Octopus Project

Release arno.2015.1.0 (b732ec9)

OPNFV

February 01, 2016

CONTENTS

Contents:

CHAPTER

ONE

PROJECT: CONTINUOUS INTEGRATION (OCTOPUS)

1.1 Introduction

1.1.1 Problem Statement:

OPNFV will use many upstream open source projects to create the reference platform. All these projects are developed and tested independently and in many cases, not have use cases of other projects in mind. Therefore it is to be expected that integration of these projects probably will unveil some gaps in functionality, since testing the OPNFV use cases needs the interworking of many upstream projects. Thus this integration work will bring major benefit to the community.

Therefore the goal of the CI project – Octopus – is to quickly provide prototype integration of a first set of upstream projects. Step by step this later will be evolved to a full blown development environment with automated test and verification as a continuous integration environment, supporting both, the parallel evolutionary work in the upstream projects, and the improvement of NFV support in this reference platform.

1.1.2 Summary

The CI project provides the starting point for all OPNFV development activities. It starts by integrating stable versions of basic upstream projects, and from there creates a full development environment for OPNFV including automatic builds and basic verification. This is a very complex task and therefore needs a step by step approach. At the same time it is urgent to have a basic environment in place very soon.

- **Create a hierarchical build environment** for the same integrated upstream projects as "getstarted",

that uses the build tools as defined by each of the upstream projects and combines them. This allows development and verification in OPNFV collaborative projects.

- **Implement automatic build process on central servers** - Provide automation and periodic builds
- **Execute the continuous automated builds and basic verification**

CHAPTER

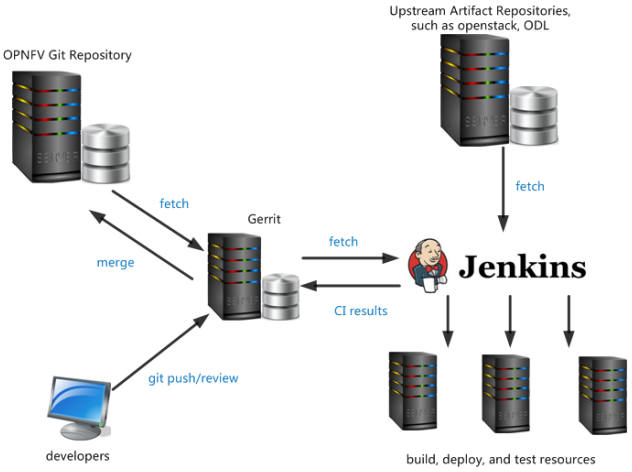
TWO

OPNFV CONTINUOUS INTEGRATION INFRASTRUCTURE

This document covers the Continuous Integration(CI) infrastructure used in the day to day operation of the OPNFV, which may be of interest to people who want to help develop this infrastructure or integrate their tools into it.

2.1 Infra-overview

The OPNFV CI infrastructure includes hardware/tools to check, build, deploy and test, etc, in pipeline. Below diagram shows the hardware/tools resources used in Octopus project,



The table below lists the tools/resources that are used in OPNFV CI,

Tools/Resources	Name
CI Engine	Jenkins
Source Code Management(SCM)	Git
Code Review	Gerrit
Artifact Repository	Google Storage
Hardware Resources	servers supplied by LF

The hardware resources are located in Linux Foundation(LF) lab and community labs, for more details, please read the Pharos project, which is shown in https://wiki.opnfv.org/pharos, a guide is provided to describe how to connect your hosting to LF Jenkins, shown in https://wiki.opnfv.org/octopus/jenkins_slave_connection.

2.2 Major Infrastructures

2.2.1 Git/Gerrit

Git is the famous open source distributed version control software, it was initially designed and developed for Linux Kernel development, and has become the most widely adopted version control system for software development. Git puts emphasis on speed, data integrity, and support for distributed, non-linear workflows.

Jenkins git plugin is used as the Source Code Manager(SCM), in some way, the Git features make it stand out apart from nearly every other SCM, for more details, please refer to http://git-scm.com/about/. For developers, you can refer to http://git-scm.com/docs for its usage.

Gerrit is used here for facilitating online code reviews for our git version control system. As a reviewer, you can see the changes are shown in a side-by-side style and add some inline comments.

2.2.2 Jenkins and JJB

Jenkins is a Continuous Integration system that runs tests and automates some parts of project operations. Jenkins mainly focuses on building/testing software projects continuously and monitoring executions of externally-run jobs. Upstream documentation is available at https://jenkins-ci.org/.

Jenkins supports plugins, which allows to be extended to meet specific requirements. Numbers of plugins have been installed, and new ones can be installed when requirements arise.

The Jenkins jobs are defined by Jenkins Job Builder(JJB) in human readable YAML format. The jobs defined are the key points of CI pipeline. To make clear how the jobs in pipeline run to complete the build, deploy, test works, you can refer to https://wiki.opnfv.org/octopus/pipelines. Moreover, to start your own job in Jenkins, you can write a JJB under the guide of https://wiki.opnfv.org/octopus/jenkins_wow.

2.2.3 Artifact Repository

An artifact repository is a collection of binary software artifacts and metadata stored in a defined directory structure, it is a kin to what subversion is to source code, i.e., it is a way of versioning artifacts produced by build systems, CI, and so on. At this moment, since there is not enough storage space of LF environment, Google Cloud Storage is used as the OPNFV artifact repository temporarily.

If you want to further find out what the artifact repository is and how to use OPNFV artifact repository, the wiki link https://wiki.opnfv.org/octopus/artifact_repository provides a good reference.

CHAPTER

THREE

OPNFV CI PIPELINE

3.1 OPNFV CI

OPNFV Continuous Integration (CI) project provides the starting point for all OPNFV development activities. It creates a full development environment for OPNFV including automatic build, deployment, and testing.

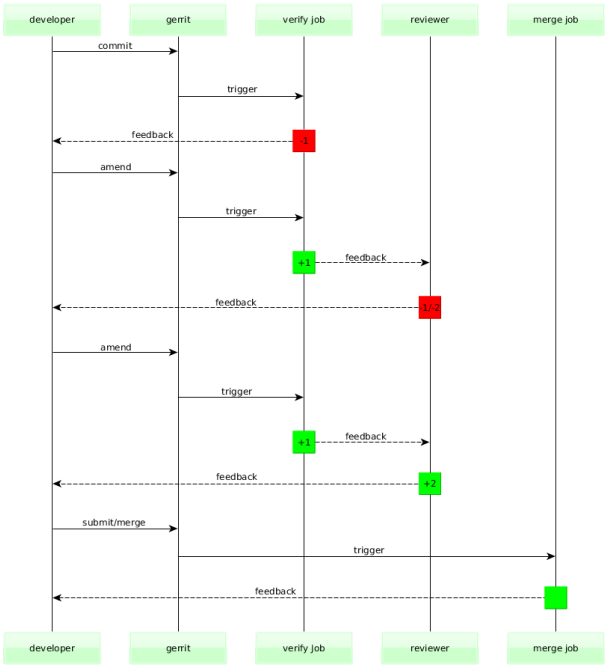
In order to provide fast and continuous feedback to OPNFV community, CI pipeline utilizes different tools, runs different type of verification activities in different phases depending on the needs of different OPNFV projects and the needs of the OPNFV community.

This document aims to provide information regarding OPNFV CI Pipeline which is currently being enabled for the projects.

3.2 CI Pipeline Overview

OPNFV CI Pipeline starts with a change (commit) and stages in the pipeline are triggered based on events that happen while the change travels through the pipeline, producing feedback based on different verification activities.

Below diagram shows overview of the OPNFV CI pipeline.



Please note that the daily job is neglected on above diagram as the daily job is currently triggered once during night time based on timer, not based on Gerrit events.

3.3 Jenkins Jobs in CI Pipeline

3.3.1 Verify Jenkins Job

OPNFV CI Pipeline has **verify** jobs for all OPNFV Projects in order to run quick verification activities for each and every patchset sent to Gerrit for review.

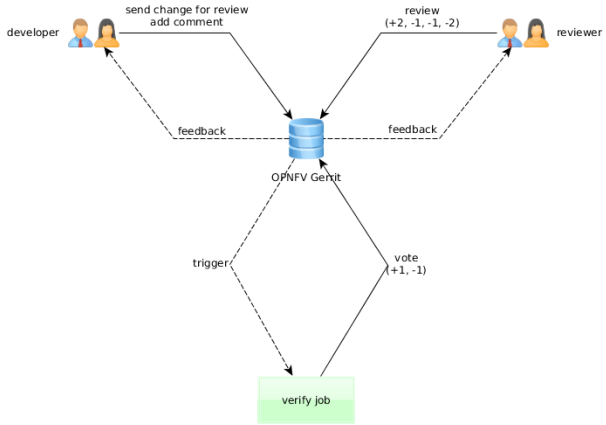
The main purpose of this job is to keep the quality of codebase on certain level so whoever clones the repo at any given time can get *stable* version of the software. It also provides feedback regarding the quality of the patchset to developer who submitted the patchset for review, reviewer(s) who are requested to do review(s) and the rest of the OPNFV community, as early as possible.

This job is triggered automatically when developers issue **git review** command to publish their changes to Gerrit. Gerrit then publishes **patchset created** event under normal circumstances, triggering the job. If the job fails to trigger or fails during execution for some reason that is not related to patchset itself, developers can retrigger it by adding a new comment to change on Gerrit and include either one of the keywords **recheck** or **reverify**.

The result of this job will be verified/failed vote (+1 or -1) on Gerrit. Depending on reviews, the commit can later be submitted to master and merged.

This job does not produce any artifact (document, ISO, etc.).

Please check the diagram below to see how the flow looks.



3.3.2 Merge Jenkins Job

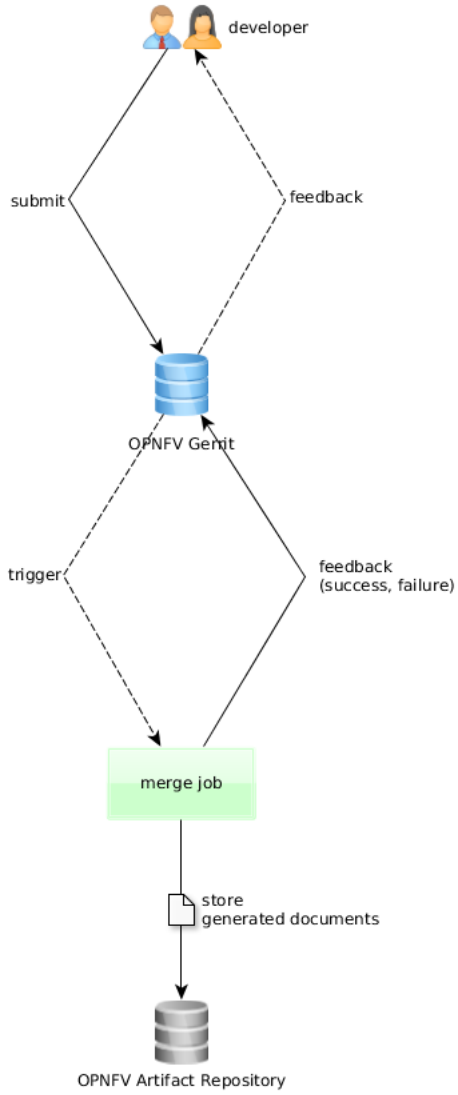
OPNFV CI Pipeline has **merge** jobs for all OPNFV Projects in order to run verification activities for each and every change that gets merged to master.

The main purpose of this job is to give feedback regarding the quality of the master branch once a certain change gets merged to master and the current scope of the job is same as verify job.

This job is triggered automatically by Gerrit **change merged** event under normal circumstances. If the job fails to trigger or fails during execution for some reason that is not related to patchset itself, developers can retrigger it by adding a new comment to change on Gerrit and include the keyword **remerge**.

This job currently produces documents and publishes them on OPNFV Artifact Repository.

Please check the diagram below to see how the flow looks.



3.3.3 Daily Jenkins Job

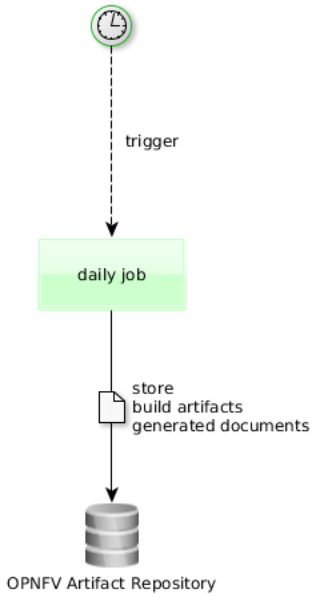
OPNFV CI Pipeline has **daily** jobs for all OPNFV Projects in order to run more extensive verification activities that can take long time to finish.

The main purpose of this job is to run full build, produce artifacts (installer ISOs, etc.), store artifacts in OPNFV Artifact Repository so they can be downloaded to target environment, do the deployment using build artifacts, run tests (Tempest, Robot, etc.) and give feedback regarding the quality of the master branch.

This job is triggered automatically every night 00:00UTC based on **timer** under normal circumstances. If the job fails to trigger or fails during execution for some reason that is not related to software itself, it can only be retriggered by LF aor Octopus team members.

This job produces build artifacts and documents and publishes them on OPNFV Artifact Repository.

Please check the diagram below to see how the flow looks.


trigger
daily job
store
build artifacts
generated documents
OPNFV Artifact Repository

CHAPTER

FOUR

CONNECTING OPNFV COMMUNITY LABS TO OPNFV JENKINS

Table of Contents

4.1 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.

4.2 License

4.3 Version History

Date	Ver.	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

4.4 Jenkins

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

Linux Foundation (LF) hosts and operates OPNFV Jenkins.

4.5 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 loads to slaves. [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.

4.6 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).

4.6.1 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.

4.6.2 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 ticket by sending mail to OPNFV LF Helpdesk first, opnfv-helpdesk@rt.linuxfoundation.org.
2. Ensure DNS is setup for your public IP addresses: DNS A records and PTR records need to be matching.
3. Create a local user on server you want to connect to OPNFV Jenkins. (named **jenkins** for example)

4. Download slave.jar using https://build.opnfv.org/ci/jnlpJars/slave.jar and place it to somewhere so jenkins user created in previous step can access. 5. Create a directory /home/jenkins/opnfv_slave_root. 6. Contact LF by sending mail to opnfv-helpdesk@rt.linuxfoundation.org as getting the server connected requires help from LF. 7. Provide needed information to LF including the IP of the server, name of the slave, slave root, and your Public PGP key in order for LF to pass credentials to you securely. Please see the notes section for details regarding how to pass your Public PGP Key.

Slave IP: x.x.x.x

Slave name: company-build

Slave Root: /home/jenkins/opnfv_slave_root

PGP Key: (attached, or exported to key server)

8. LF will provide you the key/token you need to use.
9. Try to see if you can establish connection towards OPNFV Jenkins by using below command.

```
java -jar slave.jar -jnlpUrl https://build.opnfv.org/ci/computer/<slave_name>/slave-agent.j
-secret <token>
```

10. Navigate to OPNFV Jenkins and look for your slave. It should have some executors in "Idle" state if the connection is successful. 11. Once you reach this step, you have the server connection to OPNFV Jenkins completed. You can script the command you used above so the connection between slave and Jenkins can be kept open.

4.7 Notes

4.7.1 Keeping the slave.jar Up to Date

It is important to keep the slave.jar up to date since OPNFV Jenkins version may be updated any time. In order to make sure you are using compatible version of slave.jar, you can download it from https://build.opnfv.org/ci/jnlpJars/slave.jar whenever you reopen the connection towards OPNFV Jenkins. You may experience random disconnects if you do not do this regularly.

4.7.2 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
```

4.8 References

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

CHAPTER

FIVE

HOW TO WRITE AND USE JJB?

5.1 What is Jenkins Job Builder

Jenkins Job Builder(JJB) takes simple descriptions of Jenkins jobs in YAML format, and uses them to configure Jenkins jobs. JJB keeps your job descriptions in human readable format and job template system simplifies the configuration of Jenkins jobs. Upstream documentation is available at http://ci.openstack.org/jenkins-job-builder/.

5.2 How to Write/Use Jenkins Job Builder

Job template is widely used in JJBs and makes the configuration of Jenkins jobs simple, if you need to define several jobs which are nearly identical, except perhaps in their names, SCP targets, etc., then you may use a job template to specify the particulars of the job, and then use a project to realize the job with appropriate variable substitution.

To illustrate how to configure Jenkins jobs by using job template, we can start with a simple example used in our OPNFV releng project octopus directory, which is just used to print "Hello world from Octopus", shown as below:

```
-job-template:
   name: octopus-test

   node: master

   project-type: freestyle

   logrotate:
       daysToKeep: 30
       numToKeep: 10
       artifactDaysToKeep: -1
       artifactNumToKeep: -1

   builders:
       - shell: |
           echo "Hello world from octopus"
```

the value "-1" here means keep forever, you can add this job template into the project jobs to run in jenkins:

```
- project:
    name: octopus
    jobs:
        - 'octopus-test'
```

then this job works!

Further, if you are a developer who wants to set up a job template to be used in jenkins, we should dive into much more, taking one job template in releng project used for octopus project, the 'octopus-daily-{stream}', as an example:

```
-job-template:
   name: 'octopus-daily-{stream}'

   node: master

   # Job template for daily builders
   #
   # Required Variables:
   #     stream:    branch with - in place of / (eg. stable)
   #     branch:    branch (eg. stable)

   project-type: freestyle
   varsetabove: '{somevar}'

   logrotate:
       daysToKeep: '{build-days-to-keep}'
       numToKeep: '{build-num-to-keep}'
       artifactDaysToKeep: '{build-artifact-days-to-keep}'
       artifactNumToKeep: '{build-artifact-num-to-keep}'

   parameters:
       - project-parameter:
           project: '{project}'

   scm:
       - git-scm:
           credentials-id: '{ssh-credentials}'
           refspec: ''
           branch: '{branch}'

   wrappers:
       - ssh-agent-credentials:
           user: '{ssh-credentials}'

   triggers:
       - timed: 'H H * * *'

   prebuilders:
       - test-macro

   builders:
       - shell:
           !include-raw build-upload-docu.sh

   postbulders:
       - test-macro
```

the {stream} here means when you add this into jobs, you can replace {stream} with what you want, such as:

```
stream:
       - master:
           branch: 'master'
```

the:

```
node: master
```

means to restrict this job to run in Jenkins master node. Next, several important procedures are illustrated here,

- scm, this mudule allows you to specify the source code location for the project,

and it allows referencing multiple repositories in a Jenkins job. - triggers, this module defines what causes a Jenkins job to start building. - prebuilders and postbuilders, which define job need done pre and post the builders. - builders, which defines actions that the Jenkins job should execute, usually the shell scripts or maven targets are existed there, e.g., build-upload-docu.sh used in our example.

Generally, the modules used in a job template is sequenced as

1. parameters, properties
2. scm
3. triggers
4. wrappers
5. prebuilders
6. builders
7. postbuilders
8. publishers, reporters, notifications

5.3 Working with OPNFV Jenkins Jobs

By now, the releng project of OPNFV is the release engineering project for JJBs, you can clone the repo:

```
git clone ssh://YOU@gerrit.opnfv.org:29418/releng
```

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/51
remote:
To ssh://agardner@gerrit.opnfv.org:29418/releng.git
 "* [new branch]      HEAD -> refs/publish/master
```

Follow the link to gerrit https://gerrit.opnfv.org/gerrit/51 in a few moments 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/view/builder/job/builder-verify-jjb/ The patch can be submitted by a committer.

The verify and merge jobs are retriggerable in Gerrit by simply leaving a comment with one of the keywords listed below. 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.

- Verify Job: Trigger: **recheck** or **reverify**
- Merge Job: Trigger: **remerge**

You can add below persons as reviewers to your patch in order to get it reviewed and submitted.

- Ulrich Kleber (Ulrich.Kleber@huawei.com)
- Fatih Degirmenci (fatih.degirmenci@ericsson.com)
- Xinyu Zhao(Jerry) (zhaoxinyu@huawei.com)
- Aric Gardner (agardner@linuxfoundation.org)

The Current merge and verify jobs for jenkins job builder for releng project, shown in https://git.opnfv.org/cgit/releng/tree/jjb.

Assuming that you have set up some job templates and put them into a project, then the question is that how they work? Taking the jobs 'builder-verify-jjb', 'builder-merge' used in releng project as examples, 'builder-verify-jjb' is to verify jobs you commited, you will see verified '+1' jenkins-ci in gerrit if it succeed, 'builder-merge' is to set up a merge job and update all the JJBs. If you have some new jobs need to be run, you can set up your own job templates and add them into the project.

CHAPTER

SIX

OPNFV ARTIFACT REPOSITORY

6.1 Artifact Repository

6.1.1 What is Artifact Repository

An Artifact Repository is akin to what Subversion is to source code, i.e. it is a way of versioning artifacts produced by build systems, CI, and so on. [1]

6.1.2 Why Artifact Repository is Needed

Since many developers check their source code into the GIT repository it may seem natural to just place the files you've built into the repo too. This can work okay for a single developer working on a project over the weekends but with a team working on many components that need to be tested and integrated, this won't scale.

The way git works, no revision of any file is ever lost. So if you ever check in a big file, the repository will always contain it, and a git clone will be that much slower for every clone from that point onward.

The golden rule of revision control systems applies: *check in your build scripts, not your build products*.

Unfortunately, it only takes one person to start doing this and we end up with huge repositories. Please don't do this. It will make your computers sad. Thankfully, Gerrit and code review systems are a massive disincentive to doing this.

You definitely need to avoid storing binary images in git. This is what artifact repositories are for. [2]

A "centralized image repository" is needed that can store multiple versions of various virtual machines and have something like /latest pointing to the newest uploaded image. It could be a simple nginx server that stores the output images from any jenkins job if it's successful, for instance.

6.2 OPNFV Artifact Repository

6.2.1 What is used as Artifact Repository for OPNFV

Setting up, hosting, and operating an artifact repository on OPNFV Infrastructure in Linux Foundation (LF) environment requires too much storage space. It is also not a straightforward undertaking to have robust Artifact Repository and provide 24/7 support.

OPNFV Project decided to use **Google Cloud Storage** as OPNFV Artifact Repository due to reasons summarized above. [3]

6.2.2 Usage of Artifact Repository in OPNFV CI

Binaries/packages that are produced by OPNFV Continuous Integration (CI) are deployed/uploaded to Artifact Repository making it possible to reuse artifacts during later stages of OPNFV CI. Stored artifacts can be consumed by individual developers/organizations as well.

In OPNFV, we generally produce PDF, ISO and store them on OPNFV Artifact Repository.

6.2.3 OPNFV Artifact Repository Web Interface

OPNFV Artifact Repository is accessible via link http://artifacts.opnfv.org/.

A proxy has been set up by LF for the community members located in countries with access restrictions to Google http://build.opnfv.org/artifacts/.

6.3 Access Rights to OPNFV Artifact Repository

As summarized in previous sections, OPNFV uses Google Cloud Storage as Artifact Repository. By default, everyone has read access to it and artifacts can be fetched/downloaded using browser, a curl-like command line HTTP client, or gsutil.

Write access to Artifact Repository is given per request basis and all the requests must go through LF Helpdesk with an explanation regarding the purpose of write access. Once you are given write access, you can read corresponding section to store artifacts on OPNFV Artifact Repository.

6.4 How to Use OPNFV Artifact Repository

There are 3 basic scenarios to use OPNFV Artifact repository.

- browsing artifacts
- downloading artifacts
- uploading artifacts

Please see corresponding sections regarding how to do these.

6.4.1 How to Browse Artifacts Stored on OPNFV Artifact Repository

You can browse stored artifacts using

- **Web Browser**: By navigating to the address OPNFV Artifact Storage.
- **Command Line HTTP-client** `curl -o <output_filename> http://artifacts.opnfv.org`

  Example:

  ```
  curl -o opnfv-artifact-repo.html http://artifacts.opnfv.org
  ```

- **Google Storage Util (gsutil)** `gsutil ls gs://artifacts.opnfv.org/<path_to_bucket>`

  Example:

  ```
  gsutil ls gs://artifacts.opnfv.org/octopus
  ```

6.4.2 How to Download Artifacts from OPNFV Artifact Repository

You can download stored artifacts using

- **Web Browser**: By navigating to the address OPNFV Artifact Storage and clicking the link of the artifact you want to download.
- **Command Line HTTP-client** `curl -o <output_filename> http://artifacts.opnfv.org/<path/to/art`

  Example:

  ```
  curl -o main.pdf http://artifacts.opnfv.org/octopus/docs/release/main.pdf
  ```

- **Google Storage Util (gsutil)** `gsutil cp gs://artifacts.opnfv.org/<path/to/artifact> <output_filename>`

  Example:

  ```
  gsutil cp gs://artifacts.opnfv.org/octopus/docs/release/main.pdf
  main.pdf
  ```

6.4.3 How to Upload Artifacts to OPNFV Artifact Repository

As explained in previous sections, you need to get write access for OPNFV Artifact Repository in order to upload artifacts.

Apart from write access, you also need to have Google account and have the Google Cloud Storage utility, **gsutil**, installed on your computer.

Install and Configure gsutil

Please follow steps listed below.

1. Install gsutil

   Please follow steps listed on this link to install gsutil to your computer.

2. Configure gsutil

   Issue below command and follow the instructions. You will be asked for the project-id.

The project-id is **linux-foundation-collab**.

```
gsutil config
```

3. Request write access for OPNFV Artifact Repository

   Send an email to LF Helpdesk and list the reasons for the request. Do not forget to include gmail mail address.

Upload Artifacts

Once you installed and configured gsutil and got write access from LF Helpdesk, you should be able to upload artifacts to OPNFV Artifact Repository.

The command to upload artifacts is

```
gsutil cp <file_to_upload> gs://artifacts.opnfv.org/<path/to/bucket>
```

Example:

```
gsutil cp README gs://artifacts.opnfv.org/octopus
```

Once the upload operation is completed, you can do the listing and check to see if the artifact is where it is expected to be.

```
gsutil ls gs://artifacts.opnfv.org/<path/to/bucket>
```

Example:

```
gsutil ls gs://artifacts.opnfv.org/octopus
```

6.5 Getting Help

Send an email to LF Helpdesk or join the channel **#opnfv-octopus** on IRC.

6.5.1 References

1. Why you should be using an Artifact Repository
2. Regarding VM image and Git repo
3. Google Cloud Storage

CHAPTER

SEVEN

STABLE BRANCH

This document describes the way of working with stable branches and doing maintenance.

The page is derived from https://wiki.openstack.org/wiki/StableBranch , simplified and adapted to OPNFV.

It was discussed on TSC on June 30, 2015 with minor comments.

At this time only Arno release maintenance is covered.

7.1 Overview

The stable branch is intended to be a safe source of fixes for high impact bugs and security issues which have been fixed on master since a given release. It allows users of release (stable) versions to benefit from the ongoing bugfix work after the release.

Official point releases for each project are published from the branch on a per need basis, as decided by the TSC. In later stages, a regular cadence for point releases may be introduced.

It's possible to check current maintained versions in the releases page. At this time only Arno is maintained.

OPNFV's stable branch policy borrows much from prior art, in particular from OpenStack.

In general all fixes should be made on the main branch and cherry picked to stable. If there is a case where the fix is not able to be merged backwards only then we would need to do any work directly on stable. The documented method for getting a fix into stable should be by a cherry-pick process.

7.2 Stable branch policy

7.2.1 Appropriate fixes

Only a limited class of changes are appropriate for inclusion on the stable branch.

A number of factors must be weighed when considering a change:

- **The risk of regression** - even the tiniest changes carry some risk of breaking something and we really want to avoid regressions on the stable branch
- **The user visible benefit** - are we fixing something that users might actually notice and, if so, how important is it?
- **How self-contained the fix is** - if it fixes a significant issue but also refactors a lot of code, it's probably worth thinking about what a less risky fix might look like

- Whether the fix is **already on master - a change must be a **backport** of a change already merged onto master, unless the change simply does not make sense on master (e.g. because of a change of architecture).
- If there is a suitable **work-around** for a bug, normally there won't be a fix on stable.
- Since OPNFV is using several upstream projects, typically fixes in the upstream projects

will apply as fixes for OPNFV. Therefore complete **maintenance revisions of the upstream projects** (i.e. minor versions) can be used as suitable backports for OPNFV maintenance releases. - Since OPNFV is a midstream integration effort, also test cases might be suitable backports

in case they are related to critical bugs found in stable.

Rules to maintain multiple versions and exceptions will be added later.

The stable-mtc team needs to balance the risk of any given patch with the value that it will provide to users of the stable branch. A large, risky patch for a major data corruption issue might make sense. As might a trivial fix for a fairly obscure error handling case.

The stable-mtc team also will handle exceptions, of which the most common will be that a fix on stable needs to be different to the fix on master due to some other changes on master (e.g. architectural).

Some types of changes are completely forbidden:

- New features
- Changes to the external APIs
- Changes to the notification definitions
- DB schema changes
- Incompatible config file changes
- Changes including a version upgrade of an upstream component of OPNFV (since this will typically violate the above points)

7.2.2 Support phases

Support phases will be introduced at a later time

7.2.3 Review of fixes

Each backported commit proposed to gerrit should be reviewed and +2ed by two Arno-stable-maint members before it is approved. Where a stable-maint member has backported a fix, a single other +2 is sufficient for approval.

If unsure about the technical details of a given fix, stable-maint members should consult with the appropriate developers from the affected projects for a more detailed technical review.

If unsure if a fix is appropriate for the stable branch, at this time the TSC will do the final decision.

7.2.4 Security fixes

Fixes for embargoed security issues receive special treatment. These should be reviewed in advance of disclosure by committers and stable-maint. At the time of coordinated public disclosure, the fix is proposed simultaneously to master and the stable branches and immediately approved.

7.3 Processes

7.3.1 Proposing fixes

Anyone can propose a cherry-pick to the stable-maint team.

One way is that if a bugfix on master looks like a good candidate for backporting - e.g. if it's a significant bug with the revious release - then just nominating the bug for Arno maintenance will bring it to the attention of the maintainers.

If you don't have the appropriate permissions to nominate the bug, then send an email via the user list.

The best way to get the patch merged in timely manner is to send it backported by yourself. To do so, you may try to use "Cherry Pick To" button in Gerrit UI for the original patch in master. Gerrit will take care of creating a new review, modifying commit message to include 'cherry-picked from ?' line etc.

If the patch you're proposing will not cherry-pick cleanly, you can help by resolving the conflicts yourself and proposing the resulting patch. Please keep Conflicts lines in the commit message to help reviewers! You can use git-review to propose a change to the stable branch with:

```
$> git log (find out the commit id of the patch that you want to backport from "git log" output)
$> git checkout stable/arno
$> git cherry-pick -x $master_commit_d
$> git review stable/arno
```

Note: cherry-pick -x option includes 'cherry-picked from ?' line in the commit message which is required to avoid Gerrit bug

Failing all that, just ping one of the team and mention that you think the bug/commit is a good candidate.

7.3.2 Change-Ids

When cherry-picking a commit, keep the original Change-Id and gerrit will show a separate review for the stable branch while still allowing you to use the Change-Id to see all the reviews associated with it.

Hint: Change-Id line must be in the last paragraph. Conflicts in the backport: add a new paragraph, creating a new Change-Id but you can avoid that by moving conflicts above the paragraph with Change-Id line or removing empty lines to make a single paragraph.

7.3.3 Email Notifications

If you want to be notified of these patches you can create a watch on this screen: https://gerrit.opnfv.org/gerrit/#/settings/projects click "Watched Projects"

Project Name: All-Projects

Only If: branch:stable/arno

Then check the "Email Notifications - New Changes" checkbox. That will cause gerrit to send an email whenever a matching change is proposed, and better yet, the change shows up in your 'watched changes' list in gerrit.

7.3.4 Bug Tags

will be introduced when we see the need.

7.3.5 CI Pipeline

For Arno release the jobs will be run once per day per installer (Fuel and Foreman) on stable/arno branch. Since this is in addition to the jobs for master branch and jobs have long run time, this might need re-evaluation as we go on.

The artifacts arno/stable jobs produced are stored in the new directoies on artifacts.opnfv.org.

The artifacts produced by daily jobs would be stored For stable/arno, the storage locations will be <project_name>/arno/<artifact_name>.iso

The docs produced by daily and merge jobs would be stored For stable/arno, the storage locations will be <project_name>/arno/docs/<document_name>

No changes in overall functionality in merge and verify jobs: they will continue doing builds only

```
genesis-fuel-verify-master, genesis-fuel-verify-stable-arno,
genesis-fuel-merge-master, genesis-fuel-merge-stable-arno,
genesis-foreman-verify-master, genesis-foreman-verify-stable-arno,
genesis-foreman-merge-master, genesis-foreman-merge-stable-arno
```

7.4 Team organization

7.4.1 Project specific tasks

Each of the 5 projects that contributed to Arno will dedicate some committers which would be in charge of reviewing backports for their project, following the stable branch policy. It is in the responsibility of each project how to select those committers (e.g. vote in the team).

The group of these committers here are sometimes called the "stable branch maintenance team" or "stable-mtc" without this being necessarily a team with own organization.

7.4.2 Stable branch management

Stable branches are less exercised than master branches, and they may get broken by external events.

Therefore a group of committers, the "stable branch maintenance team" (stable-mtc) is tasked with specific stable branch support, making sure the branch stays in good shape and remains usable at all times. They monitor periodic jobs failures and enlist the help of others in order to fix the branches in case of breakage. They should also raise flags if for some reason they are blocked and don't receive enough support, in which case early abandon of the branch will be considered.

The stable-mtc is responsible for the enforcement of the Stable Branch policy. Initially it is composed of a release manager and at least one committer of each of the participating projects and will have similar organization and rights as OPNFV project teams. It will be granting exceptions for all questionable backports raised by projects, providing backports reviews help everywhere, and educating projects members on the stable branch policy.

The stable-mtc will propose to TSC to decide on point releases from the stable branch. Preparation of the point release will be described in a second step. The stable-mtc can also propose to TSC to decide to abandon maintenance of the release.

When a new OPNFV version is released, a stable-mtc team for that project will start. At that time, the earlier maintenance version will go to a phase with less support. Details will be defined later.

7.4.3 Joining the team

Existing committers are greatly encouraged to join the stable-mtc in order to help with reviewing backports, judging their appropriateness for the stable branch and approving them.

We're really keen to add more folks to the stable Arno stable-mtc to help out with reviews.

All you really need is some time and the ability to apply the "safe source of high impact fixes" and "must be fixed on master first" policies. It mostly comes down to having a good sense of the risk vs benefit of applying a backport to the branch.

If you'd like to join the team, you can start by simply [+-]1ing stable branch reviews. It's best if you can add some brief thoughts to your review on why you think the fix is suitable for stable so that we know how you're applying the policy.

Same as in normal OPNFV projects, stable-mtc team will use committer votes for some decisions like proposing a new point release to TSC.