5G European Validation platform for Extensive trials

Deliverable D3.6
Interworking test suites

Project Details

Call	H2020-ICT-17-2018
Type of Action	RIA
Project start date	01/07/2018
Duration	36 months
GA No	815074

Deliverable Details

Deliverable WP:	WP3
Deliverable Task:	Task T3.4
Deliverable Identifier:	5GEVE_D3.6
Deliverable Title:	Interworking test suites
Editor(s):	Ramón Pérez (TELC)
Author(s):	Ramón Pérez (TELC); Matteo Pergolesi, Gianluca Reali (CNIT); Marc Molla (ERI-ES); Ignacio Benito (NOK-ES); Gopalasingham Aravinthan, Laurent Roullet (NOK-FR); Giacomo Bernini (NXW); Grzegorz Panek (ORA-PL); Juan Rodríguez, Lourdes Luque, Luis Miguel Contreras (TID), Evangelos Kosmatos, Christos Ntogkas (WINGS).
Reviewer(s):	Giancarlo Sacco, Mara Piccinino (ERI-IT); Andreas Tzoulis (NOK-GR), Kostas Trichias (WINGS)
Contractual Date of Delivery:	31/10/2019
Submission Date:	01/11/2019
Dissemination Level:	PU
Status:	Final
Version:	1.0
File Name:	5GEVE_D3.6_Final

Deliverable History

Version	Date	Modification	Modified by
ToC_v0.1	*11/07/2019*	*ToC proposal.*	*Ramón Pérez*
ToC_v0.2	*25/07/2019*	*Modifications to the first ToC proposal based on the feedback received from T3.4 partners.*	*Ramón Pérez*
ToC_v0.3	*06/09/2019*	*Review of the ToC after Pisa F2F Meeting. Initial assignment of partners to some sections.*	*Ramón Pérez*
ToC_v0.4	*10/09/2019*	*Final ToC proposal.*	*Ramón Pérez*
v0.1	*16/09/2019*	*Preliminary contributions to the deliverable from TELC.*	*Ramón Pérez, based on contributions by TELC.*
v0.2	*20/09/2019*	*First integrated version of the document with the integration of preliminary contributions from CNIT, NOK-FR, WINGS and ORA-PL.*	*Ramón Pérez, based on contributions by CNIT, NOK-FR, WINGS and ORA-PL.*
v0.21	*24/09/2019*	*Integration of preliminary contributions from NXW.*	*Ramón Pérez, based on contributions by NXW.*
v0.22	*24/09/2019*	*Included feedback received during the D3.3-D3.6 synchronization telco done today.*	*Ramón Pérez*
v0.23	*08/10/2019*	*Integration of contributions from CNIT, NOK-ES, NOK-FR, NXW, TID and WINGS.* *Feedback provided for all partners for revision.*	*Ramón Pérez, based on contributions by CNIT, NOK-ES, NOK-FR, NXW, TID and WINGS.*
v0.24	*14/10/2019*	*Integration of contributions from ERI-ES, NOK-ES, NXW, ORA-PL, TID and WINGS.* *Feedback provided for all partners for revision.*	*Ramón Pérez, based on contributions by ERI-ES, NOK-ES, NXW, ORA-PL, TID and WINGS.*
v0.9	*16/10/2019*	*Integration of contributions from TELC, CNIT, NXW and WINGS.* *Feedback provided for all partners for revision.* *First stable version for internal review.*	*Ramón Pérez, based on contributions by TELC, CNIT, NXW and WINGS.*
v0.91	*22/10/2019*	*Updating pending action points for the final version of the deliverable. Included feedback from internal review (NOK-GR).*	*Ramón Pérez, Andreas Tzoulis.*
v0.92	*29/10/2019*	*Included feedback from internal review (ERI-IT). Integration of final contributions from all partners. Document ready for quality checking.*	*Ramón Pérez (based on contributions by all partners), Giancarlo Sacco, Mara Piccinino.*
V1.0	*31/10/2019*	*QA review and final editing*	*Kostas Trichias*

Table of Contents

List of Acronyms and Abbreviations

Acronym	*Meaning*
5G	Fifth Generation
API	Application Programming Interface
ATDD	Acceptance-Test-Driven Development
B	Byte
CD	Continuous Development
CI	Continuous Integration
CNF	Cloud Native Network Function
CSV	Comma-Separated Values
DB	Database
DevOps	Development and Operations
DoS	Denegation of Service
DSE	Design-Space Exploration
E2E	End-to-end
EG	ETSI Guide
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
FUT	Function Under Test
GB	Gigabyte
Gbps	Gigabits per second
HTTP	Hypertext Transfer Protocol
I/W	Interworking
IBM	International Business Machines
ICMP	Internet Control Message Protocol
ICT	Information and Communications Technology
ID	Identifier
IP	Internet Protocol
JDBC	Java Database Connectivity
JMS	Java Message Service
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPI	Key Performance Indicator
LCM	Lifecycle Management
LDAP	Lightweight Directory Access Protocol
LO	Local Orchestrator

LoI	Level of Inventory
MANO	Management and Orchestration
MEC	Mobile Edge Computing
MQTT	Message Queue Telemetry Transport
ms	Milliseconds
MSC	Multi-Site Catalogue
MSI	Multi-Site Inventory
MSNO	Multi-Site Network Orchestrator
MSO	Multi-Site Orchestrator
MSO-LO	5G EVE Multi-Site NSO to Local Orchestrator
MTBF	Mean Time Between Failures
MTBE	Mean Time Between Events
MTS	Methods for Testing and Specification
NBI	Northbound Interface
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSO	Network Service Orchestrator
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
ONAP	Open Network Automation Platform
OS	Operating System
OSM	Open Source MANO
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
PNFI	Physical Network Function Instance
RAN	Radio Access Network
REST	Representational State Transfer
RFC	Request for Comments
RTT	Round Trip Time
SBI	Southbound Interface
SDN	Software Defined Network
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SUT	System Under Test

TCP	Transmission Control Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
TPC	Transaction Processing Council
TST	Testing
UDP	User Data Protocol
UPF	User Plane Function
VM	Virtual Machine
VNF	Virtual Network Function
VNFI	Virtual Network Function Instance
VPN	Virtual Private Network
WAN	Wide Area Network
WebDAV	Web Distributed Authoring and Versioning
WG	Working Group
WP	Work Package
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
YAML	YAML (Yet Another Markup Language) Ain't Markup Language
YCSB	Yahoo! Cloud Serving Benchmark

List of Figures

List of Tables

Executive Summary

Deliverable D3.6 is the first deliverable related to the Interworking Framework testing and it describes the test cases that are executed for the validation and performance assessment of the Interworking Layer implementation. This deliverable is intended to be aligned with D3.3 [1], which contains the first implementation of the Interworking Framework, as all the components and capabilities exposed by the I/W Framework must be tested in order to verify their correct operation before deploying them in a real production environment.

For achieving this, firstly the testing methodology that will be applied for the definition of the Interworking test suites is presented. In this case, the principles of software project testing are proposed, as there are a lot of similarities between this kind of developments and the I/W Framework itself; i.e. both can be decomposed into small test units that can be tested separately, verifying their correct operation in an isolated way. After that, the different components of the architecture can be integrated in a more complex solution which can be also tested in order to validate these new interactions. This process is repeated until reaching to the complete system itself, integrating all the components of the architecture and testing the overall system.

After describing the key aspects of the testing methodology to be applied, the components and capabilities to be tested in the Interworking Layer are presented, defining the testing points that must be considered for specifying the different test suites. Note that this deliverable only covers the formal definition of the test, in a tabular format, while the provision of the necessary scripts to execute the tests, the execution of these scripts and the results evaluation are tasks which are covered by D3.7, which is the second and last deliverable related to I/W testing purposes.

Furthermore, the testing plan is also aligned with the 5G EVE project roadmap planned for the final delivery regarding the Interworking Layer, which is presented in D3.3 – section 5 [1]. As a reminder, this updated roadmap defines two extra, intermediate deliveries that were not part of the original proposal, but it is foreseen they could be important to this project: Drop 1 at the end of 2019, for providing support of the execution of selected use cases, and Drop 2 for aligning the WP4 deliveries with an I/W Framework delivery that allows to test the end-to-end services provided by the Portal.

Finally, it is also defined a set of testing tools that may be useful for building the different test cases defined in the deliverable. These tools are partially inherited from all the work already done in WP5 regarding the testing and validation methodologies to be applied during the execution of tests. However, the selection of the tools and the environment to be used for testing the different test suites will be also defined in D3.7, as it is closer to the implementation part instead of the formal definition of tests.

1 Introduction

Deliverable D3.6 – *Interworking test suites* – represents the first deliverable of Task 3.4 – *Functional testing of interworking capabilities* –, whose objective is the introduction of the testing plan to be followed within Task 3.4 for testing the I/W Framework components and capabilities, defining all the possible tests to be executed, which will be based on a specific testing methodology, strategy and tools that are also described in the deliverable.

1.1 Initial context and terminology

First of all, the context in which this deliverable is involved will be presented. Mainly, deliverable D3.6 is related to the testing process of the Interworking Framework as a system, the components that compose the Interworking Framework and possible integrations between these components that may be useful to be tested.

Regarding the terminology used in this deliverable, most of the terms used in this document are explained in their corresponding section for an appropriate understanding of all the described concepts. Furthermore, it has also been included the necessary references to other deliverables or useful documents in order to clarify different aspects that may not be clear directly; e.g. concepts which have already been defined in previous deliverables and that are included again in this deliverable.

Regarding some nomenclature issues, it is worth mentioning that terms **Interworking (I/W) Framework** and **Interworking (I/W) Layer** have been used equally throughout the document. The same is true for **Multi-Site Network Orchestrator**, **Multi-Site Network Service Orchestrator**, **Multi-Site NSO** or **Multi-Site Orchestrator**.

1.1.1 Interworking Framework overview

The 5G EVE Interworking Framework, currently described in deliverables D3.1 [2], D3.2 [3] and D3.3 [1] from different points of view (reference model in the first two cases, and first implementation drop in the last one), is a unique selling point of the 5G EVE end-to-end facility, which aims at providing a unified and integrated experimentation platform spanning heterogeneous sites where diverse 5G capabilities and tools are deployed. Therefore, it is a combination of coordination features for the seamless orchestration and execution of vertical use case experiments over heterogeneous infrastructures. The Interworking Framework sits between the Experiment Portal, that is the frontend of the 5G EVE platform, and the site facilities where the vertical use case experiments must be deployed for testing and validation of 5G and service specific KPIs. In this line, Figure 1 shows the current Interworking Framework architecture and a high-level functional split.

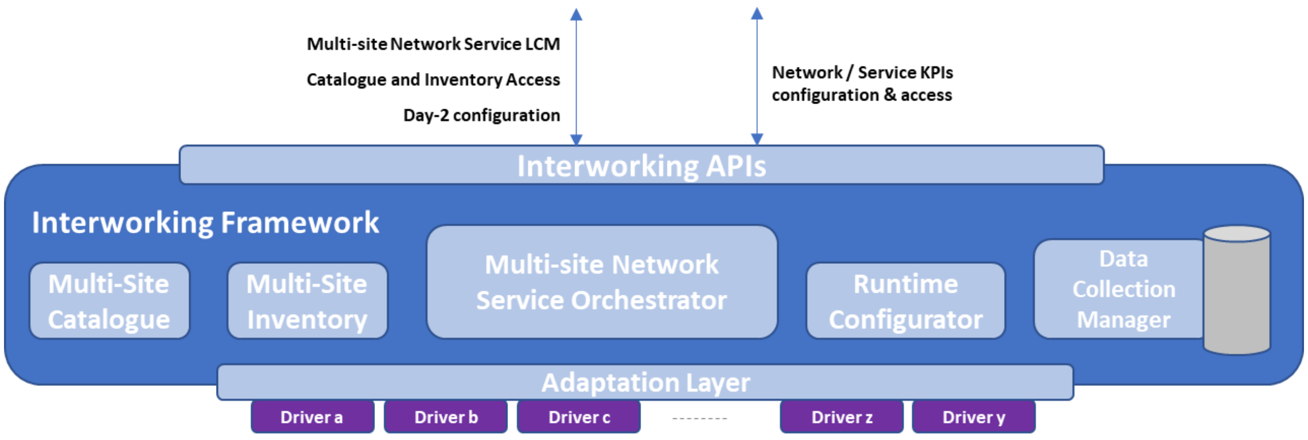

Figure 1: Interworking Framework architecture

The **two main reference points** exposed and provided by the 5G EVE Interworking Framework are:

- The **Interworking API** exposed at the northbound towards the 5G EVE experiment portal, which enables the following functionalities:
    - Access to multi-site to retrieve Network Services and VNF Descriptors, on-board new Network Services and VNF Descriptors.
    - Access to multi-site inventory information to retrieve details of already provisioned Network Services in support of vertical use case experiments.
    - Provisioning and lifecycle management of end-to-end Network Services.
    - Configuration of 5G network and service performance metrics to be monitored and collected during the execution of vertical use case experiments.
    - Common and site independent access to the collected monitoring information for each vertical use case experiment.
    - Runtime configuration of Network Services and VNFs.
- The **adaptation and abstraction interface** at the southbound provides a common, unified, abstracted access to individual site facilities services and APIs. Moreover, it is in charge of:
    - Exposing a set of common internal APIs and models for accessing transparently to per-site management, control, orchestration and monitoring services.
    - Translating them into the site-specific APIs and models via specific drivers.

The I/W Framework architecture, presented in Figure 1, is decomposed into the following functional components:

- The **Multi-Site Catalogue** decouples the Network Service Descriptors exposed to the 5G EVE experiment portal, which may span multiple sites and logically represent the actual network slice offers from the Network Service Descriptors collected from each of the site facilities, representing the actual capabilities of the sites.
- The **Multi-Site Inventory** is the counterpart of the catalogue component for what concerns the information on provisioned and instantiated network slices in the 5G EVE end-to-end facility. It is fully managed, in terms of information stored, by the Multi-Site Network Service Orchestrator, and it maintains detailed information of the running.
- The **Multi-Site Network Orchestrator** is the core component within the Interworking Framework and responsible for coordinating the provisioning and lifecycle of Network Services across the site facilities, as required to deploy end-to-end network slices for the execution of vertical use case experiments. It leverages on the per-site orchestration components and features, as they provide the fundamental logics and coordination within each 5G EVE site facility.
- The **Data Collection Manager** is a key component within the Interworking Framework, and it coordinates the collection and persistence of all those network and vertical tailored service performance metrics that are required to be monitored during the execution of experiments for testing and validation of the targeted KPIs and results for a given experiment.
- The **Runtime Configurator** allows to apply tailored, Day-2 runtime configurations to the provisioned end-to-end Network Services and VNFs in support of the vertical use case experiments**.**

1.1.2 Test suites motivation and scope clarification

The main motivation of test suites is to test and validate the Interworking Framework system, their components and its defined features. The value that these tests provide is the verification that the implementation process has been done successfully, accepting that the Interworking Framework is ready to be used in a production environment. This is crucial in order to support the deployment of several experiments in the 5G EVE platform, which implies several requirements in terms of load, latency and communications, among others.

Note that the testing methodology described in this document does not consider the end-to-end test of the 5G EVE platform, as the focus is limited only to the I/W Framework.

1.2 Document objectives

The main objectives of this document, which are aligned with the expected objectives of Task 3.4, are the following:

1. Presentation of the testing methodology to be followed in the definition of test suites.
2. Definition of the Interworking testing plan, according to the testing methodology defined beforehand. It includes the definition of the test cases that are executed for validating the implementation of the Interworking capabilities across the different site facilities, as defined by the Interworking reference model.
3. Definition, jointly with D3.3 – *First implementation of the interworking reference model* [1] and Task 3.3 – *Interworking model implementation and deployment*, of the roadmap related to the implementation and testing of the Interworking Framework.
4. Presentation of the set of tools to be considered during the Interworking testing process.

This document is aligned with deliverable D3.3 [1], in which the first drop of the implementation of the Interworking reference model is presented, so that a first complete definition of the Interworking Framework is provided by describing both the implementation of the different components that belongs to the I/W Framework and the related test cases to each component and the system itself.

The definition of test cases is done following standard principles, providing a unified format for all the test defined within the scope of this deliverable regardless of the component or capability under testing. Furthermore, note that this document just provides the test definition, leaving the execution and the evaluation of this set of tests for D3.7 – *Report on the execution of the interworking test suites*, which will complete the expected objectives to be achieved within Task 3.4.

1.3 Document structure

The Sections of this deliverable D3.6 following this Introduction are organized as follows:

- In **Section 2**, it is included the test suites general methodology which is followed for defining the test suites. This methodology is associated to the main testing methodologies that are present for the design and execution of test cases for software projects. Standard specifications related to the definition of test suites are also included in this section.
- **Section 3** provides the definition of test cases related to the Interworking Framework, mapping the general methodology presented in the previous section with the final Interworking testing plan proposed. For achieving that, the identified testing points (the I/W Framework components and aspects mainly related to site facilities' interconnection) are presented firstly, defining the test cases for these identified points afterwards. Finally, it is proposed a risk plan for covering all the possible problems and risk that may appear during the test execution.
- **Section 4** presents the roadmap which contains the different drops for the definition and execution of test cases, aligned with the implementation of the Interworking Framework (under the scope of Task 3.3).
- In **Section 5**, it is presented the set of proposed tools which can be used during the testing process, being aligned with the work already done within WP5. For that purpose, a high-level architecture which can be reproduced for the testing of all the identified testing points is also depicted.
- **Section 6** summarises the conclusions extracted from all the information included in this deliverable, regarding the definition of test cases.

2 Test suites general methodology

In this Section, it is included the test suites general methodology which will be used for defining the Interworking test suites, following the main testing principles that are present in the design and execution of test cases for software projects. Standard specifications related to the definition of test suites are also described in this Section.

2.1 Test suites overview

Expressed in a general way, a test suite is *a collection of information* (test type, priority, related tests, execution time, expected results in terms of KPIs such as latency, load, throughput and reliability…) usually presented in the form of a blueprint that is *used for deploying, operating* and *evaluating* certain scenarios.

At 5G EVE highest level, these blueprints are used by various verticals. But in the specific case of I/W Framework and of this document, test suites are defined for deploying, operating and evaluating I/W Framework system and its components itself.

Test suites can use other level; for instance, a 5G EVE Portal test suite may call one or more Interworking Layer test suites that may also call one or more per-site location test suites, depending on the case. As a result, the test suite blueprint format will differ in types depending on to which layer it belongs:

1. Requirements of verticals at 5G EVE highest level will belong to KPI that are relevant for the vertical.
2. Requirements related to general system performance, such as latency, load or throughput will pertain to 5G EVE sites levels.
3. Requirements related to 5G EVE specific infrastructure will pertain to identified testing points, such as components and capabilities.

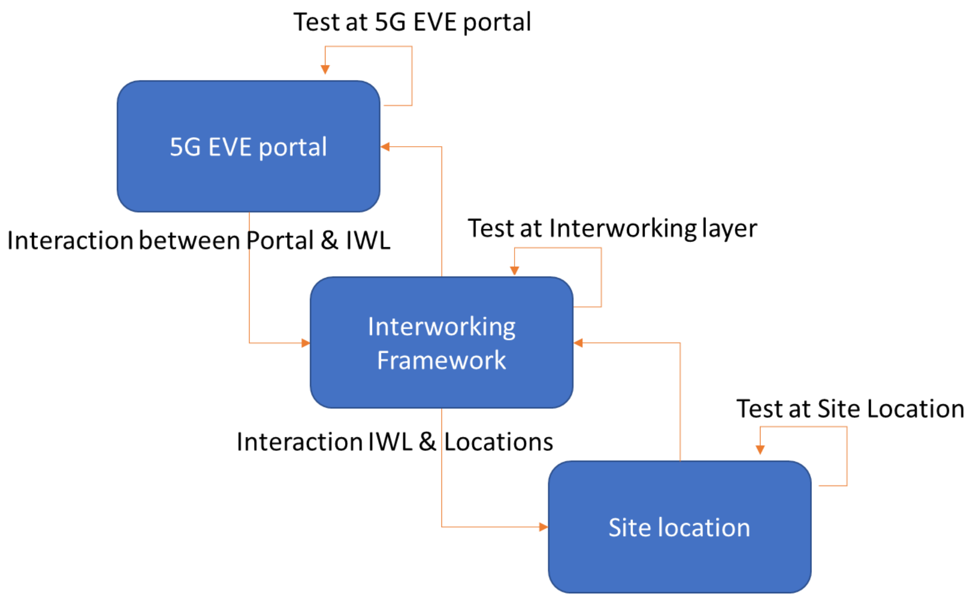

Figure 2: Different testing levels

This document focuses on the 3rd level, functional and performance aspect I/W framework components, features and interfaces. It includes required performance between IWL and other up and down layers. But it excludes end-to-end test of 5G EVE platform performed at upper layer as well as site location performance performed at lower layer.

2.1.1 Challenges

The scope of the testing methodology is limited to the functional and performance aspects of all the I/W Framework components, features and interfaces. As a result, it does not consider end-to-end test of the 5G EVE platform.

The peculiarities of the Interworking Layer testing process are listed below:

1. It is part of a system chain: Interworking Layer is part of a full system chain, between the Portal at northbound and Site facilities at southbound.
2. Interworking Layer has versatile roles: being part of a chain, some of the roles carried out by the Interworking Framework consist in relaying the information (in both directions) and in interpreting the information (also in both directions), among others
3. Interworking Layer requires new technology: as explained in D3.3 [1], no existing projects has covered such a similar need than the ones covered by the I/W Framework, therefore some new technology must be developed. It is not just a question of integration: new components must be delivered and inserted.

In that line, there are specific challenges that must be addressed by the definition of test suites:

- Since the assets (either existing or new) combination is needed, it is also needed to carefully design the test suites for validating the implementation of the components, integrations and system.
- Since the Interworking Layer layer is part of a larger system, a methodology able to cope with the increasing complexity of such a system must be provided, growing from isolated parts to an end-to-end dimension. Proxies of the larger system need to be used to perform intermediate validation.
- Finally, since there are several roles present in the Interworking Layer, the carefully design of all the variety of use cases is also needed.

2.1.2 Principles and strategy to be followed

In this scope, it is followed the general software testing principles and strategies present in most software development projects.

In fact, software testing is a well-known discipline. Various methods, approaches and philosophies have been explored over the time, adapting them to the needs of the different projects to be developed, and it would go beyond this deliverable to describe all of them, so only the most relevant options are commented in the following list:

- **Design-space exploration/In vivo testing:**
  - DSE is an exhaustive testing enabling formal verification. It can predict and guarantee performance but requires the definition on system model and is time consuming.
  - In vivo testing requires to create a testing environment usually called “staging environment” plus the deployment of in-situ probes to collect KPIs and validate pass.
  - There is a continuum between DSE, staging and in operations testing as one can consider testing is “never ending” with machine-learning applied on the field.
- **Modular vs system testing:**
  - Depending on the nature of the component (black box or system of components), various testing levels can be envisaged.
  - The lowest “testable” layer is usually a monolith already compiled and considered as a “black box”.
  - All other levels can be assembled from other blocks.
  - The levels need to be clarified to keep clarity in the testing process as it is virtually impossible to test all combinatory options.
- **Types of tests:**
  - Functional: proof that the development supports the desired function.
  - Performance: proof that the function operates with expected KPIs.
  - Stability: proof that function operate x% of the time, with MTBF/MTBE, with given recoverability…
  - Scalability: proof that function can scale along certain axis (adding parameters, usages…).

  - Security: follow Chaos engineering principles, automatic detection and recovery.
- **Automated vs manual testing:**
  - Scalability (in number of components, providers, test cases…) is an issue.
  - As complexity increases, automation becomes mandatory to scale.
  - Scripting, operating the tests, collecting, storing and interpreting data are key challenges.
  - Component/system expert knowledge is required to be captured correctly to keep test representativity.
  - DevOps technologies are needed to keep up with new software versions.
  - Machine learning are used to interpret both off and on-line collected data.
- **Microservices vs Monoliths:**
  - As a system grows, components start to have dependencies, hindering the system's speed of evolution.
  - A software system is made of multiple components assembled in larger systems and provided by various sources such as open source, previous research projects, etc.
  - Function decomposition can be refined over successive iterations.
  - Components should be "loosely coupled" to separate concerns.
  - Following 12 factors[1] design principles is a good recommendation.
- **Waterfall vs. agile**
  - Methodologies have ups and downs.
  - Waterfall assume a preliminary detailed specification and prefers exact implementation to the detriment of evolution.
  - Agile promotes quick and short iterations to obtain quickly "Minimum Viable Products" always aligned with customer demand.
  - Actually, there is no "1 size does not fit all" and methodologies should be mixed depending on the cases.
- **Open source vs. closed source**
  - Open source has faced and solved most of above-listed challenges.
  - Open source is an excellent provider of technology optimised for these purposes (e.g. Jenkins, Git…).

This analysis grid is used to position the current solution, based on a modular and system testing among others: firstly, the identification of modules to be tested is done. After it, the possible integrations between modules and the complete system follow with all the components and these integrations. The grid is provided in Table 1.

Table 1: Positioning Interworking Framework testing methodology according to software testing methodologies

Option	**Interworking Layer layer position**
Design-space exploration/In vivo testing	No design space, only "in vivo" testing using stubs.
Modular vs system testing	Both are used at two levels: FUT an SUT.
Types of tests	Mainly functional tests.
Automated vs manual testing	Both, but automated tools will be introduced in the next D3.7.
Microservices vs Monoliths	Microservices.
Waterfall vs. agile	Both.
Open source vs. closed source	Open source.

[1] https://12factor.net/

2.2 Testing standard specifications

Testing is a key aspect of any product development process and it is the mean for validating functional features, as well as compliance to standards and multi-vendor interoperability. Being such an important step for delivering new products, nowadays its role is getting more and more relevance due to wide implementation of DevOps methodologies in the industry, where testing is central in the Continuous Integration (CI) and Continuous Development (CD) cycles.

In this context, standardization is key to regulate how testing shall be implemented, with well-defined methodologies and processes to unify how products are tested and validated against standard protocols and specifications in general. Indeed, quite a lot of efforts have been spent in standardization bodies to define such methodologies and processes; one of the most active, and relevant with respect to 5G EVE (and its Interworking Framework), is ETSI.

In particular, the ETSI Methods for Testing and Specification (MTS) group identifies and defines advanced specifications and testing methods, leveraging on innovative techniques to improve the efficiency of both the standard description and related conformance and interoperability testing processes. In practice, it develops studies, guidelines and test specifications for specific ICT technologies that are not already covered by existing ETSI groups or bodies.

Two of the ETSI MTS specification documents can be considered as relevant to 5G EVE and the I/W Framework testing methodologies: **ETSI EG 202 237** [4] and **ETSI EG 202 568** [5]. These documents provide main guidelines for the use of common methods for developing test specifications in the context of standardized IP-related communications protocols, systems and products. In general, the proposed methodologies and processes are applicable to all those kinds of systems and protocols where IP-related communications happen. In terms of main relevant (with respect to the 5G EVE I/W Framework) testing aspects defined in ETSI EG 202 237 [4] and ETSI EG 202 568 [5] are the identification of methodologies for the test development process (with split into conformance testing and interoperability testing), and for the development of test specifications (as the combination of test configurations, test suites and test descriptions).

On top of these two specifications, the ETSI NFV TST Working Group (WG) has defined its own testing methodologies. Being ETSI NFV principles, architectures and interfaces at the base of the 5G EVE architecture and of the I/W Framework components and workflows in particular, it is worth to summarize how the TST WG has built and specified its testing methods. The ETSI NFV TST WG is responsible for the development of testing frameworks to validate interoperability and conformance of ETSI NFV specifications, thus driving how testing should be performed for NFV products. Moreover, it has the aim to facilitate and enable the implementation and validation of NFV related use cases.

In particular, the **ETSI NFV TST 002** – *Report on NFV Interoperability Testing Methodology* [6] defines the guidelines and methodologies for NFV products testing, differentiating between conformance and interoperability tests: conformance tests validate if NFV products correctly implement a particular standard, by defining whether or not the Function Under Test (FUT) meets the requirements specified by the standard (e.g., it tests protocol message contents and format as well as the correct sequences of messages), whereas the interoperability testing validates that NFV products can work with other NFV products, by checking that end-to-end functionality between (at least) two FUTs is as defined by the relevant NFV standards.

Based on this high-level categorization, the ETSI NFV TST 002 [6] standardizes the following testing aspects:

- Definition of basic NFV testing concepts.
- Instructions for the development of NFV test specifications, including test configurations and test descriptions.
- Description of the NFV testing process.

In summary, the main testing concepts and terminology introduced in ETSI NFV TST 002 [6], and relevant to the 5G EVE I/W Framework testing principles, are the following:

- **Function Under Test (FUT):** combination of software and/or hardware components implementing the NFV functionality that is tested through one or more NFV reference points.

- **System Under Test (SUT):** it is the combination interacting FUTs coming from different vendors, that define the whole system tested for NFV interoperability purposes.
- **Test interfaces:** it is the whole set of interfaces that FUTs and SUTs expose to enable the testing of the given NFV functionality. They are accessed by the test system to trigger and verify the test behaviour and can be used for test monitoring and analysis as well.
- **Test environment:** it is the combination of equipment, functions and procedures required to test the given NFV functionality. Entities in the test environment access the FUTs through the test interfaces and ensure the interpretation and execution of the relevant test descriptions by coordinating the required actions on the test interfaces.
- **Test purpose:** it is the full description of the objective of each test case. The collection of test purposes in a test suite should cover the entire relevant NFV specification(s). One or more test descriptions are derived from a given test purpose.
- **Test description:** it is the detailed set of instructions that need to be implemented for performing a given test. It includes descriptions of requirements for the test be executed, as well as clear definition of test objectives.

Consequently, ETSI NFV TST 002 [6] defines a list of minimum information that is required to describe a test description and that is used to execute the test itself. In particular, for each test purpose, one or multiple test descriptions can be specified. Each test description has the goal to describe all the steps required (i.e. how) to achieve a test purpose (i.e. what). In TST 002 [6], the following information is required to be part of any test description:

- **Identifier:** a unique identifier of the test description. Well-defined naming conventions are recommended.
- **Test purpose:** the description of the objective of the test description.
- **Configuration:** reference to the applicable test configuration.
- **References:** reference to the base standard specifications describing the feature under test.
- **Applicability:** list of items and features that need to be supported by the FUT in order to execute the test.
- **Pre-test conditions:** specific conditions that need to be met by the FUT prior to start executing the test. It can refer to initial state of the FUTs or specific test configurations.
- **Test Sequence:** a detailed description of the steps to be followed for achieving the given test purpose. They need to be defined with the aim of having the test applicable to a range of different types of implementation.

2.3 Testing methodology

This section describes the testing methodology for the Interworking Framework. As described above, the scope of the I/W Framework testing is to test and validate the framework itself, as an independent system. For that reason, the 5G EVE platform end-to-end testing is out of the scope of this deliverable.

Therefore, following the principles and strategy described in Section 2.1.2, it is proposed three kind of test categories for validating the I/W Framework, also aligned with the standards discussed in Section 2.2:

- The **Component Tests** (also known as Unit Test), mandatory for all the I/W Framework components, are the main kind of tests defined in the Interworking test suites and are oriented to test each FUT (or testing points related to specific components) of the I/W Framework separately.
- After these unit tests, it is also proposed to do **Integration Tests** for testing and validating the different connections between components of the I/W Framework in which a specific interaction may be required.
- The **System Tests** are intended to validate the whole I/W Framework system (i.e. SUT), with all the components already deployed in the production environment.

Tests defined for each kind of test category can be also divided in different types. Mainly, it will be executed functional tests, which are used to verify that the functions defined in the FUT or SUT operate in line with the functional requirements defined. However, other types of tests can be also defined (e.g. load tests or stress tests, for testing scalability features) in case it is needed for the specific FUT or SUT.

As a result, the basic definition of a test suite for a given FUT or SUT shall include the functional test type and, depending on the FUT or SUT, may also include load and/or stress tests and/or other useful types of tests.

In this deliverable, all tests are described using a common table in text format. Furthermore, it is planned to incorporate a more detailed description of the test suites using a specific language (script) for implementing the test (e.g. in Robot Framework format), and presenting the results obtained from the execution of the test. That description will be included in the next deliverable D3.7.

2.3.1 Component Tests

The goal of the Component Tests is to validate the behaviour of each I/W Framework component individually, which are the Functions Under Test. In this scope, all the testing process related to each component is considered internal testing (or design testing) and should follow the internal software testing methodology described by each partner, which are reflected in the tests defined in Section 3.2.

The Component Tests are defined for the identified testing points related to the components from the I/W Framework (please, refer to Section 3.1.1), generally the northbound and southbound interfaces of these components. The Component Tests must include, among other fields:

- A description of the purpose of the test.
- The type of test. In general, functional tests will be used, but there could be specific components in which it can be defined other type of tests, such as load tests or stress tests (e.g. Data Collection Manager).
- The configuration of the component to be used during the testing.
- The pre-conditions in the FUT.
- The steps for executing the test.
- The expected result.

Independently of the Component Test executed, it is required a testing environment that provides a client emulation, which generates API requests towards the component and analyse its responses, as well as server emulations for those cases that requires an external communication. The test environment to be built in this scope can be defined with the set of tools proposed in Section 5, and each component will use the necessary set of tools for executing properly all the test suites defined. These test environments will be defined in D3.7.

2.3.2 Integration Tests

The Integration Tests are required for validating the integration of some I/W Framework components which may require to be connected in specific workflows. The following integration points are identified:

- Multi-Site Network Orchestrator and Multi-Site Catalogue: integration using NSD management API.
- Multi-Site Catalogue and Multi-Site Inventory.
- Multi-Site Network Orchestrator, Multi-Site Catalogue, Data Collection Manager and Runtime Configurator with the Adaptation Layer.
- Drivers of Adaptation Layer with their respective NFVO.

The methodology used in the integration testing is the same used in the Component Tests, with the difference that instead of using mock servers for the test validation, the real components are used.

2.3.3 System Tests

The System Tests are required for validating the whole I/W Framework. The approach for this kind of testing process is the same that in Integration Tests, taking into account that the testing points are the public northbound and southbound interfaces and all I/W Framework components are integrated.

As a result, the test environment in this kind of tests would be the production environment itself, reproducing all the Component and Integration Tests in the final solution in order to test the correct behaviour of each component and the possible interconnections between them.

3 Interworking testing suites proposal

The main objective of this section is the presentation of the Interworking testing suites, specifying all the general terms and concepts presented in Section 2 within the Interworking Framework scope.

3.1 Identified testing points

The I/W Framework testing points are the features to be tested with the execution of the different test suites defined. Depending on the target element that is the objective of the test, the testing points can be determined for both I/W Framework components and capabilities:

- In the case of the different **components** placed in the I/W Framework, i.e. Multi-Site Catalogue, Multi-Site Inventory, Multi-Site Network Orchestrator, Data Collection Manager, Runtime Configurator and Adaptation layer (with the interface from Multi-Site NSO to local Orchestrators), the testing points are identified within the specific FUT which is intended to be studied by Component Tests.
- Regarding the I/W Framework **capabilities**, the testing points detected are focused mainly on validating the interconnection between different site facilities. In this case, the I/W Framework is considered a SUT in which these testing points will be tested with the different System Tests defined for that purpose.

In the following Sections 3.1.1 and 3.1.2, it will be introduced the different testing points to be considered within the Interworking testing suites, according to the two main categories aforementioned.

3.1.1 Components

3.1.1.1 Multi-Site Catalogue

The Multi-Site Catalogue stores the full set of Network Service Descriptors, PNF Descriptors and VNF Packages that can be used to compose a new end-to-end vertical experiment (from the 5G EVE Portal perspective), as well as to deploy and instantiate a vertical experiment in one or more sites (from the Multi-Site NSO perspective).

The main **testing points** for the Multi-Site Catalogue are:

- The **northbound APIs** (as specified in deliverable D3.2 [3]), for what concern **onboarding and retrieval of catalogue** content operations.
- The **southbound APIs** (as specified in deliverable D3.2 [3]), for **onboarding of Network Service Descriptors** modelling vertical experiments into the per-site orchestrators (and catalogues), for both the single and multi-site cases.

Therefore, the Multi-Site Catalogue tests described in this document mostly target the functional validation of the component when stimulated at its northbound APIs for onboarding related operations. In addition, specific tests will also validate the Multi-Site Catalogue features by stimulating it through its southbound APIs for what concern the VNF Packages and descriptors onboarding operations (in reference to the onboarding workflows specified in deliverable D3.2 [3]).

With respect to the two main categories of testing points listed above, the following **test categories** apply:

- **Test of Network Service Descriptor onboarding in single site:**
  - <u>Testing point:</u> NBI.
  - <u>Verification point:</u> NBI, SBI, internal DBs (if applicable).
- **Test of Network Service Descriptor onboarding in multiple sites:**
  - <u>Testing point:</u> NBI.
  - <u>Verification point:</u> NBI, internal DBs (if applicable).
- **Test of retrieval of VNF Packages from sites:**
  - <u>Testing point:</u> SBI.
  - <u>Verification point:</u> NBI, internal DBs (if applicable).
- **Test of retrieval of Network Service Descriptors from sites:**

  - Testing point: NBI.
  - Verification point: NBI, internal DBs (if applicable).
- **Test of retrieval of PNF Descriptors from sites:**
  - Testing point: NBI.
  - Verification point: NBI, internal DBs (if applicable).

It is worth to highlight that the single and multiple site cases listed above refer to onboarding of descriptors that are built in the 5G EVE Portal during the vertical experiment preparation phase, and that can possibly be executed either in a single or in multiple sites (i.e. the same exact experiment). Indeed, in accordance with the D3.2 [3] catalogue related workflows, it is possible that the same exact Network Service Descriptors (e.g. modelling a single-site experiment) will be onboarded to more than one site orchestrator when the related VNFs (and VNF Descriptors) are the same (i.e. same identifiers). This is independent from the Network Service Descriptors being modelling either single or multi-site vertical experiments.

3.1.1.2 Multi-Site Inventory

Similarly to the Multi-Site Catalogue storing the full set of available VNF, PNF and NS Descriptors, the Multi-Site Inventory (MSI) stores the information related to the active VNF, PNF and NS Instances. Two types of **operations** are envisioned for the MSI:

- **Write** in the inventory.
- **Query** the inventory.

The only component with rights to write in the MSI is the Multi-Site Network Service Orchestrator. There is a single exception to this rule, which is the possibility to admit manual entries in the first stages of the I/W Framework implementation. Local Network Service Orchestrators, therefore, will not have direct access to the MSI, as depicted in Figure 3; in other words, there is no SBI for the Multi-Site Inventory.

The main **testing points** for the **writing** operation are associated with the lifecycle management features of VNFs, PNFs and NSs: **instantiation, scaling, update** and **termination**.

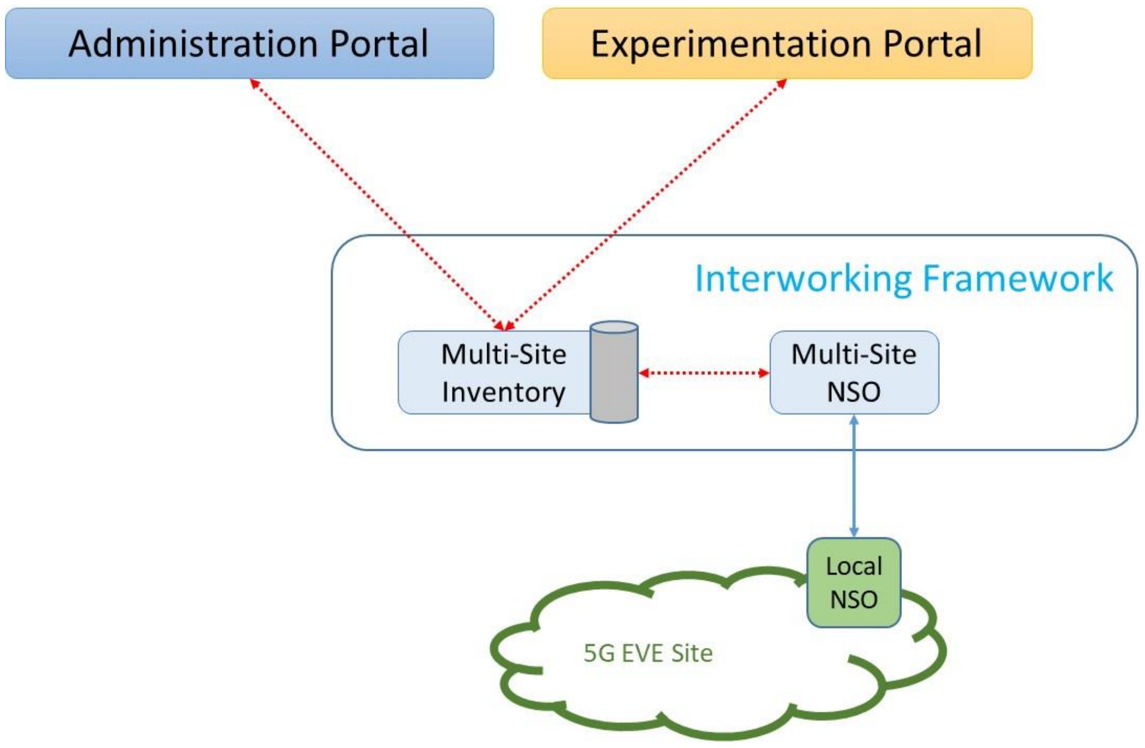

Figure 3: Multi-Site Inventory interactions

On the other hand, the **query** operation can be executed by three components. The interface will be equivalent for all three, but the rights will be different:

- The **Experimentation Portal** will be able to launch requests, on behalf of an experimenter, to request the status of its own experiment.
- The **Multi-Site NSO** and an **Administration Portal**[2] will be able to launch global requests, so that the former knows the availability of resources when new deployment requests arrive, and the latter can present the full status of the 5G EVE infrastructure to an operator (for example, for troubleshooting purposes).

In summary, the **main testing point** for the query operation is the validation of the **NBI** for the Multi-Site Inventory.

3.1.1.3 Multi-Site Network Orchestrator

The Multi-Site Network Orchestrator is in charge of coordinating all the site orchestrators for deploying Network Services that compose an end-to-end vertical experiment.

The main testing points for the Multi-Site Network Orchestrator are:

- The **northbound APIs** (as specified in deliverable D3.2 [3]), for **Network Services Lifecycle Management** operations.
- The **southbound APIs** (as specified in deliverable D3.2 [3]), for **NS LCM operations**, for both the single and multi-site cases.
- The **westbound API** with Multi-Site Catalogue (as specified in deliverable D3.2 [3]) **for retrieving NSD information**, especially the NS and site NFVO IDs.

The Multi-Site Network Orchestrator tests included in this document are functional tests of the component, simulating northbound API operations and having simulated servers of Multi-Site Catalogue and the Multi-Site NSO to LO components.

The following **test categories** are expected:

- Test of **single-site NSD deployment**.
- Test of **multi-site NSD deployment**.
- Test of **Notification system**.
- Test of **Multi-Site Inventory update**.

The main difference between the single and multi-site testing is related to the transactional service required for the multi-site environment, in order to guarantee the consistency of the NS deployments across multiple site facilities.

3.1.1.4 Data Collection Manager

The Data Collection Manager is the component responsible for the collection and persistence of all the network and vertical performance metrics that are required to be gathered during the execution of experiments, as well as the related KPIs and results calculated from these metrics, providing a publish-subscribe mechanism for the delivery of that data.

The high-level **testing points** that have been identified for the Data Collection Manager are related to the specific technology which will implement this component (i.e. the Apache Kafka[3] solution, as presented in D3.3 [1]) and the NBI and SBI with the 5G EVE Portal and Site facilities respectively:

[2] The Administration Portal, logically, is a separated entity from the Experimentation Portal, permitting the management and monitoring of the 5G EVE components by human operators. In this document, it is not covered how it is implemented (or even if it is implemented at all): it may be a hidden part (for experimenters) of the 5G EVE Portal, it may be a totally different entity, or it may not exist. The key idea is that the full status of the 5G EVE infrastructure will not be presented to experimenters, but the Multi-Site Inventory will support in its API this capability.

[3] https://kafka.apache.org/

- **Northbound interface** between the Data Collection Manager and collection-monitoring tools (from WP4) and Result Analysis and Validation tool (from WP5). The operations that are handled by the Data Collection Manager in this interface are mainly related to the data delivery towards the components subscribed to a given topic, and with the reception of data from a given topic (e.g. when the Results Analysis and Validation component publishes the KPIs and results to be presented afterwards). Moreover, the signalling topics are also present in this interface, as the Data Collection Manager needs to deal with the subscription process executed in the components from upper layers, which will conclude with the obtention, for each component that is using the publish-subscribe queue, of the topics to be subscribed. Note that topic subscription will trigger a process in the Data Collection Manager for start consuming from the specific topic.
- **Southbound interface** between the Data Collection Manager and Data shippers (from site facilities). This interface only manages the publish operation from the Data shippers (with the Kafka Producer API), which provides the metrics gathered from a given test execution. Although the Data shippers need to be configured previously by providing them the Data Collection Manager IP address and topic in which it has to publish, this configuration is out of the scope of this component, as it is closer to the behaviour of the Runtime Configurator component.
- **Operation of the selected technology** for the Data Collection Manager component (Apache Kafka). This testing point, even though it is not related to any specific external interface of the Data Collection Manager, is crucial for ensuring the correct behaviour of this component. Within this scope, there could be some tests related to the scalability of the component (e.g. deploy the component in a cluster mode and test what is the optimal number of nodes for building the cluster, check how many topics can be handled by the Data Collection Manager for a given deployment, etc.), which are more related to other kind of tests like load or stress tests.
- **Integration of other functionalities** (e.g. cold storage, fast queuing…). In order to verify that these improvements can be integrated in the Data Collection Manager, they must be tested separately. As these functionalities are currently in work-of-progress status regarding the implementation of the Data Collection Manager, they will not be considered in this first drop of the test suites.

Delving into the previous concepts, the final **testing points** to be considered for the Data Collection Manager are related to the publish, (un)subscribe and deliver **operations** presented in D3.3 [1], no matter the interface (NBI or SBI) that is used for executing these operations, as these operations are present in both. In that way, it may be more interesting to define a specific chain for the testing process (e.g. start with a subscribe operation, continue with a publish operation, and the delivery operation should be triggered automatically). Moreover, it should be also considered as testing points the different **phases** (subscription, monitoring and data collection and un-subscription) regarding the experiment monitoring and maintenance stage of an experiment, as presented in D4.1 [7]. These testing points are closer to functional tests; but, for scalability tests (load tests, stress tests…), it should also be considered the **Data Collection Manager itself**, with the different parameters and configurations that can be used for its deployment.

3.1.1.5 Runtime Configurator

The Runtime Configurator, as commented in D3.3 [1] with the update of this component, is intended to provide the necessary Day-2 configuration to the VNFs and PNFs that belongs to a given experiment. This Day-2 configuration can be divided in two main types: provision of useful configuration for the experiment before starting with it (e.g. the configuration of Data shippers aforementioned) and the execution of different commands related to each stage of the test (e.g. execute a *ping* between two VNFs/PNFs). For that purpose, the selected tool will be Ansible[4].

The main **testing points** for this component are the following:

[4] https://docs.ansible.com/ansible/latest/index.html

- **Northbound interface** between the Runtime Configurator and the Experiment Execution Manager (from WP5). This connection allows to make the necessary calls to the Runtime Configurator in order to apply all the configuration of an experiment defined in the templates consumed by the Experiment Execution Manager (which is based on Jenkins[5] and Robot Framework[6] tools). The key aspect to be tested in this connection is the interaction between Robot Framework and Ansible. As a result, having different test templates defined in Robot Framework languages, it must be tested that Robot Framework is able to establish a connection with Ansible and provide to it the necessary configuration parameters (e.g. IP addresses of the VNFs) that will be needed for the connection with the VNFs/PNFs.
- **Southbound interface** between the Runtime Configurator and VNFs/PNFs to be configured. This testing point is achieved with the usage of Runtime Configurator templates (i.e. Ansible playbooks), in which it is included all the scripts and configurations to be applied to the different VNFs/PNFs through an SSH connection. Although it can be tested separately, it makes more sense to execute these tests just after testing the northbound interface. As a result, the complete sequence would be the following: Experiment Execution Manager would call the Runtime Configurator templates, and these templates would apply the configuration in the VNFs/PNFs afterwards.

3.1.1.6 Adaptation Layer: Multi-Site NSO to local Orchestrators interface

The Multi-Site NSO to local Orchestrator interface (MSO-LO) is the portion of the Adaptation Layer that enables the I/W Framework to access the orchestration features provided by the trial sites involved in the 5G EVE project. The test of this component is extremely important to ensure the correct integration of local NFV Orchestrators with the I/W Framework.

The MSO-LO interface consists of a REST API exposed to the Interworking Layer and a series of drivers implemented to interact with local orchestrators of different technologies. The specification of features, requirements and a textual description of the API's methods are provided in D3.2 [3], while implementation details are presented in D3.3 [1].

The methods and paths of the REST API are the set of Function Under Test (FUT). Unit tests for this component are aimed to check that, given a REST API request from the I/W Framework, the correct output (JSON/YAML) is returned. The **structure** of each unit test is based on the GIVEN-WHEN-THEN and generally proceeds as follows:

- **GIVEN (preparation):** a REST API call with necessary input is prepared.
- **WHEN (execution):** the method related to the previous API call is executed.
- **THEN (assertion):** the execution output is checked against a predefined, expected output.

As the MSO-LO interface is intended to interact with local NFVO, it is necessary to provide a test environment to enable the test executions. Instead of deploying real instances of NFVOs, the behaviour of their Northbound Interface (NBI) is mocked. For example, in the case of the OSM[7] orchestrator, it is emulated its NBI by using the OpenAPI specification and Prism[8], a software that can create a fake REST API server by taking a YAML file as input. Prism can generate all the necessary HTTP status codes, sample output, and even dynamic output with fake values that respect the output schema. It is used this or similar mocking techniques also for other NFVO technologies. This methodology for unit tests makes them a hybrid between unit and integration tests as the test execution is not completely contained in the source code (Python) but is depending on an external component (Prism). This design choice is justified with the advantage of keeping the test source code simple, without the need to hard-code and maintain a lot of fake input and output to emulate NFVO interactions. Furthermore, a change in any NFVO NBI definition would only result in the download and update of a new

[5] https://jenkins.io/

[6] https://robotframework.org/

[7] https://osm.etsi.org/

[8] https://stoplight.io/open-source/prism/

OpenAPI specification and tests would be able to run. None or minor changes would be needed in test source code, while development effort can be directed to the actual application.

Unit tests organization follows the OpenAPI and application source code:

- A Python module containing tests for the **functions managing NFVO information** stored in the local database.
- A Python module containing tests for the **functions managing subscriptions information** stored in the local database.
- A Python module for each NFVO driver containing tests for the **functions related to NS and NFV instance management**.

Three or more unit tests are provided for each function/method, depending on the HTTP status codes that are worth to be tested. As a base, it is always tested code 200 (OK) or its variations, 400 (Malformed), and 404 (Not Found). Other codes can be tested depending on the specific method. The single unit test asserts the correctness of status code, header, and payload returned as output by the MSO-LO interface.

3.1.2 Site facilities' interconnection

In deliverable D3.2 [3], a proposal for inter-site connectivity was included, so that it could satisfy the requirements imposed by the Interworking Layer as well as the requirements that come from each experiment defined and deployed in site facilities. These requirements were considered by WP2 to deploy a connectivity solution for 5G EVE, but they were not the only ones. The inter-site connectivity also hosts other communication channels, like the interconnection with the 5G EVE Portal, the data and control planes for experiments, and several others.

In that sense, this test plan is not designed to verify completely all the interconnection features, but just those which are meaningful for the Interworking Framework.

3.1.2.1 Connectivity

Being the topology of 5G EVE's interconnection a star in the orchestration plane, with the I/W Framework deployed in the centre of the star, the connectivity testing on this plane will be focused on ensuring that the I/W Framework's modules implementing a SBI can reach each of the sites via the VPN interconnections. For this, specific IP address ranges must be defined per site, which should be reachable via *ping*[9] from the site at Turin.

Also, and depending on where the 5G EVE Portal is finally deployed, it might be required to repeat this connectivity test against the Web Portal (if deployed outside the project sites, which is a possibility).

The test plan is not designed to verify the availability of the connectivity. Since this is a long-term indicator, the best mechanism to measure it is via monitoring tools that continuously verify the status of the connections and extract the correct availability results.

Testing the Orchestration Plane connectivity

The objective of these tests is to verify the following communication: a) between the I/W Framework and the different components (e.g. VNF) located in different sites; b) between the I/W Framework and the 5G EVE portal located in a specific location.

In the first set of tests, the successful communication between the I/W Framework and the components located in the different sites is tested with simple ICMP tests.

In the second set of tests, it should be covered the communication between the I/W Framework and the 5G EVE Portal. The tests will be realised by utilising the Interworking API of the I/W Framework in order to communicate with the APIs defined in the 5G EVE Portal.

[9] https://ping.com/en-us/

Testing the Data Plane connectivity

In the data plane, however, and since the interworking requirements are also covered in WP3, testing of multi-site interconnections will be executed by conducting two types of tests: a) site connection tests; b) end-to-end site interconnection tests.

Site connection tests

The use of these tests will verify the connectivity in the data plane between a selected site and the Interworking Layer by using the different management IP addresses of the sites. To do this, it can be used whatever module in the I/W Layer (e.g. Runtime Configurator, Multi-Site NSO, etc.), doing *ping* tests towards one node inside each remote management network, and also verifying that the Portal is reachable.

End-to-end site interconnection tests

The use of these tests will verify the connectivity in the data plane across an end-to-end path traversing two selected sites (e.g. server placed in France and server located in Italy). The connection under test is defined as the successful communication (in IP layer) between both local components.

The test will be executed by using the *ping* tool on both components in order to test the connectivity between the two components.

The aforementioned set of tests will be repeated in order to cover all the possible site-to-site combinations, resulting in the execution of 6 set of tests.

3.1.2.2 Performance

Since the connectivity in 5G EVE is relying on the Internet, a single measurement will not be able to ensure the bandwidth nor the delay among sites when it is needed during an experiment.

Evaluating the performance of Orchestration Plane connectivity

In order to evaluate the performance of the connectivity between the I/W Framework and the different components located in different sites a set of testing scripts will be generated. These scripts will realise the successful configuration of the component located in the sites using the SBI of the I/W Framework. The main KPIs are a) the latency between the time a configuration request is generated in the I/W Layer and the time the successful response is sent back to it. The latency includes the transmission latency between the I/W Framework and components in site facilities, the processing time in the component, the time required for the realisation of the configuration command (using a script) on the NFV side and the transmission latency back to the I/W Framework; b) the throughput required for the functionality described above.

Note that the all the components placed in the I/W Framework with SBI towards Site facilities can be used for performing these tests, no matter what component is used. However, they all have different operations running over the same network, so in case of characterizing the operations for a specific component (e.g. Runtime Configurator), it should be done as well for all the other modules.

Evaluating the performance of Data Plane connectivity

As traffic will be traversing a best effort network, multiple measurements should be made to characterize the performance of the different connections. This is costly and still does not ensure the performance at key moments, but at least provides some information which may be meaningful.

The performance of the interconnection can be more or less delimitated (remember that it is not possible to validate the connectivity across Internet since it changes over time) by using two main KPIs: a) latency and specifically RTT (Round-trip Time) latency; b) throughput.

Latency

The latency KPI will be measured and validated by using the *ping* tool. A set of *ping* tests will be executed throughout a day in order to retrieve the average and maximum RTT latency (in ms) between two selected sites.

Throughput

The throughput KPI will be measured and validated using the *iperf*[10] tool. A set of *iperf* tests (including both TCP and UDP flows) will be executed throughout a day in order to retrieve the average and minimum throughput (in Gbps) and per flow type (TCP/UDP) between two selected sites.

The aforementioned performance tests will be executed between the outer nodes of the two sites (the servers in which the UPFs are located and through the N6 interfaces like the connectivity tests)[11]. Note that, if a site implements a WAN, the connection towards other sites may cross this WAN.

Similar to the connectivity tests, the performance tests will be repeated in order to cover all the possible site-to-site combinations.

As said, *ping* and a simple traffic generator like *iperf* should be enough to execute tests measuring delay and bandwidth, respectively. Therefore, it is recommended that performance tests are executed at least once a specific link between two sites is set up. Once the whole 5G EVE infrastructure is in production, it is then recommended to monitor these two parameters, similarly to the availability, by doing at least daily measurements.

3.2 Test suites definition and expected results

For defining the test suites for the identified testing points, it will be provided a table for each test case (identified with a unique ID), including the following fields:

- **Test purpose:** same as defined in Section 2.2.
- **Test type:** the type of test to which the test suite belongs. Mainly, the test defined will be functional tests, but other type of tests can be included if it is necessary.
- **Related test:** ID of related tests which are prerequisites for the execution of a given test case, if exist.
- **Priority:** a measure of the importance of the test, which can be used for specifying the order of test's execution or for contemplating different tests in the risk plan. Its value can be Low, Medium or High.
- **Execution time:** time spent for executing the test.
- **Configuration:** same as defined in Section 2.2.
- **References:** same as defined in Section 2.2.
- **Applicability:** same as defined in Section 2.2.
- **Pre-test conditions:** same as defined in Section 2.2.
- **Test Sequence:** same as defined in Section 2.2.
- **Test results:** what are the expected results after executing the test sequence.

These tables will be used for the definition of Interworking Framework Component tests and Interworking Framework System tests regarding the capabilities described in Section 3.1.2 (i.e. site facilities' interconnection).

3.2.1 Interworking Framework Component Tests

3.2.1.1 Multi-Site Catalogue

The Multi-site Catalogue tests are classified in the following categories:

- **NSD onboarding tests:**
    - Onboarding to single site – OSM.
    - Onboarding to single site – ONAP.
- **VNF onboarding tests:**

[10] https://iperf.fr/

[11] In this section (and also the previous one, for the site-to-site test), it must be decided whether tests are between the outer routers of the sites, or between the nodes implementing the UPF (wherever they may be located).

- Onboarding from per-site local orchestrator – OSM.
- Onboarding from per-site local orchestrator – ONAP.

- **PNFD onboarding tests:**
  - Onboarding from per-site local orchestrator – OSM.
  - Onboarding from per-site local orchestrator – ONAP.

As listed above, according to the current and planned status of the 5G EVE site facilities, in terms of per-site orchestrators and catalogues, the two types of deployed orchestrator tools at the time of writing are OSM and ONAP. Whenever new per-site orchestrator or catalogue tools will be used or deployed in the site facilities, the list of tests will be updated accordingly.

Moreover, the following test configurations are applicable for the Multi-site Catalogue, as depicted in Figure 4.

- MULTISITE_CAT_TO_OSM.
- MULTISTE_CAT_TO_ONAP.
- MULTISITE_CAT_TO_OSM_ONAP.

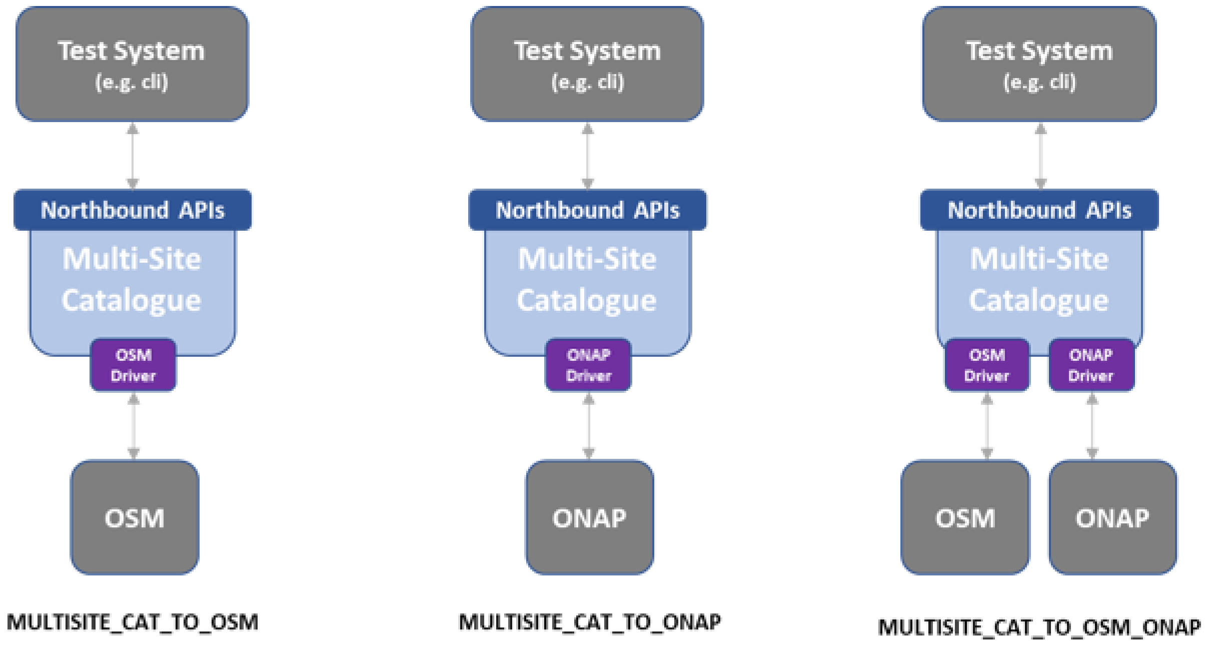

Figure 4: Multi-Site Catalogue test configurations

Table 2: Multi-Site Catalogue – NSD Management test suites

Multi-Site Catalogue – NSD Management	
Test 1 - NSD Onboarding to single site - OSM	Nsd-onboard-single-site-to-osm
Test purpose	This test aims at verifying that an NSD modelling a vertical experiment (in TOSCA format) can be successfully onboarded in the Multi-Site Catalogue from its NBI and delivered to a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.

Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue.
Test sequence	1. Trigger the onboarding of the NSD modelling vertical experiment in the Multi-Site Catalogue through its NBI, specifying the site where to onboard it. 2. Verify that a new NSD resource has been created in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully uploaded in the Multi-Site Catalogue. 4. Verify that a new NSD resource has been created in the OSM orchestrator. 5. Verify that the NSD content has been successfully translated and uploaded in the OSM orchestrator.
Expected results	The NSD modelling the vertical experiment is available in the Multi-Site Catalogue and is successfully onboarded in the catalogue of the OSM orchestrator
Test 2 - NSD Update to single site - OSM	Nsd-update-single-site-to-osm
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated in a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-osm.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the OSM orchestrator.
Test sequence	1. Trigger the update of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully updated in the Multi-site Catalogue. 3. Verify that the NSD content has been successfully translated and updated in the OSM orchestrator.
Expected results	The NSD modelling the vertical experiment is updated in the Multi-Site Catalogue and is successfully modified in the catalogue of the OSM orchestrator
Test 3 - NSD Deletion to single site - OSM	Nsd-delete-single-site-to-osm
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be deleted in the Multi-Site Catalogue from its NBI and removed from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-osm.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.

Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the OSM orchestrator.
Test sequence	1. Trigger the delete of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully deleted in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully deleted in the OSM orchestrator.
Expected results	The NSD modelling the vertical experiment is no more available in the Multi-Site Catalogue and is successfully deleted in the catalogue of the OSM orchestrator as well.

Test 4 - NSD Onboarding from single site - OSM	Nsd-onboard-single-site-from-osm
Test purpose	This test aims at verifying that an NSD modelling a single-site service can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance in a site facility is available and accessible to the Multi-Site Catalogue. • An NSD is available in the given site facility and is ready to be onboarded in the OSM catalogue.
Test sequence	1. Trigger the onboard of a new NSD in the local per-site facility OSM orchestrator. 2. Verify that the Multi-site Catalogue detects the new onboarded NSD in the OSM orchestrator. 3. Verify that a new NSD resource has been created in the Multi-Site Catalogue. 4. Verify that the NSD content has been successfully translated in TOSCA and uploaded in the Multi-Site Catalogue.
Expected results	The NSD modelling the single-site service is available in the Multi-Site Catalogue in TOSCA format.

Test 5 - NSD Update from single site - OSM	Nsd-update-single-site-from-osm
Test purpose	This test aims at verifying that an existing NSD modelling a single site service (previously onboarded from a per-site OSM orchestrator) can be modified in the Multi-Site Catalogue from its SBI when it is updated in the original per-site OSM orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-from-osm.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.

Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a single-site service is onboarded in the Multi-Site Catalogue and in the OSM orchestrator.
Test sequence	1. Trigger the update of an existing NSD in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the update to the existing NSD in the OSM orchestrator. 3. Verify that the updated NSD content has been successfully translated in TOSCA and uploaded in the Multi-Site Catalogue.
Expected results	The NSD modelling a single-site service is updated in the Multi-Site Catalogue in TOSCA format

Test 6 - NSD Deletion from single site - OSM	Nsd-delete-single-site-from-osm
Description	This test aims at verifying that an existing NSD modelling a per-site service (previously onboarded from a per-site OSM orchestrator) can be deleted in the Multi-Site Catalogue from its SBI when it is removed in the original per-site OSM orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-from-osm.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a per-site service is onboarded in the Multi-Site Catalogue and in the OSM orchestrator.
Steps	1. Trigger the delete of an existing NSD in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the deletion of the existing NSD in the OSM orchestrator. 3. Verify that the NSD content has been successfully removed in the Multi-Site Catalogue. 4. Verify that the NSD resource has been successfully removed in the Multi-Site Catalogue.
Expected results	The NSD modelling the vertical experiment is no more available in the Multi-Site Catalogue.

Test 7 - NSD Onboarding to single site - ONAP	**Nsd-onboard-single-site-to-onap**
Test purpose	This test aims at verifying that an NSD modelling a vertical experiment (in TOSCA format) can be successfully onboarded in the Multi-Site Catalogue from its NBI and delivered to a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue.

Test sequence	1. Trigger the onboarding of the NSD modelling vertical experiment in the Multi-Site Catalogue through its NBI. 2. Verify that a new NSD resource has been created in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully uploaded in the Multi-Site Catalogue. 4. Verify that a new NSD resource has been created in the ONAP orchestrator. 5. Verify that the NSD content has been successfully translated and uploaded in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is available in the Multi-Site Catalogue and is successfully onboarded in the catalogue of the ONAP orchestrator.
Test 8 - NSD Update from single site - ONAP	Nsd-update-single-site-to-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated in a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-onap.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the ONAP orchestrator.
Test sequence	1. Trigger the update of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully updated in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully translated and updated in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is updated in the Multi-Site Catalogue and is successfully modified in the catalogue of the ONAP orchestrator.
Test 9 - NSD Deletion to single site - ONAP	Nsd-delete-single-site-to-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be deleted in the Multi-Site Catalogue from its NBI and removed from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-onap.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the ONAP orchestrator.

Test sequence	1. Trigger the delete of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully deleted in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully deleted in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is no more available in the Multi-Site Catalogue and is successfully deleted in the catalogue of the OSM orchestrator as well.

Test 10 - NSD Onboarding from single site - ONAP	Nsd-onboard-single-site-from-onap
Test purpose	This test aims at verifying that an NSD modelling a single-site service can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance in a site facility is available and accessible to the Multi-Site Catalogue. • An NSD is available in the given site facility and is ready to be onboarded in the ONAP catalogue.
Test sequence	1. Trigger the onboard of a new NSD in the local per-site facility ONAP orchestrator. 2. Verify that the Multi-Site Catalogue detects the new onboarded NSD in the ONAP orchestrator. 3. Verify that a new NSD resource has been created in the Multi-Site Catalogue. 4. Verify that the NSD content has been successfully translated in TOSCA and uploaded in the Multi-Site Catalogue.
Expected results	The NSD modelling the single-site service is available in the Multi-Site Catalogue in TOSCA format

Test 11 - NSD Update from single site - ONAP	Nsd-update-single-site-from-onap
Test purpose	This test aims at verifying that an existing NSD modelling a single site service (previously onboarded from a per-site ONAP orchestrator) can be modified in the Multi-Site Catalogue from its SBI when it is updated in the original per-site ONAP orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-from-onap.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a single-site service is onboarded in the Multi-Site Catalogue and in the ONAP orchestrator.

Test sequence	1. Trigger the update of an existing NSD in the local site facility ONAP orchestrator. 2. Verify that the Multi-site Catalogue detects the update to the existing NSD in the ONAP orchestrator. 3. Verify that the updated NSD content has been successfully translated in TOSCA and uploaded in the Multi-Site Catalogue.
Expected results	The NSD modelling a single-site service is updated in the Multi-Site Catalogue in TOSCA format.

Test 12 - NSD Deletion from single site - ONAP	Nsd-delete-single-site-from-onap
Description	This test aims at verifying that an existing NSD modelling a per-site service (previously onboarded from a per-site ONAP orchestrator) can be deleted in the Multi-Site Catalogue from its SBI when it is removed in the original per-site ONAP orchestrator.
Test type	Functional.
Related tests	Nsd-onboard-single-site-from-onap.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A NSD modelling a per-site service is onboarded in the Multi-Site Catalogue and in the ONAP orchestrator.
Steps	1. Trigger the delete of an existing NSD in the local site facility ONAP orchestrator. 2. Verify that the Multi-site Catalogue detects the deletion of the existing NSD in the ONAP orchestrator. 3. Verify that the NSD content has been successfully removed in the Multi-Site Catalogue. 4. Verify that the NSD resource has been successfully removed in the Multi-Site Catalogue.
Expected results	The NSD modelling the vertical experiment is no more available in the Multi-Site Catalogue.

Test 13 - NSD Onboarding to multiple sites – OSM and ONAP	Nsd-onboard-multiple-site-to-osm-onap
Test purpose	This test aims at verifying that an NSD modelling a vertical experiment (in TOSCA format) can be successfully onboarded in the Multi-Site Catalogue from its NBI and delivered simultaneously to specific per-site OSM and ONAP orchestrators.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	700ms.
Configuration	MULTISITE_CAT_TO_OSM_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	An OSM instance and an ONAP instance are available and accessible to the Multi-Site Catalogue.

Test sequence	1. Trigger the onboarding of the NSD modelling vertical experiment in the Multi-Site Catalogue through its NBI, specifying the sites where to onboard it. 2. Verify that a new NSD resource has been created in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully uploaded in the Multi-Site Catalogue. 4. Verify that a new NSD resource has been created in the OSM orchestrator. 5. Verify that a new NSD resource has been created in the ONAP orchestrator. 6. Verify that the NSD content has been successfully translated and uploaded in the OSM orchestrator. 7. Verify that the NSD content has been successfully translated and uploaded in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is available in the Multi-Site Catalogue and is successfully onboarded in the catalogue of both the OSM and ONAP orchestrators.

Test 14 - NSD Update to multiple sites – OSM and ONAP	Nsd-update- multiple-site-to-osm-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated simultaneously in the specific per-site OSM and ONAP orchestrators.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-osm-onap.
Priority	Medium.
Execution time	700ms.
Configuration	MULTISITE_CAT_TO_OSM_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance and an ONAP instance are available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the OSM and ONAP orchestrators.
Test sequence	1. Trigger the update of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully updated in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully translated and updated in the OSM orchestrator. 4. Verify that the NSD content has been successfully translated and updated in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is updated in the Multi-Site Catalogue and is successfully modified in the catalogue of both the OSM and ONAP orchestrators.

Test 15 - NSD Deletion to multiple sites – OSM and ONAP	Nsd-delete- multiple-site-to-osm-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be deleted in the Multi-Site Catalogue from its NBI and removed simultaneously from the specific per-site OSM and ONAP orchestrators.
Test type	Functional.
Related tests	Nsd-onboard-single-site-to-osm-onap.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM_ONAP.

References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance and an ONAP instance are available and accessible to the Multi-Site Catalogue. • A NSD modelling a vertical experiment is onboarded in the Multi-Site Catalogue and in the OSM and ONAP orchestrators.
Test sequence	1. Trigger the delete of an existing NSD in the Multi-Site Catalogue through its NBI. 2. Verify that the NSD content has been successfully deleted in the Multi-Site Catalogue. 3. Verify that the NSD content has been successfully deleted in the OSM orchestrator. 4. Verify that the NSD content has been successfully deleted in the ONAP orchestrator.
Expected results	The NSD modelling the vertical experiment is no more available in the Multi-Site Catalogue and is successfully deleted in the catalogue of the OSM orchestrator as well.

Table 3: Multi-Site Catalogue – VNF Management test suites

Multi-Site Catalogue – VNF Management		
Test 1 - VNF Onboarding from site - OSM		vnf-onboard-from-osm
Test purpose	This test aims at verifying that a VNF can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.	
Test type	Functional.	
Related tests	None.	
Priority	High.	
Execution time	500ms.	
Configuration	MULTISITE_CAT_TO_OSM.	
References	NSD onboarding APIs – D3.3 [1], D3.6.	
Applicability	None.	
Pre-test conditions	• An OSM instance in a site facility is available and accessible to the Multi-Site Catalogue. • An VNF Package is available in the given site facility and is ready to be onboarded in the OSM catalogue.	
Test sequence	1. Trigger the onboard of a new VNF Package in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the new onboarded VNF package in the OSM orchestrator. 3. Verify that a new VNF package resource has been created in the Multi-Site Catalogue. 4. Verify that the content of VNF package (i.e. the VNF Descriptor) has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.	
Expected results	The VNF package is available in the Multi-Site Catalogue, with contents in TOSCA format.	
Test 2 - VNF Update from site - OSM		vnf-update-from-osm

Test purpose	This test aims at verifying that an existing VNF can be modified in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	vnf-onboard-single-site-from-osm.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A VNF Package of the given site is onboarded in both the Multi-Site Catalogue and the OSM orchestrator.
Test sequence	1. Trigger the update of an existing VNF Descriptor in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the update to the VNF Descriptor in the OSM orchestrator. 3. Verify that the updated VNF Descriptor has been successfully translated in TOSCA format and uploaded in the Multi-site Catalogue.
Expected results	The VNF Package is updated in the Multi-Site Catalogue and its content is available in TOSCA format.
Test 3 - VNF Deletion from site - OSM	vnf-delete-from-osm
Test purpose	This test aims at verifying that an existing VNF can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	vnf-onboard-single-site-from-osm.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A VNF Package of the given site is onboarded in both the Multi-Site Catalogue and the OSM orchestrator.
Test sequence	1. Trigger the deletion of an existing VNF Package in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the deletion to the VNF Package in the OSM orchestrator. 3. Verify that the updated VNF Descriptor has been successfully removed in the Multi-Site Catalogue.
Expected results	The VNF Package is no more available in the Multi-Site Catalogue.
Test 4 - VNF Onboarding from site - ONAP	vnf-onboard-from-onap
Test purpose	This test aims at verifying that a VNF can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test type	Functional.

Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance in a site facility is available and accessible to the Multi-Site Catalogue. • An VNF Package is available in the given site facility and is ready to be onboarded in the ONAP catalogue.
Test sequence	1. Trigger the onboard of a new VNF Package in the local site facility ONAP. 2. Verify that the Multi-site Catalogue detects the new onboarded VNF Package in the ONAP orchestrator. 3. Verify that a new VNF package resource has been created in the Multi-Site Catalogue. 4. Verify that the content of VNF package (i.e. the VNF Descriptor) has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.
Expected results	The VNF Package is available in the Multi-Site Catalogue, with contents in TOSCA format.

Test 5 - VNF Update from site - ONAP	vnf-update-from-onap
Test purpose	This test aims at verifying that an existing VNF can be modified in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	vnf-onboard-single-site-from-onap.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A VNF Package of the given site is onboarded in both the Multi-Site Catalogue and the ONAP orchestrator.
Test sequence	1. Trigger the update of an existing VNF Descriptor in the local site facility ONAP orchestrator. 2. Verify that the Multi-Site Catalogue detects the update to the VNF Descriptor in the ONAP orchestrator. 3. Verify that the updated VNF Descriptor has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.
Expected results	The VNF Package is updated in the Multi-Site Catalogue and its content is available in TOSCA format.

Test 6 - VNF Deletion from site - ONAP	vnf-delete-from-onap
Test purpose	This test aims at verifying that an existing VNF can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site ONAP orchestrator.
Test type	Functional.

Related tests	vnf-onboard-single-site-from-onap.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A VNF Package of the given site is onboarded in both the Multi-Site Catalogue and the ONAP orchestrator.
Test sequence	1. Trigger the deletion of an existing VNF Package in the local site facility ONAP orchestrator. 2. Verify that the Multi-Site Catalogue detects the deletion to the VNF Package in the ONAP orchestrator. 3. Verify that the updated VNF Descriptor has been successfully removed in the Multi-Site Catalogue.
Expected results	The VNF Package is no more available in the Multi-Site Catalogue

Table 4: Multi-Site Catalogue – PNFD Management test suites

Multi-Site Catalogue – PNFD Management	
Test 1 - PNFD Onboarding from site - OSM	pnfd-onboard-from-osm
Test purpose	This test aims at verifying that a PNFD can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance in a site facility is available and accessible to the Multi-Site Catalogue. • A PNFD is available in the given site facility and is ready to be onboarded in the OSM catalogue.
Test sequence	1. Trigger the onboard of a new PNFD in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the new onboarded PNFD in the OSM orchestrator. 3. Verify that a new VNF package resource has been created in the Multi-Site Catalogue. 4. Verify that the content of the PNFD has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.
Expected results	The PNFD is available in the Multi-Site Catalogue, with contents in TOSCA format.
Test 2 - PNFD Update from site - OSM	pnfd-update-from-osm
Test purpose	This test aims at verifying that an existing PNFD can be modified in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.

Test type	Functional.
Related tests	pnfd-onboard-single-site-from-osm.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A PNFD of the given site is onboarded in both the Multi-Site Catalogue and the OSM orchestrator.
Test sequence	1. Trigger the update of an existing PNFD in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the update to the PNFD in the OSM orchestrator. 3. Verify that the updated PNFD has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.
Expected results	The PNFD is updated in the Multi-Site Catalogue and its content is available in TOSCA format.

Test 3 - PNFD Deletion from site - OSM	pnfd-delete-from-osm
Test purpose	This test aims at verifying that an existing PNFD can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site OSM orchestrator.
Test type	Functional.
Related tests	pnfd-onboard-single-site-from-osm.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_OSM.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An OSM instance is available and accessible to the Multi-Site Catalogue. • A PNFD of the given site is onboarded in both the Multi-Site Catalogue and the OSM orchestrator.
Test sequence	1. Trigger the deletion of an existing PNFD in the local site facility OSM orchestrator. 2. Verify that the Multi-Site Catalogue detects the deletion to the PNFD in the OSM orchestrator. 3. Verify that the updated PNFD has been successfully removed in the Multi-Site Catalogue.
Expected results	The PNFD is no more available in the Multi-Site Catalogue.

Test 4 - PNFD Onboarding from site - ONAP	pnfd-onboard-from-onap
Test purpose	This test aims at verifying that a PNFD can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	500ms.

Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance in a site facility is available and accessible to the Multi-Site Catalogue. • A PNFD is available in the given site facility and is ready to be onboarded in the ONAP catalogue.
Test sequence	1. Trigger the onboard of a new PNFD in the local site facility ONAP. 2. Verify that the Multi-Site Catalogue detects the new onboarded PNFD in the ONAP orchestrator 3. Verify that a new PNFD resource has been created in the Multi-Site Catalogue. 4. Verify that the content of the PNFD has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue.
Expected results	The PNFD is available in the Multi-Site Catalogue, with contents in TOSCA format

Test 5 - PNFD Update from site - ONAP	pnfd-update-from-onap
Test purpose	This test aims at verifying that an existing PNFD can be modified in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	pnfd-onboard-single-site-from-onap.
Priority	Medium.
Execution time	500ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.
Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A PNFD of the given site is onboarded in both the Multi-Site Catalogue and the ONAP orchestrator.
Test sequence	1. Trigger the update of an existing PNFD in the local site facility ONAP orchestrator. 2. Verify that the Multi-Site Catalogue detects the update to the PNFD in the ONAP orchestrator. 3. Verify that the updated PNFD has been successfully translated in TOSCA format and uploaded in the Multi-Site Catalogue .
Expected results	The PNFD is updated in the Multi-Site Catalogue and its content is available in TOSCA format.

Test 6 - PNFD Deletion from site - ONAP	pnfd-delete-from-onap
Test purpose	This test aims at verifying that an existing VNF can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site ONAP orchestrator.
Test type	Functional.
Related tests	pnfd-onboard-single-site-from-onap.
Priority	High.
Execution time	250ms.
Configuration	MULTISITE_CAT_TO_ONAP.
References	NSD onboarding APIs – D3.3 [1], D3.6.

Applicability	None.
Pre-test conditions	• An ONAP instance is available and accessible to the Multi-Site Catalogue. • A PNFD of the given site is onboarded in both the Multi-Site Catalogue and the ONAP orchestrator.
Test sequence	1. Trigger the deletion of an existing PNFD in the local site facility ONAP orchestrator. 2. Verify that the Multi-Site Catalogue detects the deletion to the PNFD in the ONAP orchestrator. 3. Verify that the updated PNFD has been successfully removed in the Multi-Site Catalogue.
Expected results	The PNFD is no more available in the Multi-Site Catalogue.

3.2.1.2 Multi-Site Inventory

According to WP3 plans, the Multi-Site Network Orchestrator and the Multi-Site Inventory will initially handle just composite Network Services. Although this is the approach for the software delivered in deliverable D3.3 [1], it is included here the testing of VNF and PNF lifecycle management for covering any update in the I/W Framework in the future.

The following test configuration is applicable for the Multi-site Inventory, as depicted in Figure 5.

- MSNO_MSI_CONF.

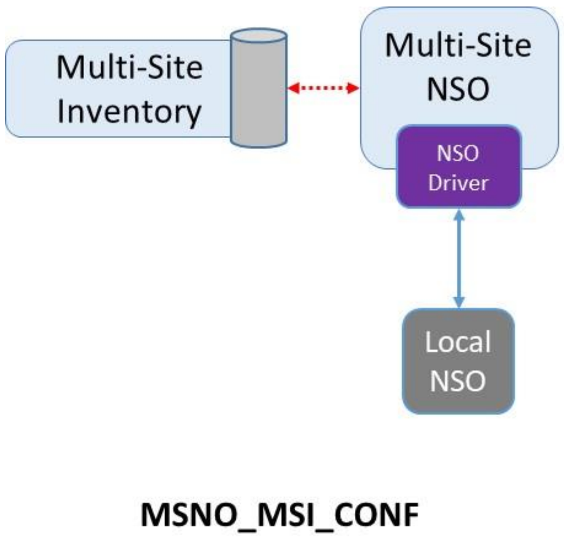

Figure 5: Multi-Site Inventory test configurations

Table 5: Multi-Site Inventory – Write Operation test suites

Multi-Site Inventory – Write Operation	
Test 1 – VNF Instance Lifecycle	write-vnfi-lifecycle
Test purpose	This test aims at validating the lifecycle management operations related to VNF Instances: instantiation, scaling, update, termination. Scaling and Update will depend on the necessity for these operations in 5G EVE.
Test type	Functional.
Related tests	None.
Priority	High.

Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. VNF lifecycle is governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory to Multi-Site NSO interface.
Pre-test conditions	• A Multi-Site NSO instance is available and with communication to the Multi-Site Inventory and at least to one local NFVO. • A testing VNF is on-boarded in the Multi-Site Catalogue and in the Multi-Site NSO. • There are local resources at the site to host the testing VNF.
Test sequence	1. The Multi-Site NSO instantiates a testing VNF in a local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “VNF instantiation” procedure at the MSI API. 2. The Multi-Site NSO scales the testing VNF; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “VNF scaling” procedure at the MSI API. 3. The Multi-Site NSO updates the testing VNF; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “VNF update” procedure at the MSI API. 4. The Multi-Site NSO terminates the testing VNF in the local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “VNF termination” procedure at the MSI API.
Expected results	1. A new VNF entry appears in the MSI, including the fields in the VNFI LoI, plus the ID of the site. 2. The VNF entry is modified with the information associated with the scaling procedure. 3. The VNF entry is modified with the information associated with the update procedure. 4. The VNF entry disappears from the MSI.
Test 2 – PNF Instance Lifecycle	write-pnfi-lifecycle
Test purpose	This test aims at validating the lifecycle management operations related to PNF Instances: instantiation, scaling, update, termination. Scaling and Update will depend on the necessity for these operations in 5G EVE, and also on the capabilities of the testing PNF.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. PNF lifecycle is governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory to Multi-Site NSO interface.
Pre-test conditions	• A Multi-Site NSO instance is available and with communication to the Multi-Site Inventory and at least to one local NFVO. • A testing PNF is deployed at the site.

Test sequence	1. The Multi-Site NSO instantiates the PNF in the local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "PNF instantiation" procedure at the MSI API. 2. The Multi-Site NSO scales the PNF; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "PNF scaling" procedure at the MSI API. 3. The Multi-Site NSO updates the PNF; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "PNF update" procedure at the MSI API. 5. The Multi-Site NSO terminates the PNF in the local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "PNF termination" procedure at the MSI API.
Expected results	1. A new PNF entry appears in the MSI, including the fields in the PNFI LoI, plus the ID of the site. 2. The PNF entry is modified with the information associated with the scaling procedure. 3. The PNF entry is modified with the information associated with the update procedure. 4. The PNF entry disappears from the MSI.
Test 3 – NS Instance Lifecycle	write-nsi-lifecycle
Test purpose	This test aims at validating the lifecycle management operations related to NS Instances: instantiation, scaling, update, termination.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. NS lifecycle is governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory to Multi-Site NSO interface.
Pre-test conditions	• A Multi-Site NSO instance is available and with communication to the Multi-Site Inventory and at least to one local NFVO. • A testing NS is on-boarded in the Multi-Site Catalogue and in the Multi-Site NSO. • There are local resources at the site to host the testing NS.
Test sequence	1. The Multi-Site NSO instantiates the testing NS in a local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "NS instantiation" procedure at the MSI API. 2. The Multi-Site NSO scales the testing NS; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "NS scaling" procedure at the MSI API. 3. The Multi-Site NSO updates the testing NS; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "NS update" procedure at the MSI API. 4. The Multi-Site NSO terminates the testing NS in the local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the "NS termination" procedure at the MSI API.
Expected results	1. A new NS entry appears in the MSI, including the fields in the NSI LoI, plus the ID of the site. 2. The NS entry is modified with the information associated with the scaling procedure.

	3. The NS entry is modified with the information associated with the update procedure. 4. The NS entry disappears from the MSI.
Test 4 – Composite NS Instance Lifecycle	write-comp-nsi-lifecycle
Test purpose	This test aims at validating the lifecycle management operations related to composite NS Instances: instantiation, scaling, update, termination.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. NS lifecycle is governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory to Multi-Site NSO interface.
Pre-test conditions	• A Multi-Site NSO instance is available and with communication to the Multi-Site Inventory and at least to two local NFVOs. • A testing composite NS is on-boarded in the Multi-Site Catalogue and in the Multi-Site NSO, using CreateNSRequest operation. • There are local resources at the sites to host the testing composite NS.
Test sequence	1. The Multi-Site NSO instantiates the testing composite NS in the two local sites; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “composite NS instantiation” procedure at the MSI API. 2. The Multi-Site NSO scales the testing composite NS; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “composite NS scaling” procedure at the MSI API. 3. The Multi-Site NSO updates the testing composite NS; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “composite NS update” procedure at the MSI API. 4. The Multi-Site NSO terminates the testing composite NS in the local site; it must be checked that after this operation is successful, the Multi-Site NSO triggers the “composite NS termination” procedure at the MSI API.
Expected results	1. A new composite NS entry appears in the MSI, including the fields in the NSI LoI, plus the IDs of the sites. 2. The composite NS entry is modified with the information associated with the scaling procedure. 3. The composite NS entry is modified with the information associated with the update procedure. 4. The composite NS entry disappears from the MSI.

Table 6: Multi-Site Inventory – Query Operation test suites

Multi-Site Inventory – Query Operation	
Test 1 – VNF Instance Management	query-vnfi-status
Test purpose	This test aims at validating the part of the MSI NBI that deals with the retrieval of specific VNF Instance(s) information.
Test type	Functional.
Related tests	None.

Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. Attributes included in the VNF LoI are governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory NBI.
Pre-test conditions	• A 5G EVE Experimentation Portal instance is available and with communication to the MSI. • There are at least two VNFI entries in the MSI.
Test sequence	1. The 5G EVE Experimentation Portal requests information about a specific VNFI from the MSI, based on the VNFI ID. 2. The 5G EVE Experimentation Portal requests information about two VNFIs from the MSI, based on the VNFI IDs.
Expected results	1. The MSI responses with the information included in the VNFI LoI for the requested VNFI ID. 2. The MSI responses with the information included in the two VNFIs LoI for the requested VNFI IDs.
Test 2 – PNF Instance Management	query-pnfi-status
Test purpose	This test aims at validating the part of the MSI NBI that deals with the retrieval of specific PNF Instance(s) information.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. Attributes included in the PNF LoI are governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory NBI.
Pre-test conditions	• A 5G EVE Experimentation Portal instance is available and with communication to the MSI. • There are at least two PNFI entries in the MSI.
Test sequence	1. The 5G EVE Experimentation Portal requests information about a specific PNFI from the MSI, based on the PNFI ID. 2. The 5G EVE Experimentation Portal requests information about two PNFIs from the MSI, based on the PNFI IDs.
Expected results	1. The MSI responses with the information included in the PNFI LoI for the requested PNFI ID. 2. The MSI responses with the information included in the two PNFIs LoI for the requested PNFI IDs.
Test 3 – NS Instance Management	query-nsi-status
Test purpose	This test aims at validating the part of the MSI NBI that deals with the retrieval of specific NS Instance(s) information.
Test type	Functional.

Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. Attributes included in the NS LoI are governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory NBI
Pre-test conditions	• A 5G EVE Experimentation Portal instance is available and with communication to the MSI. • There are at least two NSI entries in the MSI.
Test sequence	1. The 5G EVE Experimentation Portal requests information about a specific NSI from the MSI, based on the NSI ID. 2. The 5G EVE Experimentation Portal requests information about two NSIs from the MSI, based on the NSI IDs.
Expected results	1. The MSI responses with the information included in the NSI LoI for the requested NSI ID. 2. The MSI responses with the information included in the two NSIs LoI for the requested NSI IDs.
Test 4 – Infrastructure Management	query-infrast-status
Test purpose	This test aims at validating the part of the MSI NBI that deals with the retrieval of the full 5G EVE active infrastructure information.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_MSI_CONF.
References	Operations over inventories/databases can follow multiple standards; specific references may be provided depending on the implementation decision. Attributes included in the VNF, PNF and NS LoIs are governed by ETSI NFV SOL005.
Applicability	Multi-Site Inventory NBI.
Pre-test conditions	• A 5G EVE Administration Portal or a Multi-Site NSO instance is available and with communication to the MSI. • There are at least two VNFI, two PNFI and two NSI entries in the MSI.
Test sequence	1. The 5G EVE Administration Portal or the Multi-Site NSO requests the full information stored at the MSI.
Expected results	The MSI responses with the list of all stored Instances, providing for each Instance the information included in its LoI.

3.2.1.3 Multi-Site Network Orchestrator

The following configuration environment is applicable for the MSNO testing, described in Figure 6.

- MSNO_BASIC_CONF.

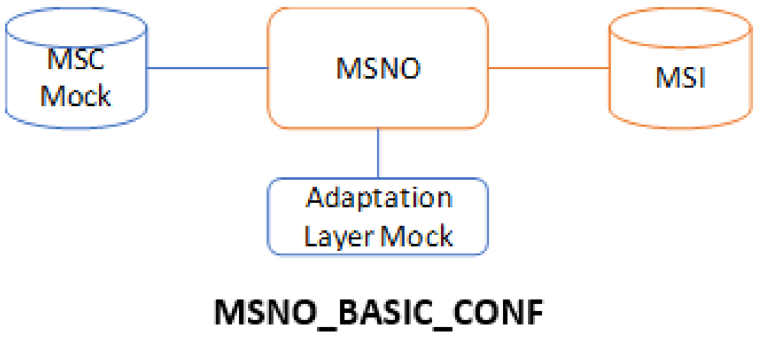

Figure 6: Multi-Site Network Orchestrator test configurations

Table 7: Multi-Site Network Orchestrator test suites

Multi-Site Network Orchestrator	
Test 1 – Network Service ID creation	ms-nso-id-creation
Test purpose	The purpose of this test is to validate the creation of a NS ID.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_BASIC_CONF.
References	NS LCM API.
Applicability	None.
Pre-test conditions	• NSD is onboarded in the Multi-Site Catalogue. • Nested NS are onboarded in each site NFVO.
Test sequence	1. Send a CreateNSRequest to the MSNO.
Expected results	• MSNO returns a positive response for the NS creation. • A notification is sent when NS creations process finishes. • NS ID is stored in the DB with NOT_INSTANTIATED status and can be queried using external MSI API.
Test 2 – Network Service instantiation	ms-nso-ns-inst
Test purpose	The purpose of this test is to validate the instantiation of a NS.
Test type	Functional.
Related tests	ms-nso-id-creation.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_BASIC_CONF.
References	NS LCM API.
Applicability	None.
Pre-test conditions	• NSD is onboarded in the Multi-Site Catalogue. • Nested NS are onboarded in each site NFVO. • NS ID is created (see ms-nso-id-creation test).

Test sequence	1. Send a instantiate request to the MSNO.
Expected results	1. MSNO returns a positive response for the instantiation. 2. A notification is sent when NS creations process starts and finish. 3. NS ID is stored in the DB with INSTANTIATED status and can be queried using external MS Inventory API. 4. Each nested NS is instantiated in each site NFVO.
Test 3 – Network Service Lifecycle Management	ms-nso-ns-lcm
Test purpose	The purpose of this test is to validate the management of the NS Lifecycle.
Test type	Functional.
Related tests	ms-nso-id-creation, ms-nso-ns-inst.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_BASIC_CONF.
References	NS LCM API.
Applicability	None.
Pre-test conditions	• NSD is onboarded in the Multi-Site Catalogue. • Nested NS are onboarded in each site NFVO. • NS ID is created (see ms-nso-id-creation test). • NS is instantiated.
Test sequence	1. Send a scale request to the MSNO. 2. Send an update request to the MSNO. 3. Send a heal request to the MSNO. 4. Send a terminate request to the MSNO.
Expected results	1. MSNO returns an Operation ID for each of the requests. 2. The NS is scaled, updated, healed and terminated correctly.
Test 4 – NS Lifecycle management operations	ms-nso-ns-lcm-op
Test purpose	The purpose of this test is to validate the management of the NS Lifecycle operations.
Test type	Functional.
Related tests	ms-nso-id-creation, ms-nso-ns-inst, ms-nso-ns-lcm.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_BASIC_CONF.
References	NS LCM API.
Applicability	None.
Pre-test conditions	• NSD is onboarded in the Multi-Site Catalogue. • Nested NS are onboarded in each site NFVO. • NS ID is created (see ms-nso-id-creation test). • NS is instantiated. • NS has not enough resources for scaling.
Test sequence	1. Send a scale request to the MSNO. 2. Send a continue operation when the previous one failed. 3. Send a retry operation when the previous one failed. 4. Send a fail operation when the previous one failed. 5. Send a scale request to the MSNO. 6. Send a rollback.

	7. Send a cancel operation.
Expected results	1. MSNO returns an Operation ID for each of the requests. 2. The scale request fails, as well as the continue and retry operations. 3. The rollback operation is cancelled.

Test 5 – Network Service Lifecycle Management notifications	ms-nso-ns-lcm-not
Test purpose	The purpose of this test is to validate the MSNO notifications.
Test type	Functional.
Related tests	ms-nso-id-creation, ms-nso-ns-inst.
Priority	High.
Execution time	As required for the full test execution.
Configuration	MSNO_BASIC_CONF.
References	NS LCM API.
Applicability	None.
Pre-test conditions	• NSD is onboarded in the Multi-Site Catalogue. • Nested NS are onboarded in each site NFVO. • NS ID is created (see ms-nso-id-creation test).
Test sequence	1. Send a subscription request to the MSNO. 2. Send an instantiate request to the MSNO.
Expected results	• MSNO sends a start notification. • MSNO sends result notification.

3.2.1.4 Data Collection Manager

There are four main test configurations that are used for the Data Collection Manager test suites, which are the following:

- DCM_PUBLISH_SUBSCRBE.
- DCM_LOAD_TESTS.
- DCM_NBI_TESTS (mainly focused on the interaction between Kafka[12] and ELK[13]).
- DCM_SBI_TESTS (mainly focused on the interaction between Kafka and Filebeat[14]).

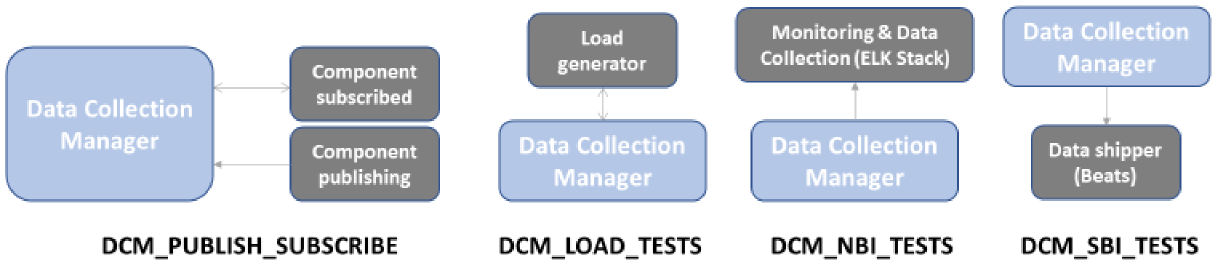

Figure 7: Data Collection Manager test configurations

[12] https://kafka.apache.org/

[13] https://www.elastic.co/es/what-is/elk-stack

[14] https://www.elastic.co/es/products/beats/filebeat

Table 8: Data Collection Manager – Kafka test suites

Data Collection Manager – Kafka	
Test 1 – Kafka cluster setup	dcm-cluster-setup
Test purpose	This test aims at validating the Kafka cluster availability, so that the rest of depending services and functionalities can make use of it.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	3 minutes.
Configuration	DCM_PUBLISH_SUBSCRIBE.
References	Data Collection Manager specification – D3.3 [1].
Applicability	None.
Pre-test conditions	• Apache Kafka is installed in the nodes that belong to the cluster. • Apache ZooKeeper[15] is installed for distributed process coordination within the cluster. • Kafka Confluent[16] is installed for managing the cluster.
Test sequence	1. Execute the necessary commands to setup the Kafka cluster.
Expected results	• The process is correctly started, according to the logs provided by Kafka. • ZooKeeper starts working with the cluster data coordination tasks. • Kafka Confluent is available, and from its interface, the cluster status can be also monitored.
Test 2 – Subscribe operation	dcm-subscribe
Test purpose	This test intends to emulate the subscribe operation defined in the Data Collection Manager, which involves the creation of a topic and the setup of a process in the Kafka cluster which starts listening to that topic.
Test type	Functional.
Related tests	dcm-cluster-setup.
Priority	High.
Execution time	1 minute.
Configuration	DCM_PUBLISH_SUBSCRIBE.
References	Data Collection Manager specification – D3.3 [1].
Applicability	None.
Pre-test conditions	• Kafka cluster is setup and running. • A topic name is chosen beforehand.
Test sequence	1. Send a subscribe operation to the Data Collection Manager, including the topic name as parameter.
Expected results	• The topic is created in the Data Collection Manager. • It has been created a process in the Kafka cluster that is consuming from that topic.

[15] https://zookeeper.apache.org/

[16] https://www.confluent.io/

Test 3 – Publish operation	dcm-publish
Test purpose	The purpose of this test is to verify that the Data Collection Manager is able to deliver the data published in the Kafka bus to the components subscribed to a given topic.
Test type	Functional.
Related tests	dcm-cluster-setup, dcm-subscribe.
Priority	High.
Execution time	2 minutes.
Configuration	DCM_PUBLISH_SUBSCRIBE.
References	Data Collection Manager specification – D3.3 [1].
Applicability	A component subscribed to the topic used during this test must be present for receiving the data from the Kafka bus.
Pre-test conditions	• Kafka cluster is setup and running. • A topic name is chosen beforehand. • A subscription process to that topic is running in the Kafka cluster.
Test sequence	1. Send a publish operation to the Data Collection Manager, including the topic name and some data (a JSON string) as parameters.
Expected results	• Data is correctly published in the Kafka cluster. • Data is automatically delivered to the subscribed component.
Test 4 – Unsubscribe operation	dcm-unsubscribe
Test purpose	This test performs the unsubscribe operation defined in the Data Collection Manager, in order to test the successful withdrawal of a given topic.
Test type	Functional.
Related tests	dcm-cluster-setup, dcm-subscribe.
Priority	Medium.
Execution time	1 minute.
Configuration	DCM_PUBLISH_SUBSCRIBE.
References	Data Collection Manager specification – D3.3 [1].
Applicability	None.
Pre-test conditions	• Kafka cluster is setup and running. • A topic name is chosen beforehand. • A subscription process to that topic is running in the Kafka cluster.
Test sequence	1. Send an unsubscribe operation to the Data Collection Manager, including the topic name as parameter.
Expected results	• It has been deleted the process that was consuming from the selected topic in the Kafka cluster. • That topic is deleted in the Data Collection Manager.
Test 5 – Stress test	dcm-stress
Test purpose	As the Data Collection Manager is a central component in the 5G EVE architecture, it is needed to verify its correct performance in terms of scalability. For that reason, it is defined this test in order to execute a stress test for validating that the Kafka cluster is able to manage a huge amount of data and topics concurrently.
Test type	Load.
Related tests	dcm-cluster-setup, dcm-subscribe.

Priority	Medium.
Execution time	1 hour.
Configuration	DCM_LOAD_TESTS.
References	Data Collection Manager specification – D3.3 [1].
Applicability	A load generator (e.g. Berserker[17]) is needed to send data to the Kafka cluster.
Pre-test conditions	• Kafka cluster is setup and running. • Subscription process has been executed for all the topics used in this test (120 in total).
Test sequence	For each topic used in this test: 1. Every 30 seconds launch the load generator configured with wanted load rate and message size for start publishing in a different topic. 2. Capture meaningful data using OS built-in tools in order to monitor the status of the server and the Kafka cluster.
Expected results	• Kafka cluster is able to manage all the data generated during the test. • Kafka cluster performance evolution during the test does not suffer important drops that may compromise the correct behaviour of the Data Collection Manager in a production environment.

Table 9: Data Collection Manager – NBI test suites

Data Collection Manager – Kafka-ELK interconnection – NBI	
Test 1 – Kafka-Elasticsearch configuration	dcm-nbi-elk-config
Test purpose	This test intends to verify that Elasticsearch component from the ELK stack is configured for receiving data from Kafka.
Test type	Functional.
Related tests	dcm-cluster-setup.
Priority	Medium.
Execution time	2 minutes.
Configuration	DCM_NBI_TESTS.
References	Data Collection Manager specification – D3.3 [1], Monitoring and Data Collection tools – D4.1 [7].
Applicability	None.
Pre-test conditions	• Kafka is deployed. • Elasticsearch is deployed. • There is connectivity between Kafka and Elasticsearch.
Test sequence	1. Modify the Elasticsearch configuration file and include Kafka information. 2. Run Kafka and start consuming from the topic defined in Elasticsearch configuration. 3. Write something to that topic. 4. Check in Kibana that Elasticsearch has received data.
Expected results	Elasticsearch is connected to Kafka and data has been received.

[17] https://github.com/smartcat-labs/berserker

Table 10: Data Collection Manager – SBI test suites

Data Collection Manager – Kafka-Filebeat interconnection - SBI	
Test 1 – Kafka-Beats service availability	dcm-sbi-filebeat-status
Test purpose	This test is intended to verify that Filebeat service is connected to the Kafka cluster.
Test type	Functional.
Related tests	dcm-cluster-setup.
Priority	Medium.
Execution time	2 minutes.
Configuration	DCM_SBI_TESTS.
References	Data Collection Manager specification – D3.3 [1], Monitoring and Data Collection tools – D4.1 [7].
Applicability	None.
Pre-test conditions	• Kafka is deployed. • Filebeat service is reported to be up and configured to be persistent at startup. • There is connectivity between Filebeat and Kafka.
Test sequence	1. Modify the Filebeat configuration file and include Kafka information. 2. Run Kafka and start consuming from the topic defined in Filebeat configuration. 3. Start Filebeat process in order to send the information to the Kafka bus. 4. Check in Kafka logs that data is being received in the selected topic.
Expected results	Data is successfully sent from Filebeat to Kafka by using the selected topic.
Test 2 - File change event leads to Kafka message	dcm-sbi-filebeat-event
Test purpose	The purpose of this test is to verify that that Filebeat is able to react to a file change event and inject a message into the Kafka bus.
Test type	Functional.
Related tests	dcm-cluster-setup, dcm-sbi-filebeat-status.
Priority	Medium.
Execution time	2 minutes.
Configuration	DCM_SBI_TESTS.
References	Data Collection Manager specification – D3.3 [1], Monitoring and Data Collection tools – D4.1 [7].
Applicability	None.
Pre-test conditions	• Kafka is deployed. • Filebeat up and running. • Filebeat service is reported to be up and configured to be persistent at startup. • There is connectivity between Filebeat and Kafka.
Test sequence	1. Make changes in the file that is being managed by Filebeat. 2. Check that a message to the Kafka bus is automatically published.
Expected results	Data is successfully sent from Filebeat to Kafka by using the selected topic.

3.2.1.5 Runtime Configurator

There are two main test configurations that are used for the Data Collection Manager test suites, which are the following:

- RC_NBI_CONF.
- RC_SBI_CONF.
- RC_COMPLETE_CONF.

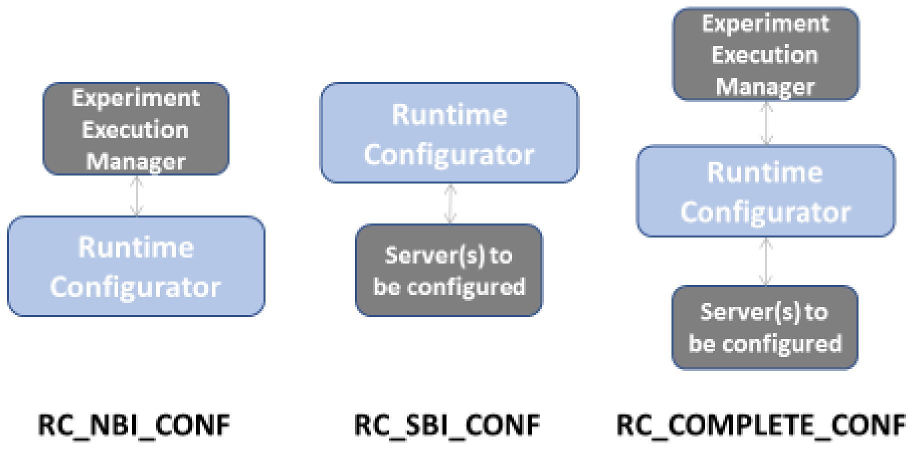
Experiment Execution Manager
Runtime Configurator
RC_NBI_CONF
Runtime Configurator
Server(s) to be configured
RC_SBI_CONF
Experiment Execution Manager
Runtime Configurator
Server(s) to be configured
RC_COMPLETE_CONF

Figure 8: Runtime Configurator test configurations

Table 11: Runtime Configurator test suites

Runtime Configurator		
Test 1 – NBI Interface		rc-nbi-if
Test purpose	The objective of this test is to verify that the Runtime Configurator is reachable by the Experiment Execution Manager (simulated with a fake-client based in Robot Framework), trying to execute a Robot Framework script related to the execution of an Ansible[18] default playbook (print "Hello World").	
Test type	Functional.	
Related tests	None.	
Priority	High.	
Execution time	As required for the full test execution.	
Configuration	RC_NBI_CONF.	
References	Runtime Configurator specification – D3.3 [1], based on SSH NBI interface.	
Applicability	None.	
Pre-test conditions	• An instance of the Runtime Configurator is deployed and available. • There exist Robot Framework testing scripts in order to interact with Ansible. • Ansible has a playbook which prints "Hello World" in the terminal.	
Test sequence	1. Build the Robot Framework script in order to execute the "Hello World" playbook. 2. Upload the playbook in Ansible. 3. Execute the Robot Framework script.	
Expected results	"Hello World" message must be printed.	

[18] https://docs.ansible.com/ansible/latest/index.html

Test 2 – SBI Interface	rc-sbi-if
Test purpose	This test intends to test the SBI between the Runtime Configurator and the VNFs/PNFs to be configured with Day-2 configurations.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	RC_SBI_CONF.
References	Runtime Configurator specification – D3.3 [1], based on SSH SBI interface.
Applicability	None.
Pre-test conditions	• An instance of the Runtime Configurator is deployed and available. • There are some virtual machines that can emulate VNFs/PNFs. • These virtual machines are reachable from Runtime Configurator and have OpenSSH server installed and enabled. • Ansible has a playbook which execute several commands in the VNFs/PNFs.
Test sequence	1. Configure hosts and group_vars files with the variables and parameters needed to reach the virtual machines. 2. Execute any Ansible playbook for testing commands in the virtual machines.
Expected results	• The Runtime Configurator is able to reach the machines to be configured. • The machines are configured with the commands provided in the playbook.
Test 3 – Complete workflow	rc-complete
Test purpose	This test composes the two previous tests. As a result, Experiment Execution Manager executes some templates of the Runtime Configurator, and the commands included there will configure several VNFs/PNFs.
Test type	Functional.
Related tests	rc-nbi-if, rc-sbi-if.
Priority	High.
Execution time	As required for the full test execution.
Configuration	RC_COMPLETE_CONF.
References	Runtime Configurator specification – D3.3 [1], based on SSH NBI and SBI interfaces.
Applicability	None.
Pre-test conditions	• An instance of the Runtime Configurator is deployed and available. • There are some virtual machines that can emulate VNFs/PNFs. • These virtual machines are reachable from Runtime Configurator and have OpenSSH server installed and enabled. • There exist Robot Framework testing scripts in order to interact with Ansible. • Ansible has a playbook which execute several commands in the VNFs/PNFs.
Test sequence	1. Build the Robot Framework script in order to execute the playbook. 2. Upload the playbook in Ansible. 3. Configure hosts and group_vars files with the variables and parameters needed to reach the virtual machines. 4. Execute the Robot Framework script.
Expected results	• Experiment Execution Manager is able to reach the Runtime Configurator. • Runtime Configurator is able to reach the VNFs/PNFs.

	• The configuration is applied in the VNFs/PNFs after starting the workflow from the Experiment Execution Manager.

3.2.1.6 Adaptation Layer: Multi-Site NSO to local Orchestrators interface

The MSO-LO interface tests are classified in the following categories:

- **NFVO information unit tests.**
- **Subscription for notifications unit test.**
- **Local NFVO driver unit test:**
  - OSM driver unit test.
  - ONAP driver unit test.

At the moment, tests for OSM and ONAP are provided, as they are the two main orchestration technologies used by the trial sites involved in 5G EVE. If new NFVOs are integrated in the future, the test set will be extended as a result.

The following test configurations are applicable for the MSO-LO, as depicted in Figure 9.

- MSO_LO_OSM_MOCK.
- MSO_LO_ONAP_MOCK.
- MSO_LO_SQLITE_MOCK.

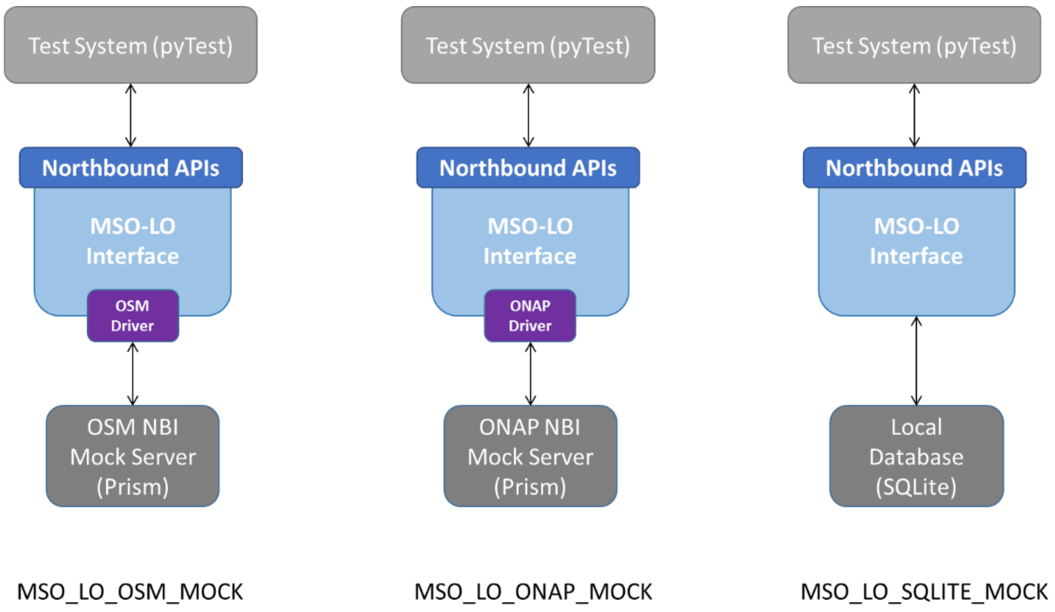

Figure 9: MSO-LO test configurations

Table 12: Multi-Site NSO to local Orchestrators interface test suites

Multi-Site NSO to local Orchestrators interface (MSO-LO)	
Test 1 – NFVO information unit test	unit-Mso-Lo-nfvo
Test purpose	Assert the correct retrieval of NFVO information from local database.
Test type	Functional.
Related tests	None.
Priority	High.

Execution time	1-2 seconds.
Configuration	MSO_LO_SQLITE_MOCK
References	MSO-LO Interface OpenAPI – D3.3 [1].
Applicability	None.
Pre-test conditions	• Creation and population with mock data of an SQLite database file.
Test sequence	1. Check status codes 200, 400, headers and payload for GET /nfvo. 2. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}.
Expected results	All assertions verified, no errors.

Test 2 – Subscription for notifications unit test	unit-Mso-Lo-subscriptions
Test purpose	Assert the correct retrieval and creation of subscriptions for notifications about the NS instance status. Local database stores the information.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	1-2 seconds.
Configuration	MSO_LO_SQLITE_MOCK
References	MSO-LO Interface OpenAPI – D3.3 [1].
Applicability	None.
Pre-test conditions	• Creation and population with mock data of an SQLite database file.
Test sequence	1. Check status codes 201, 303, 400, headers and payload for POST /nfvo/{nfvoId}/subscriptions. 2. Check status codes 200, 400, headers and payload for GET /nfvo/{nfvoId}/subscriptions. 3. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}/subscriptions/{subscriptionId}. 4. Check status codes 204, 400, 404, headers and payload for DELETE /nfvo/{nfvoId}/subscriptions/{subscriptionId}.
Expected results	All assertions verified, no errors.

Test 3 – OSM driver unit test	unit-Mso-Lo-osm
Test purpose	Assert that the request bodies sent to OSM NBI interface are correct. Assert that responses from OSM are handled correctly. Assert that responses returned by Mso-Lo API are correct. OSM server is mocked with Prism and the OpenAPI YAML specification of OSM NBI.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	5-10 seconds.
Configuration	MSO_LO_OSM_MOCK
References	MSO-LO Interface OpenAPI – D3.3 [1].
Applicability	None.
Pre-test conditions	• Prism mock server running with OSM NBI OpenAPI specification.
Test sequence	1. Check status codes 201, 400, headers and payload for POST /nfvo/{nfvoId}/ns_instances. 2. Check status codes 200, 400, headers and payload for GET /nfvo/{nfvoId}/ns_instances. 3. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}/ns_instances/{nsInstanceId}.

	4. Check status codes 204, 400, 404, headers and payload for DELETE /nfvo/{nfvoId}/ns_instances/{nsInstanceId}. 5. Check status codes 202, 400, 404, headers and payload for POST /nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate. 6. Check status codes 202, 400, headers and payload for POST /nfvo/{nfvoId}/ns_instances/{nsInstanceId}/scale. 7. Check status codes 202, 400, 404, headers and payload for POST /nfvo/{nfvoId}/ns_instance/{nsInstanceId}/terminate. 8. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}/vnf/{id}. 9. Check status codes 200, 400, headers and payload for GET /nfvo/{nfvoId}/vnf.
Expected results	All assertions verified, no errors.
Test 4 – ONAP driver unit test	unit-Mso-Lo-onap
Test purpose	Assert that the request bodies sent to ONAP NBI interface are correct. Assert that responses from ONAP are handled correctly. Assert that responses returned by Mso-Lo API are correct. ONAP server is mocked with Prism and the OpenAPI specification of ONAP NBI.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	5-10 seconds.
Configuration	MSO_LO_ONAP_MOCK
References	MSO-LO Interface OpenAPI – D3.3 [1].
Applicability	None.
Pre-test conditions	• Prism mock server running with ONAP NBI OpenAPI specification.
Test sequence	1. Check status codes 201, 400, headers and payload for POST /nfvo/{nfvoId}/ns_instances. 2. Check status codes 200, 400, headers and payload for GET /nfvo/{nfvoId}/ns_instances. 3. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}/ns_instances/{nsInstanceId}. 4. Check status codes 204, 400, 404, headers and payload for DELETE /nfvo/{nfvoId}/ns_instances/{nsInstanceId}. 5. Check status codes 202, 400, 404, headers and payload for POST /nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate. 6. Check status codes 202, 400, headers and payload for POST /nfvo/{nfvoId}/ns_instances/{nsInstanceId}/scale. 7. Check status codes 202, 400, 404, headers and payload for POST /nfvo/{nfvoId}/ns_instance/{nsInstanceId}/terminate. 8. Check status codes 200, 400, 404, headers and payload for GET /nfvo/{nfvoId}/vnf/{id}. 9. Check status codes 200, 400, headers and payload for GET /nfvo/{nfvoId}/vnf.
Expected results	All assertions verified, no errors.

3.2.2 Integration Tests between Interworking Framework components

The integration tests are required for some integration points in order to verify that the I/W Framework is working as expected. The following integration points are identified, as presented in section 2.3.2:

- Multi-Site Network Orchestrator and Multi-Site Catalogue: integration using NSD management API.
- Multi-Site Catalogue and Multi-Site Inventory.
- Multi-Site Network Orchestrator, Multi-Site Catalogue, Data Collection Manager and Runtime Configurator with the Adaptation Layer.
- Drivers of Adaptation Layer with their respective NFVO.

For the integration testing, the same testing defined in the previous sections will be used, with different environment that incorporates real components. In case of needing to contemplate new possible integrations, they will be further described in the next D3.7 deliverable.

3.2.3 Interworking Layer System Tests

Following the same approach described in the integration testing, for the system tests it is also proposed to reuse the same testing described in the functional test but having an environment with all I/W Framework components in place.

Additionally, it is proposed in the following section a set of tests for validating the inter-site connectivity.

3.2.3.1 Site facilities' interconnection

The following test configurations are applicable for the Site facilities interconnection, as depicted in Figure 10.

- ORCH_PLANE_CONF.
- ALL_PLANE_CONF.
- PORTAL_CONF.

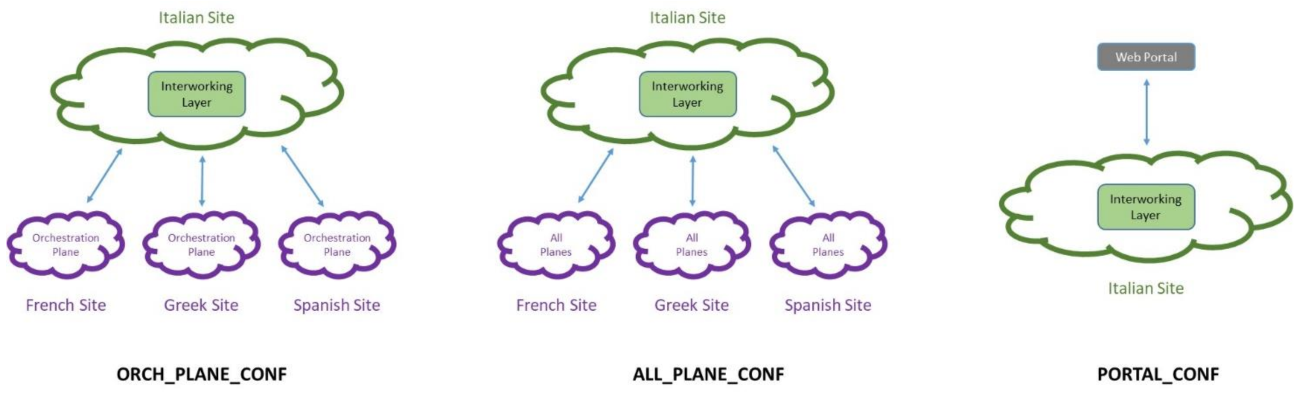

Figure 10: Site facilities' interconnection test configurations

Table 13: Site facilities' interconnection test suites

Site facilities' interconnection		
Test 1 – Connectivity between sites and I/W Framework		facilities-connect-sites
Test purpose	This test aims at ensuring that all sites are reachable from the Turin site, where the I/W Framework is to be installed. If desired from any other site, the operation would be totally equivalent.	
Test type	Functional.	
Related tests	None.	
Priority	High.	
Execution time	As required for the full test execution.	
Configuration	ORCH_PLANE_CONF.	
References	The tool must support RFC792 (Internet Control Message Protocol), plus its updates.	
Applicability	Orchestration, Control or Data Planes Interworking (depending on the plane the devices are deployed).	
Pre-test conditions	• One device at the Turin site capable of generating ICMP traffic, emulating I/W Framework modules. • At least one device at each of the other sites capable of responding to ICMP requests, emulating for example, the local NFVOs.	

Test sequence	1. ICMP echo requests are sent from the device in Turin to the device in Athens. 2. ICMP echo requests are sent from the device in Turin to the device in Madrid. 3. ICMP echo requests are sent from the device in Turin to the device in Paris Châtillon..
Expected results	1. ICMP echo responses are received from Athens, which means this connection is up. 2. ICMP echo responses are received from Madrid, which means this connection is up. 3. ICMP echo responses are received from Paris Châtillon, which means this connection is up.
Test 2 – Connectivity towards the 5G EVE Portal	facilities-connect-webportal
Test purpose	This test aims at ensuring that the 5G EVE Web Portal is reachable from the Turin site, where the I/W Framework is to be installed.
Test type	Functional.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution.
Configuration	PORTAL_CONF.
References	The tool must support RFC792 (Internet Control Message Protocol), plus its updates.
Applicability	Orchestration Plane Interworking.
Pre-test conditions	• One device in the Turin site capable of generating ICMP traffic, emulating I/W Framework modules. • One server at the local site hosting the 5G EVE Web Portal.
Test sequence	1. ICMP echo requests are sent from the device in Turin to the device emulating the 5G EVE Web Portal.
Expected results	1. ICMP echo responses are received from the device emulating the 5G EVE Web Portal, which means this connection is up.
Test 3 – Delay measurement	facilities-delay
Test purpose	This test aims at measuring the delay between Turin and another site at any precise moment.
Test type	Performance.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution. If delay wants to be monitored during a long time period, multiple measurements should be executed per day (e.g. every hour).
Configuration	ALL_PLANE_CONF.
References	The tool must support RFC792 (Internet Control Message Protocol), plus its updates.
Applicability	Orchestration, Control or Data Planes Interworking (depending on the plane the devices are deployed).
Pre-test conditions	• One device in the Turin site capable of generating ICMP traffic. • At least one device at the site towards which the delay is being measured, capable of responding to ICMP requests.
Test sequence	1. 30 ICMP requests are sent from the device in Turin to the device at the remote site.
Expected results	1. The average of the obtained RTT values can then be considered a reasonable delay measurement.
Test 4 – Capacity measurement	facilities-capacity

Test purpose	This test aims at measuring the capacity between Turin and another site at any precise moment.
Test type	Performance.
Related tests	None.
Priority	High.
Execution time	As required for the full test execution. If capacity wants to be monitored during a long time period, multiple measurements should be executed per day, but considering that bandwidth measurements may affect the running tests since they may create momentary congestion.
Configuration	ALL_PLANE_CONF.
References	None
Applicability	Orchestration, Control or Data Planes Interworking (depending on the plane the devices are deployed)
Pre-test conditions	• One device in the Turin site and another at the site towards which the capacity is being measured, capable of measuring bandwidth based on a client/server architecture (e.g. *iperf*).
Test sequence	1. The measurement tool is configured to exchange TCP traffic at maximum speed (e.g. by downloading via HTTP a 1GB file). 2. The measurement tool is configured to exchange UDP traffic at maximum speed (e.g. plain UDP traffic with 1500B payload).
Expected results	1. The average of the obtained bandwidth values can then be considered a reasonable capacity measurement for TCP traffic. 2. The average of the obtained bandwidth values can then be considered a reasonable capacity measurement for UDP traffic.

3.3 Risk plan

The risks can be analysed at three levels – Connectivity, Monitoring and Operation (see deliverable D3.2 – Sections 3.1.1 and 3.1.2 [3]), depending on whether it appears an issue in a single site execution or multi-site execution. The mitigation plan will be specific for every one of the six possible combinations.

First of all, risk identification is needed. Then, it is needed the establishment of an evaluation of the **priority** of the risks. It can be assumed that risks will have two components: probability and impact, and a priority needs to be assigned to each of them. The scale to be used is low-medium-high.

Finally, it is needed to have both action and mitigation plans. The **mitigation plan** is used when it is known in advance the existence of the risk and want to lower the probability and the severity of the risk. The **action plan** is related to what should be done once the risk has become reality and is active. A more comprehensive description about the risk plan subject can be found in [8], and also ISO/IEC 27005 can be used as reference standard [9].

Note that security, penetration or similar threatening scenarios are not under the scope of this document, which are more related to other initiatives, such as [10].

As several risks can be mapped to DoS attacks, they have been studied comprehensively in several forums. At SDN level and orchestrated VNFs and CNFs we are most interested in the DoS attacks from SBI rather than NBI or man-in-the-middle attacks.

Some additional thoughts regarding risks are:

- The presence of a VNF Validator, different from the VNF Provider, assuming that VNF Operator is the same actor that VNF Validator. The risk is different depending on the choice.
- Distinction (if needed) between Deployment and Activation in different scenarios (because they may be decoupled).

- Synchronization between PNFs and VNFs managed by one and many MANO systems.

Ideally, it is needed to manage some risk identifiers (e.g. RISK1a_001, meaning RISK for feature 1(orchestration plane networking) a(single site) #001), in order to enhance the information on the risk on a separate table describing some headlines, probability, impact and maybe historical evolution.

Coming back to the three levels described at the beginning of this section – Connectivity, Monitoring and Operation, Table 14 presents the risks identified in single-site scenarios.

Table 14: Risks identified in single-site scenarios

Key features	Cate-gory	Brief Description	Risk	Priority	Mitigation	Action	Probability
Orchestration Plane Interworking	Connec-tivity	Low bandwidth performance.	RISK1a_001	High	Mitigation Plan	Action Plan	Medium
		High latency. Messages not in order.	RISK1a_002	Medium	Mitigation Plan	Action Plan	Low
Single-site Experiment Monitoring Support	Moni-toring	Mismatch between monitoring tools (because of using different tools)	RISK1b_001	Low	Mitigation Plan	Action Plan	Medium
		Mismatch between monitoring tools (because of using different versions)	RISK1b_002	Medium	Mitigation Plan	Action Plan	Low
		Some monitoring tools need to be manually executed	RISK1b_003	Medium	Mitigation Plan	Action Plan	Medium
Single-site Applications Deployment Support	Opera-tion	Local orchestrator is different from one site to the other	RISK1c_001	High	Mitigation Plan	Action Plan	Medium
		Local orchestrator is not able to reach other orchestrators	RISK1c_002	Medium	Mitigation Plan	Action Plan	Medium
Single-site Network Automation Support	Opera-tion	Capability to deploy the required connectivity services (first phase). Different local controllers; different network infrastructure	RISK1d_001	Low	Mitigation Plan	Action Plan	Low
		Capability to deploy the required slices (second phase) to the requested site. Slicing support in the network but the specification for	RISK1d_002	Low	Mitigation Plan	Action Plan	Medium

		some slice is not possible.					

Table 15: Risks identified in multi-site scenarios

Key features	Cate-gory	Brief Description	Risk	Priority	Mitigation	Action	Probability
Control Plane Interworking	Connec-tivity	Low bandwidth performance but secure connectivity among sites for control traffic.	RISK2a_001	High	Mitigation Plan	Action Plan	Medium
			RISK2a_002	Medium	Mitigation Plan	Action Plan	Medium
Data Plane Interworking	Connec-tivity	Secure connectivity among sites for user traffic. Low bandwidth performance experiments will employ best effort connectivity. High bandwidth performance experiments will employ a parallel high bandwidth low latency network, which will be available at least between two sites.	RISK2b_001	Low	Mitigation Plan	Action Plan	High
			RISK2b_002	Medium	Mitigation Plan	Action Plan	High
Multi-site Experiment Monitoring Support	Moni-toring	Capability of translating the monitoring requirements defined by experimenters (based on selected KPIs) to the sites taking part in the same experiment. Sites will typically have different local monitoring tools and mechanisms.	RISK2c_001	High	Mitigation Plan	Action Plan	Low
			RISK2c_002	Medium	Mitigation Plan	Action Plan	Low
Multi-site E2E Orchestration Support	Opera-tion	Capability to deploy the required slices, and VNFs hosted in the 5G-EVE Catalogue on top of them, to the sites taking part in the same experiment. Sites will typically have different local	RISK2d_001	Low	Mitigation Plan	Action Plan	High
			RISK2d_002	Low	Mitigation Plan:	Action Plan	High

		orchestrators, controllers and network infrastructure.					

At the point in time of writing this document it is too early to be precise in both mitigation and action plans, so they have not been extended in the previous table. Regarding mitigation, there is a wide variety of mitigation techniques, but following elements must be considered:

- The value of the technique.
- The initial cost of the technique.
- The ongoing costs.

Typical mitigation techniques used in Information Systems are: (1) Policies and Procedures (2) Documentation (3) Training (4) Separation of duties (5) Configuration management (6) Version Control (7) Patch management (8) Intrusion detection system (9) Incident response (10) Technical controls (11) Physical controls.

Once evaluated and decided the mitigation technique(s) to be used, it is also needed the documentation of how implement them for the given use case. An example is provided in Table 16.

Table 16: Mitigation plan example (MPLAN1a_001)

Key features	Category	Brief Description	Risk	Priority	Mitigation	Action	Probability
Orchestration Plane Interworking	Connectivity	Low bandwidth performance.	RISK1a_001	High	MPLAN1a_001	ACTION_PLAN1a001	Medium
		High latency. Messages not in order.	RISK1a_002	Medium	MPLAN1a_002	Action Plan	Low

(1) <u>Policies and procedures</u>: Review VM network configurations, review physical fibre connections and increase allocated bandwidth or provide additional connection if bandwidth is under 100Mbps.

(10) <u>Technical controls</u>: increase frequency of bandwidth monitoring at orchestration plane from 1 minute to 10 seconds.

(2) (3) (4) (5) (6) (7) (8) (9) (11): Not Applicable.

The action plan captures next steps for the group. It identifies the action to be accomplished, the action agent, and the timeframe allotted for the action. This is an essential component to the meeting if progress is to be made. The action plan is usually given as a table showing the action item, the action agent, and the target date for completion.

For the **example** above it could be like:

ACTION_PLAN1a001

> ***<u>Action item description</u>: Add extra 25Gbps physical interface to BareMetal server hosting VNFs.***
>
> ***<u>Action agent</u>: Orange-PL deployment personnel.***
>
> ***<u>Target Date</u>: 10 June 2020.***

4 Interworking testing roadmap

According to deliverable D3.2 [3], there are planned four drops of the I/W Framework, until reach the official version in June 2020. For each drop is defined in D3.2 [3] the features, services and capabilities supported.

The roadmap of the Interworking testing must be aligned with the roadmap of the delivery, included in D3.3 [1] deliverable, and it is presented in Table 17.

Table 17: Interworking testing roadmap

Test	Type	D3.3	Drop 1	Drop 2	D3.4
Multi-Site Network Orchestrator	Unit Test	Unit test of the services included in D3.3	Unit testing for all services and features required for the selected use cases	Full tests	Full tests
Multi-Site Catalogue	Unit Test	Unit test of the services included in D3.3	Unit testing for all services and features required for the selected use cases	Full tests	Full tests
MSNO - MSC	Integration Test	-	Basic integration for supported UC	Full integration	Full integration
Multi-Site Inventory	Unit Test	-	Unit testing for all services and features required for the selected use cases	Full tests	Full tests
Adaptation Layer	Unit Test	-	Unit testing for all services and features required for the selected use cases	Full tests	Full tests
MSC - AL	Integration Test	-	Basic integration for supported UC	Full integration	Full integration
MSNO - AL	Integration Test	-	Basic integration for supported UC	Full integration	Full integration
Data Collection	Unit Test	-	Testing oriented to support selected UC	Single-site testing	Full tests
Runtime Configuration	Unit Test	-	Testing oriented to support selected UC	Single-site testing	Full tests
Interworking Framework	System Test	-	Single-site testing, focused on selected UC	Single-site testing	Full tests

5 Interworking testing tools

This section aims to provide an overview of the tools that will be used in order to test and validate the Interworking Framework implementation. The various components of the framework, shown in Figure 11, that will be the target of these tools are:

- Multi-Site Catalogue.
- Multi-Site Inventory.
- Multi-Site Network Orchestrator.
- Data Collection Manager.
- Runtime Configurator.
- Interworking API.
- Adaptation Layer.

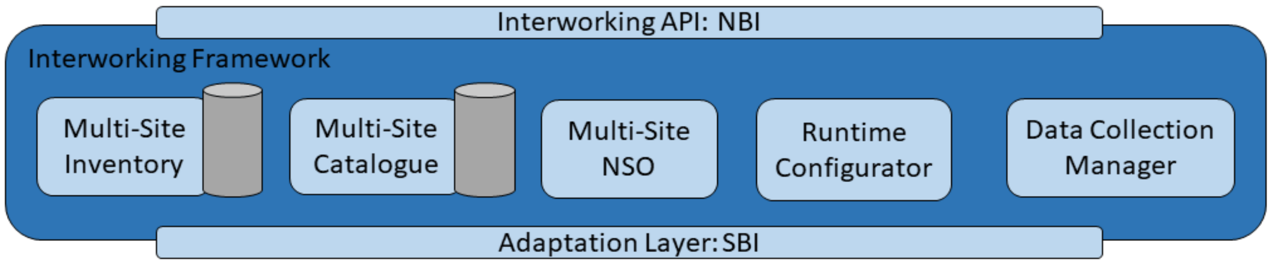

Figure 11: Interworking Framework overview

In the context of the WP5, a detailed state-of-the-art research has been done in D5.1 [11] on the various tools that can be used for testing software and network elements. Based on the results of that research and the fact that the implementation of the Interworking Framework consists only of software modules, the selection of testing tools becomes a lot more straightforward. In this section a set of tools will be proposed with the required capabilities to execute all the necessary testing for the validation of the Interworking Framework.

All the individual components of the proposed framework rely on APIs for their operation. As such, the main focus of testing will be these APIs. All the APIs in question can be simulated by using mock servers so the test preparation and execution, in some cases, can proceed independently of their implementation progress. Moreover, some of the components utilize message buses for metric collection, like Kafka, or local databases for storing data. Those are separate testing points that will also be tested. The specific testing points of each component have been identified in the previous sections of this deliverable.

The main tool that will be used for defining the proposed test suites, as mentioned in this section, will be Robot Framework, supported by other different tools for executing specific tests for the different components involved in the testing process.

5.1 High-level architecture

The Interworking Framework, depicted in Figure 11, is a multi-component implementation, and as such each component has to be tested and validated separately at first and then proceed to test the combined operation of multiple components. Finally, the complete Interworking Framework will be tested, as shown in Figure 12. Using the implementation architecture proposed by WP5, a high-level overview of the testing tool architecture, capable of testing both the individual components as well as the whole framework, can be seen in Figure 12.

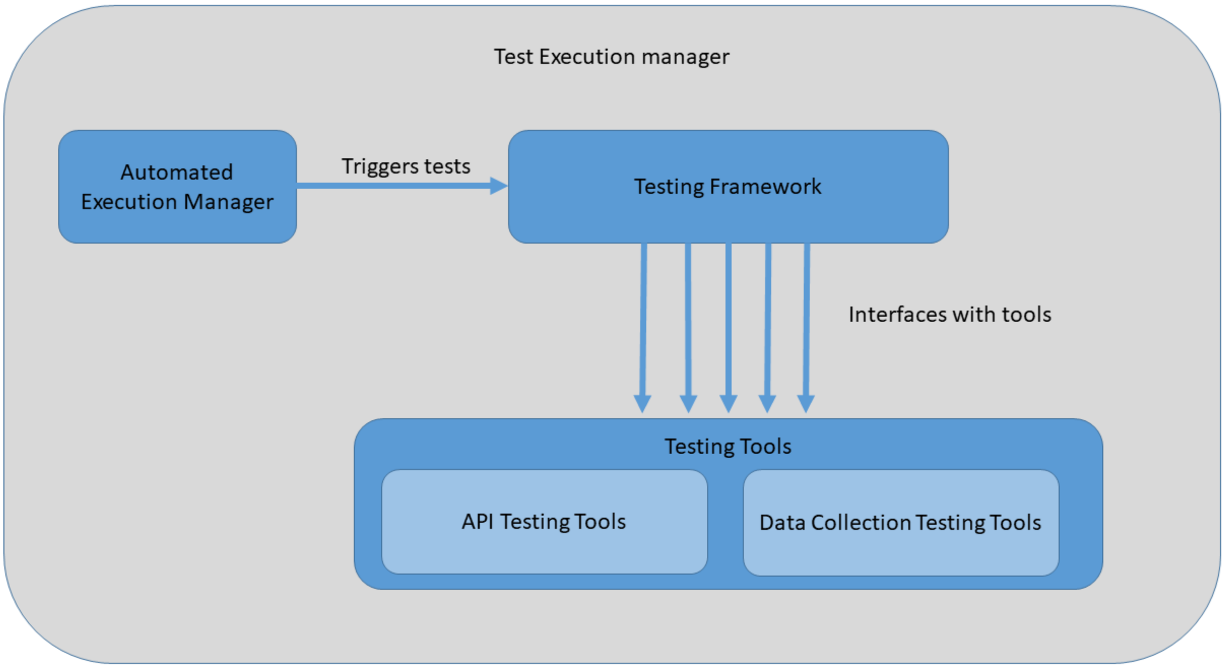

Figure 12: Testing tool high-level architecture

The Test Execution manager is responsible for the testing process. Encapsulating the necessary functionalities while keeping them separate allows for a modular implementation, open to modifications and enhancements. In this way the testing tools are not limited neither by type nor by number.

Very briefly the components of the Test Execution manager are:

- The **Automated Execution Manager** is responsible for parsing the test request and triggering the corresponding tests with the requested configurations.
- The **Testing Framework** executes the testing scripts and interfaces with the individual testing tools.
- The **Testing Tools** test the functionalities and evaluate the performance of the target components.

The main benefit of the proposed Test Execution manager is the implementation of a standalone module capable of handling the complete testing process of a component or system without external control and extensible enough so that the list of testing suites and testing tools can be increased without much effort.

5.2 Proposed tools and required capabilities

5.2.1 Testing tools overview

All the tools proposed in this section have been chosen on the basis of completely automating the testing procedure. This is in line with the work from WP5, described in deliverable D5.1 [11] and doing so it will reduce integration efforts later on in the project. Based on that work, Robot Framework is proposed as the testing framework and Jenkins as the automated execution manager for the tests.

Robot Framework is a Python-based and extensible keyword-driven test automation framework. It is widely used for end-to-end acceptance testing and acceptance-test-driven development (ATDD) and allows the definition of tests using a human-friendly format. This test definition involves the identification of all elements involved in the experiment, the actions that have to be taken and the expected results in each test. It is worth mentioning that it is an open source software initially developed at Nokia Networks, nowadays sponsored by Robot Framework Foundation. Jenkins is an open source automation server that helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. It is a friendlier execution engine than Robot Framework and also supports interaction with it through the use of plugins.

The combination of these two tools provide an automated execution manager capable of handling the execution and collection of results from the various tests. This execution manager, shown in Figure 13, will be acting as a wrapper around the individual tools tasked with the actual tests. It is lightweight and can be hosted on a central node or as a part of one or more VNFs.

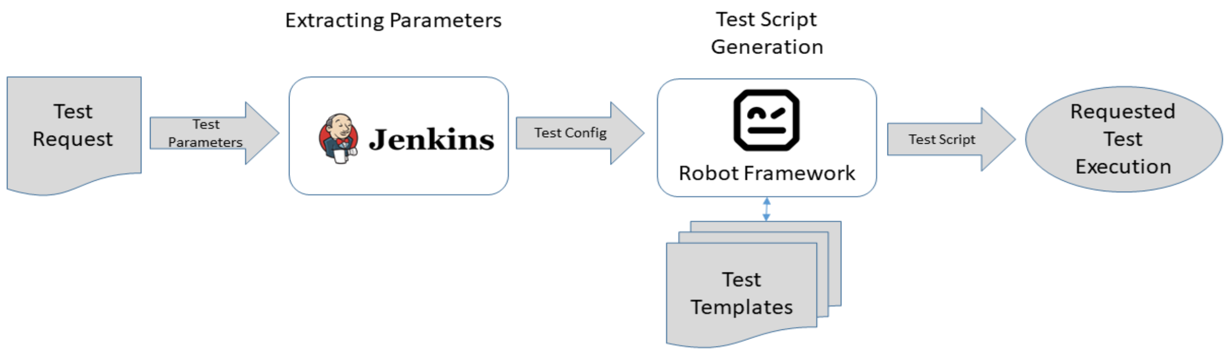

Figure 13: Test Execution Manager overview

After examining all the components of the Interworking Framework that will be under test, their testing points fall under two categories, API endpoints and data collection points, either through message buses or with local databases. All of these testing points will be handled in the same way, first their functionalities will be tested and then their performance under load will be evaluated. These two series of tests will verify the intended functionality of each component, both individually and in combination with others, in the Interworking Framework and also reveal any bottlenecks or possible future scaling issues of a specific component or the framework as a whole.

5.2.2 API testing

Regarding API functionality tests there are multiple tools capable of providing the requirement capabilities. One such tool is Robot Framework itself, capable of testing API endpoints and validating the responses based on the expected outcome of the provided requests. Implementing these tests on Robot Framework leads to a smaller number of individual tools needed and less integration effort. From an architectural point of view, if the functionalities need to be kept separate and distinct, meaning that the Robot Framework only plays the role of the test execution engine, then another tool can be chosen. One other such tool is pyresttest[19], a REST testing tool where the tests are defined in basic YAML or JSON format and there is no need for additional code. After validating the expected operation of the components, the performance under load will be evaluated. Such tools are locust[20] and httperf[21]. In this category of tools there is a wide range of available choices. The proposed tools are based purely on prior use and are likely to change if it deemed necessary for integration purposes.

- **Locust** is an easy-to-use, distributed, user load testing tool. It is commonly used for load-testing websites (or other systems) in order to figure out how many concurrent users a system can handle. The number of requests to the framework's individual components is not expected to be significantly large at first. Scaling the EVE platform to support more use cases and possibly more sites and thus increasing the load to these components, though, could potentially lead to some performance issues. The goal of using tools like locust is to identify any and all such possible issues.

[19] https://github.com/svanoort/pyresttest

[20] https://locust.io/

[21] https://github.com/httperf/httperf

- **Httperf** is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance. The focus of httperf is not on implementing one particular benchmark but on providing a robust, high-performance tool that facilitates the construction of both micro- and macro-level benchmarks.

Regarding API testing, there is another category of tools useful to our cause and that is mock servers. Mock servers are tools that allow the simulation of the behaviour of API endpoints so that request-response transactions can be tested even if the complete functionality is not implemented yet. This is beneficial for systems like the proposed Interworking Framework that are multi-component and require a lot of message exchanges between the individual components. One such tool is Prism, an open-source HTTP mock server that can also generate servers from OpenAPI documents, a functionality most useful since the use of OpenAPI has been adopted for the implementation of the Interworking Framework.

5.2.3 Data collection testing

Based on the architecture proposed in D3.2 [3], the Interworking Framework will use a publish-subscribe queue (implemented with Apache Kafka tool) as the data aggregation mechanism, also providing data storage and persistence. Thus, it is necessary to test the bus interactions to ensure their optimal operation even under stress.

5.2.3.1 Kafka bus testing

Kafka is a distributed, partitioned, replicated commit log service. It provides the functionality of a messaging system. In the context of the Data Collection Manager it will be used to collect all the metrics generated from the various nodes deployed on the facilities during the use cases experiments. Since some of the current use cases and possible future ones need many nodes to run the corresponding experiments, a large amount of data will be generated which according to the implementation plan will be separated in individual topics per node and possibly per source of generation. Therefore, it is necessary to test both the number of available topics and partitions as well as the throughput, the maximum number of message transactions, that can be supported.

Regarding the number of topics and partitions, the currently supported theoretical maximum is in the few hundreds thousands [12]. Even with multiple nodes and data sources of the use cases currently planned for the EVE platform, the requirements are unlikely to hit that limit soon. Moreover, newer versions of the Kafka software have often been increasing that limit so future larger scale experiments should not be an issue as well. The throughput is the metric that directly affects the performance of the Data Collection Manager and is one of the most import metrics for its operation. To that extent, scripts[22] specifically for testing this metric have already been provided with the tool's installation and can be used to fulfil that purpose.

Moreover, there are other specific components that can be used for testing or monitoring Kafka, which are:

- **Kafkat[23]:** it is a simplified command-line administration for Kafka brokers. It was created by AirBnB and it is quite simple to install and use so that it can be done queries for obtaining the status of the Kafka deployment by using command line tools.
- **Kafka Confluent[24]:** provides a more complex way to monitor the system.
- **Kafka Monitor[25]:** it is a framework to implement and execute long-running Kafka system tests in a real cluster. It complements Kafka's existing system tests by capturing potential bugs or regressions that are only likely to occur after prolonged period of time or with low probability.

[22] https://github.com/kafka-dev/kafka/tree/master/core/src/main/scala/kafka/tools

[23] https://github.com/airbnb/kafkat

[24] https://docs.confluent.io/current/kafka/monitoring.html

[25] https://github.com/linkedin/kafka-monitor

In the case of load and stress tests, there are more tools which can be used, such as Avalanche[26] or simple iperf or ping commands.

5.2.4 Database testing

Based on the proposed implementation of the Interworking Framework, local databases might be used in various components to store all required information for the components internal functionalities such as the metrics/KPIs/results collected from the Data Collection Manager. In the same manner the APIs were tested, the databases will also go through functional testing first and then evaluated under load. Since the volume of the data transferred between the components and the databases is expected to be high, especially for the database containing the KPIs data, stress testing the database is mandatory to prevent any data loss or corruption.

Depending on the type of the databases that will be chosen for the implementation, SQL or NoSQL, the testing tools can also be separated in the same two categories. For SQL database testing, since SQL is the most widely used type of database, there is a plethora of tools available. The proposed tools favour the open-source status as well as the configurability and ease of integration with the rest of the selected tools. Namely, some popular database testing tools are dbstress[27], Tsung[28] and HammerDB[29].

- **Dbstress** is a software for stress testing and load testing the server parts of information systems and database applications, as well as databases and servers themselves. It is suitable for solution scalability and performance testing, comparison and tuning. The tool allows users to create and configure a continuous set of requests to the server of the OLAP (query execution) and OLTP (adding/loading, modifying and deleting data in the database) types. The user can specify several virtual users to be emulated, priority and type of requests for each task (virtual user type).
- **Tsung** is an open-source multi-protocol distributed load testing tool. It can be used to stress HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP, MQTT and Jabber/XMPP servers. The purpose of Tsung is to simulate users in order to test the scalability and performance of IP based client/server applications. You can use it to do load and stress testing of your servers. Many protocols have been implemented and tested, and it can be easily extended. It can be distributed on several client machines and is able to simulate hundreds of thousands of virtual users concurrently (or even millions if you have enough hardware …). It is developed in Erlang, an open-source language made by Ericsson for building robust fault-tolerant distributed applications.
- **HammerDB** is an open source database load testing and benchmarking tool for Oracle Database, Microsoft SQL Server, IBM DB2, TimesTen, MySQL, MariaDB, PostgreSQL, Postgres Plus Advanced Server, Greenplum, Redis, Amazon Aurora and Redshift and Trafodion SQL on Hadoop. It is the highest performing database benchmarking and load testing tool with built-in workloads based on the industry standard TPC-C and TPC-H specifications.

Regarding the NoSQL type of databases, the list of testing tools is not as large. The suggested tools and also the most popular among them, are Yahoo! Cloud Serving Benchmark[30] and JMeter[31].

- **Yahoo! Cloud Serving Benchmark (YCSB**) is an open-source specification and program suite for evaluating retrieval and maintenance capabilities of computer programs. It is often used to compare relative performance of NoSQL database management systems. YCSB was contrasted with the TPC-H benchmark from the Transaction Processing Performance Council, with YCSB being called a big data benchmark while TPC-H is a decision support system benchmark. It has been used for multiple-product

[26] https://www.spirent.com/products/testing-security-app-aware-device-networks-avalanche

[27] https://github.com/semberal/dbstress

[28] http://tsung.erlang-projects.org/

[29] https://www.hammerdb.com/

[30] https://github.com/brianfrankcooper/YCSB

[31] https://jmeter.apache.org/

comparisons by industry observers such as Network World (comparing Cassandra, MongoDB, and Riak), Thumbtack Technologies (comparing Aerospike, Cassandra, Couchbase, and MongoDB), and the Polytechnic Institute and University of Coimbra (comparing Cassandra, HBase, Elasticsearch, MongoDB, Oracle NoSQL, OrientDB, Redis, Scalaris, Tarantool, and Voldemort).

- **JMeter** is an Apache project that can be used as a load testing tool for analysing and measuring the performance of a variety of services, with a focus on web applications. It can be used as a unit-test tool for JDBC database connections, FTP, LDAP, Web services, JMS, HTTP, generic TCP connections and OS native processes. One can also configure JMeter as a monitor, although this is typically considered ad hoc rather than advanced monitoring. It can be used for some functional testing as well. It supports variable parameterization, assertions (response validation), per-thread cookies, configuration variables and a variety of reports. JMeter architecture is based on plugins. Most of its "out of the box" features are implemented with plugins. Probably it has a superset of features of httperf, but the problem when using JMeter is that it requires Java to work, so there might be issues if several JVMs are required to be instantiated at the same time.

It is worth mentioning that only Yahoo! Cloud Serving Benchmark was made specifically for the purpose of testing NoSQL and cloud storage solutions. JMeter was made as a general-purpose load tester that progressively gets more support for additional storage solutions.

6 Conclusions

This deliverable presents the first drop of the Interworking Framework testing specification, derived from the current implementation status described in deliverable D3.3 [1]. Based on the current requirements obtained after the study of the different components, services and capabilities that have been defined within the Interworking Framework, the specification of a testing methodology and the definition of the different test suites for all the possible testing points that are related to the Interworking Framework have been provided along this document.

Regarding the testing methodology selected for this purpose, i.e. a methodology based on software development testing principles and on ETSI standard specifications, it has been detected that there are a lot of similarities between these methods and the desired testing process for the Interworking Framework; mainly, the division of the SUT in different FUT, i.e. the division of the test suites in different categories, from single component tests to the whole system, also including possible integration between components as an intermediate step. This kind of testing methodology really makes sense in the context that the Interworking Framework has being defined, as it is composed by several components whose operation is practically independent from the others, so this growing testing process contemplated in the I/W Framework testing methodology really fits well in this scope.

One of the key aspects that has been achieved in this deliverable is the high-level definition of the test suites for the different FUT and SUT with a textual description, avoiding in this first drop any implementation issues related to the test suites, which will be described in the future D3.7 that is the second and final drop of the Interworking test suites specification. The idea again is to follow an iterative process in the definition of the different test suites, starting with the high-level description proposed in this deliverable and finishing with the implementation of these high-level tests using a set of testing tools (mainly based on the Robot Framework component) with two main objectives: (i) test the identified testing points from the Interworking Framework, and (ii) verify the assumptions made in the high-level description, correcting any possible misalignment (contemplated in the preliminary risk plan proposed in this deliverable) in that implementation stage.

Finally, note the significance that this deliverable has for the rest of the project: it is the first deliverable only dedicated to define a testing process for a given system (in this case, the Interworking Framework), only contemplating the I/W Framework itself in an isolated way, without interacting with the 5G EVE Portal and Site facilities directly (but simulating these interactions in the tests for verifying the correct behaviour of the interfaces). This testing process definition may lay the basis for the definition of testing techniques and processes that can be implemented in other areas of the 5G EVE project if needed, replicating the methodology and the testing definition process already presented in this deliverable.

Acknowledgment

This project has received funding from the EU H2020 research and innovation programme under Grant Agreement No. 815074.

References

[1] 5G EVE Deliverable D3.3: "First implementation of the interworking reference model" – ICT-17-2018 5G EVE Project. – October 2019. To be published in: https://www.5g-eve.eu/deliverables/

[2] 5G EVE Deliverable D3.1: "Interworking Capability Definition and Gap Analysis Document" – ICT-17-2018 5G EVE Project. – December 2018. [Online]. Available: https://www.5g-eve.eu/wp-content/uploads/2019/01/5geve-d3.1-interworking-capability-gap-analysis.pdf

[3] 5G EVE Deliverable D3.2: "Interworking Reference Model" – ICT-17-2018 5G EVE Project. – June 2019. [Online]. Available: https://www.5g-eve.eu/wp-content/uploads/2019/09/5geve_d3.2-interworking-reference-model.pdf

[4] ETSI EG 202 237 V1.2.1: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic approach to interoperability testing". August 2010. [Online]. Available: https://www.etsi.org/deliver/etsi_eg/202200_202299/202237/01.02.01_60/eg_202237v010201p.pdf

[5] ETSI EG 202 568 V1.1.3: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Testing: Methodology and Framework". April 2007. [Online]. Available: https://www.etsi.org/deliver/etsi_eg/202500_202599/202568/01.01.03_60/eg_202568v010103p.pdf

[6] ETSI GS NFV-TST 002 V1.1.1: "Network Functions Virtualisation (NFV); Testing Methodology; Report on NFV Interoperability Testing Methodology". October 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/002/01.01.01_60/gs_NFV-TST002v010101p.pdf

[7] 5G EVE Deliverable D4.1: "Experimentation Tools and VNF Repository" – ICT-17-2018 5G EVE Project. – October 2019. To be published in: https://www.5g-eve.eu/deliverables/

[8] "The Risk Planning Process" – October 2016. [Online]. Available: https://www.projectengineer.net/the-risk-planning-process/

[9] "FAQ: information risk management". [Online]. Available: https://www.iso27001security.com/html/risk_mgmt.html

[10] 5G Ensure Deliverable D2.6: "Risk Assessment, Mitigation and Requirements (final)" – H2020-ICT-2014-2 Project. – November 2017. [Online]. Available: http://5gensure.eu/sites/default/files/5G-ENSURE_D2.6_Risk%20assessment%20mitigation%20and%20requirements%20%28final%29_0.pdf

[11] 5G-EVE Deliverable D5.1 "Disjoint testing and validation tools" – ICT-17-2018 5G EVE Project – April 2019. [Online]. Available: https://www.5g-eve.eu/wp-content/uploads/2019/05/5geve-d5.1-disjoint-testing-and-validation-tools.pdf

[12] "Apache Kafka Supports 200K Partitions Per Cluster" – November 2018. [Online]. Available: https://blogs.apache.org/kafka/entry/apache-kafka-supports-more-partitions