


Step 1: Create a New Jenkins Job



- Enter job name: **Assignment01**
- Select **Freestyle project**
- Click **OK**



← → ↺ ⚠ Not Secure 18.218.197.178:8080/view/all/newJob ☆ 🔍 📄 📁 📌 📎 📧 ⚙️ Finish update ⌵

 **Jenkins**

🔍 🔔 🔒 👤 Abdul Rehman Shaik ⌵ 📄 log out


Dashboard > All > New Item


New Item


Enter an item name


Assignment01


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.

 **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

 **Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

 **Multibranch Pipeline**
Creates a set of Pipelines automatically detected branches in your SCM repository.

OK

Step 2: Configure Source Code Management

📌 **Navigation:** [Configure](#) → [Source Code Management](#) → [Git](#)

📝 **Action:**

- **Repository URL:**
<https://github.com/AbdulRehaman082493/mahalogin.git>
- **Credentials:** Left as - [none](#) - (public repo, so credentials not needed)
- **Branches to build:**
[*/master](#)

👛 **Use Case:** You pull code from the [master](#) branch of a public GitHub repo. In a real team setup, this could be your main development or deployment branch.

The screenshot shows the Jenkins web interface at the URL `18.218.197.178:8080/job/Assignment01/configure`. The left sidebar contains a 'Configure' menu with options: General, Source Code Management (selected), Triggers, Environment, Build Steps, and Post-build Actions. The main content area is titled 'Source Code Management' and includes the instruction: 'Connect and manage your code repository to automatically pull the latest code for your builds.' There are two radio buttons: 'None' and 'Git' (selected). Below the 'Git' button is a 'Repositories' section with a text input for 'Repository URL' containing `https://github.com/AbdulRehaman082493/mahalogin.git` and a dropdown for 'Credentials' set to '- none -'. There is an 'Add' button and an 'Advanced' dropdown. Below this is an 'Add Repository' button. The 'Branches to build' section has a text input for 'Branch Specifier (blank for \'any\')' containing `*/master`, with an 'Add Branch' button below it. The 'Repository browser' dropdown is set to '(Auto)'. At the bottom are 'Save' and 'Apply' buttons.

GitHub Link: <https://github.com/AbdulRehaman082493/mahalogin>

Step 3: Set Environment Settings

📌 **Navigation:** `Configure → Environment`

📝 **Action:**

- ☒ **Checked:** `Add timestamps to the Console Output`
(Helps in log tracking during troubleshooting)

👛 **Use Case:** In real CI/CD pipelines, timestamps help trace delays or errors in build stages.

The screenshot shows the Jenkins web interface at the URL `18.218.197.178:8080/job/Assignment01/configure`. The left sidebar contains a 'Configure' menu with options: General, Source Code Management, Triggers, Environment (selected), Build Steps, and Post-build Actions. The main content area is titled 'Environment' and includes a description: 'Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.' Below this, there are several checkboxes: 'Delete workspace before build starts' (unchecked), 'Use secret text(s) or file(s)' (unchecked), 'Add timestamps to the Console Output' (checked), 'Inspect build log for published build scans' (unchecked), 'Terminate a build if it's stuck' (unchecked), and 'With Ant' (unchecked). The 'Build Steps' section is titled 'Automate your build process with ordered tasks like code compilation, testing, and deployment.' It contains a single step named 'Invoke top-level Maven targets' with a goal of 'clean package'. The 'Post-build Actions' section is titled 'Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.' It contains a single action named 'Archive the artifacts' with files to archive set to 'target/*war'. At the bottom, there are 'Save' and 'Apply' buttons.


Step 4: Add Build Steps

📌 **Navigation:** `Configure → Build Steps → Invoke top-level Maven targets`

📝 **Action:**

- **Goal:**
`clean package`
This tells Maven to clean previous build artifacts and package the application (i.e.,

create the `.war` file).

 **Use Case:** Used in real projects to compile and package code before testing or deployment.

Step 5: Post-Build Action


 **Navigation:** [Configure](#) → [Post-build Actions](#) → [Archive the artifacts](#)

 **Action:**

- **Files to archive:**

`target/*.war`


This saves the generated `.war` file for further use (e.g., deploy to Tomcat or store in Nexus/Artifactory).

 **Use Case:** Storing artifacts is common in enterprise pipelines to track builds and support deployments.

Step 6: Trigger the Build

 **Navigation:** [Dashboard](#) → [Assignment01](#) → [Build Now](#)

 **Action:** Click [Build Now](#) to manually trigger the first build.

 **Use Case:** Manually triggering a build is often used in early development stages or for on-demand testing.


Step 7: Check Console Output

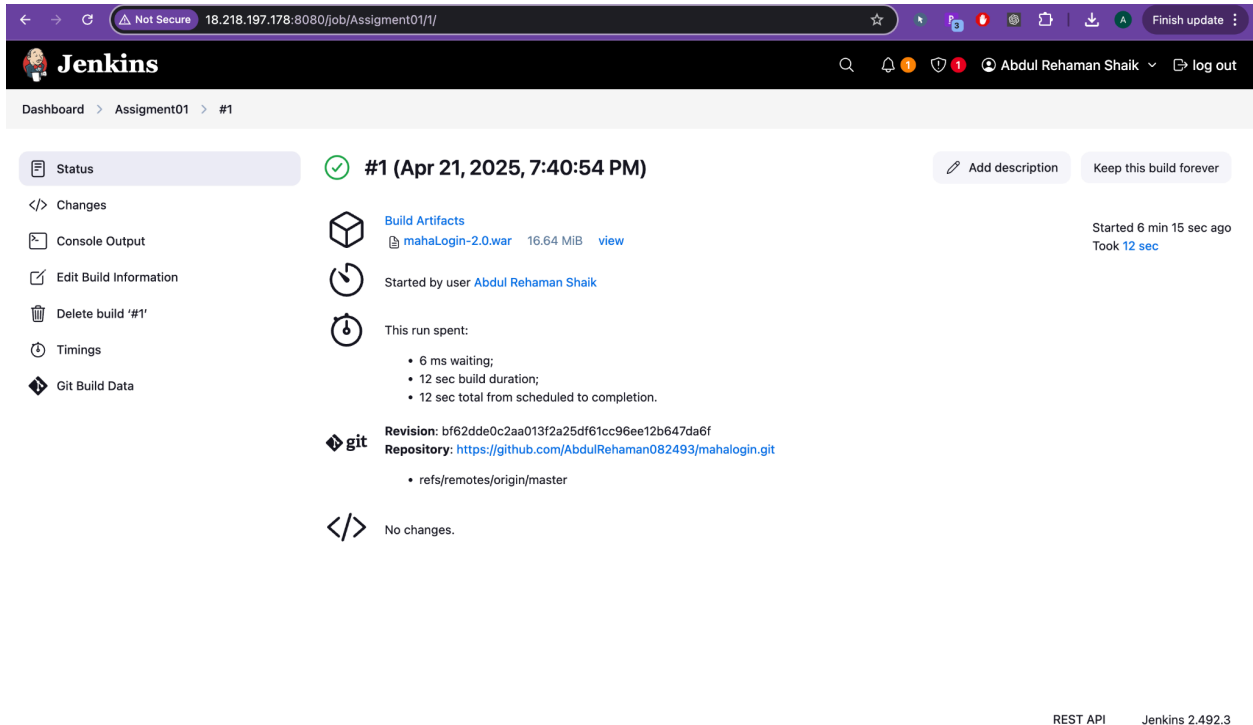
 **Navigation:** [Assignment01](#) → [#1](#) → [Console Output](#)

What Happened:

- Git cloned the repo successfully.
- Maven executed **clean package**.
- Build logs and progress are shown.
- WAR file is generated and archived.


Build Result: SUCCESS



 **Use Case:** In a real-world CI pipeline, logs help confirm that the code compiled correctly and artifacts were generated.





The screenshot shows the Jenkins web interface for a build named '#1' under the job 'Assignment01'. The build is successful, indicated by a green checkmark and the status '#1 (Apr 21, 2025, 7:40:54 PM)'. The left sidebar contains links to 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete build '#1'', 'Timings', and 'Git Build Data'. The main content area displays build details: 'Build Artifacts' showing a file 'mahaLogin-2.0.war' (16.64 MiB) with a 'view' link; 'Started by user Abdul Rehaman Shaik'; 'This run spent:' with a list of timings (6 ms waiting, 12 sec build duration, 12 sec total); 'Revision: bf62dde0c2aa013f2a25df61cc96ee12b647da6f' and 'Repository: https://github.com/AbdulRehaman082493/mahalogin.git'; and 'No changes' under the 'Changes' section. The bottom right corner shows 'REST API' and 'Jenkins 2.492.3'.

Dashboard > Assignment01 > #1


Status  #1 (Apr 21, 2025, 7:40:54 PM) [Add description](#) [Keep this build forever](#)

 Build Artifacts
 mahaLogin-2.0.war 16.64 MiB [view](#) Started 6 min 15 sec ago
Took 12 sec


 Started by user [Abdul Rehaman Shaik](#)

 This run spent:

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










 **git** Revision: bf62dde0c2aa013f2a25df61cc96ee12b647da6f
Repository: <https://github.com/AbdulRehaman082493/mahalogin.git>

- refs/remotes/origin/master

 No changes.

REST API Jenkins 2.492.3






Jenkins Build Status Colors

Color	Meaning	Details
 Blue or  Green (some themes)	Success	Build completed without errors
 Red	Failed	Build failed due to code issues, test failure, etc.
 Yellow or  Orange	Unstable	Build succeeded, but with issues (e.g., failing tests)
 Grey or  Black	Not built	Job was never built, or build was skipped
 Light Blue or  Blue (animated)	Running	Build is in progress

 Note: Jenkins originally used **blue for success**, but many themes and plugins now show **green** instead.

Jenkins Weather Report Icons (Job Health Trends)

These icons are based on the **build success percentage over time** (usually last 5 builds):

Icon	Label	Success %	Meaning
 Sun	Sunny	100%	All recent builds were successful – healthy job
 Sun with Cloud	Mostly Sunny	~80–99%	Mostly good, occasional failure
 Cloudy	Cloudy	~60–79%	Frequent issues, needs attention
 Rain Cloud	Rainy	~0–59%	Many failures – unhealthy job
 No icon (Stormy)	Worst Case	0%	All builds failed recently

