OPNFV Functest Documentation

Release master

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

May 05, 2021

Contents

Contents:

CHAPTER 1

functest

1.1 functest package

1.1.1 Subpackages

functest.core package

Submodules

functest.core.cloudify module

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).

`check_console_loop = 6`

`check_console_regex = ' login:  '`

`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=6*)

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.

clean_orphan_security_groups()

Clean all security groups which are not owned by an existing tenant

It lists all orphan security groups in use as debug to avoid misunderstanding the testcase results (it could happen if cloud admin removes accounts without cleaning the virtual machines)

count_active_hypervisors()

Count all hypervisors which are up.

count_hypervisors()

Count hypervisors.

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 subtestcases.

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 subtestcases.

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 subtestcases.

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

`get_environ`()

Get new environ

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

func_list = ['get_network_extensions', 'list_aggregates', 'list_domains', 'list_endpoi

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.

apply_blacklist(*case_file_name*, *result_file_name*)

Apply blacklist.

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

clean()

Cleanup of OpenStack resources. Should be called on completion.

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

Clean Rally config

static clean_rally_logs(*rally_conf='/etc/rally/rally.conf'*)

Clean Rally config

concurrency = 4

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(*tag*)

Get task id from command rally result.

Parameters tag –

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.

iterations_amount = 10

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

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

rally_aar4_patch_path = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api

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

run(***kwargs*)
Run testcase.

run_task(*test_name*)
Run a task.

run_tests(***kwargs*)
Execute tests.

shared_network = True

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

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

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

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

Parameters **json_raw** –

Returns Bool

task_timeout = 3600

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

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

tenants_amount = 3

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

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

users_amount = 2

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'

volume_service_type = 'volumev3'

volume_version = 3

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

Rally full testcase implementation.

task_timeout = 7200

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

Rally OpenStack CI testcase implementation.

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

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

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

stests = ['neutron']

task_timeout = 7200

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_console_loop = 12`

`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-1.3.0+stretch.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

`quota_cores = -1`

`quota_instances = -1`

`shaker_timeout = '3600'`

`ssh_connect_loops = 12`

`username = 'debian'`

Module contents

functest.opnfv_tests.openstack.tempest package

Submodules

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(*black_list*)

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.

static configure_tempest_update_params(*tempest_conf_file*, *image_id=None*, *flavor_id=None*, *compute_cnt=1*, *image_alt_id=None*, *flavor_alt_id=None*, *admin_role_name='admin'*, *cidr='192.168.120.0/24'*, *domain_id='default'*)

Add/update needed parameters into tempest.conf file

static configure_verifier(*deployment_dir*)

Execute rally verify configure-verifier, which generates tempest.conf

static create_verifier()

Create new verifier

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_deployment_dir(*verifier_id*, *deployment_id*)
Returns Rally deployment directory for current verifier

static get_verifier_id()
Returns verifier id for current Tempest

static get_verifier_repo_dir(*verifier_id*)
Returns installed verifier repo directory for Tempest

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

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

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

tempest_custom = '/home/docs/checkouts/readthedocs.org/user_builds/functest-api/envs/st

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

update_auth_section()
Update auth section in tempest.conf

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

update_scenario_section()
Update scenario section in tempest.conf

static update_tempest_conf_file(*conf_file*, *rconfig*)
Update defined paramters into tempest config file

update_validation_section()
Update validation section in tempest.conf

visibility = 'public'

class functest.opnfv_tests.openstack.tempest.tempest.**TempestHeat**(***kwargs*)

Bases: *functest.opnfv_tests.openstack.tempest.tempest.TempestCommon*

Tempest Heat testcase implementation class.

clean()

Cleanup all OpenStack objects. Should be called on completion.

configure(***kwargs*)

Create all openstack resources for tempest-based testcases and write tempest.conf.

filename_alt = '/home/opnfv/functest/images/Fedora-Cloud-Base-30-1.2.x86_64.qcow2'

flavor_alt_disk = 4

flavor_alt_ram = 512

flavor_alt_vcpus = 1

class functest.opnfv_tests.openstack.tempest.tempest.**TempestHorizon**(***kwargs*)

Bases: *functest.opnfv_tests.openstack.tempest.tempest.TempestCommon*

Tempest Horizon testcase implementation class.

configure(***kwargs*)

Create all openstack resources for tempest-based testcases and write tempest.conf.

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

ssh_retry_timeout = 240

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.SingleVm2*

Abot EPC deployed with JUJU Orchestrator Case

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

cidr = '192.168.120.0/24'

clean ()
Clean created objects/functions.

deploy_orchestrator ()
Create network, subnet, router

Bootstrap juju

deploy_vnf ()
Deploy ABOT-OAI-EPC.

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

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'

publish_image_alt (*name=None*)
Publish alternative image

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

Returns: image

Raises: expection on error

test_vnf ()
Run test on ABoT.

username = 'ubuntu'

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*, *bono_ip*, *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(*public_domain*, *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

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**(*u*

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 = <Logger functest.opnfv_tests.vnf.router.test_controller.function_test_exec (W

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 = <Logger functest.opnfv_tests.vnf.router.vnf_controller.checker (WARNING)>

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 = <Logger functest.opnfv_tests.vnf.router.vnf_controller.command_generator (WAR

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 = <Logger functest.opnfv_tests.vnf.router.vnf_controller.ssh_client (WARNING)>

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 = <Logger functest.opnfv_tests.vnf.router.vnf_controller.vm_controller (WARNING

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 = <Logger functest.opnfv_tests.vnf.router.vnf_controller.vnf_controller (WARNIN

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

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 = <Logger functest.opnfv_tests.vnf.router.utilvnf (WARNING)>

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

`functest.utils.functest_utils.`**`list_services`**`(`*cloud*`)`

Search Keystone services via $OS_INTERFACE.

It mainly conforms with Shade but allows testing vs public endpoints. It’s worth mentioning that it doesn’t support keystone v2.

Returns a list of `munch.Munch` containing the services description

Raises `OpenStackCloudException` if something goes wrong during the openstack API call.

`functest.utils.functest_utils.`**`search_services`**(*cloud*, *name_or_id=None*, *filters=None*)

Search Keystone services ia $OS_INTERFACE.

It mainly conforms with Shade but allows testing vs public endpoints. It's worth mentioning that it doesn't support keystone v2.

Parameters

- **`name_or_id`** – Name or id of the desired service.
- **`filters`** – a dict containing additional filters to use. e.g. {'type': 'network'}.

Returns a list of `munch.Munch` containing the services description

Raises `OpenStackCloudException` if something goes wrong during the openstack API call.

Module contents

1.1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index