SOFTWARE ENGINEERING LAB

EXERCISE – 6

TOPIC – 1

BUILDING THE CI/CD FREESTYLE PIPELINE USING JENKINS FOR MAVEN JAVA PROJECT

Note: At every step take screenshots and save in a document

CI/CD Pipeline

Setting up a CI/CD pipeline automates the build and testing processes for software projects, ensuring continuous integration and delivery. This helps teams identify issues early, maintain code quality, and deploy updates more efficiently. Using Jenkins for this purpose provides a robust and flexible platform to automate repetitive tasks, improve productivity, and streamline software delivery.

1. Setting Up Jenkins and Creating a Freestyle Project

- **Access Jenkins**: Open Jenkins by navigating to `localhost:8080` in your web browser. This is the Jenkins dashboard where you can manage and monitor jobs.
- **Create a New Item**: Click on "New Item" to start a new project. Enter a name for your project and select **Freestyle Project**. This type of project is a basic build job in Jenkins that provides the flexibility to run custom build steps. Click "OK" to proceed.

2. Configuring the Git Repository

- **Add Git Repository URL**: Enter the URL of the Git repository that contains the project you want Jenkins to build. This step links your project source code to Jenkins for automated builds.
- **Specify the Branch**: Indicate which branch Jenkins should use, such as `main` or `master`. This ensures Jenkins builds from the correct version of your code.

3. Setting Up Build Steps

- **Invoke Maven Targets**:
  - Go to the "Build" section and choose **Invoke top-level Maven targets**. This option tells Jenkins to run Maven commands, which are essential for building and managing Java projects.
  - Ensure the Maven path is set in Jenkins under **Manage Jenkins > Global Tool Configuration**. This configuration allows Jenkins to know where Maven is installed.
- **Specify Build Goals**: Enter Maven goals like `clean install`, which clean the project and compile the code. This step replicates the manual build process in Eclipse but in an automated manner.

4. Archiving Build Artifacts

- **Select Post-Build Actions**: Navigate to the "Post-build Actions" section, which defines what happens after a build completes.
- **Archive Artifacts**:
  - Choose **Archive the artifacts** to save build outputs. This is useful for future reference or testing purposes.
  - To archive all files, type `**/*`, which means all generated files from the build process will be stored.

5. Triggering Other Projects

- **Create a Test Project**:
  - Set up a second Freestyle Project to handle testing (e.g., `test project`). For this project, skip the Git repository configuration by selecting "None," as it will use artifacts from the previous project.
- **Copy Artifacts from Another Project**:
  - In the "Build Environment" section, check the option to discard old builds to save space.
  - Use **Copy artifacts from another project** and enter the name of the build project. This ensures the test project has access to the build outputs.

- Select "Stable build only" to ensure only successful builds are used, and type `**/*` to copy all files.

- **Add Build Steps for Testing**:
  - Choose **Invoke top-level Maven targets** and set the goal to `test`. This command runs unit tests defined in the project.

6. Finalizing Post-Build Actions

- **Archive Test Artifacts**: Repeat the artifact archiving step by typing `**/*` to save test results.
- **Apply and Save**: Click "Apply" and "Save" to store all project configurations.

7. Creating the Build Pipeline

- **Add a Pipeline View**:
  - On the Jenkins dashboard, click the "+" symbol to create a new view.
  - Select **Build Pipeline View**, which allows you to visualize and manage the sequence of jobs.
  - Provide a name for the pipeline and choose the initial build project as the starting point.
- **Run the Pipeline**:
  - Click "Run" to start the pipeline. This will trigger the sequence of jobs configured.
  - Click on the small black box in the pipeline view to open the console and monitor the build progress.
- **Verify Pipeline Status**:
  - A successful pipeline run appears in green, indicating that all linked jobs completed successfully.
  - Check the console output to confirm that both the build and test stages ran correctly and that artifacts were archived.