## **🔧 Jenkins Job Setup: Assignment01 (Freestyle Project)**

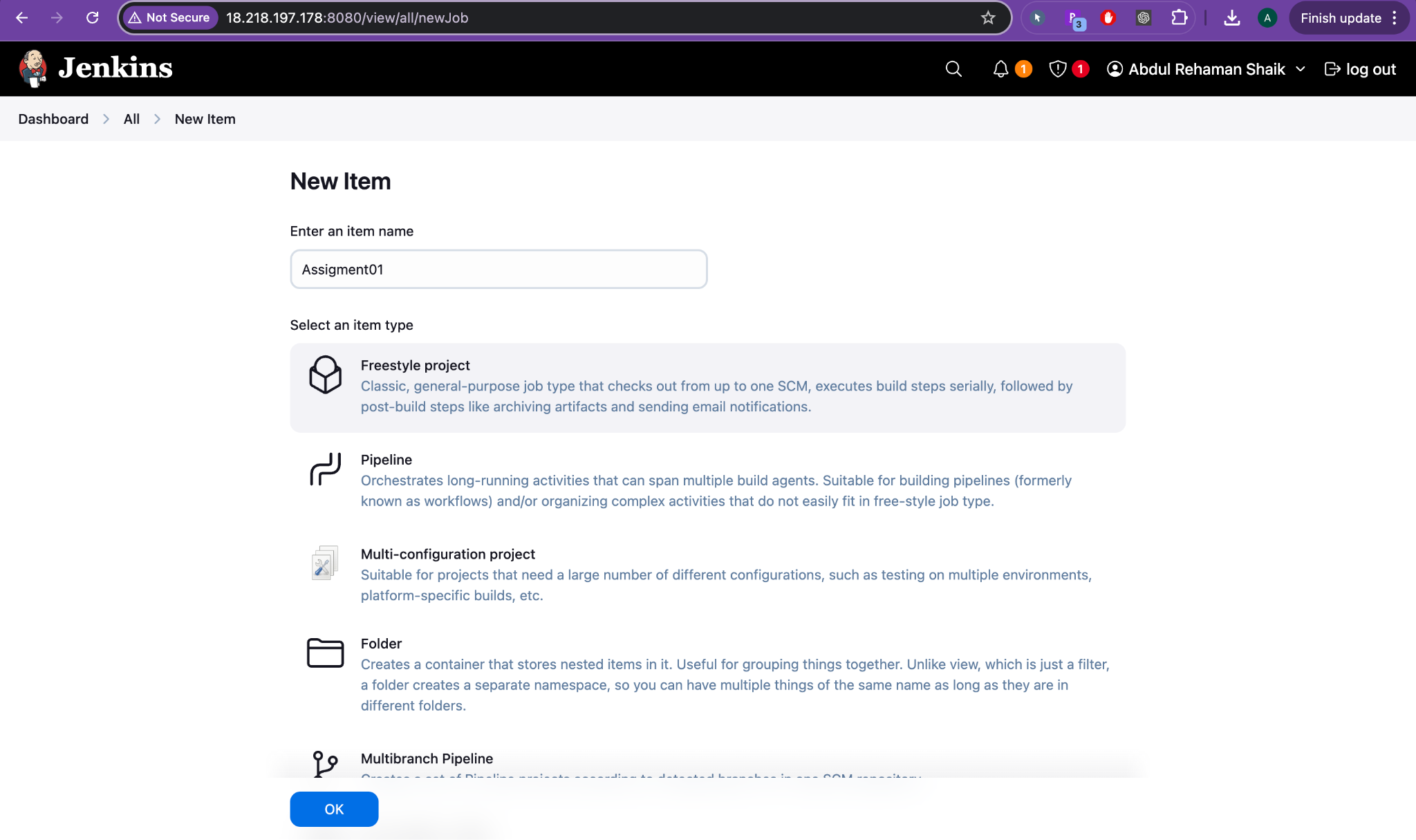
### **Step 1: Create a New Jenkins Job**

📍 **Navigation:** Dashboard → New Item

📝 **Action:**

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

💼 **Use Case:** You want to build and package a Java web application (WAR file) from a GitHub repository using Maven.



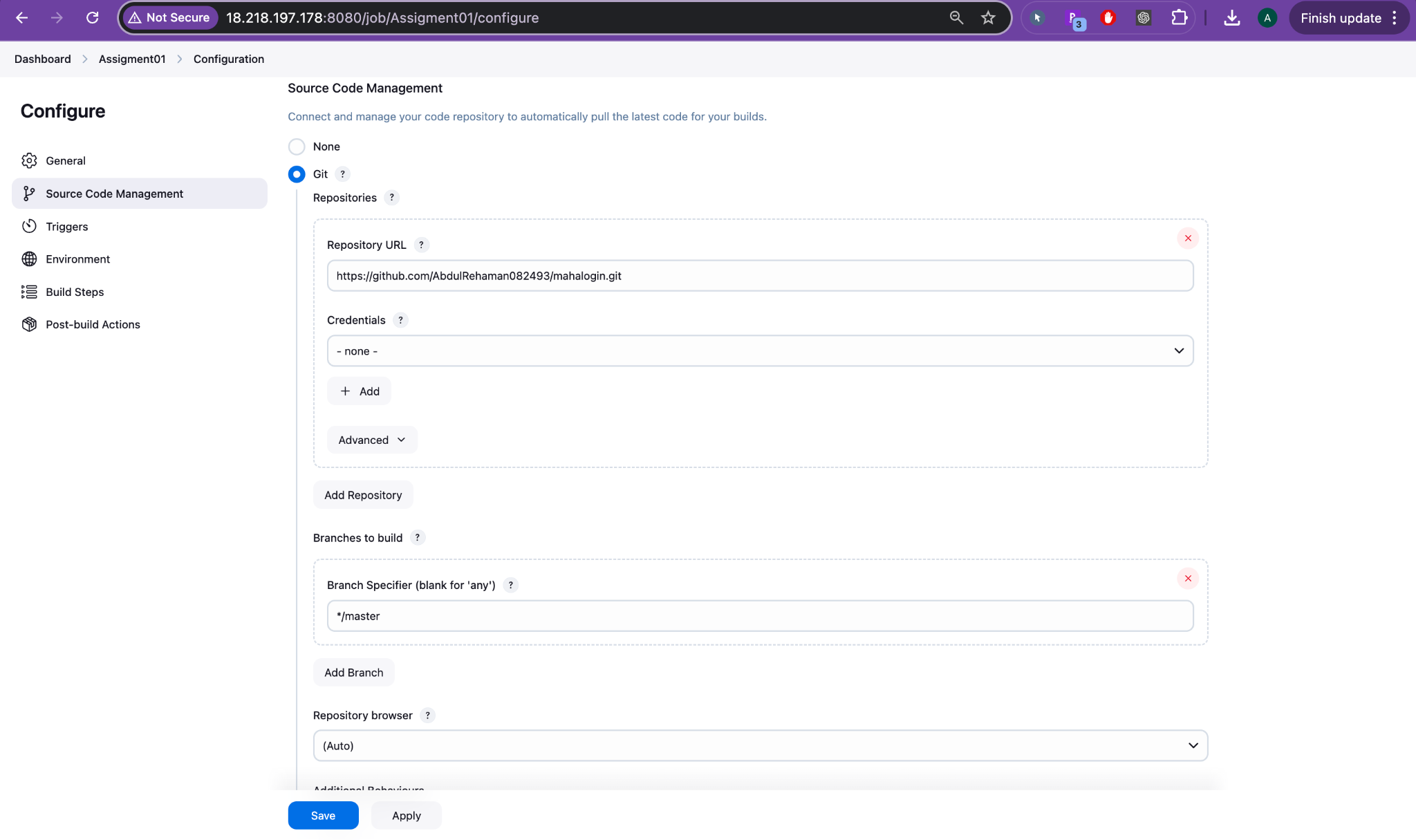
### **Step 2: Configure Source Code Management**

📍 **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.



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

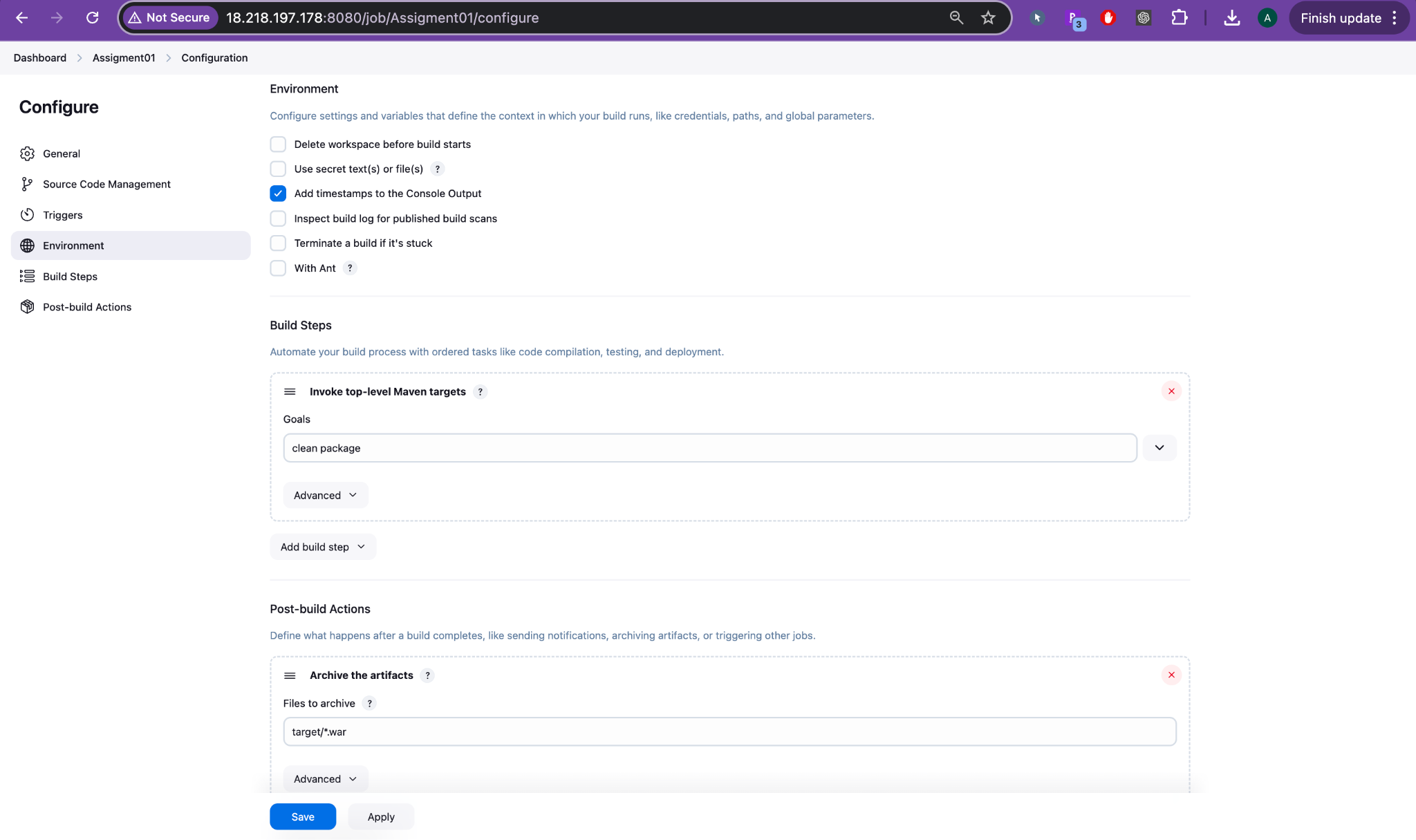
### **Step 3: Set Environment Settings**

📍 **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.



### **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**

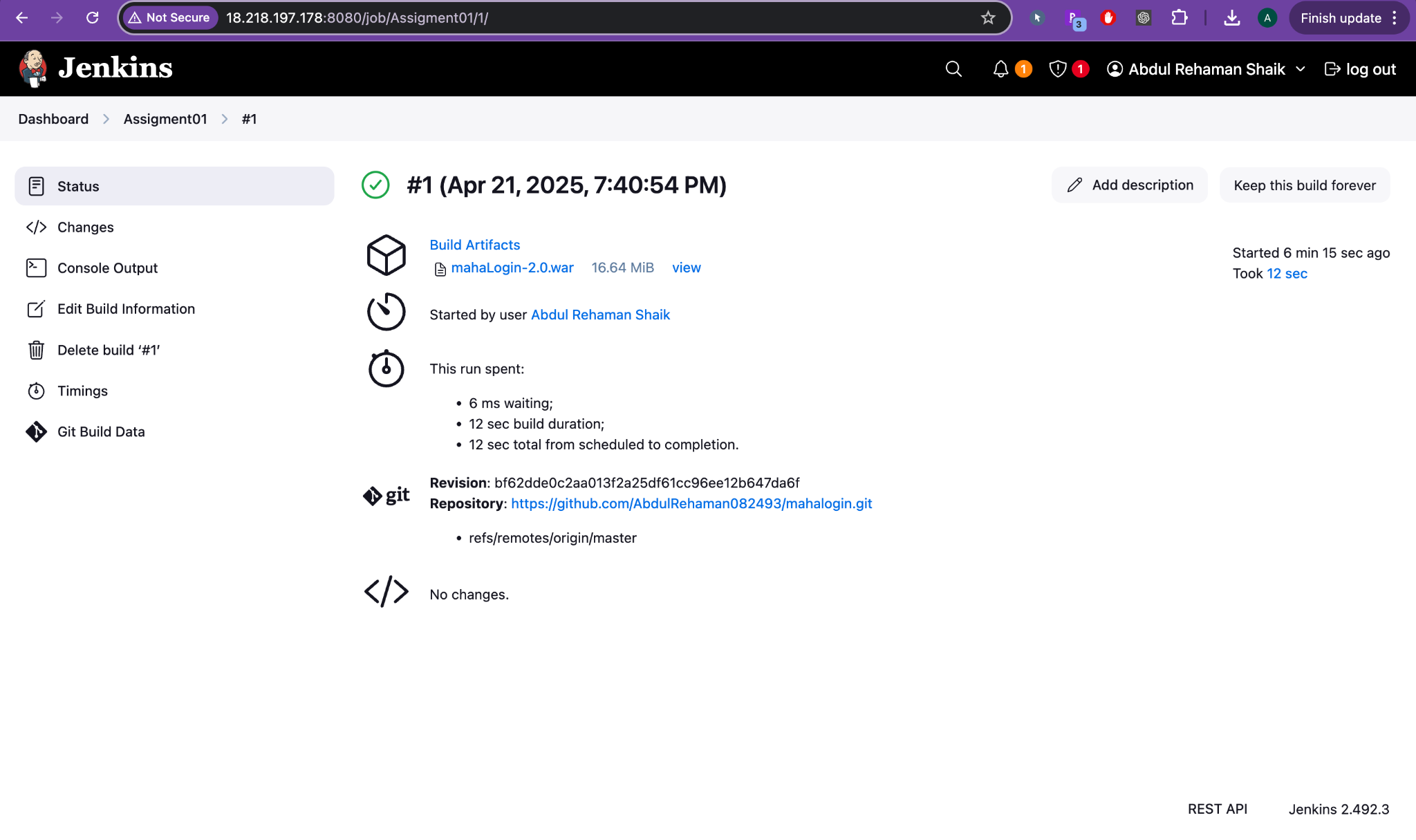
📍 **Navigation:** Assignment01 → #1 → Console Output

📝 **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.



## 

## 

## **🎨 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 |