Unit-3

Part-A

Continuous Integration with Jenkins:

1. **Introduction to Continuous Integration (CI),**
2. **Using Maven for build,**
3. **Jenkins architecture and setup,**
4. **Managing Jenkins plugins and nodes,**
5. **Building and deploying applications using Jenkins,**
6. **Creating and managing Jenkins pipelines,**
7. **Pipeline as code with Jenkins.**

1.Introduction to Continuous Integration (CI)

Definition:-

Continuous Integration (CI) is a software development practice where developers regularly merge their code changes into a central repository, and automated builds and tests are run to quickly detect and fix errors. This helps ensure that the software remains in a working state and reduces integration problems.

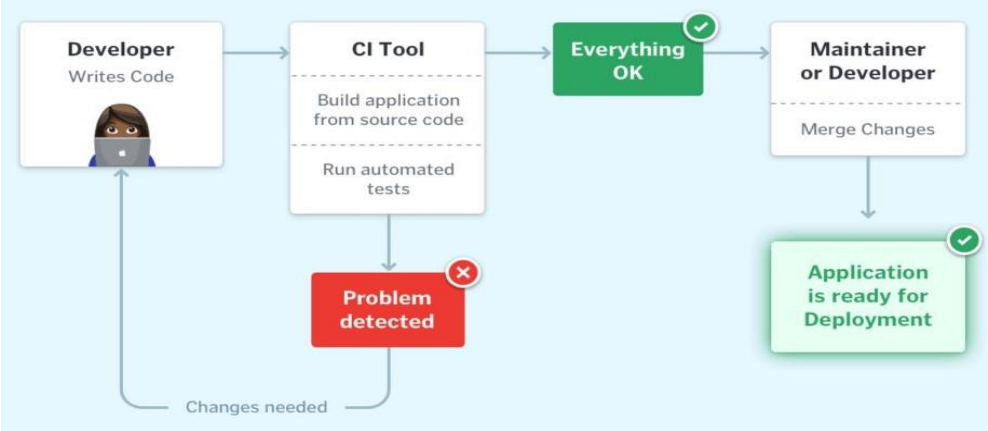
There could be scenarios when developers in a team work in isolation for an extended period and only merge their changes to the master branch once their work is completed. This not only makes the merging of code very difficult, prone to conflicts, and time-consuming but also results in bugs accumulating for a long time which are only identified in later stages of development. These factors make it harder to deliver updates to customers quickly.

With **Continuous Integration**, developers frequently commit to a shared common repository using a version control system such as Git. A continuous integration pipeline can automatically run builds, store the artifacts, run unit tests, and even conduct code reviews using tools like Sonar. We can configure the CI pipeline to be triggered every time there is a commit/merge in the codebase.

CI Workflow

Below is a pictorial representation of a CI pipeline- the workflow from developers checking in their code to its automated build, test, and final notification of the build status.

Once the developer commits their code to a version control system like Git, it triggers the CI pipeline which fetches the changes and runs automated build and unit tests. Based on the status of the step, the server then notifies the concerned developer whether the integration of the new code to the existing code base was a success or a failure.



This helps in finding and addressing the bugs much more quickly, makes the team more productive by freeing the developers from manual tasks, and helps teams deliver updates to their customers more frequently. It has been found that integrating the entire development cycle can reduce the developer's time involved by ~25 - 30%.

2.Using Maven for build

What is Maven?

- Maven is a **build automation tool** mainly for Java projects.
- It helps you **compile code, manage libraries (dependencies), run tests, and package your app** automatically.
- Uses a file called **pom.xml** to configure project details and build steps.

Why Use Maven?

- **Automates repetitive tasks** like compiling, testing, and packaging.
- **Manages dependencies** so you don’t have to download libraries manually.
- Ensures **consistent builds** on any machine.
- Integrates easily with Continuous Integration (CI) tools like Jenkins.

Common command used in builds:

mvn clean install

This cleans old files, compiles code, runs tests, and packages the project.

What is pom.xml?

- The **Project Object Model** file.
- It lists:
  - Project info (name, version)
  - Dependencies (libraries your project needs)
  - Build instructions (plugins, goals)

Install maven

In a Linux OS, We can install maven using "Sudo apt install maven" command and in a windows,for installing the maven we have to install maven environment application or we can access maven using IDE like. Eclipse, net-beans etc.

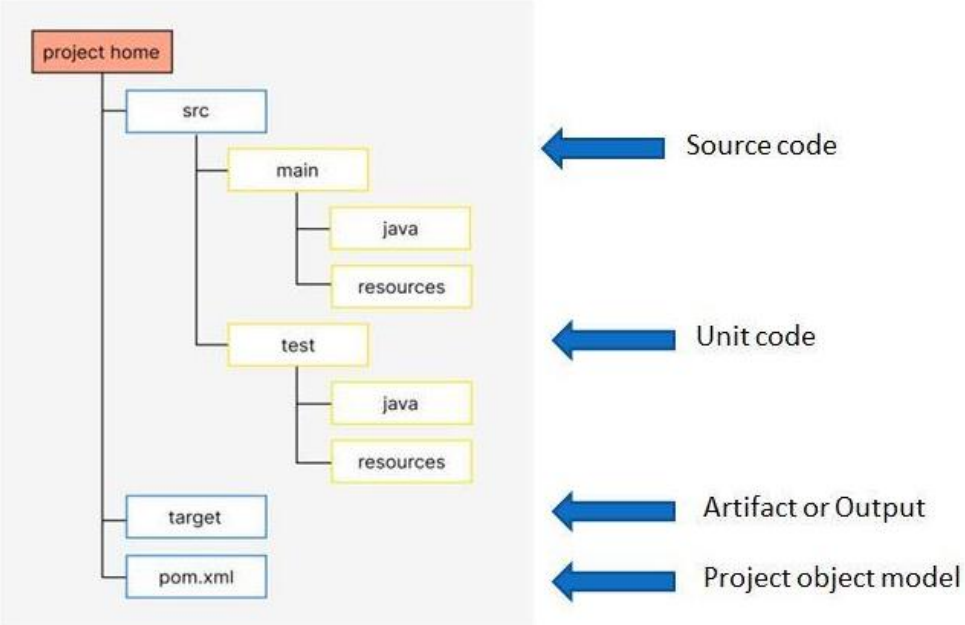
creating a Maven Project

For creating a Maven Project we have to provide two necessary things.

1. Archetype Group Id (e.g. CSD)

2. Artifact ID (e.g. Project Name)

Maven project Architecture



Maven Build Life Cycle

Maven Build life cycle consist some stages

1. Compile: For compile the Source code.

2. Test: Perform Test operation.

3. Package: Build a war file.

4. Install: Install the package in local system.

5. Deploy: Deploy the package to a remote repo

3.Jenkins architecture and setup

What is Jenkins?

- Jenkins is an **open-source automation server** used to automate software development tasks like building, testing, and deploying.
- It supports **Continuous Integration (CI)** and **Continuous Delivery (CD)**.
- Helps teams **integrate code frequently** and detect problems early.

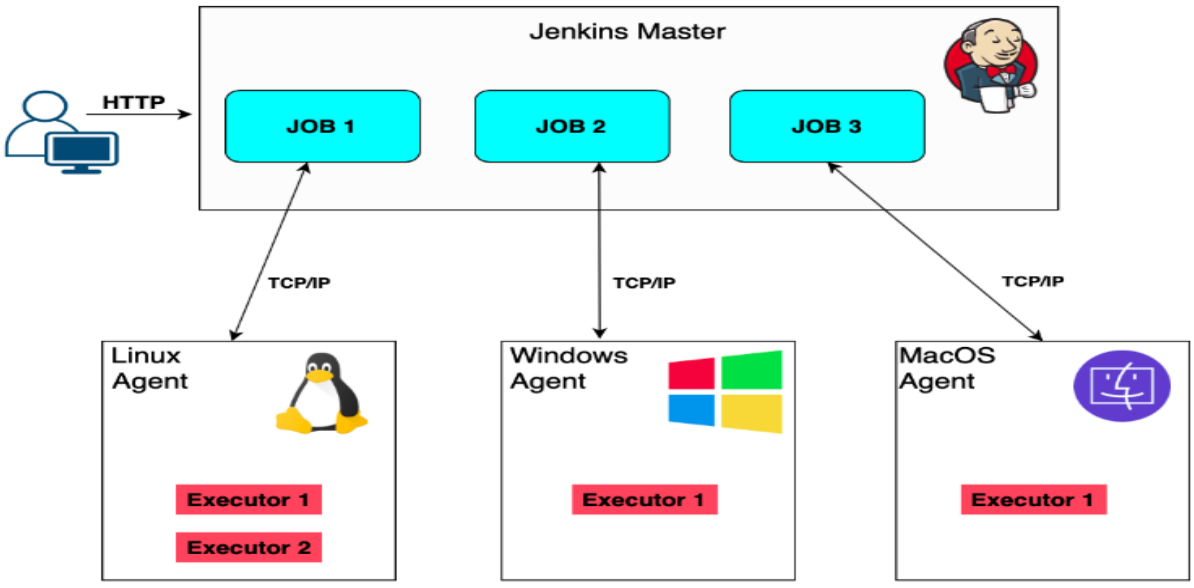


Fig.1.Jenkins Architecture

Jenkins follows a **master-agent (or master-slave)** architecture:

a) Jenkins Master

- The **central server** that manages the build environment.
- Responsible for:
    - Scheduling build jobs.
    - Dispatching builds to agents.
    - Monitoring agents.
    - Aggregating and displaying build results.
- Provides the **user interface (UI)** for configuring jobs and viewing results.

b) Jenkins Agents (or Slaves)

- Machines connected to the Jenkins master.
- Responsible for **executing build jobs** dispatched by the master.
- Can run on different platforms or environments.
- Help **distribute the workload**, speeding up the build process.

How Jenkins Works

1. Developers push code to a repository (e.g., Git).
2. Jenkins master detects the code change (via polling or webhooks).
3. Jenkins master schedules a build job.
4. Master dispatches the build to an available agent.
5. Agent performs the build steps (compile, test, package).
6. Agent sends build results back to the master.
7. Jenkins master displays the build status to users.

Jenkins Setup — Basic Steps

a) Install Jenkins

b) Step-1: first we must have to download the java jdk-11, 17 or 21 and install in our system.

c) Step-2: In a Linux OS, we can install Jenkins using “sudo apt install Jenkins” command. In a

d) windows we can directly download the environment and configure the Jenkins.

e) Step-3: In a Linux we have some command for start the Jenkins servers

f) “sudo Systemctl enable Jenkins”

g) “sudo systemctl start Jenkins”

h) “sudo systemctl status Jenkins”

(or)

- Can be installed on Windows, Linux, macOS, or run via Docker.
- Download from the official Jenkins website: https://jenkins.io

b) Initial Configuration

- Access Jenkins via web browser (default: http://localhost:8080).
- Unlock Jenkins by entering the initial admin password (found in installation logs or files).
- Install suggested plugins (supports many build tools, version control systems, etc.).
- Create the first admin user.

c) Configure Tools and Environment

- Configure JDK, Maven, Git, etc., in Jenkins global tools configuration.
- Set up credentials (like GitHub tokens, SSH keys).

d) Create Your First Job

- Click **New Item**.
- Choose job type (e.g., Freestyle project, Pipeline).
- Configure source code repository.
- Add build steps (e.g., run Maven commands).
- Save and run the job.

4. Managing Jenkins plugins and nodes

Managing Jenkins Plugins

- **Plugins** add new features to Jenkins (e.g., Git integration, build tools, notifications).
- Jenkins comes with many plugins, and you can add more as needed.

How to Manage Plugins:

1. Go to **Manage Jenkins → Manage Plugins**.
2. You will see tabs:
    - **Updates:** Plugins with available updates.
    - **Available:** Plugins you can install.
    - **Installed:** Plugins already installed.
    - **Advanced:** Upload plugins manually or change update settings.
3. To **install plugins**, go to the **Available** tab, select plugins, and click **Install**.
4. To **update plugins**, go to the **Updates** tab, select plugins, and install updates.
5. To **remove plugins**, go to the **Installed** tab, find the plugin, and click **Uninstall**.
6. Sometimes Jenkins requires a **restart** after installing or updating plugins.

Managing Jenkins Nodes (Agents)

- Jenkins master controls everything, but **nodes** (or agents) do the actual build work.
- Nodes can be on different machines or operating systems.
- Using nodes allows parallel builds and running builds in different environments.

How to Manage Nodes:

1. Go to **Manage Jenkins → Manage Nodes and Clouds**.
2. You will see a list of all nodes, including the master.
3. To **add a new node**:
    - Click **New Node**.
    - Enter a name and choose **Permanent Agent**.
    - Configure node details like remote directory, labels, and launch method (e.g., SSH).
    - Save the node.
4. Connect the node based on the launch method:
    - SSH: Jenkins connects automatically.
    - Java Web Start: Run a command on the agent machine to connect.
5. You can **disable** or **delete** nodes if needed.
6. Nodes help distribute build jobs and improve performance.

5.Building and deploying applications using Jenkins

1.What is Building?

- Compiling source code, running tests, and packaging the application.

How Jenkins Builds Applications:

1. **Pull Code:** Jenkins fetches the latest code from a repository (e.g., Git).
2. **Compile Code:** Runs build tools like Maven or Gradle (mvn clean install).
3. **Run Tests:** Executes automated tests to verify code quality.
4. **Package:** Creates the executable file or archive (like a JAR, WAR, or Docker image).
5. **Archive Artifacts:** Saves the build outputs for later use or deployment.

Setting up a Build Job:

- Create a new Jenkins job (Freestyle or Pipeline).
- Configure repository details.
- Add build steps (commands or scripts).Ex- mvn clean install
- Run the job manually or trigger automatically on code changes.

2.deploying applications using Jenkins

Deploying an application using Jenkins and Tomcat automates the process of moving your built web application (usually a WAR file) to a Tomcat server, where it can be accessed and run. After Jenkins builds the WAR file, it can automatically transfer the file to the Tomcat server and deploy it using Tomcat's manager web application or by copying it into Tomcat's deployment directory. This automation saves time, reduces manual errors, and ensures that the latest

application version is always deployed. Jenkins can use plugins like the **Deploy to Container Plugin** to make deployment easier and more integrated, or use custom scripts to copy files and restart Tomcat if needed.

Steps to Deploy Application on Tomcat Using Jenkins

1. **Prepare Tomcat Server:**
    - Install Apache Tomcat on your server or local machine.
    - Enable Tomcat Manager App (make sure user credentials with deploy permissions are set in tomcat-users.xml):

      xml
      CopyEdit
      <user username="jenkins" password="jenkins123" roles="manager-script"/>

    - Note the Tomcat server URL (e.g., http://localhost:8080/manager/text).
2. **Install Jenkins Plugins:**
    - Go to **Manage Jenkins → Manage Plugins**.
    - Install **Deploy to Container Plugin** (for easy Tomcat deployments).
3. **Create a Jenkins Job:**
    - Create a Freestyle project or Pipeline job.
    - Configure source code repository (e.g., Git).
    - Add build steps to compile and package the application (e.g., run Maven clean package to create WAR).
4. **Configure Deployment in Jenkins:**
    - In the post-build section, add **Deploy war/ear to a container**.
    - Enter:
        - WAR/EAR files: **/*.war (path to your built WAR file).
        - Context path (optional): e.g., /myapp.
        - Containers: Choose **Tomcat 7+ Remote**.
        - Credentials: Provide username/password for Tomcat Manager.
        - Tomcat URL: e.g., http://localhost:9090/manager/text.
5. **Save and Run the Job:**
    - Run the job manually or trigger via SCM changes.
    - Jenkins will build the WAR and deploy it automatically to Tomcat.
6. **Verify Deployment:**
    - Access the application in a browser at http://localhost:9090/myapp.
    - Check Jenkins console logs for deployment success messages.

6.Creating and managing Jenkins pipelines

Jenkins Pipeline

A Jenkins Pipeline is a concept that represents the fully automated Continuous Integration (CI) and Continuous Delivery (CD) process for a software using a Jenkinsfile. It consists of a series of steps that define the complete CI/CD workflow, from source code management to final deployment for end users.

Jenkin file

Jenkinfile is a text file that contain the definition and steps of jenkins Pipeline.

There are two types of pipelines in Jenkins

1. Declarative Pipeline: Simplified, structured syntax for easy pipeline creation.

2. Scripted Pipeline: More flexible, written in standard Groovy syntax.

1.Declarative Pipeline:

```
pipeline {

agent any // Runs on any available agent

stages {

stage('Build') {

steps {

echo 'Building the application...'

sh 'mvn clean package'

}

}

stage('Test') {

steps {

echo 'Running tests...'

sh 'mvn test'

}
```

```
}

}

}
```

2.Scripted Pipeline:

```
node {

stage('Checkout') {

echo 'Fetching source code from GitHub...'

git url: 'https://github.com/user/repository.git', branch: 'main'

}

stage('Build') {

echo 'Building the application...'

sh 'mvn clean package'

}

}
```

Setup Process

For setup a pipeline, first we have to add and install a Pipeline plugins from manage jenkins option.

After installation of pipeline plugins it will automatically add a jenkinfile in our git repository.

Steps for Creating a Pipeline

 Step-1: Click on new Item in Dashboard.

 Step-2: Give a Name and select Pipeline

 Step-3: Click OK and navigate to the Pipeline section.

 Step-4: Choose Pipeline Script (inline) or Pipeline from SCM (Jenkinsfile in Git).

 Step-5: Write down the pipeline script according to your need.

 Step-6: Save and run the pipeline.

Managing Jenkins Pipeline

We can Manage jenkins pipeline manually using Manually trigger a pipeline from the Jenkins UI

Dashboard,

And we have option to manage jenkins using automation tool like…

1. Upstream project builds.

2. Webhooks (GitHub/GitLab integration).

3. Poll SCM (H/5 * * * * for every 5 minutes).

7. Pipeline as code with Jenkins

Pipeline as Code is a practice where the entire Continuous Integration/Continuous Delivery (CI/CD) process is defined and managed through a Jenkinsfile. Instead of configuring pipelines manually via the Jenkins UI dashboard, they are written as code, stored in version control (e.g., Git), and executed by Jenkins automatically.

Why Pipeline as a code is important

- Version Control: Pipelines are stored jenkinfile in Git, using that one we can track the
- changes.
- Automation: Pipeline Reduce the Manuel configuration process
- Reproducibility: Ensures consistency across builds.
- Scalability: Easy to maintain and modify as projects grow.
- Collaboration: Developers and DevOps teams can work together on CI/CD pipelines.

Syntax:- of Pipeline as a code

```
pipeline {

agent any

stages {

stage(“Stage_name“) {

steps {

// Source info or command

}

}
```

}

}

Example of Pipeline as a code

For Testing

```
pipeline {

agent any

stages {

stage(„Test“) {

steps {

echo 'Running tests...'

sh 'mvn test'

}

}

}

}
```

For Source Code Management

```
pipeline {

agent any

stages {

stage('Checkout') {

steps {

git url: 'https://github.com/repo.git', branch: 'main'

}

}

}
```

}

For Build an artifact

```
pipeline {
agent any
stages {
stage(„Build') {
steps {
echo 'Building the application...'
sh 'mvn clean package'
}
}
}
}
```