**Week 8: Jenkins Automation**

1. **Steps for MavenJava Automation:**

Maven Java Automation Steps:

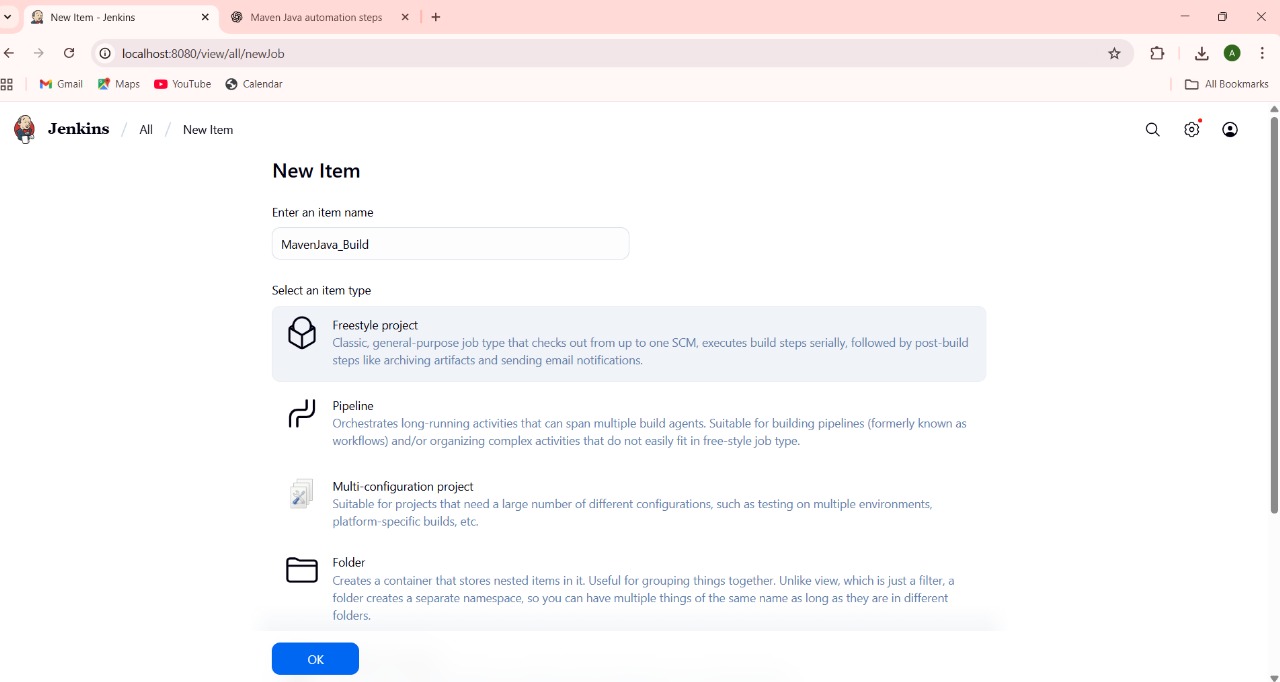
**Step 1: Open Jenkins (localhost:8080)**

├── 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"



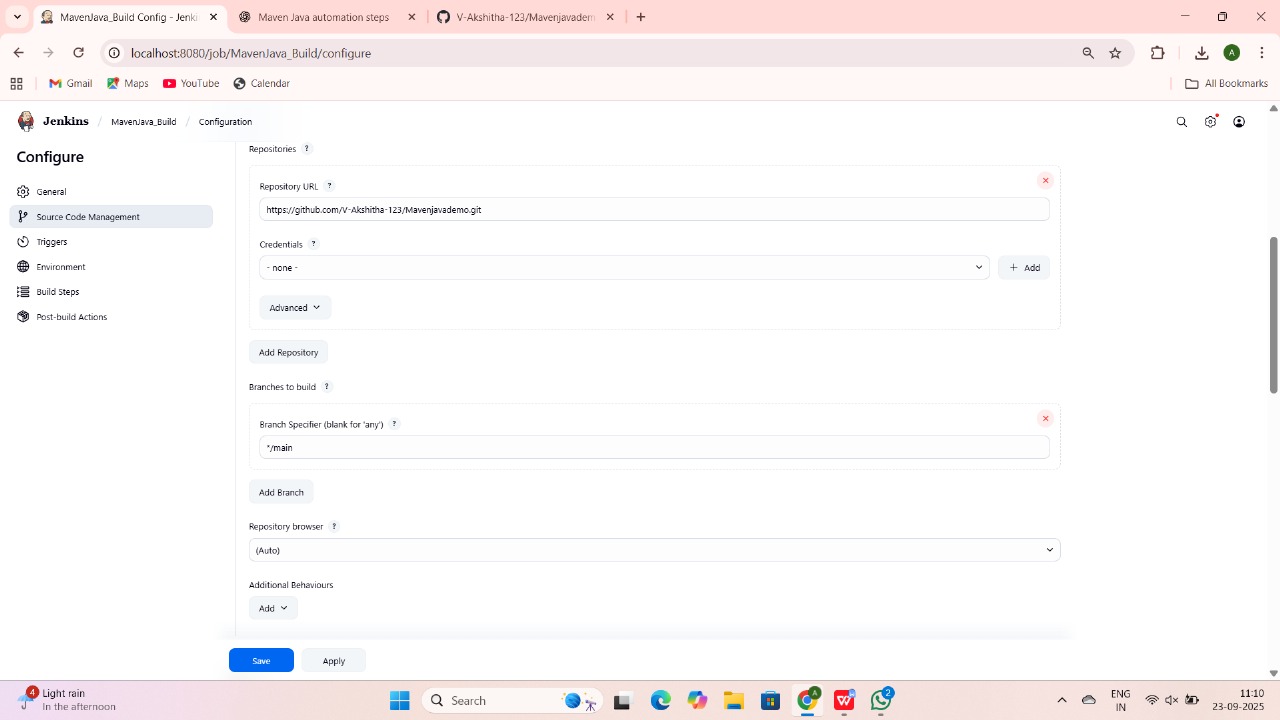
**└── Configure the project:**

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

├── **Source Code Management**:

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

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



└── **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"

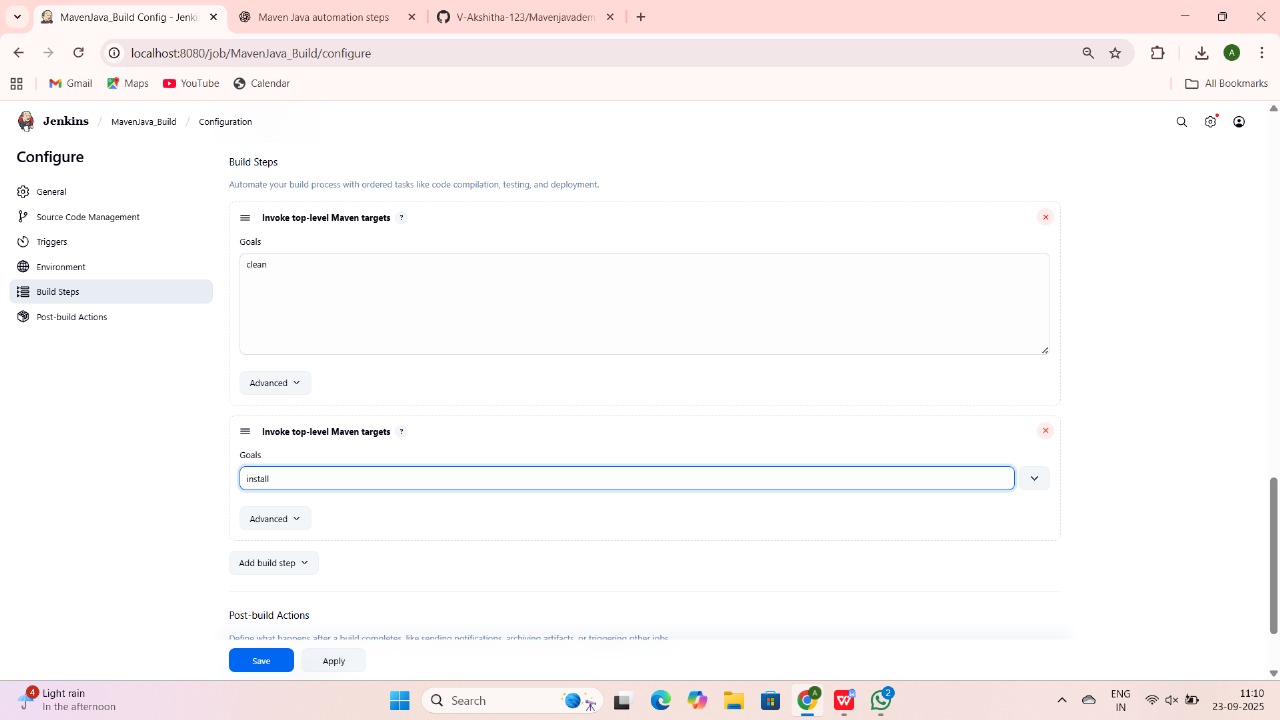
└── Files to archive: \*\*/\*

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

└── Projects to build: MavenJava\_Test

└── Trigger: Only if build is stable

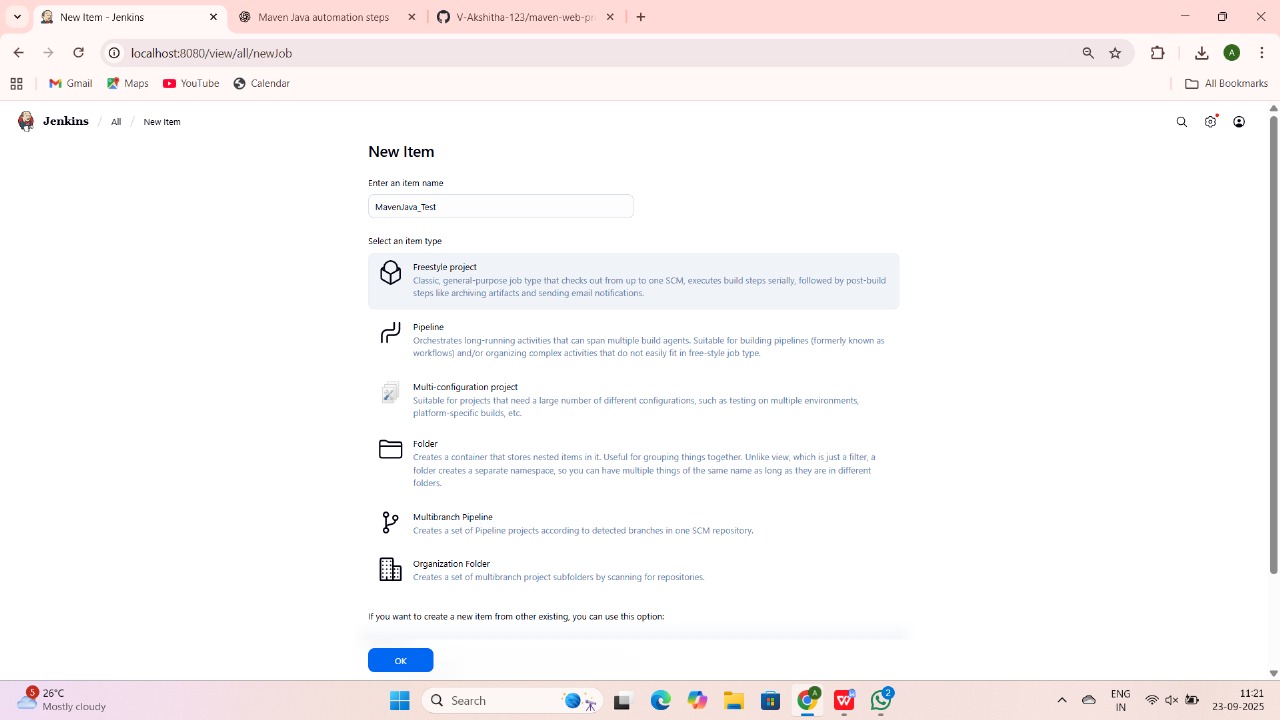
└── Apply and Save



└── **Step 3: Create Freestyle Project (e.g., MavenJava\_Test)**

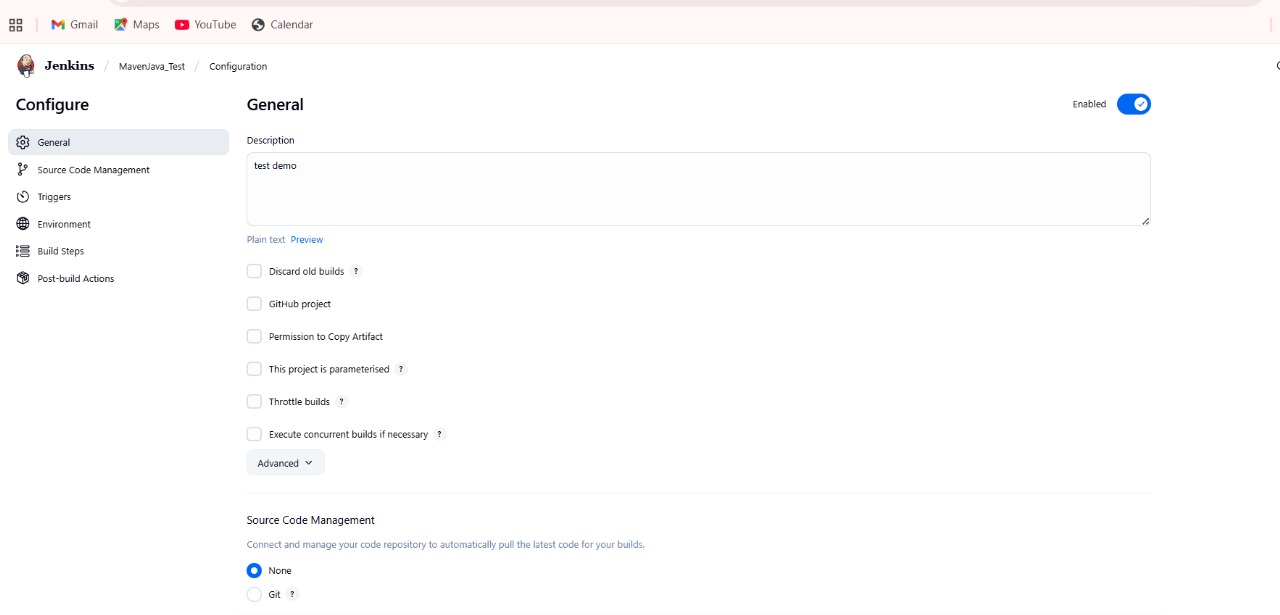
├── Enter project name (e.g., MavenJava\_Test)

├── Click "OK"



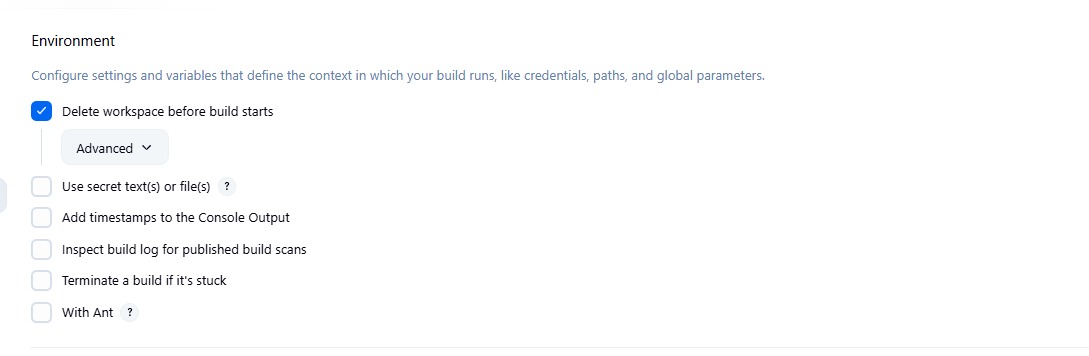
**└── Configure the project**:

├── **Description**: "Test demo"



**├── Build Environment**:

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

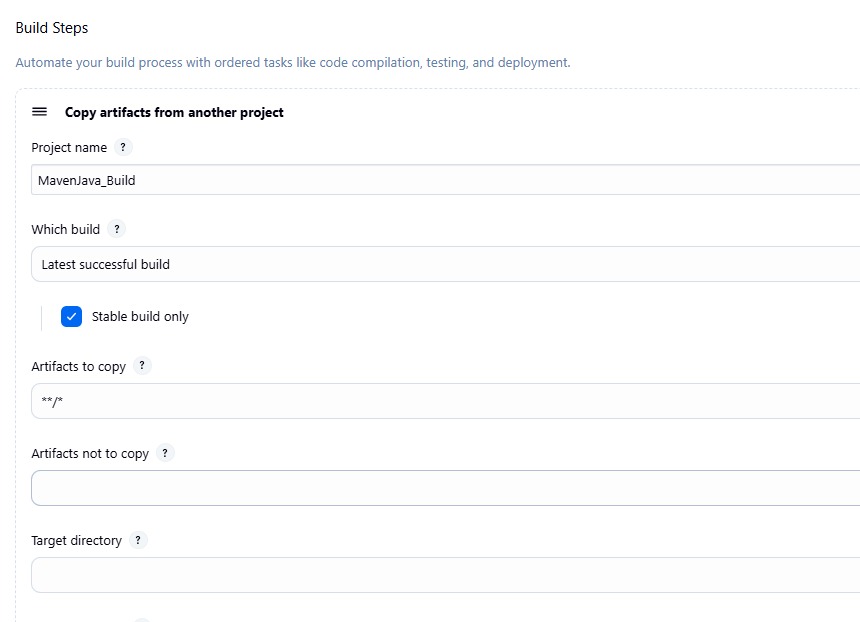


├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenJava\_Build

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

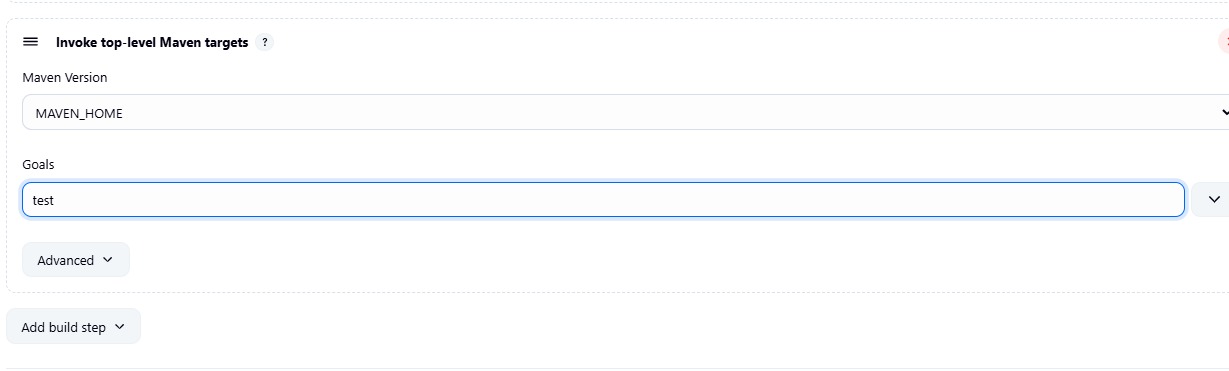
└── Artifacts to copy: \*\*/\*



├── **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

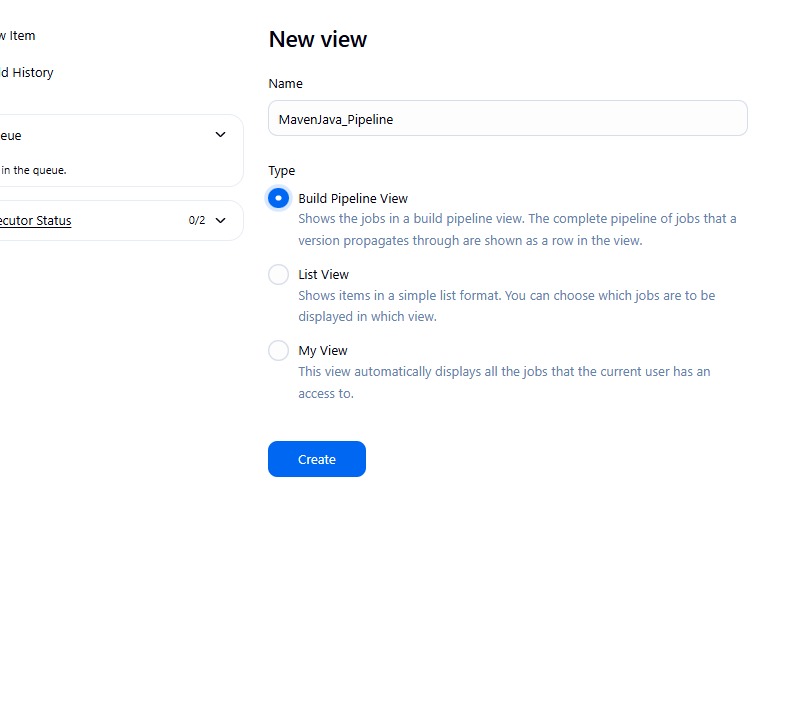
└── **Step 4**: Create Pipeline View for Maven Java project

├── 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

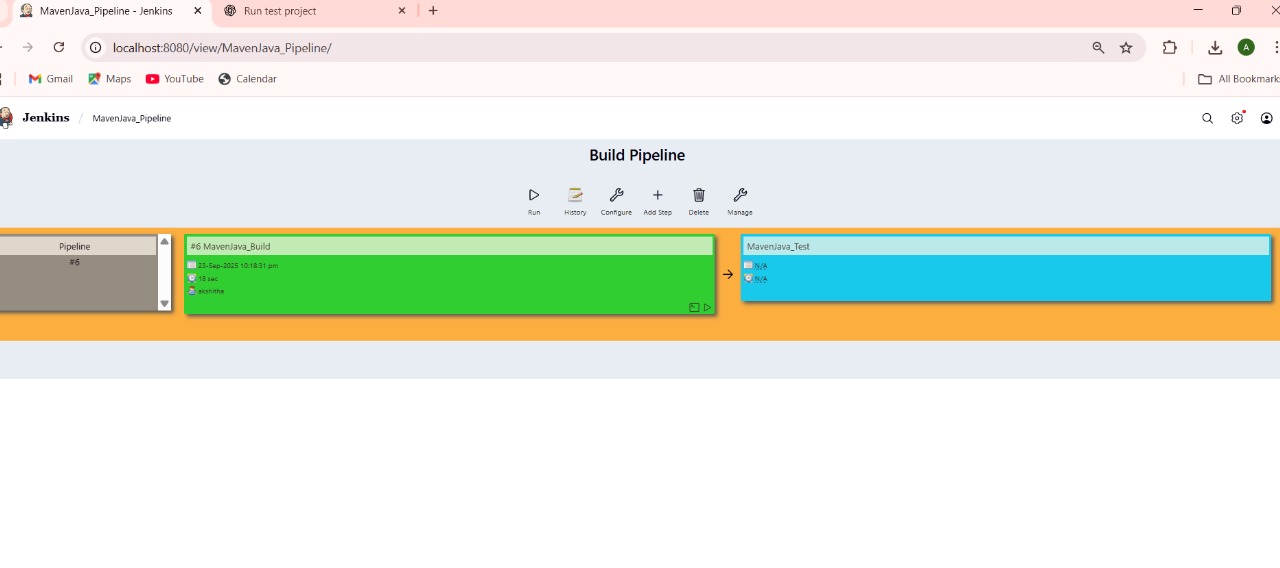
├── Initial job: MavenJava\_Build

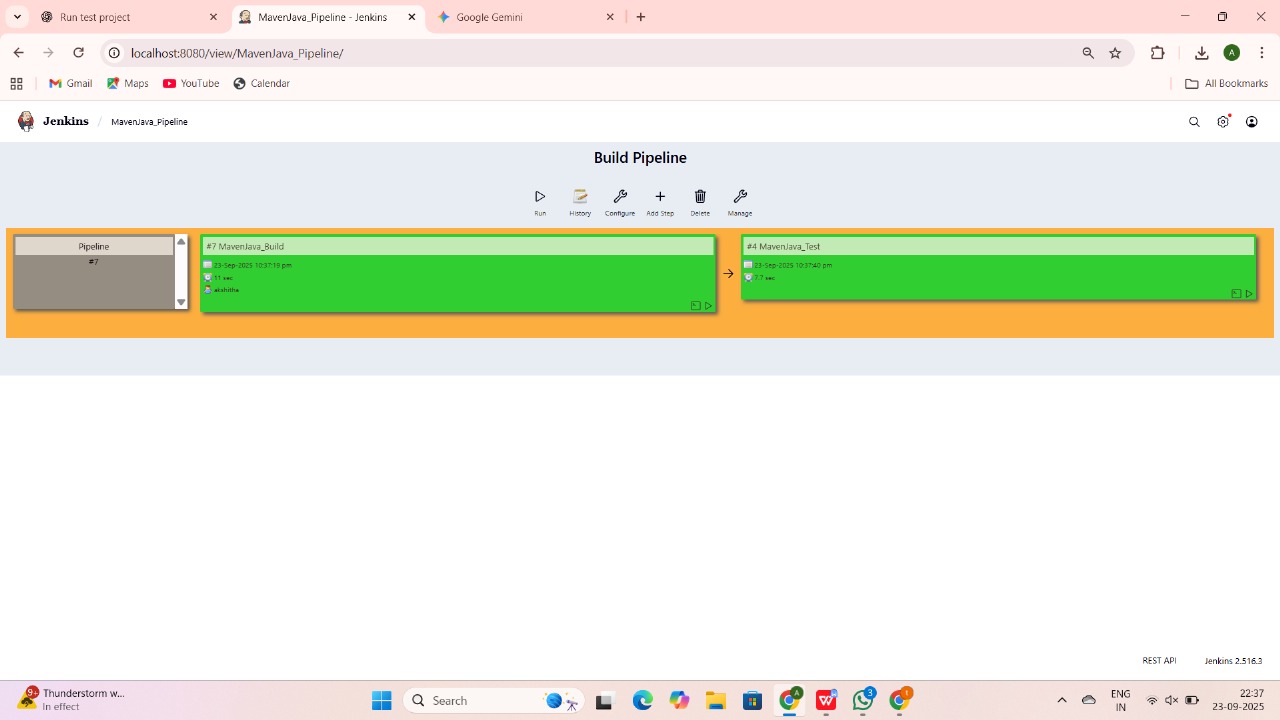
└── Apply and Save OK

└── **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



2. 

**II. Maven Web Automation Steps:**

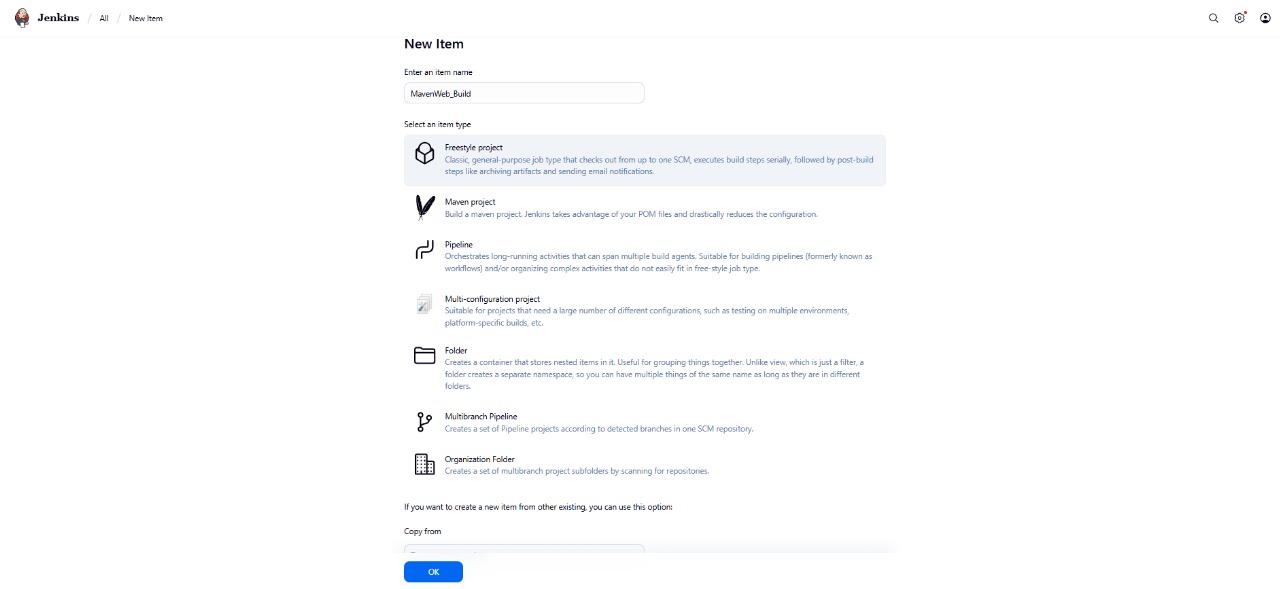
└── **Step 1:** Open Jenkins (localhost:8080)

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

└── **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"

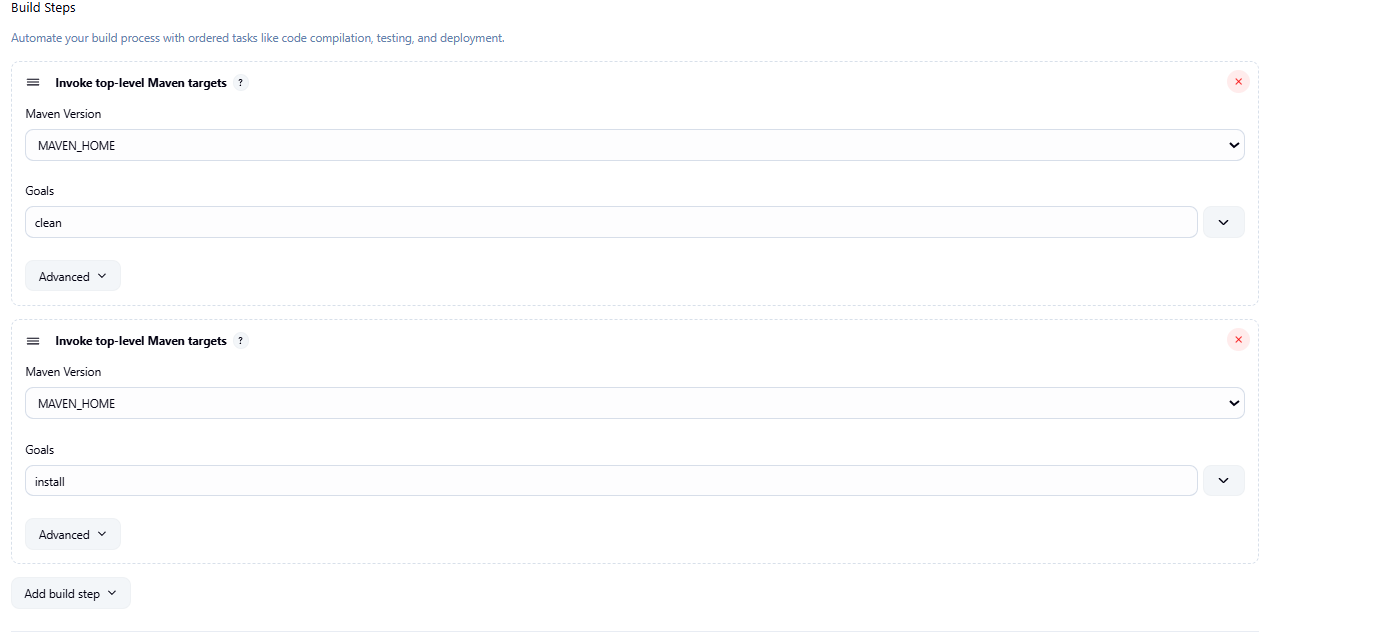
└── Maven version: MAVEN\_HOME

└── Goals: clean

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

└── Maven version: MAVEN\_HOME

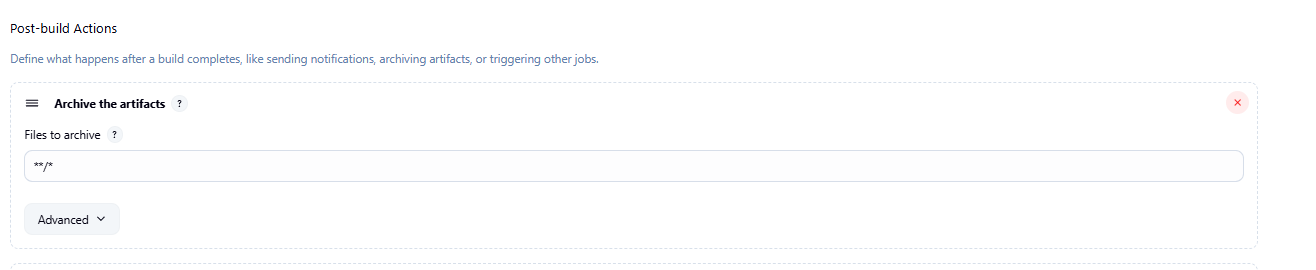
└── Goals: install



└── **Post-build Actions**:

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

└── Files to archive: \*\*/\*

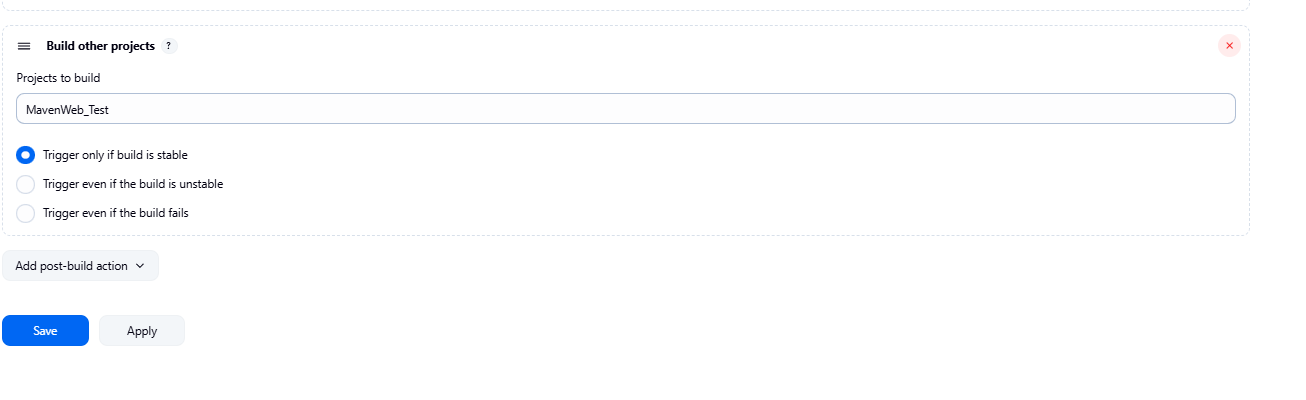


├── **Add Post Build Action** -> "Build other projects"

└── Projects to build: MavenWeb\_Test

└── Trigger: Only if build is stable

└── Apply and Save



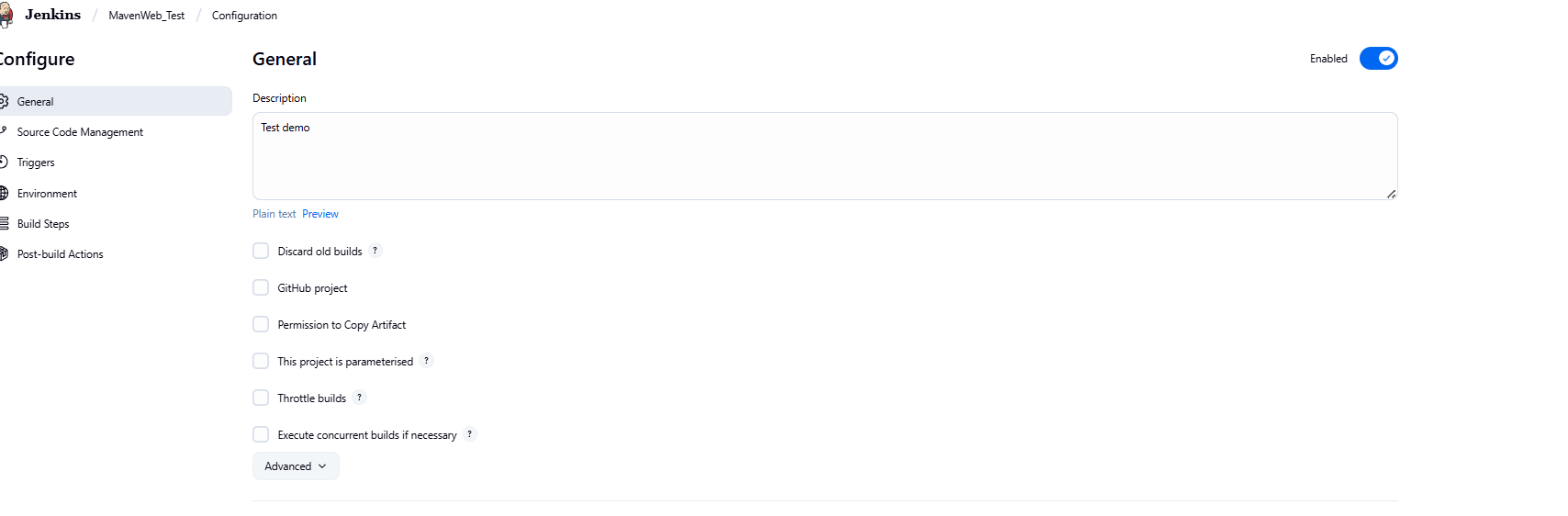
**└── Step 3**: Create Freestyle Project (e.g., MavenWeb\_Test)

├── Enter project name (e.g., MavenWeb\_Test)

├── Click "OK"

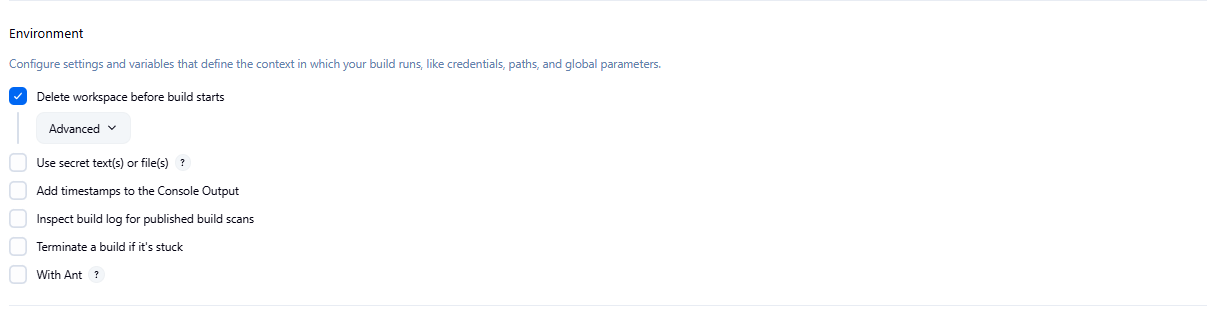
└── **Configure the project:**

├── **Description:** "Test demo"



├── **Build Environment**:

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

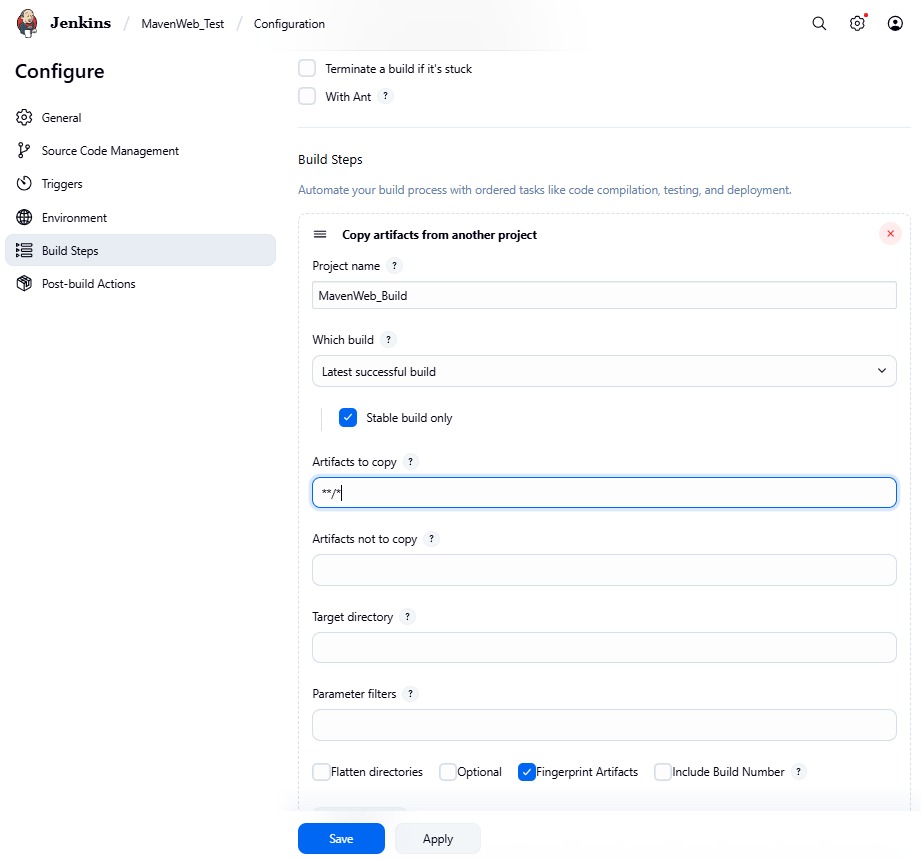


├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenWeb\_Build

└── Build: Stable build only

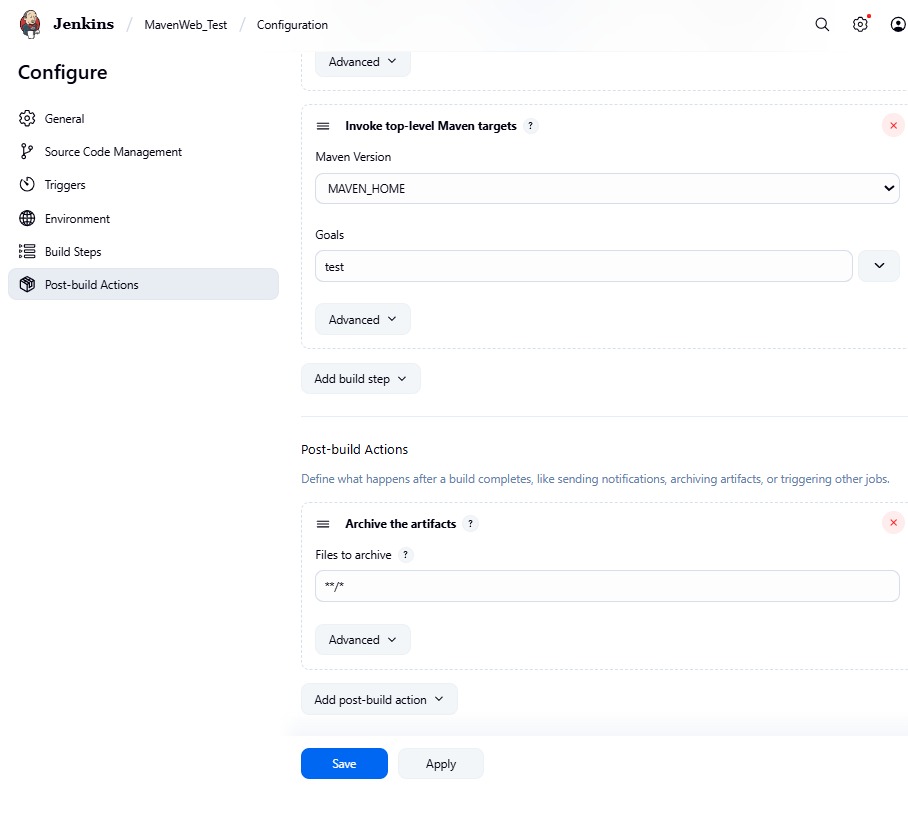
└── Artifacts to copy: \*\*/\*



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

└── Maven version: MAVEN\_HOME

└── Goals: test



└── **Post-build Actions**:

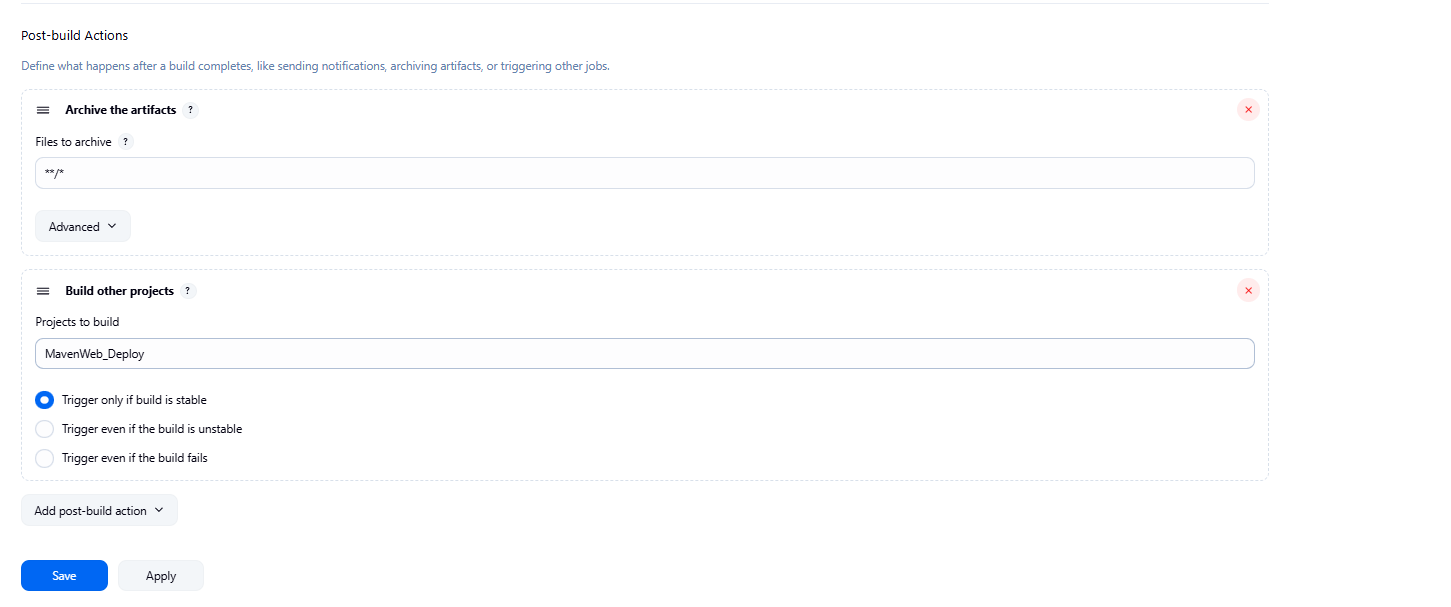
├── **Add Post Build Action** -> "Archive the artifacts"

└── Files to archive: \*\*/\*

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

└── Projects to build: MavenWeb\_Deploy

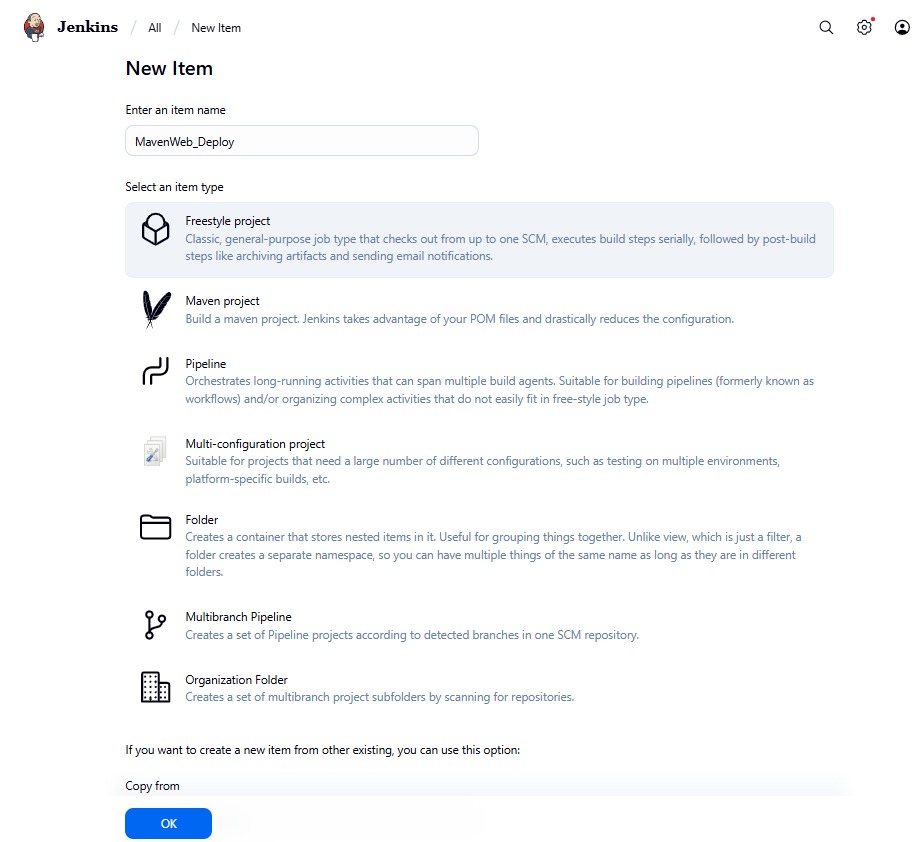
└── Apply and Save



└── **Step 4**: Create Freestyle Project (e.g., MavenWeb\_Deploy)

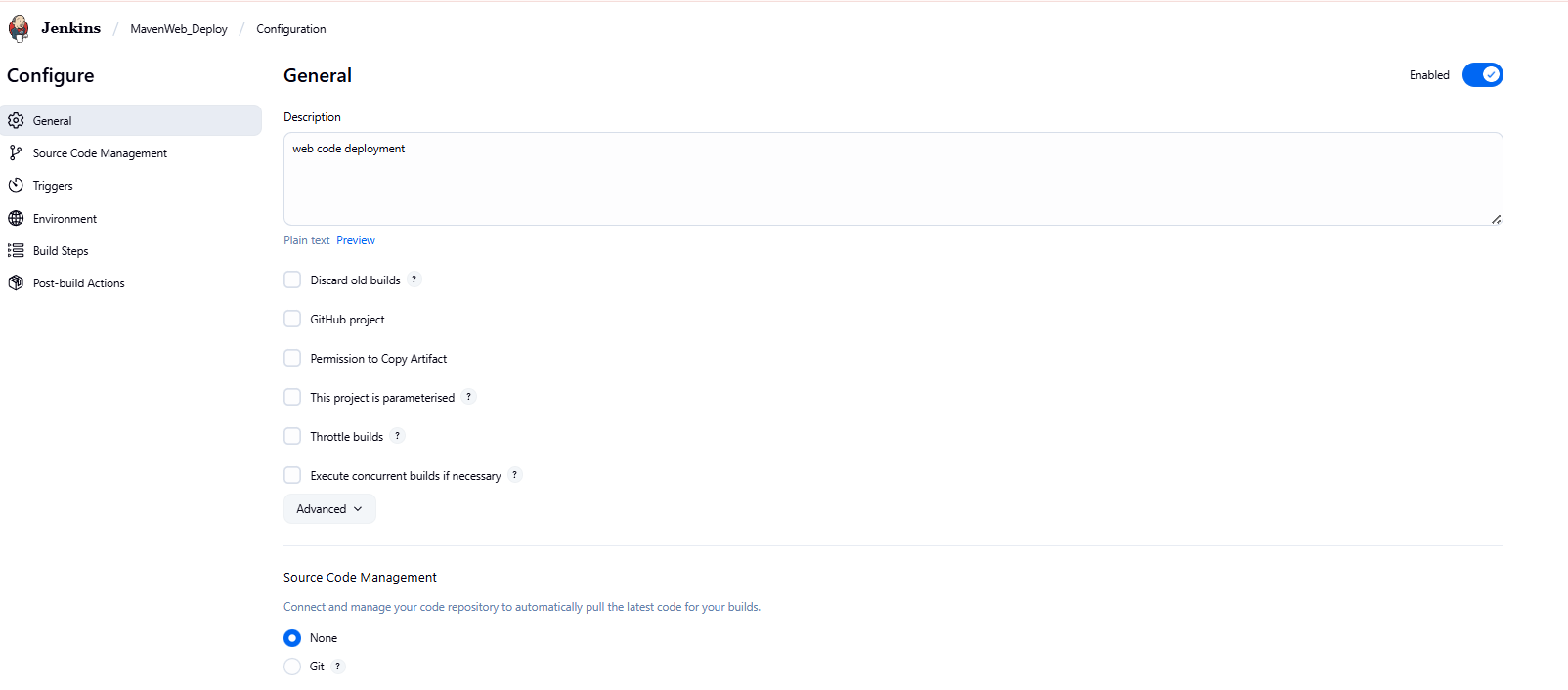
├── Enter project name (e.g., MavenWeb\_Deploy)

├── Click "OK"



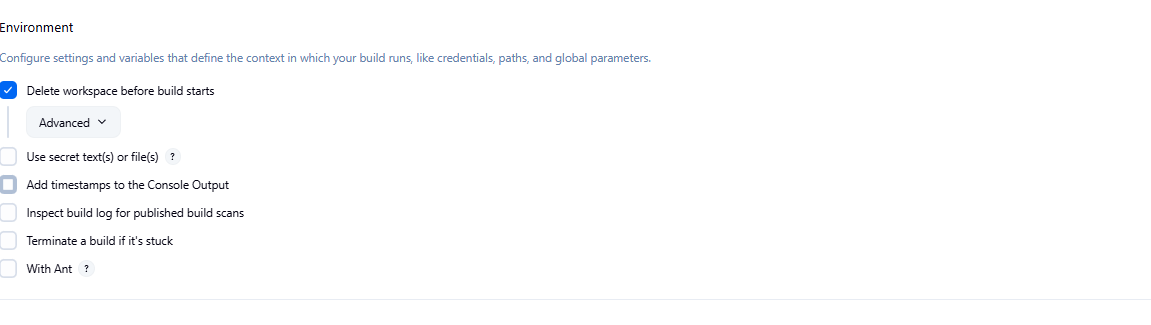
└── **Configure the projec**t:

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



├── **Build Environment**:

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

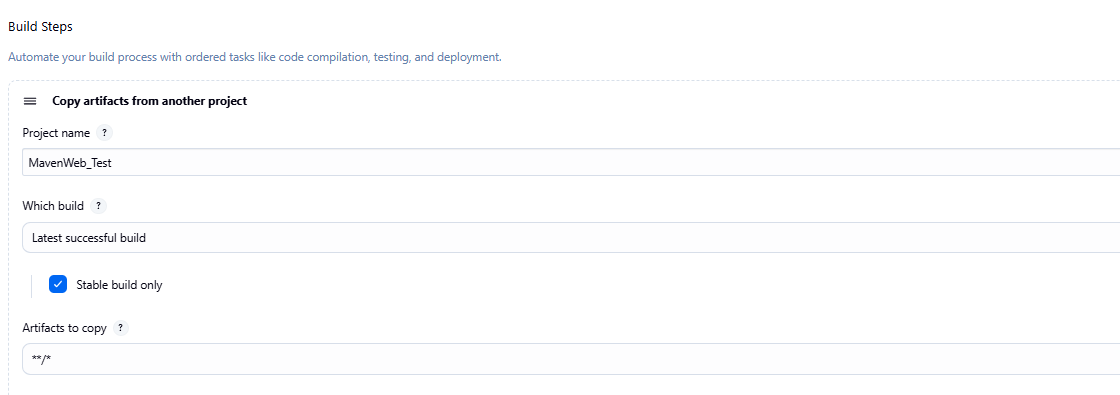


├── **Add Build Step** -> "Copy artifacts from another project"

└── Project name: MavenWeb\_Test

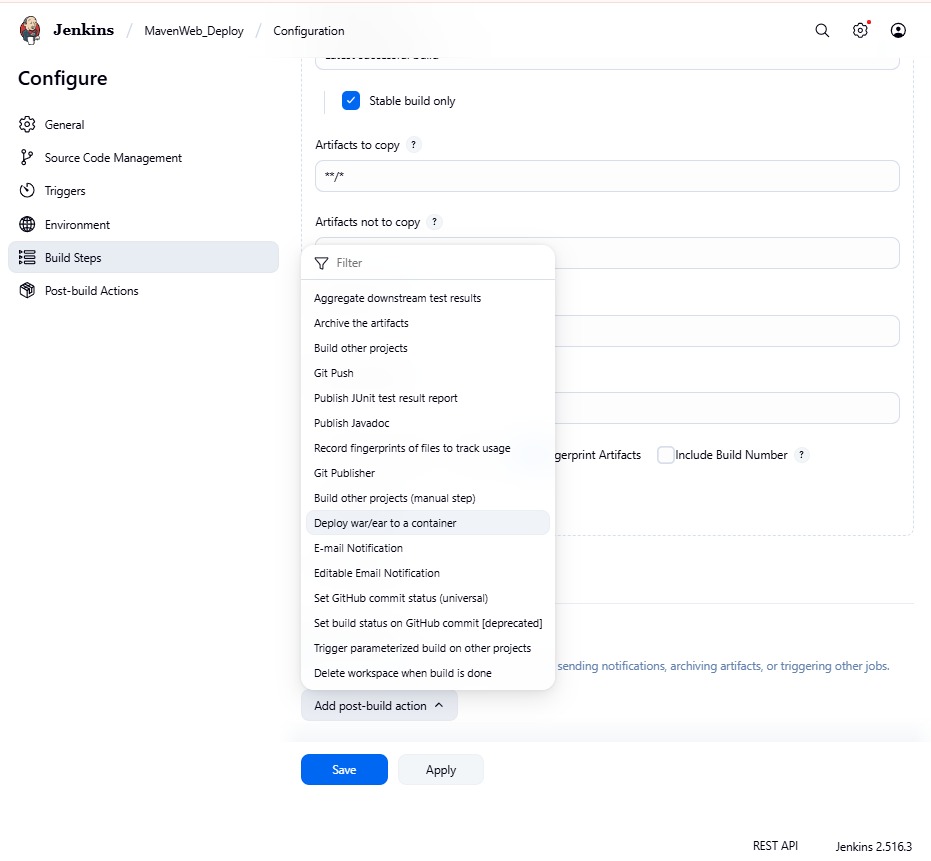
└── Build: Stable build only

└── Artifacts to copy: \*\*/\*



└── **Post-build Actions**:

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

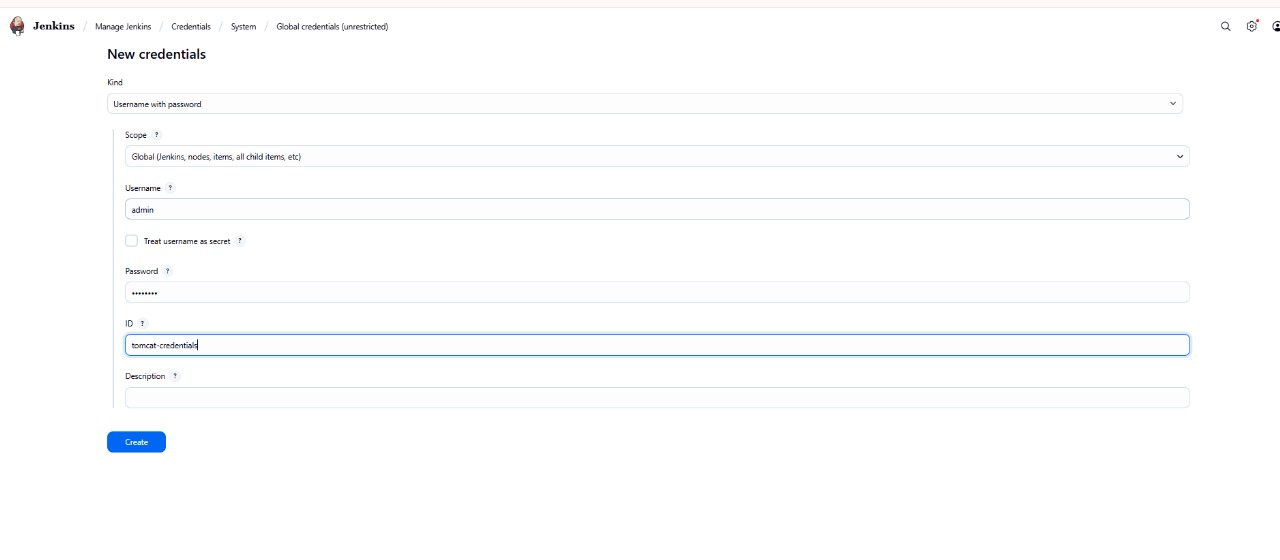


└── WAR/EAR File: \*\*/\*.war

└── Context path: Webpath

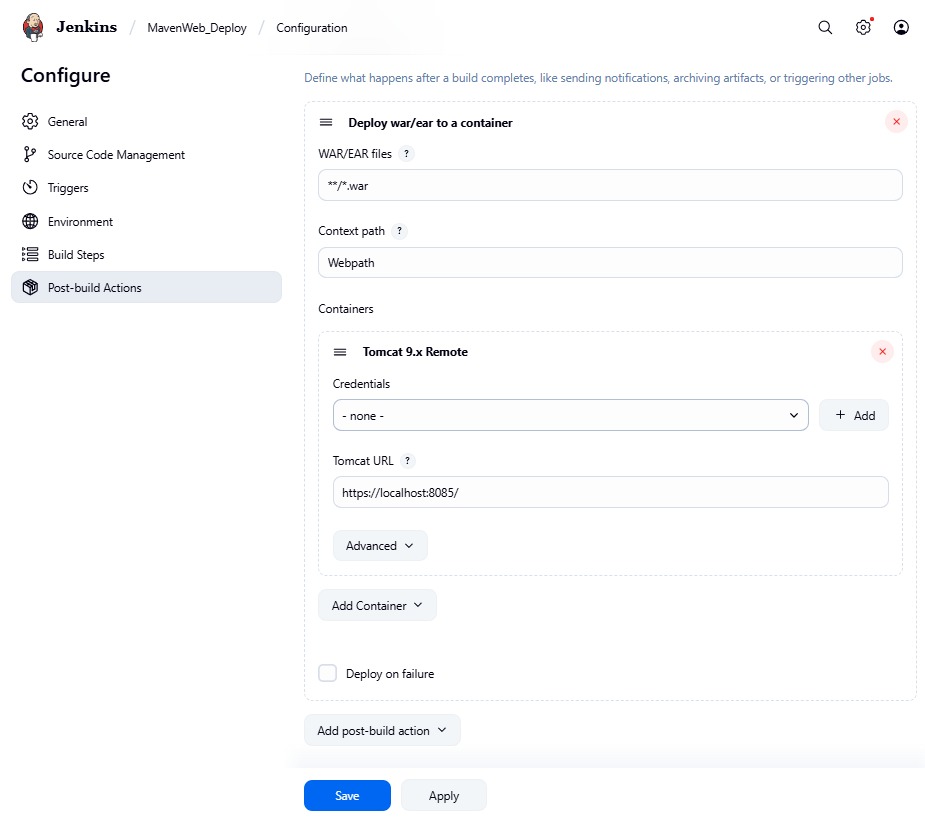
└── Add container -> Tomcat 9.x remote

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



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

└── Apply and Save

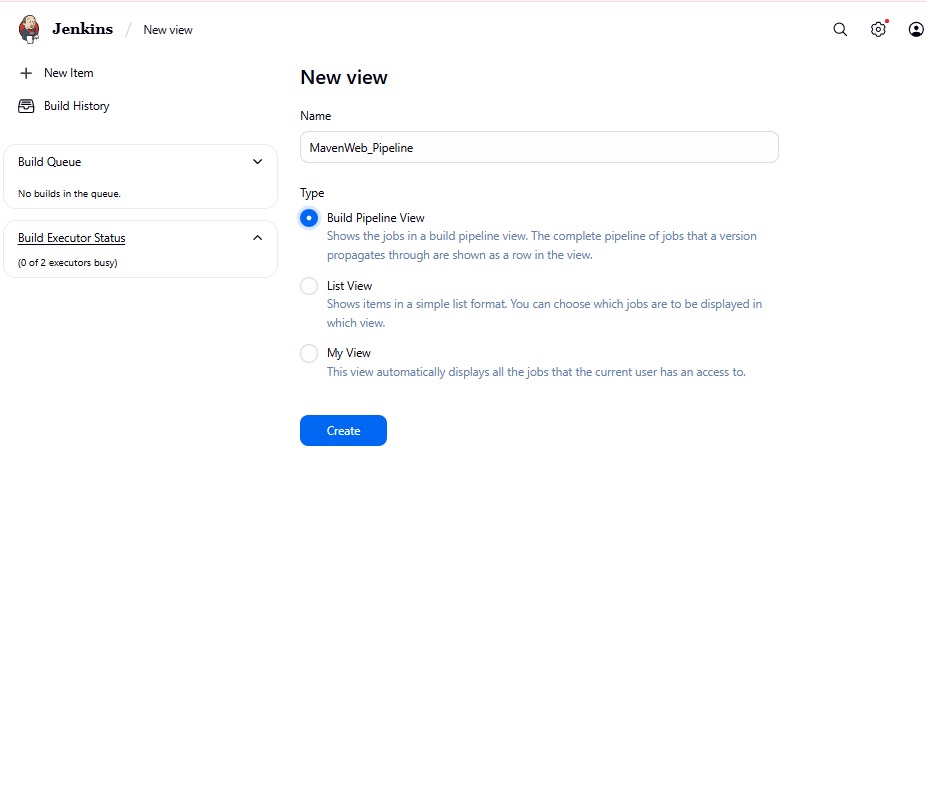


└── **Step 5**: Create Pipeline View for MavenWeb

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

├── Enter name: MavenWeb\_Pipeline

├── **Select "Build pipeline view"**

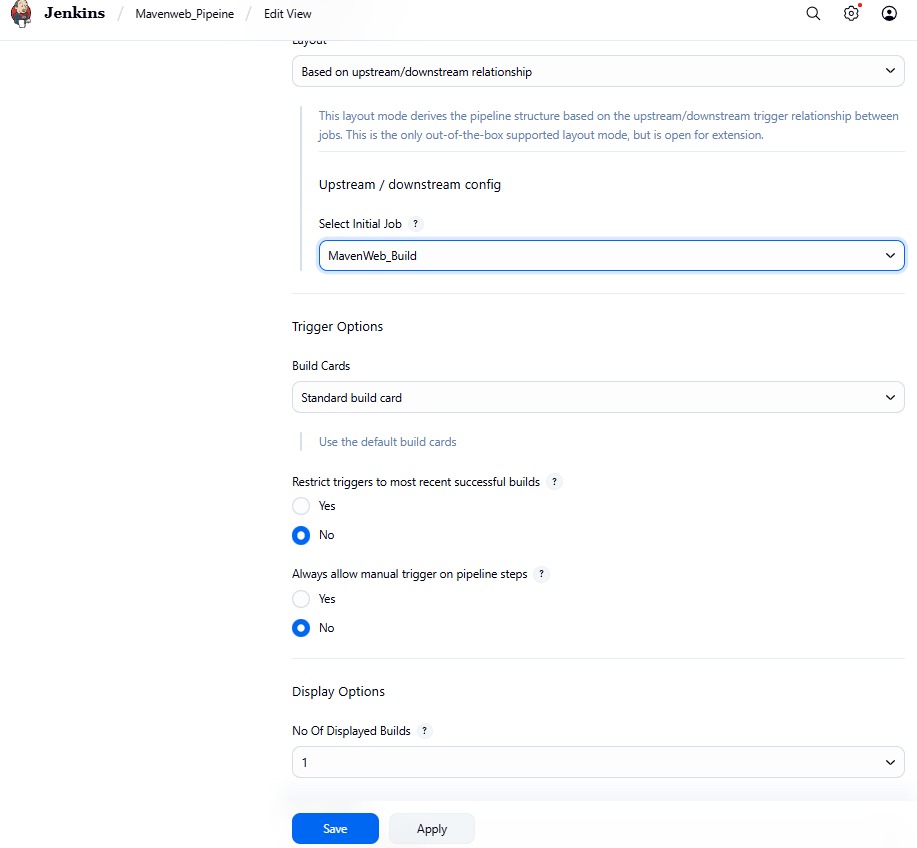
****

**└── Pipeline Flow**:

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

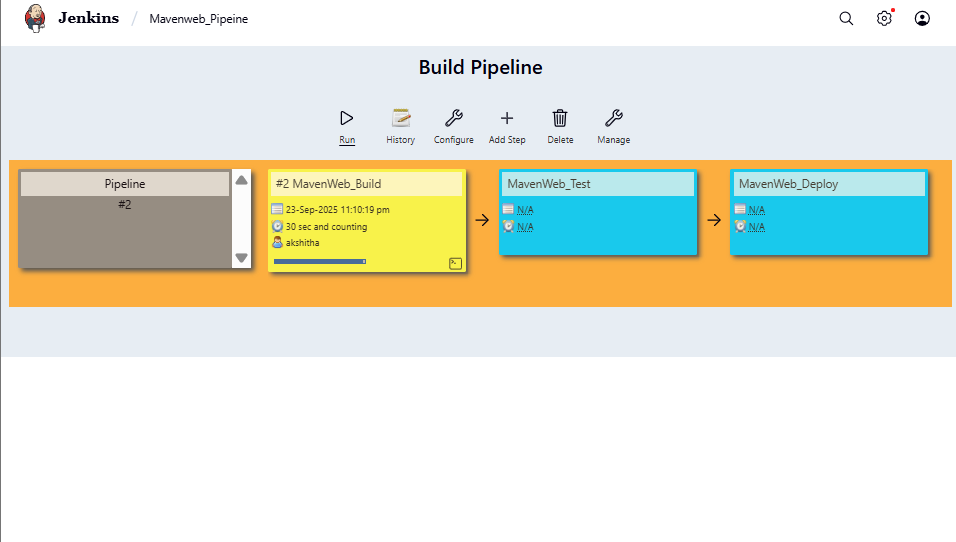
├── Initial job: MavenWeb\_Build

└── Apply and Save

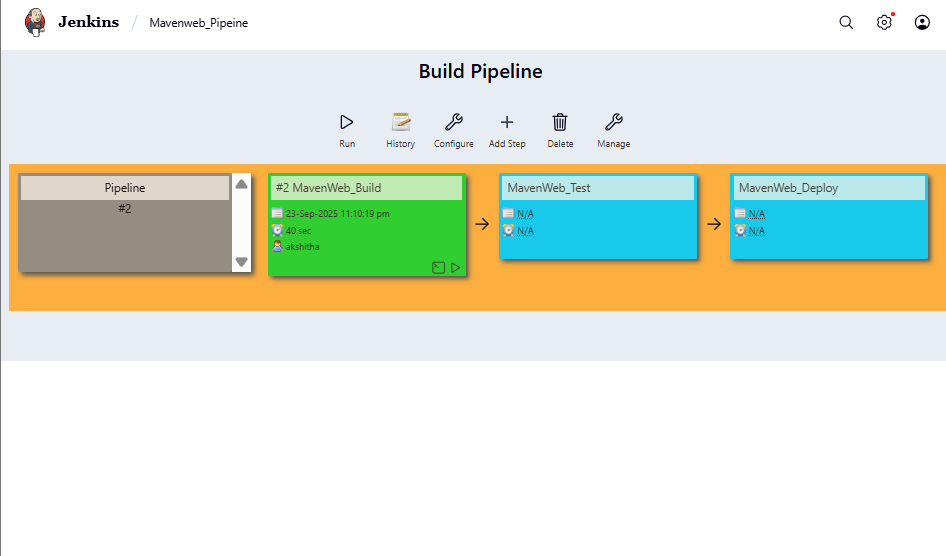


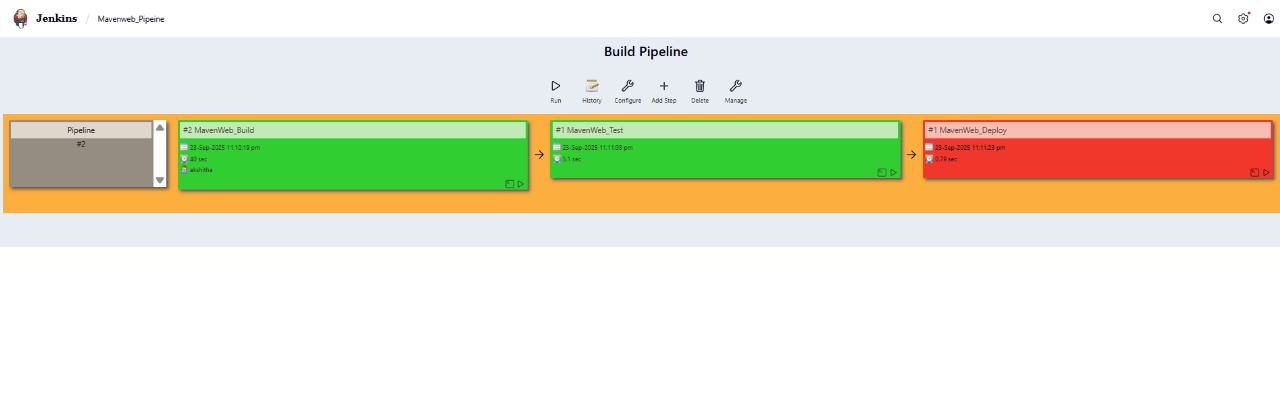
└── **Step 6**: Run the Pipeline and Check Output

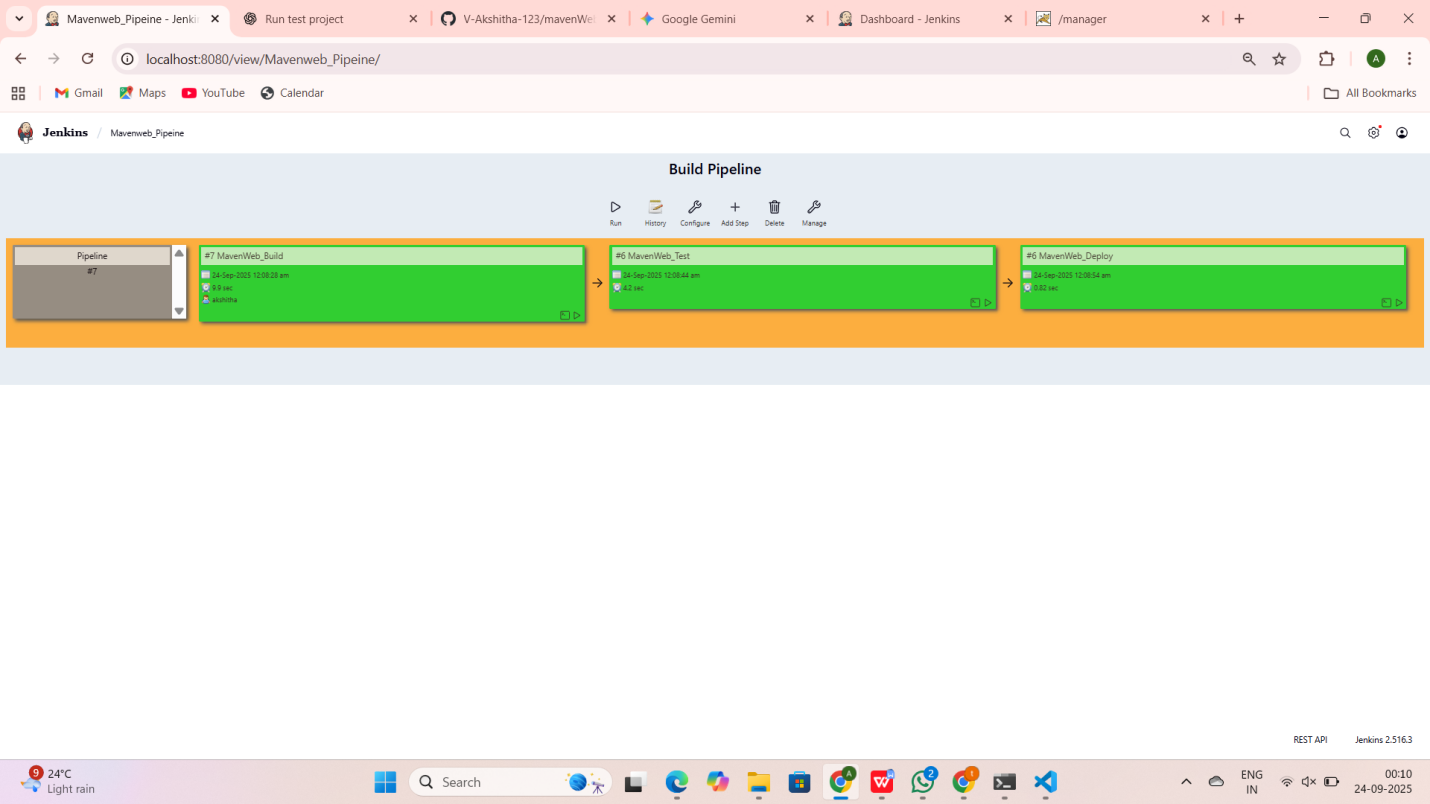
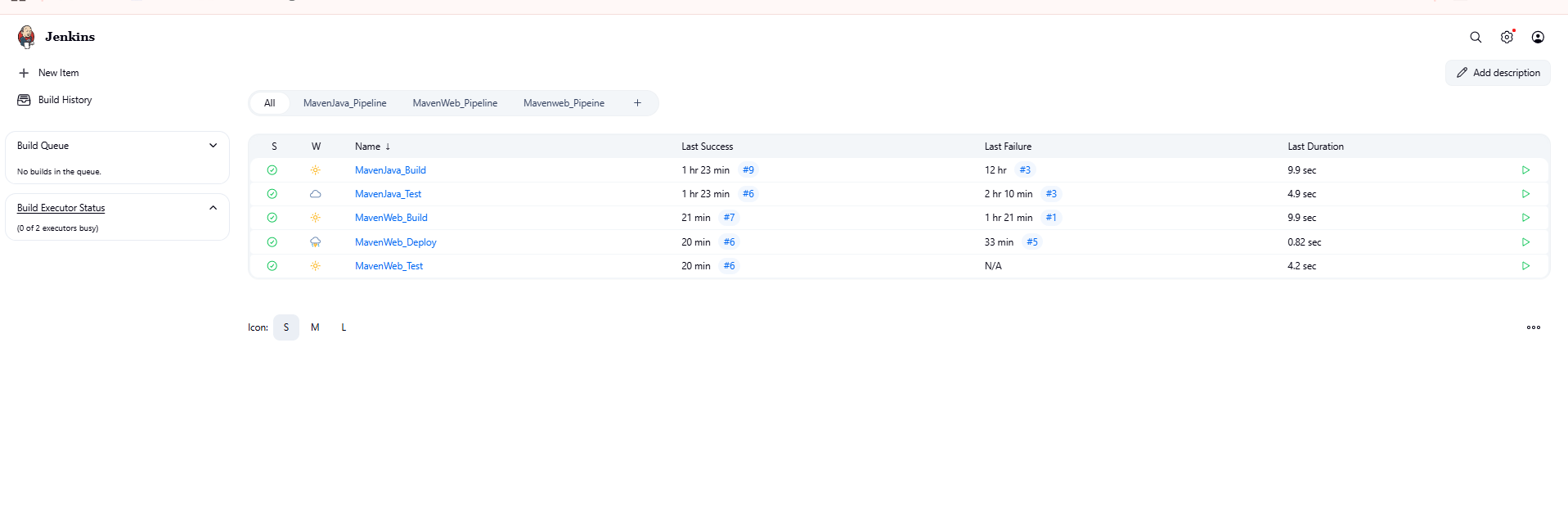
├── Click on the trigger **“RUN”** to run the pipeline



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
3. 
4. 

SBQ:

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

Automating build, test, and deployment processes — used for Continuous Integration (CI) and Continuous Delivery (CD).

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

Extensible with plugins, supports pipeline automation, distributed builds, and integration with version control systems.

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

8080

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

Git,github,bitbucket,svn

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

To extend Jenkins functionality — integrate with tools like Docker, Git, Maven, etc.

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

Freestyle projects

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

Use Credentials Plugin to securely store and access secrets.

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

A modern, visual UI for Jenkins pipelines — provides a clean, user-friendly interface for CI/CD visualization.

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

A modern, visual UI for Jenkins pipelines — provides a clean, user-friendly interface for CI/CD visualization.

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

Jenkins Master-Agent Architecture (specifically, Agents or Nodes handle distributed builds).

1. **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 Jenkins with Git repositories. |
| **Docker Plugin** | Builds and manages Docker containers during CI/CD. |
| **Pipeline Plugin** | Enables creation of CI/CD pipelines using code (Jenkinsfile). |
| **Kubernetes Plugin** | Deploys and scales builds on Kubernetes clusters. |
| **Slack Notification Plugin** | Sends build status notifications to Slack channels. |

1. **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?**
2. Go to **Manage Jenkins → Manage Plugins**
3. Select **Available** tab
4. Search for the plugin
5. Click **Install without restart**
6. Verify under **Installed** tab

**Considerations:**

* Check **version compatibility** with your Jenkins version.
* Ensure **plugin dependencies** are installed.
* Keep plugins **regularly updated** to avoid security risks.

1. **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?**
2. Go to **Manage Jenkins → Manage Plugins**
3. Select **Available** tab
4. Search for the plugin
5. Click **Install without restart**
6. Verify under **Installed** tab

**Considerations:**

* Check **version compatibility** with your Jenkins version.
* Ensure **plugin dependencies** are installed.
* Keep plugins **regularly updated** to avoid security risks.

1. **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**
2. Go to **Manage Jenkins → Configure System**
3. Find **Git** section
4. Set **Git executable path**
5. Add **GitHub credentials** under **Manage Credentials**
6. Use the plugin in your **Pipeline** or **Freestyle job** configuration.
7. **Discuss common issues that might arise when using Jenkins plugins, such as dependency conflicts or version compatibility problems. How would you troubleshoot these issues?**

| **Issue** | **Solution** |
| --- | --- |
| **Version conflicts** | Update or rollback plugins to compatible versions. |
| **Dependency errors** | Install missing dependent plugins. |
| **Build failures after updates** | Revert to previous plugin version. |
| **Security vulnerabilities** | Regularly update Jenkins and plugins. |
| **UI errors** | Clear Jenkins cache or restart the service. |