



DevOps: Jenkins Project

Name: Francis Hall

Student Number: C00220910

Date: 08/01/2020

1. Create a Pipeline

The first step is to create a pipeline via the Jenkins Dashboard. This process involves creating a Jenkinsfile to be used in the process. A Jenkinsfile is a specifically structured text file which contains the instructions for carrying out each stage of the continuous delivery pipeline. The stages, and the steps contained within, are carried out in sequence when the pipeline is built. In this case, the Jenkinsfile defines two stages.

The first stage, FetchStage, involves fetching the necessary source code from the given Github repository. This repository contains the java class and JUnit4 test file provided in the project brief.

The second stage, BuildStage, involves, using the relevant commands, compiling the two java files which are used in the process, Student.java, and StudentTest.java.



Fig 1.1: Jenkinsfile containing first two stages

Once the pipeline is created using the corresponding Jenkinsfile, it may then be built via the Jenkins Dashboard. When the pipeline is built, a Stage View is shown on the dashboard. The Stage View provides some useful information, including the date and time of the build and the progress of each stage, the time each stage takes to complete, and whether that stage has resulted in a success or failure. A log is also available for each stage, which may provide details on the steps such as error messages in the event of a failure.

In this case, the Stage View resulted in a success, as FetchStage and BuildStage completed without error.

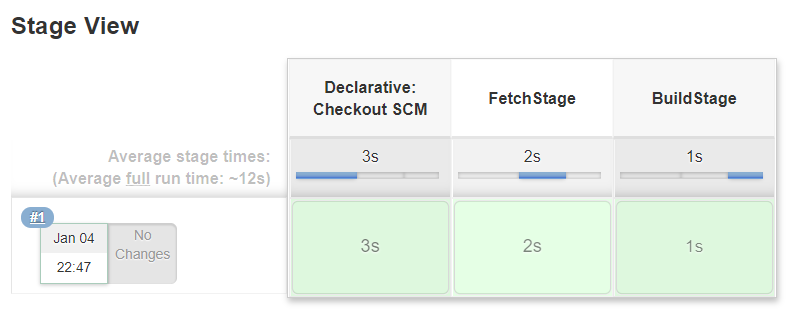


Fig 1.2: Stage View of pipeline after first build

1. Complement Your Pipeline

Once the first build is complete without errors, the next step is to add a third and final stage to the pipeline by editing the Jenkinsfile. The final stage is to be a testing stage, aptly named TestStage, using Junit, to run the tests provided in the studentTest.java file.

TestStage, like the previous stages, includes one step. This step involved running the previously compiled studentTest file with the aid of the external libraries added to the repository.



Fig 2.1: JenkinsFile after addition of TestStage

After the changes are made to the JenkinsFile, the pipeline is built again.

The pipeline fails, as expected, during the TestStage. This is denoted by the red colour on the Stage View as shown in Fig 2.2 below.

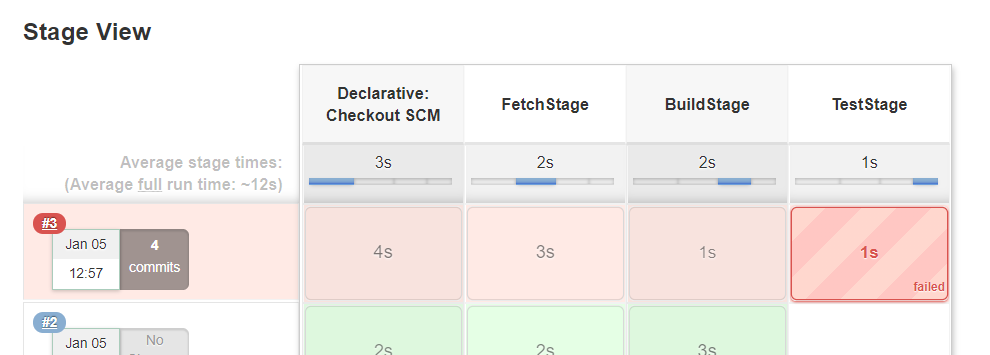


Fig 2.2: Stage View after Test Stage is added

The log for the TestStage provides more specific details of the failure, as shown below in Fig 2.3. In this case, three of the thirteen tests have failed.

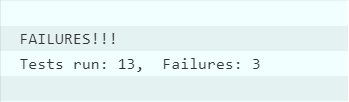


Fig 2.3: TestStage log showing number of failed tests

1. Fix the Code

The final step was to edit the source code to ensure that the provided tests pass the specification of the getAttendanceGrade method. The log from the previously failed tests are useful here as it indicates which tests have failed and what the actual results were.

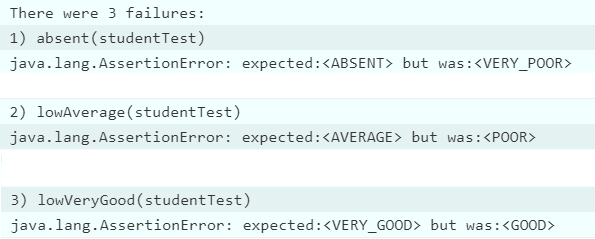


Fig 3.1. Section of log showing failed tests and expected results

The Student java class is then edited to match the table provided in the brief. The updated Student class is shown below in Fig 3.2.

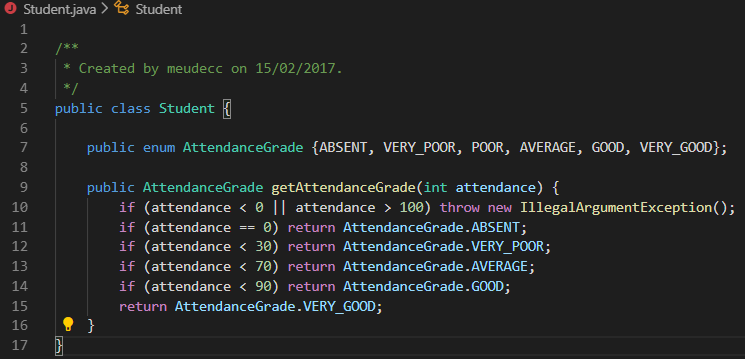


Fig 3.2. Updated Student class

Once the Student class is updated and committed to the repository, the pipeline may be built again.

As shown in Fig 3.3 below, the Stage View indicates the TestStage has completed successfully, and therefore, the pipeline as a whole has resulted in a success across all of its stages.

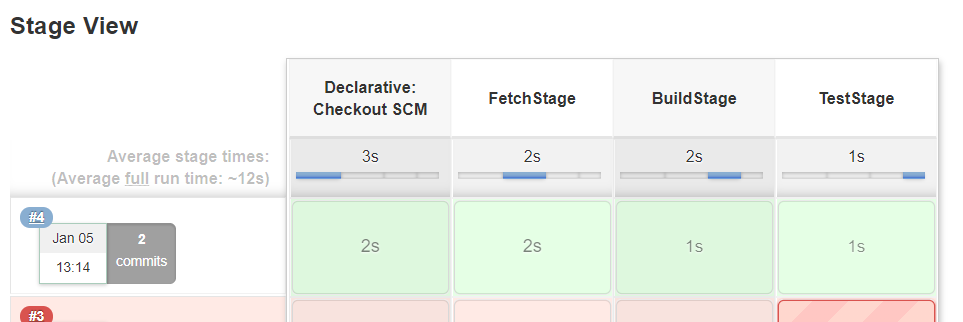


Fig 3.3: Stage View showing successful completion of pipeline