Using github, make a fork of this week's repository so you can make changes to it in your own workspace. The repository is:

<https://github.com/dlambrig/Continuous-Delivery-with-Docker-and-Jenkins-Second-Edition.git>

1. **Run the jacocoTestCoverageVerification test in your pipeline. The pipeline will fail with the error:**

***> Rule violated for bundle calculator: instructions covered ratio is 0.0, but expected minimum is 0.2***

**Describe in your own words the meaning of this error and how you can fix it. Then,** **modify the minimum limit as specified in file build.gradle so that the test will succeed. Commit your change to your forked repository. Show Jenkins logs running the pipeline before and after your change.**

Code coverage refers to the number of lines of code executed during a test. A greater ratio of lines run can sometimes be an indicator of how well the test is suited for the code, as well as whether critical code paths are being used.

In this case, code coverage is expected to be about 2%, but our tests only result in 1% or less. For purposes of demonstrating a successful run, set the coverage metric in build.gradle to ‘0.1’, which then gives a successful outcome.

Logs

Failure:

Graphical user interface, text, application, email

Description automatically generated

Success:

Text

Description automatically generated

1. **Add a stage to the lab1 pipeline to run the jacoco checkstyletest. Add a publishHTML section to generate a report named "jacoco checkstyle". The github repo will fail. Take a screenshot of the generated report to show what failed and describe the failure. Modify the java file to correct the failure and commit the change to your fork. Then rerun the test, and show it passes with a screen shot. Submit your pipeline and Jenkins screenshots.**

Failure Run

Graphical user interface, text, application, email

Description automatically generated

Success run (add JavaDoc Comment /\*\* calculator main \*/)

Text

Description automatically generated

1. Modify the checkstyle configuration config/checkstyle/checkstyle.xml.Add the rule "ConstantName". You can find documentation here. Commit it to your forked repo. Run the checkstyle test in the pipeline. It will fail. Take a screenshot of the report generated. Fix the mistake and commit it to your fork. Rerun and show it passes.

Failed run after adding ConstantName module

Graphical user interface, text, application, email

Description automatically generated

Success (after editing umlNUMBER1)

Graphical user interface, text, application, letter, email

Description automatically generated

1. ***Many shops have replaced dedicated testing employees (QA engineers) with employees who do devOps, where most tests are automated within a pipeline. Why? What are advantages and disadvantages to making this change?***

I don’t know if I would say they have “replaced” them, rather that they have “evolved” them. A properly skilled DevOps employee should be a powerful interface between the development team, the infrastructure, and the customer. Proper DevOps practices mean each piece of new code is tested when it is written, not when the project itself is complete. Code can immediately be addressed, separate from other efforts, when a failure is found. This speed of failure is what drives the value statement of a agile project.

I am sure there are some disadvantages to this strategy, a dedicated QA person can focus solely on testing – whereas a DevOps person is probably juggling a few more hats. I think the benefits far outweigh the drawbacks as code is delivered faster and with fewer bugs.

1. ***A unit test checks a given piece of code. Suppose code coverage is very high such that almost all of the code is checked this way. Given high coverage, is this sufficient testing? Why or why not? What else would ensure high quality?***

While high coverage testing is very beneficial to the development process, it is only one tool in the toolbox. High code coverage means the code does what the coder told it to – which might not be what was intended all along. Automated testing does not eliminate the need for functional testing and acceptance by developers, stakeholders, and end-users.

While the code might “work”, only a person can decide if it is doing what was intended. This is particularly true of fuzzy logic situations where the outcome is not as deterministic as say, a math equation. An example might be a simple routine to list out all customers in a particular city, without verifying the integrity of the results (and corresponding source data), you may not be able to truly tell if the test is correct/acceptable.