OPNFV FUNCTEST user guide

Release arno.2015.1.0 (9c3f0aa)

OPNFV

April 11, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

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

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

This document is a continuation of the Functest Configuration Guide‘[1]‘_ and it is assumed that the Functest Docker container is properly deployed.

IMPORTANT: All the instructions described in this guide must be performed inside the 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 in 3 main domains: VIM (Virtualised Infrastructure Manager), Controllers and Features.

Domain	Test suite	Comments
VIM	vPing	NFV "Hello World" using SSH connection and floatting IP
	vPing_userdata	Ping using userdata and cloud-init mechanism
	Tempest	OpenStack reference test suite [2]
	Rally bench	OpenStack testing tool benchmarking OpenStack modules [3]
Controllers	Open-Daylight	Opendaylight Test suite
	ONOS	Test suite of ONOS L2 and L3 functions See ONOSFW User Guide for details
Features	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
	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 Brahmaputra release , provides the two features: * Immediate Notification * Consistent resource state awareness (compute), see Doctor User Guide for details

Functest includes different test suites with several test cases within. Some of the tests 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 the tests that fit in OPNFV.

The Tempest suite has been customized but no new test cases have been created.

The results produced by the tests run from CI are pushed and collected in a NoSQL database. The goal is to populate the database with results from different sources and scenarios and to show them on a Dashboard.

There is no real notion of Test domain or Test coverage. Basic components (VIM, 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. OPNFV, since the Brahmaputra release, offers lots of potential combinations:

- 2 controllers (OpenDayligh, ONOS)
- 4 installers (Apex, Compass, Fuel, Joid)

Most of the tests are runnable on any combination, but some others might have restrictions imposed by the installers or the available deployed features.

The different test cases are described in the section hereafter.

2.1 VIM (Virtualized Infrastructure Manager)

2.1.1 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 network and verify that 2 instances can talk on a private network:

```
vPing_ssh test case
+-------------+                    +-------------+
|             |                    |             |
|             | Boot VM1 with IP1  |             |
|             +------------------->|             |
|   Tester    |                    |   System    |
|             | Boot VM2           |    Under    |
|             +------------------->|     Test    |
|             |                    |             |
|             | Create floating IP |             |
|             +------------------->|             |
|             |                    |             |
|             | Assign floating IP |             |
|             | to VM2             |             |
|             +------------------->|             |
|             |                    |             |
|             | Stablish 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 shall work on any deployment.

2.1.2 vPing_userdata

This test case is similar to vPing_ssh but without the use of floating ips and the public network. It only checks that 2 instances can talk to each other on a private network but it also verifies that the Nova metadata service is properly working:

```
vPing_userdata test case
+--------------+                    +--------------+
|              |                    |              |
|              | Boot VM1 with IP1  |              |
|              +------------------->|              |
|              |                    |              |
|              | Boot VM2 with      |              |
|              | ping.sh as userdata|              |
|              | with IP1 as $1.    |              |
|              +------------------->|              |
|   Tester     |                    |   System     |
|              | VM2 exeutes ping.sh|    Under     |
|              | (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 and the ping will be detected capturing periodically the output in the console-log of the second VM.

2.1.3 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 batteries 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 in a *log* file for further analysis.

As an addition of Arno, Brahmaputra runs a customized set of Tempest test cases. The list is specificed through *–tests-file* when executing the Rally command. This option has been introduced in the version 0.1.2 of the Rally framework.

This customized list contains more than 200 Tempest test cases and can be divided into two main sections:

1. Set of Tempest smoke test cases
2. Set of test cases from DefCore list [8]

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.

2.1.4 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) have been implemented.

2.2 SDN Controllers

Brahmaputra introduces new 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:

- Restconf.basic: Get the controller modules via Restconf
- Neutron.Networks
  - Check OpenStack Networks :: Checking OpenStack Neutron for known networks
  - Check OpenDaylight Networks :: Checking OpenDaylight Neutron API
  - Create Network :: Create new network in OpenStack
  - Check Network :: Check Network created in OpenDaylight
  - Neutron.Networks :: Checking Network created in OpenStack are pushed
- Neutron.Subnets
  - Check OpenStack Subnets :: Checking OpenStack Neutron for known Subnets
  - Check OpenDaylight subnets :: Checking OpenDaylight Neutron API
  - Create New subnet :: Create new subnet in OpenStack
  - Check New subnet :: Check new subnet created in OpenDaylight
  - Neutron.Subnets :: Checking Subnets created in OpenStack are pushed
- Neutron.Ports
  - Check OpenStack ports :: Checking OpenStack Neutron for known ports
  - Check OpenDaylight ports :: Checking OpenDaylight Neutron API
  - Create New Port :: Create new port in OpenStack
  - Check New Port :: Check new subnet created in OpenDaylight
  - Neutron.Ports :: Checking Port created in OpenStack are pushed
- Delete Ports
  - Delete previously created subnet in OpenStack
  - Check subnet deleted in OpenDaylight
  - Check subnet deleted in OpenStack
- Delete network
  - Delete previously created network in OpenStack
  - Check network deleted in OpenDaylight
  - Check network deleted in OpenStack

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 dataes for create Router and check it in ONOS
  - Update Router: Update the Router and compare it in ONOS
  - Delete Router: Delete the Router dataes and check it in ONOS
  - Create RouterInterface: Post RouterInterface data to an exist Router and check it in ONOS
  - Delete RouterInterface: Delete the RouterInterface and check the Router
  - Create FloatingIp: Post dataes 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 if it's NULL in ONOS or not
  - Create External Gateway: Post dataes for create External Gateway to an exit Router and check it
  - Update External Gateway: Update the External Gateway and compare it
  - Delete External Gateway: Delete the External Gateway and check if it's NULL in ONOS or not

2.3 Features

2.3.1 vIMS

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

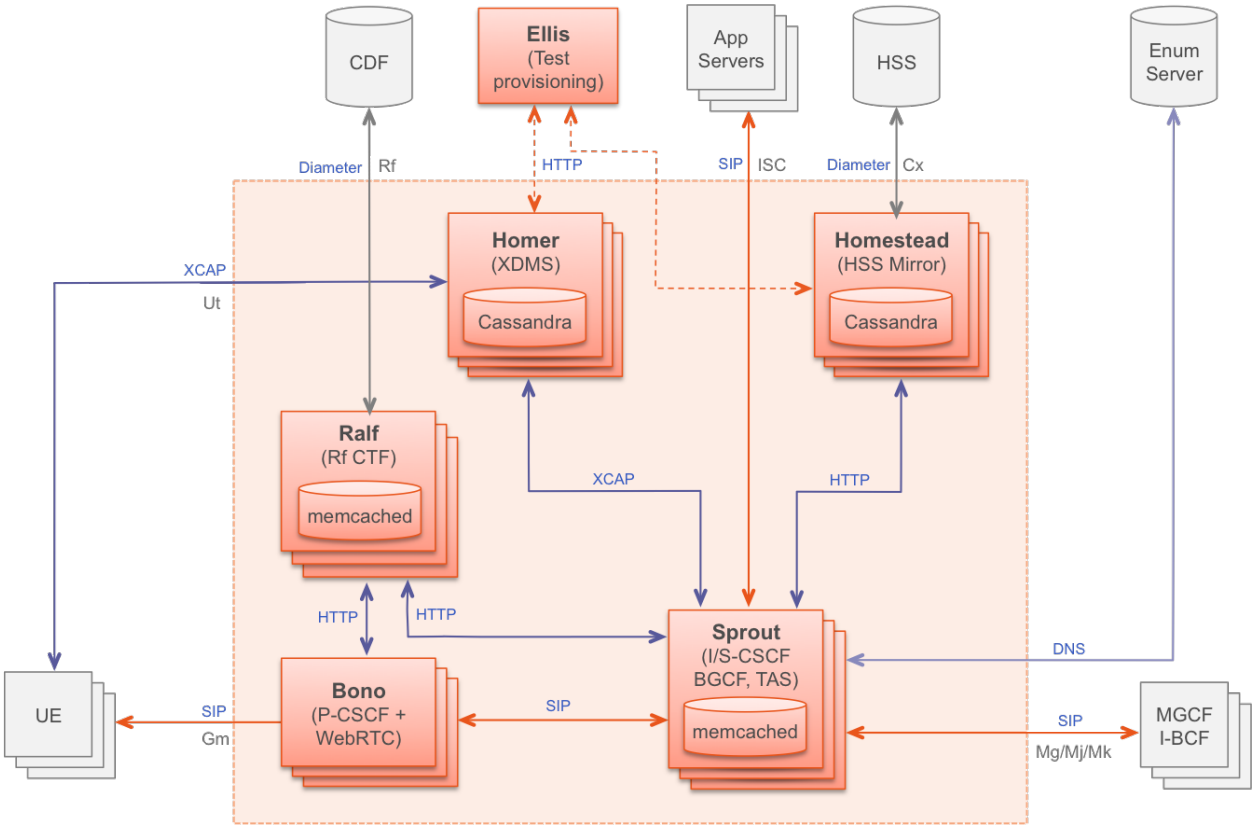
vIMS has been integrated in Functest to demonstrate the capability to deploy a relatively complex NFV scenario on the OPNFV platform. The deployment of a complete functional VNF allows the test of most of the essential functions needed for a NFV platform.

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:



2.3.2 Promise

Promise provides a basic set of test cases as part of Brahmaputra.

The available 33 test cases can be grouped into 7 test suites:

1. Add a new OpenStack provider into resource pool: Registers OpenStack into a new resource pool and adds more capacity associated with this pool.
2. Allocation without reservation: Creates a new server in OpenStack and adds a new allocation record in Promise shim-layer.
3. Allocation using reservation for immediate use: Creates a resource reservation record with no start/end time and immediately creates a new server in OpenStack and add a new allocation record in Promise shim-layer.
4. Reservation for future use: Creates a resource reservation record for a future start time, queries, modifies and cancels the newly created reservation.
5. Capacity planning: Decreases and increases the available capacity from a provider in the future and queries the available collections and utilizations.
6. Reservation with conflict: Tries to create reservations for immediate and future use with conflict.
7. Cleanup test allocations: Destroys all allocations in OpenStack.

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 (prepare_env.sh has been executed)

If any of the above steps is 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**.

The script **run_tests.sh** launches the test in an automated way. Although it is possible to execute the different tests manually, it is recommended to use the previous shell script which makes the call to the actual scripts with the appropriate parameters.

It is located in *$repos_dir/functest/docker* and it has several options:

```
./run_tests.sh -h
Script to trigger the tests automatically.

usage:
    bash run_tests.sh [-h|--help] [-r|--report] [-n|--no-clean] [-t|--test <test_name>]

where:
    -h|--help         show this help text
    -r|--report       push results to database (false by default)
    -n|--no-clean     do not clean up OpenStack resources after test run
    -s|--serial       run tests in one thread
    -t|--test         run specific set of tests
      <test_name>     one or more of the following separated by comma:
                         vping_ssh,vping_userdata,odl,onos,tempest,rally,vims,promise,doctor

examples:
    run_tests.sh
    run_tests.sh --test vping,odl
    run_tests.sh -t tempest,rally --no-clean
```

The *-r* option is used by the OPNFV Continuous Integration automation mechanisms in order to push the test results into the NoSQL results collection database. This database is read only for a regular user given that it needs special rights and special conditions to push data.

The *-t* option can be used to specify the list of a desired test to be launched, by default Functest will launch all the test suites in the following order:

1. vPing test cases
2. Tempest suite
3. SDN controller suites
4. Feature project tests cases (Promise, Doctor, ...)
5. vIMS suite
6. Rally suite

Please note that for some scenarios some test cases might not be launched. Functest calculates automatically which test can be executed and which cannot given the environment variable **DEPLOY_SCENARIO** to the docker container.

A single or set of test may be launched at once using *-t <test_name>* specifying the test name or names separated by commas in the following list: *[vping_ssh,vping_userdata,odl,onos,rally,tempest,vims,promise,doctor]*.

Functest includes cleaning mechanism in order to remove all the OpenStack resources except what was present before running any test. The script *$repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/generate_defaults.py* is called once by *prepare_env.sh* when setting up the Functest environment to snapshot all the OpenStack resources (images, networks, volumes, security groups, tenants, users) so that an eventual cleanup does not remove any of this defaults.

The *-n* option is used for preserving all the possible OpenStack resources created by the tests after their execution.

The *-s* option forces execution of test cases in a single thread. Currently this option affects Tempest test cases only and can be used e.g. for troubleshooting concurrency problems.

The script **clean_openstack.py** which is located in *$repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/* is normally called after a test execution if the *-n* is not specified. 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 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 if this flag is not specified in the *run_tests.sh* command. 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.

Within the Tempest test suite it is possible to define which test cases to execute by editing **test_list.txt** file before executing **run_tests.sh** script. This file is located in *$repos_dir/functest/testcases/VIM/OpenStack/CI/custom_tests/test_list.txt*

Although **run_tests.sh** 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/vPing/vPing_ssh.py -d
```

3.2 Automated testing

As mentioned previously, the **prepare-env.sh** and **run_test.sh** can be called within the container from Jenkins. There are 2 jobs that automate all the manual steps explained in the previous section. One job runs all the tests and the other one allows testing test suite by test suite specifying the test name. The user might use one or the other job to execute the desired test suites.

One of the most challenging task in the Brahmaputra 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).

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)
  - mode = (ha|noha)

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

```
test-dependencies:
   functest:
       vims:
           scenario: '(ocl)|(odl)|(nosdn)'
       vping:
       vping_userdata:
           scenario: '(ocl)|(odl)|(nosdn)'
       tempest:
       rally:
       odl:
           scenario: 'odl'
       onos:
           scenario: 'onos'
       ....
```

At the end of the Functest environment creation, a file */home/opnfv/functest/conf/testcase-list.txt* is created with the list of all the runnable tests. Functest considers the static constraints as regular expressions and compare them with the given scenario name. For instance, ODL suite can be run only on an scenario including 'odl' in its name.

The order of execution is also described in the Functest configuration file:

```
test_exec_priority:

   1: vping_ssh
   2: vping_userdata
   3: tempest
   4: odl
   5: onos
   6: ovno
   7: doctor
   8: promise
   9: odl-vpnservice
   10: bgpvpn
   11: openstack-neutron-bgpvpn-api-extension-tests
   12: vims
   13: rally
```

The tests are executed in the following order:

1. vPing test cases
2. Tempest suite
3. SDN controller suites
4. Feature project tests cases (Promise, Doctor, ...)
5. vIMS suite

6. Rally suite

As explained before, at the end of an automated execution, the OpenStack resources might be eventually removed.

CHAPTER

FOUR

TEST RESULTS

For Brahmaputra test results, see the functest results document at [12]

Note that the results are documented per scenario basis. Although most of the test cases might show the same output, some of them are not supported by certain scenario. Please select the appropriate scenario and compare the results to the referenced in the documentation.

CHAPTER

FIVE

TEST DASHBOARD

Based on results collected in CI, a test dashboard is dynamically generated. The URL of this dashboard is TODO LF

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 (vPing, Tempest, Rally).

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 as follows:

```
$repos_dir/functest/docker/run_tests.sh -t vping_ssh
$repos_dir/functest/docker/run_tests.sh -t vping_userdata
```

The **run_tests.sh** script is basically calling internally the corresponding vPing scripts, located in *$repos_dir/functest/testcases/vPing/CI/libraries/vPing_ssh.py* and *$repos_dir/functest/testcases/vPing/CI/libraries/vPing_userdata.py* with the appropriate flags.

After finishing the test execution, the corresponding script will remove 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. This can be achieved by adding the flag *-n*:

```
$repos_dir/functest/docker/run_tests.sh -n -t vping_ssh
$repos_dir/functest/docker/run_tests.sh -n -t vping_userdata
```

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
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
neutron security-group-rule-create sg-test --direction ingress --ethertype IPv4 --protocol tcp --port
neutron security-group-rule-create sg-test --direction egress --ethertype IPv4 --protocol tcp --port-
```

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 2 --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/vPing/CI/libraries/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 flag *-n* in **run_tests.sh** explained previously) so that the test does not clean the OpenStack resources. It means that the Container can not reach the public 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 scenario 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 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 /home/opnfv /.rally/tempest/for-deployment-<UUID> in Functest container Use "rally deployment list" command in order to check UUID of current deployment.

When some Tempest test case fails, captured traceback and possibly also 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

Same error causes than for Tempest mentioned above may lead to errors in Rally.

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

```
python $repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/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/VIM/OpenStack/CI/libraries/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: *$repos_dir/functest/testcases/VIM/OpenStack/CI/rally_cert/scenario*. For more info about Rally scenario definition please refer to the Rally official documentation.

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 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 container.

6.2 Controllers

6.2.1 ODL

2 versions are supported in Brahmaputra depending on the scenario:

- Lithium
- Berylium

The upstream test suites have not been adapted, so you may get 18 or 15 tests passed on 18 depending on your configuration. The 3 testcases are partly failed due to wrong return code.

6.2.2 ONOS

Please refer to the ONOS documentation.

6.3 Feature

6.3.1 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 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

6.3.2 Promise

Please refer to the Promise documentation.

CHAPTER

SEVEN

REFERENCES

OPNFV main site: opnfvmain.

OPNFV functional test page: opnfvfunctest.

IRC support chan: #opnfv-testperf