OPNFV Functest Documentation

Release master

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

Apr 24, 2019

Contents

Contents:

CHAPTER 1

functest

1.1 functest package

1.1.1 Subpackages

functest.ci package

Submodules

functest.ci.check_deployment module

OpenStack deployment checker

Verifies that:

- Credentials file is given and contains the right information
- OpenStack endpoints are reachable

class functest.ci.check_deployment.**CheckDeployment**(*rc_file='/var/lib/xtesting/conf/env_file'*)
Bases: object

Check deployment class.

check_all()
Calls all the class functions and returns 0 if all of them succeed. This is the method called by CLI

check_auth_endpoint()
Verifies connectivity to the OS_AUTH_URL given in the RC file and get auth token

check_ext_net()
checks if external network exists

check_glance()
checks that a simple glance operation works

check_neutron()
checks that a simple neutron operation works

check_nova()
checks that a simple nova operation works

check_public_endpoint()
Gets the public endpoint and verifies connectivity to it

check_rc()
Check if RC file exists and contains OS_AUTH_URL

check_service_endpoint(*service*)
Verifies connectivity to a given openstack service

functest.ci.check_deployment.**get_auth_token**(*os_creds*)
Get auth token

functest.ci.check_deployment.**main**()
Entry point

functest.ci.check_deployment.**verify_connectivity**(*endpoint*)
Returns true if an hostname/port is reachable

Module contents

functest.core package

Submodules

functest.core.cloudify module

Cloudify testcase implementation.

class functest.core.cloudify.**Cloudify**(***kwargs*)
Bases: *functest.core.singlevm.SingleVm2*

Cloudify Orchestrator Case.

cloudify_archive = '/home/opnfv/functest/images/cloudify-docker-manager-community-19.0

cloudify_container = 'docker-cfy-manager:latest'

create_server_timeout = 600

execute()
Deploy Cloudify Manager.

filename = '/home/opnfv/functest/images/ubuntu-16.04-server-cloudimg-amd64-disk1.img'

flavor_disk = 40

flavor_ram = 4096

flavor_vcpus = 2

ports = [80, 443, 5671, 53333]

prepare()
Create the security group and the keypair

It can be overriden to set other rules according to the services running in the VM

Raises: Exception on error

ssh_connect_loops = 12

username = 'ubuntu'

functest.core.cloudify.**get_execution_id**(*client*, *deployment_id*)
Get the execution id of a env preparation.

network, security group, fip, VM creation

functest.core.cloudify.**wait_for_execution**(*client*, *execution*, *logger*, *timeout=3600*)
Wait for a workflow execution on Cloudify Manager.

functest.core.singlevm module

Ease deploying a single VM reachable via ssh

It offers a simple way to create all tenant network resources + a VM for advanced testcases (e.g. deploying an orchestrator).

class functest.core.singlevm.**SingleVm1**(***kwargs*)
Bases: *functest.core.singlevm.VmReady1*

Deploy a single VM reachable via ssh (scenario1)

It inherits from TenantNetwork1 which creates all network resources and completes it by booting a VM attached to that network.

It ensures that all testcases inheriting from SingleVm1 could work without specific configurations (or at least read the same config data).

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

connect(*vm1*)
Connect to a virtual machine via ssh

It first adds a floating ip to the virtual machine and then establishes the ssh connection.

Returns: - (fip, ssh) - None on error

create_floating_ip_timeout = 120

execute()
Say hello world via ssh

It can be overriden to execute any command.

Returns: echo exit codes

prepare()
Create the security group and the keypair

It can be overriden to set other rules according to the services running in the VM

Raises: Exception on error

run(***kwargs*)
Boot the new VM

Here are the main actions: - add a new ssh key - boot the VM - create the security group - execute the right command over ssh

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

ssh_connect_loops = 6

ssh_connect_timeout = 1

username = 'cirros'

class functest.core.singlevm.**SingleVm2**(***kwargs*)
Bases: *functest.core.singlevm.SingleVm1*

Deploy a single VM reachable via ssh (scenario2)

It creates new user/project before creating and configuring all tenant network resources and vms required by advanced testcases.

It ensures that all testcases inheriting from SingleVm2 could work without specific configurations (or at least read the same config data).

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

class functest.core.singlevm.**VmReady1**(***kwargs*)
Bases: *functest.core.tenantnetwork.TenantNetwork1*

Prepare a single VM (scenario1)

It inherits from TenantNetwork1 which creates all network resources and prepares a future VM attached to that network.

It ensures that all testcases inheriting from SingleVm1 could work without specific configurations (or at least read the same config data).

boot_vm(*name=None*, ***kwargs*)
Boot the virtual machine

It allows booting multiple machines for the child testcases. It forces the same configuration for all subtestcases.

Returns: vm

Raises: expection on error

check_regex_in_console(*name*, *regex=' login: '*, *loop=1*)
Wait for specific message in console

Returns: True or False on errors

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

create_flavor(*name=None*)
Create flavor

It allows creating multiple flavors for the child testcases. It forces the same configuration for all subtestcases.

Returns: flavor

Raises: expection on error

create_flavor_alt (*name=None*)
Create flavor

It allows creating multiple alt flavors for the child testcases. It forces the same configuration for all sub-testcases.

Returns: flavor

Raises: expection on error

create_server_timeout = 180

extra_alt_properties = {}

extra_properties = {}

filename = '/home/opnfv/functest/images/cirros-0.4.0-x86_64-disk.img'

filename_alt = '/home/opnfv/functest/images/cirros-0.4.0-x86_64-disk.img'

flavor_alt_disk = 1

flavor_alt_extra_specs = {}

flavor_alt_ram = 1024

flavor_alt_vcpus = 1

flavor_disk = 1

flavor_extra_specs = {}

flavor_ram = 512

flavor_vcpus = 1

image_alt_format = 'qcow2'

image_format = 'qcow2'

publish_image (*name=None*)
Publish image

It allows publishing multiple images for the child testcases. It forces the same configuration for all sub-testcases.

Returns: image

Raises: expection on error

publish_image_alt (*name=None*)
Publish alternative image

It allows publishing multiple images for the child testcases. It forces the same configuration for all sub-testcases.

Returns: image

Raises: expection on error

run (***kwargs*)
Boot the new VM

Here are the main actions: - publish the image - create the flavor

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

visibility = 'private'

class functest.core.singlevm.**VmReady2**(***kwargs*)
Bases: *functest.core.singlevm.VmReady1*

Deploy a single VM reachable via ssh (scenario2)

It creates new user/project before creating and configuring all tenant network resources, flavors, images, etc. required by advanced testcases.

It ensures that all testcases inheriting from SingleVm2 could work without specific configurations (or at least read the same config data).

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

functest.core.tenantnetwork module

Ease deploying tenant networks

It offers a simple way to create all tenant network resources required by a testcase (including all Functest ones):

- TenantNetwork1 selects the user and the project set as env vars
- TenantNetwork2 creates a user and project to isolate the same resources

This classes could be reused by more complexed scenarios (Single VM)

class functest.core.tenantnetwork.**NewProject**(*cloud*, *case_name*, *guid*)
Bases: object

Ease creating new projects/users

clean()
Remove projects/users

create()
Create projects/users

class functest.core.tenantnetwork.**TenantNetwork1**(***kwargs*)
Bases: xtesting.core.testcase.TestCase

Create a tenant network (scenario1)

It creates and configures all tenant network resources required by advanced testcases (subnet, network and router).

It ensures that all testcases inheriting from TenantNetwork1 could work without network specific configurations (or at least read the same config data).

cidr = '192.168.120.0/24'

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

create_network_resources()
Create all tenant network resources

It creates a router which gateway is the external network detected. The new subnet is attached to that router.

Raises: expection on error

static get_default_role(*cloud*, *member='Member'*)
Get the default role

It also tests the role in lowercase to avoid possible conflicts.

static get_external_network(*cloud*)
Return the configured external network name or the first retrieved external network name

static get_public_auth_url(*cloud*)
Get Keystone public endpoint

run(***kwargs*)
Run the test case.

It allows running TestCase and getting its execution status.

The subclasses must override the default implementation which is false on purpose.

The new implementation must set the following attributes to push the results to DB:

- result,
- start_time,
- stop_time.

Args: kwargs: Arbitrary keyword arguments.

shared_network = False

class functest.core.tenantnetwork.**TenantNetwork2**(***kwargs*)
Bases: *functest.core.tenantnetwork.TenantNetwork1*

Create a tenant network (scenario2)

It creates new user/project before creating and configuring all tenant network resources required by a testcase (subnet, network and router).

It ensures that all testcases inheriting from TenantNetwork2 could work without network specific configurations (or at least read the same config data).

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

Module contents

functest.opnfv_tests package

Subpackages

functest.opnfv_tests.openstack package

Subpackages

functest.opnfv_tests.openstack.api package

Submodules

functest.opnfv_tests.openstack.api.connection_check module

Verify the connection to OpenStack Services

class functest.opnfv_tests.openstack.api.connection_check.**ConnectionCheck**(***kwargs*)
Bases: xtesting.core.testcase.TestCase

Perform simplest queries

run(***kwargs*)
Run all read operations to check connections

Module contents

functest.opnfv_tests.openstack.cinder package

Submodules

functest.opnfv_tests.openstack.cinder.cinder_test module

CinderCheck testcase.

class functest.opnfv_tests.openstack.cinder.cinder_test.**CinderCheck**(***kwargs*)
Bases: *functest.core.singlevm.SingleVm2*

CinderCheck testcase implementation.

Class to execute the CinderCheck test using 2 Floating IPs to connect to the VMs and one data volume

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

execute()
Execute CinderCheck testcase.

Sets up the OpenStack keypair, router, security group, and VM instance objects then validates cinder. :return: the exit code from the super.execute() method

prepare()
Create the security group and the keypair

It can be overriden to set other rules according to the services running in the VM

Raises: Exception on error

volume_timeout = 60

Module contents

functest.opnfv_tests.openstack.patrole package

Submodules

functest.opnfv_tests.openstack.patrole.patrole module

class functest.opnfv_tests.openstack.patrole.patrole.**Patrole**(***kwargs*)
Bases: *functest.opnfv_tests.openstack.tempest.tempest.TempestCommon*

configure(***kwargs*)
Create all openstack resources for tempest-based testcases and write tempest.conf.

run(***kwargs*)
Boot the new VM

Here are the main actions: - publish the image - create the flavor

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

Module contents

functest.opnfv_tests.openstack.rally package

Submodules

functest.opnfv_tests.openstack.rally.rally module

Rally testcases implementation.

class functest.opnfv_tests.openstack.rally.rally.**RallyBase**(***kwargs*)
Bases: *functest.core.singlevm.VmReady2*

Base class form Rally testcases implementation.

BLACKLIST_FILE = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/envs/s

CONCURRENCY = 4

ITERATIONS_AMOUNT = 10

RALLY_AARCH64_PATCH_PATH = '/home/docs/checkouts/readthedocs.org/user_builds/functest-

RALLY_CONF_PATH = '/etc/rally/rally.conf'

RALLY_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/envs/stable

RALLY_SCENARIO_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/en

SUPPORT_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/envs/stab

TASK_DIR = '/home/opnfv/functest/data/rally/task'

TEMPLATE_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/envs/sta

TEMP_DIR = '/home/opnfv/functest/data/rally/task/var'

TENANTS_AMOUNT = 3

TESTS = ['authenticate', 'glance', 'cinder', 'gnocchi', 'heat', 'keystone', 'neutron',

USERS_AMOUNT = 2

VOLUME_SERVICE_TYPE = 'volumev3'

VOLUME_VERSION = 3

apply_blacklist(*case_file_name*, *result_file_name*)
Apply blacklist.

clean()
Cleanup of OpenStack resources. Should be called on completion.

static clean_rally_conf(*rally_conf='/etc/rally/rally.conf'*)
Clean Rally config

static create_rally_deployment(*environ=None*)
Create new rally deployment

excl_func()
Exclude functionalities.

static excl_scenario()
Exclude scenario.

static export_task(*file_name*, *export_type='html'*)
Export all task results (e.g. html or xunit report)

Raises: subprocess.CalledProcessError: if Rally doesn't return 0

Returns: None

static file_is_empty(*file_name*)
Determine is a file is empty.

static get_task_id(*cmd_raw*)
Get task id from command rally result.

Parameters **cmd_raw** –

Returns task_id as string

static get_verifier_deployment_id()
Returns deployment id for active Rally deployment

static in_iterable_re(*needle*, *haystack*)
Check if given needle is in the iterable haystack, using regex.

Parameters

- **needle** – string to be matched
- **haystack** – iterable of strings (optionally regex patterns)

Returns True if needle is eqial to any of the elements in haystack, or if a nonempty regex pattern in haystack is found in needle.

is_successful()
The overall result of the test.

prepare_run(***kwargs*)
Prepare resources needed by test scenarios.

prepare_task(*test_name*)
Prepare resources for test run.

run(***kwargs*)
Run testcase.

run_task(*test_name*)
Run a task.

run_tests(**kwargs)
Execute tests.

shared_network = True

static task_succeed(*json_raw*)
Parse JSON from rally JSON results.

Parameters **json_raw** –

Returns Bool

static update_keystone_default_role(*rally_conf='/etc/rally/rally.conf'*)
Set keystone_default_role in rally.conf

static verify_report(*file_name*, *uuid*, *export_type='html'*)
Generate the verifier report (e.g. html or xunit report)

Raises: subprocess.CalledProcessError: if Rally doesn't return 0

Returns: None

visibility = 'public'

class functest.opnfv_tests.openstack.rally.rally.**RallyFull**(**kwargs)
Bases: *functest.opnfv_tests.openstack.rally.rally.RallyBase*

Rally full testcase implementation.

class functest.opnfv_tests.openstack.rally.rally.**RallyJobs**(**kwargs)
Bases: *functest.opnfv_tests.openstack.rally.rally.RallyBase*

Rally OpenStack CI testcase implementation.

TESTS = ['neutron']

apply_blacklist(*case_file_name*, *result_file_name*)
Apply blacklist.

clean()
Cleanup of OpenStack resources. Should be called on completion.

prepare_run(**kwargs)
Create resources needed by test scenarios.

prepare_task(*test_name*)
Prepare resources for test run.

class functest.opnfv_tests.openstack.rally.rally.**RallySanity**(**kwargs)
Bases: *functest.opnfv_tests.openstack.rally.rally.RallyBase*

Rally sanity testcase implementation.

Module contents

functest.opnfv_tests.openstack.refstack package

Submodules

functest.opnfv_tests.openstack.refstack.refstack module

Refstack testcase implementation.

class functest.opnfv_tests.openstack.refstack.refstack.**Refstack**(***kwargs*)
Bases: *functest.opnfv_tests.openstack.tempest.tempest.TempestCommon*

Refstack testcase implementation class.

defcorelist = '/home/opnfv/functest/data/refstack/defcore.txt'

generate_test_list(***kwargs*)
Generate test list based on the test mode.

Module contents

functest.opnfv_tests.openstack.shaker package

Submodules

functest.opnfv_tests.openstack.shaker.shaker module

Shaker wraps around popular system network testing tools like iperf, iperf3 and netperf (with help of flent). Shaker is able to deploy OpenStack instances and networks in different topologies. Shaker scenario specifies the deployment and list of tests to execute.

class functest.opnfv_tests.openstack.shaker.shaker.**Shaker**(***kwargs*)
Bases: *functest.core.singlevm.SingleVm2*

Run shaker full+perf l2 and l3

check_requirements()
Check the requirements of the test case.

It can be overriden on purpose.

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

create_server_timeout = 300

execute()

Returns:

- 0 if success
- 1 on operation error

filename = '/home/opnfv/functest/images/shaker-image.qcow2'

flavor_disk = 3

flavor_ram = 512

flavor_vcpus = 1

port = 9000

prepare()
Create the security group and the keypair

It can be overriden to set other rules according to the services running in the VM

Raises: Exception on error

```
shaker_timeout = '3600'
ssh_connect_loops = 12
username = 'ubuntu'
```

Module contents

functest.opnfv_tests.openstack.snaps package

Submodules

functest.opnfv_tests.openstack.snaps.api_check module

api_check test case implementation

class functest.opnfv_tests.openstack.snaps.api_check.**ApiCheck**(***kwargs*)

Bases: *functest.opnfv_tests.openstack.snaps.snaps_test_runner.SnapsTestRunner*

This test executes the Python Tests included with the SNAPS libraries that exercise many of the OpenStack APIs within Keystone, Glance, Neutron, and Nova

run(***kwargs*)

Builds the test suite then calls super.run() :param kwargs: the arguments to pass on :return:

functest.opnfv_tests.openstack.snaps.health_check module

snaps_health_check test case implementation

class functest.opnfv_tests.openstack.snaps.health_check.**HealthCheck**(***kwargs*)

Bases: *functest.opnfv_tests.openstack.snaps.snaps_test_runner.SnapsTestRunner*

This test executes the SNAPS Python Test case SimpleHealthCheck which creates a VM with a single port with an IPv4 address that is assigned by DHCP. This test then validates the expected IP with the actual

run(***kwargs*)

Builds the test suite then calls super.run() :param kwargs: the arguments to pass on :return:

functest.opnfv_tests.openstack.snaps.smoke module

snaps_smoke test case implementation

class functest.opnfv_tests.openstack.snaps.smoke.**SnapsSmoke**(***kwargs*)

Bases: *functest.opnfv_tests.openstack.snaps.snaps_test_runner.SnapsTestRunner*

This test executes the Python Tests included with the SNAPS libraries that exercise many of the OpenStack APIs within Keystone, Glance, Neutron, and Nova

run(***kwargs*)

Builds the test suite then calls super.run() :param kwargs: the arguments to pass on :return:

functest.opnfv_tests.openstack.snaps.snaps_suite_builder module

Snaps test suite including openstack client tests, api tests and integration tests. add_openstack_client_tests: for connection_check add_openstack_api_tests: for api_check add_openstack_integration_tests: for snaps_smoke

functest.opnfv_tests.openstack.snaps.snaps_suite_builder.**add_openstack_api_tests**(*suite*, *os_creds*, *ext_net_name*, *use_keystone=*, *image_metadata*, *log_level=20*)

Adds tests written to exercise all existing OpenStack APIs

Parameters

- **suite** – the unittest.TestSuite object to which to add the tests
- **os_creds** – Instance of OSCreds that holds the credentials required by OpenStack
- **ext_net_name** – the name of an external network on the cloud under test
- **use_keystone** – when True, tests requiring direct access to Keystone are added as these need to be running on a host that has access to the cloud's private network
- **image_metadata** – dict() object containing metadata for creating an image with custom config (see YAML files in examples/image-metadata)
- **log_level** – the logging level

Returns None as the tests will be adding to the 'suite' parameter object

functest.opnfv_tests.openstack.snaps.snaps_suite_builder.**add_openstack_client_tests**(*suite*, *os_creds*, *ext_net_n*, *use_keyst*, *log_level*

Adds tests written to exercise OpenStack client retrieval

Parameters

- **suite** – the unittest.TestSuite object to which to add the tests
- **os_creds** – and instance of OSCreds that holds the credentials required by OpenStack
- **ext_net_name** – the name of an external network on the cloud under test
- **use_keystone** – when True, tests requiring direct access to Keystone are added as these need to be running on a host that has access to the cloud's private network
- **log_level** – the logging level

Returns None as the tests will be adding to the 'suite' parameter object

functest.opnfv_tests.openstack.snaps.snaps_suite_builder.**add_openstack_integration_tests**(*su*
os
ex
us
fl
vo
im
ag
us
ne
co
lo

Adds tests written to exercise all long-running OpenStack integration tests meaning they will be creating VM instances and potentially performing some SSH functions through floatingIPs

Parameters

- **suite** – the unittest.TestSuite object to which to add the tests
- **os_creds** – and instance of OSCreds that holds the credentials required by OpenStack
- **ext_net_name** – the name of an external network on the cloud under test
- **use_keystone** – when True, tests requiring direct access to Keystone are added as these need to be running on a host that has access to the cloud's private network
- **image_metadata** – dict() object containing metadata for creating an image with custom config (see YAML files in examples/image-metadata)
- **flavor_metadata** – dict() object containing the metadata required by your flavor based on your configuration: (i.e. {'hw:mem_page_size': 'large'})
- **use_floating_ips** – when true, all tests requiring Floating IPs will be added to the suite
- **netconf_override** – dict() containing the reconfigured network_type, physical_network and segmentation_id
- **log_level** – the logging level

Returns None as the tests will be adding to the 'suite' parameter object

functest.opnfv_tests.openstack.snaps.snaps_test_runner module

configuration params to run snaps tests

class functest.opnfv_tests.openstack.snaps.snaps_test_runner.**SnapsTestRunner**(***kwargs*)

Bases: xtesting.core.unit.Suite

This test executes the SNAPS Python Tests

check_requirements()

Skip if OpenStack Rocky or newer.

clean()

Cleanup of OpenStack resources. Should be called on completion.

functest.opnfv_tests.openstack.snaps.snaps_utils module

Some common utils wrapping snaps functions

functest.opnfv_tests.openstack.snaps.snaps_utils.**get_active_compute_cnt**(*os_creds*)
Returns the number of active compute servers :param: os_creds: an instance of snaps OSCreds object :return: the number of active compute servers

functest.opnfv_tests.openstack.snaps.snaps_utils.**get_credentials**(*proxy_settings_str=None*, *ssh_proxy_cmd=None*, *overrides=None*)
Returns snaps OSCreds object instance :param: proxy_settings_str: proxy settings string <host>:<port> :param: ssh_proxy_cmd: the SSH proxy command for the environment :param overrides: dict() values to override in credentials :return: an instance of snaps OSCreds object

functest.opnfv_tests.openstack.snaps.snaps_utils.**get_ext_net_name**(*os_creds*)
Returns the configured external network name or the first retrieved external network name :param: os_creds: an instance of snaps OSCreds object :return:

Module contents

functest.opnfv_tests.openstack.tempest package

Submodules

functest.opnfv_tests.openstack.tempest.conf_utils module

Tempest configuration utilities.

functest.opnfv_tests.openstack.tempest.conf_utils.**CI_INSTALLER_TYPE = 'unknown'**
logging configuration

functest.opnfv_tests.openstack.tempest.conf_utils.**configure_tempest_update_params**(*tempest_con*
im-
age_id=Non
fla-
vor_id=Non
com-
pute_cnt=1,
im-
age_alt_id=
fla-
vor_alt_id=
ad-
min_role_na
cidr='192.1
do-
main_id='de
Add/update needed parameters into tempest.conf file

functest.opnfv_tests.openstack.tempest.conf_utils.**configure_verifier**(*deployment_dir*)
Execute rally verify configure-verifier, which generates tempest.conf

functest.opnfv_tests.openstack.tempest.conf_utils.**create_verifier**()
Create new verifier

functest.opnfv_tests.openstack.tempest.conf_utils.**get_verifier_deployment_dir**(*verifier_id*, *deployment_id*)
Returns Rally deployment directory for current verifier

functest.opnfv_tests.openstack.tempest.conf_utils.**get_verifier_id**()
Returns verifier id for current Tempest

functest.opnfv_tests.openstack.tempest.conf_utils.**get_verifier_repo_dir**(*verifier_id*)
Returns installed verifier repo directory for Tempest

functest.opnfv_tests.openstack.tempest.conf_utils.**update_tempest_conf_file**(*conf_file*, *rconfig*)
Update defined paramters into tempest config file

functest.opnfv_tests.openstack.tempest.tempest module

Tempest testcases implementation.

class functest.opnfv_tests.openstack.tempest.tempest.**TempestCommon**(***kwargs*)
Bases: *functest.core.singlevm.VmReady2*

TempestCommon testcases implementation class.

apply_tempest_blacklist()
Exclude blacklisted test cases.

static backup_tempest_config(*conf_file*, *res_dir*)
Copy config file to tempest results directory

check_extensions()
Check the mandatory network extensions.

check_requirements()
Check the requirements of the test case.

It can be overriden on purpose.

check_services()
Check the mandatory services.

clean()
Cleanup all OpenStack objects. Should be called on completion.

static clean_rally_conf(*rally_conf='/etc/rally/rally.conf'*)
Clean Rally config

configure(***kwargs*)
Create all openstack resources for tempest-based testcases and write tempest.conf.

filename_alt = '/home/opnfv/functest/images/cirros-0.4.0-x86_64-disk.img'

generate_test_list(***kwargs*)
Generate test list based on the test mode.

static get_verifier_result(*verif_id*)
Retrieve verification results.

is_successful()
The overall result of the test.

parse_verifier_result()
Parse and save test results.

static read_file(*filename*)
Read file and return content as a stripped list.

run(***kwargs*)
Boot the new VM

Here are the main actions: - publish the image - create the flavor

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

run_verifier_tests(***kwargs*)
Execute tempest test cases.

shared_network = True

update_compute_section()
Update compute section in tempest.conf

update_default_role(*rally_conf='/etc/rally/rally.conf'*)
Detect and update the default role if required

update_network_section()
Update network section in tempest.conf

update_rally_logs(*rally_conf='/etc/rally/rally.conf'*)
Print rally logs in res dir

update_rally_regex(*rally_conf='/etc/rally/rally.conf'*)
Set image name as tempest img_name_regex

update_scenario_section()
Update scenario section in tempest.conf

visibility = 'public'

Module contents

functest.opnfv_tests.openstack.vgpu package

Submodules

functest.opnfv_tests.openstack.vgpu.vgpu module

vGPU testcase implementation.

class functest.opnfv_tests.openstack.vgpu.vgpu.**VGPU**(***kwargs*)
Bases: *functest.core.singlevm.SingleVm2*

OpenStack vGPU Test Case.

create_server_timeout = 300

execute()
Test if the vGPU exist.

filename = '/home/opnfv/functest/images/ubuntu-16.04-server-cloudimg-amd64-disk1.img'

flavor_disk = 40

flavor_extra_specs = {'resources:VGPU': '1'}

flavor_ram = 4096

flavor_vcpus = 2

ssh_connect_loops = 12

username = 'ubuntu'

Module contents

functest.opnfv_tests.openstack.vmtp package

Submodules

functest.opnfv_tests.openstack.vmtp.vmtp module

VMTP is a small python application that will automatically perform ping connectivity, round trip time measurement (latency) and TCP/UDP throughput measurement for the following East/West flows on any OpenStack deployment:

- VM to VM same network (private fixed IP, flow #1)
- VM to VM different network using fixed IP (same as intra-tenant L3 fixed IP, flow #2)
- VM to VM different network using floating IP and NAT (same as floating IP inter-tenant L3, flow #3)

class functest.opnfv_tests.openstack.vmtp.vmtp.**Vmtp**(***kwargs*)
Bases: *functest.core.singlevm.VmReady2*

Class to run Vmtp as an OPNFV Functest testcase

check_requirements()
Check the requirements of the test case.

It can be overriden on purpose.

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

create_network_resources()
Create router

It creates a router which gateway is the external network detected.

Raises: expection on error

create_server_timeout = 300

filename = '/home/opnfv/functest/images/ubuntu-14.04-server-cloudimg-amd64-disk1.img'

flavor_disk = 0

flavor_ram = 2048

flavor_vcpus = 1

generate_keys()
Generate Keys

Raises: Exception on error

run(**kwargs)
Boot the new VM

Here are the main actions: - publish the image - create the flavor

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

run_vmtp()
Run Vmtp and generate charts

Raises: Exception on error

write_config()
Write vmtp.conf

Raises: Exception on error

Module contents

functest.opnfv_tests.openstack.vping package

Submodules

functest.opnfv_tests.openstack.vping.vping_ssh module

vPingSSH testcase.

class functest.opnfv_tests.openstack.vping.vping_ssh.**VPingSSH**(**kwargs)
Bases: *functest.core.singlevm.SingleVm2*

VPingSSH testcase implementation.

Class to execute the vPing test using a Floating IP to connect to one VM to issue the ping command to the second

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

execute()
Ping the second VM

Returns: ping exit codes

prepare()
Create the security group and the keypair

It can be overriden to set other rules according to the services running in the VM

Raises: Exception on error

functest.opnfv_tests.openstack.vping.vping_userdata module

vping_userdata testcase.

class functest.opnfv_tests.openstack.vping.vping_userdata.**VPingUserdata**(**kwargs)
Bases: *functest.core.singlevm.VmReady2*

Class to execute the vPing test using userdata and the VM's console

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

run(***kwargs*)
Sets up the OpenStack VM instance objects then executes the ping and validates. :return: the exit code from the super.execute() method

Module contents

Module contents

functest.opnfv_tests.sdn package

Subpackages

functest.opnfv_tests.sdn.odl package

Submodules

functest.opnfv_tests.sdn.odl.odl module

Define classes required to run ODL suites.

It has been designed for any context. But helpers are given for running test suites in OPNFV environment.

Example: $ python odl.py

class functest.opnfv_tests.sdn.odl.odl.**ODLParser**
Bases: object

Parser to run ODL test suites.

parse_args(*argv=None*)
Parse arguments.

It can call sys.exit if arguments are incorrect.

Returns: the arguments from cmdline

class functest.opnfv_tests.sdn.odl.odl.**ODLTests**(***kwargs*)
Bases: xtesting.core.robotframework.RobotFramework

ODL test runner.

basic_suite_dir = '/src/odl_test/csit/suites/integration/basic'

default_suites = ['/src/odl_test/csit/suites/integration/basic', '/src/odl_test/csit/s

neutron_suite_dir = '/src/odl_test/csit/suites/openstack/neutron'

odl_test_repo = '/src/odl_test'

odl_variables_file = '/src/odl_test/csit/variables/Variables.robot'

run(***kwargs*)
Run suites in OPNFV environment

It basically checks env vars to call main() with the keywords required.

Args: kwargs: Arbitrary keyword arguments.

Returns: EX_OK if all suites ran well. EX_RUN_ERROR otherwise.

run_suites(*suites=None*, ***kwargs*)
Run the test suites

It has been designed to be called in any context. It requires the following keyword arguments:

- odlusername,
- odlpassword,
- osauthurl,
- neutronurl,
- osusername,
- osprojectname,
- ospassword,
- odlip,
- odlwebport,
- odlrestconfport.

Here are the steps:

- set all RobotFramework_variables,
- create the output directories if required,
- get the results in output.xml,
- delete temporary files.

Args: kwargs: Arbitrary keyword arguments.

Returns: EX_OK if all suites ran well. EX_RUN_ERROR otherwise.

classmethod set_robotframework_vars(*odlusername='admin'*, *odlpassword='admin'*)
Set credentials in csit/variables/Variables.robot.

Returns: True if credentials are set. False otherwise.

functest.opnfv_tests.sdn.odl.odl.**main**()
Entry point

Module contents

Module contents

functest.opnfv_tests.vnf package

Subpackages

functest.opnfv_tests.vnf.epc package

Submodules

functest.opnfv_tests.vnf.epc.juju_epc module

Juju testcase implementation.

class functest.opnfv_tests.vnf.epc.juju_epc.**JujuEpc**(***kwargs*)
Bases: *functest.core.singlevm.VmReady2*

Abot EPC deployed with JUJU Orchestrator Case

check_app(*name='abot-epc-basic'*, *status='active'*)
Check application status.

check_requirements()
Check the requirements of the test case.

It can be overriden on purpose.

cidr = '192.168.121.0/24'

clean()
Clean created objects/functions.

deploy_orchestrator()
Create network, subnet, router

Bootstrap juju

deploy_vnf()
Deploy ABOT-OAI-EPC.

filename = '/home/opnfv/functest/images/ubuntu-16.04-server-cloudimg-amd64-disk1.img'

filename_alt = '/home/opnfv/functest/images/ubuntu-14.04-server-cloudimg-amd64-disk1.i

flavor_alt_disk = 10

flavor_alt_ram = 4096

flavor_alt_vcpus = 1

flavor_disk = 10

flavor_ram = 2048

flavor_vcpus = 1

juju_timeout = '4800'

prepare()
Prepare testcase (Additional pre-configuration steps).

publish_image(*name=None*)
Publish image

It allows publishing multiple images for the child testcases. It forces the same configuration for all sub-testcases.

Returns: image

Raises: expection on error

publish_image_alt(*name=None*)
Publish alternative image

It allows publishing multiple images for the child testcases. It forces the same configuration for all sub-testcases.

Returns: image

Raises: expection on error

run(***kwargs*)
Boot the new VM

Here are the main actions: - publish the image - create the flavor

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

test_vnf()
Run test on ABoT.

functest.opnfv_tests.vnf.epc.juju_epc.**process_abot_test_result**(*file_path*)
Process ABoT Result

functest.opnfv_tests.vnf.epc.juju_epc.**sig_test_format**(*sig_test*)
Process the signaling result to have a short result

functest.opnfv_tests.vnf.epc.juju_epc.**update_data**(*obj*)
Update Result data

Module contents

functest.opnfv_tests.vnf.ims package

Submodules

functest.opnfv_tests.vnf.ims.clearwater module

Ease testing any Clearwater deployment

class functest.opnfv_tests.vnf.ims.clearwater.**ClearwaterTesting**(*case_name*, *ellis_ip*)
Bases: object

vIMS clearwater base usable by several orchestrators

availability_check(*signup_code='secret'*, *two_numbers=False*)
Create one or two numbers

run_clearwater_live_test(*dns_ip*, *public_domain*, *bono_ip=None*, *ellis_ip=None*, *signup_code='secret'*)
Run the Clearwater live tests

It first runs dnsmasq to reach clearwater services by FQDN and then the Clearwater live tests. All results are saved in ims_test_output.txt.

Returns:

- a dict containing the overall results
- None on error

functest.opnfv_tests.vnf.ims.cloudify_ims module

CloudifyIms testcase implementation.

class functest.opnfv_tests.vnf.ims.cloudify_ims.**CloudifyIms**(***kwargs*)
Bases: *functest.core.cloudify.Cloudify*

Clearwater vIMS deployed with Cloudify Orchestrator Case.

check_requirements()
Check the requirements of the test case.

It can be overriden on purpose.

clean()
Clean created objects/functions.

cop_wgn = 'https://github.com/cloudify-cosmo/cloudify-openstack-plugin/releases/downlo

cop_yaml = 'https://github.com/cloudify-cosmo/cloudify-openstack-plugin/releases/downl

deploy_vnf()
Deploy Clearwater IMS.

execute()
Deploy Cloudify Manager.

network, security group, fip, VM creation

filename_alt = '/home/opnfv/functest/images/ubuntu-14.04-server-cloudimg-amd64-disk1.i

flavor_alt_disk = 3

flavor_alt_ram = 1024

flavor_alt_vcpus = 1

quota_port = 50

quota_security_group = 20

quota_security_group_rule = 100

test_vnf()
Run test on clearwater ims instance.

functest.opnfv_tests.vnf.ims.heat_ims module

HeatIms testcase implementation.

class functest.opnfv_tests.vnf.ims.heat_ims.**HeatIms**(***kwargs*)
Bases: *functest.core.singlevm.VmReady2*

Clearwater vIMS deployed with Heat Orchestrator Case.

clean()
Clean created objects/functions.

create_network_resources()
Create all tenant network resources

It creates a router which gateway is the external network detected. The new subnet is attached to that router.

Raises: expection on error

deploy_vnf()
Deploy Clearwater IMS.

execute()
Prepare Tenant/User

network, security group, fip, VM creation

filename = '/home/opnfv/functest/images/ubuntu-14.04-server-cloudimg-amd64-disk1.img'

flavor_disk = 3

flavor_ram = 1024

flavor_vcpus = 1

parameters = {'private_mgmt_net_cidr': '192.168.100.0/24', 'private_mgmt_net_gateway'

quota_port = 50

quota_security_group = 20

quota_security_group_rule = 100

run(***kwargs*)
Deploy and test clearwater

Here are the main actions: - deploy clearwater stack via heat - test the vnf instance

Returns: - TestCase.EX_OK - TestCase.EX_RUN_ERROR on error

test_vnf()
Run test on clearwater ims instance.

Module contents

functest.opnfv_tests.vnf.router package

Subpackages

functest.opnfv_tests.vnf.router.test_controller package

Submodules

functest.opnfv_tests.vnf.router.test_controller.function_test_exec module

vrouter function test execution module

class functest.opnfv_tests.vnf.router.test_controller.function_test_exec.**FunctionTestExec**(*
Bases: object

vrouter function test execution class

config_reference_vnf(*target_vnf*, *reference_vnf*, *test_kind*)

config_target_vnf(*target_vnf*, *reference_vnf*, *test_kind*)

logger = <logging.Logger object>

result_check(*target_vnf*, *reference_vnf*, *test_kind*, *test_list*)

run(*target_vnf*, *reference_vnf_list*, *test_info*, *test_list*)

Module contents

functest.opnfv_tests.vnf.router.vnf_controller package

Submodules

functest.opnfv_tests.vnf.router.vnf_controller.checker module

vrouter test result check module

class functest.opnfv_tests.vnf.router.vnf_controller.checker.**Checker**

Bases: object

vrouter test result check class

static load_check_rule(*rule_file_dir*, *rule_file_name*, *parameter*)

logger = <logging.Logger object>

static regexp_information(*response*, *rules*)

functest.opnfv_tests.vnf.router.vnf_controller.command_generator module

command generator module for vrouter testing

class functest.opnfv_tests.vnf.router.vnf_controller.command_generator.**CommandGenerator**

Bases: object

command generator class for vrouter testing

static command_create(*template*, *parameter*)

static load_template(*template_dir*, *template*)

logger = <logging.Logger object>

functest.opnfv_tests.vnf.router.vnf_controller.ssh_client module

ssh client module for vrouter testing

class functest.opnfv_tests.vnf.router.vnf_controller.ssh_client.**SshClient**(*ip_address*, *user*, *password=None*, *key_filename=None*)

Bases: object

ssh client class for vrouter testing

close()

connect(*time_out=10*, *retrycount=10*)

static error_check(*response*, *err_strs=None*)

logger = <logging.Logger object>

send(*cmd*, *prompt*, *timeout=10*)

functest.opnfv_tests.vnf.router.vnf_controller.vm_controller module

vm controll module

class functest.opnfv_tests.vnf.router.vnf_controller.vm_controller.**VmController**(*util_info*)
Bases: object

vm controll class

command_create_and_execute(*ssh*, *test_cmd_file_path*, *cmd_input_param*, *prompt_file_path*)

command_execute(*ssh*, *command*, *prompt*)

command_gen_from_template(*command_file_path*, *cmd_input_param*)

command_list_execute(*ssh*, *command_list*, *prompt*)

config_vm(*vm_info*, *test_cmd_file_path*, *cmd_input_param*, *prompt_file_path*)

connect_ssh_and_config_vm(*vm_info*, *test_cmd_file_path*, *cmd_input_param*, *prompt_file_path*)

logger = <logging.Logger object>

functest.opnfv_tests.vnf.router.vnf_controller.vnf_controller module

vrouter controll module

class functest.opnfv_tests.vnf.router.vnf_controller.vnf_controller.**VnfController**(*util_info*)
Bases: object

vrouter controll class

config_vnf(*source_vnf*, *destination_vnf*, *test_cmd_file_path*, *parameter_file_path*, *prompt_file_path*)

logger = <logging.Logger object>

output_check_result_detail_data(*res_data_list*)

result_check(*target_vnf*, *reference_vnf*, *check_rule_file_path_list*, *parameter_file_path*, *prompt_file_path*)

Module contents

Submodules

functest.opnfv_tests.vnf.router.cloudify_vrouter module

vrouter testcase implementation.

class functest.opnfv_tests.vnf.router.cloudify_vrouter.**CloudifyVrouter**(***kwargs*)
Bases: *functest.core.cloudify.Cloudify*

vrouter testcase deployed with Cloudify Orchestrator.

clean()
Clean the resources.

It can be overriden if resources must be deleted after running the test case.

cop_wgn = 'https://github.com/cloudify-cosmo/cloudify-openstack-plugin/releases/downlo

cop_yaml = 'https://github.com/cloudify-cosmo/cloudify-openstack-plugin/releases/downl

deploy_vnf()

execute()
Deploy Cloudify Manager. network, security group, fip, VM creation

filename_alt = '/home/opnfv/functest/images/vyos-1.1.8-amd64.qcow2'

flavor_alt_disk = 3

flavor_alt_ram = 1024

flavor_alt_vcpus = 1

test_vnf()

functest.opnfv_tests.vnf.router.utilvnf module

Utility module of vrouter testcase

class functest.opnfv_tests.vnf.router.utilvnf.**Utilvnf**
Bases: object

Utility class of vrouter testcase

static convert_functional_test_result(*result_data_list*)

get_address(*server_name*, *network_name*)

get_blueprint_outputs(*cfy_manager_ip*, *deployment_name*)

get_blueprint_outputs_networks(*cfy_manager_ip*, *deployment_name*)

get_blueprint_outputs_vnfs(*cfy_manager_ip*, *deployment_name*)

get_mac_address(*server_name*, *network_name*)

static get_reference_vnf_list(*vnf_info_list*)

static get_target_vnf(*vnf_info_list*)

static get_test_scenario(*file_path*)

static get_vnf_info(*vnf_info_list*, *vnf_name*)

get_vnf_info_list(*cfy_manager_ip*, *topology_deploy_name*, *target_vnf_name*)

logger = <logging.Logger object>

output_test_result_json()

request_vm_delete(*vnf_info_list*)

set_credentials(*cloud*)

write_result_data(*result_data*)

functest.opnfv_tests.vnf.router.vrouter_base module

vrouter testing base class module

class functest.opnfv_tests.vnf.router.vrouter_base.**VrouterOnBoardingBase**(*util*, *util_info*)

Bases: object

vrouter testing base class

function_test_vrouter(*target_vnf_name*, *test_info*)
function test execution

get_vnf_info_list(*target_vnf_name*)

test_vnf()
vrouter test execution

Module contents

Module contents

Module contents

functest.utils package

Submodules

functest.utils.config module

class functest.utils.config.**Config**
Bases: object

fill()

patch_file(*patch_file_path*)

functest.utils.constants module

functest.utils.env module

functest.utils.env.**get**(*env_var*)

functest.utils.env.**string**()

functest.utils.functest_utils module

functest.utils.functest_utils.**convert_dict_to_ini**(*value*)
Convert dict to oslo.conf input

functest.utils.functest_utils.**convert_ini_to_dict**(*value*)
Convert oslo.conf input to dict

functest.utils.functest_utils.**convert_ini_to_list**(*value*)
Convert list to oslo.conf input

functest.utils.functest_utils.**convert_list_to_ini**(*value*)
Convert list to oslo.conf input

functest.utils.functest_utils.**execute_command**(*cmd*, *info=False*, *error_msg=''*, *verbose=True*, *output_file=None*)

functest.utils.functest_utils.**execute_command_raise**(*cmd*, *info=False*, *error_msg=''*, *verbose=True*, *output_file=None*)

functest.utils.functest_utils.**get_nova_version**(*cloud*)
Get Nova API microversion

Returns:

- Nova API microversion
- None on operation error

functest.utils.functest_utils.**get_openstack_version**(*cloud*)
Detect OpenStack version via Nova API microversion

It follows MicroversionHistory.

Returns:

- OpenStack release
- Unknown on operation error

functest.utils.functest_utils.**get_parameter_from_yaml**(*parameter*, *yfile*)
Returns the value of a given parameter in file.yaml parameter must be given in string format with dots Example: general.openstack.image_name

Module contents

1.1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

D

E

F

G

Q

R

S

T

U

V

W