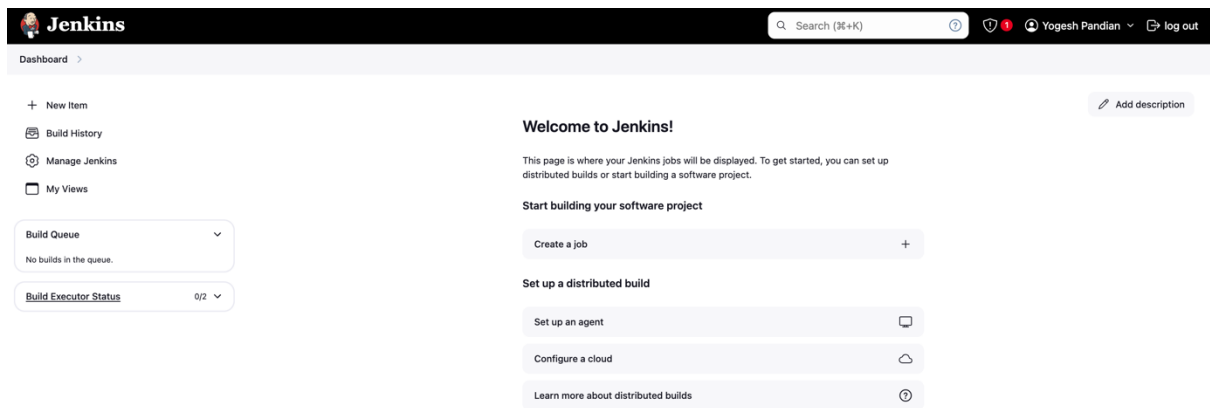


# Jenkins Configuration Guide for Testers

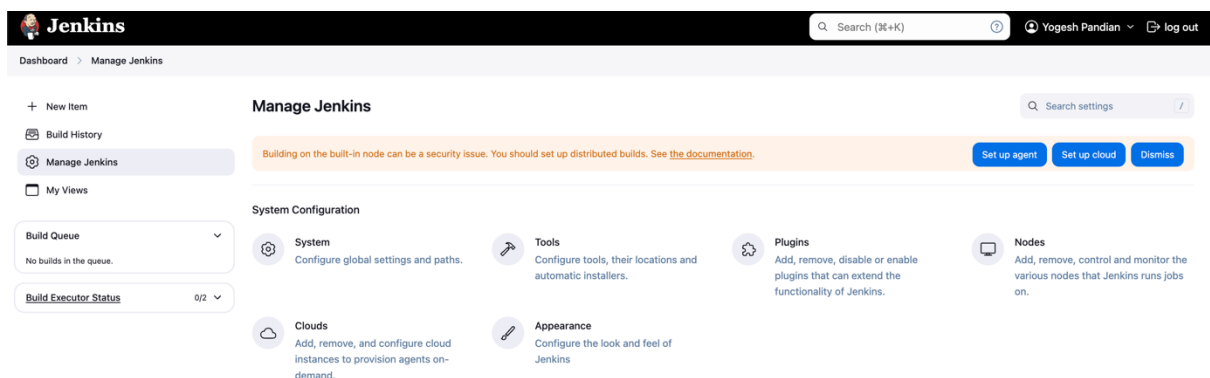
## One-Time Configuration

After installing Jenkins, open the Jenkins dashboard and ensure Jenkins is running. To begin, access **Manage Jenkins** from the main dashboard.



## Manage Jenkins

Within **Manage Jenkins**, you'll find various configuration options. Select **System Configuration** to access important settings. From here, go to **Plugins** to configure the necessary tools and integrations.



## Installing Necessary Plugins

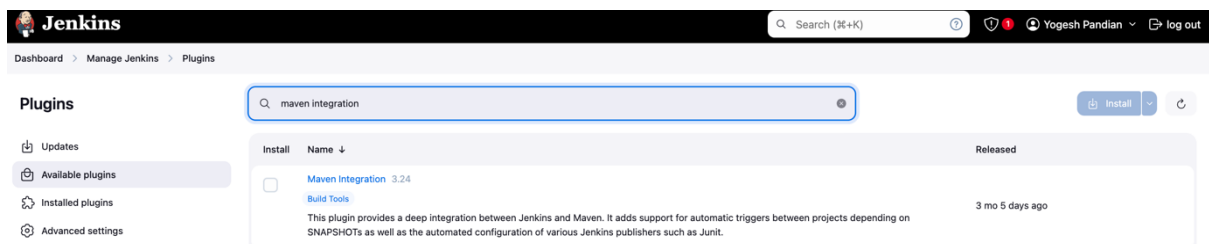
In Jenkins, plugins are essential for integrating third-party tools such as **Maven**, **Git**, and **JDK**. These tools are crucial for managing your build process and connecting Jenkins to external systems like **GitHub**.

### Key Plugins for Test Automation:

- **Maven**: Manages the build lifecycle of your project.
- **Git**: Handles version control.
- **GitHub**: Integrates with GitHub repositories for source code management.
- **JDK**: Configures Java for running your build and automation tasks.

To install plugins:

1. Go to **Manage Jenkins > Plugins**.
2. Under the **Available** tab, search for the **Maven Integration Plugin**.
3. Select the plugin and click **Install**.



Same way install all mentioned key plugins. Once the plugins are installed, ensure you've set the correct paths for **Maven**, **Git**, and **JDK**. These paths are necessary for running your automation scripts in Jenkins.

---

## Configuring Tool Paths

Now that the necessary plugins are installed, you need to set the paths for **Maven**, **Git**, and **JDK**. This step is essential for Jenkins to recognize these tools.

To do this, navigate to **Manage Jenkins > System Configuration** and click on **Tools**. Here, you can configure the locations for each tool.

## Example Paths:

- **JDK Path:**  
/opt/homebrew/opt/openjdk@17/libexec/openjdk.jdk/Contents/Home
- **Maven Path:**  
/opt/homebrew/Cellar/maven/3.9.9/libexec
- **Git Path:**  
/usr/bin/git

These paths are essential for Jenkins to locate the tools required to run your automation.

### JDK Path

Dashboard > Manage Jenkins > Tools

JDK installations

Add JDK

JDK

Name

JDK

JAVA\_HOME

/usr/libexec/java\_home

☐ Install automatically ?

### Git Path

Dashboard > Manage Jenkins > Tools

Git installations

Add Git

Git

Name

Git

Path to Git executable ?

/usr/bin/git

☐ Install automatically ?

### Maven Path

Maven installations

Add Maven

Maven

Name

Mvn

MAVEN\_HOME

/opt/homebrew/Cellar/maven/3.9.9/libexec

☐ Install automatically ?

Add Maven

Save Apply

## Summary of One-Time Setup

So far, completed the following setup steps:

- Installed Jenkins and configured the initial setup.
- Installed key **plugins**, including Maven, Git, GitHub, and JDK.
- Set the **paths** for Jdk, Maven, and Git in **System Configuration > Tools**.

## Running a Maven Project from GitHub

Now that Jenkins is set up, let's configure it to run a Maven project from GitHub.

### Creating a New Jenkins Job

1. On the **Jenkins Dashboard**, click **New Item**.
2. Select **Maven Project** and provide a name for your project.

## New Item

Enter an item name

RestAssuredFramework

Select an item type



### Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



### Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

## Configuring Source Code Management

1. In the project configuration page, scroll down to **Source Code Management**.
2. Select the **Git** option.
3. Enter the **GitHub repository URL** for your project:  
<https://github.com/AutomatedMind/RestAssuredFrameworkPractice.git>

Dashboard > RestAssuredFramework > Configuration

### Configure

- General
- Source Code Management**
- Build Triggers
- Build Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

#### Source Code Management

☐ None

☒ Git ?

Repositories ?

Repository URL ?

Credentials ?

+ Add

Advanced ▾

## Setting Up the Build Command

1. Scroll to the **Build** section.
2. Under **Goals and Options**, enter the following command:  
`test`  
This tells Maven to run the `mvn test` command, which will execute the tests defined in the `pom.xml` file.
3. Save the configuration.

## Build

Root POM ?

pom.xml

Goals and options ?

test

Advanced ▾

## Project Configuration Summary

Here's a quick recap of the steps to configure the project:

- Create a new **Maven Project** and provide a name.
- In **Source Code Management**, select **Git** and provide the **GitHub project URL**.
- Under **Build**, add the Maven command test in the **Goals and Options** field.
- Click **Save** to complete the configuration.

## Running the Project

After configuring your project, it will appear on the **Jenkins Dashboard**. To trigger a build:

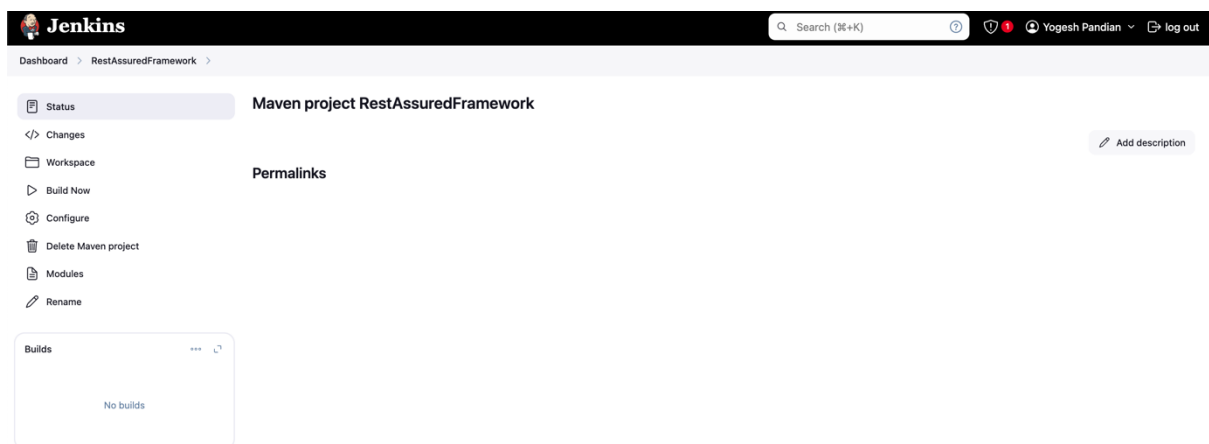
1. Click on the project name to open the project's specific dashboard.
2. From there, click **Build Now** to run the project.

## Project Dashboard

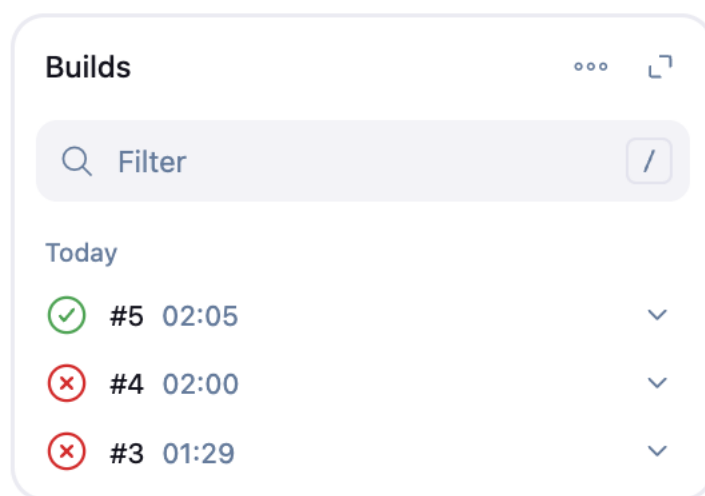
The screenshot shows the Jenkins Dashboard interface. At the top, there's a header with the Jenkins logo, a search bar, and user information (Yogesh Pandian). Below the header, the dashboard is divided into sections. On the left, there's a sidebar with links like 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. The main area displays a table of projects. The table has columns for 'S' (Status), 'W' (Webhook), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. A single project is listed: 'RestAssuredFramework' with a status of 'N/A' for both Last Success and Last Failure, and 'N/A' for Last Duration. Below the table, there are filters for 'Icon' (S, M, L) and a 'Build Queue' section showing 'No builds in the queue.' and a 'Build Executor Status' section showing '0/2'.

S	W	Name ↓	Last Success	Last Failure	Last Duration
🔄	☀️	RestAssuredFramework	N/A	N/A	N/A

## Project Specific Dashboard



The screenshot shows the Jenkins Project Specific Dashboard for the 'Maven project RestAssuredFramework'. The top navigation bar includes the Jenkins logo, a search bar, and user information (Yogesh Pandian). The left sidebar contains a list of actions: Status (selected), Changes, Workspace, Build Now, Configure, Delete Maven project, Modules, and Rename. The main content area displays the project name and a 'Permalinks' section. Below this, there is a 'Builds' section with a 'No builds' message.



The screenshot shows the 'Builds' section of the Jenkins dashboard. It features a search bar labeled 'Filter' and a list of builds. The builds are listed under the heading 'Today' and include a status icon, build number, and time. Build #5 is successful (green checkmark), while builds #4 and #3 are failed (red X).

Status	Build Number	Time
✓	#5	02:05
✗	#4	02:00
✗	#3	01:29


## Viewing Build Details

After triggering the build, you can view important details about the build process, such as:

- Who executed the build
- When the build was executed
- The time it took to complete
- The overall build status

Click on the **Console Output** to see the detailed logs and any errors that occurred during the build.

## Build Details

 **Jenkins**

Dashboard > RestAssuredFramework > #5

Status

</> Changes

Console Output

Edit Build Information

Delete build '#5'

Timings

Git Build Data

Test Result

Redeploy Artifacts

See Fingerprints

Previous Build

✓ #5 (29 Jan 2025, 02:05:53)

Started by user [Yogesh Pandian](#)

This run spent:

- 6 ms waiting;
- 30 sec build duration;
- 30 sec total from scheduled to completion.

Revision: 45048a03a123c1b08a2e2a3ce0d3248a8c2767ad

Repository: <https://github.com/AutomatedMind/RestAssuredFrameworkPractice.git>

- refs/remotes/origin/main

Test Result (no failures)

Module Builds

✓ [RestAssuredFramework](#)


16 sec

## Console Output

```
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 14.10 s -- in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] [JENKINS] Recording test results
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 16.375 s
[INFO] Finished at: 2025-01-29T02:06:20+05:30
[INFO] -----
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /Users/apple/.jenkins/workspace/RestAssuredFramework/pom.xml to org.example/RestAssuredFramework/1.0-SNAPSHOT/RestAssuredFramework-1.0-SNAPSHOT.pom
channel stopped
Finished: SUCCESS
```

## Checking Build Status

Once the build completes, you'll see a green checkmark if the build passed. If it fails, you can find detailed error messages in the **Console Output** to help diagnose the issue.



Jenkins

Search (⌘+K)

Yogesh Pandian

log out

Dashboard

+ New Item

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views



Build Queue

No builds in the queue.

Build Executor Status

0/2

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
		RestAssuredFramework	2 min 56 sec #5	8 min 16 sec #4	30 sec

Icon:

S

M

L

Add description

With these steps, we can successfully set up Jenkins to run Maven projects from GitHub. Now we know how to **configure Jenkins, install necessary plugins, link your GitHub repository, and run your automation tests** using **Maven commands**.