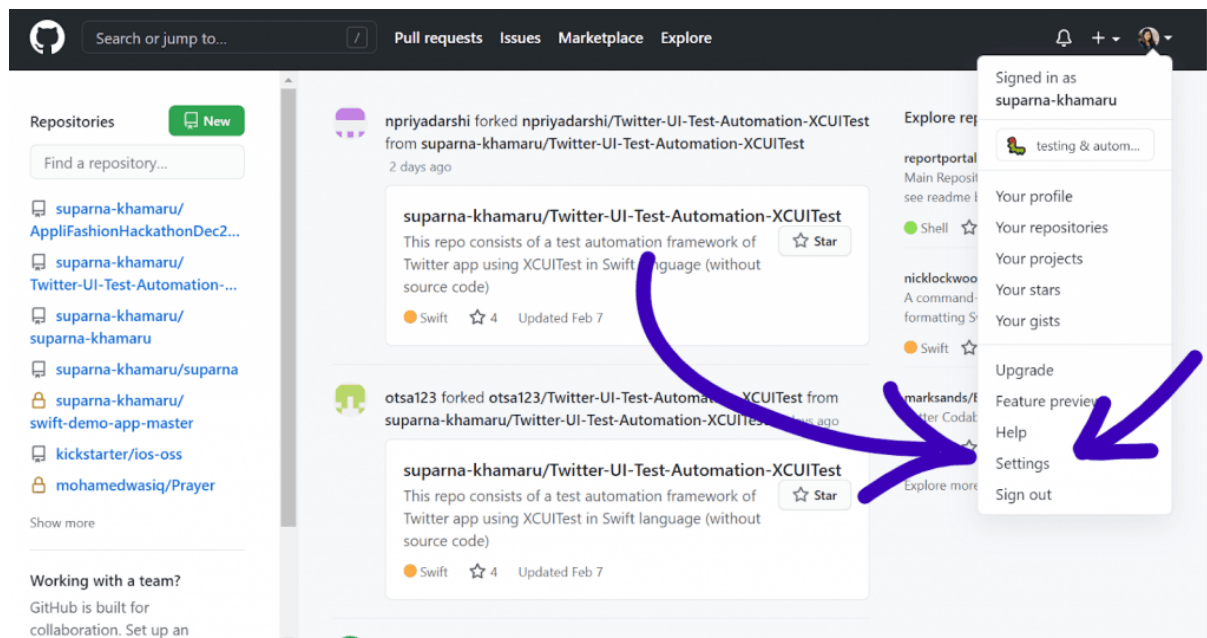


CLOUD DEVOPS(EPAM)

PROJECT

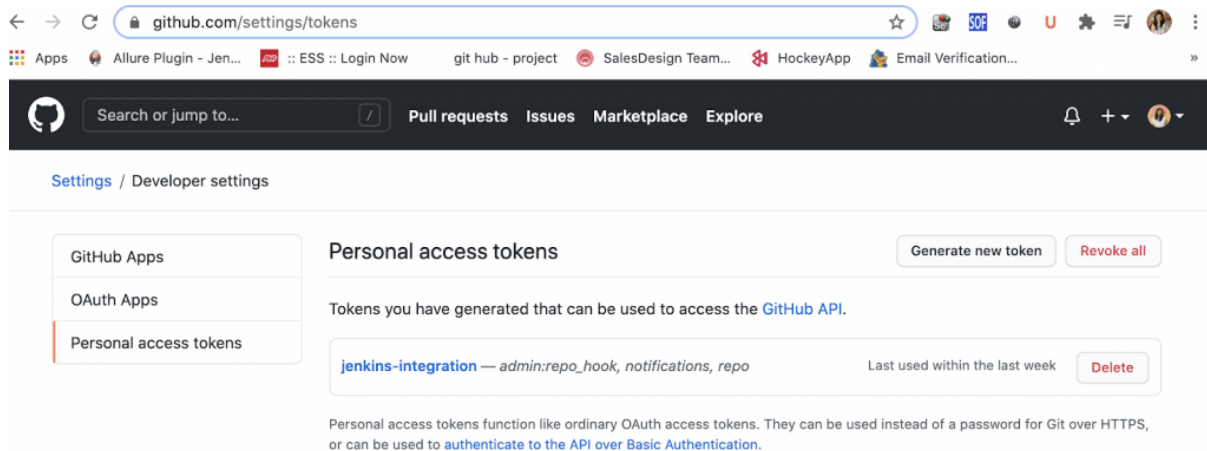
Configuring GitHub for Jenkins Continuous Integration

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.

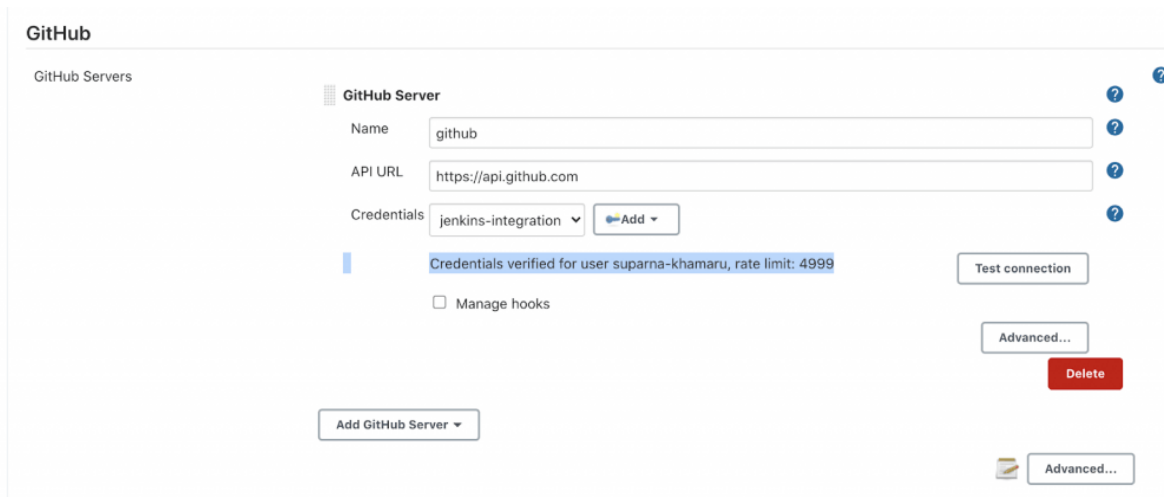


Step 4: Post successful creation of the token, the newly generated secret text in GitHub is to be copied for future usage in Jenkins.

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

localhost:8080/pluginManager/installed			☆	⚙	👤
Plugin Manager					
<input checked="" type="checkbox"/>	Git client plugin Utility plugin for Git support in Jenkins	3.6.0	Uninstall		
<input checked="" type="checkbox"/>	Git plugin This plugin integrates Git with Jenkins.	4.5.2	Uninstall		
<input checked="" type="checkbox"/>	GIT server Plugin Allows Jenkins to act as a Git server.	1.9	Uninstall		
<input checked="" type="checkbox"/>	GitHub This plugin integrates GitHub to Jenkins.	1.32.0	Uninstall		
<input checked="" type="checkbox"/>	GitHub API Plugin This plugin provides GitHub API for other plugins.	1.122	Uninstall		
<input checked="" type="checkbox"/>	GitHub Branch Source Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc.	2.9.4	Uninstall		

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.

 **Jenkins Credentials Provider: Jenkins**

 **Add Credentials**

Domain

Global credentials (unrestricted)

Kind

✓ Username with password

GitHub App

SSH Username with private key

Secret file

Secret text

Certificate

Password


ID


Description

Add

Cancel

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.

 **Jenkins Credentials Provider: Jenkins**

 **Add Credentials**

Domain

Global credentials (unrestricted)

Kind

Secret text

Scope

Global (Jenkins, nodes, items, all child items, etc)

Secret

.....

ID

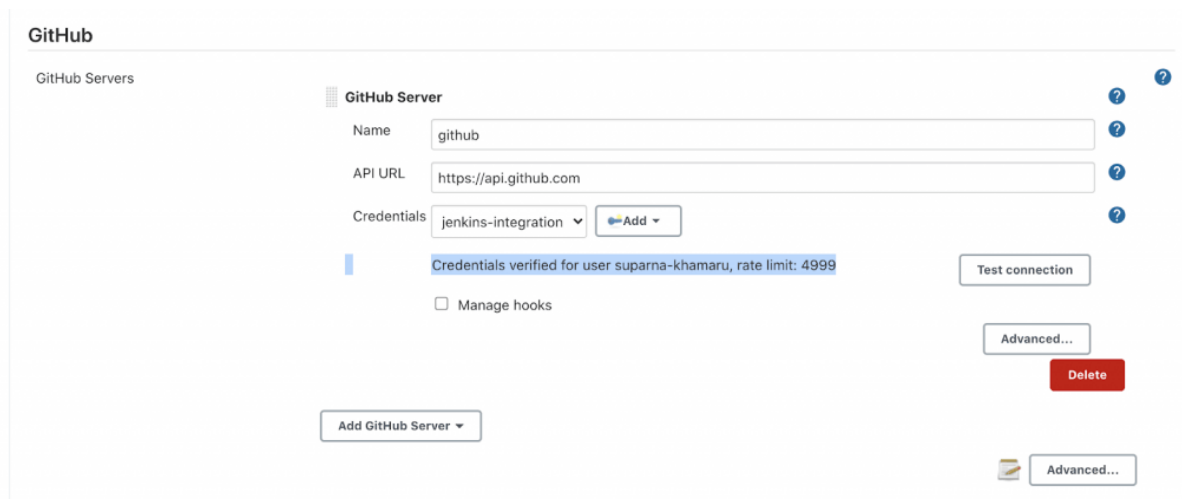
jenkins-integration1

Description

Add

Cancel

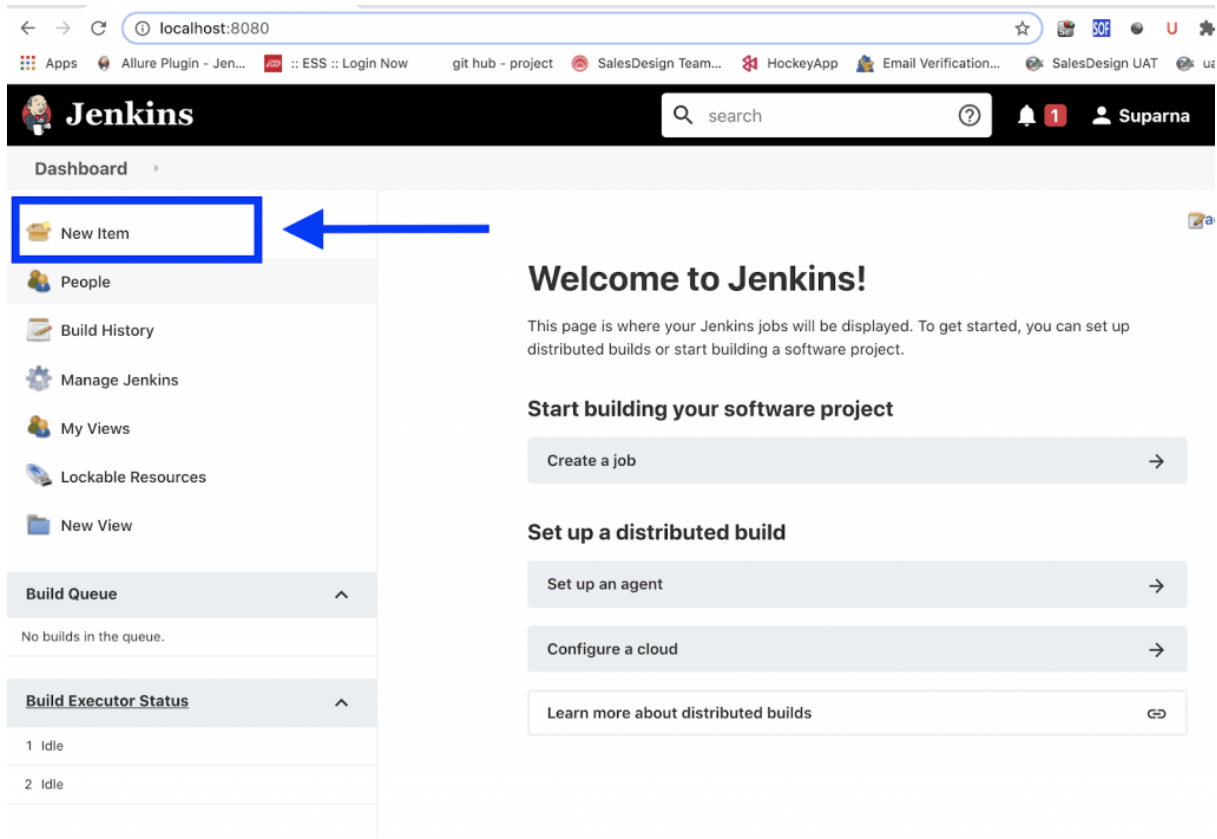
Step 8: Once the Secret text is successfully added, let us test the connection by clicking on the “**Test connection**” button and verify the confirmation message as highlighted below.



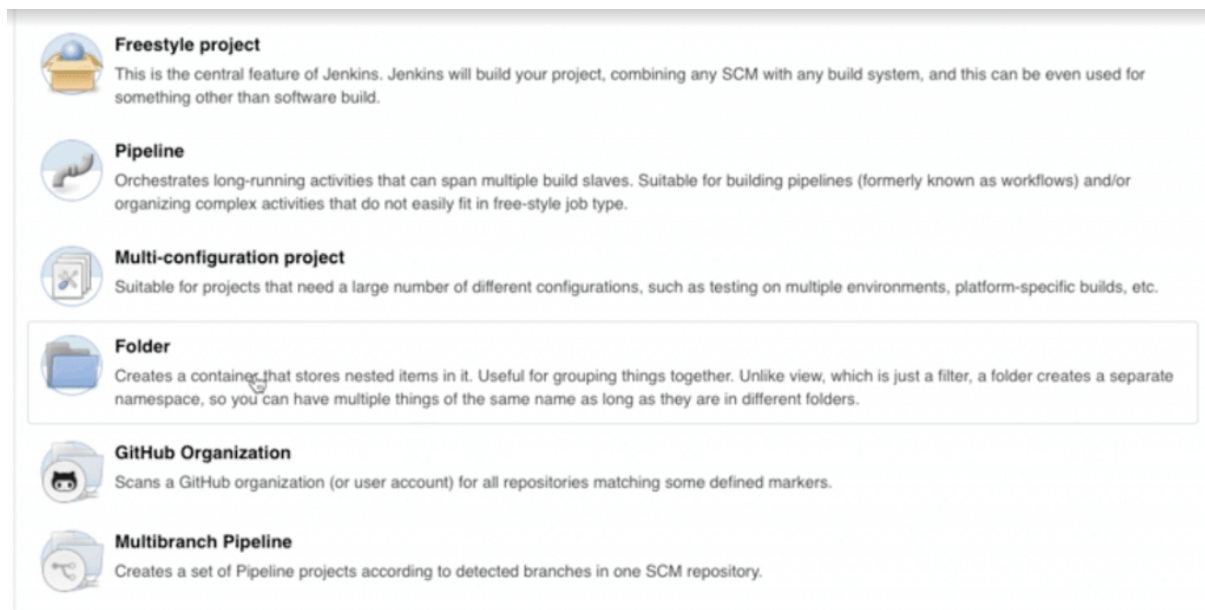
Creating Your First Jenkins Job Integrated Into a Test Project

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.

The screenshot shows the Jenkins web interface in a browser. The address bar indicates the URL is `localhost:8080/job/GithubProject/configure`. The Jenkins header is visible with the logo, a search bar, and the user name 'Suparna' with a 'log out' link. Below the header, the breadcrumb navigation shows 'Dashboard' > 'GithubProject'. The main configuration area has several tabs: 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'General' tab is selected. It contains a 'Description' text area, a checkbox for 'Discard old builds', a checked checkbox for 'GitHub project', a 'Project url' text field containing `https://github.com/suparna-khamaru/rps-ant/`, and a checkbox for 'This build requires lockable resources'. There is also a 'Preview' link and an 'Advanced...' button.

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.

Dashboard > GithubProject >

General **Source Code Management** Build Triggers Build Environment Build

Post-build Actions

Source Code Management

☐ None
☒ Git

Repositories

Repository URL ?

Credentials Add ?

Advanced...

Add Repository

Branches to build

Branch Specifier (blank for 'any') X ?

Add Branch

Repository browser ?

Additional Behaviours

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
```


Dashboard > GithubProject >

General Source Code Management Build Triggers **Build Environment** Build

Post-build Actions

Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s) ?
- ☐ Abort the build if it's stuck
- ☒ Add timestamps to the Console Output
- ☐ Inspect build log for published Gradle build scans
- ☒ With Ant ?

Ant Version ant 1.10.9 ▼

Build

Invoke Ant

Ant Version ant 1.10.9 ▼

Targets clean compile test package war ▼ ?

Advanced...

Add build step ▼

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

Build

Invoke Ant

Ant Versionant 1.10.9

Targetsclean compile test package war

Advanced...

Add build step


Post-build Actions

Add post-build action

Save


Apply


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.


 **Jenkins**


search


Dashboard > GithubProject >


 Back to Dashboard


 Status


 Changes


 Workspace

 Build Now

 Configure

 Delete Project

 GitHub

 Rename

Build History


trend ^


find

#16

02-Jan-2021, 11:35 PM

Project GithubProject

 Workspace

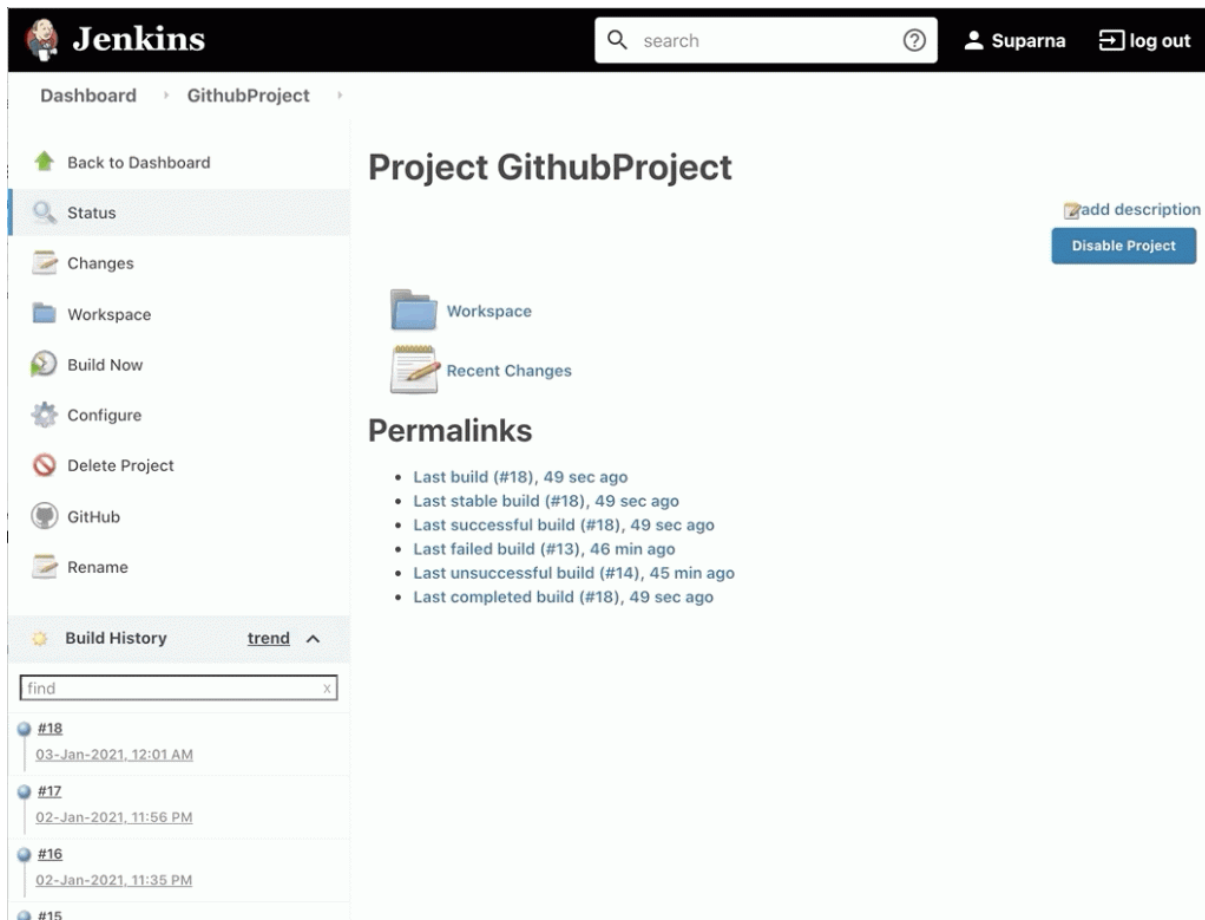
 Recent Changes

Permalinks

- Last build (#16), 18 min ago
- Last stable build (#16), 18 min ago
- Last successful build (#16), 18 min ago
- Last failed build (#13), 37 min ago
- Last unsuccessful build (#14), 36 min ago
- Last completed build (#16), 18 min ago

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.



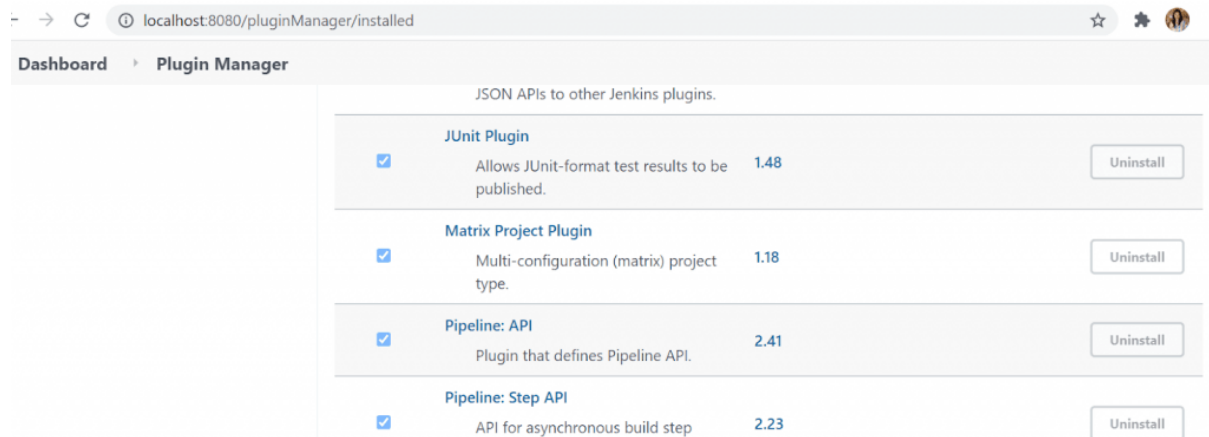
Analyzing JUnit Test Results in Jenkins

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.

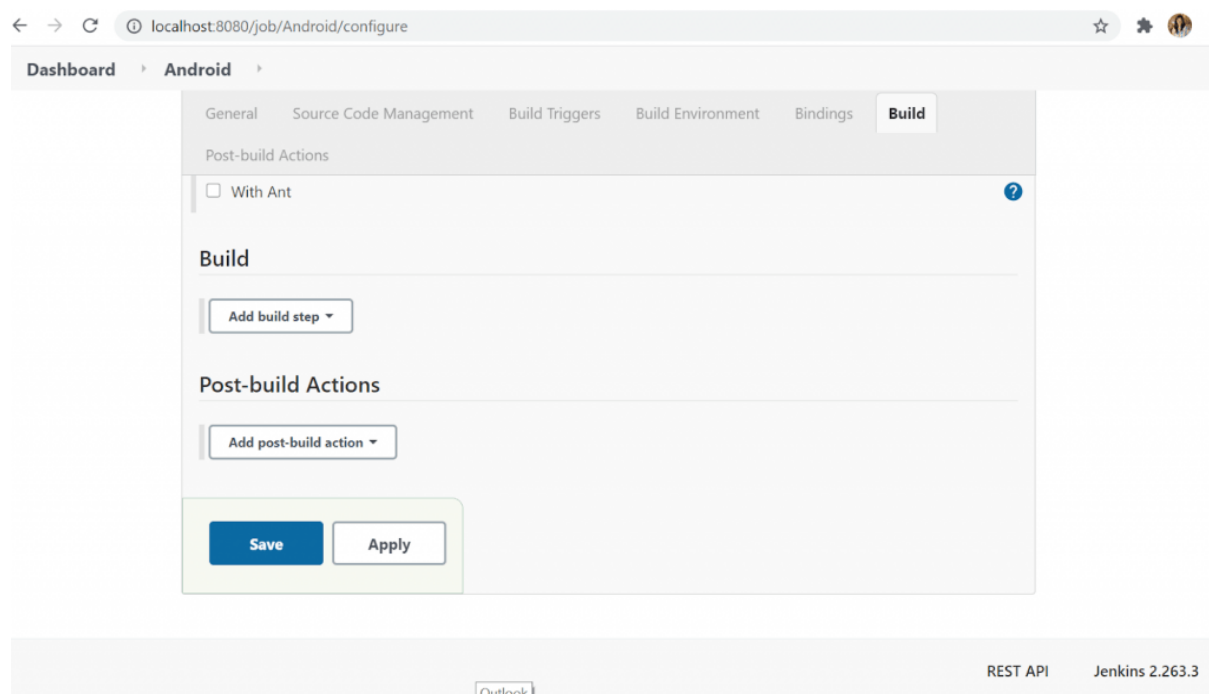
Let's see how this can be done for [JUnit](#).

Step 1: Make sure “**JUnit Plugin**” is already installed in Jenkins.

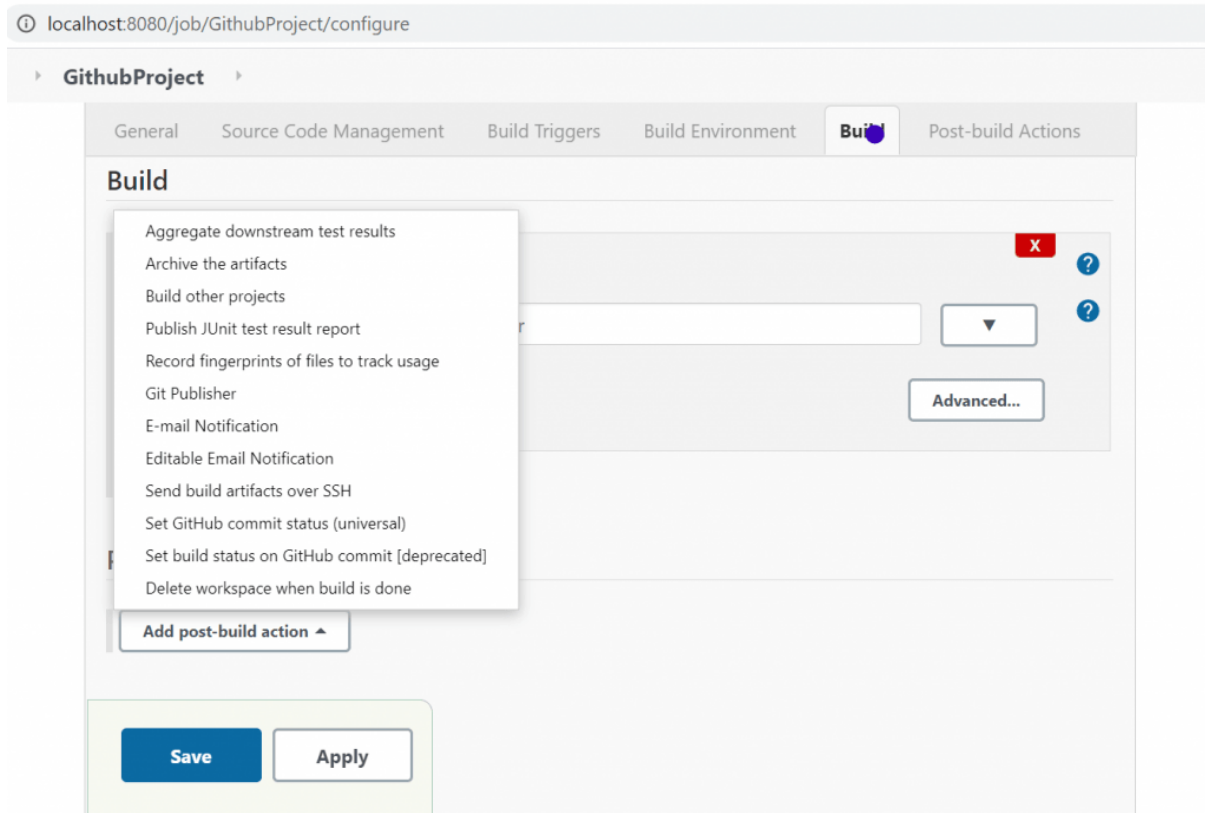
Go to Jenkins Dashboard -> Manage Jenkins -> Manage Plugins -> Click on the 'Installed' tab to view all the already installed plugins.



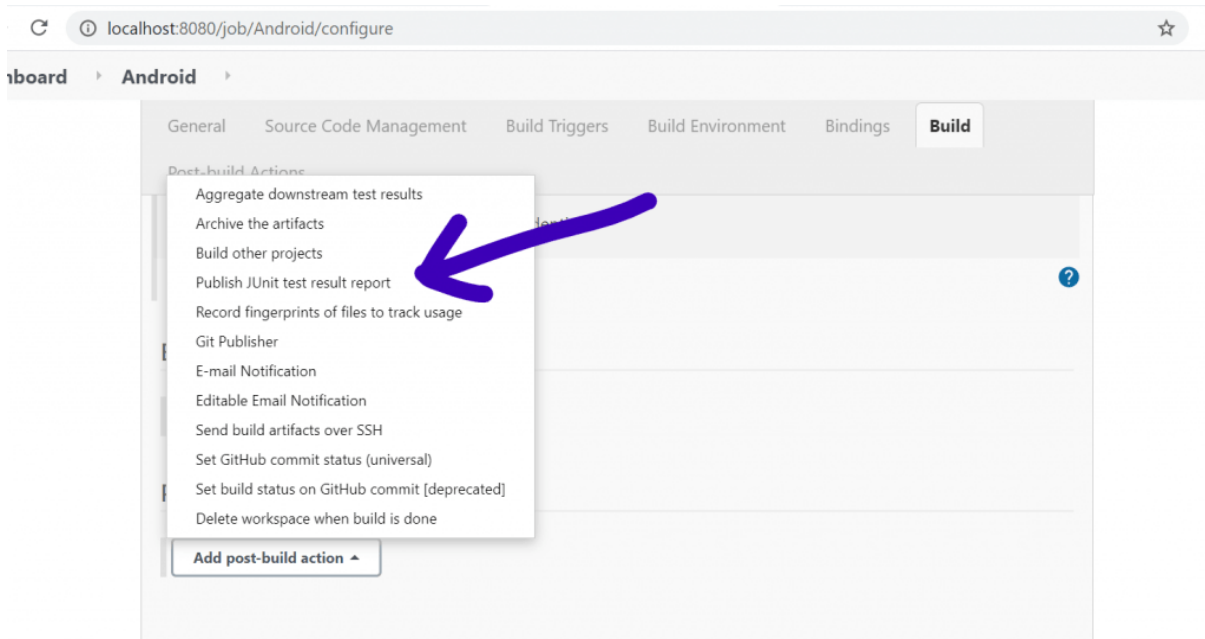
Step 2: Go to Project -> Click on '**Configure**' -> Click on '**Build**' tab.



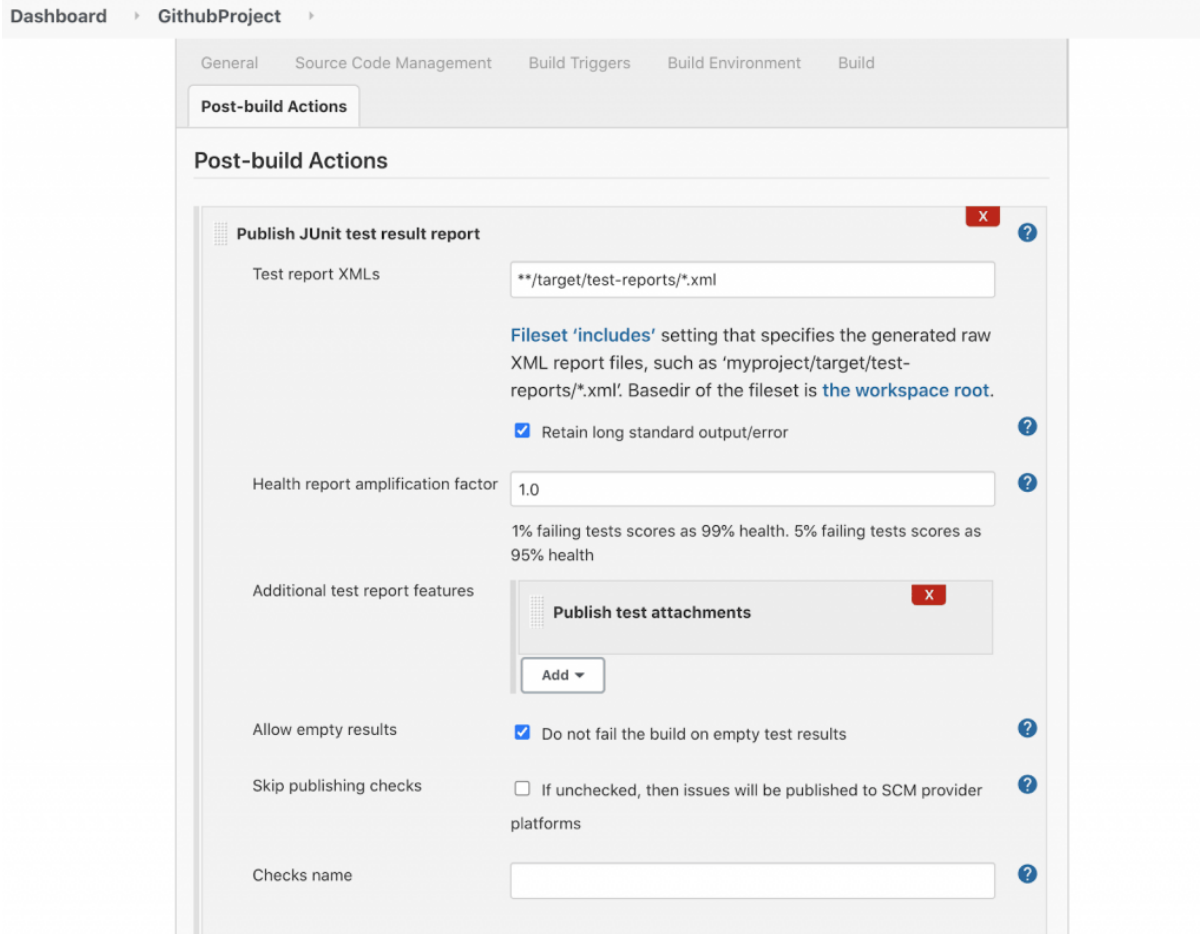
Step 3: Click on the “**Add post-build action**” button at the bottom of the page.



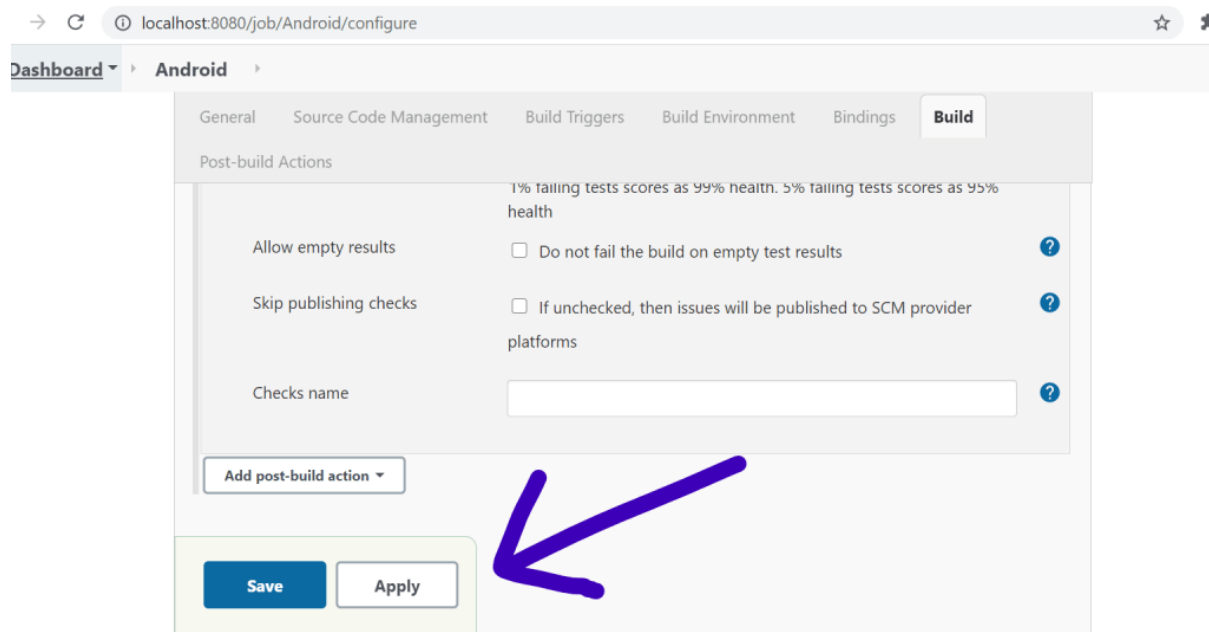
Step 4: From the above list of items displayed in the dropdown, select the “**Publish JUnit test result report**” option.



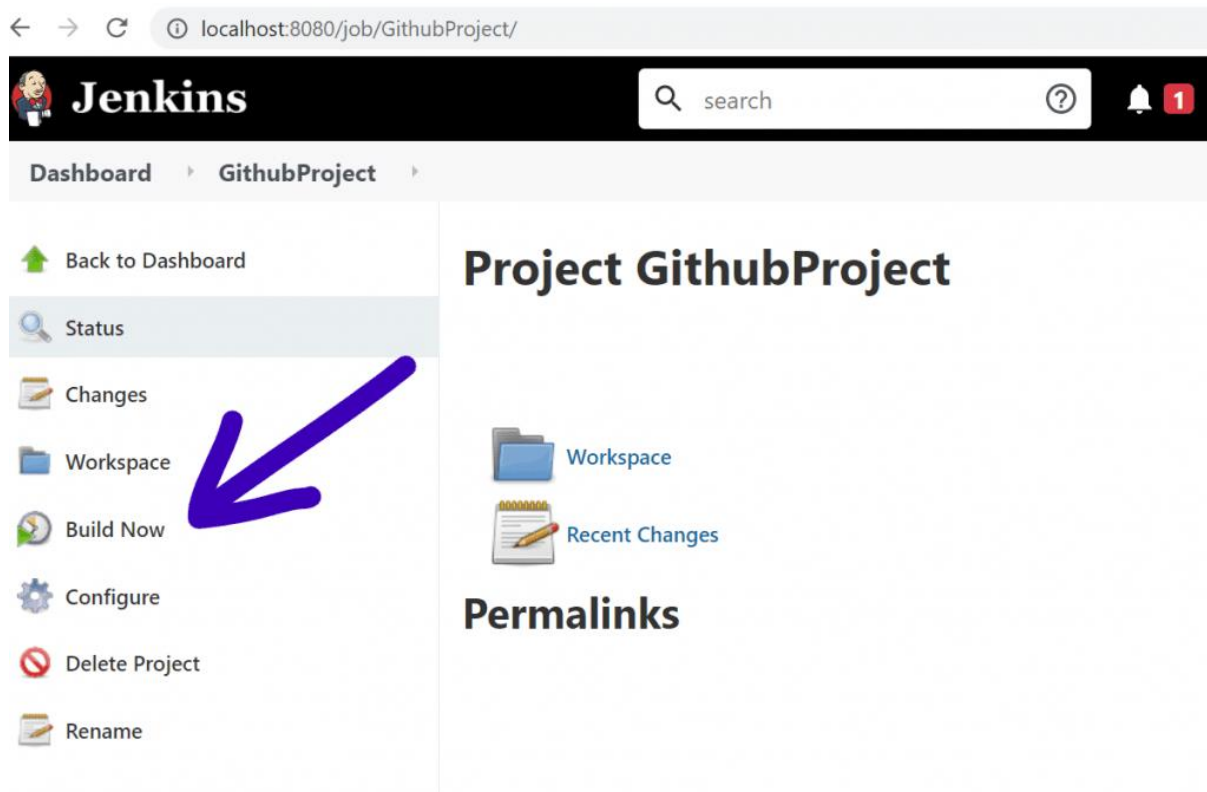
Step 5: Add the generated XML test report XML path from the workspace in the **Test report XMLs** field as shown below.



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



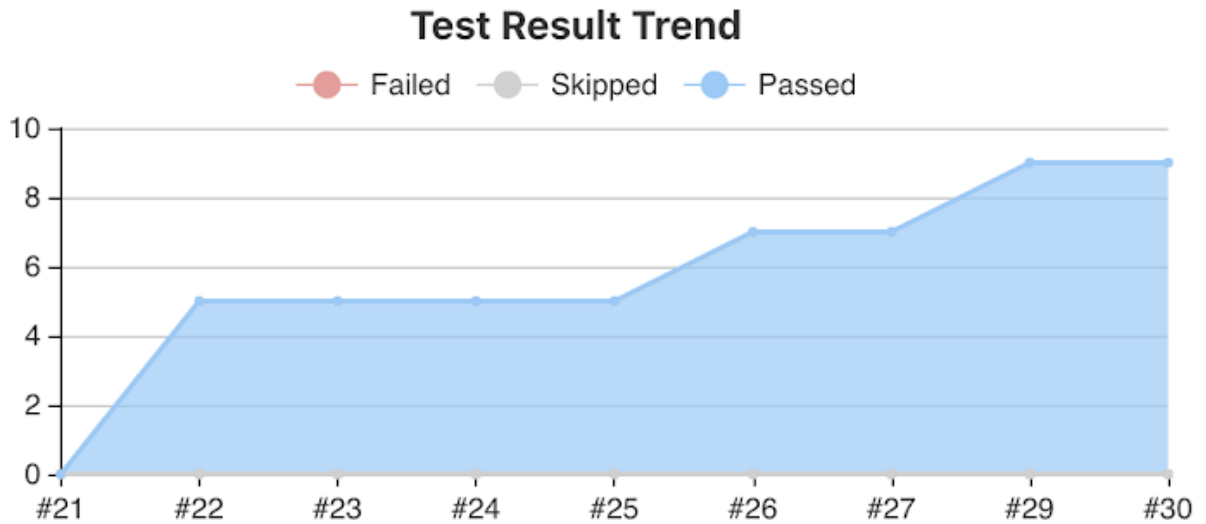
Step 7: Click on the “**Build Now**” button and allow the scheduled build to complete.



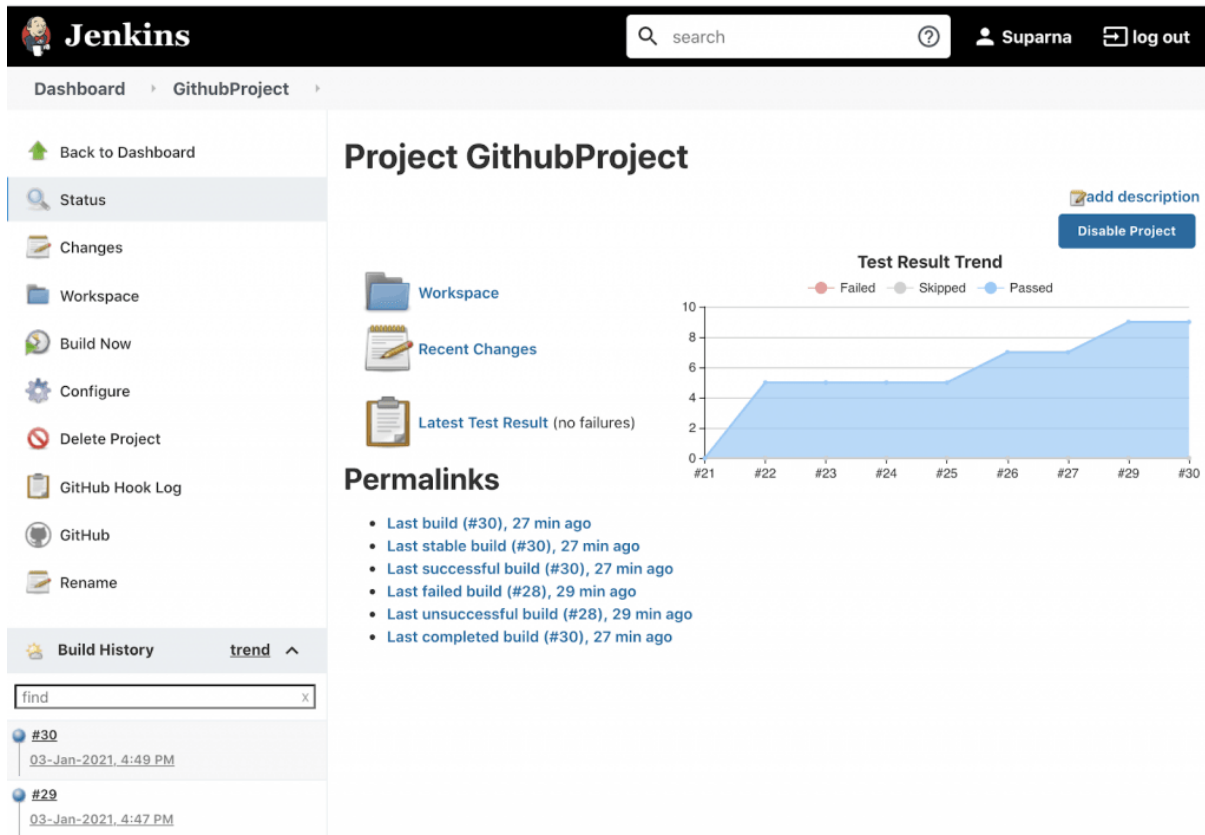
Step 8: Observe that a “**Test Result Trend**” graph is generated on the project dashboard’s right side.

 add description

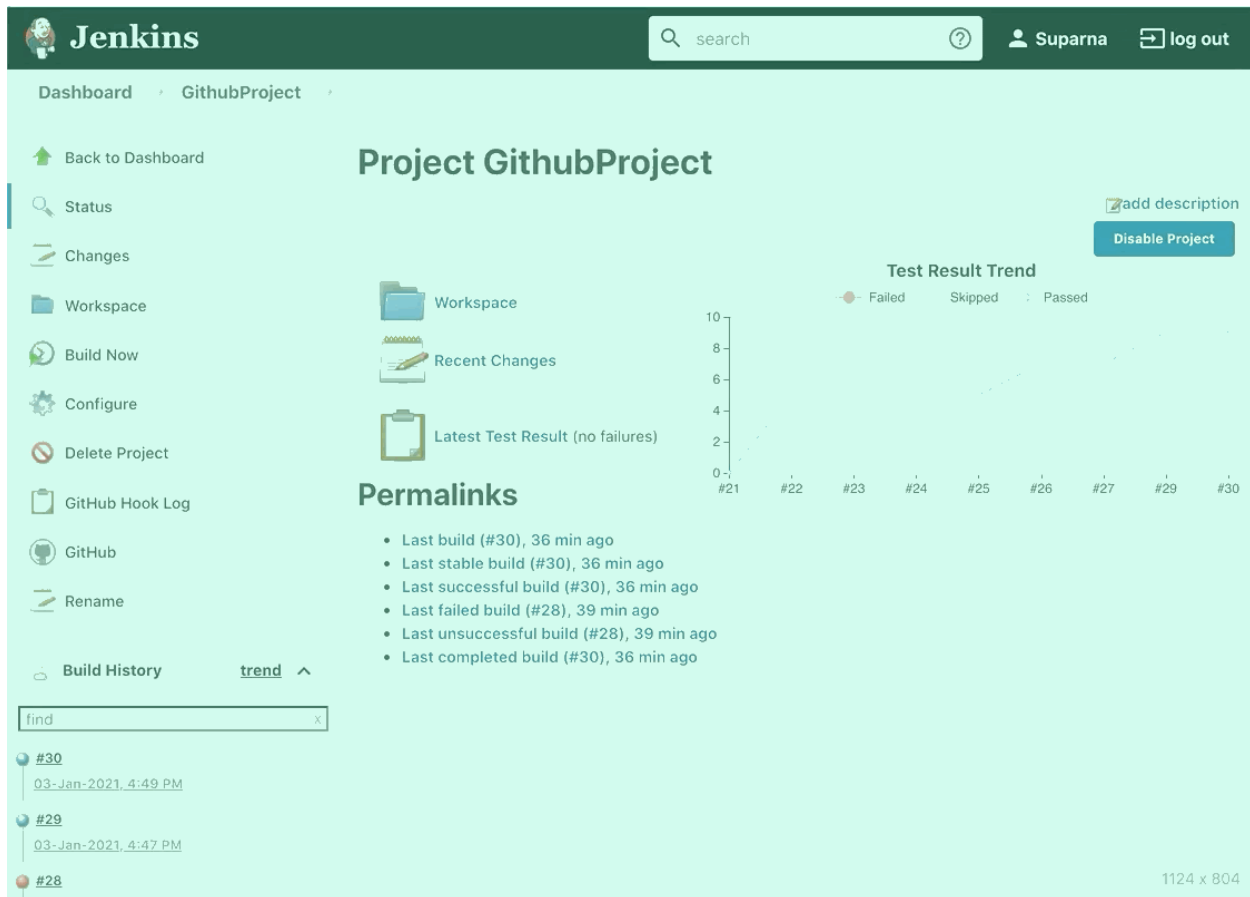
Disable Project



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.

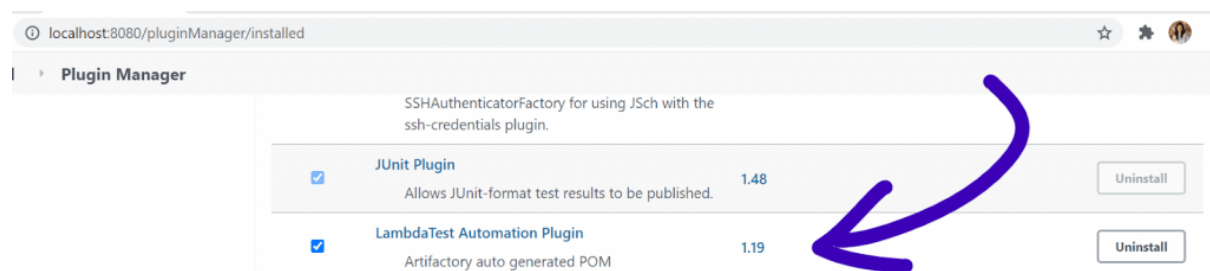


Step 10: Along with the generation of test result trend graphs, we can also observe that the “**Latest Test Result**” report is also generated in the dashboard to help users closely analyze the run tests status in each build.



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.

The screenshot shows the Jenkins web interface for build #19 of the 'GithubProject'. The left sidebar contains navigation links: 'Back to Project', 'Status', 'Changes', 'Console Output' (selected), 'View as plain text', 'Edit Build Information', 'Delete build '#19'', 'Git Build Data', 'LT(win10 Chrome-89.0 1024x768)', 'LT(win8.1 Firefox-84.0 1280x1024)', 'LT(bigsur Safari-14.0 1280x960)', and 'Previous Build'. The main area is titled 'Console Output' and displays a log of build steps. A large blue arrow points from the 'Console Output' link in the sidebar to the log content. The log shows the build starting at 18:59:36, running as SYSTEM, and building in the workspace C:\Users\khamas1\.jenkins\workspace\GithubProject. It details the git checkout process, including fetching upstream changes, checking out the master branch, and running a series of tests (clean, compile, test, package, war) which all passed successfully. The build file is located at C:\Users\khamas1\.jenkins\workspace\GithubProject\build.xml.

```
18:59:36 Started by user Suparna Khamaru
18:59:36 Running as SYSTEM
18:59:36 Building in workspace C:\Users\khamas1\.jenkins\workspace\GithubProject
18:59:36 The recommended git tool is: NONE
18:59:36 No credentials specified
18:59:36 > C:\Program Files\Git\bin\git.exe rev-parse --is-inside-work-tree # timeout=10
18:59:36 Fetching changes from the remote Git repository
18:59:36 > C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/suparna-khamaru/rps-ant.git #
timeout=10
18:59:36 Fetching upstream changes from https://github.com/suparna-khamaru/rps-ant.git
18:59:36 > C:\Program Files\Git\bin\git.exe --version # timeout=10
18:59:36 > git --version # 'git version 2.30.0.windows.2'
18:59:36 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/suparna-khamaru/rps-
ant.git +refs/heads/*:refs/remotes/origin/* # timeout=10
18:59:38 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/master^{commit}" # timeout=10
18:59:38 Checking out Revision be0af76702244d39c1e3369a095cd52bb432ae3d (refs/remotes/origin/master)
18:59:38 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
18:59:38 > C:\Program Files\Git\bin\git.exe checkout -f be0af76702244d39c1e3369a095cd52bb432ae3d # timeout=10
18:59:38 Commit message: "test names demo"
18:59:38 > C:\Program Files\Git\bin\git.exe rev-list --no-walk be0af76702244d39c1e3369a095cd52bb432ae3d # timeout=10
18:59:38 [GithubProject] $ cmd.exe /C
"C:\Users\khamas1\.jenkins\tools\hudson.tasks.Ant_AntInstallation\ant_1.10.9\bin\ant.bat clean compile test package war &&
exit %ERRORLEVEL%"
18:59:38 Buildfile: C:\Users\khamas1\.jenkins\workspace\GithubProject\build.xml
18:59:39
```

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