releng/builder

Release master

OpenDaylight Project

Aug 31, 2023

CONTENTS

This guide provides details into OpenDaylight Infrastructure and services.

Contents:

CHAPTER

ONE

JENKINS

The Release Engineering Project consolidates the Jenkins jobs from project-specific VMs to a single Jenkins server. Each OpenDaylight project has a tab for their jobs on the jenkins-master. The system utilizes Jenkins Job Builder for the creation and management of the Jenkins jobs.

Sections:

1.1 New Project Quick Start

This section attempts to provide details on how to get going as a new project quickly with minimal steps. The rest of the guide should be read and understood by those who need to create and contribute new job types that is not already covered by the existing job templates provided by OpenDaylight's JJB repo.

As a new project you will be mainly interested in getting your jobs to appear in the jenkins-master silo and this can be achieved by simply creating a <project>.yaml in the releng/builder project's jjb directory.

```
git clone --recursive https://git.opendaylight.org/gerrit/releng/builder
cd builder
mkdir jjb/<new-project>
```

Where <new-project> should be the same name as your project's git repo in Gerrit. If your project is called "aaa" then create a new jjb/aaa directory.

Next we will create <new-project>.yaml as follows:

```
---
- project:
    name: <NEW_PROJECT>-carbon
    jobs:
      - '{project-name}-clm-{stream}'
      - '{project-name}-integration-{stream}'
      - '{project-name}-merge-{stream}'
      - '{project-name}-verify-{stream}-{maven}-{jdks}'

    project: '<NEW_PROJECT>'
    project-name: '<NEW_PROJECT>'
    stream: carbon
    branch: 'master'
    jdk: openjdk8
    jdks:
      - openjdk8
    maven:
      - mvn33:
          mvn-version: 'mvn33'
    mvn-settings: '<NEW_PROJECT>-settings'
    mvn-goals: 'clean install -Dmaven.repo.local=/tmp/r -Dorg.ops4j.pax.url.mvn.
↪localRepository=/tmp/r'
    mvn-opts: '-Xmx1024m -XX:MaxPermSize=256m'
    dependencies: 'odlparent-merge-{stream},yangtools-merge-{stream},controller-merge-
↪{stream}'
    email-upstream: '[<NEW_PROJECT>] [odlparent] [yangtools] [controller]'
    archive-artifacts: ''

- project:
    name: <NEW_PROJECT>-sonar
    jobs:
      - '{project-name}-sonar'

    project: '<NEW_PROJECT>'
    project-name: '<NEW_PROJECT>'
    branch: 'master'
```

(continues on next page)

(continued from previous page)

```
    mvn-settings: '<NEW_PROJECT>-settings'
    mvn-goals: 'clean install -Dmaven.repo.local=/tmp/r -Dorg.ops4j.pax.url.mvn.
↪localRepository=/tmp/r'
    mvn-opts: '-Xmx1024m -XX:MaxPermSize=256m'
```

Replace all instances of <new-project> with the name of your project. This will create the jobs with the default job types we recommend for Java projects. If your project is participating in the simultanious-release and ultimately will be included in the final distribution, it is required to add the following job types into the job list for the release you are participating.

```
- '{project-name}-distribution-check-{stream}'
- '{project-name}-validate-autorelease-{stream}'
```

If you'd like to explore the additional tweaking options available please refer to the *Jenkins Job Templates* section.

Finally we need to push these files to Gerrit for review by the releng/builder team to push your jobs to Jenkins.

```
git add jjb/<new-project>
git commit -sm "Add <new-project> jobs to Jenkins"
git review
```

This will push the jobs to Gerrit and your jobs will appear in Jenkins once the releng/builder team has reviewed and merged your patch.

1.2 Jenkins Master

The jenkins-master is the home for all project's Jenkins jobs. All maintenance and configuration of these jobs must be done via JJB through the releng-builder-repo. Project contributors can no longer edit the Jenkins jobs directly on the server.

1.3 Build Minions

The Jenkins jobs are run on build minions (executors) which are created on an as-needed basis. If no idle build minions are available a new VM is brought up. This process can take up to 2 minutes. Once the build minion has finished a job, it will be destroyed.

Our Jenkins master supports many types of dynamic build minions. If you are creating custom jobs then you will need to have an idea of what type of minions are available. The following are the current minion types and descriptions. Minion Template Names are needed for jobs that take advantage of multiple minions as they must be specifically called out by template name instead of label.

1.3.1 Adding New Components to the Minions

If your project needs something added to one of the minions, you can help us get things added faster by doing one of the following:

- Submit a patch to RelEng/Builder for the appropriate *jenkins-scripts* definition which configure software during minion boot up.
- Submit a patch to RelEng/Builder for the *packer/provision* scripts that configures software during minion instance imaging.
- Submit a patch to RelEng/Builder for the Packer's templates in the *packer/templates* directory that configures a new instance definition along with changes in *packer/provision*.

Going the first route will be faster in the short term as we can inspect the changes and make test modifications in the sandbox to verify that it works.

Note: The first route may add additional setup time considering this is run every time the minion is booted.

The second and third routes, however, is better for the community as a whole as it will allow others to utilize our Packer setups to replicate our systems more closely. It is, however, more time consuming as an image snapshot needs to be created based on the updated Packer definitions before it can be attached to the Jenkins configuration on sandbox for validation testing.

In either case, the changes must be validated in the sandbox with tests to make sure that we don't break current jobs and that the new software features are operating as intended. Once this is done the changes will be merged and the updates applied to the RelEng Jenkins production silo. Any changes to files under *releng/builder/packer* will be validated and images would be built triggered by verify-packer and merge-packer jobs.

Please note that the combination of a Packer definitions from *vars*, *templates* and the *provision* scripts is what defines a given minion. For instance, a minion may be defined as *centos7-builder* which is a combination of Packer OS image definitions from *vars/centos.json*, Packer template definitions from *templates/builder.json* and spinup scripts from *provision/builder.sh*. This combination provides the full definition of the realized minion.

Jenkins starts a minion using the latest image which is built and linked into the Jenkins configuration. Once the base instance is online Jenkins checks out the RelEng/Builder repo on it and executes two scripts. The first is *provision/baseline.sh*, which is a baseline for all of the minions.

The second is the specialized script, which handles any system updates, new software installs or extra environment tweaks that don't make sense in a snapshot. Examples could include installing new package or setting up a virtual environment. Its imperative to ensure modifications to these spinup scripts have considered time taken to install the packages, as this could increase the build time for every job which runs on the image. After all of these scripts have executed Jenkins will finally attach the minion as an actual minion and start handling jobs on it.

1.3.2 Flavors

Performance flavors come with dedicated CPUs and are not shared with other accounts in the cloud so should ensure consistent performance.

Table 1: Flavors

Instance Type	CPUs	Memory
v3-standard-2	2	8
v3-standard-4	4	16
v3-standard-8	8	32
v3-standard-16	16	64
odl-highcpu-2	2	2
odl-highcpu-4	4	4
odl-highcpu-8	8	8

1.3.3 Pool: ODLVEX

1.3.4 Pool: ODLVEX - HOT (Heat Orchestration Templates)

HOT integration enables to spin up integration labs servers for CSIT jobs using heat, rathar than using jclouds (deprecated). Image names are updated on the project specific job templates using the variable *{odl,docker,openstack,tools}_system_image* followed by image name in the format <*platform*> - <*template*> - <*date-stamp*>.

Following are the list of published images available to Jenkins jobs.

- ZZCI - CentOS 7 - builder - x86_64 - 20190403-205252.587
- ZZCI - CentOS 7 - builder - x86_64 - 20220101-060058.758
- ZZCI - CentOS 7 - builder - x86_64 - 20220401-060107.331
- ZZCI - CentOS 7 - builder - x86_64 - 20220811-110452.412
- ZZCI - CentOS 7 - builder - x86_64 - 20220830-004905.209
- ZZCI - CentOS 7 - builder - x86_64 - 20220915-210350.650
- ZZCI - CentOS 7 - builder - x86_64 - 20221016-222911.194
- ZZCI - CentOS 7 - builder - x86_64 - 20221201-060105.225
- ZZCI - CentOS 7 - builder - x86_64 - 20230301-060101.869
- ZZCI - CentOS 7 - builder - x86_64 - 20230401-060117.151
- ZZCI - CentOS 7 - builder - x86_64 - 20230501-060110.287
- ZZCI - CentOS 7 - devstack - x86_64 - 20220401-230107.511
- ZZCI - CentOS 7 - devstack - x86_64 - 20220915-220248.057
- ZZCI - CentOS 7 - devstack - x86_64 - 20221016-125752.520
- ZZCI - CentOS 7 - devstack - x86_64 - 20230301-230109.257
- ZZCI - CentOS 7 - devstack - x86_64 - 20230401-230106.445
- ZZCI - CentOS 7 - devstack-rocky - 20190601-000116.015
- ZZCI - CentOS 7 - devstack-rocky - 20190628-065204.973
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20191002-183226.559
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20200801-000156.903
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20200811-042113.395

- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20200813-042753.841
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20220401-000109.037
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20220811-110620.848
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20220915-220323.497
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20221016-125827.911
- ZZCI - CentOS 7 - devstack-rocky - x86_64 - 20221101-000109.537
- ZZCI - CentOS 7 - devstack-stein - x86_64 - 20220401-010109.230
- ZZCI - CentOS 7 - devstack-stein - x86_64 - 20220811-110634.575
- ZZCI - CentOS 7 - devstack-stein - x86_64 - 20220915-222435.096
- ZZCI - CentOS 7 - devstack-stein - x86_64 - 20221016-222956.928
- ZZCI - CentOS 7 - devstack-stein - x86_64 - 20221101-010107.368
- ZZCI - CentOS 7 - docker - x86_64 - 20220401-220102.840
- ZZCI - CentOS 7 - docker - x86_64 - 20220811-110637.413
- ZZCI - CentOS 7 - docker - x86_64 - 20220915-220324.722
- ZZCI - CentOS 7 - docker - x86_64 - 20221016-223020.545
- ZZCI - CentOS 7 - docker - x86_64 - 20221101-220103.978
- ZZCI - CentOS 7 - docker - x86_64 - 20221201-220105.396
- ZZCI - CentOS 7 - docker - x86_64 - 20230301-220107.956
- ZZCI - CentOS 7 - docker - x86_64 - 20230401-220108.252
- ZZCI - CentOS 7 - docker - x86_64 - 20230501-220111.311
- ZZCI - CentOS 7 - helm - x86_64 - 20220401-000138.473
- ZZCI - CentOS 7 - helm - x86_64 - 20220811-110654.568
- ZZCI - CentOS 7 - helm - x86_64 - 20220915-220356.090
- ZZCI - CentOS 7 - helm - x86_64 - 20221016-223030.291
- ZZCI - CentOS 7 - helm - x86_64 - 20221101-000135.064
- ZZCI - CentOS 7 - helm - x86_64 - 20230301-000133.034
- ZZCI - CentOS 7 - robot - 20190430-080312.962
- ZZCI - CentOS 7 - robot - x86_64 - 20220401-220138.484
- ZZCI - CentOS 7 - robot - x86_64 - 20220915-220357.338
- ZZCI - CentOS 7 - robot - x86_64 - 20221016-223041.341
- ZZCI - CentOS 7 - robot - x86_64 - 20221101-220138.675
- ZZCI - CentOS 7 - robot - x86_64 - 20221201-220143.533
- ZZCI - CentOS 7 - robot - x86_64 - 20230301-220131.480
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220303-223622.243
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220405-005246.199
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220411-013651.819

- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220411-025029.496
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220601-071415.711
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220629-035812.822
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220701-160059.919
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220801-160143.906
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20220811-073719.385
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20221016-222440.331
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20221101-160106.524
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20221201-160128.560
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20230301-160121.204
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20230401-160111.589
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20230501-160107.084
- ZZCI - CentOS Stream 8 - builder - x86_64 - 20230727-135233.501
- ZZCI - CentOS Stream 8 - robot - x86_64 - 20220811-231817.668
- ZZCI - CentOS Stream 8 - robot - x86_64 - 20230301-010147.625
- ZZCI - CentOS Stream 8 - robot - x86_64 - 20230401-010209.151
- ZZCI - OPNFV - apex - compute - 0
- ZZCI - OPNFV - apex - compute - 1
- ZZCI - OPNFV - apex - controller - 0
- ZZCI - Ubuntu 16.04 - docker - x86_64 - 20190614-042302.610
- ZZCI - Ubuntu 16.04 - gbp - 20190521-223526.319
- ZZCI - Ubuntu 16.04 - kubernetes - 20190206-080347.936
- ZZCI - Ubuntu 16.04 - kubernetes - 20190211-225526.126
- ZZCI - Ubuntu 16.04 - mininet-ovs-25 - 20190416-121328.240
- ZZCI - Ubuntu 16.04 - mininet-ovs-26 - 20190521-223726.040
- ZZCI - Ubuntu 16.04 - mininet-ovs-28 - 20190415-091034.881
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20220201-040158.287
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20220501-040104.357
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20220601-040059.617
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20220701-040013.395
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20221001-040106.423
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20221201-040108.330
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20230101-040125.332
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20230301-040106.351
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20230401-040112.177
- ZZCI - Ubuntu 18.04 - docker - x86_64 - 20230501-040105.925

- ZZCI - Ubuntu 18.04 - helm - - 20210513-214525.779
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20220501-140101.102
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20220811-112321.717
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20220915-235325.735
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20221013-122339.021
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20221101-140104.772
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20221201-140107.142
- ZZCI - Ubuntu 18.04 - helm - x86_64 - 20230301-140059.950
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220201-180056.429
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220501-180100.971
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220601-180059.980
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220701-180056.799
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220801-180111.774
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20220915-223016.788
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20221013-083654.129
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20221101-180142.920
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20221201-180114.186
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20230301-180106.402
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20230401-180107.945
- ZZCI - Ubuntu 18.04 - mininet-ovs-28 - x86_64 - 20230501-180106.320

1.4 Creating Jenkins Jobs

Jenkins Job Builder takes simple descriptions of Jenkins jobs in YAML format and uses them to configure Jenkins.

- Jenkins Job Builder (JJB) documentation
- RelEng/Builder Gerrit
- RelEng/Builder Git repository

1.5 Getting Jenkins Job Builder

OpenDaylight uses Jenkins Job Builder to translate our in-repo YAML job configuration into job descriptions suitable for consumption by Jenkins. When testing new Jenkins Jobs in the *Jenkins Sandbox*, you'll need to use the *jenkins-jobs* executable to translate a set of jobs into their XML descriptions and upload them to the sandbox Jenkins server.

We document *installing* *jenkins-jobs* below.

1.6 Installing Jenkins Job Builder

We recommend using *pip* to assist with JJB installs, but we also document *installing from a git repository manually*. For both, we recommend using Python *Virtual Environments* to isolate JJB and its dependencies.

The builder/jjb/requirements.txt file contains the currently recommended JJB version. Because JJB is fairly unstable, it may be necessary to debug things by installing different versions. This is documented for both *pip-assisted* and *manual* installs.

1.7 Virtual Environments

For both *pip-assisted* and *manual* JJB installs, we recommend using Python Virtual Environments to manage JJB and its Python dependencies. The python-virtualenvwrapper tool can help you do so.

Documentation is available for installing python-virtualenvwrapper. On Linux systems with pip (typical), they amount to:

```
sudo pip install virtualenvwrapper
```

A virtual environment is simply a directory that you install Python programs into and then append to the front of your path, causing those copies to be found before any system-wide versions.

Create a new virtual environment for JJB.

```
# Virtaulenvwrapper uses this dir for virtual environments
$ echo $WORKON_HOME
/home/daniel/.virtualenvs
# Make a new virtual environment
$ mkvirtualenv jjb
# A new venv dir was created
(jjb)$ ls -rc $WORKON_HOME | tail -n 1
jjb
# The new venv was added to the front of this shell's path
(jjb)$ echo $PATH
/home/daniel/.virtualenvs/jjb/bin:<my normal path>
# Software installed to venv, like pip, is found before system-wide copies
(jjb)$ command -v pip
/home/daniel/.virtualenvs/jjb/bin/pip
```

With your virtual environment active, you should install JJB. Your install will be isolated to that virtual environment's directory and only visible when the virtual environment is active.

You can easily leave and return to your venv. Make sure you activate it before each use of JJB.

```
(jjb)$ deactivate
$ command -v jenkins-jobs
# No jenkins-jobs executable found
$ workon jjb
(jjb)$ command -v jenkins-jobs
$WORKON_HOME/jjb/bin/jenkins-jobs
```

1.8 Installing JJB using pip

The recommended way to install JJB is via pip.

First, clone the latest version of the releng-builder-repo.

```
$ git clone --recursive https://git.opendaylight.org/gerrit/p/releng/builder.git
```

Before actually installing JJB and its dependencies, make sure you've *created and activated* a virtual environment for JJB.

```
$ mkvirtualenv jjb
```

The recommended version of JJB to install is the version specified in the builder/jjb/requirements.txt file.

```
# From the root of the releng/builder repo
(jjb)$ pip install -r jjb/requirements.txt
```

To validate that JJB was successfully installed you can run this command:

```
(jjb)$ jenkins-jobs --version
```

TODO: Explain that only the currently merged jjb/requirements.txt is supported, other options described below are for troubleshooting only.

To change the version of JJB specified by builder/jjb/requirements.txt to install from the latest commit to the master branch of JJB's git repository:

```
$ cat jjb/requirements.txt
-e git+https://git.openstack.org/openstack-infra/jenkins-job-builder#egg=jenkins-job-
↪builder
```

To install from a tag, like 1.4.0:

```
$ cat jjb/requirements.txt
-e git+https://git.openstack.org/openstack-infra/jenkins-job-builder@1.4.0#egg=jenkins-
↪job-builder
```

1.8.1 Updating releng/builder repo or global-jjb

Follow these steps to update the releng/builder repo. The repo uses a submodule from a global-jjb repo so that common source can be shared across different projects. This requires updating the releng/builder repo periodically to pick up the changes. New versions of jjb could also require updating the releng/builder repo. Follow the previous steps earlier for updating jenkins-jobs using the builder/jjb/requirements.txt file. Ensure that the version listed in the file is the currently supported version, otherwise install a different version or simply upgrade using *pip install –upgrade jenkins-job-builder*.

The example below assumes the user has cloned releng/builder to *~/git/releng/builder*. Update the repo, update the submodules and then submit a test to verify it works.

```
cd ~/git/releng/builder
git checkout master
git pull
git submodule update --init --recursive
```

(continues on next page)

(continued from previous page)

```
jenkins-jobs --conf jenkins.ini test jjb/ netvirt-csit-1node-openstack-queens-upstream-
↪stateful-fluorine
```

1.9 Installing JJB Manually

This section documents installing JJB from its manually cloned repository.

Note that *installing via pip* is typically simpler.

Checkout the version of JJB's source you'd like to build.

For example, using master:

```
$ git clone https://git.openstack.org/openstack-infra/jenkins-job-builder
```

Using a tag, like 1.4.0:

```
$ git clone https://git.openstack.org/openstack-infra/jenkins-job-builder
$ cd jenkins-job-builder
$ git checkout tags/1.4.0
```

Before actually installing JJB and its dependencies, make sure you've *created and activated* a virtual environment for JJB.

```
$ mkvirtualenv jjb
```

You can then use JJB's requirements.txt file to install its dependencies. Note that we're not using *sudo* to install as root, since we want to make use of the venv we've configured for our current user.

```
# In the cloned JJB repo, with the desired version of the code checked out
(jjb)$ pip install -r requirements.txt
```

Then install JJB from the repo with:

```
(jjb)$ pip install .
```

To validate that JJB was successfully installed you can run this command:

```
(jjb)$ jenkins-jobs --version
```

1.10 Jenkins Job Templates

The OpenDaylight RelEng/Builder project provides jjb-templates that can be used to define basic jobs.

The *Gerrit Trigger* listed in the jobs are keywords that can be used to trigger the job to run manually by simply leaving a comment in Gerrit for the patch you wish to trigger against.

All jobs have a default build-timeout value of 360 minutes (6 hrs) but can be overrided via the opendaylight-infra-wrappers' build-timeout property.

TODO: Group jobs into categories: every-patch, after-merge, on-demand, etc. TODO: Reiterate that "remerge" triggers all every-patch jobs at once, because when only a subset of jobs is triggered, Gerrit forgets valid -1 from jobs outside

the subset. TODO: Document that only drafts and commit-message-only edits do not trigger every-patch jobs. TODO: Document test-{project}-{feature} and test-{project}-all.

1.11 Maven Properties

We provide a properties which your job can take advantage of if you want to do trigger a different configuration depending on job type. You can create a profile that activates on a property listed below. The JJB templated jobs will activate the profile during the build to run any custom code configuration you wish to run for this job type.

```
-Dmerge   : The Merge job sets this flag and is the same as setting the
            Maven property <merge>true</merge>.
-Dsonar   : The Sonar job sets this flag and is the same as setting the
            Maven property <sonar>true</sonar>.
```

1.12 Jenkins Sandbox

URL: https://jenkins.opendaylight.org/sandbox

Jenkins Sandbox documentation is available in the LF Jenkins Sandbox Guide.