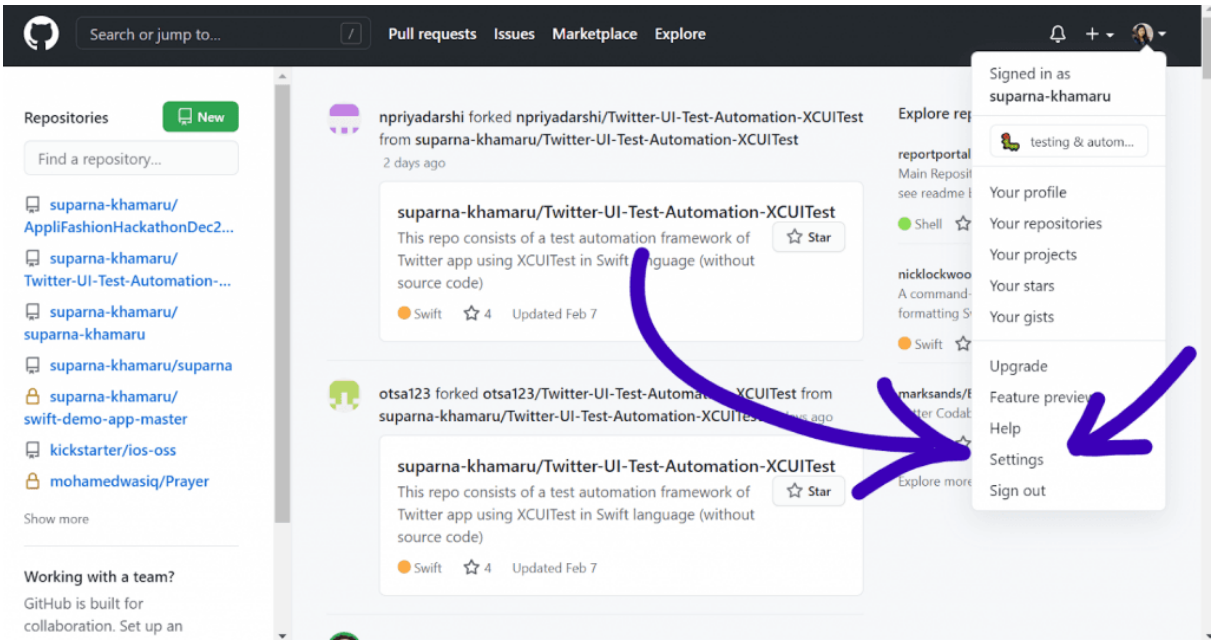
CLOUD DEVOPS(EPAM)

PROJECT

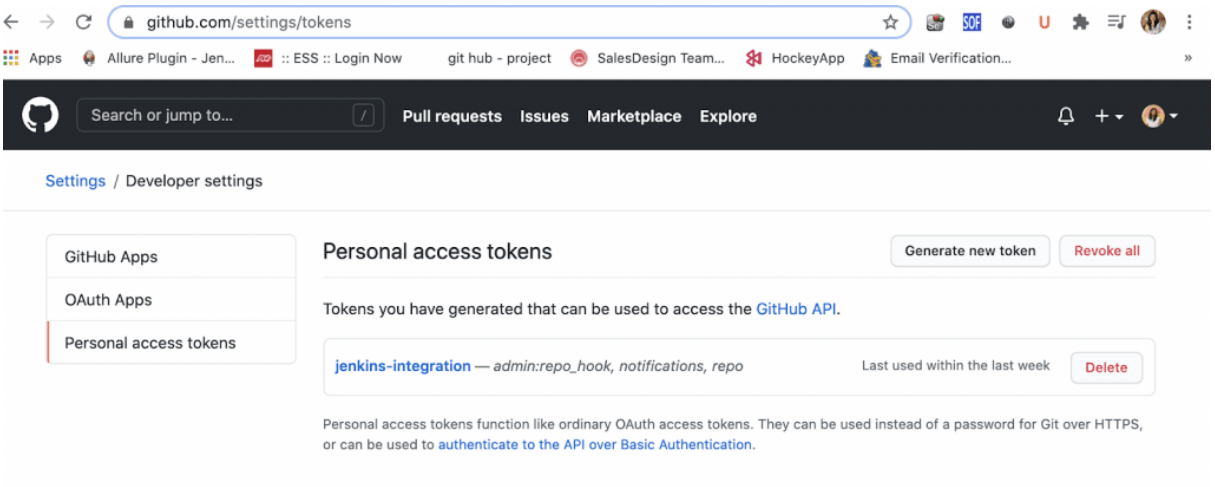
Configuring GitHub for Jenkins Continuous Integration

Step 1: Go to your Github profile and navigate to **Settings**.



Step 2: In the settings screen, click on the "**Developer settings**" menu and click on "**Personal access tokens**."

Step 3: In the "**Personal access tokens**" tab, click on the "**Generate new token**" button and provide necessary details as desired (an example is provided in the below figure), and click on the "**Generate token**" button.

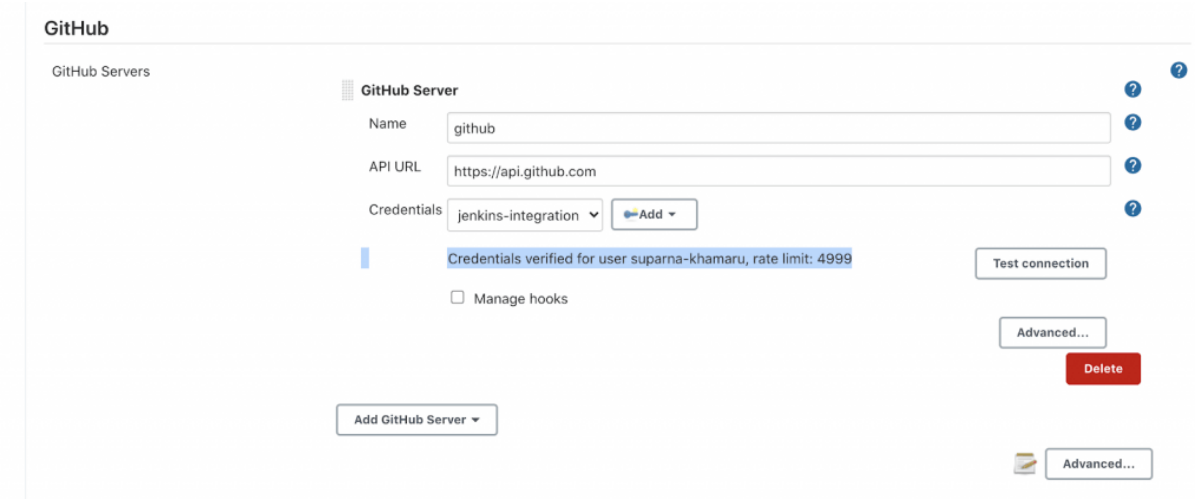


Step 4: Post successful creation of the token, the newly generated secret text in GitHub is to be copied for future usage in Jenkins.

Configuring Jenkins for GitHub Integration

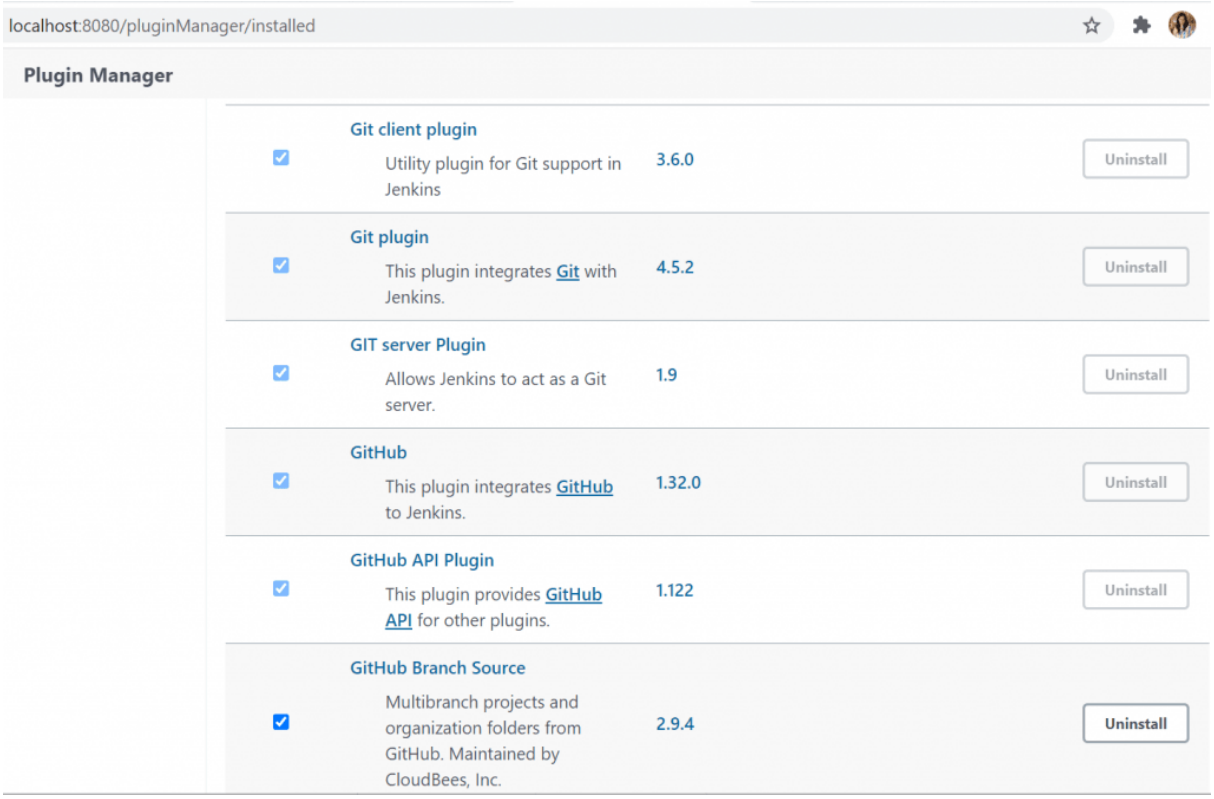
Step 5: Next, go to Jenkins dashboard.

- Click on "**Manage Jenkins**."
- Click on "**Configure System**" and go to the following '**GitHub**' section.



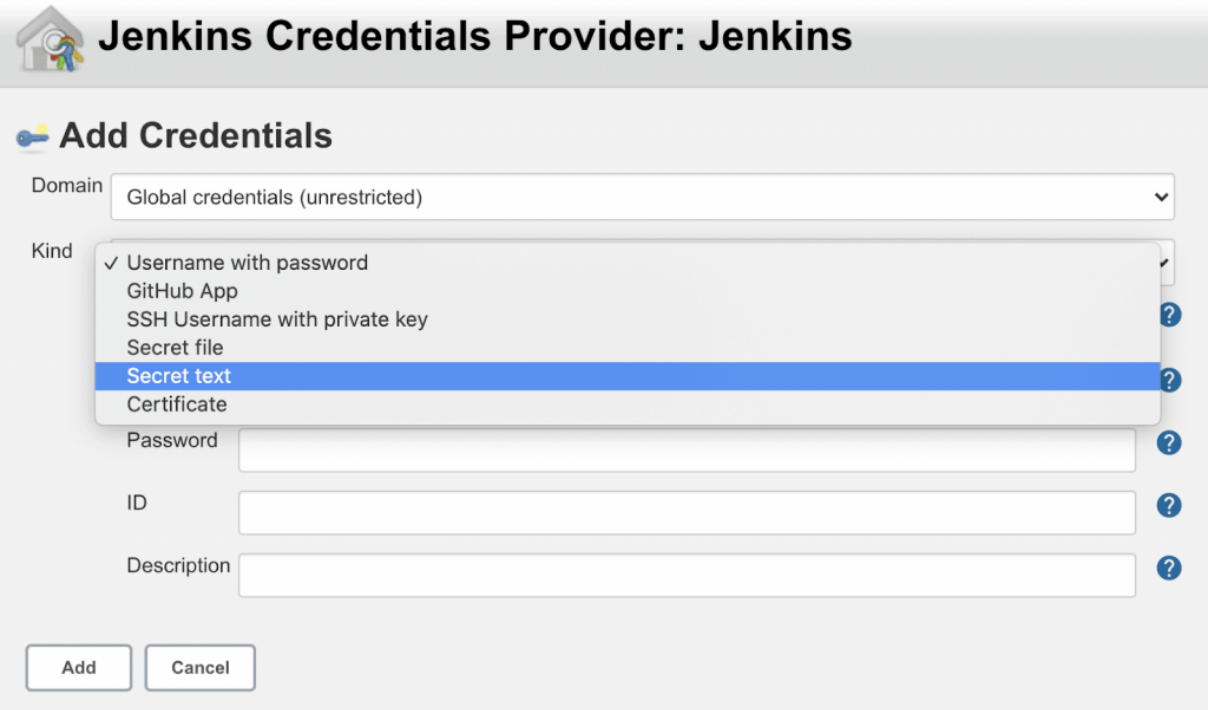
Note: If the above "**GitHub Server**" section is found missing in Jenkins, make sure to manually install the missing GitHub plugin from the installed list of tools shown below.

Steps to follow- Go to: Jenkins Dashboard -> Manage Jenkins -> Manage Plugins -> 'Available' tab -> Enter Git in search bar and filter -> Install required plugin

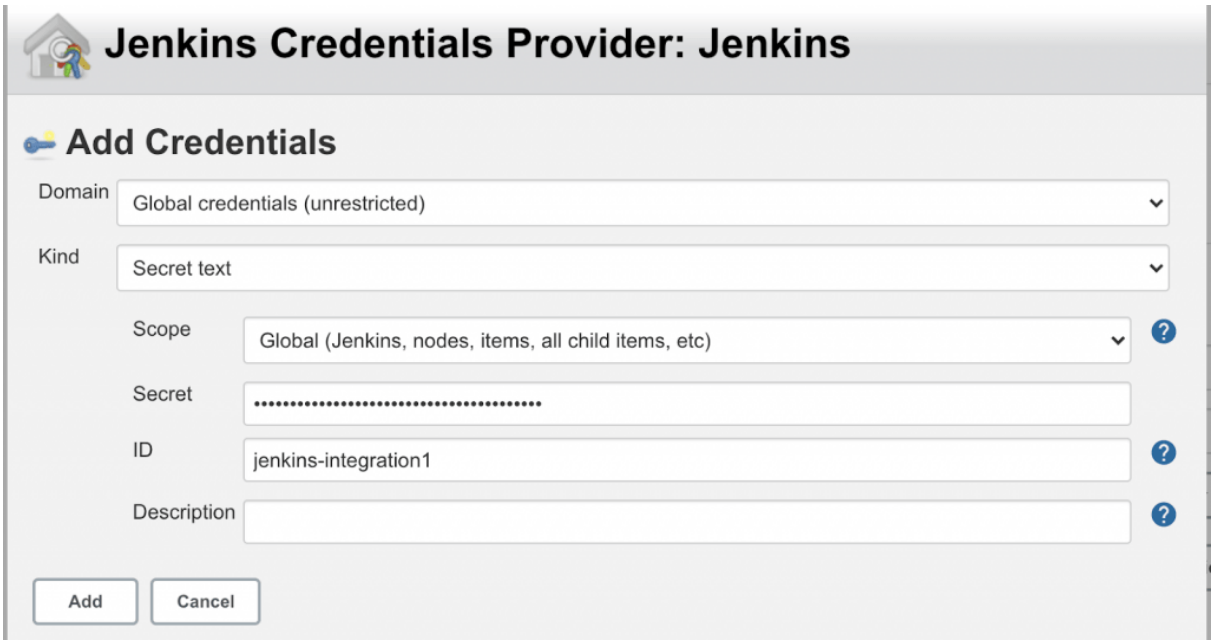


Step 6: Make sure to add the copied secret key in the above credentials by:

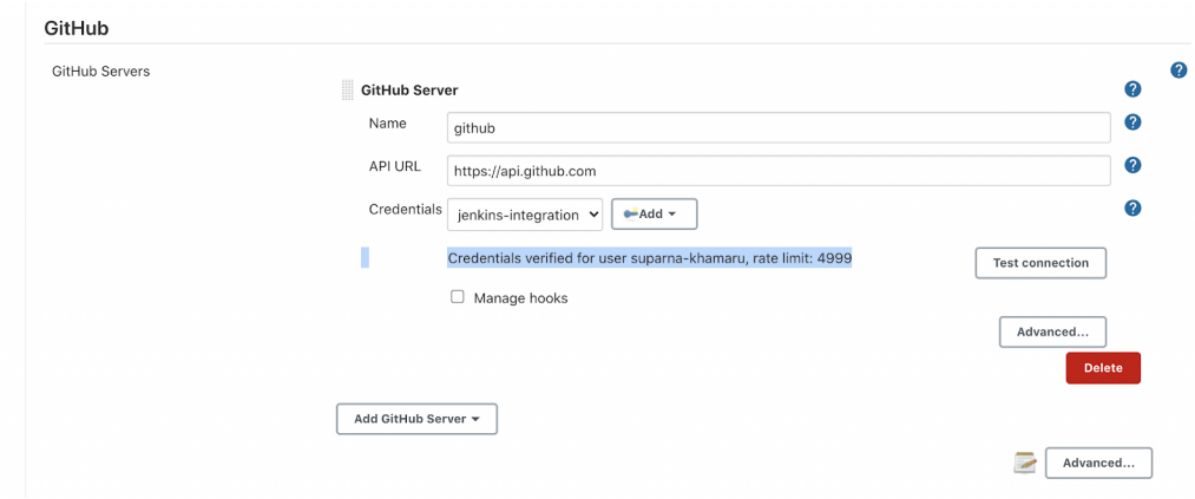
- Click on the “Add” -> “**Jenkins**” button in the Github Server section’s Credentials field.
- Select “**Secret text**” from the “Kind” dropdown.



Step 7: Paste the previously copied secret text from GitHub in the **Secret** field as displayed below, while providing an ID such as “jenkins-integration,” and then click on the “**Add**” button.



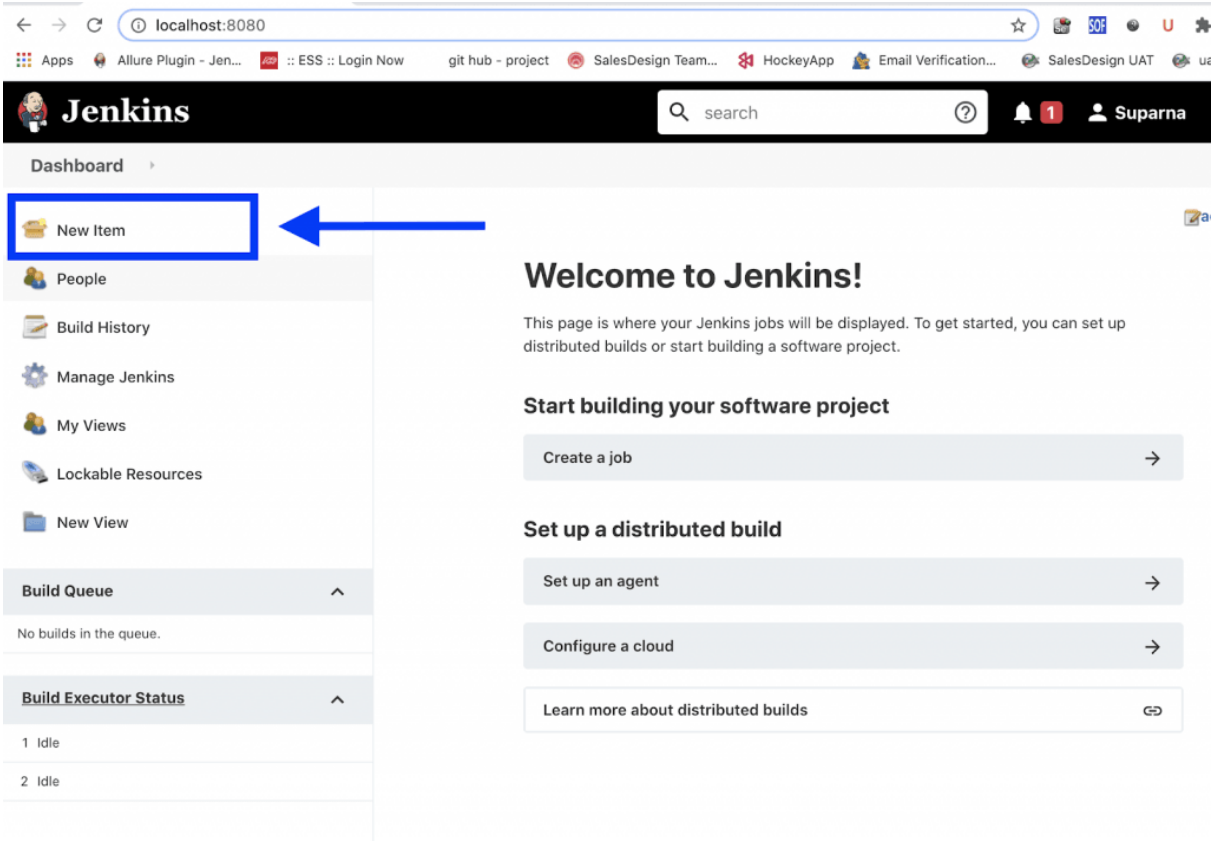
Step 8: Once the Secret text is successfully added, let us test the connection by clicking on the “**Test connection**” button and verify the confirmation message as highlighted below.



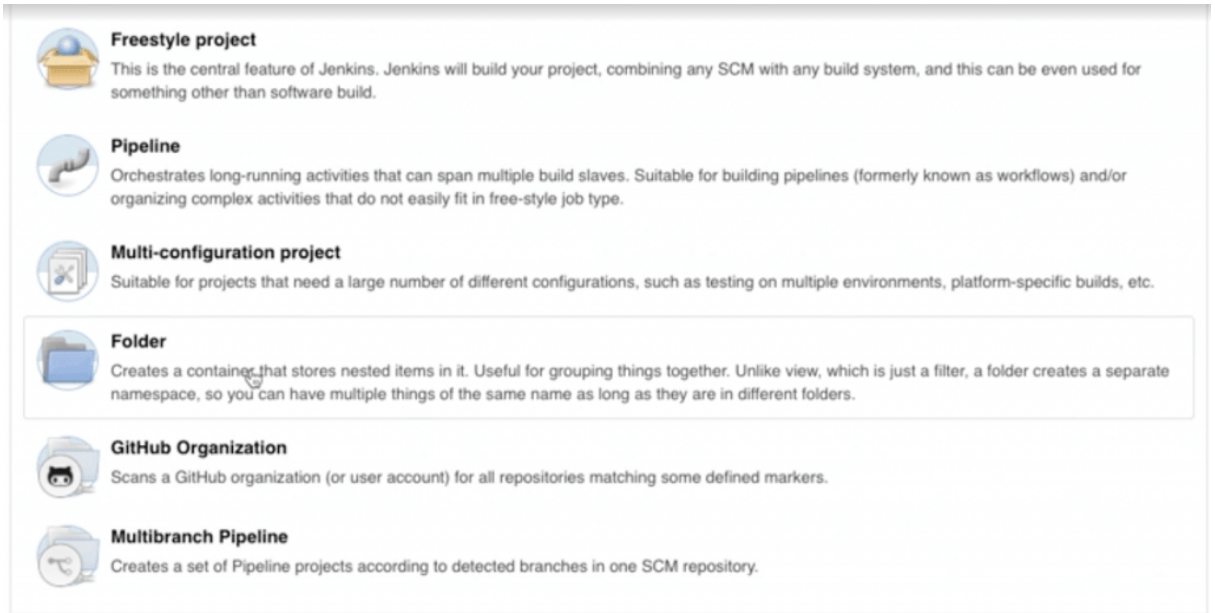
Creating Your First Jenkins Job Integrated Into a Test Project

Let us take a sample java project already integrated and configured with the Ant build tool to ease convenience. You can choose any sample project of yours if you like. Let us follow the below steps in Jenkins Dashboard to configure the same.

Step 1: Click on “**New Item**” in the Jenkins user interface dashboard on the left side.

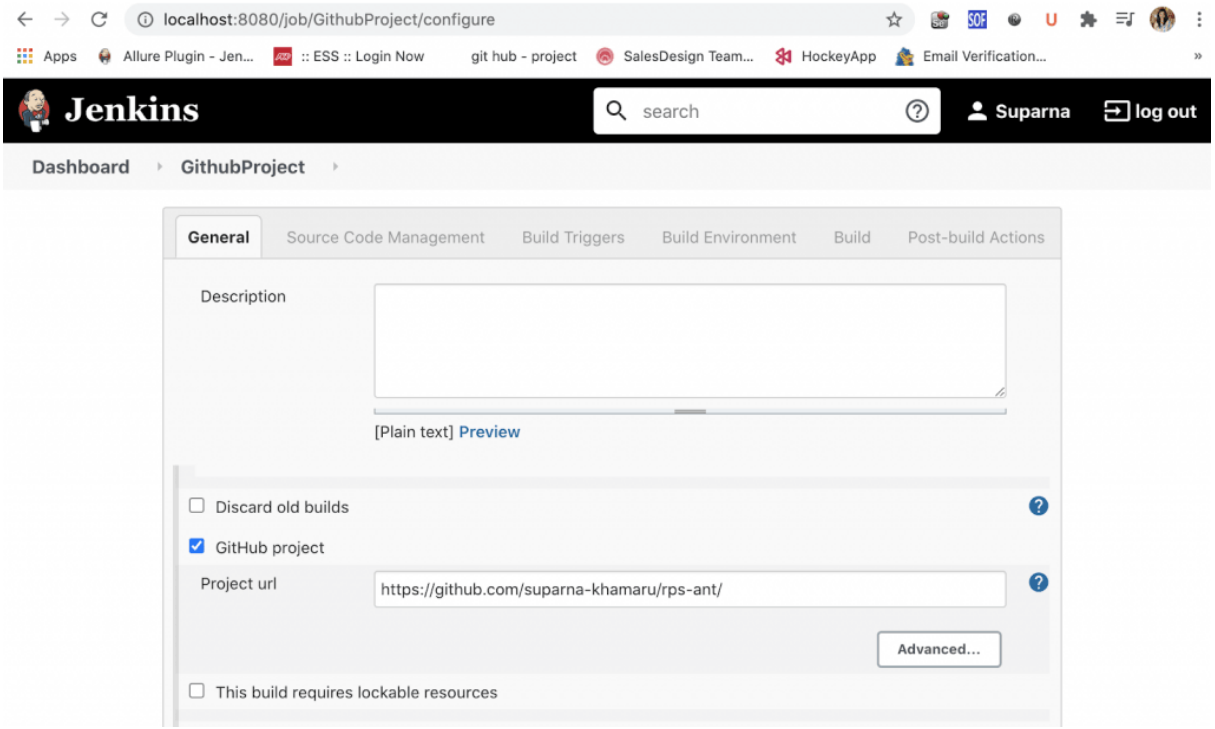


Step 2: Let us start with creating a **Freestyle project** in Jenkins for ease of understanding for a beginner. However, we can choose to select any of the following:

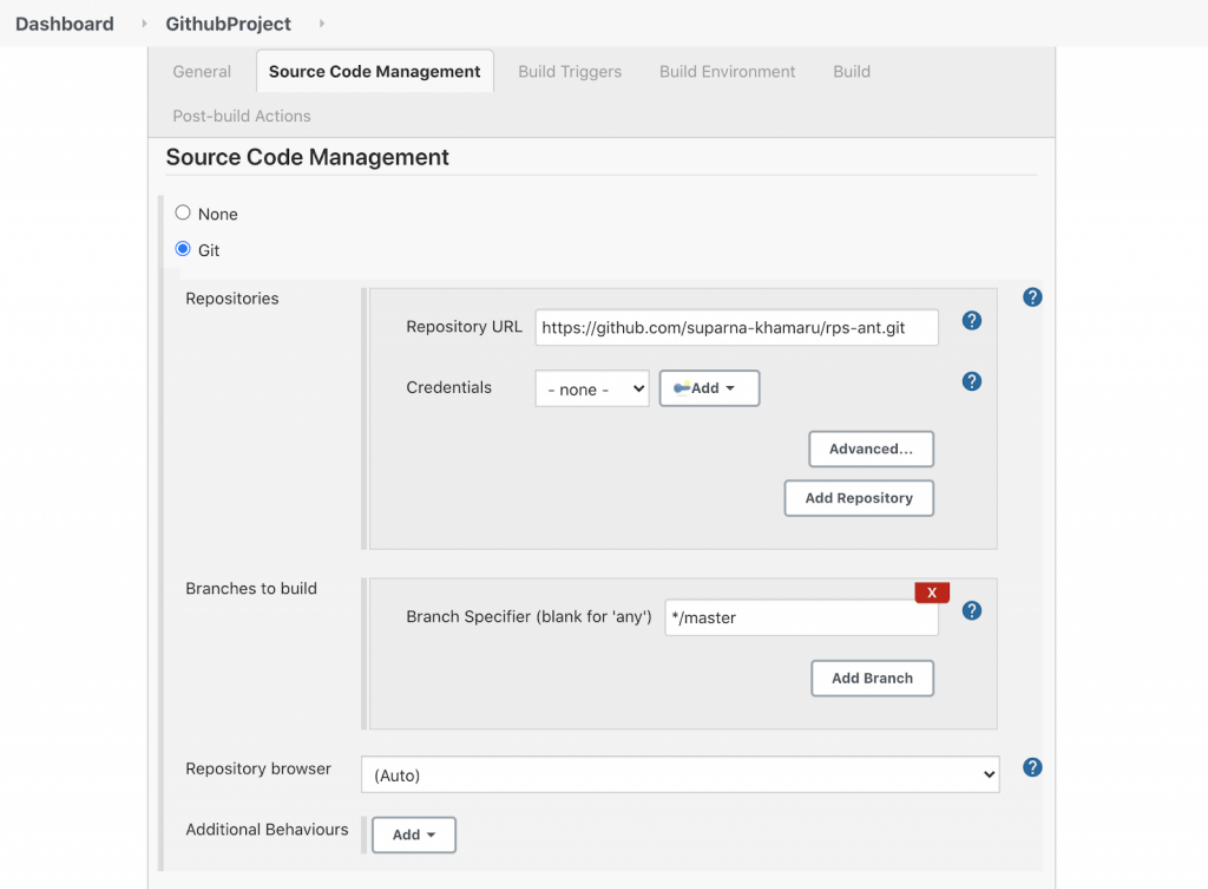


Enter a project name in the requested textbox field and select the **Freestyle project** as shown on the image above.

Step 3: In the **General** section, check the field “**GitHub project**” and enter a valid Ant project as given below.

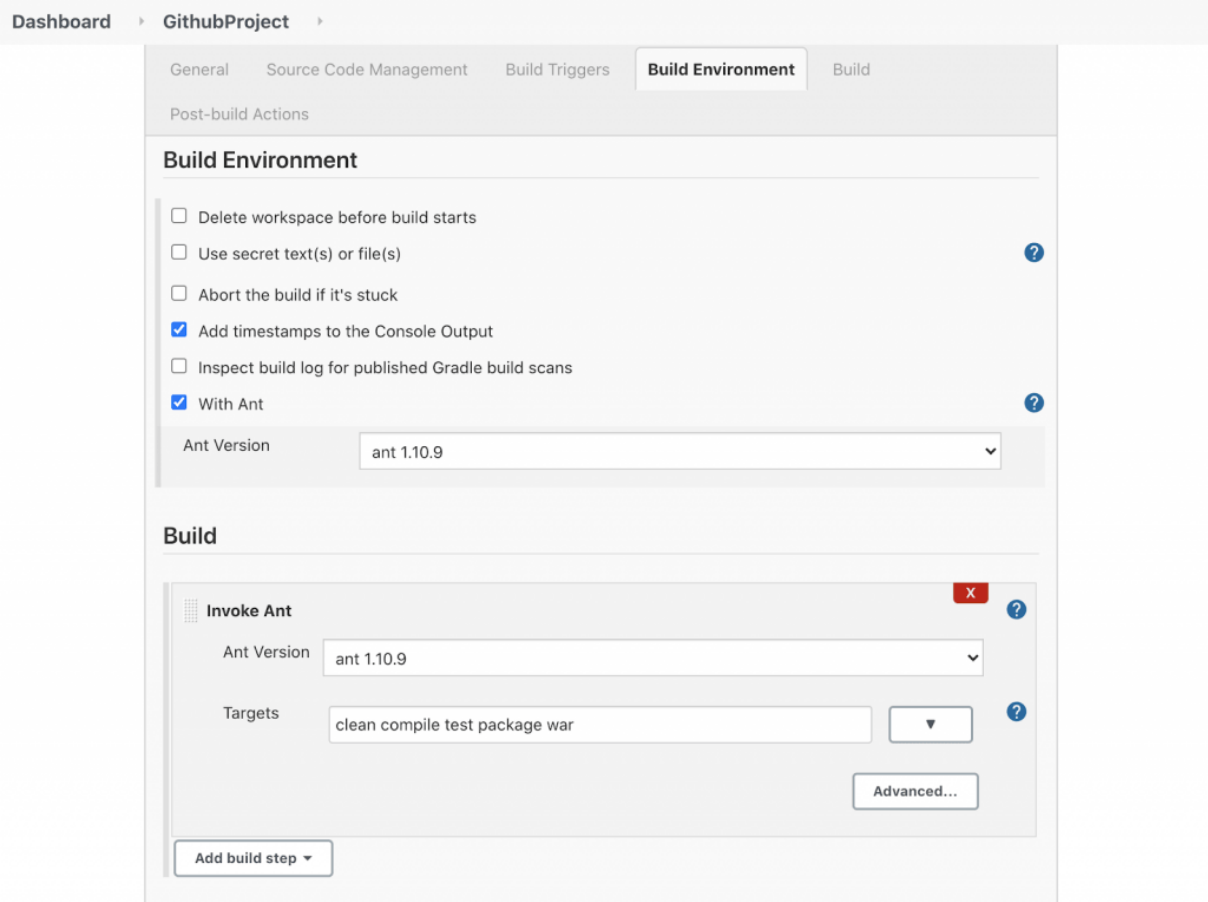


Step 4: Then, go to the “**Source Code Management**” section and enter the same link as above with an extra .git extension at the end of the URL, as shown in the below screen.

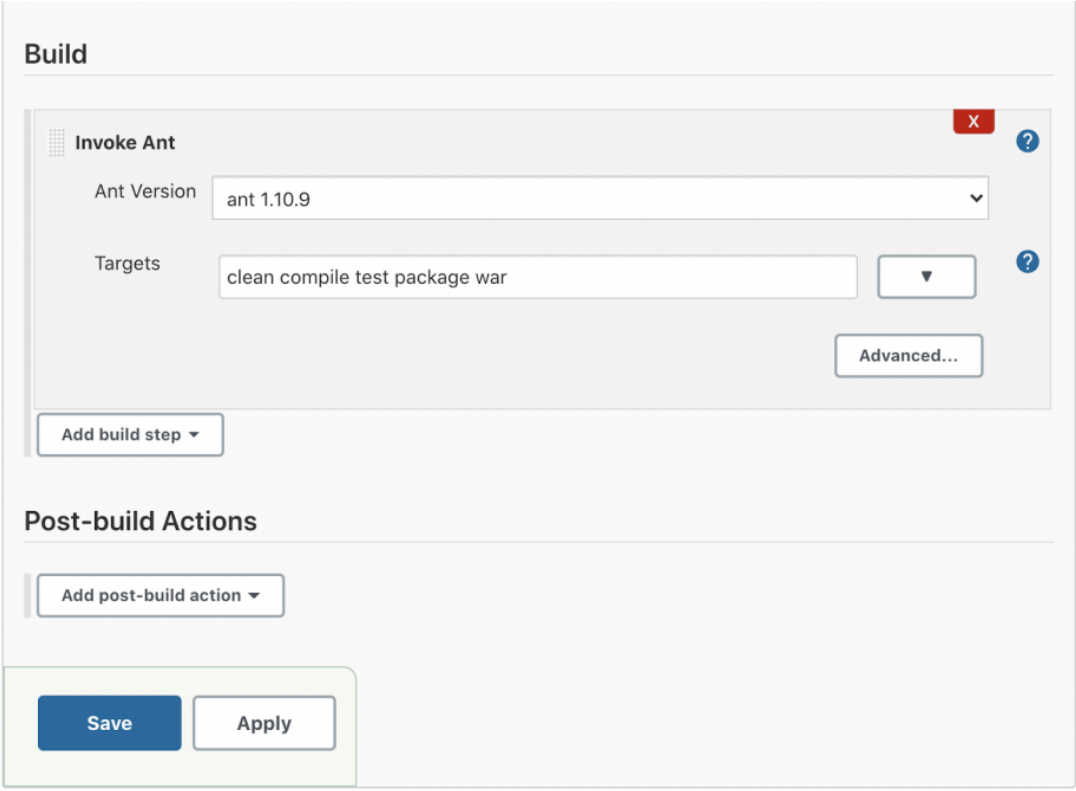


Step 5: Go to the "**Build Environment**" section and add the following steps shown in the image below. Make sure to add the command in the Targets field under the Ant Build step:

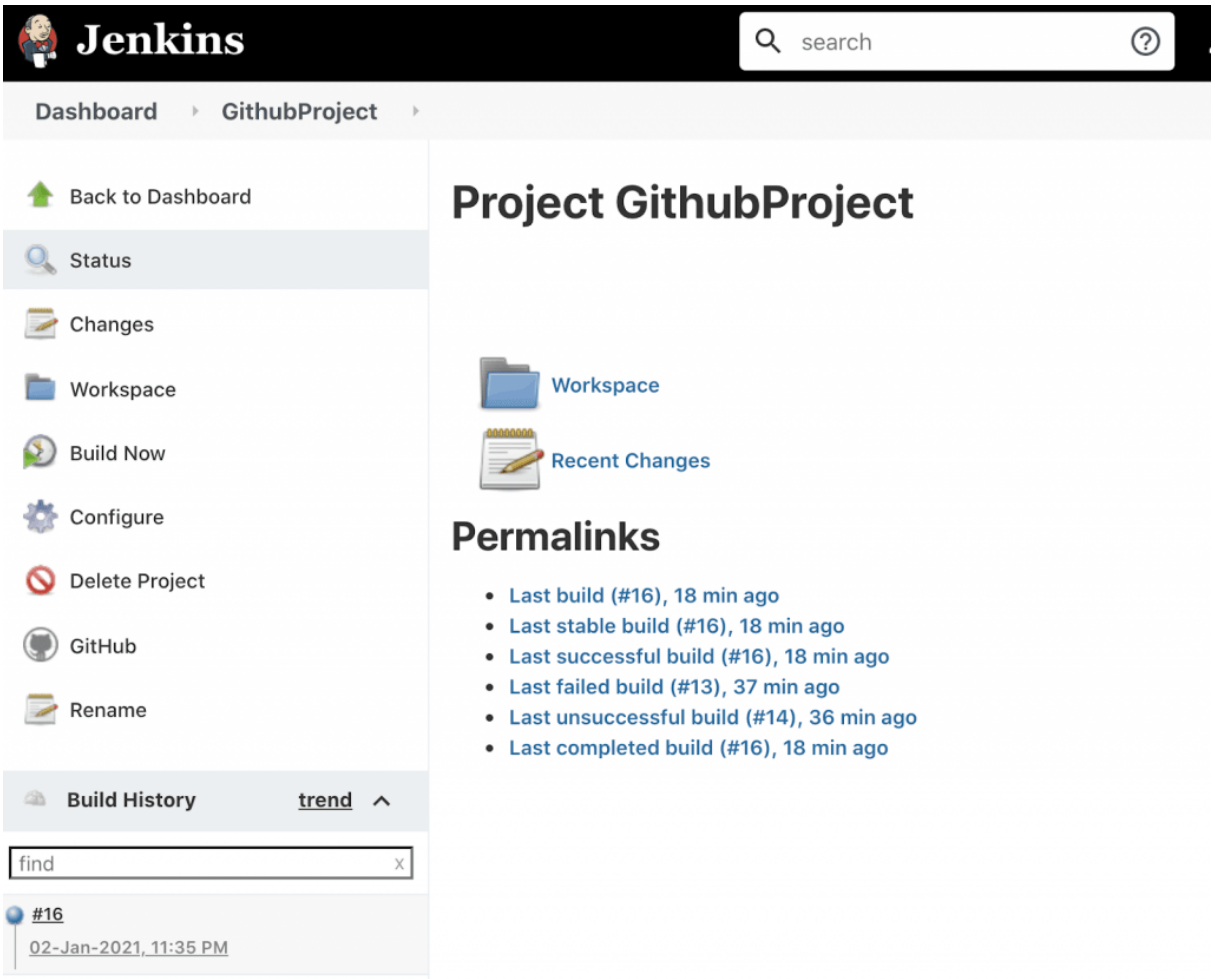
```
clean compile test package war
```



Step 6: Click on the “**Apply**” button and then click on the “**Save**” button.

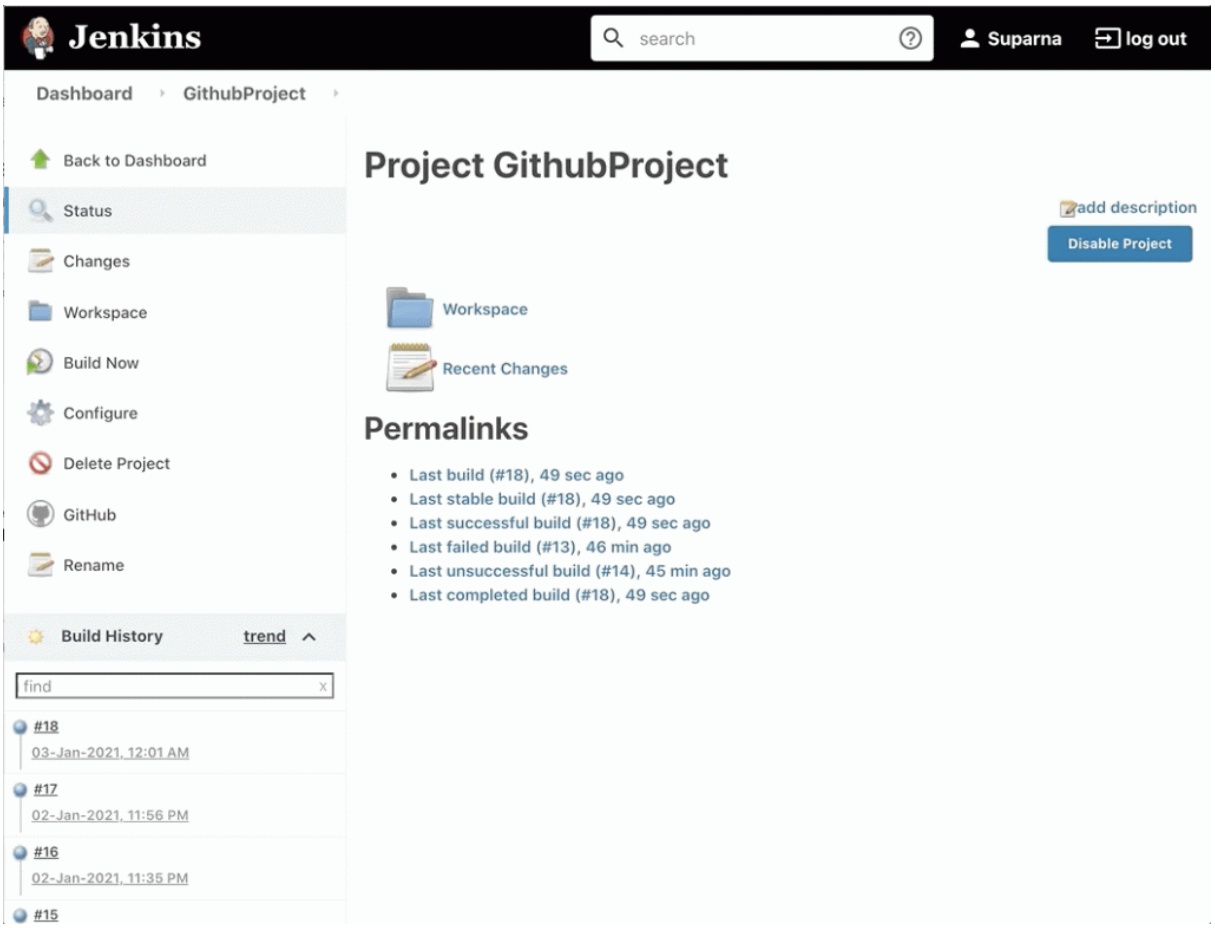


Step 7: Once the new Jenkins configurations are successfully saved, the user shall be navigated back to the Project dashboard screen, where the user now clicks on the "**Build Now**" menu on the left-hand side of the displayed dashboard screen.



Step 8: As the Build has started to run, users shall be able to note the scheduled build running in the "**Build History**" section with a visibly blinking blue round icon. Clicking it will navigate the user to the respective "**Build Status**" page.

Users must click on the "**Console Output**" menu to walk through the build logs generated in the console while building in Jenkins.

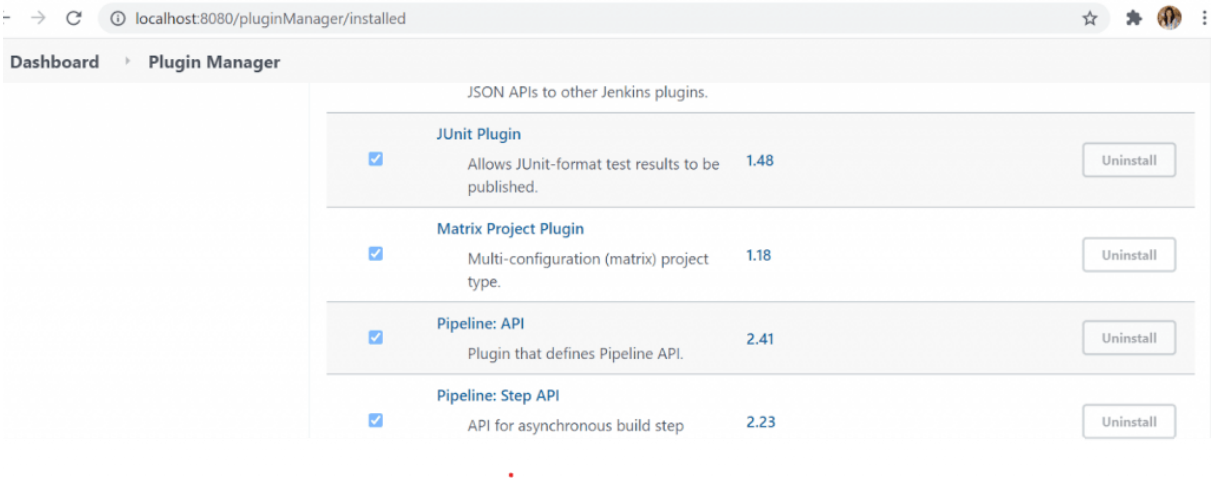


Analyzing JUnit Test Results in Jenkins

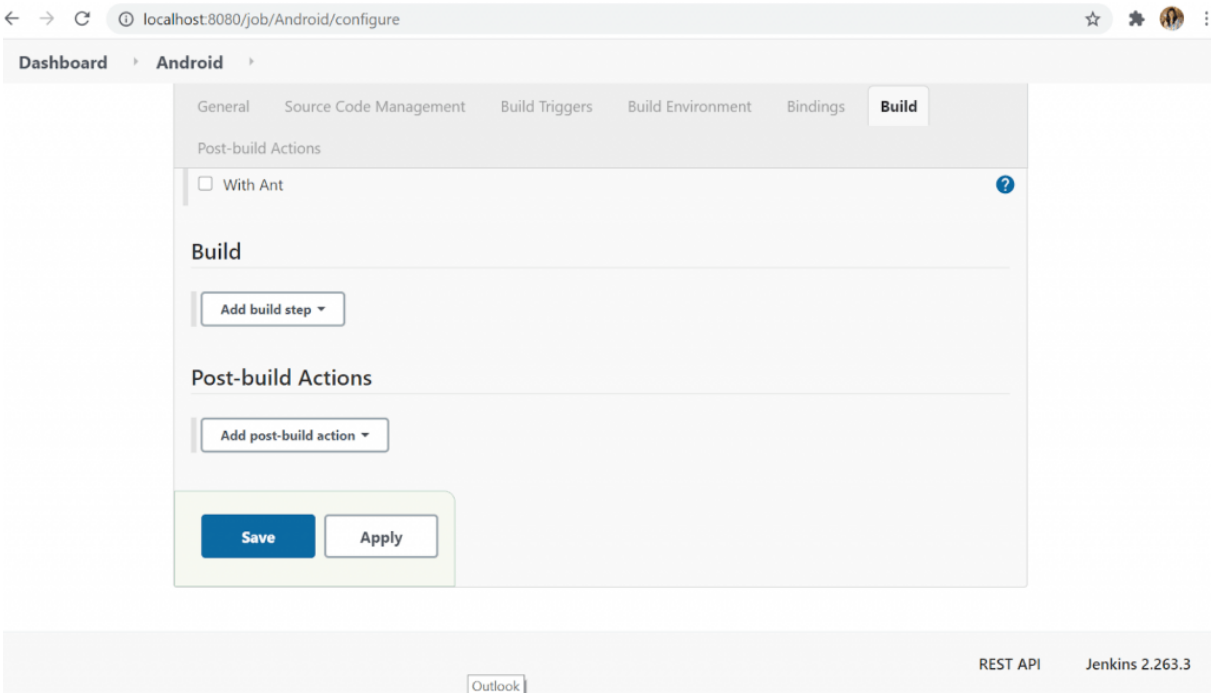
Now that we have learned how to set up continuous integration with Git Jenkins and how to create the first Jenkins Continuous integration job within a test project, it is time to take it one step ahead. After creating a job with a Java project, you might also want to analyze the test results. This can also be done using Jenkins.

Let's see how this can be done for JUnit.

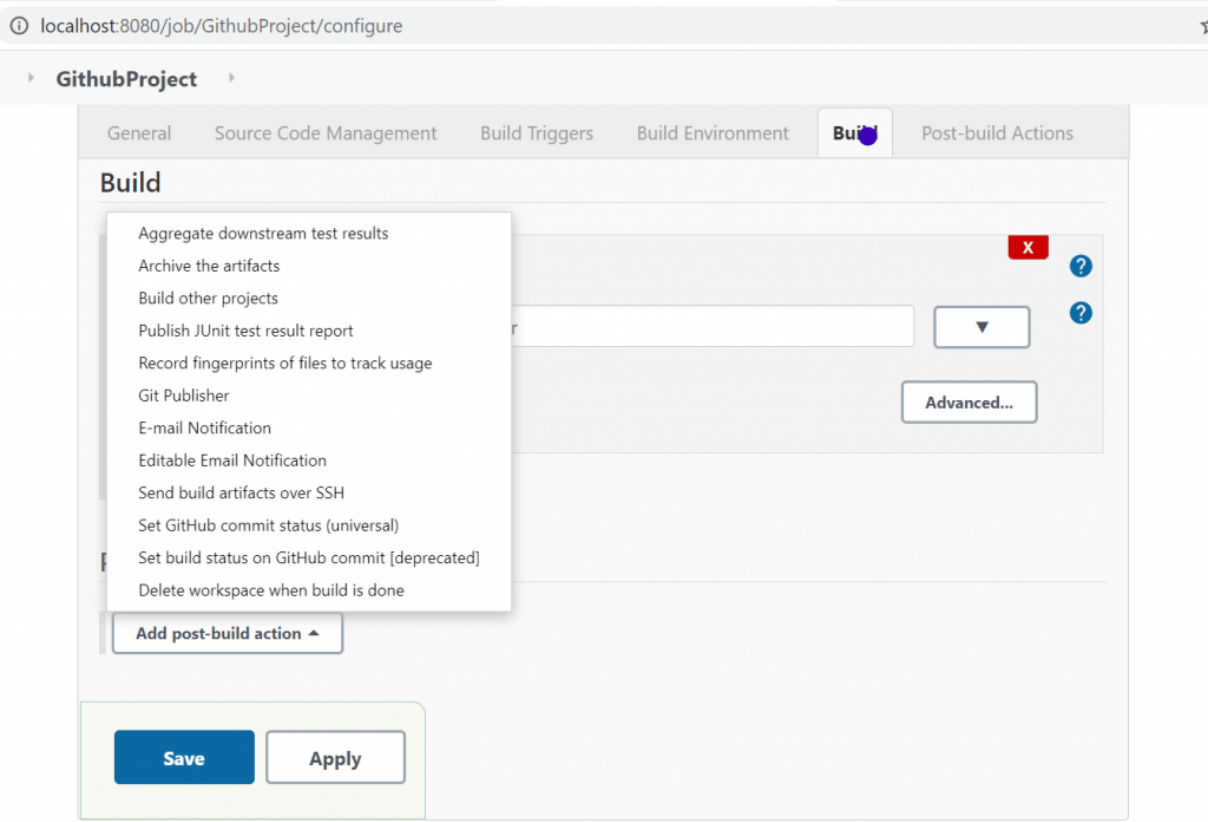
Step 1: Make sure "**JUnit Plugin**" is already installed in Jenkins.
Go to Jenkins Dashboard -> Manage Jenkins -> Manage Plugins -> Click on the '**Installed**' tab to view all the already installed plugins.



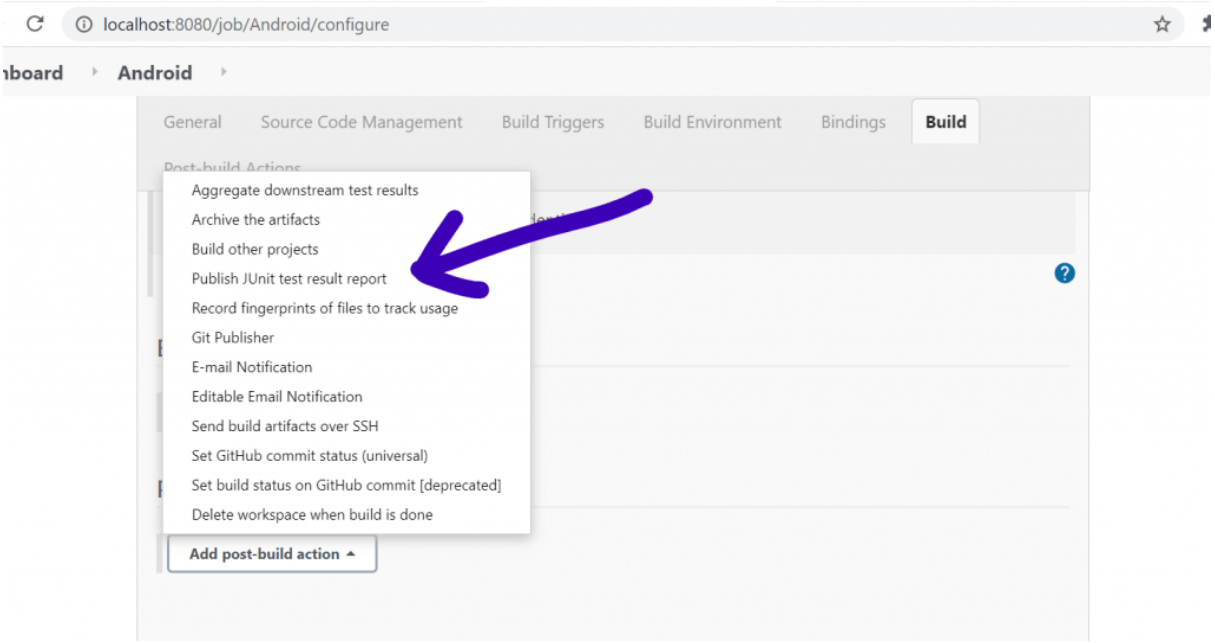
Step 2: Go to Project -> Click on '**Configure**' -> Click on '**Build**' tab.



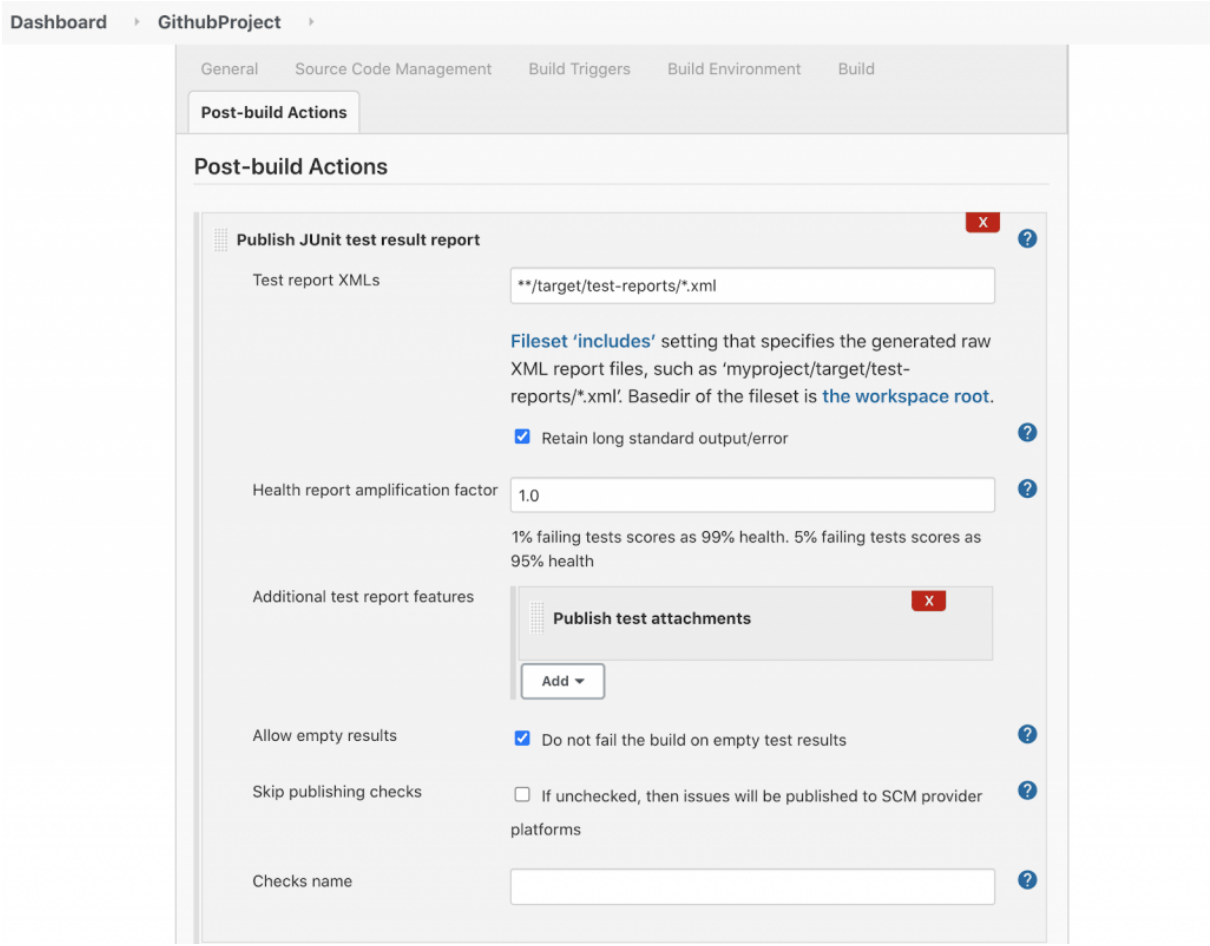
Step 3: Click on the "**Add post-build action**" button at the bottom of the page.



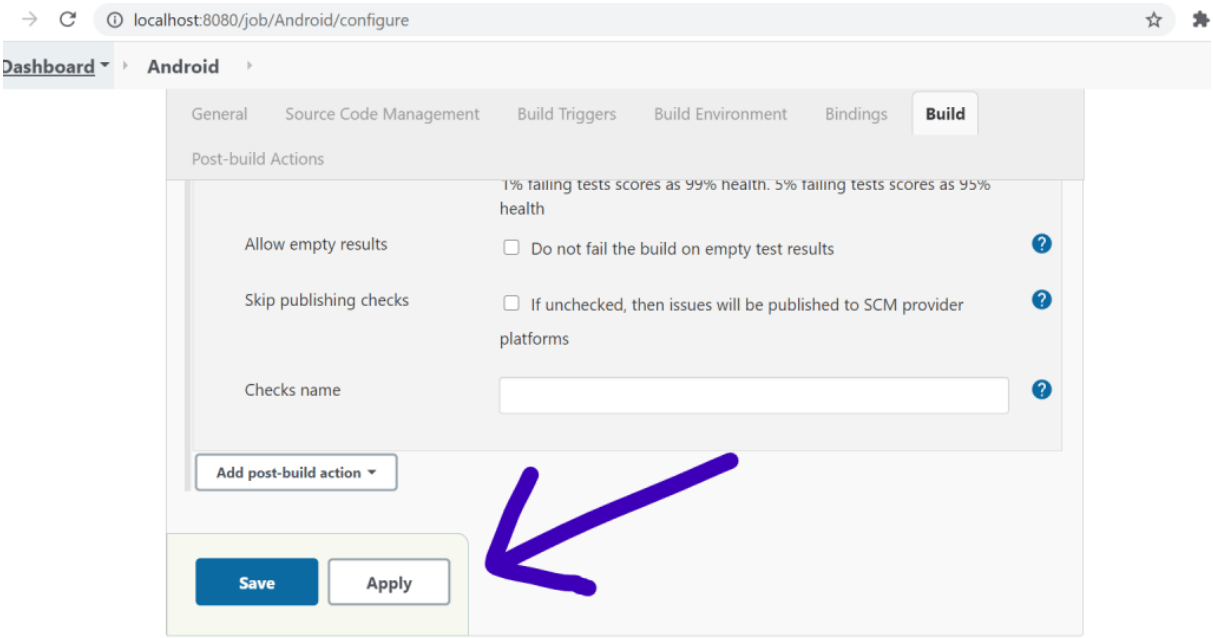
Step 4: From the above list of items displayed in the dropdown, select the "**Publish JUnit test result report**" option.



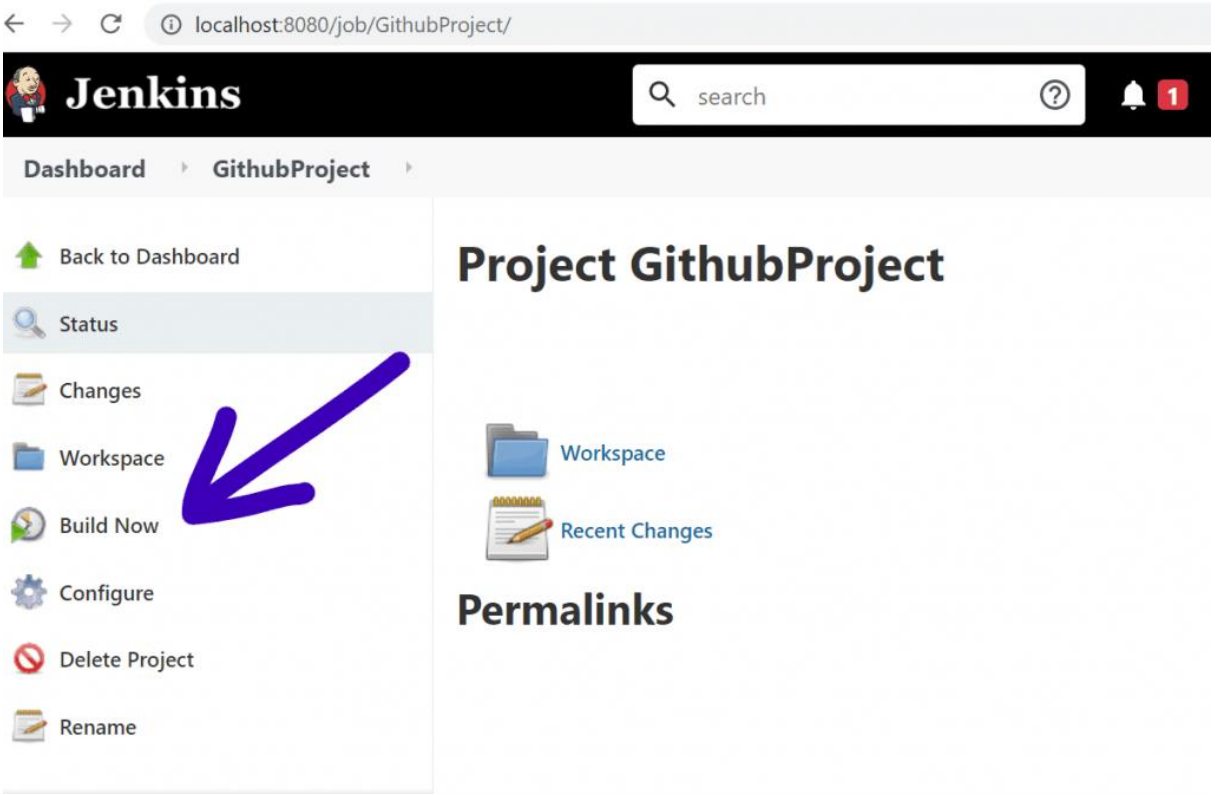
Step 5: Add the generated XML test report XML path from the workspace in the **Test report XMLs** field as shown below.



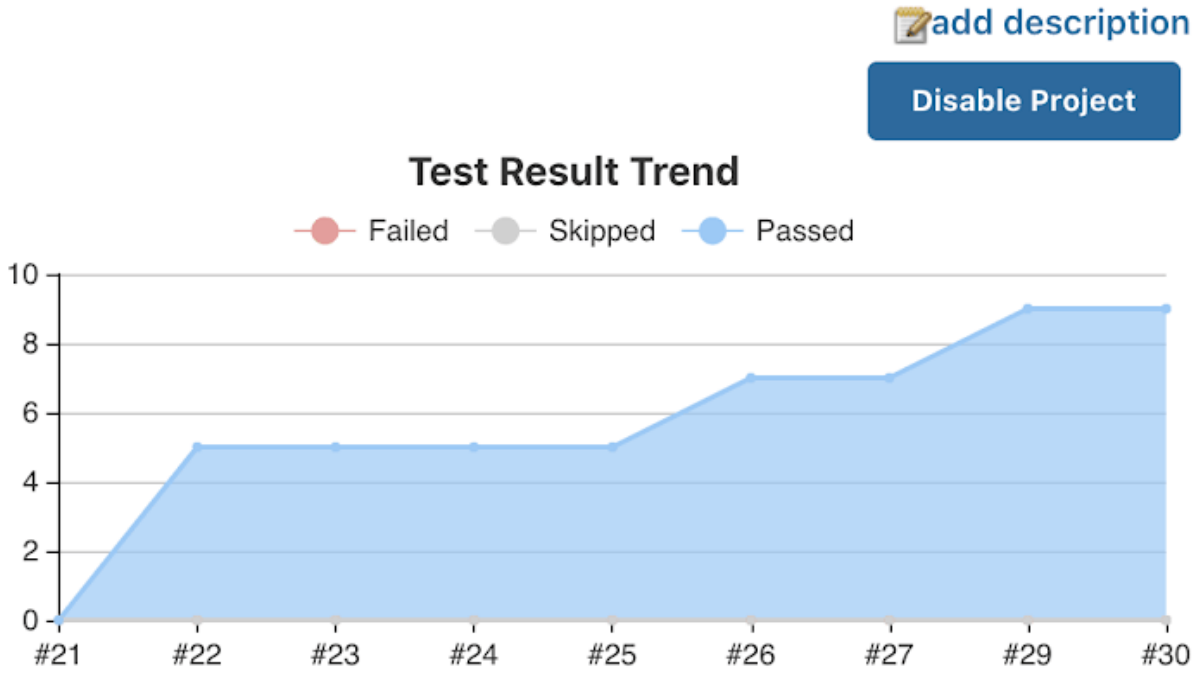
Step 6: Click on the '**Save**' and '**Apply**' buttons.



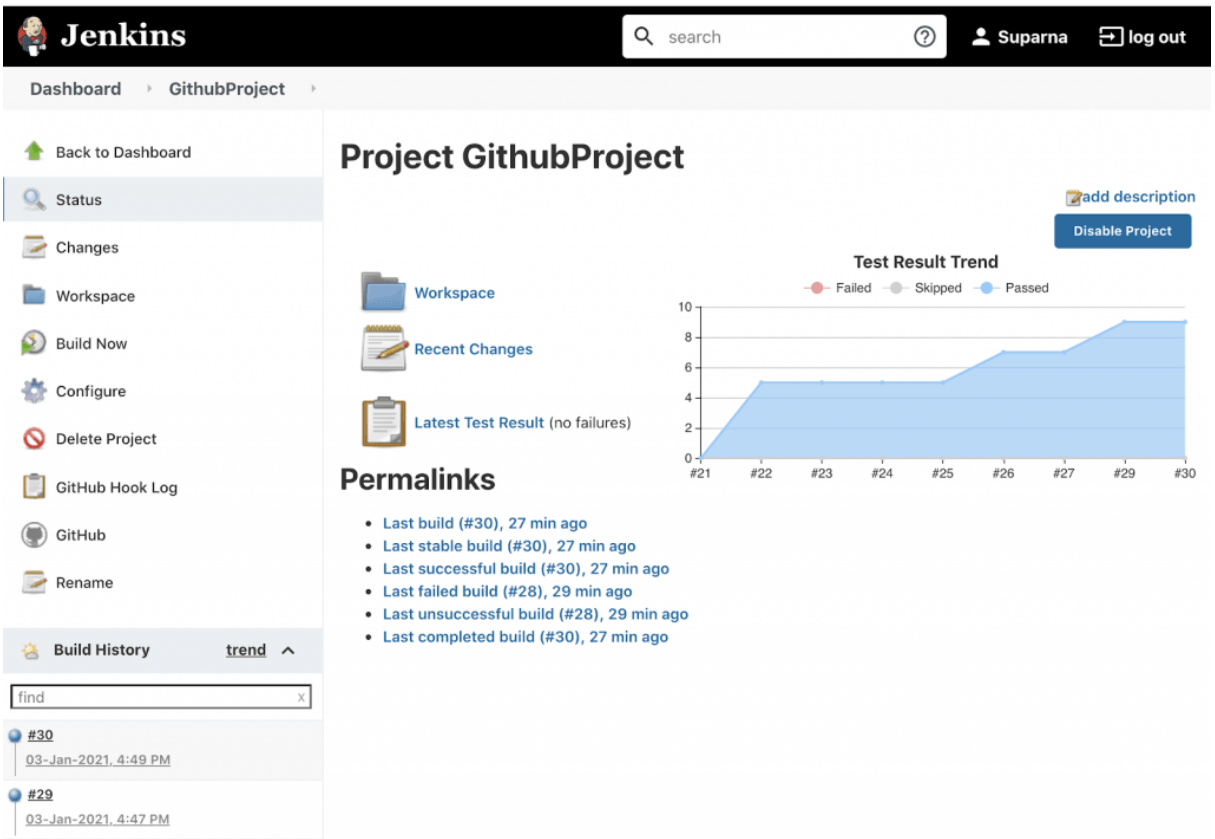
Step 7: Click on the “**Build Now**” button and allow the scheduled build to complete.



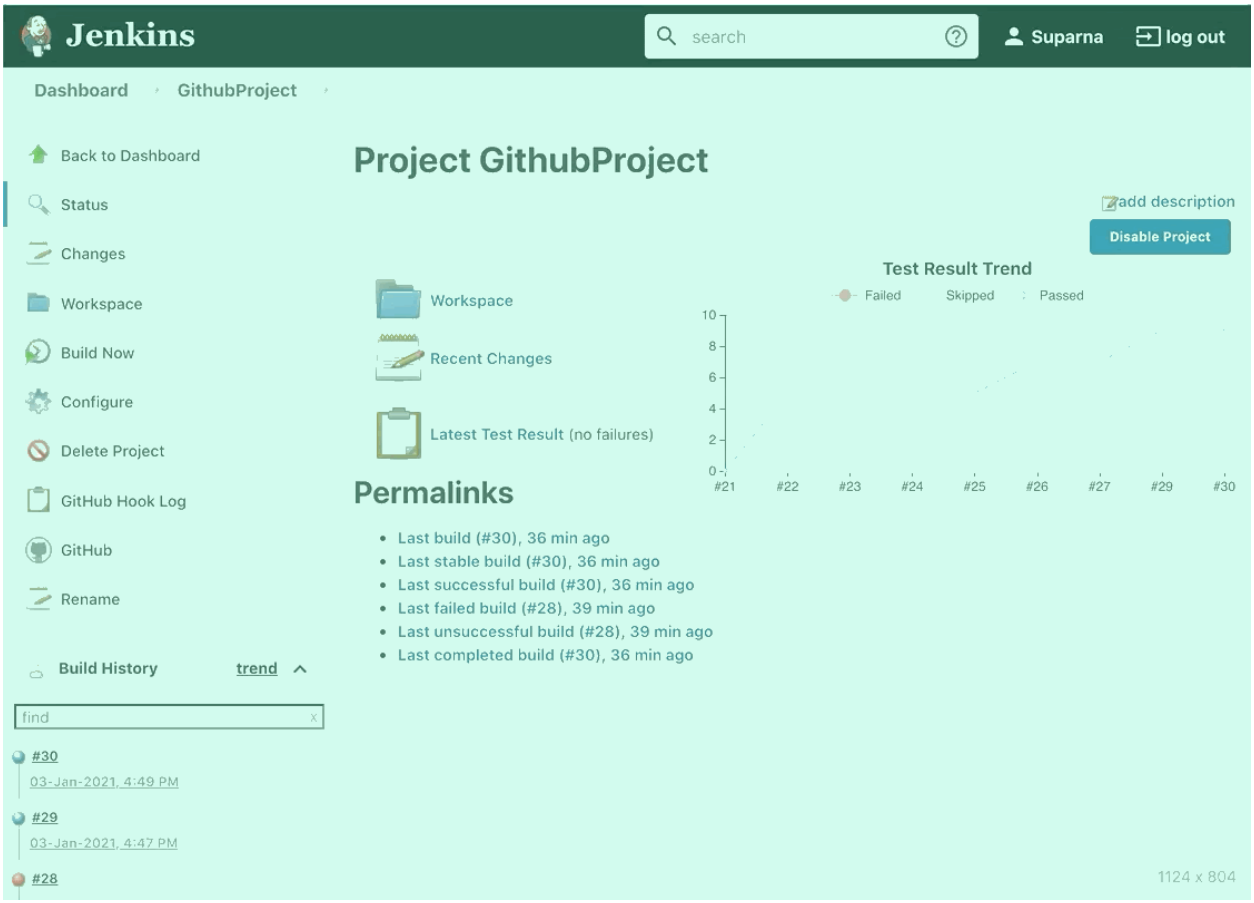
Step 8: Observe that a “**Test Result Trend**” graph is generated on the project dashboard’s right side.



Step 9: On closer observation, we can observe that **the x-axis** of the newly generated graph consists of Build Ids, and **the y-axis** is composed of several test cases run in each build.



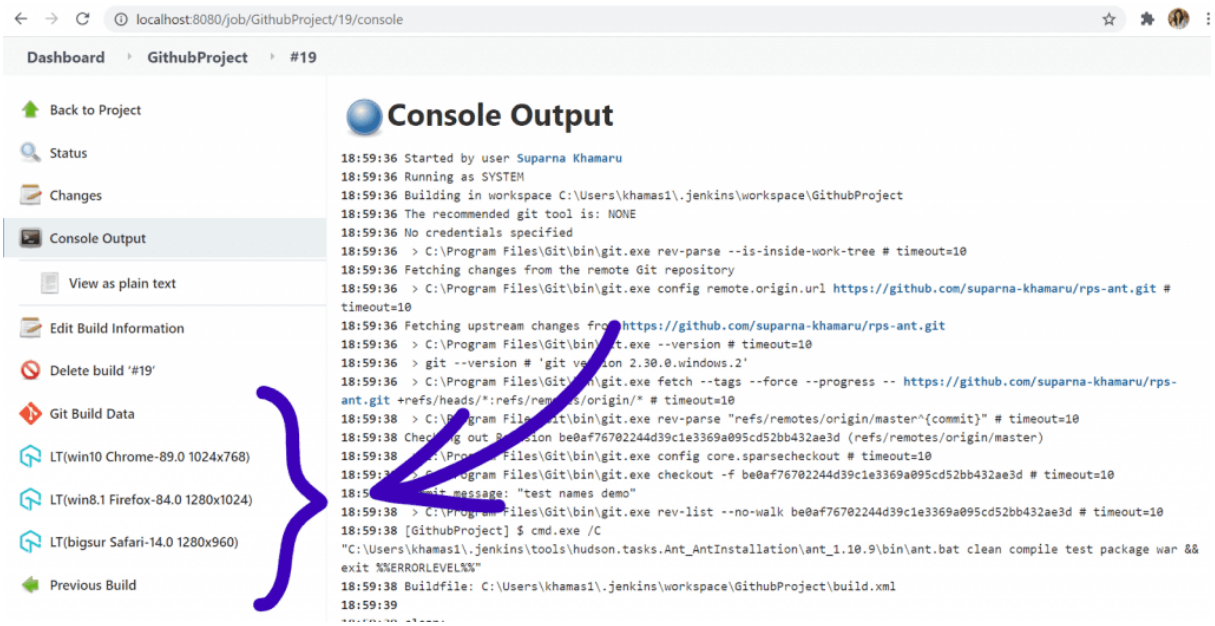
Step 10: Along with the generation of test result trend graphs, we can also observe that the "**Latest Test Result**" report is also generated in the dashboard to help users closely analyze the run tests status in each build.



Integrating LambdaTest Automation Plugin

Continuous integration with Jenkins and Git is a great way to streamline your test automation scripts. Furthermore, you can also implement a Jenkins Continuous integration to make your scripts easier to test. LambdaTest provides a Jenkins plugin to easily automate your Selenium test scripts by connecting your Jenkins CI instance to Selenium. If one follows the instructions as provided in the official website of LambdaTest, we can also observe that LambdaTest integration works smoothly to the above test run as well on any desired test environment.

I simply followed the steps provided on the official website to check how the integration works in Jenkins, and to my surprise, it worked like magic without any extra effort.



And the tests passed successfully in all the configured test environments!