Jenkins pipeline script

Jenkins and Banboo are widely used for Continuous Integration and Continuous Delivery. Since Jenkins 2.0+, it takes most of markets because it supports automation script. Before 2.0, Jenkins is purely a tool. It provides the UI to configure the workflow by pre and post actions. Now, it supports groovy script to build the workflow.

The best practice of Jenkins CD/CI is from the concrete projects. Each project implements its own delivery process/steps. Later on, Devops extracts the common portion of the functions into a so-called global libraries. then the new project can leverage the common libraries to define the workflow.

The global library is also called extended shared library. Here is [link](https://jenkins.io/doc/book/pipeline/shared-libraries/) containing the detail.

1. Create the folder structure to contain the scripts
2. Understand what inside of the folders.
   1. Src/ folder to define the lib class file likes java classes
   2. Vars/ folder to define the static method, static variable/step
   3. Resources/ folder to define the properties file
3. Declare the global lib into Jenkins

Here, we focus on how to define global static method and variables/steps because most of problems can be resolved by them. Defining in src/ folder is more difficult than static things like in java. Unless we like to store the workflow status to support pause or resume the workflow, the rest problems can be solved by static method and variables/steps.

. function in class under src/ folder

@Library(‘somelib’)

Import com.mycorp.pipeline.UsefulClass

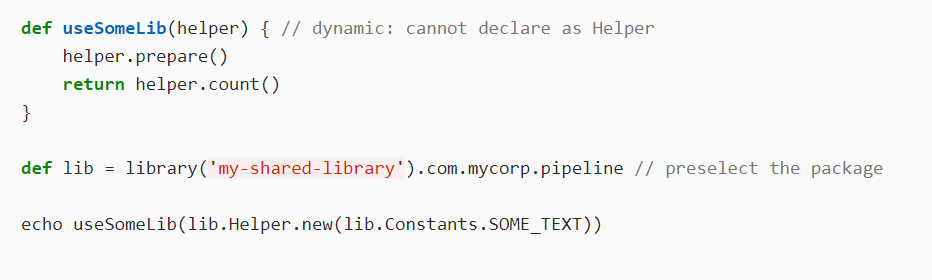
. shared variable in vars/ folder

@Library(‘my-shared-lib’) \_

Note: no import need (seems import will initialize the class)

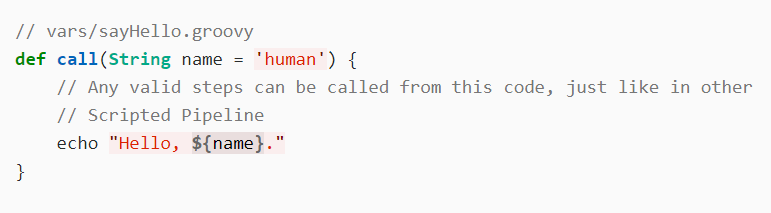
. dynamically load library

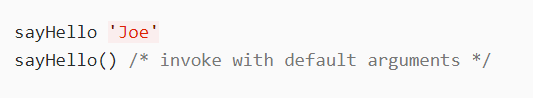
It only happens when the class is dynamically passed in as parameter



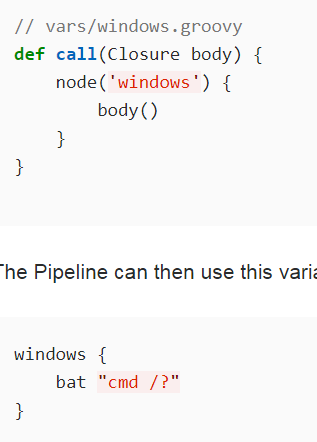
. define steps under vars/ folder

a global variable has a call method implemented. The global variable to be invoked in a manner similar to a Jenkins built-in step.





Second example:



### Defining a more structured DSL

If you have a lot of Pipelines that are mostly similar, the global variable mechanism provides a handy tool to build a higher-level DSL that captures the similarity. For example, all Jenkins plugins are built and tested in the same way, so we might write a step named buildPlugin:

// vars/buildPlugin.groovy

**def** **call**(body) {

// evaluate the body block, and collect configuration into the object

**def** config = [:]

body.resolveStrategy = Closure.DELEGATE\_FIRST

body.delegate = config

body()

// now build, based on the configuration provided

node {

git url: "https://github.com/jenkinsci/**${**config.name**}**-plugin.git"

sh "mvn install"

mail to: "...", subject: "**${**config.name**}** plugin build", body: "..."

}

}

Assuming the script has either been loaded as a [Global Shared Library](https://jenkins.io/doc/book/pipeline/shared-libraries/#global-shared-libraries) or as a [Folder-level Shared Library](https://jenkins.io/doc/book/pipeline/shared-libraries/#folder-level-shared-libraries) the resulting Jenkinsfile will be dramatically simpler:

*Jenkinsfile (Scripted Pipeline)*

buildPlugin {

name = 'git'

}