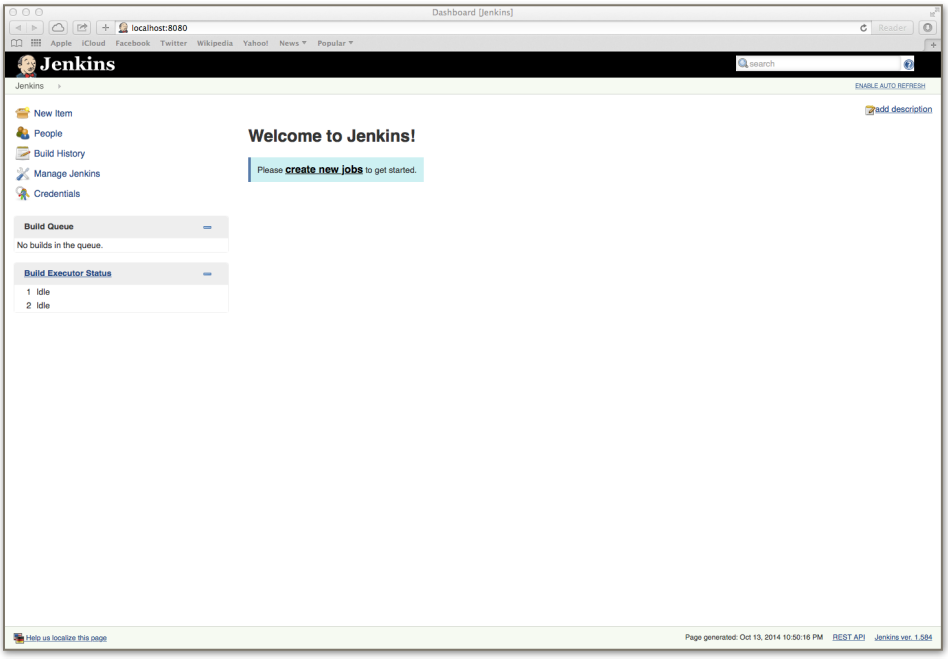
LAB 3 - Jenkins

The following document contains screenshots of the various steps carried out for setting up the lab for Jenkins. In addition to that, it contains the screenshots of expected deliverables of Jenkins showing the status with different iterations of the code.

Before installing Jenkins, a test run of the code provided in the lab using “nosetests” gave the following output

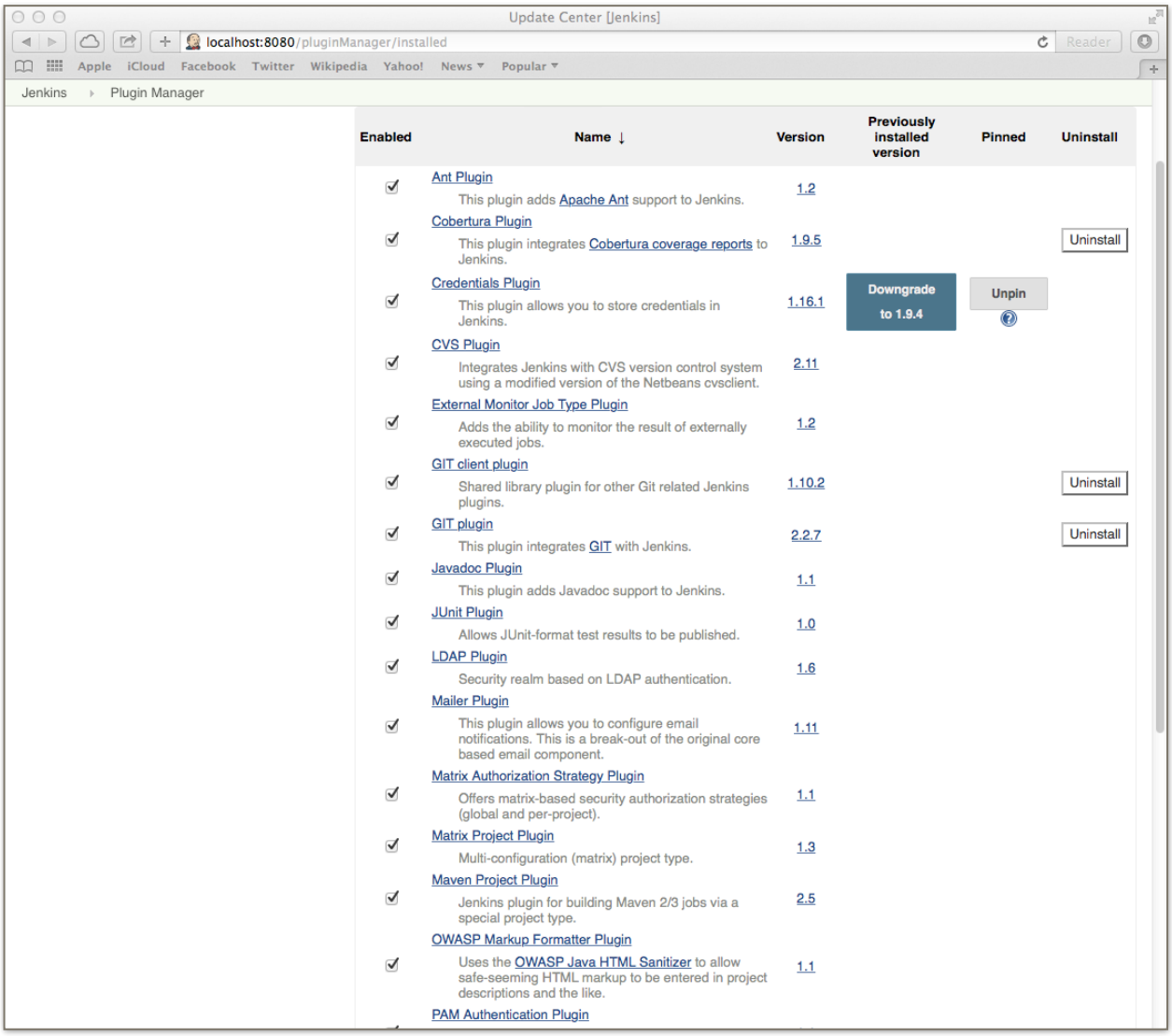
```
jenkins-testing — bash — 100×30
Asims-MacBook-Pro:jenkins-testing asim$ nosetests -v --with-coverage ExperimentTest.py
test_t1 (ExperimentTest.MyTestCase) ... ok
test_t2 (ExperimentTest.MyTestCase) ... ok
test_t3 (ExperimentTest.MyTestCase) ... ok
test_t4 (ExperimentTest.MyTestCase) ... ok
test_t5 (ExperimentTest.MyTestCase) ... ok
test_t6 (ExperimentTest.MyTestCase) ... FAIL
test_t7 (ExperimentTest.MyTestCase) ... ok

======================================================================
FAIL: test_t6 (ExperimentTest.MyTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/asim/Documents/UT - Austin/Classes/Fall 2014/APT/GitRepo/APT_Fall2014/lab3/jenkins-te
sting/ExperimentTest.py", line 28, in test_t6
    self.assertEqual(r1, -1)
AssertionError: 0 != -1

Name             Stmts   Miss  Cover   Missing
---------------------------------------------
Experiment           9      0   100%
ExperimentTest      25      1    96%   34
---------------------------------------------
TOTAL               34      1    97%
----------------------------------------------------------------------
Ran 7 tests in 0.002s

FAILED (failures=1)
```

After installing Jenkins for Mac, the following screen pops up indicating a successful installation. It started up on localhost:8080 port

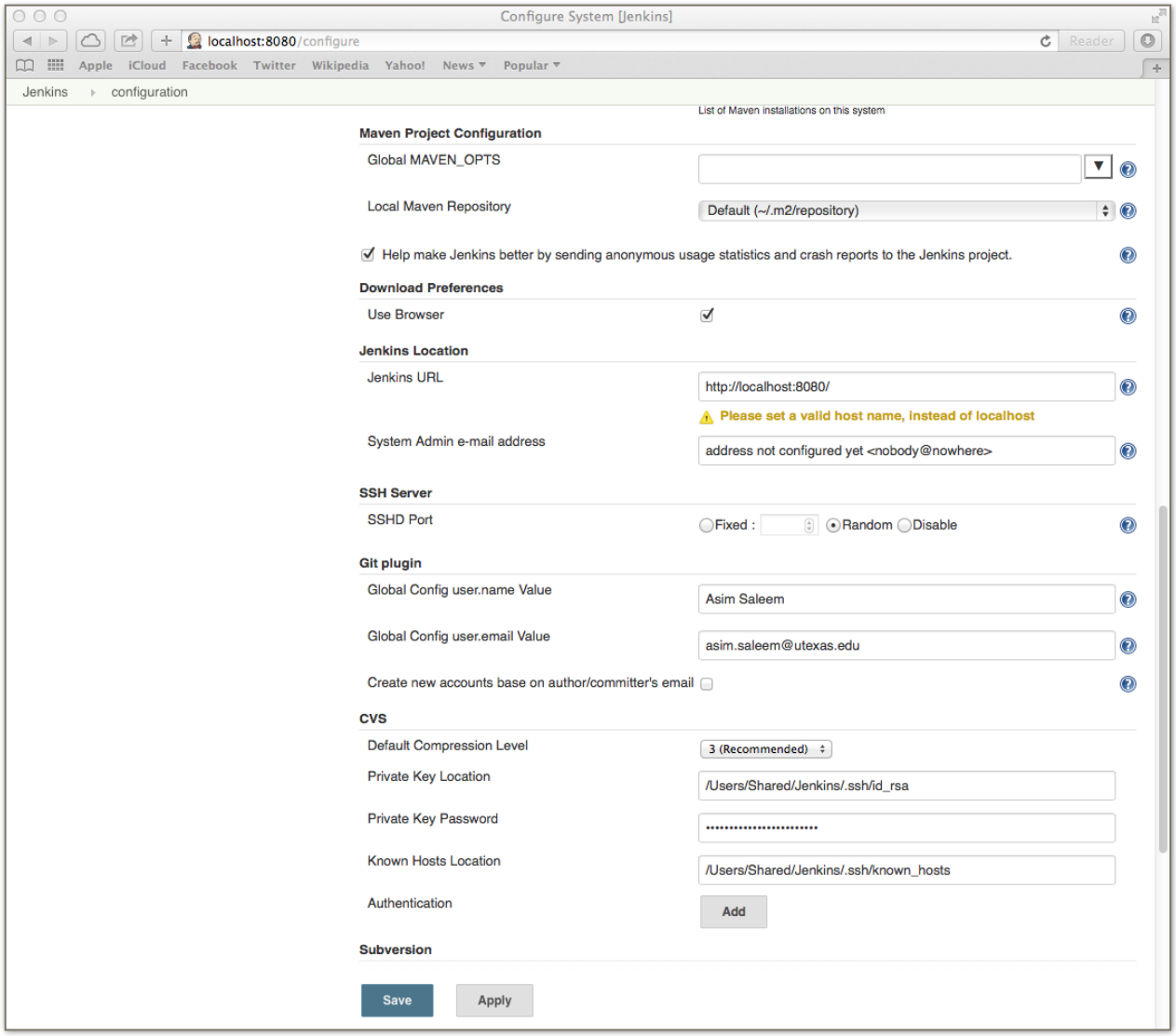


In order to continue, we need to install the plugins required for Jenkins to perform the operations we require.

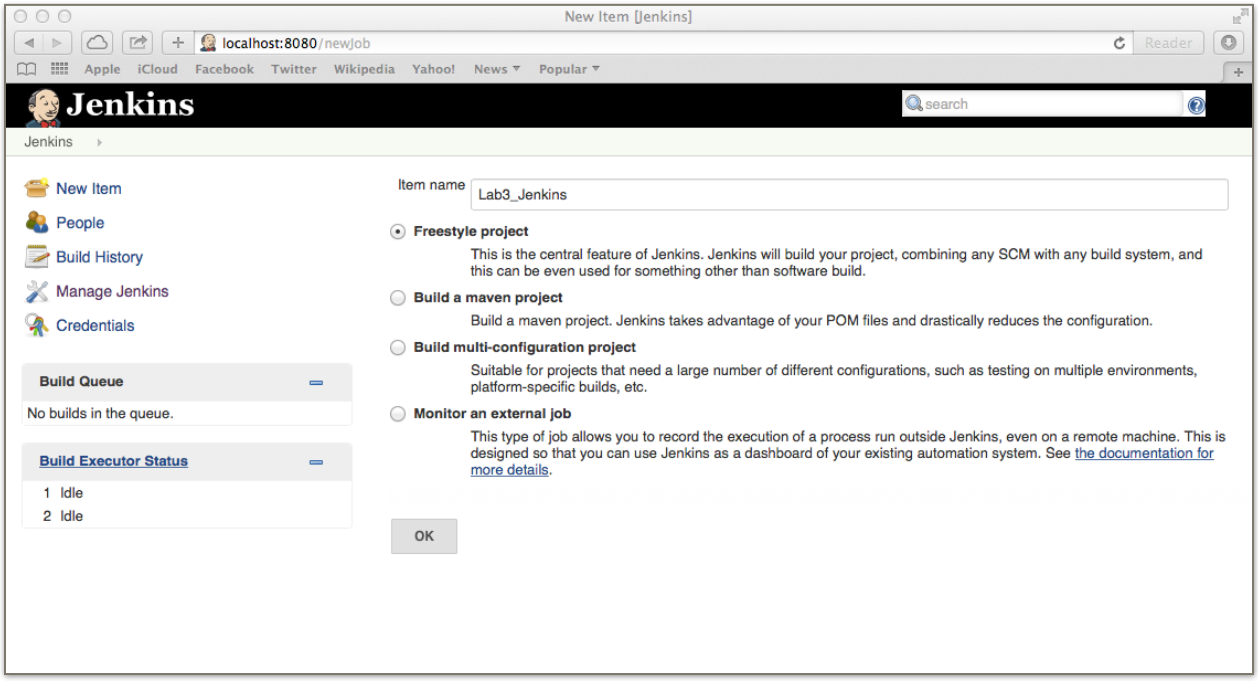
To do this, we access the Jenkins plugin repository and install the desired plugins, namely the “Jenkins GIT Plugin” and the “Jenkins Cobertura Plugin”



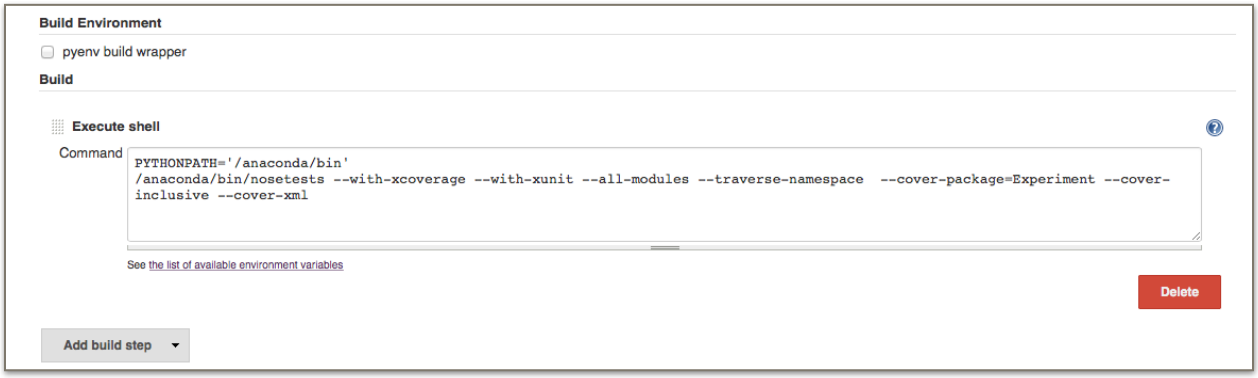
Once we install the desired plugins, it is time to setup the configuration required for the plugins to work.



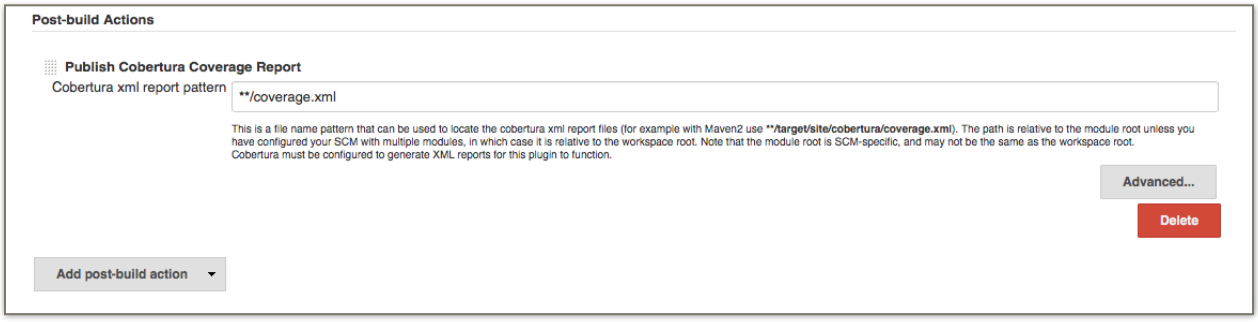
After the configurations are set, we configure the Job in Jenkins to point to the code we have already setup on Git.



We set up the Build command as follows. This takes care of executing the “nosetests” and outputting the results in a XML format

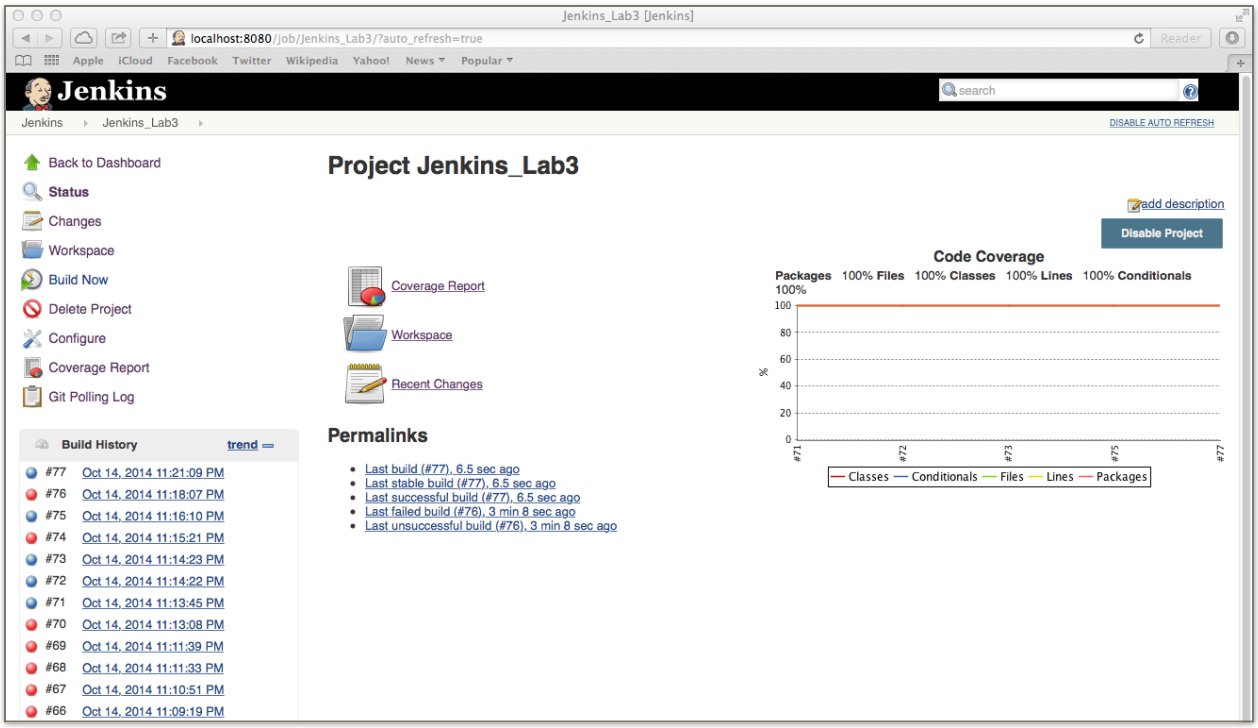


The XML file generated by the Execute Shell process in Build is then displayed using the Cobertura plugin as configured below:

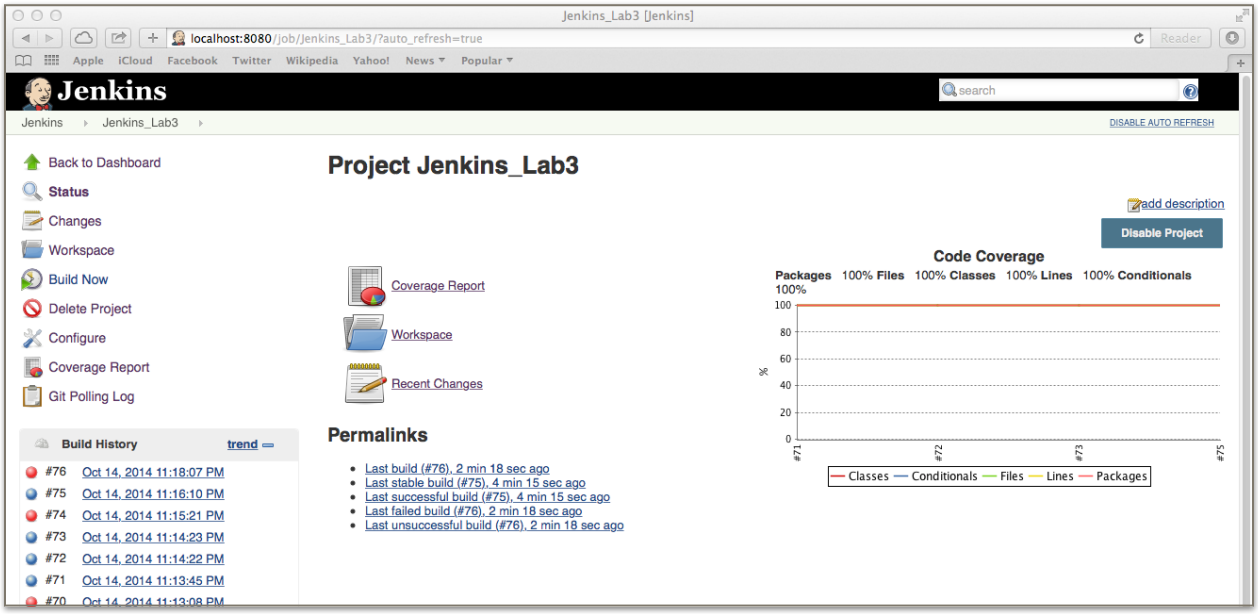


Now that the entire job has been successfully configured, we can then start seeing the status of the process whenever any code changes. Basically a code change triggers a build and the output of the build is displayed in the Jenkins UI.

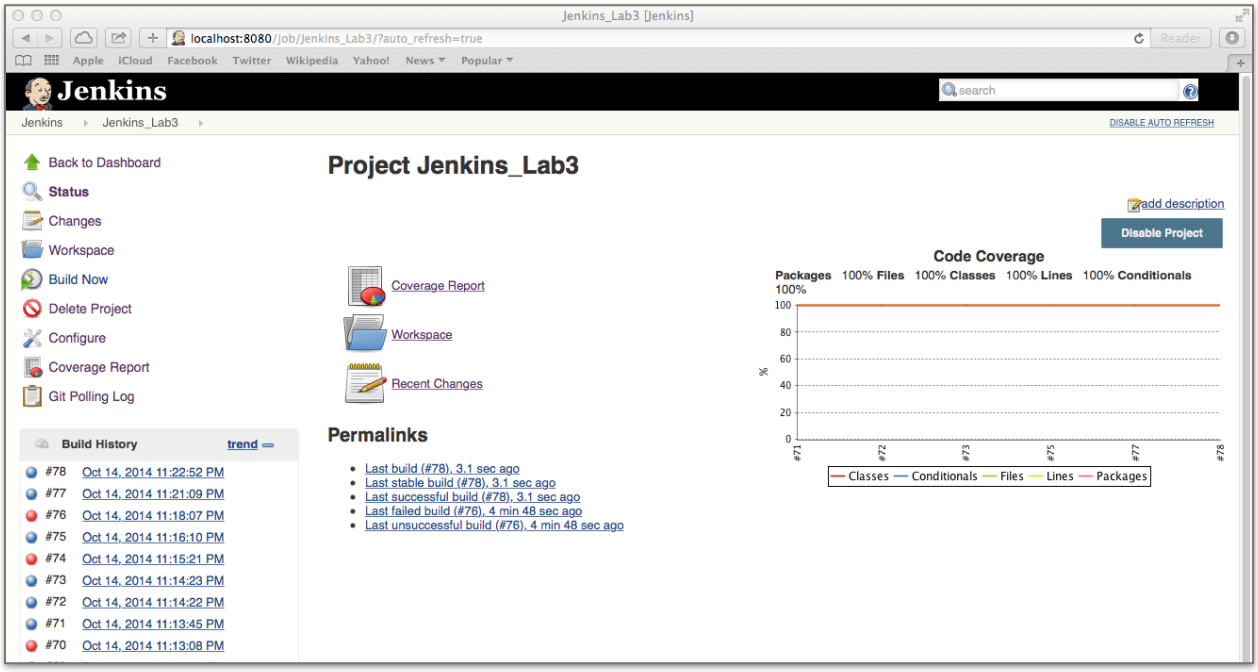
Below is an example of how it looks when the Build is Successful. In this case, the dots in the Build History section indicate the status of the individual builds. So Blue indicates success and Red indicates Failure



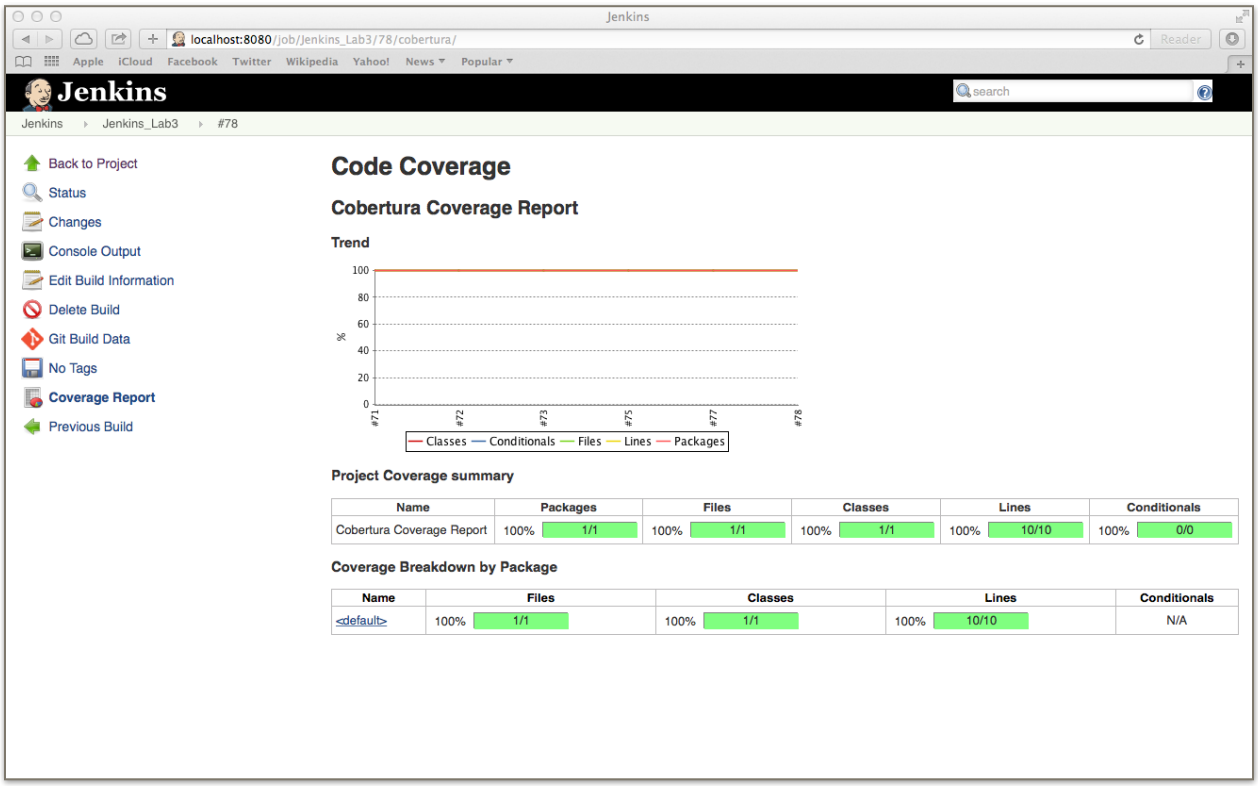
Just to indicate that the status can be Failure too, we modify the code to break the compilation process to simulate the build failing. This is captured in Jenkins as shown below:



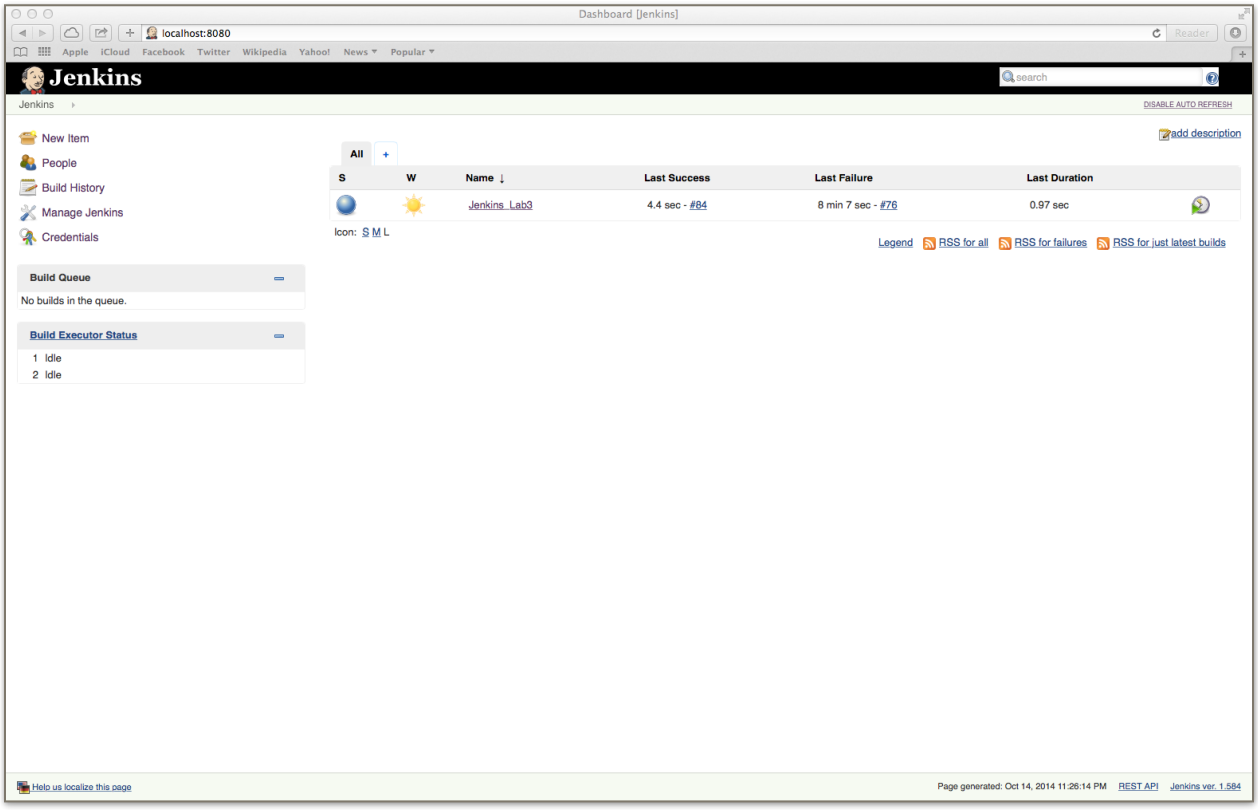
When the error is fixed and the build runs again, then we notice that the build process again changes to blue indicating the same.



By clicking on the Coverage Report link, we can also observe the coverage of the test cases that were executed.



Finally the overall status of the Job can be observed by accessing the home page of Jenkins where it is displayed as given below:



Therefore from the various screenshots that are attached, we can see how Jenkins helps us with the Continuous Integration process.