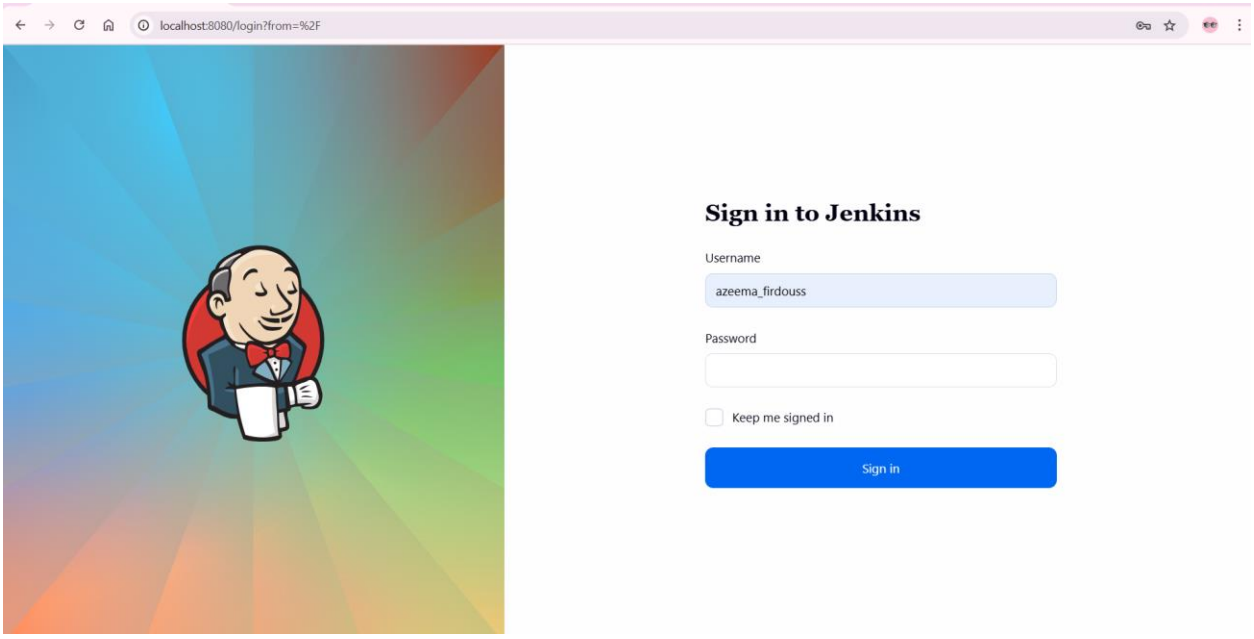
# Week 8: Jenkins Automation

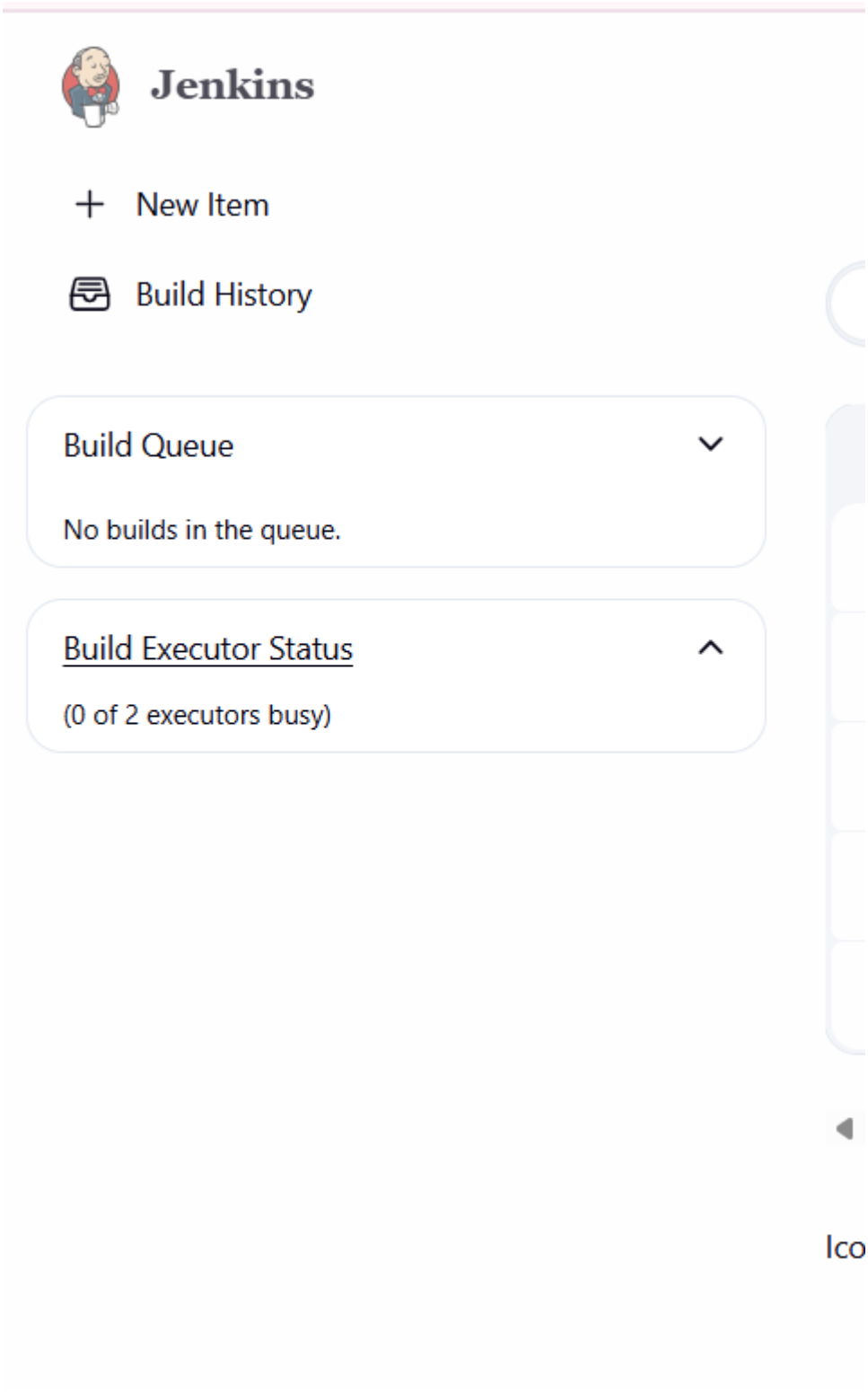
# I. Steps for MavenJava Automation:

Maven Java Automation Steps:

**Step 1: Open Jenkins (localhost:8080)**



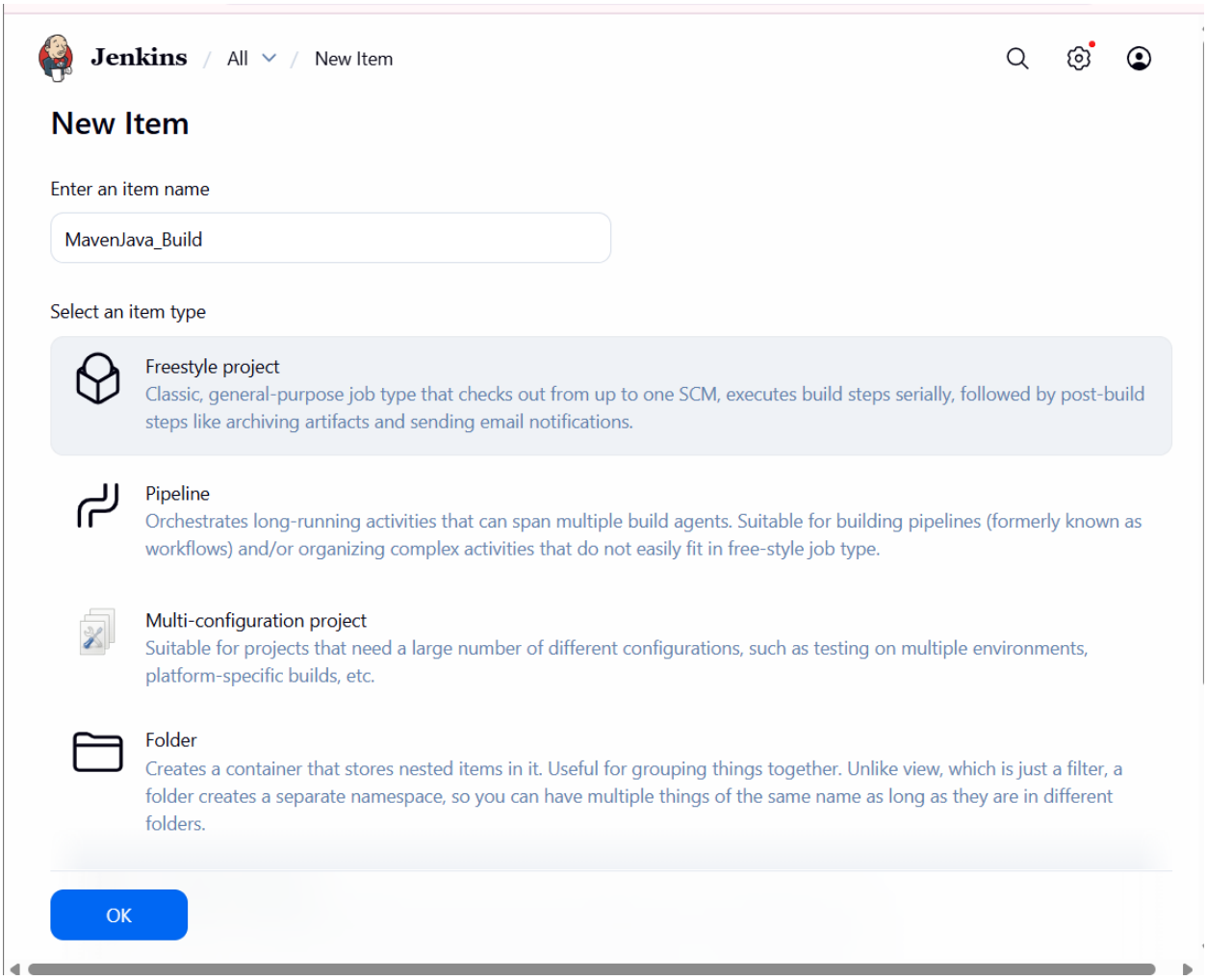

├── Click on "New Item" (left side menu)




**Step 2: Create Freestyle Project (e.g., MavenJava_Build)**

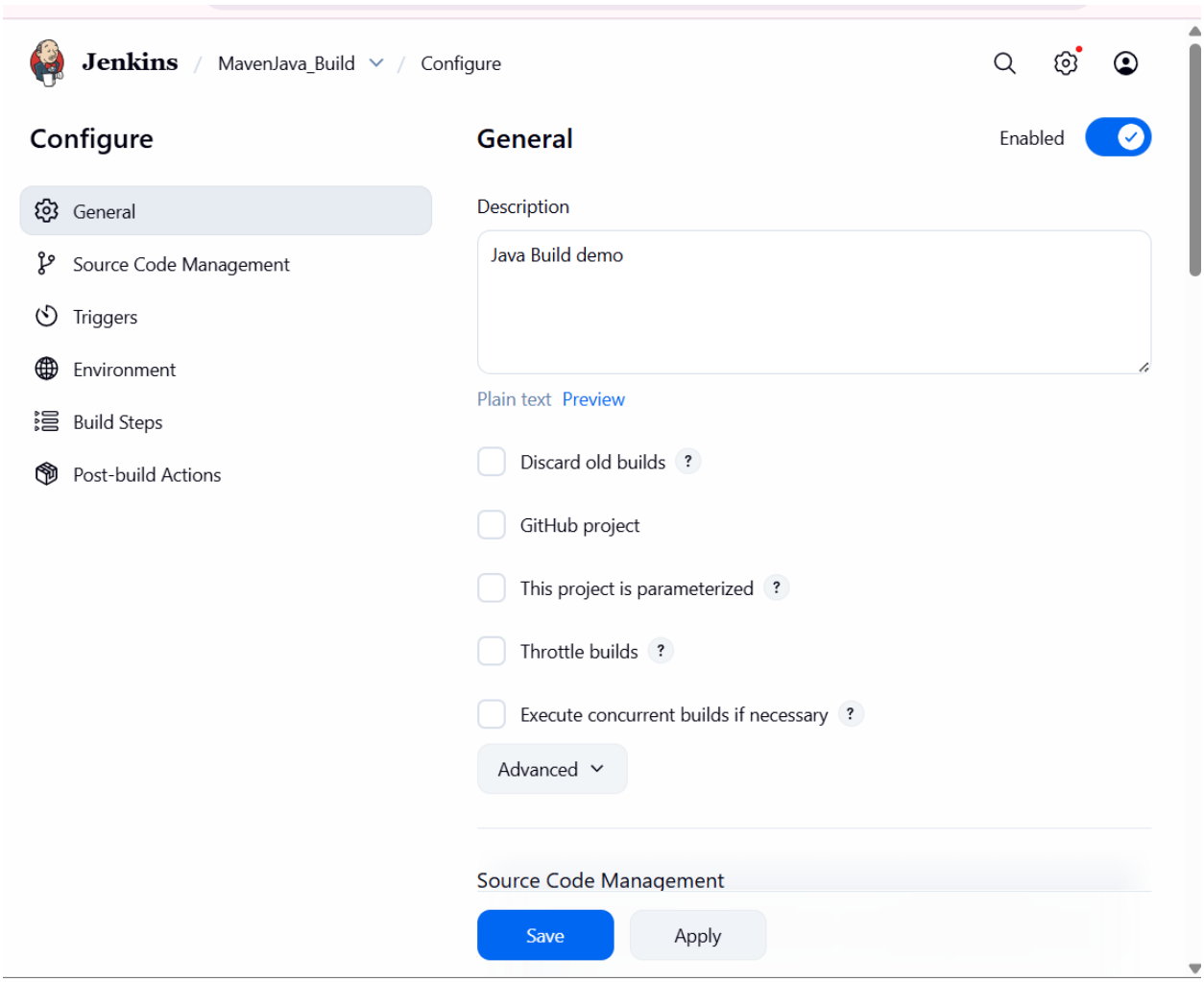
├── Enter project name (e.g., MavenJava_Build)

├── Click "OK"




└── **Configure the project:**

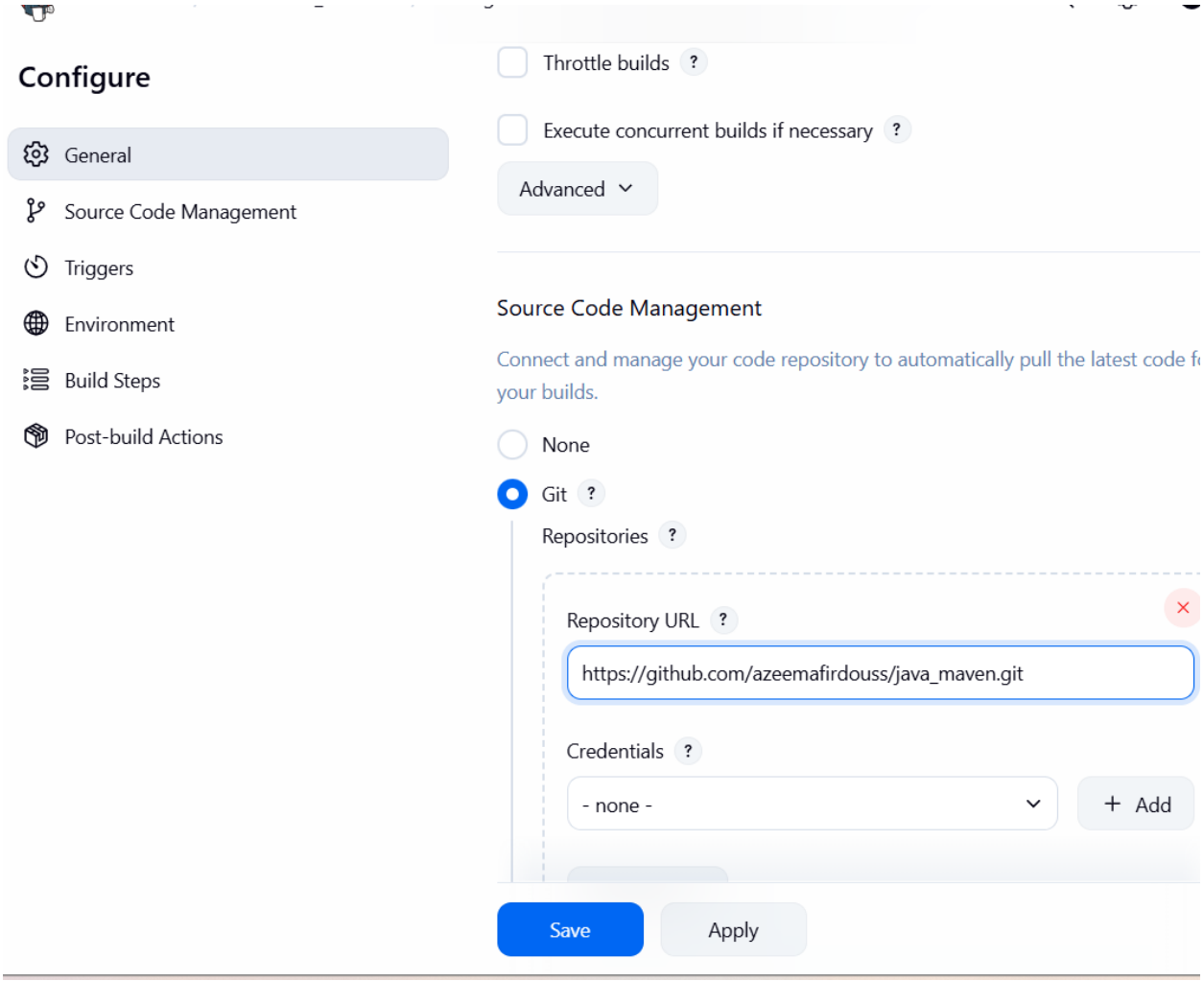
├── **Description**: "Java Build demo"

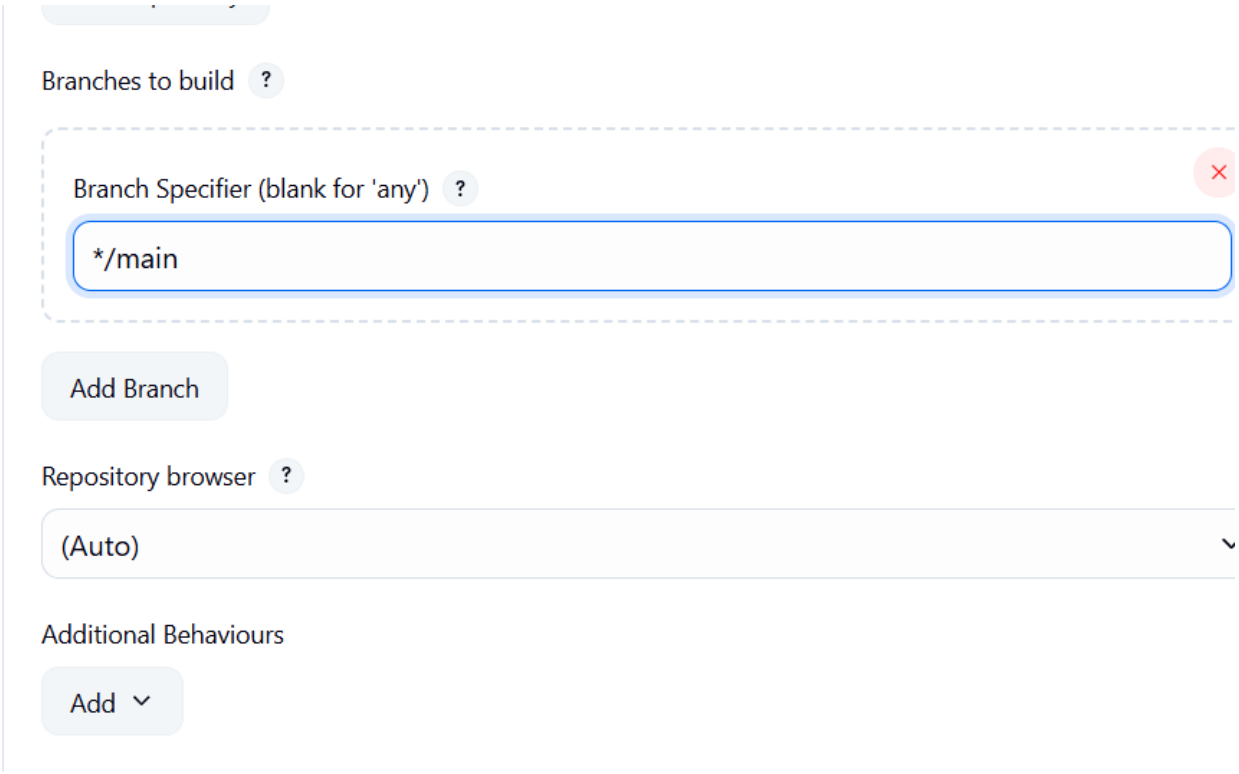



├── **Source Code Management**:

└── Git repository URL: [GitMavenJava repo URL]

├── **Branches to build**: */Main or */master


Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Throttle builds
Execute concurrent builds if necessary
Advanced
Source Code Management
Connect and manage your code repository to automatically pull the latest code f
your builds.
None
Git
Repositories
Repository URL
https://github.com/azeemafirdouss/java_maven.git
Credentials
- none -
Add
Save
Apply




└── **Build Steps**:

├── **Add Build Step** -> "Invoke top-level Maven targets"

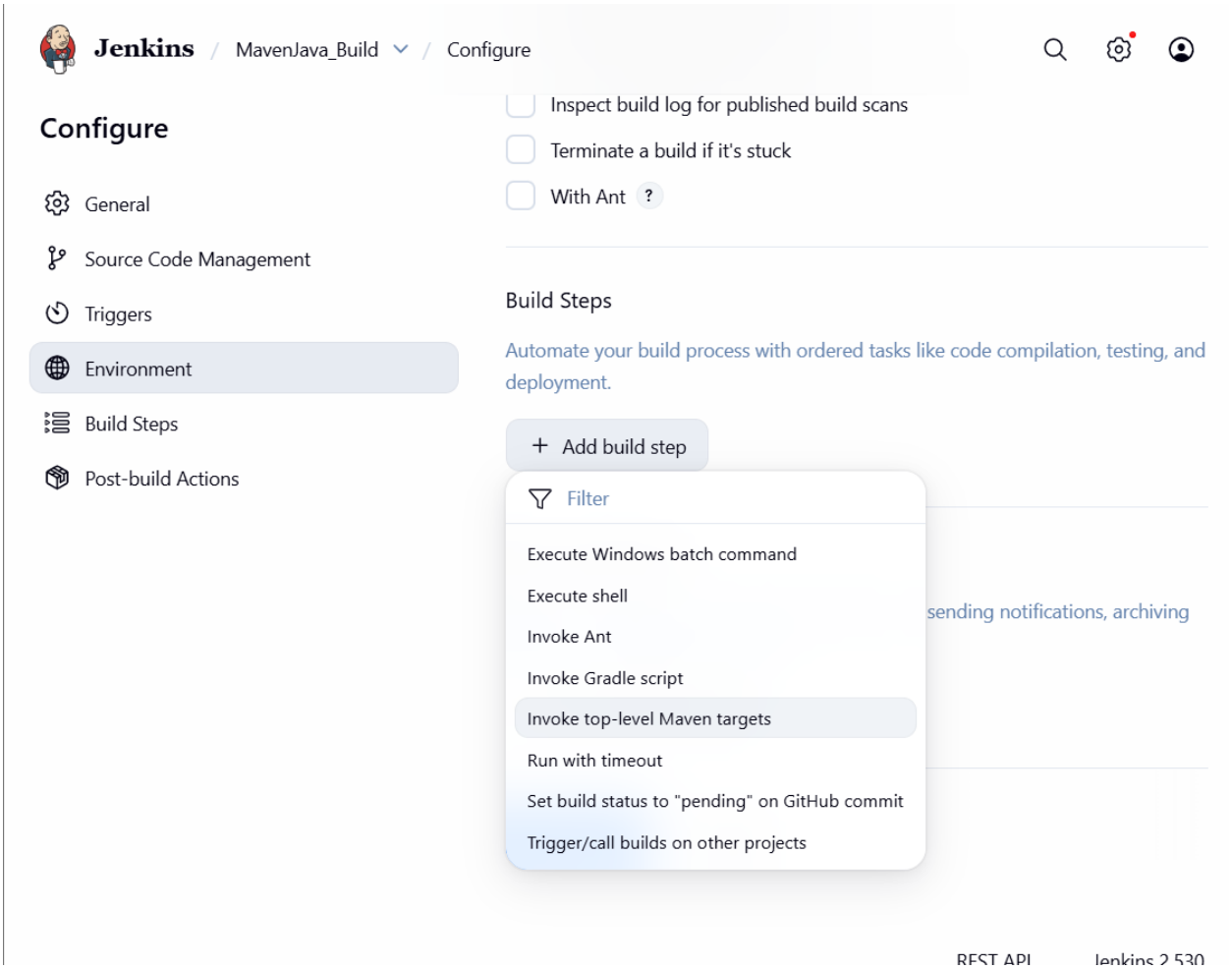
└── Maven version: MAVEN_HOME

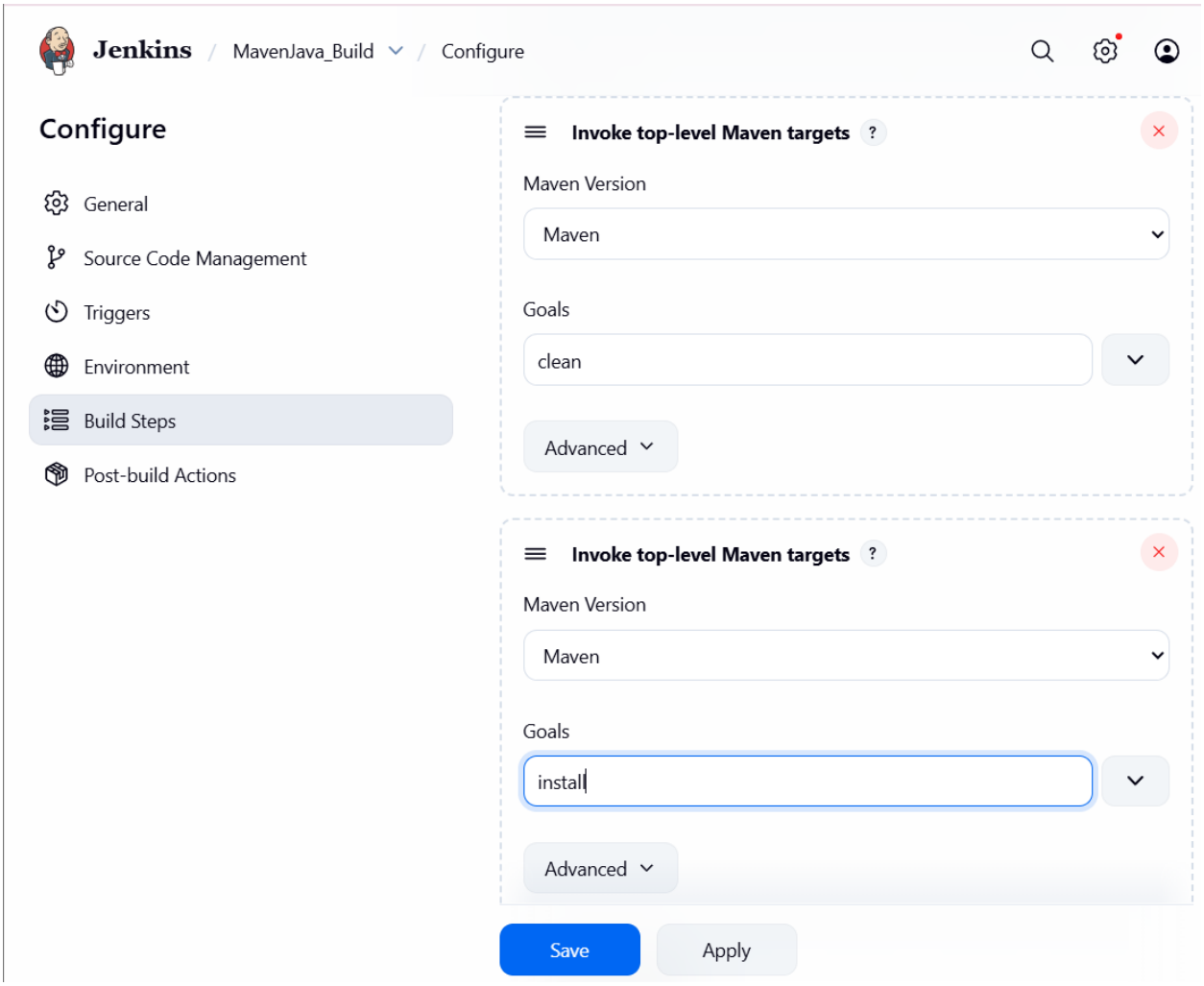
└── Goals: clean

├── **Add Build Step** -> "Invoke top-level Maven targets"

└── Maven version: MAVEN_HOME

└── Goals: install


Jenkins / MavenJava_Build / Configure
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Inspect build log for published build scans
Terminate a build if it's stuck
With Ant
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Add build step
Filter
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
Trigger/call builds on other projects
sending notifications, archiving
REST API
Jenkins 2.530




└── **Post-build Actions**:

├── Add Post Build Action -> "Archive the artifacts"

└── Files to archive: **/*

├── Add Post Build Action -> "Build other projects"

└── Projects to build: MavenJava_Test

└── Trigger: Only if build is stable

└── Apply and Save

artifacts, or triggering other jobs.

**Archive the artifacts** ?

Files to archive ?

**/*

Advanced

**Build other projects** ?

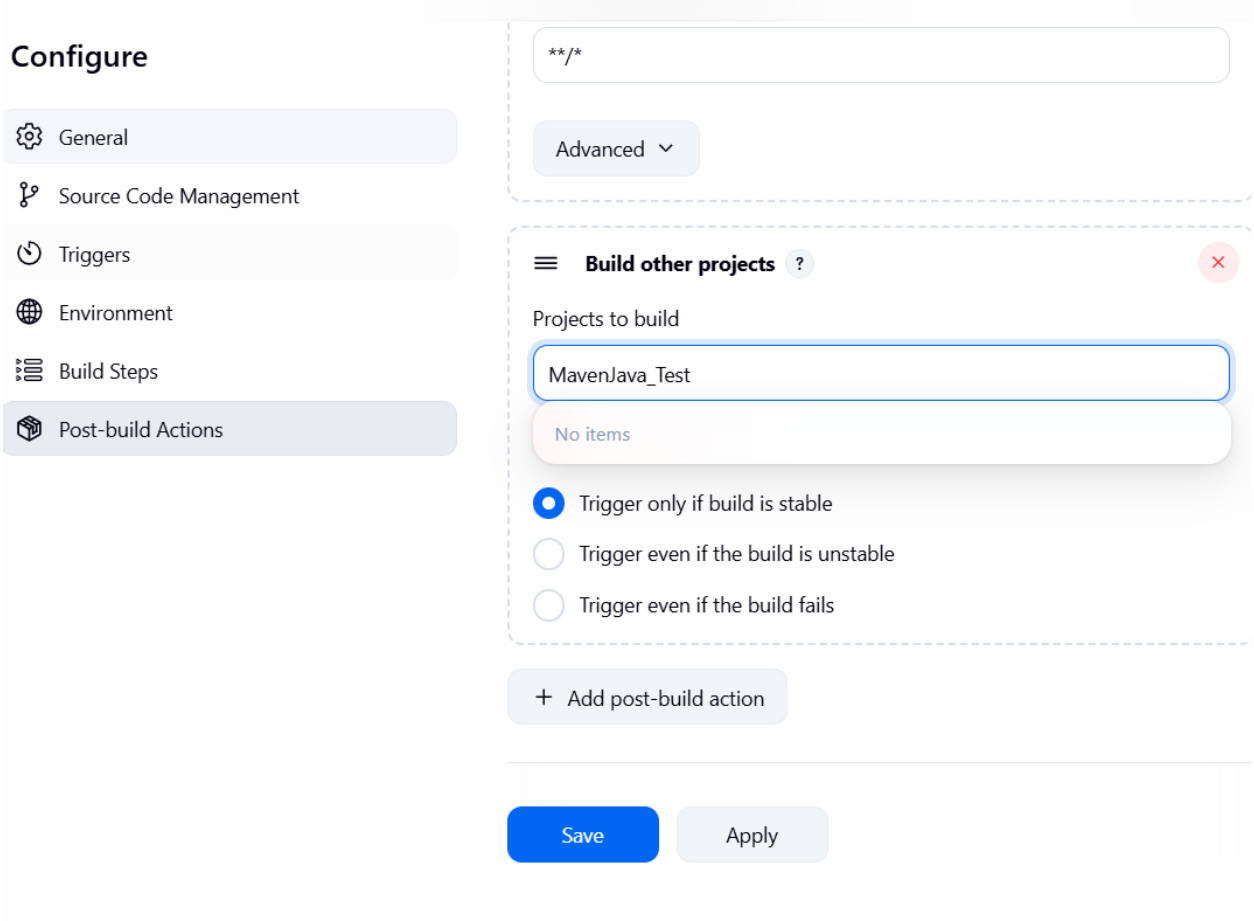
Projects to build

No project specified

- (•) Trigger only if build is stable
- ( ) Trigger even if the build is unstable
- ( ) Trigger even if the build fails

+ Add post-build action

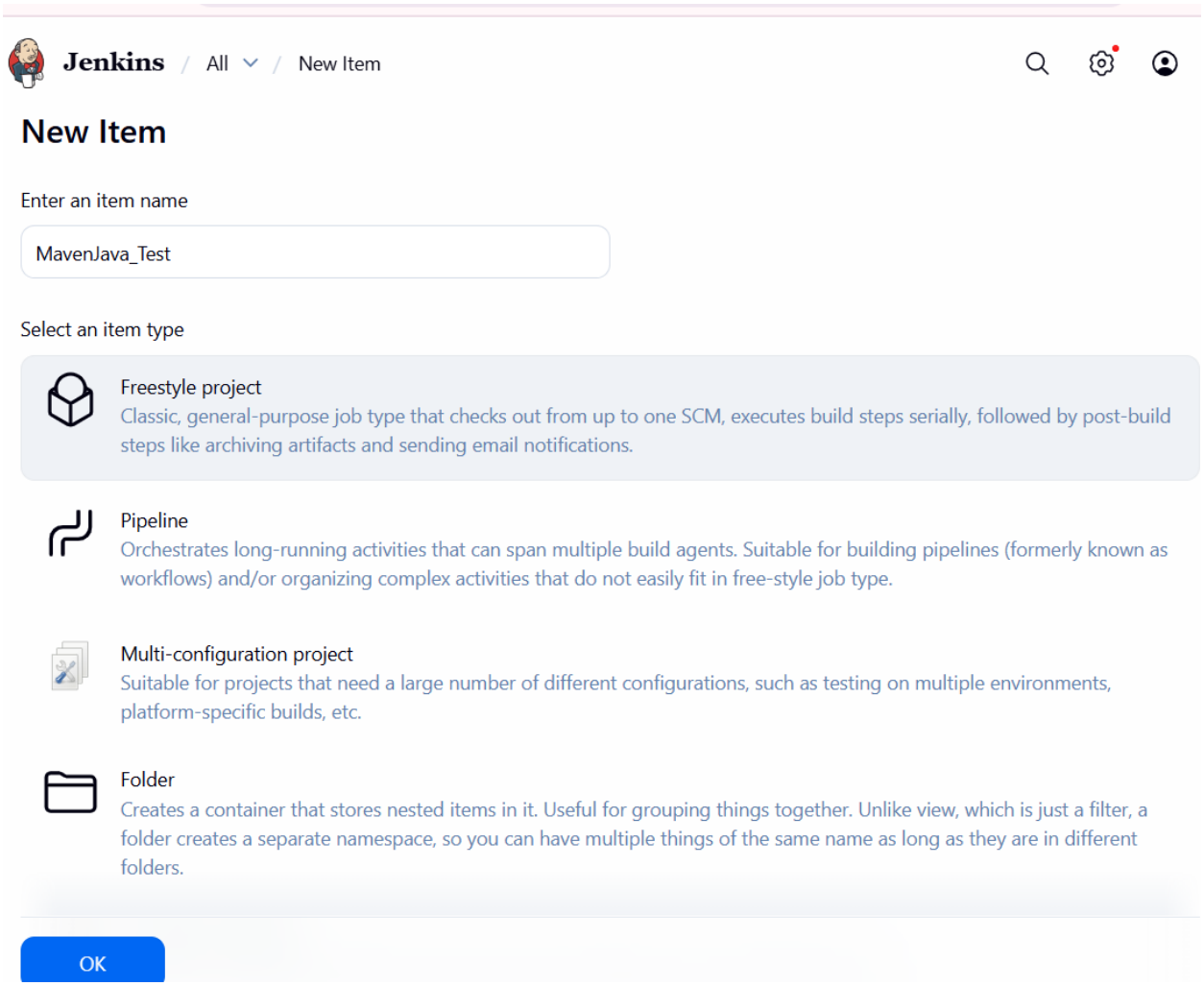
Save Apply




└── **Step 3: Create Freestyle Project (e.g., MavenJava_Test)**

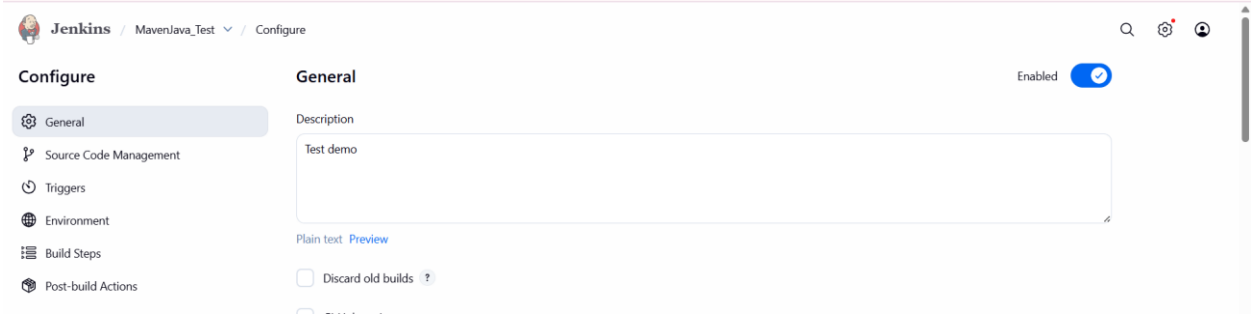
  ├── Enter project name (e.g., MavenJava_Test)

  ├── Click "OK"



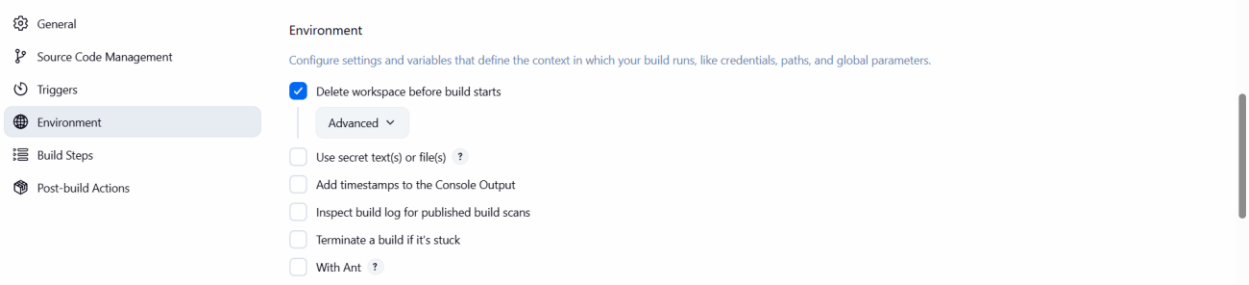

└── **Configure the project**:

├── **Description**: "Test demo"




├── **Build Environment**:

└── Check: "Delete the workspace before build starts"

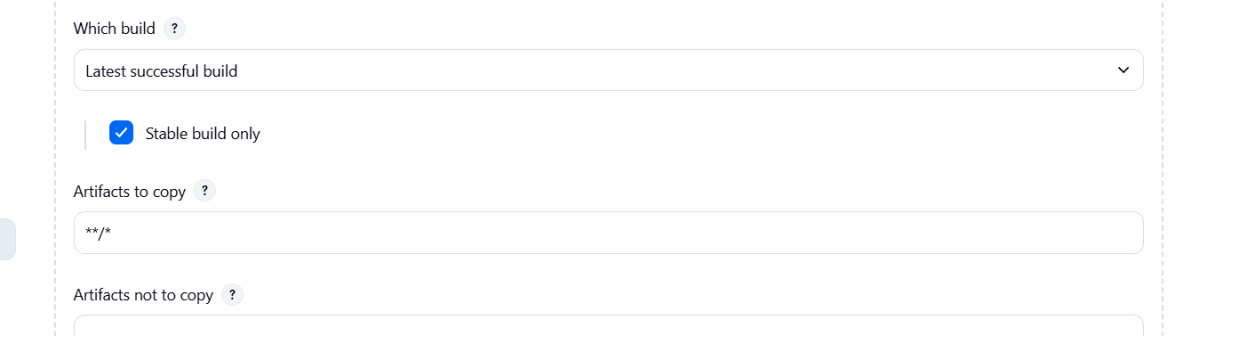



├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenJava_Build

└── Build: Stable build only  // tick at this

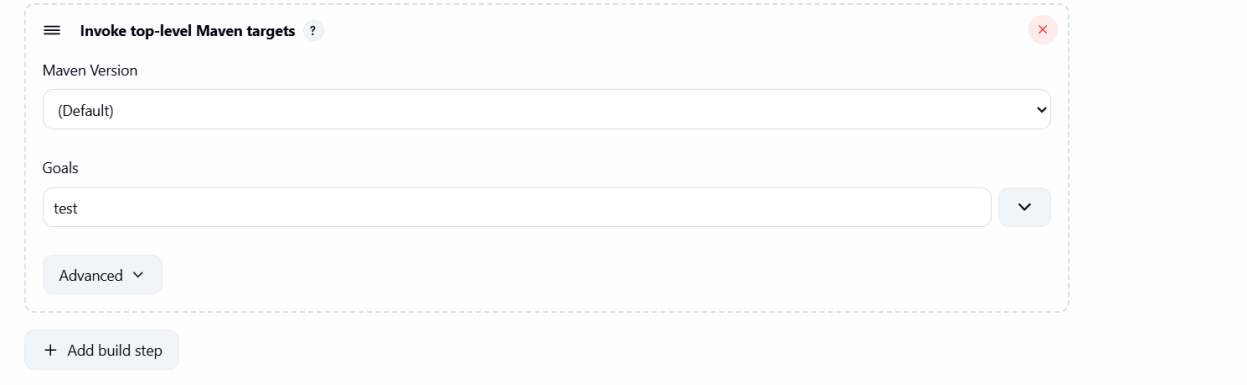

└── Artifacts to copy: **/*




├── **Add Build Step** -> "Invoke top-level Maven targets"

└── Maven version: MAVEN_HOME

└── Goals: test




└── Post-build Actions:

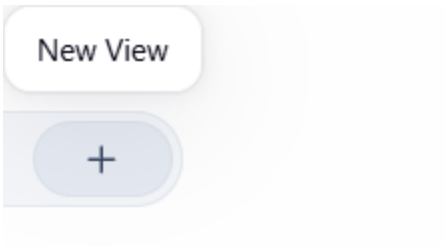

├── **Add Post Build Action** -> "Archive the artifacts"

└── Files to archive: **/*

└── Apply and Save


└── **Step 4**: Create Pipeline View for Maven Java project

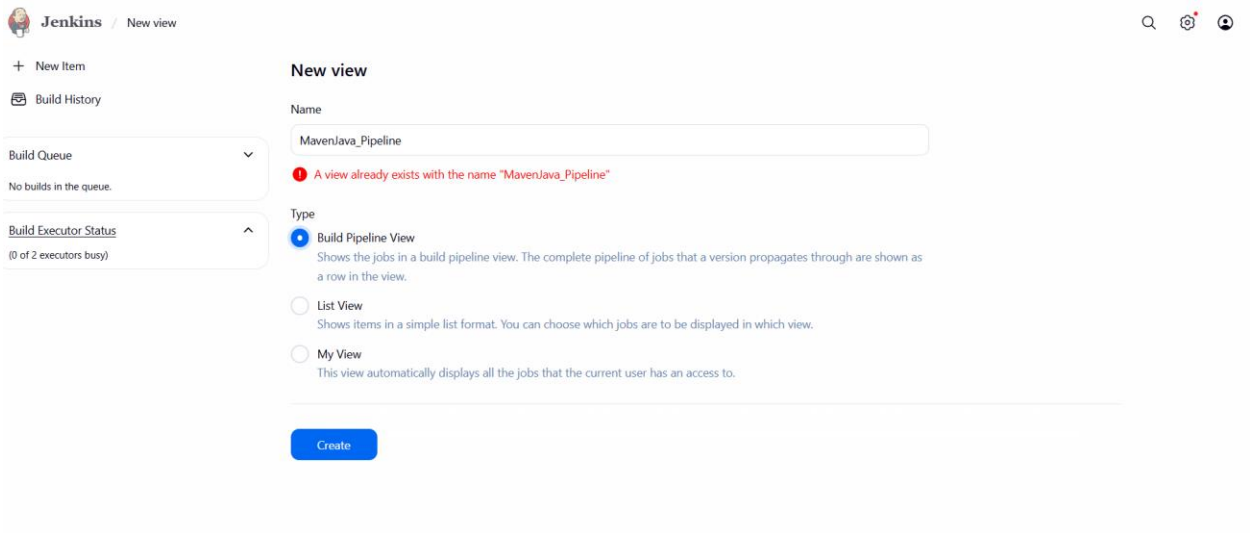
├── Click "+" beside "All" on the dashboard




├── Enter name: MavenJava_Pipeline

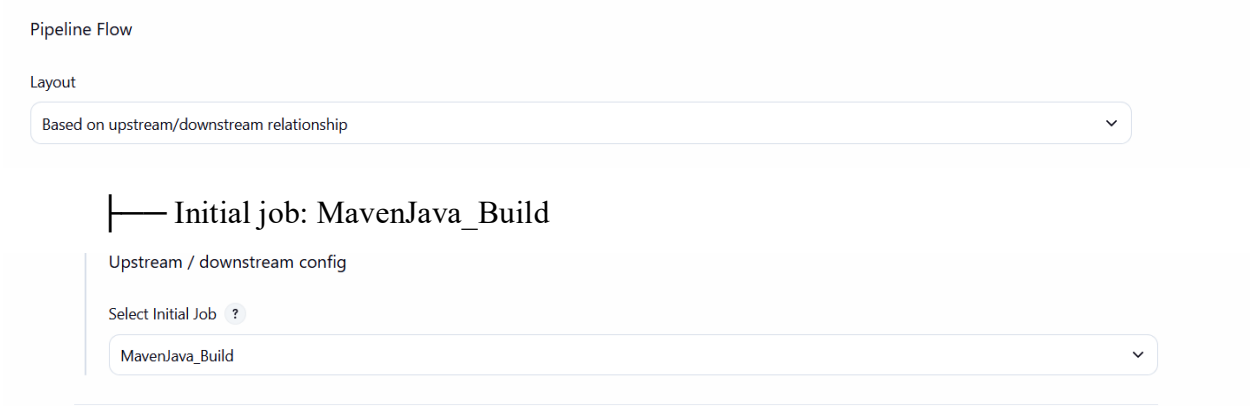
├── **Select "Build pipeline view"  // tick here**

├--- **create**



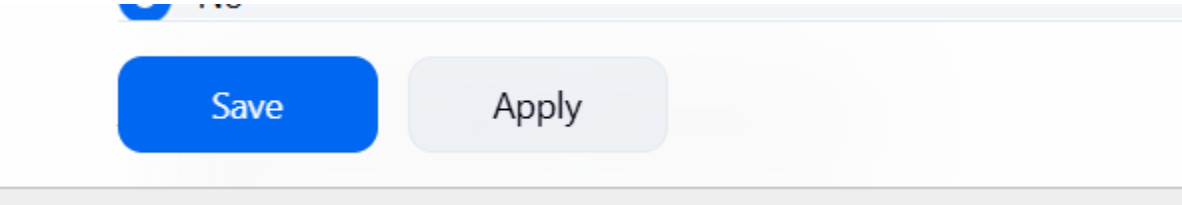

└── **Pipeline Flow**:

├── **Layout**: Based on upstream/downstream relationship




├── Initial job: MavenJava_Build

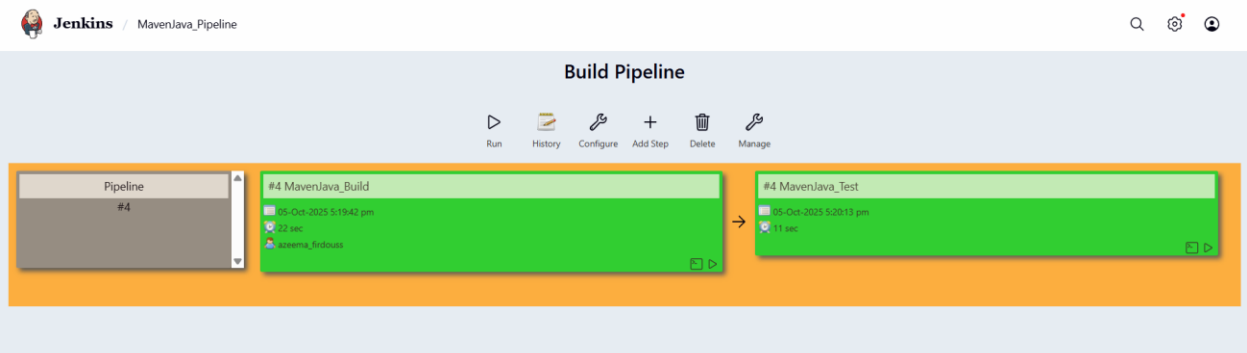
└── Apply and Save OK




└── **Step 5**: Run the Pipeline and Check Output

├── Click on the trigger to run the pipeline

├── click on the small black box to open the console to check if the build is success

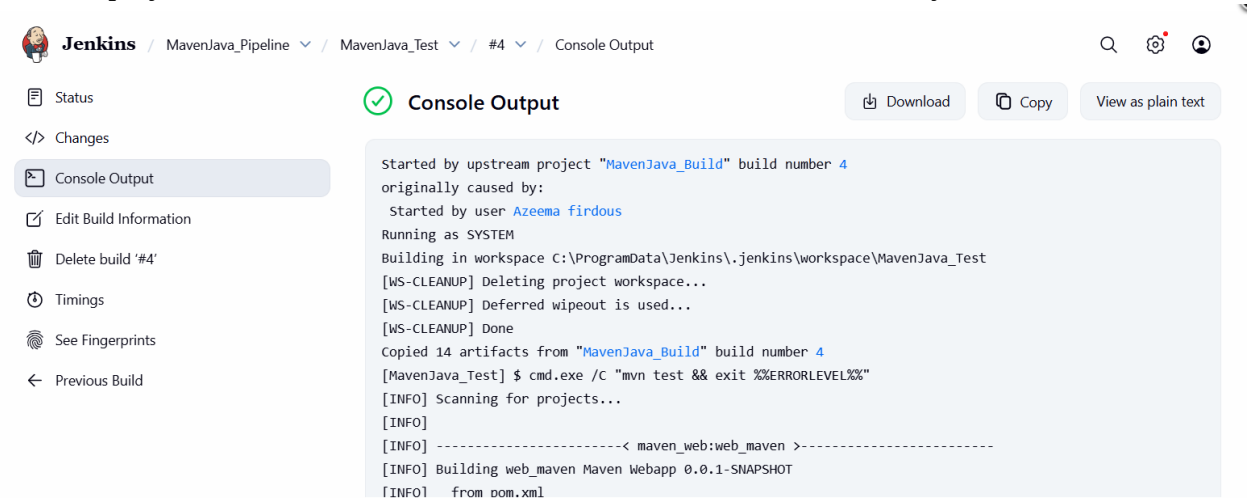


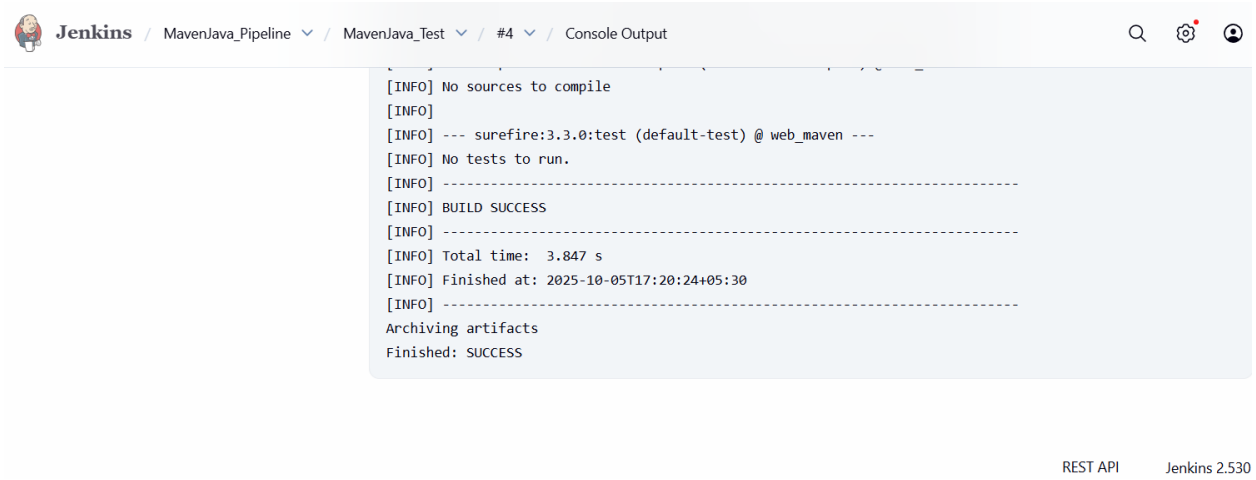

Note :

1. If build is success and the test project is also automatically triggered with name

   "MavenJava_Test"

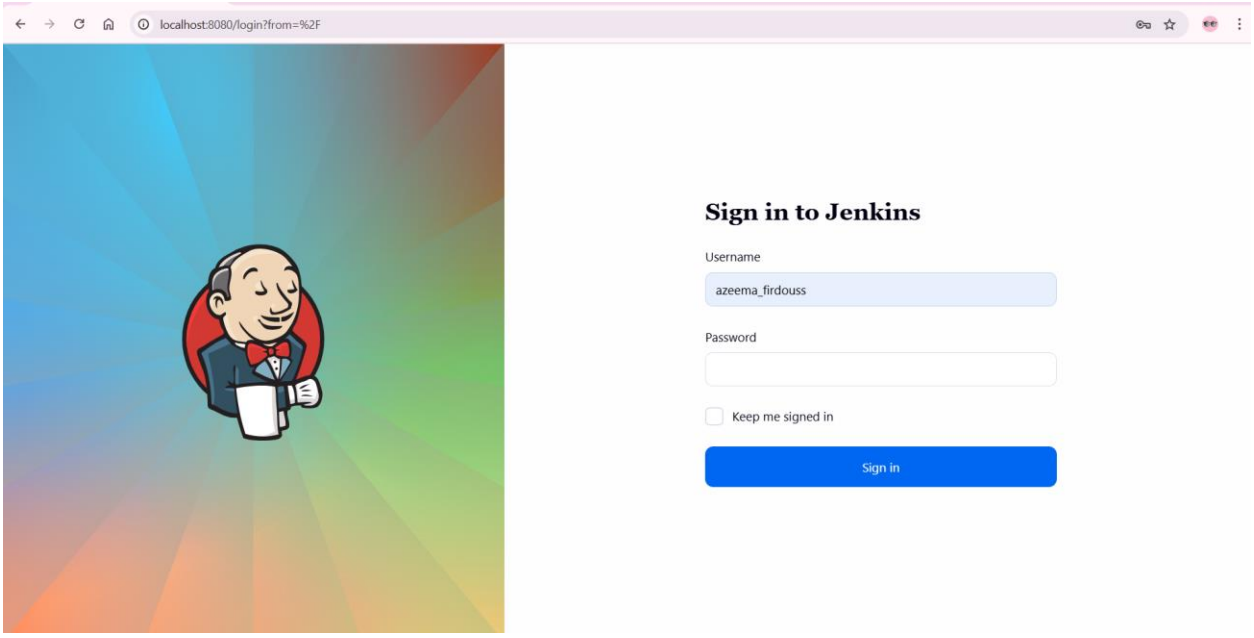
2. The pipeline is successful if it is in green color as shown ->check the console of the test project
3. The test project is successful and all the artifacts are archived successfully






# II. Maven Web Automation Steps:

└── **Step 1:** Open Jenkins (localhost:8080)




├── Click on "New Item" (left side menu)

Jenkins

+ New Item

Build History

Build Queue

No builds in the queue.

Build Executor Status

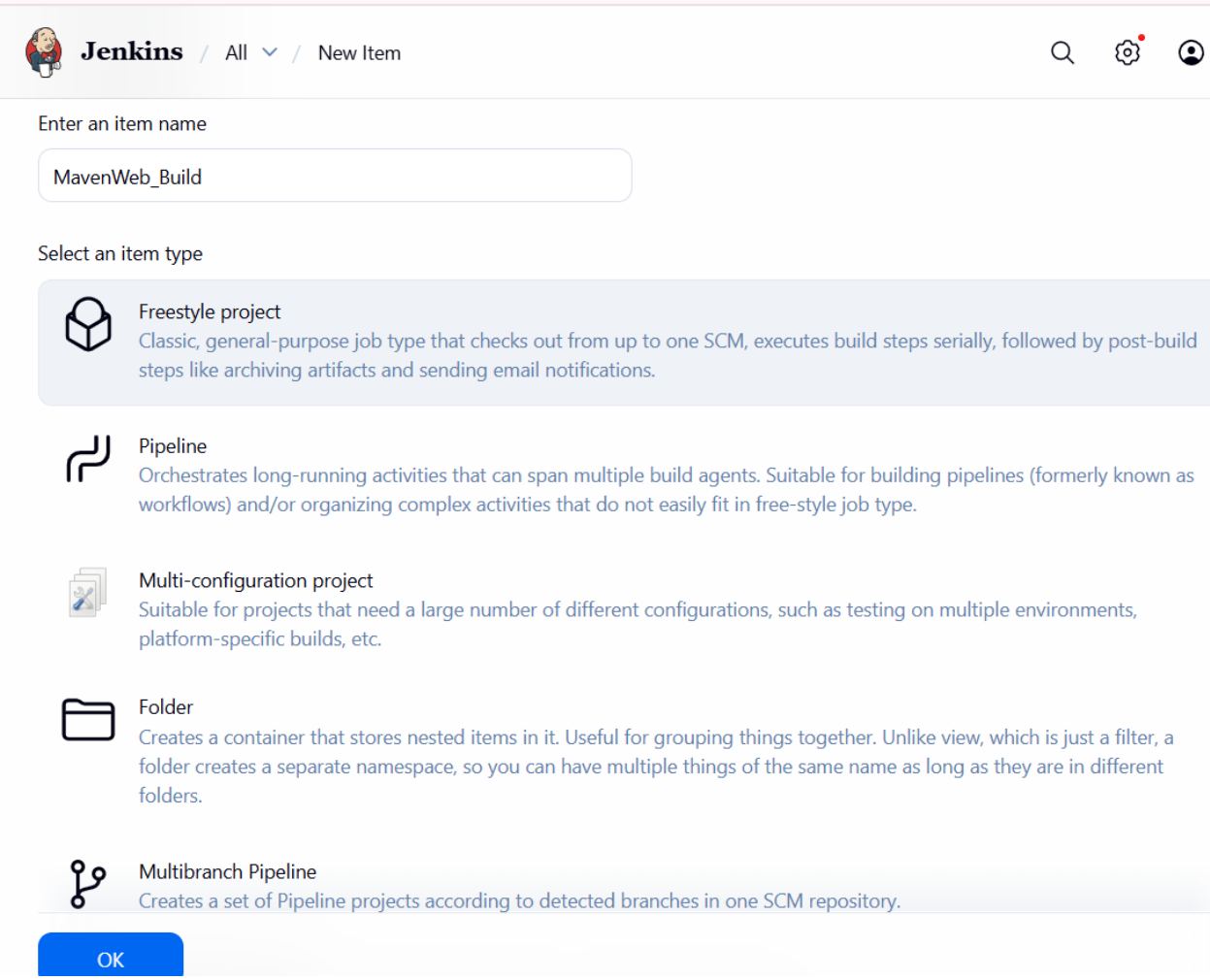
(0 of 2 executors busy)

Ico

└── **Step 2**: Create Freestyle Project (e.g., MavenWeb_Build)

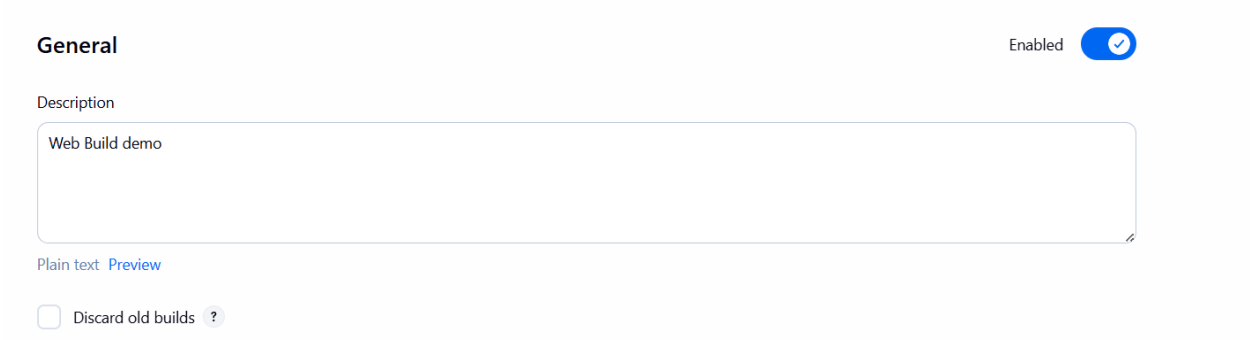
├── Enter project name (e.g., MavenWeb_Build)

├── Click "OK"




└── **Configure the project**:

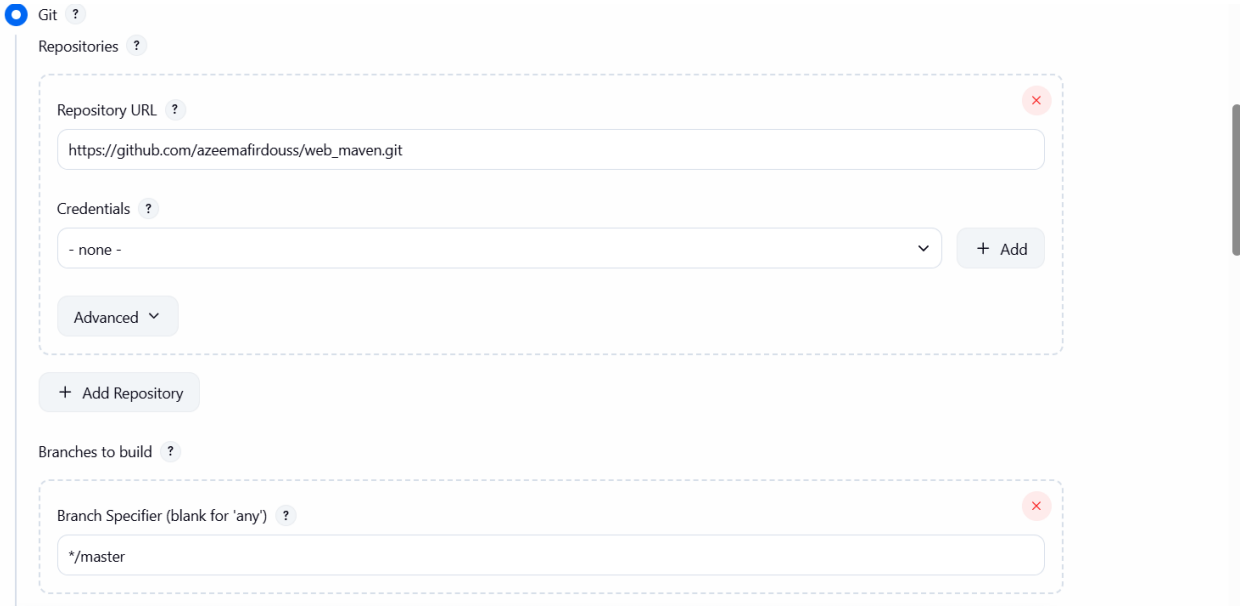
├── **Description**: "Web Build demo"




├── **Source Code Management:**

└── Git repository URL: [GitMavenWeb repo URL]

├── *Branches to build*: */Main or master




└── **Build Steps**:

├── **Add Build Step** -> "Invoke top-level Maven targets"

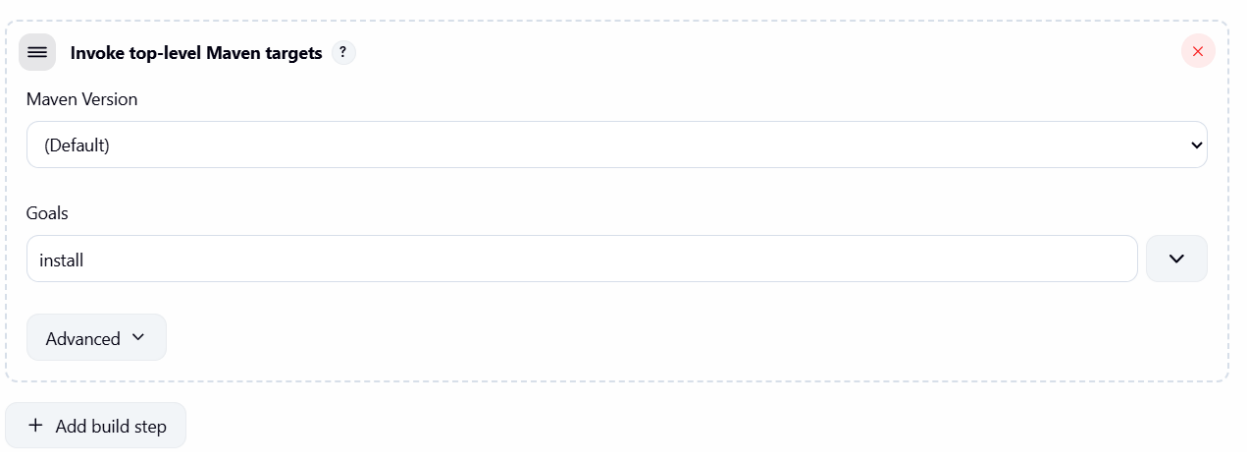
└── Maven version: MAVEN_HOME

└── Goals: clean

├── **Add Build Step** -> "Invoke top-level Maven targets"

└── Maven version: MAVEN_HOME

└── Goals: install




└── **Post-build Actions**:

├── **Add Post Build Action** -> "Archive the artifacts"

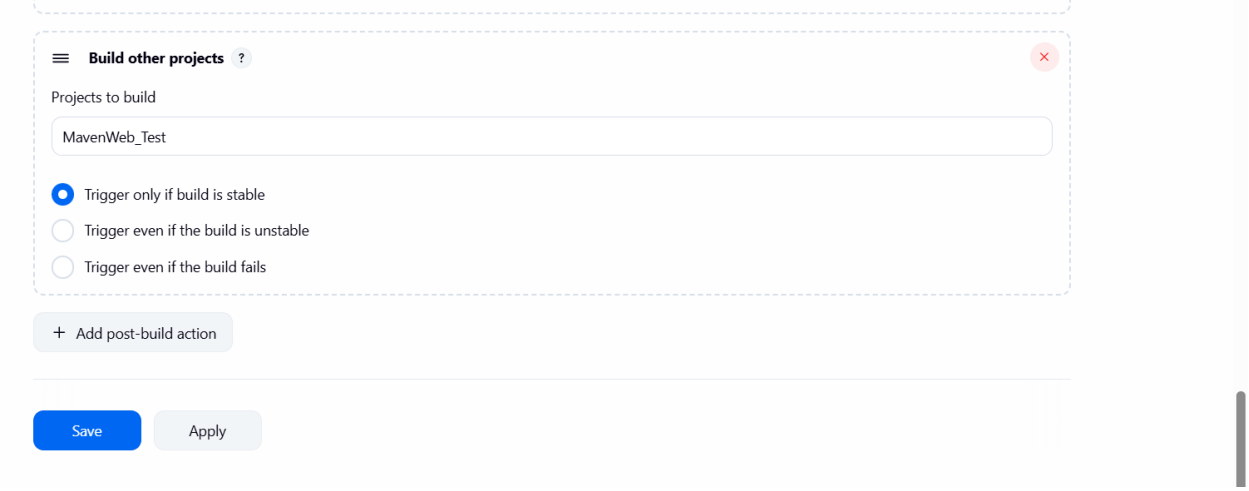
└── Files to archive: **/*


├── **Add Post Build Action** -> "Build other projects"

└── Projects to build: MavenWeb_Test

└── Trigger: Only if build is stable

└── Apply and Save

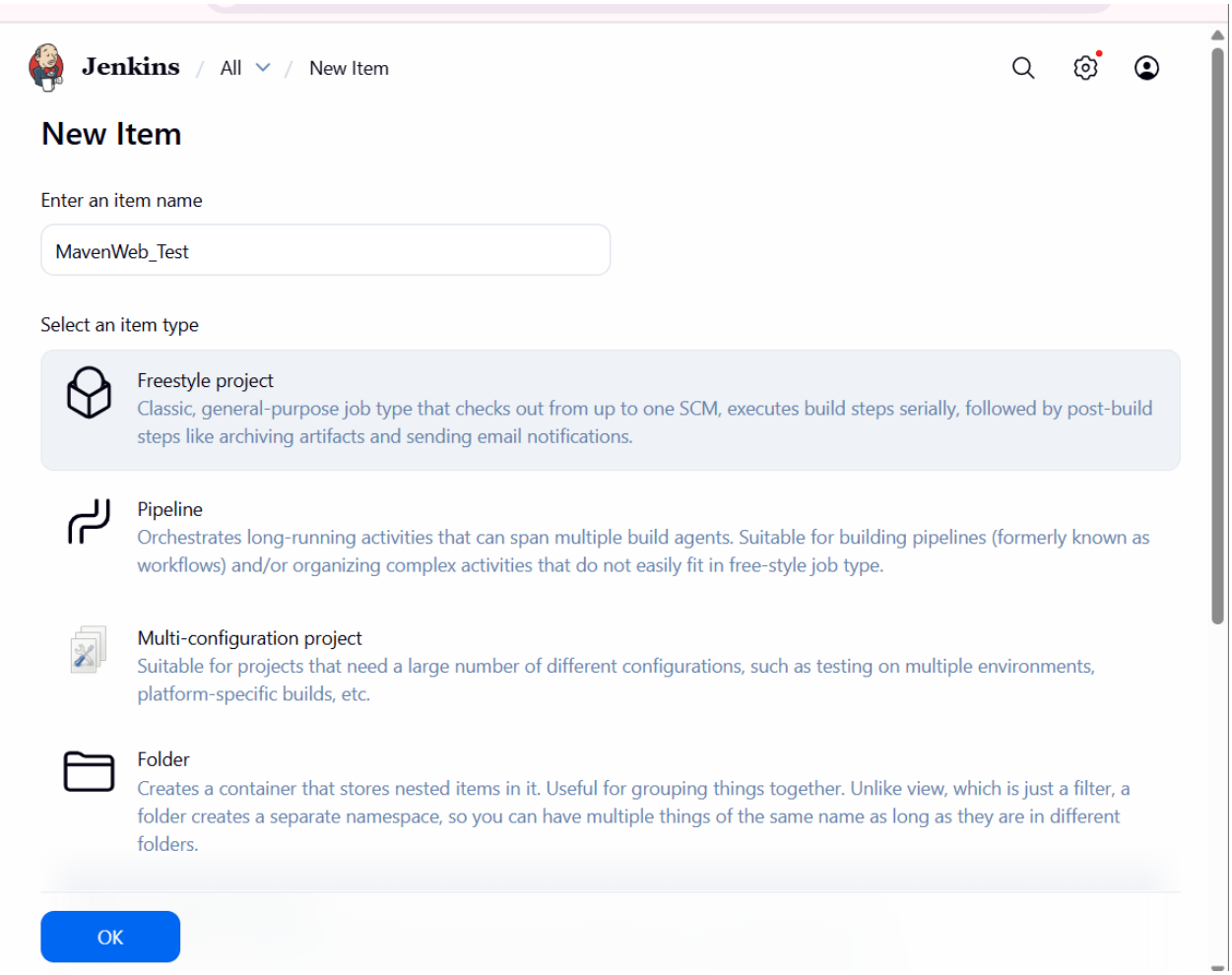



└── **Step 3**: Create Freestyle Project (e.g., MavenWeb_Test)

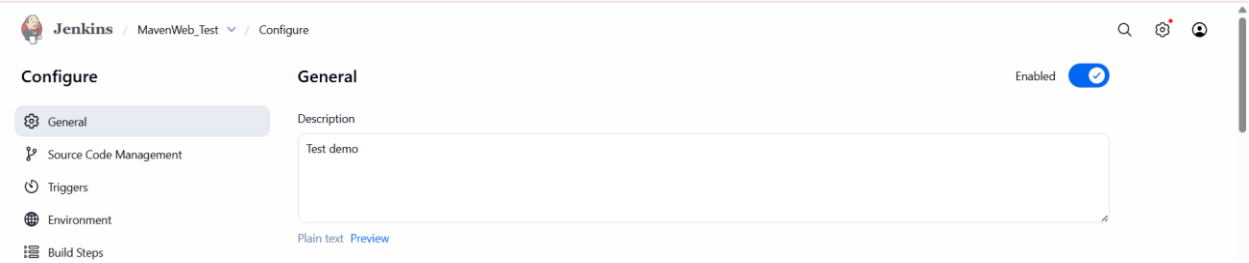
├── Enter project name (e.g., MavenWeb_Test)

├── Click "OK"

└── **Configure the project:**

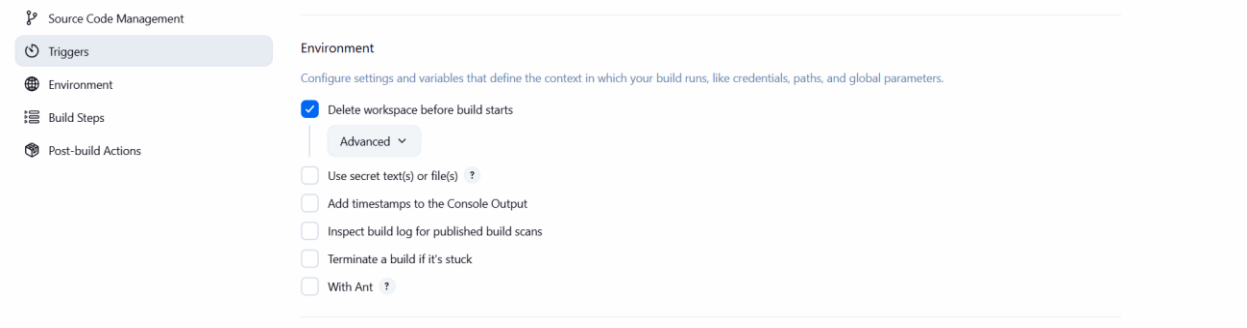



├── **Description:** "Test demo"




├── **Build Environment**:

└── Check: "Delete the workspace before build starts"

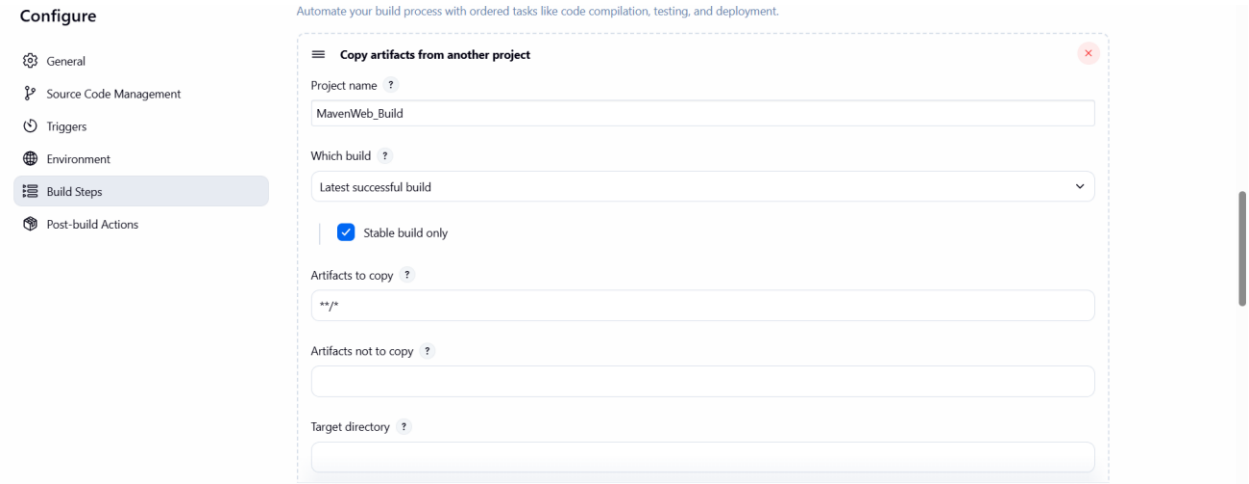



├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenWeb_Build

└── Build: Stable build only

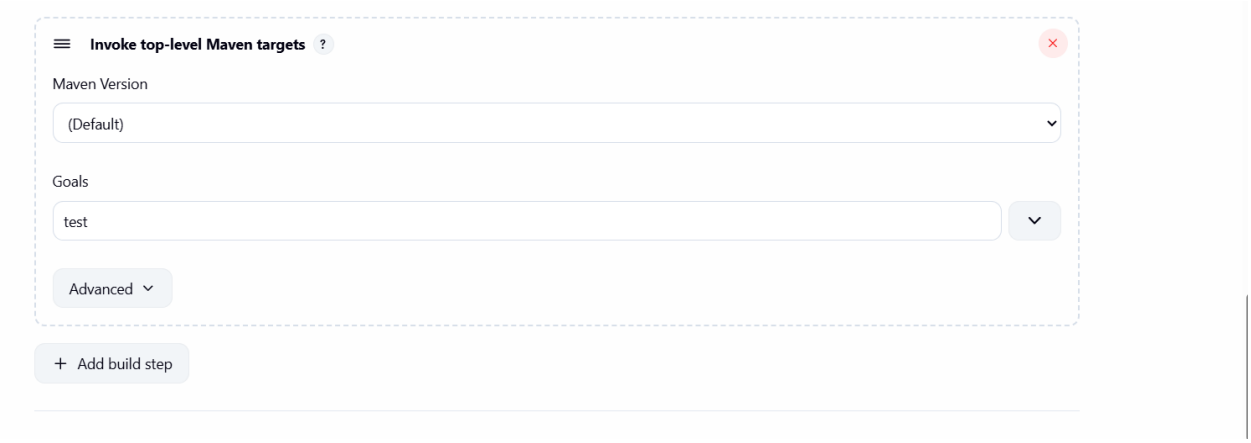
└── Artifacts to copy: **/*




├── **Add Build Step** -> "Invoke top-level Maven targets"

└── Maven version: MAVEN_HOME

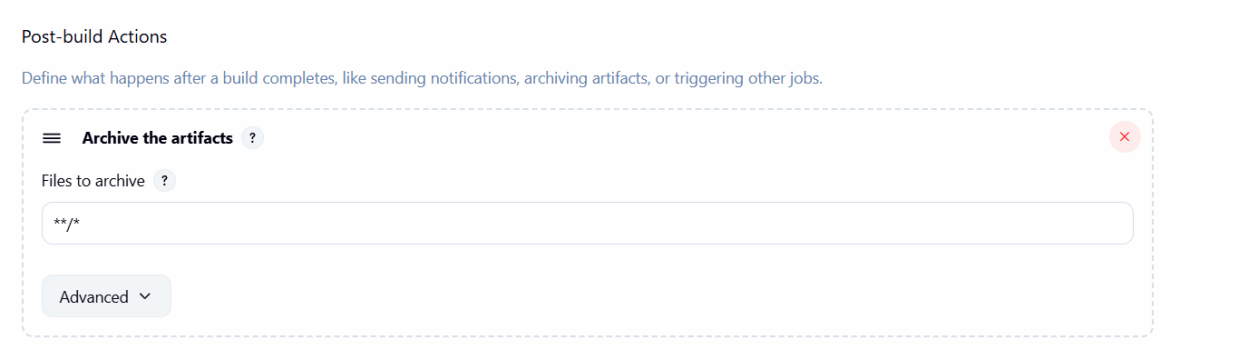
└── Goals: test



└── **Post-build Actions**:

├── **Add Post Build Action** -> "Archive the artifacts"

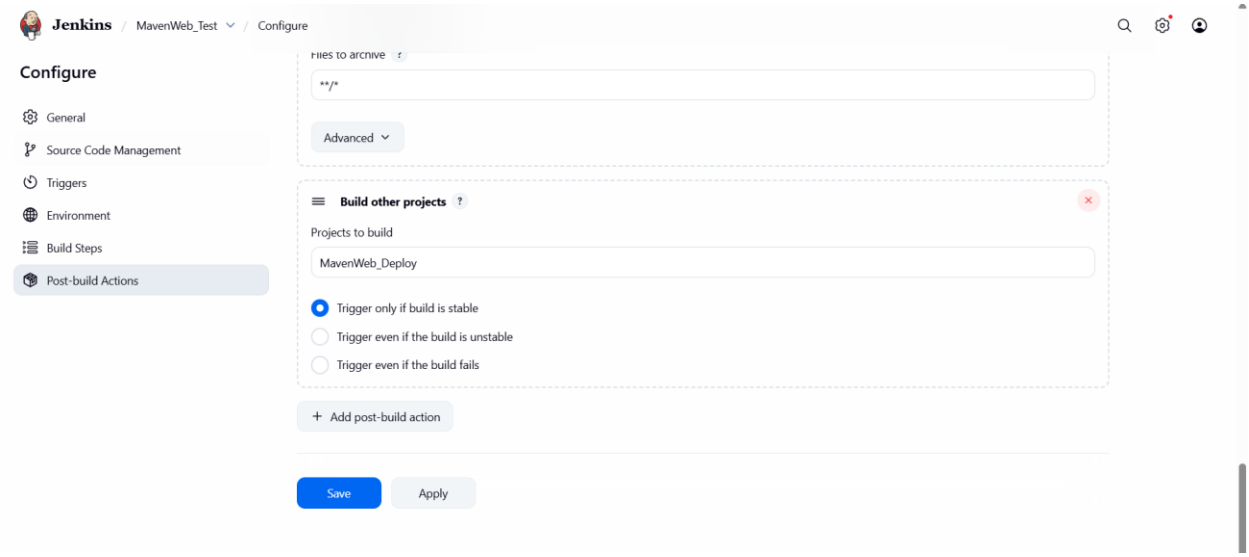
└── Files to archive: **/*




├── **Add Post Build Action** -> "Build other projects"

└── Projects to build: MavenWeb_Deploy

└── Apply and Save

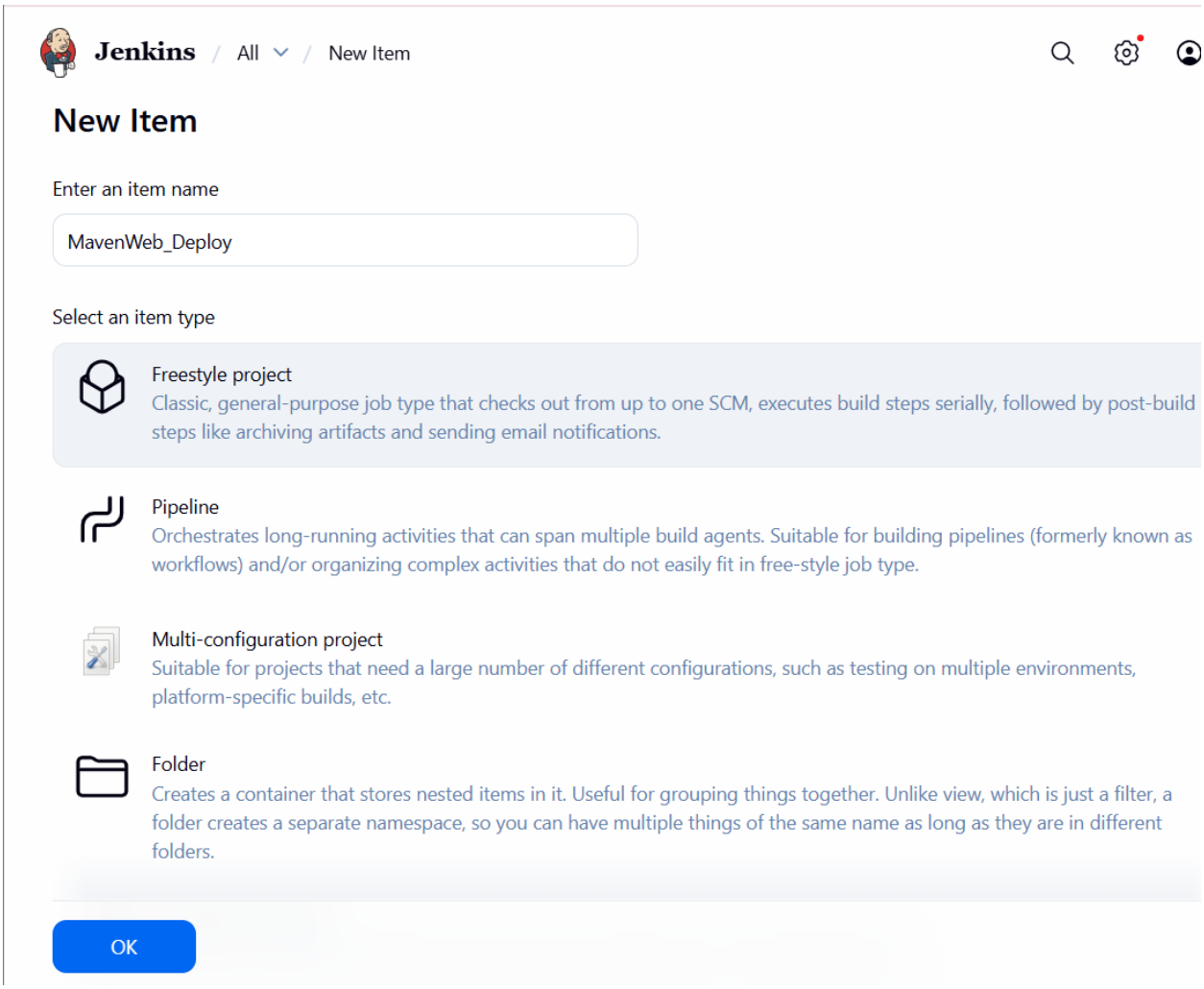



└── **Step 4**: Create Freestyle Project (e.g., MavenWeb_Deploy)

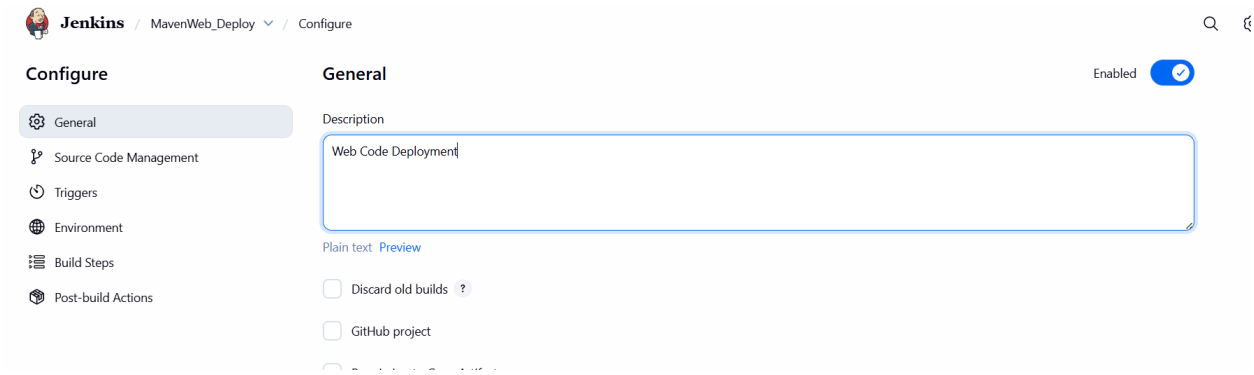
├── Enter project name (e.g., MavenWeb_Deploy)

├── Click "OK"

└── **Configure the projec**t:

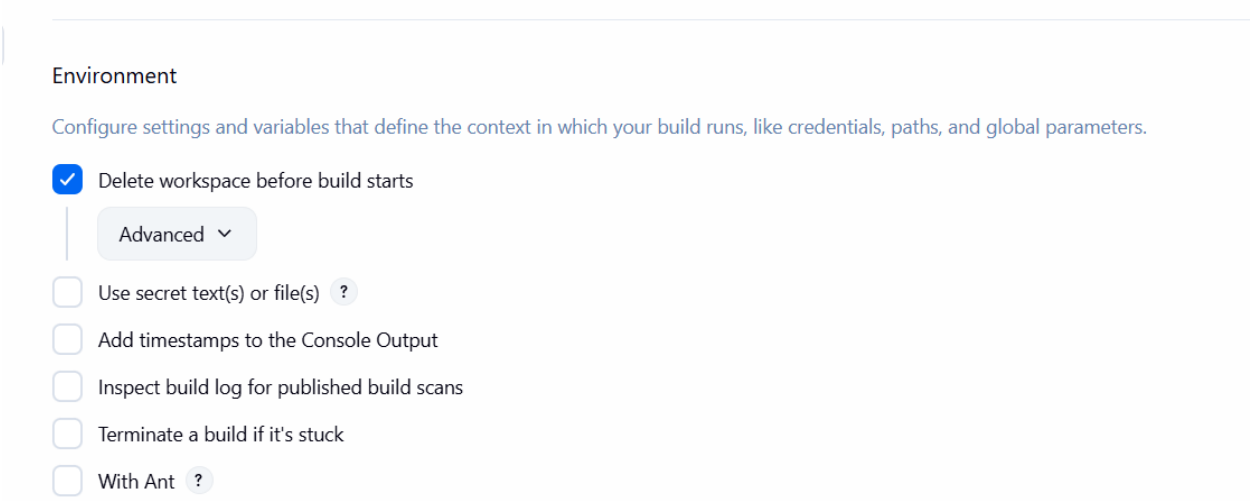



├── **Description**: "Web Code Deployment"




├── **Build Environment**:

└── Check: "Delete the workspace before build starts"

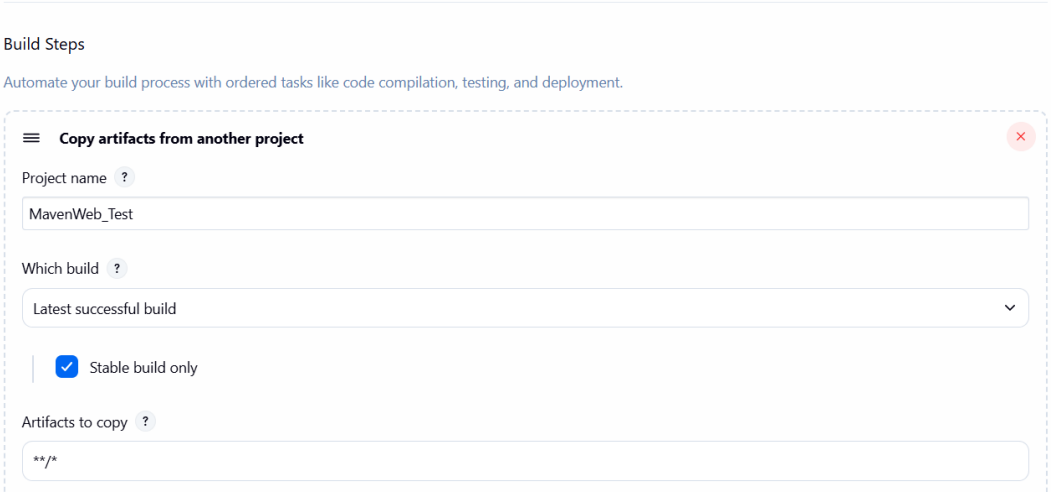



├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenWeb_Test

└── Build: Stable build only

└── Artifacts to copy: **/*




└── **Post-build Actions**:

├── **Add Post Build Action** -> "Deploy WAR/EAR to a container"

└── WAR/EAR File: **/*.war

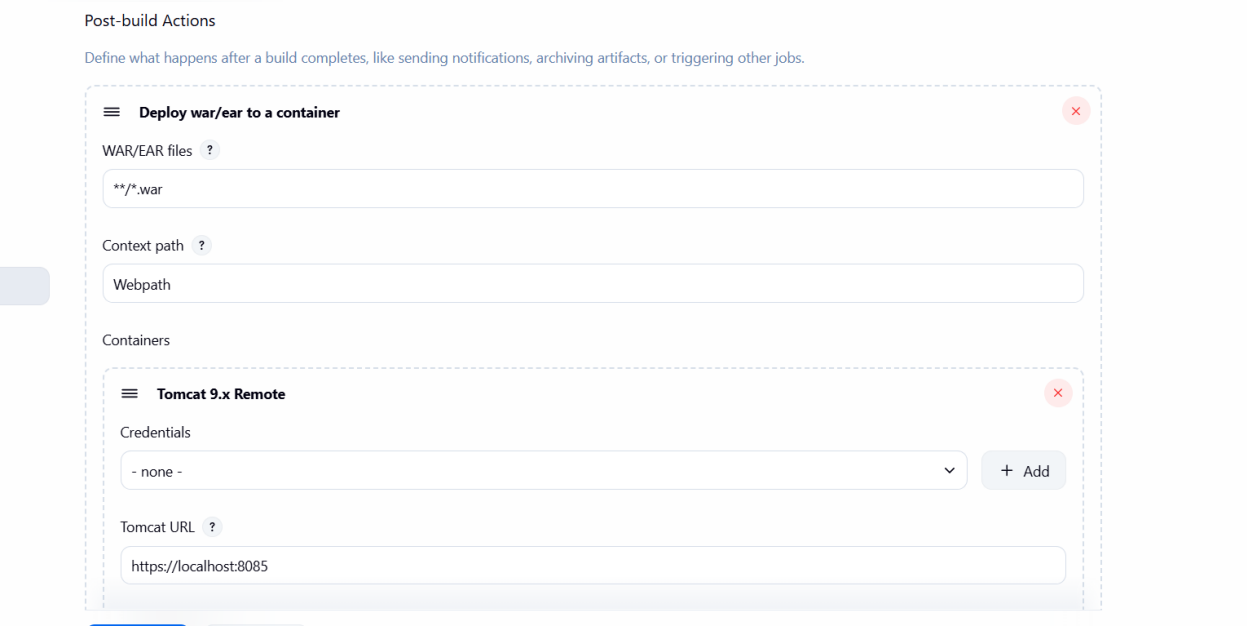
└── Context path: Webpath

└── Add container -> Tomcat 9.x remote

└── Credentials: Username: admin, Password: 1234

── Tomcat URL: https://localhost:8085/

└── Apply and Save

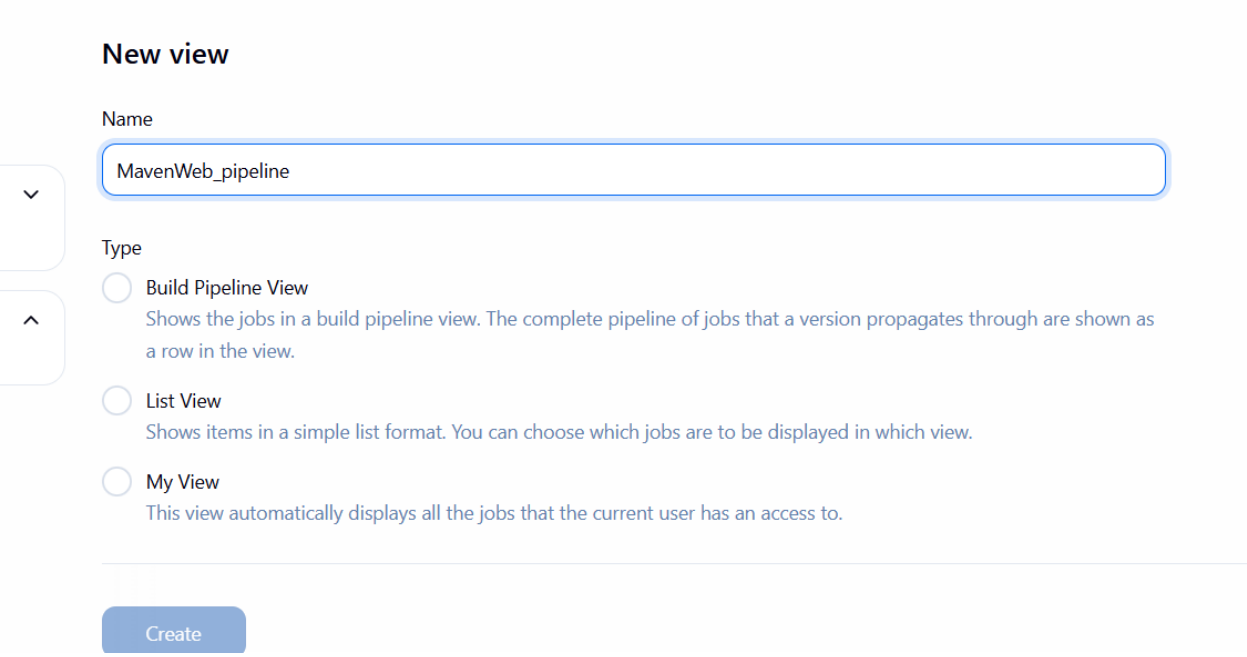



└── **Step 5**: Create Pipeline View for MavenWeb

├── Click "+" beside "All" on the dashboard

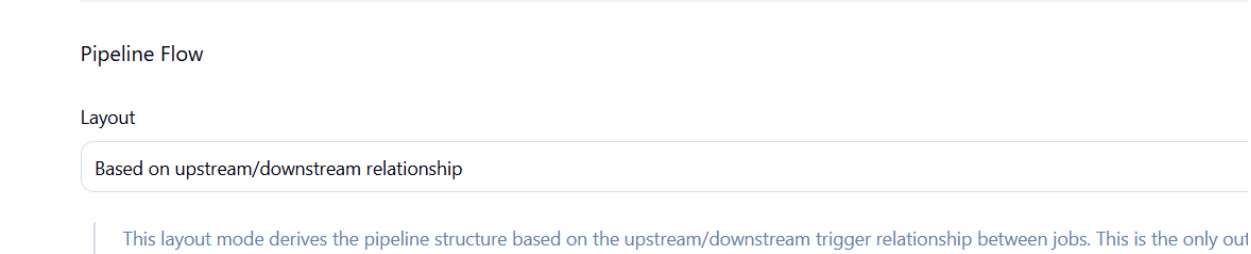
├── Enter name: MavenWeb_Pipeline

├── **Select "Build pipeline view"**

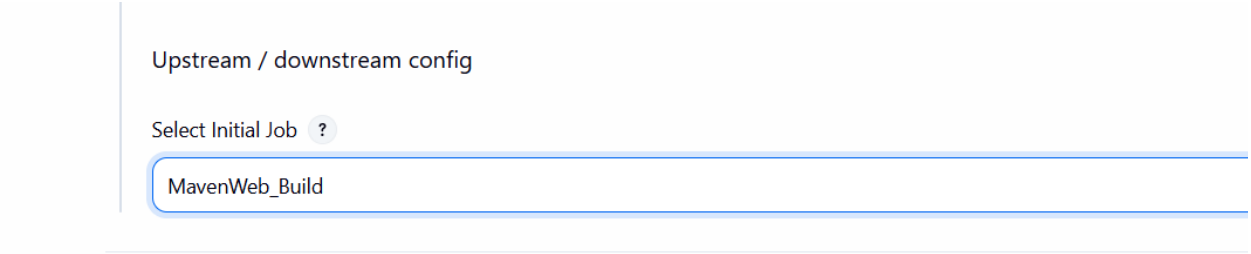


└── **Pipeline Flow**:

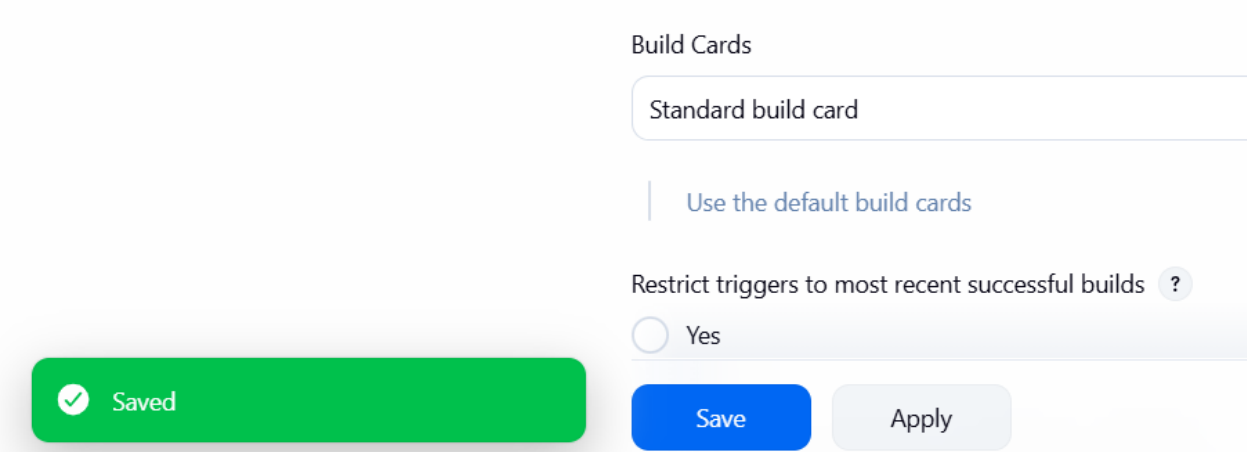
├── **Layout**: Based on upstream/downstream relationship




├── Initial job: MavenWeb_Build




└── Apply and Save



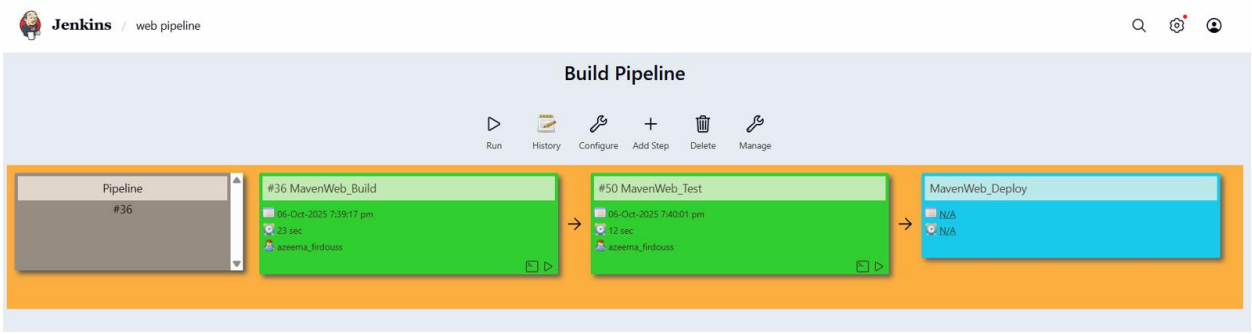

└── **Step 6**: Run the Pipeline and Check Output

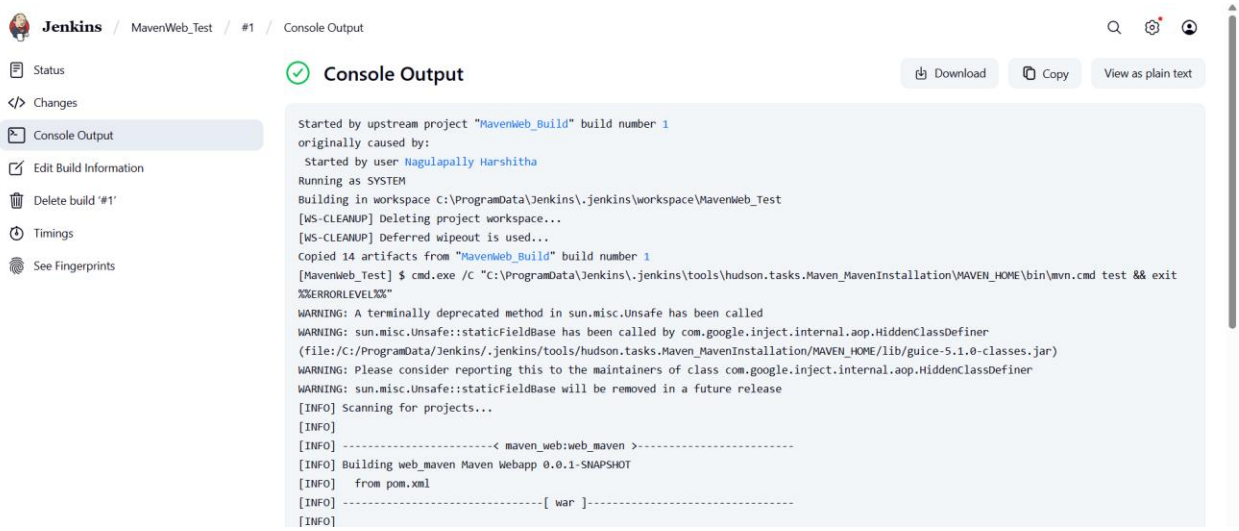
├── Click on the trigger **"RUN"** to run the pipeline

Note:

1. After Click on Run -> click on the small black box to open the console to check if the build is success

2. Now we see all the build has success if it appears in green color




```
[INFO] --- compiler:3.13.0:compile (default-compile) @ web_maven ---
[INFO] No sources to compile
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ web_maven ---
[INFO] skip non existing resourceDirectory C:\ProgramData\Jenkins\.jenkins\workspace\MavenWeb_Test\src\test\resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ web_maven ---
[INFO] No sources to compile
[INFO]
[INFO] --- surefire:3.3.0:test (default-test) @ web_maven ---
[INFO] No tests to run.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.569 s
[INFO] Finished at: 2025-09-23T20:36:10+05:30
[INFO] ------------------------------------------------------------------------
Archiving artifacts
Triggering a new build of MavenWeb_Deploy
Finished: SUCCESS
```

# III. Questions on Jenkins

1. **What is Jenkins primarily used for?**

   Jenkins is primarily used for continuous integration and continuous delivery (CI/CD) to automate the building, testing, and deployment of software projects.

2. **What is feature of Jenkins?**

   One key feature of Jenkins is its plugin-based architecture, which allows it to integrate with many tools and customize workflows.

3. **What is the default port on which Jenkins runs?**

   Jenkins runs on port 8080 by default.

4. **What can be integrated with Jenkins for version control?**

   Jenkins can be integrated with version control systems like Git, GitHub, GitLab, Bitbucket, and Subversion (SVN).

5. **What is the purpose of Jenkins plugins?**

   Jenkins plugins extend its functionality by allowing integrations with build tools, test frameworks, version control systems, cloud platforms, etc.

6. **Which type of Jenkins job is best suited for running one-off tasks or small scripts?**

   A Freestyle project is best for simple or one-off tasks.

7. **How can you manage sensitive information such as API keys in Jenkins?**

   Use the **"Credentials" plugin** to securely store and manage API keys, passwords, and other secrets.

8. **What does the "blue ocean" feature in Jenkins refer to?**

   Blue Ocean is a modern UI for Jenkins that provides a user-friendly interface for visualizing CI/CD pipelines.

9. **What does the "blue ocean" feature in Jenkins refer to?**

   Blue Ocean is a modern UI for Jenkins that provides a user-friendly interface for visualizing CI/CD pipelines.

**10. Which Jenkins component allows for distributed builds across multiple machines?**

The Jenkins agents (or nodes), managed by the Jenkins master (controller), allow for distributed builds.

**11. List at least five Jenkins plugins that you would consider important for a microservices-based application CI/CD pipeline. Briefly explain the purpose of each plugin.**

| Plugin | Purpose |
|---|---|
| Git Plugin | Integrates Git repositories for source control. |
| Pipeline Plugin | Enables defining complex CI/CD workflows as code (Jenkinsfile). |
| Docker Plugin | Supports building, deploying, and running Docker containers. |
| Kubernetes Plugin | Integrates Jenkins with Kubernetes for dynamic agent provisioning. |
| Slack Notification Plugin | Sends pipeline build notifications to Slack for team visibility. |

**12. Explain the steps you would take to install a plugin in Jenkins through the Jenkins UI. What considerations would you keep in mind regarding plugin compatibility and updates?**

Steps:

1. Go to Manage Jenkins > Plugin Manager.
2. Open the Available tab.
3. Search for the desired plugin.
4. Check the box and click Install without restart or Download now and install after restart.

Considerations:

- Check plugin compatibility with your Jenkins version.
- Review plugin dependencies before installation.
- Be aware of plugin update logs for breaking changes.
- Test in a staging environment if possible.

13. **Explain the steps you would take to install a plugin in Jenkins through the Jenkins UI. What considerations would you keep in mind regarding plugin compatibility and updates?**

**Steps:**

5. Go to Manage Jenkins > Plugin Manager.
6. Open the Available tab.
7. Search for the desired plugin.
8. Check the box and click Install without restart or Download now and install after restart.

**Considerations:**

- Check plugin compatibility with your Jenkins version.
- Review plugin dependencies before installation.
- Be aware of plugin update logs for breaking changes.
- Test in a staging environment if possible.

14. **After installing a plugin, explain how you would configure it within Jenkins. For example, if you installed the Git Plugin, what steps would you take to set it up for your pipeline?**

**Steps for Git Plugin:**

1. Go to Manage Jenkins > Global Tool Configuration.
2. Scroll to Git section.
3. Add a Git executable path or install Git automatically.
4. In your job/pipeline, use the SCM section to set the repository URL and credentials.

15. **Discuss common issues that might arise when using Jenkins plugins, such as dependency conflicts or version compatibility problems. How would you troubleshoot these issues?**

**Common issues:**

- Dependency conflicts between plugins.
- Incompatible plugin versions with Jenkins core.
- Performance degradation due to too many plugins.

- Plugins failing after Jenkins upgrade.
  Troubleshooting tips:
- Check Jenkins logs (Manage Jenkins > System Log).
- Use the Plugin Manager to identify failed or outdated plugins.
- Review the plugin's GitHub issues or changelog.
- Use safe restart after plugin updates.
- Restore from backup if plugin breaks the instance.


## Azeema firdous

## 23BD1A05A4

## CSE-F