
Anuket Reference Conformance for OpenStack (RC1)

Anuket

Sep 06, 2022

Contents

| | | |
|---|--|----|
| 1 | Introduction | 1 |
| 2 | NFVI Conformance Requirements | 7 |
| 3 | Cloud Infrastructure Test Cases and Traceability to Requirements | 23 |
| 4 | OpenStack-based cloud infrastructure Testing Cookbook | 41 |
| 5 | VNF Testing Framework Requirements | 45 |
| 6 | VNF Test Cases and Traceability to Requirements | 59 |
| 7 | VNF Testing Cookbook | 59 |
| 8 | Gap analysis and Development | 59 |

1 Introduction

1.1 Synopsis

Ensure an implementation of the Anuket Reference Architecture (RA), such as the Reference Implementation (RI), meets industry driven quality assurance standards for conformance, verification and validation. The OPNFV Verified Program (OVP), by Linux Foundation Networking (LFN), overseen by the Compliance Verification Committee (CVC), will provide tracking and governance for RC.

For the purpose of this chapter, NFVI+VNF testing will be performed to evaluate **Conformance** (i.e. adherence) to, and demonstrated proficiency with, all aspects of software delivery. More specifically, Conformance includes:

- Verified implementations of NFVI+VNF match expected design requirements
- Clearly stated guidelines for test, badging, and lifecycle management
- Inclusion of Operational run-books for 3rd party supplier instantiation and validations of NFVI+VNF
- Evidence, through test results, confirming delivered code matches industrial requirements

- Interoperability testing with Reference VNFs, ensuring integration stability and life-cycle management of the Reference VNF on the target implementation.

In summary, NFVI+VNF **Conformance** testing will be performed for **Verification** and **Validation** purposes, defined further as:

- **Verification** will be used to indicate conformance to design requirement specifications. Accomplished with Requirement Traceability and Manifest Reviews to ensure the NFVI is delivered per implementation specifications.
- **Validations** is used to indicate that testing performed confirms the NFVI+VNF meets the expected, or desired outcome, or behaviour.

All Terms utilized throughout this chapter are intended to align with CVC definitions, and their use through CVC documentation, guidelines, and standards.

1.2 Overview

Chapter Purpose

This chapter includes process flow, logistics, and requirements which must be satisfied to ensure Network Function Virtualisation Infrastructure (NFVI) meets the design, feature, and capability expectations of RM and RA. Upstream projects will define features/capabilities, test scenarios, and test cases which will be used to augment OVP test harnesses for infrastructure verification purposes. Existing processes, communication mediums, and related technologies will be utilized where feasible. Ultimately, test results of certified NFVI+VNF will reduce the amount of time and cost it takes each operator to on-board and maintain vendor provided VNFs.

Objective

Perform NFVI+VNF Verification and Validations using Anuket reference architecture, leveraging the existing Anuket and CVC Intake and Validation Process to onboard and validate new test projects for NFVI compliance. Upstream projects will define features/capabilities, test scenarios, and test cases to augment existing OVP test harnesses to be executed via the OVP Ecosystem.

Test Methodology

- Verification test to make sure if the OpenStack services have been deployed and configured correctly
- Manifest Verifications (Termed Compliance by CVC) will ensure the NFVI is compliant, and delivered for testing, with hardware and software profile specifications defined by the RM and RA.
- Empirical Validation with Reference Golden VNFs (Termed Validation by CVC) will ensure the NFVI runs with a set of VNF Families, or Classes, to mimic production-like VNF connectivity, for the purposes of interoperability checks.
- Candidate VNF Validation (Termed Validation & Performance by CVC) will ensure complete interoperability of VNF behaviour on the NFVI leverage VVP/VNFSKD test suites. Testing ensures VNF can be spun up, modified, or removed, on the target NFVI (aka Interoperability).

Different Distributions

The three step methodology described above of verifying Manifest compliance, executing Empirical Golden VNF transactions, and performing Interoperability Testing is the same validation process regardless of the Distribution used to establish a cloud topology, and the components and services used in the client software stack.

Terminology

Terminology in this document will follow [Glossary](#).

1.3 Scope

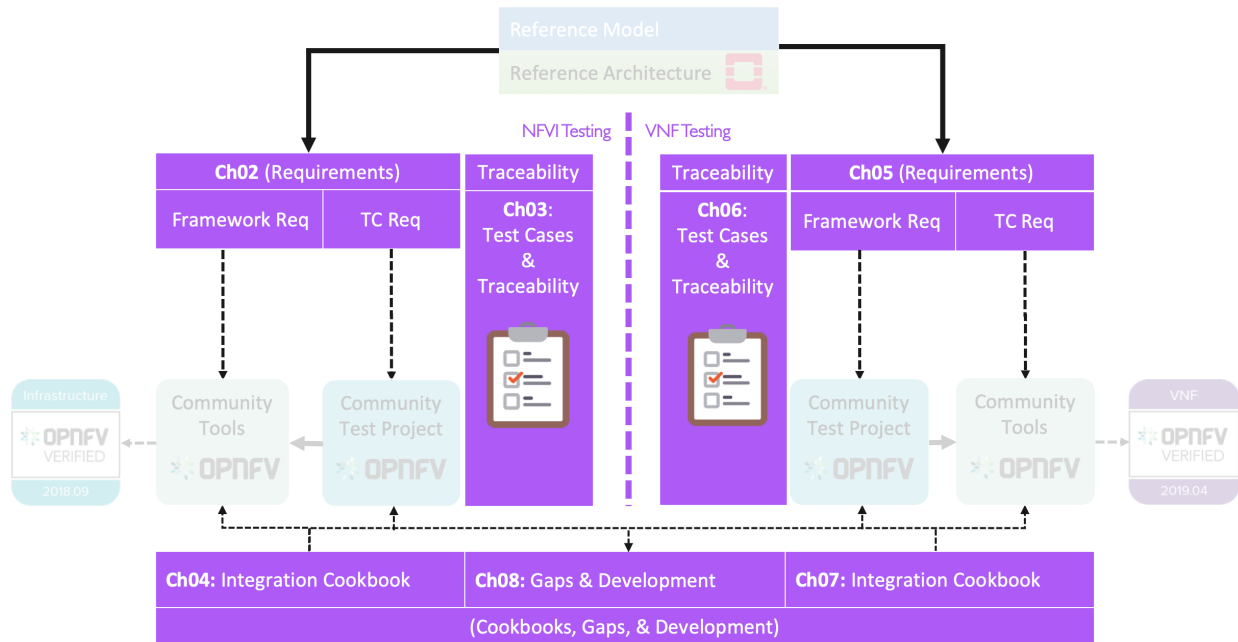


Figure 1.1: RC1 Scope

This document covers the realisation aspects of conformance of both NFVI and VNFs. The document will cover the following topics:

- Identify in details the Requirements for conformance Framework.
- Identify in details the Requirement of Test Cases (and mapping them to requirements from The Reference Model and The OpenStack Based Reference Architecture).
- Analysis of existing community projects.
- Propose an E2E Framework for conformance of NFVI and VNFs.
- Playbook of instructions, user manuals, steps of how to perform verification and conformance for both NFVI and VNFs using the proposed E2E Framework.
- Gap analysis to identify where the Gaps are in the industry (tooling, test cases, process, etc).
- Identify development efforts needed to address any gaps identified.

Not in Scope

- Functional testing / validation of the application provided by the VNF is outside the scope of this work.
- ONAP is not used in the process flow for NFVI verifications, or validations.
- Upgrades to VNFs, and the respective processes of verifying upgrade procedures and validating (testing) the success and compatibility of upgrades is not in scope.

1.4 Relation to other communities



Figure 1.2: RC-1 Relations other communities

1.5 Principles and Guidelines

The objectives of the verification program are to deliver a validated implementation of reference architecture which satisfies infrastructure needs for VNF-developer teams, leveraging the OVP ecosystem as the vehicle for delivering validated NFVI.

These core principles will guide NFV verification deliverables:

Overarching Objectives and Goals

1. Deliver verified implementation of reference architecture which satisfies infrastructure needs for VNF-developer teams.
2. All accomplished with augmentation to the current OVP ecosystem.
3. Increase probability VNFs will on-board and function with minimal problems, or issues, during initial instantiation of VNF.
4. Test Harnesses will be portable, or compatible, across all RAs/Distributions which already conform to standard interfaces and services.

1.6 Best Practices

The following best practices have been adopted to ensure verification and validation procedures are repeatable with consistent quality in test results, and RI conformances:

- Standardized test methodology / flow, Test Plan, and Test Case Suites
- Integration with Anuket Upstream Projects and OVP flow (code, docs, cert criteria, etc.)
- Leverage Network and Service Models, with identified VNF-specific parameters
- Standardized conformance criteria
- Define Anuket RA as scenarios, and have all test cases for the RA be involved in OVP
- Add test cases from operators, which operators already tested in their environment

1.7 Verification methodologies

Perform VNF interoperability verifications against an implementation of Anuket reference architecture, leveraging existing Anuket Intake Process. Upstream projects will define features/capabilities, test scenarios, and test cases to augment existing OVP test harnesses to be executed via the OVP Ecosystem.

3rd Party test platforms may also be leveraged, if desired.

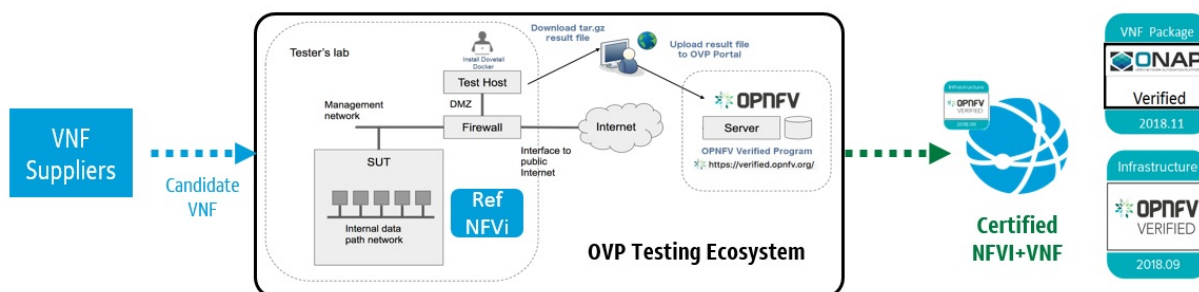


Figure 1.3: Conformance Methodology

1.8 Assumptions & Dependencies

Assumptions NFVI+VNF testing will be considered **Testable** if the follow qualifiers are present in a test execution, and subsequent result:

- Ability to perform Conformance, or Verification of Artifacts to ensure designs (RM/RA/RI) are delivered per specification
- Ability to Control (or manipulate), manifestations of RM/RA/RI for the purposes to adjust the test environment, and respective cases, scenarios, and apparatus, to support actual test validations
- Ability to monitor, measure, and report, Validations performed against a target, controlled system under test

In addition, respective Entrance criteria is a prerequisite which needs to be satisfied for NFVI+VNF to be considered **Testable**.

Dependencies NFVI+VNF verification will rely upon test harnesses, test tools, and test suites provided by Anuket projects, including dovetail, yardstick, and Bottleneck. These upstream projects will be reviewed semi-annually to verify they are still healthy and active projects. Over time, the projects representing the conformance process may change, but test parity is required if new test suites are added in place of older, stale projects.

- NFVI+VNF verifications will be performed against well defined instance types consisting of a HW and SW Profile, Configured Options, and Applied Extensions (See image.)



Figure 1.4: Instance Type

NFVI+VNF Instance Type:

- Standard compute flavours to be tested are defined in [Virtual Network Interface Specifications](#)
- Performance profiles come in the form of Basic, Network Intensive, and Compute intensive. Refer to [Analysis](#) for details on these profiles.

1.9 Results Collation & Presentation

Test suites will be categorized as functional or performance based. Results reporting will be communicated as a boolean (pass/fail). The pass/fail determination for performance-based test cases will be made by comparing results against a baseline. Example performance-based metrics include, but are not limited to: resource utilization, response times, latency, and sustained throughput per second (TPS).

Placeholder to document where results will be posted (e.g. Dovetail dashboards.)

1.10 Governance

1. Conformance badges will be presented by the CVC
2. CVC will maintain requirements for conformance

2 NFVI Conformance Requirements

2.1 Synopsis

This chapter mainly covers the overall requirement for the Reference Conformance test. The Conformance tests are conducted on the Cloud Infrastructure and VNF level. Conformance on the Cloud Infrastructure makes sure the SUT follows RM & RA requirements, and conformance on VNF makes sure that the VNF can be deployed and sufficiently work on the Cloud Infrastructure that has passed the conformance test.

All Terms utilized throughout this chapter are intended to align with CVC definitions, and their use through CVC documentation, guidelines, and standards. This chapter will outline the Requirements, Process, and Automation, needed to deliver the Cloud Infrastructure conformance.

2.2 Introduction

The Cloud Infrastructure refers to the physical and virtual resources (compute, storage and network) on which virtual network functions (VNFs) are deployed. Due to the differences in the API and under-layer configuration and capability matrix, multiple vendor VNF deployments on the shared Cloud Infrastructure becomes hard to predict, and requires a amount of cross vendor interoperability tests. With the combined effort from operators and vendors, define the RA and RM, that standardises the under layer configuration, capability and API, so as to provide the upper layer VNF with a 'common cloud infrastructure'. Based on this, Anuket also provides RC for conformance tests of SUT against RA & RM requirements. SUT passes the conformance test will be identified as NFVI that can fit into the common requirements.

In the meantime, RC also provides conformance test for VNF. The intention is to make sure VNF that passes RC test cases can be deployed on any NFVI which also passes RC without any conformance and interoperability issue.

2.3 Methodology

The Cloud Infrastructure is consumed or used by VNFs via APIs exposed by Virtualised Infrastructure Manager (VIM). The resources created by VIM on the NFVI use the underlying physical hardware (compute, storage and network) either directly or indirectly. Anuket recommends RA1 to be used as a reference architecture for the Cloud Infrastructure conformance. This would provide a set of standard interfaces to create resources on the Cloud Infrastructure. Below step by step process illustrates the NFVI conformance methodology:

- SUT (Anuket R11 or commercial NFVI) is deployed on hardware for conformance test.
- A set of tests run on SUT to determine the SUT readiness for conformance process.
- Golden KPIs are taken as a reference.
- A set of tests are run on the SUT (target for conformance).
- KPIs obtained from the SUT are collected and submitted to conformance portal.
- The SUT KPIs are reviewed and compared with Golden KPIs to determine if the conformance badge is to be provided to SUT or not.

Based on a NFVI passing RC test and getting the conformance badge, VNF conformance test can be further conducted. Such test will leverage existing Anuket Intake Process. Upstream projects will define features/capabilities, test scenarios, and test cases to augment existing OVP test harnesses to be executed via the OVP Ecosystem.

Conformance methodologies to be implemented, from a process perspective include:

- Engineering package validations will be performed against targeted infrastructure/architecture.
- Configuration settings/features/capabilities will be baseline.

- Capabilities
- Metrics
- Compute flavors
- Interface options
- Storage extensions
- Acceleration capabilities
- Profiles are offered to VNFs as an instance types with predefined compute flavors.
 - A particular set of options is an instance type
 - Compute flavors: .tiny, .small etc as defined in [Profiles](#)
- NFVI performance profiles, for which NFVI validations will support and be verified against, are defined as basic and network intensive. Details for each of these profiles can be found in [Analysis](#).

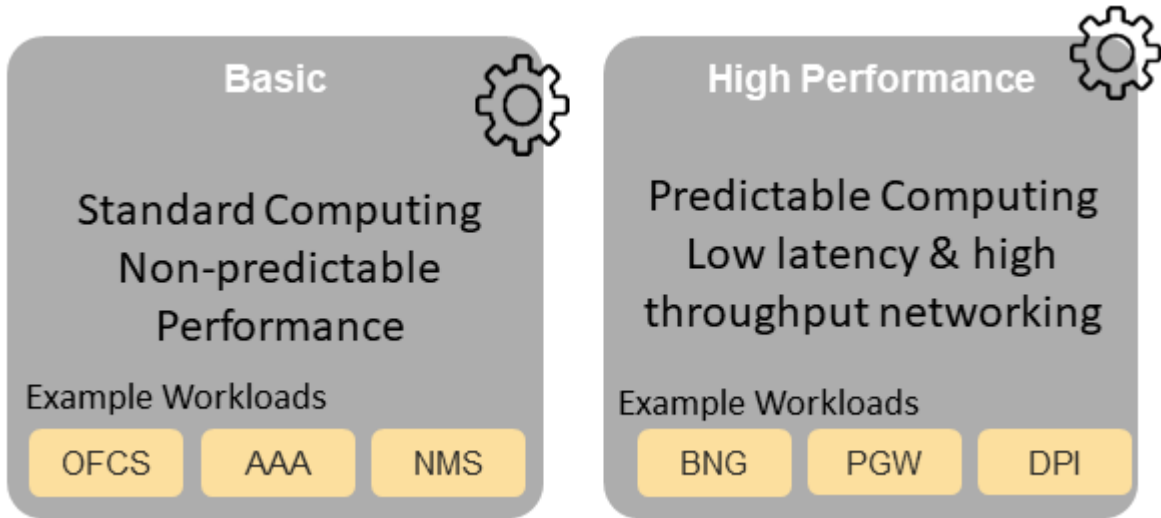


Figure 2.2: NFVI Profiles

2.5 Profiles Reference

Different vendors have different types of VNFs to serve different use-cases. A VNF like Broadband Network Gateway (BNG) would require high networking throughput whereas a VNF like Mobility Management Entity (MME) would require high computing performance. As such, BNG would require high KPI values for network throughput and MME would require high CPU performance KPIs like Index Score, Instructions Per Second (IPS) etc. The target NFVI to cater these needs would have different characteristics. Depending on VNF's requirements, the NFVI can be categorized into below profiles:

- Basic (B) profile for standard computing and
- Network intensive (N) profile offering predictable computing performance along with low latency and high networking throughput Similarly, different NFVI vendors may specialize in different hardware profiles and some may specialize in both VNFs and NFVI.

To cater to different needs from multiple NFVI vendors, Anuket allows different types of NFVI Conformance based on their types of profile [Analysis](#)

- Certify Vendor NFVI Hardware solution: This allows for Conformance of only NFVI.
- Certify Vendor NFVI Hardware and Software Solution: This allows for Conformance for NFVI running a particular VNF.

2.6 Compliance, Verification, and Conformance

The below set of steps define the compliance, verification and Conformance process for NFVI

- Based on VNF’s requirements, the Cloud Infrastructure profile is selected - B, N
- The Cloud Infrastructure readiness is checked for Conformance.
- The test VNFs are on-boarded using automation scripts on the NFVI.
- VNF on-boarding is validated by running functional tests to ensure that the on-boarding is successful.
- VNF performance tests are executed and NFVI KPIs are recorded during the tests.
- KPI comparison is run to compare NFVI KPIs with Golden KPIs, which serve as a reference for NFVI Conformance.
- If NFVI KPIs meet Golden KPIs, NFVI is certified and granted a Conformance badge.
- If NFVI KPIs do not meet Golden KPIs, no Conformance is provided.



Figure 2.3: Reference NFVI Profiles Implementation

2.7 Entry & Exit Criteria

Entry criteria: Before entering into NFVI Conformance, NFVI needs to satisfy the following requirements as entry pass:

- Design & Requirements
 - Design, Configuration, Features, SLAs, and Capability documentation complete
 - Users stories / Adherence to Anuket Model principles and guidelines
 - Chosen Reference Architecture matches the Architecture from the product catalog
- Environment

- Lab assets/resources and respective software revision levels are specified, with confirmation of compatibility across external systems
- Tenant needs identified
- All connectivity, network, image, VMs, delivered with successful pairwise tests
- Lab instrumented for proper monitoring
- Lab needs to be setup according to RA1/RA2 as defined by Anuket specifications and should be in the NFVI required state.
- Planning & Delivery
 - Kickoff / Acceptance Criteria reviews performed
 - Delivery commitments, timelines, and cadence accepted
 - Confirm backward compatibility across software/flavor revision levels
- Data/VNFs/Security
 - Images, Heat Templates, Preload Sheets available
 - Images uploaded to tenant space
 - External system test data needs identified
 - Owners (NFVI, VNF, PTL, etc) documented
 - Security Compliance Satisfied (Refer to Anuket specification Chapter XXXX Security for additional tests, scans, and vulnerabilities validations)

Exit criteria: NFVI Conformance testing should complete with following exit criteria:

- All mandatory test cases should pass.
- Test results collated, centralized, and normalized, with a final report generated showing status of the test scenario/case (e.g. Pass, Fail, Skip, Measurement Success/Fail, etc), along with trace-ability to a functional, or non-functional, requirement.

2.8 Framework Requirements

The NFVI Conformance framework deals with the process of testing NFVI in below three areas:

- Compliance: The Cloud Infrastructure needs to comply to Anuket RA1/RA2.
- Validation: Validation deals with the ability of NFVI to respond to Cloud APIs and interfaces.
- Performance: Performance deals with running tests on NFVI depending on the NFVI profile and collecting KPIs.

The Cloud Infrastructure KPIs are compared with Golden KPIs, which serve as a reference for the Cloud Infrastructure Conformance. If the Cloud Infrastructure KPIs meet Golden KPIs, The Cloud Infrastructure is certified and granted a Conformance badge. If the Cloud Infrastructure KPIs do not meet Golden KPIs, no Conformance badge is provided.

Best Practices (General)

The NFVI Conformance framework will be guided by the following core principles:

- Implementing, and adhering to, Standardized Test Methodology / flow, Test Plan, and Test Case Suites, which promotes scalability using repeatable processes.
- Integration with Automated Tool-Chains, such as XTesting or Dovetail, for continuous deployment, validation, and centralization of test harnesses and results visualization.
- Alliance and execution of OVP flows and methodologies, which supports common structures for code, artifact generation and repository, Conformance criteria, etc.)
- Where possible, leveraging ONAP Network and Service Models, with identified VNF-specific parameters
- Utilizing Standard Conformance criteria.
- Defining reference architecture (RA) as scenarios, and having all test cases for the RA be involved in OVP
- Add test cases from operators, which operators already tested in their environment

Testing

Testing for NFVI Conformance falls under three broad categories - Compliance, Validation and Performance. Target NFVI for Conformance needs to pass all these tests in order to obtain the Conformance badge.

Test Categories

The following five test categories have been identified as **minimal testing required** to verify NFVI interoperability to satisfy the needs of VNF developer teams.

1. Baremetal validation: To validate control and compute nodes hardware
2. VNF Interoperability: After VNFs are on-boarded, Openstack resources like Tenant, Network (L2/L3), CPU Pining, security policies, Affinity anti-affinity roles and flavors etc. would be validated.
3. Compute components: Validate VMs status and connectivity result after performing each of listed steps. Best candidate for this testing would be identify compute node that holds VMs which has L2 and L3 connectivity.
4. Control plane components: Validations for RabbitMQ, Ceph, MariaDB etc. and OpenStack components like Nova/Glance/Heat etc. APIs.
5. Security: Validation for use RBAC roles and user group policies. See *VNF Testing Cookbook* for complete list.

The following **Optional Test Categories** which can be considered by the Operator, or Supplier, for targeted validations to complement required testing for Conformance:

- On-Boarding (MANO agnostic)
- VNF Functional Testing
- Charging / Revenue Assurance Verification
- MicroServices Support
- Closed Loop Testing
- VNF Coexistence (ETSI NFV-TST001 “Noisy Neighbor”)
- VNF Interactions with Extended NFVi Topology
- VNF Interactions with Complex NFVi (Akraino)

- Scalability Testing
- HA Testing
- Fault Recovery Testing
- PM/KPI/Service Assurance Testing

Test Harnesses

In addition to General Best Practices for NFVI Conformance, the following Quality Engineering (QE) standards will be applied when defining and delivering test scenarios for Conformance:

1. Standardized test methodologies / flows capturing requirements from RA's, goals and scenarios for test execution, and normalizing test results.
2. Establishing, and leveraging, working test-beds which can be referenced in subsequent test scenario designs.
3. Leveraging standardized cloud-based facilities such as storage, IAM, etc.
4. Test Script libraries need to enable Data-Driven testing of On-Boarding, Instantiation, etc.
5. Standards base Test Plan and Test Case suite needs to include sample VNFs, CSAR, and Automated Test Cases.
6. Documentation needs to be dynamic, and consumable.
7. Harnesses need to apply a “Just add Water” deployment strategy, enabling test teams to readily implement test harnesses which promotes Conformance scalability.

Test Results

Categorization. Test suites will be categorized as Functional or Performance based.

Results. Test results reporting will be communicated as a boolean (pass/fail), or Measurements Only.

- **Functional Pass/Fail** signals the assertions set in a test script verify the Functional Requirements (FR) has met its stated objective as delivered by the developer. This will consist of both positive validation of expected behavior, as well as negative based testing when to confirm error handling is working as expected.
- **Performance-based Pass/Fail** determination will be made by comparing Non-Functional (NFR) NFVI KPIs (obtained after testing) with the Golden KPIs. Some of the examples of performance KPIs include, but not limited to: TCP bandwidth, UDP throughput, Memory latency, Jitter, IOPS etc. See [Infrastructure Capabilities, Measurements and Catalogue](#) for a complete list of metrics and requirements.
- **Measurement Results.** Baseline Measurements will be performed when there are no benchmark standards to compare results, or established FRs/NFRs for which to gauge application / platform behavior in an integrated environment, or under load conditions. In these cases, test results will be executed to measure the application, platform, then prepare FRs/NFRs for subsequent enhancements and test runs.

Collation | Portal. The following criteria will be applied to the collation and presentation of test-runs seeking NFVI Conformance:

- RA number and name (e.g. RA-1 OpenStack)
- Version of software tested (e.g. OpenStack Ocata)
- Normalized results will be collated across all test runs (i.e. centralized database)
- Clear time stamps of test runs will be provided.
- Identification of test engineer / executor.

- Traceability to requirements.
- Summarized conclusion if conditions warrant test Conformance (see Badging Section).
- Portal contains links to Conformance badge(s) received.

Badging

Defined. *Badging* refers to the granting of a Conformance badge by the OVP to Suppliers/Testers of Anuket NFVI upon demonstration the testing performed confirms:

- NFVI adheres to Anuket RA/RM requirements.
- Anuket certified VNFs functionally perform as expected (i.e. test cases pass) on NFVI with acceptable levels of stability and performance.

The below figure shows the targeted badge for NFVI.



Figure 2.4: NFVI badge

Specifics. More specifically, suppliers of NFVI testing seeking infrastructure Conformance are required to furnish the following:

Table 2.1: OVP/CVC Expectation

| Category | OVP/CVC Expectation | Supporting Artifact(s) |
|-------------------|--|----------------------------------|
| Lab | Verification that the delivered test lab conforms to RI-x lab requirements for topology, # of nodes, network fabric, etc | Bare-metal H/W Validations |
| Compliance | Verification that the installed software conforms to RM/RA requirements for required components and configured options and extensions, etc | Manifest S/W Validations |
| Validation | FR Validation of Component and API functional behavior meets requirements specified in RM/RA-x requirements documents | API & Platform Test Results |
| Performance | NFR Validation of Component, Interface, and API, results are within tolerance, or achieve baseline measurements | Performance Test Results |
| Results Reporting | Published of Test Results into centralized and common repository and reporting portal | Normalized Results per Standards |
| Release Notes | Supplier provides concluding remarks, links to artifacts, and demonstration of having met exit criteria for testing | Release Notes |

Conformance Process

Conformance and issuance of NFVI badges will be as follows:

- NFVI supplier utilizes, or installs a target RM/RA-x in a RI lab.
- Required artifacts are submitted/supplied to the OVP, demonstrating proper Lab Installation, Compliance, Validation, Performance, and Release of Results & Known Issues.
- Artifact validations will be corroborated and confirmed by the OVP. with direct comparison between measured results and documented FRs/NFRs for applications, hardware and software configuration settings, and host systems.
- All OVP inquiries, requests for re-tests, or reformatting / re-uploading of results data are closed.



Figure 2.5: NFVI Badges

2.9 NFVI Test Cases Requirements

The objective of this chapter is to describe the requirements for NFVI test cases as derived from the reference model and architecture for the LFN-based compliance program. This set of requirements eventually determines the scope of the compliance program and the corresponding list of test cases included in the compliance program. In particular, this chapter extends the generic list of NFVI test case requirements which is provided in Section Test Case Selection Requirements [Multi-Cloud Interactions Model](#) of the reference model.

Generic Requirements on Test Cases

All test cases must fulfill the generic requirements listed in Section [Multi-Cloud Interactions Model](#) of the reference model.

In addition, for test cases targeting the NFVI compliance program, the following requirements must be met:

Table 2.2: NFVI compliance requirements

| Reference | Description |
|-----------|---|
| x | All NFVI test cases <i>must</i> be automated. Once the pre-conditions of a test case are met, i.e., the system under test is configured and in a state according to the pre-conditions of the particular test case, no manual steps must be required to run a test case to completion. |
| x | All NFVI test cases <i>must</i> be implemented using publicly available open source tools. This enables access to test tools and test case implementations to all interested parties and organizations. |
| x | All NFVI test cases <i>must</i> be integrated and run in the Anuket CI/CD pipeline. This requirement ensures that test cases are functionally correct, reliable, mature and pass on the NFVI reference implementation. |
| x | All NFVI test cases <i>must</i> treat the NFVI platform as a black box. In particular, test cases must not perform actions on or change the state of the system under test outside the scope of well-defined APIs as listed by RA1. This requirement ensures applicability of test cases across different implementations: reference implementations as well as commercial implementations. |

Requirement Types

The compliance and Conformance program intends to validate four different types of requirements and system properties:

- **API compliance:** This is the most relevant type of test case, validating the functional correctness of the system under test. API compliance test cases exercise only the specific well-defined APIs described in the reference architecture (see Interfaces and APIs in [Interfaces and APIs](#)).
- **Performance:** Test cases covering this type of requirement measure specific performance characteristics of the system under test as defined in the reference model, the corresponding reference architectures and in sections further below in this chapter.
- **Resilience:** Test cases covering this type of requirement measure specific resilience characteristics of the system under test as defined in the reference model, the corresponding reference architectures and in sections further below in this chapter.
- **Hardware configuration:** Validation of the bare-metal hardware itself in terms of specs and configuration should be included in the scope of the compliance test suite eventually. This validation step ensures that the underlying hardware is correctly configured according to Anuket hardware specification (TODO: add reference to updated “Pharos specs”). The purpose of this validation is to act as a pre-flight check before performing the extensive compliance test suite. Moreover, by validating key hardware configuration aspects, it ensures comparability of performance-related test results.

The extend to which these different types of requirements are included in the compliance and Conformance test suite is subject to the availability of test cases. See Section [NFVI Test Cases Requirements](#) below.

Profile Catalog

Section Infrastructure Profiles Catalogue [Profiles and Workload Flavours](#) of the reference model defines two software profiles, targeting two different use cases:

- Basic
- Network intensive

The test cases selected for validating compliance of the two profiles must cover the functional and non-functional requirements as listed in Section Instance Capabilities Mapping [Virtual Network Interface Specifications](#) and Section Instance Performance Measurement Mapping in [Storage Extensions](#) of the reference model.

TODO: what actually needs to be done here is to reference the table from chapter 4.2.5 and mark for which of those requirements test cases are actually available in the set of test tools available to us.

Software & Hardware Reference

The LFN-based compliance and Conformance program comprises three distinct types of NFVI deployment and runtime environments:

1. A reference implementation deployed in the CI/CD environment,
2. A commercial NFVI product deployed in a vendor's internal development and testing environment, and
3. A reference implementation of a commercial NFVI product deployed in a 3rd party lab providing testing and Conformance services.

The test tooling, harnesses and corresponding test cases which are part of the compliance and Conformance test suite must be capable of running across all of those environments. This results in the following list of requirements:

Table 2.3: Test case requirements

| Reference | Description |
|-----------|---|
| x | NFVI test cases <i>must not</i> interact with remote (Internet) services apart from downloading container or VM images. In particular, test tools and test cases must not automatically upload test data to any system or service run by LFN or GSMA. The purpose of this requirement is to protect the confidentiality of (intermediate) test data. |
| x | NFVI test cases <i>must</i> support a means of running in an internal enterprise lab environment. This could be achieved by either i) natively supporting proxied Internet connectivity and non-public DNS servers or ii) by providing a high-level description of remote dependencies (e.g., container and VM images, network services (DNS), etc.) such that local mirrors can be set up. |

Options & Extensions

Measurement Criteria

Test validations will be corroborated, and confirmed, with direct comparison between measured results and documented non-functional requirements (NFRs) for applications, hardware and software configuration settings, and host systems. Throughput, latency, concurrent connections/threads, are all examples of non-functional requirements which specify criteria which can be used to judge the operation of a system, rather than specific behavior of the application which are defined by functional requirements.

This section attempts to summarize a categorical list of metrics used for test validations. **For a complete list of metrics, and requirements, please refer to Reference Model**

Storage and IOPS

IOPS validations for Storage, and/or Storage Extensions, will be included as part of the final NFVI verification, and validation, process.

From a definition perspective, IOPS is the standard unit of measurement for I/O (Input/Output) operations per second. This measurement is a performance-based measurement and is usually seen written as**(1)**:

- **Total IOPS**: Average number of I/O operations per second.
- **Read IOPS**: Average number of read I/O operations per second.
- **Write IOPS**: Average number of write I/O operations per second.

For example, if you have a disk that is capable of doing a 100 IOPS, it means that it is theoretically capable of issuing a 100 read and or write operations per second. This is in theory. In reality, additional time is needed to actually process the 100 reads/writes. This additional time is referred to as “latency”, which reduces the total IOPS that is calculated, and measured. Latency needs to be measured, and included in the IOPS calculation. Latency will tell us how long it takes to process a single I/O request, and is generally in the 2 millisecond (ms) range per IO operation for a physical disk, through 20+ ms, at which time users will notice an impact in their experience**(2)**.

Additional factors to consider when measuring IOPS:

- Take into consideration the percentage of Input (write) vs. Output (reads) operations, as Writes can be more resource intensive.
- Determine if Reads were performed from Cache, as this may (will) result in faster performance, and faster IOPS.
- Confirm the storage types (Physical, RAID), as storage arrays with linear, or sequential reading/writing may (will) be slower.
- Identify the block size used, as using large block sizes vs. small block sizes can (will) impact IOPS performance.
- Determine Hard Disk Speeds (HDD in RPMs) used, as the higher the RPMS, the potential for faster IOPS performance.
- Quantify the number of disk controllers used to process the number of requested IO requests.
- Determine the specific work-load requirements, as this will dictate speed, controllers, disk RPM, and latency tolerances.

For additional insight, or deeper understanding and reading of IOPS, refer to the references below.

Measurement Types

Performance Measurements

Objectives

The NFVI performance measurements aim at assessing the performance of a given NFVI implementation on the execution plan (i.e., excluding VIM) by providing it with a set of significant metrics to be measured.

They should allow validating the performance of any software and/or hardware NFVI implementation as described in Reference Model.

Of course, they can also be used for other purposes, such as:

- fine tuning of software and/or hardware NFVI configuration (e.g., the number of cores dedicated to the DPDK vSwitch)
- comparing the performances of different software or hardware technologies (e.g., DPDK vSwitch vs hardware-offloaded vSwitch)

- assessing the performance impact of specific features (e.g., with or without encapsulation)

Metrics Baseline

For the purpose of validation, a baseline of the performance metrics is required for comparison with the results of their measurements on the NFVI implementation to be validated.

That baseline is a set of threshold values which could be determined by **measuring the performance metrics on Reference Implementations**.

The validation can then be based on simple pass/fail test results or on a grade (e.g., “class” A, B or C) provided by the combination of pass/fail results for 2 different threshold values of some (or all) metrics.

Metrics Description

Two categories of metrics are considered depending on whether they are related to either the VNF domain or the NFVI domain itself:

- Metrics related to the VNF domain are defined from VNF perspective (i.e., per VNFC, per vNIC, per vCPU...) and should concern VNF as well as NFVI actors.
- Metrics related to the NFVI domain are defined per NFVI node ; their measurement is based on virtual workloads (i.e., VM or container) in order to reflect the performance of a NFVI node with a given profile ; they should only concern NFVI actors.

The following table contains the list of performance metrics related to the VNF domain.

Table 2.4: Performance metrics related to the VNF domain

| Reference | Name | Unit | Definition/Notes |
|-------------------|--------------------------------|---------------------|---|
| vnf.nfvi.perf.001 | vNIC throughput | bits/s | Throughput per vNIC |
| vnf.nfvi.perf.002 | vNIC latency | second | Frame transfer time to vNIC at the throughput (vnf.nfvi.perf.001) |
| vnf.nfvi.perf.003 | vNIC delay variation | second | Frame Delay Variation (FDV) to vNIC at the throughput (vnf.nfvi.perf.001) |
| vnf.nfvi.perf.004 | vNIC simultaneous active flows | number | Simultaneous active L3/L4 flows per vNIC before a new flow is dropped |
| vnf.nfvi.perf.005 | vNIC new flows rate | flows/s | New L3/L4 flows rate per vNIC |
| vnf.nfvi.perf.006 | Storage throughput | bytes/s | Throughput per virtual storage unit |
| vnf.nfvi.perf.007 | vCPU capacity | test-specifics core | Compute capacity per vCPU |

The following table contains the list of performance metrics related to the NFVI domain.

Table 2.5: Performance metrics related to the NFVI domain

| Reference | Name | Unit | Definition/Notes |
|---------------------|--------------------------------|---------------------|---|
| infra.nfvi.perf.001 | Node network throughput | bits/s | Network throughput per node |
| infra.nfvi.perf.002 | Node simultaneous active flows | number | Simultaneous active L3/L4 flows per node before a new flow is dropped |
| infra.nfvi.perf.003 | Node new flows rate | flows/s | New L3/L4 flows rate per node |
| infra.nfvi.perf.004 | Node storage throughput | bytes/s | Storage throughput per node |
| infra.nfvi.perf.005 | Physical core capacity | test-specifics core | Compute capacity per physical core usable by VNFs |
| infra.nfvi.perf.006 | Energy consumption | W | Energy consumption of the node without hosting any VNFC |
| infra.nfvi.perf.007 | Network energy efficiency | W/bits/s | Energy consumption of the node at the network throughput, (infra.nfvi.perf.001), normalized to the measured bit rate |
| infra.nfvi.perf.008 | Storage energy efficiency | W/bytes/s | Energy consumption of the node at the storage throughput (infra.nfvi.perf.004), normalized to the measured byte rate |
| infra.nfvi.perf.009 | Compute energy efficiency | W/core | Energy consumption of the node during compute capacity test (vnf.nfvi.perf.007 or infra.nfvi.perf.005), normalized to the number of physical cores usable by VNFs |

MVP Metrics

The following metrics should be considered as MVP:

- vnf.nfvi.perf.001,002,006,007
- infra.nfvi.perf.001,005,006,007,009

Network Metrics Measurement Test Cases

The network performance metrics are vnf.nfvi.perf.001-005 and infra.nfvi.perf.001-003,006.

The different possible test cases are defined by each of the 3 following test traffic conditions.

- **Test traffic path across NFVI**

3 traffic path topologies should be considered:

- **North/South traffic**, between VNFCs within a node and outside NFVI
This can be provided by PVP test setup of ETSI GS NFV-TST009.
- **East/West intra-node traffic**, between VNFCs within a node
This can be provided by a V2V (Virtual-to-Virtual) test setup and, in some cases, by PVVP test setup of ETSI GS NFV-TST009.
- **East/West inter-node traffic**, between VNFCs in different nodes
This can be provided by VPV (Virtual-Physical-Virtual) test setup and, in some cases, by PVVP test setup between 2 nodes.

- **Test traffic processing by NFVI**

Different processing complexity applicable to the traffic crossing the NFVI should be considered, including especially (but not exhaustively):

- **L2 processing** (Ethernet switching), possibly including VLAN tagging/mapping and encapsulation (e.g., VXLAN)
- **L3 processing** (IP routing), possibly including L2 processing
- **L4 stateful processing** (e.g., FW, NAT, SFC), also including L3 processing
- **Encryption** (e.g., IPSec ESP tunneling)

• **Test traffic profile**

Two different test traffic profiles should be considered according to the two VNF types that must be provided with network connectivity by the NFVI.

- **Forwarded traffic** for L3/L4 forwarding VNF (e.g., PGW, FW)

It is based on ETSI GS NFV-TST009 and it should be:

- * **bidirectional UDP traffic** with **0.001%** frame loss ratio, **300B** average frame size, **10k** L3/L4 flows,
- * between a **traffic generator** and a **traffic receiver** through a **L3 forwarding** pseudo-VNF with sufficient capacity not to be the test bottleneck.

Latency and delay variation measurement should be the 99th percentile of measured values for one-way frame transfer (i.e. from generator to receiver).

The main Anuket test tools candidates for that purpose are NFVbench and VSPerf.

Note: to be studied whether additional frame sizes and flows number should be considered

- **Client-server traffic** for L4/L7 endpoint VNF (e.g., MME, CDN)

It should be:

- * **bidirectional TCP traffic** with **1400B** maximum frame size, **5k** TCP sessions,
- * between **2 TCP client&server endpoints**, one or both as pseudo-VNF, with sufficient capacity not to be the test bottleneck.

Note: the maximum TCP frame size can be forced by configuring TCP endpoint link MTU.

The main Anuket test tool candidate for that purpose is Functest (VMTP and Shaker).

Note: to be studied whether metrics related to latency and flows for that traffic profile should be considered (how? with UDP and/or ICMP test traffic in addition?)

The combination of each of those 3 test conditions types and the different NFVI profiles results in a wide matrix of test cases (potentially more than 50 cases). Furthermore, these test cases should be combined with the different metrics resulting in a huge number of measurements (potentially more than 400 measurements). For the efficiency of the validation, only the most relevant combinations should be kept.

This optimization should be based on the following principles:

1. NFVI domain metrics measurement: on PVP topology only
2. Metrics measurement with forwarded traffic: with no L4 stateful processing
3. Basic profile metrics measurement: client-server traffic profile only
4. Flows & latency related metrics measurement: for PVP only

The following table proposed a possible optimized matrix model of the test cases against the metrics to be measured.

Table 2.6: Optimized matrix model of the test cases against the metrics to be measured

| | NFVI Profiles | B | | | | N | |
|-----------------|---------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| | Test Cases | V2V - L2 - SRV | VPV - L3 - SRV | PVP - L2 - SRV | PVP - L4 - SRV | PVP - L2 - SRV | PVP - L2 - FWD |
| MVP Metrics | vnf.nfvi.perf.001 | 50Gbps | 20Gbps | 20Gbps | 10Gbps | 40Gbps | 40Gbps |
| | vnf.nfvi.perf.002 | n/a (4) | n/a (4) | ? | ? | ? | 0.5ms |
| | infra.nfvi.perf.001 | n/a (1) | n/a (1) | 40Gbps | 20Gbps | 60Gbps | 80Gbps |
| | infra.nfvi.perf.007 | n/a (1) | n/a (1) | ? W/Gbps | ? W/Gbps | ? W/Gbps | ? W/Gbps |
| Non-MVP Metrics | vnf.nfvi.perf.003 | n/a (4) | n/a (4) | ? | ? | ? | 1ms |
| | vnf.nfvi.perf.004 | n/a (4) | n/a (4) | ? | ? | ? | 500k |
| | vnf.nfvi.perf.005 | n/a (4) | n/a (4) | ? | ? | ? | 110kfps |
| | infra.nfvi.perf.002 | n/a (1) | n/a (1) | ? | ? | ? | 1G |
| | infra.nfvi.perf.003 | n/a (1) | n/a (1) | ? | ? | ? | 200kfps |

Table notes:

- Values are only indicative (see “Metrics Baseline” below)
- L2/L3/L4 refers to network processing layer
 - L2 for Ethernet switching
 - L3 for IP routing
 - L4 for IP routing with L4 stateful processing (e.g. NAT)
- SRV/FWD refers to the traffic profile (and pseudo-VNF type implied)
 - SRV for client-server traffic (and L4/L7 endpoint pseudo-VNF)
 - FWD for forwarded traffic (and L3/L4 forwarding pseudo-VNF)

Energy Metrics Measurement Test Cases

Energy metrics (infra.nfvi.perf.006-009) should be considered carefully for NFVI validation since energy consumption may vary a lot across processor architectures, models and power management features.

They mainly enable to have metrics available regarding NFVI environment footprint. They also allow energy-based comparison of different NFVI software implementations running on a same physical NFVI hardware implementation.

Storage Metrics Measurement Test Cases

Metric (MVP): vnf.nfvi.perf.006 and infra.nfvi.perf.004,008

Compute Metrics Measurement Test Cases

The compute performance metrics are vnf.nfvi.perf.007 and infra.nfvi.perf.004,009.

For normalized results, the compute performance test requires all of the possible vCPUs available for running workloads to execute workloads. You need to start as many VMs as needed to force all of the possible CPUs on the node to run a workload. In this case, the result is normalized:

- to the number of vCPU, for the vCPU capacity measurements (vnf.nfvi.perf.007)
- to the number of physical core usable by VNFs, for the physical core capacity and compute energy efficiency measurements (infra.nfvi.perf.004,009)

Resiliency Measurements

3 Cloud Infrastructure Test Cases and Traceability to Requirements

3.1 Introduction

The scope of this chapter is to identify and list down test cases based on requirements defined in [Reference Architecture for OpenStack based Cloud Infrastructure \(RA1\)](#). This will serve as traceability between test cases and requirements.

Note that each requirement may have one or more test cases associated with it.

must: Test Cases that are marked as must are considered mandatory and must pass successfully.

should: Test Cases that are marked as should are expected to be fulfilled by NFVI but it is up to each service provider to accept an NFVI targeting reference architecture that is not reflecting on any of those requirements. The same applies to should not.

may: Test cases that are marked as may are considered optional. The same applies to may not.

3.2 Selection Criteria

Test cases below are selected based on available test cases in open-source tools like FuncTest etc.

3.3 Traceability Matrix

The following is a Requirements Traceability Matrix (RTM) mapping Test Case, and/or Test Case Coverage, to RM and RA-1 requirements (config and deployment).

The RTM contains RM config (i.e. .conf) requirements listed “per profile”, followed by RA-1 requirements. Requirements fall into 8 domains: general(gen), infrastructure(inf), VIM(vim), Interface & API(int), Tenants(tnt), LCM(lcm), Assurance(asr), Security(sec).

For detailed information on RM & RA-1 NFVI and VNF requirements, please refer to [Cloud Infrastructure + VNF Target State & Specification](#).

Architecture and OpenStack Requirements

Infrastructure

VIM

Interfaces & APIs

The [OpenStack Gates](#) verify all changes proposed mostly by running thousands of Tempest tests completed by Rally scenarios in a few cases. Skipping tests is allowed in all OpenStack Gates and only failures rate the review -1 because of the multiple capabilities and backends selected in the different Gate jobs. The classical [FuncTest containers](#) conform to this model which also fits the heterogeneous user deployments.

From a Anuket Compliance state point, the capabilities are well described in [Interfaces and APIs](#) which allows tuning the test configurations and the test lists to avoid skipping any test. It results that all tests covering optional capabilities and all upstream skipped tests due to known bugs are not executed. All remaining tests must be executed and must pass successfully.

New [Functest containers](#) have been proposed for Anuket Compliance which simply override the default test configurations and the default test lists. Any optional capability or services (e.g. Barbican) can be still verified by the classical Functest containers.

The next subsections only detail the Tempest tests which must not be executed from a Compliance state point. The remaining tests have to pass successfully. They cover all together the API testing requirements as asked by [Interfaces and APIs](#)

The following software versions are considered here to verify OpenStack Wallaby selected by Anuket:

Table 3.1: Software versions

| Software | Version |
|-------------------------|-------------|
| Functest | wallaby |
| Cinder Tempest plugin | 1.4.0 |
| Keystone Tempest plugin | 0.7.0 |
| Heat Tempest plugin | 1.2.0 |
| Neutron Tempest plugin | 1.4.0 |
| Rally OpenStack | 2.2.1.dev11 |
| Tempest | 27.0.0 |

Identity - Keystone API testing

Keystone API is covered in the OpenStack Gates via [Tempest](#) and [keystone-tempest-plugin](#) as integrated in [Functest Smoke CNTT](#).

According to [Interfaces and APIs](#) the following test names must not be executed:

Table 3.2: Keystone API testing

| Test rejection regular expressions | Reasons |
|---|--------------|
| .*api.identity.v3.test_oauth1_tokens | oauth1 |
| .*scenario.test_federated_authentication | federation |
| .*identity.admin.v2 | API v2 |
| .*identity.v2 | API v2 |
| .*identity.v3.test_access_rules | access_rules |
| .*identity.v3.test_application_credentials.\ApplicationCredentialsV3Test.\test_create_application_credential_access_rules | access_rules |

Keystone API is also covered by [Rally](#).

Here are the mainline tasks integrated in [Functest Smoke CNTT](#):

- Authenticate.keystone
- KeystoneBasic.add_and_remove_user_role
- KeystoneBasic.create_add_and_list_user_roles
- KeystoneBasic.create_and_list_tenants
- KeystoneBasic.create_and_delete_role
- KeystoneBasic.create_and_delete_service
- KeystoneBasic.get_entities
- KeystoneBasic.create_update_and_delete_tenant

- KeystoneBasic.create_user
- KeystoneBasic.create_tenant
- KeystoneBasic.create_and_list_users
- KeystoneBasic.create_tenant_with_users

Image - Glance API testing

Glance API is covered in the OpenStack Gates via [Tempest](#) as integrated in [Functest Smoke CNTT](#).

According to [Interfaces and APIs](#) the following test names must not be executed:

Table 3.3: Glance API testing

| Test rejection regular expressions | Reasons |
|---|--------------------|
| .*image.v1 | API v1 |
| .*image.v2.admin.test_images.ImportCopyImagesTest | import_image |
| .*image.v2.test_images_negative.ImagesNegativeTest.\test_create_image_reserved_property | os_glance_reserved |
| .*image.v2.test_images_negative.ImagesNegativeTest.\test_update_image_reserved_property | os_glance_reserved |
| .*image.v2.test_images_negative.ImportImagesNegativeTest.\test_image_web_download_import_with_bad_url | web-downloadimport |
| .*image.v2.test_images.ImportImagesTest | import_image |
| .*image.v2.test_images.MultiStoresImportImages | import_image |

Glance API is also covered by [Rally](#).

Here are the mainline tasks integrated in [Functest Smoke CNTT](#):

- Authenticate.validate_glance
- GlanceImages.create_and_delete_image
- GlanceImages.create_and_list_image
- GlanceImages.list_images
- GlanceImages.create_image_and_boot_instances

Block Storage - Cinder API testing

Cinder API is covered in the OpenStack Gates via [Tempest](#) and [cinder-tempest-plugin](#) as integrated in [Functest Smoke CNTT](#).

According to [Interfaces and APIs](#) the following test names must not be executed:

Table 3.4: Cinder API testing

| Test rejection regular expressions | Reasons |
|---|---|
| .*test_incremental_backup | https://gerrit.opnfv.org/gerrit/68881 |
| .*test_consistencygroups | consistency_group |
| .*test_backup_crossproject_admin_negative | https://gerrit.opnfv.org/gerrit/71011 |
| .*test_backup_crossproject_user_negative | https://gerrit.opnfv.org/gerrit/71011 |
| .*test_volume_encrypted.TestEncryptedCinderVolumes | attach_encrypted_volume |
| .*test_encrypted_volumes_extend | extend_attached_encrypted_volume |
| .*test_group_snapshots.GroupSnapshotsV319Test.\ntest_reset_group_snapshot_status | https://launchpad.net/bugs/1770179 |
| .*test_multi_backend | multi-backend |
| .*test_volume_retype.VolumeRetypeWithMigrationTest | multi-backend |
| .*test_volume_delete_cascade.VolumesDeleteCascade.\ntest_volume_from_snapshot_cascade_delete | https://launchpad.net/bugs/1677525 |
| .*test_volumes_backup.VolumesBackupsTest.\ntest_volume_backup_create_get_detailed_list_restore_delete | ceph |
| .*test_volumes_extend.VolumesExtendAttachedTest.\ntest_extend_attached_volume | extend_attached_volume |
| .*tempest.scenario.test_volume_migrate_attached | multi-backend |

Cinder API is also covered by Rally.

Here are the mainline tasks integrated in Functest Smoke CNTT:

- Authenticate.validate_cinder
- CinderVolumes.create_and_delete_snapshot
- CinderVolumes.create_and_delete_volume
- CinderVolumes.create_and_extend_volume
- CinderVolumes.create_from_volume_and_delete_volume
- CinderQos.create_and_list_qos
- CinderQos.create_and_set_qos
- CinderVolumeTypes.create_and_list_volume_types
- CinderVolumeTypes.create_volume_type_and_encryption_type
- Quotas.cinder_update_and_delete
- Quotas.cinder_update

Object Storage - Swift API testing

Swift API is covered in the OpenStack Gates via Tempest as integrated in Functest Smoke CNTT.

According to Interfaces and APIs the following test names must not be executed:

Table 3.5: Swift API testing

| Test rejection regular expressions | Reasons |
|---|---|
| .*test_container_sync.ContainerSyncTest.\ntest_container_synchronization | https://launchpad.net/bugs/1317133 |
| .*test_container_sync_middleware.ContainerSyncMiddlewareTest.\ntest_container_synchronization | container_sync |
| .*test_object_services.ObjectTest.\ntest_create_object_with_transfer_encoding | https://launchpad.net/bugs/1905432 |

Swift API is also covered by Rally.

Here are the mainline tasks integrated in Functest Smoke CNTT:

- SwiftObjects.create_container_and_object_then_list_objects
- SwiftObjects.list_objects_in_containers
- SwiftObjects.create_container_and_object_then_download_object
- SwiftObjects.create_container_and_object_then_delete_all
- SwiftObjects.list_and_download_objects_in_containers

Networking - Neutron API testing

Neutron API is covered in the OpenStack Gates via Tempest and neutron-tempest-plugin as integrated in Functest Smoke CNTT.

According to Interfaces and APIs the following test names must not be executed:

Table 3.6: Neutron API testing

| Test rejection regular expressions | Reasons |
|---|-----------------------------|
| .*admin.test_agent_availability_zone | DHCP agent and L3 agent |
| .*admin.test_dhcp_agent_scheduler | dhcp_agent_scheduler |
| .*admin.test_l3_agent_scheduler | l3_agent_scheduler |
| .*admin.test_logging | logging |
| .*admin.test_logging_negative | logging |
| .*admin.test_network_segment_range | network-segment-range |
| .*admin.test_ports.PortTestCasesAdmin.\ntest_regenerate_mac_address | port-mac-address-regenerate |
| .*admin.test_ports.PortTestCasesResourceRequest | port-resource-request |
| .*admin.test_routers_dvr | dvr |
| .*admin.test_routers_flavors | l3-flavors |
| .*admin.test_routers_ha | l3-ha |
| .*test_floating_ips.FloatingIPPoolTestJSON | floatingip-pools |
| .*test_floating_ips.FloatingIPTestJSON.\ntest_create_update_floatingip_port_details | fip-port-details |
| .*test_metering_extensions | metering |
| .*test_metering_negative | metering |
| .*test_networks.NetworksSearchCriteriaTest.\ntest_list_validation_filters | filter-validation |
| .*test_networks.NetworksTestAdmin.\ntest_create_tenant_network_vxlan | vxlan |

continues on next page

Table 3.6 – continued from previous page

| Test rejection regular expressions | Reasons |
|---|---|
| .*test_networks.NetworksTestJSON.\ test_create_update_network_dns_domain | dns-integration |
| .*test_port_forwardings | floating-ip-port-forwarding |
| .*test_port_forwarding_negative | floating-ip-port-forwarding |
| .*test_ports.PortsTaggingOnCreation | tag-ports-during-bulk-creation |
| .*test_ports.PortsTestJSON.\ test_create_port_with_propagate_uplink_status | uplink-status-propagation |
| .*test_ports.PortsTestJSON.\ test_create_port_without_propagate_uplink_status | uplink-status-propagation |
| .*test_ports.PortsTestJSON.\ test_create_update_port_with_dns_domain | dns-domain-ports |
| .*test_ports.PortsTestJSON.\ test_create_update_port_with_dns_name | dns-integration |
| .*test_ports.PortsTestJSON.\ test_create_update_port_with_no_dns_name | dns-integration |
| .*test_revisions.TestRevisions.\ test_update_dns_domain_bumps_revision | dns-integration |
| .*test_revisions.TestRevisions.\ test_update_router_extra_attributes\ bumps_revision | l3-ha |
| .*test_router_interface_fip | router-interface-fip |
| .*test_routers.DvrRoutersTest | dvr |
| .*test_routers.HaRoutersTest | l3-ha |
| .*test_routers.RoutersIPv6Test.\ test_extra_routes_atomic | extraroute-atomic |
| .*test_routers.RoutersTest.\ test_extra_routes_atomic | extraroute-atomic |
| .*test_routers_negative.DvrRoutersNegativeTest | dvr |
| .*test_routers_negative.\ DvrRoutersNegativeTestExtended | dvr |
| .*test_routers_negative.HaRoutersNegativeTest | l3-ha |
| .*test_security_groups.RbacSharedSecurityGroupTest | rbac-security-groups |
| .*test_subnetpool_prefix_ops | subnetpool-prefix-ops |
| .*test_subnetpools.RbacSubnetPoolTest | rbac-subnetpool |
| .*test_subnetpools_negative.SubnetPoolsNegativeTestJSON.\ test_tenant_create_subnetpool_associate_shared_address_scope | rbac-subnetpool |
| .*test_subnetpools.SubnetPoolsSearchCriteriaTest.\ test_list_validation_filters | filter-validation |
| .*test_subnets.SubnetsSearchCriteriaTest.\ test_list_validation_filters | filter-validation |
| .*test_timestamp.TestTimeStamp.\ test_segment_with_timestamp | standard-attr-segment |
| .*test_trunk.TrunkTestInheritJSONBase.\ test_add_subport | https://launchpad.net/bugs/1863707 |
| .*test_trunk.TrunkTestMtusJSON | vxlan |
| .*test_trunk_negative.TrunkTestJSON.\ test_create_subport_invalid_inherit_network\ segmentation_type | vxlan |
| .*test_trunk_negative.TrunkTestMtusJSON | vxlan |
| .*test_qos.QosMinimumBandwidthRuleTestJSON | https://gerrit.opnfv.org/gerrit/69105 |

continues on next page

Table 3.6 – continued from previous page

| Test rejection regular expressions | Reasons |
|--|---|
| .*network.test_tags | tag-ext |
| .*test_routers.RoutersIPv6Test\ test_create_router_set_gateway_with_fixed_ip | https://launchpad.net/bugs/1676207 |
| .*test_routers.RoutersTest\ test_create_router_set_gateway_with_fixed_ip | https://launchpad.net/bugs/1676207 |
| .*test_network_basic_ops\ TestNetworkBasicOps.test_router_rescheduling | l3_agent_scheduler |
| .*test_network_advanced_server_ops\ TestNetworkAdvancedServerOps\ test_server_connectivity_cold_migration_revert | https://launchpad.net/bugs/1836595 |

Neutron API is also covered by Rally.

Here are the mainline tasks integrated in Functest Smoke CNTT:

- Authenticate.validate_neutron
- NeutronNetworks.create_and_delete_networks
- NeutronNetworks.create_and_delete_ports
- NeutronNetworks.create_and_delete_routers
- NeutronNetworks.create_and_delete_subnets
- NeutronNetworks.create_and_list_networks
- NeutronNetworks.create_and_list_ports
- NeutronNetworks.create_and_list_routers
- NeutronNetworks.create_and_list_subnets
- NeutronSecurityGroup.create_and_delete_security_groups
- NeutronSecurityGroup.create_and_delete_security_group_rule
- NeutronNetworks.set_and_clear_router_gateway
- Quotas.neutron_update

Compute - Nova API testing

Nova API is covered in the OpenStack Gates via Tempest as integrated in Functest Smoke CNTT.

According to Interfaces and APIs the following test names must not be executed:

Table 3.7: Nova API testing

| Test rejection regular expressions | Reasons |
|--|------------------------|
| .*admin.test_agents | xenapi_apis |
| .*test_fixed_ips | neutron |
| .*test_fixed_ips_negative | neutron |
| .*test_auto_allocate_network | shared networks |
| .*test_flavors_microversions.FlavorsV255TestJSON | max_microversion: 2.53 |
| .*test_flavors_microversions.FlavorsV261TestJSON | max_microversion: 2.53 |
| .*test_floating_ips_bulk | nova-network |

continues on next page

Table 3.7 – continued from previous page

| Test rejection regular expressions | Reasons |
|---|---|
| .*test_live_migration.\nLiveAutoBlockMigrationV225Test.test_iscsi_volume | block live migration |
| .*test_live_migration.\nLiveAutoBlockMigrationV225Test.\ntest_live_block_migration | block live migration |
| .*test_live_migration.\nLiveAutoBlockMigrationV225Test.\ntest_live_block_migration_paused | block live migration |
| .*test_live_migration.\nLiveAutoBlockMigrationV225Test.\ntest_volume_backed_live_migration | volume-backed live migration |
| .*test_live_migration.LiveMigrationTest.\ntest_iscsi_volume | block live migration |
| .*test_live_migration.LiveMigrationTest.\ntest_live_block_migration | block live migration |
| .*test_live_migration.LiveMigrationTest.\ntest_live_block_migration_paused | block live migration |
| .*test_live_migration.LiveMigrationTest.\ntest_volume_backed_live_migration | volume-backed live migration |
| .*test_live_migration.\nLiveMigrationRemoteConsolesV26Test | serial_console |
| .*test_quotas.QuotasAdminTestV257 | max_microversion: 2.53 |
| .*test_servers.ServersAdminTestJSON.\ntest_reset_network_inject_network_info | xenapi_apis |
| .*certificates.test_certificates | cert |
| .*test_quotas_negative.\nQuotasSecurityGroupAdminNegativeTest | https://launchpad.net/bugs/1186354 |
| .*test_novnc | vnc_console |
| .*test_server_personality | personality |
| .*test_servers.ServerShowV263Test.\ntest_show_update_rebuild_list_server | certified_image_ref |
| .*test_servers_microversions.ServerShowV254Test | max_microversion: 2.53 |
| .*test_servers_microversions.ServerShowV257Test | max_microversion: 2.53 |
| .*test_servers_negative.ServersNegativeTestJSON.\ntest_personality_file_contents_not_encoded | personality |
| .*test_server_actions.ServerActionsTestJSON.\ntest_change_server_password | change_password |
| .*test_server_actions.ServerActionsTestJSON.\ntest_get_vnc_console | vnc_console |
| .*test_server_actions.ServerActionsTestJSON.\ntest_reboot_server_soft | https://launchpad.net/bugs/1014647 |
| .*test_server_rescue.\nServerBootFromVolumeStableRescueTest | stable_rescue |
| .*test_server_rescue.ServerStableDeviceRescueTest | stable_rescue |
| .*test_security_group_default_rules | https://launchpad.net/bugs/1311500 |
| .*test_security_groups_negative.\nSecurityGroupsNegativeTestJSON.\ntest_security_group_create_with_duplicate_name | neutron |

continues on next page

Table 3.7 – continued from previous page

| Test rejection regular expressions | Reasons |
|--|---|
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_security_group_create_with_invalid_group_description | https://launchpad.net/bugs/1161411 |
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_security_group_create_with_invalid_group_name | https://launchpad.net/bugs/1161411 |
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_update_security_group_with_invalid_sg_description | neutron |
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_update_security_group_with_invalid_sg_description | neutron |
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_update_security_group_with_invalid_sg_id | neutron |
| .*test_security_groups_negative.\ SecurityGroupsNegativeTestJSON.\ test_update_security_group_with_invalid_sg_name | neutron |
| .*test_server_metadata.ServerMetadataTestJSON | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_delete_metadata_non_existent_server | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_metadata_items_limit | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_set_metadata_invalid_key | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_set_metadata_non_existent_server | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_set_server_metadata_blank_key | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_set_server_metadata_missing_metadata | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_update_metadata_non_existent_server | xenapi_apis |
| .*test_server_metadata_negative.\ ServerMetadataNegativeTestJSON.\ test_update_metadata_with_blank_key | xenapi_apis |
| .*test_list_server_filters.\ ListServerFiltersTestJSON.\ test_list_servers_filtered_by_ip_regex | https://launchpad.net/bugs/1540645 |
| .*servers.test_virtual_interfaces | nova-network |
| .*compute.test_virtual_interfaces_negative | nova-network |
| .*compute.test_networks | nova-network |

continues on next page

Table 3.7 – continued from previous page

| Test rejection regular expressions | Reasons |
|---|-------------------------|
| .*test_attach_volume.AttachVolumeMultiAttach | volume_multiattach |
| .*test_volume_boot_pattern.\nTestVolumeBootPattern.\n test_boot_server_from_encrypted_volume_luks | attach_encrypted_volume |
| .*test_volume_swap | swap_volume |
| .*test_encrypted_cinder_volumes | attach_encrypted_volume |
| .*test_minbw_allocation_placement | microversion |
| .*test_volumes_negative.\nUpdateMultiattachVolumeNegativeTest.\n test_multiattach_rw_volume_update_failure | volume_multiattach |
| .*test_shelve_instance.TestShelveInstance.\n test_cold_migrate_unshelved_instance | shelve_migrate |

Nova API is also covered by [Rally](#).

Here are the mainline tasks integrated in [Functest Smoke CNTT](#):

- Authenticate.validate_nova
- NovaServers.boot_and_live_migrate_server
- NovaServers.boot_server_attach_created_volume_and_live_migrate
- NovaServers.boot_server_from_volume_and_live_migrate
- NovaKeypair.boot_and_delete_server_with_keypair
- NovaServers.boot_server_from_volume_and_delete
- NovaServers.pause_and_unpause_server
- NovaServers.boot_and_migrate_server
- NovaServers.boot_server_and_list_interfaces
- NovaServers.boot_server_associate_and_dissociate_floating_ip
- NovaServerGroups.create_and_delete_server_group
- Quotas.nova_update

Orchestration - Heat API testing

Heat API is covered in the OpenStack Gates via [heat-tempest-plugin](#) as integrated in [Functest Smoke CNTT](#)

According to [Interfaces and APIs](#) the following test names must not be executed:

Table 3.8: Heat API testing

| Test rejection regular expressions | Reasons |
|--|---|
| .*functional.test_lbaasv2 | lbaasv2 |
| .*functional.test_encryption_vol_type | https://storyboard.openstack.org/#!/story/2007804 |
| .*RemoteStackTest.\ test_stack_create_with_cloud_credential | https://gerrit.opnfv.org/gerrit/c/funcstest/+/-/69926 |
| .*scenario.test_aodh_alarm | aodh |
| .*tests.scenario.test_autoscaling_lb | lbaas |
| .*scenario.test_autoscaling_lbv2 | lbaasv2 |
| .*scenario.test_server_software_config | https://gerrit.opnfv.org/gerrit/c/funcstest/+/-/69926 |
| .*test_volumes.\ VolumeBackupRestoreIntegrationTest | https://gerrit.opnfv.org/gerrit/c/funcstest/+/-/69931 |
| .*scenario.test_octavia_lbaas | octavia |
| .*scenario.test_server_cfn_init | https://gerrit.opnfv.org/gerrit/c/funcstest/+/-/70004 |

Heat API is also covered by Rally.

Here are the mainline tasks integrated in Functest Smoke CNTT:

- Authenticate.validate_heat
- HeatStacks.create_update_delete_stack
- HeatStacks.create_check_delete_stack
- HeatStacks.create_suspend_resume_delete_stack
- HeatStacks.list_stacks_and_resources

Dashboard

Horizon is covered in the OpenStack Gates via tempest-horizon as integrated in Functest Healthcheck.

OpenStack API benchmarking

Rally is tool and framework that allows to perform OpenStack API benchmarking.

Here are the Rally-based test cases proposed by Functest Benchmarking CNTT:

- rally_full: Functest scenarios iterating 10 times the mainline Rally scenarios
- rally_jobs: Neutron scenarios executed in the OpenStack gates

At the time of writing, no KPI is defined in Interfaces and APIs which would have asked for an update of the default SLA (maximum failure rate of 0%) proposed in Functest Benchmarking CNTT

Identity - Keystone API benchmarking

Functest rally_full_cntt:

Table 3.9: Keystone API benchmarking

| Scenarios | Iterations |
|---|------------|
| Authenticate.keystone | 10 |
| KeystoneBasic.add_and_remove_user_role | 10 |
| KeystoneBasic.create_add_and_list_user_roles | 10 |
| KeystoneBasic.create_and_list_tenants | 10 |
| KeystoneBasic.create_and_delete_role | 10 |
| KeystoneBasic.create_and_delete_service | 10 |
| KeystoneBasic.get_entities | 10 |
| KeystoneBasic.create_update_and_delete_tenant | 10 |
| KeystoneBasic.create_user | 10 |
| KeystoneBasic.create_tenant | 10 |
| KeystoneBasic.create_and_list_users | 10 |
| KeystoneBasic.create_tenant_with_users | 10 |

Image - Glance API benchmarking

Functest rally_full_cntt:

Table 3.10: Glance API benchmarking

| Scenarios | Iterations |
|--|------------|
| Authenticate.validate_glance | 10 |
| GlanceImages.create_and_delete_image | 10 |
| GlanceImages.create_and_list_image | 10 |
| GlanceImages.list_images | 10 |
| GlanceImages.create_image_and_boot_instances | 10 |
| GlanceImages.create_and_deactivate_image | 10 |
| GlanceImages.create_and_download_image | 10 |
| GlanceImages.create_and_get_image | 10 |
| GlanceImages.create_and_update_image | 10 |

Block Storage - Cinder API benchmarking

Functest rally_full_cntt:

Table 3.11: Cinder API benchmarking

| Scenarios | Iterations |
|---|------------|
| Authenticate.validate_glance | 10 |
| CinderVolumes.create_and_attach_volume | 10 |
| CinderVolumes.create_and_list_snapshots | 10 |
| CinderVolumes.create_and_list_volume | 10 |
| CinderVolumes.create_and_upload_volume_to_image | 10 |
| CinderVolumes.create_nested_snapshots_and_attach_volume | 10 |
| CinderVolumes.create_snapshot_and_attach_volume | 10 |
| CinderVolumes.create_volume | 10 |
| CinderVolumes.list_volumes | 10 |
| CinderVolumes.create_and_delete_snapshot | 10 |
| CinderVolumes.create_and_delete_volume | 10 |
| CinderVolumes.create_and_extend_volume | 10 |
| CinderVolumes.create_from_volume_and_delete_volume | 10 |
| CinderQos.create_and_get_qos | 10 |
| CinderQos.create_and_list_qos | 10 |
| CinderQos.create_and_set_qos | 10 |
| CinderVolumeTypes.create_and_get_volume_type | 10 |
| CinderVolumeTypes.create_and_list_volume_types | 10 |
| CinderVolumeTypes.create_and_update_volume_type | 10 |
| CinderVolumeTypes.create_volume_type_and_encryption_type | 10 |
| CinderVolumeTypes.create_volume_type_add_and_list_type_access | 10 |
| Quotas.cinder_update_and_delete | 10 |
| Quotas.cinder_update | 10 |

Object Storage - Swift API benchmarking

Functest rally_full_cntt:

Table 3.12: Swift API benchmarking

| Scenarios | Iterations |
|---|------------|
| SwiftObjects.create_container_and_object_then_list_objects | 10 |
| SwiftObjects.list_objects_in_containers | 10 |
| SwiftObjects.create_container_and_object_then_download_object | 10 |
| SwiftObjects.create_container_and_object_then_delete_all | 10 |
| SwiftObjects.list_and_download_objects_in_containers | 10 |

Networking - Neutron API benchmarking

Functest rally_full_cntt:

Table 3.13: Neutron API benchmarking

| Scenarios | Iterations |
|--|------------|
| Authenticate.validate_neutron | 10 |
| NeutronNetworks.create_and_update_networks | 10 |
| NeutronNetworks.create_and_update_ports | 10 |
| NeutronNetworks.create_and_update_routers | 10 |
| NeutronNetworks.create_and_update_subnets | 10 |
| NeutronNetworks.create_and_delete_networks | 10 |
| NeutronNetworks.create_and_delete_ports | 10 |
| NeutronNetworks.create_and_delete_routers | 10 |
| NeutronNetworks.create_and_delete_subnets | 10 |
| NeutronNetworks.create_and_list_networks | 10 |
| NeutronNetworks.create_and_list_ports | 10 |
| NeutronNetworks.create_and_list_routers | 10 |
| NeutronNetworks.create_and_list_subnets | 10 |
| NeutronSecurityGroup.create_and_delete_security_groups | 10 |
| NeutronSecurityGroup.create_and_delete_security_group_rule | 10 |
| NeutronSecurityGroup.create_and_list_security_group_rules | 10 |
| NeutronSecurityGroup.create_and_show_security_group | 10 |
| NeutronNetworks.set_and_clear_router_gateway | 10 |
| NeutronNetworks.create_and_show_ports | 10 |
| NeutronNetworks.create_and_show_routers | 10 |
| NeutronNetworks.create_and_show_subnets | 10 |
| Quotas.neutron_update | 10 |

Functest rally_jobs_cntt:

Table 3.14: Neutron API benchmarking

| Scenarios | Iterations |
|--|------------|
| NeutronNetworks.create_and_delete_networks | 40 |
| NeutronNetworks.create_and_delete_ports | 40 |
| NeutronNetworks.create_and_delete_routers | 40 |
| NeutronNetworks.create_and_delete_subnets | 40 |
| NeutronNetworks.create_and_list_networks | 100 |
| NeutronNetworks.create_and_list_ports | 8 |
| NeutronNetworks.create_and_list_routers | 40 |
| NeutronNetworks.create_and_list_subnets | 40 |
| NeutronNetworks.create_and_update_networks | 40 |
| NeutronNetworks.create_and_update_ports | 40 |
| NeutronNetworks.create_and_update_routers | 40 |
| NeutronNetworks.create_and_update_subnets | 100 |
| NeutronTrunks.create_and_list_trunks | 4 |
| Quotas.neutron_update | 40 |

Compute - Nova API benchmarking

Functest rally_full_cntt:

Table 3.15: Nova API benchmarking

| Scenarios | Iterations |
|--|------------|
| Authenticate.validate_nova | 10 |
| NovaKeypair.create_and_delete_keypair | 10 |
| NovaKeypair.create_and_list_keypairs | 10 |
| NovaServers.boot_and_bounce_server | 10 |
| NovaServers.boot_and_delete_server | 10 |
| NovaServers.boot_and_list_server | 10 |
| NovaServers.boot_and_rebuild_server | 10 |
| NovaServers.snapshot_server | 10 |
| NovaServers.boot_server_from_volume | 10 |
| NovaServers.boot_server | 10 |
| NovaServers.list_servers | 10 |
| NovaServers.resize_server | 10 |
| NovaServers.boot_and_live_migrate_server | 10 |
| NovaServers.boot_server_attach_created_volume_and_live_migrate | 10 |
| NovaServers.boot_server_from_volume_and_live_migrate | 10 |
| NovaKeypair.boot_and_delete_server_with_keypair | 10 |
| NovaServers.boot_server_from_volume_and_delete | 10 |
| NovaServers.pause_and_unpause_server | 10 |
| NovaServers.boot_and_migrate_server | 10 |
| NovaServers.boot_server_and_list_interfaces | 10 |
| NovaServers.boot_and_get_console_url | 10 |
| NovaServers.boot_server_and_attach_interface | 10 |
| NovaServers.boot_server_attach_volume_and_list_attachments | 10 |
| NovaServers.boot_server_associate_and_dissociate_floating_ip | 10 |
| NovaServers.boot_and_associate_floating_ip | 10 |
| NovaServerGroups.create_and_delete_server_group | 10 |
| NovaServerGroups.create_and_get_server_group | 10 |
| NovaServerGroups.create_and_list_server_groups | 10 |
| Quotas.nova_update | 10 |

Orchestration - Heat API benchmarking

Functest rally_full_cntt:

Table 3.16: Heat API benchmarking

| Scenarios | Iterations |
|---|------------|
| Authenticate.validate_heat | 10 |
| HeatStacks.create_and_delete_stack | 10 |
| HeatStacks.create_and_list_stack | 10 |
| HeatStacks.create_update_delete_stack | 10 |
| HeatStacks.create_check_delete_stack | 10 |
| HeatStacks.create_suspend_resume_delete_stack | 10 |
| HeatStacks.list_stacks_and_resources | 10 |

Dataplane benchmarking

Functest Benchmarking CNTT offers two benchmarking dataplane test cases leveraging on:

- VMTP
- Shaker

VMTP is a small python application that will automatically perform ping connectivity, round trip time measurement (latency) and TCP/UDP throughput measurement on any OpenStack deployment.

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

At the time of writing, no KPIs are defined in Anuket specifications which would have asked for an update of the default SLA proposed in [Functest Benchmarking CNTT](#)

On top of this dataplane benchmarking described in VMTP & Shaker, we need to integrate testing as described in [ETSI GS NFV-TST 009: Specification of Networking Benchmarks and Measurement Methods for NFVI](#). This type of testing is better suited to measure the networking capabilities of a compute node. The [rapid scripts](#) in conjunction with the [PROX](#) tool offers an open source implementation for this type of testing.

VMTP

Here are the [scenarios](#) executed by [Functest vmtp](#): - VM to VM same network fixed IP (intra-node) - VM to VM different network fixed IP (intra-node) - VM to VM different network floating IP (intra-node) - VM to VM same network fixed IP (inter-node) - VM to VM different network fixed IP (inter-node) - VM to VM different network floating IP (inter-node)

Here are all results per scenario:

Table 3.17: All results per scenario

| protocol | pkt_size | results |
|----------|----------|-----------------|
| ICMP | 64 | rtt_avg_ms |
| ICMP | 64 | rtt_max_ms |
| ICMP | 64 | rtt_min_ms |
| ICMP | 64 | rtt_stddev |
| ICMP | 391 | rtt_avg_ms |
| ICMP | 391 | rtt_max_ms |
| ICMP | 391 | rtt_min_ms |
| ICMP | 391 | rtt_stddev |
| ICMP | 1500 | rtt_avg_ms |
| ICMP | 1500 | rtt_max_ms |
| ICMP | 1500 | rtt_min_ms |
| ICMP | 1500 | rtt_stddev |
| UDP | 128 | loss_rate |
| UDP | 128 | throughput_kbps |
| UDP | 1024 | loss_rate |
| UDP | 1024 | throughput_kbps |
| UDP | 8192 | loss_rate |
| UDP | 8192 | throughput_kbps |
| TCP | 65536 | rtt_ms |
| TCP | 65536 | throughput_kbps |

Shaker

Here are the [scenarios](#) executed by Shaker:

- OpenStack L2
- OpenStack L3 East-West
- OpenStack L3 North-South
- OpenStack L3 North-South Performance

Here are all samples:

Table 3.18: All samples

| test | samples |
|----------------|------------------------|
| Bi-directional | ping_icmp (ms) |
| Bi-directional | tcp_download (Mbits/s) |
| Bi-directional | tcp_upload (Mbits/s) |
| Download | ping_icmp (ms) |
| Download | tcp_download (Mbits/s) |
| Upload | ping_icmp (ms) |
| Upload | tcp_upload (Mbits/s) |
| Ping | ping_icmp (ms) |
| Ping | ping_udp (ms) |
| TCP | bandwidth (bit/s) |
| TCP | retransmits |
| UDP | packets (pps) |

PROX

The generator used with the rapid scripts is PROX with a specific generator configuration file. When multiple flows are requested, the generator starts randomizing bits in the source and destination UDP ports. The number of flows to be generated during each run of the test is specified in the test files (e.g. TST009_Throughput.test). Packet size used during the test is also defined in the test file. IMIX is not supported yet, but you could take the average packet size of the IMIX for now. When defining n packet sizes with m different flow sizes, the test will run n x m times and will produce the results for these n x m combinations. All throughput benchmarking is done by a generator sending packets to a reflector. This results in bidirectional traffic which should be identical (src and dest IP and ports swapped) if all traffic goes through. The VMs or containers use only 1 vNIC for incoming and outgoing traffic. Multiple queues can be used. Multiple VMs or containers can be deployed prior to running any tests. This allows to use generator-reflector pairs on the same or different compute nodes, on the same or different NUMA nodes.

Opensource VNF onboarding and testing

Running opensource VNFs is a key technical solution to ensure that the platforms meet Network Functions Virtualization requirements. [Functest VNF](#) offers 5 test cases which automatically onboard and test the following 3 opensource VNFs:

- Clearwater IMS
- VyOS vRouter
- OpenAirInterface vEPC

Here are the full list of orchestrators used for all these deployments:

- Cloudify
- Heat
- Juju

The VNF are covered by upstream tests when possible (see [clearwater-live-test](#)) and by Functest VNF tests in the other cases.

Tenants

LCM

Assurance

Security

Resilience

Bare-metal validations

3.4 Test Cases Traceability to Requirements

RM/RA-1 Requirements

According to *OpenStack-based cloud infrastructure Testing Cookbook* the following test cases must pass as they are for Anuket NFVI Conformance:

Table 3.19: Anuket NFVI Conformance

| container | test case | criteria |
|--|-----------------------|----------|
| opnfv/functest-healthcheck:wallaby | tempest_horizon | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_neutron_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_cinder_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_keystone_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | rally_sanity_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_full_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_scenario_cntt | PASS |
| opnfv/functest-smoke-cntt:wallaby | tempest_slow_cntt | PASS |
| opnfv/functest-benchmarking-cntt:wallaby | rally_full_cntt | PASS |
| opnfv/functest-benchmarking-cntt:wallaby | rally_jobs_cntt | PASS |
| opnfv/functest-benchmarking-cntt:wallaby | vmtp | PASS |
| opnfv/functest-benchmarking-cntt:wallaby | shaker | PASS |
| opnfv/functest-vnf:wallaby | cloudify | PASS |
| opnfv/functest-vnf:wallaby | cloudify_ims | PASS |
| opnfv/functest-vnf:wallaby | heat_ims | PASS |
| opnfv/functest-vnf:wallaby | vyos_vrouter | PASS |
| opnfv/functest-vnf:wallaby | juju_epc | PASS |

TC Mapping to Requirements

| test case | requirements |
|-----------------------|--|
| tempest_horizon | Horizon testing |
| tempest_neutron_cntt | Neutron API testing |
| tempest_cinder_cntt | Cinder API testing |
| tempest_keystone_cntt | Keystone API testing |
| rally_sanity_cntt | Keystone, Glance, Cinder, Swift, Neutron, Nova and Heat API testing |
| tempest_full_cntt | Keystone, Glance, Cinder, Swift, Neutron and Nova API testing |
| tempest_scenario_cntt | Keystone, Glance, Cinder, Swift, Neutron and Nova API testing |
| tempest_slow_cntt | Keystone, Glance, Cinder, Swift, Neutron and Nova API testing |
| rally_full_cntt | Keystone, Glance, Cinder, Swift, Neutron, Nova and Heat API benchmarking |
| rally_jobs_cntt | Neutron API benchmarking |
| vmtpt | Dataplane benchmarking |
| shaker | Dataplane benchmarking |
| cloudify | opensource VNF onboarding and testing |
| cloudify_ims | opensource VNF onboarding and testing |
| heat_ims | opensource VNF onboarding and testing |
| vyos_vrouter | opensource VNF onboarding and testing |
| juju_epc | opensource VNF onboarding and testing |

4 OpenStack-based cloud infrastructure Testing Cookbook

4.1 Introduction

Define the purpose of the chapter which is to:

- Identify Framework Needs, Goals, and Dependencies
- Define Opensource Integration (OVP, Functest, CVC, others)
- Provide Automation Toolchain (list, topology, flow)

4.2 Relevant Community Projects and Initiatives

Functest

Functest was initially created to verify OPNFV Installers and Scenarios and then to publish fair, trustable and public results regarding the status of the different opensource technologies, especially for Neutron backends (e.g. Neutron agents, OpenDaylight, OVN, etc.). It has been continuously updated to offer the best testing coverage for any kind of OpenStack and Kubernetes deployments including production environments. It also ensures that the platforms meet Network Functions Virtualization requirements by running and testing VNFs amongst all tests available.

Functest is driven by a true verification of the platform under test as opposed to the interoperability programs such as RefStack or OPNFV Verification Program which select a small subset of Functional tests passing in many different opensource software combinations:

- tests are skipped if an optional support is missing (e.g. Barbican or networking features such as BGPVPN interconnection or Service Function Chaining)
- tests are parameterized (e.g. shared vs non-shared live migration)
- blacklist mechanisms are available if needed

It should be noted that the [RefStack lists](#) are included as they are in Functest in the next 3 dedicated testcases:

- [refstack_compute](#) (OpenStack Powered Compute)
- [refstack_object](#) (OpenStack Powered Object Storage)
- [refstack_platform](#) (OpenStack Powered Platform)

Functest also integrates [Kubernetes End-to-end tests](#) and allows verifying Kubernetes Conformance (see [k8s-conformance](#)).

Dovetail (OVP) mostly leverages on Functest but only runs a small part of Functest (~15% of all functional tests, no benchmarking tests, no VNF deployment and testing). It's worth mentioning that Functest is patched to [disable API verification](#) which has differed from OpenStack rules for years.

Then Functest conforms with the upstream rules (versions, code quality, etc.) and especially their [gates](#) (a.k.a. the automatic verification prior to any code review) to preserve the quality between code and deployment. In that case, Functest can be considered as a smooth and lightweight integration of tests developed upstream (and the Functest team directly contributes in these projects: [Rally](#), [Tempest](#), etc.). It's worth mentioning that, as opposed to the OpenStack Gates leveraging on [DevStack](#), it can check the same already deployed SUT over and over even from a [Raspberry PI](#). Here the testcases can be executed in parallel vs the same deployment instead of being executed vs different pools of virtual machines.

Here are the functional tests (>2000) running in OpenStack gates integrated in Functest Smoke (see [Functest daily jobs](#) for more details):

Table 4.1: Functional tests

| Testcases | Gates |
|-------------------|--------------------|
| tempest_neutron | Neutron |
| tempest_cinder | Cinder |
| tempest_keystone | Keystone |
| rally_sanity | General |
| refstack_defcore | General |
| tempest_full | General |
| tempest_slow | General |
| tempest_scenario | General |
| patrole | Patrole |
| tempest_barbican | Barbican |
| networking-bgpvpn | Networking BGP VPN |
| networking-sfc | Networking SFC |

To complete functional testing, Functest also integrates a few [performance tools](#) (2-3 hours) as proposed by OpenStack:

Table 4.2: Performance tools

| Testcases | Benchmarking |
|------------|-----------------------------|
| rally_full | Control Plane (API) testing |
| rally_jobs | Control Plane (API) testing |
| vmtp | Data Plane testing |
| shaker | Data Plane testing |

And VNFs automatically deployed and tested :

Table 4.3: VNFs

| Testcases | Benchmarking |
|--------------|--------------------------------------|
| cloudify | Cloudify deployment |
| cloudify_ims | Clearwater IMS deployed via Cloudify |
| heat_ims | Clearwater IMS deployed via Heat |
| vyos_vrouter | VyOS deployed via Cloudify |
| juju_epc | OAI deployed via Juju |

Functest should be considered as a whole as it meets multiple objectives about the reference implementation:

- verify all APIs (services, advances, features, etc.) exposed by the reference implementation
- compare the reference implementation and local deployments from a functional standpoint and from OpenStack control plane and dataplane capabilities

It's worth mentioning that Functest already takes into account the first Anuket [profiles](#). Anuket should simply add the next Functest inputs according the reference implementation:

- [Functest inputs](#)
- [tempest specific configuration](#)

Additional links:

- [Homepage](#)
- [Run Alpine Functest containers \(Wallaby\)](#)
- [Deploy your own Functest CI/CD toolchains](#)
- [Functest gates](#)

Yardstick

Bottlenecks

Test Tools

1. Shaker: <https://pyshaker.readthedocs.io/en/latest/> (The distributed data-plane testing tool built for OpenStack)
2. Sonobuoy: <https://sonobuoy.io/> It is a diagnostic tool that makes it easier to understand the state of a Kubernetes cluster by running a set of plugins (including Kubernetes conformance tests) in an accessible and non-destructive manner.

Scenario Descriptor File (SDF)

As defined by Anuket, Scenario Descriptor File's (SDF) will be utilized to relay information from the Scenario Designer (or Test Manager), to Release Managers, CI Pipeline Owners, and Installer Agents, to define test scenario content, and specifications.

SDF's will contain, but not limited to, the following Metadata, Components, Deployment Options, Deployment Tools, and Hardware prerequisites:

- **Metadata**
 - Name
 - History

- Purpose
- Owner
- **Components**
 - e.g. SDN controllers
 - Versions
 - Optional features, e.g. NFV features
- **Deployment Options**
 - Hardware types
 - Virtual deploy
 - HA, NUMA
- **Deployment Tools**
 - Supporting installers.
 - Valid options per installer.

4.3 OpenStack Testing Cookbook

At the time of writing, the CI description file is hosted in Functest and only runs the containers listed in RM/RA-1 Requirements. It will be completed by the next Anuket mandatory test cases and then a new CI description file will be proposed in CIRV tree.

Please note the next two points depending on the GNU/Linux distributions and the network settings:

- SELinux: you may have to add `--system-site-packages` when creating the virtualenv (“Aborting, target uses selinux but python bindings (libselinux-python) aren’t installed!”)
- Proxy: you may set your proxy in env for Ansible and in systemd for Docker <https://docs.docker.com/config/daemon/systemd/#httphttps-proxy>

To deploy your own CI toolchain running Anuket Compliance:

```
virtualenv functest --system-site-packages
. functest/bin/activate
pip install ansible
ansible-galaxy install collivier.xtesting
ansible-galaxy collection install ansible.posix community.general community.grafana
↪kubernetes.core community.docker community.postgresql
git clone https://gerrit.opnfv.org/gerrit/functest functest-src
(cd functest-src && git checkout -b stable/wallaby origin/stable/wallaby)
ansible-playbook functest-src/ansible/site.cntt.yml
```

OpenStack API testing configuration

Here is the default Functest tree as proposed in [Functest Wallaby](#):

- /home/opnfv/functest/openstack.creds
- /home/opnfv/functest/images

Download the images and fill /home/opnfv/functest/openstack.creds as proposed in [Functest Wallaby](#)

You may have to modify a few Functest env vars according to the SUT (see env in [Functest Wallaby](#)). Be free to modify functest-src/ansible/host_vars/127.0.0.1 at your convenience and then to reconfigure the toolchain:

```
ansible-playbook functest-src/ansible/site.cntt.yml
```

Run Anuket OpenStack Testing

Open <http://127.0.0.1:8080/job/functest-wallaby-daily/> in a web browser, login as admin/admin and click on “Build with Parameters” (keep the default build_tag value).

If the System under test (SUT) is Anuket compliant, a link to the full archive containing all test results and artifacts will be printed in functest-wallaby-zip’s console. Be free to download it and then to send it to any reviewer committee.

To clean your working dir:

```
deactivate
rm -rf functest-src functest
```

5 VNF Testing Framework Requirements

5.1 Introduction

This chapter covers comprehensive VNF Conformance requirements for enabling required process and steps to provide VNF badging based on define scope of compliance and validation. This includes end to end test framework requirements, badging entry and exit criteria, profiles to reference, different stake holders and Conformance Methodologies by using certified NFVi under NFVi badging program.

5.2 Conformance Methodology

It defines the end-end framework and process required for certifying the given VNF.

End-End framework:

Here, the steps 1-4 are NFVI related steps are covered in detail in [NFVI Conformance Requirements](#).

Step-5. Interoperability validations for VNF functional testing defined.

Step-6. Interoperability validations for VNF performance testing defined (IOPS, connection, threading, resource consumption).

Step-7. Sending requirements to the VNF requirements projects in terms of t-shirt sizes, config settings, required for VNF/orchestration validation.

Conformance flow:

The entry and exit criteria defined in below section are pre-requisities for this flow.

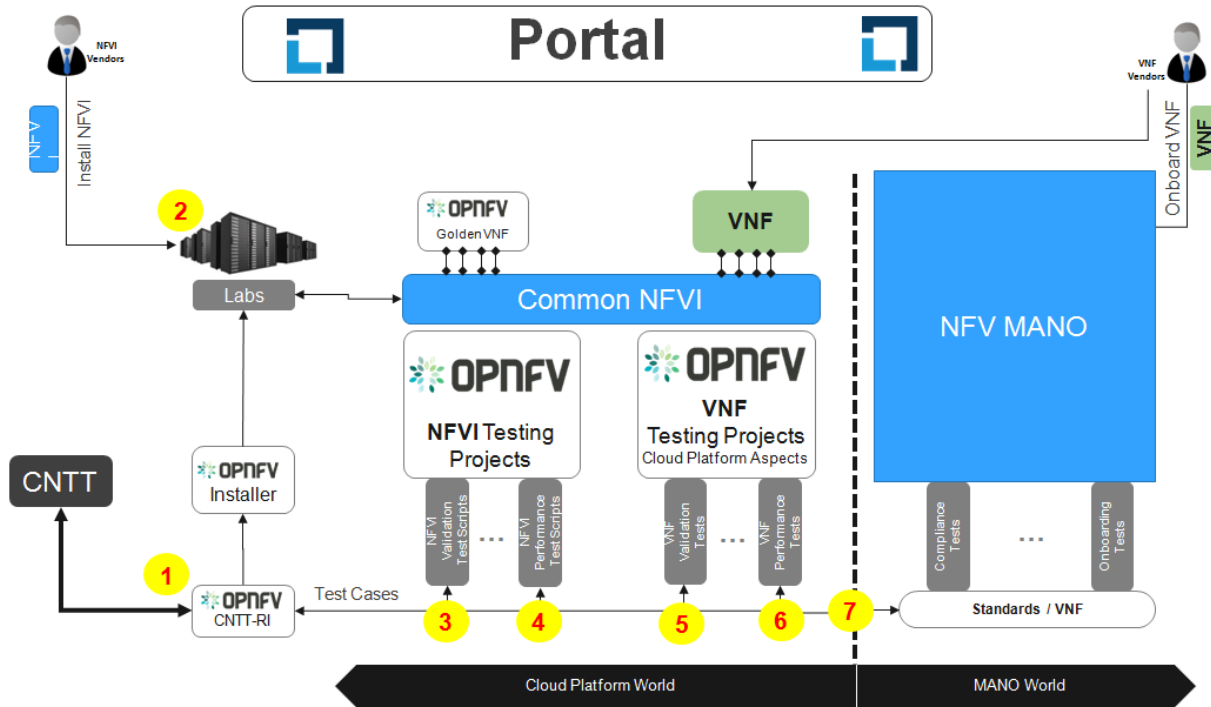


Figure 5.1: End-End framework

1. VNF Vendors submit the VNF into OVP Lab for Conformance (Fulfilling the entry criteria is pre-requisites for this step.)
2. As part of OVP lab, already required test cases, test tools, eco-system like MANO and appropriate certified NFVi to be setup as defined part of entry criteria. This lab could either OVP 3rd party lab or VNF vendors.
3. Once testing is completed done, test results will be submitted to the OVP portal for community review along with additional information such as product name, documentation links, primary company contact information, etc.
4. LFN CVC community team reviewers will review the results submitted and will approve or reject it based the details provided.
5. If reviewer rejected it, then step 2 and 3 will be ran again to address the review comments. Otherwise once reviewer approved it, corresponding VNF will be published into OVP VNF Portal with OVP badge.
6. LFN staff will provide the certificate badge graphics and graphical usage guidelines.
The OVP portal will reflect LFN's disposition and assignment of the certified VNF badge.

Now VNF is ready and Telco Operators can start consume it.



Figure 5.2: OVP 3rd party lab

Profiles Reference

The NFV Infrastructure (NFVI) is the totality of all hardware and software components which build up the environment in which VNFs are deployed, managed and executed. It is, therefore, inevitable that different VNFs would require different capabilities and have different expectations from it. So one of the main targets of Anuket is to define an agnostic NFVI and removes any dependencies between VNFs and deployed Infrastructure (NFVI) and offer NFVI to VNFs in an abstracted way with defined capabilities and metrics. This would help operators to host their Telco Workload (VNF) with different traffic types, behaviour and from any vendor on a unified consistent Infrastructure. so as part of VNF Conformance, its important to certify the VNF based on profiled defined in :doc:ref_model/chapters/chapter02`.

In [Workload Requirements & Analysis](#), following NFVi profiles are proposed as reference:

- **Basic:** for VNF that can tolerate resource over-subscription and variable latency.
- **Network Intensive:** for VNF that require predictable computing performance, high network throughput and low network latency.

Protoype VNFs

A portion of the NFVI badging methodology includes Empirical Validation with Reference Golden VNFs (aka CVC Validation) which will ensure the NFVI runs with a set of VNF Families, or Classes, to mimic production-like VNF connectivity. These tests are to 1) ensure interoperability checks pass, and 2) there is an established baseline of VNF behaviors and characters before vendor supplied VNFs are tested and certified. In other words, empirical validations will confirm performance and stability between Platform and VNF, such as validating packet loss is within acceptable tolerances.

5.3 Badging Requirements

Defined. *Badging* refers to the granting of a Conformance badge by the OVP to Suppliers/Testers of Anuket NFVI+VNF upon demonstration the testing performed confirms:

- NFVI adheres to Anuket RA/RM requirements.
- Anuket certified VNFs functionally perform as expected (i.e. test cases pass) on NFVI with acceptable levels of stability and performance.

Following table shows the badging requirements with scope of mandatory (must) or optional.

Table 5.1: The badging requirements

| Requirement id | Scope | Details |
|----------------|----------|--|
| CVreq.VNF.001 | must | Receive NFVi badge in lab setup per RI-1 standards, performing h/w validations, performing s/w manifest validations, running nfvi compliance, validation, and performance checks |
| CVreq.VNF.002 | must | met all entry and exit criteria |
| CVreq.VNF.003 | must | run interoperability validations, including instantiation, communication / health, and removal |
| CVreq.VNF.004 | shall | utilize automation frameworks to run all required tests. Conformance process would improve, if test framework satisfy the required defined in this chapter under <i>VNF Test Conformance platform requirements section</i> |
| CVreq.VNF.005 | must | pass all required tests |
| CVreq.VNF.006 | must | prepare release notes, with issues known, their severity and magnitude, mitigation plan |
| CVreq.VNF.007 | must | publish results in defined normalized output |
| CVreq.VNF.008 | must | respond /closed badging inquiries |
| CVreq.VNF.010 | optional | for badging VNF supplier can choose to run their own test harnesses/suites to validate VNF functional and performance behaviors and performance |

Badging Scope

The VNF badging includes:

1. NFVi Verifications (Compliance): Manifest Verifications will ensure the NFVI is compliant, and delivered for testing, with hardware and software profile specifications defined by the Ref Model and Ref Architecture.
2. Empirical Validation with Reference VNF (Validation): Empirical Validation with Reference Golden VNFs will ensure the NFVI runs with a set of VNF Families, or Classes, to mimic production-like VNFs to baseline infrastructure conformance.
3. Candidate VNF Validation (Validation & Performance): Candidate VNF Validation will ensure complete interoperability of VNF behavior on the NFVI leveraging VVP/VNFSDK test suites to ensure VNF can be spun up, modified, or removed, on the target NFVI (aka Interoperability).

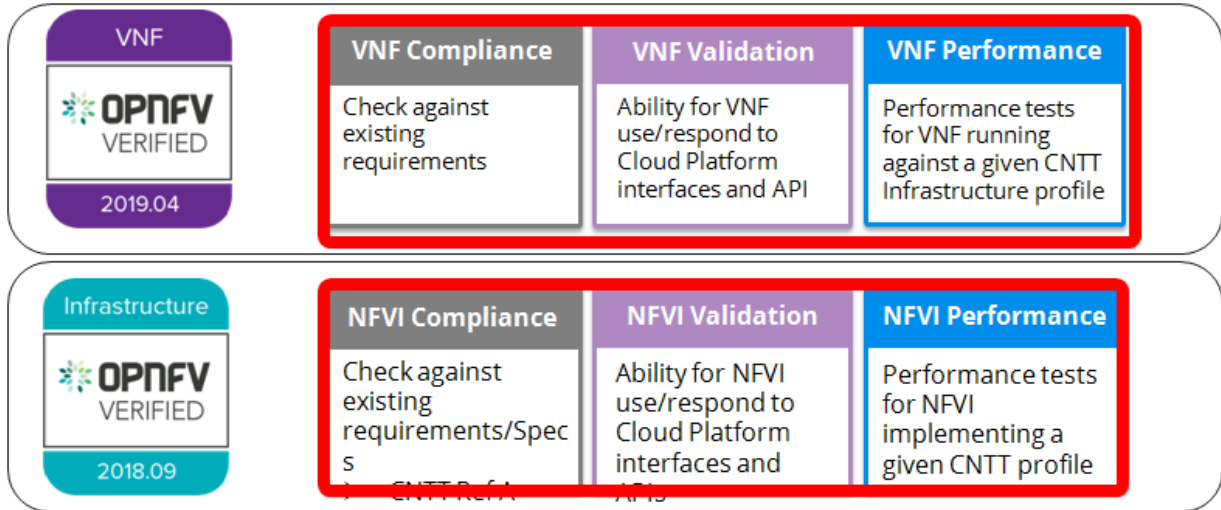


Figure 5.3: Candidate VNF Validation

Entry criteria

Before entering into the VNF badging process, VNF needs to satisfy the following requirements as entry criteria:

- *Environment Requirements* : Published details providing evidence that a RAX compliant lab has been implemented, meeting requirements set forth in respective RM and RAX documentation for features, options, and capabilities needed for VNF test validations. Expected information includes:
 - Lab Flavor
 - Component software rev levels
 - Confirmation of compatibility with external systems
 - Tenant needs identified
 - All connectivity, network, image, VMs, delivered with successful pairwise tests
 - Lab instrumented for proper monitoring
- *VNF artifact* : VNF cloud (native) image, VNF configurations and guidelines, automation scripts, etc
- *NFVi profiles*: List of supporting OVP Certified Anuket compliant NFVi
- Completed Security review report
- Vendor specific test cases and its deployment and usage guidelines

Exit criteria

VNF Conformance testing should be completed with following exit criteria:

- All required test cases should be passed
- No outstanding high severity issues and other known issues to be documented
- Release notes
- Provided with required installation guide, configuration guide, etc.

- Test results collated, centralized, and normalized, with a final report generated showing status of the test scenario/case (e.g. Pass, Fail, Skip, Measurement Success/Fail, etc), along with traceability to a functional, or non-functional, requirement

5.4 VNF Test Conformance platform Requirements

Test platform requirements are provided to address test case design, distribution, execution and result reporting along with required artifacts and environments in place and are defined based on below scope.



Figure 5.4: Test platform requirements

Standards/Profiles

- ETSI (TOSCA)
- GSMA
- ONAP VNFREQS (HOT)

Test cases

Refer *chapter RC-06* for more details on test case requirements defined for VNF under Anuket. Platform should support to managed and execute these test cases.

Note: For Conformance, only compliance and verification test cases will be considered, but in future, it could be extent to validation and Performance related testing.

Compliance

Perform compliance check based on

- TOSCA using ETSI SOL004 & SOL001
- OpenStack HOT using ONAP VNFREQS
- GSMA profile as defined in *chapter RM-04*.

Verification

Perform on-boarding/ verification life cycle operation (from instantiation, configuration, update, termination) using MANO supporting Anuket compliant NFVI.

Validation

Perform various VNF type specific functionality operations on Anuket RA & RM compliant NFVI

Performance

Perform various performance related testing and facilitate for benchmarking the VNF performance on different profile and scenarios.

Eco-system MANO/NFVI

Platform would support to execute various test cases on Anuket RA & RM compliant NFVi along with required MANO system supporting these NFVi.

VNF

Suppliers of VNFs/CNFs seeking to receive VNF Conformance badges must first ensure their testing is performed against a compliant RM/RA architecture supporting all capabilities, features, and services defined by the respective *RM/RA requirements*. More specifically, the VNF Supplier must ensure their implementation of the RM/RA receives the NFVI Conformance badge prior to starting VNF testing. Finally, to receive VNF Conformance, the test platform will need to support TOSCA and HOT based VNF distros.

In addition, Platform should be able to perform the required test case management and executions and produce the result the CVC OVP portal for Conformance process along with required testing foot print details. So overall scoped example architecture could be as below:



Figure 5.5: VNF Test Certification Platform

Test Case Model

As there are more number of VNF at different levels of networking such as access, transport and core level as well as OSI level L0-L7. Every network function provides set of pre-defined features and functionalities. So its important to model test cases for every functionality to identify it uniquely and use it as part of test flow design.

As part of modeling its very important to capture the following details

- Test case Name
- Test case description
- Virtual Network function Name
- Network function Feature/functionality name
- Test case input parameters
- Test case result attributes
- Test case version

while implementing the test cases, this model would act as specification and as it captures the input and output, it would help while designing the test flow which will help to execute set of test cases in pre-defined flow.

Test case management

- **Test case** : On-board/discover, update, disable/enable, delete
- **Test suite** : On-board/discover, update, disable/enable, delete
- **Test flow** : design/discover, update, disable/enable, delete

Test Execution management

- **Run-time:** One of the common nature of the test environment is heterogeneous and multiple vendors and open communities would provide various test tool and environment to support execution of test cases developed under different run-times (JVM, Python, Shell, Container, Cloud VM, etc)
- **RPC:** In order to enable the scaling/remote execution, it should be enabled with required RPC support.

When VNF test platform execute the test cases, it captures the footprints of test case execution along with results, which are made available to user and integrated system for consuming.

Test Result management

Categorization. Test suites will be categorized as Functional/Platform or Performance based.

Results. Test results reporting will be communicated as a boolean (pass/fail), or Measurements Only.

- **Functional Pass/Fail** signals the assertions set in a test script verify the Functional Requirements (FR) has met its stated objective as delivered by the developer. This will consist of both positive validation of expected behavior, as well as negative based testing when to confirm error handling is working as expected.
- **Performance-based Pass/Fail** determination will be made by comparing Non-Functional (NFR) KPIs (obtained after testing) with the Golden KPIs. Some of the examples of performance KPIs include, but not limited to: TCP bandwidth, UDP throughput, Memory latency, Jitter, IOPS etc.
- **Measurement Results.** Baseline Measurements will be performed when there are no benchmark standards to compare results, or established FRs/NFRs for which to gauge application / platform behavior in an integrated environment, or under load conditions. In these cases, test results will be executed to measure the application, platform, then prepare FRs/NFRs for subsequent enhancements and test runs.

Formats. As part of execution management, system produces the result in JSON format which can be represented in various form like YAML, CSV, Table, etc.

Search & Reporting. Search would help to query the test results based on various fact such as test case, VNF, date of execution, environment, etc. and produce the report in various format like pie-chart, success rates, etc

Collation | Portal. The following criteria will be applied to the collation and presentation of test-runs seeking Conformance:

- RA number and name (e.g. RA-1 OpenStack)
- Version of software tested (e.g. OpenStack Ocata)
- Normalized results will be collated across all test runs (i.e. centralized database)
- Clear time stamps of test runs will be provided.
- Identification of test engineer / executor.
- Traceability to requirements.
- Summarized conclusion if conditions warrant test Conformance (see Badging Section).
- Portal contains links to Conformance badge(s) received.

Test Artifact management

As part of testing various binaries, configurations, images, scripts ,etc would be used during test cases building or execution and Version artifact supports such as VNF CSAR.

Test Scenario management

Allow to create repeatable scenario includes test cases, artifacts and profiles.

It helps to create dynamic testing scenario development and testing from the existing test cases and flows along with required artifacts and profiles. It allows to run repeated testing with one or different profiles.

Test Profile management

For every test case execution needs to be configured with required environments and predefined test input parameter values. This is provided by means of profile

Profile should be having option to include other profiles to manage the hierarchy of them.

As part of profile, testing environment URL, credentials and related security keys are captured and while running the test cases, user would be able to inputs the required profile in place of actual inputs and artifacts.

Also helps in Managing System under test configuration and multiple MANO / NFVI and related eco system management elements.

Tenant & User management

Testing involves design, distribution by different user roles and executed across multiple tenant's environments.

Conformance management & integration

Platform should have integration with OVP Conformance portal for submitting results with OVP defined format.

It should enable repository of certified VNFs which can be used for testing validation and performance.

User & System interfaces

User interface:

- CLI
- Web portal

Programming interface:

- REST API
- gRPC

Deliverables

Platform should be able to get deployed in both container and cloud environments. so following model deliverables would enable it:

- Docker image based installation
- Standalone installation scripts and zip artifact

5.5 VNF Test Cases Requirements

Rationale

Network functions virtualization (NFV) and softwaredefined networking (SDN) offer service providers increased service agility, OpEx improvements, and back-office automation. Disaggregation, the approach of decoupling the various layers of the stack, from hardware, to NFVI/VIM software, to dataplane acceleration, SDN controllers, MANO components, and VNFs, enables multi-vendor deployments with best-of-breed options at each layer.

The Anuket specifications define the required architecture and model for NFVI which will help to decouple the various commercial product layers and it is important to define and certify the VNF and NFVI. Therefore, in addition to verify general NFVI capabilities based on Anuket RM/RA/RI, it is also necessary to verify that VNFs can provide virtualization functions normally based on the Anuket-compliant NFVI. So the VNF testing should at least include: Compliance, verification, validation, Performance. With the improvement of specifications, the types of tests may continue to add in the future.

In this chapter, the scope and requirements of VNF test cases are defined as reference for VNF Conformance, which helps to perform the various compliance and verification (C&V) testing and submit results to LFN OVP Conformance portal.

Assumptions

Here lists the assumptions for VNF Conformance: - NFVI is ready and it should be an Anuket-compliant NFVI - VNF template is ready to deploy and certificate - VNF Test environment is ready, the test environment contains test functions and entities(NFVI, MANO, VNF Test Platform, VNF Test Tools) to enable controlling the test execution and collecting the test measurements. - VNF Test Platform has been integrated with CICD chain - VNF test result can be generated with OVP defined format

Developer Deliverables

This section define the developer Deliverables (artifacts),the following list the expectations and deliverables we expect from developers in order to achieve the VNF Conformance: - VNF test cases model/scripts/programs - VNF test cases configuration/profile - VNF test tools

Requirement Type

VNF test cases are used to verify whether the virtualization network functions can be deployed on the Anuket-compliant NFVI and provide normal functions and meet performance, security and other requirements.

By running these VNF test cases and analysis the test results, can be used for VNF compliance, verification, validation and performance Conformance and help on Anuket-compliant NFVI validation and performance Conformance.

All the VNF test cases should be supported and run by VNF E2E Conformance and verification Framework and generate outputs, logs to identify whether the test passed or failed.

Anuket defines the following four category testing which should be consistent with the VNF test category defined by OVP.

Table 5.2: VNF Test Case categories

| VNF Test Case Category | Requirement Number | Type (Measurement/Boolean) | Definition/Description |
|------------------------|--------------------|----------------------------|---|
| Compliance | VNF.COMPreq.001 | Boolean (i.e. Pass/Fail) | Test case “must” perform a platform check against the OpenStack requirements and VNF package structure and syntax requirements |
| Verification | VNF.VERIFYreq.001 | Boolean (i.e. Pass/Fail) | Test case “must” perform on-boarding/ verification life cycle operation validation |
| Validation | VNF.VALIDreq.001 | Boolean (i.e. Pass/Fail) | Test case “must” perform API validation tests to verify operability |
| Performance | VNF.PERFreq.001 | Measurement | Test case “must” execute various performance related testing and facilitate for benchmarking the VNF performance on different profile and scenarios |

Note: The four category testing can be gradually supported and in the future, will also cover security and other test category.

Interaction Type

- Describe the types of Interactions: Extended Topology, Complex (Akraio), Functional, HA, Fault, Interoperability

Performance Profiles

Performance profiles are not in the scope of current release, and in future it would need to align with *chapter RM-4* defined measurements.

VNF Class/Family and Characteristics

- Describe and provide a Table of VNF Class/Family & Characteristics of Each

The communication network usually consists of three parts: access network, transmission network/bearer network and core network. Following are some examples of network elements for each type of network

Table 5.3: Communication network

| Network Type | Network Elements |
|------------------------------------|--|
| Access Network | Including mobile access network, wireless access network, wired access network |
| Transport network & Bearer network | Including Trunk Optical Transport Network, Metro transport network, IP backbone network, etc. |
| Core Network | Circuit domain, including MSC/VLR, GMSC, MGW, NPMSC, HLR/AUC, NPHLR, HSS, etc, Packet domain devices, including MME, SAE GW, EPC CG, EPC DNS, PCC, etc; Core network equipment for IoT private network, including PGW/GGSN, PCRF, HSS/HLR, etc, 5G core network element, including AMF, SMF, UPF, UDM/UDR/AUS F, PCF, NSSF, NRF, SMSF, etc |

In addition to the above network elements, there are some other data communication network element, including FW, DNS, Router, GW, etc

According to the current level of the entire network virtualization, the core network already has many VNFs, and also includes some datacom-type(data communication) VNFs.

We can also classify VNFs based on the level of VNF operation:

- VNFs that operate at Layer 2 or Layer 3 are primarily involved in switching or routing packets at these layers. Examples include vRouter, vBNG, vCE device, or vSwitch.
- VNFs that operate at Layer 4 through Layer 7 and are involved in forwarding, dropping, filtering or redirecting packets at Layer 4 through 7. Examples include vFirewall, vADC, vIDS/vIPS, or vWAN Accelerator.
- VNFs that are involved in the dataplane forwarding through the evolved packet core.

Measurement

As part of Conformance testing, following measurement would help for evaluating the badging:

- VNF type defined as part of *Chapter RM-02* and its profile used for testing.
- Test cases and their test results including the test case outputs, logs
- VNF model type (TOSCA/HOT)
- Test case pass/failed
- Different NFVi profiles used and LAB reference identifier
- Test owner (point of contact)

VNF Test Cases

Compliance test cases

Currently, there VNFs can be packaged as HEAT templates or in a CSAR file using TOSCA and OVP has supported the VNF compliance test cases (compliance check based on TOSCA using ETSI SOL004 & SOL001; OpenStack HOT using ONAP VNFREQS; GSMA profile), all the OVP supported test case can be found in the following two link:

Table 5.4: OVP supported test case

| Test Cases | Link |
|------------------|---|
| Heat Test Cases | https://docs.onap.org/projects/onap-vnfrqts-testcases/en/istanbul/Appendix.html#list-of-requirements-with-associated-tests |
| Tosca Test Cases | https://docs.onap.org/projects/onap-vnfsdk-model/en/istanbul/files/csar-validation.html |

Above compliance test cases definition can be found <https://github.com/onap/vnfsdk-validation/tree/master/csarvalidation/src/main/resources/open-cli-schema>

In order to adapt Anuket specification, more compliance test case will be added here.

Verification test cases

In general, the VNF Manager, in collaboration with the NFV Orchestrator, the VIM and the EM, is responsible for managing a VNF's lifecycle. The lifecycle phases are listed below:

- VNF on-boarding, it refers to VNF package onboarding to service/resource Orchestrator
- VNF instantiation, once the VNF is instantiated, its associated VNFCs have been successfully instantiated and have been allocated necessary NFVI resources-
- VNF scaling/updating, it means the VNF can scale or update by allocating more or less NFVI resources
- VNF termination, any NFVI resources consumed by the VNF can be cleaned up and released.

OVP has also supported the lifecycle test case: <https://wiki.lfnetworking.org/display/LN/VNF+Validation+Minimum+Viable+Product?src=...>

Validation Test cases

From the current situation of operators, there are usually corresponding functional test specifications for each types of VNFs. Therefore, different types of VNFs have different functional test cases. Normally, functional tests for VNFs require the cooperation of surrounding VNFs. Or use the instruments to simulate the functions of surrounding VNFs for testing. Therefore, different test cases need to be defined according to different types of VNFs

Performance Test cases

This is the same as what described in validation test cases, the performance test cases need to be defined according to different types of VNFs. Combined with the classification of VNF, according to the protocol level that VNF operates, it can include:

- VNF data plane benchmarking, like forwarding Performance Benchmarking, Long duration traffic testing, low misrouting and so on.
- VNF control plane benchmarking, like throughput

- VNF user plane benchmarking, like Packet Loss, Latency, Packet Delay

ETSI spec has also defined the testing method https://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/001/01.01.01_60/gs_nfv-tst001v010101p.pdf

6 VNF Test Cases and Traceability to Requirements

6.1 Introduction

- Provide an overview of the purpose for the VNF TC Traceability to RM Requirements chapter
- Start with defining requirements
- Finish with tracing test cases to requirements

6.2 RM/RA1 Requirements

- Define RM/RA-1 Openstack requirements

6.3 Test Case Traceability

- Define and provide a table mapping Test Cases to Requirements

7 VNF Testing Cookbook

7.1 Introduction

Define the purpose of the chapter which is to:

- Identify Framework Needs, Goals, and Dependencies
- Define Opensource Integration (OVP, Functest, CVC, others)
- Provide Automation Toolchain (list, topology, flow)

8 Gap analysis and Development

8.1 Introduction

- Describe the purpose of this chapter, which includes, but not limited to:
- Test Case Gaps (analysis)
- Automation Gaps
- OpenStack release Comparisons

8.2 Openstack Release Comparisons

- Provide details, preferably in table format, comparing OpenStack releases based on Wallaby baseline for RI-1

8.3 Test Case Gaps

Anuket has developed many test cases in the different [test projects](#) which can quickly improve RC. As listed in *NFVI Conformance Requirements*, porting all the existing testcases to Xtesting will unify the test case execution and simplify the test integration as required by RC. Here are all the related issues:

- [port VinePerf to Xtesting](#)
- [port NFVbench testcases to Xtesting](#)

Here are the possible new test cases which could be integrated in the existing Anuket projects to improve RC:

Table 8.1: Possible new test cases

| Issues | Requirements |
|---|-------------------|
| integrate KloudBuster in Functest | disk benchmarking |
| add tempest-stress in Functest | stress testing |

8.4 Framework Gaps

As proposed in [port VTP test cases to Xtesting](#), VTP selected in *VNF Testing Framework Requirements* requires small adaptations to fully fulfill the current *NFVI Conformance Requirements*. It seems trivial changes as VTP proposed a REST API but will ensure that both NFVI and VNF testing can be executed in the same CI toolchain very easily.