OPNFV FUNCTEST user guide

Release arno.2015.1.0 (880ad2f)

OPNFV

February 23, 2016

CHAPTER

ONE

INTRODUCTION

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

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

It is assumed that Functest container has been properly installed [1].

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:

Domain	Test suite	Comments
VIM (Virtualised Infrastructure Manager)	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	OpenDaylight	Opendaylight Test suite
	ONOS	Test suite of ONOS L2 and L3 functions
	OpenContrail	
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.
	SDNVPN	

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. Some OPNFV feature projects (e.g. SDNVPN) have written some Tempest tests cases and pushed upstream to be used by Functest.

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:

- 3 controllers (OpenDayligh, ONOS, OpenContrail)
- 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.

Details on working with the functest suites can be found at http://artifacts.opnfv.org/functest/brahmaputra/userguide/index.html

The different scenarios are described in the section hereafter.

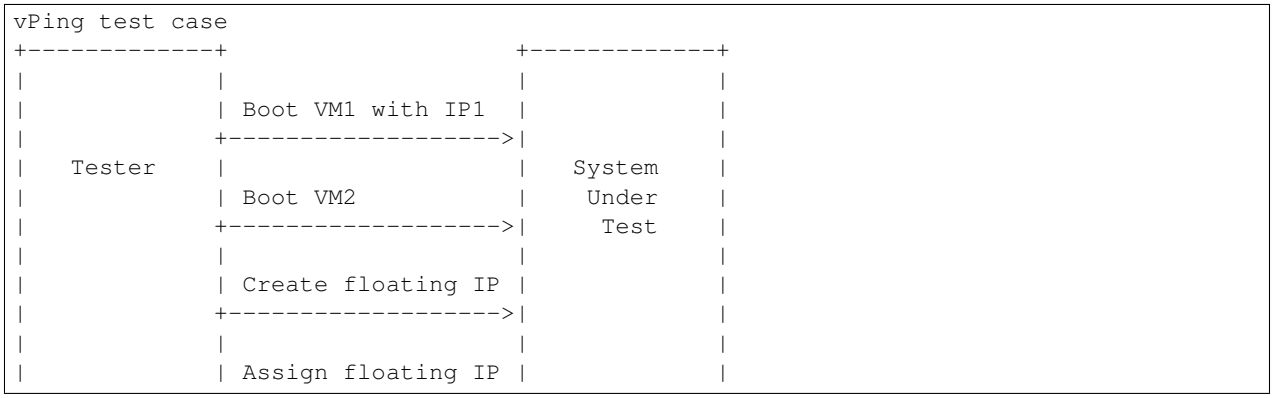
VIM (Virtualized Infrastructure Manager) —

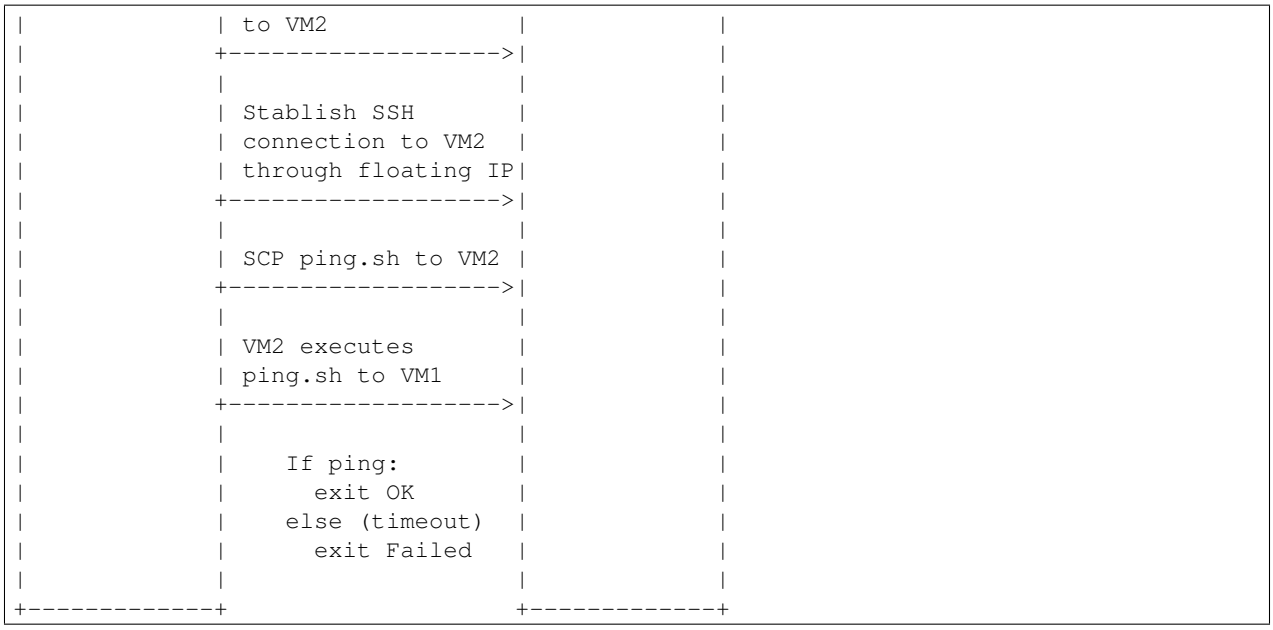
2.1 vPing_SSH

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 described as follows:

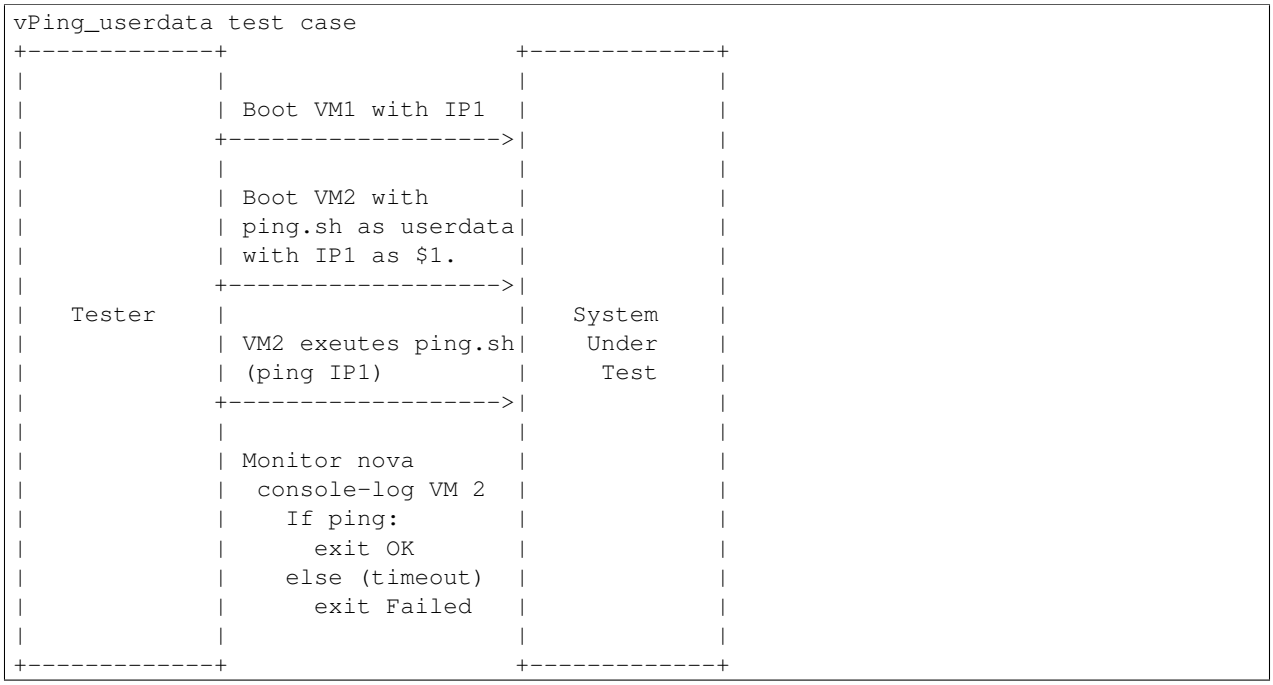




This test can be considered as an "Hello World" example. It is the first basic use case which shall work on any deployment.

2.2 vPing_userdata

The goal of this test can be described as follows:



This scenario is similar to the previous one but it uses cloud-init (nova metadata service) instead of floating IPs and SSH connection. When the second VM boots it will execute the script automatically and the ping will be detected capturing periodically the output in the console-log of the second VM.

2.3 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 and updates the appropriate parameters to the configuration file. When the Tempest suite is executed, each test duration is measured and the full console output is stored in the tempest.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 running Rally. This option has been introduced in Rally in version 0.1.2.

The customized test list is available in the Functest repo [4]. This 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 different OpenStack components on an OPNFV fresh installation using the corresponding REST API interfaces.

2.4 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 the different OpenStack modules and get significant figures that could help to define Telco Cloud KPIs.

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

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

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

2.4.1 SDN Controllers

Brahmaputra introduces new SDN controllers. There are currently 3 possible controllers:

- OpenDaylight (ODL)
- ONOS
- OpenContrail (OCL)

2.5 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

2.6 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 may be 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

2.7 OpenContrail

TODO OVNO

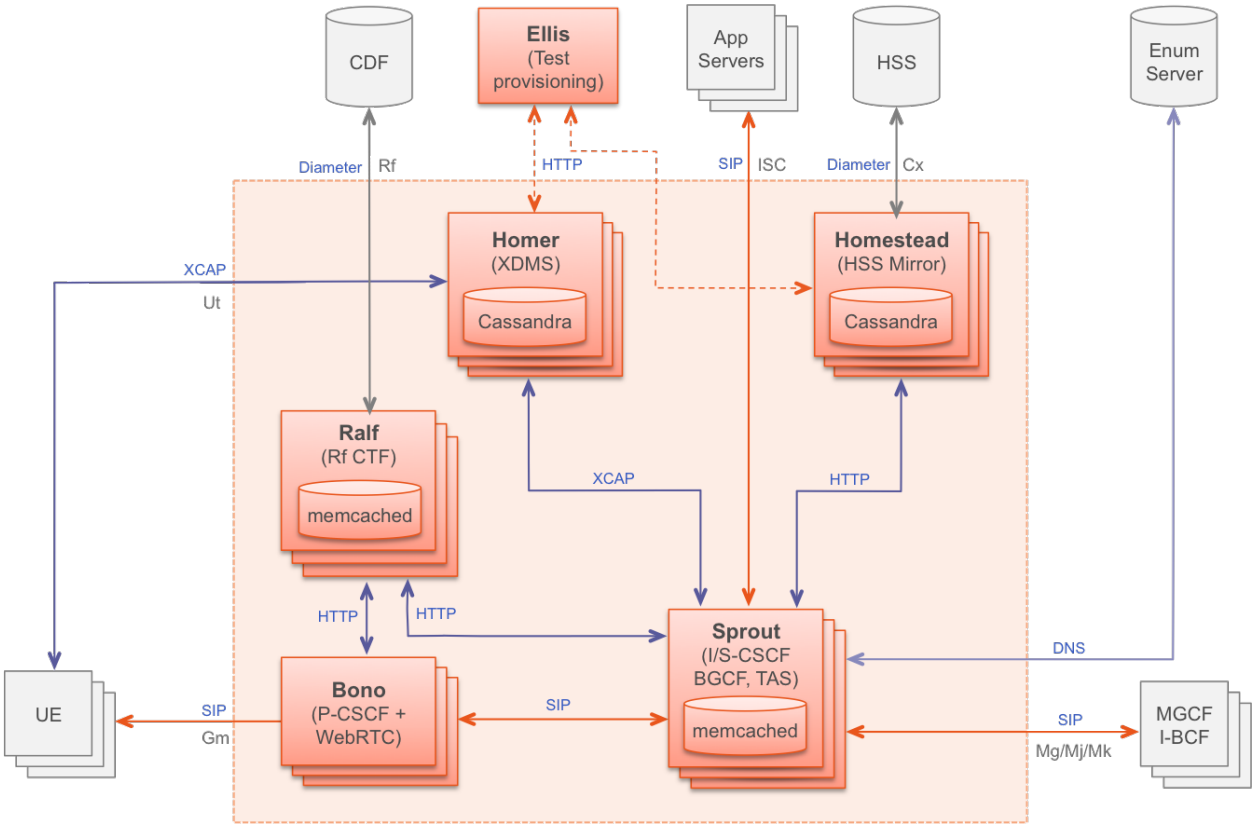
2.7.1 Features

2.8 vIMS

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:



Two types of information are stored in the Test Database:

- the duration of each step (orchestion deployment, VNF deployment and test)
- the test results

The deployment of a complete functional VNF allows the test of most of the essential functions needed for a NFV platform.

2.9 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.

The specific parameters for Promise can be found in config_functest.yaml and include:

```
promise:
  general:
      tenant_name: Name of the OpenStack tenant/project (e.g. promise)
      tenant_description: Description of the OpenStack tenant (e.g. promise Functionality Testing)
      user_name: Name of the user tenant (e.g. promiser)
      user_pwd: Password of the user tenant (e.g. test)
      image_name: Name of the software image (e.g. promise-img)
      flavor_name: Name of the flavor (e.g. promise-flavor with 1 vCPU and 512 MB RAM)
      flavor_vcpus: 1
      flavor_ram: 512
      flavor_disk: 0
```

However, these parameters must not be changed, as they are the values expected by the Promise test suite.

CHAPTER

THREE

EXECUTING THE FUNCTEST SUITES

Once the Functest docker container is running and Functest environment ready (through */home/opnfv/repos/functest/docker/prepare_env.sh* script), the system is ready to run the tests.

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,rally,tempest,vims,onos,promise,ovno

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: vPing, Tempest, vIMS, Rally.

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,vping_userdata,odl,rally,tempest,vims,onos,promise]*.

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

Please note that Functest includes cleaning mechanism in order to remove all the VIM 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 *-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 *$repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/clean_openstack.py* 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 cleaning 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.py -d
```

As mentioned in *[1]*, 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 (prepare_env.sh see [1]), 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:

- Basic scenario (vPing_ssh, vPing_userdata, Tempest)
- Controller suites: ODL or ONOS or OpenContrail
- Feature projects (promise, vIMS)
- Rally (benchmark scenario)

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: http://artifacts.opnfv.org/functest/brahmaputra/docs/results/index.html

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

6.1 vPing_SSH

vPing should work on all the scenarios. In case of timeout, check your network connectivity. The test case creates its own security group to allow SSH access, check your network settings and your security rules.

6.2 vPing_userdata

Cloud-init in not supported on scenario dealing with ONOS.

6.3 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.

6.4 Rally

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

6.4.1 Controllers

6.5 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.

6.6 ONOS

TODO

6.7 OpenContrail

6.7.1 Feature

6.8 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 | Impossible to create vIMS user and | | id | tenant | +----------+----------+ | Error when uploading image from | impossible to deploy VNF | | OpenStack to glance | | +----------+----------+ | 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 | if vPing test fails, vIMS test will| Test container and the VM | fail... | +----------+----------+ | No Internet access from a VM | the VMs of the VNF must have an | | | external access to Internet | +----------+----------+

6.9 Promise

TODO

CHAPTER

SEVEN

REFERENCES

OPNFV main site: opnfvmain.

OPNFV functional test page: opnfvfunctest.

IRC support chan: #opnfv-testperf