OPNFV FUNCTEST user guide

Release arno.2015.1.0 (604d2fc)

OPNFV

August 19, 2016

CHAPTER

ONE

INTRODUCTION

The goal of this document is to describe the OPNFV Functest test cases and to provide a procedure to execute them. In the OPNFV Colorado system release, a Functest CLI utility is introduced for easier execution of test procedures.

A overview presentation has been created for the first OPNFV Summit [4].

This document is a continuation of the OPNFV Functest Configuration Guide [1].

IMPORTANT: It is assumed here that the Functest Docker container is already properly deployed and that all instructions described in this guide are to be performed from *inside* the deployed Functest Docker container.

CHAPTER

TWO

OVERVIEW OF THE FUNCTEST SUITES

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

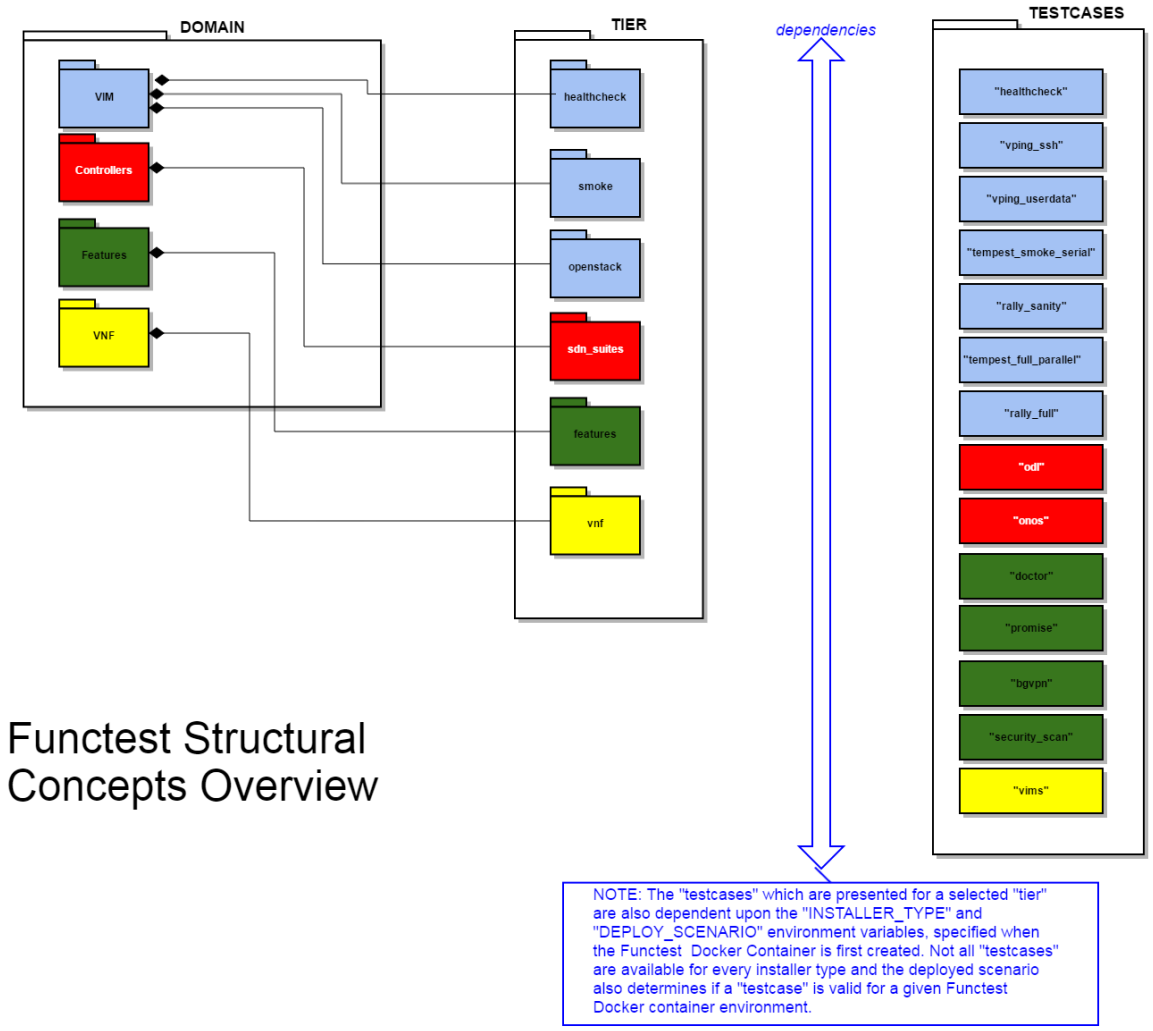
The current list of test suites can be distributed over 4 main domains: VIM (Virtualised Infrastructure Manager), Controllers (i.e. SDN Controllers), Features and VNF (Virtual Network Functions).

Domain	Tier	Test case	Comments
VIM	healthcheck	healthcheck	Verify basic operation in VIM
	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_serial	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
	openstack	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].
Controller	sdn_suites	odl	Opendaylight Test suite Limited test suite to check the basic neutron (Layer 2) operations mainly based on upstream testcases. See below for details
		onos	Test suite of ONOS L2 and L3 functions. See ONOSFW User Guide for details.
Features	features	Promise	Resource reservation and management project to identify NFV related requirements and realize resource reservation for future usage by capacity management of resource pools regarding compute, network and storage. See Promise User Guide for details.
		Doctor	Doctor platform, as of Colorado release, provides the three features: * Immediate Notification * Consistent resource state awareness for compute host down * Valid compute host status given to VM owner See Doctor User Guide for details
		bgpvpn	Implementation of the OpenStack bgpvpn API from the SDNVPN feature project. It allows for the creation of BGP VPNs. See SDNVPN User Guide for details
		security_scan	Implementation of a simple security scan. (Currently available only for the Apex installer environment)
		onos-sfc	SFC testing for onos scenarios TODO See for details
		odl-sfc	SFC testing for odl scenarios TODO See for details
		domino	Domino provides TOSCA template distribution service for network service and VNF descriptors among MANO components e.g., NFVO, VNFM, VIM, SDN-C, etc., as well as OSS/BSS functions. See Domino User Guide for details
		copper	Deployment policy TODO See for details
		multisites	Multisites TODO See for details
		moon	Security management system TODO See for details
VNF	vnf	vims	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
		parser	Parser is an integration project which aims to provide placement/deployment templates translation for OPNFV platform, including TOSCA -> HOT, POLICY -> TOSCA and YANG -> TOSCA. See Parser User Guide for details

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.

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.

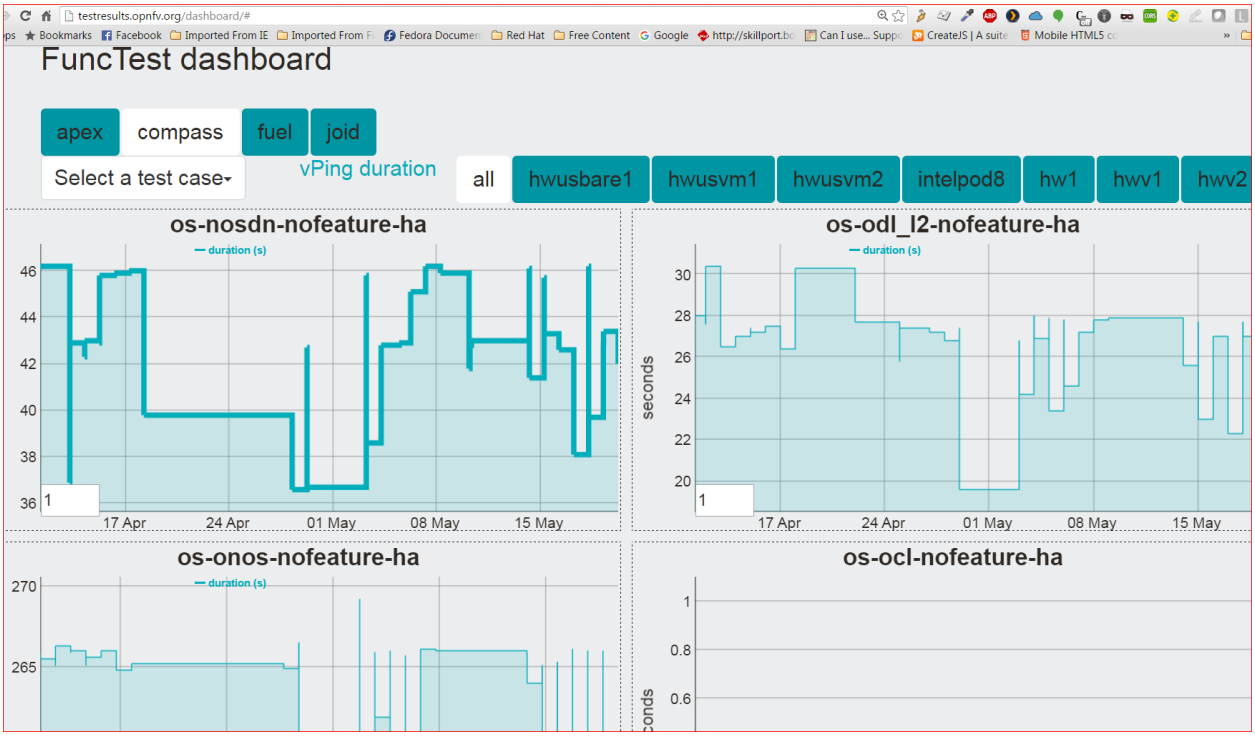
The results produced by the tests run from CI are pushed and collected into a NoSQL database. The goal is to populate the database with results from different sources and scenarios and to show them on a Functest Dashboard. A screenshot of a live Functest Dashboard is shown below:

There is no real notion of Test domain or Test coverage. 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.

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

Functest considers OPNFV as a black box. As of Colorado release the OPNFV offers a lot of potential combinations:

- 3 controllers (OpenDaylight, ONOS, OpenContrail)



- 4 installers (Apex, Compass, 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 features. The system uses the environment variables (INSTALLER_IP and DEPLOY_SCENARIO) to automatically determine the valid test cases; for each given environment.

In the Colorado OPNFV System release a convenience Functest CLI utility is also introduced 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 allow 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 detail in the section *Executing the functest suites* of this document.

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

2.1 VIM (Virtualized Infrastructure Manager)

2.1.1 Healthcheck

In Colorado release a new Tier 'healthcheck' with one testcase 'healthcheck' is introduced. The healthcheck testcase verifies that some basic IP connectivity and essential operations of OpenStack functionality over the command line are working correctly.

In particular, the following verifications are performed:

- DHCP agent functionality for IP address allocation
- Openstack Authentication management functionality via the Keystone API
- OpenStack Image management functionality via the Glance API
- OpenStack Block Storage management functionality via the Cinder API

- OpenStack Networking management functionality via the Neutron API
- Openstack Compute management functionality via the NOVA API

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

2.1.2 vPing_ssh

Given the script **ping.sh**:

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

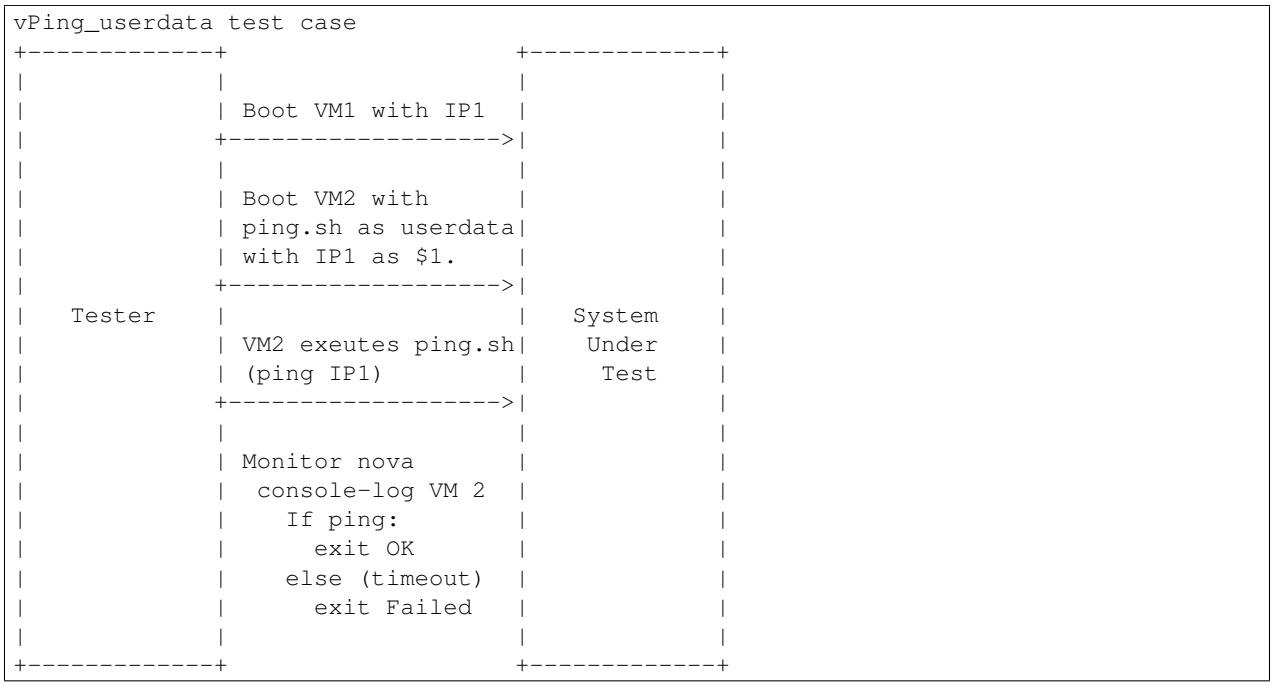
The goal of this test is to establish an SSH connection using a floating IP on the Public/External network and verify that 2 instances can talk over a Private Tenant network:

```
vPing_ssh test case
+-------------+                    +-------------+
|             |                    |             |
|             | Boot VM1 with IP1  |             |
|             +------------------->|             |
|   Tester    |                    |   System    |
|             | Boot VM2           |    Under    |
|             +------------------->|     Test    |
|             |                    |             |
|             | Create floating IP |             |
|             +------------------->|             |
|             |                    |             |
|             | Assign floating IP |             |
|             | to VM2             |             |
|             +------------------->|             |
|             |                    |             |
|             | Establish SSH      |             |
|             | connection to VM2  |             |
|             | through floating IP|             |
|             +------------------->|             |
|             |                    |             |
|             | SCP ping.sh to VM2 |             |
|             +------------------->|             |
|             |                    |             |
|             | VM2 executes       |             |
|             | ping.sh to VM1     |             |
|             +------------------->|             |
|             |                    |             |
|             |    If ping:        |             |
|             |      exit OK       |             |
|             |    else (timeout): |             |
|             |      exit Failed   |             |
```

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

2.1.3 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:



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.

2.1.4 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 accross two Tiers:

- Smoke Tier - Test Case 'tempest_smoke_serial'
- Openstack Tier - Test case 'tempest_full_parallel'

NOTE: Test case 'tempest_smoke_serial' executes a defined set of tempest smoke tests with a single thread (i.e. serial mode). Test case 'tempest_full_parallel' executes all defined Tempest tests using several concurrent threads (i.e. parallel mode). The number of threads activated corresponds to the number of available logical CPUs.

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.

2.1.5 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
- requests

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

The Rally testcases are distributed accross two Tiers:

- Smoke Tier - Test Case 'rally_sanity'
- Openstack 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.

2.2 SDN Controllers

There are currently 2 available controllers:

- OpenDaylight (ODL)
- ONOS

2.2.1 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 succesfully 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 succesfully deleted in OpenDaylight

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

2.2.2 ONOS

TestON Framework is used to test the ONOS SDN controller functions. The test cases deal with L2 and L3 functions. The ONOS test suite can be run on any ONOS compliant scenario.

The test cases are described as follows:

- onosfunctest: The main executable file contains the initialization of the docker environment and functions called by FUNCvirNetNB and FUNCvirNetNBL3
- FUNCvirNetNB
  - Create Network: Post Network data and check it in ONOS
  - Update Network: Update the Network and compare it in ONOS
  - Delete Network: Delete the Network and check if it's NULL in ONOS or not
  - Create Subnet: Post Subnet data and check it in ONOS
  - Update Subnet: Update the Subnet and compare it in ONOS
  - Delete Subnet: Delete the Subnet and check if it's NULL in ONOS or not
  - Create Port: Post Port data and check it in ONOS
  - Update Port: Update the Port and compare it in ONOS
  - Delete Port: Delete the Port and check if it's NULL in ONOS or not
- FUNCvirNetNBL3
  - Create Router: Post data for create Router and check it in ONOS

- Update Router: Update the Router and compare it in ONOS
- Delete Router: Delete the Router data and check it in ONOS
- Create RouterInterface: Post Router Interface data to an existing Router and check it in ONOS
- Delete RouterInterface: Delete the RouterInterface and check the Router
- Create FloatingIp: Post data for create FloatingIp and check it in ONOS
- Update FloatingIp: Update the FloatingIp and compare it in ONOS
- Delete FloatingIp: Delete the FloatingIp and check that it is 'NULL' in ONOS
- Create External Gateway: Post data to create an External Gateway for an existing Router and check it in ONOS
- Update External Gateway: Update the External Gateway and compare the change
- Delete External Gateway: Delete the External Gateway and check that it is 'NULL' in ONOS

2.3 Features

Most of the features have been developped by feature projects. Security_scan has been initiated in Functest repository but should soon be declared in its own repository as well.

Please refer to the dedicated feature user guides for details:

- bgpvpn: ** TODO link **
- copper: ** TODO link **
- doctor: ** TODO link **
- domino: ** TODO link **
- moon: ** TODO link **
- multisites: ** TODO link **
- onos-sfc: ** TODO link **
- odl-sfc: ** TODO link **
- promise: ** TODO link **

2.3.1 security_scan

Security Scanning, is a project to insure security compliance and vulnerability checks, as part of an automated CI / CD platform delivery process.

The project makes use of the existing SCAP format[6] to perform deep scanning of NFVi nodes, to insure they are hardened and free of known CVE reported vulnerabilities.

The SCAP content itself, is then consumed and run using an upstream opensource tool known as OpenSCAP[7].

The OPNFV Security Group have developed the code that will called by the OPNFV Jenkins build platform, to perform a complete scan. Resulting reports are then copied to the OPNFV functest dashboard.

The current work flow is as follows:

- Jenkins Build Initiated
- security_scan.py script is called, and a config file is passed to the script as an argument.

- The IP addresses of each NFVi node (compute / control), is gathered.
- A scan profile is matched to the node type.
- The OpenSCAP application is remotely installed onto each target node gathered on step 3, using upstream packaging (rpm and .deb).
- A scan is made against each node gathered within step 3.
- HTML Reports are downloaded for rendering on a dashboard.
- If the config file value 'clean' is set to 'True' then the application installed in step 5 is removed, and all reports created at step 6 are deleted.

At present, only the Apex installer is supported, with support for other installers due within D-release.

2.4 VNF

2.4.1 vIMS

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.

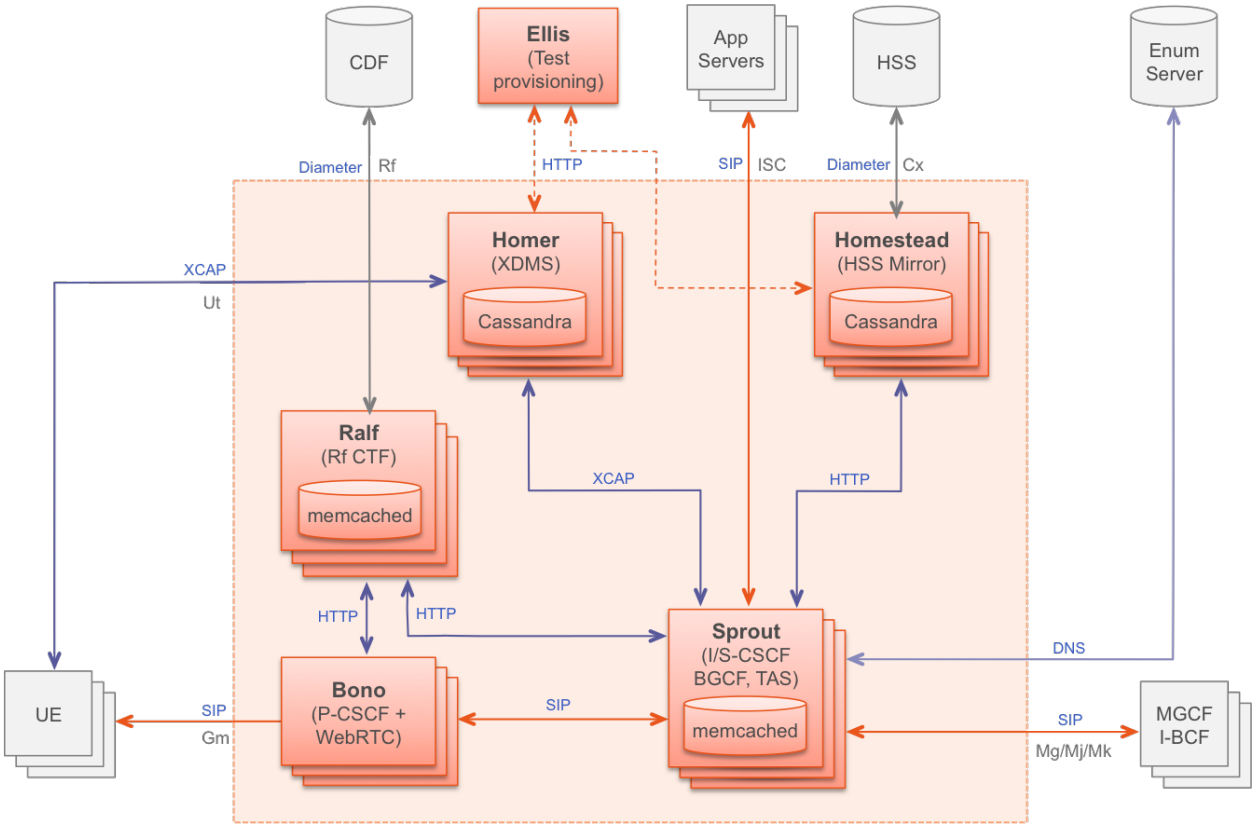
The goal of this test suite consists of:

- 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:

2.4.2 parser

See parser user guide for details: [12]


CDF
Ellis
(Test
provisioning)
App
Servers
HSS
Enum
Server
Diameter
Rf
HTTP
SIP
ISC
Diameter
Cx
Homer
(XDMS)
Cassandra
Homestead
(HSS Mirror)
Cassandra
XCAP
Ut
Ralf
(Rf CTF)
memcached
XCAP
HTTP
HTTP
HTTP
Sprout
(I/S-CSCF
BGCF, TAS)
memcached
DNS
UE
SIP
Gm
Bono
(P-CSCF +
WebRTC)
SIP
SIP
Mg/Mj/Mk
MGCF
I-BCF

CHAPTER

THREE

EXECUTING THE FUNCTEST SUITES

3.1 Manual testing

This section assumes the following:

- The Functest Docker container is running
- The docker prompt is shown
- The Functest environment is ready (Functest CLI command 'functest env prepare' has been executed)

If any of the above steps are missing please refer to the Functest Config Guide as they are a prerequisite and all the commands explained in this section **must** be performed **inside the container**.

Note: In Colorado release, the scripts **run_tests.sh** is now replaced with a new Functest CLI. One difference, is that tests run through the Functest CLI will always clean-up OpenStack resources. See the *Troubleshooting* section of this document, where this difference is discussed.

The Functest CLI offers two commands (functest tier ...) and (functest testcase ...) for the execution of Test Tiers or Test Cases:

```
root@22e436918db0:~/repos/functest/ci# functest tier --help
Usage: functest tier [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  get-tests  Prints the tests in a tier.
  list       Lists the available tiers.
  run        Executes all the tests within a tier.
  show       Shows information about a tier.
root@22e436918db0:~/repos/functest/ci# functest testcase --help

Usage: functest testcase [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  list  Lists the available testcases.
  run   Executes a test case.
  show  Shows information about a test case.
```

More details on the existing Tiers and Test Cases can be seen with the 'list' command:

```
root@22e436918db0:~/repos/functest/ci# functest tier list
    - 0. healthcheck:
           ['healthcheck']
    - 1. smoke:
           ['vping_ssh', 'vping_userdata', 'tempest_smoke_serial', 'rally_sanity']
    - 2. sdn_suites:
           ['odl']
    - 3. features:
           ['doctor', 'security_scan']
    - 4. openstack:
           ['tempest_full_parallel', 'rally_full']
    - 5. vnf:
           ['vims']

and

root@22e436918db0:~/repos/functest/ci# functest testcase list
healthcheck
vping_ssh
vping_userdata
tempest_smoke_serial
rally_sanity
odl
doctor
security_scan
tempest_full_parallel
rally_full
vims
```

More specific details on specific Tiers or Test Cases can be seen wih the 'show' command:

```
root@22e436918db0:~/repos/functest/ci# functest tier show smoke
+==============================================================================+
| Tier:  smoke                                                                 |
+==============================================================================+
| Order: 1                                                                     |
| CI Loop: (daily)|(weekly)                                                    |
| Description:                                                                 |
|    Set of basic Functional tests to validate the OpenStack                   |
|    deployment.                                                               |
| Test cases:                                                                  |
|    - vping_ssh                                                               |
|    - vping_userdata                                                          |
|    - tempest_smoke_serial                                                    |
|    - rally_sanity                                                            |
|                                                                              |
+------------------------------------------------------------------------------+

and

root@22e436918db0:~/repos/functest/ci# functest testcase  show tempest_smoke_serial
+==============================================================================+
| Testcase:  tempest_smoke_serial                                              |
+==============================================================================+
| Description:                                                                 |
|    This test case runs the smoke subset of the OpenStack Tempest             |
|    suite. The list of test cases is generated by Tempest                     |
|    automatically and depends on the parameters of the OpenStack              |
|    deplopyment.                                                              |
```

```
| Dependencies:                                                        |
|   - Installer:                                                       |
|   - Scenario :                                                       |
|                                                                      |
+----------------------------------------------------------------------+
```

To execute a Test Tier or Test Case, the 'run' command is used:

```
root@22e436918db0:~/repos/functest/ci# functest tier run healthcheck
Executing command: 'python /home/opnfv/repos/functest/ci/run_tests.py -t healthcheck'
2016-06-30 11:44:56,933 - run_tests - INFO - Sourcing the OpenStack RC file...
2016-06-30 11:44:56,937 - run_tests - INFO - ################################################
2016-06-30 11:44:56,938 - run_tests - INFO - Running tier 'healthcheck'
2016-06-30 11:44:56,938 - run_tests - INFO - ################################################
2016-06-30 11:44:56,938 - run_tests - INFO - ================================================
2016-06-30 11:44:56,938 - run_tests - INFO - Running test case 'healthcheck'...
2016-06-30 11:44:56,938 - run_tests - INFO - ================================================
2016-06-30 11:44:56,953 - healtcheck - INFO -  Testing Keystone API...
2016-06-30 11:45:05,351 - healtcheck - INFO -  ...Keystone OK!
2016-06-30 11:45:05,354 - healtcheck - INFO -  Testing Glance API...
2016-06-30 11:45:29,746 - healtcheck - INFO -  ... Glance OK!
2016-06-30 11:45:29,749 - healtcheck - INFO -  Testing Cinder API...
2016-06-30 11:45:37,502 - healtcheck - INFO -  ...Cinder OK!
2016-06-30 11:45:37,505 - healtcheck - INFO -  Testing Neutron API...
2016-06-30 11:45:39,664 - healtcheck - INFO -  External network found. ccd98ad6-d34a-4768-b03c-e28ec
2016-06-30 11:45:39,667 - healtcheck - INFO -  1. Create Networks...
2016-06-30 11:45:44,227 - healtcheck - INFO -  2. Create subnets...
2016-06-30 11:45:46,805 - healtcheck - INFO -  4. Create Routers...
2016-06-30 11:45:54,261 - healtcheck - INFO -  ...Neutron OK!
2016-06-30 11:45:54,264 - healtcheck - INFO -  Testing Nova API...
2016-06-30 11:47:12,272 - healtcheck - INFO -  ...Nova OK!
2016-06-30 11:47:12,274 - healtcheck - INFO -  Checking if instances get an IP from DHCP...
:
:
2016-06-30 11:48:17,832 - healtcheck - INFO -  ...DHCP OK!
2016-06-30 11:48:17,835 - healtcheck - INFO -  Health check passed!
2016-06-30 11:48:17,837 - clean_openstack - INFO - ++++++++++++++++++++++++++++++++
2016-06-30 11:48:17,837 - clean_openstack - INFO - Cleaning OpenStack resources...
2016-06-30 11:48:17,837 - clean_openstack - INFO - ++++++++++++++++++++++++++++++++
Version 1 is deprecated, use alternative version 2 instead.
WARNING:cinderclient.api_versions:Version 1 is deprecated, use alternative version 2 instead.
2016-06-30 11:48:18,272 - clean_openstack - INFO - Removing Nova instances...
2016-06-30 11:48:24,439 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:24,440 - clean_openstack - INFO - Removing Glance images...
2016-06-30 11:48:35,853 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:35,854 - clean_openstack - INFO - Removing Cinder volumes...
2016-06-30 11:48:37,344 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:37,344 - clean_openstack - INFO - Removing floating IPs...
2016-06-30 11:48:37,467 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:37,467 - clean_openstack - INFO - Removing Neutron objects
2016-06-30 11:48:53,633 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:53,633 - clean_openstack - INFO - Removing Security groups...
2016-06-30 11:48:53,689 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:53,689 - clean_openstack - INFO - Removing Users...
2016-06-30 11:48:54,444 - clean_openstack - INFO - ----------------------------------------
2016-06-30 11:48:54,444 - clean_openstack - INFO - Removing Tenants...
2016-06-30 11:48:54,711 - clean_openstack - INFO - ----------------------------------------

and
```

```
root@22e436918db0:~/repos/functest/ci# functest testcase run vping_ssh
Executing command: 'python /home/opnfv/repos/functest/ci/run_tests.py -t vping_ssh'
2016-06-30 11:50:31,861 - run_tests - INFO - Sourcing the OpenStack RC file...
2016-06-30 11:50:31,865 - run_tests - INFO - ================================================
2016-06-30 11:50:31,865 - run_tests - INFO - Running test case 'vping_ssh'...
2016-06-30 11:50:31,865 - run_tests - INFO - ================================================
2016-06-30 11:50:32,977 - vping_ssh - INFO - Creating image 'functest-vping' from '/home/opnfv/functe
2016-06-30 11:50:45,470 - vping_ssh - INFO - Creating neutron network vping-net...
2016-06-30 11:50:47,645 - vping_ssh - INFO - Creating security group  'vPing-sg'...
2016-06-30 11:50:48,843 - vping_ssh - INFO - Using existing Flavor 'm1.small'...
2016-06-30 11:50:48,927 - vping_ssh - INFO - vPing Start Time:'2016-06-30 11:50:48'
2016-06-30 11:50:48,927 - vping_ssh - INFO - Creating instance 'opnfv-vping-1'...
2016-06-30 11:51:34,664 - vping_ssh - INFO - Instance 'opnfv-vping-1' is ACTIVE.
2016-06-30 11:51:34,818 - vping_ssh - INFO - Adding 'opnfv-vping-1' to security group 'vPing-sg'...
2016-06-30 11:51:35,209 - vping_ssh - INFO - Creating instance 'opnfv-vping-2'...
2016-06-30 11:52:01,439 - vping_ssh - INFO - Instance 'opnfv-vping-2' is ACTIVE.
2016-06-30 11:52:01,439 - vping_ssh - INFO - Adding 'opnfv-vping-2' to security group 'vPing-sg'...
2016-06-30 11:52:01,754 - vping_ssh - INFO - Creating floating IP for VM 'opnfv-vping-2'...
2016-06-30 11:52:01,969 - vping_ssh - INFO - Floating IP created: '10.17.94.140'
2016-06-30 11:52:01,969 - vping_ssh - INFO - Associating floating ip: '10.17.94.140' to VM 'opnfv-vpi
2016-06-30 11:52:02,792 - vping_ssh - INFO - Trying to establish SSH connection to 10.17.94.140...
2016-06-30 11:52:19,915 - vping_ssh - INFO - Waiting for ping...
2016-06-30 11:52:21,108 - vping_ssh - INFO - vPing detected!
2016-06-30 11:52:21,108 - vping_ssh - INFO - vPing duration:'92.2' s.
2016-06-30 11:52:21,109 - vping_ssh - INFO - vPing OK
2016-06-30 11:52:21,153 - clean_openstack - INFO - ++++++++++++++++++++++++++++++++
2016-06-30 11:52:21,153 - clean_openstack - INFO - Cleaning OpenStack resources...
2016-06-30 11:52:21,153 - clean_openstack - INFO - ++++++++++++++++++++++++++++++++
Version 1 is deprecated, use alternative version 2 instead.
:
:
etc.
```

To list the test cases which are part of a specific Test Tier, the 'get-tests' command is used with 'functest tier':

```
root@22e436918db0:~/repos/functest/ci# functest tier get-tests sdn_suites
Test cases in tier 'sdn_suites':
 ['odl']
```

Please note that for some scenarios some test cases might not be launched. For example, the last example displayed only the 'odl' testcase for the given environment. In this particular system the deployment does not support the 'onos' SDN Controller Test Case; for example.

Important If you use the command 'functest tier run <tier_name>', then the Functest CLI utility will call **all valid Test Cases**, which belong to the specified Test Tier, as relevant to scenarios deployed to the SUT environment. Thus, the Functest CLI utility calculates automatically which tests can be executed and which cannot, given the environment variable **DEPLOY_SCENARIO**, which is passed in to the Functest docker container.

Currently, the Functest CLI command 'functest testcase run <testcase_name>', supports two possibilities:

```
*  Run a single Test Case, specified by a valid choice of <testcase_name>
*  Run ALL test Test Cases (for all Tiers) by specifying <testcase_name> = 'all'
```

Example:

```
root@22e436918db0:~/repos/functest/ci# functest testcase run all
Executing command: 'python /home/opnfv/repos/functest/ci/run_tests.py -t all'
2016-06-30 12:03:28,628 - run_tests - INFO - Sourcing the OpenStack RC file...
2016-06-30 12:03:28,634 - run_tests - INFO - Tiers to be executed:
```

```
    - 0. healthcheck:
           ['healthcheck']
    - 1. smoke:
           ['vping_ssh', 'vping_userdata', 'tempest_smoke_serial', 'rally_sanity']
    - 2. sdn_suites:
           ['odl']
    - 3. features:
           ['doctor', 'security_scan']
    - 4. openstack:
           ['tempest_full_parallel', 'rally_full']
    - 5. vnf:
           ['vims']
2016-06-30 12:03:28,634 - run_tests - INFO - ################################################
2016-06-30 12:03:28,635 - run_tests - INFO - Running tier 'healthcheck'
2016-06-30 12:03:28,635 - run_tests - INFO - ################################################
2016-06-30 12:03:28,635 - run_tests - INFO - ================================================
2016-06-30 12:03:28,635 - run_tests - INFO - Running test case 'healthcheck'...
2016-06-30 12:03:28,635 - run_tests - INFO - ================================================
2016-06-30 12:03:28,651 - healtcheck - INFO -  Testing Keystone API...
2016-06-30 12:03:36,676 - healtcheck - INFO -  ...Keystone OK!
2016-06-30 12:03:36,679 - healtcheck - INFO -  Testing Glance API...
:
:
etc.
```

Functest includes a cleaning mechanism in order to remove all the OpenStack resources except those present before running any test. The script *$repos_dir/functest/utils/generate_defaults.py* is called once when setting up the Functest environment (i.e. CLI command 'functest env prepare') to snapshot all the OpenStack resources (images, networks, volumes, security groups, tenants, users) so that an eventual cleanup does not remove any of these defaults.

The script **clean_openstack.py** which is located in *$repos_dir/functest/utils/* is normally called after a test execution. It is in charge of cleaning the OpenStack resources that are not specified in the defaults file generated previously which is stored in */home/opnfv/functest/conf/os_defaults.yaml* in the Functest docker container.

It is important to mention that if there are new OpenStack resources created manually after preparing the Functest environment, they will be removed, unless you use the special method of invoking the test case with specific suppression of clean up. (See the *Troubleshooting* section).

The reason to include this cleanup meachanism in Functest is because some test suites such as Tempest or Rally create a lot of resources (users, tenants, networks, volumes etc.) that are not always properly cleaned, so this function has been set to keep the system as clean as it was before a full Functest execution.

Although the Functest CLI provides an easy way to run any test, it is possible to do a direct call to the desired test script. For example:

```
python $repos_dir/functest/testcases/OpenStack/vPing/vPing_ssh.py -d
```

3.2 Automated testing

As mentioned previously, the Functest Docker container preparation as well as invocation of Test Cases can be called within the container from the Jenkins CI system. There are 3 jobs that automate the whole process. The first job runs all the tests referenced in the daily loop (i.e. that must been run daily), the second job runs the tests referenced in the weekly loop (usually long duration tests run once a week maximum) and the third job allows testing test suite by test suite specifying the test suite name. The user may also use either of these Jenkins jobs to execute the desired test suites.

One of the most challenging task in the Colorado release consists in dealing with lots of scenarios and installers. Thus,

when the tests are automatically started from CI, a basic algorithm has been created in order to detect whether a given test is runnable or not on the given scenario. Some Functest test suites cannot be systematically run (e.g. ODL suite can not be run on an ONOS scenario). Moreover since Colorado, we also introduce the notion of daily/weekly in order to save CI time and avoid running systematically long duration tests.

CI provides some useful information passed to the container as environment variables:

- Installer (apex|compass|fuel|joid), stored in INSTALLER_TYPE
- Installer IP of the engine or VM running the actual deployment, stored in INSTALLER_IP
- The scenario [controller]-[feature]-[mode], stored in DEPLOY_SCENARIO with
  - controller = (odl|onos|oc|nosdn)
  - feature = (ovs(dpdk)|kvm|sfc|bgpvpn|moon|multisites)
  - mode = (ha|noha)

The constraints per test case are defined in the Functest configuration file */home/opnfv/repos/functest/ci/testcases.yaml*:

```
tiers:
   -
       name: healthcheck
       order: 0
       ci_loop: '(daily)|(weekly)'
       description : >-
           First tier to be executed to verify the basic
           operations in the VIM.
       testcases:
           -
               name: healthcheck
               criteria: 'status == "PASS"'
               blocking: true
               description: >-
                   This test case verifies the basic OpenStack services like
                   Keystone, Glance, Cinder, Neutron and Nova.

               dependencies:
                   installer: ''
                   scenario: ''

  -
       name: smoke
       order: 1
       ci_loop: '(daily)|(weekly)'
       description : >-
           Set of basic Functional tests to validate the OpenStack deployment.
       testcases:
           -
               name: vping_ssh
               criteria: 'status == "PASS"'
               blocking: true
               description: >-
                   This test case verifies: 1) SSH to an instance using floating
                   IPs over the public network. 2) Connectivity between 2 instances
                   over a private network.
               dependencies:
                   installer: ''
                   scenario: '^((?!bgpvpn|odl_l3).)*$'
       ....
```

We may distinguish 2 levels in the test case description:

- Tier level
- Test case level

At the tier level, we define the following parameters:

- ci_loop: indicate if in automated mode, the test case must be run in daily and/or weekly jobs
- description: a high level view of the test case

For a given test case we defined:

- the name of the test case
- the criteria (experimental): a criteria used to declare the test case as PASS or FAIL
- blocking: if set to true, if the test is failed, the execution of the following tests is canceled
- the description of the test case
- the dependencies: a combination of 2 regex on the scenario and the installer name

The order of execution is the one defined in the file if all test cases are selected.

In CI daily job the tests are executed in the following order:

1. healthcheck (blocking)
2. smoke: both vPings are blocking
3. SDN controller suites (blocking)
4. Feature project tests cases

In CI weekly job we add 2 tiers:

5. vIMS suite
6. Rally suite

As explained before, at the end of an automated execution, the OpenStack resources might be eventually removed. Please note that a system snapshot is taken before any test case execution.

This testcase.yaml file is used for CI, for the CLI and for the automatic reporting.

CHAPTER

FOUR

TEST RESULTS

4.1 Manual testing

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

4.2 Automated testing

In automated mode, test results are displayed in jenkins logs, a summary is provided at the end of the job and can be described as follow:

```
+==============================================================================================
|                                                                 FUNCTEST REPORT
+==============================================================================================
|
|  Deployment description:
|    INSTALLER: fuel
|    SCENARIO:  os-odl_l2-nofeature-ha
|    BUILD TAG: jenkins-functest-fuel-baremetal-daily-master-324
|    CI LOOP:   daily
|
+=========================+===============+============+===============+=============================
| TEST CASE               | TIER          | DURATION   | RESULT        | URL
+=========================+===============+============+===============+=============================
| healthcheck             | healthcheck   | 03:07      | PASS          |
+-------------------------+---------------+------------+---------------+-----------------------------
| vping_ssh               | smoke         | 00:56      | PASS          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
| vping_userdata          | smoke         | 00:41      | PASS          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
| tempest_smoke_serial    | smoke         | 16:05      | FAIL          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
| rally_sanity            | smoke         | 12:19      | PASS          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
| odl                     | sdn_suites    | 00:24      | PASS          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
| promise                 | features      | 00:41      | PASS          | http://testresults.opnfv.org
+-------------------------+---------------+------------+---------------+-----------------------------
```

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
- Tempest: Tempest test case including reported errors per scenario and installer
- vIMS: vIMS details per scenario and installer

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
					

CHAPTER

FIVE

TEST DASHBOARD

Based on results collected in CI, a test dashboard is dynamically generated.

CHAPTER

SIX

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/openstack.creds
```

6.1 VIM

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

6.1.1 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:

```
$ functest testcase run vping_ssh
$ functest testcase run vping_userdata
```

The Functest CLI is designed to route a call to the corresponding internal python scripts, located in paths: *$repos_dir/functest/testcases/vPing/CI/libraries/vPing_ssh.py* and *$repos_dir/functest/testcases/vPing/CI/libraries/vPing_userdata.py*

Notes:

1. In this Colorado Funtest Userguide, the use of the Functest CLI is emphasized. The Functest CLI replaces the earlier Bash shell script *run_tests.sh*.

2. 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:** '/home/opnfv/repos/functest/ci/run_test.py'. 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-0.3.4-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.

6.1.2 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 *$repos_dir/functest/testcases/OpenStack/vPing/ping.sh* and takes an IP as a parameter. When the SCP is completed, the test will do an 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.

6.1.3 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.

NOTE: Cloud-init in not supported on scenarios dealing with ONOS and the tests have been excluded from CI in those scenarios.

6.1.4 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 '/home/opnfv/.rally/tempest/for-deployment-<UUID>' in the Functest Docker container. Use the "rally deployment list" command in order to check the UUID 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.

6.1.5 Rally

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

It is possible to run only one Rally scenario, instead of the whole suite. To do that, call the alternative python script as follows:

```
python $repos_dir/functest/testcases/OpenStack/rally/run_rally-cert.py -h
usage: run_rally-cert.py [-h] [-d] [-r] [-s] [-v] [-n] test_name

positional arguments:
  test_name      Module name to be tested. Possible values are : [
                 authenticate | glance | cinder | heat | keystone | neutron |
                 nova | quotas | requests | vm | all ] The 'all' value
                 performs all possible test scenarios

optional arguments:
  -h, --help     show this help message and exit
  -d, --debug    Debug mode
  -r, --report   Create json result file
  -s, --smoke    Smoke test mode
  -v, --verbose  Print verbose info about the progress
  -n, --noclean  Don't clean the created resources for this test.
```

For example, to run the Glance scenario with debug information:

```
python $repos_dir/functest/testcases/OpenStack/rally/run_rally-cert.py -d glance
```

Possible scenarios are:

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

To know more about what those scenarios are doing, they are defined in directory: *$repos_dir/functest/testcases/OpenStack/rally/scenario* For more info about Rally scenario definition please refer to the Rally official documentation. [3]

If the flag *all* is specified, it will run all the scenarios one by one. Please note that this might take some time (~1,5hr), taking around 1 hour alone to complete the Nova scenario.

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

6.2 Controllers

6.2.1 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.).

6.2.2 ONOS

Please refer to the ONOS documentation. ONOSFW User Guide .

6.3 Features

Please refer to the dedicated feature user guides for details.

6.3.1 security_scan

** TODO **

6.4 NFV

6.4.1 vIMS

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

6.4.2 parser

For now log info is the only way to do trouble shooting

CHAPTER

SEVEN

REFERENCES

OPNFV main site: opnfvmain.

OPNFV functional test page: opnfvfunctest.

IRC support chan: #opnfv-testperf