

Week 8: Jenkins Automation

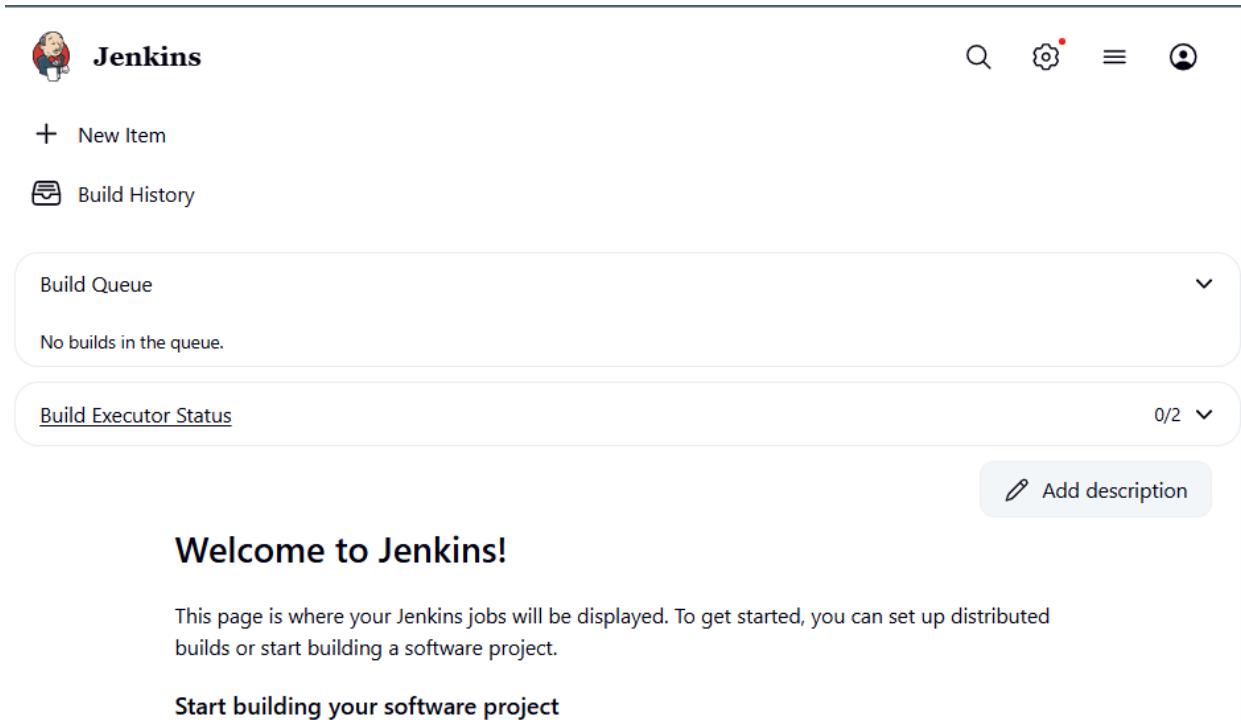
- I. Hands-on practice on manual creation of Jenkins pipeline using Maven projects from Github
- II. Create the job and build the pipeline for maven-java and maven-web project.
- III. Questions on Jenkins
- IV. Upload the Screenshots.

I. Steps for MavenJava Automation:

Maven Java Automation Steps:

Step 1: Open Jenkins (localhost:8080)

|—— Click on "New Item" (left side menu



The screenshot shows the Jenkins dashboard. At the top left is the Jenkins logo. To its right are search, settings, and user icons. Below the header, there's a 'New Item' button with a plus sign. Underneath it is a 'Build History' link. A 'Build Queue' section shows a message: 'No builds in the queue.' A 'Build Executor Status' section shows '0/2'. At the bottom right is a 'Add description' button with a pencil icon. The main content area features a large heading 'Welcome to Jenkins!' followed by a subtext: 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.' Below that is a 'Start building your software project' button.

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

|—— Enter project name (e.g., MavenJava_Build)

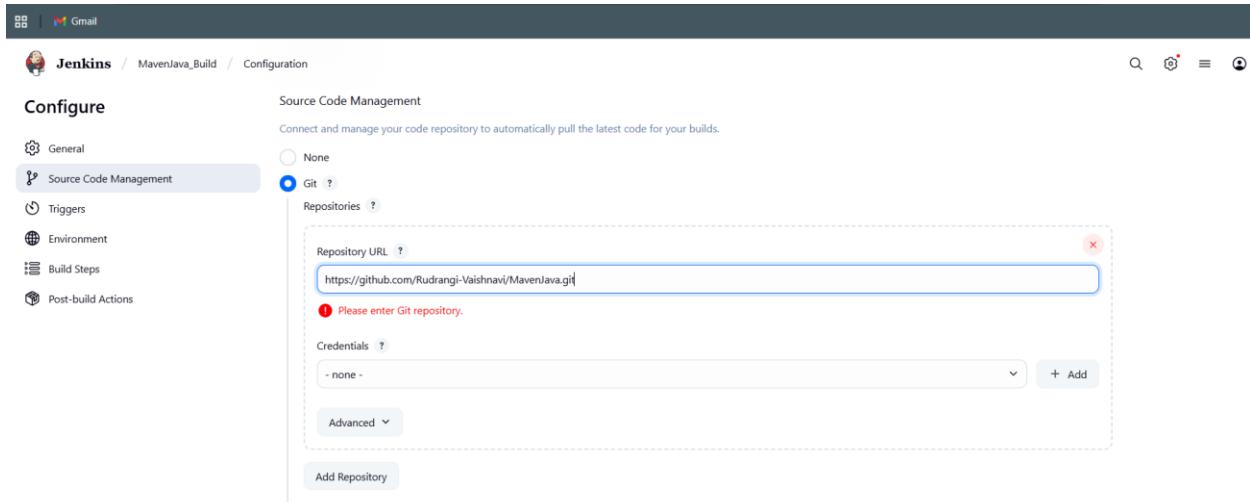
|—— Click "OK"

|—— 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

Build Steps

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

The screenshot shows the 'Invoke top-level Maven targets' build step configuration. It includes fields for 'Maven Version' (set to 'MAVEN_HOME') and 'Goals' (set to 'clean'). An 'Advanced' dropdown is also present. At the bottom, there is a 'Add build step' button.

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

 └─ Maven version: MAVEN_HOME

└─ Goals: install

The screenshot shows a configuration dialog for a build step. At the top, it says "Invoke top-level Maven targets". Below that is a "Maven Version" dropdown set to "MAVEN_HOME". Under "Goals", the value "install" is entered. There is also an "Advanced" button with a dropdown arrow.

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.

The screenshot shows a configuration dialog for archiving artifacts. It has a "Files to archive" dropdown containing "**/*". An "Advanced" button with a dropdown arrow is also present.

└─ 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"

The screenshot shows the Jenkins configuration interface for a project named 'MavenJava_Test'. The top navigation bar includes the Jenkins logo, the project name, and a 'Configuration' link. To the right are search, settings, and other management icons. The main title is 'Configure'. On the left, a sidebar lists configuration sections: General (selected), Source Code Management, Triggers, Environment, Build Steps, and Post-build Actions. The 'General' section is expanded, showing a 'Description' field containing 'Test demo', which is highlighted with a blue border. Below the description are two checkboxes: 'Discard old builds' (unchecked) and 'GitHub project' (unchecked). The status 'Enabled' is shown with a checked blue button.

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 ?

MavenJava_Build

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

MAVEN_HOME

Goals

test

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 a configuration dialog for 'Archive the artifacts'. At the top left is a three-dot menu icon and a question mark icon. Below that is a 'Files to archive' field with a question mark icon, containing the value '**/*'. To the right of the field is a red circular button with a white 'X'. Below the field is an 'Advanced' dropdown menu with a downward arrow. At the bottom left of the dialog is a 'Add post-build action' button with a downward 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

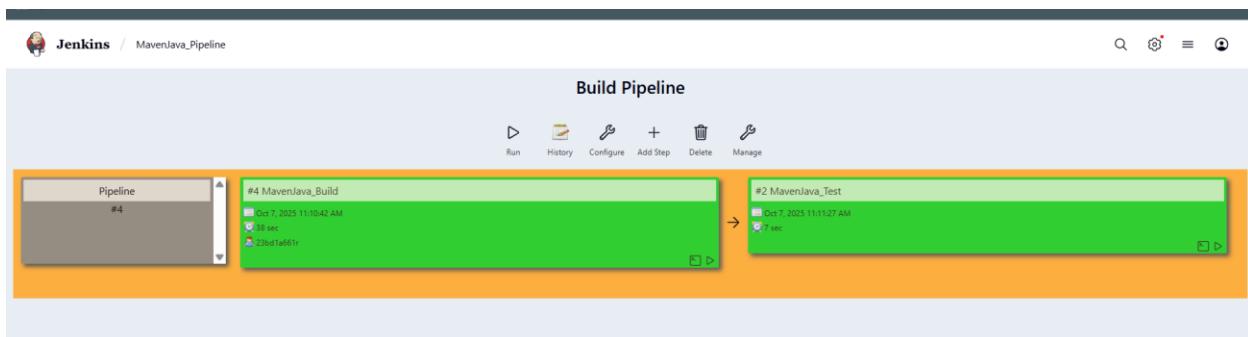
Restrict triggers to most recent successful builds ?

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** ? X

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** ? X

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.

The screenshot shows the 'Archive the artifacts' configuration screen. It includes a header with three horizontal bars, the title 'Archive the artifacts', a help icon, and a red 'X' button. Below the title is a section for 'Files to archive' with a placeholder '**/*'. An 'Advanced' dropdown menu is visible. At the bottom is a button labeled 'Add post-build action' with a dropdown arrow.

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

 └─ Projects to build: MavenWeb_Test

 └─ Trigger: Only if build is stable

 └─ Apply and Save

The screenshot shows the 'Build other projects' configuration screen. It includes a header with three horizontal bars, the title 'Build other projects', a help icon, and a red 'X' button. Below the title is a section for 'Projects to build' containing the text 'MavenWeb_Test'. A warning message '⚠ No such project 'MavenWeb_Test''. Did you mean 'MavenJava_Test'' is displayed. Below the input field are three trigger options: 'Trigger only if build is stable' (selected), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. At the bottom is a button labeled 'Add post-build action' with a dropdown arrow.

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"

Jenkins / MavenWeb_Test / Configuration

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 [?](#)

└─ **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.

The screenshot shows a configuration dialog for a build step. At the top, there's a title bar with a close button (red circle with an 'X'). Below it, the section title is "Copy artifacts from another project". The configuration fields include:

- Project name:** A dropdown menu showing "MavenWeb_Build".
- Which build:** A dropdown menu showing "Latest successful build".
- Stable build only:** A checkbox that is checked.
- Artifacts to copy:** A text input field containing "**/*".
- Artifacts not to copy:** An empty text input field.

└─ **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

The screenshot shows a Jenkins configuration dialog titled "Build other projects". In the "Projects to build" field, "MavenWeb_Deploy" is entered. A red error message below the field states: "No such project 'MavenWeb_Deploy'. Did you mean 'MavenWeb_Build'?". There are three trigger options: "Trigger only if build is stable" (selected), "Trigger even if the build is unstable", and "Trigger even if the build fails". Below the configuration area is a button labeled "Add post-build action" with a dropdown arrow. At the bottom are two buttons: "Save" (highlighted in blue) and "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

[Copy](#)

└─ 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

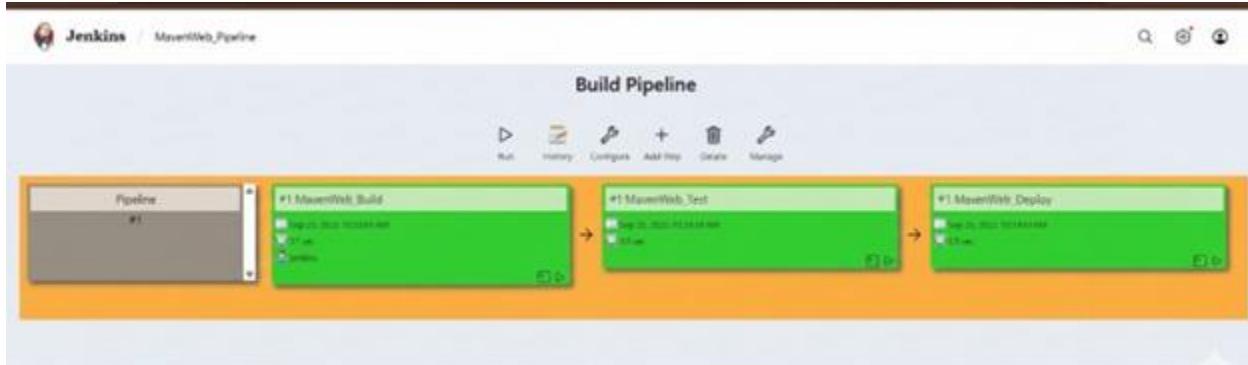
Trigger Options

└─ 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

The screenshot shows the Tomcat Web Application Manager interface at the URL "localhost:8085/manager/html". The page title is "Tomcat Web Application Manager". A message bar at the top says "Message: OK". Below it is a "Manager" section with tabs for "List Applications", "HTML Manager Help", "Manager Help", and "Server Status". The main area is titled "Applications" and contains a table listing seven deployed applications:

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/MavenWeb	None specified	Archetype Created Web Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

At the bottom, there is a "Deploy" section with the placeholder "Deploy directory or WAR file located on server". The browser's address bar shows "localhost:8085/manager/html".

Note:

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

2. It provide the page with our 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.