dovetail

Release Latest

Open Platform for NFV

Feb 14, 2020

CONTENTS

CHAPTER ONE

OPNFV VERIFICATION PROGRAM APPLICATION FORM

Field	Description
Organization name	*Organization name*
Organization website	*Organization website if it is public*
Product name and/or identifier	*Product name and/or identifier*
Product specifications	*A link of product specifications*
Product public documentation	*A link of product public documentation*
Product categories	*Choose one: (i) software and hardware (ii) software and third party hardware*
Primary contact name	*Name*
Primary business email	*Only the Business email address should be used for official communication with OPNFV OVP*
Primary postal address	*Address*
Primary phone number	*Phone Number*
User ID for OVP web portal	*Choose one: (i) Linux Foundation (ii) Openstack (iii) Github (iv) Google (v) Facebook ID* *User ID:*
Location	*Choose one: (i)internal vendor lab (ii) third-party lab* *Name and address:*
Information of the 3rd-party lab	*If the test is to be conducted by a third-party lab, including name, email, address and phone numbe*

Continued on next pag

Table 1 – continued from previous page

Field	Description

CHAPTER

TWO

OVP WORKFLOW

2.1 Introduction

This document provides guidance for prospective participants on how to obtain "OPNFV Verified" for products and services. The OPNFV Verification Program (OVP) is administered by the LF Networking Compliance and Verification (C&V) Committee.

For further information about the workflow and general inquiries about the program, please check out the OVP landing page, or contact the C&V committee by email address ovp-support@lfnetworking.org. This email address should be used for all communication with the OVP.

2.2 Step 1: Participation Form Submission

A participant should start the process by submitting an online participation form. The participation form can be found on the OVP landing page. Submitters are asked for the following information:

- Organization name
- Contact name
- Contact address
- Contact telephone
- Contact email
- Membership status in LF Networking
- Qualifying Offerings Information
- Testing Category
- Product Category
- User ID
- Version of Testing Tools
- Testing Date
- Testing Location
- Third-party lab information (if any)
- Signature

Once the participation form information is received and in order, an email response will be sent to the primary contact with confirmation and information to proceed. The primary contact specified in the participation form will be entered into the corresponding OVP web portal back-end by the program administrator and will be permitted to submit results for review on behalf of their organization. The OVP NFVI Portal can be found here (https://nfvi-verified.lfnetworking.org) and the OVP VNF Portal can be found here (https://vnf-verified.lfnetworking.org).

There is no fee at this time for participation in the OVP for LF Networking (LFN) members or non-profit organizations. If not a member of LFN, please contact LF Networking for participation fee information.

2.3 Step 2: Testing

2.3.1 NFVI Testing

The following documents guide testers to prepare the NFVI test environment and run tests:

- *OVP NFVI System Preparation Guide*
- *OVP Test Specifications*
- *OVP Testing User Guide*

A unique Test ID is generated by the Dovetail tool for each test run and can only be submitted to the OVP web portal once.

2.3.2 VNF Testing

The following document guide testers to prepare the environment and run the VNF tests:

- Deploy ONAP via OOM
- ONAP VNF Test Specifications
- *Conducting ONAP VNF Testing for OVP*

2.4 Step 3: Submitting Test Results

Users/testers other than the primary contact may use the OVP web portal as a resource to upload, evaluate and share results in a private manner. Testers can upload the test results to the OVP web portal (either NFVI or VNF). By default, the results are visible only to the tester who uploaded the data.

Testers can self-review the test results through the portal until they are ready to ask for OVP review. They may also add new test results as needed.

Once the tester is satisfied with the test result, the primary contact grants access to the test result for OVP review using a 'submit for review' operation via the portal. During this step, a new window may be appeared to ask primary contact to complete the application form online. Then the test result is identified by the unique Test ID and becomes visible to a review group comprised of OPNFV community members.

When a test result is made visible to the reviewers, the program administrator will ask for volunteers from the review group using the ovp-support@lfnetworking.org email and CC the primary contact email that a review request has been made. The program administrator will supply the Test ID and owner field (primary contact user ID) to the reviewers to identify the results.

2.5 Step 4: OVP Review

Upon receiving the email request from the program administrator, the review group conducts a peer based review of the test result using reviewer guidelines published per OVP release. Persons employed by the same organization that submitted the test results or by affiliated organizations will not be part of the reviewers.

The primary contact may be asked via email for any missing information or clarification of the test results. The reviewers will make a determination and recommend compliance or non-compliance to the C&V Committee. A positive review requires a minimum of two approvals from two distinct organizations without any negative reviews. The program administrator sends an email to OVP/C&V emails announcing a positive review. A one week limit is given for issues to be raised. If no issue is raised, the C&V Committee approves the result and the program administrator sends an email to OVP/C&V emails stating the result is approved.

Normally, the outcome of the review should be communicated to the primary contact within 10 business days after all required information is in order.

If a test result is denied, an appeal can be made to the C&V Committee for arbitration.

2.6 Step 5: Grant of Use of Program Marks

If an application is approved, further information will be communicated to the primary contact that includes badges and on the guidelines of using OVP Program Marks for marketing and promotional purposes.

CHAPTER

THREE

OVP TESTING USER GUIDE PER INSTALLER

3.1 Conducting OVP Testing with Dovetail using APEX installer

3.1.1 Overview

The purpose of this document is to give tips for the dovetail deployment on APEX installer. The general structure of the document is remaining according to the user guide document and the APEX related tips will be added under of the respective chapter's name.

3.1.2 Installing Dovetail

Checking the Test Host Readiness

Installing Prerequisite Packages on the Test Host

Configuring the Test Host Environment

In order to run the test scenarios properly and having access to all OS components that each scenario needs, the undercloud credentials should be used and copied in the docker container along with ssh key.

The environment preparation should be applied on the Test Host environment. Therefore, the containers which are going to be used as part of this configuration, fetch the information, the files and the rest input from Test Host environment directly as part of the Docker command.

Setting up Primary Configuration File

Two new environment variables could be introduced in the `env_config.sh` file.

```
# Set the name of the installer type as environment variable (e.g. apex, fuel, etc)
# Optional parameter
export INSTALLER_TYPE=xxxx

# Set the deployed scenario name (e.g. os-sdn-nofeature-noha)
# Optional parameter
export DEPLOY_SCENARIO=xxxx
```

For the OS_PASSWORD, OpenStack password from undercloud environment should be used.

Configuration for Running Tempest Test Cases (Mandatory)

Configuration for Running HA Test Cases (Mandatory)

Below is a sample of `${DOVETAIL_HOME}/pre_config/pod.yaml` file with the required syntax when key_filename is used instead of password is employed by the controller. Moreover, the 'heat-admin' should be used as user.

```
nodes:
-
    # This can not be changed and must be node0.
    name: node0

    # This must be Jumpserver.
    role: Jumpserver

    # This is the instance IP of a node which has ipmitool installed.
    ip: xx.xx.xx.xx

    # User name of the user of this node. This user **must** have sudo privileges.
    user: heat-admin

    # Password of the user.
    #password: root
    key_filename: /root/.ssh/id_rsa

-
    # This can not be changed and must be node1.
    name: node1

    # This must be controller.
    role: Controller

    # This is the instance IP of a controller node, which is the haproxy primary node
    ip: xx.xx.xx.xx

    # User name of the user of this node. This user **must** have sudo privileges.
    user: heat-admin

    # Password of the user.
    #password: root
    key_filename: /root/.ssh/id_rsa

process_info:
-
    # The default attack process of yardstick.ha.rabbitmq is 'rabbitmq-server'.
    # Here can be reset to 'rabbitmq'.
    testcase_name: yardstick.ha.rabbitmq
    attack_process: rabbitmq

-
    # The default attack host for all HA test cases is 'node1'.
    # Here can be reset to any other node given in the section 'nodes'.
    testcase_name: yardstick.ha.glance_api
    attack_host: node2
```

Configuration of Hosts File (Optional)

Installing Dovetail on the Test Host

Online Test Host

Offline Test Host

3.1.3 Starting Dovetail Docker

3.1.4 Running the OVP Test Suite

Making Sense of OVP Test Results

3.1.5 OVP Portal Web Interface

3.1.6 Updating Dovetail or a Test Suite

CHAPTER

FOUR

DISABLING STRICT API VALIDATION IN TEMPEST

4.1 Introduction

In 2015, the OpenStack QA team introduced a validation mechanism for Nova API responses in Tempest[1] with the goal of enforcing Nova API micro-versions. The API validation mechanism verifies that API responses only contain data elements (properties) as explicitly defined in API response schemas[2]. In case additional data elements are found in Nova API responses, the corresponding tests fail immediately without asserting whether or not the particular API operation actually succeeded or not.

Independently, cloud vendors have extended their commercial OpenStack cloud implementations with additional functionality which requires API extensions. Consequently, such cloud implementations do not pass Tempest tests which validate API responses despite actually implementing and providing the tested functionality.

This document describes an exemption process for use within the OPNFV Verification Program which

i) allows vendors to pass Tempest tests if the tested functionality is fully supported despite the presence of additional data elements in API responses, and

ii) makes the application of the exemption process transparently visible in test results.

4.2 Background and benefits for OVP

Vendors of commercial NFV products have extended OpenStack to provide additional (NFV) functionality to their customers and to fill functional gaps in OpenStack. These add-ons potentially extend the OpenStack API in two ways:

i) new API endpoints and

ii) additional attributes returned by existing API endpoints.

New API endpoints typically go unnoticed by OpenStack Tempest tests and hence do not interfere with existing tests. In contrast, (Nova) Tempest tests actively validate the responses returned by existing API endpoints against pre-defined schemas. An API response is considered invalid if additional attributes are present (see example below). Hence, this particular type of functional extension of OpenStack causes existing Tempest tests to fail, irrespective of whether or not the functionality which is supposed to be tested is actually available. As a result, a Tempest test failing due to extended API responses does not provide information about whether the tested functionality is available or not.

The OPNFV Verification Program has inherited the policy to strictly validate API responses from OpenStack by including a selection of Tempest tests in its compliance test suite. However, it was never discussed if OVP should adopt this policy as well. It turns out that this policy causes challenges for vendors of commercial NFV offerings to pass the OVP test suite. The exemption process outlined in this document aims at allowing to selectively disable strict API response validation in order to enable vendors to adopt OVP **if** the tested functionality is supported.

[1] https://review.opendev.org/gitweb?p=openstack%2Ftempest.git;a=commitdiff;h=f0c30bc241e5160e3fe7402e738ea8f56a8b1315

[2] https://github.com/openstack/tempest/tree/master/tempest/lib/api_schema/response/compute

It must be clearly understood that this exemption targets **only** the scenario in which additional attributes are included in API responses. It does not provide a loophole for passing OVP tests if the OpenStack APIs have been altered significantly as this is in conflict with the objective of OVP to create industry alignment.

In conclusion, the exemption process described here is deemed beneficial for both the broader industry as well as for OVP: Enabling adoption of OVP by vendors which extended OpenStack API responses facilitates adoption of OVP in the industry. The limited validity period of an exemption incentivizes eventual alignment within the industry around a clearly specified set of APIs.

4.2.1 Example: additional attributes per VM for HA policy

This fictional example showcases the presence of an additional attribute in an API response. The example shows that the 'server details'[3], i.e. the VM metadata, includes an additional attribute 'ha-policy' which is used to associate high-availability policies with a VM instance. This attribute is utilized by a proprietary add-on component to manage VM migration and recovery in case of compute host failures:

```
{
  "server": {
    "accessIPv4": "1.2.3.4",
    "config_drive": "",
    "flavor": {...},
    "image": {...},
    "ha_policy": "migrate"        <-- additional attribute
    "name": "new-server-test",
    "status": "ACTIVE"
  }
}
```

4.3 Precedent in OpenStack

In the OpenStack community, the OpenStack Interoperability Working Group (Interop WG)[4] is maintaining multiple API interoperability compliance programs[5]. These programs utilize Tempest-based tests to determine if a given commercial cloud is compliant to a selected set of OpenStack APIs. After introduction of the strict API response validation, various cloud products which previously passed the compliance program failed validation because of the reasons outlined above.

In order to mitigate this situation, the Interop WG consulted with the broader OpenStack community[6] and eventually introduced an "additional properties waiver" for its API compliance programs in July 2016. The waiver was created with a clearly defined validity period, covering roughly one year - equivalent to three iterations of interoperability guidelines (2015.07, 2016.01, and 2016.08). The limited lifetime of the waiver was intended to give cloud product vendors a transition period for adapting their products to achieve full API compliance by the end of the exemption period. All details of the waiver are listed in[7]. Finally, the waiver was officially canceled in October 2017[8] after about 15 months.

[3] https://docs.openstack.org/api-ref/compute/#show-server-details

[4] https://wiki.openstack.org/wiki/Governance/InteropWG

[5] https://www.openstack.org/brand/interop/

[6] http://lists.openstack.org/pipermail/openstack-dev/2016-June/097349.html

[7] https://review.opendev.org/gitweb?p=openstack%2Finterop.git;a=commitdiff;h=c38e18b343505f16a74a97b748362fa7f1a01e57

[8] https://review.opendev.org/gitweb?p=openstack%2Finterop.git;a=commitdiff;h=5748c296a658cf5efebc16ad9d7644ca1125b073

4.4 Exemption process for additional properties in API responses in the OVP

The details of the exemption process for disabling strict validation of API responses is as follows:

1. The Dovetail tool provides a new command line option "–no-api-validation" for disabling strict API validation. This option needs to be explicitly given on the command line to disable strict API validation. If this command line option is omitted, the default behavior (i.e., strict API validation) is applied.
2. The test results created by the Dovetail tool includes an explicit print-out stating if strict API validation was disabled during the test run or not.
3. The OVP portal reads the uploaded result files and indicates for all uploaded test results if strict API validation was disabled or not.
4. Together with the application for certification, a company can request an exemption from the requirement for strict API response checking. A rationale for requesting the exemption has to be provided. The request is either granted or rejected by the C&C committee. The rationale provided must establish that the need for exemption is not in violation of the OVP's objectives.
5. Compliance badges obtained under exemption are valid for one year.
6. OPNFV expects OVP participants to aim for full compliance without requiring exemptions as soon as possible. Hence, an exemption can only be requested twice for the same product (addressing new versions of OVP or new versions of the product).
7. The same logo will be used regardless of being obtained under exemption or not.
8. The exemption will be made available to participants of OVP as part of a service release of OVP 2018.01, 2018.09 and 2019.12.
9. The C&C committee will monitor the situation around exemptions and may decide changes to the above process at any time, including the possibility to stop issuing exemptions.

CHAPTER

FIVE

GUIDELINES ADDENDUM FOR 2019.12 RELEASE

5.1 Introduction

This addendum provides a high-level description of the testing scope and pass/fail criteria used in the OPNFV Verification Program (OVP) for the 2019.12 release. This information is intended as an overview for OVP testers and for the Dovetail Project to help guide test-tool and test-case development for the OVP 2019.12 release. The Dovetail project is responsible for documenting test-case specifications as well as implementing the OVP tool-chain through collaboration with the OPNFV and ONAP testing communities. OVP testing focuses on establishing the ability of the System Under Test (SUT) to perform NFVI and VIM operations and support Service Provider oriented features that ensure manageable, resilient and secure networks.

5.2 Meaning of Compliance

OPNFV Compliance indicates adherence of an NFV platform and VNF to behaviors defined through specific platform capabilities, allowing to prepare, instantiate, operate and remove VNFs running on the NFVI. OVP 2019.12 compliance evaluates the ability of a platform to support Service Provider network capabilities and workloads that are supported in the OPNFV and ONAP platforms as of this release. Test cases are designated as compulsory or optional based on the maturity of capabilities as well as industry expectations. Compulsory test cases may for example include NFVI management capabilities whereas tests for certain high-availability features may be deemed as optional.

Test coverage and pass/fail criteria are designed to ensure an acceptable level of compliance but not be so restrictive as to disqualify variations in platform implementations, capabilities and features.

5.3 SUT Assumptions

Assumptions about the NFVI System Under Test (SUT) for the OVP Infrastructure badge include . . .

- The minimal specification of physical infrastructure, including controller nodes, compute nodes and networks, is defined for the NFVI by the Pharos specification.
- The SUT is fully deployed and operational, i.e. SUT deployment tools are out of scope of testing.

Assumptions about the VNF System Under Test (SUT) for the OVP VNF badge include . . .

- The VNF templates and disk image(s) file are available, and the disk image(s) have been deployed to the ONAP Cloud Site.
- The required value for the VNF pre-load files are available for the selected ONAP Cloud Site.

5.4 Scope of Testing

The OVP Governance Guidelines, as approved by the Board of Directors, outlines the key objectives of the OVP as follows:

- Help build the market for
  - LFN based infrastructure
  - applications designed to run on that infrastructure
- Reduce adoption risks for end-users
- Decrease testing costs by verifying hardware and software platform interfaces and components
- Enhance interoperability

The guidelines further directs the scope to be constrained to “features, capabilities, components, and interfaces included in an OPNFV and ONAP releases that are generally available in the industry (e.g., through adoption by an upstream community)”, and that compliance verification is evaluated using “functional tests that focus on defined interfaces and/or behaviors without regard to the implementation of the underlying system under test”.

OPNFV provides a broad range of capabilities, including the reference platform itself as well as tools-chains and methodologies for building infrastructures, and deploying and testing the platform. Not all these aspects are in scope for OVP and not all functions and components are tested in the initial versions of OVP. For example, the deployment tools for the SUT and CI/CD toolchain are currently out of scope. Similarly, performance benchmarking related testing is also out of scope or for further study. Newer functional areas such as MANO (outside of APIs in the NFVI and VIM) are still developing and are for future considerations.

ONAP provides a comprehensive platform for real-time, policy-driven orchestration and automation of physical and virtual network functions that will enable software, network, IT and cloud providers and developers to rapidly automate new services and support complete lifecycle management. By unifying member resources, ONAP is accelerating the development of a vibrant ecosystem around a globally shared architecture and implementation for network automation–with an open standards focus– faster than any one product could on its own.

5.4.1 General Approach

In order to meet the above objectives for OVP, we aim to follow a general approach by first identifying the overall requirements for all stake-holders, then analyzing what OPNFV and the upstream communities can effectively test and verify presently to derive an initial working scope for OVP, and to recommend what the community should strive to achieve in future releases.

The overall requirements for OVP can be categorized by the basic cloud capabilities representing common operations needed by basic VNFs, and additional requirements for VNFs that go beyond the common cloud capabilities including functional extensions, operational capabilities and additional carrier grade requirements.

For the basic NFV requirements, we will analyze the required test cases, leverage or improve upon existing test cases in OPNFV projects and upstream projects whenever we can, and bridge the gaps when we must, to meet these basic requirements.

We are not yet ready to include compliance requirements for capabilities such as hardware portability, carrier grade performance, fault management and other operational features, security, MANO and VNF verification. These areas are being studied for consideration in future OVP releases.

In some areas, we will start with a limited level of verification initially, constrained by what community resources are able to support at this time, but still serve a basic need that is not being fulfilled elsewhere. In these areas, we bring significant value to the community we serve by starting a new area of verification, breaking new ground and expanding it in the future.

In other areas, the functions being verified have yet to reach wide adoption but are seen as important requirements in NFV, or features are only needed for specific NFV use cases but an industry consensus about the APIs and behaviors is still deemed beneficial. In such cases, we plan to incorporate the test areas as optional. An optional test area will not have to be run or passed in order to achieve compliance. Optional tests provide an opportunity for vendors to demonstrate compliance with specific OPNFV features beyond the mandatory test scope.

5.4.2 Analysis of Scope

In order to define the scope of the 2019.12 release of the compliance and verification program, this section analyzes NFV-focused platform capabilities with respect to the high-level objectives and the general approach outlined in the previous section. The analysis determines which capabilities are suitable for inclusion in this release of the OVP and which capabilities are to be addressed in future releases.

1. Basic Cloud Capabilities

The intent of these tests is to verify that the SUT has the required capabilities that a basic VNF needs, and these capabilities are implemented in a way that enables this basic VNF to run on any OPNFV compliant deployment.

A basic VNF can be thought of as a single virtual machine that is networked and can perform the simplest network functions, for example, a simple forwarding gateway, or a set of such virtual machines connected only by simple virtual network services. Running such basic VNF leads to a set of common requirements, including:

- image management (testing Glance API)
- identity management (testing Keystone Identity API)
- virtual compute (testing Nova Compute API)
- virtual storage (testing Cinder API)
- virtual networks (testing Neutron Network API)
- forwarding packets through virtual networks in data path
- filtering packets based on security rules and port security in data path
- dynamic network runtime operations through the life of a VNF (e.g. attach/detach, enable/disable, read stats)
- correct behavior after common virtual machine life cycles events (e.g. suspend/resume, reboot, migrate)
- simple virtual machine resource scheduling on multiple nodes

OPNFV mainly supports OpenStack as the VIM up to the 2019.12 release. The VNFs used in the OVP NFVI program, and features in scope for the program which are considered to be basic to all VNFs, require commercial OpenStack distributions to support a common basic level of cloud capabilities, and to be compliant to a common specification for these capabilities. This requirement significantly overlaps with OpenStack community's Interop working group's goals, but they are not identical. The OVP runs the OpenStack Refstack-Compute test cases to verify compliance to the basic common API requirements of cloud management functions and VNF (as a VM) management for OPNFV. Additional NFV specific requirements are added in network data path validation, packet filtering by security group rules and port security, life cycle runtime events of virtual networks, multiple networks in a topology, validation of VNF's functional state after common life-cycle events including reboot, pause, suspense, stop/start and cold migration. In addition, the basic requirement also verifies that the SUT can allocate VNF resources based on simple anti-affinity rules.

The combined test cases help to ensure that these basic operations are always supported by a compliant platform and they adhere to a common standard to enable portability across OPNFV compliant platforms.

2. NFV specific functional requirements

NFV has functional requirements beyond the basic common cloud capabilities, esp. in the networking area. Examples like BGPVPN, IPv6, SFC may be considered additional NFV requirements beyond general purpose cloud computing. These feature requirements expand beyond common OpenStack (or other VIM) requirements. OPNFV OVP will incorporate test cases to verify compliance in these areas as they become mature. Because these extensions may impose new API demands, maturity and industry adoption is a prerequisite for making them a mandatory requirement for OPNFV compliance. At the time of the 2019.12 release, we have promoted tests of the OpenStack IPv6 API from optional to mandatory while keeping BGPVPN as optional test area. Passing optional tests will not be required to pass OPNFV compliance verification.

BGPVPNs are relevant due to the wide adoption of MPLS/BGP based VPNs in wide area networks, which makes it necessary for data centers hosting VNFs to be able to seamlessly interconnect with such networks. SFC is also an important NFV requirement, however its implementation has not yet been accepted or adopted in the upstream at the time of the 2019.12 release.

3. High availability

High availability is a common carrier grade requirement. Availability of a platform involves many aspects of the SUT, for example hardware or lower layer system failures or system overloads, and is also highly dependent on configurations. The current OPNFV high availability verification focuses on OpenStack control service failures and resource overloads, and verifies service continuity when the system encounters such failures or resource overloads, and also verifies the system heals after a failure episode within a reasonable time window. These service HA capabilities are commonly adopted in the industry and should be a mandatory requirement.

The current test cases in HA cover the basic area of failure and resource overload conditions for a cloud platform's service availability, including all of the basic cloud capability services, and basic compute and storage loads, so it is a meaningful first step for OVP. We expect additional high availability scenarios be extended in future releases.

4. Stress Testing

Resiliency testing involves stressing the SUT and verifying its ability to absorb stress conditions and still provide an acceptable level of service. Resiliency is an important requirement for end-users.

The 2019.12 release of OVP includes a load test which spins up a number of VMs pairs in parallel to assert that the system under test can process the workload spike in a stable and deterministic fashion.

5. Security

Security is among the top priorities as a carrier grade requirement by the end-users. Some of the basic common functions, including virtual network isolation, security groups, port security and role based access control are already covered as part of the basic cloud capabilities that are verified in OVP. These test cases however do not yet cover the basic required security capabilities expected of an end-user deployment. It is an area that we should address in the near future, to define a common set of requirements and develop test cases for verifying those requirements.

The 2019.12 release includes new test cases which verify that the role-based access control (RBAC) functionality of the VIM is behaving as expected.

Another common requirement is security vulnerability scanning. While the OPNFV security project integrated tools for security vulnerability scanning, this has not been fully analyzed or exercised in 2019.12 release. This area needs further work to identify the required level of security for the purpose of OPNFV in order to be integrated into the OVP. End-user inputs on specific requirements in security is needed.

6. Service assurance

Service assurance (SA) is a broad area of concern for reliability of the NFVI/VIM and VNFs, and depends upon multiple subsystems of an NFV platform for essential information and control mechanisms. These subsystems include telemetry, fault management (e.g. alarms), performance management, audits, and control mechanisms such as security and configuration policies.

The current 2019.12 release implements some enabling capabilities in NFVI/VIM such as telemetry, policy, and fault management. However, the specification of expected system components, behavior and the test cases to verify them have not yet been adequately developed. We will therefore not be testing this area at this time but defer to future study.

7. Use case testing

Use-case test cases exercise multiple functional capabilities of a platform in order to realize a larger end-to-end scenario. Such end-to-end use cases do not necessarily add new API requirements to the SUT per se, but exercise aspects of the SUT's functional capabilities in more complex ways. For instance, they allow for verifying the complex interactions among multiple VNFs and between VNFs and the cloud platform in a more realistic fashion. End-users consider use-case-level testing as a significant tool in verifying OPNFV compliance because it validates design patterns and support for the types of NFVI features that users care about.

There are a lot of projects in OPNFV developing use cases and sample VNFs. The 2019.12 release of OVP features two such use-case tests, spawning and verifying a vIMS and a vEPC, correspondingly.

8. Additional NFVI capabilities

In addition to the capabilities analyzed above, there are further system aspects which are of importance for the OVP. These comprise operational and management aspects such as platform in-place upgrades and platform operational insights such as telemetry and logging. Further aspects include API backward compatibility / micro-versioning, workload migration, multi-site federation and interoperability with workload automation platforms, e.g. ONAP. Finally, efficiency aspects such as the hardware and energy footprint of the platform are worth considering in the OVP.

OPNFV is addressing these items on different levels of details in different projects. However, the contributions developed in these projects are not yet considered widely available in commercial systems in order to include them in the OVP. Hence, these aspects are left for inclusion in future releases of the OVP.

9. VNF Compliance

VNF Compliance verifies the VNF template files conform to the requirements documented in by ONAP VNFRQTS project.

10. VNF Validation

VNF Validation verifies the VNF is able to onbroad within ONAP and ONAP is able to perform basic orchestration operations with the VNF, including instantiating the VNF on the Cloud Site.

5.4.3 Scope of the 2019.12 release of the OVP

Summarizing the results of the analysis above, the scope of the 2019.12 release of OVP is as follows:

- Mandatory NFVI test scope:
  - functest.vping.userdata
  - functest.vping.ssh
  - functest.tempest.osinterop*
  - functest.tempest.compute
  - functest.tempest.identity_v3
  - functest.tempest.image
  - functest.tempest.network_api
  - functest.tempest.volume
  - functest.tempest.neutron_trunk_ports
  - functest.tempest.ipv6_api
  - functest.security.patrole
  - yardstick.ha.nova_api
  - yardstick.ha.neutron_server
  - yardstick.ha.keystone

  - yardstick.ha.glance_api
  - yardstick.ha.cinder_api
  - yardstick.ha.cpu_load
  - yardstick.ha.disk_load
  - yardstick.ha.haproxy
  - yardstick.ha.rabbitmq
  - yardstick.ha.database
  - bottlenecks.stress.ping
- Optional NFVI test scope:
  - functest.tempest.ipv6_scenario
  - functest.tempest.multi_node_scheduling
  - functest.tempest.network_security
  - functest.tempest.vm_lifecycle
  - functest.tempest.network_scenario
  - functest.tempest.bgpvpn
  - yardstick.ha.neutron_l3_agent
  - yardstick.ha.controller_restart
  - functest.vnf.vims
  - functest.vnf.vepc
- Mandatory VNF test scope:
  - Refer to ONAP VNF Test Case Descriptions

* The OPNFV OVP utilizes the same set of test cases as the OpenStack interoperability program *OpenStack Powered Compute*. Passing the OPNFV OVP does **not** imply that the SUT is certified according to the *OpenStack Powered Compute* program. *OpenStack Powered Compute* is a trademark of the OpenStack foundation and the corresponding certification label can only be awarded by the OpenStack foundation.

Note: The SUT is limited to NFVI and VIM functions. While testing MANO component capabilities is out of scope, certain APIs exposed towards MANO are used by the current OPNFV compliance testing suite. MANO and other operational elements may be part of the test infrastructure; for example used for workload deployment and provisioning.

5.4.4 Scope considerations for future OVP releases

Based on the previous analysis, the following items are outside the scope of the 2019.12 release of OVP but are being considered for inclusion in future releases:

- service assurance
- use case testing
- platform in-place upgrade
- API backward compatibility / micro-versioning
- workload migration
- multi-site federation

- service function chaining
- platform operational insights, e.g. telemetry, logging
- efficiency, e.g. hardware and energy footprint of the platform
- interoperability with workload automation platforms e.g. ONAP
- resilience
- security and vulnerability scanning
- performance measurements

5.5 Criteria for Awarding Compliance

This section provides guidance on compliance criteria for each test area. The criteria described here are high-level, detailed pass/fail metrics are documented in Dovetail test specifications.

1. All mandatory test cases must pass.

Exceptions to this rule may be legitimate, e.g. due to imperfect test tools or reasonable circumstances that we can not foresee. These exceptions must be documented and accepted by the reviewers.

2. Optional test cases are optional to run. Its test results, pass or fail, do not impact compliance.

Applicants who choose to run the optional test cases can include the results of the optional test cases to highlight the additional compliance.

5.6 Exemption from strict API response validation

Vendors of commercial NFVI products may have extended the Nova API to support proprietary add-on features. These additions can cause Nova Tempest API tests to fail due to unexpected data in API responses. In order to resolve this transparently in the context of OVP, a temporary exemption process has been created. More information on the exemption can be found in section *Disabling Strict API Validation in Tempest*.

CHAPTER

SIX

OVP REVIEWER GUIDE

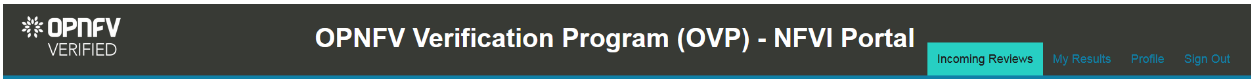
6.1 Introduction

This document provides detailed guidance for reviewers on how to handle the result review process.

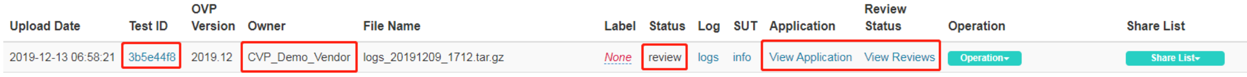
The OPNFV Verification Program (OVP) provides the ability for users to upload test results in OVP portal and request from OVP community to review them.

OVP administrator will ask for review volunteers using the ovp-support@lfnetworking.org email alias. The incoming results for review will be identified by the administrator with particular **Test ID** and **Owner** values.

Volunteers that will accept the review request can access the test results by login to the OVP portal and then click on the **Incoming Reviews** tab in top-level navigation bar.



After the user submit for review the test results **Status** is changed from 'private' to 'review'. Reviewers can find that the corresponding OVP portal result will have a status of 'review'. Also there are **Application** information list here for review. All the application information is submitted by users at the same time they submit their results for review. Reviewers can also find who has already approve/not approve the test results by clicking on the **View Reviews**.



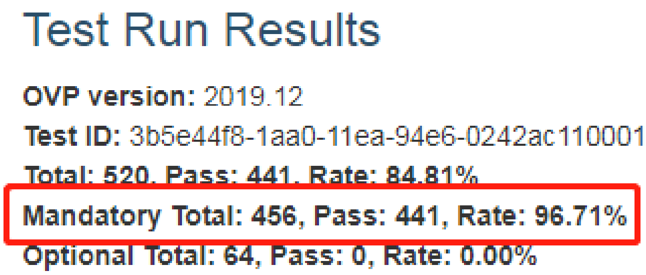
Upload Date	Test ID	OVP Version	Owner	File Name	Label	Status	Log	SUT	Application	Review Status	Operation	Share List
2019-12-13 06:58:21	3b5e44f8	2019.12	CVP_Demo_Vendor	logs_20191209_1712.tar.gz	None	review	logs	info	View Application	View Reviews	Operation▾	Share List▾

Reviewers must follow the checklist below to ensure review consistency for the OPNFV Verification Program (OVP) 2019.12 (Hunter) release at a minimum.

1. **Test Case Pass Percentage** - Ensure all mandatory tests have passed (100% pass rate).
2. **Mandatory Test Case Results** - Validate that results for all mandatory test cases are present.
3. **Log File Verification** - Inspect the log file for each test case.
4. **SUT Info Verification** - Validate the system under test (SUT) hardware and software endpoint info is present.

6.2 Test Case Pass Percentage

All mandatory test cases have to run successfully. The below figure of the **Test Run Results** is one method and shows that 96.71% of the mandatory test cases have passed. This value must not be lower than 100%.



Test Run Results

OVP version: 2019.12

Test ID: 3b5e44f8-1aa0-11ea-94e6-0242ac110001

Total: 520, Pass: 441, Rate: 84.81%

Mandatory Total: 456, Pass: 441, Rate: 96.71%

Optional Total: 64, Pass: 0, Rate: 0.00%

6.3 Mandatory Test Case Results

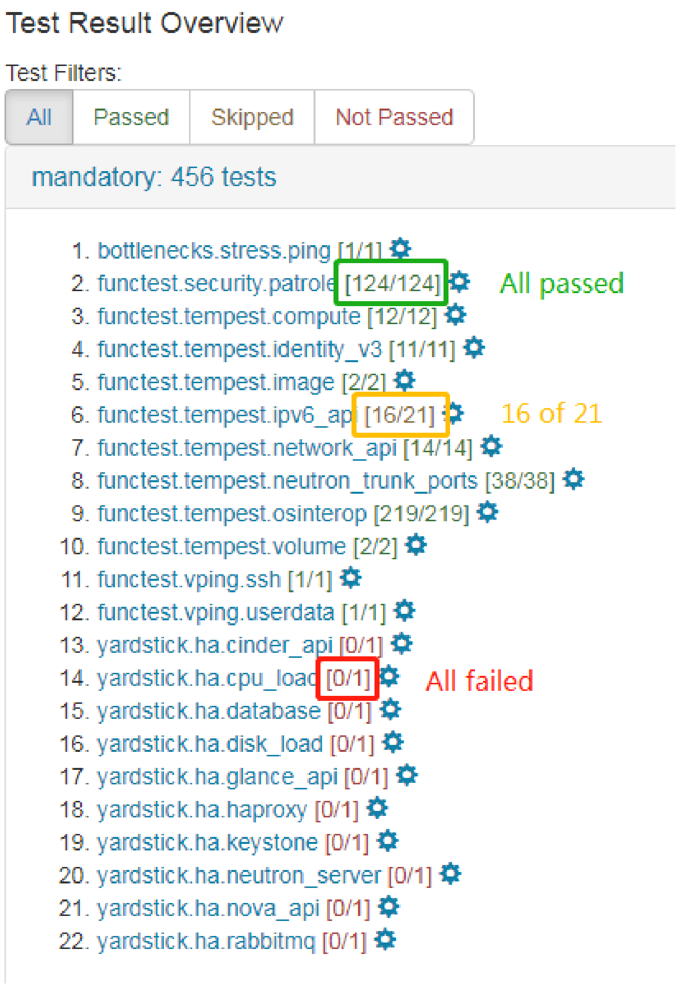
Test results can be displayed by clicking on the hyperlink under the **Test ID** column. Reviewers should validate that results for all mandatory test cases are included in the overall test suite. The required mandatory test cases are:

- bottlenecks.stress.ping
- functest.security.patrole
- functest.tempest.compute
- functest.tempest.identity_v3
- functest.tempest.image
- functest.tempest.ipv6_api
- functest.tempest.network_api
- functest.tempest.neutron_trunk_ports
- functest.tempest.osinterop
- functest.tempest.volume
- functest.vping.ssh
- functest.vping.userdata
- yardstick.ha.cinder_api
- yardstick.ha.cpu_load
- yardstick.ha.database
- yardstick.ha.disk_load
- yardstick.ha.glance_api
- yardstick.ha.haproxy
- yardstick.ha.keystone
- yardstick.ha.neutron_server
- yardstick.ha.nova_api
- yardstick.ha.rabbitmq

Note, that the 'Test ID' column in this view condenses the UUID used for 'Test ID' to eight characters even though the 'Test ID' is a longer UUID in the back-end.

Failed test cases can be easy identified by the color of pass/total number:

- **Green** when all test cases pass
- **Orange** when at least one fails/skips
- **Red** when all test cases fail/skip



mandatory: 456 tests

1. bottlenecks.stress.ping [1/1] ⚙
2. functest.security.patrole [124/124] ⚙
3. functest.tempest.compute [12/12] ⚙
4. functest.tempest.identity_v3 [11/11] ⚙
5. functest.tempest.image [2/2] ⚙
6. functest.tempest.ipv6_api [16/21] ⚙
7. functest.tempest.network_api [14/14] ⚙
8. functest.tempest.neutron_trunk_ports [38/38] ⚙
9. functest.tempest.osinterop [219/219] ⚙
10. functest.tempest.volume [2/2] ⚙
11. functest.vping.ssh [1/1] ⚙
12. functest.vping.userdata [1/1] ⚙
13. yardstick.ha.cinder_api [0/1] ⚙
14. yardstick.ha.cpu_load [0/1] ⚙
15. yardstick.ha.database [0/1] ⚙
16. yardstick.ha.disk_load [0/1] ⚙
17. yardstick.ha.glance_api [0/1] ⚙
18. yardstick.ha.haproxy [0/1] ⚙
19. yardstick.ha.keystone [0/1] ⚙
20. yardstick.ha.neutron_server [0/1] ⚙
21. yardstick.ha.nova_api [0/1] ⚙
22. yardstick.ha.rabbitmq [0/1] ⚙

All passed

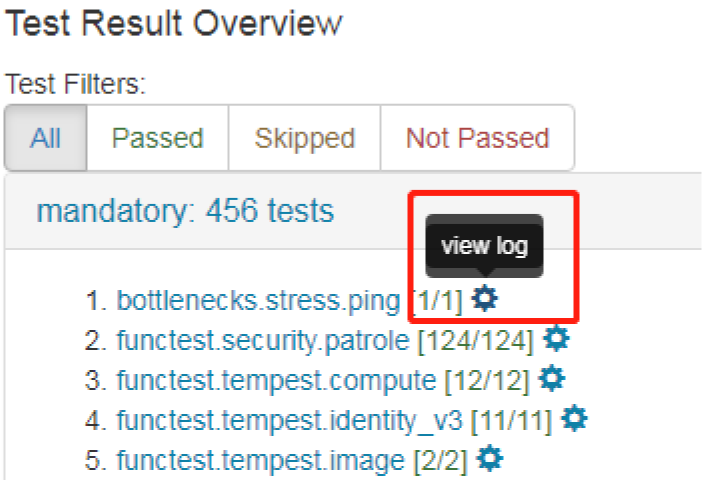
16 of 21

All failed

6.4 Log File Verification

Each log file of the mandatory test cases have to be verified for content.

Log files can be displayed by clicking on the setup icon to the right of the results, as shown in the figure below.



mandatory: 456 tests

1. bottlenecks.stress.ping [1/1]
2. functest.security.patrole [124/124]
3. functest.tempest.compute [12/12]
4. functest.tempest.identity_v3 [11/11]
5. functest.tempest.image [2/2]

Note, all log files can be found at results/ directory as shown at the following table.

Mandatory Test Case	Location
bottlenecks	results/stress_logs/
functest.vping	results/vping_logs/
functest.tempest	results/tempest_logs/
functest.security	results/security_logs/
yardstick	results/ha_logs/

6.4.1 Bottlenecks Logs

It must contain the 'SUCCESS' result at the end of Bottlenecks log as shown in following example:

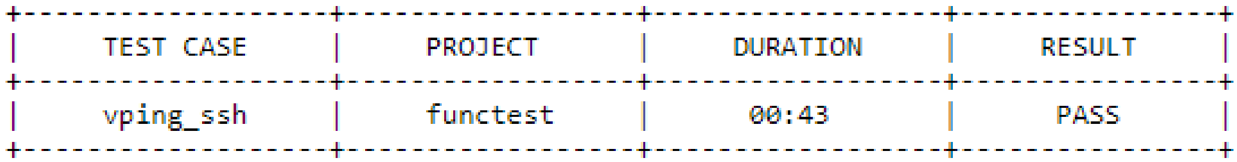
> 2019-12-03 07:35:14,630 [INFO] yardstick.benchmark.core.task task.py:129 Testcase: "ping_bottlenecks" SUCCESS!!!

6.4.2 Functest Logs

There are 2 different types of Functest logs, one is plain text for **vping** test cases and the other is html file for **tempest** and **security** test cases.

For **vping** test cases, two entries displayed in the tables below must be present in log files.

functest.vping.ssh

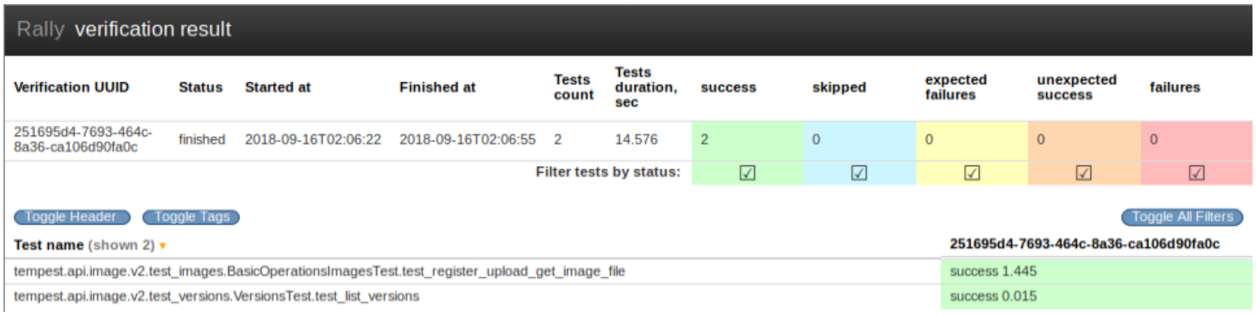


```
+-------------------+------------------+-------------------+-----------------+
|     TEST CASE     |     PROJECT      |     DURATION      |     RESULT      |
+-------------------+------------------+-------------------+-----------------+
|     vping_ssh     |     functest     |       00:43       |      PASS       |
+-------------------+------------------+-------------------+-----------------+
```

functest.vping.userdata

```
+--------------------------+------------------+------------------+----------------+
|        TEST CASE         |     PROJECT      |     DURATION     |     RESULT     |
+--------------------------+------------------+------------------+----------------+
|      vping_userdata      |    functest      |      01:40       |      PASS      |
+--------------------------+------------------+------------------+----------------+
```

For **tempest** and **security** test cases, it opens an html page that lists all test cases as shown below. All test cases must have run successfully.



Rally verification result

Verification UUID	Status	Started at	Finished at	Tests count	Tests duration, sec	success	skipped	expected failures	unexpected success	failures
251695d4-7693-464c-8a36-ca106d90fa0c	finished	2018-09-16T02:06:22	2018-09-16T02:06:55	2	14.576	2	0	0	0	0
					Filter tests by status:	☑	☑	☑	☑	☑

Test name (shown 2)	251695d4-7693-464c-8a36-ca106d90fa0c
tempest.api.image.v2.test_images.BasicOperationsImagesTest.test_register_upload_get_image_file	success 1.445
tempest.api.image.v2.test_versions.VersionsTest.test_list_versions	success 0.015

6.4.3 Yardstick Logs

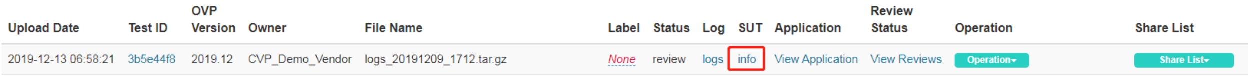
The yardstick log must contain the 'SUCCESS' result for each of the test-cases within this test area. This can be verified by searching the log for the keyword 'SUCCESS'.

An example of a FAILED and a SUCCESS test case are listed below:

> 2018-08-28 10:25:09,946 [ERROR] yardstick.benchmark.scenarios.availability.monitor.monitor_multi monitor_multi.py:78 SLA **failure**: 14.015082 > 5.000000
>
> 2018-08-28 10:23:41,907 [INFO] yardstick.benchmark.core.task task.py:127 Testcase: "opnfv_yardstick_tc052" **SUCCESS**!!!

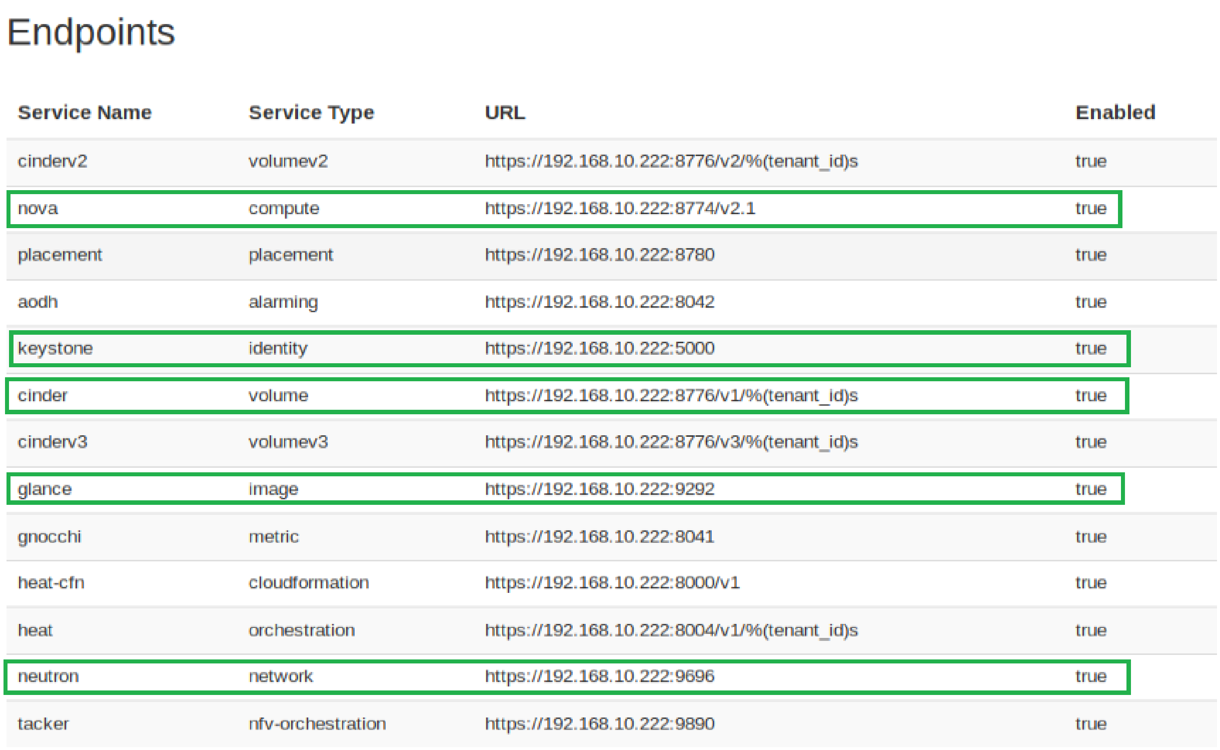
6.5 SUT Info Verification

SUT information must be present in the results to validate that all required endpoint services and at least two controllers were present during test execution. For the results shown below, click the **info** hyperlink in the **SUT** column to navigate to the SUT information page.



Upload Date	Test ID	OVP Version	Owner	File Name	Label	Status	Log	SUT	Application	Review Status	Operation	Share List
2019-12-13 06:58:21	3b5e44f8	2019.12	CVP_Demo_Vendor	logs_20191209_1712.tar.gz	None	review	logs	info	View Application	View Reviews	Operation▾	Share List▾

In the **Endpoints** listing shown below for the SUT VIM component, ensure that services are present for identify, compute, image, volume and network at a minimum by inspecting the **Service Type** column.



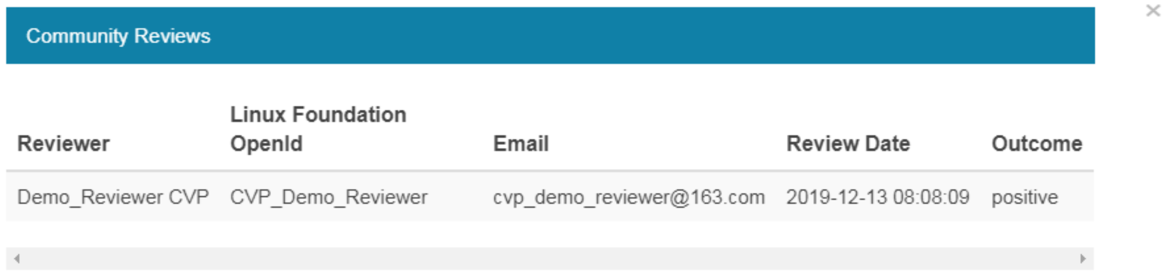
Endpoints

Service Name	Service Type	URL	Enabled
cinderv2	volumev2	https://192.168.10.222:8776/v2/%(tenant_id)s	true
nova	compute	https://192.168.10.222:8774/v2.1	true
placement	placement	https://192.168.10.222:8780	true
aodh	alarming	https://192.168.10.222:8042	true
keystone	identity	https://192.168.10.222:5000	true
cinder	volume	https://192.168.10.222:8776/v1/%(tenant_id)s	true
cinderv3	volumev3	https://192.168.10.222:8776/v3/%(tenant_id)s	true
glance	image	https://192.168.10.222:9292	true
gnocchi	metric	https://192.168.10.222:8041	true
heat-cfn	cloudformation	https://192.168.10.222:8000/v1	true
heat	orchestration	https://192.168.10.222:8004/v1/%(tenant_id)s	true
neutron	network	https://192.168.10.222:9696	true
tacker	nfv-orchestration	https://192.168.10.222:9890	true

Inspect the **Hosts** listing found below the Endpoints secion of the SUT info page and ensure at least two hosts are present, as two controllers are required the for the mandatory HA test cases.

6.6 Approve or Not Approve Results

When you decide to approve or not approve this test, you can click the **Operation** and choose **approve** or **not approve**. Once you have approved or not approved the test, you can click **View Reviews** to find the review status as shown below.



Community Reviews

Reviewer	Linux Foundation OpenId	Email	Review Date	Outcome
Demo_Reviewer CVP	CVP_Demo_Reviewer	cvp_demo_reviewer@163.com	2019-12-13 08:08:09	positive

CHAPTER

SEVEN

OVP NFVI SYSTEM PREPARATION GUIDE

This document provides a general guide to hardware system prerequisites and expectations for running OPNFV OVP testing. For detailed guide of preparing software tools and configurations, and conducting the test, please refer to the User Guide :ref:dovetail-testing_user_guide.

The OVP test tools expect that the hardware of the System Under Test (SUT) is Pharos compliant Pharos specification

The Pharos specification itself is a general guideline, rather than a set of specific hard requirements at this time, developed by the OPNFV community. For the purpose of helping OVP testers, we summarize the main aspects of hardware to consider in preparation for OVP testing.

As described by the OVP Testing User Guide, the hardware systems involved in OVP testing includes a Test Node, a System Under Test (SUT) system, and network connectivity between them.

The Test Node can be a bare metal machine or a virtual machine that can support Docker container environment. If it is a bare metal machine, it needs to be a x86 based at this time. Detailed information of how to configure and prepare the Test Node can be found in the User Guide.

The System Under Test (SUT) system is expected to consist of a set of general purpose servers, storage devices or systems, and networking infrastructure connecting them together. The set of servers are expected to be of the same architecture, either x86-64 or ARM-64. Mixing different architectures in the same SUT is not supported.

A minimum of 5 servers, 3 configured for controllers and 2 or more configured for compute resource are expected. However this is not a hard requirement at this phase. The OVP 1.0 mandatory test cases only require one compute server. At lease two compute servers are required to pass some of the optional test cases in the current OVP release. OVP control service high availability tests expect two or more control nodes to pass, depending on the HA mechanism implemented by the SUT.

The SUT is also expected to include components for persistent storage. The OVP testing does not expect or impose significant storage size or performance requirements.

The SUT is expected to be connected with high performance networks. These networks are expected in the SUT:

- A management network by which the Test Node can reach all identity, image, network, and compute services in the SUT
- A data network that supports the virtual network capabilities and data path testing

Additional networks, such as Light Out Management or storage networks, may be beneficial and found in the SUT, but they are not a requirement for OVP testing.

CHAPTER

EIGHT

OVP TEST SPECIFICATIONS

8.1 Introduction

The OPNFV OVP provides a series or test areas aimed to evaluate the operation of an NFV system in accordance with carrier networking needs. Each test area contains a number of associated test cases which are described in detail in the associated test specification.

All tests in the OVP are required to fulfill a specific set of criteria in order that the OVP is able to provide a fair assessment of the system under test. Test requirements are described in the :ref:dovetail-test_case_requirements document.

All tests areas addressed in the OVP are covered in the following test specification documents.

8.1.1 OpenStack Services HA test specification

Scope

The HA test area evaluates the ability of the System Under Test to support service continuity and recovery from component failures on part of OpenStack controller services(“nova-api”, “neutron-server”, “keystone”, “glance-api”, “cinder-api”) and on “load balancer” service.

The tests in this test area will emulate component failures by killing the processes of above target services, stressing the CPU load or blocking disk I/O on the selected controller node, and then check if the impacted services are still available and the killed processes are recovered on the selected controller node within a given time interval.

References

This test area references the following specifications:

- ETSI GS NFV-REL 001
  - https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_nfv-rel001v010101p.pdf
- OpenStack High Availability Guide
  - https://docs.openstack.org/ha-guide/

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- SUT - system under test
- Monitor - tools used to measure the service outage time and the process outage time

- Service outage time - the outage time (seconds) of the specific OpenStack service
- Process outage time - the outage time (seconds) from the specific processes being killed to recovered

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

SUT is assumed to be in high availability configuration, which typically means more than one controller nodes are in the System Under Test.

Test Area Structure

The HA test area is structured with the following test cases in a sequential manner.

Each test case is able to run independently. Preceding test case's failure will not affect the subsequent test cases.

Preconditions of each test case will be described in the following test descriptions.

Test Descriptions

Test Case 1 - Controller node OpenStack service down - nova-api

Short name

yardstick.ha.nova_api

Yardstick test case: opnfv_yardstick_tc019.yaml

Use case specification

This test case verifies the service continuity capability in the face of the software process failure. It kills the processes of OpenStack "nova-api" service on the selected controller node, then checks whether the "nova-api" service is still available during the failure, by creating a VM then deleting the VM, and checks whether the killed processes are recovered within a given time interval.

Test preconditions

There is more than one controller node, which is providing the "nova-api" service for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for verifying service continuity and recovery

The service continuity and process recovery capabilities of "nova-api" service is evaluated by monitoring service outage time, process outage time, and results of nova operations.

Service outage time is measured by continuously executing "openstack server list" command in loop and checking if the response of the command request is returned with no failure. When the response fails, the "nova-api" service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is measured by checking the status of “nova-api” processes on the selected controller node. The time of “nova-api” processes being killed to the time of the “nova-api” processes being recovered is the process outage time. Process recovery is verified by checking the existence of “nova-api” processes.

All nova operations are carried out correctly within a given time interval which suggests that the “nova-api” service is continuously available.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that “nova-api” processes are running on Node1
- Test action 2: Create a image with “openstack image create test-cirros –file cirros-0.3.5-x86_64-disk.img –disk-format qcow2 –container-format bare”
- Test action 3: Execute”openstack flavor create m1.test –id auto –ram 512 –disk 1 –vcpus 1” to create flavor “m1.test”.
- Test action 4: Start two monitors: one for “nova-api” processes and the other for “openstack server list” command. Each monitor will run as an independent process
- Test action 5: Connect to Node1 through SSH, and then kill the “nova-api” processes
- Test action 6: When “openstack server list” returns with no error, calculate the service outage time, and execute command “openstack server create –flavor m1.test –image test-cirros test-instance”
- Test action 7: Continuously Execute “openstack server show test-instance” to check if the status of VM “test-instance” is “Active”
- Test action 8: If VM “test-instance” is “Active”, execute “openstack server delete test-instance”, then execute “openstack server list” to check if the VM is not in the list
- Test action 9: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

The nova operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the process of “nova-api” if they are not running.

Delete image with “openstack image delete test-cirros”.

Delete flavor with “openstack flavor delete m1.test”.

Test Case 2 - Controller node OpenStack service down - neutron-server

Short name

yardstick.ha.neutron_server

Yardstick test case: opnfv_yardstick_tc045.yaml

Use case specification

This test verifies the high availability of the “neutron-server” service provided by OpenStack controller nodes. It kills the processes of OpenStack “neutron-server” service on the selected controller node, then checks whether the “neutron-server” service is still available, by creating a network and deleting the network, and checks whether the killed processes are recovered.

Test preconditions

There is more than one controller node, which is providing the “neutron-server” service for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of “neutron-server” service is evaluated by monitoring service outage time, process outage time, and results of neutron operations.

Service outage time is tested by continuously executing “openstack router list” command in loop and checking if the response of the command request is returned with no failure. When the response fails, the “neutron-server” service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of “neutron-server” processes on the selected controller node. The time of “neutron-server” processes being killed to the time of the “neutron-server” processes being recovered is the process outage time. Process recovery is verified by checking the existence of “neutron-server” processes.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that “neutron-server” processes are running on Node1
- Test action 2: Start two monitors: one for “neutron-server” process and the other for “openstack router list” command. Each monitor will run as an independent process.
- Test action 3: Connect to Node1 through SSH, and then kill the “neutron-server” processes
- Test action 4: When “openstack router list” returns with no error, calculate the service outage time, and execute “openstack network create test-network”
- Test action 5: Continuously executing “openstack network show test-network”, check if the status of “test-network” is “Active”
- Test action 6: If “test-network” is “Active”, execute “openstack network delete test-network”, then execute “openstack network list” to check if the “test-network” is not in the list

- Test action 7: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

The neutron operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of "neutron-server" if they are not running.

Test Case 3 - Controller node OpenStack service down - keystone

Short name

yardstick.ha.keystone

Yardstick test case: opnfv_yardstick_tc046.yaml

Use case specification

This test verifies the high availability of the "keystone" service provided by OpenStack controller nodes. It kills the processes of OpenStack "keystone" service on the selected controller node, then checks whether the "keystone" service is still available by executing command "openstack user list" and whether the killed processes are recovered.

Test preconditions

There is more than one controller node, which is providing the "keystone" service for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of "keystone" service is evaluated by monitoring service outage time and process outage time

Service outage time is tested by continuously executing "openstack user list" command in loop and checking if the response of the command request is reutrned with no failure. When the response fails, the "keystone" service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of "keystone" processes on the selected controller node. The time of "keystone" processes being killed to the time of the "keystone" processes being recovered is the process outage time. Process recovery is verified by checking the existence of "keystone" processes.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that "keystone" processes are running on Node1
- Test action 2: Start two monitors: one for "keystone" process and the other for "openstack user list" command. Each monitor will run as an independent process.
- Test action 3: Connect to Node1 through SSH, and then kill the "keystone" processes
- Test action 4: Calculate the service outage time and process outage time
- Test action 5: The test passes if process outage time is less than 20s and service outage time is less than 5s
- Test action 6: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of "keystone" if they are not running.

Test Case 4 - Controller node OpenStack service down - glance-api

Short name

yardstick.ha.glance_api

Yardstick test case: opnfv_yardstick_tc047.yaml

Use case specification

This test verifies the high availability of the "glance-api" service provided by OpenStack controller nodes. It kills the processes of OpenStack "glance-api" service on the selected controller node, then checks whether the "glance-api" service is still available, by creating image and deleting image, and checks whether the killed processes are recovered.

Test preconditions

There is more than one controller node, which is providing the "glance-api" service for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of “glance-api” service is evaluated by monitoring service outage time, process outage time, and results of glance operations.

Service outage time is tested by continuously executing “openstack image list” command in loop and checking if the response of the command request is returned with no failure. When the response fails, the “glance-api” service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of “glance-api” processes on the selected controller node. The time of “glance-api” processes being killed to the time of the “glance-api” processes being recovered is the process outage time. Process recovery is verified by checking the existence of “glance-api” processes.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that “glance-api” processes are running on Node1
- Test action 2: Start two monitors: one for “glance-api” process and the other for “openstack image list” command. Each monitor will run as an independent process.
- Test action 3: Connect to Node1 through SSH, and then kill the “glance-api” processes
- Test action 4: When “openstack image list” returns with no error, calculate the service outage time, and execute “openstack image create test-image –file cirros-0.3.5-x86_64-disk.img –disk-format qcow2 –container-format bare”
- Test action 5: Continuously execute “openstack image show test-image”, check if status of “test-image” is “active”
- Test action 6: If “test-image” is “active”, execute “openstack image delete test-image”. Then execute “openstack image list” to check if “test-image” is not in the list
- Test action 7: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

The glance operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of “glance-api” if they are not running.

Delete image with “openstack image delete test-image”.

Test Case 5 - Controller node OpenStack service down - cinder-api

Short name

yardstick.ha.cinder_api

Yardstick test case: opnfv_yardstick_tc048.yaml

Use case specification

This test verifies the high availability of the "cinder-api" service provided by OpenStack controller nodes. It kills the processes of OpenStack "cinder-api" service on the selected controller node, then checks whether the "cinder-api" service is still available by executing command "openstack volume list" and whether the killed processes are recovered.

Test preconditions

There is more than one controller node, which is providing the "cinder-api" service for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of "cinder-api" service is evaluated by monitoring service outage time and process outage time

Service outage time is tested by continuously executing "openstack volume list" command in loop and checking if the response of the command request is returned with no failure. When the response fails, the "cinder-api" service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of "cinder-api" processes on the selected controller node. The time of "cinder-api" processes being killed to the time of the "cinder-api" processes being recovered is the process outage time. Process recovery is verified by checking the existence of "cinder-api" processes.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that "cinder-api" processes are running on Node1
- Test action 2: Start two monitors: one for "cinder-api" process and the other for "openstack volume list" command. Each monitor will run as an independent process.
- Test action 3: Connect to Node1 through SSH, and then execute kill the "cinder-api" processes
- Test action 4: Continuously measure service outage time from the monitor until the service outage time is more than 5s
- Test action 5: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

The cinder operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of “cinder-api” if they are not running.

Test Case 6 - Controller Node CPU Overload High Availability

Short name

yardstick.ha.cpu_load

Yardstick test case: opnfv_yardstick_tc051.yaml

Use case specification

This test verifies the availability of services when one of the controller node suffers from heavy CPU overload. When the CPU usage of the specified controller node is up to 100%, which breaks down the OpenStack services on this node, the Openstack services should continue to be available. This test case stresses the CPU usage of a specific controller node to 100%, then checks whether all services provided by the SUT are still available with the monitor tools.

Test preconditions

There is more than one controller node, which is providing the “cinder-api”, “neutron-server”, “glance-api” and “keystone” services for API end-point.

Denoted a controller node as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of related OpenStack service is evaluated by monitoring service outage time

Service outage time is tested by continuously executing “openstack router list”, “openstack stack list”, “openstack volume list”, “openstack image list” commands in loop and checking if the response of the command request is returned with no failure. When the response fails, the related service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Methodology for stressing CPU usage

To evaluate the high availability of target OpenStack service under heavy CPU load, the test case will first get the number of logical CPU cores on the target controller node by shell command, then use the number to execute 'dd' command to continuously copy from /dev/zero and output to /dev/null in loop. The 'dd' operation only uses CPU, no I/O operation, which is ideal for stressing the CPU usage.

Since the 'dd' command is continuously executed and the CPU usage rate is stressed to 100%, the scheduler will schedule each 'dd' command to be processed on a different logical CPU core. Eventually to achieve all logical CPU cores usage rate to 100%.

Test execution

- Test action 1: Start four monitors: one for "openstack image list" command, one for "openstack router list" command, one for "openstack stack list" command and the last one for "openstack volume list" command. Each monitor will run as an independent process.
- Test action 2: Connect to Node1 through SSH, and then stress all logical CPU cores usage rate to 100%
- Test action 3: Continuously measure all the service outage times until they are more than 5s
- Test action 4: Kill the process that stresses the CPU usage

Pass / fail criteria

All the service outage times are less than 5s.

A negative result will be generated if the above is not met in completion.

Post conditions

No impact on the SUT.

Test Case 7 - Controller Node Disk I/O Overload High Availability

Short name

yardstick.ha.disk_load

Yardstick test case: opnfv_yardstick_tc052.yaml

Use case specification

This test verifies the high availability of control node. When the disk I/O of the specific disk is overload, which breaks down the OpenStack services on this node, the read and write services should continue to be available. This test case blocks the disk I/O of the specific controller node, then checks whether the services that need to read or write the disk of the controller node are available with some monitor tools.

Test preconditions

There is more than one controller node. Denoted a controller node as Node1 in the following configuration. The controller node has at least 20GB free disk space.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of nova service is evaluated by monitoring service outage time

Service availability is tested by continuously executing "openstack flavor list" command in loop and checking if the response of the command request is returned with no failure. When the response fails, the related service is considered in outage.

Methodology for stressing disk I/O

To evaluate the high availability of target OpenStack service under heavy I/O load, the test case will execute shell command on the selected controller node to continuously writing 8kb blocks to /test.dbf

Test execution

- Test action 1: Connect to Node1 through SSH, and then stress disk I/O by continuously writing 8kb blocks to /test.dbf
- Test action 2: Start a monitor: for "openstack flavor list" command
- Test action 3: Create a flavor called "test-001"
- Test action 4: Check whether the flavor "test-001" is created
- Test action 5: Continuously measure service outage time from the monitor until the service outage time is more than 5s
- Test action 6: Stop writing to /test.dbf and delete file /test.dbf

Pass / fail criteria

The service outage time is less than 5s.

The nova operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

Delete flavor with "openstack flavor delete test-001".

Test Case 8 - Controller Load Balance as a Service High Availability

Short name

yardstick.ha.haproxy

Yardstick test case: opnfv_yardstick_tc053.yaml

Use case specification

This test verifies the high availability of "haproxy" service. When the "haproxy" service of a specified controller node is killed, whether "haproxy" service on other controller nodes will work, and whether the controller node will restart the "haproxy" service are checked. This test case kills the processes of "haproxy" service on the selected controller node, then checks whether the request of the related OpenStack command is processed with no failure and whether the killed processes are recovered.

Test preconditions

There is more than one controller node, which is providing the "haproxy" service for rest-api.

Denoted as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for monitoring high availability

The high availability of "haproxy" service is evaluated by monitoring service outage time and process outage time

Service outage time is tested by continuously executing "openstack image list" command in loop and checking if the response of the command request is returned with no failure. When the response fails, the "haproxy" service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of processes of "haproxy" service on the selected controller node. The time of those processes being killed to the time of those processes being recovered is the process outage time. Process recovery is verified by checking the existence of processes of "haproxy" service.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that processes of "haproxy" service are running on Node1
- Test action 2: Start two monitors: one for processes of "haproxy" service and the other for "openstack image list" command. Each monitor will run as an independent process
- Test action 3: Connect to Node1 through SSH, and then kill the processes of "haproxy" service
- Test action 4: Continuously measure service outage time from the monitor until the service outage time is more than 5s
- Test action 5: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

The process outage time is less than 30s.

The service outage time is less than 5s.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of "haproxy" if they are not running.

Test Case 9 - Controller node OpenStack service down - Database

Short name

yardstick.ha.database

Yardstick test case: opnfv_yardstick_tc090.yaml

Use case specification

This test case verifies that the high availability of the data base instances used by OpenStack (mysql) on control node is working properly. Specifically, this test case kills the processes of database service on a selected control node, then checks whether the request of the related OpenStack command is OK and the killed processes are recovered.

Test preconditions

In this test case, an attacker called "kill-process" is needed. This attacker includes three parameters: fault_type, process_name and host.

The purpose of this attacker is to kill any process with a specific process name which is run on the host node. In case that multiple processes use the same name on the host node, all of them are going to be killed by this attacker.

Basic test flow execution description and pass/fail criteria

Methodology for verifying service continuity and recovery

In order to verify this service two different monitors are going to be used.

As first monitor is used a OpenStack command and acts as watcher for database connection of different OpenStack components.

For second monitor is used a process monitor and the main purpose is to watch whether the database processes on the host node are killed properly.

Therefore, in this test case, there are two metrics:

- service_outage_time, which indicates the maximum outage time (seconds) of the specified OpenStack command request
- process_recover_time, which indicates the maximum time (seconds) from the process being killed to recovered

Test execution

- Test action 1: Connect to Node1 through SSH, and check that "database" processes are running on Node1
- Test action 2: Start two monitors: one for "database" processes on the host node and the other for connection toward database from OpenStack components, verifying the results of openstack image list, openstack router list, openstack stack list and openstack volume list. Each monitor will run as an independent process
- Test action 3: Connect to Node1 through SSH, and then kill the "mysql" process(es)
- Test action 4: Stop monitors after a period of time specified by "waiting_time". The monitor info will be aggregated.
- Test action 5: Verify the SLA and set the verdict of the test case to pass or fail.

Pass / fail criteria

Check whether the SLA is passed: - The process outage time is less than 30s. - The service outage time is less than 5s.

The database operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

The database service is up and running again. If the database service did not recover successfully by itself, the test explicitly restarts the database service.

Test Case 10 - Controller Messaging Queue as a Service High Availability

Short name

yardstick.ha.rabbitmq

Yardstick test case: opnfv_yardstick_tc056.yaml

Use case specification

This test case will verify the high availability of the messaging queue service (RabbitMQ) that supports OpenStack on controller node. This test case expects that message bus service implementation is RabbitMQ. If the SUT uses a different message bus implementations, the Dovetail configuration (pod.yaml) can be changed accordingly. When messaging queue service (which is active) of a specified controller node is killed, the test case will check whether messaging queue services (which are standby) on other controller nodes will be switched active, and whether the cluster manager on the attacked controller node will restart the stopped messaging queue.

Test preconditions

There is more than one controller node, which is providing the "messaging queue" service. Denoted as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for verifying service continuity and recovery

The high availability of "messaging queue" service is evaluated by monitoring service outage time and process outage time.

Service outage time is tested by continuously executing "openstack image list", "openstack network list", "openstack volume list" and "openstack stack list" commands in loop and checking if the responses of the command requests are returned with no failure. When the response fails, the "messaging queue" service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of processes of "messaging queue" service on the selected controller node. The time of those processes being killed to the time of those processes being recovered is the process outage time. Process recovery is verified by checking the existence of processes of "messaging queue" service.

Test execution

- Test action 1: Start five monitors: one for processes of "messaging queue" service and the others for "openstack image list", "openstack network list", "openstack stack list" and "openstack volume list" command. Each monitor will run as an independent process
- Test action 2: Connect to Node1 through SSH, and then kill all the processes of "messaging queue" service
- Test action 3: Continuously measure service outage time from the monitors until the service outage time is more than 5s
- Test action 4: Continuously measure process outage time from the monitor until the process outage time is more than 30s

Pass / fail criteria

Test passes if the process outage time is no more than 30s and the service outage time is no more than 5s.

A negative result will be generated if the above is not met in completion.

Post conditions

Restart the processes of "messaging queue" if they are not running.

Test Case 11 - Controller node OpenStack service down - Controller Restart

Short name

yardstick.ha.controller_restart

Yardstick test case: opnfv_yardstick_tc025.yaml

Use case specification

This test case verifies that the high availability of controller node is working properly. Specifically, this test case shutdowns a specified controller node via IPMI, then checks whether all services provided by the controller node are OK with some monitor tools.

Test preconditions

In this test case, an attacker called "host-shutdown" is needed. This attacker includes two parameters: fault_type and host.

The purpose of this attacker is to shutdown a controller and check whether the services are handled by this controller are still working normally.

Basic test flow execution description and pass/fail criteria

Methodology for verifying service continuity and recovery

In order to verify this service one monitor is going to be used.

This monitor is using an OpenStack command and the respective command name of the OpenStack component that we want to verify that the respective service is still running normally.

In this test case, there is one metric: 1)service_outage_time: which indicates the maximum outage time (seconds) of the specified OpenStack command request.

Test execution

- Test action 1: Connect to Node1 through SSH, and check that controller services are running normally
- Test action 2: Start monitors: each monitor will run as independently process, monitoring the image list, router list, stack list and volume list accordingly. The monitor info will be collected.
- Test action 3: Using the IPMI component, the Node1 is shut-down remotely.
- Test action 4: Stop monitors after a period of time specified by "waiting_time". The monitor info will be aggregated.
- Test action 5: Verify the SLA and set the verdict of the test case to pass or fail.

Pass / fail criteria

Check whether the SLA is passed: - The process outage time is less than 30s. - The service outage time is less than 5s.

The controller operations are carried out in above order and no errors occur.

A negative result will be generated if the above is not met in completion.

Post conditions

The controller has been restarted

Test Case 12 - OpenStack Controller Virtual Router Service High Availability

Short name

yardstick.ha.neutron_l3_agent

Yardstick test case: opnfv_yardstick_tc058.yaml

Use case specification

This test case will verify the high availability of virtual routers(L3 agent) on controller node. When a virtual router service on a specified controller node is shut down, this test case will check whether the network of virtual machines will be affected, and whether the attacked virtual router service will be recovered.

Test preconditions

There is more than one controller node, which is providing the Neutron API extension called "neutron-l3-agent" virtual router service API.

Denoted as Node1 in the following configuration.

Basic test flow execution description and pass/fail criteria

Methodology for verifying service continuity and recovery

The high availability of "neutrol-l3-agent" virtual router service is evaluated by monitoring service outage time and process outage time.

Service outage is tested using ping to virtual machines. Ping tests that the network routing of virtual machines is ok. When the response fails, the virtual router service is considered in outage. The time between the first response failure and the last response failure is considered as service outage time.

Process outage time is tested by checking the status of processes of "neutron-l3-agent" service on the selected controller node. The time of those processes being killed to the time of those processes being recovered is the process outage time.

Process recovery is verified by checking the existence of processes of "neutron-l3-agent" service.

Test execution

- Test action 1: Two host VMs are booted, these two hosts are in two different networks, the networks are connected by a virtual router.
- Test action 2: Start monitors: each monitor will run with independently process. The monitor info will be collected.
- Test action 3: Do attacker: Connect the host through SSH, and then execute the kill process script with param value specified by "process_name"
- Test action 4: Stop monitors after a period of time specified by "waiting_time" The monitor info will be aggregated.
- Test action 5: Verify the SLA and set the verdict of the test case to pass or fail.

Pass / fail criteria

Check whether the SLA is passed: - The process outage time is less than 30s. - The service outage time is less than 5s.

A negative result will be generated if the above is not met in completion.

Post conditions

Delete image with "openstack image delete neutron-l3-agent_ha_image".

Delete flavor with "openstack flavor delete neutron-l3-agent_ha_flavor".

8.1.2 Patrole Tempest Tests

Scope

This test area evaluates the ability of a system under test to support the role-based access control (RBAC) implementation. The test area specifically validates services image and networking.

References

- OpenStack image service API reference
- OpenStack metadata definitions service API reference
- OpenStack layer 2 networking service API reference
- OpenStack layer 3 networking service API reference
- OpenStack network security API reference
- OpenStack resource management API reference

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM deployed on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links.

Image basic RBAC test:

These tests cover the RBAC tests of image basic operations.

Implementation: BasicOperationsImagesRbacTest

- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_create_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_create_image_tag
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_deactivate_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_delete_image

- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_delete_image_tag
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_download_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_list_images
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_publicize_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_reactivate_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_show_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_update_image
- patrole_tempest_plugin.tests.api.image.test_images_rbac.BasicOperationsImagesRbacTest.test_upload_image

Image namespaces RBAC test:

These tests cover the RBAC tests of image namespaces.

Implementation: ImageNamespacesRbacTest

- patrole_tempest_plugin.tests.api.image.test_image_namespace_rbac.ImageNamespacesRbacTest.test_create_metadef_namespace
- patrole_tempest_plugin.tests.api.image.test_image_namespace_rbac.ImageNamespacesRbacTest.test_modify_metadef_namespac

Image namespaces objects RBAC test:

These tests cover the RBAC tests of image namespaces objects.

Implementation: ImageNamespacesObjectsRbacTest

- patrole_tempest_plugin.tests.api.image.test_image_namespace_objects_rbac.ImageNamespacesObjectsRbacTest.test_create_met
- patrole_tempest_plugin.tests.api.image.test_image_namespace_objects_rbac.ImageNamespacesObjectsRbacTest.test_list_metade
- patrole_tempest_plugin.tests.api.image.test_image_namespace_objects_rbac.ImageNamespacesObjectsRbacTest.test_show_meta
- patrole_tempest_plugin.tests.api.image.test_image_namespace_objects_rbac.ImageNamespacesObjectsRbacTest.test_update_me

Image namespaces property RBAC test:

These tests cover the RBAC tests of image namespaces property.

Implementation: NamespacesPropertyRbacTest

- patrole_tempest_plugin.tests.api.image.test_image_namespace_property_rbac.NamespacesPropertyRbacTest.test_add_md_prope
- patrole_tempest_plugin.tests.api.image.test_image_namespace_property_rbac.NamespacesPropertyRbacTest.test_get_md_proper
- patrole_tempest_plugin.tests.api.image.test_image_namespace_property_rbac.NamespacesPropertyRbacTest.test_get_md_proper
- patrole_tempest_plugin.tests.api.image.test_image_namespace_property_rbac.NamespacesPropertyRbacTest.test_modify_md_pr

Image namespaces tags RBAC test:

These tests cover the RBAC tests of image namespaces tags.

Implementation: NamespaceTagsRbacTest

- patrole_tempest_plugin.tests.api.image.test_image_namespace_tags_rbac.NamespaceTagsRbacTest.test_create_namespace_tag
- patrole_tempest_plugin.tests.api.image.test_image_namespace_tags_rbac.NamespaceTagsRbacTest.test_create_namespace_tags
- patrole_tempest_plugin.tests.api.image.test_image_namespace_tags_rbac.NamespaceTagsRbacTest.test_list_namespace_tags
- patrole_tempest_plugin.tests.api.image.test_image_namespace_tags_rbac.NamespaceTagsRbacTest.test_show_namespace_tag
- patrole_tempest_plugin.tests.api.image.test_image_namespace_tags_rbac.NamespaceTagsRbacTest.test_update_namespace_tag

Image resource types RBAC test:

These tests cover the RBAC tests of image resource types.

Implementation: ImageResourceTypesRbacTest

- patrole_tempest_plugin.tests.api.image.test_image_resource_types_rbac.ImageResourceTypesRbacTest.test_add_metadef_resour
- patrole_tempest_plugin.tests.api.image.test_image_resource_types_rbac.ImageResourceTypesRbacTest.test_get_metadef_resourc
- patrole_tempest_plugin.tests.api.image.test_image_resource_types_rbac.ImageResourceTypesRbacTest.test_list_metadef_resourc

Image member RBAC test:

These tests cover the RBAC tests of image member.

Implementation: ImagesMemberRbacTest

- patrole_tempest_plugin.tests.api.image.test_images_member_rbac.ImagesMemberRbacTest.test_add_image_member
- patrole_tempest_plugin.tests.api.image.test_images_member_rbac.ImagesMemberRbacTest.test_delete_image_member
- patrole_tempest_plugin.tests.api.image.test_images_member_rbac.ImagesMemberRbacTest.test_list_image_members
- patrole_tempest_plugin.tests.api.image.test_images_member_rbac.ImagesMemberRbacTest.test_show_image_member

Network agents RBAC test:

These tests cover the RBAC tests of network agents.

Implementation: AgentsRbacTest and DHCPAgentSchedulersRbacTest.

- patrole_tempest_plugin.tests.api.network.test_agents_rbac.AgentsRbacTest.test_show_agent
- patrole_tempest_plugin.tests.api.network.test_agents_rbac.AgentsRbacTest.test_update_agent
- patrole_tempest_plugin.tests.api.network.test_agents_rbac.DHCPAgentSchedulersRbacTest.test_add_dhcp_agent_to_network
- patrole_tempest_plugin.tests.api.network.test_agents_rbac.DHCPAgentSchedulersRbacTest.test_delete_network_from_dhcp_age
- patrole_tempest_plugin.tests.api.network.test_agents_rbac.DHCPAgentSchedulersRbacTest.test_list_networks_hosted_by_one_d

Network floating ips RBAC test:

These tests cover the RBAC tests of network floating ips.

Implementation: FloatingIpsRbacTest

- patrole_tempest_plugin.tests.api.network.test_floating_ips_rbac.FloatingIpsRbacTest.test_create_floating_ip
- patrole_tempest_plugin.tests.api.network.test_floating_ips_rbac.FloatingIpsRbacTest.test_create_floating_ip_floatingip_address
- patrole_tempest_plugin.tests.api.network.test_floating_ips_rbac.FloatingIpsRbacTest.test_delete_floating_ip
- patrole_tempest_plugin.tests.api.network.test_floating_ips_rbac.FloatingIpsRbacTest.test_show_floating_ip
- patrole_tempest_plugin.tests.api.network.test_floating_ips_rbac.FloatingIpsRbacTest.test_update_floating_ip

Network basic RBAC test:

These tests cover the RBAC tests of network basic operations.

Implementation: NetworksRbacTest

- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network_is_default
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network_router_external
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network_shared

- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_delete_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_list_dhcp_agents_on_hosting_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_show_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_show_network_provider_network_type
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_show_network_provider_physical_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_show_network_provider_segmentation_id
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_show_network_router_external
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_update_network
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_update_network_provider_physical_networ
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_update_network_provider_segmentation_id
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_update_network_router_external
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_update_network_shared

Network ports RBAC test:

These tests cover the RBAC tests of network ports.

Implementation: PortsRbacTest

- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_allowed_address_pairs
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_binding_host_id
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_binding_profile
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_device_owner
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_fixed_ips_ip_address
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_mac_address
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_create_port_security_enabled
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_delete_port
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_show_port
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_show_port_binding_host_id
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_show_port_binding_profile
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_show_port_binding_vif_details
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_show_port_binding_vif_type
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_allowed_address_pairs
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_binding_host_id
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_binding_profile
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_device_owner
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_fixed_ips_ip_address
- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_mac_address

- patrole_tempest_plugin.tests.api.network.test_ports_rbac.PortsRbacTest.test_update_port_security_enabled

Network routers RBAC test:

These tests cover the RBAC tests of network routers.

Implementation: RouterRbacTest

- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_add_router_interface
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_create_router
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_create_router_enable_snat
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_create_router_external_fixed_ips
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_delete_router
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_remove_router_interface
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_show_high_availability_router
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_show_router
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_update_router
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_update_router_enable_snat
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_update_router_external_fixed_ips
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_update_router_external_gateway_info
- patrole_tempest_plugin.tests.api.network.test_routers_rbac.RouterRbacTest.test_update_router_external_gateway_info_network_

Network security groups RBAC test:

These tests cover the RBAC tests of network security groups.

Implementation: SecGroupRbacTest

- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_create_security_group
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_create_security_group_rule
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_delete_security_group
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_delete_security_group_rule
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_list_security_group_rules
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_list_security_groups
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_show_security_group_rule
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_show_security_group
- patrole_tempest_plugin.tests.api.network.test_security_groups_rbac.SecGroupRbacTest.test_update_security_group

Network service providers RBAC test:

These tests cover the RBAC tests of network service providers.

Implementation: ServiceProvidersRbacTest

- patrole_tempest_plugin.tests.api.network.test_service_providers_rbac.ServiceProvidersRbacTest.test_list_service_providers

Network subnetpools RBAC test:

These tests cover the RBAC tests of network subnetpools.

Implementation: SubnetPoolsRbacTest

- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_create_subnetpool
- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_create_subnetpool_shared
- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_delete_subnetpool
- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_show_subnetpool
- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_update_subnetpool
- patrole_tempest_plugin.tests.api.network.test_subnetpools_rbac.SubnetPoolsRbacTest.test_update_subnetpool_is_default

Network subnets RBAC test:

These tests cover the RBAC tests of network subnets.

Implementation: SubnetsRbacTest

- patrole_tempest_plugin.tests.api.network.test_subnets_rbac.SubnetsRbacTest.test_create_subnet
- patrole_tempest_plugin.tests.api.network.test_subnets_rbac.SubnetsRbacTest.test_delete_subnet
- patrole_tempest_plugin.tests.api.network.test_subnets_rbac.SubnetsRbacTest.test_list_subnets
- patrole_tempest_plugin.tests.api.network.test_subnets_rbac.SubnetsRbacTest.test_show_subnet
- patrole_tempest_plugin.tests.api.network.test_subnets_rbac.SubnetsRbacTest.test_update_subnet

Network flavors RBAC test:

These tests cover the RBAC tests of network flavors.

Implementation: FlavorsRbacTest

- patrole_tempest_plugin.tests.api.network.test_flavors_rbac.FlavorsPluginRbacTest.test_create_flavor
- patrole_tempest_plugin.tests.api.network.test_flavors_rbac.FlavorsPluginRbacTest.test_delete_flavor
- patrole_tempest_plugin.tests.api.network.test_flavors_rbac.FlavorsPluginRbacTest.test_list_flavors
- patrole_tempest_plugin.tests.api.network.test_flavors_rbac.FlavorsPluginRbacTest.test_show_flavor
- patrole_tempest_plugin.tests.api.network.test_flavors_rbac.FlavorsPluginRbacTest.test_update_flavor

Network segments RBAC test:

These tests cover the RBAC tests of network segments.

Implementation: SegmentsRbacTest

- patrole_tempest_plugin.tests.api.network.test_network_segments_rbac.NetworkSegmentsRbacTest.test_create_network_segment
- patrole_tempest_plugin.tests.api.network.test_network_segments_rbac.NetworkSegmentsRbacTest.test_show_network_segments
- patrole_tempest_plugin.tests.api.network.test_network_segments_rbac.NetworkSegmentsRbacTest.test_update_network_segmen

8.1.3 Patrole Tempest Tests Depend on Vxlan

Scope

This test area includes some tempest role-based access control (RBAC) tests which depend on vxlan physical networks.

References

- OpenStack image service API reference
- OpenStack metadata definitions service API reference
- OpenStack layer 2 networking service API reference
- OpenStack layer 3 networking service API reference
- OpenStack network security API reference
- OpenStack resource management API reference

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM deployed on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links.

Network basic RBAC test:

These tests cover the RBAC tests of network basic operations by creating a vxlan provider network.

Implementation: NetworksRbacTest

- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network_provider_network_type
- patrole_tempest_plugin.tests.api.network.test_networks_rbac.NetworksRbacTest.test_create_network_provider_segmentation_id

8.1.4 Stress Test Specification

Scope

The stress test involves testing and verifying the ability of the SUT to withstand stress and other challenging factors. Main purpose behind the testing is to make sure the SUT is able to absorb failures while providing an acceptable level of service.

References

This test area references the following specifications, definitions and reviews:

- Upstream OpenStack NOVA Resiliency
  - https://wiki.openstack.org/wiki/NovaResiliency
- Stress Testing over OPNFV Platform
  - https://wiki.opnfv.org/display/bottlenecks/Stress+Testing+over+OPNFV+Platform

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- iff - if and only if
- NFVI - Network Functions Virtualization Infrastructure
- NaN - Not a Number
- Service Level - Measurable terms that describe the quality of the service provided by the SUT within a given time period
- SUT - System Under Test
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

According to the testing goals stated in the test scope section, preceding test will not affect the subsequent test as long as the SUT is able to sustain the given stress while providing an acceptable level of service. Any FAIL result from a single test case will cause the SUT failing the whole test.

Test Descriptions

Test Case 1 - Concurrent capacity based on life-cycle ping test

Short name

bottlenecks.stress.ping

Use case specification

This test case verifies the ability of the SUT concurrently setting up VM pairs for different tenants (through different OpenStack related components) and providing acceptable capacity under stressful conditions. The connectivity between VMs in a VM pair for a tenant is validated through Ping test. A life-cycle event in this test case is particularly referred to a VM pair life-cycle consisting of spawning, pinging and destroying.

Test preconditions

- heat_template_version: 2013-05-23
- ElasticSearch Port: 9200
- LogStash Port: 5044
- Kibana Port: 5601
- Yardstick Port: 5000

Basic test flow execution description and pass/fail criteria

Methodology for validating capacity of the SUT

Validating capacity of the SUT based on life-cycle ping test generally involves 2 subtests which provides secondary validation for the SUT furnishing users with reliable capacity without being crushed.

Let *N1*, *N2*, *N3* and *P1* be certain preset numbers, respectively. In subtest 1, the SUT concurrently setting up *N1* VM pairs with each VM pair belonging to a different tenant. Then VM1 in a VM pair pings VM2 for *P1* times with *P1* packets. The connectivity could be validated iff VM1 successfully pings VM2 with these *P1* packets. Subtest 1 is finished iff all the concurrent (*N1*) requests for creating VM pairs are fulfilled with returned values that indicate the statuses of the VM pairs creations.

Subtest 2 is executed after subtest 1 as secondary validation of the capacity. It follows the same workflow as subtest 1 does to set up *N2* VM pairs.

Assume *S1* and *S2* be the numbers of VM pairs that are successfully created in subtest 1 and subtest 2, respectively. If *min(S1,S2)>=N3*, then the SUT is considered as PASS. Otherwise, we denote the SUT with FAIL.

Note that for subtest 1, if the number of successfully created VM pairs, i.e., *S1*, is smaller than *N3*. Subtest 2 will not be executed and SUT will be marked with FAIL.

Test execution

- Test action 1: Install the testing tools by pulling and running the Bottlenecks Docker container
- Test action 2: Prepare the test by sourcing openstack credential file, eliminating the environment constraints, i.e., Quota setting, setting up Yardstick docker, pulling and registering OS images and VM flavor
- Test action 3: Call Yardstick to concurrently creating *N1* VM pairs for N1 tenants
- Test action 4: In each VM pair, VM1 pings VM2 for *P1* times with *P1* packets while recording the successful numbers
- Test action 5: Mark the VM pairs with *P1* successful pings as PASS and record the total number of PASS VM pairs as *S1*
- Test action 6: Destroy all the VM pairs
- Test action 7: If *S1<N3*, the SUT is marked with FAIL and the test return. Otherwise go to *Test action 8*
- Test action 8: Go to *Test action 3* and do the test again to create *N2* VM pairs with PASS VM pairs counted as *S2*
- Test action 9: If *S2<N3*, the SUT is marked with FAIL. Otherwise marked with PASS.

Pass / Fail criteria

Typical setting of *(N1, N2, N3, P1)* is *(5, 5, 5, 10)*. The reference setting above is acquired based on the results from OPNFV CI jobs and testing over commercial products.

The connectivity within a VM pair is validated iff:

- VM1 successfully pings VM2 for *P1* times with *P1* packets

The SUT is considered passing the test iff:

- *min(S1,S2)>=N3*

Note that after each subtest, the program will check if the successfully created number of VM pairs is smaller than *N3*. If true, the program will return and the SUT will be marked with FAIL. Then the passing criteria is equal to the equation above. When subtest 1 returns, *S2* here is denoted by NaN.

Post conditions

N/A

8.1.5 Tempest Compute test specification

Scope

The Tempest Compute test area evaluates the ability of the System Under Test (SUT) to support dynamic network runtime operations through the life of a VNF. The tests in this test area will evaluate IPv4 network runtime operations functionality.

These runtime operations includes:

- Create, list and show flavors
- Create and list security group rules
- Create, delete and list security groups
- Create, delete, show and list interfaces; attach and deattach ports to servers
- List server addresses
- Individual version endpoints info works
- Servers Test Boot From Volume

References

Security Groups:

- create security group
- delete security group

Networks:

- create network
- delete network

Routers and interface:

- create router
- update router
- delete router
- add interface to router

Subnets:

- create subnet
- update subnet

- delete subnet

Servers:

- create keypair
- create server
- delete server
- add/assign floating IP
- disassociate floating IP

Ports:

- create port
- update port
- delete port

Floating IPs:

- create floating IP
- delete floating IP

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

All these test cases are included in the test case functest.tempest.compute of OVP test suite.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

- **Flavor V2 test**
  - tempest.api.compute.flavors.test_flavors.FlavorsV2TestJSON.test_get_flavor
  - tempest.api.compute.flavors.test_flavors.FlavorsV2TestJSON.test_list_flavors
- **Security Group Rules test**
  - tempest.api.compute.security_groups.test_security_group_rules.SecurityGroupRulesTestJSON.test_security_group_rul
  - tempest.api.compute.security_groups.test_security_group_rules.SecurityGroupRulesTestJSON.test_security_group_rul
- **Security Groups test**
  - tempest.api.compute.security_groups.test_security_groups.SecurityGroupsTestJSON.test_security_groups_create_list_
- **Attach Interfaces test**
  - tempest.api.compute.servers.test_attach_interfaces.AttachInterfacesUnderV243Test.test_add_remove_fixed_ip

- **Server Addresses test**
  - tempest.api.compute.servers.test_server_addresses.ServerAddressesTestJSON.test_list_server_addresses
  - tempest.api.compute.servers.test_server_addresses.ServerAddressesTestJSON.test_list_server_addresses_by_network
- **Test Versions**
  - tempest.api.compute.test_versions.TestVersions.test_get_version_details
- **Servers Test Boot From Volume**
  - tempest.api.compute.servers.test_create_server.ServersTestBootFromVolume.test_verify_server_details
  - tempest.api.compute.servers.test_create_server.ServersTestBootFromVolume.test_list_servers
- **Server Basic Operations test**
  - tempest.scenario.test_server_basic_ops.TestServerBasicOps.test_server_basic_ops

8.1.6 Tempest Identity v3 test specification

Scope

The Tempest Identity v3 test area evaluates the ability of the System Under Test (SUT) to create, list, delete and verify users through the life of a VNF. The tests in this test area will evaluate IPv4 network runtime operations functionality.

These runtime operations may include that create, list, verify and delete:

- credentials
- domains
- endpoints
- user groups
- policies
- regions
- roles
- services
- identities
- API versions

References

Identity API v3.0

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

All these test cases are included in the test case functest.tempest.identity_v3 of OVP test suite.

- **Create, Get, Update and Delete Credentials**
    - tempest.api.identity.admin.v3.test_credentials.CredentialsTestJSON.test_credentials_create_get_update_delete
- **Create and Verify Domain**
    - tempest.api.identity.v3.test_domains.DefaultDomainTestJSON.test_default_domain_exists
- **Create, Update and Delete Domain**
    - tempest.api.identity.admin.v3.test_domains.DomainsTestJSON.test_create_update_delete_domain
- **Create and Update endpoint**
    - tempest.api.identity.admin.v3.test_endpoints.EndPointsTestJSON.test_update_endpoint
- **Create, List and Delete Group Users**
    - tempest.api.identity.admin.v3.test_groups.GroupsV3TestJSON.test_group_users_add_list_delete
- **Update Policy**
    - tempest.api.identity.admin.v3.test_policies.PoliciesTestJSON.test_create_update_delete_policy
- **Create a Region with a Specific Id**
    - tempest.api.identity.admin.v3.test_regions.RegionsTestJSON.test_create_region_with_specific_id
- **Create, Update and Show Role List**
    - tempest.api.identity.admin.v3.test_roles.RolesV3TestJSON.test_role_create_update_show_list
- **Create a Service**
    - tempest.api.identity.admin.v3.test_services.ServicesTestJSON.test_create_update_get_service
- **Create and List Trusts**
    - tempest.api.identity.admin.v3.test_trusts.TrustsV3TestJSON.test_get_trusts_all
- **List API Versions**
    - tempest.api.identity.v3.test_api_discovery.TestApiDiscovery.test_list_api_versions

8.1.7 Tempest Image test specification

Scope

The Tempest Image test area tests the basic operations of Images of the System Under Test (SUT) through the life of a VNF. The tests in this test area will evaluate IPv4 network runtime operations functionality.

References

Image Service API v2

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

All these test cases are included in the test case functest.tempest.image of OVP test suite.

- **Register, Upload, Get Image and Get Image File API's**
  - tempest.api.image.v2.test_images.BasicOperationsImagesTest.test_register_upload_get_image_file
- **List Versions**
  - tempest.api.image.v2.test_versions.VersionsTest.test_list_versions

8.1.8 IPv6 test specification

Scope

The IPv6 test area will evaluate the ability for a SUT to support IPv6 Tenant Network features and functionality. The tests in this test area will evaluate,

- network, subnet, port, router API CRUD operations
- interface add and remove operations
- security group and security group rule API CRUD operations
- IPv6 address assignment with dual stack, dual net, multiprefix in mode DHCPv6 stateless or SLAAC

References

- upstream openstack API reference
  - https://docs.openstack.org/api-ref/
- upstream openstack IPv6 reference
  - https://docs.openstack.org/newton/networking-guide/config-ipv6.html

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- CIDR - Classless Inter-Domain Routing
- CRUD - Create, Read, Update, and Delete
- DHCP - Dynamic Host Configuration Protocol
- DHCPv6 - Dynamic Host Configuration Protocol version 6
- ICMP - Internet Control Message Protocol

- NFVI - Network Functions Virtualization Infrastructure
- NIC - Network Interface Controller
- RA - Router Advertisements
- radvd - The Router Advertisement Daemon
- SDN - Software Defined Network
- SLAAC - Stateless Address Auto Configuration
- TCP - Transmission Control Protocol
- UDP - User Datagram Protocol
- VM - Virtual Machine
- vNIC - virtual Network Interface Card

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM deployed with a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on network, port and subnet operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test.

Test Descriptions

API Used and Reference

Networks: https://docs.openstack.org/api-ref/network/v2/index.html#networks

- show network details
- update network
- delete network
- list networks
- create netowrk
- bulk create networks

Subnets: https://docs.openstack.org/api-ref/network/v2/index.html#subnets

- list subnets
- create subnet
- bulk create subnet
- show subnet details
- update subnet
- delete subnet

Routers and interface: https://docs.openstack.org/api-ref/network/v2/index.html#routers-routers

- list routers
- create router
- show router details
- update router
- delete router
- add interface to router
- remove interface from router

Ports: https://docs.openstack.org/api-ref/network/v2/index.html#ports

- show port details
- update port
- delete port
- list port
- create port
- bulk create ports

Security groups: https://docs.openstack.org/api-ref/network/v2/index.html#security-groups-security-groups

- list security groups
- create security groups
- show security group
- update security group
- delete security group

Security groups rules: https://docs.openstack.org/api-ref/network/v2/index.html#security-group-rules-security-group-rules

- list security group rules
- create security group rule
- show security group rule
- delete security group rule

Servers: https://docs.openstack.org/api-ref/compute/

- list servers
- create server
- create multiple servers
- list servers detailed
- show server details
- update server
- delete server

All IPv6 api and scenario test cases addressed in OVP are covered in the following test specification documents.

Test Case 1 - Create and Delete Bulk Network, IPv6 Subnet and Port

Short name

functest.tempest.ipv6_api.bulk_network_subnet_port_create_delete

Use case specification

This test case evaluates the SUT API ability of creating and deleting multiple networks, IPv6 subnets, ports in one request, the reference is,

tempest.api.network.test_networks.BulkNetworkOpsIpV6Test.test_bulk_create_delete_network tempest.api.network.test_networks.BulkNetworkOpsIpV6Test.test_bulk_create_delete_subnet tempest.api.network.test_networks.BulkNetworkOpsIpV6Test.test_bulk_create_delete_port

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 2 networks using bulk create, storing the “id” parameters returned in the response
- Test action 2: List all networks, verifying the two network id’s are found in the list
- **Test assertion 1:** The two “id” parameters are found in the network list
- Test action 3: Delete the 2 created networks using the stored network ids
- Test action 4: List all networks, verifying the network ids are no longer present
- **Test assertion 2:** The two “id” parameters are not present in the network list
- Test action 5: Create 2 networks using bulk create, storing the “id” parameters returned in the response
- Test action 6: Create an IPv6 subnets on each of the two networks using bulk create commands, storing the associated “id” parameters
- Test action 7: List all subnets, verify the IPv6 subnets are found in the list
- **Test assertion 3:** The two IPv6 subnet “id” parameters are found in the network list
- Test action 8: Delete the 2 IPv6 subnets using the stored “id” parameters
- Test action 9: List all subnets, verify the IPv6 subnets are no longer present in the list
- **Test assertion 4:** The two IPv6 subnet “id” parameters, are not present in list
- Test action 10: Delete the 2 networks created in test action 5, using the stored network ids
- Test action 11: List all networks, verifying the network ids are no longer present
- **Test assertion 5:** The two “id” parameters are not present in the network list
- Test action 12: Create 2 networks using bulk create, storing the “id” parameters returned in the response

- Test action 13: Create a port on each of the two networks using bulk create commands, storing the associated "port_id" parameters
- Test action 14: List all ports, verify the port_ids are found in the list
- **Test assertion 6:** The two "port_id" parameters are found in the ports list
- Test action 15: Delete the 2 ports using the stored "port_id" parameters
- Test action 16: List all ports, verify port_ids are no longer present in the list
- **Test assertion 7:** The two "port_id" parameters, are not present in list
- Test action 17: Delete the 2 networks created in test action 12, using the stored network ids
- Test action 18: List all networks, verifying the network ids are no longer present
- **Test assertion 8:** The two "id" parameters are not present in the network list

Pass / Fail criteria

This test evaluates the ability to use bulk create commands to create networks, IPv6 subnets and ports on the SUT API. Specifically it verifies that:

- Bulk network create commands return valid "id" parameters which are reported in the list commands
- Bulk IPv6 subnet commands return valid "id" parameters which are reported in the list commands
- Bulk port commands return valid "port_id" parameters which are reported in the list commands
- All items created using bulk create commands are able to be removed using the returned identifiers

Post conditions

N/A

Test Case 2 - Create, Update and Delete an IPv6 Network and Subnet

Short name

functest.tempest.ipv6_api.network_subnet_create_update_delete

Use case specification

This test case evaluates the SUT API ability of creating, updating, deleting network and IPv6 subnet with the network, the reference is

tempest.api.network.test_networks.NetworksIpV6Test.test_create_update_delete_network_subnet

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the “id” and “status” parameters returned in the response
- Test action 2: Verify the value of the created network’s “status” is ACTIVE
- **Test assertion 1:** The created network’s “status” is ACTIVE
- Test action 3: Update this network with a new_name
- Test action 4: Verify the network’s name equals the new_name
- **Test assertion 2:** The network’s name equals to the new_name after name updating
- Test action 5: Create an IPv6 subnet within the network, storing the “id” parameters returned in the response
- Test action 6: Update this IPv6 subnet with a new_name
- Test action 7: Verify the IPv6 subnet’s name equals the new_name
- **Test assertion 3:** The IPv6 subnet’s name equals to the new_name after name updating
- Test action 8: Delete the IPv6 subnet created in test action 5, using the stored subnet id
- Test action 9: List all subnets, verifying the subnet id is no longer present
- **Test assertion 4:** The IPv6 subnet “id” is not present in the subnet list
- Test action 10: Delete the network created in test action 1, using the stored network id
- Test action 11: List all networks, verifying the network id is no longer present
- **Test assertion 5:** The network “id” is not present in the network list

Pass / Fail criteria

This test evaluates the ability to create, update, delete network, IPv6 subnet on the SUT API. Specifically it verifies that:

- Create network commands return ACTIVE “status” parameters which are reported in the list commands
- Update network commands return updated “name” parameters which equals to the “name” used
- Update subnet commands return updated “name” parameters which equals to the “name” used
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 3 - Check External Network Visibility

Short name

functest.tempest.ipv6_api.external_network_visibility

Use case specification

This test case verifies user can see external networks but not subnets, the reference is,

tempest.api.network.test_networks.NetworksIpV6Test.test_external_network_visibility

Test preconditions

1. The SUT has at least one external network.
2. In the external network list, there is no network without external router, i.e., all networks in this list are with external router.
3. There is one external network with configured public network id and there is no subnet on this network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all networks with external router, storing the "id"s parameters returned in the response
- Test action 2: Verify list in test action 1 is not empty
- **Test assertion 1:** The network with external router list is not empty
- Test action 3: List all netowrks without external router in test action 1 list
- Test action 4: Verify list in test action 3 is empty
- **Test assertion 2:** networks without external router in the external network list is empty
- Test action 5: Verify the configured public network id is found in test action 1 stored "id"s
- **Test assertion 3:** the public network id is found in the external network "id"s
- Test action 6: List the subnets of the external network with the configured public network id
- Test action 7: Verify list in test action 6 is empty
- **Test assertion 4:** There is no subnet of the external network with the configured public network id

Pass / Fail criteria

This test evaluates the ability to use list commands to list external networks, pre-configured public network. Specifically it verifies that:

- Network list commands to find visible networks with external router
- Network list commands to find visible network with pre-configured public network id
- Subnet list commands to find no subnet on the pre-configured public network

Post conditions

None

Test Case 4 - List IPv6 Networks and Subnets

Short name

functest.tempest.ipv6_api.network_subnet_list

Use case specification

This test case evaluates the SUT API ability of listing netowrks, subnets after creating a network and an IPv6 subnet, the reference is

tempest.api.network.test_networks.NetworksIpV6Test.test_list_networks tempest.api.network.test_networks.NetworksIpV6Test.test_lis

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" parameter returned in the response
- Test action 2: List all networks, verifying the network id is found in the list
- **Test assertion 1:** The "id" parameter is found in the network list
- Test action 3: Create an IPv6 subnet of the network created in test action 1. storing the "id" parameter returned in the response
- Test action 4: List all subnets of this network, verifying the IPv6 subnet id is found in the list
- **Test assertion 2:** The "id" parameter is found in the IPv6 subnet list
- Test action 5: Delete the IPv6 subnet using the stored "id" parameters
- Test action 6: List all subnets, verify subnet_id is no longer present in the list
- **Test assertion 3:** The IPv6 subnet "id" parameter is not present in list
- Test action 7: Delete the network created in test action 1, using the stored network ids
- Test action 8: List all networks, verifying the network id is no longer present
- **Test assertion 4:** The network "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use create commands to create network, IPv6 subnet, list commands to list the created networks, IPv6 subnet on the SUT API. Specifically it verifies that:

- Create commands to create network, IPv6 subnet
- List commands to find that netowrk, IPv6 subnet in the all networks, subnets list after creating
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 5 - Show Details of an IPv6 Network and Subnet

Short name

functest.tempest.ipv6_api.network_subnet_show

Use case specification

This test case evaluates the SUT API ability of showing the network, subnet details, the reference is,

tempest.api.network.test_networks.NetworksIpV6Test.test_show_network tempest.api.network.test_networks.NetworksIpV6Test.test_s

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" and "name" parameter returned in the response
- Test action 2: Show the network id and name, verifying the network id and name equal to the "id" and "name" stored in test action 1
- **Test assertion 1:** The id and name equal to the "id" and "name" stored in test action 1
- Test action 3: Create an IPv6 subnet of the network, storing the "id" and CIDR parameter returned in the response
- Test action 4: Show the details of the created IPv6 subnet, verifying the id and CIDR in the details are equal to the stored id and CIDR in test action 3.
- **Test assertion 2:** The "id" and CIDR in show details equal to "id" and CIDR stored in test action 3
- Test action 5: Delete the IPv6 subnet using the stored "id" parameter
- Test action 6: List all subnets on the network, verify the IPv6 subnet id is no longer present in the list
- **Test assertion 3:** The IPv6 subnet "id" parameter is not present in list
- Test action 7: Delete the network created in test action 1, using the stored network id
- Test action 8: List all networks, verifying the network id is no longer present
- **Test assertion 4:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use create commands to create network, IPv6 subnet and show commands to show network, IPv6 subnet details on the SUT API. Specifically it verifies that:

- Network show commands return correct “id” and “name” parameter which equal to the returned response in the create commands
- IPv6 subnet show commands return correct “id” and CIDR parameter which equal to the returned response in the create commands
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 6 - Create an IPv6 Port in Allowed Allocation Pools

Short name

functest.tempest.ipv6_api.port_create_in_allocation_pool

Use case specification

This test case evaluates the SUT API ability of creating an IPv6 subnet within allowed IPv6 address allocation pool and creating a port whose address is in the range of the pool, the reference is,

tempest.api.network.test_ports.PortsIpV6TestJSON.test_create_port_in_allowed_allocation_pools

Test preconditions

There should be an IPv6 CIDR configuration, which prefixlen is less than 126.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the “id” parameter returned in the response
- Test action 2: Check the allocation pools configuration, verifying the prefixlen of the IPv6 CIDR configuration is less than 126.
- **Test assertion 1:** The prefixlen of the IPv6 CIDR configuration is less than 126
- Test action 3: Get the allocation pool by setting the start_ip and end_ip based on the IPv6 CIDR configuration.
- Test action 4: Create an IPv6 subnet of the network within the allocation pools, storing the “id” parameter returned in the response
- Test action 5: Create a port of the network, storing the “id” parameter returned in the response
- Test action 6: Verify the port’s id is in the range of the allocation pools which is got is test action 3

- **Test assertion 2:** the port's id is in the range of the allocation pools
- Test action 7: Delete the port using the stored "id" parameter
- Test action 8: List all ports, verify the port id is no longer present in the list
- **Test assertion 3:** The port "id" parameter is not present in list
- Test action 9: Delete the IPv6 subnet using the stored "id" parameter
- Test action 10: List all subnets on the network, verify the IPv6 subnet id is no longer present in the list
- **Test assertion 4:** The IPv6 subnet "id" parameter is not present in list
- Test action 11: Delete the network created in test action 1, using the stored network id
- Test action 12: List all networks, verifying the network id is no longer present
- **Test assertion 5:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use create commands to create an IPv6 subnet within allowed IPv6 address allocation pool and create a port whose address is in the range of the pool. Specifically it verifies that:

- IPv6 subnet create command to create an IPv6 subnet within allowed IPv6 address allocation pool
- Port create command to create a port whose id is in the range of the allocation pools
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 7 - Create an IPv6 Port with Empty Security Groups

Short name

functest.tempest.ipv6_api.port_create_empty_security_group

Use case specification

This test case evaluates the SUT API ability of creating port with empty security group, the reference is,

tempest.api.network.test_ports.PortsIpV6TestJSON.test_create_port_with_no_securitygroups

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the “id” parameter returned in the response
- Test action 2: Create an IPv6 subnet of the network, storing the “id” parameter returned in the response
- Test action 3: Create a port of the network with an empty security group, storing the “id” parameter returned in the response
- Test action 4: Verify the security group of the port is not none but is empty
- **Test assertion 1:** the security group of the port is not none but is empty
- Test action 5: Delete the port using the stored “id” parameter
- Test action 6: List all ports, verify the port id is no longer present in the list
- **Test assertion 2:** The port “id” parameter is not present in list
- Test action 7: Delete the IPv6 subnet using the stored “id” parameter
- Test action 8: List all subnets on the network, verify the IPv6 subnet id is no longer present in the list
- **Test assertion 3:** The IPv6 subnet “id” parameter is not present in list
- Test action 9: Delete the network created in test action 1, using the stored network id
- Test action 10: List all networks, verifying the network id is no longer present
- **Test assertion 4:** The “id” parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use create commands to create port with empty security group of the SUT API. Specifically it verifies that:

- Port create commands to create a port with an empty security group
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 8 - Create, Update and Delete an IPv6 Port

Short name

functest.tempest.ipv6_api.port_create_update_delete

Use case specification

This test case evaluates the SUT API ability of creating, updating, deleting IPv6 port, the reference is,

tempest.api.network.test_ports.PortsIpV6TestJSON.test_create_update_delete_port

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" parameter returned in the response
- Test action 2: Create a port of the network, storing the "id" and "admin_state_up" parameters returned in the response
- Test action 3: Verify the value of port's 'admin_state_up' is True
- **Test assertion 1:** the value of port's 'admin_state_up' is True after creating
- Test action 4: Update the port's name with a new_name and set port's admin_state_up to False, storing the name and admin_state_up parameters returned in the response
- Test action 5: Verify the stored port's name equals to new_name and the port's admin_state_up is False.
- **Test assertion 2:** the stored port's name equals to new_name and the port's admin_state_up is False
- Test action 6: Delete the port using the stored "id" parameter
- Test action 7: List all ports, verify the port is no longer present in the list
- **Test assertion 3:** The port "id" parameter is not present in list
- Test action 8: Delete the network created in test action 1, using the stored network id
- Test action 9: List all networks, verifying the network id is no longer present
- **Test assertion 4:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use create/update/delete commands to create/update/delete port of the SUT API. Specifically it verifies that:

- Port create commands return True of 'admin_state_up' in response
- Port update commands to update 'name' to new_name and 'admin_state_up' to false
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 9 - List IPv6 Ports

Short name

functest.tempest.ipv6_api.port_list

Use case specification

This test case evaluates the SUT ability of creating a port on a network and finding the port in the all ports list, the reference is,

tempest.api.network.test_ports.PortsIpV6TestJSON.test_list_ports

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" parameter returned in the response
- Test action 2: Create a port of the network, storing the "id" parameter returned in the response
- Test action 3: List all ports, verify the port id is found in the list
- **Test assertion 1:** The "id" parameter is found in the port list
- Test action 4: Delete the port using the stored "id" parameter
- Test action 5: List all ports, verify the port is no longer present in the list
- **Test assertion 2:** The port "id" parameter is not present in list
- Test action 6: Delete the network created in test action 1, using the stored network id
- Test action 7: List all networks, verifying the network id is no longer present
- **Test assertion 3:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use list commands to list the networks and ports on the SUT API. Specifically it verifies that:

- Port list command to list all ports, the created port is found in the list.
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 10 - Show Key/Valus Details of an IPv6 Port

Short name

functest.tempest.ipv6_api.port_show_details

Use case specification

This test case evaluates the SUT ability of showing the port details, the values in the details should be equal to the values to create the port, the reference is,

tempest.api.network.test_ports.PortsIpV6TestJSON.test_show_port

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the “id” parameter returned in the response
- Test action 2: Create a port of the network, storing the “id” parameter returned in the response
- Test action 3: Show the details of the port, verify the stored port’s id in test action 2 exists in the details
- **Test assertion 1:** The “id” parameter is found in the port shown details
- Test action 4: Verify the values in the details of the port are the same as the values to create the port
- **Test assertion 2:** The values in the details of the port are the same as the values to create the port
- Test action 5: Delete the port using the stored “id” parameter
- Test action 6: List all ports, verify the port is no longer present in the list
- **Test assertion 3:** The port “id” parameter is not present in list
- Test action 7: Delete the network created in test action 1, using the stored network id
- Test action 8: List all networks, verifying the network id is no longer present
- **Test assertion 4:** The “id” parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to use show commands to show port details on the SUT API. Specifically it verifies that:

- Port show commands to show the details of the port, whose id is in the details
- Port show commands to show the details of the port, whose values are the same as the values to create the port
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 11 - Add Multiple Interfaces for an IPv6 Router

Short name

functest.tempest.ipv6_api.router_add_multiple_interface

Use case specification

This test case evaluates the SUT ability of adding multiple interface to a router, the reference is,

tempest.api.network.test_routers.RoutersIpV6Test.test_add_multiple_router_interfaces

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 2 networks named network01 and network02 sequentially, storing the "id" parameters returned in the response
- Test action 2: Create an IPv6 subnet01 in network01, an IPv6 subnet02 in network02 sequentially, storing the "id" parameters returned in the response
- Test action 3: Create a router, storing the "id" parameter returned in the response
- Test action 4: Create interface01 with subnet01 and the router
- Test action 5: Verify the router_id stored in test action 3 equals to the interface01's 'device_id' and subnet01_id stored in test action 2 equals to the interface01's 'subnet_id'
- **Test assertion 1:** the router_id equals to the interface01's 'device_id' and subnet01_id equals to the interface01's 'subnet_id'
- Test action 5: Create interface02 with subnet02 and the router
- Test action 6: Verify the router_id stored in test action 3 equals to the interface02's 'device_id' and subnet02_id stored in test action 2 equals to the interface02's 'subnet_id'
- **Test assertion 2:** the router_id equals to the interface02's 'device_id' and subnet02_id equals to the interface02's 'subnet_id'
- Test action 7: Delete the interfaces, router, IPv6 subnets and networks, networks, subnets, then list all interfaces, ports, IPv6 subnets, networks, the test passes if the deleted ones are not found in the list.
- **Test assertion 3:** The interfaces, router, IPv6 subnets and networks ids are not present in the lists after deleting

Pass / Fail criteria

This test evaluates the ability to use bulk create commands to create networks, IPv6 subnets and ports on the SUT API. Specifically it verifies that:

- Interface create commands to create interface with IPv6 subnet and router, interface 'device_id' and 'subnet_id' should equal to the router id and IPv6 subnet id, respectively.
- Interface create commands to create multiple interface with the same router and multiple IPv6 subnets.
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 12 - Add and Remove an IPv6 Router Interface with port_id

Short name

functest.tempest.ipv6_api.router_interface_add_remove_with_port

Use case specification

This test case evaluates the SUT abiltiy of adding, removing router interface to a port, the subnet_id and port_id of the interface will be checked, the port's device_id will be checked if equals to the router_id or not. The reference is,

tempest.api.network.test_routers.RoutersIpV6Test.test_add_remove_router_interface_with_port_id

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" parameter returned in the response
- Test action 2: Create an IPv6 subnet of the network, storing the "id" parameter returned in the response
- Test action 3: Create a router, storing the "id" parameter returned in the response
- Test action 4: Create a port of the network, storing the "id" parameter returned in the response
- Test action 5: Add router interface to the port created, storing the "id" parameter returned in the response
- Test action 6: Verify the interface's keys include 'subnet_id' and 'port_id'
- **Test assertion 1:** the interface's keys include 'subnet_id' and 'port_id'
- Test action 7: Show the port details, verify the 'device_id' in port details equals to the router id stored in test action 3
- **Test assertion 2:** 'device_id' in port details equals to the router id
- Test action 8: Delete the interface, port, router, subnet and network, then list all interfaces, ports, routers, subnets and networks, the test passes if the deleted ones are not found in the list.
- **Test assertion 3:** interfaces, ports, routers, subnets and networks are not found in the lists after deleting

Pass / Fail criteria

This test evaluates the ability to use add/remove commands to add/remove router interface to the port, show commands to show port details on the SUT API. Specifically it verifies that:

- Router_interface add commands to add router interface to a port, the interface's keys should include 'subnet_id' and 'port_id'
- Port show commands to show 'device_id' in port details, which should be equal to the router id
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 13 - Add and Remove an IPv6 Router Interface with subnet_id

Short name

functest.tempest.ipv6_api.router_interface_add_remove

Use case specification

This test case evaluates the SUT API ability of adding and removing a router interface with the IPv6 subnet id, the reference is

tempest.api.network.test_routers.RoutersIpV6Test.test_add_remove_router_interface_with_subnet_id

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network, storing the "id" parameter returned in the response
- Test action 2: Create an IPv6 subnet with the network created, storing the "id" parameter returned in the response
- Test action 3: Create a router, storing the "id" parameter returned in the response
- Test action 4: Add a router interface with the stored ids of the router and IPv6 subnet
- **Test assertion 1:** Key 'subnet_id' is included in the added interface's keys
- **Test assertion 2:** Key 'port_id' is included in the added interface's keys
- Test action 5: Show the port info with the stored interface's port id
- **Test assertion 3:**: The stored router id is equal to the device id shown in the port info
- Test action 6: Delete the router interface created in test action 4, using the stored subnet id

- Test action 7: List all router interfaces, verifying the router interface is no longer present
- **Test assertion 4:** The router interface with the stored subnet id is not present in the router interface list
- Test action 8: Delete the router created in test action 3, using the stored router id
- Test action 9: List all routers, verifying the router id is no longer present
- **Test assertion 5:** The router "id" parameter is not present in the router list
- Test action 10: Delete the subnet created in test action 2, using the stored subnet id
- Test action 11: List all subnets, verifying the subnet id is no longer present
- **Test assertion 6:** The subnet "id" parameter is not present in the subnet list
- Test action 12: Delete the network created in test action 1, using the stored network id
- Test action 13: List all networks, verifying the network id is no longer present
- **Test assertion 7:** The network "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to add and remove router interface with the subnet id on the SUT API. Specifically it verifies that:

- Router interface add command returns valid 'subnet_id' parameter which is reported in the interface's keys
- Router interface add command returns valid 'port_id' parameter which is reported in the interface's keys
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 14 - Create, Show, List, Update and Delete an IPv6 router

Short name

functest.tempest.ipv6_api.router_create_show_list_update_delete

Use case specification

This test case evaluates the SUT API ability of creating, showing, listing, updating and deleting routers, the reference is

tempest.api.network.test_routers.RoutersIpV6Test.test_create_show_list_update_delete_router

Test preconditions

There should exist an OpenStack external network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a router, set the admin_state_up to be False and external_network_id to be public network id, storing the "id" parameter returned in the response
- **Test assertion 1:** The created router's admin_state_up is False
- **Test assertion 2:** The created router's external network id equals to the public network id
- Test action 2: Show details of the router created in test action 1, using the stored router id
- **Test assertion 3:** The router's name shown is the same as the router created
- **Test assertion 4:** The router's external network id shown is the same as the public network id
- Test action 3: List all routers and verify if created router is in response message
- **Test assertion 5:** The stored router id is in the router list
- Test action 4: Update the name of router and verify if it is updated
- **Test assertion 6:** The name of router equals to the name used to update in test action 4
- Test action 5: Show the details of router, using the stored router id
- **Test assertion 7:** The router's name shown equals to the name used to update in test action 4
- Test action 6: Delete the router created in test action 1, using the stored router id
- Test action 7: List all routers, verifying the router id is no longer present
- **Test assertion 8:** The "id" parameter is not present in the router list

Pass / Fail criteria

This test evaluates the ability to create, show, list, update and delete router on the SUT API. Specifically it verifies that:

- Router create command returns valid "admin_state_up" and "id" parameters which equal to the "admin_state_up" and "id" returned in the response
- Router show command returns valid "name" parameter which equals to the "name" returned in the response
- Router show command returns valid "external network id" parameters which equals to the public network id
- Router list command returns valid "id" parameter which equals to the stored router "id"
- Router update command returns updated "name" parameters which equals to the "name" used to update
- Router created using create command is able to be removed using the returned identifiers

Post conditions

None

Test Case 15 - Create, List, Update, Show and Delete an IPv6 security group

Short name

functest.tempest.ipv6_api.security_group_create_list_update_show_delete

Use case specification

This test case evaluates the SUT API ability of creating, listing, updating, showing and deleting security groups, the reference is

tempest.api.network.test_security_groups.SecGroupIPv6Test.test_create_list_update_show_delete_security_group

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group, storing the "id" parameter returned in the response
- Test action 2: List all security groups and verify if created security group is there in response
- **Test assertion 1:** The created security group's "id" is found in the list
- Test action 3: Update the name and description of this security group, using the stored id
- Test action 4: Verify if the security group's name and description are updated
- **Test assertion 2:** The security group's name equals to the name used in test action 3
- **Test assertion 3:** The security group's description equals to the description used in test action 3
- Test action 5: Show details of the updated security group, using the stored id
- **Test assertion 4:** The security group's name shown equals to the name used in test action 3
- **Test assertion 5:** The security group's description shown equals to the description used in test action 3
- Test action 6: Delete the security group created in test action 1, using the stored id
- Test action 7: List all security groups, verifying the security group's id is no longer present
- **Test assertion 6:** The "id" parameter is not present in the security group list

Pass / Fail criteria

This test evaluates the ability to create list, update, show and delete security group on the SUT API. Specifically it verifies that:

- Security group create commands return valid "id" parameter which is reported in the list commands
- Security group update commands return valid "name" and "description" parameters which are reported in the show commands

- Security group created using create command is able to be removed using the returned identifiers

Post conditions

None

Test Case 16 - Create, Show and Delete IPv6 security group rule

Short name

functest.tempest.ipv6_api.security_group_rule_create_show_delete

Use case specification

This test case evaluates the SUT API ability of creating, showing, listing and deleting security group rules, the reference is

tempest.api.network.test_security_groups.SecGroupIPv6Test.test_create_show_delete_security_group_rule

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group, storing the "id" parameter returned in the response
- Test action 2: Create a rule of the security group with protocol tcp, udp and icmp, respectively, using the stored security group's id, storing the "id" parameter returned in the response
- Test action 3: Show details of the created security group rule, using the stored id of the security group rule
- **Test assertion 1:** All the created security group rule's values equal to the rule values shown in test action 3
- Test action 4: List all security group rules
- **Test assertion 2:** The stored security group rule's id is found in the list
- Test action 5: Delete the security group rule, using the stored security group rule's id
- Test action 6: List all security group rules, verifying the security group rule's id is no longer present
- **Test assertion 3:** The security group rule "id" parameter is not present in the list
- Test action 7: Delete the security group, using the stored security group's id
- Test action 8: List all security groups, verifying the security group's id is no longer present
- **Test assertion 4:** The security group "id" parameter is not present in the list

Pass / Fail criteria

This test evaluates the ability to create, show, list and delete security group rules on the SUT API. Specifically it verifies that:

- Security group rule create command returns valid values which are reported in the show command
- Security group rule created using create command is able to be removed using the returned identifiers

Post conditions

None

Test Case 17 - List IPv6 Security Groups

Short name

functest.tempest.ipv6_api.security_group_list

Use case specification

This test case evaluates the SUT API ability of listing security groups, the reference is

tempest.api.network.test_security_groups.SecGroupIPv6Test.test_list_security_groups

Test preconditions

There should exist a default security group.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all security groups
- Test action 2: Verify the default security group exists in the list, the test passes if the default security group exists
- **Test assertion 1:** The default security group is in the list

Pass / Fail criteria

This test evaluates the ability to list security groups on the SUT API. Specifically it verifies that:

- Security group list command return valid security groups which include the default security group

Post conditions

None

Test Case 1 - IPv6 Address Assignment - Dual Stack, SLAAC, DHCPv6 Stateless

Short name

functest.tempest.ipv6_scenario.dhcpv6_stateless

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless'. In this case, guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6 stateless. This test case then verifies the ping6 available VM can ping the other VM's v4 and v6 addresses as well as the v6 subnet's gateway ip in the same network, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_dhcp6_stateless_from_os

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the "id" parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the "id" parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create one IPv6 subnet of the network created in test action 1 in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless', storing the "id" parameter returned in the response
- Test action 6: Connect the IPv6 subnet to the router, using the stored IPv6 subnet id
- Test action 7: Boot two VMs on this network, storing the "id" parameters returned in the response
- **Test assertion 1:** The vNIC of each VM gets one v4 address and one v6 address actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 address as well as the v6 subnet's gateway ip
- Test action 8: Delete the 2 VMs created in test action 7, using the stored ids
- Test action 9: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 10: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 11: Delete the IPv6 subnet created in test action 5, using the stored id
- Test action 12: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The "id" parameters of IPv4 and IPv6 are not present in the list

- Test action 13: Delete the network created in test action 1, using the stored id
- Test action 14: List all networks, verifying the id is no longer present
- **Test assertion 6:** The “id” parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode ‘dhcpv6_stateless’ and ipv6_address_mode ‘dhcpv6_stateless’, and verify the ping6 available VM can ping the other VM’s v4 and v6 addresses as well as the v6 subnet’s gateway ip in the same network. Specifically it verifies that:

- The IPv6 addresses in mode ‘dhcpv6_stateless’ assigned successfully
- The VM can ping the other VM’s IPv4 and IPv6 private addresses as well as the v6 subnet’s gateway ip
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 2 - IPv6 Address Assignment - Dual Net, Dual Stack, SLAAC, DHCPv6 Stateless

Short name

functest.tempest.ipv6_scenario.dualnet_dhcpv6_stateless

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode ‘dhcpv6_stateless’ and ipv6_address_mode ‘dhcpv6_stateless’. In this case, guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6 stateless. This test case then verifies the ping6 available VM can ping the other VM’s v4 address in one network and v6 address in another network as well as the v6 subnet’s gateway ip, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_dualnet_dhcp6_stateless_from_os

Test preconditions

There should exists a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the “id” parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the “id” parameter returned in the response

- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create another network, storing the "id" parameter returned in the response
- Test action 6: Create one IPv6 subnet of network created in test action 5 in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless', storing the "id" parameter returned in the response
- Test action 7: Connect the IPv6 subnet to the router, using the stored IPv6 subnet id
- Test action 8: Boot two VMs on these two networks, storing the "id" parameters returned in the response
- Test action 9: Turn on 2nd NIC of each VM for the network created in test action 5
- **Test assertion 1:** The 1st vNIC of each VM gets one v4 address assigned and the 2nd vNIC of each VM gets one v6 address actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 address as well as the v6 subnet's gateway ip
- Test action 10: Delete the 2 VMs created in test action 8, using the stored ids
- Test action 11: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 12: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 13: Delete the IPv6 subnet created in test action 6, using the stored id
- Test action 14: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The "id" parameters of IPv4 and IPv6 are not present in the list
- Test action 15: Delete the 2 networks created in test action 1 and 5, using the stored ids
- Test action 16: List all networks, verifying the ids are no longer present
- **Test assertion 6:** The two "id" parameters are not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless', and verify the ping6 available VM can ping the other VM's v4 address in one network and v6 address in another network as well as the v6 subnet's gateway ip. Specifically it verifies that:

- The IPv6 addresses in mode 'dhcpv6_stateless' assigned successfully
- The VM can ping the other VM's IPv4 address in one network and IPv6 address in another network as well as the v6 subnet's gateway ip
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 3 - IPv6 Address Assignment - Multiple Prefixes, Dual Stack, SLAAC, DHCPv6 Stateless

Short name

functest.tempest.ipv6_scenario.multiple_prefixes_dhcpv6_stateless

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless'. In this case, guest instance obtains IPv6 addresses from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6 stateless. This test case then verifies the ping6 available VM can ping the other VM's one v4 address and two v6 addresses with different prefixes as well as the v6 subnets' gateway ips in the same network, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_multi_prefix_dhcpv6_stateless

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the "id" parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the "id" parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create two IPv6 subnets of the network created in test action 1 in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless', storing the "id" parameters returned in the response
- Test action 6: Connect the two IPv6 subnets to the router, using the stored IPv6 subnet ids
- Test action 7: Boot two VMs on this network, storing the "id" parameters returned in the response
- **Test assertion 1:** The vNIC of each VM gets one v4 address and two v6 addresses with different prefixes actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 addresses as well as the v6 subnets' gateway ips
- Test action 8: Delete the 2 VMs created in test action 7, using the stored ids
- Test action 9: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 10: Delete the IPv4 subnet created in test action 2, using the stored id

- Test action 11: Delete two IPv6 subnets created in test action 5, using the stored ids
- Test action 12: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The "id" parameters of IPv4 and IPv6 are not present in the list
- Test action 13: Delete the network created in test action 1, using the stored id
- Test action 14: List all networks, verifying the id is no longer present
- **Test assertion 6:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless', and verify the ping6 available VM can ping the other VM's v4 address and two v6 addresses with different prefixes as well as the v6 subnets' gateway ips in the same network. Specifically it verifies that:

- The different prefixes IPv6 addresses in mode 'dhcpv6_stateless' assigned successfully
- The VM can ping the other VM's IPv4 and IPv6 private addresses as well as the v6 subnets' gateway ips
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 4 - IPv6 Address Assignment - Dual Net, Multiple Prefixes, Dual Stack, SLAAC, DHCPv6 Stateless

Short name

functest.tempest.ipv6_scenario.dualnet_multiple_prefixes_dhcpv6_stateless

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'dhcpv6_stateless' and ipv6_address_mode 'dhcpv6_stateless'. In this case, guest instance obtains IPv6 addresses from OpenStack managed radvd using SLAAC and optional info from dnsmasq using DHCPv6 stateless. This test case then verifies the ping6 available VM can ping the other VM's v4 address in one network and two v6 addresses with different prefixes in another network as well as the v6 subnets' gateway ips, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_dualnet_multi_prefix_dhcpv6_stateless

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the “id” parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the “id” parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create another network, storing the “id” parameter returned in the response
- Test action 6: Create two IPv6 subnets of network created in test action 5 in ipv6_ra_mode ‘dhcpv6_stateless’ and ipv6_address_mode ‘dhcpv6_stateless’, storing the “id” parameters returned in the response
- Test action 7: Connect the two IPv6 subnets to the router, using the stored IPv6 subnet ids
- Test action 8: Boot two VMs on these two networks, storing the “id” parameters returned in the response
- Test action 9: Turn on 2nd NIC of each VM for the network created in test action 5
- **Test assertion 1:** The vNIC of each VM gets one v4 address and two v6 addresses with different prefixes actually assigned
- **Test assertion 2:** Each VM can ping the other’s v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other’s v6 addresses as well as the v6 subnets’ gateway ips
- Test action 10: Delete the 2 VMs created in test action 8, using the stored ids
- Test action 11: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two “id” parameters are not present in the VM list
- Test action 12: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 13: Delete two IPv6 subnets created in test action 6, using the stored ids
- Test action 14: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The “id” parameters of IPv4 and IPv6 are not present in the list
- Test action 15: Delete the 2 networks created in test action 1 and 5, using the stored ids
- Test action 16: List all networks, verifying the ids are no longer present
- **Test assertion 6:** The two “id” parameters are not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode ‘dhcpv6_stateless’ and ipv6_address_mode ‘dhcpv6_stateless’, and verify the ping6 available VM can ping the other VM’s v4 address in one network and two v6 addresses with different prefixes in another network as well as the v6 subnets’ gateway ips. Specifically it verifies that:

- The IPv6 addresses in mode ‘dhcpv6_stateless’ assigned successfully
- The VM can ping the other VM’s IPv4 and IPv6 private addresses as well as the v6 subnets’ gateway ips
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 5 - IPv6 Address Assignment - Dual Stack, SLAAC

Short name

functest.tempest.ipv6_scenario.slaac

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac'. In this case, guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC. This test case then verifies the ping6 available VM can ping the other VM's v4 and v6 addresses as well as the v6 subnet's gateway ip in the same network, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_slaac_from_os

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the "id" parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the "id" parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create one IPv6 subnet of the network created in test action 1 in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac', storing the "id" parameter returned in the response
- Test action 6: Connect the IPv6 subnet to the router, using the stored IPv6 subnet id
- Test action 7: Boot two VMs on this network, storing the "id" parameters returned in the response
- **Test assertion 1:** The vNIC of each VM gets one v4 address and one v6 address actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 address as well as the v6 subnet's gateway ip
- Test action 8: Delete the 2 VMs created in test action 7, using the stored ids
- Test action 9: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 10: Delete the IPv4 subnet created in test action 2, using the stored id

- Test action 11: Delete the IPv6 subnet created in test action 5, using the stored id
- Test action 12: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The "id" parameters of IPv4 and IPv6 are not present in the list
- Test action 13: Delete the network created in test action 1, using the stored id
- Test action 14: List all networks, verifying the id is no longer present
- **Test assertion 6:** The "id" parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac', and verify the ping6 available VM can ping the other VM's v4 and v6 addresses as well as the v6 subnet's gateway ip in the same network. Specifically it verifies that:

- The IPv6 addresses in mode 'slaac' assigned successfully
- The VM can ping the other VM's IPv4 and IPv6 private addresses as well as the v6 subnet's gateway ip
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 6 - IPv6 Address Assignment - Dual Net, Dual Stack, SLAAC

Short name

functest.tempest.ipv6_scenario.dualnet_slaac

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac'. In this case, guest instance obtains IPv6 address from OpenStack managed radvd using SLAAC. This test case then verifies the ping6 available VM can ping the other VM's v4 address in one network and v6 address in another network as well as the v6 subnet's gateway ip, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_dualnet_slaac_from_os

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the "id" parameter returned in the response

- Test action 2: Create one IPv4 subnet of the created network, storing the "id" parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create another network, storing the "id" parameter returned in the response
- Test action 6: Create one IPv6 subnet of network created in test action 5 in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac', storing the "id" parameter returned in the response
- Test action 7: Connect the IPv6 subnet to the router, using the stored IPv6 subnet id
- Test action 8: Boot two VMs on these two networks, storing the "id" parameters returned in the response
- Test action 9: Turn on 2nd NIC of each VM for the network created in test action 5
- **Test assertion 1:** The 1st vNIC of each VM gets one v4 address assigned and the 2nd vNIC of each VM gets one v6 address actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 address as well as the v6 subnet's gateway ip
- Test action 10: Delete the 2 VMs created in test action 8, using the stored ids
- Test action 11: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 12: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 13: Delete the IPv6 subnet created in test action 6, using the stored id
- Test action 14: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The "id" parameters of IPv4 and IPv6 are not present in the list
- Test action 15: Delete the 2 networks created in test action 1 and 5, using the stored ids
- Test action 16: List all networks, verifying the ids are no longer present
- **Test assertion 6:** The two "id" parameters are not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac', and verify the ping6 available VM can ping the other VM's v4 address in one network and v6 address in another network as well as the v6 subnet's gateway ip. Specifically it verifies that:

- The IPv6 addresses in mode 'slaac' assigned successfully
- The VM can ping the other VM's IPv4 address in one network and IPv6 address in another network as well as the v6 subnet's gateway ip
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 7 - IPv6 Address Assignment - Multiple Prefixes, Dual Stack, SLAAC

Short name

functest.tempest.ipv6_scenario.multiple_prefixes_slaac

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac'. In this case, guest instance obtains IPv6 addresses from OpenStack managed radvd using SLAAC. This test case then verifies the ping6 available VM can ping the other VM's one v4 address and two v6 addresses with different prefixes as well as the v6 subnets' gateway ips in the same network, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_multi_prefix_slaac

Test preconditions

There should exists a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the "id" parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the "id parameter returned in the response
- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create two IPv6 subnets of the network created in test action 1 in ipv6_ra_mode 'slaac' and ipv6_address_mode 'slaac', storing the "id" parameters returned in the response
- Test action 6: Connect the two IPv6 subnets to the router, using the stored IPv6 subnet ids
- Test action 7: Boot two VMs on this network, storing the "id" parameters returned in the response
- **Test assertion 1:** The vNIC of each VM gets one v4 address and two v6 addresses with different prefixes actually assigned
- **Test assertion 2:** Each VM can ping the other's v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other's v6 addresses as well as the v6 subnets' gateway ips
- Test action 8: Delete the 2 VMs created in test action 7, using the stored ids
- Test action 9: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two "id" parameters are not present in the VM list
- Test action 10: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 11: Delete two IPv6 subnets created in test action 5, using the stored ids
- Test action 12: List all subnets, verifying the ids are no longer present

- **Test assertion 5:** The “id” parameters of IPv4 and IPv6 are not present in the list
- Test action 13: Delete the network created in test action 1, using the stored id
- Test action 14: List all networks, verifying the id is no longer present
- **Test assertion 6:** The “id” parameter is not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode ‘slaac’ and ipv6_address_mode ‘slaac’, and verify the ping6 available VM can ping the other VM’s v4 address and two v6 addresses with different prefixes as well as the v6 subnets’ gateway ips in the same network. Specifically it verifies that:

- The different prefixes IPv6 addresses in mode ‘slaac’ assigned successfully
- The VM can ping the other VM’s IPv4 and IPv6 private addresses as well as the v6 subnets’ gateway ips
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

Test Case 8 - IPv6 Address Assignment - Dual Net, Dual Stack, Multiple Prefixes, SLAAC

Short name

functest.tempest.ipv6_scenario.dualnet_multiple_prefixes_slaac

Use case specification

This test case evaluates IPv6 address assignment in ipv6_ra_mode ‘slaac’ and ipv6_address_mode ‘slaac’. In this case, guest instance obtains IPv6 addresses from OpenStack managed radvd using SLAAC. This test case then verifies the ping6 available VM can ping the other VM’s v4 address in one network and two v6 addresses with different prefixes in another network as well as the v6 subnets’ gateway ips, the reference is

tempest.scenario.test_network_v6.TestGettingAddress.test_dualnet_multi_prefix_slaac

Test preconditions

There should exist a public router or a public network.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create one network, storing the “id” parameter returned in the response
- Test action 2: Create one IPv4 subnet of the created network, storing the “id” parameter returned in the response

- Test action 3: If there exists a public router, use it as the router. Otherwise, use the public network to create a router
- Test action 4: Connect the IPv4 subnet to the router, using the stored IPv4 subnet id
- Test action 5: Create another network, storing the “id” parameter returned in the response
- Test action 6: Create two IPv6 subnets of network created in test action 5 in ipv6_ra_mode ‘slaac’ and ipv6_address_mode ‘slaac’, storing the “id” parameters returned in the response
- Test action 7: Connect the two IPv6 subnets to the router, using the stored IPv6 subnet ids
- Test action 8: Boot two VMs on these two networks, storing the “id” parameters returned in the response
- Test action 9: Turn on 2nd NIC of each VM for the network created in test action 5
- **Test assertion 1:** The vNIC of each VM gets one v4 address and two v6 addresses with different prefixes actually assigned
- **Test assertion 2:** Each VM can ping the other’s v4 private address
- **Test assertion 3:** The ping6 available VM can ping the other’s v6 addresses as well as the v6 subnets’ gateway ips
- Test action 10: Delete the 2 VMs created in test action 8, using the stored ids
- Test action 11: List all VMs, verifying the ids are no longer present
- **Test assertion 4:** The two “id” parameters are not present in the VM list
- Test action 12: Delete the IPv4 subnet created in test action 2, using the stored id
- Test action 13: Delete two IPv6 subnets created in test action 6, using the stored ids
- Test action 14: List all subnets, verifying the ids are no longer present
- **Test assertion 5:** The “id” parameters of IPv4 and IPv6 are not present in the list
- Test action 15: Delete the 2 networks created in test action 1 and 5, using the stored ids
- Test action 16: List all networks, verifying the ids are no longer present
- **Test assertion 6:** The two “id” parameters are not present in the network list

Pass / Fail criteria

This test evaluates the ability to assign IPv6 addresses in ipv6_ra_mode ‘slaac’ and ipv6_address_mode ‘slaac’, and verify the ping6 available VM can ping the other VM’s v4 address in one network and two v6 addresses with different prefixes in another network as well as the v6 subnets’ gateway ips. Specifically it verifies that:

- The IPv6 addresses in mode ‘slaac’ assigned successfully
- The VM can ping the other VM’s IPv4 and IPv6 private addresses as well as the v6 subnets’ gateway ips
- All items created using create commands are able to be removed using the returned identifiers

Post conditions

None

8.1.9 VM Resource Scheduling on Multiple Nodes test specification

Scope

The VM resource scheduling test area evaluates the ability of the system under test to support VM resource scheduling on multiple nodes. The tests in this test area will evaluate capabilities to schedule VM to multiple compute nodes directly with scheduler hints, and create server groups with policy affinity and anti-affinity.

References

- Availability zone
  - https://docs.openstack.org/newton/networking-guide/config-az.html
- Scheduling
  - https://docs.openstack.org/kilo/config-reference/content/section_compute-scheduler.html

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- NFVi - Network Functions Virtualization infrastructure
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on server group operations and server operations on multiple nodes. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

All these test cases are included in the test case functest.tempest.multi_node_scheduling of OVP test suite.

Test Descriptions

API Used and Reference

Security Groups: https://docs.openstack.org/api-ref/network/v2/index.html#security-groups-security-groups

- create security group
- delete security group

Networks: https://docs.openstack.org/api-ref/network/v2/index.html#networks

- create network

- delete network

Routers and interface: https://docs.openstack.org/api-ref/network/v2/index.html#routers-routers

- create router
- delete router
- add interface to router

Subnets: https://docs.openstack.org/api-ref/network/v2/index.html#subnets

- create subnet
- delete subnet

Servers: https://docs.openstack.org/api-ref/compute/

- create keypair
- create server
- show server
- delete server
- add/assign floating IP
- create server group
- delete server group
- list server groups
- show server group details

Ports: https://docs.openstack.org/api-ref/network/v2/index.html#ports

- create port
- delete port

Floating IPs: https://docs.openstack.org/api-ref/network/v2/index.html#floating-ips-floatingips

- create floating IP
- delete floating IP

Availability zone: https://docs.openstack.org/api-ref/compute/

- get availability zone

Test Case 1 - Schedule VM to compute nodes

Test case specification

tempest.scenario.test_server_multinode.TestServerMultinode.test_schedule_to_all_nodes

Test preconditions

- At least 2 compute nodes
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Get all availability zones AZONES1 in the SUT
- Test action 2: Get all compute nodes in AZONES1
- Test action 3: Get the value of ‘min_compute_nodes’ which is set by user in tempest config file and means the minimum number of compute nodes expected
- **Test assertion 1:** Verify that SUT has at least as many compute nodes as specified by the ‘min_compute_nodes’ threshold
- Test action 4: Create one server for each compute node, up to the ‘min_compute_nodes’ threshold
- **Test assertion 2:** Verify the number of servers matches the ‘min_compute_nodes’ threshold
- Test action 5: Get every server’s ‘hostId’ and store them in a set which has no duplicate values
- **Test assertion 3:** Verify the length of the set equals to the number of servers to ensure that every server ended up on a different host
- Test action 6: Delete the created servers

Pass / Fail criteria

This test evaluates the functionality of VM resource scheduling. Specifically, the test verifies that:

- VMs are scheduled to the requested compute nodes correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Test create and delete multiple server groups with same name and policy

Test case specification

tempest.api.compute.servers.test_server_group.ServerGroupTestJSON.test_create_delete_multiple_server_groups_with_same_name_po

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Generate a random name N1

- Test action 2: Create a server group SERG1 with N1 and policy affinity
- Test action 3: Create another server group SERG2 with N1 and policy affinity
- **Test assertion 1:** The names of SERG1 and SERG2 are the same
- **Test assertion 2:** The 'policies' of SERG1 and SERG2 are the same
- **Test assertion 3:** The ids of SERG1 and SERG2 are different
- Test action 4: Delete SERG1 and SERG2
- Test action 5: List all server groups
- **Test assertion 4:** SERG1 and SERG2 are not in the list

Pass / Fail criteria

This test evaluates the functionality of creating and deleting server groups with the same name and policy. Specifically, the test verifies that:

- Server groups can be created with the same name and policy.
- Server groups with the same name and policy can be deleted successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Test create and delete server group with affinity policy

Test case specification

tempest.api.compute.servers.test_server_group.ServerGroupTestJSON.test_create_delete_server_group_with_affinity_policy

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Generate a random name N1
- Test action 2: Create a server group SERG1 with N1 and policy affinity
- **Test assertion 1:** The name of SERG1 returned in the response is the same as N1
- **Test assertion 2:** The 'policies' of SERG1 returned in the response is affinity
- Test action 3: Delete SERG1 and list all server groups
- **Test assertion 3:** SERG1 is not in the list

Pass / Fail criteria

This test evaluates the functionality of creating and deleting server group with affinity policy. Specifically, the test verifies that:

- Server group can be created with affinity policy and deleted successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - Test create and delete server group with anti-affinity policy

Test case specification

tempest.api.compute.servers.test_server_group.ServerGroupTestJSON.test_create_delete_server_group_with_anti_affinity_policy

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Generate a random name N1
- Test action 2: Create a server group SERG1 with N1 and policy anti-affinity
- **Test assertion 1:** The name of SERG1 returned in the response is the same as N1
- **Test assertion 2:** The ‘policies’ of SERG1 returned in the response is anti-affinity
- Test action 3: Delete SERG1 and list all server groups
- **Test assertion 3:** SERG1 is not in the list

Pass / Fail criteria

This test evaluates the functionality of creating and deleting server group with anti-affinity policy. Specifically, the test verifies that:

- Server group can be created with anti-affinity policy and deleted successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Test list server groups

Test case specification

tempest.api.compute.servers.test_server_group.ServerGroupTestJSON.test_list_server_groups

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Generate a random name N1
- Test action 2: Create a server group SERG1 with N1 and policy affinity
- Test action 3: List all server groups
- **Test assertion 1:** SERG1 is in the list
- Test action 4: Delete SERG1

Pass / Fail criteria

This test evaluates the functionality of listing server groups. Specifically, the test verifies that:

- Server groups can be listed successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 6 - Test show server group details

Test case specification

tempest.api.compute.servers.test_server_group.ServerGroupTestJSON.test_show_server_group

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Generate a random name N1
- Test action 2: Create a server group SERG1 with N1 and policy affinity, and stored the details (D1) returned in the response
- Test action 3: Show the details (D2) of SERG1
- **Test assertion 1:** All values in D1 are the same as the values in D2
- Test action 4: Delete SERG1

Pass / Fail criteria

This test evaluates the functionality of showing server group details. Specifically, the test verifies that:

- Server groups can be shown successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.10 Tempest Network API test specification

Scope

The Tempest Network API test area tests the basic operations of the System Under Test (SUT) through the life of a VNF. The tests in this test area will evaluate IPv4 network runtime operations functionality.

These runtime operations may include that create, list, verify or delete:

- Floating IP
- Network
- Subnet
- Port
- External Network Visibility
- Router
- Subnetpools
- API Version Resources

References

Networks:

- create network

- delete network

Routers and interface:

- create router
- update router
- delete router
- add interface to router

Subnets:

- create subnet
- update subnet
- delete subnet

Subnetpools:

- create subnetpool
- update subnetpool
- delete subnetpool

Ports:

- create port
- update port
- delete port

Floating IPs:

- create floating IP
- delete floating IP

Api Versions

- list version
- show version

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

All these test cases are included in the test case functest.tempest.network of OVP test suite.

List, Show and Verify the Details of the Available Extensions

- tempest.api.network.test_extensions.ExtensionsTestJSON.test_list_show_extensions

Floating IP tests

- Create a Floating IP

  - Update a Floating IP
  - Delete a Floating IP
  - List all Floating IPs
  - Show Floating IP Details
  - Associate a Floating IP with a Port and then Delete that Port
  - Associate a Floating IP with a Port and then with a Port on Another Router

- tempest.api.network.test_floating_ips.FloatingIPTestJSON.test_create_floating_ip_specifying_a_fixed_ip_address
- tempest.api.network.test_floating_ips.FloatingIPTestJSON.test_create_list_show_update_delete_floating_ip

Network tests

  - Bulk Network Creation & Deletion
  - Bulk Subnet Create & Deletion
  - Bulk Port Creation & Deletion
  - List Project's Networks

- tempest.api.network.test_networks.BulkNetworkOpsTest.test_bulk_create_delete_network
- tempest.api.network.test_networks.BulkNetworkOpsTest.test_bulk_create_delete_port
- tempest.api.network.test_networks.BulkNetworkOpsTest.test_bulk_create_delete_subnet

External Network Visibility test

- tempest.api.network.test_networks.NetworksTest.test_external_network_visibility

Create Port with No Security Groups test

- tempest.api.network.test_ports.PortsTestJSON.test_create_port_with_no_securitygroups

Router test

- tempest.api.network.test_routers.RoutersTest.test_add_multiple_router_interfaces
- tempest.api.network.test_routers.RoutersTest.test_add_remove_router_interface_with_port_id
- tempest.api.network.test_routers.RoutersTest.test_add_remove_router_interface_with_subnet_id
- tempest.api.network.test_routers.RoutersTest.test_create_show_list_update_delete_router

Create, List, Show, Update and Delete Subnetpools

- tempest.api.network.test_subnetpools_extensions.SubnetPoolsTestJSON.test_create_list_show_update_delete_subnetpools

API Version Resources test

- tempest.api.network.test_versions.NetworksApiDiscovery.test_api_version_resources

8.1.11 Tempest Network Scenario test specification

Scope

The Tempest Network scenario test area evaluates the ability of the system under test to support dynamic network runtime operations through the life of a VNF (e.g. attach/detach, enable/disable, read stats). The tests in this test area will evaluate IPv4 network runtime operations functionality. These runtime operations includes hotpluging network interface, detaching floating-ip from VM, attaching floating-ip to VM, updating subnet's DNS, updating VM instance port admin state and updating router admin state.

References

- DNS
  - https://docs.openstack.org/newton/networking-guide/config-dns-res.html

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- DNS - Domain Name System
- ICMP - Internet Control Message Protocol
- MAC - Media Access Control
- NIC - Network Interface Controller
- NFVi - Network Functions Virtualization infrastructure
- SSH - Secure Shell
- TCP - Transmission Control Protocol
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on dynamic network runtime operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

All these test cases are included in the test case functest.tempest.network_scenario of OVP test suite.

Test Descriptions

API Used and Reference

Security Groups: https://docs.openstack.org/api-ref/network/v2/index.html#security-groups-security-groups

- create security group
- delete security group

Networks: https://docs.openstack.org/api-ref/network/v2/index.html#networks

- create network
- delete network

Routers and interface: https://docs.openstack.org/api-ref/network/v2/index.html#routers-routers

- create router
- update router
- delete router
- add interface to router

Subnets: https://docs.openstack.org/api-ref/network/v2/index.html#subnets

- create subnet
- update subnet
- delete subnet

Servers: https://docs.openstack.org/api-ref/compute/

- create keypair
- create server
- delete server
- add/assign floating IP
- disassociate floating IP

Ports: https://docs.openstack.org/api-ref/network/v2/index.html#ports

- create port
- update port
- delete port

Floating IPs: https://docs.openstack.org/api-ref/network/v2/index.html#floating-ips-floatingips

- create floating IP
- delete floating IP

Test Case 1 - Basic network operations

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_network_basic_ops

Test preconditions

- Nova has been configured to boot VMs with Neutron-managed networking
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic

- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating IP FIP1 (via R1) to VM1
- **Test assertion 1:** Ping FIP1 and SSH to VM1 via FIP1 successfully
- **Test assertion 2:** Ping the internal gateway from VM1 successfully
- **Test assertion 3:** Ping the default gateway from VM1 using its floating IP FIP1 successfully
- Test action 6: Detach FIP1 from VM1
- **Test assertion 4:** VM1 becomes unreachable after FIP1 disassociated
- Test action 7: Create a new server VM2 with NET1, and associate floating IP FIP1 to VM2
- **Test assertion 5:** Ping FIP1 and SSH to VM2 via FIP1 successfully
- Test action 8: Delete SG1, NET1, SUBNET1, R1, VM1, VM2 and FIP1

Pass / Fail criteria

This test evaluates the functionality of basic network operations. Specifically, the test verifies that:

- The Tempest host can ping VM's IP address. This implies, but does not guarantee (see the ssh check that follows), that the VM has been assigned the correct IP address and has connectivity to the Tempest host.
- The Tempest host can perform key-based authentication to an ssh server hosted at VM's IP address. This check guarantees that the IP address is associated with the target VM.
- The Tempest host can ssh into the VM via the IP address and successfully ping the internal gateway address, implying connectivity to another VM on the same network.
- The Tempest host can ssh into the VM via the IP address and successfully ping the default gateway, implying external connectivity.
- Detach the floating-ip from the VM and VM becomes unreachable.
- Associate attached floating ip to a new VM and the new VM is reachable.
- Floating IP status is updated correctly after each change.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Hotplug network interface

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_hotplug_nic

Test preconditions

- Nova has been configured to boot VMs with Neutron-managed networking
- Compute interface_attach feature is enabled
- VM vnic_type is not set to 'direct' or 'macvtap'
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating IP FIP1 (via R1) to VM1
- **Test assertion 1:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- Test action 6: Create a second neutron network NET2 and subnet SUBNET2, and attach VM1 to NET2
- Test action 7: Get VM1's ethernet interface NIC2 for NET2
- **Test assertion 2:** Ping NET2's internal gateway successfully
- Test action 8: Delete SG1, NET1, NET2, SUBNET1, SUBNET2, R1, NIC2, VM1 and FIP1

Pass / Fail criteria

This test evaluates the functionality of adding network to an active VM. Specifically, the test verifies that:

- New network interface can be added to an existing VM successfully.
- The Tempest host can ssh into the VM via the IP address and successfully ping the new network's internal gateway address, implying connectivity to the new network.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Update subnet's configuration

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_subnet_details

Test preconditions

- Nova has been configured to boot VMs with Neutron-managed networking
- DHCP client is available
- Tenant networks should be non-shared and isolated
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface, configure SUBNET1 with dns nameserver '1.2.3.4'
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating IP FIP1 (via R1) to VM1
- **Test assertion 1:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- **Test assertion 2:** Retrieve the VM1's configured dns and verify it matches the one configured for SUBNET1
- Test action 6: Update SUBNET1's dns to '9.8.7.6'
- **Test assertion 3:** After triggering the DHCP renew from the VM manually, retrieve the VM1's configured dns and verify it has been successfully updated
- Test action 7: Delete SG1, NET1, SUBNET1, R1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the functionality of updating subnet's configurations. Specifically, the test verifies that:

- Updating subnet's DNS server configurations are affecting the VMs.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - Update VM port admin state

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_update_instance_port_admin_state

Test preconditions

- Nova has been configured to boot VMs with Neutron-managed networking
- Network port_admin_state_change feature is enabled
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating IP FIP1 (via R1) to VM1
- Test action 6: Create a server VM2 with SG1 and NET1, and assign a floating IP FIP2 to VM2
- Test action 7: Get a SSH client SSHCLNT1 to VM2
- **Test assertion 1:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- **Test assertion 2:** Ping FIP1 via SSHCLNT1 successfully
- Test action 8: Update admin_state_up attribute of VM1 port to False
- **Test assertion 3:** Ping FIP1 and SSH to VM1 with FIP1 failed
- **Test assertion 4:** Ping FIP1 via SSHCLNT1 failed
- Test action 9: Update admin_state_up attribute of VM1 port to True
- **Test assertion 5:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- **Test assertion 6:** Ping FIP1 via SSHCLNT1 successfully
- Test action 10: Delete SG1, NET1, SUBNET1, R1, SSHCLNT1, VM1, VM2 and FIP1, FIP2

Pass / Fail criteria

This test evaluates the VM public and project connectivity status by changing VM port admin_state_up to True & False. Specifically, the test verifies that:

- Public and project connectivity is reachable before updating admin_state_up attribute of VM port to False.
- Public and project connectivity is unreachable after updating admin_state_up attribute of VM port to False.
- Public and project connectivity is reachable after updating admin_state_up attribute of VM port from False to True.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Update router admin state

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_update_router_admin_state

Test preconditions

- Nova has been configured to boot VMs with Neutron-managed networking
- Multi-tenant networks capabilities
- Openstack nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating IP FIP1 (via R1) to VM1
- **Test assertion 1:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- Test action 6: Update admin_state_up attribute of R1 to False
- **Test assertion 2:** Ping FIP1 and SSH to VM1 with FIP1 failed
- Test action 7: Update admin_state_up attribute of R1 to True
- **Test assertion 3:** Ping FIP1 and SSH to VM1 with FIP1 successfully
- Test action 8: Delete SG1, NET1, SUBNET1, R1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the router public connectivity status by changing router admin_state_up to True & False. Specifically, the test verifies that:

- Public connectivity is reachable before updating admin_state_up attribute of router to False.
- Public connectivity is unreachable after updating admin_state_up attribute of router to False.
- Public connectivity is reachable after updating admin_state_up attribute of router. from False to True

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.12 Security Group and Port Security test specification

Scope

The security group and port security test area evaluates the ability of the system under test to support packet filtering by security group and port security. The tests in this test area will evaluate preventing MAC spoofing by port security, basic security group operations including testing cross/in tenant traffic, testing multiple security groups, using port security to disable security groups and updating security groups.

References

N/A

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- ICMP - Internet Control Message Protocol
- MAC - Media Access Control
- NFVi - Network Functions Virtualization infrastructure
- SSH - Secure Shell
- TCP - Transmission Control Protocol
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on the basic operations of security group and port security. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

All these test cases are included in the test case functest.tempest.network_security of OVP test suite.

Test Descriptions

API Used and Reference

Security Groups: https://docs.openstack.org/api-ref/network/v2/index.html#security-groups-security-groups

- create security group
- delete security group

Networks: https://docs.openstack.org/api-ref/network/v2/index.html#networks

- create network
- delete network
- list networks
- create floating ip
- delete floating ip

Routers and interface: https://docs.openstack.org/api-ref/network/v2/index.html#routers-routers

- create router
- delete router
- list routers
- add interface to router

Subnets: https://docs.openstack.org/api-ref/network/v2/index.html#subnets

- create subnet
- list subnets
- delete subnet

Servers: https://docs.openstack.org/api-ref/compute/

- create keypair
- create server
- delete server
- add/assign floating ip

Ports: https://docs.openstack.org/api-ref/network/v2/index.html#ports

- update port
- list ports
- show port details

Test Case 1 - Port Security and MAC Spoofing

Test case specification

tempest.scenario.test_network_basic_ops.TestNetworkBasicOps.test_port_security_macspoofing_port

Test preconditions

- Neutron port-security extension API
- Neutron security-group extension API
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 2: Create a neutron network NET1
- Test action 3: Create a tenant router R1 which routes traffic to public network
- Test action 4: Create a subnet SUBNET1 and add it as router interface
- Test action 5: Create a server VM1 with SG1 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 6: Verify can ping FIP1 successfully and can SSH to VM1 with FIP1
- Test action 7: Create a second neutron network NET2 and subnet SUBNET2, and attach VM1 to NET2
- Test action 8: Get VM1's ethernet interface NIC2 for NET2
- Test action 9: Create second server VM2 on NET2
- Test action 10: Verify VM1 is able to communicate with VM2 via NIC2
- Test action 11: Login to VM1 and spoof the MAC address of NIC2 to "00:00:00:00:00:01"
- Test action 12: Verify VM1 fails to communicate with VM2 via NIC2
- **Test assertion 1:** The ping operation is failed
- Test action 13: Update 'security_groups' to be none for VM1's NIC2 port
- Test action 14: Update 'port_security_enable' to be False for VM1's NIC2 port
- Test action 15: Verify now VM1 is able to communicate with VM2 via NIC2
- **Test assertion 2:** The ping operation is successful
- Test action 16: Delete SG1, NET1, NET2, SUBNET1, SUBNET2, R1, VM1, VM2 and FIP1

Pass / Fail criteria

This test evaluates the ability to prevent MAC spoofing by using port security. Specifically, the test verifies that:

- With port security, the ICMP packets from a spoof server cannot pass the port.
- Without port security, the ICMP packets from a spoof server can pass the port.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Test Security Group Cross Tenant Traffic

Test case specification

tempest.scenario.test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_cross_tenant_traffic

Test preconditions

- Neutron security-group extension API
- Two tenants
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a neutron network NET1 for primary tenant
- Test action 2: Create a primary tenant router R1 which routes traffic to public network
- Test action 3: Create a subnet SUBNET1 and add it as router interface
- Test action 4: Create 2 empty security groups SG1 and SG2 for primary tenant
- Test action 5: Add a tcp rule to SG1
- Test action 6: Create a server VM1 with SG1, SG2 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 7: Repeat test action 1 to 6 and create NET2, R2, SUBNET2, SG3, SG4, FIP2 and VM2 for an alt_tenant
- Test action 8: Verify VM1 fails to communicate with VM2 through FIP2
- **Test assertion 1:** The ping operation is failed
- Test action 9: Add ICMP rule to SG4
- Test action 10: Verify VM1 is able to communicate with VM2 through FIP2
- **Test assertion 2:** The ping operation is successful
- Test action 11: Verify VM2 fails to communicate with VM1 through FIP1
- **Test assertion 3:** The ping operation is failed
- Test action 12: Add ICMP rule to SG2
- Test action 13: Verify VM2 is able to communicate with VM1 through FIP1
- **Test assertion 4:** The ping operation is successful
- Test action 14: Delete SG1, SG2, SG3, SG4, NET1, NET2, SUBNET1, SUBNET2, R1, R2, VM1, VM2, FIP1 and FIP2

Pass / Fail criteria

This test evaluates the ability of the security group to filter packets cross tenant. Specifically, the test verifies that:

- Without ICMP security group rule, the ICMP packets cannot be received by the server in another tenant which differs from the source server.
- With ingress ICMP security group rule enabled only at tenant1, the server in tenant2 can ping server in tenant1 but not the reverse direction.
- With ingress ICMP security group rule enabled at tenant2 also, the ping works from both directions.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Test Security Group in Tenant Traffic

Test case specification

tempest.scenario.test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_in_tenant_traffic

Test preconditions

- Neutron security-group extension API
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a neutron network NET1
- Test action 2: Create a tenant router R1 which routes traffic to public network
- Test action 3: Create a subnet SUBNET1 and add it as router interface
- Test action 4: Create 2 empty security groups SG1 and SG2
- Test action 5: Add a tcp rule to SG1
- Test action 6: Create a server VM1 with SG1, SG2 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 7: Create second server VM2 with default security group and NET1
- Test action 8: Verify VM1 fails to communicate with VM2 through VM2's fixed ip
- **Test assertion 1:** The ping operation is failed
- Test action 9: Add ICMP security group rule to default security group
- Test action 10: Verify VM1 is able to communicate with VM2 through VM2's fixed ip
- **Test assertion 2:** The ping operation is successful

- Test action 11: Delete SG1, SG2, NET1, SUBNET1, R1, VM1, VM2 and FIP1

Pass / Fail criteria

This test evaluates the ability of the security group to filter packets in one tenant. Specifically, the test verifies that:

- Without ICMP security group rule, the ICMP packets cannot be received by the server in the same tenant.
- With ICMP security group rule, the ICMP packets can be received by the server in the same tenant.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - Test Multiple Security Groups

Test case specification

tempest.scenario.test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_multiple_security_groups

Test preconditions

- Neutron security-group extension API
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a neutron network NET1
- Test action 2: Create a tenant router R1 which routes traffic to public network
- Test action 3: Create a subnet SUBNET1 and add it as router interface
- Test action 4: Create 2 empty security groups SG1 and SG2
- Test action 5: Add a tcp rule to SG1
- Test action 6: Create a server VM1 with SG1, SG2 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 7: Verify failed to ping FIP1
- **Test assertion 1:** The ping operation is failed
- Test action 8: Add ICMP security group rule to SG2
- Test action 9: Verify can ping FIP1 successfully
- **Test assertion 2:** The ping operation is successful
- Test action 10: Verify can SSH to VM1 with FIP1

- **Test assertion 3:** Can SSH to VM1 successfully
- Test action 11: Delete SG1, SG2, NET1, SUBNET1, R1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability of multiple security groups to filter packets. Specifically, the test verifies that:

- A server with 2 security groups, one with TCP rule and without ICMP rule, cannot receive the ICMP packets sending from the tempest host machine.
- A server with 2 security groups, one with TCP rule and the other with ICMP rule, can receive the ICMP packets sending from the tempest host machine and be connected via the SSH client.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Test Port Security Disable Security Group

Test case specification

tempest.scenario.test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_port_security_disable_security_group

Test preconditions

- Neutron security-group extension API
- Neutron port-security extension API
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a neutron network NET1
- Test action 2: Create a tenant router R1 which routes traffic to public network
- Test action 3: Create a subnet SUBNET1 and add it as router interface
- Test action 4: Create 2 empty security groups SG1 and SG2
- Test action 5: Add a tcp rule to SG1
- Test action 6: Create a server VM1 with SG1, SG2 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 7: Create second server VM2 with default security group and NET1
- Test action 8: Update ‘security_groups’ to be none and ‘port_security_enabled’ to be True for VM2’s port
- Test action 9: Verify VM1 fails to communicate with VM2 through VM2’s fixed ip

- **Test assertion 1:** The ping operation is failed
- Test action 10: Update 'security_groups' to be none and 'port_security_enabled' to be False for VM2's port
- Test action 11: Verify VM1 is able to communicate with VM2 through VM2's fixed ip
- **Test assertion 2:** The ping operation is successful
- Test action 12: Delete SG1, SG2, NET1, SUBNET1, R1, VM1, VM2 and FIP1

Pass / Fail criteria

This test evaluates the ability of port security to disable security group. Specifically, the test verifies that:

- The ICMP packets cannot pass the port whose 'port_security_enabled' is True and security_groups is none.
- The ICMP packets can pass the port whose 'port_security_enabled' is False and security_groups is none.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 6 - Test Update Port Security Group

Test case specification

tempest.scenario.test_security_groups_basic_ops.TestSecurityGroupsBasicOps.test_port_update_new_security_group

Test preconditions

- Neutron security-group extension API
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a neutron network NET1
- Test action 2: Create a tenant router R1 which routes traffic to public network
- Test action 3: Create a subnet SUBNET1 and add it as router interface
- Test action 4: Create 2 empty security groups SG1 and SG2
- Test action 5: Add a tcp rule to SG1
- Test action 6: Create a server VM1 with SG1, SG2 and NET1, and assign a floating ip FIP1 (via R1) to VM1
- Test action 7: Create third empty security group SG3
- Test action 8: Add ICMP rule to SG3

- Test action 9: Create second server VM2 with default security group and NET1
- Test action 10: Verify VM1 fails to communicate with VM2 through VM2's fixed ip
- **Test assertion 1:** The ping operation is failed
- Test action 11: Update 'security_groups' to be SG3 for VM2's port
- Test action 12: Verify VM1 is able to communicate with VM2 through VM2's fixed ip
- **Test assertion 2:** The ping operation is successful
- Test action 13: Delete SG1, SG2, SG3, NET1, SUBNET1, R1, VM1, VM2 and FIP1

Pass / Fail criteria

This test evaluates the ability to update port with a new security group. Specifically, the test verifies that:

- Without ICMP security group rule, the VM cannot receive ICMP packets.
- Update the port's security group which has ICMP rule, the VM can receive ICMP packets.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.13 OpenStack Interoperability Test Specification

The test cases documented here are the API test cases in the OpenStack Interop guideline 2018.11 as implemented by the RefStack client.

References

- OpenStack Governance/Interop
  - https://wiki.openstack.org/wiki/Governance/InteropWG
- OpenStack Interoperability guidelines (version 2018.11)
  - https://github.com/openstack/interop/blob/master/2018.11.json
- Refstack client
  - https://github.com/openstack/refstack-client

All OpenStack interop test cases addressed in OVP are covered in the following test specification documents.

VIM compute operations test specification

Scope

The VIM compute operations test area evaluates the ability of the system under test to support VIM compute operations. The test cases documented here are the compute API test cases in the OpenStack Interop guideline 2018.11 as implemented by the RefStack client. These test cases will evaluate basic OpenStack (as a VIM) compute operations, including:

- Image management operations
- Basic support operations
- API version support operations
- Quotas management operations
- Basic server operations
- Volume management operations

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- NFVi - Network Functions Virtualization infrastructure
- SUT - System Under Test
- UUID - Universally Unique IDentifier
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM deployed with a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on VIM compute API operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

For brevity, the test cases in this test area are summarized together based on the operations they are testing.

All these test cases are included in the test case functest.tempest.osinterop of OVP test suite.

Test Descriptions

API Used and Reference

Servers: https://docs.openstack.org/api-ref/compute/

- create server
- delete server
- list servers
- start server
- stop server
- update server

- get server action
- set server metadata
- update server metadata
- rebuild server
- create image
- delete image
- create keypair
- delete keypair

Block storage: https://docs.openstack.org/api-ref/block-storage/

- create volume
- delete volume
- attach volume to server
- detach volume from server

Test Case 1 - Image operations within the Compute API

Test case specification

tempest.api.compute.images.test_images_oneserver.ImagesOneServerTestJSON.test_create_delete_image tempest.api.compute.images.test_images_oneserver.ImagesOneServerTestJSON.test_create_image_specify_multibyte_character_image_na

Test preconditions

- Compute server extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1 with an image IMG1 and wait for VM1 to reach 'ACTIVE' status
- Test action 2: Create a new server image IMG2 from VM1, specifying image name and image metadata. Wait for IMG2 to reach 'ACTIVE' status, and then delete IMG2
- **Test assertion 1:** Verify IMG2 is created with correct image name and image metadata; verify IMG1's 'minRam' equals to IMG2's 'minRam' and IMG2's 'minDisk' equals to IMG1's 'minDisk' or VM1's flavor disk size
- **Test assertion 2:** Verify IMG2 is deleted correctly
- Test action 3: Create another server IMG3 from VM1, specifying image name with a 3 byte utf-8 character
- **Test assertion 3:** Verify IMG3 is created correctly
- Test action 4: Delete VM1, IMG1 and IMG3

Pass / fail criteria

This test evaluates the Compute API ability of creating image from server, deleting image, creating server image with multi-byte character name. Specifically, the test verifies that:

- Compute server create image and delete image APIs work correctly.
- Compute server image can be created with multi-byte character name.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Action operation within the Compute API

Test case specification

tempest.api.compute.servers.test_instance_actions.InstanceActionsTestJSON.test_get_instance_action tempest.api.compute.servers.test_instance_actions.InstanceActionsTestJSON.test_list_instance_actions

Test preconditions

- Compute server extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1 and wait for VM1 to reach 'ACTIVE' status
- Test action 2: Get the action details ACT_DTL of VM1
- **Test assertion 1:** Verify ACT_DTL's 'instance_uuid' matches VM1's ID and ACT_DTL's 'action' matched 'create'
- Test action 3: Create a server VM2 and wait for VM2 to reach 'ACTIVE' status
- Test action 4: Delete server VM2 and wait for VM2 to reach termination
- Test action 5: Get the action list ACT_LST of VM2
- **Test assertion 2:** Verify ACT_LST's length is 2 and two actions are 'create' and 'delete'
- Test action 6: Delete VM1

Pass / fail criteria

This test evaluates the Compute API ability of getting the action details of a provided server and getting the action list of a deleted server. Specifically, the test verifies that:

- Get the details of the action in a specified server.

- List the actions that were performed on the specified server.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Generate, import and delete SSH keys within Compute services

Test case specification

tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_specify_keypair

Test preconditions

- Compute server extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1 and list all existing keypairs
- Test action 2: Create a server VM1 with KEYP1 and wait for VM1 to reach 'ACTIVE' status
- Test action 3: Show details of VM1
- **Test assertion 1:** Verify value of 'key_name' in the details equals to the name of KEYP1
- Test action 4: Delete KEYP1 and VM1

Pass / fail criteria

This test evaluates the Compute API ability of creating a keypair, listing keypairs and creating a server with a provided keypair. Specifically, the test verifies that:

- Compute create keypair and list keypair APIs work correctly.
- While creating a server, keypair can be specified.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - List supported versions of the Compute API

Test case specification

tempest.api.compute.test_versions.TestVersions.test_list_api_versions

Test preconditions

- Compute versions extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Get a List of versioned endpoints in the SUT
- **Test assertion 1:** Verify endpoints versions start at ‘v2.0’

Pass / fail criteria

This test evaluates the functionality of listing all available APIs to API consumers. Specifically, the test verifies that:

- Compute list API versions API works correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Quotas management in Compute API

Test case specification

tempest.api.compute.test_quotas.QuotasTestJSON.test_get_default_quotas tempest.api.compute.test_quotas.QuotasTestJSON.test_get_

Test preconditions

- Compute quotas extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Get the default quota set using the tenant ID
- **Test assertion 1:** Verify the default quota set ID matches tenant ID and the default quota set is complete

- Test action 2: Get the quota set using the tenant ID
- **Test assertion 2:** Verify the quota set ID matches tenant ID and the quota set is complete
- Test action 3: Get the quota set using the user ID
- **Test assertion 3:** Verify the quota set ID matches tenant ID and the quota set is complete

Pass / fail criteria

This test evaluates the functionality of getting quota set. Specifically, the test verifies that:

- User can get the default quota set for its tenant.
- User can get the quota set for its tenant.
- User can get the quota set using user ID.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 6 - Basic server operations in the Compute API

Test case specification

This test case evaluates the Compute API ability of basic server operations, including:

- Create a server with admin password
- Create a server with a name that already exists
- Create a server with a numeric name
- Create a server with a really long metadata
- Create a server with a name whose length exceeding 255 characters
- Create a server with an unknown flavor
- Create a server with an unknown image ID
- Create a server with an invalid network UUID
- Delete a server using a server ID that exceeds length limit
- Delete a server using a negative server ID
- Get a nonexistent server details
- Verify the instance host name is the same as the server name
- Create a server with an invalid access IPv6 address
- List all existent servers
- Filter the (detailed) list of servers by flavor, image, server name, server status or limit
- Lock a server and try server stop, unlock and retry
- Get and delete metadata from a server

- List and set metadata for a server
- Reboot, rebuild, stop and start a server
- Update a server’s access addresses and server name

The reference is,

tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_server_with_admin_password
tempest.api.compute.servers.test_servers.ServersTestJSON.test_create_with_existing_server_name
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_numeric_server_name
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_server_metadata_exceeds_length_limit
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_server_name_length_exceeds_256
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_with_invalid_flavor
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_with_invalid_image
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_create_with_invalid_network_uuid
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_delete_server_pass_id_exceeding_length_limit
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_delete_server_pass_negative_id
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_get_non_existent_server
tempest.api.compute.servers.test_create_server.ServersTestJSON.test_host_name_is_same_as_server_name
tempest.api.compute.servers.test_create_server.ServersTestManualDisk.test_host_name_is_same_as_server_name
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_invalid_ip_v6_address
tempest.api.compute.servers.test_create_server.ServersTestJSON.test_list_servers
tempest.api.compute.servers.test_create_server.ServersTestJSON.test_list_servers_with_detail
tempest.api.compute.servers.test_create_server.ServersTestManualDisk.test_list_servers
tempest.api.compute.servers.test_create_server.ServersTestManualDisk.test_list_servers_with_detail
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_detailed_filter_by_flavor
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_detailed_filter_by_image
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_detailed_filter_by_server_name
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_detailed_filter_by_server_status
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_detailed_limit_results
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filter_by_flavor
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filter_by_image
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filter_by_limit
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filter_by_server_name
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filter_by_active_status
tempest.api.compute.servers.test_list_server_filters.ListServerFiltersTestJSON.test_list_servers_filtered_by_name_wildcard
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_changes_since_future_date
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_changes_since_invalid_date
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_limits_greater_than_actual_c
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_limits_pass_negative_value
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_limits_pass_string
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_non_existing_flavor
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_non_existing_image
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_by_non_existing_server_name
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_detail_server_is_deleted
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_status_non_existing
tempest.api.compute.servers.test_list_servers_negative.ListServersNegativeTestJSON.test_list_servers_with_a_deleted_server
tempest.api.compute.servers.test_server_actions.ServerActionsTestJSON.test_lock_unlock_server
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_delete_server_metadata_item
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_get_server_metadata_item
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_list_server_metadata
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_set_server_metadata
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_set_server_metadata_item
tempest.api.compute.servers.test_server_metadata.ServerMetadataTestJSON.test_update_server_metadata
tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_server_name_blank

tempest.api.compute.servers.test_server_actions.ServerActionsTestJSON.test_reboot_server_hard tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_reboot_non_existent_server tempest.api.compute.servers.test_server_actions.ServerActionsTestJSON.test_rebuild_server tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_rebuild_deleted_server tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_rebuild_non_existent_server tempest.api.compute.servers.test_server_actions.ServerActionsTestJSON.test_stop_start_server tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_stop_non_existent_server tempest.api.compute.servers.test_servers.ServersTestJSON.test_update_access_server_address tempest.api.compute.servers.test_servers.ServersTestJSON.test_update_server_name tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_update_name_of_non_existent_server tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_update_server_name_length_exceeds_256 tempest.api.compute.servers.test_servers_negative.ServersNegativeTestJSON.test_update_server_set_empty_name tempest.api.compute.servers.test_create_server.ServersTestJSON.test_verify_created_server_vcpus tempest.api.compute.servers.test_create_server.ServersTestJSON.test_verify_server_details tempest.api.compute.servers.test_create_server.ServersTestManualDisk.test_verify_created_server_vcpus tempest.api.compute.servers.test_create_server.ServersTestManualDisk.test_verify_server_details tempest.api.compute.servers.test_delete_server.DeleteServersTestJSON.test_delete_active_server

Test preconditions

- Compute quotas extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1 with a admin password 'testpassword'
- **Test assertion 1:** Verify the password returned in the response equals to 'testpassword'
- Test action 2: Generate a VM name VM_NAME
- Test action 3: Create 2 servers VM2 and VM3 both with name VM_NAME
- **Test assertion 2:** Verify VM2's ID is not equal to VM3's ID, and VM2's name equal to VM3's name
- Test action 4: Create a server VM4 with a numeric name '12345'
- **Test assertion 3:** Verify creating VM4 failed
- Test action 5: Create a server VM5 with a long metadata '{'a': 'b' * 260}'
- **Test assertion 4:** Verify creating VM5 failed
- Test action 6: Create a server VM6 with name length exceeding 255 characters
- **Test assertion 5:** Verify creating VM6 failed
- Test action 7: Create a server VM7 with an unknown flavor '-1'
- **Test assertion 6:** Verify creating VM7 failed
- Test action 8: Create a server VM8 with an unknown image ID '-1'
- **Test assertion 7:** Verify creating VM8 failed
- Test action 9: Create a server VM9 with an invalid network UUID 'a-b-c-d-e-f-g-h-i-j'
- **Test assertion 8:** Verify creating VM9 failed

- Test action 10: Delete a server using a server ID that exceeds system's max integer limit
- **Test assertion 9:** Verify deleting server failed
- Test action 11: Delete a server using a server ID '-1'
- **Test assertion 10:** Verify deleting server failed
- Test action 12: Get a nonexistent server by using a random generated server ID
- **Test assertion 11:** Verify get server failed
- Test action 13: SSH into a provided server and get server's hostname
- **Test assertion 12:** Verify server's host name is the same as the server name
- Test action 14: SSH into a provided server and get server's hostname (manual disk configuration)
- **Test assertion 13:** Verify server's host name is the same as the server name (manual disk configuration)
- Test action 15: Create a server with an invalid access IPv6 address
- **Test assertion 14:** Verify creating server failed, a bad request error is returned in response
- Test action 16: List all existent servers
- **Test assertion 15:** Verify a provided server is in the server list
- Test action 17: List all existent servers in detail
- **Test assertion 16:** Verify a provided server is in the detailed server list
- Test action 18: List all existent servers (manual disk configuration)
- **Test assertion 17:** Verify a provided server is in the server list (manual disk configuration)
- Test action 19: List all existent servers in detail (manual disk configuration)
- **Test assertion 18:** Verify a provided server is in the detailed server list (manual disk configuration)
- Test action 20: List all existent servers in detail and filter the server list by flavor
- **Test assertion 19:** Verify the filtered server list is correct
- Test action 21: List all existent servers in detail and filter the server list by image
- **Test assertion 20:** Verify the filtered server list is correct
- Test action 22: List all existent servers in detail and filter the server list by server name
- **Test assertion 21:** Verify the filtered server list is correct
- Test action 23: List all existent servers in detail and filter the server list by server status
- **Test assertion 22:** Verify the filtered server list is correct
- Test action 24: List all existent servers in detail and filter the server list by display limit '1'
- **Test assertion 23:** Verify the length of filtered server list is 1
- Test action 25: List all existent servers and filter the server list by flavor
- **Test assertion 24:** Verify the filtered server list is correct
- Test action 26: List all existent servers and filter the server list by image
- **Test assertion 25:** Verify the filtered server list is correct
- Test action 27: List all existent servers and filter the server list by display limit '1'
- **Test assertion 26:** Verify the length of filtered server list is 1

- Test action 28: List all existent servers and filter the server list by server name
- **Test assertion 27:** Verify the filtered server list is correct
- Test action 29: List all existent servers and filter the server list by server status
- **Test assertion 28:** Verify the filtered server list is correct
- Test action 30: List all existent servers and filter the server list by server name wildcard
- **Test assertion 29:** Verify the filtered server list is correct
- Test action 31: List all existent servers and filter the server list by part of server name
- **Test assertion 30:** Verify the filtered server list is correct
- Test action 32: List all existent servers and filter the server list by a future change-since date
- **Test assertion 31:** Verify the filtered server list is empty
- Test action 33: List all existent servers and filter the server list by a invalid change-since date format
- **Test assertion 32:** Verify a bad request error is returned in the response
- Test action 34: List all existent servers and filter the server list by a display limit value greater than the length of the server list
- **Test assertion 33:** Verify the length of filtered server list equals to the length of server list
- Test action 35: List all existent servers and filter the server list by display limit '-1'
- **Test assertion 34:** Verify a bad request error is returned in the response
- Test action 36: List all existent servers and filter the server list by a string type limit value 'testing'
- **Test assertion 35:** Verify a bad request error is returned in the response
- Test action 37: List all existent servers and filter the server list by a nonexistent flavor
- **Test assertion 36:** Verify the filtered server list is empty
- Test action 38: List all existent servers and filter the server list by a nonexistent image
- **Test assertion 37:** Verify the filtered server list is empty
- Test action 39: List all existent servers and filter the server list by a nonexistent server name
- **Test assertion 38:** Verify the filtered server list is empty
- Test action 40: List all existent servers in detail and search the server list for a deleted server
- **Test assertion 39:** Verify the deleted server is not in the server list
- Test action 41: List all existent servers and filter the server list by a nonexistent server status
- **Test assertion 40:** Verify the filtered server list is empty
- Test action 42: List all existent servers in detail
- **Test assertion 41:** Verify a provided deleted server's id is not in the server list
- Test action 43: Lock a provided server VM10 and retrieve the server's status
- **Test assertion 42:** Verify VM10 is in 'ACTIVE' status
- Test action 44: Stop VM10
- **Test assertion 43:** Verify stop VM10 failed
- Test action 45: Unlock VM10 and stop VM10 again

- **Test assertion 44:** Verify VM10 is stopped and in ‘SHUTOFF’ status
- Test action 46: Start VM10
- **Test assertion 45:** Verify VM10 is in ‘ACTIVE’ status
- Test action 47: Delete metadata item ‘key1’ from a provided server
- **Test assertion 46:** Verify the metadata item is removed
- Test action 48: Get metadata item ‘key2’ from a provided server
- **Test assertion 47:** Verify the metadata item is correct
- Test action 49: List all metadata key/value pair for a provided server
- **Test assertion 48:** Verify all metadata are retrieved correctly
- Test action 50: Set metadata {‘meta2’: ‘data2’, ‘meta3’: ‘data3’} for a provided server
- **Test assertion 49:** Verify server’s metadata are replaced correctly
- Test action 51: Set metadata item nova’s value to ‘alt’ for a provided server
- **Test assertion 50:** Verify server’s metadata are set correctly
- Test action 52: Update metadata {‘key1’: ‘alt1’, ‘key3’: ‘value3’} for a provided server
- **Test assertion 51:** Verify server’s metadata are updated correctly
- Test action 53: Create a server with empty name parameter
- **Test assertion 52:** Verify create server failed
- Test action 54: Hard reboot a provided server
- **Test assertion 53:** Verify server is rebooted successfully
- Test action 55: Soft reboot a nonexistent server
- **Test assertion 54:** Verify reboot failed, an error is returned in the response
- Test action 56: Rebuild a provided server with new image, new server name and metadata
- **Test assertion 55:** Verify server is rebuilt successfully, server image, name and metadata are correct
- Test action 57: Create a server VM11
- Test action 58: Delete VM11 and wait for VM11 to reach termination
- Test action 59: Rebuild VM11 with another image
- **Test assertion 56:** Verify rebuild server failed, an error is returned in the response
- Test action 60: Rebuild a nonexistent server
- **Test assertion 57:** Verify rebuild server failed, an error is returned in the response
- Test action 61: Stop a provided server
- **Test assertion 58:** Verify server reaches ‘SHUTOFF’ status
- Test action 62: Start the stopped server
- **Test assertion 59:** Verify server reaches ‘ACTIVE’ status
- Test action 63: Stop a provided server
- **Test assertion 60:** Verify stop server failed, an error is returned in the response
- Test action 64: Create a server VM12 and wait it to reach ‘ACTIVE’ status

- Test action 65: Update VM12's IPv4 and IPv6 access addresses
- **Test assertion 61:** Verify VM12's access addresses have been updated correctly
- Test action 66: Create a server VM13 and wait it to reach 'ACTIVE' status
- Test action 67: Update VM13's server name with non-ASCII characters 'u00CDu00F1stu00E1u00F1cu00E9'
- **Test assertion 62:** Verify VM13's server name has been updated correctly
- Test action 68: Update the server name of a nonexistent server
- **Test assertion 63:** Verify update server name failed, an 'object not found' error is returned in the response
- Test action 69: Update a provided server's name with a 256-character long name
- **Test assertion 64:** Verify update server name failed, a bad request is returned in the response
- Test action 70: Update a provided server's server name with an empty string
- **Test assertion 65:** Verify update server name failed, a bad request error is returned in the response
- Test action 71: Get the number of vcpus of a provided server
- Test action 72: Get the number of vcpus stated by the server's flavor
- **Test assertion 66:** Verify that the number of vcpus reported by the server matches the amount stated by the server's flavor
- Test action 73: Create a server VM14
- **Test assertion 67:** Verify VM14's server attributes are set correctly
- Test action 74: Get the number of vcpus of a provided server (manual disk configuration)
- Test action 75: Get the number of vcpus stated by the server's flavor (manual disk configuration)
- **Test assertion 68:** Verify that the number of vcpus reported by the server matches the amount stated by the server's flavor (manual disk configuration)
- Test action 76: Create a server VM15 (manual disk configuration)
- **Test assertion 69:** Verify VM15's server attributes are set correctly (manual disk configuration)
- Test action 77: Create a server VM16 and then delete it when its status is 'ACTIVE'
- **Test assertion 70:** Verify VM16 is deleted successfully
- Test action 78: Delete all VMs created

Pass / fail criteria

This test evaluates the functionality of basic server operations. Specifically, the test verifies that:

- If an admin password is provided on server creation, the server's root password should be set to that password
- Create a server with a name that already exists is allowed
- Create a server with a numeric name or a name that exceeds the length limit is not allowed
- Create a server with a metadata that exceeds the length limit is not allowed
- Create a server with an invalid flavor, an invalid image or an invalid network UUID is not allowed
- Delete a server with a server ID that exceeds the length limit or a nonexistent server ID is not allowed
- Delete a server which status is 'ACTIVE' is allowed

- A provided server's host name is the same as the server name
- Create a server with an invalid IPv6 access address is not allowed
- A created server is in the (detailed) list of servers
- Filter the (detailed) list of servers by flavor, image, server name, server status, and display limit, respectively.
- Filter the list of servers by a future date
- Filter the list of servers by an invalid date format, a negative display limit or a string type display limit value is not allowed
- Filter the list of servers by a nonexistent flavor, image, server name or server status is not allowed
- Deleted servers are not in the list of servers
- Deleted servers do not show by default in list of servers
- Locked server is not allowed to be stopped by non-admin user
- Can get and delete metadata from servers
- Can list, set and update server metadata
- Create a server with name parameter empty is not allowed
- Hard reboot a server and the server should be power cycled
- Reboot, rebuild and stop a nonexistent server is not allowed
- Rebuild a server using the provided image and metadata
- Stop and restart a server
- A server's name and access addresses can be updated
- Update the name of a nonexistent server is not allowed
- Update name of a server to a name that exceeds the name length limit is not allowed
- Update name of a server to an empty string is not allowed
- The number of vcpus reported by the server matches the amount stated by the server's flavor
- The specified server attributes are set correctly

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 7 - Retrieve volume information through the Compute API

Test case specification

This test case evaluates the Compute API ability of attaching volume to a specific server and retrieve volume information, the reference is,

tempest.api.compute.volumes.test_attach_volume.AttachVolumeTestJSON.test_attach_detach_volume tempest.api.compute.volumes.test_attach_volume.AttachVolumeTestJSON.test_list_get_volume_attachments

Test preconditions

- Compute volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1 and a volume VOL1
- Test action 2: Attach VOL1 to VM1
- **Test assertion 1:** Stop VM1 successfully and wait VM1 to reach 'SHUTOFF' status
- **Test assertion 2:** Start VM1 successfully and wait VM1 to reach 'ACTIVE' status
- **Test assertion 3:** SSH into VM1 and verify VOL1 is in VM1's root disk devices
- Test action 3: Detach VOL1 from VM1
- **Test assertion 4:** Stop VM1 successfully and wait VM1 to reach 'SHUTOFF' status
- **Test assertion 5:** Start VM1 successfully and wait VM1 to reach 'ACTIVE' status
- **Test assertion 6:** SSH into VM1 and verify VOL1 is not in VM1's root disk devices
- Test action 4: Create a server VM2 and a volume VOL2
- Test action 5: Attach VOL2 to VM2
- Test action 6: List VM2's volume attachments
- **Test assertion 7:** Verify the length of the list is 1 and VOL2 attachment is in the list
- Test action 7: Retrieve VM2's volume information
- **Test assertion 8:** Verify volume information is correct
- Test action 8: Delete VM1, VM2, VOL1 and VOL2

Pass / fail criteria

This test evaluates the functionality of retrieving volume information. Specifically, the test verifies that:

- Stop and start a server with an attached volume work correctly.
- Retrieve a server's volume information correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 8 - List Compute service availability zones with the Compute API

Test case specification

This test case evaluates the Compute API ability of listing availability zones with a non admin user, the reference is,

tempest.api.compute.servers.test_availability_zone.AZV2TestJSON.test_get_availability_zone_list_with_non_admin_user

Test preconditions

- Compute volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List availability zones with a non admin user
- **Test assertion 1:** The list is not empty

Pass / fail criteria

This test evaluates the functionality of listing availability zones with a non admin user. Specifically, the test verifies that:

- Non admin users can list availability zones.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 9 - List Flavors within the Compute API

Test case specification

This test case evaluates the Compute API ability of listing flavors, the reference is,

tempest.api.compute.flavors.test_flavors.FlavorsV2TestJSON.test_list_flavors tempest.api.compute.flavors.test_flavors.FlavorsV2TestJS

Test preconditions

- Compute volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all flavors
- **Test assertion 1:** One given flavor is list in the all flavors' list
- Test action 2: List all flavors with details
- **Test assertion 2:** One given flavor is list in the all flavors' list

Pass / fail criteria

This test evaluates the functionality of listing flavors within the Compute API. Specifically, the test verifies that:

- Can list flavors with/without details within the Compute API.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 10 - Keypair operations within the Compute API

Test case specification

This test case evaluates the Compute API ability of creating keypair with type, the reference is,

tempest.api.compute.keypairs.test_keypairs_v22.KeyPairsV22TestJSON.test_keypairsv22_create_list_show_with_type

Test preconditions

- Compute server extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair with type 'x509' and a random name
- **Test assertion 1:** The keypair type received in the response body is equal to 'x509'
- Test action 2: Show the details of this created keypair
- **Test assertion 2:** The keypair type received in the response body is equal to 'x509'
- Test action 3: List all keypairs and find the one with the same name as given in test action 1
- **Test assertion 3:** The keypair type of this keypair is equal to 'x509'

Pass / fail criteria

This test evaluates the functionality of keypair operations within the Compute API. Specifically, the test verifies that:

- Can create keypair by specifying keypair type.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

VIM identity operations test specification

Scope

The VIM identity test area evaluates the ability of the system under test to support VIM identity operations. The tests in this area will evaluate API discovery operations within the Identity v3 API, auth operations within the Identity API.

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- NFVi - Network Functions Virtualisation infrastructure
- VIM - Virtual Infrastructure Manager

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on an Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on VIM identity operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test.

All these test cases are included in the test case functest.tempest.osinterop of OVP test suite.

Dependency Description

The VIM identity operations test cases are a part of the OpenStack interoperability tempest test cases. For Fraser based dovetail release, the OpenStack interoperability guidelines (version 2018.11) is adopted, which is valid for Mitaka, Newton, Ocata and Pike releases of Openstack.

Test Descriptions

Test Case 1 - API discovery operations within the Identity v3 API

Use case specification

tempest.api.identity.v3.test_api_discovery.TestApiDiscovery.test_api_version_resources tempest.api.identity.v3.test_api_discovery.TestApiDiscovery.test_api_media_types tempest.api.identity.v3.test_api_discovery.TestApiDiscovery.test_api_version_statuses

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Show the v3 identity api description, the test passes if keys 'id', 'links', 'media-types', 'status', 'updated' are all included in the description response message.
- Test action 2: Get the value of v3 identity api 'media-types', the test passes if api version 2 and version 3 are all included in the response.
- Test action 3: Show the v3 indentity api description, the test passes if 'current', 'stable', 'experimental', 'supported', 'deprecated' are all of the identity api 'status' values.

Pass / Fail criteria

This test case passes if all test action steps execute successfully and all assertions are affirmed. If any test steps fails to execute successfully or any of the assertions is not met, the test case fails.

Post conditions

None

Test Case 2 - Auth operations within the Identity API

Use case specification

tempest.api.identity.v3.test_tokens.TokensV3Test.test_create_token tempest.api.identity.v3.test_tokens.TokensV3Test.test_validate_toke

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Get the token by system credentials, the test passes if the returned token_id is not empty and is string type.
- Test action 2: Get the user_id in getting token response message, the test passes if it is equal to the user_id which is used to get token.
- Test action 3: Get the user_name in getting token response message, the test passes if it is equal to the user_name which is used to get token.
- Test action 4: Get the method in getting token response message, the test passes if it is equal to the password which is used to get token.
- Test action 5: Get the token by system credentials and show the token, the test passes if the response bodies of the get and show operations are the same.
- Test action 6: Get the user_id in showing token response message, the test passes if it is equal to the user_id which is used to get token.
- Test action 7: Get the username in showing token response message, the test passes if it is equal to the username which is used to get token.
- Test action 8: Delete this token by non-admin compute client, the test passes if it raises a NotFound exception.

Pass / Fail criteria

This test case passes if all test action steps execute successfully and all assertions are affirmed. If any test steps fails to execute successfully or any of the assertions is not met, the test case fails.

Post conditions

None

Test Case 3 - Catalog operations within the Identity API

Use case specification

tempest.api.identity.v3.test_catalog.IdentityCatalogTest.test_catalog_standardization

Test preconditions

None

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Show all catalogs by non-admin catalog client, the test passes if the catalog types getting in the show response message equal to the standard service values. Standard catalog types of ‘keystone’, ‘nova’, ‘glance’ and ‘swift’ should be ‘identity’, ‘compute’, ‘image’ and ‘object-store’ respectively.

Pass / Fail criteria

This test case passes if all test action steps execute successfully and all assertions are affirmed. If any test steps fails to execute successfully or any of the assertions is not met, the test case fails.

Post conditions

None

VIM image operations test specification

Scope

The VIM image test area evaluates the ability of the system under test to support VIM image operations. The test cases documented here are the Image API test cases in the Openstack Interop guideline 2018.11 as implemented by the Refstack client. These test cases will evaluate basic Openstack (as a VIM) image operations including image creation, image list, image update and image deletion capabilities using Glance v2 API.

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- CRUD - Create, Read, Update, and Delete
- NFVi - Network Functions Virtualization infrastructure
- VIM - Virtual Infrastructure Manager

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on VIM image operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test.

For brevity, the test cases in this test area are summarized together based on the operations they are testing.

All these test cases are included in the test case functest.tempest.osinterop of OVP test suite.

Test Descriptions

API Used and Reference

Images: https://docs.openstack.org/api-ref/image/v2/

- create image
- delete image
- show image details
- show images
- show image schema
- show images schema
- upload binary image data
- add image tag
- delete image tag

Test Case 1 - Image get tests using the Glance v2 API

Test case specification

tempest.api.image.v2.test_images.ListUserImagesTest.test_get_image_schema tempest.api.image.v2.test_images.ListUserImagesTest.test_get_images_schema tempest.api.image.v2.test_images_negative.ImagesNegativ tempest.api.image.v2.test_images_negative.ImagesNegativeTest.test_get_image_null_id tempest.api.image.v2.test_images_negative.ImagesNegativeTest.test_get_non_existent_image

Test preconditions

Glance is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 6 images and store their ids in a created images list.
- Test action 2: Use image v2 API to show image schema and check the body of the response.
- **Test assertion 1:** In the body of the response, the value of the key 'name' is 'image'.
- Test action 3: Use image v2 API to show images schema and check the body of the response.
- **Test assertion 2:** In the body of the response, the value of the key 'name' is 'images'.
- Test action 4: Create an image with name 'test', container_formats 'bare' and disk_formats 'raw'. Delete this image with its id and then try to show it with its id. Delete this deleted image again with its id and check the API's response code.
- **Test assertion 3:** The operations of showing and deleting a deleted image with its id both get 404 response code.

- Test action 5: Use a null image id to show a image and check the API's response code.
- **Test assertion 4:** The API's response code is 404.
- Test action 6: Generate a random uuid and use it as the image id to show the image.
- **Test assertion 5:** The API's response code is 404.
- Test action 7: Delete the 6 images with the stored ids. Show all images and check whether the 6 images' ids are not in the show list.
- **Test assertion 6:** The 6 images' ids are not found in the show list.

Pass / Fail criteria

The first two test cases evaluate the ability to use Glance v2 API to show image and images schema. The latter three test cases evaluate the ability to use Glance v2 API to show images with a deleted image id, a null image id and a non-existing image id. Specifically it verifies that:

- Glance image get API can show the image and images schema.
- Glance image get API can't show an image with a deleted image id.
- Glance image get API can't show an image with a null image id.
- Glance image get API can't show an image with a non-existing image id.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

Test Case 2 - CRUD image operations in Images API v2

Test case specification

tempest.api.image.v2.test_images.ListUserImagesTest.test_list_no_params

Test preconditions

Glance is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 6 images and store their ids in a created images list.
- Test action 2: List all images and check whether the ids listed are in the created images list.
- **Test assertion 1:** The ids get from the list images API are in the created images list.

Pass / Fail criteria

This test case evaluates the ability to use Glance v2 API to list images. Specifically it verifies that:

- Glance image API can show the images.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

Test Case 3 - Image list tests using the Glance v2 API

Test case specification

tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_container_format
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_disk_format
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_limit
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_min_max_size
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_size
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_status
tempest.api.image.v2.test_images.ListUserImagesTest.test_list_images_param_visibility

Test preconditions

Glance is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 6 images with a random size ranging from 1024 to 4096 and visibility 'private'; set their (container_format, disk_format) pair to be (ami, ami), (ami, ari), (ami, aki), (ami, vhd), (ami, vmdk) and (ami, raw); store their ids in a list and upload the binary images data.
- Test action 2: Use Glance v2 API to list all images whose container_format is 'ami' and store the response details in a list.
- **Test assertion 1:** The list is not empty and all the values of container_format in the list are 'ami'.
- Test action 3: Use Glance v2 API to list all images whose disk_format is 'raw' and store the response details in a list.
- **Test assertion 2:** The list is not empty and all the values of disk_format in the list are 'raw'.
- Test action 4: Use Glance v2 API to list one image by setting limit to be 1 and store the response details in a list.
- **Test assertion 3:** The length of the list is one.
- Test action 5: Use Glance v2 API to list images by setting size_min and size_max, and store the response images' sizes in a list. Choose the first image's size as the median, size_min is median-500 and size_max is median+500.

- **Test assertion 4:** All sizes in the list are no less than size_min and no more than size_max.
- Test action 6: Use Glance v2 API to show the first created image with its id and get its size from the response. Use Glance v2 API to list images whose size is equal to this size and store the response details in a list.
- **Test assertion 5:** All sizes of the images in the list are equal to the size used to list the images.
- Test action 7: Use Glance v2 API to list the images whose status is active and store the response details in a list.
- **Test assertion 6:** All status of images in the list are active.
- Test action 8: Use Glance v2 API to list the images whose visibility is private and store the response details in a list.
- **Test assertion 7:** All images' values of visibility in the list are private.
- Test action 9: Delete the 6 images with the stored ids. Show images and check whether the 6 ids are not in the show list.
- **Test assertion 8:** The stored 6 ids are not found in the show list.

Pass / Fail criteria

This test case evaluates the ability to use Glance v2 API to list images with different parameters. Specifically it verifies that:

- Glance image API can show the images with the container_format.
- Glance image API can show the images with the disk_format.
- Glance image API can show the images by setting a limit number.
- Glance image API can show the images with the size_min and size_max.
- Glance image API can show the images with the size.
- Glance image API can show the images with the status.
- Glance image API can show the images with the visibility type.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

Test Case 4 - Image update tests using the Glance v2 API

Test case specification

tempest.api.image.v2.test_images.BasicOperationsImagesTest.test_update_image tempest.api.image.v2.test_images_tags.ImagesTagsTest.test_update_delete_tags_for_image tempest.api.image.v2.test_images_tags_negative.ImagesTagsNegativeTest.test_update_tags_for_non_existing_image

Test preconditions

Glance is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create an image with container_formats 'ami', disk_formats 'ami' and visibility 'private' and store its id returned in the response. Check whether the status of the created image is 'queued'.
- **Test assertion 1:** The status of the created image is 'queued'.
- Test action 2: Use the stored image id to upload the binary image data and update this image's name. Show this image with the stored id. Check if the stored id and name used to update the image are equal to the id and name in the show list.
- **Test assertion 2:** The id and name returned in the show list are equal to the stored id and name used to update the image.
- Test action 3: Create an image with container_formats 'bare', disk_formats 'raw' and visibility 'private' and store its id returned in the response.
- Test action 4: Use the stored id to add a tag. Show the image with the stored id and check if the tag used to add is in the image's tags returned in the show list.
- **Test assertion 3:** The tag used to add into the image is in the show list.
- Test action 5: Use the stored id to delete this tag. Show the image with the stored id and check if the tag used to delete is not in the show list.
- **Test assertion 4:** The tag used to delete from the image is not in the show list.
- Test action 6: Generate a random uuid as the image id. Use the image id to add a tag into the image's tags.
- **Test assertion 5:** The API's response code is 404.
- Test action 7: Delete the images created in test action 1 and 3. Show the images and check whether the ids are not in the show list.
- **Test assertion 6:** The two ids are not found in the show list.

Pass / Fail criteria

This test case evaluates the ability to use Glance v2 API to update images with different parameters. Specifically it verifies that:

- Glance image API can update image's name with the existing image id.
- Glance image API can update image's tags with the existing image id.
- Glance image API can't update image's tags with a non-existing image id.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

Test Case 5 - Image deletion tests using the Glance v2 API

Test case specification

tempest.api.image.v2.test_images.BasicOperationsImagesTest.test_delete_image tempest.api.image.v2.test_images_negative.ImagesNegativeTest.test_delete_image_null_id tempest.api.image.v2.test_images_negative.ImagesNegativeTest.test_delete_non_existing_image tempest.api.image.v2.test_images_tags_negative.ImagesTagsNegativeTest.test_delete_non_existing_tag

Test preconditions

Glance is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create an image with container_formats 'ami', disk_formats 'ami' and visibility 'private'. Use the id of the created image to delete the image. List all images and check whether this id is in the list.
- **Test assertion 1:** The id of the created image is not found in the list of all images after the deletion operation.
- Test action 2: Delete images with a null id and check the API's response code.
- **Test assertion 2:** The API's response code is 404.
- Test action 3: Generate a random uuid and delete images with this uuid as image id. Check the API's response code.
- **Test assertion 3:** The API's response code is 404.
- Test action 4: Create an image with container_formats 'bare', disk_formats 'raw' and visibility 'private'. Delete this image's tag with the image id and a random tag Check the API's response code.
- **Test assertion 4:** The API's response code is 404.
- Test action 5: Delete the images created in test action 1 and 4. List all images and check whether the ids are in the list.
- **Test assertion 5:** The two ids are not found in the list.

Pass / Fail criteria

The first three test cases evaluate the ability to use Glance v2 API to delete images with an existing image id, a null image id and a non-existing image id. The last one evaluates the ability to use the API to delete a non-existing image tag. Specifically it verifies that:

- Glance image deletion API can delete the image with an existing id.
- Glance image deletion API can't delete an image with a null image id.
- Glance image deletion API can't delete an image with a non-existing image id.
- Glance image deletion API can't delete an image tag with a non-existing image tag.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

VIM network operations test specification

Scope

The VIM network test area evaluates the ability of the system under test to support VIM network operations. The test cases documented here are the network API test cases in the Openstack Interop guideline 2018.11 as implemented by the Refstack client. These test cases will evaluate basic Openstack (as a VIM) network operations including basic CRUD operations on L2 networks, L2 network ports and security groups.

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- CRUD - Create, Read, Update and Delete
- NFVi - Network Functions Virtualization infrastructure
- VIM - Virtual Infrastructure Manager

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on VIM network operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

For brevity, the test cases in this test area are summarized together based on the operations they are testing.

All these test cases are included in the test case functest.tempest.osinterop of OVP test suite.

Test Descriptions

API Used and Reference

Network: https://docs.openstack.org/api-ref/network/v2/index.html

- create network
- update network
- list networks
- show network details

- delete network
- create subnet
- update subnet
- list subnets
- show subnet details
- delete subnet
- create port
- bulk create ports
- update port
- list ports
- show port details
- delete port
- create security group
- update security group
- list security groups
- show security group
- delete security group
- create security group rule
- list security group rules
- show security group rule
- delete security group rule

Test Case 1 - Basic CRUD operations on L2 networks and L2 network ports

Test case specification

tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_with_allocation_pools
tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_with_dhcp_enabled
tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_with_gw tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_with_gw_and_allocation_pools tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_with_host_routes_and_dns_nameservers
tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_without_gateway
tempest.api.network.test_networks.NetworksTest.test_create_delete_subnet_all_attributes tempest.api.network.test_networks.NetworksTest.test_create_update_delete_network_subnet
tempest.api.network.test_networks.NetworksTest.test_delete_network_with_subnet tempest.api.network.test_networks.NetworksTest.test_external_network_visibility tempest.api.network.test_networks.NetworksTest.test_list_networks tempest.api.network.test_networks.NetworksTest.test_list_networks_fi
tempest.api.network.test_networks.NetworksTest.test_list_subnets tempest.api.network.test_networks.NetworksTest.test_list_subnets_fi
tempest.api.network.test_networks.NetworksTest.test_show_network tempest.api.network.test_networks.NetworksTest.test_show_netw
tempest.api.network.test_networks.NetworksTest.test_show_subnet tempest.api.network.test_networks.NetworksTest.test_show_subnet
tempest.api.network.test_networks.NetworksTest.test_update_subnet_gw_dns_host_routes_dhcp tempest.api.network.test_ports.PortsTestJSON.test_create_bulk_port tempest.api.network.test_ports.PortsTestJSON.test_create_port_in_all

tempest.api.network.test_ports.PortsTestJSON.test_create_update_delete_port tempest.api.network.test_ports.PortsTestJSON.test_list_ports tempest.api.network.test_ports.PortsTestJSON.test_list_ports_fields tempest.api.network.test_ports.PortsTestJSON.test_port_list_filter_by_router_id tempest.api.network.test_ports.PortsTestJSON.test_show_port tempest.api.network.test_ports.PortsTestJSON.test_show_port_fields

Test preconditions

Neutron is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a network and create a subnet of this network by setting allocation_pools, then check the details of the subnet and delete the subnet and network
- **Test assertion 1:** The allocation_pools returned in the response equals to the one used to create the subnet, and the network and subnet ids are not found after deletion
- Test action 2: Create a network and create a subnet of this network by setting enable_dhcp "True", then check the details of the subnet and delete the subnet and network
- **Test assertion 2:** The enable_dhcp returned in the response is "True" and the network and subnet ids are not found after deletion
- Test action 3: Create a network and create a subnet of this network by setting gateway_ip, then check the details of the subnet and delete the subnet and network
- **Test assertion 3:** The gateway_ip returned in the response equals to the one used to create the subnet, and the network and subnet ids are not found after deletion
- Test action 4: Create a network and create a subnet of this network by setting allocation_pools and gateway_ip, then check the details of the subnet and delete the subnet and network
- **Test assertion 4:** The allocation_pools and gateway_ip returned in the response equal to the ones used to create the subnet, and the network and subnet ids are not found after deletion
- Test action 5: Create a network and create a subnet of this network by setting host_routes and dns_nameservers, then check the details of the subnet and delete the subnet and network
- **Test assertion 5:** The host_routes and dns_nameservers returned in the response equal to the ones used to create the subnet, and the network and subnet ids are not found after deletion
- Test action 6: Create a network and create a subnet of this network without setting gateway_ip, then delete the subnet and network
- **Test assertion 6:** The network and subnet ids are not found after deletion
- Test action 7: Create a network and create a subnet of this network by setting enable_dhcp "true", gateway_ip, ip_version, cidr, host_routes, allocation_pools and dns_nameservers, then check the details of the subnet and delete the subnet and network
- **Test assertion 7:** The values returned in the response equal to the ones used to create the subnet, and the network and subnet ids are not found after deletion
- Test action 8: Create a network and update this network's name, then create a subnet and update this subnet's name, delete the subnet and network

- **Test assertion 8:** The network's status and subnet's status are both 'ACTIVE' after creation, their names equal to the new names used to update, and the network and subnet ids are not found after deletion
- Test action 9: Create a network and create a subnet of this network, then delete this network
- **Test assertion 9:** The subnet has also been deleted after deleting the network
- Test action 10: List all external networks, find the one with the same public_network_id as defined in tempest.conf and list its subnets
- **Test assertion 10:** The external network can be found, no internal network got by the list operation, if this external network is shared, the subnets list is not empty, otherwise, it should be empty
- Test action 11: Create a network and list all networks
- **Test assertion 11:** The network created is found in the list
- Test action 12: Create a network and list networks with the id and name of the created network
- **Test assertion 12:** The id and name of the list network equal to the created network's id and name
- Test action 13: Create a network and create a subnet of this network, then list all subnets
- **Test assertion 13:** The subnet created is found in the list
- Test action 14: Create a network and create a subnet of this network, then list subnets with the id and network_id of the created subnet
- **Test assertion 14:** The id and network_id of the list subnet equal to the created subnet
- Test action 15: Create a network and show network's details with the id of the created network
- **Test assertion 15:** The id and name returned in the response equal to the created network's id and name
- Test action 16: Create a network and just show network's id and name info with the id of the created network
- **Test assertion 16:** The keys returned in the response are only id and name, and the values of all the keys equal to network's id and name
- Test action 17: Create a network and create a subnet of this network, then show subnet's details with the id of the created subnet
- **Test assertion 17:** The id and cidr info returned in the response equal to the created subnet's id and cidr
- Test action 18: Create a network and create a subnet of this network, then show subnet's id and network_id info with the id of the created subnet
- **Test assertion 18:** The keys returned in the response are just id and network_id, and the values of all the keys equal to subnet's id and network_id
- Test action 19: Create a network and create a subnet of this network, then update subnet's name, host_routes, dns_nameservers and gateway_ip
- **Test assertion 19:** The name, host_routes, dns_nameservers and gateway_ip returned in the response equal to the values used to update the subnet
- Test action 20: Create 2 networks and bulk create 2 ports with the ids of the created networks
- **Test assertion 20:** The network_id of each port equals to the one used to create the port and the admin_state_up of each port is True
- Test action 21: Create a network and create a subnet of this network by setting allocation_pools, then create a port with the created network's id
- **Test assertion 21:** The ip_address of the created port is in the range of the allocation_pools

- Test action 22: Create a network and create a port with its id, then update the port's name and set its admin_state_up to be False
- **Test assertion 22:** The name returned in the response equals to the name used to update the port and the port's admin_state_up is False
- Test action 23: Create a network and create a port with its id, then list all ports
- **Test assertion 23:** The created port is found in the list
- Test action 24: Create a network and create a port with its id, then list ports with the id and mac_address of the created port
- **Test assertion 24:** The created port is found in the list
- Test action 25: Create a network and create a subnet, port with its id, create a router and add this port as this router's interface, then list ports with this router id
- **Test assertion 25:** The number of the ports list is 1, the port id and device_id getting with the list operation are the same as the ones got when creating them
- Test action 26: Create a network and create a port with its id, then show the port's details
- **Test assertion 26:** The key 'id' is in the details
- Test action 27: Create a network and create a port with its id, then show the port's id and mac_address info with the port's id
- **Test assertion 27:** The keys returned in the response are just id and mac_address, and the values of all the keys equal to port's id and mac_address

Pass / Fail criteria

These test cases evaluate the ability of basic CRUD operations on L2 networks and L2 network ports. Specifically it verifies that:

- Subnets can be created successfully by setting different parameters.
- Subnets can be updated after being created.
- Ports can be bulk created with network ids.
- Port's security group(s) can be updated after being created.
- Networks/subnets/ports can be listed with their ids and other parameters.
- All details or special fields' info of networks/subnets/ports can be shown with their ids.
- Networks/subnets/ports can be successfully deleted.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Basic CRUD operations on security groups

Test case specification

tempest.api.network.test_security_groups.SecGroupTest.test_create_list_update_show_delete_security_group tempest.api.network.test_security_groups.SecGroupTest.test_create_security_group_rule_with_additional_args tempest.api.network.test_security_groups.SecGroupTest.test_create_security_group_rule_with_icmp_type_code tempest.api.network.test_security_groups.SecGroupTest.test_create_security_group_rule_with_protocol_integer_value
tempest.api.network.test_security_groups.SecGroupTest.test_create_security_group_rule_with_remote_group_id
tempest.api.network.test_security_groups.SecGroupTest.test_create_security_group_rule_with_remote_ip_prefix
tempest.api.network.test_security_groups.SecGroupTest.test_create_show_delete_security_group_rule
tempest.api.network.test_security_groups.SecGroupTest.test_list_security_groups tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_additional_default_security_group_fails
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_duplicate_security_group_rule_fails
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_bad_ethertype
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_bad_protocol
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_bad_remote_ip_prefi
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_invalid_ports
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_non_existent_remote_
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_create_security_group_rule_with_non_existent_security
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_delete_non_existent_security_group
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_show_non_existent_security_group
tempest.api.network.test_security_groups_negative.NegativeSecGroupTest.test_show_non_existent_security_group_rule

Test preconditions

Neutron is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a security group SG1, list all security groups, update the name and description of SG1, show details of SG1 and delete SG1
- **Test assertion 1:** SG1 is in the list, the name and description of SG1 equal to the ones used to update it, the name and description of SG1 shown in the details equal to the ones used to update it, and SG1's id is not found after deletion
- Test action 2: Create a security group SG1, and create a rule with protocol 'tcp', port_range_min and port_range_max
- **Test assertion 2:** The values returned in the response equal to the ones used to create the rule
- Test action 3: Create a security group SG1, and create a rule with protocol 'icmp' and icmp_type_codes
- **Test assertion 3:** The values returned in the response equal to the ones used to create the rule
- Test action 4: Create a security group SG1, and create a rule with protocol '17'
- **Test assertion 4:** The values returned in the response equal to the ones used to create the rule
- Test action 5: Create a security group SG1, and create a rule with protocol 'udp', port_range_min, port_range_max and remote_group_id
- **Test assertion 5:** The values returned in the response equal to the ones used to create the rule

- Test action 6: Create a security group SG1, and create a rule with protocol 'tcp', port_range_min, port_range_max and remote_ip_prefix
- **Test assertion 6:** The values returned in the response equal to the ones used to create the rule
- Test action 7: Create a security group SG1, create 3 rules with protocol 'tcp', 'udp' and 'icmp' respectively, show details of each rule, list all rules and delete all rules
- **Test assertion 7:** The values in the shown details equal to the ones used to create the rule, all rules are found in the list, and all rules are not found after deletion
- Test action 8: List all security groups
- **Test assertion 8:** There is one default security group in the list
- Test action 9: Create a security group whose name is 'default'
- **Test assertion 9:** Failed to create this security group because of name conflict
- Test action 10: Create a security group SG1, create a rule with protocol 'tcp', port_range_min and port_range_max, and create another tcp rule with the same parameters
- **Test assertion 10:** Failed to create this security group rule because of duplicate protocol
- Test action 11: Create a security group SG1, and create a rule with ethertype 'bad_ethertype'
- **Test assertion 11:** Failed to create this security group rule because of bad ethertype
- Test action 12: Create a security group SG1, and create a rule with protocol 'bad_protocol_name'
- **Test assertion 12:** Failed to create this security group rule because of bad protocol
- Test action 13: Create a security group SG1, and create a rule with remote_ip_prefix '92.168.1./24', '192.168.1.1/33', 'bad_prefix' and '256' respectively
- **Test assertion 13:** Failed to create these security group rules because of bad remote_ip_prefix
- Test action 14: Create a security group SG1, and create a tcp rule with (port_range_min, port_range_max) (-16, 80), (80, 79), (80, 65536), (None, 6) and (-16, 65536) respectively
- **Test assertion 14:** Failed to create these security group rules because of bad ports
- Test action 15: Create a security group SG1, and create a tcp rule with remote_group_id 'bad_group_id' and a random uuid respectively
- **Test assertion 15:** Failed to create these security group rules because of nonexistent remote_group_id
- Test action 16: Create a security group SG1, and create a rule with a random uuid as security_group_id
- **Test assertion 16:** Failed to create these security group rules because of nonexistent security_group_id
- Test action 17: Generate a random uuid and use this id to delete security group
- **Test assertion 17:** Failed to delete security group because of nonexistent security_group_id
- Test action 18: Generate a random uuid and use this id to show security group
- **Test assertion 18:** Failed to show security group because of nonexistent id of security group
- Test action 19: Generate a random uuid and use this id to show security group rule
- **Test assertion 19:** Failed to show security group rule because of nonexistent id of security group rule

Pass / Fail criteria

These test cases evaluate the ability of Basic CRUD operations on security groups and security group rules. Specifically it verifies that:

- Security groups can be created, list, updated, shown and deleted.
- Security group rules can be created with different parameters, list, shown and deleted.
- Cannot create an additional default security group.
- Cannot create a duplicate security group rules.
- Cannot create security group rules with bad ethertype, protocol, remote_ip_prefix, ports, remote_group_id and security_group_id.
- Cannot show or delete security groups or security group rules with nonexistent ids.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - CRUD operations on subnet pools

Test case specification

tempest.api.network.test_subnetpools_extensions.SubnetPoolsTestJSON.test_create_list_show_update_delete_subnetpools

Test preconditions

Neutron is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a subnetpool SNP1 with a specific name and get the name from the response body
- **Test assertion 1:** The name got from the body is the same as the name used to create SNP1
- Test action 2: Show SNP1 and get the name from the response body
- **Test assertion 2:** The name got from the body is the same as the name used to create SNP1
- Test action 3: Update the name of SNP1 and get the new name from the response body
- **Test assertion 3:** The name got from the body is the same as the name used to update SNP1
- Test action 4: Delete SNP1

Pass / Fail criteria

These test cases evaluate the ability of Basic CRUD operations on subnetpools. Specifically it verifies that:

- Subnetpools can be created, updated, shown and deleted.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - CRUD operations on routers

Test case specification

tempest.api.network.test_routers.RoutersTest.test_add_multiple_router_interfaces tempest.api.network.test_routers.RoutersTest.test_add_remove_router_interface_with_port_id tempest.api.network.test_routers.RoutersTest.test_add_remove_router_interface_with_subnet_id tempest.api.network.test_routers.RoutersTest.test_create_show_list_update_delete_router tempest.api.network.test_routers.RoutersTest.test_update_delete_extra_route tempest.api.network.test_routers.RoutersTest.test_update_rou tempest.api.network.test_routers_negative.RoutersNegativeTest.test_add_router_interfaces_on_overlapping_subnets_returns_400 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_delete_non_existent_router_returns_404 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_router_add_gateway_invalid_network_returns_404 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_router_add_gateway_net_not_external_returns_400 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_router_remove_interface_in_use_returns_409 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_show_non_existent_router_returns_404 tempest.api.network.test_routers_negative.RoutersNegativeTest.test_update_non_existent_router_returns_404

Test preconditions

Neutron is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create 2 networks NET1 and NET2, create SubNet1 of NET1 and SubNet2 with cidr of NET2, create a router R1, add SubNet1 and SubNet2 to be R1's interfaces and get port details
- **Test assertion 1:** The device_id and subnet_id of the port equals to the router id and subnet id getting from the creating response body
- Test action 2: Create a network NET1, create SubNet1 of NET1, create P1 of NET1, create a router R1, add P1 to be its interface, store the response body B1, show P1 and store the response body B2 and remove the interface with port id
- **Test assertion 2:** B1 includes keys 'subnet_id' and 'port_id', 'device_id' of B2 equals to router id
- Test action 3: Create a network NET1, create SubNet1 of NET1, create a router R1, add SubNet1 to be its interface, store the response body B1, show port details with 'port_id' in B1

- **Test assertion 3:** B1 includes keys 'subnet_id' and 'port_id', 'device_id' equals to router id
- Test action 4: Create a router R1 with name, admin_state_up False and external_network_id Ext-Net, store the request body B1, show R1 with 'id' in B1, list all routers, update R1's name
- **Test assertion 4:** 'name', 'admin_state_up' and 'network_id' in B1 equal to the name, False and Ext-Net, 'name' in show details equals to 'name' in B1, 'network_id' in show details equals to Ext-Net, R1 is in the router list, the 'name' has been updated
- Test action 5: Create a router R1 with admin_state_up=True, create 4 networks and 4 subnets with different cidr, add these 4 subnets to R1 as its interfaces, update R1 by given routes and show R1, delete extra route of R1 and then show R1
- **Test assertion 5:** The number of routes queals to 4 and routes is empty after deletion
- Test action 6: Create a router R1, update 'admin_state_up' of R1 to be True
- **Test assertion 6:** 'admin_state_up' of R1 is False, 'admin_state_up' of R1 is True after updating
- Test action 7: Create 2 networks NET1 and NET2, create SubNet1 of NET1 and SubNet2 of NET2, create a router R1, add SubNet1 to be R1's interface and then trying to add SubNet2 to be R1's interface
- **Test assertion 7:** It raises a BadRequest exception when trying to add SubNet2
- Test action 8: Try to delete router with a random name
- **Test assertion 8:** It raises a NotFound exception when trying delete operation
- Test action 9: Create a router R1 and try to update it with wrong external_gateway_info
- **Test assertion 9:** It raises a NotFound exception when trying to update R1
- Test action 10: Create an internal network NET1, create SubNet1 with cidr of NET1, create a router R1 and try to update it with NET1 as external_gateway_info
- **Test assertion 10:** It raises a BadRequest exception when trying to update R1
- Test action 11: Create a network NET1, create SubNet1 of NET1, create a router R1, add SubNet1 to the interface of R1 and try to delete R1
- **Test assertion 11:** It raises a Conflict exception when trying to delete R1
- Test action 12: Try to show router with a random name
- **Test assertion 12:** It raises a NotFound exception when trying to show router
- Test action 13: Try to update router with a random name
- **Test assertion 13:** It raises a NotFound exception when trying to update router

Pass / Fail criteria

These test cases evaluate the ability of Basic CRUD operations on routers. Specifically it verifies that:

- Routers can be created, updated, shown and deleted.
- Can not show, update and delete non existent router.
- Can not remove interface in use.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - List versions within Network API

Test case specification

tempest.api.network.test_versions.NetworksApiDiscovery.test_api_version_resources

Test preconditions

Neutron is available.

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List network API versions
- **Test assertion 1:** The network API version is 'v2.0'

Pass / Fail criteria

This test case evaluates the ability of listing network API versions. Specifically it verifies that:

- The network API version is 'v2.0'.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

VIM volume operations test specification

Scope

The VIM volume operations test area evaluates the ability of the system under test to support VIM volume operations. The test cases documented here are the volume API test cases in the OpenStack Interop guideline 2018.11 as implemented by the RefStack client. These test cases will evaluate basic OpenStack (as a VIM) volume operations, including:

- Volume attach and detach operations
- Volume service availability zone operations
- Volume cloning operations
- Image copy-to-volume operations

- Volume creation and deletion operations
- Volume service extension listing
- Volume metadata operations
- Volume snapshot operations

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- NFVi - Network Functions Virtualization infrastructure
- SUT - System Under Test
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM deployed with a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on VIM volume API operations. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

For brevity, the test cases in this test area are summarized together based on the operations they are testing.

All these test cases are included in the test case functest.tempest.osinterop of OVP test suite.

Test Descriptions

API Used and Reference

Block storage: https://docs.openstack.org/api-ref/block-storage/

- create volume
- delete volume
- update volume
- attach volume to server
- detach volume from server
- create volume metadata
- update volume metadata
- delete volume metadata
- list volume

- create snapshot
- update snapshot
- delete snapshot

Test Case 1 - Volume service availability zone operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_availability_zone.AvailabilityZoneTestJSON.test_get_availability_zone_list

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all existent availability zones
- **Test assertion 1:** Verify the availability zone list length is greater than 0

Pass / Fail criteria

This test case evaluates the volume API ability of listing availability zones. Specifically, the test verifies that:

- Availability zones can be listed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Volume cloning operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_get.VolumesGetTest.test_volume_create_get_update_delete_as_clone

Test preconditions

- Volume extension API
- Cinder volume clones feature is enabled

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a volume VOL1
- Test action 2: Create a volume VOL2 from source volume VOL1 with a specific name and metadata
- Test action 2: Wait for VOL2 to reach 'available' status
- **Test assertion 1:** Verify the name of VOL2 is correct
- Test action 3: Retrieve VOL2's detail information
- **Test assertion 2:** Verify the retrieved volume name, ID and metadata are the same as VOL2
- **Test assertion 3:** Verify VOL2's bootable flag is 'False'
- Test action 4: Update the name of VOL2 with the original value
- Test action 5: Update the name of VOL2 with a new value
- **Test assertion 4:** Verify the name of VOL2 is updated successfully
- Test action 6: Create a volume VOL3 with no name specified and a description contains characters '@#$%^*'
- **Test assertion 5:** Verify VOL3 is created successfully
- Test action 7: Update the name of VOL3 and description with the original value
- **Test assertion 6:** Verify VOL3's bootable flag is 'False'

Pass / Fail criteria

This test case evaluates the volume API ability of creating a cloned volume from a source volume, getting cloned volume detail information and updating cloned volume attributes.

Specifically, the test verifies that:

- Cloned volume can be created from a source volume.
- Cloned volume detail information can be retrieved.
- Cloned volume detail information can be updated.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Image copy-to-volume operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_actions.VolumesActionsTest.test_volume_bootable tempest.api.volume.test_volumes_get.VolumesGetTest.test_volume_create_get_update_delete_from_image

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Set a provided volume VOL1's bootable flag to 'True'
- Test action 2: Retrieve VOL1's bootable flag
- **Test assertion 1:** Verify VOL1's bootable flag is 'True'
- Test action 3: Set a provided volume VOL1's bootable flag to 'False'
- Test action 4: Retrieve VOL1's bootable flag
- **Test assertion 2:** Verify VOL1's bootable flag is 'False'
- Test action 5: Create a bootable volume VOL2 from one image with a specific name and metadata
- Test action 6: Wait for VOL2 to reach 'available' status
- **Test assertion 3:** Verify the name of VOL2 name is correct
- Test action 7: Retrieve VOL2's information
- **Test assertion 4:** Verify the retrieved volume name, ID and metadata are the same as VOL2
- **Test assertion 5:** Verify VOL2's bootable flag is 'True'
- Test action 8: Update the name of VOL2 with the original value
- Test action 9: Update the name of VOL2 with a new value
- **Test assertion 6:** Verify the name of VOL2 is updated successfully
- Test action 10: Create a volume VOL3 with no name specified and a description contains characters '@#$%^*'
- **Test assertion 7:** Verify VOL3 is created successfully
- Test action 11: Update the name of VOL3 and description with the original value
- **Test assertion 8:** Verify VOL3's bootable flag is 'True'

Pass / Fail criteria

This test case evaluates the volume API ability of updating volume's bootable flag and creating a bootable volume from an image, getting bootable volume detail information and updating bootable volume.

Specifically, the test verifies that:

- Volume bootable flag can be set and retrieved.
- Bootable volume can be created from a source volume.
- Bootable volume detail information can be retrieved.
- Bootable volume detail information can be updated.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - Volume creation and deletion operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_get.VolumesGetTest.test_volume_create_get_update_delete tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_invalid_size tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_nonexistent_source_volid tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_nonexistent_volume_type tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_without_passing_size tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_size_negative tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_size_zero

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a volume VOL1 with a specific name and metadata
- Test action 2: Wait for VOL1 to reach 'available' status
- **Test assertion 1:** Verify the name of VOL1 is correct
- Test action 3: Retrieve VOL1's information
- **Test assertion 2:** Verify the retrieved volume name, ID and metadata are the same as VOL1
- **Test assertion 3:** Verify VOL1's bootable flag is 'False'
- Test action 4: Update the name of VOL1 with the original value
- Test action 5: Update the name of VOL1 with a new value
- **Test assertion 4:** Verify the name of VOL1 is updated successfully
- Test action 6: Create a volume VOL2 with no name specified and a description contains characters '@#$%^*'
- **Test assertion 5:** Verify VOL2 is created successfully
- Test action 7: Update the name of VOL2 and description with the original value
- **Test assertion 6:** Verify VOL2's bootable flag is 'False'
- Test action 8: Create a volume with an invalid size '#$%'
- **Test assertion 7:** Verify create volume failed, a bad request error is returned in the response
- Test action 9: Create a volume with a nonexistent source volume
- **Test assertion 8:** Verify create volume failed, a 'Not Found' error is returned in the response

- Test action 10: Create a volume with a nonexistent volume type
- **Test assertion 9:** Verify create volume failed, a 'Not Found' error is returned in the response
- Test action 11: Create a volume without passing a volume size
- **Test assertion 10:** Verify create volume failed, a bad request error is returned in the response
- Test action 12: Create a volume with a negative volume size
- **Test assertion 11:** Verify create volume failed, a bad request error is returned in the response
- Test action 13: Create a volume with volume size '0'
- **Test assertion 12:** Verify create volume failed, a bad request error is returned in the response

Pass / Fail criteria

This test case evaluates the volume API ability of creating a volume, getting volume detail information and updating volume, the reference is, Specifically, the test verifies that:

- Volume can be created from a source volume.
- Volume detail information can be retrieved/updated.
- Create a volume with an invalid size is not allowed.
- Create a volume with a nonexistent source volume or volume type is not allowed.
- Create a volume without passing a volume size is not allowed.
- Create a volume with a negative volume size is not allowed.
- Create a volume with volume size '0' is not allowed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Volume service extension listing operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_extensions.ExtensionsTestJSON.test_list_extensions

Test preconditions

- Volume extension API
- At least one Cinder extension is configured

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all cinder service extensions
- **Test assertion 1:** Verify all extensions are list in the extension list

Pass / Fail criteria

This test case evaluates the volume API ability of listing all existent volume service extensions.

- Cinder service extensions can be listed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 6 - Volume GET operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_get_invalid_volume_id tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_get_volume_without_passing_volume_id tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_volume_get_nonexistent_volume_id

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Retrieve a volume with an invalid volume ID
- **Test assertion 1:** Verify retrieve volume failed, a 'Not Found' error is returned in the response
- Test action 2: Retrieve a volume with an empty volume ID
- **Test assertion 2:** Verify retrieve volume failed, a 'Not Found' error is returned in the response
- Test action 3: Retrieve a volume with a nonexistent volume ID
- **Test assertion 3:** Verify retrieve volume failed, a 'Not Found' error is returned in the response

Pass / Fail criteria

This test case evaluates the volume API ability of getting volumes. Specifically, the test verifies that:

- Get a volume with an invalid/an empty/a nonexistent volume ID is not allowed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 7 - Volume listing operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_by_name
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_details_by_name
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_param_display_name_and_status
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_with_detail_param_display_name_and_status
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_with_detail_param_metadata
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_with_details
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_with_param_metadata
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volumes_list_by_availability_zone
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volumes_list_by_status
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volumes_list_details_by_availability_zone
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volumes_list_details_by_status
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_list_volumes_detail_with_invalid_status
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_list_volumes_detail_with_nonexistent_name
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_list_volumes_with_invalid_status
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_list_volumes_with_nonexistent_name
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_details_pagination
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_details_with_multiple_params
tempest.api.volume.test_volumes_list.VolumesListTestJSON.test_volume_list_pagination

Test preconditions

- Volume extension API
- The backing file for the volume group that Nova uses has space for at least 3 1G volumes

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List all existent volumes
- **Test assertion 1:** Verify the volume list is complete
- Test action 2: List existent volumes and filter the volume list by volume name

- **Test assertion 2:** Verify the length of filtered volume list is 1 and the retrieved volume is correct
- Test action 3: List existent volumes in detail and filter the volume list by volume name
- **Test assertion 3:** Verify the length of filtered volume list is 1 and the retrieved volume is correct
- Test action 4: List existent volumes and filter the volume list by volume name and status 'available'
- **Test assertion 4:** Verify the name and status parameters of the fetched volume are correct
- Test action 5: List existent volumes in detail and filter the volume list by volume name and status 'available'
- **Test assertion 5:** Verify the name and status parameters of the fetched volume are correct
- Test action 6: List all existent volumes in detail and filter the volume list by volume metadata
- **Test assertion 6:** Verify the metadata parameter of the fetched volume is correct
- Test action 7: List all existent volumes in detail
- **Test assertion 7:** Verify the volume list is complete
- Test action 8: List all existent volumes and filter the volume list by volume metadata
- **Test assertion 8:** Verify the metadata parameter of the fetched volume is correct
- Test action 9: List existent volumes and filter the volume list by availability zone
- **Test assertion 9:** Verify the availability zone parameter of the fetched volume is correct
- Test action 10: List all existent volumes and filter the volume list by volume status 'available'
- **Test assertion 10:** Verify the status parameter of the fetched volume is correct
- Test action 11: List existent volumes in detail and filter the volume list by availability zone
- **Test assertion 11:** Verify the availability zone parameter of the fetched volume is correct
- Test action 12: List all existent volumes in detail and filter the volume list by volume status 'available'
- **Test assertion 12:** Verify the status parameter of the fetched volume is correct
- Test action 13: List all existent volumes in detail and filter the volume list by an invalid volume status 'null'
- **Test assertion 13:** Verify the filtered volume list is empty
- Test action 14: List all existent volumes in detail and filter the volume list by a non-existent volume name
- **Test assertion 14:** Verify the filtered volume list is empty
- Test action 15: List all existent volumes and filter the volume list by an invalid volume status 'null'
- **Test assertion 15:** Verify the filtered volume list is empty
- Test action 16: List all existent volumes and filter the volume list by a non-existent volume name
- **Test assertion 16:** Verify the filtered volume list is empty
- Test action 17: List all existent volumes in detail and paginate the volume list by desired volume IDs
- **Test assertion 17:** Verify only the desired volumes are listed in the filtered volume list
- Test action 18: List all existent volumes in detail and filter the volume list by volume status 'available' and display limit '2'
- Test action 19: Sort the filtered volume list by IDs in ascending order
- **Test assertion 18:** Verify the length of filtered volume list is 2
- **Test assertion 19:** Verify the status of retrieved volumes is correct

- **Test assertion 20:** Verify the filtered volume list is sorted correctly
- Test action 20: List all existent volumes in detail and filter the volume list by volume status 'available' and display limit '2'
- Test action 21: Sort the filtered volume list by IDs in descending order
- **Test assertion 21:** Verify the length of filtered volume list is 2
- **Test assertion 22:** Verify the status of retrieved volumes is correct
- **Test assertion 23:** Verify the filtered volume list is sorted correctly
- Test action 22: List all existent volumes and paginate the volume list by desired volume IDs
- **Test assertion 24:** Verify only the desired volumes are listed in the filtered volume list

Pass / Fail criteria

This test case evaluates the volume API ability of getting a list of volumes and filtering the volume list. Specifically, the test verifies that:

- Get a list of volumes (in detail) successful.
- Get a list of volumes (in detail) and filter volumes by name/status/metadata/availability zone successful.
- Volume list pagination functionality is working.
- Get a list of volumes in detail using combined condition successful.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 8 - Volume metadata operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volume_metadata.VolumesMetadataTest.test_crud_volume_metadata tempest.api.volume.test_volume_metadata.VolumesMetadataTest.test_update_show_volume_metadata_item

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create metadata for a provided volume VOL1
- Test action 2: Get the metadata of VOL1

- **Test assertion 1:** Verify the metadata of VOL1 is correct
- Test action 3: Update the metadata of VOL1
- **Test assertion 2:** Verify the metadata of VOL1 is updated
- Test action 4: Delete one metadata item 'key1' of VOL1
- **Test assertion 3:** Verify the metadata item 'key1' is deleted
- Test action 5: Create metadata for a provided volume VOL2
- **Test assertion 4:** Verify the metadata of VOL2 is correct
- Test action 6: Update one metadata item 'key3' of VOL2
- **Test assertion 5:** Verify the metadata of VOL2 is updated

Pass / Fail criteria

This test case evaluates the volume API ability of creating metadata for a volume, getting the metadata of a volume, updating volume metadata and deleting a metadata item of a volume. Specifically, the test verifies that:

- Create metadata for volume successfully.
- Get metadata of volume successfully.
- Update volume metadata and metadata item successfully.
- Delete metadata item of a volume successfully.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 9 - Verification of read-only status on volumes with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_actions.VolumesActionsTest.test_volume_readonly_update

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Update a provided volume VOL1's read-only access mode to 'True'
- **Test assertion 1:** Verify VOL1 is in read-only access mode
- Test action 2: Update a provided volume VOL1's read-only access mode to 'False'

- **Test assertion 2:** Verify VOL1 is not in read-only access mode

Pass / Fail criteria

This test case evaluates the volume API ability of setting and updating volume read-only access mode. Specifically, the test verifies that:

- Volume read-only access mode can be set and updated.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 10 - Volume snapshot creation/deletion operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_snapshot_metadata.SnapshotMetadataTestJSON.test_crud_snapshot_metadata tempest.api.volume.test_snapshot_metadata.SnapshotMetadataTestJSON.test_update_show_snapshot_metadata_item tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_create_volume_with_nonexistent_snapshot_id tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_delete_invalid_volume_id tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_delete_volume_without_passing_volume_id tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_volume_delete_nonexistent_volume_id tempest.api.volume.test_volumes_snapshots.VolumesSnapshotTestJSON.test_snapshot_create_get_list_update_delete tempest.api.volume.test_volumes_snapshots.VolumesSnapshotTestJSON.test_volume_from_snapshot tempest.api.volume.test_volumes_snapshots_list.VolumesSnapshotListTestJSON.test_snapshots_list_details_with_params tempest.api.volume.test_volumes_snapshots_list.VolumesSnapshotListTestJSON.test_snapshots_list_with_params tempest.api.volume.test_volumes_snapshots_negative.VolumesSnapshotNegativeTestJSON.test_create_snapshot_with_nonexistent_vol tempest.api.volume.test_volumes_snapshots_negative.VolumesSnapshotNegativeTestJSON.test_create_snapshot_without_passing_volu

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create metadata for a provided snapshot SNAP1
- Test action 2: Get the metadata of SNAP1
- **Test assertion 1:** Verify the metadata of SNAP1 is correct
- Test action 3: Update the metadata of SNAP1
- **Test assertion 2:** Verify the metadata of SNAP1 is updated
- Test action 4: Delete one metadata item ‘key3’ of SNAP1

- **Test assertion 3:** Verify the metadata item 'key3' is deleted
- Test action 5: Create metadata for a provided snapshot SNAP2
- **Test assertion 4:** Verify the metadata of SNAP2 is correct
- Test action 6: Update one metadata item 'key3' of SNAP2
- **Test assertion 5:** Verify the metadata of SNAP2 is updated
- Test action 7: Create a volume with a nonexistent snapshot
- **Test assertion 6:** Verify create volume failed, a 'Not Found' error is returned in the response
- Test action 8: Delete a volume with an invalid volume ID
- **Test assertion 7:** Verify delete volume failed, a 'Not Found' error is returned in the response
- Test action 9: Delete a volume with an empty volume ID
- **Test assertion 8:** Verify delete volume failed, a 'Not Found' error is returned in the response
- Test action 10: Delete a volume with a nonexistent volume ID
- **Test assertion 9:** Verify delete volume failed, a 'Not Found' error is returned in the response
- Test action 11: Create a snapshot SNAP2 from a provided volume VOL1
- Test action 12: Retrieve SNAP2's detail information
- **Test assertion 10:** Verify SNAP2 is created from VOL1
- Test action 13: Update the name and description of SNAP2
- **Test assertion 11:** Verify the name and description of SNAP2 are updated in the response body of update snapshot API
- Test action 14: Retrieve SNAP2's detail information
- **Test assertion 12:** Verify the name and description of SNAP2 are correct
- Test action 15: Delete SNAP2
- Test action 16: Create a volume VOL2 with a volume size
- Test action 17: Create a snapshot SNAP3 from VOL2
- Test action 18: Create a volume VOL3 from SNAP3 with a bigger volume size
- Test action 19: Retrieve VOL3's detail information
- **Test assertion 13:** Verify volume size and source snapshot of VOL3 are correct
- Test action 20: List all snapshots in detail and filter the snapshot list by name
- **Test assertion 14:** Verify the filtered snapshot list is correct
- Test action 21: List all snapshots in detail and filter the snapshot list by status
- **Test assertion 15:** Verify the filtered snapshot list is correct
- Test action 22: List all snapshots in detail and filter the snapshot list by name and status
- **Test assertion 16:** Verify the filtered snapshot list is correct
- Test action 23: List all snapshots and filter the snapshot list by name
- **Test assertion 17:** Verify the filtered snapshot list is correct
- Test action 24: List all snapshots and filter the snapshot list by status

- **Test assertion 18:** Verify the filtered snapshot list is correct
- Test action 25: List all snapshots and filter the snapshot list by name and status
- **Test assertion 19:** Verify the filtered snapshot list is correct
- Test action 26: Create a snapshot from a nonexistent volume by using an invalid volume ID
- **Test assertion 20:** Verify create snapshot failed, a 'Not Found' error is returned in the response
- Test action 27: Create a snapshot from a volume by using an empty volume ID
- **Test assertion 21:** Verify create snapshot failed, a 'Not Found' error is returned in the response

Pass / Fail criteria

This test case evaluates the volume API ability of managing snapshot and snapshot metadata. Specifically, the test verifies that:

- Create metadata for snapshot successfully.
- Get metadata of snapshot successfully.
- Update snapshot metadata and metadata item successfully.
- Delete metadata item of a snapshot successfully.
- Create a volume from a nonexistent snapshot is not allowed.
- Delete a volume using an invalid volume ID is not allowed.
- Delete a volume without passing the volume ID is not allowed.
- Delete a non-existent volume is not allowed.
- Create snapshot successfully.
- Get snapshot's detail information successfully.
- Update snapshot attributes successfully.
- Delete snapshot successfully.
- Creates a volume and a snapshot passing a size different from the source successfully.
- List snapshot details by display_name and status filters successfully.
- Create a snapshot from a nonexistent volume is not allowed.
- Create a snapshot from a volume without passing the volume ID is not allowed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 11 - Volume update operations with the Cinder v2 or v3 API

Test case specification

tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_update_volume_with_empty_volume_id
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_update_volume_with_invalid_volume_id
tempest.api.volume.test_volumes_negative.VolumesNegativeTest.test_update_volume_with_nonexistent_volume_id

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Update a volume by using an empty volume ID
- **Test assertion 1:** Verify update volume failed, a 'Not Found' error is returned in the response
- Test action 2: Update a volume by using an invalid volume ID
- **Test assertion 2:** Verify update volume failed, a 'Not Found' error is returned in the response
- Test action 3: Update a non-existent volume by using a random generated volume ID
- **Test assertion 3:** Verify update volume failed, a 'Not Found' error is returned in the response

Pass / Fail criteria

This test case evaluates the volume API ability of updating volume attributes. Specifically, the test verifies that:

- Update a volume without passing the volume ID is not allowed.
- Update a volume using an invalid volume ID is not allowed.
- Update a non-existent volume is not allowed.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 12 - Volume list version operation with the Cinder v3 API

Test case specification

tempest.api.volume.test_versions.VersionsTest.test_list_versions

Test preconditions

- Volume API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: List volume API versions
- **Test assertion 1:** The list versions operation is successful executed

Pass / Fail criteria

This test case evaluates the volume API ability of listing volume API versions. Specifically, the test verifies that:

- Successfully listing volume API versions.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.14 Neutron Trunk Port Tempest Tests

Scope

This test area evaluates the ability of a system under test to support Neutron trunk ports. The test area specifically validates port and sub-port API CRUD operations, by means of both positive and negative tests.

References

- OpenStack API reference

System Under Test (SUT)

The system under test is assumed to be the NFVI and VIM deployed on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

- Neutron Trunk API tests
- Neutron Trunk API trunk details
- Neutron Trunk API negative tests

Trunk port and sub-port CRUD operations:

These tests cover the CRUD (Create, Read, Update, Delete) life-cycle operations of trunk ports and subports.

Implementation: TrunkTestJSON

- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_add_subports
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_create_show_delete_trunk
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_create_trunk_empty_subports_list
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_create_trunk_subports_not_specified
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_delete_trunk_with_subport_is_allowed
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_get_subports
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_list_trunks
- neutron_tempest_plugin.api.test_trunk.TrunkTestJSON.test_remove_subport

API for listing query results:

These tests verify that listing operations of trunk port objects work. This functionality is required for CLI and UI operations.

Implementation: TrunksSearchCriteriaTest

- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_no_pagination_limit_0
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_pagination
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_pagination_page_reverse_asc
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_pagination_page_reverse_desc
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_pagination_with_marker
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_sorts_asc
- neutron_tempest_plugin.api.test_trunk.TrunksSearchCriteriaTest.test_list_sorts_desc

Query trunk port details:

These tests validate that all attributes of trunk port objects can be queried.

Implementation: TestTrunkDetailsJSON

- neutron_tempest_plugin.api.test_trunk_details.TestTrunkDetailsJSON.test_port_resource_empty_trunk_details
- neutron_tempest_plugin.api.test_trunk_details.TestTrunkDetailsJSON.test_port_resource_trunk_details_no_subports
- neutron_tempest_plugin.api.test_trunk_details.TestTrunkDetailsJSON.test_port_resource_trunk_details_with_subport

Negative tests:

These group of tests comprise negative tests which verify that invalid operations are handled correctly by the system under test.

Implementation: TrunkTestNegative

- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_add_subport_duplicate_segmentation_details
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_add_subport_passing_dict
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_add_subport_port_id_disabled_trunk
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_add_subport_port_id_uses_parent_port_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_subport_missing_segmentation_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_subport_nonexistent_port_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_subport_nonexistent_trunk
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_duplicate_subport_segmentation_ids

- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_nonexistent_port_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_nonexistent_subport_port_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_with_subport_missing_port_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_with_subport_missing_segmentation_id
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_create_trunk_with_subport_missing_segmentation_type
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_delete_port_in_use_by_subport
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_delete_port_in_use_by_trunk
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_delete_trunk_disabled_trunk
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_remove_subport_not_found
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_remove_subport_passing_dict
- neutron_tempest_plugin.api.test_trunk_negative.TrunkTestJSON.test_remove_subport_port_id_disabled_trunk

Scenario tests (tests covering more than one functionality):

In contrast to the API tests above, these tests validate more than one specific API capability. Instead they verify that a simple scenario (example workflow) functions as intended. To this end, they boot up two VMs with trunk ports and sub ports and verify connectivity between those VMs.

Implementation: TrunkTest

- neutron_tempest_plugin.scenario.test_trunk.TrunkTest.test_trunk_subport_lifecycle

8.1.15 Common virtual machine life cycle events test specification

Scope

The common virtual machine life cycle events test area evaluates the ability of the system under test to behave correctly after common virtual machine life cycle events. The tests in this test area will evaluate:

- Stop/Start a server
- Reboot a server
- Rebuild a server
- Pause/Unpause a server
- Suspend/Resume a server
- Resize a server
- Resizing a volume-backed server
- Sequence suspend resume
- Shelve/Unshelve a server
- Cold migrate a server
- Live migrate a server

References

- iSCSI
  - https://docs.openstack.org/liberty/config-reference/content/config-iscsi-storage.html

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- API - Application Programming Interface
- NFVi - Network Functions Virtualization infrastructure
- VIM - Virtual Infrastructure Manager
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured based on common virtual machine life cycle events. Each test case is able to run independently, i.e. irrelevant of the state created by a previous test. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

All these test cases are included in the test case functest.tempest.vm_lifecycle of OVP test suite.

Test Descriptions

API Used and Reference

Block storage: https://docs.openstack.org/api-ref/block-storage/

- create volume
- delete volume
- attach volume to server
- detach volume from server

Security Groups: https://docs.openstack.org/api-ref/network/v2/index.html#security-groups-security-groups

- create security group
- delete security group

Networks: https://docs.openstack.org/api-ref/network/v2/index.html#networks

- create network
- delete network

Routers and interface: https://docs.openstack.org/api-ref/network/v2/index.html#routers-routers

- create router

- delete router
- add interface to router

Subnets: https://docs.openstack.org/api-ref/network/v2/index.html#subnets

- create subnet
- delete subnet

Servers: https://docs.openstack.org/api-ref/compute/

- create keypair
- create server
- show server
- delete server
- add/assign floating IP
- resize server
- revert resized server
- confirm resized server
- pause server
- unpause server
- start server
- stop server
- reboot server
- rebuild server
- suspend server
- resume suspended server
- shelve server
- unshelve server
- migrate server
- live-migrate server

Ports: https://docs.openstack.org/api-ref/network/v2/index.html#ports

- create port
- delete port

Floating IPs: https://docs.openstack.org/api-ref/network/v2/index.html#floating-ips-floatingips

- create floating IP
- delete floating IP

Availability zone: https://docs.openstack.org/api-ref/compute/

- get availability zone

Test Case 1 - Minimum basic scenario

Test case specification

tempest.scenario.test_minimum_basic.TestMinimumBasicScenario.test_minimum_basic_scenario

Test preconditions

- Nova, cinder, glance, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create an image IMG1
- Test action 2: Create a keypair KEYP1
- Test action 3: Create a server VM1 with IMG1 and KEYP1
- **Test assertion 1:** Verify VM1 is created successfully
- Test action 4: Create a volume VOL1
- **Test assertion 2:** Verify VOL1 is created successfully
- Test action 5: Attach VOL1 to VM1
- **Test assertion 3:** Verify VOL1's status has been updated after attached to VM1
- Test action 6: Create a floating IP FIP1 and assign FIP1 to VM1
- **Test assertion 4:** Verify VM1's addresses have been refreshed after associating FIP1
- Test action 7: Create and add security group SG1 to VM1
- **Test assertion 5:** Verify can SSH to VM1 via FIP1
- Test action 8: Reboot VM1
- **Test assertion 6:** Verify can SSH to VM1 via FIP1
- **Test assertion 7:** Verify VM1's disk count equals to 1
- Test action 9: Delete the floating IP FIP1 from VM1
- **Test assertion 8:** Verify VM1's addresses have been refreshed after disassociating FIP1
- Test action 10: Delete SG1, IMG1, KEYP1, VOL1, VM1 and FIP1

Pass / Fail criteria

This test evaluates a minimum basic scenario. Specifically, the test verifies that:

- The server can be connected before reboot.
- The server can be connected after reboot.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Cold migration

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_cold_migration

Test preconditions

- At least 2 compute nodes
- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Get VM1's host info SRC_HOST
- Test action 5: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 1:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 6: Cold migrate VM1
- Test action 7: Wait for VM1 to reach 'VERIFY_RESIZE' status
- Test action 8: Confirm resize VM1
- Test action 9: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 2:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 10: Get VM1's host info DST_HOST
- **Test assertion 3:** Verify SRC_HOST does not equal to DST_HOST
- Test action 11: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to cold migrate VMs. Specifically, the test verifies that:

- Servers can be cold migrated from one compute node to another computer node.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 3 - Pause and unpause server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_pause_unpause

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Pause VM1
- Test action 5: Wait for VM1 to reach 'PAUSED' status
- **Test assertion 1:** Verify FIP1 status is 'ACTIVE'
- **Test assertion 2:** Verify ping FIP1 failed and SSH to VM1 via FIP1 failed
- Test action 6: Unpause VM1
- Test action 7: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 3:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 8: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to pause and unpause VMs. Specifically, the test verifies that:

- When paused, servers cannot be reached.
- When unpaused, servers can recover its reachability.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 4 - Reboot server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_reboot

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Soft reboot VM1
- Test action 5: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 1:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 6: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to reboot servers. Specifically, the test verifies that:

- After reboot, servers can still be connected.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 5 - Rebuild server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_rebuild

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Rebuild VM1 with another image
- Test action 5: Wait for VM1 to reach ‘ACTIVE’ status
- **Test assertion 1:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 6: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to rebuild servers. Specifically, the test verifies that:

- Servers can be rebuilt with specific image correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 6 - Resize server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_resize

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Resize VM1 with another flavor
- Test action 5: Wait for VM1 to reach 'VERIFY_RESIZE' status
- Test action 6: Confirm resize VM1
- Test action 7: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 1:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 8: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to resize servers. Specifically, the test verifies that:

- Servers can be resized with specific flavor correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 7 - Stop and start server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_stop_start

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1

- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Stop VM1
- Test action 5: Wait for VM1 to reach ‘SHUTOFF’ status
- **Test assertion 1:** Verify ping FIP1 failed and SSH to VM1 via FIP1 failed
- Test action 6: Start VM1
- Test action 7: Wait for VM1 to reach ‘ACTIVE’ status
- **Test assertion 2:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 8: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to stop and start servers. Specifically, the test verifies that:

- When stopped, servers cannot be reached.
- When started, servers can recover its reachability.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 8 - Suspend and resume server

Test case specification

tempest.scenario.test_network_advanced_server_ops.TestNetworkAdvancedServerOps.test_server_connectivity_suspend_resume

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a server VM1 with KEYP1
- Test action 3: Create a floating IP FIP1 and assign FIP1 to VM1
- Test action 4: Suspend VM1
- Test action 5: Wait for VM1 to reach ‘SUSPENDED’ status

- **Test assertion 1:** Verify ping FIP1 failed and SSH to VM1 via FIP1 failed
- Test action 6: Resume VM1
- Test action 7: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 2:** Verify can ping FIP1 successfully and can SSH to VM1 via FIP1
- Test action 8: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to suspend and resume servers. Specifically, the test verifies that:

- When suspended, servers cannot be reached.
- When resumed, servers can recover its reachability.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 9 - Suspend and resume server in sequence

Test case specification

tempest.scenario.test_server_advanced_ops.TestServerAdvancedOps.test_server_sequence_suspend_resume

Test preconditions

- Nova, neutron services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1
- Test action 2: Suspend VM1
- Test action 3: Wait for VM1 to reach 'SUSPENDED' status
- **Test assertion 1:** Verify VM1's status is 'SUSPENDED'
- Test action 4: Resume VM1
- Test action 5: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 2:** Verify VM1's status is 'ACTIVE'
- Test action 6: Suspend VM1

- Test action 7: Wait for VM1 to reach 'SUSPENDED' status
- **Test assertion 3:** Verify VM1 status is 'SUSPENDED'
- Test action 8: Resume VM1
- Test action 9: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 4:** Verify VM1 status is 'ACTIVE'
- Test action 10: Delete KEYP1, VM1 and FIP1

Pass / Fail criteria

This test evaluates the ability to suspend and resume servers in sequence. Specifically, the test verifies that:

- Servers can be suspend and resume in sequence correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 10 - Resize volume backed server

Test case specification

tempest.api.compute.servers.test_server_actions.ServerActionsTestJSON.test_resize_volume_backed_server_confirm

Test preconditions

- Nova, neutron, cinder services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a volume backed server VM1
- Test action 2: Resize VM1 with another flavor
- Test action 3: Wait for VM1 to reach 'VERIFY_RESIZE' status
- Test action 4: Confirm resize VM1
- Test action 5: Wait for VM1 to reach 'ACTIVE' status
- **Test assertion 1:** VM1's status is 'ACTIVE'
- Test action 6: Delete VM1

Pass / Fail criteria

This test evaluates the ability to resize volume backed servers. Specifically, the test verifies that:

- Volume backed servers can be resized with specific flavor correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 11 - Shelve and unshelve server

Test case specification

tempest.scenario.test_shelve_instance.TestShelveInstance.test_shelve_instance

Test preconditions

- Nova, neutron, image services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a security group SG1, which has rules for allowing incoming SSH and ICMP traffic
- Test action 3: Create a server with SG1 and KEYP1
- Test action 4: Create a timestamp and store it in a file F1 inside VM1
- Test action 5: Shelve VM1
- Test action 6: Unshelve VM1
- Test action 7: Wait for VM1 to reach 'ACTIVE' status
- Test action 8: Read F1 and compare if the read value and the previously written value are the same or not
- **Test assertion 1:** Verify the values written and read are the same
- Test action 9: Delete SG1, KEYP1 and VM1

Pass / Fail criteria

This test evaluates the ability to shelve and unshelve servers. Specifically, the test verifies that:

- Servers can be shelved and unshelved correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 12 - Shelve and unshelve volume backed server

Test case specification

tempest.scenario.test_shelve_instance.TestShelveInstance.test_shelve_volume_backed_instance

Test preconditions

- Nova, neutron, image, cinder services are available
- One public network

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a keypair KEYP1
- Test action 2: Create a security group SG1, which has rules for allowing incoming and outgoing SSH and ICMP traffic
- Test action 3: Create a volume backed server VM1 with SG1 and KEYP1
- Test action 4: SSH to VM1 to create a timestamp T_STAMP1 and store it in a file F1 inside VM1
- Test action 5: Shelve VM1
- Test action 6: Unshelve VM1
- Test action 7: Wait for VM1 to reach 'ACTIVE' status
- Test action 8: SSH to VM1 to read the timestamp T_STAMP2 stored in F1
- **Test assertion 1:** Verify T_STAMP1 equals to T_STAMP2
- Test action 9: Delete SG1, KEYP1 and VM1

Pass / Fail criteria

This test evaluates the ability to shelve and unshelve volume backed servers. Specifically, the test verifies that:

- Volume backed servers can be shelved and unshelved correctly.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.16 Tempest Volume test specification

Scope

This test area evaluates the ability of a system under test to manage volumes.

The test area specifically validates the creation, the deletion and the attachment/detach volume operations. tests.

References

N/A

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in individual tests as listed below. For detailed information on the individual steps and assertions performed by the tests, review the Python source code accessible via the following links:

All these test cases are included in the test case functest.tempest.volume of OVP test suite.

Test Case 1 - Attach Detach Volume to Instance

Test case specification

Implementation: Attach Detach Volume to Instance

- tempest.api.volume.test_volumes_actions.VolumesActionsTest.test_attach_detach_volume_to_instance

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1: Create a server VM1
- Test action 2: Attach a provided VOL1 to VM1
- **Test assertion 1:** Verify VOL1 is in 'in-use' status
- Test action 3: Detach VOL1 from VM1
- **Test assertion 2:** Verify detach volume VOL1 successfully and VOL1 is in 'available' status

Pass / fail criteria

This test evaluates the volume API ability of attaching a volume to a server and detaching a volume from a server. Specifically, the test verifies that:

- Volumes can be attached and detached from servers.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

Test Case 2 - Volume Boot Pattern test

Test case specification

Implementation: Volume Boot Pattern test

- tempest.scenario.test_volume_boot_pattern.TestVolumeBootPattern.test_volume_boot_pattern

Test preconditions

- Volume extension API

Basic test flow execution description and pass/fail criteria

Test execution

- Test action 1:Create in Cinder some bootable volume VOL1 importing a Glance image
- Test action 2:Boot an instance VM1 from the bootable volume VOL1
- Test action 3:Write content to the VOL1
- Test action 4:Delete VM1 and Boot a new instance VM2 from the volume VOL1
- Test action 5:Check written content in the instance
- **Test assertion 1:** Verify the content of written file in action 3
- Test action 6:Create a volume snapshot VOL2 while the instance VM2 is running
- Test action 7:Boot an additional instance VM3 from the new snapshot based volume VOL2
- Test action 8:Check written content in the instance booted from snapshot
- **Test assertion 2:** Verify the content of written file in action 3

Pass / fail criteria

This test evaluates the volume storage consistency. Specifically, the test verifies that:

- The content of written file in the volume.

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

8.1.17 VNF test specification

Scope

The VNF test area evaluates basic NFV capabilities of the system under test. These capabilities include creating a small number of virtual machines, establishing the SUT VNF, VNFs which are going to support the test activities and an Orchestrator as well as verifying the proper behavior of the basic VNF.

References

This test area references the following specifications and guides:

- Functest repo for detailed description of the vEPC testcase
  - https://github.com/opnfv/functest/blob/master/docs/testing/user/userguide/test_details.rst#juju_epc
- Functest repo for detailed description of the vIMS testcase
  - https://github.com/opnfv/functest/blob/master/docs/testing/user/userguide/test_details.rst#cloudify_ims
- 3GPP LTE
  - https://www.3gpp.org/technologies/keywords-acronyms/98-lte
- ETSI - TS 24.301
  - https://www.etsi.org/deliver/etsi_ts/124300_124399/124301/10.03.00_60/ts_124301v100300p.pdf
- Cloudify clearwater: opnfv-cloudify-clearwater [1]
  - https://github.com/Orange-OpenSource/opnfv-cloudify-clearwater

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- 3GPP - 3rd Generation Partnership Project
- EPC - Evolved Packet Core
- ETSI - European Telecommunications Standards Institute
- IMS - IP Multimedia Core Network Subsystem
- LTE - Long Term Evolution
- NFV - Network functions virtualization
- OAI - Open Air Interface
- TS - Technical Specifications
- VM - Virtual machine
- VNF - Virtual Network Function

System Under Test (SUT)

The system under test is assumed to be the VNF and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in two separate tests which are executed sequentially. The order of the tests is arbitrary as there are no dependencies across the tests. Specifically, every test performs clean-up operations which return the system to the same state as before the test.

Test Descriptions

Test Case 1 - vEPC

Short name

functest.vnf.vepc

Use case specification

The Evolved Packet Core (EPC) is the main component of the System Architecture Evolution (SAE) which forms the core of the 3GPP LTE specification.

vEPC has been integrated in Functest to demonstrate the capability to deploy a complex mobility-specific NFV scenario on the OPNFV platform. The OAI EPC supports most of the essential functions defined by the 3GPP Technical Specs; hence the successful execution of functional tests on the OAI EPC provides a good endorsement of the underlying NFV platform.

Test preconditions

At least one compute node is available. No further pre-configuration needed.

Basic test flow execution description and pass/fail criteria

Methodology for verifying connectivity

This integration also includes ABot, a Test Orchestration system that enables test scenarios to be defined in high-level DSL. ABot is also deployed as a VM on the OPNFV platform; and this provides an example of the automation driver and the Test VNF being both deployed as separate VNFs on the underlying OPNFV platform.

Test execution

- Test action 1: Deploy Juju controller (VNF Manager) using Bootstrap command.
- Test action 2: Deploy ABot (Orchestrator) and OAI EPC as Juju charms. Configuration of ABot and OAI EPC components is handled through built-in Juju relations.
- Test action 3: Execution of ABot feature files triggered by Juju actions. This executes a suite of LTE signalling tests on the OAI EPC.

- Test action 4: ABot test results are parsed accordingly.
- Test action 5: The deployed VMs are deleted.

Pass / Fail criteria

The VNF Manager (juju) should be deployed successfully

Test executor (ABot), test Orchestration system is deployed and enables test scenarios to be defined in high-level DSL

VMs which are act as VNFs (including the VNF that is the SUT for test case) are following the 3GPP technical specifications accordingly.

Post conditions

The clean-up operations are run.

Test Case 2 - vIMS

Short name

functest.vnf.vims

Use case specification

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

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

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.

Test preconditions

Certain ubuntu server and cloudify images version refer to dovetail testing user guide.

At least 30G RAMs and 50 vcpu cores required.

Basic test flow execution description and pass/fail criteria

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

Test execution

- Test action 1: Deploy a VNF orchestrator (Cloudify).
- Test action 2: Deploy a Clearwater vIMS (IP Multimedia Subsystem) VNF from this orchestrator based on a TOSCA blueprint defined in repository of opnfv-cloudify-clearwater [1].
- Test action 3: Run suite of signaling tests on top of this VNF
- Test action 4: Collect test results.
- Test action 5: The deployed VMs are deleted.

Pass / Fail criteria

The VNF orchestrator (Cloudify) should be deployed successfully.

The Clearwater vIMS (IP Multimedia Subsystem) VNF from this orchestrator should be deployed successfully.

The suite of signaling tests on top of vIMS should be run successfully.

The test scenarios on the NFV platform should be executed successfully following the ETSI standards accordingly.

Post conditions

All resources created during the test run have been cleaned-up

8.1.18 Vping Test Specification

Scope

The vping test area evaluates basic NFVi capabilities of the system under test. These capabilities include creating a small number of virtual machines, establishing basic L3 connectivity between them and verifying connectivity by means of ICMP packets.

References

- OpenStack Shade
  - https://docs.openstack.org/shade/latest/user/index.html
- SSHClient
  - http://docs.paramiko.org/en/2.2/
- SCPClient
  - https://pypi.org/project/scp/

Definitions and Abbreviations

The following terms and abbreviations are used in conjunction with this test area

- ICMP - Internet Control Message Protocol
- L3 - Layer 3

- NFVi - Network Functions Virtualization infrastructure
- SCP - Secure CoPy
- SSH - Secure SHell
- VM - Virtual Machine

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in two separate tests which are executed sequentially. The order of the tests is arbitrary as there are no dependencies across the tests.

Test Descriptions

Test Case 1 - vPing using userdata provided by nova metadata service

Short name

functest.vping.userdata

Use case specification

This test evaluates the use case where an NFVi tenant boots up two VMs and requires L3 connectivity between those VMs. The target IP is passed to the VM that will initiate pings by using a custom userdata script provided by nova metadata service.

Test preconditions

At least one compute node is available. No further pre-configuration needed.

Basic test flow execution description and pass/fail criteria

Methodology for verifying connectivity

Connectivity between VMs is tested by sending ICMP ping packets between selected VMs. The target IP is passed to the VM sending pings by using a custom userdata script by means of the config driver mechanism provided by Nova metadata service. Whether or not a ping was successful is determined by checking the console output of the source VMs.

Test execution

- **Test action 1:**
    - Create a private tenant network by using neutron client
    - Create one subnet and one router in the network by neutron client
    - Add one interface between the subnet and router
    - Add one gateway route to the router by neutron client
    - Store the network id in the response
- **Test assertion 1:** The network id, subnet id and router id can be found in the response
- **Test action 2:**
    - Create an security group by using neutron client
    - Store the security group id parameter in the response
- **Test assertion 2:** The security group id can be found in the response
- **Test action 3:**
    - Boot VM1 by using nova client with configured name, image, flavor, private tenant network created in test action 1 and security group created in test action 2
- **Test assertion 3:** The VM1 object can be found in the response
- **Test action 4:**
    - Generate ping script with the IP of VM1 to be passed as userdata provided by the **nova metadata service**
- **Test action 5:**
    - Boot VM2 by using nova client with configured name, image, flavor, private tenant network created in test action 1, security group created in test action 2, and userdata created in test action 4
- **Test assertion 4:** The VM2 object can be found in the response
- **Test action 6:**
    - Inside VM2, the ping script is executed automatically when booted and it contains a loop doing the ping until the return code is 0 or timeout reached
    - For each ping, when the return code is 0, “vPing OK” is printed in the VM2 console-log, otherwise, “vPing KO” is printed
    - Monitoring the console-log of VM2 to see the response generated by the script
- **Test assertion 5:** “vPing OK” is detected, when monitoring the console-log in VM2
- **Test action 7:**
    - Delete VM1, VM2
- **Test assertion 6:** VM1 and VM2 are not present in the VM list
- **Test action 8:**
    - Delete security group, gateway, interface, router, subnet and network
- **Test assertion 7:** The security group, gateway, interface, router, subnet and network are no longer present in the lists after deleting

Pass / Fail criteria

This test evaluates basic NFVi capabilities of the system under test. Specifically, the test verifies that:

- Neutron client network, subnet, router, interface create commands return valid "id" parameters which are shown in the create response message
- Neutron client interface add command to add between subnet and router returns success code
- Neutron client gateway add command to add to router returns success code
- Neutron client security group create command returns valid "id" parameter which is shown in the response message
- Nova client VM create command returns valid VM attributes response message
- Nova metadata server can transfer userdata configuration at nova client VM booting time
- Ping command from one VM to the other in same private tenant network returns valid code
- All items created using neutron client or nova client create commands are able to be removed by using the returned identifiers

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

Test Case 2 - vPing using SSH to a floating IP

Short name

functest.vping.ssh

Use case specification

This test evaluates the use case where an NFVi tenant boots up two VMs and requires L3 connectivity between those VMs. An SSH connection is establised from the host to a floating IP associated with VM2 and `ping` is executed on VM2 with the IP of VM1 as target.

Test preconditions

At least one compute node is available. There should exist an OpenStack external network and can assign floating IP.

Basic test flow execution description and pass/fail criteria

Methodology for verifying connectivity

Connectivity between VMs is tested by sending ICMP ping packets between selected VMs. To this end, the test establishes an SSH connection from the host running the test suite to a floating IP associated with VM2 and executes `ping` on VM2 with the IP of VM1 as target.

Test execution

- **Test action 1:**
  - Create a private tenant network by neutron client
  - Create one subnet and one router are created in the network by using neutron client
  - Create one interface between the subnet and router
  - Add one gateway route to the router by neutron client
  - Store the network id in the response
- **Test assertion 1:** The network id, subnet id and router id can be found in the response
- **Test action 2:**
  - Create an security group by using neutron client
  - Store the security group id parameter in the response
- **Test assertion 2:** The security group id can be found in the response
- **Test action 3:**
  - Boot VM1 by using nova client with configured name, image, flavor, private tenant network created in test action 1 and security group created in test action 2
- **Test assertion 3:** The VM1 object can be found in the response
- **Test action 4:**
  - Boot VM2 by using nova client with configured name, image, flavor, private tenant network created in test action 1 and security group created in test action 2
- **Test assertion 4:** The VM2 object can be found in the response
- **Test action 5:**
  - Create one floating IP by using neutron client, storing the floating IP address returned in the response
- **Test assertion 5:** Floating IP address can be found in the response
- **Test action 6:**
  - Assign the floating IP address created in test action 5 to VM2 by using nova client
- **Test assertion 6:** The assigned floating IP can be found in the VM2 console log file
- **Test action 7:**
  - Establish SSH connection between the test host and VM2 through the floating IP
- **Test assertion 7:** SSH connection between the test host and VM2 is established within 300 seconds
- **Test action 8:**
  - Copy the Ping script from the test host to VM2 by using SCPClient
- **Test assertion 8:** The Ping script can be found inside VM2
- **Test action 9:**
  - Inside VM2, to execute the Ping script to ping VM1, the Ping script contains a loop doing the ping until the return code is 0 or timeout reached
  - For each ping, when the return code is 0, “vPing OK” is printed in the VM2 console-log, otherwise, “vPing KO” is printed

  - Monitoring the console-log of VM2 to see the response generated by the script

- **Test assertion 9:** “vPing OK” is detected, when monitoring the console-log in VM2
- **Test action 10:**
  - Delete VM1, VM2
- **Test assertion 10:** VM1 and VM2 are not present in the VM list
- **Test action 11:**
  - Delete floating IP, security group, gateway, interface, router, subnet and network
- **Test assertion 11:** The security group, gateway, interface, router, subnet and network are no longer present in the lists after deleting

Pass / Fail criteria

This test evaluates basic NFVi capabilities of the system under test. Specifically, the test verifies that:

- Neutron client network, subnet, router, interface create commands return valid “id” parameters which are shown in the create response message
- Neutron client interface add command to add between subnet and router return success code
- Neutron client gateway add command to add to router return success code
- Neutron client security group create command returns valid “id” parameter which is shown in the response message
- Nova client VM create command returns valid VM attributes response message
- Neutron client floating IP create command return valid floating IP address
- Nova client add floating IP command returns valid response message
- SSH connection can be established using a floating IP
- Ping command from one VM to another in same private tenant network returns valid code
- All items created using neutron client or nova client create commands are able to be removed by using the returned identifiers

In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

None

8.1.19 VPN test specification

Scope

The VPN test area evaluates the ability of the system under test to support VPN networking for virtual workloads. This test area tests CRUD (Create, Read, Update, Delete) operations of BGPVPN API.

References

This test area evaluates the ability of the system to perform selected actions defined in the following specifications. Details of specific features evaluated are described in the test descriptions.

- RFC 4364 - BGP/MPLS IP Virtual Private Networks
  - https://tools.ietf.org/html/rfc4364
- RFC 4659 - BGP-MPLS IP Virtual Private Network
  - https://tools.ietf.org/html/rfc4659
- RFC 2547 - BGP/MPLS VPNs
  - https://tools.ietf.org/html/rfc2547

Definitions and abbreviations

The following terms and abbreviations are used in conjunction with this test area

- BGP - Border gateway protocol
- NFVi - Network functions virtualization infrastructure
- VM - Virtual machine
- VPN - Virtual private network

System Under Test (SUT)

The system under test is assumed to be the NFVi and VIM in operation on a Pharos compliant infrastructure.

Test Area Structure

The test area is structured in several tempest tests which are executed sequentially. The order of the tests is arbitrary as there are no dependencies across the tests. Specifially, every test performs clean-up operations which return the system to the same state as before the test.

Test Descriptions

Test Case 1 - Tempest API CRUD Tests

Short Name

functest.tempest.bgpvpn

Use case specification

This test case combines multiple CRUD (Create, Read, Update, Delete) tests for the objects defined by the BGPVPN API extension of Neutron.

These tests are implemented in the upstream networking-bgpvpn project repository as a Tempest plugin.

Test preconditions

The VIM is operational and the networking-bgpvpn service plugin for Neutron is correctly configured and loaded. At least one compute node is available.

Basic test flow execution description and pass/fail criteria

List of test cases

- networking_bgpvpn_tempest.tests.api.test_create_bgpvpn
- networking_bgpvpn_tempest.tests.api.test_create_bgpvpn_as_non_admin_fail
- networking_bgpvpn_tempest.tests.api.test_delete_bgpvpn_as_non_admin_fail
- networking_bgpvpn_tempest.tests.api.test_show_bgpvpn_as_non_owner_fail
- networking_bgpvpn_tempest.tests.api.test_list_bgpvpn_as_non_owner_fail
- networking_bgpvpn_tempest.tests.api.test_show_netassoc_as_non_owner_fail
- networking_bgpvpn_tempest.tests.api.test_list_netassoc_as_non_owner_fail
- networking_bgpvpn_tempest.tests.api.test_associate_disassociate_network
- networking_bgpvpn_tempest.tests.api.test_update_route_target_non_admin_fail
- networking_bgpvpn_tempest.tests.api.test_create_bgpvpn_with_invalid_routetargets
- networking_bgpvpn_tempest.tests.api.test_update_bgpvpn_invalid_routetargets
- networking_bgpvpn_tempest.tests.api.test_associate_invalid_network
- networking_bgpvpn_tempest.tests.api.test_disassociate_invalid_network
- networking_bgpvpn_tempest.tests.api.test_associate_disassociate_router
- networking_bgpvpn_tempest.tests.api.test_attach_associated_subnet_to_associated_router

The tests include both positive tests and negative tests. The latter are identified with the suffix "_fail" in their name.

Test execution

The tests are executed sequentially and a separate pass/fail result is recorded per test.

In general, every test case performs the API operations indicated in its name and asserts that the action succeeds (positive test) or a specific exception is triggered (negative test). The following describes the test execution per test in further detail.

networking_bgpvpn_tempest.tests.api.test_create_bgpvpn

- Create a BGPVPN as an admin.
- **Test assertion**: The API call succeeds.

networking_bgpvpn_tempest.tests.api.test_create_bgpvpn_as_non_admin_fail

- Attempt to create a BGPVPN as non-admin.
- **Test assertion**: Creating a BGPVPN as non-admin fails.

networking_bgpvpn_tempest.tests.api.test_delete_bgpvpn_as_non_admin_fail

- Create BGPVPN vpn1 as admin.
- Attempt to delete vpn1 as non-admin.
- **Test assertion**: The deletion of vpn1 as non-admin fails.

networking_bgpvpn_tempest.tests.api.test_show_bgpvpn_as_non_owner_fail

- Create a BGPVPN vpn1 as admin in project1.
- **Test assertion**: Attempting to retrieve detailed properties of vpn1 in project2 fails.

networking_bgpvpn_tempest.tests.api.test_list_bgpvpn_as_non_owner_fail

- Create a BGPVPN vpn1 as admin in project1.
- Retrieve a list of all BGPVPNs in project2.
- **Test assertion**: The list of BGPVPNs retrieved in project2 does not include vpn1.

networking_bgpvpn_tempest.tests.api.test_show_netassoc_as_non_owner_fail

- Create BGPVPN vpn1 as admin in project1.
- Associate vpn1 with a Neutron network in project1
- **Test assertion**: Retrieving detailed properties of the network association fails in project2.

networking_bgpvpn_tempest.tests.api.test_list_netassoc_as_non_owner_fail

- Create BGPVPN vpn1 as admin in project1.
- Create network association net-assoc1 with vpn1 and Neutron network net1 in project1.
- Retrieve a list of all network associations in project2.
- **Test assertion**: The retrieved list of network associations does not include network association net-assoc1.

networking_bgpvpn_tempest.tests.api.test_associate_disassociate_network

- Create a BGPVPN vpn1 as admin.
- Associate vpn1 with a Neutron network net1.
- **Test assertion**: The metadata of vpn1 includes the UUID of net1.
- Diassociate vpn1 from the Neutron network.

- **Test assertion**: The metadata of vpn1 does not include the UUID of net1.

networking_bgpvpn_tempest.tests.api.test_update_route_target_non_admin_fail

- Create a BGPVPN vpn1 as admin with specific route targets.
- Attempt to update vpn1 with different route targets as non-admin.
- **Test assertion**: The update fails.

networking_bgpvpn_tempest.tests.api.test_create_bgpvpn_with_invalid_routetargets

- Attempt to create a BGPVPN as admin with invalid route targets.
- **Test assertion**: The creation of the BGPVPN fails.

networking_bgpvpn_tempest.tests.api.test_update_bgpvpn_invalid_routetargets

- Create a BGPVPN vpn1 as admin with empty route targets.
- Attempt to update vpn1 with invalid route targets.
- **Test assertion**: The update of the route targets fails.

networking_bgpvpn_tempest.tests.api.test_associate_invalid_network

- Create BGPVPN vpn1 as admin.
- Attempt to associate vpn1 with a non-existing Neutron network.
- **Test assertion**: Creating the network association fails.

networking_bgpvpn_tempest.tests.api.test_disassociate_invalid_network

- Create BGPVPN vpn1 as admin.
- Create network association net-assoc1 with vpn1 and Neutron network net1.
- Attempt to delete net-assoc1 with an invalid network UUID.
- **Test assertion**: The deletion of the net-assoc fails.

networking_bgpvpn_tempest.tests.api.test_associate_disassociate_router

- Create a BGPVPN vpn1 as admin.
- Associate vpn1 with a Neutron router router1.
- **Test assertion**: The metadata of vpn1 includes the UUID of router1.
- Disassociate router1 from vpn1.
- **Test assertion**: The metadata of vpn1 does not include the UUID of router1.

networking_bgpvpn_tempest.tests.api.test_attach_associated_subnet_to_associated_router

- Create BGPVPN vpn1 as admin.
- Associate vpn1 with Neutron network net1.
- Create BGPVPN vpn2
- Associate vpn2 with Neutron router router1.
- Attempt to add the subnet of net1 to router1
- **Test assertion**: The association fails.

Pass / fail criteria

This test validates that all supported CRUD operations (create, read, update, delete) can be applied to the objects of the Neutron BGPVPN extension. In order to pass this test, all test assertions listed in the test execution above need to pass.

Post conditions

N/A

CHAPTER

NINE

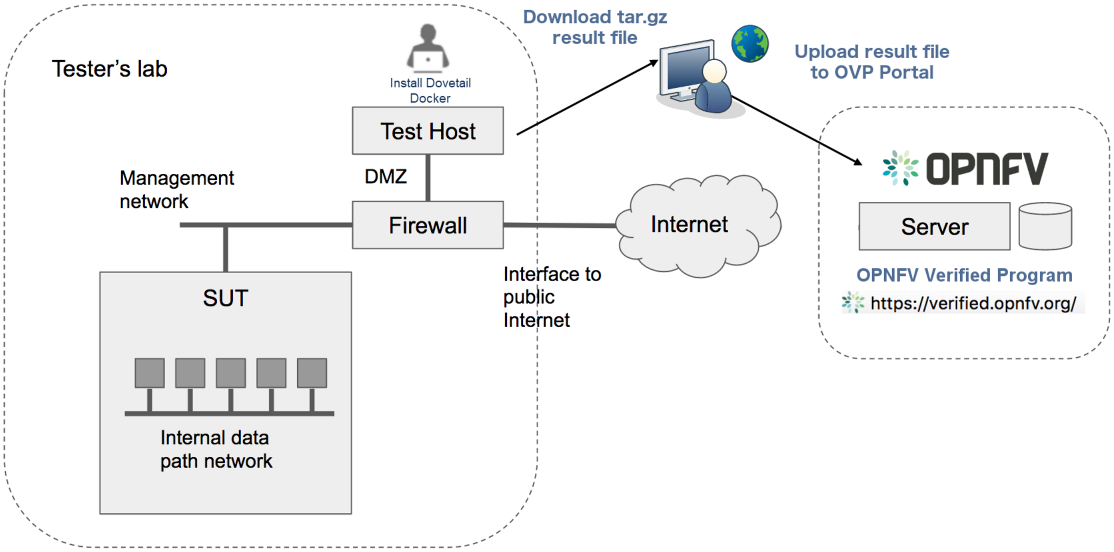
OVP TESTING USER GUIDE

9.1 Conducting OVP NFVI Testing with Dovetail

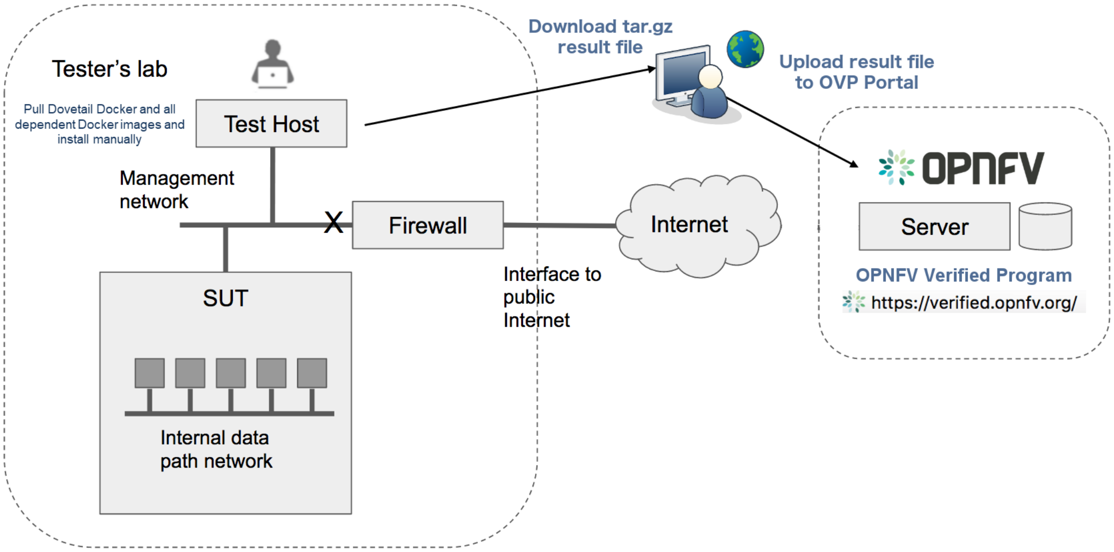
9.1.1 Overview

This guide provides the instructions for the OVP Infrastructure testing. For the OVP VNF testing, please refer to the next section.

The Dovetail testing framework for OVP consists of two major parts: the testing client which executes all test cases in a lab (vendor self-testing or a third party lab), and the server system which is hosted by the OVP administrator to store and view test results based on a web API. The following diagram illustrates this overall framework.


Tester's lab
Install Dovetail Docker
Test Host
DMZ
Management network
Firewall
SUT
Internal data path network
Interface to public Internet
Internet
Download tar.gz result file
Upload result file to OVP Portal
OPNFV
Server
OPNFV Verified Program
https://verified.opnfv.org/

Within the tester's lab, the Test Host is the machine where Dovetail executes all automated test cases. As it hosts the test harness, the Test Host must not be part of the System Under Test (SUT) itself. The above diagram assumes that the tester's Test Host is situated in a DMZ, which has internal network access to the SUT and external access via the public Internet. The public Internet connection allows for easy installation of the Dovetail containers. A single compressed file that includes all the underlying results can be pulled from the Test Host and uploaded to the OPNFV OVP server. This arrangement may not be supported in some labs. Dovetail also supports an offline mode of installation which is illustrated in the next diagram.


Tester's lab
Pull Dovetail Docker and all dependent Docker images and install manually
Test Host
Management network
Firewall
SUT
Internal data path network
Interface to public Internet
Internet
Download tar.gz result file
Upload result file to OVP Portal
OPNFV
Server
OPNFV Verified Program
https://verified.opnfv.org/

In the offline mode, the Test Host only needs to have access to the SUT via the internal network, but does not need to connect to the public Internet. This user guide will highlight differences between the online and offline modes of the Test Host. While it is possible to run the Test Host as a virtual machine, this user guide assumes it is a physical machine for simplicity.

The rest of this guide will describe how to install the Dovetail tool as a Docker container image, go over the steps of running the OVP test suite, and then discuss how to view test results and make sense of them.

Readers interested in using Dovetail for its functionalities beyond OVP testing, e.g. for in-house or extended testing, should consult the Dovetail developer's guide for additional information.

9.1.2 Installing Dovetail

In this section, we describe the procedure to install Dovetail client tool on the Test Host. The Test Host must have network access to the management network with access rights to the Virtual Infrastructure Manager's API.

Checking the Test Host Readiness

The Test Host must have network access to the Virtual Infrastructure Manager's API hosted in the SUT so that the Dovetail tool can exercise the API from the Test Host. It must also have `ssh` access to the Linux operating system of the compute nodes in the SUT. The `ssh` mechanism is used by some test cases to generate test events in the compute nodes. You can find out which test cases use this mechanism in the test specification document.

We have tested the Dovetail tool on the following host operating systems. Other versions or distributions of Linux may also work, but community support may be more available on these versions.

- Ubuntu 16.04.2 LTS (Xenial) or 14.04 LTS (Trusty)
- CentOS-7-1611
- Red Hat Enterprise Linux 7.3
- Fedora 24 or 25 Server

Use of Ubuntu 16.04 is highly recommended, as it has been most widely employed during testing. Non-Linux operating systems, such as Windows and Mac OS, have not been tested and are not supported.

If online mode is used, the tester should also validate that the Test Host can reach the public Internet. For example,

```
$ ping www.opnfv.org
PING www.opnfv.org (50.56.49.117): 56 data bytes
64 bytes from 50.56.49.117: icmp_seq=0 ttl=48 time=52.952 ms
64 bytes from 50.56.49.117: icmp_seq=1 ttl=48 time=53.805 ms
64 bytes from 50.56.49.117: icmp_seq=2 ttl=48 time=53.349 ms
...
```

Or, if the lab environment does not allow ping, try validating it using HTTPS instead.

```
$ curl https://www.opnfv.org
<!doctype html>


<html lang="en-US" class="no-js">
<head>
...
```

Installing Prerequisite Packages on the Test Host

The main prerequisite software for Dovetail is Docker.

Dovetail does not work with Docker versions prior to 1.12.3. We have validated Dovetail with Docker 17.03 CE. Other versions of Docker later than 1.12.3 may also work, but community support may be more available on Docker 17.03 CE or greater.

```
$ sudo docker version
Client:
Version:      17.03.1-ce
API version:  1.27
Go version:   go1.7.5
Git commit:   c6d412e
Built:        Mon Mar 27 17:10:36 2017
OS/Arch:      linux/amd64

Server:
Version:      17.03.1-ce
API version:  1.27 (minimum version 1.12)
Go version:   go1.7.5
Git commit:   c6d412e
Built:        Mon Mar 27 17:10:36 2017
OS/Arch:      linux/amd64
Experimental: false
```

If your Test Host does not have Docker installed, or Docker is older than 1.12.3, or you have Docker version other than 17.03 CE and wish to change, you will need to install, upgrade, or re-install in order to run Dovetail. If you need further assistance with Docker installation process, you should refer to the official Docker installation guide that is relevant to your Test Host's operating system.

The above installation steps assume that the Test Host is in the online mode. For offline testing, use the following offline installation steps instead. For instance, download Docker static binaries and copy the tar file to the Test Host, such as for Ubuntu14.04, you may follow the following link:

```
https://github.com/meetyg/docker-offline-install
```

Configuring the Test Host Environment

The Test Host needs a few environment variables set correctly in order to access the OpenStack API which is required to drive the Dovetail tests. For convenience and as a convention, we will also create a home directory for storing all Dovetail related config files and results files:

```
$ mkdir -p ${HOME}/dovetail
$ export DOVETAIL_HOME=${HOME}/dovetail
```

For example, here we set dovetail home directory to be `${HOME}/dovetail`. Then create two directories named `pre_config` and `images` under this directory to store all Dovetail related config files and all test images respectively:

```
$ mkdir -p ${DOVETAIL_HOME}/pre_config
$ mkdir -p ${DOVETAIL_HOME}/images
```

Setting up Primary Configuration File

At this point, you will need to consult your SUT (OpenStack) administrator to correctly set the configurations in a file named `env_config.sh`. The OpenStack settings need to be configured such that the Dovetail client has all the necessary credentials and privileges to execute all test operations. If the SUT uses terms somewhat differently from the standard OpenStack naming, you will need to adjust this file accordingly.

Create and edit the file `${DOVETAIL_HOME}/pre_config/env_config.sh` so that all parameters are set correctly to match your SUT. Here is an example of what this file should contain.

```
$ cat ${DOVETAIL_HOME}/pre_config/env_config.sh

# Project-level authentication scope (name or ID), admin project is recommended.
export OS_PROJECT_NAME=admin

# Authentication username, belongs to the project above, admin user is recommended.
export OS_USERNAME=admin

# Authentication password. Use your own password
export OS_PASSWORD=xxxxxxxx

# Authentication URL, one of the endpoints of keystone service. If this is v3 version,
# there needs some extra variables as follows.
export OS_AUTH_URL='http://xxx.xxx.xxx.xxx:5000/v3'

# Default is 2.0. If use keystone v3 API, this should be set as 3.
export OS_IDENTITY_API_VERSION=3

# Domain name or ID containing the user above.
# Command to check the domain: openstack user show <OS_USERNAME>
export OS_USER_DOMAIN_NAME=default

# Domain name or ID containing the project above.
# Command to check the domain: openstack project show <OS_PROJECT_NAME>
export OS_PROJECT_DOMAIN_NAME=default

# Special environment parameters for https.
# If using https + cacert, the path of cacert file should be provided.
# The cacert file should be put at $DOVETAIL_HOME/pre_config.
export OS_CACERT=/path/to/pre_config/cacert.pem
```

(continues on next page)

(continued from previous page)

```
# If using https + no cacert, should add OS_INSECURE environment parameter.
export OS_INSECURE=True

# The name of a network with external connectivity for allocating floating
# IPs. It is required that at least one Neutron network with the attribute
# 'router:external=True' is pre-configured on the system under test.
# This network is used by test cases to SSH into tenant VMs and perform
# operations there.
export EXTERNAL_NETWORK=xxx

# Set an existing role used to create project and user for vping test cases.
# Otherwise, it will create a role 'Member' to do that.
export NEW_USER_ROLE=xxx

# For XCI installer the following environment parameters should be added in
# this file. Otherwise, those parameters could be ignored.
export INSTALLER_TYPE=osa
export DEPLOY_SCENARIO=os-nosdn-nofeature
export XCI_FLAVOR=noha
```

The OS_AUTH_URL variable is key to configure correctly, as the other admin services are collected from the identity service. HTTPS should be configured in the SUT so either OS_CACERT or OS_INSECURE should be uncommented. However, if SSL is disabled in the SUT, comment out both OS_CACERT and OS_INSECURE variables. Ensure the '/path/to/pre_config' directory in the above file matches the directory location of the cacert file for the OS_CACERT variable.

The next three sections outline additional configuration files used by Dovetail. The tempest (tempest_conf.yaml) configuration file is required for executing all tempest test cases (e.g. functest.tempest.compute, functest.tempest.ipv6 ...) and functest.security.patrole. The HA (pod.yaml) configuration file is required for HA test cases and is also employed to collect SUT hardware info. The hosts.yaml is optional for hostname/IP resolution.

Configuration for Running Tempest Test Cases (Mandatory)

The test cases in the test areas *tempest* and *security* are based on Tempest. A SUT-specific configuration of Tempest is required in order to run those test cases successfully. The corresponding SUT-specific configuration options must be supplied in the file `$DOVETAIL_HOME/pre_config/tempest_conf.yaml`.

Create and edit file `$DOVETAIL_HOME/pre_config/tempest_conf.yaml`. Here is an example of what this file should contain.

```
compute:
  # The minimum number of compute nodes expected.
  # This should be no less than 2 and no larger than the compute nodes the SUT␣
↪actually has.
  min_compute_nodes: 2

  # Expected device name when a volume is attached to an instance.
  volume_device_name: vdb

  # One sub test case of functest.tempest.osinterop will be skipped if not provide␣
↪this version.
  # The default range of microversion for tempest is [None - None].
  # Test case functest.tempest.osinterop required the range to be [2.2 - latest].
  max_microversion: 2.65
```

Use the listing above as a minimum to execute the mandatory test cases.

If the optional BGPVPN Tempest API tests shall be run, Tempest needs to be told that the BGPVPN service is available. To do that, add the following to the $DOVETAIL_HOME/pre_config/tempest_conf.yaml configuration file:

```
service_available:
  bgpvpn: True
```

Configuration for Running HA Test Cases (Mandatory)

The HA test cases require OpenStack controller node info. It must include the node's name, role, ip, as well as the user and key_filename or password to login to the node. Users must create the file ${DOVETAIL_HOME}/pre_config/pod.yaml to store the info. For some HA test cases, they will log in the controller node 'node1' and kill the specific processes. The names of the specific processes may be different with the actual ones of the SUTs. The processes' names can also be changed with file ${DOVETAIL_HOME}/pre_config/pod.yaml.

This file is also used as a basis to collect SUT hardware information which is stored alongside results and uploaded to the OVP web portal. The SUT hardware information can be viewed within the 'My Results' view in the OVP web portal by clicking the SUT column 'info' link. In order to collect SUT hardware information holistically, ensure this file has an entry for each of the controller and compute nodes within the SUT.

Below is a sample with the required syntax when password is employed by the controller.

```
nodes:
-
    # This info of node0 is used only for one optional test case 'yardstick.ha.controller_restart'.
    # If you don't plan to test it, this Jumpserver node can be ignored.
    # This can not be changed and **must** be node0.
    name: node0

    # This **must** be Jumpserver.
    role: Jumpserver

    # This is the instance IP of a node which has ipmitool installed.
    ip: xx.xx.xx.xx

    # User name of the user of this node. This user **must** have sudo privileges.
    user: root

    # Password of the user.
    password: root

-
    # Almost all HA test cases are trying to login to a controller node named 'node1'
    # and then kill some processes running on it.
    # If you don't want to reset the attack node name for each test case, this
    # name can not be changed and **must** be node1.
    name: node1

    # This **must** be controller.
    role: Controller

    # This is the instance IP of a controller node, which is the haproxy primary node
    ip: xx.xx.xx.xx
```

(continues on next page)

(continued from previous page)

```
    # User name of the user of this node. This user **must** have sudo privileges.
    user: root

    # Password of the user.
    password: root

process_info:
-
    # For all HA test cases, there are 2 parameters, 'attack_process' and 'attack_host
↪',
    # which support to be set by users instead of using the default values.
    # The 'attack_process' is the process name of one HA test case which it try to␣
↪kill.
    # The 'attack_host' is the host name which the test case try to login and then␣
↪kill
    # the process running on it.
    # Fllowing is 2 samples.

    # The default attack process of yardstick.ha.rabbitmq is 'rabbitmq-server'.
    # Here can be reset to 'rabbitmq'.
    testcase_name: yardstick.ha.rabbitmq
    attack_process: rabbitmq

-
    # The default attack host for all HA test cases is 'node1'.
    # Here can be reset to any other node given in the section 'nodes'.
    testcase_name: yardstick.ha.glance_api
    attack_host: node2
```

Besides the ‘password’, a ‘key_filename’ entry can be provided to login to the controller node. Users need to create file `$DOVETAIL_HOME/pre_config/id_rsa` to store the private key. A sample is provided below to show the required syntax when using a key file.

```
nodes:
-
    name: node1
    role: Controller
    ip: 10.1.0.50
    user: root

    # Private ssh key for accessing the controller nodes. If a keyfile is
    # being used instead of password, it **must** be put under
    # $DOVETAIL_HOME/pre_config/ and named 'id_rsa'.
    key_filename: /home/dovetail/pre_config/id_rsa
```

Under nodes, repeat entries for name, role, ip, user and password or key file for each of the controller/compute nodes that comprise the SUT. Use a ‘-‘ to separate each of the entries. Specify the value for the role key to be either ‘Controller’ or ‘Compute’ for each node.

Under process_info, repeat entries for testcase_name, attack_host and attack_process for each HA test case. Use a ‘-‘ to separate each of the entries. The default attack host of all HA test cases is **node1**. The default attack processes of all HA test cases are list here:

Test Case Name	Attack Process Name
yardstick.ha.cinder_api	cinder-api
yardstick.ha.database	mysql
yardstick.ha.glance_api	glance-api
yardstick.ha.haproxy	haproxy
yardstick.ha.keystone	keystone
yardstick.ha.neutron_l3_agent	neutron-l3-agent
yardstick.ha.neutron_server	neutron-server
yardstick.ha.nova_api	nova-api
yardstick.ha.rabbitmq	rabbitmq-server

Configuration of Hosts File (Optional)

If your SUT uses a hosts file to translate hostnames into the IP of OS_AUTH_URL, then you need to provide the hosts info in a file `$DOVETAIL_HOME/pre_config/hosts.yaml`.

Create and edit file `$DOVETAIL_HOME/pre_config/hosts.yaml`. Below is an example of what this file should contain. Note that multiple hostnames can be specified for each IP address, as shown in the generic syntax below the example.

```
$ cat ${DOVETAIL_HOME}/pre_config/hosts.yaml

---
hosts_info:
  192.168.141.101:
    - identity.endpoint.url
    - compute.endpoint.url

  <ip>:
    - <hostname1>
    - <hostname2>
```

Installing Dovetail on the Test Host

The Dovetail project maintains a Docker image that has Dovetail test tools preinstalled. This Docker image is tagged with versions. Before pulling the Dovetail image, check the OPNFV's OVP web page first to determine the right tag for OVP testing.

Online Test Host

If the Test Host is online, you can directly pull Dovetail Docker image, then all other dependent docker images will automatically be downloaded. Also you can download other related VM images such as Ubuntu and Cirros images which are used by Dovetail for image creation and VM instantiation within the SUT.

Following given the download url for each VM images. Cirros-0.4.0 and Ubuntu-16.04 are used by mandatory test cases, so they are the only 2 images **must** be downloaded before doing the test. There are also 2 other optional VM images, Ubuntu-14.04 and Cloudify-manager, which are used by optional test cases functest.vnf.vepc and functest.vnf.vims. If you don't plan to test these 2 test cases, you can skip downloading these 2 images.

```
$ wget -nc http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img -P $
↪{DOVETAIL_HOME}/images
$ wget -nc https://cloud-images.ubuntu.com/releases/16.04/release/ubuntu-16.04-server-
↪cloudimg-amd64-disk1.img -P ${DOVETAIL_HOME}/images
```

(continues on next page)

(continued from previous page)

```
$ wget -nc https://cloud-images.ubuntu.com/releases/14.04/release/ubuntu-14.04-server-
↪cloudimg-amd64-disk1.img -P ${DOVETAIL_HOME}/images
$ wget -nc http://repository.cloudifysource.org/cloudify/19.01.24/community-release/
↪cloudify-docker-manager-community-19.01.24.tar -P ${DOVETAIL_HOME}/images

$ sudo docker pull opnfv/dovetail:ovp-3.0.0
ovp-3.0.0: Pulling from opnfv/dovetail
324d088ce065: Pull complete
2ab951b6c615: Pull complete
9b01635313e2: Pull complete
04510b914a6c: Pull complete
83ab617df7b4: Pull complete
40ebbe7294ae: Pull complete
d5db7e3e81ae: Pull complete
0701bf048879: Pull complete
0ad9f4168266: Pull complete
d949894f87f6: Pull complete
Digest: sha256:7449601108ebc5c40f76a5cd9065ca5e18053be643a0eeac778f537719336c29
Status: Downloaded newer image for opnfv/dovetail:ovp-3.0.0
```

Offline Test Host

If the Test Host is offline, you will need to first pull the Dovetail Docker image and all the dependent images that Dovetail uses, to a host that is online. The reason that you need to pull all dependent images is because Dovetail normally does dependency checking at run-time and automatically pulls images as needed, if the Test Host is online. If the Test Host is offline, then all these dependencies will need to be manually copied.

The Docker images, Ubuntu and Cirros image below are necessary for all mandatory test cases.

```
$ sudo docker pull opnfv/dovetail:ovp-3.0.0
$ sudo docker pull opnfv/functest-smoke:hunter
$ sudo docker pull opnfv/functest-healthcheck:hunter
$ sudo docker pull opnfv/yardstick:opnfv-8.0.0
$ sudo docker pull opnfv/bottlenecks:8.0.1-latest
$ wget -nc http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img -P
↪{ANY_DIR}
$ wget -nc https://cloud-images.ubuntu.com/releases/16.04/release/ubuntu-16.04-server-
↪cloudimg-amd64-disk1.img -P ${DOVETAIL_HOME}/images
```

The other Docker images and test images below are only used by optional test cases.

```
$ sudo docker pull opnfv/functest-vnf:hunter
$ wget -nc https://cloud-images.ubuntu.com/releases/14.04/release/ubuntu-14.04-server-
↪cloudimg-amd64-disk1.img -P {ANY_DIR}
$ wget -nc http://repository.cloudifysource.org/cloudify/19.01.24/community-release/
↪cloudify-docker-manager-community-19.01.24.tar -P ${DOVETAIL_HOME}/images
```

Once all these images are pulled, save the images, copy them to the Test Host, and then load the Dovetail image and all dependent images at the Test Host.

At the online host, save the images with the command below.

```
$ sudo docker save -o dovetail.tar opnfv/dovetail:ovp-3.0.0 \
  opnfv/functest-smoke:hunter opnfv/functest-healthcheck:hunter \
  opnfv/functest-vnf:hunter \
  opnfv/yardstick:opnfv-8.0.0 opnfv/bottlenecks:8.0.1-latest
```

The command above creates a dovetail.tar file with all the images, which can then be copied to the Test Host. To load the Dovetail images on the Test Host execute the command below.

```
$ sudo docker load --input dovetail.tar
```

Now check to see that all Docker images have been pulled or loaded properly.

```
$ sudo docker images
REPOSITORY                     TAG                 IMAGE ID            CREATED              ␣
↪     SIZE
opnfv/dovetail                 ovp-3.0.0           4b68659da24d        22 hours ago         ␣
↪     825MB
opnfv/functest-smoke           hunter              c0253f6de153        3 weeks ago          ␣
↪     556MB
opnfv/functest-healthcheck     hunter              fb6d766e38e0        3 weeks ago          ␣
↪     379MB
opnfv/functest-vnf             hunter              31466d52d155        21 hours ago         ␣
↪     1.1GB
opnfv/yardstick                opnfv-8.0.0         189d7d9fbcb2        7 months ago         ␣
↪     2.54GB
opnfv/bottlenecks              8.0.1-latest        44c1b9fb25aa        5 hours ago          ␣
↪     837MB
```

After copying and loading the Dovetail images at the Test Host, also copy the test images (Ubuntu, Cirros and cloudify-manager) to the Test Host.

- Copy image `cirros-0.4.0-x86_64-disk.img` to `${DOVETAIL_HOME}/images/`.
- Copy image `ubuntu-14.04-server-cloudimg-amd64-disk1.img` to `${DOVETAIL_HOME}/images/`.
- Copy image `ubuntu-16.04-server-cloudimg-amd64-disk1.img` to `${DOVETAIL_HOME}/images/`.
- Copy image `cloudify-docker-manager-community-19.01.24.tar` to `${DOVETAIL_HOME}/images/`.

9.1.3 Starting Dovetail Docker

Regardless of whether you pulled down the Dovetail image directly online, or loaded it from a static image tar file, you are now ready to run Dovetail. Use the command below to create a Dovetail container and get access to its shell.

```
$ sudo docker run --privileged=true -it \
          -e DOVETAIL_HOME=$DOVETAIL_HOME \
          -v $DOVETAIL_HOME:$DOVETAIL_HOME \
          -v /var/run/docker.sock:/var/run/docker.sock \
          opnfv/dovetail:<tag> /bin/bash
```

The `-e` option sets the DOVETAIL_HOME environment variable in the container and the `-v` options mounts files from the test host to the destination path inside the container. The latter option allows the Dovetail container to read the configuration files and write result files into DOVETAIL_HOME on the Test Host. The user should be within the Dovetail container shell, once the command above is executed.

9.1.4 Running the OVP Test Suite

All or a subset of the available tests can be executed at any location within the Dovetail container prompt. You can refer to *Dovetail Command Line Interface Reference* for the details of the CLI.

```
$ dovetail run --testsuite <test-suite-name>
```

The `--testsuite` option is used to control the set of tests intended for execution at a high level. For the purposes of running the OVP test suite, the test suite name follows the following format, `ovp.<release-version>`. The latest and default test suite is ovp.2019.12.

```
$ dovetail run
```

This command is equal to

```
$ dovetail run --testsuite ovp.2019.12
```

Without any additional options, the above command will attempt to execute all mandatory and optional test cases with test suite ovp.2019.12. To restrict the breadth of the test scope, it can also be specified using options `--mandatory` or `--optional`.

```
$ dovetail run --mandatory
```

Also there is a `--testcase` option provided to run a specified test case.

```
$ dovetail run --testcase functest.tempest.osinterop
```

Dovetail allows the user to disable strict API response validation implemented by Nova Tempest tests by means of the `--no-api-validation` option. Usage of this option is only advisable if the SUT returns Nova API responses that contain additional attributes. For more information on this command line option and its intended usage, refer to *Disabling Strict API Validation in Tempest*.

```
$ dovetail run --testcase functest.tempest.osinterop --no-api-validation
```

By default, during test case execution, the respective feature is responsible to decide what flavor is going to use for the execution of each test scenario which is under of its umbrella. In parallel, there is also implemented a mechanism in order for the extra specs in flavors of executing test scenarios to be hugepages instead of the default option. This is happening if the name of the scenario contains the substring "ovs". In this case, the flavor which is going to be used for the running test case has 'hugepage' characteristics.

Taking the above into consideration and having in mind that the DEPLOY_SCENARIO environment parameter is not used by dovetail framework (the initial value is 'unknown'), we set as input, for the features that they are responsible for the test case execution, the DEPLOY_SCENARIO environment parameter having as substring the feature name "ovs" (e.g. os-nosdn-ovs-ha).

Note for the users:

- if their system uses DPDK, they should run with `--deploy-scenario <xx-yy-ovs-zz>` (e.g. os-nosdn-ovs-ha)
- this is an experimental feature

```
$ dovetail run --testcase functest.tempest.osinterop --deploy-scenario os-nosdn-ovs-ha
```

By default, results are stored in local files on the Test Host at `$DOVETAIL_HOME/results`. Each time the 'dovetail run' command is executed, the results in the aforementioned directory are overwritten. To create a singular compressed result file for upload to the OVP portal or for archival purposes, the tool provides an option `--report`.

```
$ dovetail run --report
```

If the Test Host is offline, `--offline` should be added to support running with local resources. Otherwise, it will try to download resources online during the run time.

```
$ dovetail run --offline
```

Below is an example of running one test case and the creation of the compressed result file on the Test Host.

```
$ dovetail run --offline --testcase functest.vping.userdata --report
2019-12-04 07:31:13,156 - run - INFO -␣
↪=================================================
2019-12-04 07:31:13,157 - run - INFO - Dovetail compliance: ovp.2019.12!
2019-12-04 07:31:13,157 - run - INFO -␣
↪=================================================
2019-12-04 07:31:13,157 - run - INFO - Build tag: daily-master-0c9184e6-1668-11ea-
↪b1cd-0242ac110002
2019-12-04 07:31:13,610 - run - INFO - >>[testcase]: functest.vping.userdata
2019-12-04 07:31:13,612 - dovetail.test_runner.DockerRunner - WARNING - There is no␣
↪hosts file /home/ovp/pre_config/hosts.yaml. This may cause some issues with domain␣
↪name resolution.
2019-12-04 07:31:14,587 - dovetail.test_runner.DockerRunner - INFO - Get hardware␣
↪info of all nodes list in file /home/ovp/pre_config/pod.yaml ...
2019-12-04 07:31:14,587 - dovetail.test_runner.DockerRunner - INFO - Hardware info of␣
↪all nodes are stored in file /home/dovetail/results/all_hosts_info.json.
2019-12-04 07:31:14,612 - dovetail.container.Container - WARNING - There is no hosts␣
↪file /home/ovp/pre_config/hosts.yaml. This may cause some issues with domain name␣
↪resolution.
2019-12-04 07:32:13,804 - dovetail.report.Report - INFO - Results have been stored␣
↪with files: ['/home/ovp/results/functest_results.txt'].
2019-12-04 07:32:13,808 - dovetail.report.Report - INFO -

Dovetail Report
Version: 2019.12
Build Tag: daily-master-0c9184e6-1668-11ea-b1cd-0242ac110002
Test Date: 2019-12-04 07:32:13 UTC
Duration: 60.20 s

Pass Rate: 100.00% (1/1)
vping:                         pass rate 100.00%
-functest.vping.userdata       PASS
```

When test execution is complete, a tar file with all result and log files is written in `$DOVETAIL_HOME` on the Test Host. An example filename is `${DOVETAIL_HOME}/logs_20191204_0732.tar.gz`. The file is named using a timestamp that follows the convention 'YearMonthDay_HourMinute'. In this case, it was generated at 07:32 on December 4th, 2019. This tar file is used for uploading the logs and results to the OVP portal.

Making Sense of OVP Test Results

When a tester is performing trial runs, Dovetail stores results in local files on the Test Host by default within directory `$DOVETAIL_HOME/results`.

- Log file: dovetail.log
  - Review the dovetail.log to see if all important information has been captured
    - In default mode without DEBUG.
    - Adding option `-d/--debug` to change the mode to be DEBUG.
- Result file: results.json
  - Review the results.json to see all results data including criteria for PASS or FAIL.

- Tempest and security test cases
    - Can see the log details in `tempest_logs/functest.tempest.XXX.html` and `security_logs/functest.security.XXX.html` respectively, which has the passed, skipped and failed test cases results.
        - This kind of files need to be opened with a web browser.
        - The skipped test cases are accompanied with the reason tag for the users to see why these test cases skipped.
        - The failed test cases have rich debug information for the users to see why these test cases failed.
- Vping test cases
    - Its log is stored in `vping_logs/functest.vping.XXX.log`.
- HA test cases
    - Its log is stored in `ha_logs/yardstick.ha.XXX.log`.
- Stress test cases
    - Its log is stored in `stress_logs/bottlenecks.stress.XXX.log`.
- VNF test cases
    - Its log is stored in `vnf_logs/functest.vnf.XXX.log`.

9.1.5 OVP Portal Web Interface

The OVP portal is a public web interface for the community to collaborate on results and to submit results for official OPNFV compliance verification. The portal can be used as a resource by users to navigate and inspect results more easily than by manually inspecting the log files. The portal also allows users to share results in a private manner until they are ready to submit results for peer community review.

- Web Site URL
    - https://nfvi-verified.lfnetworking.org
- Sign In / Sign Up Links
    - Accounts are exposed through Linux Foundation.
    - If you already have a Linux Foundation ID, you can sign in directly with your ID.
    - If you do not have a Linux Foundation ID, you can sign up for a new one using 'Sign Up'.
- My Results Tab
    - This is the primary view where most of the workflow occurs.
    - This page lists all results uploaded by you after signing in.
    - Following the two steps below, the results are uploaded and in status 'private'.
        - Obtain results tar file located at `${DOVETAIL_HOME}/`, e.g. `logs_20180105_0858.tar.gz`.
        - Use the *Choose File* button where a file selection dialog allows you to choose your result file from the hard-disk. Then click the *Upload result* button and see a results ID once your upload succeeds.
    - Results are remaining in status 'private' until they are submitted for review.
    - Use the *Operation* column drop-down option *submit to review*, to expose results to OPNFV community peer reviewers. Use the *withdraw submit* option to reverse this action.
        - Results status are changed to be 'review' after submit to review.

  - Use the *View Reviews* to find the review status including reviewers' names and the outcome.
  - The administrator will approve the results which have got 2 positive outcome from 2 reviewers. Then the status will be changed to be 'verified'.
  - Use the *Operation* column drop-down option *share with* to share results with other users by supplying either the login user ID or the email address associated with the share target account. The result is exposed to the share target but remains private otherwise.

- Profile Tab
  - This page shows your account info after you sign in.
  - There are 3 different roles: administrator, user and reviewer.

9.1.6 Updating Dovetail or a Test Suite

Follow the instructions in section *Installing Dovetail on the Test Host* and *Running the OVP Test Suite* by replacing the docker images with new_tags:

```
sudo docker pull opnfv/dovetail:<dovetail_new_tag>
sudo docker pull opnfv/functest:<functest_new_tag>
sudo docker pull opnfv/yardstick:<yardstick_new_tag>
sudo docker pull opnfv/bottlenecks:<bottlenecks_new_tag>
```

This step is necessary if dovetail software or the OVP test suite have updates.

9.2 Dovetail Command Line Interface Reference

Dovetail command line is to have a simple command line interface in Dovetail to make easier for users to handle the functions that dovetail framework provides.

9.2.1 Commands List

Commands	Action
dovetail –help | -h	Show usage of command “dovetail”
dovetail –version	Show version number
Dovetail List Commands	
dovetail list –help | -h	Show usage of command “dovetail list”
dovetail list	List all available test suites and all test cases within each test suite
dovetail list <test_suite_name>	List all available test areas within test suite <test_suite_name>
Dovetail Show Commands	
dovetail show –help | -h	Show usage of command “dovetail show”
dovetail show <test_case_name>	Show the details of one test case
Dovetail Run Commands	
dovetail run –help | -h	Show usage of command “dovetail run”
dovetail run	Run Dovetail with all test cases within default test suite
dovetail run –testsuite <test_suite_name>	Run Dovetail with all test cases within test suite <test_suite_name>
dovetail run –testsuite <test_suite_name> –testarea <test_area_name>	Run Dovetail with test area <test_area_name> within test suite <test_suite_name>. Test area can be chosen from (vping, tempest, security, ha, stress, bgpvpn, vnf, snaps). Repeat option to set multiple test areas.
dovetail run –testcase <test_case_name>	Run Dovetail with one or more specified test cases. Repeat option to set multiple test cases.
dovetail run –mandatory –testsuite <test_suite_name>	Run Dovetail with all mandatory test cases within test suite <test_suite_name>
dovetail run –optional –testsuite <test_suite_name>	Run Dovetail with all optional test cases within test suite <test_suite_name>
dovetail run –debug | -d	Run Dovetail with debug mode and show all debug logs
dovetail run –offline	Run Dovetail offline, use local docker images instead of download online
dovetail run –report | -r <db_url>	Package the results directory which can be used to upload to OVP web portal
dovetail run –deploy-scenario <deploy_scenario_name>	Specify the deploy scenario having as project name ‘ovs’
dovetail run –no-api-validation	Disable strict API response validation
dovetail run –no-clean | -n	Keep all Containers created for debuging
dovetail run –stop | -s	Stop immediately when one test case failed

9.2.2 Commands Examples

Dovetail Commands

```
root@1f230e719e44:~/dovetail/dovetail# dovetail --help
Usage: dovetail [OPTIONS] COMMAND [ARGS]...

Options:
  --version   Show the version and exit.
```

(continues on next page)

(continued from previous page)

```
  -h, --help  Show this message and exit.

Commands:
  list  list the testsuite details
  run   run the testcases
  show  show the testcases details
```

```
root@1f230e719e44:~/dovetail/dovetail# dovetail --version
dovetail, version 2018.9.0
```

Dovetail List Commands

```
root@1f230e719e44:~/dovetail/dovetail# dovetail list --help
Usage: dovetail list [OPTIONS] [TESTSUITE]

  list the testsuite details

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

```
root@1f230e719e44:~/dovetail/dovetail# dovetail list ovp.2019.12
- mandatory
    functest.vping.userdata
    functest.vping.ssh
    functest.tempest.osinterop
    functest.tempest.compute
    functest.tempest.identity_v3
    functest.tempest.image
    functest.tempest.network_api
    functest.tempest.volume
    functest.tempest.neutron_trunk_ports
    functest.tempest.ipv6_api
    functest.security.patrole
    yardstick.ha.nova_api
    yardstick.ha.neutron_server
    yardstick.ha.keystone
    yardstick.ha.glance_api
    yardstick.ha.cinder_api
    yardstick.ha.cpu_load
    yardstick.ha.disk_load
    yardstick.ha.haproxy
    yardstick.ha.rabbitmq
    yardstick.ha.database
    bottlenecks.stress.ping
- optional
    functest.tempest.ipv6_scenario
    functest.tempest.multi_node_scheduling
    functest.tempest.network_security
    functest.tempest.vm_lifecycle
    functest.tempest.network_scenario
    functest.tempest.bgpvpn
    functest.security.patrole_vxlan_dependent
    yardstick.ha.neutron_l3_agent
    yardstick.ha.controller_restart
```

(continues on next page)

(continued from previous page)

```
    functest.vnf.vims
    functest.vnf.vepc
```

Dovetail Show Commands

```
root@1f230e719e44:~/dovetail/dovetail# dovetail show --help
Usage: dovetail show [OPTIONS] TESTCASE

  show the testcases details

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

```
root@1f230e719e44:~/dovetail/dovetail# dovetail show functest.vping.ssh
---
functest.vping.ssh:
  name: functest.vping.ssh
  objective: testing for vping using ssh
  validate:
    type: functest
    testcase: vping_ssh
    image_name: opnfv/functest-healthcheck
  report:
    source_archive_files:
      - functest.log
    dest_archive_files:
      - vping_logs/functest.vping.ssh.log
    check_results_file:
      - 'functest_results.txt'
    portal_key_file: vping_logs/functest.vping.ssh.log
    sub_testcase_list:
```

```
root@1f230e719e44:~/dovetail/dovetail# dovetail show functest.tempest.image
---
functest.tempest.image:
  name: functest.tempest.image
  objective: tempest smoke test cases about image
  validate:
    type: functest
    testcase: tempest_custom
    pre_condition:
      - 'cp /home/opnfv/userconfig/pre_config/tempest_conf.yaml /usr/lib/python2.7/
↪site-packages/functest/opnfv_tests/openstack/tempest/custom_tests/tempest_conf.yaml'
      - 'cp /home/opnfv/userconfig/tempest_custom_testcases.yaml /usr/lib/python2.7/
↪site-packages/xtesting/ci/testcases.yaml'
      - 'cp /home/opnfv/functest/results/tempest_custom.txt /usr/lib/python2.7/site-
↪packages/functest/opnfv_tests/openstack/tempest/custom_tests/test_list.txt'
  report:
    source_archive_files:
      - functest.log
      - tempest_custom/rally.log
      - tempest_custom/tempest-report.html
    dest_archive_files:
      - tempest_logs/functest.tempest.image.functest.log
```

(continues on next page)

(continued from previous page)

```
      - tempest_logs/functest.tempest.image.log
      - tempest_logs/functest.tempest.image.html
    check_results_file:
      - 'functest_results.txt'
    portal_key_file: tempest_logs/functest.tempest.image.html
    sub_testcase_list:
      - tempest.api.image.v2.test_images.BasicOperationsImagesTest.test_register_
↪upload_get_image_file[id-139b765e-7f3d-4b3d-8b37-3ca3876ee318,smoke]
      - tempest.api.image.v2.test_versions.VersionsTest.test_list_versions[id-
↪659ea30a-a17c-4317-832c-0f68ed23c31d,smoke]
```

Dovetail Run Commands

```
root@1f230e719e44:~/dovetail/dovetail# dovetail run --help
Usage: run.py [OPTIONS]

Dovetail compliance test entry!

Options:
--opnfv-ci               Only enabled when running with OPNFV CI jobs and pushing␣
↪results to TestAPI DB
--optional               Run all optional test cases.
--mandatory              Run all mandatory test cases.
--deploy-scenario TEXT   Specify the DEPLOY_SCENARIO which will be used as input by␣
↪each testcase respectively
-n, --no-clean           Keep all Containers created for debuging.
--no-api-validation      disable strict API response validation
--offline                run in offline method, which means not to update the docker␣
↪upstream images, functest, yardstick, etc.
-r, --report             Create a tarball file to upload to OVP web portal
-s, --stop               Flag for stopping on test case failure.
-d, --debug              Flag for showing debug log on screen.
--testcase TEXT          Compliance testcase. Specify option multiple times to include␣
↪multiple test cases.
--testarea TEXT          Compliance testarea within testsuite. Specify option multiple␣
↪times to include multiple test areas.
--testsuite TEXT         compliance testsuite.
-h, --help               Show this message and exit.
```

```
root@1f230e719e44:~/dovetail/dovetail# dovetail run --testcase functest.vping.ssh --
↪offline -r --deploy-scenario os-nosdn-ovs-ha
2019-12-06 02:51:52,634 - run - INFO -␣
↪================================================
2019-12-06 02:51:52,634 - run - INFO - Dovetail compliance: ovp.2019.12!
2019-12-06 02:51:52,634 - run - INFO -␣
↪================================================
2019-12-06 02:51:52,634 - run - INFO - Build tag: daily-master-5b58584a-17d3-11ea-
↪878a-0242ac110002
2019-12-06 02:51:52,634 - run - INFO - DEPLOY_SCENARIO : os-nosdn-ovs-ha
2019-12-06 02:51:53,077 - run - INFO - >>[testcase]: functest.vping.ssh
2019-12-06 02:51:53,078 - dovetail.test_runner.DockerRunner - WARNING - There is no␣
↪hosts file /home/ovp/pre_config/hosts.yaml. This may cause some issues with domain␣
↪name resolution.
2019-12-06 02:51:54,048 - dovetail.test_runner.DockerRunner - INFO - Get hardware␣
↪info of all nodes list in file /home/ovp/pre_config/pod.yaml ...
```

(continues on next page)

(continued from previous page)

```
2019-12-06 02:51:54,049 - dovetail.test_runner.DockerRunner - INFO - Hardware info of␣
↪all nodes are stored in file /home/dovetail/results/all_hosts_info.json.
2019-12-06 02:51:54,073 - dovetail.container.Container - WARNING - There is no hosts␣
↪file /home/ovp/pre_config/hosts.yaml. This may cause some issues with domain name␣
↪resolution.
2019-12-06 02:52:57,982 - dovetail.report.Report - INFO - Results have been stored␣
↪with files: ['/home/ovp/results/functest_results.txt'].
2019-12-06 02:52:57,986 - dovetail.report.Report - INFO -

Dovetail Report
Version: 2019.12
Build Tag: daily-master-5b58584a-17d3-11ea-878a-0242ac110002
Test Date: 2019-12-06 02:52:57 UTC
Duration: 64.91 s

Pass Rate: 100.00% (1/1)
vping:                       pass rate 100.00%
-functest.vping.ssh          PASS
```

9.3 Running Dovetail by RESTful API

9.3.1 Overview

Dovetail framework provides RESTful APIs for end users to run all OVP test cases. Also it provides a Swagger UI for users to find out all APIs and try them out.

9.3.2 Definitions and abbreviations

- REST - Representational State Transfer
- API - Application Programming Interface
- OVP - OPNFV Verification Program
- UI - User Interface

9.3.3 Environment Preparation

Install Docker

The main prerequisite software for Dovetail is Docker. Please refer to official Docker installation guide that is relevant to your Test Host's operating system.

Configuring the Test Host Environment

For convenience and as a convention, we will create a home directory for storing all Dovetail related config items and results files:

```
$ mkdir -p ${HOME}/dovetail
$ export DOVETAIL_HOME=${HOME}/dovetail
```

9.3.4 Installing Dovetail API

The Dovetail project maintains a Docker image that has both Dovetail API and Dovetail CLI preinstalled. This Docker image is tagged with versions. Before pulling the Dovetail image, check the OPNFV's OVP web page first to determine the right tag for OVP testing.

Downloading Dovetail Docker Image

The first version of Dovetail API is ovp-3.0.0.

```
$ sudo docker pull opnfv/dovetail:ovp-3.0.0
ovp-3.0.0: Pulling from opnfv/dovetail
6abc03819f3e: Pull complete
05731e63f211: Pull complete
0bd67c50d6be: Pull complete
3f737f5d00b2: Pull complete
c93fd0792ebd: Pull complete
77d9a9603ec6: Pull complete
9463cdd9c628: Pull complete
Digest: sha256:45e2ffdbe217a4e6723536afb5b6a3785d318deff535da275f34cf8393af458d
Status: Downloaded newer image for opnfv/dovetail:ovp-3.0.0
```

Deploying Dovetail API

The Dovetail API can be deployed by running a Dovetail container with the Docker image downloaded before.

```
$ docker run -itd -p <swagger_port>:80 -p <api_port>:5000 --privileged=true \
  -e SWAGGER_HOST=<host_ip>:<api_port> -e DOVETAIL_HOME=/home/ovp \
  -v /home/ovp:/home/ovp -v /var/run/docker.sock:/var/run/docker.sock \
  opnfv/dovetail:<version>
```

In the container, it uses 2 ports for Swagger UI (port 80) and API (port 5000) respectively. So in order to access to these 2 services outside the container, it needs to map them to the host ports. It can be any available ports in the host.

The env SWAGGER_HOST is optional. If you will access the Swagger UI webpage with the same host deploying this container, there is no need to set SWAGGER_HOST. Otherwise, if you will access the Swagger UI webpage from other machines, then it needs to set SWAGGER_HOST.

9.3.5 Using Dovetail API

Here give the guide of where to find out all APIs and how to use them.

Swagger UI Webpage

After deploying Dovetail container, the Swagger UI webpage can be accessed with any browser. The url is `http://localhost:<swagger_port>/dovetail-api/index.html` if accessing from the same host as deploying this container. Otherwise, the url is `http://<host_ip>:<swagger_port>/dovetail-api/index.html`.

Calling APIs

There are totally 5 APIs provided by Dovetail.

- Get all test suites
- Get all test cases
- Run test cases
- Run test cases with execution ID
- Get status of test cases

Here give some easy guide of how to call these APIs. For more detailed infomation, please refer to the Swagger UI page.

Getting All Test Suites

- This is a **GET** function with no parameter to get all test suites defined in Dovetail container.
- The request URL is `http://<host_ip>:<api_port>/api/v1/scenario/nfvi/testsuites`.
- The response body is structured as:

```
{
  "testsuites": {
    "debug": {
      "name": "debug",
      "testcases_list": {
        "optional": [
          "functest.vping.userdata"
        ]
      }
    },
    "healthcheck": {
      "name": "healthcheck",
      "testcases_list": {
        "optional": [
          "functest.healthcheck.connection_check"
        ]
      }
    }
  }
}
```

Getting All Test Cases

- This is a **GET** function without no parameter to get all test cases integrated in Dovetail container.
- The request URL is `http://<host_ip>:<api_port>/api/v1/scenario/nfvi/testcases`.
- The response body is structured as:

```
{
  "testcases": [
    {
      "description": "This test case will verify the high availability of the␣
↪user service provided by OpenStack (keystone) on control node.",
```

(continues on next page)

(continued from previous page)

```
      "scenario": "nfvi",
      "subTestCase": null,
      "testCaseName": "yardstick.ha.keystone"
    },
    {
      "description": "testing for vping using userdata",
      "scenario": "nfvi",
      "subTestCase": null,
      "testCaseName": "functest.vping.userdata"
    },
    {
      "description": "tempest smoke test cases about volume",
      "scenario": "nfvi",
      "subTestCase": [
        "tempest.api.volume.test_volumes_actions.VolumesActionsTest.test_attach_
↪detach_volume_to_instance[compute,id-fff42874-7db5-4487-a8e1-ddda5fb5288d,smoke]
↪",
        "tempest.scenario.test_volume_boot_pattern.TestVolumeBootPattern.test_
↪volume_boot_pattern[compute,id-557cd2c2-4eb8-4dce-98be-f86765ff311b,image,slow,
↪volume]"
      ],
      "testCaseName": "functest.tempest.volume"
    }
  ]
}
```

Running Test Cases

- This is a **POST** function with some parameters to run a subset of the whole test cases.
- The request URL is `http://<host_ip>:<api_port>/api/v1/scenario/nfvi/execution`.
- The request body is structured as following. The `conf` section is used to give all configuration items those are required to run test cases. They are the same as all configuration files provided under `$DOVETAIL_HOME/pre_config/`. If you already have these files under this directory, the whole `conf` section can be ignored. If you provide these configuration items with the request body, then the corresponding files under `$DOVETAIL_HOME/pre_config/` will be ignored by Dovetail. The `testcase`, `testsuite`, `testarea` and `deploy_scenario` correspond to `--testcase`, `--testsuite`, `--testarea` and `--deploy-scenario` defined with Dovetail CLI. The `options` section support to set all options which have already been implemented by Dovetail CLI including `--optional`, `--mandatory`, `--no-clean`, `--no-api-validation`, `--offline`, `--report`, `--stop` and `--debug`. For options list in `options` section, they are set to be `True`, otherwise, they are set to be `False`.

```
{
  "conf": {
    "vm_images": "/home/ovp/images",
    "pods": {
      "nodes": [
        {
          "name": "node1",
          "role": "Controller",
          "ip": "192.168.117.222",
          "user": "root",
          "password": "root",
        }
```

(continues on next page)

(continued from previous page)

```
      ],
      "process_info": [
        {
          "testcase_name": "yardstick.ha.rabbitmq",
          "attack_host": "node1",
          "attack_process": "rabbitmq"
        }
      ]
    },
    "tempest_conf": {
      "compute": {
        "min_compute_nodes": "2",
        "volume_device_name": "vdb",
        "max_microversion": "2.65"
      }
    },
    "hosts": {
      "192.168.141.101": [
        "volume.os.com",
        "compute.os.com"
      ]
    },
    "envs": {
      "OS_USERNAME": "admin",
      "OS_PASSWORD": "admin",
      "OS_AUTH_URL": "https://192.168.117.222:5000/v3",
      "EXTERNAL_NETWORK": "ext-net"
    }
  },
  "testcase": [
    "functest.vping.ssh",
    "yardstick.ha.rabbitmq"
  ],
  "testsuite": "ovp.2019.12",
  "testarea": [
    "vping",
    "ha"
  ],
  "deploy_scenario": "os-nosdn-ovs-ha",
  "options": [
    "debug",
    "report"
  ]
}
```

- The response body is structured as:

```
{
  "result": [
    {
      "endTime": null,
      "executionId": "a65e24c0-1803-11ea-84f4-0242ac110004",
      "results": null,
      "scenario": "nfvi",
      "status": "IN_PROGRESS",
      "testCaseName": "functest.vping.ssh",
      "testSuiteName": "ovp.2019.12",
```

(continues on next page)

(continued from previous page)

```
      "timestart": null
    }
  ]
}
```

Running Test Cases with Execution ID

- This is a **POST** function with some parameters to run a subset of whole test cases and set the execution ID instead of using the random one.
- The request URL is `http://<host_ip>:<api_port>/api/v1/scenario/nfvi/execution/{exec_id}`.
- It's almost the same as the above running test cases API except the execution ID.

Getting Status of Test Cases

- This is a **POST** function to get the status of some test cases by using the execution ID received in the response body of *Running Test Cases* or *Running Test Cases with Execution ID* APIs.
- The request URL is `http://<host_ip>:<api_port>/api/v1/scenario/nfvi/execution/status/{exec_id}`.
- The request body is structured as:

```
{
  "testcase": [
    "functest.vping.ssh"
  ]
}
```

- The response body is structured as:

```
{
  "result": [
    {
      "endTime": "2019-12-06 08:39:23",
      "executionId": "a65e24c0-1803-11ea-84f4-0242ac110004",
      "results": {
        "criteria": "PASS",
        "sub_testcase": [],
        "timestart": "2019-12-06 08:38:40",
        "timestop":"2019-12-06 08:39:23"
      },
      "scenario": "nfvi",
      "status": "COMPLETED",
      "testCaseName": "functest.vping.ssh",
      "testSuiteName": "ovp.2019.12",
      "timestart":"2019-12-06 08:38:40"
    }
  ]
}
```

Getting Test Results

Each time you call the running test case API, Dovetail creates a directory with the execution ID as the name under `$DOVETAIL_HOME` to store results on the host. You can find all result files under `$DOVETAIL_HOME/<executionId>/results`. If you run test cases with `report` option, then there will be a tarball file under `$DOVETAIL_HOME/<executionId>` which can be upload to OVP portal.

9.4 Conducting ONAP VNF Testing for OVP

9.4.1 Overview

As the LFN verification framework, the Dovetail team has worked with the ONAP VVP, and VTP projects to enable VNF testing, results submission, and results review to be completed throught the same web portal and processes used for the NFVI testing. For more details about VNF SDK and VVP, please refer to ONAP VNF SDK Compliance Verification Program and ONAP VVP.

Testing is available for both HEAT and TOSCA defined VNFs, but the process is different depending on the template language. This userguide covers the testing process for both VNF types in the two sections below.

9.4.2 Definitions and abbreviations

- LFN - Linux Foundation Networking
- ONAP - Open Network Automation Platform
- OVP - OPNFV Verification Program
- VNF - Virtual Network Function
- VNF SDK - VNF Software Development Kit
- VTP - VNF Test Platform
- VVP - VNF Validation Program

9.4.3 Testing of HEAT based VNFs

Environment Preparation

Prerequisites

- ONAP ElAlto Release deployed via OOM
- An OpenStack deployment is available and privisioned as ONAP's Cloud Site
- kubectl is installed on the system used to start the testing
- bash
- VNF Heat Templates
- Preload json files

After deploying ONAP, you need to configure ONAP with:

- A cloud owner

- A cloud region
- A subscriber
- A service type
- A project name
- An owning entity
- A platform
- A line of business
- A cloud site

If you're not familiar with how to configure ONAP, there are guides that use robot or direct api requests available to help, as well as a guide for adding a new OpenStack site to ONAP.

VVP Test Tool Setup

On your local machine, or the system from which you will run the tests, you will need to clone the ONAP OOM project repo:

```
git clone --branch 5.0.1-ONAP ssh://<username>@gerrit.onap.org:29418/oom --recurse-
↪submodules
```

VNF Preparation

The vnf lifecycle validation testsuite requires the VNF to be packaged into a specific directory hierarchy, shown below.

```
vnf_folder
├── /templates
|   └── base.yaml
|   └── base.env
|   └── incremental_0.yaml
|   └── incremental_0.env
|   └── ...
├── /preloads
|   └── base_preload.json
|   └── incremental_0_preload.json
|   └── ...
└── vnf-details.json
```

- The name for vnf_folder is free-form, and can be located anywhere on your computer. The path to this folder will be passed to the testsuite as an argument.
- /templates should contain your VVP-compliant VNF heat templates.
- **/preloads should contain a preload file for each VNF module (TODO: add link to preload documentation).**
    - For a VNF-API preload: vnf-name, vnf-type, generic-vnf-type, and generic-vnf-name should be empty strings.
    - For a GR-API preload: vnf-name, vnf-type, vf-module-type, and vf-module-name should be empty strings.
    - This information will be populated at runtime by the testsuite.

- vnf-details should be a json file with the information that will be used by ONAP to instantiate the VNF. The structure of vnf-details is shown below.
- VNF disk image must be uploaded and available in the OpenStack project being managed by ONAP
- Modules must contain an entry for each module of the VNF. Only one module can be a base module.
- api_type should match the format of the preloads that are provided in the package.
- The other information should match what was used to configure ONAP during the pre-requisite section of this guide.

```
{
    "vnf_name": "The Vnf Name",
    "description": "Description of the VNF",
    "modules": [
        {
        "filename": "base.yaml",
        "isBase": "true",
        "preload": "base_preload.json"
        },
        {
        "filename": "incremental_0.yaml",
        "isBase": "false",
        "preload": "incremental_0.json"
        },
        ...
    ],
    "api_type": "[gr_api] or [vnf_api]",
    "subscriber": "<subscriber name>",
    "service_type": "<service type>",
    "tenant_name": "<name of tenant>",
    "region_id": "<name of region>",
    "cloud_owner": "<name of cloud owner>",
    "project_name": "<name of project>",
    "owning_entity": "<name of owning entity>",
    "platform": "<name of platform>",
    "line_of_business": "<name of line of business>",
    "os_password": "<openstack password>"
}
```

Runnign the HEAT VNF Test

The ONAP OOM Robot framework will run the test, using kubectl to manage the execution. The framework will copy your VNF template files to the robot container required to execute the test.

```
 cd oom/kubernetes/robot
$ ./instantiate-k8s.sh --help
./instantiate-k8s.sh [options]

required:
-n, --namespace <namespace>       namespace that robot pod is running under.
-f, --folder <folder>             path to folder containing heat templates, preloads,␣
↪and vnf-details.json.

additional options:
-p, --poll                        some cloud environments (like azure) have a short␣
↪time out value when executing
```

(continues on next page)

(continued from previous page)

```
                                  kubectl. If your shell exits before the testsuite␣
↪finishes, using this option
                                  will poll the testsuite logs every 30 seconds until␣
↪the test finishes.
-t, --tag <tag>                   robot testcase tag to execute (default is␣
↪instantiate_vnf).

This script executes the VNF instantiation robot testsuite.
- It copies the VNF folder to the robot container that is part of the ONAP deployment.
- It models, distributes, and instantiates a heat-based VNF.
- It copies the logs to an output directory, and creates a tarball for upload to the␣
↪OVP portal.
```

Sample execution:

```
$ ./instantiate-k8s.sh --namespace onap --folder /tmp/vnf-instantiation/examples/VNF_
↪API/pass/multi_module/ --poll
...
...
...
...
------------------------------------------------------------------------------
Testsuites.Vnf Instantiation :: The main driver for instantiating ... | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Testsuites                                                            | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /share/logs/0003_ete_instantiate_vnf/output.xml
+ set +x
testsuite has finished
Copying Results from pod...
/tmp/vnf-instantiation /tmp/vnf-instantiation
a log.html
a results.json
a stack_report.json
a validation-scripts.json
/tmp/vnf-instantiation
VNF test results: /tmp/vnfdata.46749/vnf_heat_results.tar.gz
```

The testsuite takes about 10-15 minutes for a simple VNF, and will take longer for a more complicated VNF.

Reporting Results

Once the testsuite is finished, it will create a directory and tarball in /tmp (the name of the directory and file is shown at the end of the stdout of the script). There will be a results.json in that directory that has the ultimate outcome of the test, in the structure shown below.

Log Files

The output tar file will have 4 log files in it.

- results.json: This is high-level results file of all of the test steps, and is consumed by the OVP portal.
- report.json: This is the output of the vvp validation scripts.

- stack_report.json: This is the output from querying openstack to validate the heat modules.
- log.html: This is the robot log, and contains each execution step of the testcase.

If the result is “PASS”, that means the testsuite was successful and the tarball is ready for submission to the OVP portal.

results.json

```
{
    "vnf_checksum": "afc57604a3b3b7401d5b8648328807b594d7711355a2315095ac57db4c334a50
↪",
    "build_tag": "vnf-validation-53270",
    "version": "2019.09",
    "test_date": "2019-09-04 17:50:10.575",
    "duration": 437.002,
    "vnf_type": "heat",
    "testcases_list": [
        {
            "mandatory": "true",
            "name": "onap-vvp.validate.heat",
            "result": "PASS",
            "objective": "onap heat template validation",
            "sub_testcase": [],
            "portal_key_file": "report.json"
        },
        {
            "mandatory": "true",
            "name": "onap-vvp.lifecycle_validate.heat",
            "result": "PASS",
            "objective": "onap vnf lifecycle validation",
            "sub_testcase": [
                {
                    "name": "model-and-distribute",
                    "result": "PASS"
                },
                {
                    "name": "instantiation",
                    "result": "PASS"
                }
            ],
            "portal_key_file": "log.html"
        },
        {
            "mandatory": "true",
            "name": "stack_validation",
            "result": "PASS",
            "objective": "onap vnf openstack validation",
            "sub_testcase": [],
            "portal_key_file": "stack_report.json"
        }
    ]
}
```

Additional Resources

- ONAP VVP Project
- VVP Wiki Users Guide (this will track current ONAP master)

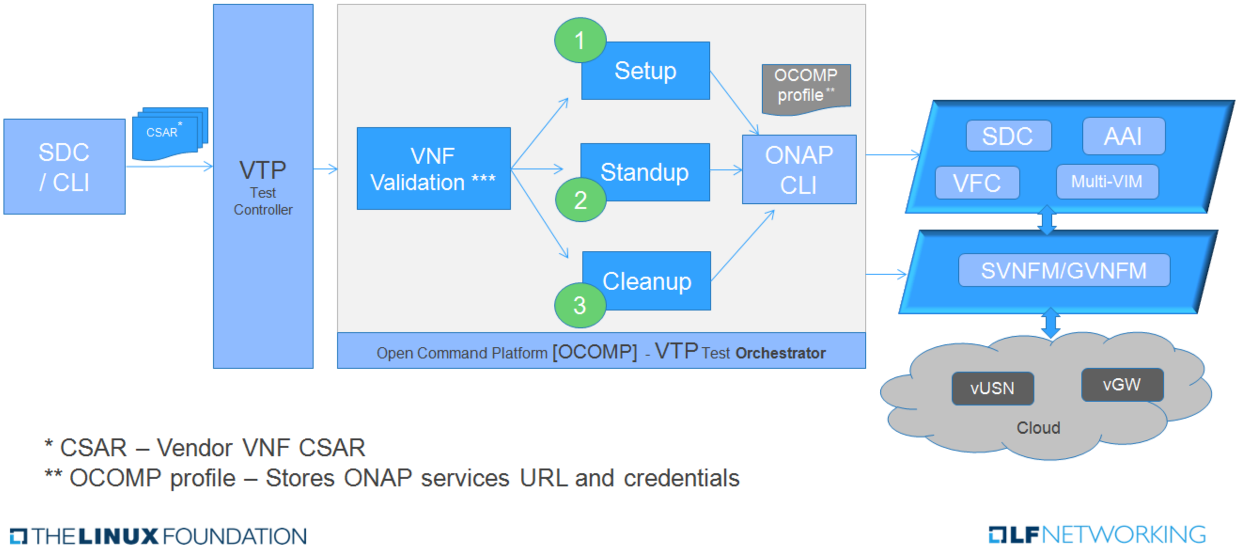
Sample VNF templates are available on the VVP Wiki Users Guide page.

9.4.4 Testing of TOSCA based VNFs

VNF Test Platform (VTP) provides an platform to on-board different test cases required for OVP for various VNF testing provided by VNFSDK (for TOSCA) and VVP(for HEAT) projects in ONAP. And it generates the test case outputs which would be uploaded into OVP portal for VNF badging.

TOSCA VNF Test Environment

As pre-requestiests steps, Its assumed that, successful ONAP, Vendor VNFM and OpenStack cloud are already available. Below installation steps help to setup VTP components and CLI.



* CSAR – Vendor VNF CSAR
** OCOMP profile – Stores ONAP services URL and credentials

Installation

Clone the VNFSDK repo.

```
git clone --branch elalto https://git.onap.org/vnfsdk/refrepo
```

Install the VTP by using script *refrepo/vnfmarket-be/deployment/install/vtp_install.sh*

Follow the steps as below (in sequence):

- vtp_install.sh –download : It will download all required artifacts into /opt/vtp_stage
- vtp_install.sh –install : It will install VTP (/opt/controller) and CLI (/opt/oclip)
- vtp_install.sh –start : It will start VTP controller as tomcat service and CLI as oclip service
- vtp_install.sh –verify : It will verify the setup is done properly by running some test cases.

Last step (verify) would check the health of VTP components and TOSCA VNF compliance and validation test cases.

Check Available Test Cases

VTP supports to check the compliance of VNF and PNF based on ONAP VNFREQS.

To check:

- Go to command console
- Run command oclip
- Now it will provide a command prompt:

oclip:open-cli>

Now run command as below and check the supported compliance testcases for VNFREQS.

- csar-validate - Helps to validate given VNF CSAR for all configured VNFREQS.
- csar-validate-rxxx - Helps to validate given VNF CSAR for a given VNFREQS xxx.

```
oclip:open-cli>schema-list --product onap-dublin --service vnf-compliance
+--------------+----------------+------------------------+--------------+----------+--
↪----+
|product       |service         |command                 |ocs-version   |enabled   ␣
↪|rpc   |
+--------------+----------------+------------------------+--------------+----------+--
↪----+
|onap-dublin   |vnf-compliance  |csar-validate-r10087    |1.0           |true      | ␣
↪    |
+--------------+----------------+------------------------+--------------+----------+--
↪----+
|onap-dublin   |vnf-compliance  |csar-validate           |1.0           |true      | ␣
↪    |
+--------------+----------------+------------------------+--------------+----------+--
↪----+
|onap-dublin   |vnf-compliance  |csar-validate-r26885    |1.0           |true      | ␣
↪    |
+--------------+----------------+------------------------+--------------+----------+--
↪----+
|onap-dublin   |vnf-compliance  |csar-validate-r54356    |1.0           |true      | ␣
↪    |
...
```

To know the details of each VNFREQS, run as below.

```
oclip:open-cli>use onap-dublin
oclip:onap-dublin>csar-validate-r54356 --help
usage: oclip csar-validate-r54356

Data types used by NFV node and is based on TOSCA/YAML constructs specified in draft␣
↪GS NFV-SOL 001.
The node data definitions/attributes used in VNFD MUST comply.
```

Now run command as below and check the supported validation testcases

```
oclip:onap-dublin>use open-cli
oclip:open-cli>schema-list --product onap-dublin --service vnf-validation
+--------------+----------------+----------------------+--------------+----------+----
↪--+
|product       |service         |command               |ocs-version   |enabled   |rpc␣
↪  |
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------+----------------------+--------------+----------+----
↪--+
|onap-dublin   |vnf-validation  |vnf-tosca-provision   |1.0           |true      |   ␣
↪  |
+--------------+----------------+----------------------+--------------+----------+----
↪--+
```

Configure ONAP with required VNFM and cloud details

1. Setup the OCOMP profile onap-dublin

Run following command to configure the ONAP service URL and creadentials as given below, which will be used by VTP while executing the test cases

```
oclip:open-cli>use onap-dublin
oclip:onap-dublin>profile onap-dublin
oclip:onap-dublin>set sdc.onboarding:host-url=http://159.138.8.8:30280
oclip:onap-dublin>set sdc.onboarding:host-username=cs0008
oclip:onap-dublin>set sdc.onboarding:host-password=demo123456!
oclip:onap-dublin>set sdc.catalog:host-url=http://159.138.8.8:30205
oclip:onap-dublin>set sdc.catalog:host-password=demo123456\!
oclip:onap-dublin>set sdc.catalog:host-username=cs0008
oclip:onap-dublin>set sdc.catalog:service-model-approve:host-username=gv0001
oclip:onap-dublin>set sdc.catalog:service-model-distribute:host-username=op0001
oclip:onap-dublin>set sdc.catalog:service-model-test-start:host-username=jm0007
oclip:onap-dublin>set sdc.catalog:service-model-test-accept:host-username=jm0007
oclip:onap-dublin>set sdc.catalog:service-model-add-artifact:host-username=ocomp
oclip:onap-dublin>set sdc.catalog:vf-model-add-artifact:host-username=ocomp
oclip:onap-dublin>set aai:host-url=https://159.138.8.8:30233
oclip:onap-dublin>set aai:host-username=AAI
oclip:onap-dublin>set aai:host-password=AAI
oclip:onap-dublin>set vfc:host-url=http://159.138.8.8:30280
oclip:onap-dublin>set multicloud:host-url=http://159.138.8.8:30280
```

NOTE: Mostly all above entries value would be same execept the IP address used in the URL, which would be ONAP k8s cluser IP.

By default, SDC onboarding service does not provide node port, which is available to access from external ONAP network. so to enable for external access, register the SDC onboarding service into MSB and use MSB url for sdc.onboarding:host-url.

```
oclip:onap-dublin> microservice-create --service-name sdcob --service-version v1.0 --
↪service-url /onboarding-api/v1.0 --path /onboarding-api/v1.0 --node-ip 172.16.1.0 --
↪node-port 8081
```

NOTE: To find the node-ip and node-port, use the following steps.

Find out SDC onboarding service IP and port details as given here:

```
[root@onap-dublin-vfw-93996-50c1z ~]# kubectl get pods -n onap -o wide | grep sdc-
↪onboarding-be
dev-sdc-sdc-onboarding-be-5564b877c8-vpwr5 2/2 Running 0 29d 172.16.1.0 192.168.2.163
↪<none> <none>
dev-sdc-sdc-onboarding-be-cassandra-init-mtvz6 0/1 Completed 0 29d 172.16.0.220 192.
↪168.2.163 <none> <none>
[root@onap-dublin-vfw-93996-50c1z ~]#
```

Note down the IP address for sdc-onboarding-be 172.16.1.0

```
[root@onap-dublin-vfw-93996-50c1z ~]# kubectl get services -n onap -o wide | grep sdc-
↪onboarding-be
sdc-onboarding-be ClusterIP 10.247.198.92 <none> 8445/TCP,8081/TCP 29d app=sdc-
↪onboarding-be,release=dev-sdc
[root@onap-dublin-vfw-93996-50c1z ~]#
```

Note down the port for sdc-onboarding-be 8445 8081

Similarly, other service IP and Port could be discovered like above, in case not know earlier :)

Verify these details once by typing 'set'

```
oclip:onap-dublin> set
```

This profile would be used by user while running the test cases with ONAP setup configured in it, as below oclip –profile onap-dublin vnf-tosca-provision . . . .

2. Setup SDC consumer

SDC uses consumer concept to configure required VN model and service model artifacts. So following commands required to run, which will create consumer named ocomp, which is already configured in onap-dublin profile created in above steps.

```
oclip --product onap-dublin --profile onap-dublin sdc-consumer-create --consumer-name␣
↪ocomp
```

NOTE: command oclip could be used in scripting mode as above or in interactive mode as used in earlier steps

3. Update the cloud and vnfm driver details

In the configuration file /opt/oclip/conf/vnf-tosca-provision.json, update the cloud and VNFM details.

```
"cloud": {
        "identity-url": "http://10.12.11.1:5000/v3",
        "username": "admin",
        "password": "password",
        "region": "RegionOVP",
        "version": "ocata",
        "tenant": "ocomp"
    },
    "vnfm":{
        "hwvnfmdriver":{
            "version": "v1.0",
            "url": "http://159.138.8.8:38088",
            "username": "admin",
            "password": "xxxx"
        },
        "gvnfmdriver":{
            "version": "v1.0",
            "url": "http://159.138.8.8:30280"
        }
    }
```

4.Configure the decided VNFRES (optional) VTP allows to configure the set of VNFREQS to be considered while running the VNF compliance test cases in the configuration file /opt/oclip/conf/vnfreqs.properties.

If not available, please create this file with following entries:

```
vnfreqs.enabled=r02454,r04298,r07879,r09467,r13390,r23823,r26881,r27310,r35851,r40293,
↪r43958,r66070,r77707,r77786,r87234,r10087,r21322,r26885,r40820,r35854,r65486,r17852,
↪r46527,r15837,r54356,r67895,r95321,r32155,r01123,r51347,r787965,r130206
pnfreqs.enabled=r10087,r87234,r35854,r15837,r17852,r293901,r146092,r57019,r787965,
↪r130206
# ignored all chef and ansible related tests
vnferrors.ignored=
pnferrors.ignored=
```

Runnign the TOSCA VNF Test

Every test provided in VTP is given with guidelines on how to use it. On every execution of test cases, use the following additional arguments based on requirements

- –product onap-dublin - It helps VTP choose the test cases written for onap-dublin version
- –profile onap-dublin - It helps VTP to use the profile settings provided by admin (optional)
- –request-id - It helps VTP to track the progress of the test cases execution and user could use this id for same. (optional)

So, final test case execution would be as below. To find the test case arguments details, run second command below.

```
oclip --product onap-dublin --profile onap-dublin --request-id req-1 <test case name>
↪<test case arguments>
oclip --product onap-dublin <test case name> --help
```

Running TOSCA VNF Compliance Testing

To run compliance test as below with given CSAR file

It will produce the result format as below:

```
{
    "date": "Fri Sep 20 17:34:24 CST 2019",
    "criteria": "PASS",
    "contact": "ONAP VTP Team onap-discuss@lists.onap.org",
    "results": [
    {
        "description": "V2.4.1 (2018-02)",
        "passed": true,
        "vnfreqName": "SOL004",
        "errors": []
    },
    {
        "description": "If the VNF or PNF CSAR Package utilizes Option 2 for package␣
↪security, then the complete CSAR file MUST be digitally signed with the VNF or PNF␣
↪provider private key. The VNF or PNF provider delivers one zip file consisting of␣
↪the CSAR file, a signature file and a certificate file that includes the VNF or PNF␣
↪provider public key. The certificate may also be included in the signature␣
↪container, if the signature format allows that. The VNF or PNF provider creates a␣
↪zip file consisting of the CSAR file with .csar extension, signature and␣
↪certificate files. The signature and certificate files must be siblings of the CSAR␣
↪file with extensions .cms and .cert respectively.\n",
        "passed": true,
        "vnfreqName": "r787965",
```

(continues on next page)

(continued from previous page)

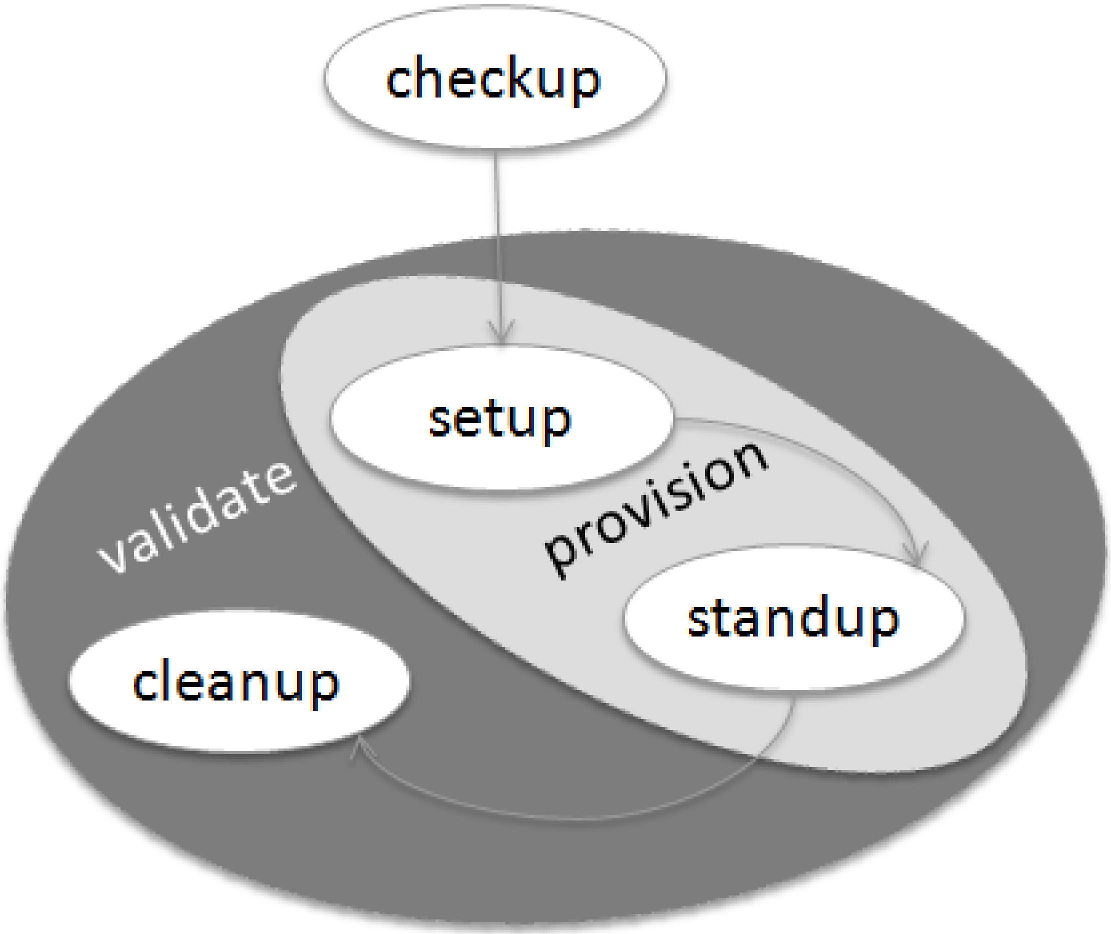
```
        "errors": []
    }
    ],
    "platform": "VNFSDK - VNF Test Platform (VTP) 1.0",
    "vnf": {
    "mode": "WITH_TOSCA_META_DIR",
    "vendor": "ONAP",
    "name": null,
    "type": "TOSCA",
    "version": null
    }
}
```

In case of errors, the errors section will have list of details as below. Each error block, will be given with error code and error details. Error code would be very useful to provide the troubleshooting guide in future. Note, to generate the test result in OVP archieve format, its recommended to run this compliance test with request-id similar to running validation test as below.

```
[
{
    "vnfreqNo": "R66070",
    "code": "0x1000",
    "message": "MissinEntry-Definitions file",
    "lineNumber": -1
}
]
```

Running TOSCA VNF Validation Testing

VTP provides validation test case with following modes:



setup: Create requires Vendor, Service Subscription and VNF cloud in ONAP standup: From the given VSP csar, VNF csar and NS csar, it creates VF Model, NS Model and NS service cleanup: Remove those entries created during provision provision: Runs setup -> standup validate: Runs setup -> standup -> cleanup checkup: mode helps to verify automation is deployed properly.

For OVP badging, validate mode would be used as below:

```
oclip --request-id WkVVu9fD--product onap-dublin --profile onap-dublin vnf-tosca-
 ↪provision --vsp <vsp csar> --vnf-csar <v
```

Validation testing would take for a while to complete the test execution, so user could use the above given request-id, to tracking the progress as below:

```
oclip execution-list --request-id WkVVu9fD
+------------+------------------------+--------------+------------------+-------------
 ↪-----------------+--------------+------------+--------------------------+-----------
 ↪---------------+
|request-id  |execution-id            |product       |service           |command     ␣
 ↪                 |profile       |status      |start-time                |end-time  ␣
 ↪               |
+------------+------------------------+--------------+------------------+-------------
 ↪-----------------+--------------+------------+--------------------------+-----------
 ↪---------------+
|WkVVu9fD    |WkVVu9fD-1568731678753  |onap-dublin   |vnf-validation    |vnf-tosca-
 ↪provision           |              |in-progress |2019-09-17T14:47:58.000   |      ␣
 ↪                |
```

(continues on next page)

(continued from previous page)

```
+------------+------------------------+--------------+------------------+-------------
↪-----------------+--------------+------------+--------------------------+-----------
↪---------------+
|WkVVu9fD    |WkVVu9fD-1568731876397  |onap-dublin   |sdc.catalog       |service-
↪model-test-request     |onap-dublin   |in-progress |2019-09-17T14:51:16.000   |      ␣
↪                          |
+------------+------------------------+--------------+------------------+-------------
↪-----------------+--------------+------------+--------------------------+-----------
↪---------------+
|WkVVu9fD    |WkVVu9fD-1568731966966  |onap-dublin   |sdc.onboarding    |vsp-archive ␣
↪                |onap-dublin   |completed   |2019-09-17T14:52:46.000   |2019-09-
↪17T14:52:47.000   |
+------------+------------------------+--------------+------------------+-------------
↪-----------------+--------------+------------+--------------------------+-----------
↪---------------+
|WkVVu9fD    |WkVVu9fD-1568731976982  |onap-dublin   |aai               |subscription-
↪delete           |onap-dublin   |completed   |2019-09-17T14:52:56.000   |2019-09-
↪17T14:52:57.000   |
+------------+------------------------+--------------+------------------+-------------
↪-----------------+--------------+------------+--------------------------+-----------
↪---------------+
|WkVVu9fD    |WkVVu9fD-1568731785780  |onap-dublin   |aai               |vnfm-create ␣
↪                |onap-dublin   |completed   |2019-09-17T14:49:45.000   |2019-09-
↪17T14:49:46.000   |
......
```

While executing the test cases, VTP provides unique execution-id (2nd column) for each step. As you note in the example above, some steps are in-progress, while others are completed already. If there is error then status will be set to failed.

To find out the foot-print of each step, following commands are available

```
oclip execution-show-out --execution-id WkVVu9fD-1568731785780       - Reports the␣
↪standard output logs
oclip execution-show-err --execution-id WkVVu9fD-1568731785780        - Reports the␣
↪standard error logs
oclip execution-show-debug --execution-id WkVVu9fD-1568731785780  - Reports the debug␣
↪details like HTTP request and responseoclip execution-show --execution-id WkVVu9fD-
↪1568731785780             - Reports the complete foot-print of inputs, outputs of␣
↪steps
```

Track the progress of the vnf-tosca-provision test cases until its completed. Then the out of the validation test cases could be retrieved as below:

```
oclip execution-show --execution-id WkVVu9fD-1568731678753              - use vnf␣
↪tosca test case execution id here
```

It will provides the output format as below:

```
{
"output": {
    "ns-id": null,
    "vnf-id": "",
    "vnfm-driver": "hwvnfmdriver",
    "vnf-vendor-name": "huawei",
    "onap-objects": {
    "ns_instance_id": null,
```

(continues on next page)

(continued from previous page)

```
    "tenant_version": null,
    "service_type_id": null,
    "tenant_id": null,
    "subscription_version": null,
    "esr_vnfm_id": null,
    "location_id": null,
    "ns_version": null,
    "vnf_status": "active",
    "entitlement_id": null,
    "ns_id": null,
    "cloud_version": null,
    "cloud_id": null,
    "vlm_version": null,
    "esr_vnfm_version": null,
    "vlm_id": null,
    "vsp_id": null,
    "vf_id": null,
    "ns_instance_status": "active",
    "service_type_version": null,
    "ns_uuid": null,
    "location_version": null,
    "feature_group_id": null,
    "vf_version": null,
    "vsp_version": null,
    "agreement_id": null,
    "vf_uuid": null,
    "ns_vf_resource_id": null,
    "vsp_version_id": null,
    "customer_version": null,
    "vf_inputs": null,
    "customer_id": null,
    "key_group_id": null,
    },
    "vnf-status": "active",
    "vnf-name": "vgw",
    "ns-status": "active"
},
"input": {
    "mode": "validate",
    "vsp": "/tmp/data/vtp-tmp-files/1568731645518.csar",
    "vnfm-driver": "hwvnfmdriver",
    "config-json": "/opt/oclip/conf/vnf-tosca-provision.json",
    "vnf-vendor-name": "huawei",
    "ns-csar": "/tmp/data/vtp-tmp-files/1568731660745.csar",
    "onap-objects": "{}",
    "timeout": "600000",
    "vnf-name": "vgw",
    "vnf-csar": "/tmp/data/vtp-tmp-files/1568731655310.csar"
},
"product": "onap-dublin",
"start-time": "2019-09-17T14:47:58.000",
"service": "vnf-validation",
"end-time": "2019-09-17T14:53:46.000",
"request-id": "WkVVu9fD-1568731678753",
"command": "vnf-tosca-provision",
"status": "completed"
}
```

Reporting Results

VTP provides translation tool to migrate the VTP result into OVP portal format and generates the tar file for the given test case execution. Please refer https://github.com/onap/vnfsdk-refrepo/tree/master/vnfmarket-be/deployment/vtp2ovp for more details.

Once tar is generated, it can be used to submit into OVP portal https://vnf-verified.lfnetworking.org/

CHAPTER

TEN

OVP TEST CASE REQUIREMENTS

10.1 OVP Test Suite Purpose and Goals

The OVP test suite is intended to provide a method for validating the interfaces and behaviors of an NFVI platform according to the expected capabilities exposed in OPNFV. The behavioral foundation evaluated in these tests should serve to provide a functional baseline for VNF deployment and portability across NFVI instances. All OVP tests are available in open source and are executed in open source test frameworks.

10.2 Test Case Requirements

The following requirements are mandatory for a test to be submitted for consideration in the OVP test suite:

- All test cases must be fully documented, in a common format. Please consider the existing *OVP Test Specifications* as examples.
  - Clearly identifying the test procedure and expected results / metrics to determine a "pass" or "fail" result.
- Tests must be validated for the purpose of OVP, tests should be run with both an expected positive and negative outcome.
- At the current stage of OVP, only functional tests are eligible, performance testing is out of scope.
  - Performance test output could be built in as "for information only", but must not carry pass/fail metrics.
- Test cases should favor implementation of a published standard interface for validation.
  - Where no standard is available provide API support references.
  - If a standard exists and is not followed, an exemption is required. Such exemptions can be raised in the project meetings first, and if no consensus can be reached, escalated to the TSC.
- Test cases must pass on applicable OPNFV reference deployments and release versions.
  - Tests must not require a specific NFVI platform composition or installation tool.
    - Tests and test tools must run independently of the method of platform installation and architecture.
    - Tests and test tools must run independently of specific OPNFV components allowing different components such as storage backends or SDN controllers.
  - Tests must not require un-merged patches to the relevant upstream projects.
  - Tests must not require features or code which are out of scope for the latest release of the OPNFV project.
  - Tests must have a documented history of recent successful verification in OPNFV testing programs including CI, Functest, Yardstick, Bottlenecks, Dovetail, etc. (i.e., all testing programs in OPNFV that regularly validate tests against the release, whether automated or manual).

  - Tests must be considered optional unless they have a documented history for ALL OPNFV scenarios that are both
    - applicable, i.e., support the feature that the test exercises, and
    - released, i.e., in the OPNFV release supported by the OVP test suite version.
- Tests must run against a fully deployed and operational system under test.
- Tests and test implementations must support stand alone OPNFV and commercial OPNFV-derived solutions.
  - There can be no dependency on OPNFV resources or infrastructure.
  - Tests must not require external resources while a test is running, e.g., connectivity to the Internet. All resources required to run a test, e.g., VM and container images, are downloaded and installed as part of the system preparation and test tool installation.
- The following things must be documented for the test case:
  - Use case specification
  - Test preconditions
  - Basic test flow execution description and test assertions
  - Pass/Fail criteria
- The following things may be documented for the test case:
  - Parameter border test cases descriptions
  - Fault/Error test case descriptions
  - Post conditions where the system state may be left changed after completion

New test case proposals should complete a OVP test case worksheet to ensure that all of these considerations are met before the test case is approved for inclusion in the OVP test suite.

10.3 Dovetail Test Suite Naming Convention

Test case naming and structuring must comply with the following conventions. The fully qualified name of a test case must comprise three sections:

<test_project>.<test_area>.<test_case_name>

- **test_project**: The fully qualified test case name must identify the test project which developed and maintains the test case.
- **test_area**: The fully qualified test case name must identify the test case area. The test case area is a single word identifier describing the broader functional scope of a test case, such as ha (high-availability), tempest, vnf, etc.
- **test_case_name**: The fully qualified test case name must include a concise description of the purpose of the test case.

An example of a fully qualified test case name is *functest.tempest.compute*.

CHAPTER

ELEVEN

DOVETAIL AS A GENERIC TEST FRAMEWORK

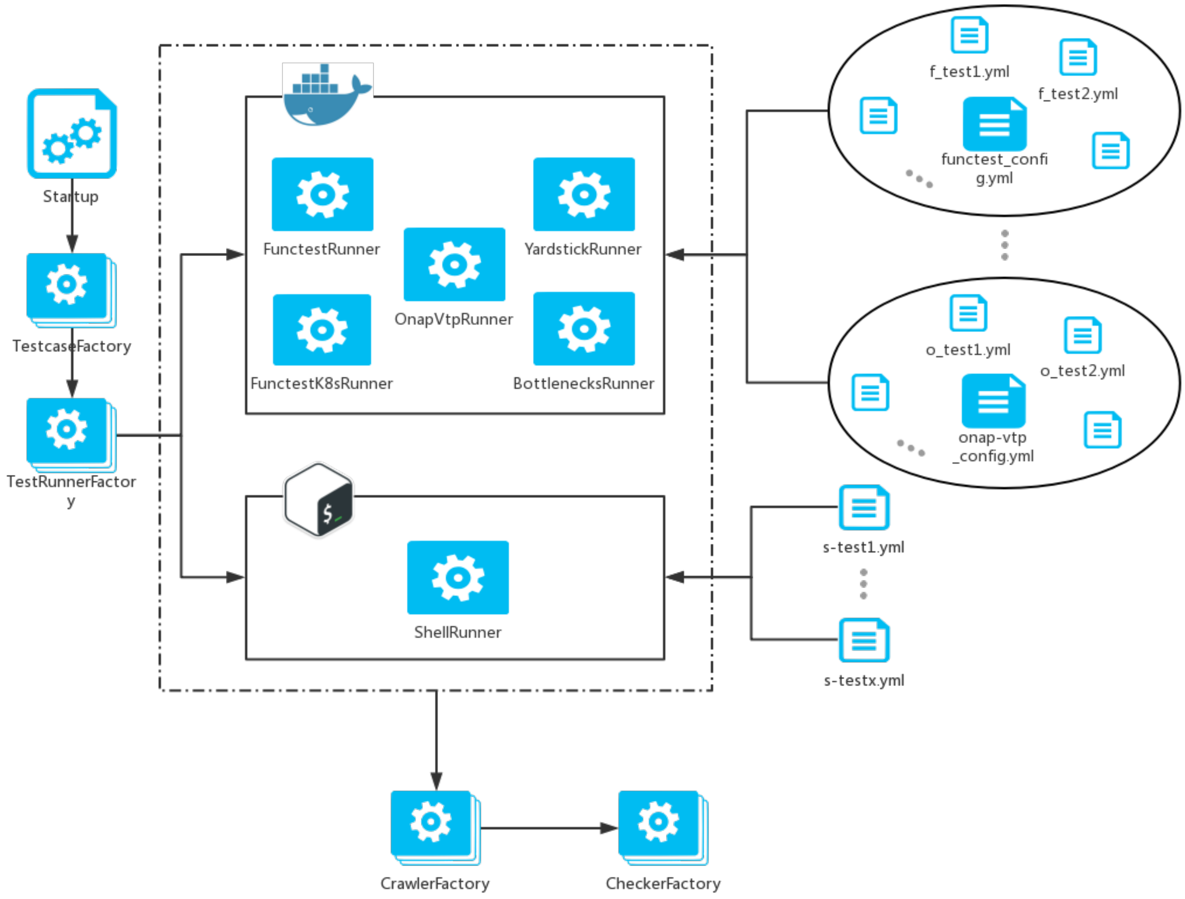
11.1 Overview

Dovetail is responsible for the technical realization of the OPNFV Verification Program (OVP) and other compliance verification projects within the scope of the Linux Foundation Networking (LFN) umbrella projects. Dovetail provides a generic framework for executing a specific set of test cases which define the scope of a given compliance verification program, such as OVP.

This document aims at introducing what Dovetail generic framework looks like and how to develop within this framework.

11.2 Introduction of Dovetail Framework

The following diagram illustrates Dovetail generic framework.



In this diagram, there are 5 main parts, *TestcaseFactory*, *TestRunnerFactory*, *CrawlerFactory*, *CheckerFactory* and *test case groups*.

- **TestcaseFactory**: For each project, there needs to create its own testcase class such as *FunctestTestcase* and *OnapVtpTestcase*. All these classes are based on class *Testcase*. There are already many functions in this base class which are mainly used to parse test case configuration files. If no other special requirements exist, it only needs to initialize these classes with different types. Otherwise, it needs to overload or add some functions.
- **TestRunnerFactory**: Similar to *TestcaseFactory*, each project has its own test runner class. Dovetail supports 2 kinds of test runners, *DockerRunner* and *ShellRunner*. For projects based on Docker, it needs to create their own test runner classes such as *FunctestRunner* which inherit from class *DockerRunner*. For other projects that are based on Shell, it can use *ShellRunner* directly. Test case runners provide many functions to support test cases runs such as preparing test tool of each project, run all the commands defined by each test case and clean the environment.
- **Test case groups**: Each group is composed of one project configuration file and a set of test cases belonging to this project. These groups are used as the input of test runners to provide information of projects and test cases. For *ShellRunner*, it only needs test case configuratons as the input.
- **CrawlerFactory**: This is used to parse the results of test cases and record them with unified format. The original result data report by each project is different. So it needs to create different crawler classes for different projects to parse their results.
- **CheckerFactory**: This is used to check the result data generated by crawler. Each project should have its own checker class due to the different requirements of different projects.

11.3 Development with Dovetail Framework

Everyone who is interested in developing Dovetail framework to integrate new upstream test cases will face one of the two following scenarios:

- **Adding test cases that belong to integrated projects**: There are already some projects integrated in Dovetail. These projects are coming from OPNFV (Open Platform for NFV) and ONAP (Open Network Automation Platform) communities. It will be much easier to add new test cases that belong to these projects.
- **Adding test cases that not belong to integrated projects**: The test cases may belong to other projects that haven't been integrated into Dovetail yet. These projects could be in OPNFV, ONAP or other communities. This scenario is a little more complicated.

11.3.1 Test cases belonging to integrated projects

Dovetail framework already includes a large amount of test cases. All these test cases are implemented by upstream projects in OPNFV and ONAP. The upstream projects already integrated in Dovetail are Functest, Yardstick and Bottlenecks from OPNFV and VNF SDK and VVP from ONAP.

In order to add a test case belonging to one of these projects, there only need to add one test case configuration file which is in yaml format. Following is the introduction about how to use the file to add one new test case. Please refer to Dovetail test case github for all configuration files of all test cases.

```
---
Test case name in Dovetail:
  name: Test case name in Dovetail
  objective: Test case description
  validate:
    type: 'shell' or name of the project already integrated in Dovetail
    testcase: The original test case name called in the project that it is developed
    image_name: Name of the Docker image used to run this test
    pre_condition:
      - 'Commands needed to be executed before running this test'
      - 'e.g. cp src_file dest_file'
    cmds:
      - 'Commands used to run this test case'
    post_condition:
      - 'Commands needed to be executed after running this test case'
  report:
    source_archive_files:
      - test.log
    dest_archive_files:
      - path/to/archive/test.log
    check_results_files:
      - results.json
    portal_key_file: path/to/key/logs/xxx.log
    sub_testcase_list:
      - sub_test_1
      - sub_test_2
      - sub_test_3
```

This is the complete format of test case configuration file. Here are some detailed description for each of the configuration options.

- **Test case name in Dovetail**: All test cases should be named as 'xxx.yyy.zzz'. This is the alias in Dovetail and has no relationship with its name in its own project. The first part is used to identify the project where this test case come from (e.g. functest, onap-vtp). The second part is used to classify this test case according to test area (e.g. healthcheck, ha). Dovetail supports to run whole test cases in one test suite with the same test area. Also the area is used to group all test cases and generate the summary report at the end of the test. The last part is special for this test case itself (e.g. image, haproxy, csar). It's better to keep the file name the same as the test case name to make it easier to find the config file according to this test case alias in Dovetail.

- **validate**: This is the main section to define how to run this test case.
  - **type**: This is the type of this test case. It can be *shell* which means running this test case with Linux bash commands within Dovetail container. Also it can be one of the projects already integrated in Dovetail (functest, yardstick, bottlenecks, onap-vtp and onap-vvp). Then this type is used to map to its project configuration yaml file. For example, in order to add a test case in OPNFV project Functest to Dovetail framework, the type here should be *functest*, and will map to *functest_config.yml* for more configurations in project level. Please refer to Dovetail project config github for more details.
  - **testcase**: This is the name defined in its own project. One test case can be uniquely identified by *type* and *testcase*. Take the test case *functest.vping.ssh* as an example. Its *type* is 'functest' and *testcase* is 'vping_ssh'. With these 2 properties, it can be uniquely identified. End users only need to know that there is a test case named *functest.vping.ssh* in OVP compliance test scope. Dovetail Framework will run *vping_ssh* within Functest Docker container.
  - **image_name**: [optional] If the type is *shell*, there is no need to give this. For other types, there are default docker images defined in their project configuration files. If this test case uses a different docker image, it needs to overwrite it by adding *image_name* here. The *image_name* here should only be the docker image name without tag. The tag is defined in project's configuration file for all test cases belonging to this project.
  - **pre_condition**: [optional] A list of all preparations needed by this test case. If the list is the same as the default one in its project configuration file, then there is no need to repeat it here. Otherwise, it's necessary to overwrite it. If its type is *shell*, then all commands in *pre_condition*, *cmds* and *post_condition* should be executable within Dovetail Ubuntu 14.04 Docker container. If its type is one of the Docker runner projects, then all commands should be executable within their own containers. For Functest, it's alpine 3.8. For Yardstick and Bottlenecks it's Ubuntu 16.04. For VNF SDK it's Ubuntu 14.04. Also all these commands should not require network connection because some commercial platforms may be offline environments in private labs.
  - **cmds**: [optional] A list of all commands used to run this test case.
  - **post_condition**: [optional] A list of all commands needed after executing this test case such as some clean up operations.
- **report**: This is the section for this test case to archive some log files and provide the result file for reporting PASS or FAIL.
  - **source_archive_files**: [optional] If there is no need to archive any files, this section can be removed. Otherwise, this is a list of all source files needed to be archived. All files generated by all integrated projects will be put under *$DOVETAIL_HOME/results*. In order to classify and avoid overwriting them, it needs to rename some important files or move them to new directories. Navigating directory *$DOVETAIL_HOME/results* to find out all files needed to be archived. The paths here should be relative ones according to *$DOVETAIL_HOME/results*.
  - **dest_archive_files**: [optional] This should be a list corresponding to the list of *source_archive_files*. Also all paths here should be relative ones according to *$DOVETAIL_HOME/results*.
  - **check_results_files**: This should be a list of relative paths of the result files generated by this test case. Dovetail will parse these files to get the result (PASS or FAIL).
  - **portal_key_file**: This should be the key log file of this test case which will be used by the OVP portal for review.

- **sub_testcase_list**: [optional] This section is almost only for Tempest tests in Functest. Take *functest.tempest.osinterop* as an example. The *sub_testcase_list* here is an check list for this kind of tempest tests. Only when all sub test cases list here are passed, this test case can be taken as PASS. The other kind of tempest tests is *tempest_custom* such as *functest.tempest.image*. Besides taking the *sub_testcase_list* as the check list, it's also used to generate an input file of Functest to define the list of sub test cases to be tested.

11.3.2 Test cases not belonging to integrated projects

If test cases waiting to be added into Dovetail do not belong to any project that is already integrated into Dovetail framework, then besides adding the test case configuration files introduced before, there are some other files needed to be added or modified.

Step 1: Add a project configuration file

For a new test case that belongs to a new project, it needs to create a project configuration file to define this new project in Dovetail first. Now Dovetail only supports integration of projects by using their Docker images. If this test case should be run with shell runner, then can only add test case configuration files with *type* 'shell' as describing before and skip the following steps. Following is the introduction of how to use project configuration file to add one new project into Dovetail. Please refer to Dovetail projects configuration github for all configuration files of all integrated projects.

```
---

{% set validate_testcase = validate_testcase or '' %}
{% set testcase = testcase or '' %}
{% set dovetail_home = dovetail_home or '' %}
{% set debug = debug or 'false' %}
{% set build_tag = build_tag or '' %}
{% set userconfig_dir = '/tmp/userconfig' %}
{% set patches_dir = '/tmp/patches' %}
{% set result_dir = '/tmp/results' %}
{% set openrc_file = '/home/conf/env_file' %}

project name:
  image_name: name of the docker image
  docker_tag: tag of the docker image
  opts:
    detach: true
    stdin_open: true
    privileged: true
  shell: '/bin/bash'
  envs:
    - 'CI_DEBUG={{debug}}'
    - 'DEPLOY_SCENARIO={{deploy_scenario}}'
    - 'ENV_NAME=env_value'
  volumes:
    - '{{dovetail_home}}/userconfig:{{userconfig_dir}}'
    - '{{dovetail_home}}/results:{{result_dir}}'
    - '/path/on/host:/path/in/container'
    - '/path/of/host/file:/file/path/in/container'
  mounts:
    - 'source={{dovetail_home}}/pre_config/env_config.sh,target={{openrc_file}}
    - 'source={{dovetail_home}}/pre_config,target=/home/opnfv/pre_config'
    - 'source=/file/or/derectory/on/host,target=/file/or/derectory/in/container'
```

(continues on next page)

(continued from previous page)

```
patches_dir: {{patches_dir}}
pre_condition:
  - 'Commands needed to be executed before running this test'
cmds:
  - 'Commands used to run this test case'
post_condition:
  - 'Commands needed to be executed after running this test case'
openrc: absolute path of openstack credential files
extra_container:
  - container1_name
  - container2_name
```

This is the complete format of project configuration file. Here are some detailed description for each of the configuration options.

- **Jinja Template**: At the begining of this yaml file, it uses Jinja template to define some parameters that will be used somewhere in this file (e.g. result_dir and openrc_file). Besides those, there are some other parameters providing by Dovetail framework as input of this file, and other parameters can be defined by using these ones (e.g. testcase and dovetail_home). The whole input parameters which can be used are list below.
  - **attack_host**: This is the attack host name of the test case which calls this project configuration file. It's only for HA test cases and can be given in HA configuration file *pod.yaml*.
  - **attack_process**: This is the attack process name of the test case which calls this project configuration file. It's only for HA test cases and can be given in HA configuration file *pod.yaml*.
  - **build_tag**: This is a string includes the UUID generated by Dovetail.
  - **cacert**: This is also only for OpenStack test cases. It is the absolute path of the OpenStack certificate provided in *env_config.sh* file.
  - **deploy_scenario**: This is the input when running Dovetail with option *–deploy-scenario*.
  - **debug**: This is *True* or *False* according to the command running test cases with or without option *–debug*.
  - **dovetail_home**: This is the *DOVETAIL_HOME* getting from the ENV.
  - **os_insecure**: This is only for test cases aiming at OpenStack. This is *True* or *False* according to *env_config.sh* file.
  - **testcase**: This is the name of the test case which calls this project configuration file. Different from *validate_testcase*, this is the alias defined in Dovetail not in its own project.
  - **validate_testcase**: This is the name of the test case instance which calls this project configuration file. The name is provided by the configuration file of this test case (validate -> testcase).
- **project name**: This is the project name defined in Dovetail. For example OPNFV Functest project is named as 'functest' here in Dovetail. This project name will be used by test case configuration files as well as somewhere in Dovetail source code.
- **image_name**: This is the name of the default Docker image for most test cases within this project. Each test case can overwrite it with its own configuration.
- **docker_tag**: This is the tag of all Docker images for all test cases within this project. For each release, it should use one Docker image with a stable and official release version.
- **opts**: Here are all options used to run Docker containers except 'image', 'command', 'environment', 'volumes', 'mounts' and 'extra_hosts'. For example, the options include 'detach', 'privileged' and 'tty'. The full list of all options can be found in Docker python SDK docs.
- **shell**: This is the command used to run in the container.

- **envs**: This is a list of all envs used to run Docker containers.
- **volumes**: A volume mapping list used to run Docker containers. The source volumes list here are allowed to be nonexistent and Docker will create new directories for them on the host. Every project should at least map the *$DOVETAIL_HOME/results* in the test host to containers to collect all result files.
- **mounts**: A mount mapping list used to run Docker containers. More powerful alternative to **volumes**. The source volumes list here are not allowed to be nonexistent. Every project should at least mount the *$DOVETAIL_HOME/pre_config* in the test host to containers to get config files.
- **patches_dir**: [optional] This is an absolute path of the patches applied to the containers.
- **pre_condition**: A list of all default preparations needed by this project. It can be overwritten by configurations of test cases.
- **cmds**: A list of all default commands used to run all test cases within this project. Also it can be overwritten by configurations of test cases.
- **post_condition**: A list of all default cleaning commands needed by this project.
- **openrc**: [optional] If the system under test is OpenStack, then it needs to provide the absolute path here to copy the credential file in the Test Host to containers.
- **extra_container**: [optional] The extra containers needed to be removed at the end of the test. These containers are created by the test cases themselves at runtime rather than created by Dovetail.

Step 2: Add related classes

After adding the project and test case configuration files, there also need to add some related classes into the source code.

- **Test Case class**: Each project should have its own test case class in *testcase.py* for *TestcaseFactory*.
- **Test Runner class**: Each project should have its own test runner class in *test_runner.py* for *TestRunnerFactory*.
- **Crawler class**: Each project should have its own test results crawler class in *report.py* for *CrawlerFactory*.
- **Checker class**: Each project should have its own test results checker class in *report.py* for *CheckerFactory*.

Step 3: Create related logs

If the classes added in step2 have function *create_log*, then need to call these functions in *run.py* to initial the log instances at the very begining.

Step 4: Update unit tests

A patch is not going to be verified without 100% coverage when applying acceptance check.

CHAPTER

TWELVE

OPNFV VERIFICATION PROGRAM (OVP) 2019.12 / DOVETAIL 3.0.0 RELEASE NOTE

12.1 OPNFV 2019.12 Release

The OPNFV Verification Program (OVP) allows vendors and operators to obtain 'OPNFV Verified' status based on an agreed upon set of compliance verification test cases that align to OPNFV releases. The reference System under Test (SUT) is either the NFV components deployed by the OPNFV installers for a given release, where OVP 2019.12 is based on the OPNFV Hunter release, or a VNF being on-boarded and orchestrated by the ONAP El Alto release. Participants of the program can verify commercial or open source offerings against an OVP release. This implies that the SUT used for verification has interfaces, components, functions and behaviors that align to OPNFV installer integrations and ONAP deployments.

Dovetail is the overall framework used to execute tests and collect results for the OVP Infrastructure badge. Dovetail does not deliver test content directly. Rather, test content is developed in other OPNFV test frameworks such as Functest and upstream test communities such as OpenStack's RefStack/Tempest projects. Dovetail leverages this upstream test content and provides a common set of test platform services for the OVP.

Approved test tools (OPNFV Dovetail, ONAP VTP, and ONAP VVP) work in conjunction with a web portal interface dubbed the 'OVP web portal' to allow users to upload test results to a centralized community repository. This facilitates user collaboration, result sharing, self-testing and community reviews. It also serves as a hub for new participants to learn about the program and access key resources. The link for this portal is at: OPNFV Verification Program.

Use of the OVP web portal is open to all and only requires a valid Linux Foundation ID to login. Users are welcome to use the portal to upload, inspect and share results in a private manner. In order to submit results for official review, the first step is apply for acceptance into the program with the participation form provided in the link: OPNFV Verification Program Participation Form

12.1.1 NFVI Test Suites and Test Areas

OVP/Dovetail groups test cases into test suites and test areas. Test suites are currently a basic categorization around releases for the most part. Executing the test suite 'ovp.2019.12' without further specification will run all the test cases in the OVP 2019.12 release. Test suites are divided into test areas that can be executed separately.

Test cases include a division into **'mandatory'** and **'optional'** in an overarching categorization.

All the mandatory test cases are required to be executed with passing results for all inclusive test cases for results to be reviewed and approved by the community made up of peer reviewers. The optional test cases are not required to be executed for the official compliance verification review in the OVP 2019.12 release. However, execution of these cases is encouraged, as some optional test cases may become mandatory in future releases.

12.1.2 NFVI Test Cases and Sub Test Cases

Each test area consists of multiple test cases where each test case can be a single test or broken down into sub test cases. A listing of test cases with the number of sub test cases noted in parenthesis is shown below for the OVP 2019.12 release.

Mandatory

- bottlenecks.stress.ping (1)
- functest.vping.userdata (1)
- functest.vping.ssh (1)
- functest.tempest.osinterop (219)
- functest.tempest.compute (12)
- functest.tempest.identity_v3 (11)
- functest.tempest.image (2)
- functest.tempest.network_api (14)
- functest.tempest.volume (2)
- functest.tempest.neutron_trunk_ports (38)
- functest.tempest.ipv6_api (21)
- functest.security.patrole (124)
- yardstick.ha.nova_api (1)
- yardstick.ha.neutron_server (1)
- yardstick.ha.keystone (1)
- yardstick.ha.glance_api (1)
- yardstick.ha.cinder_api (1)
- yardstick.ha.cpu_load (1)
- yardstick.ha.disk_load (1)
- yardstick.ha.haproxy (1)
- yardstick.ha.rabbitmq (1)
- yardstick.ha.database (1)

There are a total of 456 mandatory test cases.

Optional

- functest.vnf.vims (1)
- functest.vnf.vepc (1)
- functest.tempest.ipv6_scenario (8)
- functest.tempest.multi_node_scheduling (6)
- functest.tempest.network_security (6)
- functest.tempest.vm_lifecycle (12)
- functest.tempest.network_scenario (5)

- functest.tempest.bgpvpn (21)
- functest.security.patrole_vxlan_dependent (2)
- yardstick.ha.neutron_l3_agent (1)
- yardstick.ha.controller_restart (1)

There are a total of 64 optional test cases.

12.1.3 OPNFV Test Projects and Components

The OPNFV test frameworks integrated into the Dovetail framework that deliver test content are:

- Functest (leverages OpenStack RefStack/Tempest projects in addition to supplying native test cases)
- Yardstick
- Bottlenecks

12.1.4 ONAP Test Projects and Components

The ONAP test projects and components used with this OVP release to provide the test requirements and test scripting are:

- VNFRQTS
- VNFSDK
- VVP

12.1.5 Acceptence and Marketing

Upon successful community review of results for OVP 2019.12, the Linux Foundation Compliance Verification Committee (LFN CVC) on behalf of the Board of Directors can award a product 'OPNFV Verified' status. Use of 'OPNFV Verified' Program Marks shall be awarded to the platform used for compliance verification. The category label of 'Infrastructure' is used within the Program Marks logo and limits the scope of this OVP release to a SUT consisting of NFVI and VIM components using ETSI terminology. It does not provide compliance verification for specific VNFs in any fashion. The date '2019.12' corresponds to a reference SUT that aligns to the OPNFV Hunter release and currently aligns to the Dovetail framework version 3.0.0.

Organizations shall not use the Program Marks in any way that would associate it with any individual or company logo or brand, beyond the association to the specific platform to which it was awarded. While OpenStack RefStack interoperability and Tempest integration test cases are executed as part of the OVP 2019.12 compliance verification test suites, the OVP does not grant or award OpenStack Marks in any fashion. 'OPNFV Verified' status does not assert readiness for commercial deployment.

Please refer to the program governance guidelines and term & conditions documents for additional details using the respective links:

- OVP Governance Guidelines
- OVP Terms and Conditions

12.2 Release Data

Project	Dovetail
Repo tag	ovp-3.0.1
Release designation	OPNFV Verification Program (OVP) 2019.12 (Hunter)
Release date	December 2019
Purpose of the delivery	Support OVP 2019.12 release with OPNFV Hunter release as reference SUT
Notes	Point release ovp-3.0.1 updates, changes, and corrects the documentation only.

12.3 Deliverables

12.3.1 Software

OPNFV Software

Docker Container	Docker Image	Tag
dovetail	opnfv/dovetail	ovp-3.0.0
functest	opnfv/functest-smoke	hunter
functest	opnfv/functest-healthcheck	hunter
functest	opnfv/functest-vnf	hunter
yardstick	opnfv/yardstick	opnfv-8.0.0
bottlenecks	opnfv/bottlenecks	8.0.1-latest

Docker images:

- Dovetail Docker images
- Functest-smoke Docker images
- Functest-healthcheck Docker images
- Functest-vnf Docker images
- Yardstick Docker images
- Bottlenecks Docker images

ONAP Software

Item	Repo Link
VTP/VNFSDK Test Scripts	https://gerrit.onap.org/r/admin/repos/vnfsdk/refrepo
VVP Robot Test Scripts	https://gerrit.onap.org/r/admin/repos/oom

12.3.2 Documents

- System Preparation Guide
- NFVI User Guide
- VNF User Guide:

- OVP NFVI Test Specifications
- ONAP VNF Test Specifications
- Dovetail CLI Reference
- Dovetail RESTful API
- OVP Workflow
- OVP Reviewer Guide

12.4 Testing with OPNFV Hunter Installers

OVP 2019.12 and Dovetail 3.0.0 are known to be have been tested with the following OPNFV Hunter installer versions.

Installer	Version
Fuel	stable/hunter

12.5 Hunter Known Restrictions/Issues

Please refer to the OPNFV and ONAP JIRA for known issues with each applicable project:

- https://jira.opnfv.org/projects/DOVETAIL
- https://jira.onap.org/projects/VVP
- https://jira.onap.org/projects/VNFSDK

12.6 Useful Links

- OVP Web Portal
- Wiki Project Page
- Dovetail Repo
- Dovetail CI dashboard
- JIRA dashboard
- Dovetail IRC Channel: #opnfv-dovetail
- Dovetail Test Configuration