Acumos Developer's Guide to CI-CD Resources and Processes at the LF

This is a guide for Acumos software developers and testers about the continuous integration/continuous deployment (CI/CD) resources and processes supported by the Linux Foundation.

CI/CD Network Resources

These resources require a Linux Foundation identity.

URL	Description
https://identity.linuxfoundation.org	The Linux Foundation identity management web site.
https://jira.linuxfoundation.org/servicedesk/customer/portals	The Linux Foundation release engineering team's service desk.
https://gerrit.acumos.org	The Gerrit code review server hosts the Git repositories and supports the review, merge and release processes. The review process basically requires all users to follow something like a conventional git pull-request process, but restricts the publication (push) of private branches.
https://jenkins.acumos.org	Jenkins is the continuous-integration server aka the build server. All users can see the contents of the Jenkins; no users can modify the configuration nor start a job in this Jenkins.
https://jenkins.acumos.org/sandbox	The Jenkins sandbox is a place for testing Jenkins job configurations. After requesting access, users can create jobs, reconfigure jobs, trigger builds etc. After JJB templates have been tested in the sandbox they can be submitted for the main Jenkins server.
https://jira.acumos.org	Jira is the web site that tracks issues, epics, stories, etc.
https://nexus.acumos.org	This Nexus 2 repository holds Maven (i.e., jar) artifacts produced by builds. Snapshot and staging builds are all deployed to this repository. Release artifacts are created by promoting artifacts manually from the staging repository after suitable approval. Publicly accessible to users without credentials. All users should be able to access and browse artifacts through this URL.
https://nexus3.acumos.org	This Nexus 3 server publishes external and internal Docker images. Supports the following registries: • Public mirror registry: nexus3.acumos.org:10001. This acts as a mirror of the public registry at docker.io. • Release registry: nexus3.acumos.org:10002. This is a VIEW offering access to the public registry mirror AND the Acumos releases (docker images promoted manually from the staging registry after suitable approval) • Snapshot registry: nexus3.acumos.org:10003. This contains docker images produced by some merge jobs, primarily Java projects. • Staging registry: nexus3.acumos.org:10004. This contains docker images produced by some merge jobs and also stage-release jobs. These registries are open for anonymous read access. The Jenkins server has credentials to push images to the snapshot and staging registries, and to promote images to the release registry. Manual push of images is not supported.

https://sonar.acumos.org	Sonar analyzes Java code for quality issues and code coverage achieved by unit tests (JUnit).

Development Procedures and Policies

This section guides the developer in submitting code, reviewing code, tracking the status of builds and requesting creation of release versions. These recommended practices come from the Linux Foundation.

Maven POM version numbers and dependencies

All Java project POM files should always use the suffix "-SNAPSHOT" in their project version strings, that's the "V" in the GAV triple defined at the top, at all times. This is slightly counter intuitive. The Jenkins jobs expect this and manage the version strings appropriately.

In contrast, Java project POM files should use release versions of dependencies (not SNAPSHOT versions). It may be necessary to use SNAPSHOT versions of dependencies for development, but that must be regarded as temporary. All releases given to the test team and published for use by the world MUST use only release versions.

A peculiarity of the LF build environment is that every Jenkins job that stages a release candidate strips all "-SNAPSHOT" suffixes within POM files. This affects the project's own version and all named dependencies.

Maven repositories and pluginRepositories URLs are managed externally

Per the LF, Java project POM files should have neither a "repositories" nor a "pluginRepositories" section. These sections can be used to store URLs of the servers that supply dependencies - jar and plugin files - other than the well-known Maven Central (central.maven.org) server. Those sections are provided on the LF Jenkins build server by a configured "settings.xml" file. Individual developers who build the software on their own machines must create a settings file at path ~/.m2/settings.xml (i.e., in their home directory where maven looks by default) with this information.

Always create a branch

When working with a git repository cloned from Gerrit you can create as many local branches as you like. Note that none of the branches will be pushed to the remote Gerrit server, the branches remain forever private. Creating branches for each task will allow you to work on multiple independent tasks in parallel, let you recover gracefully from various situations, and generally save aggravation and time. Also see these instructions on tagging and branching for releases:

Acumos Tagging and Branching Steps Process

Write good git commit messages

The Linux Foundation strongly recommends (and eventually will enforce) Gerrit commit message content. A git commit message must meet these requirements:

1. The first line should have a short message, up to 50 characters
2. The second line must be blank
3. The body of the commit message should have a detailed description of the change
4. The line before the footer must be blank
5. The footer (last block of lines following a blank line) must consist of these lines:
    a. Issue-ID: line with a valid Acumos Jira ticket number, composed and inserted manually by the committer
    b. Change-Id: line, which is automatically generated and inserted by git review
    c. Signed-off-by line, which is automatically generated and inserted by git commit

An example is shown below:

```
Null check for ClientResponse

NullPointerException might be thrown as cres is nullable here

Issue-ID: ACUMOS-999
Change-Id: I14dc792fb67198ebcbabfe80d90c48389af6cc91
Signed-off-by: First Last <fl1234567890@techmahindra.com>
```

Please note that the Jira system's pattern matcher that finds issue IDs in Gerrit reviews is extremely limited. The "Issue-ID" line must be EXACTLY as shown above! The keyword must be written in mixed case as shown, then a colon, then a space, then the word ACUMOS in ALL CAPS, then a hyphen, then the number.

Working with Docker images

The docker image name and tag are pulled from the project's pom.xml file. The development team must manage the version string in the POM, as noted above, always with a SNAPSHOT suffix. Please note that this same procedure ALSO applies to Python projects that use a standard JJB configuration for building a docker! Those projects should create a tiny POM file with just enough information for the docker image builder.

Creation of snapshot images happens on standard Jenkins verify and merge jobs. The Jenkins job sets the PUSH registry to SNAPSHOT.

Creation of staging (release candidate) images happens on standard Jenkins release jobs. To run that job, post the comment "stage-release" in a Gerrit review of that project. The Jenkins release job sets the PUSH registry to staging. The team does NOT have to fiddle the POM file to change the version suffix. There is some magic in the LF build job for release, it modifies the POM files to strip any "-SNAPSHOT" strings out!

Creation of a final release image requires opening a ticket with LF. The LF release engineering team signs the image and moves it manually.

Using Gerrit

Acumos uses Gerrit to automate the process of reviewing all code changes before they are committed to the Git repository. For a tutorial on git you might start here:

https://git-scm.com/doc

Here is a tutorial and reference on using Gerrit:

https://gerrit.acumos.org/r/Documentation/index.html

Also see this guide published by the LF Release Engineering team about using Gerrit:

https://docs.releng.linuxfoundation.org/en/latest/gerrit.html

Prohibited Content

Gerrit is designed to host text files – source code. It enforces a size threshold on every commit, the default limit is 5MB. Further the Linux Foundation prohibits committing binary files such as compiled executables, jar files, zip archives and so on. An exception is made for binary picture (image) files in GIF, JPG and PNG formats, but the size limit must still be honored.

Quickstart Guides

The quickstart guides below describe the command-line procedures for performing common tasks on Gerrit. The command-line tool "git review" is the most reliable and can be used on any platform. Windows users should install "Git Bash" to gain support for command-line git.

Quickstart: Create and submit a change for review

```
git checkout -b my-local-branch
# work work work
git commit -as
    # Write a commit message including a line "Issue-ID:" with Jira ticket nr
git review master
```

Quickstart: Revise your open gerrit review

```
git checkout my-local-branch
# revise revise revise
git commit -a --amend
    # Revise your message but do NOT alter the Change-Id: line
git review master
```

Quickstart: Revise any open gerrit review

```
# Look up the change number shown top-left in Gerrit web, here "999".
git review -d 999
# revise revise revise
git commit -a --amend
    # Check your message -- do NOT alter the Change-Id: line
git review --no-rebase master
```

Quickstart: Resolve a conflicted review

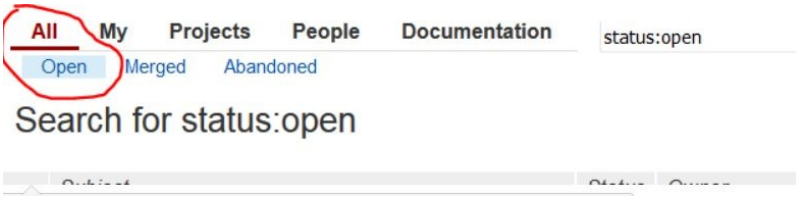
```
git checkout master
git pull origin master
# Look up the change number in Gerrit, it's shown top-left, here 999
git review -d 999
git rebase master
# fix conflicts
git add .
git rebase --continue
git review master
```

Quickstart: Squash commits after you forgot the "--amend" flag

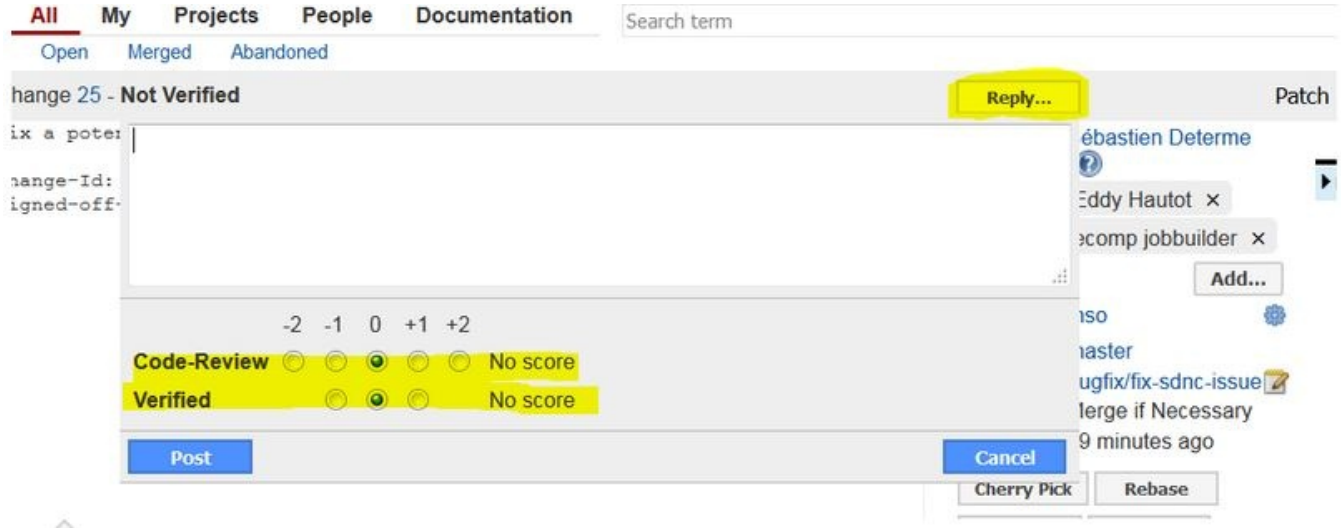
```
# count number of commits to be squashed, the 2 below is for
two
git reset --soft HEAD~2
git commit
```

Reviewing and merging on Gerrit web site

Submitted reviews with changes should appear in the Gerrit web interface. The Jenkins job builder will publish test results, giving "Verified +1" if the test succeeds and -1 if the test fails. A contributor or committer can review it with a -1/0/+1. A committer then must approve it with a +2 rating and merge the code to the master branch. All actions are done on the gerrit.acumos.org web site.



The committer may take any of several actions, such as clicking on the "Reply" button, adding reviewers, adding a review comment, and moving the flags to +2 and +1



Once a committer/contributor approves it, the code can be merged to the master branch.

Triggering Jenkins jobs from Gerrit

Jobs can be triggered at Jenkins by posting a comment in the Gerrit system on a review.

Post "**recheck**" to re-run a verify job. This may be necessary due to intermittent failures at Jenkins, because a dependency has become available, etc.

Post "**remerge**" to re-run a merge job. This may be necessary due to intermittent failures at Jenkins, to recreate a SNAPSHOT version of an artifact, etc.

Post "**stage-release**" to run a stage-for-release job. This should be done to produce and deploy a release candidate. Obviously this should follow successful testing of the SNAPSHOT version.

Post "**run-sonar**" to run a Sonar analysis job.

Post "**run-clm**" to run a CLM job. The CLM jobs are still scheduled to run every Saturday, this feature can be useful for debugging on demand. Commenting "run-clm" in a gerrit that is not merged, will not trigger the CLM job based on that revision but will trigger the job based on the tip of the branch. This job is designed to always run on the latest tip of the branch to avoid inconsistencies on the reports.

Staging a release candidate

Once testing of the snapshot version has been completed, prepare a release candidate in the staging area. First ensure that your POM file has only release versions in the dependency sections. Then trigger the appropriate Jenkins job to stage a release candidate as noted above. The job uses magic (ok a shell script) to strip the string "-SNAPSHOT" from every line of all of your project's POM files. This means any lingering dependency version string with -SNAPSHOT will also be modified, not just the project version string. If the modified build succeeds, the build outputs are deployed or pushed to a staging repository.

Jar files in the staging repository are not used by the LF Jenkins server - there is no Maven profile for this. My advice is to treat the staging repository as a brief resting stop on the way to full release status.

If you force a local build job to pull artifacts from a staging repository by fiddling your local settings, be warned that the staging server will give you the **OLDEST** artifact in the staging repository, not the newest. This is a very real practical limitation on the usefulness of the staging repository.

Releasing your Java/maven artifact

Once testing against the staging repo version has been completed (see above ⚠️) and the project has determined that the artifact in the staged repository is ready for release, the project team can use the new-for-2019 self-release process as follows, once the required jobs are added to the Jenkins server via appropriate entries in the JJB templates. Also see documentation at https://docs.releng.linuxfoundation.org/projects/global-jjb/en/latest/jjb/lf-release-jobs.html

1. Find the Jenkins stage job that created the release candidate. Look among its output logs for the file with the name: staging-repo.txt.gz, it will have a URL like this:
    a. https://logs.acumos.org/production/vex-yul-acumos-jenkins-1/common-dataservice-maven-dl-stage-master/4/staging-repo.txt.gz
2. Create a new release yaml file in the directory "releases/" at the repo root.
    a. The file should have the version being released plus a word or two, for example "1.0.0-maven.yaml". An example of the content appears below.
    b. The file content has the version, the project and (critically) the log directory you found above, altho in abbreviated form.
3. Create a new change set with just the new file, commit to git locally and submit the change set to Gerrit.
4. After the verify job succeeds, a project committer should merge the change set. This will tag the repository with the version string.

Example release yaml file content:

```
---
distribution_type: 'maven'
version: '1.0.0'
project: 'example-project'
log_dir: 'example-project-maven-stage-master/17/'
```

Once complete, the artifacts should appear in the Nexus2 release repository.

Releasing your Docker artifact

Once testing against the staging repo version has been completed (see above ⚠️) and the project has determined that the artifact in the staged registry is ready for release, a release can then be performed as follows:

1. Self service release of docker containers using gerrit and jenkins – you must setup jenkins – more info here: Self Service Docker Release-adoption

    - Add releases folder into the top level directory of your repository - one time change
    - Add releases/release-docker.yaml to your repo (you can name it anything just a suggestion) – just make sure you only have one file change per gerrit commit.
    - Example:

```
distribution_type: container
container_release_tag: 0.26.2
container_pull_registry: nexus3.acumos.org:10004
container_push_registry: nexus3.acumos.org:10002
project: license-usage-manager
ref: d1b9cd2dd345fbeec0d3e2162e008358b8b663b2
containers:
     - name: lum-db
       version: 0.26.2
     - name: lum-server
       version: 0.26.2
```

    - Fill out the release-docker.yml – with the version you are releasing and the git commit (ref) that contains the commit that was built.
        - ref – the gerrit commit sha for the tagging of the gerrit repo
        - containers – can have 1 to many docker images, container names do not need to have acumos/ in the name property.
        - container-release-tag - should match the version you have in pom.xml (plus the any build number if added to tag) (in maven if you want to tag a specific docker build your version would be 0.26.3-b03 for example or in container-tag.yaml
        - Example release in gerrit for docker
            - https://gerrit.acumos.org/r/c/license-usage-manager/+/5430
    - Add release-docker.yaml as one file change in one commit:
        - do not add multiple files into this commit

        - do not rename files either it counts as two file changes.
    - Make sure that release-verify Jenkins job completes – you will see the update in the gerrit commit then merge. (Wait for a plus +1 get a second pair of eyes to look at your change)
        - After merge of gerrit release commit the docker image will be pulled from nexus3.acumos.org:10004 (staging) and pushed into the release nexus repo (nexus3.acumos.org:10002)
    - After release-merge job completes you should be able to run
        - example:
            - docker pull nexus3.acumos.org:10002/acumos/lum-server:0.26.2
    - The release job handles the automatic tagging of the repo as well
        - For example - https://gerrit.acumos.org/r/admin/repos/license-usage-manager,tags
2. Full Service Process Back up option - if self service is not available then follow this older process.
    a. Find the Jenkins stage job that created the release candidate. Look among its output logs for the file with the name: staging-repo.txt.gz, it will have a URL like this:
        i. https://logs.acumos.org/production/vex-yul-acumos-jenkins-1/common-dataservice-maven-dl-stage-master/4/staging-repo.txt.gz
    b. Open a ticket with the LF service desk to sign the artifact(s) in the staging repo and promote them to the release repo. Example request is below.
        i. Login at https://jira.linuxfoundation.org/servicedesk/customer/portals
        ii. Create a new request - search for "Release Artifacts"
        iii. Paste in the Jenkins stage job URL and Jenkins log URL from above.
        iv. Paste in the name of the autorelease repository.

Example service request content:

```
Please release the Acumos PROJECT-NAME docker image at version VERSION
Stage job:
    https://jenkins.acumos.org/job/PROJECT-stage-master/201/
Logs:
    https://logs.acumos.org/production/vex-yul-acumos-jenkins-1/PROJECT-stage-master/201/
Autorelease repo name:
    autorelease-1150
Thanks in advance
```

Once complete, the artifacts should appear in the Nexus3 release registry.

Finally, add a tag for the release on the Gerrit web site.

Capturing the release in Gerrit history

This addresses the need for a project to reproduce from git the EXACT, byte-for-byte state of the project's files as built, which is not a requirement for all projects. The Jenkins build server's stage-release job modifies POM files to strip --SNAPSHOT suffixes. The Jenkins server stores in the local git repository the state of the project after that modification. That state is published via a git bundle file that is pushed to the log server, it is NOT pushed directly back to Gerrit, because Jenkins generally cannot push changes back to Gerrit. To add that item into project history, that commit can be downloaded, merged to the local repository and submitted via a Gerrit review into the global history. Steps:

1. Download the taglist.log and project.bundle files from the log server
2. Using the SHA code from taglist.log, checkout the commit hash
3. Merge the project.bundle patch into the local repository
4. Submit the change as a git review to Gerrit, and merge the patch set
5. Git tag the release
6. Push the release tag to Gerrit.

Jenkins Job Builder Templates

All jobs in the Jenkins server are generated from Jenkins Job Builder (JJB) templates. The templates for Acumos projects are maintained in the "ci-management" git repository. A project template is defined for each project in a YAML file (e.g., "common-dataservice.yaml"). The project templates use job templates defined by the Linux Foundation Global JJB as well as job templates custom defined for the Acumos project.