A Shared Library in Jenkins is a reusable collection of Groovy scripts and functions that can be shared across multiple pipelines.

It allows you to centralize and standardize common pipeline tasks, such as building, testing, deploying, or setting up environments.

Key Features of Shared Libraries:

Reusability: Write once and use it in multiple pipelines.

Modularity: Organize reusable into functions or classes for better maintenance.

Consistency: Ensure consistent behavior across pipelines by using standardized scripts.

Scalability: Simplify pipeline management for large-scale projects with many jobs.

Shared Library Function: Install Java and Maven

vars/installJavaAndMaven.groovy

def call(String javaVersion = '17', String mavenVersion = 'latest') {

echo "Installing Java ${javaVersion} and Maven ${mavenVersion}..."

// Update package index

sh 'sudo apt update'

// Install the specified version of Java

sh "sudo apt install -y openjdk-${javaVersion}-jdk"

// Install Maven (either a specific version or the latest)

if (mavenVersion == 'latest') {

sh 'sudo apt install -y maven'

} else {

echo "Installing Maven version ${mavenVersion} is not implemented in this example."

error "Currently only 'latest' Maven installation is supported."

}

// Verify installations

sh 'java -version'

sh 'mvn -version'

echo "Java ${javaVersion} and Maven installation completed!"

}

Explanation

Purpose:

Automates the installation of Java (default version: 17) and Maven (default: latest).

Ensures a consistent setup across pipelines.

Parameters:

javaVersion (optional): The version of Java to install (default: 17).

mavenVersion (optional): The Maven version to install (default: latest).

Functionality:

Updates the package index using apt update.

Installs Java using apt install.

Installs Maven (currently supports only the latest version).

Verifies the installations by running java -version and mvn -version.

Usage in Jenkinsfile

groovy

@Library('my-shared-library') \_

pipeline {

agent any

stages {

stage('Install Java and Maven') {

steps {

script {

installJavaAndMaven('17', 'latest')

}

}

}

stage('Build with Maven') {

steps {

echo 'Building the project...'

sh 'mvn clean package'

}

}

}

}

Output

Console Output:

csharp

[Pipeline] echo

Installing Java 17 and Maven latest...

[Pipeline] sh

+ sudo apt update

[Pipeline] sh

+ sudo apt install -y openjdk-17-jdk

[Pipeline] sh

+ sudo apt install -y maven

[Pipeline] sh

+ java -version

[Pipeline] sh

+ mvn -version

[Pipeline] echo

Java 17 and Maven installation completed!

Why Use This Function?

Streamlined Setup:

Ensures Java and Maven are always available for the pipeline without manual intervention.

Flexibility:

Allows specifying Java and Maven versions, making it adaptable for various projects.

Reusability:

Centralizes the setup logic for Java and Maven, making it easy to update and maintain.