**Components and Workflow Explanation**

1. **Jenkins Pipeline:**
   * **Action:** Initiates the build process by invoking the shared library method setupBuildTool.
   * **Inputs Provided:**
     + toolName (e.g., maven)
     + toolVersion (e.g., 3.8.7)
     + downloadUrl (URL to download the tool from Artifactory)
     + toolbinpath(where the tool or exe present in downloaded zip)
2. **Shared Library Method (setupBuildTool):**
   * **Purpose:** Manages the verification, downloading, setup, and path configuration of the specified build tool.
   * **Inputs:**
     + toolName: Name of the build tool (e.g., maven, node)
     + toolVersion: Specific version required
     + downloadUrl: Artifactory URL for downloading the tool
     + toolbinpath(where the tool or exe present in downloaded zip)
3. **Detect Operating System:**
   * **Function:** Determines whether the Jenkins agent is running on Windows or Linux.
   * **Implementation:** Utilize Jenkins environment variables or Groovy's System properties to detect the OS.
   * **Purpose:** Ensures that subsequent steps are executed with OS-specific commands and paths.
4. **Path Setup (Windows vs. Linux):**
   * **Windows:**
     + **Central Tools Directory:** Typically C:\Jenkins\tools\
   * **Linux:**
     + **Central Tools Directory:** /opt/jenkins/tools/
5. **Check Central Tools Directory:**
   * **Function:** Verifies if the specified tool and version already exist in the central directory.
   * **Decision Points:**
     + **Yes:** Tool exists (/opt/jenkins/tools/${toolName}-${toolVersion+toolBinPath}.
     + **No:** Tool does not exist and needs to be downloaded
6. **If Tool Exists:**
   * **Return Existing Tool Path:**
     + **Windows Example:** C:\Jenkins\tools\maven-3.8.7\bin\mvn.exe
     + **Linux Example:** /opt/jenkins/tools/maven-3.8.7/bin/mvn
   * **Create Symlink in Workspace:**
     + **Windows:** Use mklink to create a directory junction or symbolic link.
   * **Update PATH Environment Variable:**
     + **Windows:**
     + env.PATH = "C:\\Jenkins\\workspace\\tools\\${toolName}-${toolVersion}\\bin;${env.PATH}"
     + **Linux:** env.PATH = "${workspaceToolPath}:${env.PATH}"
   * **Output:** Returns the path to the tool's bin directory for use in subsequent build stages.
7. **If Tool Does Not Exist:**
   * **Download Tool from Artifactory:**
     + **Windows:**
       - Use PowerShell or curl to download.
       - Invoke-WebRequest -Uri $downloadUrl -OutFile "C:\Jenkins\tools\${toolName}-${toolVersion}.zip"
     + **Linux:**
       - Use curl or wget to download.
       - curl -L -o /opt/jenkins/tools/${toolName}-${toolVersion}.zip ${downloadUrl}
   * **Uncompress Tool Archive:**
     + **Windows:**
       - Use PowerShell's Expand-Archive or a tool like 7-Zip.
       - Expand-Archive -Path "C:\Jenkins\tools\${toolName}-${toolVersion}.zip" -DestinationPath "C:\Jenkins\tools\${toolName}-${toolVersion}"
     + **Linux:**
       - Use unzip or tar based on archive type.
       - unzip /opt/jenkins/tools/${toolName}-${toolVersion}.zip -d /opt/jenkins/tools/${toolName}-${toolVersion}
   * **Set Executable Permissions (If Necessary):**
     + **Windows:** Generally not required, but ensure executables are accessible.
     + **Linux:** chmod +x /opt/jenkins/tools/${toolName}-${toolVersion}/bin/\*
   * **Create Symlink in Workspace:** *(Same as in "If Tool Exists" step)*
   * **Update PATH Environment Variable:** *(Same as in "If Tool Exists" step)*
   * **Output:** Returns the path to the tool's bin directory.
8. **Return Build Tool Path to Pipeline:**
   * **Windows Example:** C:\Jenkins\workspace\tools\maven-3.8.7\bin\
   * **Linux Example:** /workspace/tools/maven-3.8.7/bin/

**Detailed Workflow Steps**

1. **Start: Provide Inputs**
   * **Inputs:**
     + toolName: Name of the build tool (e.g., maven, node)
     + toolVersion: Specific version (e.g., 3.8.7, 14.17.0)
     + downloadUrl: Direct URL to download the tool archive from Artifactory
2. **Initialize: Load Shared Library and Define Paths**
   * **Action:** The Jenkins pipeline loads the shared library containing the setupBuildTool method.
   * **Define Paths:** Based on the detected OS, set base directories for tools and workspace symlinks.
3. **Detect Operating System**
   * **Windows:**
     + Paths use backslashes (\)
     + Executable extensions like .exe
   * **Linux:**
     + Paths use forward slashes (/)
     + Executables do not have extensions
4. **Check Central Tools Directory**
   * **Function:** Verify if the tool exists in the central tools directory.
   * **Condition:** Check for the presence of the tool's binary.
5. **Decision Point: Tool Exists?**
   * **Yes:**
     + **Action:** Proceed to set up environment paths and create symlinks.
   * **No:**
     + **Action:** Download and set up the tool as per OS-specific steps.
6. **Download and Uncompress Tool (If Needed)**
   * **Windows:**
     + Use PowerShell commands or Windows-specific utilities to handle downloads and extraction.
   * **Linux:**
     + Utilize shell commands like curl/wget and unzip/tar.
7. **Set Executable Permissions (Linux Only)**
   * **Action:** Ensure that binaries have execute permissions using chmod.
8. **Create Symlink in Workspace**
   * **Purpose:** Provide workspace-specific access to the tool without duplicating the files.
   * **Windows:** Use mklink for directory junctions or symbolic links.
   * **Linux:** Use ln -sfn to create or update symbolic links.
9. **Update PATH Environment Variable**
   * **Purpose:** Make the tool's binaries accessible in the build environment.
   * **Windows:** Prepend the tool's bin directory to env.PATH using Windows path syntax.
   * **Linux:** Prepend the tool's bin directory to env.PATH using Unix path syntax.
10. **Return Build Tool Path to Pipeline**
    * **Output:** The method returns the path to the tool's bin directory, which the Jenkins pipeline can use to execute build commands.

Creating a Groovy Maven project for a shared library is a great way to modularize code and make it reusable across different projects. Here’s a basic structure and configuration to help you get started.

**1. Project Structure**

Assume your project structure is as follows:

css

Copy code

shared-library/

├── src

│ ├── main

│ │ ├── groovy

│ │ │ └── com

│ │ │ └── example

│ │ │ └── MyLibrary.groovy

│ │ └── resources

│ └── test

│ ├── groovy

│ │ └── com

│ │ └── example

│ │ └── MyLibraryTest.groovy

│ └── resources

├── pom.xml

└── README.md

In this structure:

* The src/main/groovy folder contains your main Groovy code.
* The src/test/groovy folder holds test code.

**2. pom.xml Configuration**

Here’s a pom.xml that sets up a Groovy Maven project for a shared library.

xml

Copy code

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>shared-library</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>Shared Library</name>

<description>A shared library project in Groovy</description>

<properties>

<groovy.version>3.0.9</groovy.version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Groovy dependency -->

<dependency>

<groupId>org.codehaus.groovy</groupId>

<artifactId>groovy</artifactId>

<version>${groovy.version}</version>

</dependency>

<!-- JUnit for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<!-- Spock for BDD-style testing (optional) -->

<dependency>

<groupId>org.spockframework</groupId>

<artifactId>spock-core</artifactId>

<version>2.0-M5-groovy-3.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Groovy Maven Plugin -->

<plugin>

<groupId>org.codehaus.gmavenplus</groupId>

<artifactId>gmavenplus-plugin</artifactId>

<version>1.13.0</version>

<executions>

<execution>

<goals>

<goal>compile</goal>

<goal>testCompile</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

**3. Sample Groovy Class (MyLibrary.groovy)**

Here’s a sample Groovy class in src/main/groovy/com/example/MyLibrary.groovy:

groovy

Copy code

package com.example

class MyLibrary {

static String greet(String name) {

return "Hello, $name!"

}

}

**4. Sample Test Class (MyLibraryTest.groovy)**

Add a basic unit test for your library in src/test/groovy/com/example/MyLibraryTest.groovy:

groovy

Copy code

package com.example

import org.junit.Test

import static org.junit.Assert.\*

class MyLibraryTest {

@Test

void testGreet() {

String result = MyLibrary.greet("World")

assertEquals("Hello, World!", result)

}

}

**5. Building and Testing**

To compile and test the project, you can use the following Maven commands:

bash

Copy code

# Compile the project

mvn clean compile

# Run tests

mvn test

# Package the shared library as a JAR

mvn package

This project setup will allow you to build a shared Groovy library and make it available to other projects as a Maven dependency.

4o

**You said:**

1. Jenkins Pipeline: o Action: Initiates the build process by invoking the shared library method setupBuildTool. o Inputs Provided: ♣ toolName (e.g., maven) ♣ toolVersion (e.g., 3.8.7) ♣ downloadUrl (URL to download the tool from Artifactory) ♣ toolbinpath(where the tool or exe present in downloaded zip) 2. Shared Library Method (setupBuildTool): o Purpose: Manages the verification, downloading, setup, and path configuration of the specified build tool. o Inputs: ♣ toolName: Name of the build tool (e.g., maven, node) ♣ toolVersion: Specific version required ♣ downloadUrl: Artifactory URL for downloading the tool ♣ toolbinpath(where the tool or exe present in downloaded zip) 3. Detect Operating System: o Function: Determines whether the Jenkins agent is running on Windows or Linux. o Implementation: Utilize Jenkins environment variables or Groovy's System properties to detect the OS. o Purpose: Ensures that subsequent steps are executed with OS-specific commands and paths. 4. Path Setup (Windows vs. Linux): o Windows: ♣ Central Tools Directory: Typically C:\Jenkins\tools\ o Linux: ♣ Central Tools Directory: /opt/jenkins/tools/ 5. Check Central Tools Directory: o Function: Verifies if the specified tool and version already exist in the central directory. o Decision Points: ♣ Yes: Tool exists (/opt/jenkins/tools/${toolName}-${toolVersion+toolBinPath}. ♣ No: Tool does not exist and needs to be downloaded 6. If Tool Exists: o Return Existing Tool Path: ♣ Windows Example: C:\Jenkins\tools\maven-3.8.7\bin\mvn.exe ♣ Linux Example: /opt/jenkins/tools/maven-3.8.7/bin/mvn o Create Symlink in Workspace: ♣ Windows: Use mklink to create a directory junction or symbolic link. o Update PATH Environment Variable: ♣ Windows: ♣ env.PATH = "C:\\Jenkins\\workspace\\tools\\${toolName}-${toolVersion}\\bin;${env.PATH}" ♣ Linux: env.PATH = "${workspaceToolPath}:${env.PATH}" o Output: Returns the path to the tool's bin directory for use in subsequent build stages. 7. If Tool Does Not Exist: o Download Tool from Artifactory: ♣ Windows: ♣ Use PowerShell or curl to download. ♣ Invoke-WebRequest -Uri $downloadUrl -OutFile "C:\Jenkins\tools\${toolName}-${toolVersion}.zip" ♣ Linux: ♣ Use curl or wget to download. ♣ curl -L -o /opt/jenkins/tools/${toolName}-${toolVersion}.zip ${downloadUrl} o Uncompress Tool Archive: ♣ Windows: ♣ Use PowerShell's Expand-Archive or a tool like 7-Zip. ♣ Expand-Archive -Path "C:\Jenkins\tools\${toolName}-${toolVersion}.zip" -DestinationPath "C:\Jenkins\tools\${toolName}-${toolVersion}" ♣ Linux: ♣ Use unzip or tar based on archive type. ♣ unzip /opt/jenkins/tools/${toolName}-${toolVersion}.zip -d /opt/jenkins/tools/${toolName}-${toolVersion} o Set Executable Permissions (If Necessary): ♣ Windows: Generally not required, but ensure executables are accessible. ♣ Linux: chmod +x /opt/jenkins/tools/${toolName}-${toolVersion}/bin/\* o Create Symlink in Workspace: (Same as in "If Tool Exists" step) o Update PATH Environment Variable: (Same as in "If Tool Exists" step) o Output: Returns the path to the tool's bin directory. 8. Return Build Tool Path to Pipeline: o Windows Example: C:\Jenkins\workspace\tools\maven-3.8.7\bin\ o Linux Example: /workspace/tools/maven-3.8.7/bin/

**ChatGPT said:**

ChatGPT

Here's how you can implement a Jenkins Pipeline and a shared library function (setupBuildTool) to meet these requirements.

**Jenkins Pipeline Script**

The following Jenkins Pipeline script initiates the build process by calling the setupBuildTool method from the shared library, passing the required parameters like toolName, toolVersion, downloadUrl, and toolBinPath.

groovy

Copy code

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

pipeline {

agent any

environment {

TOOL\_NAME = 'maven'

TOOL\_VERSION = '3.8.7'

DOWNLOAD\_URL = 'https://artifactory.example.com/path/to/maven-3.8.7.zip'

TOOL\_BIN\_PATH = 'bin'

}

stages {

stage('Setup Build Tool') {

steps {

script {

def toolPath = setupBuildTool(

toolName: env.TOOL\_NAME,

toolVersion: env.TOOL\_VERSION,

downloadUrl: env.DOWNLOAD\_URL,

toolbinpath: env.TOOL\_BIN\_PATH

)

echo "Tool setup at path: ${toolPath}"

}

}

}

}

}

**Shared Library Script (setupBuildTool)**

The setupBuildTool method checks the OS, verifies the tool's presence, downloads it if necessary, and configures the environment path.

groovy

Copy code

def call(Map config) {

def toolName = config.toolName

def toolVersion = config.toolVersion

def downloadUrl = config.downloadUrl

def toolBinPath = config.toolbinpath

def isWindows = isWindowsOS()

def toolsDir = isWindows ? "C:\\Jenkins\\tools\\" : "/opt/jenkins/tools/"

def toolDir = "${toolsDir}${toolName}-${toolVersion}"

def toolPath = "${toolDir}/${toolBinPath}"

// Check if the tool exists

if (fileExists(toolPath)) {

echo "Tool already exists at: ${toolPath}"

} else {

// Download and setup the tool

downloadAndSetupTool(toolName, toolVersion, downloadUrl, toolsDir, isWindows)

}

// Create symlink in workspace and update PATH

createSymlink(toolDir, isWindows)

env.PATH = isWindows ? "C:\\Jenkins\\workspace\\tools\\${toolName}-${toolVersion}\\${toolBinPath};${env.PATH}" :

"${toolPath}:${env.PATH}"

return toolPath

}

def isWindowsOS() {

return System.getProperty("os.name").toLowerCase().contains("windows")

}

def downloadAndSetupTool(String toolName, String toolVersion, String downloadUrl, String toolsDir, boolean isWindows) {

def toolZip = "${toolsDir}${toolName}-${toolVersion}.zip"

if (isWindows) {

powershell """

Invoke-WebRequest -Uri '${downloadUrl}' -OutFile '${toolZip}'

Expand-Archive -Path '${toolZip}' -DestinationPath '${toolsDir}${toolName}-${toolVersion}'

"""

} else {

sh """

curl -L -o '${toolZip}' '${downloadUrl}'

unzip '${toolZip}' -d '${toolsDir}${toolName}-${toolVersion}'

chmod +x '${toolsDir}${toolName}-${toolVersion}/bin/\*'

"""

}

}

def createSymlink(String toolDir, boolean isWindows) {

if (isWindows) {

powershell """

if (!(Test-Path -Path "C:\\Jenkins\\workspace\\tools")) {

New-Item -ItemType Directory -Force -Path "C:\\Jenkins\\workspace\\tools"

}

if (!(Test-Path -Path "C:\\Jenkins\\workspace\\tools\\${toolDir}")) {

cmd /c mklink /D "C:\\Jenkins\\workspace\\tools\\${toolDir}" "${toolDir}"

}

"""

} else {

sh """

mkdir -p "${env.WORKSPACE}/tools"

if [ ! -L "${env.WORKSPACE}/tools/${toolDir}" ]; then

ln -s "${toolDir}" "${env.WORKSPACE}/tools/"

fi

"""

}

}

**Explanation of setupBuildTool**

1. **Check OS:** The isWindowsOS method checks the operating system by reading the system properties.
2. **Set Directory Paths:** Based on the OS, it sets toolsDir, the central tools directory path.
3. **Verify Tool Existence:** It checks if the specified version of the tool is already installed in the tools directory. If it exists, it skips the download.
4. **Download and Extract Tool:**
   * If the tool doesn’t exist, it downloads the .zip from Artifactory.
   * For Windows, PowerShell is used to download and expand the archive. For Linux, curl and unzip are used.
5. **Create Symlink:** The method creates a symlink in the Jenkins workspace, making it easier to access the tool.
6. **Update PATH Variable:** Updates the PATH environment variable to include the tool’s bin directory.

This setup gives flexibility for various operating systems and ensures tools are only downloaded if necessary.

4o

**You said:**

please write this code as structured and as professional developer

**ChatGPT said:**

ChatGPT

To make the code more structured, I’ll break it into reusable methods with clear separation of concerns, provide descriptive comments, and format it to improve readability. Here’s the refactored version of the Jenkins shared library function setupBuildTool, along with the pipeline script.

**Jenkins Pipeline Script**

This pipeline script is straightforward, invoking the setupBuildTool method with configuration parameters. It’s designed to demonstrate usage in a CI/CD process.

groovy

Copy code

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

pipeline {

agent any

environment {

TOOL\_NAME = 'maven'

TOOL\_VERSION = '3.8.7'

DOWNLOAD\_URL = 'https://artifactory.example.com/path/to/maven-3.8.7.zip'

TOOL\_BIN\_PATH = 'bin'

}

stages {

stage('Setup Build Tool') {

steps {

script {

def toolPath = setupBuildTool(

toolName: env.TOOL\_NAME,

toolVersion: env.TOOL\_VERSION,

downloadUrl: env.DOWNLOAD\_URL,

toolBinPath: env.TOOL\_BIN\_PATH

)

echo "Build tool setup complete. Tool path: ${toolPath}"

}

}

}

}

}

**Shared Library Script (setupBuildTool)**

The function has been organized into helper methods, each handling a distinct task to simplify testing, maintainability, and readability.

groovy

Copy code

def call(Map config) {

def toolName = config.toolName

def toolVersion = config.toolVersion

def downloadUrl = config.downloadUrl

def toolBinPath = config.toolBinPath

def isWindows = detectOS()

def toolsDir = isWindows ? "C:\\Jenkins\\tools\\" : "/opt/jenkins/tools/"

def toolDir = "${toolsDir}${toolName}-${toolVersion}"

def toolPath = "${toolDir}/${toolBinPath}"

// Ensure tool is available or download and install it

if (toolExists(toolPath)) {

echo "Tool ${toolName} version ${toolVersion} already exists at ${toolPath}."

} else {

downloadAndInstallTool(toolName, toolVersion, downloadUrl, toolsDir, isWindows)

}

// Create symlink in workspace and configure PATH

configureWorkspaceSymlink(toolDir, isWindows)

updateEnvironmentPath(toolPath, isWindows)

// Return tool path for further use

return toolPath

}

/\*\*

\* Detects the operating system (Windows or Linux) using system properties.

\* @return true if Windows, false otherwise (Linux assumed).

\*/

def detectOS() {

return System.getProperty("os.name").toLowerCase().contains("windows")

}

/\*\*

\* Checks if the specified tool already exists at the given path.

\* @param toolPath Path to the tool's bin directory.

\* @return true if the tool exists, false otherwise.

\*/

def toolExists(String toolPath) {

return fileExists(toolPath)

}

/\*\*

\* Downloads and installs the specified tool from the given URL.

\* @param toolName Name of the tool.

\* @param toolVersion Tool version.

\* @param downloadUrl URL for downloading the tool.

\* @param toolsDir Directory to install the tool.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def downloadAndInstallTool(String toolName, String toolVersion, String downloadUrl, String toolsDir, boolean isWindows) {

def toolZip = "${toolsDir}${toolName}-${toolVersion}.zip"

if (isWindows) {

powershell """

Write-Host "Downloading ${toolName} version ${toolVersion} from ${downloadUrl}"

Invoke-WebRequest -Uri '${downloadUrl}' -OutFile '${toolZip}'

Write-Host "Extracting ${toolZip} to ${toolsDir}${toolName}-${toolVersion}"

Expand-Archive -Path '${toolZip}' -DestinationPath '${toolsDir}${toolName}-${toolVersion}'

"""

} else {

sh """

echo "Downloading ${toolName} version ${toolVersion} from ${downloadUrl}"

curl -L -o '${toolZip}' '${downloadUrl}'

echo "Extracting ${toolZip} to ${toolsDir}${toolName}-${toolVersion}"

unzip '${toolZip}' -d '${toolsDir}${toolName}-${toolVersion}'

chmod +x '${toolsDir}${toolName}-${toolVersion}/bin/\*'

"""

}

}

/\*\*

\* Creates a symbolic link in the workspace for the installed tool.

\* @param toolDir Path to the installed tool's directory.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def configureWorkspaceSymlink(String toolDir, boolean isWindows) {

def workspaceToolPath = isWindows ? "C:\\Jenkins\\workspace\\tools" : "${env.WORKSPACE}/tools"

def symlinkPath = "${workspaceToolPath}/${toolDir}"

if (isWindows) {

powershell """

if (!(Test-Path -Path "${workspaceToolPath}")) {

New-Item -ItemType Directory -Force -Path "${workspaceToolPath}"

}

if (!(Test-Path -Path "${symlinkPath}")) {

cmd /c mklink /D "${symlinkPath}" "${toolDir}"

}

"""

} else {

sh """

mkdir -p "${workspaceToolPath}"

if [ ! -L "${symlinkPath}" ]; then

ln -s "${toolDir}" "${workspaceToolPath}/"

fi

"""

}

}

/\*\*

\* Updates the PATH environment variable to include the tool's bin directory.

\* @param toolPath Path to the tool's bin directory.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def updateEnvironmentPath(String toolPath, boolean isWindows) {

if (isWindows) {

env.PATH = "C:\\Jenkins\\workspace\\tools\\${toolPath};${env.PATH}"

} else {

env.PATH = "${toolPath}:${env.PATH}"

}

}

**Explanation of Code Organization**

* **detectOS**: Determines if the agent is on Windows or Linux, allowing OS-specific commands.
* **toolExists**: Checks if the required tool and version are already available at the specified path.
* **downloadAndInstallTool**: Downloads and extracts the tool if it doesn't already exist. It uses PowerShell commands on Windows and curl and unzip on Linux.
* **configureWorkspaceSymlink**: Sets up a symbolic link in the workspace directory, making the tool easily accessible. On Windows, it uses mklink; on Linux, it uses ln -s.
* **updateEnvironmentPath**: Updates the PATH environment variable to include the tool’s bin directory.

This code provides a clean, organized approach to setup, download, and configuration based on OS. The pipeline and library script work together to ensure the tool is prepared correctly, allowing Jenkins to use it in subsequent stages.

4o

**You said:**

instead of direct quires in powershell and unix, is there anyway right it using groovy code

**ChatGPT said:**

ChatGPT

Yes, you can use Groovy’s built-in file manipulation and process execution capabilities to replace direct shell and PowerShell commands. Groovy provides methods to handle files, download URLs, and create symbolic links programmatically. Here’s how you can adjust your setupBuildTool method using Groovy code instead of inline PowerShell and shell commands.

Here's the refactored code that uses Groovy for OS operations:

**Updated setupBuildTool Using Groovy**

groovy

Copy code

import groovy.io.FileType

def call(Map config) {

def toolName = config.toolName

def toolVersion = config.toolVersion

def downloadUrl = config.downloadUrl

def toolBinPath = config.toolBinPath

def isWindows = detectOS()

def toolsDir = isWindows ? "C:\\Jenkins\\tools\\" : "/opt/jenkins/tools/"

def toolDir = "${toolsDir}${toolName}-${toolVersion}"

def toolPath = "${toolDir}/${toolBinPath}"

// Ensure tool is available or download and install it

if (toolExists(toolPath)) {

echo "Tool ${toolName} version ${toolVersion} already exists at ${toolPath}."

} else {

downloadAndInstallTool(toolName, toolVersion, downloadUrl, toolsDir, isWindows)

}

// Create symlink in workspace and configure PATH

configureWorkspaceSymlink(toolDir, isWindows)

updateEnvironmentPath(toolPath, isWindows)

// Return tool path for further use

return toolPath

}

/\*\*

\* Detects the operating system (Windows or Linux) using system properties.

\* @return true if Windows, false otherwise (Linux assumed).

\*/

def detectOS() {

return System.getProperty("os.name").toLowerCase().contains("windows")

}

/\*\*

\* Checks if the specified tool already exists at the given path.

\* @param toolPath Path to the tool's bin directory.

\* @return true if the tool exists, false otherwise.

\*/

def toolExists(String toolPath) {

return new File(toolPath).exists()

}

/\*\*

\* Downloads and installs the specified tool from the given URL.

\* @param toolName Name of the tool.

\* @param toolVersion Tool version.

\* @param downloadUrl URL for downloading the tool.

\* @param toolsDir Directory to install the tool.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def downloadAndInstallTool(String toolName, String toolVersion, String downloadUrl, String toolsDir, boolean isWindows) {

def toolZip = new File("${toolsDir}${toolName}-${toolVersion}.zip")

def toolDir = new File("${toolsDir}${toolName}-${toolVersion}")

// Ensure tools directory exists

if (!toolDir.parentFile.exists()) {

toolDir.parentFile.mkdirs()

}

// Download the tool

def url = new URL(downloadUrl)

url.withInputStream { i ->

toolZip.withOutputStream { it << i }

}

echo "Downloaded ${toolName} version ${toolVersion} to ${toolZip}"

// Unzip the downloaded file

if (isWindows) {

unzipWindows(toolZip, toolDir)

} else {

unzipLinux(toolZip, toolDir)

}

}

/\*\*

\* Unzips a file on Windows.

\* @param zipFile The zip file to unzip.

\* @param destDir The destination directory for the unzipped content.

\*/

def unzipWindows(File zipFile, File destDir) {

def zip = new java.util.zip.ZipFile(zipFile)

zip.entries().each { entry ->

def entryFile = new File(destDir, entry.name)

if (entry.isDirectory()) {

entryFile.mkdirs()

} else {

entryFile.parentFile.mkdirs()

entryFile.withOutputStream { out ->

zip.getInputStream(entry).withStream { it -> out << it }

}

}

}

zip.close()

echo "Unzipped ${zipFile} to ${destDir}"

}

/\*\*

\* Unzips a file on Linux.

\* @param zipFile The zip file to unzip.

\* @param destDir The destination directory for the unzipped content.

\*/

def unzipLinux(File zipFile, File destDir) {

def ant = new AntBuilder()

ant.unzip(src: zipFile.absolutePath, dest: destDir.absolutePath, overwrite: "true")

echo "Unzipped ${zipFile} to ${destDir}"

}

/\*\*

\* Creates a symbolic link in the workspace for the installed tool.

\* @param toolDir Path to the installed tool's directory.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def configureWorkspaceSymlink(String toolDir, boolean isWindows) {

def workspaceToolDir = isWindows ? new File("C:\\Jenkins\\workspace\\tools") : new File("${env.WORKSPACE}/tools")

def symlinkPath = new File(workspaceToolDir, toolDir.name)

if (!workspaceToolDir.exists()) {

workspaceToolDir.mkdirs()

}

if (!symlinkPath.exists()) {

if (isWindows) {

def command = "cmd /c mklink /D \"${symlinkPath}\" \"${toolDir}\""

def proc = command.execute()

proc.waitFor()

echo "Created symlink at ${symlinkPath} for ${toolDir}"

} else {

symlinkPath.parentFile.mkdirs()

symlinkPath.withOutputStream { it << toolDir.toURI().toURL().openStream() }

echo "Created symlink at ${symlinkPath} for ${toolDir}"

}

}

}

/\*\*

\* Updates the PATH environment variable to include the tool's bin directory.

\* @param toolPath Path to the tool's bin directory.

\* @param isWindows Boolean indicating if the OS is Windows.

\*/

def updateEnvironmentPath(String toolPath, boolean isWindows) {

env.PATH = isWindows ? "C:\\Jenkins\\workspace\\tools\\${toolPath};${env.PATH}" :

"${toolPath}:${env.PATH}"

}

**Explanation of Changes**

* **downloadAndInstallTool**: Downloads the zip file from the provided URL using Groovy’s URL class and unzips it with unzipWindows or unzipLinux methods, depending on the OS.
* **unzipWindows and unzipLinux**: These methods handle extracting zip files on each platform. The Windows unzip function uses java.util.zip.ZipFile to extract files, while the Linux version leverages Groovy's AntBuilder to run an Ant unzip task, which works cross-platform for Linux.
* **configureWorkspaceSymlink**: Creates a symbolic link in the workspace directory for easier access to the tool. On Windows, it executes a command with mklink for symlinks, while on Linux, it uses Groovy file operations to create a link.
* **updateEnvironmentPath**: Sets the PATH environment variable using the path to the tool’s bin directory. It ensures the pipeline uses the tool in subsequent steps without the need to manually update the system PATH.

Using Groovy’s capabilities this way helps avoid inline OS-specific commands and provides a cleaner, more maintainable solution.