved and merged into the dev branch to be tested again in the pipeline.

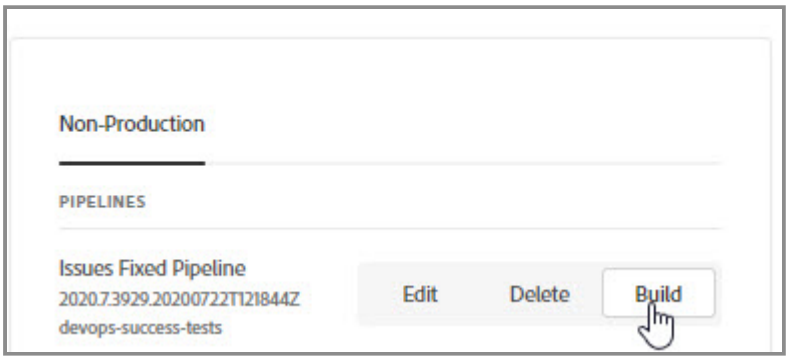
Task 2: Run a code quality pipeline

Now that the fixed issues are implemented, they need to be tested once again in a code quality pipeline. In this task, you will run a code quality Pipeline.

1. Open Cloud manager and create a new non-production pipeline:
    - Name: **Issues Fixed Pipeline**
    - Pipeline Type: **Code Quality Pipeline**
    - Git Branch: **devops-success-tests**
    - Pipeline Options: **On Git Changes**

2. Click **Save**. The **Non-Production** page opens.

3. Hover over **Issues Fixed Pipeline** and click **Build**, as shown:



The **Build and Code Scanning** screen opens.

4. Click **Build** to start the pipeline. The pipeline starts to build.

Note: At this point, the pipeline will run through the entire code quality pipeline, which includes building a new image. This can take from 20 to 40 minutes for a successful completion.

Notice the pipeline failed on the **Code Scanning** step.

5. Click **Close** on **Your build failed** wizard.

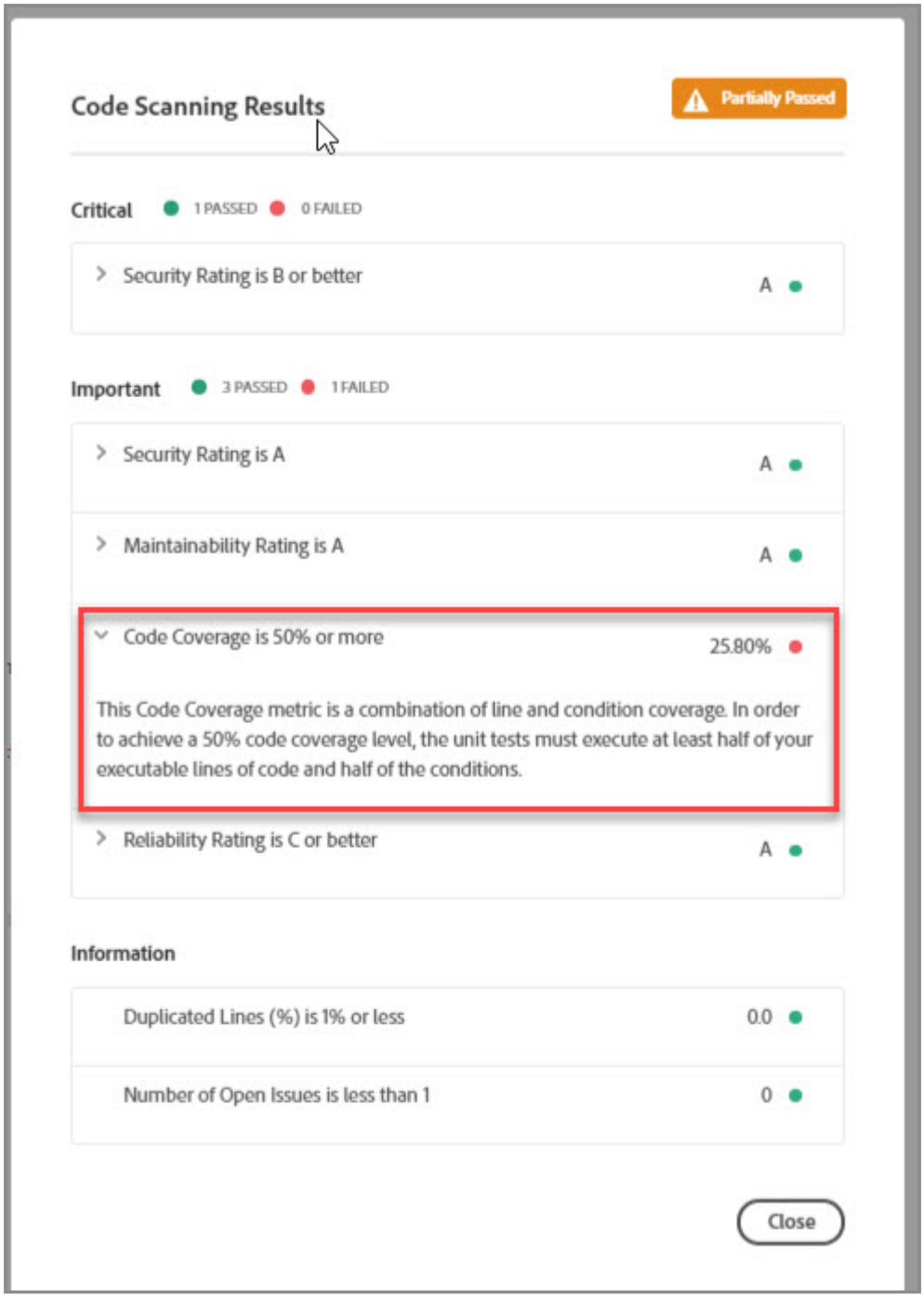
6. Click the **Download Log** button. The **build_code_quality.log** file is downloaded.

7. Open the log file using a text editor(such as Notepad++) and notice there are no errors.

Note: If you are no longer on the Pipeline screen, you can navigate there by selecting **Activity** at the top of the Cloud Manager. Find the failed Pipeline and click **Details**.

8. Click **Review Summary**. The Code Scanning Results page opens.

Notice that Code Coverage is less than 50%, as shown:



This is considered an Important issue and, therefore, the pipeline failed.

Note: Code quality pipelines automatically fail on important and critical issues. Development Pipelines (attached to an environment) have the option to continue with important issues, if needed. Development Pipelines will still fail with critical issues.

References

1. AEM Mocks https://wcm.io/testing/aem-mock/usage.html ↩

2. SonarQube https://www.sonarqube.org/ ↩

3. OakPal http://adamcin.net/oakpal/oakpal-maven-plugin/usage.html ↩

4. Code Quality Testing: https://docs.adobe.com/content/help/en/experience-manager-cloud-manager/using/how-to-use/understand-your-test-results.html#code-quality-testing ↩

5. SonarQube Rules https://rules.sonarsource.com/java ↩ ↩

6. CQ Rules https://docs.adobe.com/content/help/en/experience-manager-cloud-manager/using/how-to-use/custom-code-quality-rules.html ↩ ↩

7. AEM Rules https://github.com/Cognifide/AEM-Rules-for-SonarQube#good-practices ↩ ↩

8. SonarQube Issues https://docs.sonarqube.org/latest/user-guide/issues/ ↩ ↩