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]

├── **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

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"

└── **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 project:
        ├── 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

├── 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?
2. What is feature of Jenkins?
3. What is the default port on which Jenkins runs?
4. What can be integrated with Jenkins for version control?
5. What is the purpose of Jenkins plugins?
6. Which type of Jenkins job is best suited for running one-off tasks or small scripts?
7. How can you manage sensitive information such as API keys in Jenkins?
8. What does the "blue ocean" feature in Jenkins refer to?
9. What does the "blue ocean" feature in Jenkins refer to?
10. Which Jenkins component allows for distributed builds across multiple machines?
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.

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?
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?
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?
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?

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