5G European Validation platform for Extensive trials

Deliverable D3.3
First implementation of the interworking reference model

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 T3.3
Deliverable Identifier:	5G_EVE_D3.3
Deliverable Title:	First implementation of the interworking reference model
Editor(s):	Marc Mollà Roselló
Author(s):	Ramón Perez, Aitor Zabala (TELC); F.Lombardo, S.Salsano, P.Lungaroni, M. Pergolesi, M. Femminella, G. Reali (CNIT); Gopolasingham Aravinthan (NOK-FR); Marc Mollà Roselló (ERI-ES); Jaime Garcia-Reinoso, Pablo Serrano Yañez-Mingot (UC3M); Grzegorz Panek (ORA-PL)
Reviewer(s):	Marius Iordache Juan Rodriguez Martinez
Contractual Date of Delivery:	31/10/2019
Submission Date:	31/10/2019
Dissemination Level:	PU
Status:	Final
Version:	1.0
File Name:	5G_EVE_D3.3

Deliverable History

Version	Date	Modification	Modified by
V0.1	*09/09/2019*	*First draft*	*Marc Mollà Roselló*
V0.2	*25/09/2019*	*First contributions*	*Marc Mollà Roselló*
V0.3	*07/10/2019*	*First internal release*	*WP3 team*
V0.4	*11/10/2019*	*First draft for internal review*	*WP3 team*
V0.5	*15/10/2019*	*First proposal for internal review*	*WP3 team*
V0.6	*16/10/2019*	*Updated proposal for internal review (French site)*	*WP3 team*
V0.7	*25/10/2019*	*Updated with the internal revision*	*WP3 team*
V0.9	*28/10/2019*	*Ready for delivery*	*WP3 team*
V1.0	*30/10/2019*	*QA review*	*Kostas Trichias*

Table of Contents

List of Acronyms and Abbreviations

Acronym	*Meaning*
3GPP	Third Generation Partnership Project
5G	Fifth Generation
ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LCM	Lifecycle Management
MANO	Management and Orchestration
MSNO	5G EVE Multi-Site Network Orchestrator
MSO-LO	5G EVE Multi-Site NSO to Local Orchestrator
NBI	North-Bound Interface
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSO	Network Service Orchestrator
ONAP	Open Network Automation Platform
OSM	Open Source MANO
PNF	Physical Network Function
PNFD	PNF Descriptor
REST	Representational State Transfer (software architectural style)
SBI	South-Bound Interface
SQL	Structured Query Language
SSH	Secure Shell
TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modeling Language
VNF	Virtual Network Function
VNFD	VNF Descriptor
YAML	YAML Ain’t Markup Language

List of Figures

List of Tables

Executive Summary

Deliverable D3.3 is the first deliverable dealing with the implementation of the Interworking Framework. As defined in the proposal, this deliverable is basically a software deliverable that contains part of the services and features of the I/W Framework. In addition to the software delivery, we have produced this document in order to explain details about the internal architecture and to also provide updates of the previous deliverables.

The high-level design of the Interworking Framework, as well as its services, capabilities and features are defined in previous deliverables of the WP3 (D3.1 and D3.2); and they are the foundations of the I/W Framework development.

For each of the components defined in the I/W Framework architecture we include an open definition of the implemented API, regardless of whether the API is public or for internal use. This information follows the openness and modular principles that we follow in the design and development of the I/W Framework. The APIs are defined using the OpenAPI specifications and they are available in a the 5G EVE public repository[1]. This deliverable includes the first release of the Multi-Site Catalog and the Multi-Sire Network Orchestrator. For both components, this document describes the supported services in this deliverable. We also include a description of the internal component architecture, design and workflows of the services implemented by: Multi-Site Catalog, Multi-Site Network Orchestrator and Adaptation Layer. In the case of Runtime Configurator and Data Collection Manager, we update in this deliverable the design included in D3.2, with more detailed workflows and descriptions.

This document also contains the roadmap of the I/W Framework for the next project year, mapping the service, capabilities and features required for the I/W Framework with the status in each of the deliveries. According to the proposal, the I/W Framework design has two deliverables: this one and the final delivery in M24. We foresee the need of intermediates deliveries of the I/W Framework for supporting the MS8 (M18) and for supporting the WP4 deliverables. For that reason, we include two additional deliveries: Drop 1 and Drop 2.

Finally, we include some updates about the status of the inter-site connectivity and the technical solution adopted to provide this interconnection.

[1] https://github.com/5GEVE/OpenAPI

1 Introduction

Deliverable D3.3 "*First implementation of the interworking reference model*" is the first software deliverable of the Interworking (I/W) Framework, developed in the scope of Task 3.3 "*Interworking model implementation and deployment*". The deliverable contains the public definitions of the Interworking Framework components' interfaces, the low-level description of the components as well as the references to the public repository for the software artefacts.

1.1 Initial context

This work is based on the analysis, architecture and interface definitions included in D3.1 ([1]) and D3.2 ([2]). A summary of the design decisions contained in these documents is included in section 2.

1.2 Structure of the document

The main structure of this deliverable can be summarized as follows:

- Section 2 contains a summary of the Interworking Framework design that is included in the D3.1 and D3.2
- Section 3 contains the description of the software included in this deliverable. The minimum content is the public specification of the API provided for each component, with a reference of the public open API definition. If the component implements a subset of the required features, a description is also included
- Section 4 contains an update of the status of the inter-site connectivity
- Section 5 describes the Interworking Framework roadmap at the time of this delivery

2 Interworking Framework design

The I/W framework provides a set of key features to the 5G EVE experimentation platform as it abstracts the specific logics and tools available in each site facility, aiming at exposing the entire multi-site infrastructure through a unified and site-agnostic information model towards the 5G EVE portal. The heterogeneous capabilities adopted in each site facility are properly abstracted by the I/W framework to enable the realization of a common and unified end-to-end experimentation and validation facility, thus ensuring interoperability of validation services offered by the various sites while achieving their end-to-end integration. In terms of main features, on the one hand the I/W framework provides a common interworking model for the abstraction of the heterogeneous capabilities and orchestration solutions implemented in each 5G EVE site facility, exposing seamless services and APIs to the 5G EVE portal to facilitate the experimentation of vertical services. On the other hand, the I/W framework enables the site facility interconnection at different levels (from orchestration to control and data plane layers) to allow the execution of experiments (and thus instantiation of vertical services) in different sites (and when and where possible and required, in multiple sites) in a transparent way.

Deliverables D3.1 [1] and D3.2 [2] provide a detailed specification of the I/W framework architecture, and set the foundations for the software development of the I/W framework functional components reported in this document. While D3.1 defined an initial architectural decomposition as a result of the analysis of a list of use case driven technical requirements for the deployment, runtime operation and monitoring of the vertical experiments, deliverable D3.2 augmented this early specification with the definition of the I/W framework APIs (at both northbound and southbound) and a more detailed functional specification of each component including I/W workflows and mechanisms.

For the sake of completeness of this document, the 5G EVE I/W framework architecture is depicted in Figure 1. The five main components building the I/W framework are: The Multi-site Network Service Orchestrator, the Multi-site Catalogue, the Multi-site Inventory, the Runtime Configurator and the Data Collection Manager. All of them are specified in terms of main features in deliverable D3.2, including northbound and southbound APIs. As a general principle, the 5G EVE I/W framework architecture re-uses (and enhance where required) existing relevant standards and solutions provided by other European funded projects to not design and develop from scratch such a complex architecture.

The interworking APIs depicted in Figure 1 are built by the collection of the APIs exposed by each of the core I/W framework components. These APIs are detailed in deliverable D3.2 and are reported in the form of Open API representations in section 3 of this deliverable. These OpenAPI representations have to be considered as the REST implementation of the APIs defined in D3.2. Moreover, most of these interworking APIs highly leverages on existing ETSI NFV SOL005 APIs for the lifecycle management of NFV Network Services implementing the vertical experiments. As described in section 3 (under each per-component subsection) these SOL005 APIs (and the related data models) are enhanced where needed to support the I/W reference model defined in D3.2.

At the southbound, the I/W framework adaptation layer plays a key role as it implements the required abstraction features on top of the site facilities, harmonizing under a common information model and set of APIs the heterogeneous capabilities offered by each site in terms of orchestration, catalogues, monitoring and runtime configuration. Where applicable, the common APIs exposed by the adaptation layer to the other I/W framework services are based on existing standard specifications, like those offered to the Multi-site Network Service Orchestrator and the Multi-site Catalogue which are based on ETSI NFV IFA specifications.

Deliverable D3.2 also provided two different options for the interaction and integration of each I/W framework component with the adaptation layer for their access to the different sites' services (from orchestration to catalogues and monitoring). In the first option (named as "A"), each of the I/W framework components independently interacts with the per-site services and tools (e.g. orchestrators, SDN controller, catalogues, monitoring platforms, etc.) through the adaptation layer. In the second option (named as "B), the Multi-site Network Service Orchestrator acts as a proxy for all the I/W framework interactions with the local per-site services, thus relaxing the complexity of the abstraction layer. At the time of writing D3.2 none of the two options was considered clearly preferable with respect to the other, and the decision was left open for the software implementation stage to have each component to choose the best suited option. As a result, in this

document, the option A is followed by the Multi-site catalogue only, while option B is followed by all the other components.

This means the Multi-site Catalogue provides on its own the whole set of I/W features for what concerns the management of Network Service and VNF Descriptors, i.e. unified northbound APIs based on ETSI NFV SOL005 [3] and adaptation (with data model and API logics translation) towards the heterogeneous per-site orchestrators and catalogues.

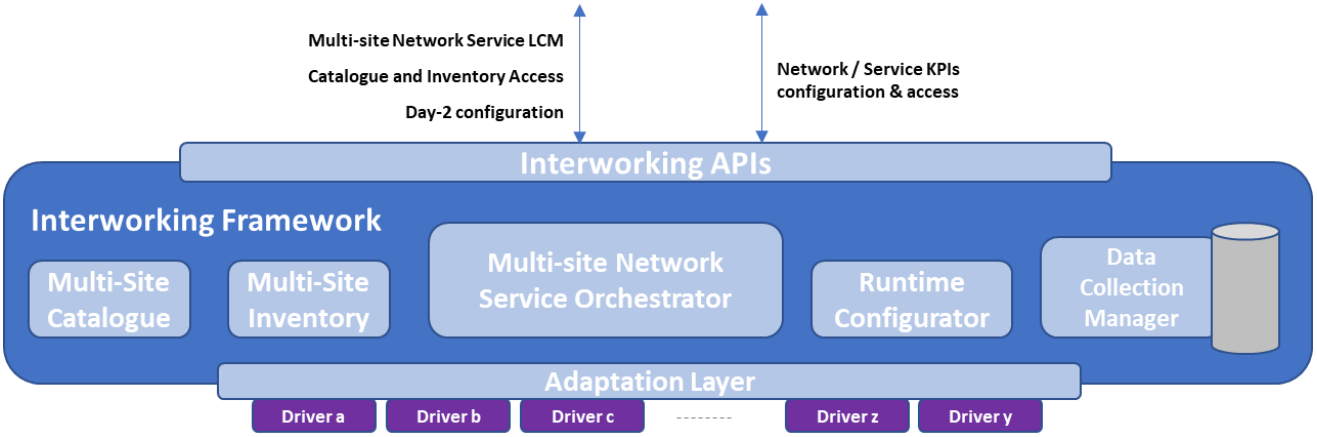


Figure 1: I/W Framework Architecture

3 Software artefacts

This section contains the description of the software artefacts included in this delivery. For all Interworking framework components, we include the description of the public interface provided by the component, in Open API format. For the Multi-Site Catalogue, the Multi-Site Network Orchestrator and the Adaptation Layer we include a description of the services and features included in this deliverable. Additionally, we also include a low-level detail of the internal design of components

3.1 Multi-Site Catalogue

The 5G EVE Multi-site Catalogue stores all the information related to Networks Service Descriptors (NSDs) and VNF Descriptors (VNFDs) that can be used by the 5G EVE portal for the provisioning of single-site and multi-site vertical experiments. It is the central repository of the whole 5G EVE platform for what concerns the capabilities offered by the site facilities which are exposed through a common model that follows the ETSI NFV SOL specifications. The Multi-site Catalogue implements mechanisms for the synchronization between the information available in the catalogues at the different sites and the centralized I/W catalogue, by maintaining the association between the local NSDs and VNFDs and the corresponding descriptors at the sites' catalogues.

Moreover, dedicated APIs are exposed to the 5G EVE Portal for the automated onboarding of NSD modelling the vertical experiments into the catalogues at the target sites. These APIs leverage on logics for the automated translation between the I/W information model and the site-specific languages and information models implemented in each of the sites' catalogues. Indeed, as stated above, the Multi-site Catalogue embeds part of the I/W framework adaptation layer features.

With reference to the I/W workflows defined in deliverable D3.2, the Multi-site Catalogue implements those related to Network Service and VNF on-boarding, in particular:

- *Catalogues synchronization*, to let the Multi-site Catalogue automatically retrieve and store available NSDs and VNFDs from each site facility.
- *VNF onboarding*, to allow dynamic and automated retrieval and store of new VNFDs that are on-boarded directly in the 5G EVE sites at runtime.
- *VNF removal*, to allow dynamic and automated removal of existing VNFDs in the Multi-site Catalogue when the related VNFs are no longer available in the 5G EVE sites.
- *Network Service Descriptor onboarding*, to allow the 5G EVE portal to onboard in the I/W framework new NSDs for single- and multi-site vertical experiments.
- *Network Service Descriptor removal*, to allow the 5G EVE portal to remove existing NSDs whenever the related single- or multi-site vertical experiment is no longer available in the 5G EVE platform.

3.1.1 Software architecture

The Multi-site Catalogue software prototype is developed in WP3 (while its software design is reported in WP4) as an extension of the Nextworks' 5G Apps and Services Catalogue[4], already used in the context of the 5G-MEDIA project [5]. The high-level software design of the Multi-site Catalogue is fully detailed in deliverable D4.1 [6], while the enhancements applied on top of the original 5G Apps and Services Catalogue can be summarized as follows:

- New Catalogue Service logics for the support of the Multi-site Catalogue onboarding workflows defined in deliverable D3.2
- New Policy Management features to regulate and configure the interactions with the per-site orchestrators and catalogues (e.g. what is onboarded where)
- Enhanced internal information model and ETSI NFV SOL005 REST APIs to:
    - Support additional location constraints (i.e. specification of which site and orchestrator) when new NSDs are onboarded from the 5G EVE portal

    - Expose additional information related to location constraints of NSDs and VNFDs (i.e. where descriptors are available for instantiation)
- Enhanced Graphical User Interface (GUI) features to expose full multi-site NSDs and VNFDs information (mostly targeting admin access)
- Integration with authorization and authentication mechanisms, as defined in deliverable D4.1

In particular, the main relevant and impacting enhancements with respect to the existing software are those related to the implementation of the multi-site onboarding workflows, which are specific for the 5G EVE platform and are those that truly enable the integration of heterogeneous site facilities under the same catalogue, hiding the complexity of the different per-site API logics and data models. As a consequence, in support of the various per-site orchestrators and catalogues, the 5G Apps and Services Catalogue is planned to be enhanced with additional NFVO drivers to fully support the 5G EVE multi-site onboarding and NSD/VNFD management logics and workflows.

3.1.2 Open API description

The Multi-site Catalogue northbound interface is defined in the deliverable D3.2, and it is implemented in the software prototype described in this document as a set of REST APIs. These REST APIs are based on the ETSI NFV SOL005 v2.4.1 specification, with focus on NSD Management and VNF Package Management.

The OpenAPI representations for the Multi-site Catalogue northbound REST APIs are available in the 5G EVE github repository, at: https://github.com/5GEVE/OpenAPI/tree/v0.2/MultiSiteCatalogue.

The APIs and related endpoints implemented and exposed by the Multi-site Catalogue are detailed in deliverable D4.1 [6], as part of the software design documentation.

The Multi-site Catalogue OpenAPI representations are in turn based on the available standard ETSI NFV SOL005 v2.4.1 OpenAPIs. However, in accordance with the deliverable D3.2, the following enhancements (and differences) are applied:

- NSD Management API
    - The PNFD related resources are not allowed to be created, updated or patched: POST, PUT, PATCH operations are therefore not included for collective and individual resources. This is aligned with the assumption defined in deliverable D3.2 for avoiding the direct and automatic onboard of PNFs from the 5G EVE portal
    - Extensions:
        - Usage of “userDefinedData” key-value pairs attribute in CreateNsdInfoRequest for specifying in which sites to onboard an NSD
            - Example:

```
. . .
“userDefinedData”: {
                "ITALY_TURIN": "yes",
                "SPAIN_5TONIC": "yes"
}
. . .
```

        - Usage of “userDefinedData” key-value pairs attribute in NsdInfo for identification (e.g. in NSD GET operations from the 5G EVE portal) where an NSD is available (i.e. which sites)
            - Example:

```
. . .
“userDefinedData”: {
                "FRANCE_PARIS": "yes"
}
. . .
```

        - Usage of “userDefinedData” key-value pairs attribute in PnfdInfo for identifying (e.g. in PNFD GET operations from the 5G EVE portal) where a PNFD is available (i.e. which sites)
            - Example:

```
. . .
“userDefinedData”: {
                "GREECE_ATHENS": "yes"
```

```
}
. . .
```

- VNF Package Management API
  - VNF Package related resources are not allowed to be created, updated or patched: POST, PUT, PATCH operations are therefore not included for VNF Package and VNFD collective and individual resources. This is aligned with the assumption defined in deliverable D3.2 for avoiding the direct and automated onboard of VNFs from the 5G EVE portal down to the sites
  - Extensions:
    - Usage of "userDefinedData" key-value pairs attribute in VnfPckgInfo for identification (e.g. in VNF Package GET operations from the 5G EVE portal) where an VNFD is available (i.e. which sites)
      - Example:

```
. . .
"userDefinedData": {
        "SPAIN_5TONIC": "yes"
        "GREECE_ATHENS": "yes"
}
. . .
```

3.1.3 Service description

With respect to the high-level Multi-site Catalogue software architecture described in D4.1 [6], this document reports the first release of the software prototype, which covers part of the overall catalogue features. As said, the Multi-site Catalogue software prototype is implemented on top of the Nextworks' 5G Apps and Services Catalogue, which is therefore the software reference baseline. In particular, this D3.3 software release is providing a subset of the software extensions defined in section 3.1.1, mostly targeting the enhancements to the Catalogue Service for supporting the multi-site onboarding workflows, and the enhancements to the northbound REST APIs and ETSI NFV SOL005 data model[2].

For what concerns the Catalogue Service, this D3.3 software release provides the implementation and support of the following Multi-Site Catalogue onboarding workflows (as defined in D3.2):

- Catalogues synchronization: this version of the Multi-site Catalogue implements the collection of all VNF Packages, VNFDs and NSDs from the different site orchestrators and catalogues, and maintain them in the common enhanced ETSI NFV SOL005 and SOL001 formats. This synchronization is a procedure that is triggered by the Catalogue Service at the start of the Multi-site Catalogue service (as a kind of initial synchronization), and then is kept as a periodic process to ensure that the Multi-site Catalogue is aligned with the per-site catalogues.
- NSD onboarding workflow: this version of the Multi-site Catalogue allows to onboard new NSDs from the 5G EVE Portal, and specify the sites where the NSD should be onboarded to run the related experiments (after the proper adaptation and translation to the specific site data model).
- NSD removal workflow: this version of the Multi-site Catalogue allows to remove NSDs previously onboarded from the 5G EVE Portal, specifying the site where the NSD should be deleted.

Additional workflows will be developed for the next releases of the I/W framework prototype.

In terms of NFVO southbound drivers, this D3.3 software release of the Multi-Site Catalogue includes an OSM driver (that is compatible with OSM R4, R5, and R6) that allows to integrate with the Italian and Spanish site facilities (since both of them make use of OSM as site orchestrator). Additional NFVO southbound drivers (e.g. for adapting and translating ONAP API logics and data model) will be developed for the next releases of the I/W framework prototype. If required, specific drivers will be also implemented to integrate with commercial orchestrators, e.g. from Ericsson and Nokia in the Italian and the Greek sites.

[2] Multi-site Catalogue software prototype: https://github.com/nextworks-it/5g-catalogue/tree/v3.0

3.2 Multi-Site Inventory

The Multi-Site Inventory is the component in charge of maintaining the status of the Network Services instantiated in any of the 5G EVE sites. As described in D3.2 [2], the Multi-site Inventory is fully managed by the Multi-site Network Service Orchestrator, who is in charge of notifying of all the changes that have to do with service provisioning.

In this delivery we include the Multi-Site Inventory features integrated in the Multi-Site Network Orchestrator. In future releases it is expected to delegate all the Inventory catalogue to the Multi-Site Inventory component.

3.2.1 Open API description

The API of the Multi-Site Inventory can be found at: https://github.com/5GEVE/OpenAPI/tree/v0.2/MSI

At the moment of the delivery, only the status of the Network Service (NS) deployed by the 5G EVE Portal are available, including the status of each nested NS that are part of the NS definition. In Table 1 we show the services provided by the Multi-Site Inventory.

Table 1: Services provided by Multi-Site Inventory

Service	Path	Method	Input	Output	Description
Multi-Site Inventory	/nslcm/v1/ns_instances/{nsInstanceId}	GET	nsInstanceID	NsInstance	Retrieve the status of a NS

3.3 Multi-Site Network Orchestrator

The Multi-Site Network Orchestrator (MSNO) provides the Network Services Lifecycle Management API to the 5G EVE Portal. The MSNO orchestrates the Network Services through multiple site orchestrators.

3.3.1 Open API description

The API published by the Multi-Site Network Orchestrator is available at:

https://github.com/5GEVE/OpenAPI/v0.2//MSNO

It contains the full ETSI NFV SOL 005 Network Service LCM API, as defined in the standard. Table 2 states the services supported in this delivery.

Table 2: Services provided by Multi-Site Orchestrator

Service	Path	Method	Input	Output	Description
Multi-Site Network Orchestrator	/nslcm/v1/ns_instances	POST	CreateNSRequest	NsInstance	Create a new Network Service in the MSNO.
Multi-Site Network Orchestrator	/nslcm/v1/ns_instances	GET	-	Array of NsInstance	Returns the information of all NS available at MSNO/MSI
Multi-Site Network Orchestrator	/nslcm/v1/ns_instances/{nsInstanceId}/instantiate	POST	InstantiateNsRequest	Location	Instantiate a NS in the local NFV-O

3.3.2 Service description

In this delivery, the Multi-Site Network Orchestrator provides a subset of the Network Service Lifecycle Management API described above. The goal is to support the minimum set of services that allows to deploy a single-site scenario of a 5G EVE Use Case.

The following services are provided:

- List of Network Services: Provides the list of Network Services onboarded in the I/W Framework
- Information of a Network Service (via Multi-Site Inventory)

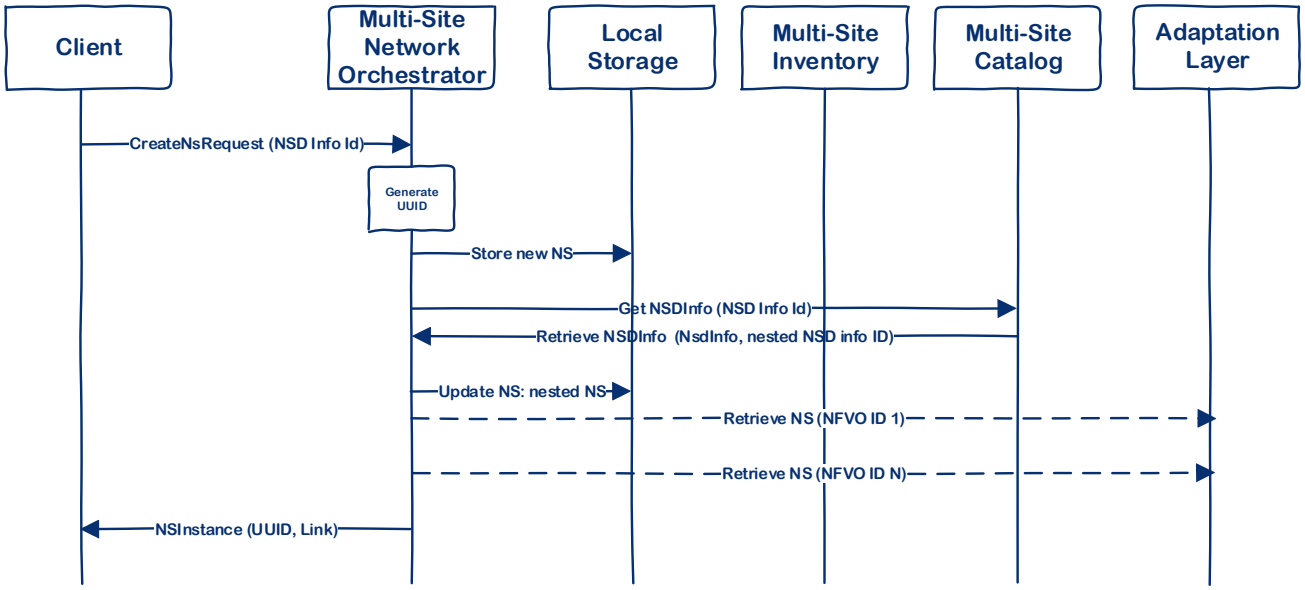


Figure 2: Workflow of the creation of a Network Service ID

- Creation of a Network Service Identification: It is the first step for deploying a Network Service and it is detailed in Figure 2. The main features implemented are:
    - Persistent storage (Local Storage) of the Network Service Information, shared between MSNO and MSI
    - Retrieval from the Multi-Site Catalogue of the NSD information
    - (optionally) Consistency check with local NFV-O (through Adaptation Layer)
- Instantiate a Network Service: Detailed in Figure 3, it instantiates a Network Service in the 5G EVE sites. The main features supported in this delivery are:
    - Instantiation of each nested NS in the correspondent site NFV-O
    - Transactional control of the operation

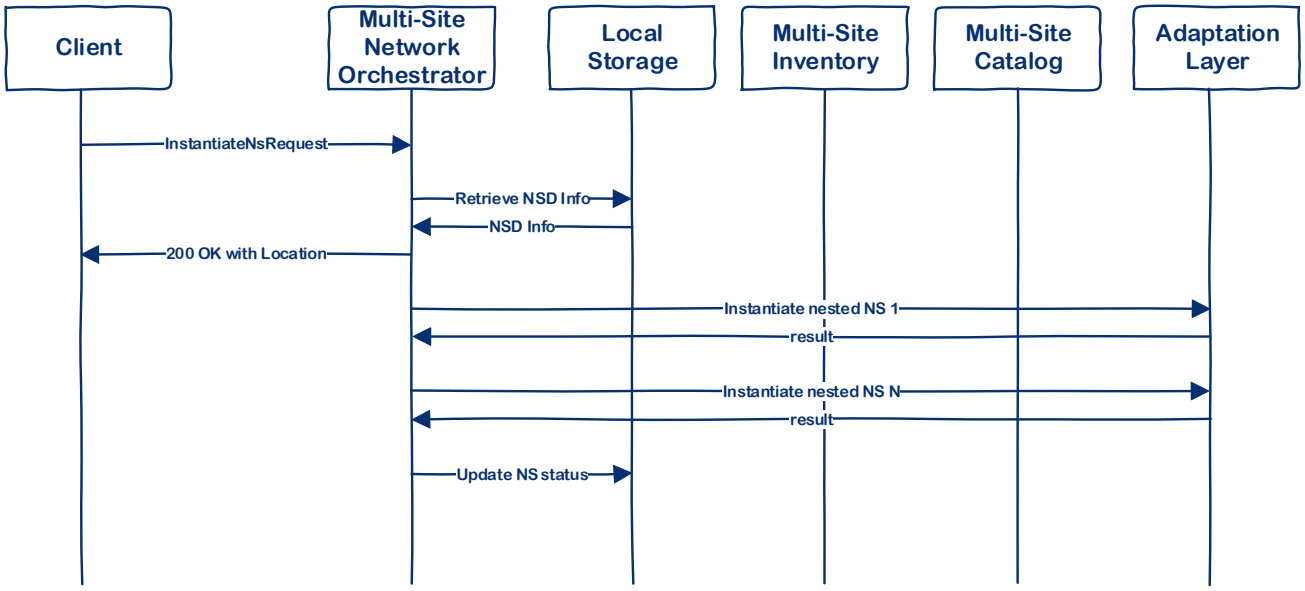


Figure 3: Workflow of NS instantiation

The services implemented have the following limitations:

- Notifications are not supported. They will be implemented in Drop 1
- A transaction cannot be resumed in case of MSNO crash. A persistent transaction model is planned for Drop 2
- Transaction rollback not implemented. Planned for Drop 2.

The deliverable of Multi-Site Network Orchestrator can be found at:

https://github.com/5GEVE/I-W_Framework/tree/v0.2/MSNO

3.4 Data Collection Manager

The Data Collection Manager component, as explained in D3.2 ([2]) – section 4.2.5, is 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, with two objectives: monitor the experiment and validate the targeted KPIs.

As a result, one specific feature that is necessary to be provided is the delivery of the monitoring information (being more precise: the metrics[3] gathered from the monitored components – VNFs, PNFs and other components – and the KPIs and results inferred from these metrics) to the components that are going to consume that data. For that reason, the Data Collection Manager is a central component in the 5G EVE architecture, as it has to obtain the necessary metrics from site facilities, and store and deliver them to the components on demand. These components include: The Portal developed in WP4, and the KPI Analysis and Validation framework developed in WP5.

Note that this component has had different updates regarding the proposed architecture, operations, information model and workflows. These changes have been mentioned in Annex A.

[3] This concept replaces the "data" term presented in D3.2 – section 4.2.5.1. Currently, "logs" term has the same meaning than explained in that section, but now it will be used "metrics" instead of "data" for talking about the processed logs, after using a transformation function (called "log-to-metric transformation"), which are delivered and used by upper layers. Data shipper components (presented in D3.2 – section 4.2.5.2) will be responsible for implementing that log-to-metric transformation function.

3.4.1 Software architecture

The Data Collection Manager software architecture will be based on Apache Kafka[4], which is a well-known publish-subscribe tool that acts as a bus, being able to manage data and send them to different applications connected to it.

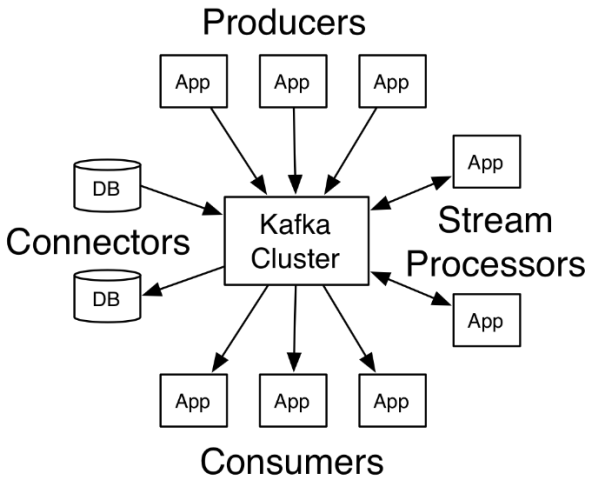


Figure 4: Apache Kafka's main APIs.

Moreover, Kafka is able to replicate data among different nodes, which can be useful for a cluster deployment. In fact, there are different deployment modes for achieving that purpose:

- **Cluster mode:** data from different topics is replicated within a cluster.
- **Mirroring mode:** data from different topics is replicated to another Kafka cluster. This solution is interesting in case it is necessary to have separate Kafka clusters (e.g. for managing different data in each cluster).

Some interesting services offered by Kafka are the following:

- **Kafka Streams:** is a library for building streaming applications, specifically applications that transform input Kafka topics into output Kafka topics (or calls to external services, or updates to databases, or whatever).
- **Kafka Store:** Kafka also offers the possibility of persisting data with storage services.
- **Kafka ACL:** for controlling who access to a specific topic and when, ACLs can be defined as well.

These services are described in Figure 5.

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

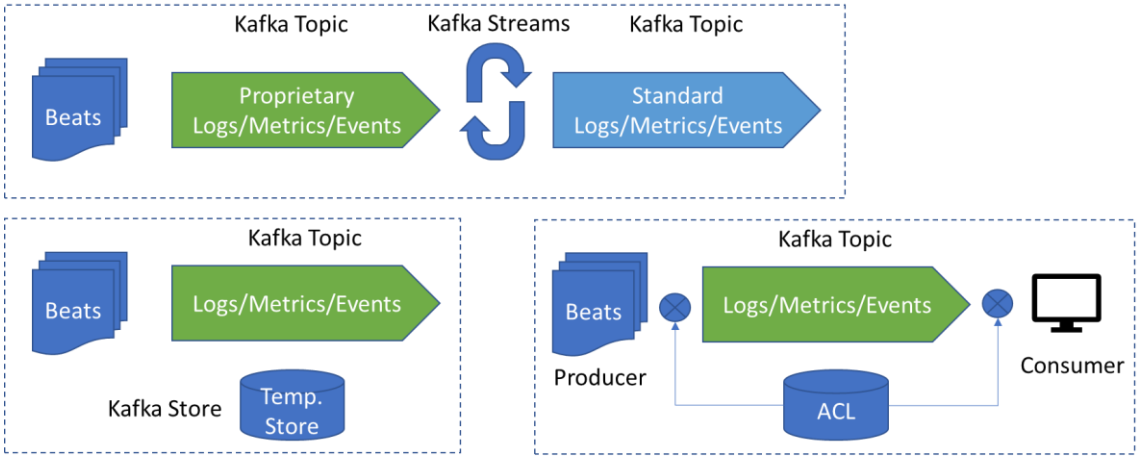


Figure 5: Main services and functionalities offered by Kafka.

The access to the Kafka bus from upper layers is also transparent, hiding the presence of different site facilities to these layers, which is the main objective to be achieved with the use of the Interworking Layer. This can be achieved with an intermediate layer to provide seamless access to upper components, which can also provide the capabilities of defining, maintaining and updating topics.

Currently, the implementation that has been already done regarding the Data Collection Manager is a dockerized environment for testing the 5G EVE Monitoring & Data Collection tools from WP4, which includes a specific Kafka container for performing the publish-subscribe queue functionality. That environment is available in the 5G EVE repository[5].

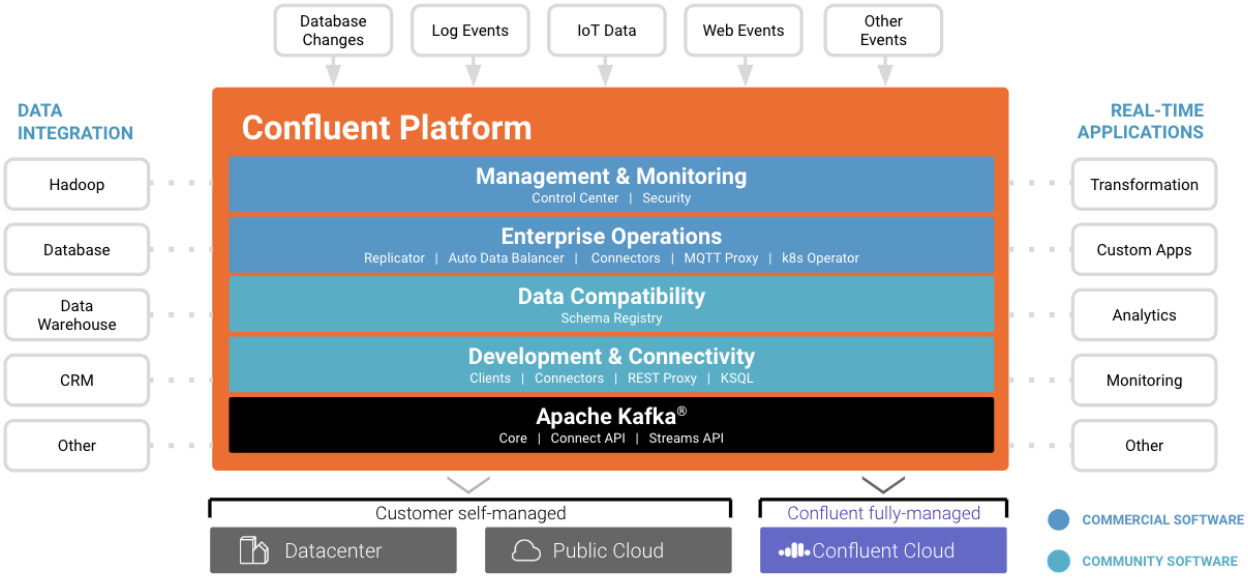


Figure 6: Kafka Confluent components (from https://www.confluent.io)

However, the solution that is intended to be deployed is a clustered Apache Kafka environment, coordinated by a dedicated tool such as Apache ZooKeeper[6], and complemented with a management platform like Kafka

[5] https://github.com/5GEVE/5geve-wp4-monitoring-dockerized-env

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

Confluent[7], which provides, among other utilities, a REST API proxy for accessing to the Kafka cluster without using the native Kafka protocol. This complete architecture is expected to be finalized for the next D3.4 deliverable. Moreover, this Kafka Confluent platform also provides several clients[8]for enabling different functionalities that can be considered for the future. An example of an architecture which covers these software requirements is presented in Figure 6, which belongs to the Kafka Confluent documentation. For Data Collection Manager, it will be used the Community version (open source), and also combining Apache Kafka with Apache ZooKeeper to enable a distributed coordination and consensus system in the cluster. The final specification, including other main functionalities and capabilities such as security mechanisms, will be provided in the next D3.4 deliverable.

This architecture can also be complemented with other technologies for offering cold storage or fast queuing as improvements, related to the enhanced architecture already presented. This configuration is not currently available and will be studied for future deliverables.

3.4.2 Open API description

As a first approach, the Data Collection Manager main API will be based on the native Kafka protocol[9], as it is a well-known protocol that is present in lots of production environments for managing the exchange of a huge amount of data between lots of components. The API is based on the Kafka publish-subscribe implementation, supporting the three main operations presented before (subscribe, unsubscribe and publish – which triggers the deliver operation too). In fact, practically all the components which are connected to this Data Collection Manager are able to use that protocol or can be adapted somehow in order to use it (e.g. by including an application logic with programming languages which include Kafka libraries, such as Python).

However, in case of having specific components that are not capable of using the Kafka protocol, the Kafka Confluent platform provides a REST Proxy[10] component in order to deal with that issue. Note that it does not cover all the functionalities that Kafka manages with its native protocol, but the main operations that will be used by the Data Collection Manager are defined in the specification.

The Open API specification of the REST Proxy component (which has been obtained from the Kafka Confluent documentation) has been included here:

https://github.com/5GEVE/OpenAPI/tree/v0.2/DataCollectionManager

An example of possible operations supported by this REST Proxy component related to the operations presented in Annex A.2 (subscribe, unsubscribe and publish) are presented in Table 3 (the final operations to be used will be defined in the next D3.4 deliverable).

Table 3: Example of Data Collection Manager operations from its Open API specification.

Service	Path	Method	Input	Output	Description
Data Collection Manager	/consumers/{group_name}/instances/{instance}/subscription	POST	group_name (name of consumer group) instance (ID of consumer instance)	204 –success 404 – consumer not found 409 – subscription	Subscribe to the given list of topics or a topic pattern to get dynamically assigned partition. If a prior subscription exists, it would be replaced by the latest

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

[8] https://docs.confluent.io/current/clients/index.html and https://cwiki.apache.org/confluence/display/KAFKA/Clients

[9] https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol

[10] https://docs.confluent.io/current/kafka-rest/index.html

			Body with either topic subscription or topic pattern subscription schema.	mutually exclusive	subscription (subscribe operation)
Data Collection Manager	/consumers/{group_name}/instances/{instance}/subscription	DELETE	group_name (name of consumer group) instance (ID of consumer instance)	204 –success 404 – consumer not found	Unsubscribe from the topics currently subscribed (unsubscribe operation).
Data Collection Manager	/topics/{topicName}	POST	topicName message(s) to produce to the given topic	200 – message(s) posted 404 – topic not found 422 – invalid request	Post messages to a given topic (publish operation)

Nevertheless, to support the workflows presented in Annex A.3, it is necessary to chain several operations present in both the Kafka protocol and the REST API in order to achieve the specific purpose of each operation. Specifically, each operation must do the following actions:

- **Subscribe:** when the Data Collection Manager receives the list of topics to be subscribed:
  - It will check if each topic has been previously created in the Kafka cluster. So, for each topic:
    - In case it was created, it will do nothing.
    - In case it was not created, it will create it and start consuming from that topic.
- **Unsubscribe:** it is the opposite case. When the Data Collection Manager receives the list of topics to be unsubscribed:
  - It will check if each topic has been previously created in the Kafka cluster. So, for each topic:
    - In case it was not created, it will do nothing.
    - In case it was created, it will stop consuming from that topic and delete it afterwards.
- **Publish:** when the message is received in the Data Collection Manager through a specific topic, it will be automatically delivered to the components subscribed to that topic, as long as Kafka is consuming from that topic (the topic must exist). In other case, the operation would be rejected.

3.4.3 Service description

In this section, the architecture proposed in D3.2 – section 4.2.5.2 will be extended with the introduction of the Topic framework definition and the extension of the basic workflow to be followed in the Data Collection Manager operation (mentioned in D3.2 – section 4.2.5.2). Architecture's update from D3.2 is described in Annex A.1.

3.4.3.1 Topic framework proposal

The topic framework represents the information model that will be used to define the data that will be handled by the Data Collection Manager in a concurrent way. In this way, each topic defined, which is identified by a unique ID, will manage a specific set of data that will be different to the information consumed by the other topics, allowing the isolation between datasets.

In this first proposal, two main types of topics which will be used by the Data Collection Manager have been identified:

- **Signalling topics:** these topics are used to communicate specific topics which will be used to transport the data related to the metrics obtained and provided by the Data shippers, or the results and KPIs calculated by the Results Analysis and Validation component, in order to execute the corresponding subscribe or unsubscribe operations, depending on the stage in which these signalling topics are involved. In this current proposal, there have been identified three signalling topics, whose identifiers are:
  - **signalling.metric:** signalling topic for metric's data.
  - **signalling.kpi:** signalling topic for KPI's data.
  - **signalling.result:** signalling topic for result's data.

  The reason of having a different signalling topic for each kind of data managed by the Data Collection Manager is due to the need of isolation between the information exchanged in the Publish-subscribe queue. As a result, for example, the *signalling.metric* topic will only handle topics related to metrics, and will assure that the components subscribed to that signalling topic will only receive information regarding metrics, and not related to results or KPIs.
- **Data topics:** they transport the value for the metrics/KPIs/results that are identified in the topic ID, followed by some extra information (depending on the data; e.g. the VNF/PNF ID) in case they were needed for later processing. The generic format of this kind of topic is the following: ***<exp>.<site>.[metric|kpi|result].<value>***, where:
  - ***<exp>:*** is the experiment ID, identifying the experiment (e.g. industry4.0, etc.).
  - ***<site>:*** identifies the site facility where the data is extracted (e.g. Spain, Italy, etc.).
  - ***[metric|kpi|result]:*** it is a key word that indicates the type of information to which belongs the next parameter. For example, if the data which is being consumed in the topic is a metric, the value of this field will be "*metric"*.
  - ***<value>:*** set the name of the parameter to be monitored (e.g. latency), identifying the metric/KPI/result.

In case there is a need to manage a new source of information by the Data Collection Manager (e.g. data from the orchestrator), a new set of signalling and data topics are needed in order to achieve that, and also updating the corresponding workflows for including that new exchange of information.

The specific structure of data that will be handled by each kind of topic is briefly presented in Annex A.2 with the update of the Data Collection Manager's NBI and SBI, but it is not the main objective of this deliverable, so it will be further described in D3.4.

3.4.3.2 Updated basic workflow

In this section, it is updated the basic workflow defined in D3.2 – section 4.2.5.2, having this sequence of high-level operations (which are described with more depth in Annex A.3):

i. Data shipper components will gather the logs from the components which belong to a given experiment and that are liable to be monitored. They will then perform the log-to-metric transformation, in order to transform the heterogeneous, raw logs obtained from components and collection tools into metrics with a common, homogeneous format, previously agreed in the blueprint's definition. These data shippers can be placed within each component as a lightweight software or can be in a separated server, but in both cases, they must be connected to the Data Collection Manager with a logical connection (or "queue"). As mentioned in D4.1, the responsible for configuring correctly the data shippers is the experiment developer, but dynamic information such as the configuration parameters for the connection to the Data Collection Manager, in case of not being defined before the experiment, can be included as Day-2 configuration by the Runtime Configuration component (as long as that configuration is defined in the test case).

ii. When test cases are executed in site facilities, the different components used during the experiment (both physical and virtual) generate logs (raw information) that are captured by the data shippers and transformed into metrics. Leveraging the connection with the publish-subscribe queue (again, by publishing into a topic), metrics can be delivered to the Data Collection Manager.

iii. This new workflow avoids the necessity of compute resources (mainly storage) in each site facility in order to save the metrics before sending them to the Interworking Publish-subscribe queue, as the "publish" operation is performed automatically by Data shippers.

iv. All the metrics provided by Data shippers located in each site facility are gathered in a common queue (i.e. the Interworking Publish-subscribe queue), which also provides persistence for all the data published there. As different topics for a specific use case – site facility – metric combination are used, the information that arrives to the Data Collection Manager through a topic is automatically isolated from the other topics from a logical point of view, being also able to define control access policies for implementing more restrictive security procedures.

v. The metrics are received by the Results Analysis and Validation component in order to obtain the related KPIs and results from a specific metric, which are in turn published in the Data Collection Manager (in both cases, using specific topics for that purpose, and performing the necessary subscriptions for achieving that behaviour). Therefore, if other components (e.g. monitoring components in the Portal, experiment tools or the orchestrator itself) want to receive data (metrics, KPIs and/or results) managed in the Interworking queue, they only have to subscribe to the specific topic in which they can obtain the desired data. This is represented in Figure 15 and Figure 16 with the block called "transparent and seamless access to results, metrics & KPI values to other queues".

3.5 Runtime Configurator

The Runtime Configurator component is intended to apply tailored runtime configurations to the provisioned end-to-end VNFs and PNFs for supporting the Vertical use case experiments, acting as Day-2 configurator. It was briefly described in D3.2 – section 4.2.4, and its definition will be expanded in this deliverable, mainly regarding its interaction with the Experiment Execution Manager component and the subsequent connection to the components to be configured by it.

Note that this component has had different updates regarding the proposed architecture, operations, information model and workflows. These changes have been mentioned in Annex B.

3.5.1 Software architecture

The Runtime Configurator will be based on Red Hat Ansible Community Edition[11] open-source project. The main component of this platform is Ansible Core[12], which is an IT automation tool that can configure systems, deploy software and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

The configuration, deployment and orchestration language used by Ansible are represented by playbooks, which are the implementation of the templates commented in previous sections. These playbooks describe the instructions to be executed in order to manage configurations of and deployments to remote machines (i.e. the VNFs and PNFs to be configured).

The access to the Ansible component can be made by accessing to the server which holds the Ansible implementation (e.g. by using SSH protocol, for example) or by using the API provided by the AWX project[13], which is the open-source version of the commercial Red Hat Ansible Tower platform, and provides a web-based user interface, a REST API and a task engine built on top of Ansible.

Furthermore, the access to the configured components is achieved with OpenSSH as the main alternative, but also having other transports and pull modes as alternatives. For non-compatible components, another solution

[11] https://www.ansible.com/community

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

[13] https://github.com/ansible/awx

to make them reachable to Ansible is to provide the Proxy Component presented in Annex B.1, which will have an open connection with Ansible and will forward the configuration provided by Ansible to these non-compatible VNFs and PNFs. Possible functionalities and examples to be covered by the Proxy Component could be:

- A proprietary management system from a given vendor, which can be contacted by the Runtime Configurator in order to interact with non-compatible VNFs-PNFs. The vendors must be the responsible for deploying and managing these components.
- A solution to reachability problems (e.g. for cases where the VNFs/PNFs to be configured are isolated in private networks and needs to be reachable by the Runtime Configurator. This functionality may be also present for the last case (i.e. a management system isolated in a private network). These connectivity issues must be developed within 5G EVE project.

For the deployment of the Runtime Configurator, it has been agreed the interface between the Experiment Execution Manager and the Runtime Configurator and it has also been confirmed that OpenSSH protocol can be used in order to configure the components deployed for a given experiment, having the alternative of the Proxy Component in case of incompatibilities. However, the implementation of the specific platform (currently, it has been only tested in a local environment), operations and information model presented in Annex B.2, and workflows described in Annex B.3 and the deployment of the templates (playbook) to be used in the test are still in progress and they will be further described in future deliverables.

Other issues that must be solved in the following drops of the Interworking Framework implementation are the following:

- The configuration of VNFs and PNFs whose IP addressing is based on private networks which are not reachable by the Interworking Layer. In that case, as there are other components in the Interworking Layer that interacts with site facilities (e.g. the Network Service Orchestrator or the Data Collection Manager), there must be an access point where the Runtime Configurator can access to site facilities and can provide the configuration to the VNFs and PNFs; for example, the Proxy Component already mentioned.
- The specification of the Proxy Component. In theory, site facilities define a multi-tenant infrastructure which holds different experiments from different users. As a result, the definition of the virtual networks to be used by each experiment could be different among them in order to assure the isolation between use cases. Consequently, the Proxy Component must be configured for reaching all the virtual networks where there can be non-compatible components with Ansible technology, which is an implementation issue that can be achieved by many ways (e.g. deploying a Proxy Component for each virtual network, using one single Proxy Component which is connected to all virtual networks, or deploying a Reverse SSH Proxy as an alternative to direct SSH connection). The selection of the most appropriate way of performing this and the definition of the actors responsible for the management and operation of this Proxy Component will be also described in future deliverables.

3.5.2 Open API description

The main interface to be used between the Experiment Execution Manager and the Runtime Configurator will be based on the OpenSSH protocol, because it is intended to define direct calls between them exchanged through that protocol. However, as mentioned before, the AWX component provides a REST API in case of needing a RESTful interface for the access to the Runtime Configurator functionality.

That Open API specification is defined in the Ansible official documentation, which can be found here: https://docs.ansible.com/ansible-tower/latest/html/towerapi/api_ref.html. Furthermore, the correct reference to the swagger file containing that OpenAPI specification in the 5G EVE repository is the following one: https://github.com/5GEVE/OpenAPI/tree/master/RuntimeConfigurator

An example of possible operations supported by the AWX REST API related to the operations presented previously (mainly, executing a specific template) are presented in Table 4 (the final operations to be used will be defined in the next D3.4 deliverable).

Table 4: Example of Runtime Configurator operations from its OpenAPI specification.

Service	Path	Method	Input	Output	Description
Runtime Configurator	/job_templates	POST	Among others: • Name of the job template. • Playbook to be executed. • Extra variables to be used. • Timeout.	201 – accepted, with the job template ID. 403 – permissions' problem.	Create a job template.
Runtime Configurator	/jobs/{id}/create_schedule	POST	Job template ID.	201 – accepted.	Create a schedule that launches the job template that launched this job

3.5.3 Service description

Currently, the Runtime Configurator service description is still under discussion in order to close the definition of the necessary interactions and connections with the Experiment Execution Manager component. The current status of this is presented in Annex B and will be updated in future drops of the Interworking Framework.

3.6 Adaptation Layer

The Adaptation Layer at the SBI provides a common access interface to trial site services and resources. The implementation is organized in multiple software components depending on the specific feature to access (i.e., orchestration, configuration, monitoring).

A thorough presentation of requirements, features, and high-level design is reported in D3.2 [2].

The following subsections provide a description of the implementation process, software architecture and technologies for each part of the Adaptation Layer.

3.6.1 Multi-Site Catalogue SBI

The Multi-Site Catalogue described in Section 3.1 includes SBI operations for the management of NSDs, VNFDs and PNFDs in the local NFVOs. Furthermore, it features a synchronization mechanism to keep the Interworking Layer aligned with the information available in the local NFVO catalogues. To achieve this, the Multi-Site Catalogue is structured with a driver-based architecture. Each driver is in charge of the translation of the NSD, VNFD, and PNFD from the standard ETSI NFV SOL 001 [7] model to the specific model of the targeted NFVO type. Reverse translation is supported as well. The driver also implements the required adaptations to interact with the NSD Management API of the specific NFVO.

This D3.3 software release of the Multi-Site Catalogue includes an OSM driver (that is compatible with OSM R4, R5, and R6) that allows to integrate with the Italian and Spanish site facilities (i.e. both of them make use of OSM as site orchestrator). Additional NFVO southbound drivers (e.g. for adapting and translating ONAP API logics and data model) will be developed for the next releases of the I/W framework prototype. If required, specific drivers will be also implemented to integrate with commercial orchestrators, e.g. from Ericsson and Nokia in the Italian and the Greek sites. The alternative will be that all sites deploy at least one instance of OSM or ONAP, independently of the presence of these commercial orchestrators.

3.6.2 Multi-Site NSO to local Orchestrators interface (MSO-LO)

The Multi-Site NSO to local Orchestrator interface (MSO-LO) is the portion of the Adaptation Layer that enables the Interworking Layer to access the orchestration features provided by the trial sites involved in the 5G EVE project. Trial sites can expose one or more local Network Function Virtualization Orchestrator (NFVO), each one managing a different set of resources. The specification of features, requirements, and a textual description of the API's methods is provided in D3.2 [2].

This interface provides the following features:

- Operations to retrieve information about local NFVOs registered with the 5G EVE platform.
- Operations to retrieve, create, instantiate, scale, terminate, and delete NS instances.
- Operations to retrieve, create, and delete subscriptions to notifications about the status of one or more NS instance.

3.6.2.1 Software architecture

The MSO-LO must provide a common interface to access the features of various NFVO types (e.g., OSM, ONAP) without restrictions on open-source or proprietary licenses. The only constraint is for the NFVO implementation to provide the complete specification of its North-Bound Interface (NBI), possibly in OpenAPI format. To support the NFVO implementations declared at the moment in the 5G EVE project and to support the inclusion of new NFVO technologies in the future, we designed the software with a driver-based structure as depicted in Figure 16.

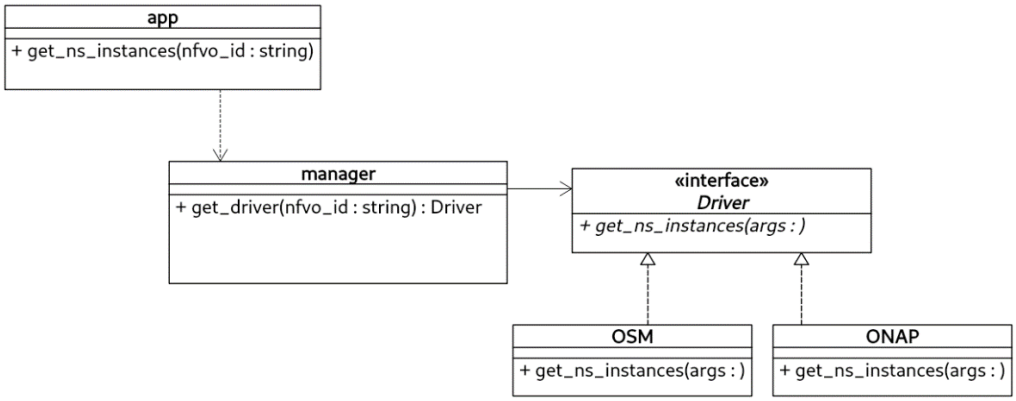


Figure 7: UML class diagram with just one example method of the interface.

Figure 7 shows a simplified version of the UML class diagram with just one method of the API, which is enough to explain how the software works. As we use Python for the software implementation, not all the classes shown in the diagram are Python classes. Some entities are just modules, as they do not need to store any status but are a mere collection of static methods.

The *app* module contains the declaration of the REST API paths and related HTTP operations. Furthermore, it includes the mapping between the previous entities and the relevant Python method. The method implementation here is generic and agnostic from the specific type of local NFVO.

The other entities in the figure implement the factory method design pattern [8]. The pattern is very effective for our use-case as it separates the creation of driver instances from the *app* module that actually use them. Indeed, the *app* module depends (dashed arrow) from the *manager* module for the selection of the specific NFVO implementation. The *manager* module implements the factory method plus some utility methods for the interaction with the NFVO database.

The *Driver* abstract class realizes a contract for the driver implementation and is implemented by means of the Abstract Base Class Python module (ABC, [9]). The addition of a specific NFVO driver simply consists in the extension of *Driver* and in the implementation of all its declared methods.

To better understand the interactions between the modules and class instances we use a message sequence chart. Figure 8 shows the interactions needed to obtain the list of NSD from the catalogue of an OSM instance NFVO named ‘osm_1’.

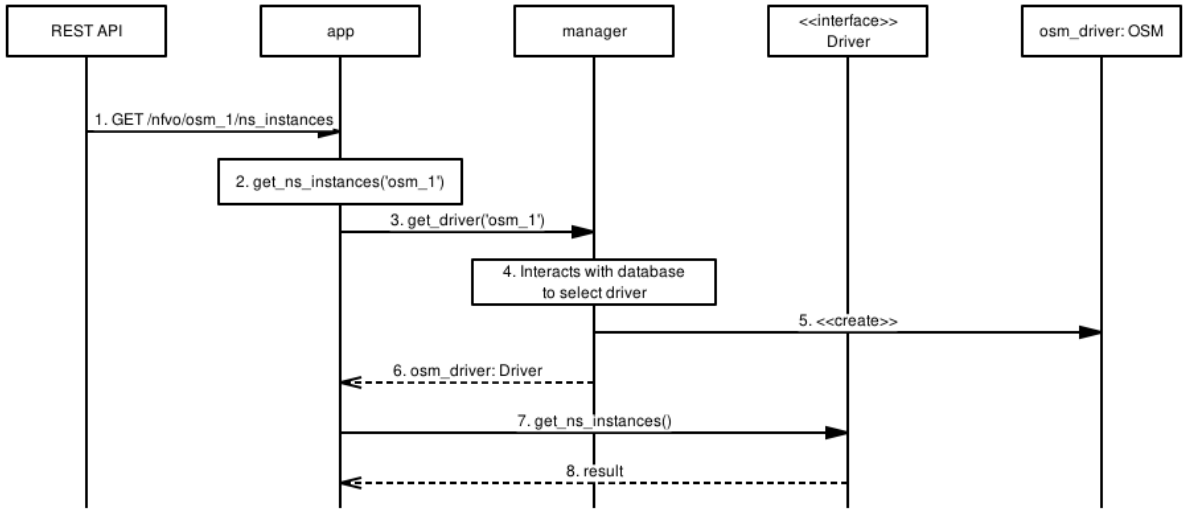


Figure 8: Message sequence chart to show modules and objects interactions.

At step 1, a GET request to the path “/nfvo/osm_1/ns_instances” is routed to the *app* module. The *app* module executes the relevant method (step 2). At step 3, the *app* module requests the appropriate driver to the *manager*. The *manager* determines the appropriate driver (step 4), creates an instance of the OSM driver (step 5), and returns it to the *app* module (step 6). The *app* module actually receives an object of generic type *Driver*. In step 7, *app* calls the relevant method and in step 8 it receives the result.

At the moment, the *manager* creates a new driver instance for each request. This can of course be improved with the association of a driver instance to each NFVO instance that is contacted. The feature needs to store and track objects and it is planned for a future stage of the development.

3.6.2.2 Storage, database

In addition to supporting multiple types of local NFVOs, we also need to interact with multiple instances of them. To support this requirement, we implement a registry to track registered NFVOs and related information, as IP address and credentials. The registry is designed as a simple relational database. Figure 9 shows the table diagram of the database.

In this first implementation the database is composed of four tables:

- NFVO: each record contains the general information about a local NFVO
- NFVO_CREDENTIALS: each record contains the credentials needed to interact with the NBI of the local NFVO
- NS_SUBSCRIPTION: each record contains the subscription information to receive notifications about the status of one or more NS instances.
- NS_INSTANCE: each record contains the association of a NS instance to the subscription for notifications about its status.

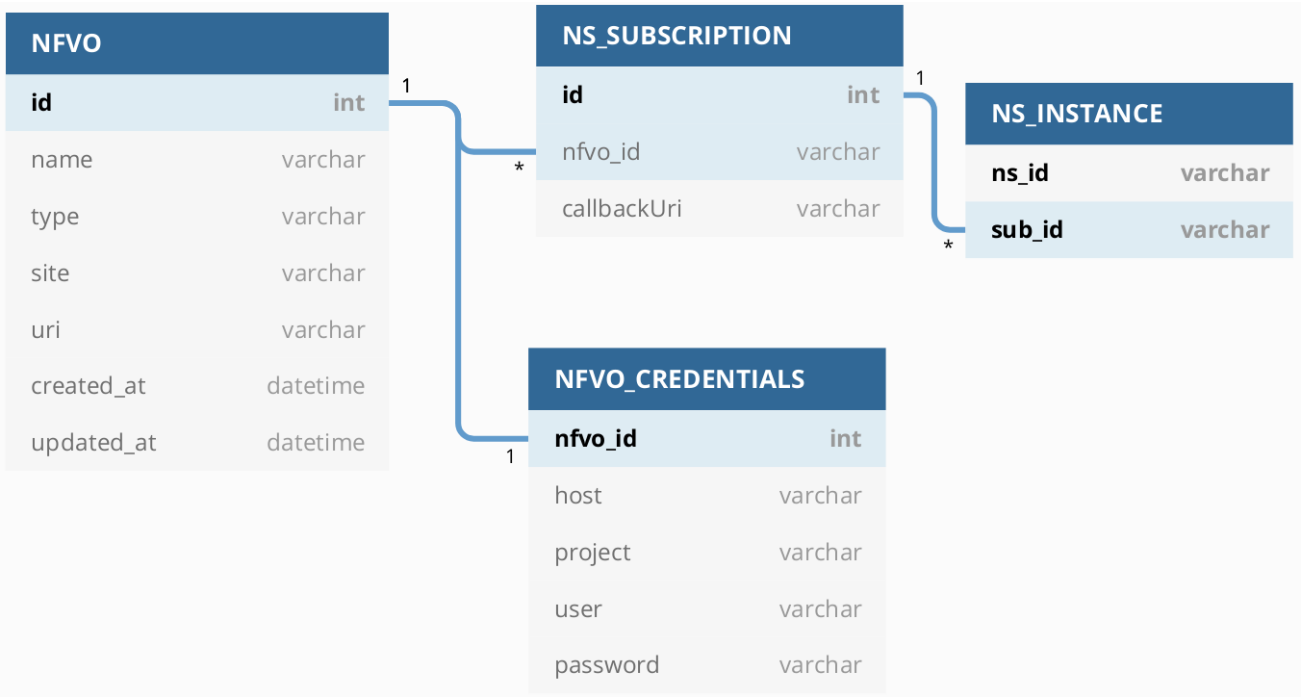


Figure 9: Table diagram of the database for the local NFVO registry.

The relationship between NFVO and NFVO_CREDENTIALS is 1-to-1 and all the information could be merged in the same table. Anyway, since NFVO_CREDENTIALS stores sensible data we decided to keep it separate to avoid accidental leaks due to programming errors.

Because the data model is extremely simple and the database is not going to be accessed by a large number of users, we have discarded the selection of a client/server SQL database engine for the implementation. We have decided to use SQLite [10], a database engine that "strives to provide local data storage for individual applications and devices". SQLite provides SQL features as well as ACID transactions. The database is completely contained in a single disk file with support to sizes up to the terabyte. The API is simple to use and it has great integration with python. We believe SQLite can widely support our use-case requirements, while keeping the application architecture as simple as possible.

3.6.2.3 Application architecture

The MSO-LO is a web service that exposes to NBI based on the REST architectural style. To implement this module, we have mainly used tools for the API description / documentation and tools for the webservice implementation. Due to the used technology, a reverse proxy is used to bundle all communication between MSO and the MSO-LO backend.

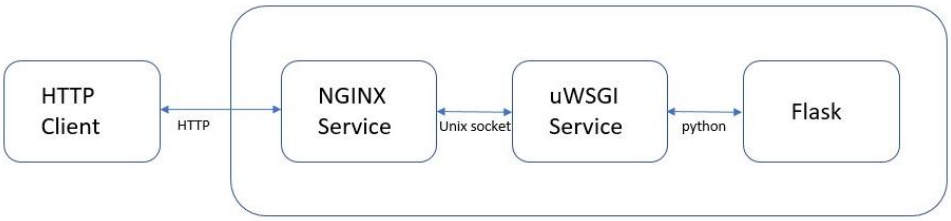


Figure 10: Architecture of the application

As shown in Figure 10 the architecture of the application is composed by the following components:

- Reverse proxy: we used the popular NGINX [11]
- WSGI server: we used uWSGI [12] typically used for running Python web applications

- Webservice: we used the lightweight micro-framework Flask [13] written in python, that is designed for a quick and easy "getting started", with the ability to scale up to complex applications

Then the execution flow is the following: a generic HTTP Client initiates a request by NGINX that forwards the request to uWSGI using a unix socket. Then uWSGI forwards the request to Flask that handles the request and generates the response. The response is passed back though the stack.

3.6.2.4 Open API description

The API has been described in D3.2. As a standard we reference the ETSI NFV-IFA 013 [14] and we extend it in order to support the management of the information about local NFVOs (e.g., name, location, radio coverage, capabilities, etc.) and to fully define the subscription and notification system (partially defined by the standards). For the implementation we adopt and extend the proposed implementation described in ETSI NFV-SOL 005 [3]. We try to adhere to the proposed standard as much as possible with similar paths, request bodies, and responses.

The API description / documentation has been realized with Swagger [15], an online tool to describe API adhering to the OpenAPI Specification [16]. The API is available as a YAML file in the 5G EVE public organization hosted on GitHub [17]. Table 5 summarizes the API methods offered by MSO-LO interface.

Table 5: Services provided by MSO-LO interface

Service	Path	Method	Input	Output	Description
MSO-LO	/nfvo	GET	-	Array of NFVO info	Retrieve list of local NFVO.
MSO-LO	/nfvo/{nfvoId}	GET	-	NFVO info	Read an individual NFVO.
MSO-LO	/nfvo/{nfvoId}/ns_instances	POST	nsdId, nsName, nsDescription	NS identifier	Creates and returns a Network Service identifier (nsId) in a Nfvo
MSO-LO	/nfvo/{nfvoId}/ns_instances	GET	-	Array of NS instances	Retrieve list of NS instances.
MSO-LO	/nfvo/{nfvoId}/ns_instances/{nsInstanceId}	GET	-	NS instance	Read an individual NS instance resource.
MSO-LO	/nfvo/{nfvoId}/ns_instances/{nsInstanceId}	DELETE	-	-	Delete an individual NS instance resource.
MSO-LO	/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate	POST	nsFlavourId	-	Instantiate a NS instance.
MSO-LO	/nfvo/{nfvoId}/ns_instances/{nsInstanceId}/scale	POST	scaleType, scaleVnfData, scaleNsData	-	Scale NS instance.

MSO-LO	/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/terminate	POST	terminationTime	-	Terminate a NS instance.
MSO-LO	/nfvo/{nfvoId}/subscriptions	POST	filter, callbackUri	subscription identifier	Subscribe to NS lifecycle change notifications.
MSO-LO	/nfvo/{nfvoId}/subscriptions	GET	-	Array of subscription information	Query multiple subscriptions.
MSO-LO	/nfvo/{nfvoId}/subscriptions/{subscriptionId}	GET	-	subscription information	Read an individual subscription resource.
MSO-LO	/nfvo/{nfvoId}/subscriptions/{subscriptionId}	DELETE	-	-	Terminate a subscription.

3.6.3 Runtime Configurator SBI

As described in Section 3.5, the Runtime Configurator is in charge of applying Day-2 configurations on the VNF instances deployed in the trial sites. To achieve this goal, the configuration management software Ansible is used. As VNFs are usually Linux-based system, an SSH connection is needed to run commands and apply configurations with Ansible.

The Runtime Configurator SBI is therefore realized by the SSH protocol. Given a deployed NS instance, SSH connections or tunnels are dynamically established between the Interworking Layer and the VNF instances deployed in the local trial sites to apply Day-2 configurations during the experiment execution.

This approach has the advantage of being simple and based on a well-known, highly maintained, and secure protocol as SSH. At the same time some challenges arise (also presented in Section 3.5): e.g. not all VNF instances can be reached directly from the Interworking Layer. In fact, probably no VNFs will have a public IP address in the management plane, so the solution must go in line with using the VPN connections among sites in order to provide access to local management networks to the Interworking Layer. Only in cases where there is equipment in management networks not published in 5G EVE, other solutions must be proposed; e.g. the use of a reverse proxy.

3.6.4 Data Collection Manager SBI

The Data Collection Manager has been presented in Section 3.4and its task is to collect data about the experiment execution. It is based on Kafka, a well-known and widely used publish-subscribe broker providing a lot of clients to interact with [20].

On the SBI, the Data Collection Manager must be able to receive data from the VNF instances realizing the NS of the experiment. This is achieved by equipping all the required VNFs in 5G EVE with a data shipper including a Kafka client, e.g. the Elastic Beat [21] component.

The solution is simple but effective. Anyway, if some VNF cannot include the data shipper for whatever reason, Kafka can be extended with a REST API thanks to an external plugin: the Confluent REST Proxy [22]. It has some limitations with respect to the native Kafka protocol, but it supports all the basic operations like topic subscription, message publications, and message consumption.

3.7 Site adaptations

All the commands executed by Adaptation Layer's components need to be translated to a form understandable by local orchestrators. Each of the Sites in the project requires a special driver that allows communication between the MSO-LO interface end-points. As different Sites use different implementations of a Local Orchestrator, different drivers are needed to be implemented (for instance: ONAP driver, OSM driver, etc.). A thorough mapping of functionalities to OSM and ONAP operations is reported in D3.2 [2].

The following subsections provide a description of the implementation process, software architecture and technologies for each Site of 5G EVE platform.

3.7.1 French site

The French Site facility is deployed in various French cities with a central Management and Orchestration (MANO) tool (local orchestrator) based in Chatillon, Paris, in Orange's headquarter. The commercial and pre-commercial experiments have been deployed locally at companies' premises at Paris, Lannion and Sophia Antipolis.

The ONAP solution has been selected as a main tool of management and orchestration in French Site. It is the major point of contact with I/W Framework. It creates a kind of a gateway and umbrella over all the experiments developed by the French companies. ONAP is an open-source, complete solution that provides a comprehensive platform for real-time, policy-driven service orchestration and automation. In the French Site we attach four different VIMs to be orchestrated by ONAP (Openstack instances, Kubernetes, OpenShift). ONAP is also a platform with full range of life-cycle management operations that can be steered by Runtime Configurator of I/W Framework. To fully integrate ONAP and the I/W Framework, a special adapter needs to be implemented. The adapter will translate requests made by I/W Framework components to the form understandable by ONAP via using the ONAP external North Bound Interface.

In D3.3we will focus on some operations that basically allow to integrate ONAP with I/W Framework services like Service Orchestration, Runtime Configurator or Multi-Site Catalogue. It allows to run first experiments from the level of 5G EVE Portal. The first delivery of the MSO-LO interface (that provides operations for the management of NS and VNF instances) will focus on VNF onboarding and initialization.

The very first prototype released in this D3.3 should provide the adaptation and translation features towards the ONAP APIs like:

- Onboard NSD /nbi/api/serviceSpecification
- Retrieve NSD /nbi/api/serviceSpecification
- Onboard VNFD /nbi/api/serviceSpecification
- Retrieve VNFD /nbi/api/serviceSpecification
- Instantiate NSI /nbi/api/serviceOrder
- Retrieve list of NSI /nbi/api/serviceOrder

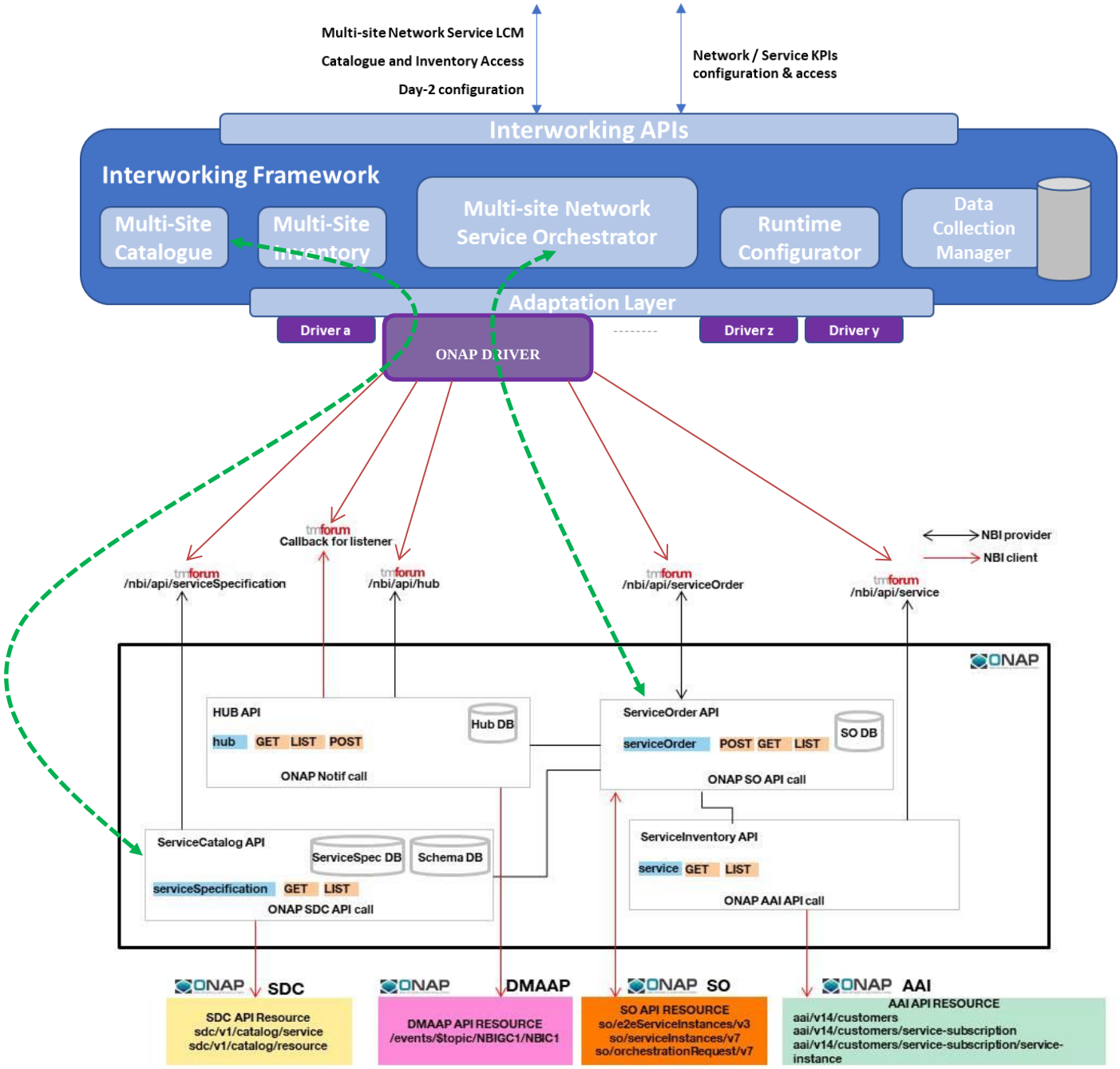


Figure 11: Architecture of the ONAP driver with ONAP NBI

3.7.2 Greek site

For Greece site, we are analysing the possibility of reusing the ONAP approach followed in French site. At the moment of closing this deliverable, the decision is not confirmed and for that is not included here.

Following, we include the current interface exposed by the site

3.7.2.1 Nokia CloudBand Application Manager APIs

Interworking is based on NOKIA CloudBand Application Manager (CBAM) APIs and that performs the required functionalities. The following main operations belong to VNF lifecycle management:

The Nokia CBAM is a VNF manager that automates VNF lifecycle management and cloud resource management. CBAM has standards-based APIs to allow it to work with any vendor VNF, EMS, VIM, or NFV Orchestrator (NFVO).

The NFM-P provides an interface with CBAM, which acts as the VNF manager in this solution. The NFM-P uses the Ve-Vnfm-Em reference point to exchange notifications on VNF lifecycle changes and monitor virtual resources with CBAM. The Network Supervision application allows you to configure a CBAM access point to monitor managed VNFs and execute lifecycle management actions.

The NFM-P provides the following functions when interfacing with CBAM:

- element management for VNFs
- VNF assurance, alarm monitoring, and status tracking
- VNF KPI monitoring
- lifecycle management proxy actions
- policy-based lifecycle changes

CBAM access point

The NFM-P supports CBAM integration through a CBAM access point created in the Network Supervision application. It is necessary to input CBAM and NFM-P login credentials in order to create the access point. It is necessary to also specify an NFM-P discovery rule to enable VNF automatic discovery. This procedure requires administrator or nfvMgmt access privileges.

Once the access point is successfully created, you can view a list of CBAM access points associated with the NFM-P. From the list of CBAM access points, you can cross-launch to the CBAM GUI, rescan CBAM VNFs, or open the Details tab for an access point. The Details tab allows you to view the associated discovery rule and the connection status for the specified CBAM access point. The connection status is verified by the NFM-P once every two minutes.

The following are prerequisites before creating a CBAM access point:

- Ensure the specified CBAM login credentials have access to the CBAM APIs via ReST. See the *Installing CloudBand Application Manager Guide* for more information.
- Ensure the NFM-P SSL certificates are installed on CBAM. See the *CloudBand Application Manager Administrator Guide* for more information.

VNF discovery

It is not possible to instantiate VNFs using a CBAM interface. We can only discover VNFs from CBAM using an NFM-P discovery rule specified during access point creation. The NFM-P requires that the VNFD template for discovered VNFs have post-instantiation scripts to enable SNMP and configure other protocols necessary for automatic node discovery. When the NFM-P discovers VNFs from a CBAM access point, it adds them as a rule element for the associated discovery rule.

The Unmanaged VNFs tab lists the VNFs managed by the CBAM instance that were not discovered by the specified NFM-P discovery rule. The list is updated once every two minutes.

Lifecycle change notifications

Lifecycle change notifications (LCNs) are messages sent from CBAM to the NFM-P with details on VNF lifecycle updates. When the CBAM access point is created, the NFM-P requests two different LCN subscriptions for the CMG and CMM. LCNs are used to inform the NFM-P of changes related to VNF instantiation, termination, scaling, healing, or variable modifications. When the NFM-P receives an LCN, it scans the VNF information from the CBAM access point and updates its VNF database accordingly.

Regardless of LCNs, the NFM-P automatically polls the CBAM access point for VNF updates once every hour.

VNF lifecycle management

Certain VNF lifecycle changes can be initiated from CBAM or the Network Supervision application. Whenever a lifecycle change is triggered in CBAM, it informs the NFM-P via an LCN.

VNFs can be instantiated in CBAM and advertised to the NFM-P via an LCN. When the NFM-P receives information on a newly instantiated VNF, it creates an associated VNF object and attaches a discovery rule to that object automatically. When the NFM-P discovers a VNF, it retrieves information related to supported operations, scaling and healing templates, extensions, and compute resources.

VNFs can be terminated in CBAM and advertised to the NFM-P via an LCN. When the NFM-P receives information on a terminated VNF, it unmanages the VNF object and removes all associated VNFCs.

VNFs can be deleted in CBAM and advertised to the NFM-P via an LCN. When the NFM-P receives information on a deleted VNF, it removes the VNF from its database and unmanages the associated network element.

VNFs can be healed to trigger a reboot in CBAM or the Network Supervision application. Healing must be enabled in the CBAM VNFD before this operation can be performed in the GUI. If the VNFD requires additional parameters for VNF healing, the parameters are visible in the Network Supervision application.

VNFs can be scaled in or scaled out in CBAM or the Network Supervision application. Scaling must be enabled in the CBAM VNFD before this operation can be performed in the GUI. When performing a scaling operation, it is necessary to specify a scaling aspect and a new level. The scaling level cannot exceed the maximum scaling level specified in the CBAM VNFD. If the VNFD requires additional parameters for VNF scaling, the parameters are visible in the Network Supervision application

3.7.3 Italian site

The Italian site facility is deployed in the city of Turin, where a test lab environment is running in TIM headquarters for the verification of components and functionalities of 5G NSA architecture. In parallel, 5G technologies provided by Ericsson are being deployed in different sites in Turin for the experimentation of Smart Transport and Smart City use cases.

For what concerns the NFV orchestration tools, two solutions are envisaged to coexist for the lifecycle management of Network Services and VNFs in the Italian site facility: an open source OSM orchestrator, and an Ericsson orchestrator. In particular, the OSM orchestrator is mostly dedicated to the coordination of the vertical VNFs (i.e. those providing the specific Use Case applications) and related Network Services lifecycle. For what concerns the coordination and lifecycle management of the 5G/4G network related VNFs and functionalities (e.g. EPC, 5G Core, RAN), a dedicated orchestrator is planned to be provided by Ericsson, and to be integrated with the I/W framework and the 5G EVE platform at a later stage.

Therefore, in this D3.3 delivery, for what concerns the multi-site Network Service lifecycle management, including the management of Network Service and VNF descriptors, the Italian site adaptation features mostly focus on the translation of the ETSI NFV SOL and IFA APIs, workflows and data models towards those of OSM.

As said in previous sections, the Multi-site Catalogue embeds part of the I/W framework adaptation layer features targeting the management of NSDs, VNFDs and PNFDs. In practice, the Multi-site Catalogue is equipped with specific per-site orchestrator drivers that adapt the unified catalogue northbound APIs and translate the common TOSCA based data models into those specific of each local per-site orchestrator. In particular, the Multi-site Catalogue prototype released in this D3.3 delivery provides the adaptation and translation features towards the OSM APIs and data model, as follows:

- Adaptation of NSD Management Multi-site Catalogue APIs into OSM NSD Management API
- Implementation of catalogue synchronization workflow (see D3.2) with OSM
- Translation of TOSCA NSD into OSM NSD, and viceversa
- Translation of TOSCA VNFD into OSM VNFD, and viceversa

On the other side, the MSO-LO interface provides operations for the management of NS and VNF instances deployed in local orchestrators of different type. The API implementation is inspired by the ETSI NFV SOL 005 specification to ensure a common and standardized interface. Following a modular design, a driver implements the adaptation and translation toward the specific orchestrator interface. In this release, a prototype driver for OSM is included supporting the following features:

- Adaptation of NS instance management Multi-site Catalogue APIs into OSM NS instance management API. Operations to retrieve, create, instantiate, scale, terminate, and delete NS instances are supported
- Preliminary implementation of a subscription and notification system. This feature is not available in OSM and it is realized in the driver.

3.7.4 Spanish site

Spanish site has a similar approach to the Italian site for the orchestration. As agreed during the WP3 discussions, we are going to reuse the OSM site adaptation driver for the OSM deployment in Spanish site, which is in charge of orchestrating the VNF deployed in the cloud services. In a future, we might include a driver for the Ericsson technology to orchestrate the 5G Core network

For the connection to the Data Collection Manager component, it is only needed that the Data shippers are configured correctly with the IP addressing and port used by the Data Collection Manager and also the topics to be used for the publication of metrics. That configuration can be defined by the experiment developer in advance or provided by the Runtime Configurator during the experiment execution. For achieving this, the Data Collection Manager must be reachable from the site facility.

In the case of the Runtime Configurator, it must be confirmed that the technology proposed (based on Ansible platform) is able to reach all the possible components that can be deployed for a given experiment and that are intended to consume Day-2 configuration. After analysing the possible constraints in that communication, the Proxy Component might be configured and deployed consequently.

4 Inter-site connectivity status

This section is focused on updating the site facilities interconnection solution described in D3.2 with all the new decisions that have been agreed since then. The key aspects covered in D3.2 related to the inter-site connectivity status are the following:

- Identified requirements for sites interconnection, inherited from D3.1 with some specific corrections.
- Potential technologies that can be used to interconnect the site facilities among them and with the I/W Framework, connecting the different interconnection planes.
- Basic topologies to be considered (star and mesh topologies, mainly), in which it should be distinguished both control and data plane, whose scope and requirements are slightly different:
    - Control plane: the main site hosting the portal and the I/W Framework (assuming they are placed in one of the site facilities, but this is not a specific requirement) must be connected to the remaining of the site facilities.
    - Data plane: all sites should be able to reach the remaining sites preserving the constraints enunciated in both D3.1 and D3.2.
- Example of a possible interconnection based on the French site facility.
- The interconnection proposal itself, which will be extended below.
- Additional requirements (covered in the section 6 of the D3.2). Some examples are:
    - Vertical's remote access to deployed VNFs.
    - Internet access for deployed VNFs.
    - Monitoring and Support platform.

  In this scope, it is also proposed the usage of a centralized IdM for all the services provided by the site facilities to end users, in order to manage the access to these different services from the users.

Furthermore, there are other aspects which can impact on the final solution proposed which are still under discussion and evaluation by WP2, such as the addressing plan (including the orchestration, data and control plane) to be used in each site facility for this inter-site connectivity and with the I/W Framework, or the possibility of joining certain ICT-19 projects infrastructure to 5G EVE facilities, and which will be defined in future deliverables (starting from a brief list of requirements which was already included in D3.2).

4.1 Updated Inter-site connectivity proposal

It will be updated the interconnection proposal provided in D3.2 with new considerations and requirements that must be considered. The proposal is based on the following architecture:

- The topology selected for inter-site connectivity would be, initially, a star topology for both control and data planes, having the main site located at Turin, due to its geographical centralized position. As mentioned before, this central hub is intended to host I/W Framework.
- Connectivity will be based on S2S IPsec tunnels among sites, using a routable private plan (e.g. using virtual routers).

This solution would be the first phase in the interconnection between site facilities, and it is depicted in Figure 12.

Figure 12: Site facilities interconnection – first phase.

The main advantage of this solution is that it makes the inclusion of new nodes easier. However, there are some risks related to the existence of new possible nodes that come from close to existing locations or the link capacity and resilience in Turin site facility. Moreover, once this setup is up and running, there should be specific monitoring tools to collect the interconnection metrics which will allow to check whether the setup copes with the requirements identified in D3.1, which are completed in D3.2 – section 5.1.

Taking into consideration the advantages and disadvantages of the first phase solution, and also considering that, in case the performance requirements are not met with the above solution, an evolution of the last proposal has been also proposed, identified as the second phase for site facilities interconnection, whose architecture is depicted in Figure 13.

Figure 13: Site facilities interconnection – second phase

This type of interconnection has the following characteristics:

- The topology selected for inter-site connectivity would be a star topology for control plane and a full or partial mesh for data plane (as the required paths are not optimized in a star, and the required performance is higher than in the control plane). The main site would be also located at Turin, due to its geographical centralized position, but it could be any other.
- Again, connectivity will be based on S2S IPsec tunnels among sites, probably using a routable private plan too.

Leveraging in a different topology for the control and data plane has many advantages, which are: the avoidance of potential bottlenecks and the reduction of the latency in the data plane, but having the risks of having difficulties with scaling in the mesh topology or increasing the operations overhead.

5 Updated roadmap

According to the proposal (Figure 14), the 5G EVE project has two official deliveries of the I/W framework: the delivery described in this document and one in M24 with the full I/W framework.

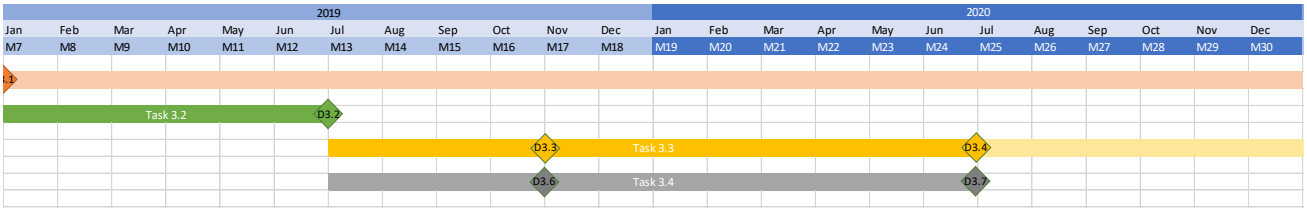


Figure 14: Task plan for WP3 for the 5G EVE second year

In Table 6 we show the plan of supporting each of the features included in D3.2, based on WP3 plan and the 5G EVE project roadmap published in WP1.

Table 6: 5G EVE Interworking Framework Roadmap

Key Features	Category	Brief Description	D3.3 (M16)	Drop 1 (M18)	Drop 2 (M22)	D3.4 (M24)
Single-Site scenarios						
Orchestration Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for orchestration traffic	Italian and French sites	Full provided	Full provided	Full provided
Single-site Experiment Monitoring Support	Monitoring	Capability of translating the monitoring requirements defined by experimenters (based on selected KPIs) to the requested site. Sites will typically have different local monitoring tools and mechanisms.	-	Support of basic UC (one per site)	Full provided	Full provided
Single-site Applications Deployment Support	Operation	Capability to deploy the required VNFs, hosted in the 5G EVE Catalogue, at the requested site. Sites will typically have different local orchestrators.	-	Support of basic UC (one per site)	Full provided	Full provided
Single-site Network Automation Support	Operation	Capability to deploy the required connectivity services (first phase) and slices (second phase) to the requested site. Sites will typically have available different local controllers and network infrastructure.	-	Support of basic UC (one per site)	Full provided	Full provided
Multi-Site scenarios						
Control Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for control traffic	-	-	Demons trated with one selected UC	Full provided
Data Plane Interworking	Connectivity	Secure connectivity among sites for user traffic. Low bandwidth performance experiments will employ best effort connectivity. High bandwidth performance experiments will	-	-	-	Depends on final UC requirements

		employ a parallel high bandwidth low latency network, which will be available at least between two sites.				
Multi-Site Experiment Monitoring support	Monitoring	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.	-	-	Selected UC working in multi-site deployment	Full provided
Multi-Site E2E Orchestration Support	Operation	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 orchestrators, controllers and network infrastructure.	-	-	Selected UC working in multi-site deployment	Full provided
Multi-Site slicing						
Vertical Slicing	Slicing	End-to-end slice that satisfies the requirement of the Vertical	-	Selected UC	Full provided	Full provided
Multi-Site Slicing	Slicing	Vertical Slice that spans across multiple sites	-	-	Selected UC	Full provided

6 Conclusions

This document describes the deliverable D3.3, which is the first software delivery of the Interworking Framework. The main content of the software delivery are the APIs definitions, including both external and internal interface definitions, which are available at the project repository[14] following the standard de-facto of OpenAPI. By making public all the interfaces we contribute to the openness and modularity of the 5G EVE Framework, which is one of the main objectives of this project. As we explained, all interfaces try to follow as much as possible the existing standard that in our case are: ETSI NFV SOL 001, SOL 003 and IFA 013. Where the standard does not arrive, we extend it or we propose an implementation, as we do for the Adaptation Layer northbound interface. The delivery also includes the Multi-Site Network Orchestrator and Multi-Site Catalog components, with a subset of the target services, features and capabilities described in previous deliveries (D3.1 and D3.2). For those components not included in the delivery we describe the current design status, with all the internal architecture and workflows.

In addition, in this document we updated the status of the inter-site connectivity, which is one of the pillars of the Interworking Framework. We also include the roadmap of the Interworking Framework, where we define two extra deliveries, called Drop 1 and Drop2, that we identify as necessary for supporting the milestones and deliveries of this project.

[14] https://github.com/5GEVE/OpenAPI/tree/v2.0

Acknowledgment

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

References

[1] 5G EVE D3.1: Interworking capability definition and gap analysis document, available at https://www.5g-eve.eu/wp-content/uploads/2019/01/5geve-d3.1-interworking-capability-gap-analysis.pdf

[2] 5G EVE D3.2: Interworking reference model, available at https://www.5g-eve.eu/wp-content/uploads/2019/09/5geve_d3.2-interworking-reference-model.pdf

[3] "ETSI GS NFV-SOL 005 V2.6.1 (2019-04)" [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.06.01_60/gs_NFV-SOL005v020601p.pdf

[4] 5G Catalogue, available at https://github.com/nextworks-it/5g-catalogue

[5] 5G Media, available at www.5gmedia.eu

[6] 5G EVE D4.1: 5G-MEDIA Catalogue APIs and Network Apps, available at http://www.5gmedia.eu/cms/wp-content/uploads/2018/09/5G-MEDIA-D4.1-5G-MEDIA-Catalogue-APIs-and-Network-Apps_v1.0.pdf

[7] ETSI GS NFV-SOL 001 NFV descriptors based on TOSCA specification, available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/001/02.05.01_60/gs_NFV-SOL001v020501p.pdf

[8] E. Gamma, "Design patterns: elements of reusable object-oriented software," *Pearson Education India*, 1995.

[9] "abc - Abstract Base Classes," [Online]. Available: https://docs.python.org/3/library/abc.html.

[10] "SQLite Home Page," [Online]. Available: https://www.sqlite.org/index.html?.

[11] "nginx," [Online]. Available: http://nginx.org/.

[12] "uWSGI," [Online]. Available: https://uwsgi-docs.readthedocs.io/en/latest/.

[13] "Flask Web Framework," [Online]. Available: https://palletsprojects.com/p/flask/.

[14] "ETSI GS NFV-IFA 022 V3.1.1 (2018-04)" [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/022/03.01.01_60/gr_NFV-IFA022v030101p.pdf

[15] "Swagger," [Online]. Available: https://swagger.io/.

[16] "OpenAPI Initiative," [Online]. Available: https://www.openapis.org/.

[17] "AdaptationLayer OpenAPI," [Online]. Available: https://github.com/5GEVE/OpenAPI/tree/v0.2/AdaptationLayer

[18] 5G EVE D4.1: Experimentation tools and VNF repository (https://www.5g-eve.eu/deliverables/)

[19] 5G-Ensure, "Deliverable D3.6 5G-PPP security enablers open specification," 2017.

[20] "Clients – Apache Kafka" [Online]. Available: https://cwiki.apache.org/confluence/display/KAFKA/Clients

[21] "Beats: Data shippers for Elasticsearch" [Online]. Available: https://www.elastic.co/products/beats

[22] "Confluent REST Proxy" [Online]. Available: https://docs.confluent.io/current/kafka-rest/index.html

Annex A – Data Collection Manager update from D3.2

In this section, the architecture proposed in D3.2 – section 4.2.5.2 will be extended with an updated version of the requirements and capabilities that the Data Collection Manager must comply. Moreover, the operations and information models described in D3.2 – sections 4.4.4.1 (NBI) and 4.5.5 (SBI) and the Data Collection Manager workflows presented in D3.2 – section 4.3.3.6 will be updated, due to the change on the architecture and the workflows of the Data Collection Manager.

A.1 Updated architecture

According to D3.2 – section 4.2.5.2, the technology envisioned to build the Data Collection Manager architecture is the publish/subscribe messaging paradigm. However, this document describes some changes in more detail. As a result, Figure 15 depicts the representation of the updated Data Collection Framework in which the Data Collection Manager component is involved.

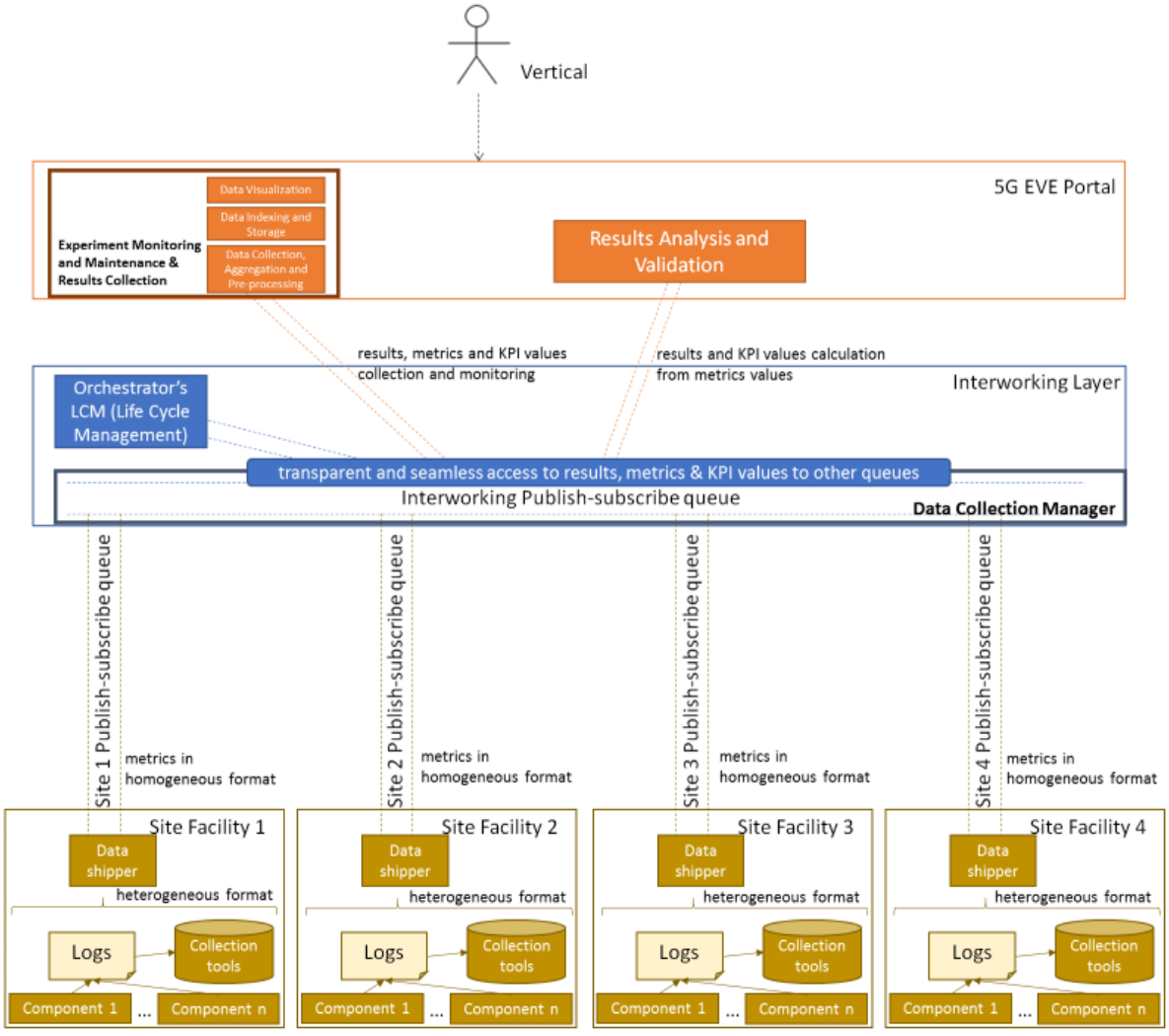


Figure 15: Improved version of the Data Collection Framework.

The changes, redefinitions and new concepts which have been included in this update are the following:

- All the publish-subscribe queues presented in the previous diagram are referred to the logical connection which is established between specific components (e.g. Data shippers, Results Analysis and Validation,

etc.) and the Data Collection Manager. In order to achieve this connection, these components must be configured properly in order to exchange information through the publish-subscribe mechanism implemented by the Data Collection Manager.

- The logical connection aforementioned will be based on topics, which are the information unit in charge of separating and isolating the information that is consumed by the Data Collection Manager. The information model based on topics that will be used in this operation is defined in a separate subsection (3.4.3.1). Note that the Data Collection Manager is able to handle the information managed by multiple topics at the same time.
- The Experiment Definition block defined in D3.2 – Figure 18 has been replaced by the Results Analysis and Validation block (from WP5) as described in Figure 15, whose main objective is to validate the test execution according to the metrics received from the monitored components, also calculating the KPIs and results in order to perform that validation. These KPIs and results can also be published into the Data Collection Manager in case they were required by other components of the global architecture (e.g. for monitoring purposes within WP4 scope). Therefore, the Data Collection Manager module would not only handle processed metrics (as mentioned in D3.2), but also handle KPIs and result values provided by the Results Analysis and Validation module.
- According to the specification provided in D4.1 ([18]), the Data Visualization block presented in D3.2 – Figure 18 has been replaced by the Experiment Monitoring and Maintenance & Results Collection, whose objective is to collect all the information (metrics, KPIs and results) related to monitoring purposes, apply filters and pre-processing functions to them, and finally represent them in an intuitive GUI.
- Figure 15 presents a connection between the Orchestrator's LCM and the Data Collection Manager, which means that it is possible to connect other components to the publish-subscribe queue as long as they are able to communicate by using the same protocol. Currently, this integration is not available, but it could be easily integrated if required. Other components which could leverage from monitoring purposes could be some components integrated in the site facilities, such as the local orchestrators or the VIMs, in order to take intelligent decisions regarding the location of certain functions (VNFs) in site facilities for a multi-site experiment, deploying these functions in the most suitable site facility depending on the resource utilization in each site. In the previous examples, note that the Data Collection Manager would only offer the capability of collecting and delivering information between connected components, but other actions (like the application logic that make these "more intelligent decisions") must be implemented in other specific components.
- Potential capabilities that could be included in this general architecture (presented in D3.2 – section 4.2.5.2 and Figure 19), can be also integrated, maintaining all the assumptions done. However, the fast data processing queue (using of the in-memory data storage in the Data Collection Manager server(s) for processing the data that needs fast processing) does not need to have a new connection established towards the Data Collection Manager. The usage of that in-memory data storage would depend, in fact, on the topic framework proposed (e.g. having specific topics which will be used by the in-memory storage). As a result, in Figure 16 these additional capabilities and changes to accommodate this fact have been included.

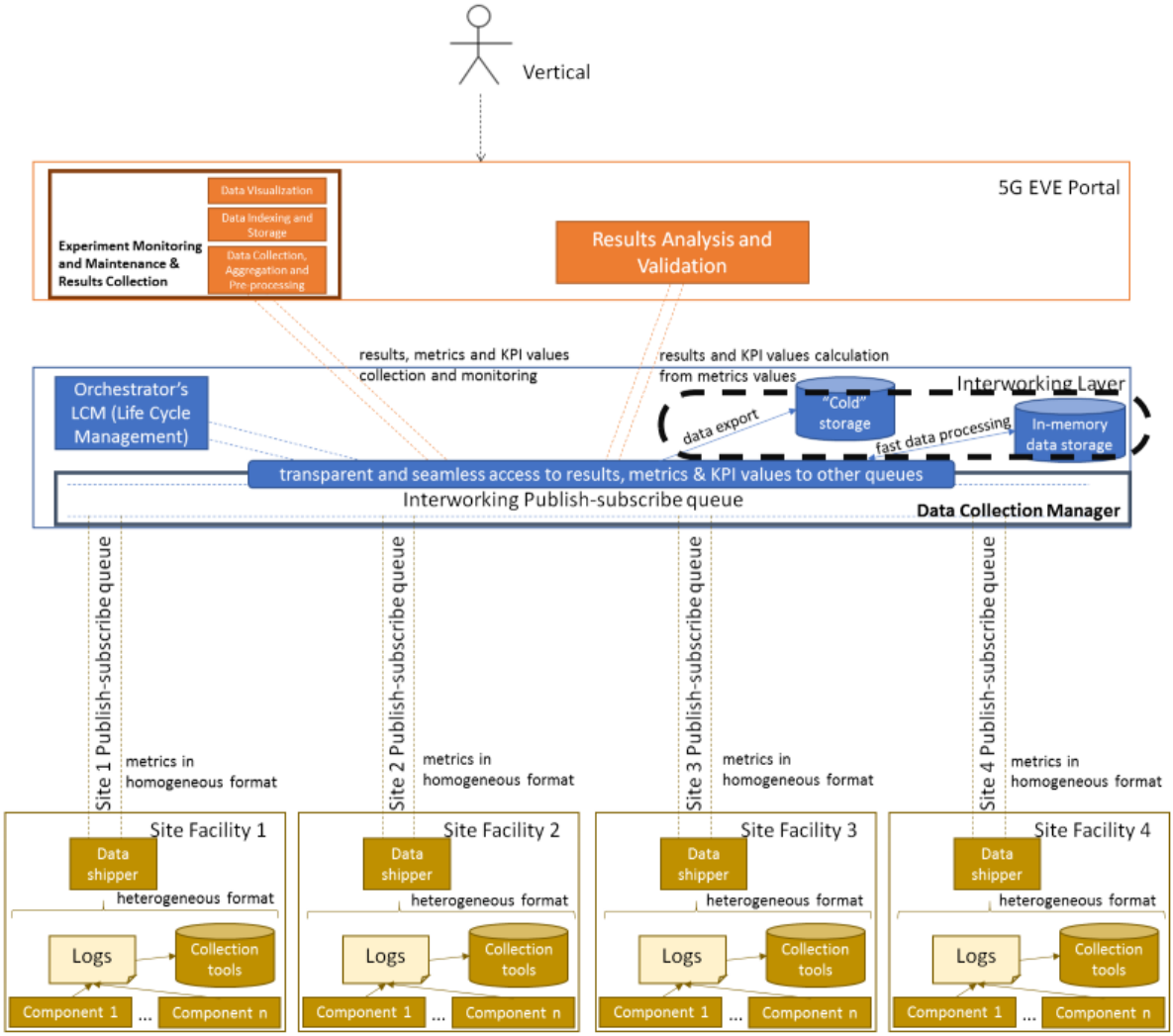


Figure 16: Enhanced Data Collection Manager architecture.

A.2 Updated operations and information model

As mentioned in D3.2, there will be three main operations regarding the Data Collection Manager: **subscribe** to a topic, **publish** data into a topic (which will automatically trigger a delivery operation – deliver – of that data to the components subscribed to that topic) and **unsubscribe** from a topic. As a result, the NBI and SBI operations are depicted in Table 7 and

Table 8.

Table 7: Data Collection Manager NBI

Data Collection Manager NBI	
Management of experimentation data	
Description	Application-independent access to monitored experimentation data obtained during the execution of use cases. Monitored data can be related to network conditions in general, or Network/Service metrics, KPIs and results. The former will be valid to ensure that conditions are adequate for the execution of the test. The later will also be used for validation purposes.

Reference Standards	Interface based on Publish/Subscribe model. Regarding network monitoring: RMON (RFC2819, extensions and updates), Syslog (RFC5424, extensions and updates), SNMP (RFC1157, extensions and updates). Regarding testing: NGMN Alliance "Definition of the testing framework for the NGMN 5G pre-commercial networks trials".	
Operations Exposed	***Information Exchanged***	***Information Model***

Topic subscription (subscribe)	• Topic(s) to be subscribed, classified in two main categories:
	  ○ Signalling topics.
	  ○ Data topics, related to different metrics, KPIs and/or results obtained from a specific use case.
		A list of topics to be subscribed, formatted in a JSON chain. The name to be used for each topic is defined in the Topic framework (which is still under discussion).

Topic un-subscription (unsubscribe)	• Topic(s) to be unsubscribed, with the same two categories aforementioned.	Same as topic subscription operation but including the topics to be unsubscribed.

Publish data (publish)	• Topic in which you want to publish the data.
	• Data to be delivered in the message. The data content depends on the topic category:
	  ○ Signalling topics: transport the list of topics to which the components subscribed to a specific signalling topic will be eventually (un)subscribed. In order to decide what operation should be done, there is an action field whose value can be either subscribe or unsubscribe, depending on the operation chosen.
	  ○ Data topics: transport the value for the metrics/KPIs/results and some extra information in case they were needed.
	As a result of this operation, the Data Collection Manager performs the delivery of the data (deliver) provided in the publish operation through the defined topic to the components subscribed to that topic.
		The information model for the data provided would depend on the topic category. For signalling topics, it will include the list of topics to be (un)subscribed and a action field with the operation to be executed afterwards (subscribe or unsubscribe). For metrics/KPIs/results topics, its definition would depend on the experiment developer, as it is the actor responsible for the definition of the mechanisms and procedures for obtaining data from the monitored components deployed in a experiment. That issue is currently under discussion.

Table 8: Data Collection Manager SBI

Data Collection Manager SBI		
Collection of metrics		
Description	Data collection from the different components deployed in the site facilities for a given experiment.	
Reference Standards	Interface based on Publish/Subscribe model. Regarding network monitoring: RMON (RFC2819, extensions and updates), Syslog (RFC5424, extensions and updates), SNMP (RFC1157, extensions and updates). Regarding testing: NGMN Alliance "Definition of the testing framework for the NGMN 5G pre-commercial networks trials".	
Operations Exposed	***Information Exchanged***	***Information Model***

Publish data (publish)	• Metric topic in which you want to publish the data. • Data to be delivered in the message related to the monitored metric. As a result of this operation, the Data Collection Manager performs the delivery of the data (deliver) provided in the publish operation through the defined topic to the components subscribed to that topic.	The information model would depend on the experiment developer, as it is the actor responsible for the definition of the mechanisms and procedures for obtaining data from the monitored components deployed in a experiment. That issue is currently under discussion.

Note that the operations themselves have not changed, but the information exchanged and the information model have been updated with all the considerations already commented.

A.3 Updated specific-purpose workflows

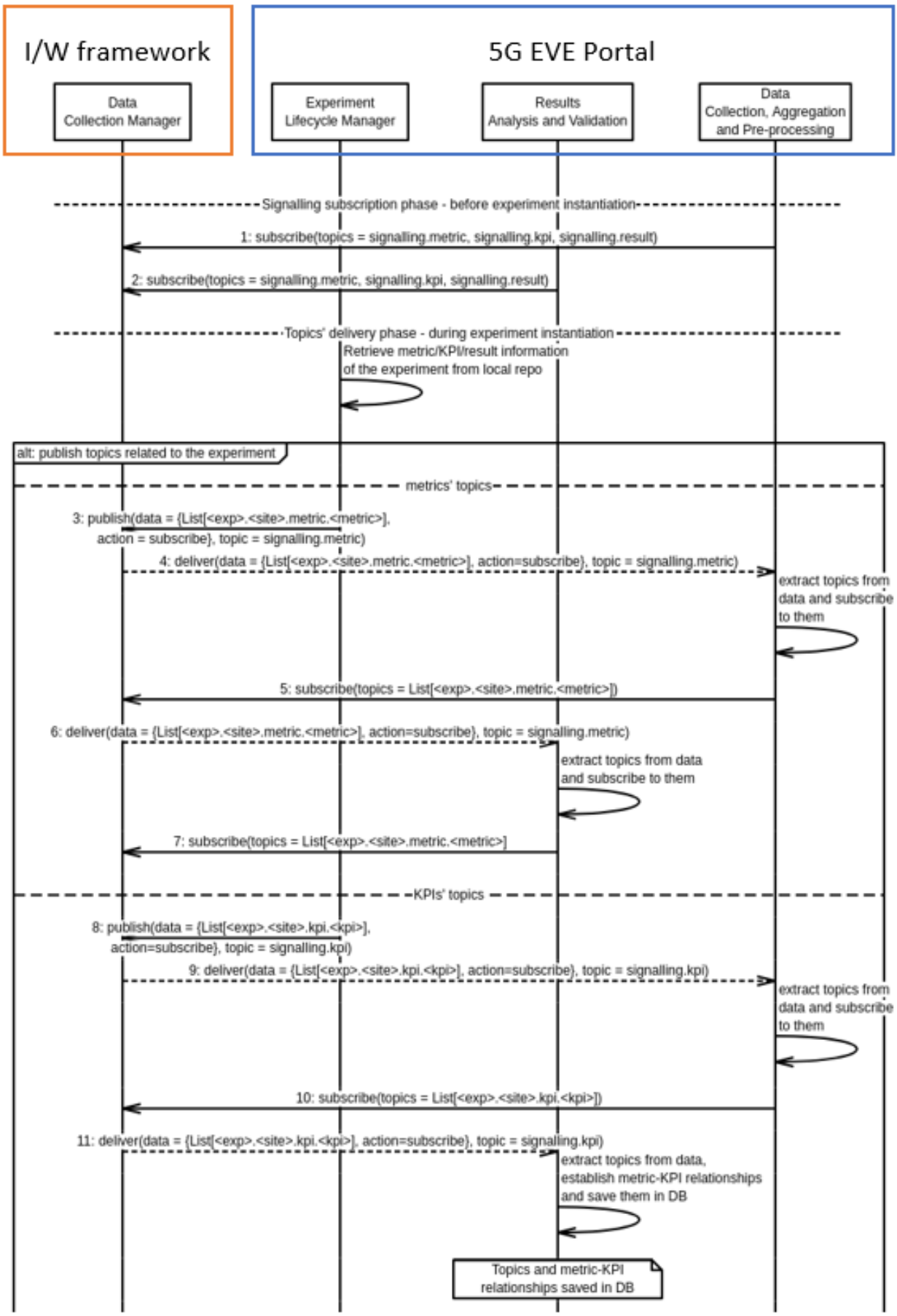
In D3.2 – section 4.3.3.6, it was defined a brief workflow presenting the main monitoring operations that involve the Data Collection Manager (or Publish-Subscribe queue, as mentioned in that section): subscribe (to a topic), publish (data in a topic) and deliver (data to the subscribed components to a specific topic in which data has been published). Moreover, it was also presented a first distinction between topics, differentiating between signalling topics and topics related to the monitored data. In fact, the variety of topics that can be used in the Data Collection Manager operation has been increased with the introduction of the Topic framework proposal already described.

That preliminary workflow has been updated in D4.1 – section 3.2.3.3, regarding the experiment monitoring and performance analysis stage, which is the final phase for a given experiment. In this deliverable, it will be only mentioned the key aspects that should be considered regarding the Data Collection Manager functionality and that have not been covered at all in D4.1.

Having said that, in that experiment monitoring and performance analysis stage, three different phases[15] were detected:

- **Subscription phase**: carried out before and during the experiment instantiation. In this case, the components that perform functions related to monitoring and performance analysis are subscribed to the corresponding topics related to the experiment, depending on the specific experiment that is going to be executed and the metrics/KPIs/results to be monitored. These topics are provided by the Experiment Lifecycle Manager and distributed by the Data Collection Manager through specific signalling topics. This workflow is depicted in Figure 17.

[15] In the phases, it is presented the following components: Data Collection, Aggregation and Pre-processing, Data Indexing and Storage, and Data Visualization. These three components belong to the Experiment monitoring and Maintenance & Results Collection component presented in the Data Collection Manager architecture.


I/W framework
Data Collection Manager
5G EVE Portal
Experiment Lifecycle Manager
Results Analysis and Validation
Data Collection, Aggregation and Pre-processing
Signalling subscription phase - before experiment instantiation
1: subscribe(topics = signalling.metric, signalling.kpi, signalling.result)
2: subscribe(topics = signalling.metric, signalling.kpi, signalling.result)
Topics' delivery phase - during experiment instantiation
Retrieve metric/KPI/result information of the experiment from local repo
alt: publish topics related to the experiment
metrics' topics
3: publish(data = {List[<exp>.<site>.metric.<metric>], action = subscribe}, topic = signalling.metric)
4: deliver(data = {List[<exp>.<site>.metric.<metric>], action=subscribe}, topic = signalling.metric)
extract topics from data and subscribe to them
5: subscribe(topics = List[<exp>.<site>.metric.<metric>])
6: deliver(data = {List[<exp>.<site>.metric.<metric>], action=subscribe}, topic = signalling.metric)
extract topics from data and subscribe to them
7: subscribe(topics = List[<exp>.<site>.metric.<metric>]
KPIs' topics
8: publish(data = {List[<exp>.<site>.kpi.<kpi>], action=subscribe}, topic = signalling.kpi)
9: deliver(data = {List[<exp>.<site>.kpi.<kpi>], action=subscribe}, topic = signalling.kpi)
extract topics from data and subscribe to them
10: subscribe(topics = List[<exp>.<site>.kpi.<kpi>])
11: deliver(data = {List[<exp>.<site>.kpi.<kpi>], action=subscribe}, topic = signalling.kpi)
extract topics from data, establish metric-KPI relationships and save them in DB
Topics and metric-KPI relationships saved in DB

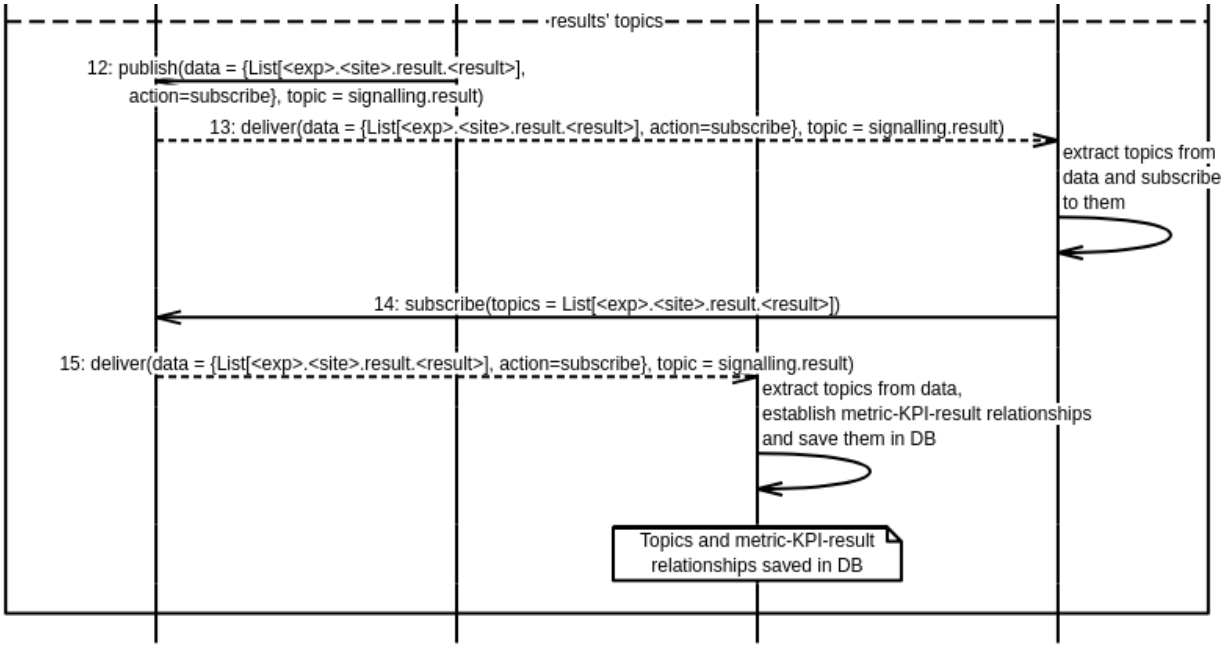


Figure 17: Subscription to the topics used for experiment monitoring and performance analysis purposes.

- **Monitoring and data collection phase:** the metrics that have been published (each *N* seconds until the end of the experiment) by the distributed Data shippers (with a previous log-to-metric transformation done by these components) in the Data Collection Manager are delivered, during the experiment execution, to the two components that are subscribed to the topics which correspond to each metric (which are the Results Analysis and Validation component and the Data Collection, Aggregation and Pre-processing tool that belongs to the Experiment monitoring and Maintenance & Results Collection component). In the case of the Results Analysis and Validation component, there is another loop which also involves the Data Collection Manager, because the metrics and KPIs calculated from these metrics are published in other specific topics (according to the topic framework defined) and deliver afterwards to the Data Collection, Aggregation and Pre-processing tool. Remember that Data shippers must know in advance the topics in which they have to subscribe, information that can be provided by the experiment developer in the corresponding configuration files (and always following the topic framework proposed) or can be also included as Day-2 configuration by the Runtime Configuration component (as long as that configuration is defined in the test case). This workflow is presented in Figure 18.

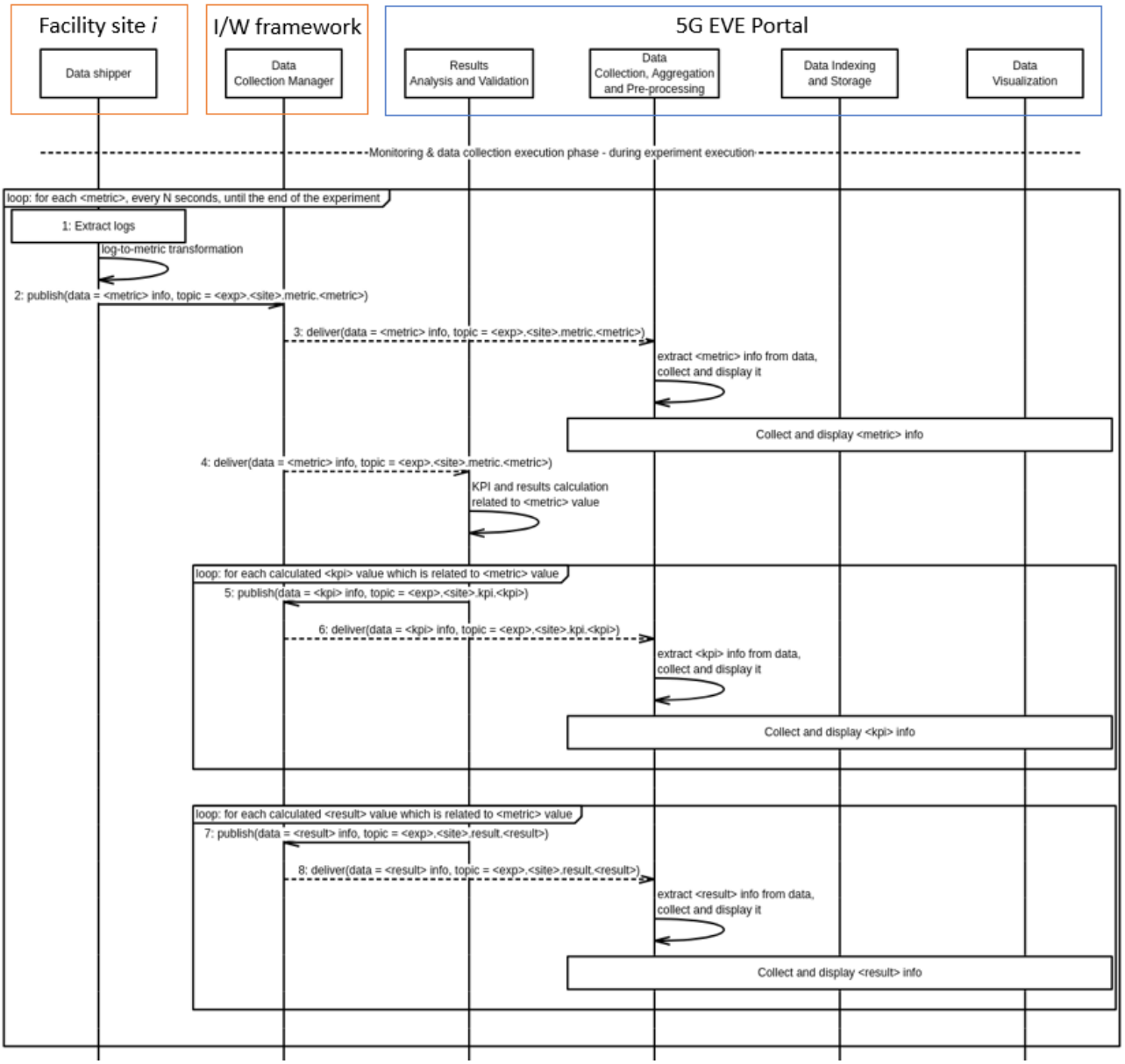
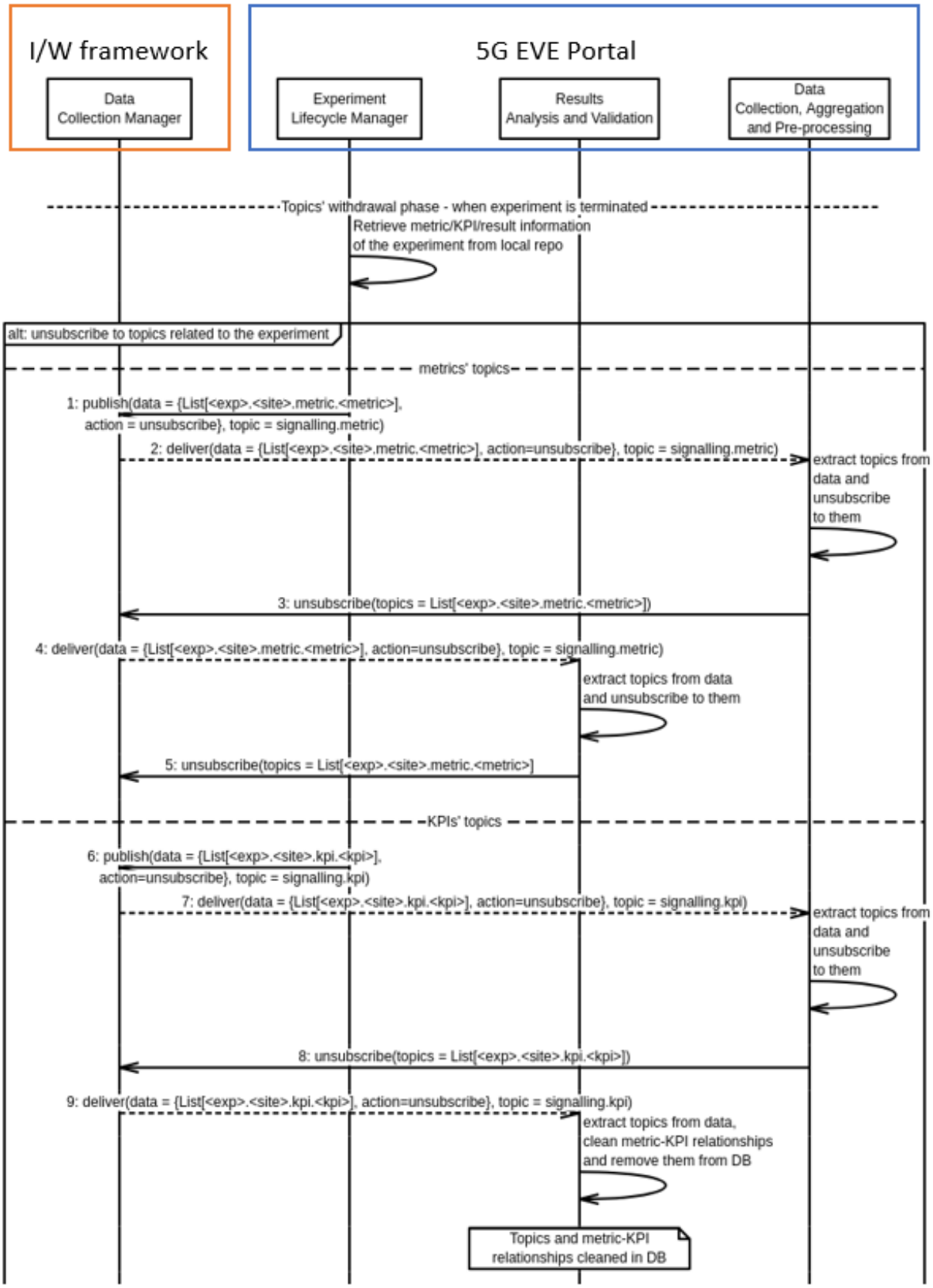


Figure 18: Delivery and management of monitoring information during the experiment execution.

- **Withdrawal phase:** performed as the opposite operation to the subscription phase, in order to trigger the withdrawal to the topics used during the experiment, using again the signalling topics for that purpose. Withdrawal to signalling topics is not done because they can be used during the execution of other test cases, but the procedure of withdrawal would be similar. Figure 19 presents this workflow in detail.


I/W framework
Data Collection Manager
5G EVE Portal
Experiment Lifecycle Manager
Results Analysis and Validation
Data Collection, Aggregation and Pre-processing
Topics' withdrawal phase - when experiment is terminated
Retrieve metric/KPI/result information of the experiment from local repo
alt: unsubscribe to topics related to the experiment
metrics' topics
1: publish(data = {List[<exp>.<site>.metric.<metric>], action = unsubscribe}, topic = signalling.metric)
2: deliver(data = {List[<exp>.<site>.metric.<metric>], action=unsubscribe}, topic = signalling.metric)
extract topics from data and unsubscribe to them
3: unsubscribe(topics = List[<exp>.<site>.metric.<metric>])
4: deliver(data = {List[<exp>.<site>.metric.<metric>], action=unsubscribe}, topic = signalling.metric)
extract topics from data and unsubscribe to them
5: unsubscribe(topics = List[<exp>.<site>.metric.<metric>]
KPIs' topics
6: publish(data = {List[<exp>.<site>.kpi.<kpi>], action=unsubscribe}, topic = signalling.kpi)
7: deliver(data = {List[<exp>.<site>.kpi.<kpi>], action=unsubscribe}, topic = signalling.kpi)
extract topics from data and unsubscribe to them
8: unsubscribe(topics = List[<exp>.<site>.kpi.<kpi>])
9: deliver(data = {List[<exp>.<site>.kpi.<kpi>], action=unsubscribe}, topic = signalling.kpi)
extract topics from data, clean metric-KPI relationships and remove them from DB
Topics and metric-KPI relationships cleaned in DB

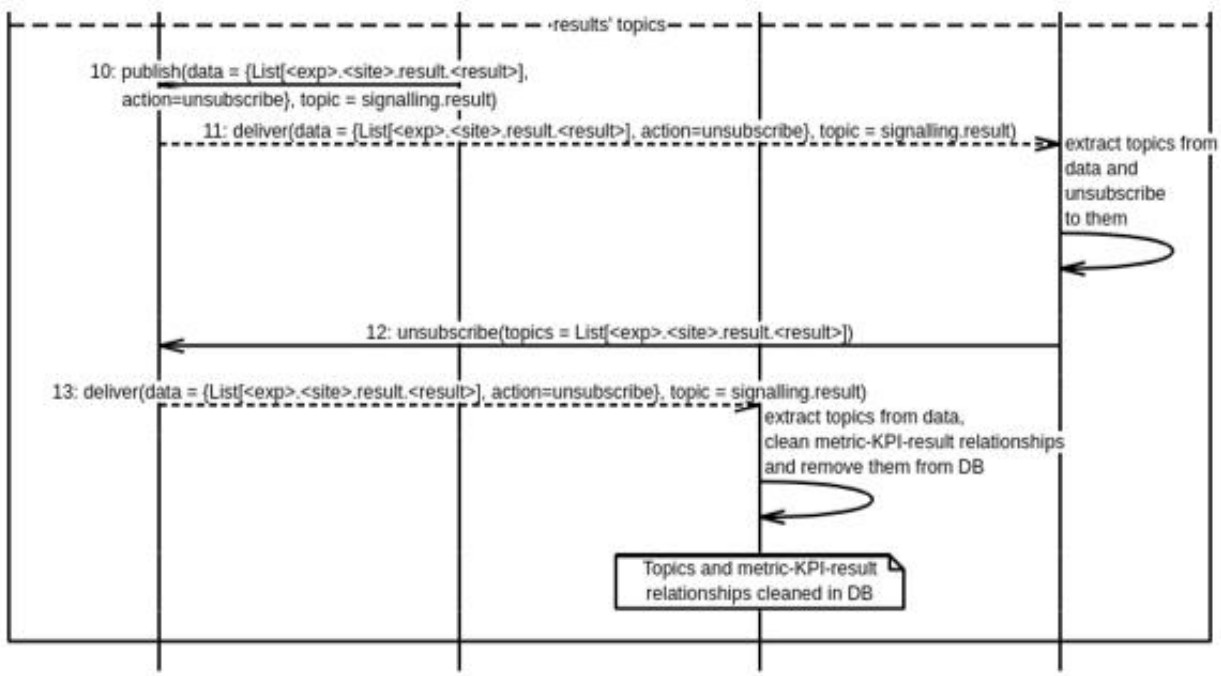


Figure 19: Withdrawal of the topics used for experiment monitoring and performance analysis purposes

Annex B – Runtime Configurator update from D3.2

Firstly, it will be presented the updated architecture for this component, with all the new requirements and functionalities that must be considered for its correct definition. Afterwards, it will be updated the operations and information model included in D3.2 – sections 4.4.3.2 (for NBI) and 4.5.4 (for SBI), and the specific-purpose workflows presented in D3.2 – section 4.3.3.7, being the last one aligned with the experiment execution phase that is currently described in D4.1 – section 3.2.3.

B.1 Updated architecture

In Figure 20 the Experiment Configuration Framework is depicted, in which the Runtime Configurator is involved It should be noted that it has suffered some small changes from its previous version that will be commented in sub-sequent sections.

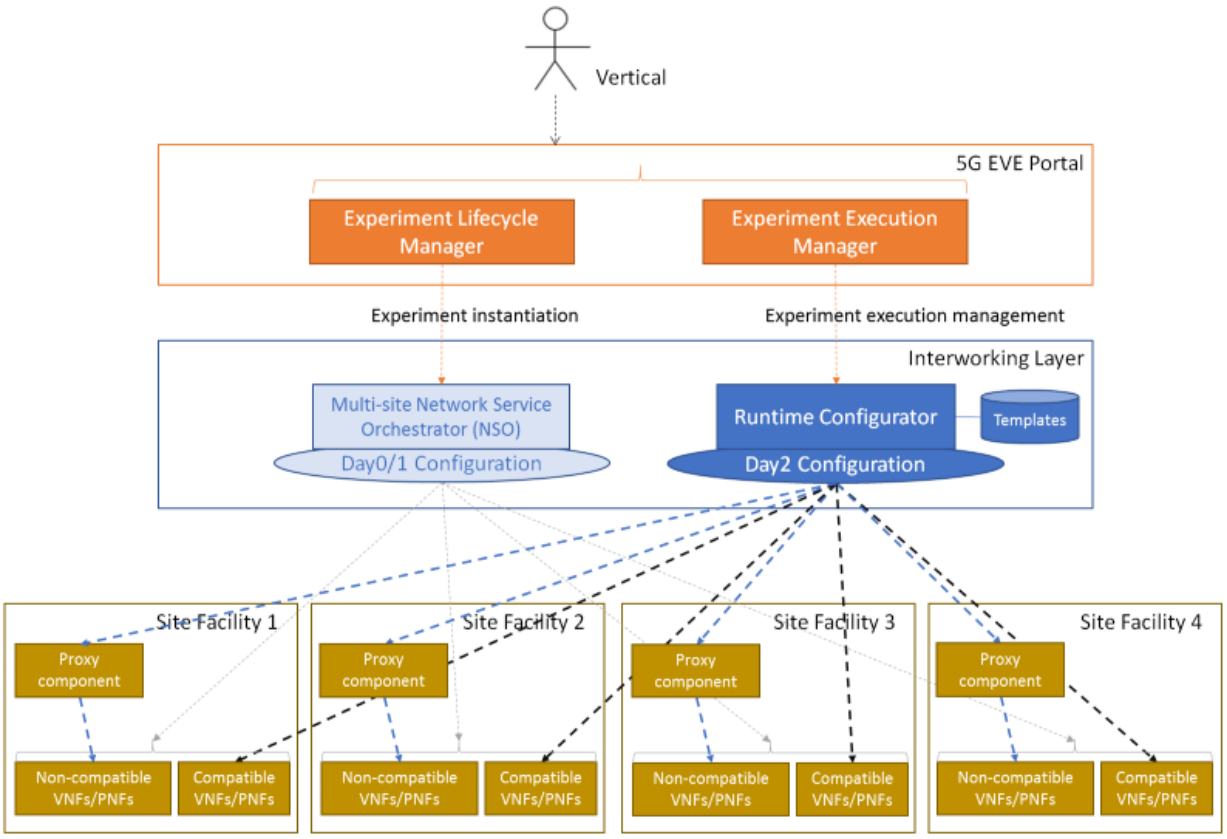


Figure 20: Improved version of the Experiment Configuration Framework.

The main components are the same than in its first version, all of them being involved in the experiment execution phase, in which it can be distinguished two main workflows:

- The **experiment instantiation workflow**, which involves the Experiment Lifecycle Manager and the Multi-site Network Service Orchestrator, is responsible for instantiating the network service, with its corresponding VNFs and PNFs, providing the Day-0/1 configuration as depicted in the previous diagram with the grey arrows. The Runtime Configurator does not participate in this configuration.
- The **experiment execution management workflow**, executed after the experiment instantiation workflow, involves the Experiment Execution Manager and the Runtime Configurator. In this case, the Experiment Execution Manager will define the test cases with the necessary steps to be followed in order to execute and monitor the test (defined in the Test Case Blueprints, presented in D4.1 – section

5.2.7). These high-level instructions include specific calls to templates located in the Runtime Configurator, which contain the related commands and specific configurations that must be provided to the VNFs and PNFs involved in the experiment. As a result, the execution of that Day-2 configuration is delegated on the Runtime Configurator, and the Experiment Execution Manager only needs to indicate the template(s) that must be called during the experiment execution. These templates must be saved in a catalogue managed by the Runtime Configurator, and the experiment developer must be the responsible for establishing the necessary correspondences between the Test Case Blueprints and the template(s) that must be used for the use case.

After triggering the templates to be executed by the Runtime Configurator, it is established a connection between the Runtime Configurator and the VNFs/PNFs to be configured (black arrows in the previous diagram), and then the Day-2 configuration is applied, returning the results to the Experiment Execution Manager for allowing the monitoring of the experiment execution procedure. However, there could be special components that are not reachable directly by the Runtime Configurator (e.g. proprietary, non-standard components). In that case, the access could be achieved by using a Proxy Component reachable by the Runtime Configurator, implementing in that Proxy Component the necessary adaptations in order to provide the Day-2 configuration to these non-compatible VNFs/PNFs (blue arrows in the previous diagram).

B.2 Updated operations and information model

In D3.2, sections 4.4.3.2 (for NBI) and 4.5.4 (for SBI), it was included the first version of the operations and information model for the Runtime Configurator, which have been updated as a result of the changes already commented. Both updated NBI and SBI for the Runtime Configurator are presented in Table 9 and Table 10.

Table 9: Runtime Configurator NBI.

Runtime Configurator NBI		
Execution of templates for a given step of a test case execution		
Description	Invocation of the templates related to a given step of a test case execution, in order to provide Day-2 configuration to the VNFs/PNFs referenced in the test case step.	
Reference Standards	OpenSSH protocol or REST API interface in case of incompatibilities.	
Operations Exposed	***Information Exchanged***	***Information Model***
Execute a template for providing pre-configuration operations (execute_config_templates operation)	• A list with the templates ID. • The experiment execution ID. It will return the result of the execution of the commands included in the used templates.	In case of using OpenSSH, the input information will be provided in a specific script to be executed in the Runtime Configurator server, which will include the necessary logic to process it, obtain the IP addressing configuration of the components to be configured and apply the templates to these components. This behaviour is still under development.

Execute a template for providing test-related operations (execute_exec_templates operation)	• A list with the templates ID. • The experiment execution ID. It will return the result of the execution of the commands included in the used templates.	In case of using OpenSSH, the input information will be provided in a specific script to be executed in the Runtime Configurator server, which will include the necessary logic to process it, obtain the IP addressing configuration of the components to be configured and apply the templates to these components. This behaviour is still under development.

Table 10: Data Collection Manager SBI.

Runtime Configurator SBI		
Provision of the configuration defined in the templates for a given step of a test case execution		
Description	Execution of the actions described in the templates for enabling the Day-2 configuration in the VNFs/PNFs which are intended to be configured.	
Reference Standards	Openconfig/IETF Yang models, Netconf, Ansible (OpenSSH protocol, mainly)	
Operations Exposed	***Information Exchanged***	***Information Model***
Apply pre-configuration (apply_configuration operation)	• A list with the commands to be executed in the VNFs/PNFs. It will return the result of the commands execution, which is then forwarded to the Experiment Execution Manager.	YANG model or configuration template depending on the required configuration. This behaviour is still under development.
Execute the commands related to a test step (execute_commands operation)	• A list with the commands to be executed in the VNFs/PNFs. It will return the result of the commands execution, which is then forwarded to the Experiment Execution Manager.	YANG model or configuration template depending on the required configuration. This behaviour is still under development.

B.3 Updated specific-purpose workflows

The Runtime Configurator workflow presented in D3.2 – section 4.3.3.7 only presented a small interaction between the Runtime Configurator and the Experiment Execution Manager, without specifying the data exchanged between them. The following workflows extend the information in D3.2.

First of all, when the experiment execution is started and in "configuring" status, there is a first phase[16] in which the VNFs and PNFs related to the experiment could need some extra configuration in order to work properly during the tests; for example, the topics in which they have to publish the metrics data to be collected by the Data Collection Manager (in case this configuration has not been included beforehand). In Figure 21, this behaviour is presented. For achieving that goal, the Runtime Configurator can handle some specific templates that can carry out these configuring actions, being called by the *execute_config_templates* operation (message 1). The Runtime Configurator, after receiving this request, will obtain all the IP addressing information related to the components by using the Exec_ID (i.e. the ID which identifies the experiment), in order to be able to

[16] This extends the "Configure VNFs" block defined in the workflow presented in D4.1 – section 3.2.3.2, related to the Experiment execution management.

reach the components to be configured. Then, and depending on the type of component (if it is a non-compatible VNF/PNF, it is necessary to use the Proxy Component as broker), the Runtime Configurator will apply the commands contained in the selected templates in the VNFs/PNFs with the *apply_configuration* operation (message 2 in compatible components, messages 4 and 5 in non-compatible components), which will return the result of the configuration operation (message 3 in compatible components, messages 6 and 7 in non-compatible components), which is forwarded to the Experiment Execution Manager (message 8) in order to monitor the test execution. This workflow is executed for each step related to this pre-configuration feature in a sequential way.

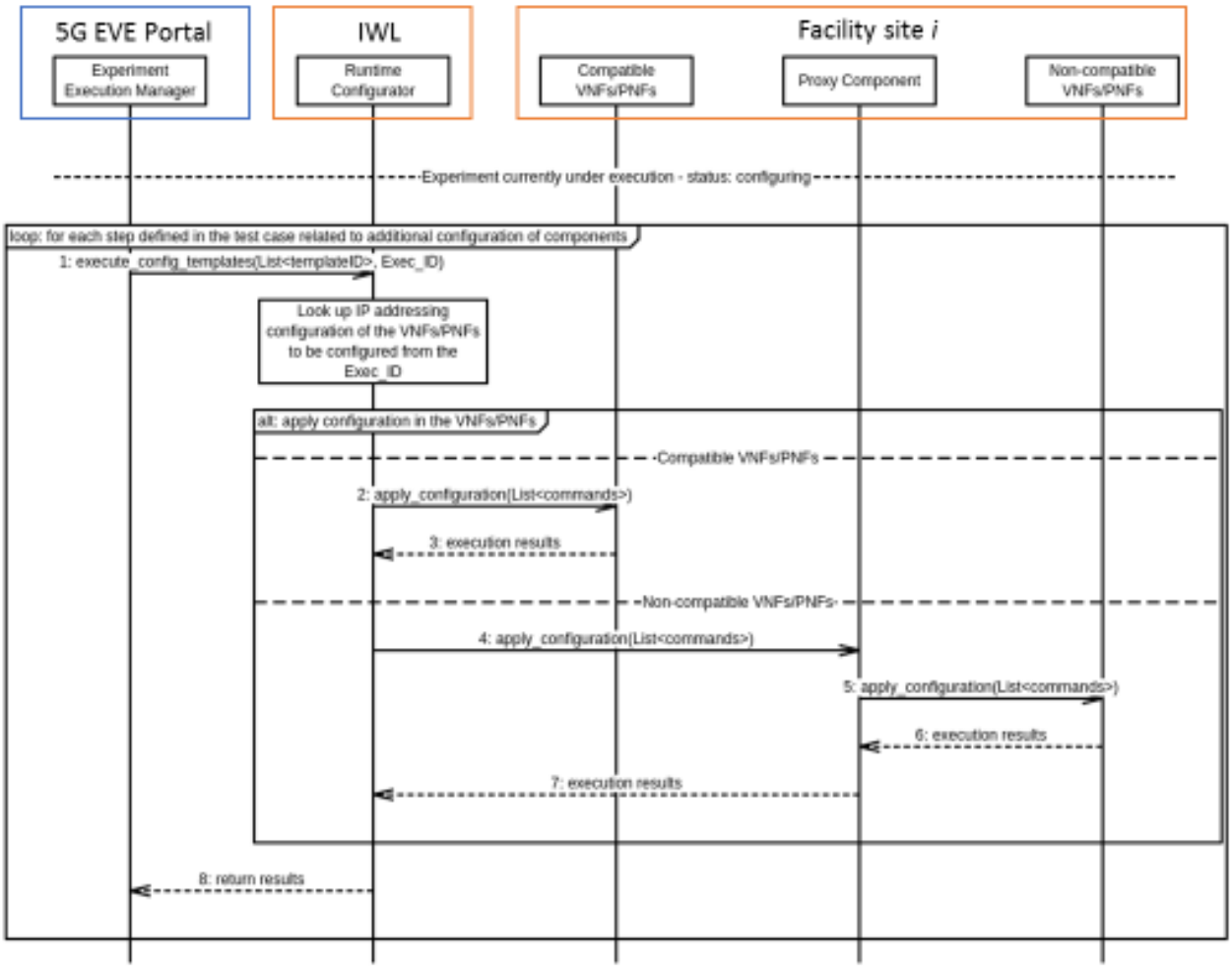


Figure 21: Application of Day-2 configuration – experiment in configuring state.

When this phase finishes, it starts a second phase[17], depicted in Figure 22, related to the test execution itself; i.e., the Experiment Execution Manager will update the experiment status to "running" and will start with the execution of test steps. As a result, the templates which correspond to these steps are called by the Experiment Execution Manager through the *execute_exec_templates* operation (message 1), whose format is similar to the *execute_config_templates*. From this point, the workflow is completely the same as in the first phase but changing the *apply_configuration* operation for the *execute_commands* operation.

[17] This extends the "Execute test" block defined in the workflow presented in D4.1 – section 3.2.3.2, related to the Experiment execution management.

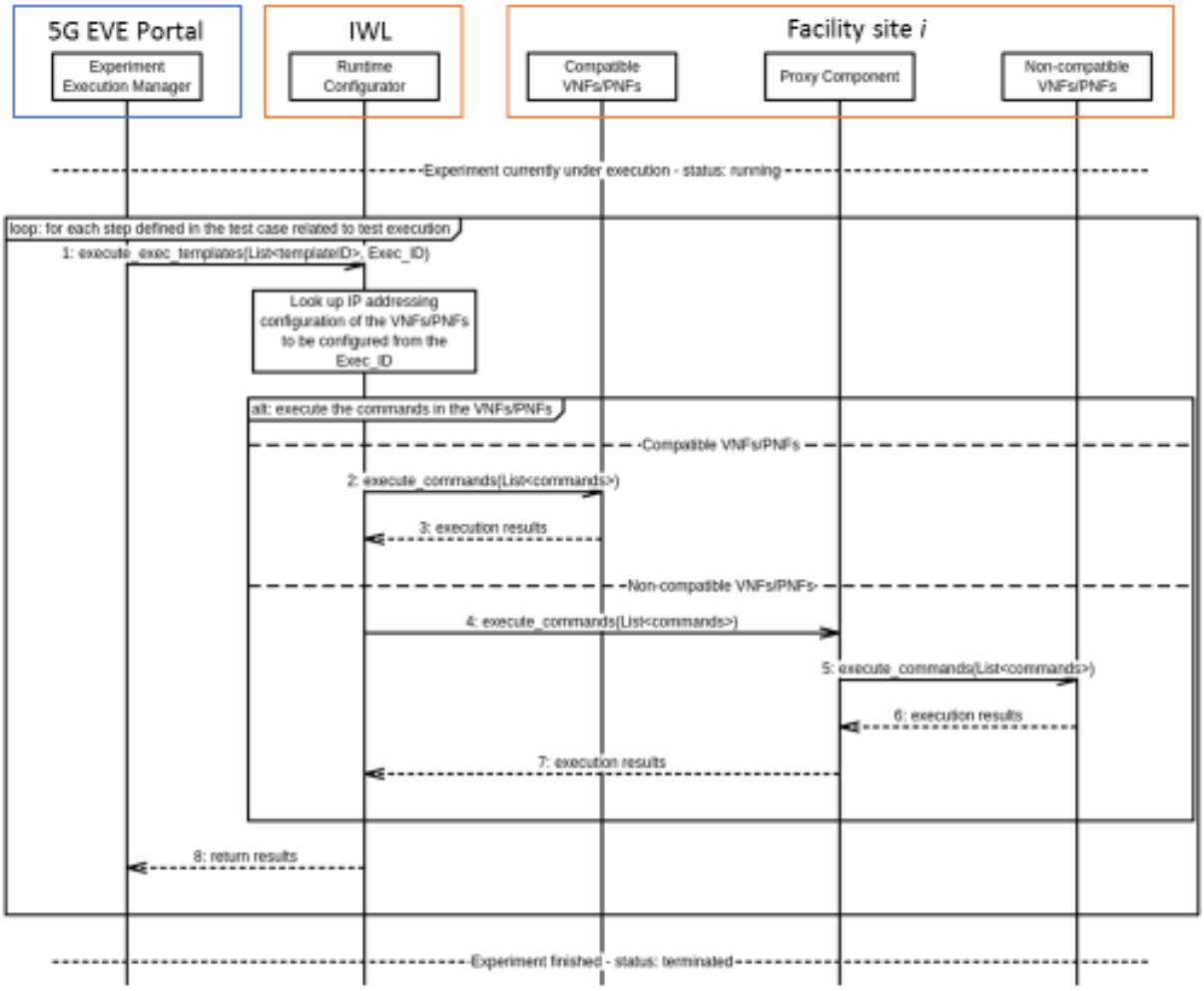


Figure 22: Application of Day-2 configuration – experiment in running state.

This distinction has been done in order to differentiate between the pre-configuration operations related to a test and the specific steps to be executed during the test execution, as the second case may have particular temporary constraints that are not present in the first case, which is always applied at the beginning. Note that the first phase may be optional in case of not needing to include additional configuration for the test, and its definition will depend on each Test Case Blueprint.