**Centralized Tool Setup for Jenkins Agents**

1. **JSON Configuration**:
   * Tools (name, version, binary paths) are defined in a JSON file.
   * Example structure:

{

"tools": [

{ "name": "maven", "version": "3.8.1", "binPath": "/bin/mvn", "windowsExe": "mvn.exe" },

{ "name": "gradle", "version": "7.2", "binPath": "/bin/gradle", "windowsExe": "gradle.exe" }

]

}

1. **Groovy Script**:
   * Parse the JSON to get tool details (name, version, binPath).
   * Based on OS (Linux/Windows), append correct bin path or .exe.
2. **Pipeline Integration**:
   * Detect OS using isUnix().
   * Set tool path dynamically based on the tool's binary/executable path.
   * Example paths:
     + **Linux**: /opt/jenkins/tools/<tool>/<version>/bin
     + **Windows**: C:\\JenkinsTools\\<tool>\\<version>\\bin
3. **Avoid Redownloads**:
   * Check if the tool exists in the centralized location.
   * Download from Artifactory if not found, then extract and store.
4. **Reuse Across Jobs**:
   * Centralized tool location ensures tools are reused across Jenkins jobs without redownloading.
5. **Cross-Platform Support**:
   * Automatically handles both Linux and Windows paths, adding .exe for Windows tools.

**Centralized Tool Setup Using Key-Value Metadata**

1. **Key-Value Metadata File (tool-metadata.properties)**:
   * Contains tool info (name, version, paths).
   * Example:

name=maven

version=3.8.1

binPath=/bin/mvn

windowsExe=mvn.exe

1. **Groovy Script**:
   * Parse tool-metadata.properties after extraction.
   * Extract key values like binPath (Linux) or windowsExe (Windows).
2. **Pipeline Steps**:
   * **Check for Tool**: Skip download if tool-metadata.properties exists.
   * **Set Tool Path**: Read metadata, set binPath (Linux) or windowsExe (Windows), and use it for the build.
3. **Cross-Platform Support**:
   * Automatically selects the correct executable path based on OS.
4. **Avoid Redownloads**:
   * Tools are cached centrally in Jenkins and only downloaded if missing.

**Tool Path Setup Using Jenkins Params (Downloaded Zip)**

1. **Jenkins Build Parameters**:
   * **TOOL\_EXE**: Name of the executable (e.g., mvn, gradle.exe).
   * **TOOL\_PATH** (optional): Directory path of the tool.
2. **Download and Extract Tool**:
   * Download tool from Artifactory if not already present.
   * Extract to a centralized location (e.g., /opt/jenkins/tools/ or C:\\JenkinsTools\\).
3. **Detect Executable Using Params**:
   * Use TOOL\_EXE to locate the executable in the extracted directory.
   * Set TOOL\_PATH dynamically:
   * def exeFile = isUnix() ? "${toolPath}/bin/${params.TOOL\_EXE}" : "${toolPath}\\${params.TOOL\_EXE}"
4. **Build Execution**:
   * Use detected TOOL\_PATH to run the build:
   * sh "${env.TOOL\_PATH} clean install" // or bat on Windows.

**High-Level Design: Tool Path Setup Using Naming Convention**

1. **Naming Convention**:
   * Use predefined names for executables:
     + **Maven**: mvn (Linux), mvn.exe (Windows).
     + **Gradle**: gradle (Linux), gradle.exe (Windows).
2. **Download and Extract**:
   * Download tool from Artifactory if not present.
   * Extract to:
     + **Linux**: /opt/jenkins/tools/<tool>/<version>.
     + **Windows**: C:\\JenkinsTools\\<tool>\\<version>.
3. **Find Executable**:
   * Construct path based on naming convention:
   * def exeFile = isUnix() ? "${toolDir}/bin/${toolName}" : "${toolDir}\\${toolName}.exe"
4. **Set Tool Path**:
   * Verify if the executable exists, set TOOL\_PATH:
   * env.TOOL\_PATH = exeFile
5. **Build Execution**:
   * Run the build using TOOL\_PATH:
   * sh "${env.TOOL\_PATH} clean install" // or bat on Windows