Functest Documentation

Release master

Functest <opnfv-tech-discuss@lists.opnfv.org>

Oct 14, 2019

Contents

1 Functest Installation Guide 1

2 Functest User Guide 15

3 Functest Developer Guide 39

4 Functest Release Notes 45

CHAPTER 1

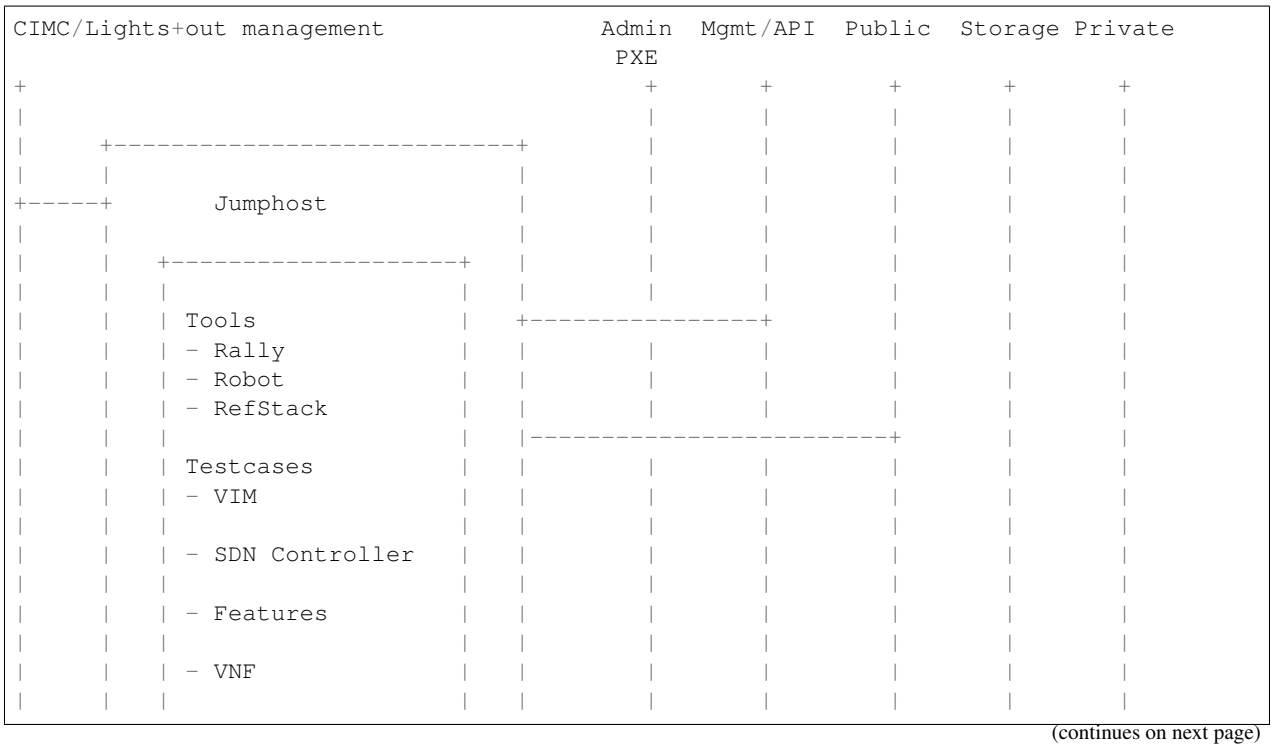
Functest Installation Guide

1.1 Introduction

This document describes how to install and configure Functest in OPNFV.

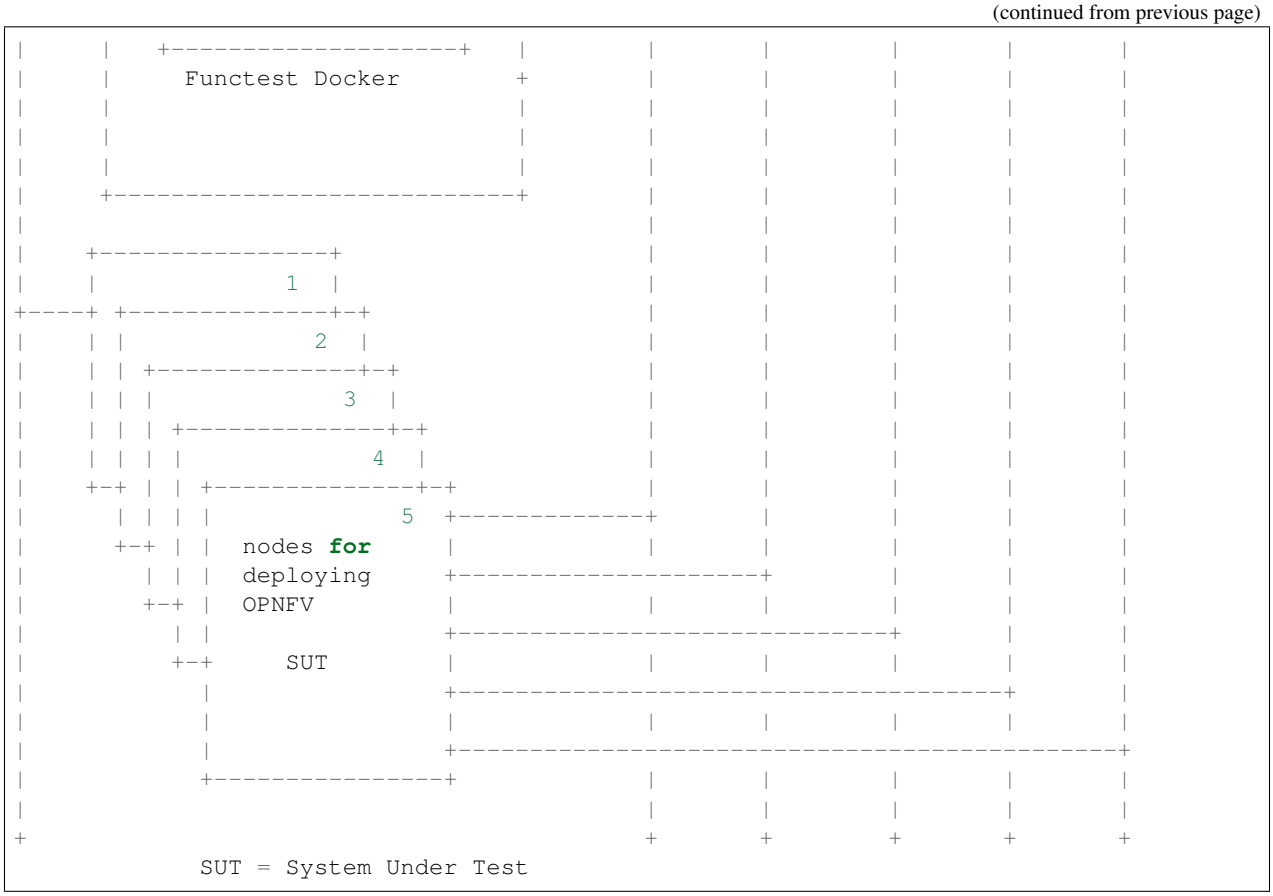
1.1.1 High level architecture

The high level architecture of Functest within OPNFV can be described as follows:



(continues on next page)

(continued from previous page)



All the libraries and dependencies needed by all of the Functest tools are pre-installed into the Docker images. This allows running Functest on any platform.

The automated mechanisms inside the Functest Docker containers will:

- Prepare the environment according to the System Under Test (SUT)
- Perform the appropriate functional tests
- Push the test results into the OPNFV test result database (optional)

The OpenStack credentials file must be provided to the container.

These Docker images can be integrated into CI or deployed independently.

Please note that the Functest Docker images have been designed for OPNFV, however, it would be possible to adapt them to any OpenStack based VIM + controller environment, since most of the test cases are integrated from upstream communities.

The functional test cases are described in the *Functest User Guide*

1.2 Prerequisites

The OPNFV deployment is out of the scope of this document but it can be found in https://docs.opnfv.org/en/stable-hunter/. The OPNFV platform is considered as the SUT in this document.

Several prerequisites are needed for Functest:

1. A Jumphost to run Functest on
2. A Docker daemon shall be installed on the Jumphost
3. A public/external network created on the SUT
4. An admin/management network created on the SUT
5. Connectivity from the Jumphost to the SUT public/external network

WARNING: Connectivity from Jumphost is essential and it is of paramount importance to make sure it is working before even considering to install and run Functest. Make also sure you understand how your networking is designed to work.

NOTE: **Jumphost** refers to any server which meets the previous requirements. Normally it is the same server from where the OPNFV deployment has been triggered previously, but it could be any server with proper connectivity to the SUT.

NOTE: If your Jumphost is operating behind a company http proxy and/or firewall, please consult first the section *Proxy support*, towards the end of this document. The section details some tips/tricks which *may* be of help in a proxified environment.

1.2.1 Docker installation

Docker installation and configuration is only needed to be done once through the life cycle of Jumphost.

If your Jumphost is based on Ubuntu, SUSE, RHEL or CentOS linux, please consult the references below for more detailed instructions. The commands below are offered as a short reference.

Tip: For running docker containers behind the proxy, you need first some extra configuration which is described in section *Docker Installation on CentOS behind http proxy*. You should follow that section before installing the docker engine.

Docker installation needs to be done as root user. You may use other userid's to create and run the actual containers later if so desired. Log on to your Jumphost as root user and install the Docker Engine (e.g. for CentOS family):

```
curl -sSL https://get.docker.com/ | sh
systemctl start docker

*Tip:* If you are working through proxy, please set the https_proxy
environment variable first before executing the curl command.
```

Add your user to docker group to be able to run commands without sudo:

```
sudo usermod -aG docker <your_user>
```

A reconnection is needed. There are 2 ways for this:

1. Re-login to your account
2. su - <username>

References - Installing Docker Engine on different Linux Operating Systems:

- Ubuntu
- RHEL
- CentOS
- SUSE

1.2.2 Public/External network on SUT

Some of the tests against the VIM (Virtual Infrastructure Manager) need connectivity through an existing public/external network in order to succeed. This is needed, for example, to create floating IPs to access VM instances through the public/external network (i.e. from the Docker container).

By default, the five OPNFV installers provide a fresh installation with a public/external network created along with a router. Make sure that the public/external subnet is reachable from the Jumphost and an external router exists.

Hint: For the given OPNFV Installer in use, the IP sub-net address used for the public/external network is usually a planning item and should thus be known. Ensure you can reach each node in the SUT, from the Jumphost using the 'ping' command using the respective IP address on the public/external network for each node in the SUT. The details of how to determine the needed IP addresses for each node in the SUT may vary according to the used installer and are therefore ommitted here.

1.3 Installation and configuration

Alpine containers have been introduced in Euphrates. Alpine allows Functest testing in several very light containers and thanks to the refactoring on dependency management should allow the creation of light and fully customized docker images.

1.3.1 Functest Dockers for OpenStack deployment

Docker images are available on the dockerhub:

- opnfv/functest-core
- opnfv/functest-healthcheck
- opnfv/functest-smoke
- opnfv/functest-benchmarking
- opnfv/functest-vnf

Preparing your environment

cat env:

```
DEPLOY_SCENARIO=XXX  # if not os-nosdn-nofeature-noha scenario
NAMESERVER=XXX  # if not 8.8.8.8
EXTERNAL_NETWORK=XXX # if not first network with router:external=True
NEW_USER_ROLE=XXX # if not member
SDN_CONTROLLER_IP=XXX  # if odl scenario
VOLUME_DEVICE_NAME=XXX  # if not vdb
FLAVOR_EXTRA_SPECS=hw:mem_page_size:large    # if fdio scenarios
```

See section on environment variables for details.

cat env_file:

```
export OS_AUTH_URL=XXX
export OS_USER_DOMAIN_NAME=XXX
export OS_PROJECT_DOMAIN_NAME=XXX
export OS_USERNAME=XXX
```

(continues on next page)

(continued from previous page)

```
export OS_PROJECT_NAME=XXX
export OS_PASSWORD=XXX
export OS_IDENTITY_API_VERSION=3
export OS_REGION_NAME=XXX
```

See section on OpenStack credentials for details.

Create a directory for the different images (attached as a Docker volume):

```
mkdir -p images && wget -q -O- https://git.opnfv.org/functest/plain/functest/ci/
↪download_images.sh | bash -s -- images && ls -1 images/*

images/cirros-0.4.0-aarch64-disk.img
images/cirros-0.4.0-x86_64-disk.img
images/cloudify-docker-manager-community-19.01.24.tar
images/shaker-image-1.3.0+stretch.qcow2
images/ubuntu-14.04-server-cloudimg-amd64-disk1.img
images/ubuntu-14.04-server-cloudimg-arm64-uefi1.img
images/ubuntu-16.04-server-cloudimg-amd64-disk1.img
images/vyos-1.1.8-amd64.qcow2
```

Testing healthcheck suite

Run healthcheck suite:

```
sudo docker run --env-file env \
    -v $(pwd)/openstack.creds:/home/opnfv/functest/conf/env_file \
    -v $(pwd)/images:/home/opnfv/functest/images \
    opnfv/functest-healthcheck
```

Results shall be displayed as follows:

```
+-------------------------+------------------+---------------------+------------------
↪-+----------------+
|        TEST CASE        |     PROJECT      |         TIER        |     DURATION    ⌴
↪ |     RESULT     |
+-------------------------+------------------+---------------------+------------------
↪-+----------------+
|     connection_check    |     functest     |     healthcheck     |       00:02     ⌴
↪ |       PASS     |
|      tenantnetwork1     |     functest     |     healthcheck     |       00:06     ⌴
↪ |       PASS     |
|      tenantnetwork2     |     functest     |     healthcheck     |       00:06     ⌴
↪ |       PASS     |
|         vmready1        |     functest     |     healthcheck     |       00:06     ⌴
↪ |       PASS     |
|         vmready2        |     functest     |     healthcheck     |       00:08     ⌴
↪ |       PASS     |
|        singlevm1        |     functest     |     healthcheck     |       00:28     ⌴
↪ |       PASS     |
|        singlevm2        |     functest     |     healthcheck     |       00:25     ⌴
↪ |       PASS     |
|        vping_ssh        |     functest     |     healthcheck     |       00:36     ⌴
↪ |       PASS     |
|      vping_userdata     |     functest     |     healthcheck     |       00:34     ⌴
↪ |       PASS     |
```

(continues on next page)

(continued from previous page)

```
|       cinder_test         |     functest     |     healthcheck     |       01:03      ␣
↪ |       PASS       |
|      tempest_smoke        |     functest     |     healthcheck     |       05:13      ␣
↪ |       PASS       |
|          odl              |     functest     |     healthcheck     |       00:00      ␣
↪ |       SKIP       |
+---------------------------+------------------+---------------------+------------------
↪-+----------------+
```

NOTE: the duration is a reference and it might vary depending on your SUT.

Testing smoke suite

Run smoke suite:

```
sudo docker run --env-file env \
    -v $(pwd)/openstack.creds:/home/opnfv/functest/conf/env_file \
    -v $(pwd)/images:/home/opnfv/functest/images \
    opnfv/functest-smoke
```

Results shall be displayed as follows:

```
+-----------------------------------+------------------+---------------+-------------
↪-----+----------------+
|             TEST CASE             |     PROJECT      |      TIER     |             ␣
↪DURATION     |     RESULT     |
+-----------------------------------+------------------+---------------+-------------
↪-----+----------------+
|     neutron-tempest-plugin-api    |     functest     |     smoke     |       09:12      ␣
↪     |      PASS      |
|           rally_sanity            |     functest     |     smoke     |       16:29      ␣
↪     |      PASS      |
|         refstack_compute          |     functest     |     smoke     |       06:25      ␣
↪     |      PASS      |
|         refstack_object           |     functest     |     smoke     |       01:54      ␣
↪     |      PASS      |
|        refstack_platform          |     functest     |     smoke     |       06:52      ␣
↪     |      PASS      |
|           tempest_full            |     functest     |     smoke     |       30:26      ␣
↪     |      PASS      |
|         tempest_scenario          |     functest     |     smoke     |       09:23      ␣
↪     |      PASS      |
|           tempest_slow            |     functest     |     smoke     |       24:42      ␣
↪     |      PASS      |
|            patrole                |     functest     |     smoke     |       02:36      ␣
↪     |      PASS      |
|            barbican               |     functest     |     smoke     |       02:13      ␣
↪     |      PASS      |
|          neutron_trunk            |     functest     |     smoke     |       00:00      ␣
↪     |      SKIP      |
|        networking-bgpvpn          |     functest     |     smoke     |       00:00      ␣
↪     |      SKIP      |
|          networking-sfc           |     functest     |     smoke     |       00:00      ␣
↪     |      SKIP      |
|            octavia               |     functest     |     smoke     |       00:00      ␣
↪     |      SKIP      |
```

(continues on next page)

(continued from previous page)

```
+----------------------------------+-----------------+---------------+-------------
↪-----+----------------+
```

Note: if the scenario does not support some tests, they are indicated as SKIP. See User guide for details.

Testing benchmarking suite

Run benchmarking suite:

```
sudo docker run --env-file env \
    -v $(pwd)/openstack.creds:/home/opnfv/functest/conf/env_file \
    -v $(pwd)/images:/home/opnfv/functest/images \
    opnfv/functest-benchmarking
```

Results shall be displayed as follows:

```
+-------------------+------------------+---------------------+------------------+---
↪-------------+
|     TEST CASE     |     PROJECT      |         TIER        |     DURATION     |  ␣
↪  RESULT     |
+-------------------+------------------+---------------------+------------------+---
↪-------------+
|     rally_full    |     functest     |     benchmarking    |       92:16      |  ␣
↪   PASS      |
|     rally_jobs    |     functest     |     benchmarking    |       18:49      |  ␣
↪   PASS      |
|        vmtp       |     functest     |     benchmarking    |       15:28      |  ␣
↪   PASS      |
|       shaker      |     functest     |     benchmarking    |       24:04      |  ␣
↪   PASS      |
+-------------------+------------------+---------------------+------------------+---
↪-------------+
```

Note: if the scenario does not support some tests, they are indicated as SKIP. See User guide for details.

Testing vnf suite

Run vnf suite:

```
sudo docker run --env-file env \
    -v $(pwd)/openstack.creds:/home/opnfv/functest/conf/env_file \
    -v $(pwd)/images:/home/opnfv/functest/images \
    opnfv/functest-vnf
```

Results shall be displayed as follows:

```
+---------------------+------------------+-------------+------------------+---------
↪-------+
|      TEST CASE      |     PROJECT      |     TIER    |     DURATION     |    ␣
↪RESULT     |
+---------------------+------------------+-------------+------------------+---------
↪-------+
|       cloudify      |     functest     |     vnf     |       03:49      |    ␣
↪PASS       |
```

(continues on next page)

(continued from previous page)

```
|     cloudify_ims      |     functest     |     vnf      |       24:20       |  ␣
↪PASS      |
|       heat_ims        |     functest     |     vnf      |       32:13       |  ␣
↪PASS      |
|     vyos_vrouter      |     functest     |     vnf      |       14:55       |  ␣
↪PASS      |
|       juju_epc        |     functest     |     vnf      |       41:24       |  ␣
↪PASS      |
+-----------------------+------------------+--------------+-------------------+---------
↪-------+
```

1.3.2 Functest Dockers for Kubernetes deployment

Docker images are available on the dockerhub:

- opnfv/functest-kubernetes-core
- opnfv/functest-kubernetest-healthcheck
- opnfv/functest-kubernetest-smoke

Preparing your environment

cat env:

```
DEPLOY_SCENARIO=k8s-XXX
```

Testing healthcheck suite

Run healthcheck suite:

```
sudo docker run -it --env-file env \
    -v $(pwd)/config:/root/.kube/config \
    opnfv/functest-kubernetes-healthcheck
```

A config file in the current dir 'config' is also required, which should be volume mapped to ~/.kube/config inside kubernetes container.

Results shall be displayed as follows:

```
+------------------+------------------+--------------------+------------------+-----
↪-----------+
|     TEST CASE     |     PROJECT      |         TIER       |     DURATION     |  ␣
↪RESULT     |
+------------------+------------------+--------------------+------------------+-----
↪-----------+
|     k8s_smoke     |     functest     |     healthcheck    |       01:09      |  ␣
↪ PASS      |
+------------------+------------------+--------------------+------------------+-----
↪-----------+
```

Testing smoke suite

Run smoke suite:

```
sudo docker run -it --env-file env \
    -v $(pwd)/config:/root/.kube/config \
    opnfv/functest-kubernetes-smoke
```

Results shall be displayed as follows:

```
+---------------------------+------------------+--------------+------------------+---
↪-------------+
|         TEST CASE         |     PROJECT      |      TIER    |     DURATION     |  ␣
↪  RESULT     |
+---------------------------+------------------+--------------+------------------+---
↪-------------+
|     xrally_kubernetes     |     functest     |     smoke    |       22:04      |  ␣
↪   PASS      |
|      k8s_conformance      |     functest     |     smoke    |       173:48     |  ␣
↪   PASS      |
+---------------------------+------------------+--------------+------------------+---
↪-------------+
```

1.4 Environment variables

Several environement variables may be specified:

- INSTALLER_IP=<Specific IP Address>
- DEPLOY_SCENARIO=<vim>-<controller>-<nfv_feature>-<ha_mode>
- NAMESERVER=XXX # if not 8.8.8.8
- VOLUME_DEVICE_NAME=XXX # if not vdb
- EXTERNAL_NETWORK=XXX # if not first network with router:external=True
- NEW_USER_ROLE=XXX # if not member

INSTALLER_IP is required by Barometer in order to access the installer node and the deployment.

The format for the DEPLOY_SCENARIO env variable can be described as follows:

- vim: (os|k8s) = OpenStack or Kubernetes
- controller is one of (nosdn | odl)
- nfv_feature is one or more of (ovs | kvm | sfc | bgpvpn | nofeature)
- ha_mode (high availability) is one of (ha | noha)

If several features are pertinent then use the underscore character '_' to separate each feature (e.g. ovs_kvm). 'nofeature' indicates that no OPNFV feature is deployed.

The list of supported scenarios per release/installer is indicated in the release note.

NOTE: The scenario name is mainly used to automatically detect if a test suite is runnable or not (e.g. it will prevent ODL test suite to be run on 'nosdn' scenarios). If not set, Functest will try to run the default test cases that might not include SDN controller or a specific feature.

NOTE: An HA scenario means that 3 OpenStack controller nodes are deployed. It does not necessarily mean that the whole system is HA. See installer release notes for details.

Finally, three additional environment variables can also be passed in to the Functest Docker Container, using the -e "<EnvironmentVariable>=<Value>" mechanism. The first two parameters are only relevant to Jenkins CI invoked testing and **should not be used** when performing manual test scenarios:

- INSTALLER_TYPE=(apex|compass|daisy|fuel)
- NODE_NAME=<Test POD Name>
- BUILD_TAG=<Jenkins Build Tag>

where:

- **<Test POD Name> = Symbolic name of the POD where the tests are run.** Visible in test results files, which are stored to the database. This option is only used when tests are activated under Jenkins CI control. It indicates the POD/hardware where the test has been run. If not specified, then the POD name is defined as "Unknown" by default. DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.
- **<Jenkins Build tag> = Symbolic name of the Jenkins Build Job.** Visible in test results files, which are stored to the database. This option is only set when tests are activated under Jenkins CI control. It enables the correlation of test results, which are independently pushed to the results database from different Jenkins jobs. DO NOT USE THIS OPTION IN MANUAL TEST SCENARIOS.

1.5 Openstack credentials

OpenStack credentials are mandatory and must be provided to Functest. When running the command "functest env prepare", the framework will automatically look for the Openstack credentials file "/home/opnfv/functest/conf/env_file" and will exit with error if it is not present or is empty.

There are 2 ways to provide that file:

- by using a Docker volume with -v option when creating the Docker container. This is referred to in docker documentation as "Bind Mounting". See the usage of this parameter in the following chapter.
- or creating manually the file '/home/opnfv/functest/conf/env_file' inside the running container and pasting the credentials in it. Consult your installer guide for further details. This is however not instructed in this document.

In proxified environment you may need to change the credentials file. There are some tips in chapter: *Proxy support*

1.5.1 SSL Support

If you need to connect to a server that is TLS-enabled (the auth URL begins with "https") and it uses a certificate from a private CA or a self-signed certificate, then you will need to specify the path to an appropriate CA certificate to use, to validate the server certificate with the environment variable OS_CACERT:

```
echo $OS_CACERT
/etc/ssl/certs/ca.crt
```

However, this certificate does not exist in the container by default. It has to be copied manually from the OpenStack deployment. This can be done in 2 ways:

1. Create manually that file and copy the contents from the OpenStack controller.
2. (Recommended) Add the file using a Docker volume when starting the container:

```
-v <path_to_your_cert_file>:/etc/ssl/certs/ca.cert
```

You might need to export OS_CACERT environment variable inside the credentials file:

```
export OS_CACERT=/etc/ssl/certs/ca.crt
```

Certificate verification can be turned off using OS_INSECURE=true. For example, Fuel uses self-signed cacerts by default, so an pre step would be:

```
export OS_INSECURE=true
```

1.6 Logs

By default all the logs are put un /home/opnfv/functest/results/functest.log. If you want to have more logs in console, you may edit the logging.ini file manually. Connect on the docker then edit the file located in /usr/lib/python3.7/site-packages/xtesting/ci/logging.ini

Change wconsole to console in the desired module to get more traces.

1.7 Configuration

You may also directly modify the python code or the configuration file (e.g. testcases.yaml used to declare test constraints) under /usr/lib/python3.7/site-packages/xtesting and /usr/lib/python3.7/site-packages/functest

1.8 Tips

1.8.1 Docker

When typing **exit** in the container prompt, this will cause exiting the container and probably stopping it. When stopping a running Docker container all the changes will be lost, there is a keyboard shortcut to quit the container without stopping it: <CTRL>-P + <CTRL>-Q. To reconnect to the running container **DO NOT** use the *run* command again (since it will create a new container), use the *exec* or *attach* command instead:

```
docker ps  # <check the container ID from the output>
docker exec -ti <CONTAINER_ID> /bin/bash
```

There are other useful Docker commands that might be needed to manage possible issues with the containers.

List the running containers:

```
docker ps
```

List all the containers including the stopped ones:

```
docker ps -a
```

Start a stopped container named "FunTest":

```
docker start FunTest
```

Attach to a running container named "StrikeTwo":

```
docker attach StrikeTwo
```

It is useful sometimes to remove a container if there are some problems:

```
docker rm <CONTAINER_ID>
```

Use the *-f* option if the container is still running, it will force to destroy it:

```
docker rm -f <CONTAINER_ID>
```

Check the Docker documentation [dockerdocs] for more information.

1.8.2 Checking Openstack and credentials

It is recommended and fairly straightforward to check that Openstack and credentials are working as expected.

Once the credentials are there inside the container, they should be sourced before running any Openstack commands:

```
source /home/opnfv/functest/conf/env_file
```

After this, try to run any OpenStack command to see if you get any output, for instance:

```
openstack user list
```

This will return a list of the actual users in the OpenStack deployment. In any other case, check that the credentials are sourced:

```
env|grep OS_
```

This command must show a set of environment variables starting with *OS_*, for example:

```
OS_REGION_NAME=RegionOne
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_NAME=admin
OS_AUTH_VERSION=3
OS_IDENTITY_API_VERSION=3
OS_PASSWORD=da54c27ae0d10dfae5297e6f0d6be54ebdb9f58d0f9dfc
OS_AUTH_URL=http://10.1.0.9:5000/v3
OS_USERNAME=admin
OS_TENANT_NAME=admin
OS_ENDPOINT_TYPE=internalURL
OS_INTERFACE=internalURL
OS_NO_CACHE=1
OS_PROJECT_DOMAIN_NAME=Default
```

If the OpenStack command still does not show anything or complains about connectivity issues, it could be due to an incorrect url given to the OS_AUTH_URL environment variable. Check the deployment settings.

1.8.3 Proxy support

If your Jumphost node is operating behind a http proxy, then there are 2 places where some special actions may be needed to make operations succeed:

1. Initial installation of docker engine First, try following the official Docker documentation for Proxy settings. Some issues were experienced on CentOS 7 based Jumphost. Some tips are documented in section: *Docker Installation on CentOS behind http proxy* below.

If that is the case, make sure the resolv.conf and the needed http_proxy and https_proxy environment variables, as well as the 'no_proxy' environment variable are set correctly:

```
# Make double sure that the 'no_proxy=...' line in the
# 'env_file' file is commented out first. Otherwise, the
# values set into the 'no_proxy' environment variable below will
# be ovewrwritten, each time the command
# 'source ~/functest/conf/env_file' is issued.

cd ~/functest/conf/
sed -i 's/export no_proxy/#export no_proxy/' env_file
source ./env_file

# Next calculate some IP addresses for which http_proxy
# usage should be excluded:

publicURL_IP=$(echo $OS_AUTH_URL | grep -Eo "([0-9]+\.){3}[0-9]+")

adminURL_IP=$(openstack catalog show identity | \
grep adminURL | grep -Eo "([0-9]+\.){3}[0-9]+")

export http_proxy="<your http proxy settings>"
export https_proxy="<your https proxy settings>"
export no_proxy="127.0.0.1,localhost,$publicURL_IP,$adminURL_IP"

# Ensure that "git" uses the http_proxy
# This may be needed if your firewall forbids SSL based git fetch
git config --global http.sslVerify True
git config --global http.proxy <Your http proxy settings>
```

For example, try to use the **nc** command from inside the functest docker container:

```
nc -v opnfv.org 80
Connection to opnfv.org 80 port [tcp/http] succeeded!

nc -v opnfv.org 443
Connection to opnfv.org 443 port [tcp/https] succeeded!
```

Note: In a Jumphost node based on the CentOS family OS, the **nc** commands might not work. You can use the **curl** command instead.

curl https://www.opnfv.org/

<HTML><HEAD><meta http-equiv="content-type" . . </BODY></HTML>

curl https://www.opnfv.org:443

<HTML><HEAD><meta http-equiv="content-type" . . </BODY></HTML>

(Ignore the content. If command returns a valid HTML page, it proves the connection.)

1.8.4 Docker Installation on CentOS behind http proxy

This section is applicable for CentOS family OS on Jumphost which itself is behind a proxy server. In that case, the instructions below should be followed **before** installing the docker engine:

```
1) # Make a directory '/etc/systemd/system/docker.service.d'
   # if it does not exist
   sudo mkdir /etc/systemd/system/docker.service.d

2) # Create a file called 'env.conf' in that directory with
   # the following contents:
   [Service]
   EnvironmentFile=-/etc/sysconfig/docker

3) # Set up a file called 'docker' in directory '/etc/sysconfig'
   # with the following contents:
   HTTP_PROXY="<Your http proxy settings>"
   HTTPS_PROXY="<Your https proxy settings>"
   http_proxy="${HTTP_PROXY}"
   https_proxy="${HTTPS_PROXY}"

4) # Reload the daemon
   systemctl daemon-reload

5) # Sanity check - check the following docker settings:
   systemctl show docker | grep -i env

   Expected result:
   ----------------
   EnvironmentFile=/etc/sysconfig/docker (ignore_errors=yes)
   DropInPaths=/etc/systemd/system/docker.service.d/env.conf
```

Now follow the instructions in [Install Docker on CentOS] to download and install the **docker-engine**. The instructions conclude with a "test pull" of a sample "Hello World" docker container. This should now work with the above pre-requisite actions.

1.9 Integration in CI

In CI we use the Docker images and execute the appropriate commands within the container from Jenkins.

4 steps have been defined::

- functest-cleanup: clean existing functest dockers on the jumphost
- functest-daily: run dockers opnfv/functest-* (healthcheck, smoke, benchmarking, vnf)
- functest-store-results: push logs to artifacts

See [1] for details.

1.10 References

[1] : Functest Jenkins jobs

IRC support channel: #opnfv-functest

CHAPTER 2

Functest User Guide

2.1 Introduction

The goal of this document is to describe the OPNFV Functest test cases and to provide a procedure to execute them.

IMPORTANT: It is assumed here that Functest has been properly deployed following the installation guide procedure *Functest Installation Guide*.

2.2 Overview of the Functest suites

Functest is the OPNFV project primarily targeting functional testing. In the Continuous Integration pipeline, it is launched after an OPNFV fresh installation to validate and verify the basic functions of the infrastructure.

The current list of test suites can be distributed over 5 main domains:

- VIM (Virtualised Infrastructure Manager)
- Controllers (i.e. SDN Controllers)
- VNF (Virtual Network Functions)
- Kubernetes

Functest test suites are also distributed in the OPNFV testing categories: healthcheck, smoke, benchmarking, VNF, Stress tests.

All the Healthcheck and smoke tests of a given scenario must be succesful to validate the scenario for the release.

Domain	Tier	Test case	Comments
VIM	healthcheck	connection_check	Check OpenStack connectivity
	smoke	vping_ssh	NFV "Hello World" using an SSH connection to a destination VM over a created floating IP address on the SUT Public / External network. Using the SSH connection a test script is then copied to the destination VM and then executed via SSH. The script will ping another VM on a specified IP address over the SUT Private Tenant network
		vping_userdata	Uses Ping with given userdata to test intra-VM connectivity over the SUT Private Tenant network. The correct operation of the NOVA Metadata service is also verified in this test
		tempest_smoke	Generate and run a relevant Tempest Test Suite in smoke mode. The generated test set is dependent on the OpenStack deployment environment
		rally_sanity	Run a subset of the OpenStack Rally Test Suite in smoke mode
		refstack_defcore	Reference RefStack suite tempest selection for NFV
		patrole	Patrole is a tempest plugin for testing and verifying RBAC policy enforcement, which offers testing for the following OpenStack services: Nova, Neutron, Glance, Cinder and Keystone
		neutron_trunk	The neutron trunk port testcases have been introduced and they are supported by installers : Apex, Fuel and Compass.
	components	tempest_full_parallel	Generate and run a full set of the OpenStack Tempest Test Suite. See the OpenStack reference test suite [2]. The generated test set is dependent on the OpenStack deployment environment
		rally_full	Run the OpenStack testing tool benchmarking OpenStack modules See the Rally documents [3]
Controllers	smoke	odl	Opendaylight Test suite Limited test suite to check the basic neutron (Layer 2) operations mainly based on upstream testcases. See below for details
VNF	vnf	cloudify_ims	Example of a real VNF deployment to show the NFV capabilities of the platform. The IP Multimedia Subsytem is a typical Telco test case, referenced by ETSI. It provides a fully functional VoIP System
		vyos_vrouter	vRouter testing
		juju_epc	Validates deployment of a complex mobility VNF on OPNFV Platform. Uses Juju for deploying the OAI EPC and ABot for defining test scenarios using high-level DSL. VNF tests reference 3GPP Technical Specs and are executed through protocol drivers provided by ABot.
Kubernetes	healthcheck	k8s_smoke	Test a running Kubernetes cluster and ensure it satisfies minimal functional requirements
	smoke	k8s_conformance	Run a subset of Kubernetes End-to-End tests, expected to pass on any Kubernetes cluster

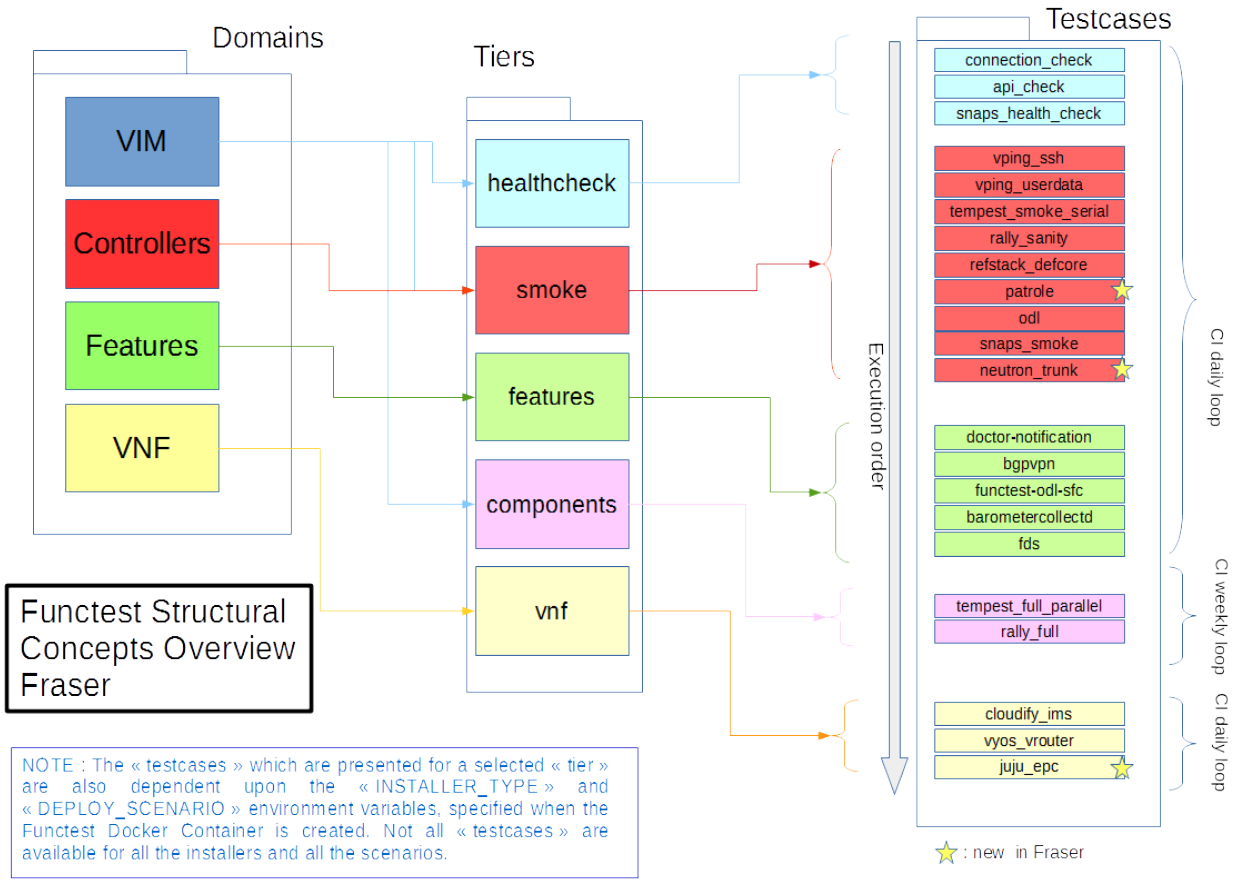
As shown in the above table, Functest is structured into different 'domains', 'tiers' and 'test cases'. Each 'test case'

usually represents an actual 'Test Suite' comprised -in turn- of several test cases internally.

Test cases also have an implicit execution order. For example, if the early 'healthcheck' Tier testcase fails, or if there are any failures in the 'smoke' Tier testcases, there is little point to launch a full testcase execution round.

In Danube, we merged smoke and sdn controller tiers in smoke tier.

An overview of the Functest Structural Concept is depicted graphically below:



Some of the test cases are developed by Functest team members, whereas others are integrated from upstream communities or other OPNFV projects. For example, Tempest is the OpenStack integration test suite and Functest is in charge of the selection, integration and automation of those tests that fit suitably to OPNFV.

The Tempest test suite is the default OpenStack smoke test suite but no new test cases have been created in OPNFV Functest.

The results produced by the tests run from CI are pushed and collected into a NoSQL database. The goal is to populate the database with results from different sources and scenarios and to show them on a Functest Dashboard. A screenshot of a live Functest Dashboard is shown below:

Basic components (VIM, SDN controllers) are tested through their own suites. Feature projects also provide their own test suites with different ways of running their tests.

The notion of domain has been introduced in the description of the test cases stored in the Database. This parameters as well as possible tags can be used for the Test case catalog.

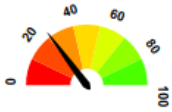
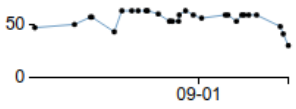
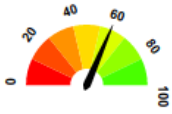
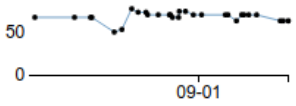
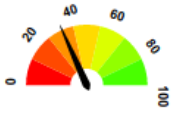
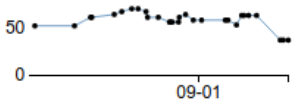
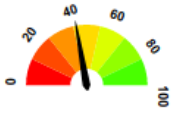
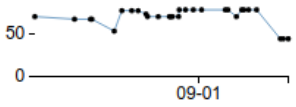
vIMS test case was integrated to demonstrate the capability to deploy a relatively complex NFV scenario on top of the OPNFV infrastructure.

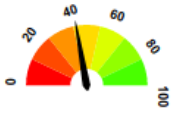
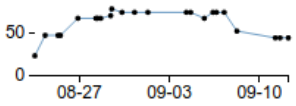
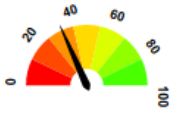
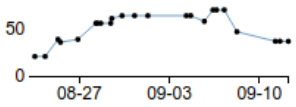
Functest status page (master, 2017-09-13 06:22)

Home Apex Compass Daisy fuel@x86 fuel@aarch64 Joid

compass

List of last scenarios (master) run over the last 10 days

HA Scenario	Status	Trend	Score	Iteration
os-odl_l3-nofeature-ha			8/27	3
os-nosdn-kvm-ha			17/27	4
os-odl-sfc-ha			11/30	4
os-nosdn-nofeature-ha			12/27	2

NOHA Scenario	Status	Trend	Score	Iteration
os-nosdn-kvm-noha			12/27	3
os-odl-sfc-noha			11/30	2

Functest considers OPNFV as a black box. OPNFV offers a lot of potential combinations (which may change from one version to another):

- 3 controllers (OpenDaylight, ONOS, OpenContrail)
- 5 installers (Apex, Compass, Daisy, Fuel, Joid)

Most of the tests are runnable by any combination, but some tests might have restrictions imposed by the utilized installers or due to the available deployed services. The system uses the environment variables to automatically determine the valid test cases, for each given environment.

A convenience Functest CLI utility is also available to simplify setting up the Functest evironment, management of the OpenStack environment (e.g. resource clean-up) and for executing tests. The Functest CLI organised the testcase into logical Tiers, which contain in turn one or more testcases. The CLI allows execution of a single specified testcase, all test cases in a specified Tier, or the special case of execution of **ALL** testcases. The Functest CLI is introduced in more details in next section.

The different test cases are described in the remaining sections of this document.

2.3 VIM (Virtualized Infrastructure Manager)

2.3.1 Healthcheck tests

Since Danube, healthcheck tests have been refactored and rely on SNAPS, an OPNFV middleware project.

SNAPS stands for "SDN/NFV Application development Platform and Stack". SNAPS is an object-oriented OpenStack library packaged with tests that exercise OpenStack. More information on SNAPS can be found in [13]

Three tests are declared as healthcheck tests and can be used for gating by the installer, they cover functionally the tests previously done by healthcheck test case.

The tests are:

- *connection_check*

Connection_check consists in test cases (test duration < 5s) checking the connectivity with Glance, Keystone, Neutron, Nova and the external network.

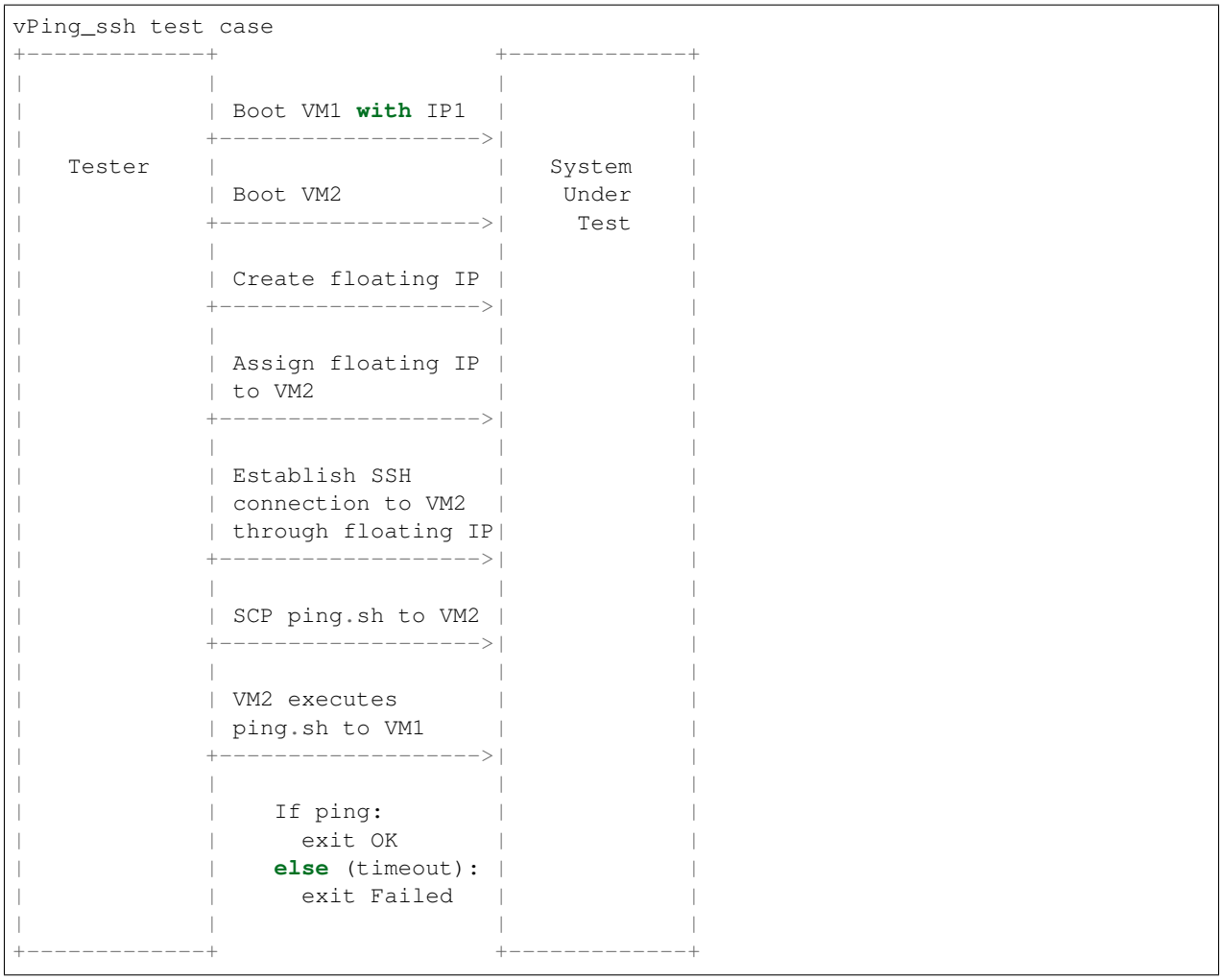
Self-obviously, successful completion of the 'healthcheck' testcase is a necessary pre-requisite for the execution of all other test Tiers.

2.3.2 vPing_ssh

Given the script **ping.sh**:

```
#!/bin/sh
ping -c 1 $1 2>&1 >/dev/null
RES=$?
if [ "Z$RES" = "Z0" ] ; then
    echo 'vPing OK'
else
    echo 'vPing KO'
fi
```

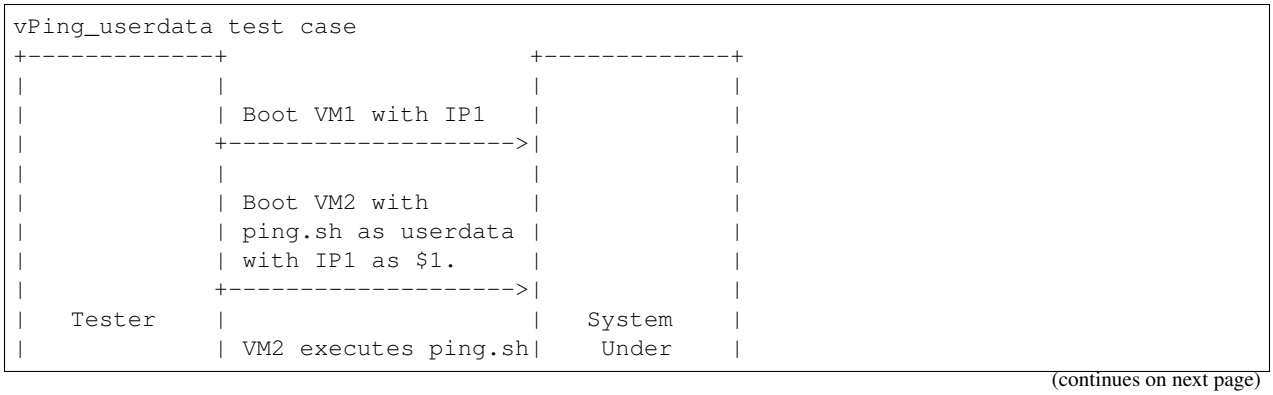
The goal of this test is to establish an SSH connection using a floating IP on the Public/External network and verify that 2 instances can talk over a Private Tenant network:



This test can be considered as an "Hello World" example. It is the first basic use case which **must** work on any deployment.

2.3.3 vPing_userdata

This test case is similar to vPing_ssh but without the use of Floating IPs and the Public/External network to transfer the ping script. Instead, it uses Nova metadata service to pass it to the instance at booting time. As vPing_ssh, it checks that 2 instances can talk to each other on a Private Tenant network:



(continues on next page)

(continued from previous page)

```
|              | (ping IP1)           |     Test     |
|              +--------------------->|              |
|              |                      |              |
|              | Monitor nova         |              |
|              |  console-log VM 2    |              |
|              |    If ping:          |              |
|              |      exit OK         |              |
|              |    else (timeout)    |              |
|              |      exit Failed     |              |
|              |                      |              |
+--------------+                      +--------------+
```

When the second VM boots it will execute the script passed as userdata automatically. The ping will be detected by periodically capturing the output in the console-log of the second VM.

2.3.4 Tempest

Tempest [2] is the reference OpenStack Integration test suite. It is a set of integration tests to be run against a live OpenStack cluster. Tempest has suites of tests for:

- OpenStack API validation
- Scenarios
- Other specific tests useful in validating an OpenStack deployment

Functest uses Rally [3] to run the Tempest suite. Rally generates automatically the Tempest configuration file **tempest.conf**. Before running the actual test cases, Functest creates the needed resources (user, tenant) and updates the appropriate parameters into the configuration file.

When the Tempest suite is executed, each test duration is measured and the full console output is stored to a *log* file for further analysis.

The Tempest testcases are distributed across three Tiers:

- Smoke Tier - Test Case 'tempest_smoke'
- Components Tier - Test case 'tempest_full'
- Neutron Trunk Port - Test case 'neutron_trunk'
- OpenStack interop testcases - Test case 'refstack_defcore'
- Testing and verifying RBAC policy enforcement - Test case 'patrole'

NOTE: Test case 'tempest_smoke' executes a defined set of tempest smoke tests. Test case 'tempest_full' executes all defined Tempest tests.

NOTE: The 'neutron_trunk' test set allows to connect a VM to multiple VLAN separated networks using a single NIC. The feature neutron trunk ports have been supported by Apex, Fuel and Compass, so the tempest testcases have been integrated normally.

NOTE: Rally is also used to run Openstack Interop testcases [9], which focus on testing interoperability between OpenStack clouds.

NOTE: Patrole is a tempest plugin for testing and verifying RBAC policy enforcement. It runs Tempest-based API tests using specified RBAC roles, thus allowing deployments to verify that only intended roles have access to those APIs. Patrole currently offers testing for the following OpenStack services: Nova, Neutron, Glance, Cinder and Keystone. Currently in functest, only neutron and glance are tested.

The goal of the Tempest test suite is to check the basic functionalities of the different OpenStack components on an OPNFV fresh installation, using the corresponding REST API interfaces.

2.3.5 Rally bench test suites

Rally [3] is a benchmarking tool that answers the question:

How does OpenStack work at scale?

The goal of this test suite is to benchmark all the different OpenStack modules and get significant figures that could help to define Telco Cloud KPIs.

The OPNFV Rally scenarios are based on the collection of the actual Rally scenarios:

- authenticate
- cinder
- glance
- heat
- keystone
- neutron
- nova
- quotas

A basic SLA (stop test on errors) has been implemented.

The Rally testcases are distributed across two Tiers:

- Smoke Tier - Test Case 'rally_sanity'
- Components Tier - Test case 'rally_full'

NOTE: Test case 'rally_sanity' executes a limited number of Rally smoke test cases. Test case 'rally_full' executes the full defined set of Rally tests.

2.4 SDN Controllers

2.4.1 OpenDaylight

The OpenDaylight (ODL) test suite consists of a set of basic tests inherited from the ODL project using the Robot [11] framework. The suite verifies creation and deletion of networks, subnets and ports with OpenDaylight and Neutron.

The list of tests can be described as follows:

- Basic Restconf test cases
  - Connect to Restconf URL
  - Check the HTTP code status
- Neutron Reachability test cases
  - Get the complete list of neutron resources (networks, subnets, ports)
- Neutron Network test cases
  - Check OpenStack networks

  - Check OpenDaylight networks
  - Create a new network via OpenStack and check the HTTP status code returned by Neutron
  - Check that the network has also been successfully created in OpenDaylight
- Neutron Subnet test cases
  - Check OpenStack subnets
  - Check OpenDaylight subnets
  - Create a new subnet via OpenStack and check the HTTP status code returned by Neutron
  - Check that the subnet has also been successfully created in OpenDaylight
- Neutron Port test cases
  - Check OpenStack Neutron for known ports
  - Check OpenDaylight ports
  - Create a new port via OpenStack and check the HTTP status code returned by Neutron
  - Check that the new port has also been successfully created in OpenDaylight
- Delete operations
  - Delete the port previously created via OpenStack
  - Check that the port has been also successfully deleted in OpenDaylight
  - Delete previously subnet created via OpenStack
  - Check that the subnet has also been successfully deleted in OpenDaylight
  - Delete the network created via OpenStack
  - Check that the network has also been successfully deleted in OpenDaylight

Note: the checks in OpenDaylight are based on the returned HTTP status code returned by OpenDaylight.

2.5 VNF

2.5.1 cloudify_ims

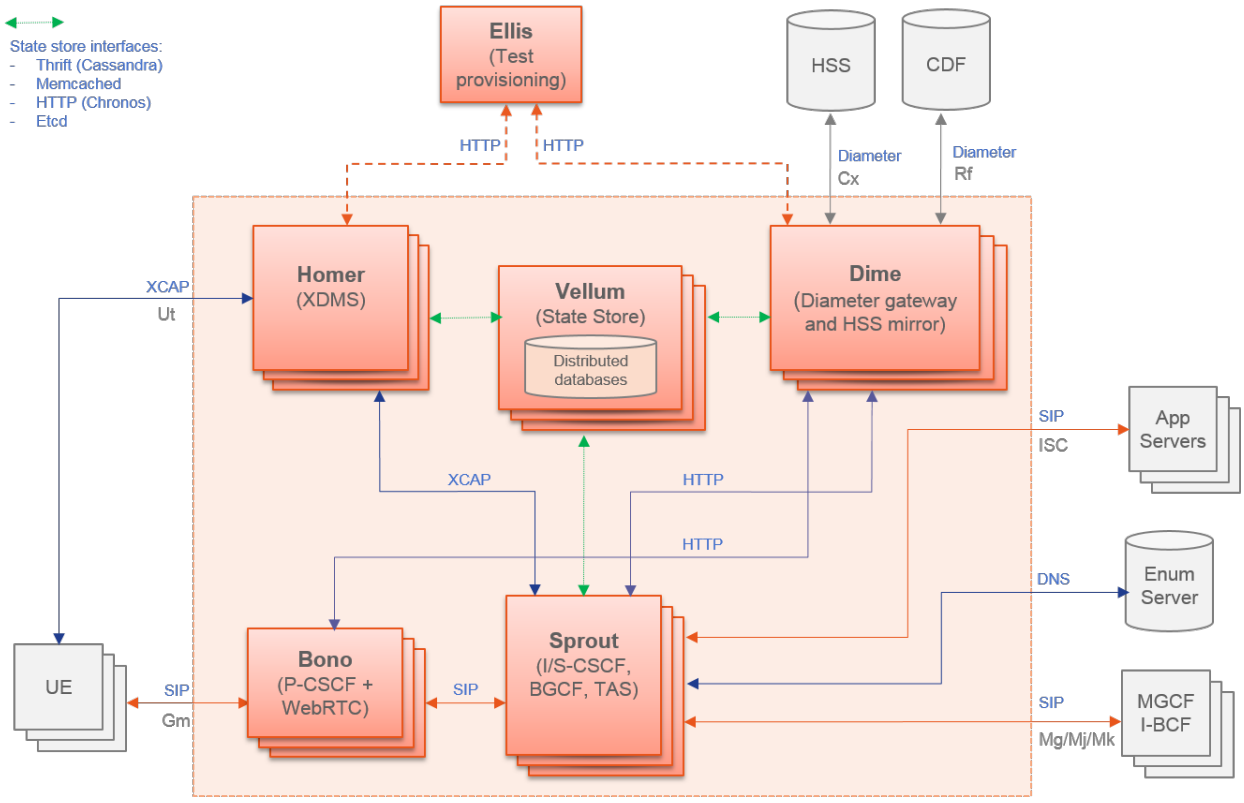
The IP Multimedia Subsystem or IP Multimedia Core Network Subsystem (IMS) is an architectural framework for delivering IP multimedia services.

vIMS has been integrated in Functest to demonstrate the capability to deploy a relatively complex NFV scenario on the OPNFV platform. The deployment of a complete functional VNF allows the test of most of the essential functions needed for a NFV platform.

The goal of this test suite consists of:

- deploy a VNF orchestrator (Cloudify)
- deploy a Clearwater vIMS (IP Multimedia Subsystem) VNF from this orchestrator based on a TOSCA blueprint defined in [5]
- run suite of signaling tests on top of this VNF

The Clearwater architecture is described as follows:



2.5.2 heat_ims

The IP Multimedia Subsystem or IP Multimedia Core Network Subsystem (IMS) is an architectural framework for delivering IP multimedia services.

vIMS has been integrated in Functest to demonstrate the capability to deploy a relatively complex NFV scenario on the OPNFV platform. The deployment of a complete functional VNF allows the test of most of the essential functions needed for a NFV platform.

The goal of this test suite consists of:

- deploy a Clearwater vIMS (IP Multimedia Subsystem) VNF using OpenStack Heat orchestrator based on a HOT template defined in [17]
- run suite of signaling tests on top of this VNF

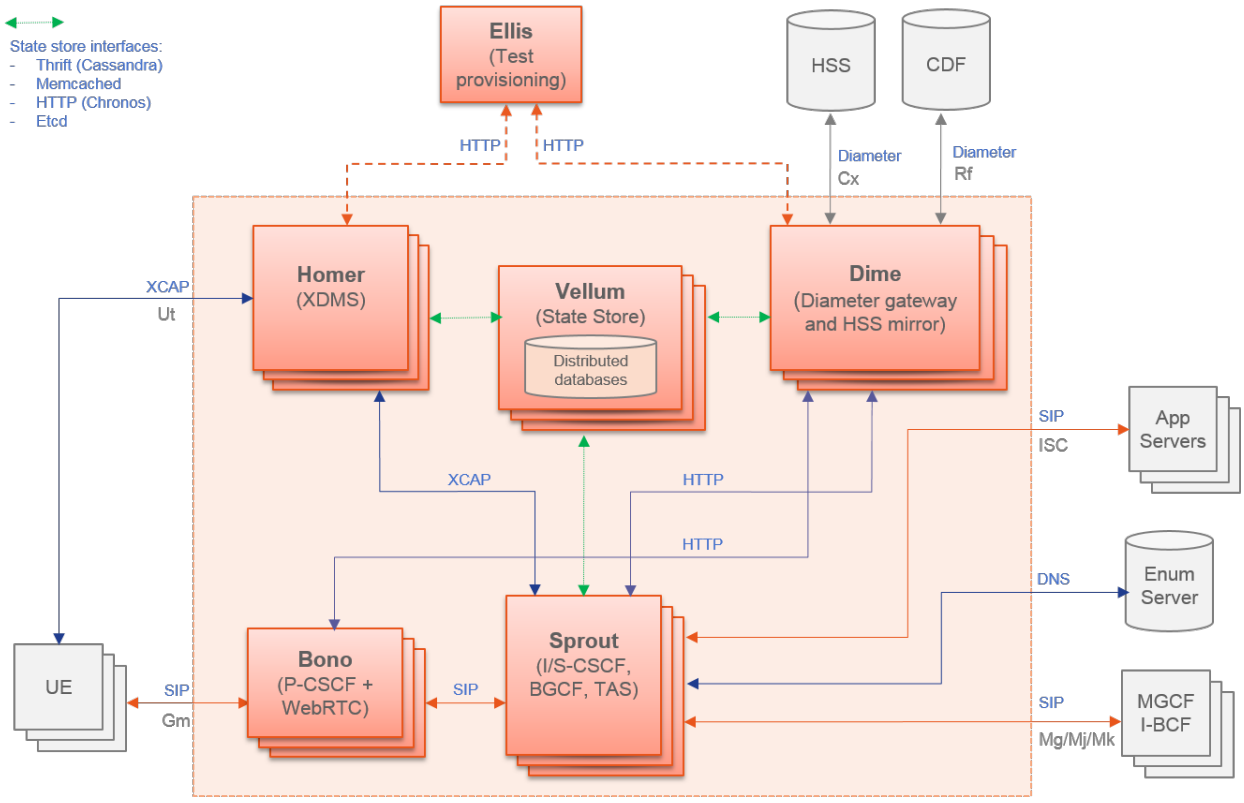
The Clearwater architecture is described as follows:

2.5.3 vyos-vrouter

This test case deals with the deployment and the test of vyos vrouter with Cloudify orchestrator. The test case can do testing for interchangeability of BGP Protocol using vyos.

The Workflow is as follows:

- **Deploy** Deploy VNF Testing topology by Cloudify using blueprint.
- **Configuration** Setting configuration to Target VNF and reference VNF using ssh
- **Run** Execution of test command for test item written YAML format file. Check VNF status and behavior.



- **Reporting** Output of report based on result using JSON format.

The vyos-vrouter architecture is described in [14]

2.5.4 juju_epc

The Evolved Packet Core (EPC) is the main component of the System Architecture Evolution (SAE) which forms the core of the 3GPP LTE specification.

vEPC has been integrated in Functest to demonstrate the capability to deploy a complex mobility-specific NFV scenario on the OPNFV platform. The OAI EPC supports most of the essential functions defined by the 3GPP Technical Specs; hence the successful execution of functional tests on the OAI EPC provides a good endorsement of the underlying NFV platform.

This integration also includes ABot, a Test Orchestration system that enables test scenarios to be defined in high-level DSL. ABot is also deployed as a VM on the OPNFV platform; and this provides an example of the automation driver and the Test VNF being both deployed as separate VNFs on the underlying OPNFV platform.

The Workflow is as follows:

- **Deploy Orchestrator** Deploy Juju controller using Bootstrap command.
- **Deploy VNF** Deploy ABot orchestrator and OAI EPC as Juju charms. Configuration of ABot and OAI EPC components is handled through built-in Juju relations.
- **Test VNF** Execution of ABot feature files triggered by Juju actions. This executes a suite of LTE signalling tests on the OAI EPC.
- **Reporting** ABot test results are parsed accordingly and pushed to Functest Db.

Details of the ABot test orchestration tool may be found in [15]

2.6 Kubernetes (K8s)

Kubernetes testing relies on sets of tests, which are part of the Kubernetes source tree, such as the Kubernetes End-to-End (e2e) tests [16].

The kubernetes testcases are distributed across various Tiers:

- Healthcheck Tier
  - k8s_smoke Test Case: Creates a Guestbook application that contains redis server, 2 instances of redis slave, frontend application, frontend service and redis master service and redis slave service. Using frontend service, the test will write an entry into the guestbook application which will store the entry into the backend redis database. Application flow MUST work as expected and the data written MUST be available to read.
- Smoke Tier
  - k8s_conformance Test Case: Runs a series of k8s e2e tests expected to pass on any Kubernetes cluster. It is a subset of tests necessary to demonstrate conformance grows with each release. Conformance is thus considered versioned, with backwards compatibility guarantees and are designed to be run with no cloud provider configured.

2.7 Test results

2.7.1 Manual testing

In manual mode test results are displayed in the console and result files are put in /home/opnfv/functest/results.

If you want additional logs, you may configure the logging.ini under /usr/lib/python3.7/site-packages/xtesting/ci.

2.7.2 Automated testing

In automated mode, tests are run within split Alpine containers, and test results are displayed in jenkins logs. The result summary is provided at the end of each suite and can be described as follow.

Healthcheck suite:

```
+-------------------------+------------------+---------------------+-----------------
↪-+----------------+
|        TEST CASE        |     PROJECT      |         TIER        |     DURATION
↪ |     RESULT     |
+-------------------------+------------------+---------------------+-----------------
↪-+----------------+
|     connection_check    |     functest     |     healthcheck     |       00:02
↪ |      PASS      |
|      tenantnetwork1     |     functest     |     healthcheck     |       00:06
↪ |      PASS      |
|      tenantnetwork2     |     functest     |     healthcheck     |       00:06
↪ |      PASS      |
|         vmready1        |     functest     |     healthcheck     |       00:06
↪ |      PASS      |
|         vmready2        |     functest     |     healthcheck     |       00:08
↪ |      PASS      |
|        singlevm1        |     functest     |     healthcheck     |       00:28
↪ |      PASS      |
```

(continues on next page)

(continued from previous page)

```
|        singlevm2          |     functest     |     healthcheck      |       00:25      |
↪ |      PASS      |
|        vping_ssh          |     functest     |     healthcheck      |       00:36      |
↪ |      PASS      |
|      vping_userdata       |     functest     |     healthcheck      |       00:34      |
↪ |      PASS      |
|       cinder_test         |     functest     |     healthcheck      |       01:03      |
↪ |      PASS      |
|      tempest_smoke        |     functest     |     healthcheck      |       05:13      |
↪ |      PASS      |
|           odl             |     functest     |     healthcheck      |       00:00      |
↪ |      SKIP      |
+--------------------------+------------------+----------------------+------------------
↪-+----------------+
```

Smoke suite:

```
+----------------------------------+------------------+---------------+-------------
↪-----+----------------+
|             TEST CASE            |     PROJECT      |      TIER     |
↪DURATION     |     RESULT     |
+----------------------------------+------------------+---------------+-------------
↪-----+----------------+
|     neutron-tempest-plugin-api     |     functest     |     smoke     |       09:12
↪      |      PASS      |
|            rally_sanity           |     functest     |     smoke     |       16:29
↪      |      PASS      |
|          refstack_compute         |     functest     |     smoke     |       06:25
↪      |      PASS      |
|          refstack_object          |     functest     |     smoke     |       01:54
↪      |      PASS      |
|         refstack_platform         |     functest     |     smoke     |       06:52
↪      |      PASS      |
|           tempest_full            |     functest     |     smoke     |       30:26
↪      |      PASS      |
|          tempest_scenario         |     functest     |     smoke     |       09:23
↪      |      PASS      |
|           tempest_slow            |     functest     |     smoke     |       24:42
↪      |      PASS      |
|            patrole                |     functest     |     smoke     |       02:36
↪      |      PASS      |
|            barbican               |     functest     |     smoke     |       02:13
↪      |      PASS      |
|          neutron_trunk            |     functest     |     smoke     |       00:00
↪      |      SKIP      |
|        networking-bgpvpn          |     functest     |     smoke     |       00:00
↪      |      SKIP      |
|          networking-sfc           |     functest     |     smoke     |       00:00
↪      |      SKIP      |
|            octavia               |     functest     |     smoke     |       00:00
↪      |      SKIP      |
+----------------------------------+------------------+---------------+-------------
↪-----+----------------+
```

Benchmarking suite:

```
+--------------------+------------------+---------------------+------------------+---
↪-------------+
|     TEST CASE      |     PROJECT      |         TIER        |     DURATION     |  ⌴
↪  RESULT     |
+--------------------+------------------+---------------------+------------------+---
↪-------------+
|     rally_full     |     functest     |     benchmarking    |       92:16      |  ⌴
↪   PASS      |
|     rally_jobs     |     functest     |     benchmarking    |       18:49      |  ⌴
↪   PASS      |
|        vmtp        |     functest     |     benchmarking    |       15:28      |  ⌴
↪   PASS      |
|       shaker       |     functest     |     benchmarking    |       24:04      |  ⌴
↪   PASS      |
+--------------------+------------------+---------------------+------------------+---
↪-------------+
```

Vnf suite:

```
+---------------------+------------------+-------------+------------------+---------
↪-------+
|      TEST CASE      |     PROJECT      |     TIER    |     DURATION     |    ⌴
↪RESULT     |
+---------------------+------------------+-------------+------------------+---------
↪-------+
|       cloudify      |     functest     |     vnf     |       03:49      |    ⌴
↪PASS       |
|     cloudify_ims    |     functest     |     vnf     |       24:20      |    ⌴
↪PASS       |
|       heat_ims      |     functest     |     vnf     |       32:13      |    ⌴
↪PASS       |
|     vyos_vrouter    |     functest     |     vnf     |       14:55      |    ⌴
↪PASS       |
|       juju_epc      |     functest     |     vnf     |       41:24      |    ⌴
↪PASS       |
+---------------------+------------------+-------------+------------------+---------
↪-------+
```

Kubernetes healthcheck suite:

```
+------------------+------------------+--------------------+------------------+-----
↪-----------+
|     TEST CASE    |     PROJECT      |         TIER       |     DURATION     |    ⌴
↪RESULT     |
+------------------+------------------+--------------------+------------------+-----
↪-----------+
|     k8s_smoke    |     functest     |     healthcheck    |       01:09      |    ⌴
↪ PASS      |
+------------------+------------------+--------------------+------------------+-----
↪-----------+
```

Kubernetes smoke suite:

```
+-------------------------+------------------+-------------+------------------+---
↪-------------+
|         TEST CASE       |     PROJECT      |     TIER    |     DURATION     |  ⌴
↪  RESULT     |
```

(continues on next page)

(continued from previous page)

```
+---------------------------+------------------+--------------+------------------+---
↪-------------+
|     xrally_kubernetes     |     functest     |     smoke    |      22:04       | ␣
↪   PASS      |
|      k8s_conformance      |     functest     |     smoke    |      173:48      | ␣
↪   PASS      |
+---------------------------+------------------+--------------+------------------+---
↪-------------+
```

Results are automatically pushed to the test results database, some additional result files are pushed to OPNFV artifact web sites.

Based on the results stored in the result database, a Functest reporting portal is also automatically updated. This portal provides information on the overall status per scenario and per installer

2.8 Test reporting

An automatic reporting page has been created in order to provide a consistent view of the Functest tests on the different scenarios.

In this page, each scenario is evaluated according to test criteria.

The results are collected from the centralized database every day and, per scenario. A score is calculated based on the results from the last 10 days. This score is the addition of single test scores. Each test case has a success criteria reflected in the criteria field from the results.

As an illustration, let's consider the scenario os-odl_l2-nofeature-ha scenario, the scenario scoring is the addition of the scores of all the runnable tests from the categories (tiers, healthcheck, smoke and features) corresponding to this scenario.

Test	Apex	Compass	Fuel	Joid
vPing_ssh	X	X	X	X
vPing_userdata	X	X	X	X
tempest_smoke	X	X	X	X
rally_sanity	X	X	X	X
odl	X	X	X	X
promise			X	X
doctor	X		X	
security_scan	X			
parser			X	
copper	X			X

src: os-odl_l2-nofeature-ha Colorado (see release note for the last matrix version)

All the testcases (X) listed in the table are runnable on os-odl_l2-nofeature scenarios. Please note that other test cases (e.g. sfc_odl, bgpvpn) need ODL configuration addons and, as a consequence, specific scenario. There are not considered as runnable on the generic odl_l2 scenario.

If no result is available or if all the results are failed, the test case get 0 point. If it was successful at least once but not anymore during the 4 runs, the case get 1 point (it worked once). If at least 3 of the last 4 runs were successful, the case get 2 points. If the last 4 runs of the test are successful, the test get 3 points.

In the example above, the target score for fuel/os-odl_l2-nofeature-ha is 3 x 8 = 24 points and for compass it is 3 x 5 = 15 points .

The scenario is validated per installer when we got 3 points for all individual test cases (e.g 24/24 for fuel, 15/15 for compass).

Please note that complex or long duration tests are not considered yet for the scoring. In fact the success criteria are not always easy to define and may require specific hardware configuration.

Please also note that all the test cases have the same "weight" for the score calculation whatever the complexity of the test case. Concretely a vping has the same weight than the 200 tempest tests. Moreover some installers support more features than others. The more cases your scenario is dealing with, the most difficult to rich a good scoring.

Therefore the scoring provides 3 types of indicators:

- the richness of the scenario: if the target scoring is high, it means that the scenario includes lots of features
- the maturity: if the percentage (scoring/target scoring * 100) is high, it means that all the tests are PASS
- the stability: as the number of iteration is included in the calculation, the pecentage can be high only if the scenario is run regularly (at least more than 4 iterations over the last 10 days in CI)

In any case, the scoring is used to give feedback to the other projects and does not represent an absolute value of the scenario.

See reporting page for details. For the status, click on the version, Functest then the Status menu.

2.9 Troubleshooting

This section gives some guidelines about how to troubleshoot the test cases owned by Functest.

IMPORTANT: As in the previous section, the steps defined below must be executed inside the Functest Docker container and after sourcing the OpenStack credentials:

```
. $creds
```

or:

```
source /home/opnfv/functest/conf/env_file
```

2.9.1 VIM

This section covers the test cases related to the VIM (healthcheck, vping_ssh, vping_userdata, tempest_smoke, tempest_full, rally_sanity, rally_full).

vPing common

For both vPing test cases (**vPing_ssh**, and **vPing_userdata**), the first steps are similar:

- Create Glance image
- Create Network
- Create Security Group
- Create Instances

After these actions, the test cases differ and will be explained in their respective section.

These test cases can be run inside the container, using new Functest CLI as follows:

os-nosdn-kvm-noha

vPing (ssh)	vPing (userdata)	Tempest (smoke)	Rally (smoke)	Promise
				

os-onos-sfc-ha

vPing (ssh)	vPing (userdata)	Tempest (smoke)	Rally (smoke)	ONOS	Promise	SFC
						

os-odl_l3-nofeature-ha

vPing (userdata)	Tempest (smoke)	Rally (smoke)	ODL	Promise
				

os-onos-nofeature-ha

vPing (ssh)	vPing (userdata)	Tempest (smoke)	Rally (smoke)	ONOS	Promise
					

```
$ run_tests -t vping_ssh
$ run_tests -t vping_userdata
```

The Functest CLI is designed to route a call to the corresponding internal python scripts, located in paths:

```
/usr/lib/python3.7/site-packages/functest/opnfv_tests/openstack/vping/vping_ssh.py
/usr/lib/python3.7/site-packages/functest/opnfv_tests/openstack/vping/vping_userdata.
↪py
```

Notes:

1. There is one difference, between the Functest CLI based test case execution compared to the earlier used Bash shell script, which is relevant to point out in troubleshooting scenarios:

   > The Functest CLI does **not yet** support the option to suppress clean-up of the generated OpenStack resources, following the execution of a test case.

   Explanation: After finishing the test execution, the corresponding script will remove, by default, all created resources in OpenStack (image, instances, network and security group). When troubleshooting, it is advisable sometimes to keep those resources in case the test fails and a manual testing is needed.

   It is actually still possible to invoke test execution, with suppression of OpenStack resource cleanup, however this requires invocation of a **specific Python script:** 'run_tests'. The OPNFV Functest Developer Guide provides guidance on the use of that Python script in such troubleshooting cases.

Some of the common errors that can appear in this test case are:

```
vPing_ssh- ERROR - There has been a problem when creating the neutron network....
```

This means that there has been some problems with Neutron, even before creating the instances. Try to create manually a Neutron network and a Subnet to see if that works. The debug messages will also help to see when it failed (subnet and router creation). Example of Neutron commands (using 10.6.0.0/24 range for example):

```
neutron net-create net-test
neutron subnet-create --name subnet-test --allocation-pool start=10.6.0.2,end=10.6.0.
↪100 \
--gateway 10.6.0.254 net-test 10.6.0.0/24
neutron router-create test_router
neutron router-interface-add <ROUTER_ID> test_subnet
neutron router-gateway-set <ROUTER_ID> <EXT_NET_NAME>
```

Another related error can occur while creating the Security Groups for the instances:

```
vPing_ssh- ERROR - Failed to create the security group...
```

In this case, proceed to create it manually. These are some hints:

```
neutron security-group-create sg-test
neutron security-group-rule-create sg-test --direction ingress --protocol icmp \
--remote-ip-prefix 0.0.0.0/0
neutron security-group-rule-create sg-test --direction ingress --ethertype IPv4 \
--protocol tcp --port-range-min 80 --port-range-max 80 --remote-ip-prefix 0.0.0.0/0
neutron security-group-rule-create sg-test --direction egress --ethertype IPv4 \
--protocol tcp --port-range-min 80 --port-range-max 80 --remote-ip-prefix 0.0.0.0/0
```

The next step is to create the instances. The image used is located in */home/opnfv/functest/data/cirros-0.4.0-x86_64-disk.img* and a Glance image is created with the name **functest-vping**. If booting the instances fails (i.e. the status is not **ACTIVE**), you can check why it failed by doing:

```
nova list
nova show <INSTANCE_ID>
```

It might show some messages about the booting failure. To try that manually:

```
nova boot --flavor m1.small --image functest-vping --nic net-id=<NET_ID> nova-test
```

This will spawn a VM using the network created previously manually. In all the OPNFV tested scenarios from CI, it never has been a problem with the previous actions. Further possible problems are explained in the following sections.

vPing_SSH

This test case creates a floating IP on the external network and assigns it to the second instance **opnfv-vping-2**. The purpose of this is to establish a SSH connection to that instance and SCP a script that will ping the first instance. This script is located in the repository under /usr/lib/python3.7/site-packages/functest/opnfv_tests/openstack/vping/ping.sh and takes an IP as a parameter. When the SCP is completed, the test will do a SSH call to that script inside the second instance. Some problems can happen here:

```
vPing_ssh- ERROR - Cannot establish connection to IP xxx.xxx.xxx.xxx. Aborting
```

If this is displayed, stop the test or wait for it to finish, if you have used the special method of test invocation with specific supression of OpenStack resource clean-up, as explained earler. It means that the Container can not reach the Public/External IP assigned to the instance **opnfv-vping-2**. There are many possible reasons, and they really depend on the chosen scenario. For most of the ODL-L3 and ONOS scenarios this has been noticed and it is a known limitation.

First, make sure that the instance **opnfv-vping-2** succeeded to get an IP from the DHCP agent. It can be checked by doing:

```
nova console-log opnfv-vping-2
```

If the message *Sending discover* and *No lease, failing* is shown, it probably means that the Neutron dhcp-agent failed to assign an IP or even that it was not responding. At this point it does not make sense to try to ping the floating IP.

If the instance got an IP properly, try to ping manually the VM from the container:

```
nova list
<grab the public IP>
ping <public IP>
```

If the ping does not return anything, try to ping from the Host where the Docker container is running. If that solves the problem, check the iptable rules because there might be some rules rejecting ICMP or TCP traffic coming/going from/to the container.

At this point, if the ping does not work either, try to reproduce the test manually with the steps described above in the vPing common section with the addition:

```
neutron floatingip-create <EXT_NET_NAME>
nova floating-ip-associate nova-test <FLOATING_IP>
```

Further troubleshooting is out of scope of this document, as it might be due to problems with the SDN controller. Contact the installer team members or send an email to the corresponding OPNFV mailing list for more information.

vPing_userdata

This test case does not create any floating IP neither establishes an SSH connection. Instead, it uses nova-metadata service when creating an instance to pass the same script as before (ping.sh) but as 1-line text. This script will be

executed automatically when the second instance **opnfv-vping-2** is booted.

The only known problem here for this test to fail is mainly the lack of support of cloud-init (nova-metadata service). Check the console of the instance:

```
nova console-log opnfv-vping-2
```

If this text or similar is shown:

```
checking http://169.254.169.254/2009-04-04/instance-id
failed 1/20: up 1.13. request failed
failed 2/20: up 13.18. request failed
failed 3/20: up 25.20. request failed
failed 4/20: up 37.23. request failed
failed 5/20: up 49.25. request failed
failed 6/20: up 61.27. request failed
failed 7/20: up 73.29. request failed
failed 8/20: up 85.32. request failed
failed 9/20: up 97.34. request failed
failed 10/20: up 109.36. request failed
failed 11/20: up 121.38. request failed
failed 12/20: up 133.40. request failed
failed 13/20: up 145.43. request failed
failed 14/20: up 157.45. request failed
failed 15/20: up 169.48. request failed
failed 16/20: up 181.50. request failed
failed 17/20: up 193.52. request failed
failed 18/20: up 205.54. request failed
failed 19/20: up 217.56. request failed
failed 20/20: up 229.58. request failed
failed to read iid from metadata. tried 20
```

it means that the instance failed to read from the metadata service. Contact the Functest or installer teams for more information.

Tempest

In the upstream OpenStack CI all the Tempest test cases are supposed to pass. If some test cases fail in an OPNFV deployment, the reason is very probably one of the following

Error	Details
Resources required for test-case execution are missing	Such resources could be e.g. an external network and access to the management subnet (adminURL) from the Functest docker container.
OpenStack components or services are missing or not configured properly	Check running services in the controller and compute nodes (e.g. with "systemctl" or "service" commands). Configuration parameters can be verified from the related .conf files located under '/etc/<component>' directories.
Some resources required for execution test cases are missing	The tempest.conf file, automatically generated by Rally in Functest, does not contain all the needed parameters or some parameters are not set properly. The tempest.conf file is located in directory 'root/.rally/verification/verifier-<UUID> /for-deployment-<UUID>' in the Functest Docker container. Use the "rally deployment list" command in order to check the UUID of the current deployment.

When some Tempest test case fails, captured traceback and possibly also the related REST API requests/responses are output to the console. More detailed debug information can be found from tempest.log file stored into related Rally deployment folder.

Functest offers a possibility to test a customized list of Tempest test cases. To enable that, add a new entry in docker/smoke/testcases.yaml on the "smoke" container with the following content:

```
-
    case_name: tempest_custom
    project_name: functest
    criteria: 100
    blocking: false
    description: >-
        The test case allows running a customized list of tempest
        test cases
    dependencies:
        installer: ''
        scenario: ''
    run:
        module: 'functest.opnfv_tests.openstack.tempest.tempest'
        class: 'TempestCustom'
```

Also, a list of the Tempest test cases must be provided to the container or modify the existing one in /usr/lib/python3.7/site-packages/functest/opnfv_tests/openstack/tempest/custom_tests/test_list.txt

Example of custom list of tests 'my-custom-tempest-tests.txt':

```
tempest.scenario.test_server_basic_ops.TestServerBasicOps.test_server_basic_
↪ops[compute,id-7fff3fb3-91d8-4fd0-bd7d-0204f1f180ba,network,smoke]
tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_network_basic_
↪ops[compute,id-f323b3ba-82f8-4db7-8ea6-6a895869ec49,network,smoke]
```

This is an example of running a customized list of Tempest tests in Functest:

```
sudo docker run --env-file env \
    -v $(pwd)/openstack.creds:/home/opnfv/functest/conf/env_file \
    -v $(pwd)/images:/home/opnfv/functest/images \
    -v $(pwd)/my-custom-testcases.yaml:/usr/lib/python3.7/site-packages/functest/ci/
↪testcases.yaml \
    -v $(pwd)/my-custom-tempest-tests.txt:/usr/lib/python3.7/site-packages/functest/
↪opnfv_tests/openstack/tempest/custom_tests/test_list.txt \
    opnfv/functest-smoke run_tests -t tempest_custom
```

Rally

The same error causes which were mentioned above for Tempest test cases, may also lead to errors in Rally as well.

Possible scenarios are:

- authenticate
- glance
- cinder
- heat
- keystone
- neutron

- nova
- quotas
- vm

To know more about what those scenarios are doing, they are defined in directory: /usr/lib/python3.7/site-packages/functest/opnfv_tests/openstack/rally/scenario For more info about Rally scenario definition please refer to the Rally official documentation. [3]

To check any possible problems with Rally, the logs are stored under */home/opnfv/functest/results/rally/* in the Functest Docker container.

2.9.2 Controllers

Opendaylight

If the Basic Restconf test suite fails, check that the ODL controller is reachable and its Restconf module has been installed.

If the Neutron Reachability test fails, verify that the modules implementing Neutron requirements have been properly installed.

If any of the other test cases fails, check that Neutron and ODL have been correctly configured to work together. Check Neutron configuration files, accounts, IP addresses etc.).

2.9.3 Features

Please refer to the dedicated feature user guides for details.

2.9.4 VNF

cloudify_ims

vIMS deployment may fail for several reasons, the most frequent ones are described in the following table:

Error	Comments
Keystone admin API not reachable	Impossible to create vIMS user and tenant
Impossible to retrieve admin role id	Impossible to create vIMS user and tenant
Error when uploading image from OpenStack to glance	impossible to deploy VNF
Cinder quota cannot be updated	Default quotas not sufficient, they are adapted in the script
Impossible to create a volume	VNF cannot be deployed
SSH connection issue between the Test Docker container and the VM	if vPing test fails, vIMS test will fail...
No Internet access from the VM	the VMs of the VNF must have an external access to Internet
No access to OpenStack API from the VM	Orchestrator can be installed but the vIMS VNF installation fails

Please note that this test case requires resources (8 VM (2Go) + 1 VM (4Go)), it is there fore not recommended to run it on a light configuration.

2.10 References

[2]: OpenStack Tempest documentation

[3]: Rally documentation

[4]: Functest in depth (Danube)

[5]: Clearwater vIMS blueprint

[6]: Security Content Automation Protocol

[7]: OpenSCAP web site

[8]: Refstack client

[9]: Defcore

[10]: OpenStack interoperability procedure

[11]: Robot Framework web site

[13]: SNAPS wiki

[14]: vRouter

[15]: Testing OpenStack Tempest part 1

[16]: OPNFV Test API

OPNFV main site: OPNFV official web site

Functest page: Functest wiki page

IRC support chan: #opnfv-functest

CHAPTER 3

Functest Developer Guide

3.1 Introduction

Functest is a project dealing with functional testing. The project produces its own internal test cases but can also be considered as a framework to support feature and VNF onboarding project testing.

Therefore there are many ways to contribute to Functest. You can:

- Develop new internal test cases
- Integrate the tests from your feature project
- Develop the framework to ease the integration of external test cases

Additional tasks involving Functest but addressing all the test projects may also be mentioned:

- The API / Test collection framework
- The dashboards
- The automatic reporting portals
- The testcase catalog

This document describes how, as a developer, you may interact with the Functest project. The first section details the main working areas of the project. The Second part is a list of "How to" to help you to join the Functest family whatever your field of interest is.

3.2 Functest developer areas

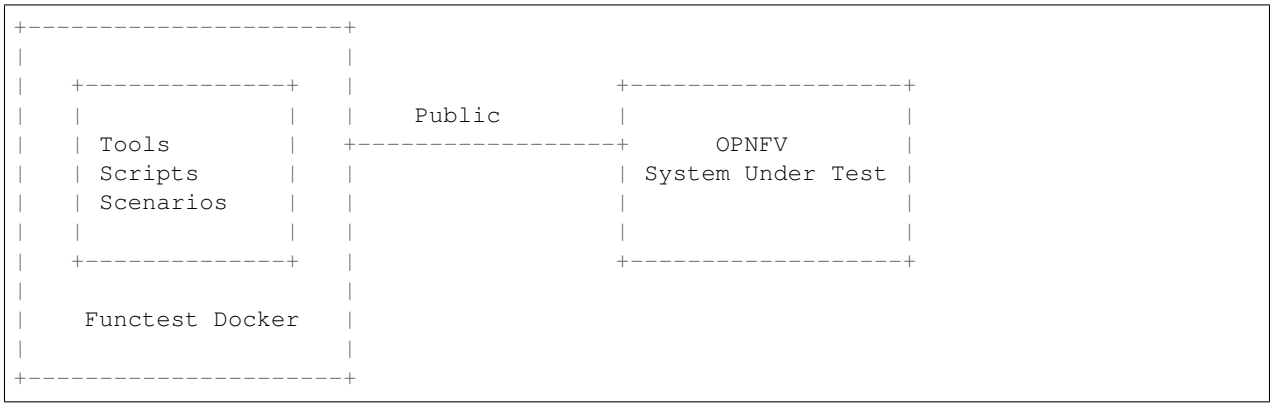
3.2.1 Functest High level architecture

Functest is a project delivering test containers dedicated to OPNFV. It includes the tools, the scripts and the test scenarios. In Euphrates Alpine containers have been introduced in order to lighten the container and manage testing slicing. The new containers are created according to the different tiers:

- functest-core: https://hub.docker.com/r/opnfv/functest-core/
- functest-healthcheck: https://hub.docker.com/r/opnfv/functest-healthcheck/
- functest-smoke: https://hub.docker.com/r/opnfv/functest-smoke/
- functest-vnf: https://hub.docker.com/r/opnfv/functest-vnf/
- functest-restapi: https://hub.docker.com/r/opnfv/functest-restapi/

Standalone functest dockers are maintained for Euphrates but Alpine containers are recommended.

Functest can be described as follow:



3.2.2 Functest internal test cases

The internal test cases in Euphrates are:

- connection_check
- vping_ssh
- vping_userdata
- odl
- rally_full
- rally_sanity
- tempest_smoke
- tempest_full
- cloudify_ims

By internal, we mean that this particular test cases have been developed and/or integrated by functest contributors and the associated code is hosted in the Functest repository. An internal case can be fully developed or a simple integration of upstream suites (e.g. Tempest/Rally developed in OpenStack, or odl suites are just integrated in Functest).

The structure of this repository is detailed in *[1]*. The main internal test cases are in the opnfv_tests subfolder of the repository, the internal test cases can be grouped by domain:

- sdn: odl, odl_fds
- openstack: connection_check, vping_ssh, vping_userdata, tempest_*, rally_*
- vnf: cloudify_ims

If you want to create a new test case you will have to create a new folder under the testcases directory (See next section for details).

3.2.3 Functest framework

Functest is a framework.

Historically Functest is released as a docker file, including tools, scripts and a CLI to prepare the environment and run tests. It simplifies the integration of external test suites in CI pipeline and provide commodity tools to collect and display results.

Since Colorado, test categories also known as **tiers** have been created to group similar tests, provide consistent sub-lists and at the end optimize test duration for CI (see How To section).

The definition of the tiers has been agreed by the testing working group.

The tiers are:

- healthcheck
- smoke
- benchmarking
- features
- vnf

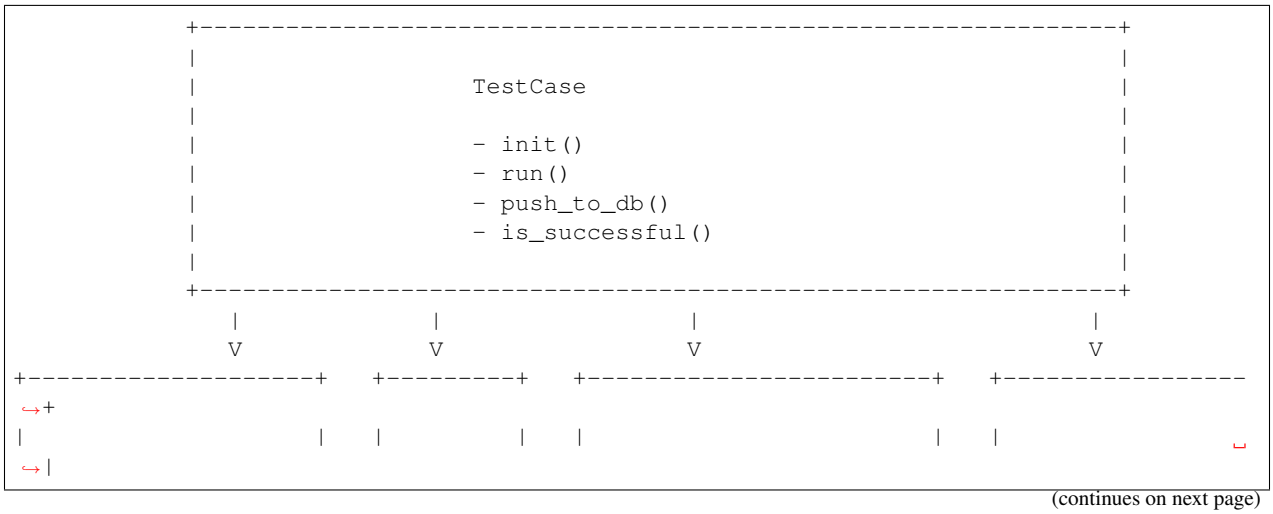
3.2.4 Functest abstraction classes

In order to harmonize test integration, abstraction classes have been introduced:

- testcase: base for any test case
- unit: run unit tests as test case
- feature: abstraction for feature project
- vnf: abstraction for vnf onboarding

The goal is to unify the way to run tests in Functest.

Feature, unit and vnf_base inherit from testcase:



(continues on next page)

(continued from previous page)

```
|    feature          |   |  unit   |   |    vnf                    |   | robotframework |
↪|
|                     |   |         |   |                           |   |                |
↪|
|                     |   |         |   |- prepare()                |   |                |
↪|
|  - execute()        |   |         |   |- deploy_orchestrator()    |   |                |
↪|
| BashFeature class   |   |         |   |- deploy_vnf()             |   |                |
↪|
|                     |   |         |   |- test_vnf()               |   |                |
↪|
|                     |   |         |   |- clean()                  |   |                |
↪|
+---------------------+   +---------+   +---------------------------+   +----------------
↪+
```

3.2.5 Functest util classes

In order to simplify the creation of test cases, Functest develops also some functions that are used by internal test cases. Several features are supported such as logger, configuration management and Openstack capabilities (tacker,..). These functions can be found under <repo>/functest/utils and can be described as follows:

```
functest/utils/
|-- config.py
|-- constants.py
|-- decorators.py
|-- env.py
|-- functest_utils.py
|-- openstack_tacker.py
`-- openstack_utils.py
```

3.2.6 TestAPI

Functest is using the Test collection framework and the TestAPI developed by the OPNFV community. See *[4]* for details.

3.2.7 Reporting

A web page is automatically generated every day to display the status based on jinja2 templates *[3]*.

3.2.8 Dashboard

Additional dashboarding is managed at the testing group level, see *[5]* for details.

3.3 How TOs

See How to section on Functest wiki *[6]*

3.4 References

[1]: http://artifacts.opnfv.org/functest/docs/configguide/index.html Functest configuration guide

[2]: http://artifacts.opnfv.org/functest/docs/userguide/index.html functest user guide

[3]: https://github.com/opnfv/releng-testresults/tree/master/reporting

[4]: https://wiki.opnfv.org/display/functest/2017+Beijing?preview=%2F11699623%2F11700523%2FTestAPI+-+test+results+collection+service.pptx

[5]: https://opnfv.biterg.io/login?nextUrl=%2Fgoto%2F283dba93ca18e95964f852c63af1d1ba

[6]: https://wiki.opnfv.org/pages/viewpage.action?pageId=7768932

IRC support chan: #opnfv-functest

CHAPTER 4

Functest Release Notes

4.1 OPNFV master release note for Functest

4.1.1 Abstract

This document contains the release notes of the Functest project.

4.1.2 OPNFV master Release

Functest deals with functional testing of the OPNFV solution. It includes test cases developed within the project, test cases developed in other OPNFV projects and it also integrates test cases from other upstream communities.

OpenStack

The internal test cases are:

- connection_check
- tenantnetwork1
- tenantnetwork2
- vmready1
- vmready2
- singlevm1
- singlevm2
- vping_ssh
- vping_userdata
- cinder_test

- tempest_smoke
- odl
- neutron-tempest-plugin-api
- rally_sanity
- refstack_compute
- refstack_object
- refstack_platform
- tempest_full
- tempest_scenario
- tempest_slow
- patrole
- barbican
- neutron_trunk
- networking-bgpvpn
- networking-sfc
- octavia
- rally_full
- rally_jobs
- vmtp
- shaker
- cloudify
- cloudify_ims
- heat_ims
- vyos_vrouter
- juju_epc

Kubernetes

The internal test cases are:

- k8s_smoke
- xrally_kubernetes
- k8s_conformance

4.1.3 Release Data

Project	functest
Repository branch	master

4.1.4 Deliverables

Software

Functest Docker images (OpenStack):

- https://hub.docker.com/r/opnfv/functest-healthcheck
- https://hub.docker.com/r/opnfv/functest-smoke
- https://hub.docker.com/r/opnfv/functest-benchmarking
- https://hub.docker.com/r/opnfv/functest-vnf

Functest Docker images (Kubernetes):

- https://hub.docker.com/r/opnfv/functest-kubernetes-healthcheck
- https://hub.docker.com/r/opnfv/functest-kubernetes-smoke

Docker tag for master: latest

Documents

- Functests Guides: https://functest.readthedocs.io/en/latest/
- API Docs: https://functest-api.readthedocs.io/en/latest/

4.1.5 Version change

Key changes

- update testcases and containers to OpenStack master and to Kubernetes master

Key benefits

- the enduser can easily deploy its own Functest toolchains in few commands
- everyone can pick stable Functest rolling releases (latest included)
- Functest can verify VIM in production even on Raspberry PI
- all testcases can run in parallel (tested with 4 executors in our gates)
- no remaining resources detected in our gates after multiple runs

Code quality

- pylint: 10.00/10
- code coverage: 70%

4.1.6 Useful links

- wiki project page: https://wiki.opnfv.org/display/functest/Opnfv+Functional+Testing
- Functest git repository: https://github.com/opnfv/functest
- Functest CI dashboard: https://build.opnfv.org/ci/view/functest/
- JIRA dashboard: https://jira.opnfv.org/secure/Dashboard.jspa?selectPageId=10611
- Functest IRC channel: #opnfv-functest
- Reporting page: http://testresults.opnfv.org/reporting/master/functest/functest.html

Build date: Oct 14, 2019