Automating WSO2 Releases with Continuous Delivery via Jenkins

Design & Configuration Guide

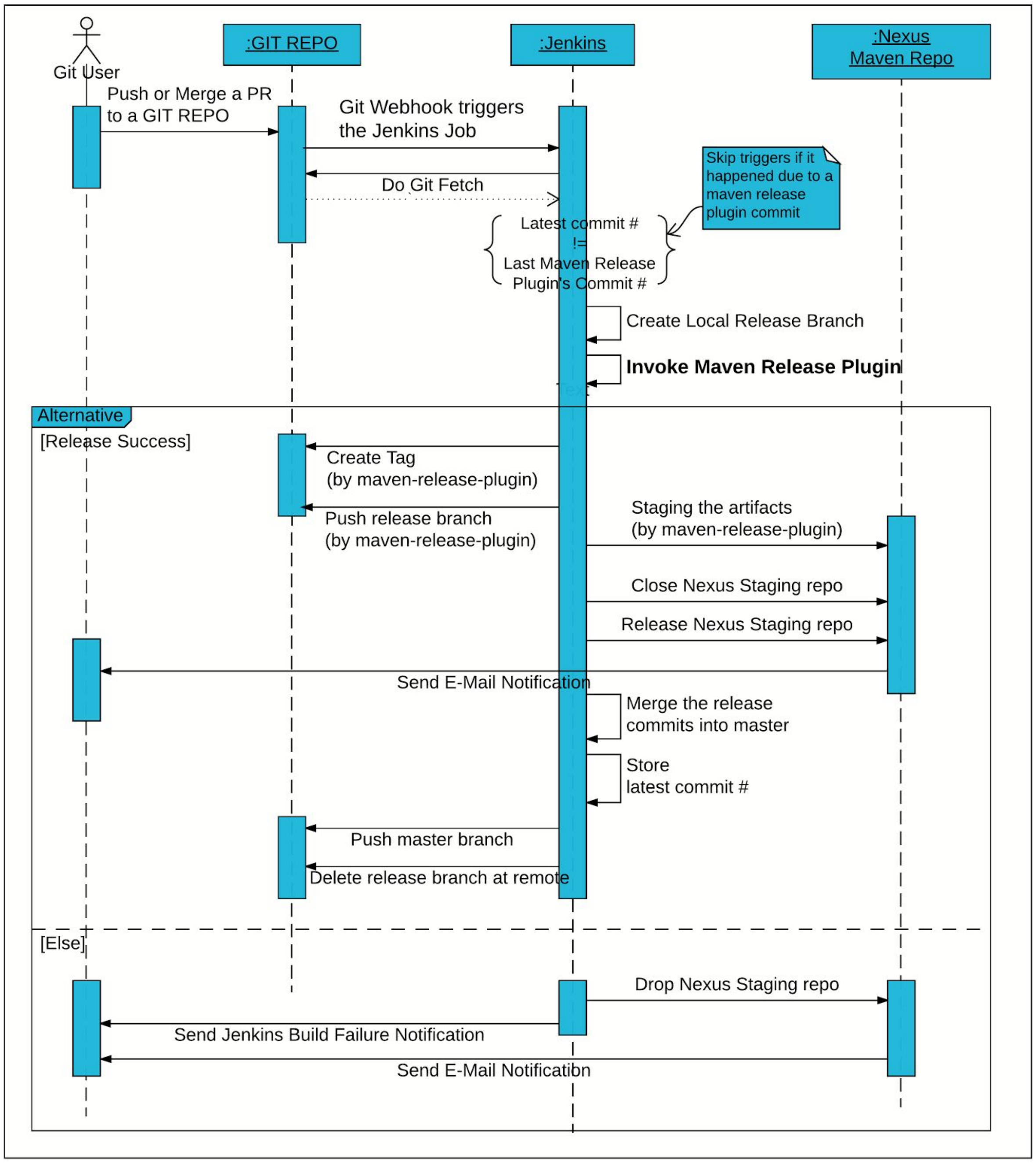
WSO2 Continuous Delivery Sequence

This document explain the effort went into automatically doing a release after each push by incrementing the micro part of the version number. This is one more step towards CI/CD. So what needs to happen is, Jenkins should automatically take over and do the release & push to WSO2 Nexus.

The rationale behind this is, developers feel guilty to increment version numbers and this leads to a long release train being created when a product has to be released. So we will follow the feature branch approach and when the feature in completed, we will merge to the master, and the release will happen. It could be even bug fixes which are pushed to GitHub which will trigger the automatic release.

The next step would be if some product or another component is using a WSO2 dependency which is old, a notification will be sent to the repo owners and engineering list saying that newer versions are available.

Following is the sequence diagram of the process for this flow. It shows the interactions of Git users, Git repos, Jenkins and Nexus. The process was implemented via a Jenkins plugin.


Git User
:GIT REPO
:Jenkins
:Nexus Maven Repo
Push or Merge a PR to a GIT REPO
Git Webhook triggers the Jenkins Job
Do Git Fetch
Skip triggers if it happened due to a maven release plugin commit
Latest commit # != Last Maven Release Plugin's Commit #
Create Local Release Branch
Invoke Maven Release Plugin
Alternative
[Release Success]
Create Tag (by maven-release-plugin)
Push release branch (by maven-release-plugin)
Staging the artifacts (by maven-release-plugin)
Close Nexus Staging repo
Release Nexus Staging repo
Send E-Mail Notification
Merge the release commits into master
Store latest commit #
Push master branch
Delete release branch at remote
[Else]
Drop Nexus Staging repo
Send Jenkins Build Failure Notification
Send E-Mail Notification

Raw link of the image for later modifications:
https://www.lucidchart.com/invitations/accept/41dd6dc5-b787-486d-8f02-6034ee87032b

Configuration Guide

Per Job Configuration

Following need to be done for each job that moves into CD.

Reference: https://wso2.org/jenkins/job/carbon-caching/configure

1. Source Code Management -> Git:
   a. Set credentials - (wso2-jenkins-bot - github user for maven release plugin)
   b. Add ‘Additional behaviors’:
      i. Wipe out repository and force clone

2. Set build trigger.

   ADD : Build when a change is pushed to GitHub
   REMOVE : Poll SCM

3. Tick the following in “Build Environment” ->
   a. Mask password - To mask the GitHub passwords from build logs.
   b. Maven Release Build - To enable maven-release-plugin and its goals.
      i. CHANGE Release goal:
      “release:clean release:prepare release:perform ***-P wso2-release***”
      (Since the *wso2-release* profile has the needed plugins configured such as the gpg-plugin, we need to specify the profile explicitly.)

4. Disable nexus snapshot deployment if it is a release build.
   a. Post Build Actions -> Deploy artifacts to Maven repository -> “**Release environment variable**” -> Set this to **IS_M2RELEASEBUILD**

5. (optional) Add a Quiet period of **10 minutes** - If someone need to do one release for multiple PRs, then this gives a window of 10 minutes to merge all the needed PRs.

Needed Jenkins Plugins

(not complete)

GitHub Plugin
Mask Passwords Plugin
Customized M2Release Plugin

Global configuration

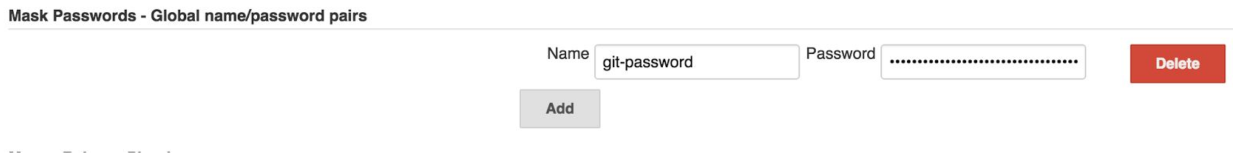
These are one-time configurations that needs to be done. These configs are already done in production.

1. Make sure keystore config is correct.
    a. This is configured through catalina.sh

2. Install m2release-plugin [1] and github-plugin.
   M2Release-Plugin needs to be built from -
https://github.com/wso2/wso2-jenkins-m2release-plugin

3. Enable Nexus Pro Support
    a. https://maven.wso2.org/nexus/
    b. Credentials - https://docs.google.com/document/d/1wznAJVK95_oQS3SEfpKY_803Br-dwMo8pYwZ-Uz7Z08/edit#
4. Install the mask passwords plugin. Otherwise, the passwords are visible in the build logs.

    a. Global -> Set the passwords to mask
        i. Tick “Non-stored password”, “**Text**”, “**String**”, “**Password**”.
        ii. Give the password to mask. Choose any name.
        iii. The github user-password and Nexus user-password needs to be masked. Otherwise, these appear in the logs!!
    (This is set for git password)


Mask Passwords - Global name/password pairs

Name: git-password
Password: ••••••••••••••••••••••••••••••••••

5. Add settings.xml via **managed scripts**.
    a. You only need to configure the new servers - nexus-releases and my-scm-server.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <localRepository>/build/m2repos/repository/</localRepository>
  <pluginGroups>
  </pluginGroups>
   <proxies>

  </proxies>

  <servers>
    <!-- The passwords should be picked from jenkins server credentials. -KasunG  -->
    <server>
      <id>wso2.snapshots</id>
      <username>wso2</username>
      <password></password>
    </server>

    <server>
      <id>wso2.releases</id>
      <username>wso2</username>
      <password></password>
    </server>
    <server>

<id>nexus-releases</id>
  <username>kasunbg</username>
  <password></password>
</server>

    <server>
    <id>my-scm-server</id>
    <username>wso2</username>
      <password>xyz</password>
</server>
   </servers>
</settings>
```

6. Let Jenkins manage the GitHub Webhooks. To do that, add a new GitHub Server under the GitHub section -
    a. API URL: https://api.github.com
    b. Set credentials
    c. Tick Manage hooks

Other configurations

1. Configure GitHub WebHook *for each repo*.
    a. Login to GitHub, and head over to Settings -> **Webhooks & services**
    b. Click on Add Webhook
    c. Set the Payload URL: **https://wso2.org/jenkins/github-webhook/**
    d. Click on Add Webhook

NOTE: No need to do this if Jenkins is automatically managing the webhooks.

2. Add **wso2-jenkins-bot** github user to all the repos. -
   NOTE: This user has access to all the repos inside *‘wso2’* GitHub organization. Need to add it to other organizations such as wso2-extensions.

means the change is DONE in production Jenkins.

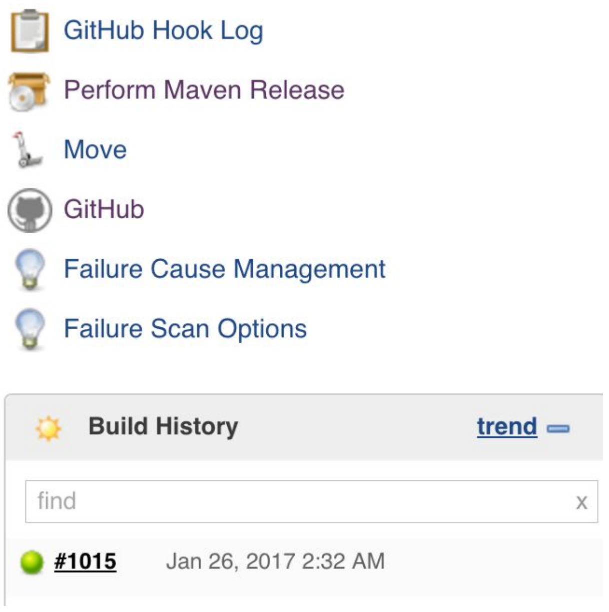
Triggering a Manual Release via Jenkins

Let’s suppose CD is not enabled for a repo. But still, if you like to use Jenkins to do the releases for you, then it indeed is possible. :-) What happens then is that you make your Jenkins job CD-ready, but the release per each push option is set to disable.

Steps -

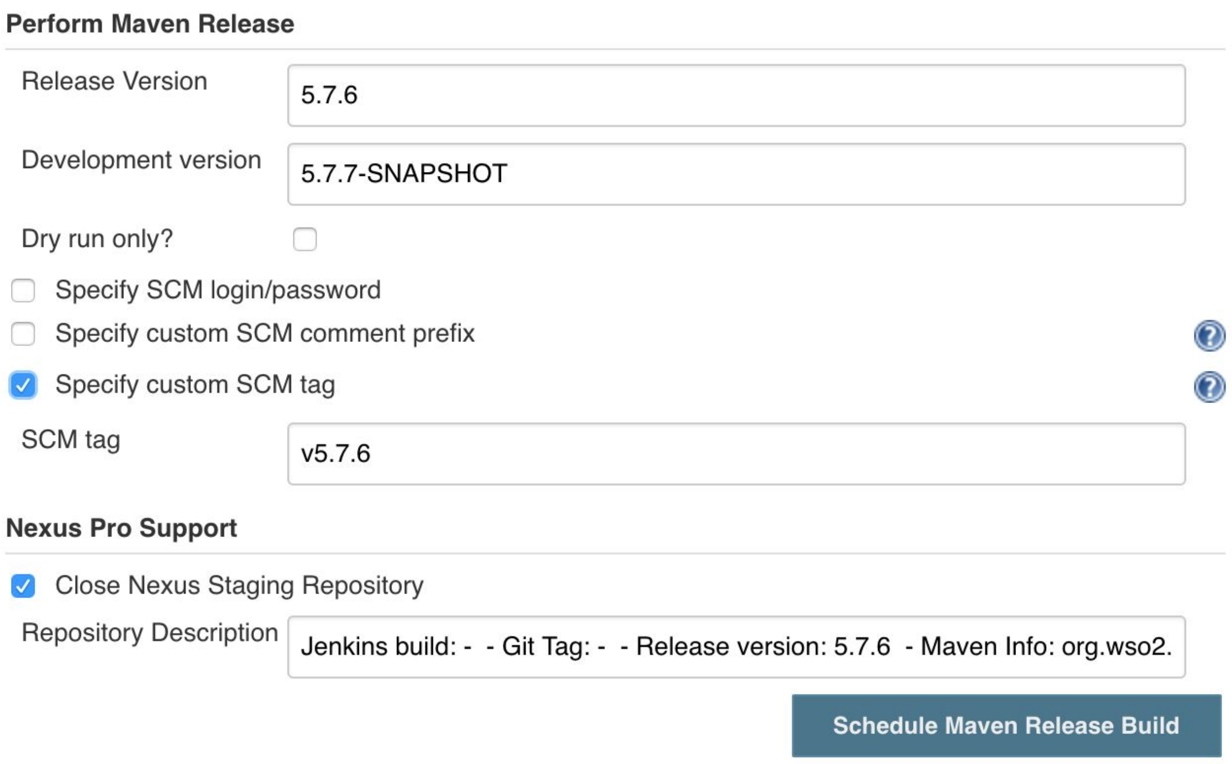
1. Follow the steps mentioned in “Per Job Configuration” section.
2. Modify the build triggers of the job’s configuration as follows.
   REMOVE : Build when a change is pushed to GitHub
   ADD : Poll SCM

Now, when you need to do a release, click on the “**Perform Maven Release**” option in the left menu. (If you do not need see this, then you may need to request the permission)



Now, in the “Perform Maven Release” page, provide the following (usual) stuff.

- Release Version
- Next Development Version
- SCM Tag name



Perform Maven Release

Release Version	5.7.6
Development version	5.7.7-SNAPSHOT
Dry run only?	☐

☐ Specify SCM login/password

☐ Specify custom SCM comment prefix

☑ Specify custom SCM tag

SCM tag	v5.7.6

Nexus Pro Support

☑ Close Nexus Staging Repository

Repository Description	Jenkins build: - - Git Tag: - - Release version: 5.7.6 - Maven Info: org.wso2.

Temporarily Disabling CD of a Job

Rarely, some repos may need to disable CD temporarily. In that case, you do not have to reverse entire steps mentioned above. You only need to reverse the steps mentioned in Step #2. Ie.

1. Modify the build triggers of the job’s configuration as follows.

   REMOVE : Build when a change is pushed to GitHub
   ADD : Poll SCM

The advantage here is that you can you can still perform manual releases via Jenkins itself even though releases will not happen for each git push.

Please note that there needs to be a good enough reason to do this.

How to Upload Custom wso2 m2release-plugin Plugin to Jenkins

1. Build https://github.com/wso2/wso2-jenkins-m2release-plugin
2. Grab target/m2release-plugin.hpi
3. Install the plugin as per https://wiki.jenkins-ci.org/display/JENKINS/Plugins#Plugins-Byhand . You may upload the plugin via UI as well.
4. Restart Jenkins via the link - https://wso2.org/jenkins/safeRestart

Rough work

- **Settings.xml** should have the credentials
- Tag version format should be v${project.version}
- Need to install the modified m2 release plugin at kasunbg repo
- Cleanup the local git checkout after the release
- Enable nexus pro support

- Install the mask passwords plugin
    - Per job “Build Environment” -> Mask Password
    - Global -> Set the passwords to mask
        - Mask the github and nexus username and passwords
- Add the new m2release.hpi

Creating a new repo

- Create the repo

git remote add legend https://github.com/kasunbg/legendary-journey-stg.git
git push -u legend master

- Change SCM
- Add wso2-jenkins-bot user to the repo.