releng

Release Latest

Open Platform for NFV

Aug 18, 2021

CONTENTS

CHAPTER

ONE

RELEASING OPNFV

1.1 Release Process

TBD

1.2 Release Automation

This page describes how projects can take advantage of the release automation introduced in Fraser for creating their stable branch, and stable branch Jenkins jobs.

It also describes the structures of the `releases` directory and the associated scripts.

1.2.1 Stable Branch Creation

If your project participated in the last release (beginning with Euphrates), perform the following steps:

1. Copy your project's release file to the new release directory. For example:

```
cp releases/euphrates/apex.yaml releases/fraser/apex.yaml
```

2. For projects who are participating the in the stable release process for the first time, you can either copy a different project's file and changing the values to match your project, or use the following template, replacing values marked with < and >:

```
---
project: <opnfv-project-name>
project-type: <opnfv-project-type>
release-model: stable

branches:
  - name: stable/<release>
    location:
      <project-repo>: <git-sha1>
```

3. Modify the file, replacing the previous stable branch name with the new release name, and the commit the branch will start at. For example:

```
branches:
  - name: stable/fraser
    location:
      apex: <git-full-sha1>
```

4. If your project contains multiple repositories, add them to the list of branches. They can also be added later if more time is needed before the stable branch window closes.

```
branches:
  - name: stable/fraser
    location:
      apex: <git-sha1>
  - name: stable/fraser
    location:
      apex-puppet-tripleo: <git-sha1>
```

5. Git add, commit, and git-review the changes. A job will be triggered to verify the commit exists on the branch, and the yaml file follows the scheme listed in `releases/schema.yaml`
6. Once the commit has been reviewed and merged by Releng, a job will be triggered to create the stable branch Jenkins jobs under `jjb/`.

1.2.2 Stable Release Tagging

TBD

1.2.3 Release File Fields

The following is a description of fields in the Release file, which are verified by the scheme file at `releases/schema.yaml`

project Project team in charge of the release.

release-model Release model the project follows.

One of: stable, non-release

project-type Classification of project within OPNFV.

One of: installer, feature, testing, tools, infra

upstream (Optional) Upstream OpenStack project assocated with this project.

releases List of released versions for the project.

version Version of the release, must be in the format `opnfv-X.Y.Z`.

location Combination of repository and git hash to locate the release version.

Example:

```
opnfv-project: f15d50c2009f1f865ac6f4171347940313727547
```

branches List of stable branches for projects following the `stable` release-model.

name Stable branch name. Must start with the string `stable/`

location Same syntax as `location` under `releases`

release-notes Link to release notes for the projects per-release.

1.2.4 Scripts

- `create_branch.py -f <RELEASE_FILE>`

  Create branches in Gerrit listed in the release file.

  Must be ran from the root directory of the releng repository as the release name is extracted from the subdirectory under `releases/`

  The Gerrit server can be changed by creating a `~/releases.cfg` file with the following content:

  ```
  [gerrit]
  url=http://gerrit.example.com
  ```

  This will override the default configuration of using the OPNFV Gerrit server at https://gerrit.opnfv.org, and is primarily used for testing.

- `create_jobs.py -f <RELEASE_FILE>`

  Modifies the jenkins job files for a project to add the stable branch stream. Assumes the jenkins jobs are found in the releng repository under `jjb/<project>/`

- `verify_schema -s <SCHEMA_FILE> -y <YAML_FILE>`

  Verifies the yaml file matches the specified jsonschema formatted file. Used to verify the release files under `releases/`

1.3 Stable Branch

TBD

1.4 Versioning

TBD

CHAPTER

TWO

OPNFV CI

OPNFV continuous integration (CI) is ran on a variety of *hardware* connected to Jenkins and mangaged through YAML files in the Releng repository. These YAML files are read by Jenkins Job Builder to generate and upload Jenkins jobs to the server. See the *User Guide* for resources on getting started with CI for your project.

2.1 CI User Guide

2.1.1 Structure of the Releng Repository

jjb/<projects> Individual project CI configurations.

jjb/global Collection of JJB defaults and templates shared by all projects.

global-jjb/ Git submodule pointing to Global-JJB, which provides a variety of common CI jobs such as ReadTheDocs (RTD) builds.

docs/ This documentation.

releases/ Release configuration files for creating stable branches and tagging repositories and related automation scripts.

utils/ Collection of common utilities used by projects

utils/build-server-ansible Ansible configuration for managing build servers. This is where projects can add packages they need for their CI to the servers.

2.1.2 CI Setup

Basic Setup

All projects are required to have a **+1 Verified** vote in Gerrit in order to merge their code. As a new project that comes in may not yet know how they want to setup CI, they can pass this validation by configuring a 'no-op' job to run against their changesets.

1. Clone the Releng repository, using the *Clone with commit-msg hook* command under the *SSH* tab (displayed after logging in and uploading an SSH key):

   Note: <gerrit username> in the command below will be your username in Gerrit when viewing the command on the website.

   For example:

```
git clone "ssh://<gerrit username>@gerrit.opnfv.org:29418/releng" && \
scp -p -P 29418 <gerrit username>@gerrit.opnfv.org:hooks/commit-msg "releng/.git/
↪hooks/"
```

2. Create a project directory under the *jjb/* directory, and an intial project YAML file:

```
mkdir jjb/myproject
touch jjb/myproject/myproject-ci-jobs.yaml
```

3. Modify the project YAML file to add the basic validation job:

```
$EDITOR jjb/myproject/myproject-ci-jobs.yaml
```

```
---
- project:
    name: myproject
    project:
      - '{name}'
    jobs:
      - '{project}-verify-basic'
```

Docker Builds

Docker build are managed through the **jjb/releng/opnfv-docker.yaml** file. Modify this file with your project details to enable docker builds on merges and tags to your project repository:

```
---
- project:
    name: opnfv-docker'

    [...]

    dockerrepo:
      [...]
      - 'myproject':
        project: 'myproject'
        <<: *master
```

Documentation Builds

Documentation is build using they Python Sphinx project. You can read more about how these build work and how your documentation should be setup in the opnfvdocs project.

Create a file at **jjb/myproject/myproject-rtd-builds.yaml** with the following content:

```
---
- project:
    name: myproject-rtd
    project: myproject
    project-name: myproject

    project-pattern: 'myproject'
    rtd-build-url: <request from LFN IT>
    rtd-token: <request from LFN IT>
```

(continues on next page)

(continued from previous page)

```
jobs:
  - '{project-name}-rtd-jobs'
```

Note: Open a ticket with a link to the change adding your documentation at support.linuxfoundation.org and the LFN IT team will provide you the *rtd-build-url* and *rtd-token*.

This will create jobs to build your project documentation (under *docs/* in your project repository) on proposed changes, and trigger a rebuild on the RTD site when code is merged in your project.

2.2 CI Resources

CI for OPNFV requires a range of resources in order to meet testing and verification needs. Each resource must meet a set of criteria in order to be part of CI for an OPNFV release. There are three types of resources:

- Baremetal PODs (PODs)
- Virtual PODs (vPODs)
- Build Servers

2.2.1 Baremetal PODs

Baremetal PODs are used to deploy OPNFV on to baremetal hardware through one of the installer projects. They enable the full range of scenarios to be deployed and tested.

Requirements

In order of a POD to be considered CI-Ready the following requirements must be met:

1. Pharos Compliant and has a PDF
2. Connected to Jenkins
3. 24/7 Uptime
4. No Development
5. No manual intervention

Table 1: CI Servers for Baremetal Deployment

Node	Usage	Jumphost OS / Version	PDF	IDF
arm-pod9	Armband	Ubuntu 16.04	PDF	IDF
arm-pod10	Fuel	Ubuntu 16.04	PDF	IDF
ericsson-pod1	Fuel	Ubuntu 16.04	PDF	IDF
ericsson-pod2	XCI	Ubuntu 16.04	PDF	IDF
flex-pod1	Yardstick		PDF	IDF
flex-pod2	Apex		PDF	IDF
huawei-pod1	Compass4NFV		PDF	IDF
huawei-pod2	Compass4NFV	Ubuntu 14.04	PDF	IDF
huawei-pod3	Yardstick	Ubuntu 14.04	PDF	IDF
huawei-pod4	Dovetail		PDF	IDF
huawei-pod6		Ubuntu 14.04	PDF	IDF
huawei-pod7	Dovetail	Ubuntu 14.04	PDF	IDF
huawei-pod8	Compass4NFV	Ubuntu 16.04 (aarch64)	PDF	IDF
huawei-pod12	JOID	Ubuntu 16.04	PDF	IDF
intel-pod10	KVMforNFV	CentOS 7	PDF	IDF
intel-pod11	Apex		PDF	IDF
intel-pod12	VSPerf	CentOS 7	PDF	IDF
intel-pod17	Airship		PDF	IDF
intel-pod18	Airship		PDF	IDF
lf-pod1	Apex	CentOS 7	PDF	IDF
lf-pod2	Fuel	CentOS 7	PDF	IDF
unh-pod1	Auto	Ubuntu 16.04 (aarch64)	PDF	IDF
zte-pod1			PDF	IDF
zte-pod2			PDF	IDF
zte-pod3			PDF	IDF
zte-pod4			PDF	IDF
zte-pod9			PDF	IDF

2.2.2 Virtual PODs

Virtual PODs are used to deploy OPNFV in a virtualized environment generally on top of KVM through libvirt.

Requirements

1. Have required virtualization packages installed
2. Meet the Pharos resource specification for virtual PODs
3. Connected to Jenkins
4. 24/7 Uptime

Table 2: CI Servers for Virtual Deployment

Node	Architecture	OS	Contact
arm-virtual2	aarch64	Ubuntu 16.04	Armband ENEA Team
arm-virtual3	aarch64	Ubuntu 16.04	*Xuan Jia*
arm-virtual4	aarch64	Ubuntu 16.04	*Xuan Jia*
ericsson-virtual-pod1bl01	x86_64	CentOS 7	
ericsson-virtual1	x86_64	Ubuntu 16.04	
ericsson-virtual2	x86_64	Ubuntu 16.04	

continues on next page

Table 2 – continued from previous page

Node	Architecture	OS	Contact
ericsson-virtual3	x86_64	Ubuntu 16.04	
ericsson-virtual4	x86_64	Ubuntu 16.04	
ericsson-virtual5	x86_64	Ubuntu 16.04	
huawei-virtual1	x86_64	Ubuntu 14.04	
huawei-virtual2	x86_64	Ubuntu 14.04	
huawei-virtual3	x86_64	Ubuntu 14.04	
huawei-virtual4	x86_64	Ubuntu 14.04	
huawei-virtual5	x86_64		
huawei-virtual6	x86_64	Ubuntu 16.04	
huawei-virtual7	x86_64	Ubuntu 14.04	
huawei-virtual8	x86_64	Ubuntu 14.04	
huawei-virtual9	x86_64	Ubuntu 14.04	
intel-virtual3	x86_64		
intel-virtual11	x86_64		
intel-virtual12	x86_64		
intel-virtual13	x86_64		
intel-virtual14	x86_64		
intel-virtual15	x86_64		
intel-virtual16	x86_64		
lf-virtual1	x86_64	Ubuntu 14.04	Linux Foundation
lf-virtual2	x86_64	CentOS 7	Linux Foundation
lf-virtual3	x86_64	CentOS 7	Linux Foundation
ool-virtual1	x86_64		
ool-virtual2	x86_64		
ool-virtual3	x86_64		
zte-virtual1	x86_64		
zte-virtual2	x86_64		
zte-virtual3	x86_64		
zte-virtual4	x86_64		
zte-virtual5	x86_64		
zte-virtual6	x86_64		

2.2.3 Build Servers

Build servers are used to build project, run basic verifications (such as unit tests and linting), and generate documentation.

Requirements

1. Have required *packages_* installed
2. 24/7 Uptime
3. Connected to Jenkins

Table 3: CI Build Servers

Node	Architecture	OS	Contact
arm-build3	aarch64	CentOS 7.4	Armband ENEA Team
arm-build4	aarch64	Ubuntu 16.04	Armband ENEA Team
lf-build5	x86_64	Ubuntu 18.04	Linux Foundation
lf-build6	x86_64	CentOS 8	Linux Foundation

2.3 Development Resources

Table 4: Baremetal Development Servers

Node	Usage	Jumphost OS / Version	PDF	IDF
cacti-pod1				
cengn-pod1				
itri-pod1				
lf-pod4				
lf-pod5				
nokia-pod1				
ool-pod1				
bii-pod1				

2.4 CI Resources Labels

ci-resource Resource devoted to CI

ci-pod POD devoted to CI

opnfv-build Node is for builds - independent of OS

opnfv-build-centos Node is for builds needing CentOS

opnfv-build-centos-arm Node is for ARM builds on CentOS

opnfv-build-ubuntu Node is for builds needing Ubuntu

opnfv-build-ubuntu-arm Node is for ARM builds on Ubuntu

{installer}-baremetal POD is devoted to {installer} for baremetal deployments

{installer}-virtual Server is devoted to {installer} for virtual deployments

CHAPTER

THREE

SOFTWARE INFRASTRUCTURE

OPNFV Software Infrastructure consists of set of components and tools that realize OPNFV Continuous Integration (CI) and provide means for community to contribute to OPNFV in most efficient way. OPNFV Software Infrastructure enables and orchestrates development, integration and testing activities for the components OPNFV consumes from upstream communities and for the development work done in scope of OPNFV. Apart from orchestration aspects, providing timely feedback that is fit for purpose to the OPNFV community is one of its missions.

CI is the top priority for OPNFV Software Infrastructure. Due to the importance the OPNFV community puts into it, the resulting CI machinery is highly powerful, capable and runs against distributed hardware infrastructure managed by OPNFV Pharos Project. The hardware infrastructure OPNFV CI relies on is located in 3 different continents, 5+ different countries and 10+ different member companies.

OPNFV CI is continuously evolved in order to fulfill the needs and match the expectations of the OPNFV community.

OPNFV Software Infrastructure is developed, maintained and operated by OPNFV Releng Project with the support from Linux Foundation.

3.1 Continuous Integration Server

Jenkins

3.1.1 Connecting OPNFV Community Labs to OPNFV Jenkins

Table of Contents

Abstract

This document describes how to connect resources (servers) located in Linux Foundation (LF) lab and labs provided by the OPNFV Community to OPNFV Jenkins.

License

Version History

Date	Version	Author	Comment
2015-05-05	0.1.0	Fatih Degirmenci	First draft
2015-09-25	1.0.0	Fatih Degirmenci	Instructions for the Arno SR1 release
2016-01-25	1.1.0	Jun Li	Change the format for new doc toolchain
2016-01-27	1.2.0	Fatih Degirmenci	Instructions for the Brahmaputra release
2016-05-25	1.3.0	Julien	Add an additional step after step9 to output the correct monit config file

Jenkins

Jenkins is an extensible open source Continuous Integration (CI) server. [1]

Linux Foundation (LF) hosts and operates OPNFV Jenkins.

Jenkins Slaves

Slaves are computers that are set up to build projects for a **Jenkins Master**. [2]

Jenkins runs a separate program called "**slave agent**" on slaves. When slaves are registered to a master, the master starts distributing load to slaves by scheduling jobs to run on slaves if the jobs are set to run on them. [2]

Term **Node** is used to refer to all machines that are part of Jenkins grid, slaves and master. [2]

Two types of slaves are currently connected to OPNFV Jenkins and handling different tasks depending on the purpose of connecting the slave.

- Slaves hosted in LF Lab
- Slaves hosted in Community Test Labs

The slaves connected to OPNFV Jenkins can be seen using this link: https://build.opnfv.org/ci/computer/

Slaves without red cross next to computer icon are fully functional.

Connecting Slaves to OPNFV Jenkins

The method that is normally used for connecting slaves to Jenkins requires direct SSH access to servers. [3] This is the method that is used for connecting slaves hosted in LF Lab.

Connecting slaves using direct SSH access can become a challenge given that OPNFV Project has number of different labs provided by community as mentioned in previous section. All these labs have different security requirements which can increase the effort and the time needed for connecting slaves to Jenkins. In order to reduce the effort and the time needed for connecting slaves and streamline the process, it has been decided to connect slaves using Java Network Launch Protocol (JNLP).

Connecting Slaves from LF Lab to OPNFV Jenkins

Slaves hosted in LF Lab are handled by LF. All the requests and questions regarding these slaves should be submitted to OPNFV LF Helpdesk.

Connecting Slaves from Community Labs to OPNFV Jenkins

As noted in corresponding section, slaves from Community Labs are connected using JNLP. Via JNLP, slaves open connection towards Jenkins Master instead of Jenkins Master accessing to them directly.

Servers connecting to OPNFV Jenkins using this method must have access to internet.

Please follow below steps to connect a slave to OPNFV Jenkins.

1. Create a user named **jenkins** on the machine you want to connect to OPNFV Jenkins and give the user sudo rights.
2. Install needed software on the machine you want to connect to OPNFV Jenkins as slave.
   - openjdk 8
   - monit
3. If the slave will be used for running virtual deployments, Functest, and Yardstick, install below software and make jenkins user the member of the groups.
   - docker
   - libvirt
4. Create slave root in Jenkins user home directory.

   ```
   mkdir -p /home/jenkins/opnfv/slave_root
   ```

5. Clone OPNFV Releng Git repository.

   ```
   mkdir -p /home/jenkins/opnfv/repos
   cd /home/jenkins/opnfv/repos
   git clone https://gerrit.opnfv.org/gerrit/p/releng.git
   ```

6. Contact LF by creating a ticket to Connect my 3rd party CI/Lab Include the following information in your ticket.

- Slave root (/home/jenkins/opnfv/slave_root)
- Public IP of the slave (You can get the IP by executing `curl http://icanhazip.com/`)
- PGP Key (attached to the mail or exported to a key server)

7. Once you get confirmation from LF stating that your slave is created on OPNFV Jenkins, check if the firewall on LF is open for the server you are trying to connect to Jenkins.

```
cp /home/jenkins/opnfv/repos/releng/utils/jenkins-jnlp-connect.sh /home/jenkins/
cd /home/jenkins/
sudo ./jenkins-jnlp-connect.sh -j /home/jenkins -u jenkins -n <slave name on␣
↪OPNFV Jenkins> -s <the token you received from LF> -f
```

   - If you receive an error, follow the steps listed on the command output.

8. Run the same script with test(-t) on foreground in order to make sure no problem on connection. You should see **INFO: Connected** in the console log.

   ```
   sudo ./jenkins-jnlp-connect.sh -j /home/jenkins -u jenkins -n <slave name
   on OPNFV Jenkins> -s <the token you received from LF> -t
   ```

   - If you receive an error similar to the one shown on this link, you need to check your firewall and allow outgoing connections for the port.

9. Kill the Java slave.jar process.

10. Run the same script normally without test(-t) in order to get monit script created.

    ```
    sudo ./jenkins-jnlp-connect.sh -j /home/jenkins -u jenkins -n <slave name
    on OPNFV Jenkins> -s <the token you received from LF>
    ```

11. Edit monit configuration and enable http interface. The file to edit is /etc/monit/monitrc on Ubuntu systems. Uncomment below lines.

```
set httpd port 2812 and
    use address localhost  # only accept connection from localhost
    allow localhost        # allow localhost to connect to the server and
```

12. Restart monit service.

    - Without systemd:

      ```
      sudo service monit restart
      ```

    - With systemd: you have to enable monit service first and then restart it.

      ```
      sudo systemctl enable monit
      sudo systemctl restart monit
      ```

13. Check to see if jenkins comes up as managed service in monit.

    ```
    sudo monit status
    ```

14. Connect slave to OPNFV Jenkins using monit.

    ```
    sudo monit start jenkins
    ```

15. Check slave on OPNFV Jenkins to verify the slave is reported as connected.

    - The slave on OPNFV Jenkins should have some executors in “Idle” state if the connection is successful.

Notes

PGP Key Instructions

Public PGP Key can be uploaded to public key server so it can be taken from there using your mail address. Example command to upload the key to key server is

```
gpg --keyserver hkp://keys.gnupg.net:80 --send-keys XXXXXXX
```

The Public PGP Key can also be attached to the email by storing the key in a file and then attaching it to the email.

```
gpg --export -a '<your email address>' > pgp.pubkey
```

References

- What is Jenkins
- Jenkins Terminology
- Jenkins SSH Slaves Plugin

3.1.2 Jenkins User Guide

TBD

3.1.3 Creating/Configuring/Verifying Jenkins Jobs

Clone and setup the repo:

```
git clone --recursive ssh://YOU@gerrit.opnfv.org:29418/releng
cd releng
git review -s
```

Make changes:

```
git commit -sv
git review
remote: Resolving deltas: 100% (3/3)
remote: Processing changes: new: 1, refs: 1, done
remote:
remote: New Changes:
remote:   https://gerrit.opnfv.org/gerrit/<CHANGE_ID>
remote:
To ssh://YOU@gerrit.opnfv.org:29418/releng.git
 * [new branch]      HEAD -> refs/publish/master
```

Test with tox:

```
tox -e jjb
```

Note: You can also test the jobs under a single jjb directory by specifying the directory. For example to test only the releng jobs, you could run:

tox -e jjb – jjb/global:jjb/global-jjb:jjb/releng

Submit the change to gerrit:

```
git review -v
```

Follow the link given in the stdoutput to gerrit eg: `https://gerrit.opnfv.org/gerrit/<CHANGE_ID>` the verify job will have completed and you will see Verified +1 jenkins-ci in the gerrit ui.

If the changes pass the verify job https://build.opnfv.org/ci/job/releng-verify-jjb/ , the patch can be submitited by a committer.

Job Types

- Verify Job
  - Trigger: **recheck** or **reverify**
- Merge Job
  - Trigger: **remerge**
- Experimental Job
  - Trigger: **check-experimental**

The verify and merge jobs are retriggerable in Gerrit by simply leaving a comment with one of the keywords listed above. This is useful in case you need to re-run one of those jobs in case if build issues or something changed with the environment.

The experimental jobs are not triggered automatically. You need to leave a comment with the keyword list above to trigger it manually. It is useful for trying out experimental features.

Note that, experimental jobs skip vote for verified status, which means it will reset the verified status to 0. If you want to keep the verified status, use **recheck-experimental** in commit message to trigger both verify and experimental jobs.

You can look in the releng/INFO file for a list of current committers to add as reviewers to your patch in order to get it reviewed and submitted.

Or Add the group releng-contributors

Or just email a request for review to helpdesk@opnfv.org

The Current merge and verify jobs for jenkins job builder can be found in releng-jobs.yaml.

3.1.4 Jenkins Node Labels

TBD

3.2 Source Control and Code Review

Gerrit

3.2.1 Gerrit User Guide

3.3 Artifact and Image Repositories

Google Storage & Docker Hub

3.3.1 Artifact Repository

TBD

3.3.2 Docker Hub

TBD

3.4 Issue and Bug Tracking

JIRA

3.4.1 JIRA User Guide

TBD

3.5 Dashboards and Analytics

- Pharos Dashboard
- Test Results
- Bitergia Dashboard