

## Week 8: Jenkins Automation

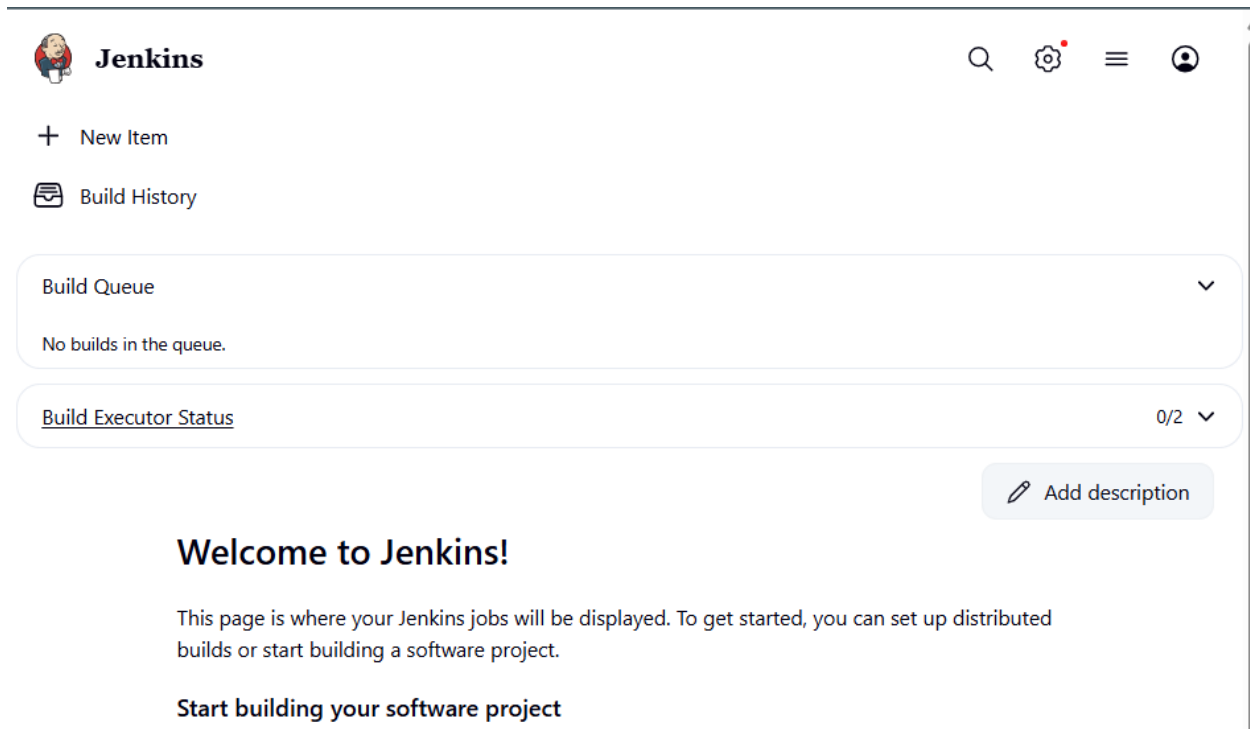
- I. Hands-on practice on manual creation of Jenkins pipeline using Maven projects from Github
- II. Create the job and build the pipeline for maven-java and maven-web project.
- III. Questions on Jenkins
- IV. Upload the Screenshots.

### 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]

The screenshot shows the Jenkins 'Configure' page for a job named 'MavenJava\_Build'. The 'Source Code Management' tab is selected. Under 'Source Code Management', the 'Git' option is chosen. A 'Repository URL' field contains 'https://github.com/Rudrangi-Vaishnavi/MavenJava.git'. A red error message 'Please enter Git repository.' is displayed below the field. The 'Credentials' dropdown is set to 'none'. An 'Add Repository' button is at the bottom.

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

└─ Build Steps:

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

└─ Maven version: MAVEN\_HOME

└─ Goals: clean

## Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

The screenshot shows the configuration for the 'Invoke top-level Maven targets' build step. The 'Maven Version' dropdown is set to 'MAVEN\_HOME'. The 'Goals' dropdown is set to 'clean'. An 'Advanced' dropdown is visible at the bottom. An 'Add build step' button is at the bottom left.

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

└─ Maven version: MAVEN\_HOME

└─ Goals: install

The screenshot shows the configuration for the 'Invoke top-level Maven targets' build step. It includes a 'Maven Version' dropdown set to 'MAVEN\_HOME', a 'Goals' text field containing 'install', and an 'Advanced' expandable section. Below the configuration box is an 'Add build step' button.

└─ **Post-build Actions:**

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

└─ Files to archive: \*\*/\*

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

The screenshot shows the configuration for the 'Archive the artifacts' post-build action. It features a 'Files to archive' text field with the pattern '\*\*/\*' and an 'Advanced' expandable section.

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

└─ Projects to build: MavenJava\_Test

└─ Trigger: Only if build is stable

└─ Apply and Save

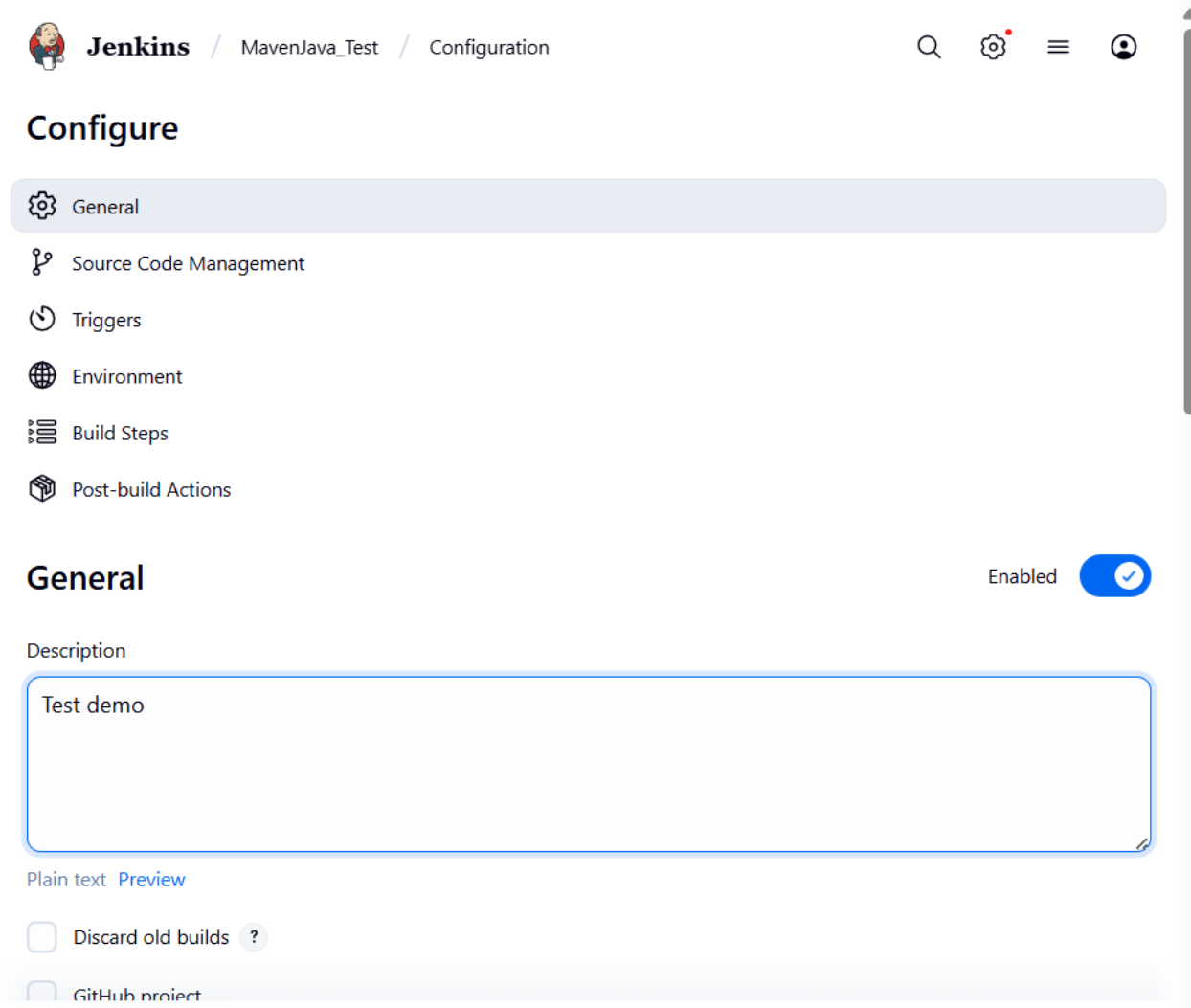
└─ **Step 3: Create Freestyle Project (e.g., MavenJava\_Test)**


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





└─ Click "OK"

└─ **Configure the project:**







└─ **Description:** "Test demo"




 **Jenkins** / MavenJava\_Test / Configuration

## Configure

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


### General

Enabled 

Description

Test demo

Plain text [Preview](#)

☐ Discard old builds 

☐ GitHub project

└─ **Build Environment:**

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

## Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

☒ Delete workspace before build starts

Advanced ▾

- ☐ Use secret text(s) or file(s) ?
  - ☐ Add timestamps to the Console Output
  - ☐ Inspect build log for published build scans
  - ☐ Terminate a build if it's stuck
  - ☐ With Ant ?
- 

## Build Steps

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

└─ Project name: MavenJava\_Build

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

└─ Artifacts to copy: \*\*/\*

## Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

≡

**Copy artifacts from another project**

×

Project name ?

Which build ?  

Latest successful build ▼

☒ Stable build only

Artifacts to copy ?

Artifacts not to copy ?

Target directory ?

Parameter filters ?

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

└─ Maven version: MAVEN\_HOME

└─ Goals: test

≡

**Invoke top-level Maven targets** ?

×

Maven Version  
 ▼

Goals  
 ▼

Advanced ▼

Add build step ▼

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

└─ Files to archive: \*\*/\*

└─ Apply and Save

## Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

The screenshot shows the 'Archive the artifacts' configuration window in Jenkins. It has a title bar with a hamburger menu, the title 'Archive the artifacts', a help icon, and a close button. Below the title bar, there is a label 'Files to archive' with a help icon, followed by a text input field containing '\*\*/\*'. Below the input field is an 'Advanced' dropdown button. At the bottom of the window is a button labeled 'Add post-build action' with a dropdown arrow.

└─ **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

## New view

Name

MavenJava\_Pipeline

Type



Build Pipeline View

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.



List View

Shows items in a simple list format. You can choose which jobs are to be displayed in which view.



My View

This view automatically displays all the jobs that the current user has an access to.

Create

└─ **Pipeline Flow:**

- └─ **Layout:** Based on upstream/downstream relationship
- └─ Initial job: MavenJava\_Build
- └─ Apply and Save OK

## Pipeline Flow

### Layout

Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config

Select Initial Job ?

MavenJava\_Build

### Trigger Options

#### Build Cards

Standard build card

Use the default build cards

Restrict triggers to most recent successful builds ?

☐ Yes

## └─ **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)

└─ **Step 2:** Create Freestyle Project (e.g., MavenWeb\_Build)

└─ Enter project name (e.g., MavenWeb\_Build)

└─ Click "OK"

## New Item

Enter an item name

MavenWeb Build

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.



### 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.

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

## Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

≡ **Invoke top-level Maven targets** ?

Maven Version

MAVEN\_HOME

Goals

clean

Advanced ▾

Add build step ▾

- └─ **Add Build Step** -> "Invoke top-level Maven targets"
- └─ Maven version: MAVEN\_HOME
- └─ Goals: install

≡ **Invoke top-level Maven targets** ?

Maven Version

MAVEN\_HOME

Goals

install

Advanced ▾

Add build step ▾

## Post-build Actions

- └─ **Post-build Actions:**
- └─ **Add Post Build Action** -> "Archive the artifacts"
- └─ Files to archive: \*\*/\*

## Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

≡ **Archive the artifacts** ?

Files to archive ?

Advanced ▾

Add post-build action ▾

- └─ **Add Post Build Action** -> "Build other projects"
- └─ Projects to build: MavenWeb\_Test
- └─ Trigger: Only if build is stable
- └─ Apply and Save

≡ **Build other projects** ?

Projects to build

❗ No such project 'MavenWeb\_Test'. Did you mean 'MavenJava\_Test'?

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Add post-build action ▾

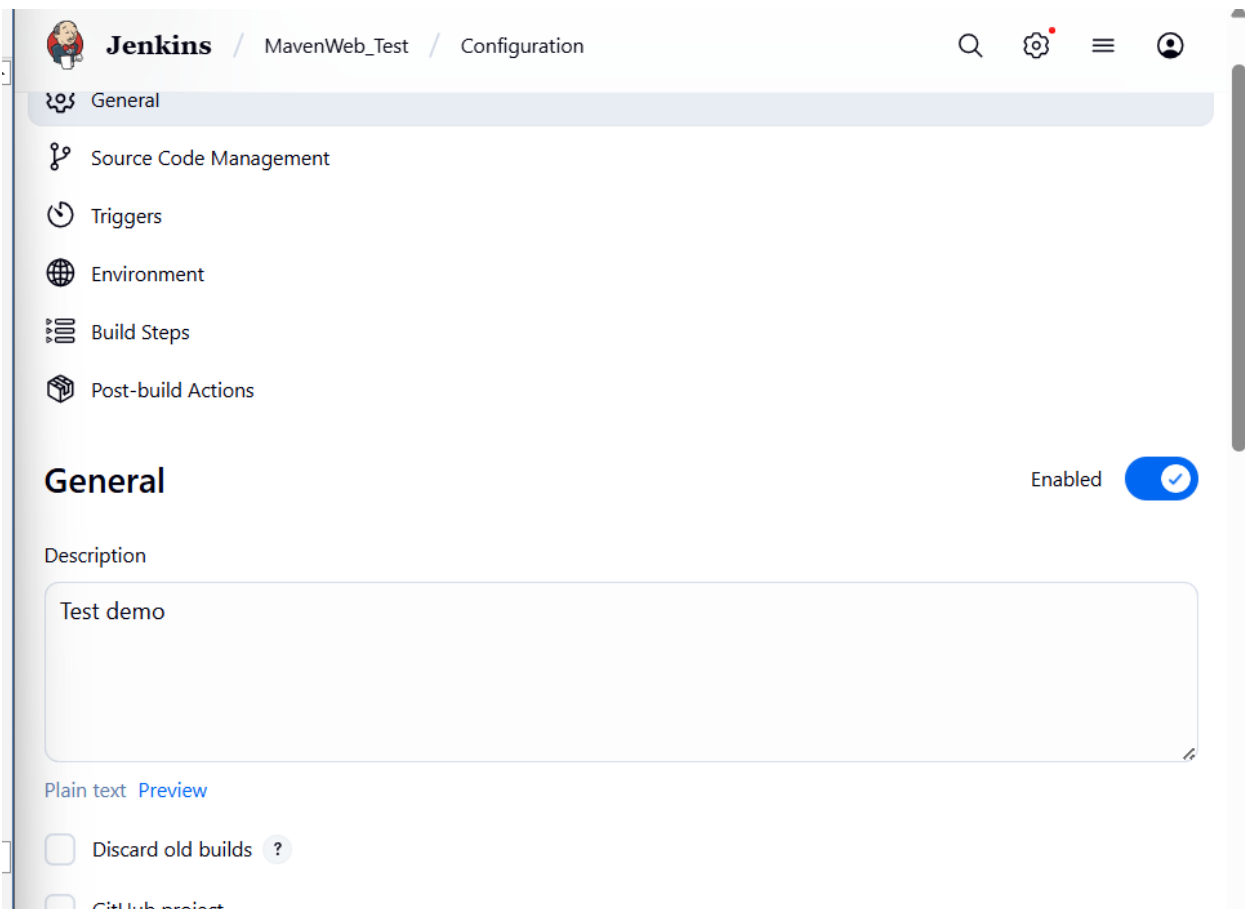
Save

Apply

- └─ **Step 3:** Create Freestyle Project (e.g., MavenWeb\_Test)
  - └─ Enter project name (e.g., MavenWeb\_Test)
  - └─ Click "OK"

## └─ Configure the project:

### └─ Description: "Test demo"



The screenshot shows the Jenkins web interface. At the top, the breadcrumb navigation reads 'Jenkins / MavenWeb\_Test / Configuration'. On the left sidebar, a list of configuration categories is shown: General (selected), Source Code Management, Triggers, Environment, Build Steps, and Post-build Actions. The main content area is titled 'General' and features a toggle switch labeled 'Enabled' which is turned on. Below this, there is a 'Description' section with a text area containing the text 'Test demo'. At the bottom of the visible section, there are two unchecked checkboxes: 'Discard old builds' and 'GitHub project'.

## └─ Build Environment:

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

#### Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

☒ Delete workspace before build starts

Advanced ▾

☐ Use secret text(s) or file(s) ?

☐ Add timestamps to the Console Output

☐ Inspect build log for published build scans

☐ Terminate a build if it's stuck

☐ With Ant ?

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

└─ Project name: MavenWeb\_Build

└─ Build: Stable build only

└─ Artifacts to copy: \*\*/\*

## Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

≡ **Copy artifacts from another project** ✕

Project name ?

Which build ?  

Latest successful build ▼

☒ Stable build only

Artifacts to copy ?

Artifacts not to copy ?

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

└─ Maven version: MAVEN\_HOME

└─ Goals: test

≡ Invoke top-level Maven targets ?

×

Maven Version

MAVEN\_HOME

Goals

test

Advanced

Add build step

└─ **Post-build Actions:**

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

└─ Files to archive: \*\*/\*

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

≡ Archive the artifacts ?

×

Files to archive ?

\*\*/\*

Advanced

Add post-build action

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

└─ Projects to build: MavenWeb\_Deploy

└─ Apply and Save

Build other projects ?

×

Projects to build

MavenWeb\_Deploy

! No such project 'MavenWeb\_Deploy'. Did you mean 'MavenWeb\_Build'?

☒ 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 4:** Create Freestyle Project (e.g., MavenWeb\_Deploy)


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





└─ Click "OK"


└─ **Configure the project:**


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




 **Jenkins** / MavenWeb\_Deploy / Configuration




 Environment

 Build Steps

 Post-build Actions


## General

Enabled 

Description

Web Code Deployment

Plain text [Preview](#)

☐ Discard old builds 

☐ [Start a new build](#)

└─ **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: \*\*/\*

Copy artifacts from another project

Project name ?

MavenWeb\_Test

Which build ?

Latest successful build

☒ Stable build only

Artifacts to copy ?

\*\*/\*

Artifacts not to copy ?

Target directory ?

### └─ 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"

## New view

Name

MavenWeb\_Pipeline

Type



Build Pipeline View

Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.



List View

Shows items in a simple list format. You can choose which jobs are to be displayed in which view.



My View

This view automatically displays all the jobs that the current user has an access to.

Create

### └─ Pipeline Flow:

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

└─ Initial job: MavenWeb\_Build

└─ Apply and Save

Pipeline Flow

Layout

Based on upstream/downstream relationship



This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config

Select Initial Job ?

MavenWeb\_Build



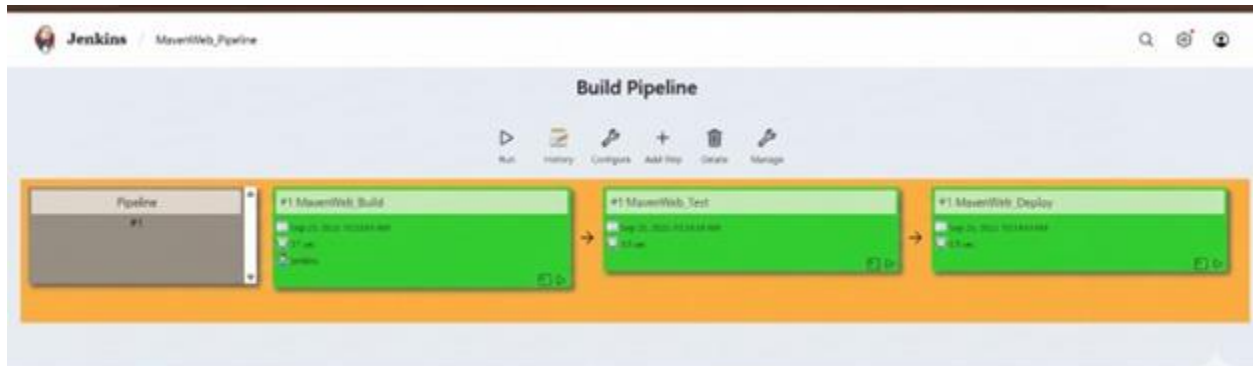
Trigger Options

## Step 6: Run the Pipeline and Check Output

Click on the trigger “RUN” to run the pipeline

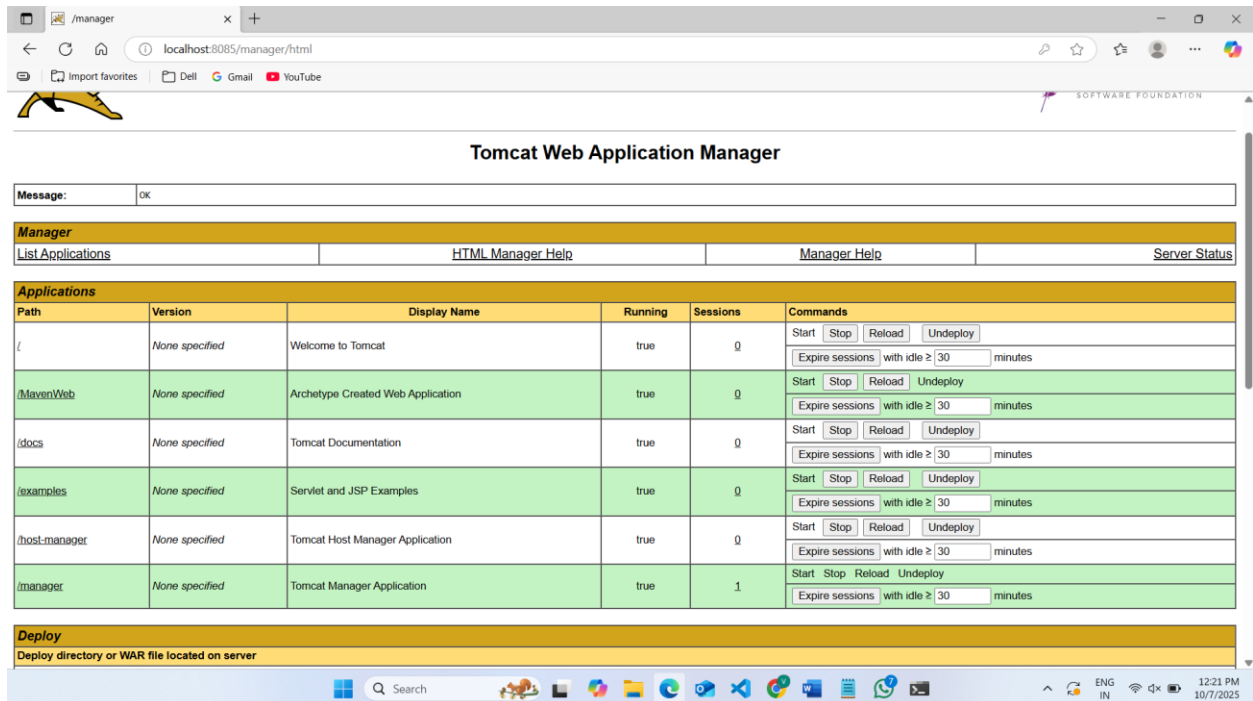
Note:

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



Open Tomcat homepage in another tab

Click on the "/webpath" option under the manager app



Note:

1. It ask for user credentials for login ,provide the credentials of tomcat.

2. It provide the page with out project name which is highlighted.
3. After clicking on our project we can see output.

### **III. Questions on Jenkins**

1. What is Jenkins primarily used for?
  - A. Jenkins is used for Continuous Integration and Continuous Deployment (CI/CD) — automating build, test, and deployment processes.
2. What is feature of Jenkins?
  - A. Jenkins is open-source, extensible with plugins, supports pipeline automation, and integrates with many tools (Git, Docker, etc.).
3. What is the default port on which Jenkins runs?
  - A. Port 8080.
4. What can be integrated with Jenkins for version control?
  - A. Git, GitHub etc
5. What is the purpose of Jenkins plugins?
  - A. Plugins extend Jenkins functionality
6. Which type of Jenkins job is best suited for running one-off tasks or small scripts?
  - A. Freestyle Project
7. How can you manage sensitive information such as API keys in Jenkins?
  - A. Use Credentials Manager to securely store and access secrets like passwords, tokens, and API keys.
8. What does the "blue ocean" feature in Jenkins refer to?
  - A. A modern, user-friendly UI for visualizing and managing Jenkins pipelines.
9. What does the "blue ocean" feature in Jenkins refer to?
  - A. A modern, user-friendly UI .
10. Which Jenkins component allows for distributed builds across multiple machines?
  - A. Jenkins Master-Agent (Node) Architecture
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.
  - A. Git Plugin - Integrates Jenkins with Git repositories , Pipeline Plugin - Enables code-based CI/CD pipelines, Docker Plugin - Builds and deploys Docker containers.
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?
  - A. Manage Jenkins->Manage Plugins->open available tab-> search install without restart

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?

A. Manage Jenkins->Manage Plugins->open available tab-> search install without restart.

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?

A. Manage Jenkins->Configure System->Find git section-> set the credentials ->choose source code management->git and enter the url, branch

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?

A. Dependency conflicts → update or roll back plugins

Version incompatibility → check Jenkins + plugin versions

Missing dependencies → install required plugins

**Conclusion:** In this week student learnt automating Maven projects through Jenkins.