5G European Validation platform for Extensive trials

Deliverable D3.4
Second 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_D3.4
Deliverable Title:	Second implementation of the interworking reference model
Editor(s):	Marc Mollà Roselló
Author(s):	Ramón Perez, Aitor Zabala (TELC); M. Pergolesi, M. Femminella, F. Lombardo Francesco, P. Lungaroni, S. Salsano, N. Blefari Melazzi (CNIT); Giacomo Bernini, Leonardo Agueci, Giada Landi (NXW); Gopolasingham Aravinthan (NOK-FR); Marc Mollà Roselló (ERI-ES); Jaime Garcia-Reinoso, Pablo Serrano Yañez-Mingot (UC3M); Grzegorz Panek (ORA-PL); Luis Miguel Contreras, Lourdes Luque, Juan Rodriguez (TID); Federico Alvarez, David Jimenez, Javier Serrano, Alberto del Rio (UMP); P. Vlacheas, V. Foteinos, I. Belikaidis, V. Kosmatos, K. Trichias, A. Skalidi, C. Ntogkas, A. Gkiolias, I. Chondroulis, A. Demou, A. Athanesselis, M. Mitrou (WINGS)
Reviewer(s):	Silvia Canale (Ares2T); Vincenzo Suraci (Ares2T)
Contractual Date of Delivery:	30/06/2020
Submission Date:	29/06/2020
Dissemination Level:	PU
Status:	Final
Version:	1.0
File Name:	5G_EVE_D3.4

Deliverable History

Version	Date	Modification	Modified by
V0.1	*14/05/2020*	*First draft*	*Marc Mollà Roselló*
V0.2	*29/05/2020*	*Second draft with initial contributions*	*WP3 team*
V0.3	*08/06/2020*	*First version for internal review*	*WP3 team*
V0.4	*09/06/2020*	*Second version for internal review*	*WP3 team*
V0.5	*12/06/2020*	*Version ready for internal review*	*WP3 team*
V0.6	*14/06/2020*	*Internal review*	*WP3 team*
V0.9	*15/06/2020*	*Ready for official review*	*WP3 team*
V0.91	*26/06/2020*	*Internal review*	*WP3 team*
V1.0	*28/06/2020*	*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
DCI	Data Center Interconnection
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IWF	Inter-Working Framework
IWL	Inter-Working Layer
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 Modelling Language
VNF	Virtual Network Function
VNFD	VNF Descriptor
YAML	YAML Ain't Mark-up Language

List of Figures

List of Tables

Executive Summary

Deliverable D3.4 is the second deliverable dealing with the implementation of the Interworking Layer (IWL).

This deliverable includes all IWL components as they are tested and integrated internally and with other 5G EVE components (5G EVE Portal, 5G EVE sites), as we describe in the deliverable D3.7.

One of the biggest efforts done in the scope of WP3 during the last third of the second year of the project is the integration of the IWL components. We integrated and tested the components first internally, and later with the other 5G EVE components of the 5G EVE Portal and the 5G EVE sites. Most of this integration has been done during the COVID-19 situation, which impacted the normal execution of the project due to the restrictions on working physically in the sites. However, we delivered in time the IWL components with the required scope and now IWL is integrated with the 5G EVE Portal and we tested the integration with OSM and ONAP NFV-Os.

As we explain in the Roadmap section, we managed the scope of the IWL roadmap for delaying those part that are not essential today for the project, without impacting the services and features offered by 5G EVE. In addition, we also managed the incorporation of new requirements: (i) a new Gaming use case coming from Telefonica and incorporated in the last amendment of the project, (ii) new requirements for integrating a new kind of orchestrators that are focused in RAN management and (iii) the development of a new component, called IWL repository, which exposes information related to the architecture of the 5G EVE sites. The seamless incorporation of these new requirements demonstrates that the open and modular IWL architecture allows to change, evolve or enhance any part of the IWL for adapting it to future requirements.

We also update in this report the information related to inter-site connectivity, which is core for supporting the multi-site experiments execution.

1 Introduction

Deliverable D3.4 "*Second implementation of the interworking reference model*" contains the report of the Inter-Working Layer software deliverables for MS8 and MS9 of the 5G EVE project, 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]). The software delivery is an update of the first delivery reported at D3.3 ([3]). 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, D3.2 and D3.3. It also contains the description of the new requirements and components included in the IWL since the delivery of D3.3.
- Section 3 contains the description of the software included in this deliverable. We included the public specification of the API provided for each component, with a reference of the public open API definition. Also, we included the features provided by each component as well as the internal design.
- 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 5G EVE I/W layer (IWL) enables the 5G EVE platform to execute multi-site vertical experiments without caring of the specific technology constraints that each site implements to provide 5G services. Indeed, the IWL is the core 5G EVE component for the abstraction of specific logics and management, control and orchestration tools that each site facility implements, by exposing common and unified procedures (and data models) for managing the vertical experiments on top of the heterogeneous multi-site infrastructure. The 5G EVE Portal benefits from this approach by accessing the IWL to first retrieve the capabilities of the multi-site 5G EVE infrastructure in terms of services and VNFs, and then to request for the provisioning of multi-site 5G vertical experiments designed on top of such capabilities, all by using a common and unified set of APIs based on the ETSI NFV SOL specifications that the I/W framework expose. At its southbound, the I/W framework takes care to interface with the various site tools (e.g. the local site orchestrators, each with its own information model for network service and VNF description). This is enabled by a dedicated Adaptation Layer that abstracts the specific site technologies, data models and procedures while exposing common and unified APIs to the I/W framework internals.

The detailed design of the I/W framework capabilities and features is reported in deliverables D3.1 ([1]) and D3.2 ([2]), where the functional decomposition together with APIs and workflows are described. Moreover, deliverable D3.3 ([3]) provides details about the first software release of the I/W framework, which is already integrated with the 5G EVE site facilities and used by the 5G EVE Portal for the validation of the first set of 5G vertical experiments. With respect to the software delivery in D3.3, the internal architecture of the I/W framework of the new release described in this document (for the milestone MS9) is slightly updated as depicted in **Figure 1**, where in addition to the existing functional blocks (the Multi-Site Network Orchestrator, the Multi-Site Catalogue, the Multi-Site Inventory, the Runtime Configurator and the Data Collection Manager), a new component is introduced to address additional I/W framework requirements and constraints emerged during the first set of integration and validation activities. This new component is the IWF repository, depicted in green in **Figure 1**, takes care to store information about site facility capabilities and features, e.g. in terms of available local orchestrators (type, credentials,), virtualized infrastructures and availability zones. The IWF repository is therefore a central storage point within the I/W framework for all the site facility information that are useful for the I/W framework logics, even if not strictly related to the 5G vertical experiments themselves. It exposes a well-defined set of APIs to be consumed by the other I/W framework components (mostly the Multi-Site Catalogue, the Multi-Site Network Service Orchestrator, the Runtime Configurator), as fully described in section 3.7.

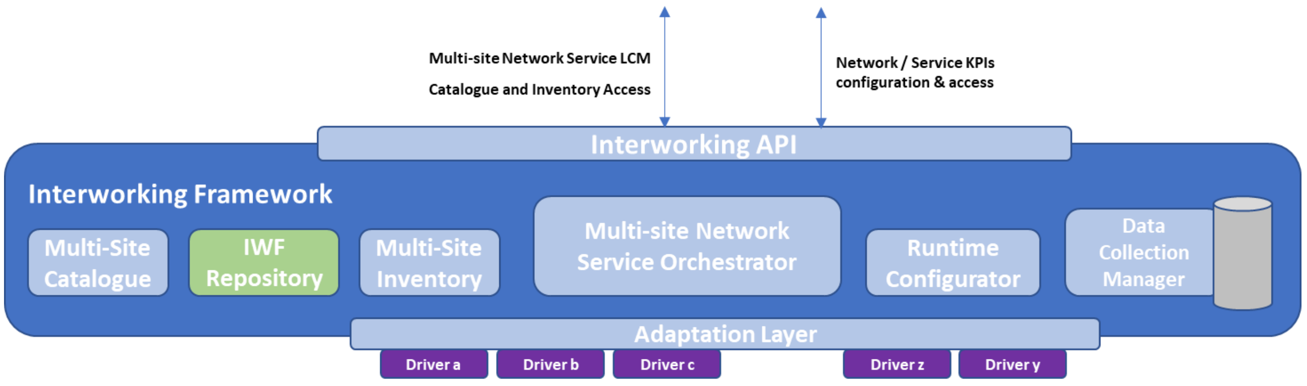

Figure 1: I/W Framework updated architecture

2.1 New features in MS9 version

During the design and development of the IWL we introduce new requirements and components due to the following reasons:

1. new requirements coming from the new Telefónica's gaming use case

2. new requirements from the integration of IWL and the 5G EVE sites and
3. new requirements from internal 5G EVE use cases and the new use cases coming from ICT-19 projects.

In the amendment of the 5G EVE project, Telefónica (TID) introduced a new gaming business case that is interesting for the IWL because the use case introduces the integration of SDN controllers and the IWL as well as a close loop that connects the metrics collection with the orchestration decision. The use case is briefly described in the following sections and the IWL will support it for the next release of the IWL.

One of the biggest efforts in WP3 during this year has been the integration of the IWL components and the integration of the 5G EVE Portal, the IWL and the 5G EVE sites. During the integration we realized that even using standard and open interfaces we needed to know per-site details in order to integrate the systems: credentials, internal architecture, configuration, etc. For that reason, we decided to create the IWL repository, a new IWL component that will store all the per-site information required at the IWL. This approach allows us to maintain most of the IWL components agnostic to the 5G EVE site architecture and configuration.

Finally, we introduce the RAN orchestration feature as the first feature in our roadmap towards the multi-site experiments. The aim of the feature is to configure the (v)RAN of the experiment according to the requirements of the experiment developer. This feature allows us to introduce RAN controllers that configure the RAN according to the requirements or even RAN orchestrators that deploy virtual RAN on demand, all of this transparently to the user of the 5G platform.

In the following sections we detail all these new features.

2.1.1 Extended Automation in Connectivity Interworking Services

In Deliverable D3.2 ([2]) – *Interworking Reference Model* – several "Interworking Services" were introduced. These were classified following the below taxonomy:

- Interworking Framework services, divided in single-site and multi-site scenarios, and which deal with orchestration, monitoring, and automated deployment.
- Data Plane Connectivity services, divided themselves in traditional services, like VLANs, VPNs, etc., and slices.

Regarding the automated provisioning of the latter, some discussions were introduced about different SDN architectures with the capabilities to achieve this. In 5G EVE, generally, SDN Controllers in charge of managing the datacentre fabric (or the Open vSwitches at the compute infrastructure) are themselves managed by the local VIMs, or alternatively, by the local NFVO (as depicted in Figure 2).

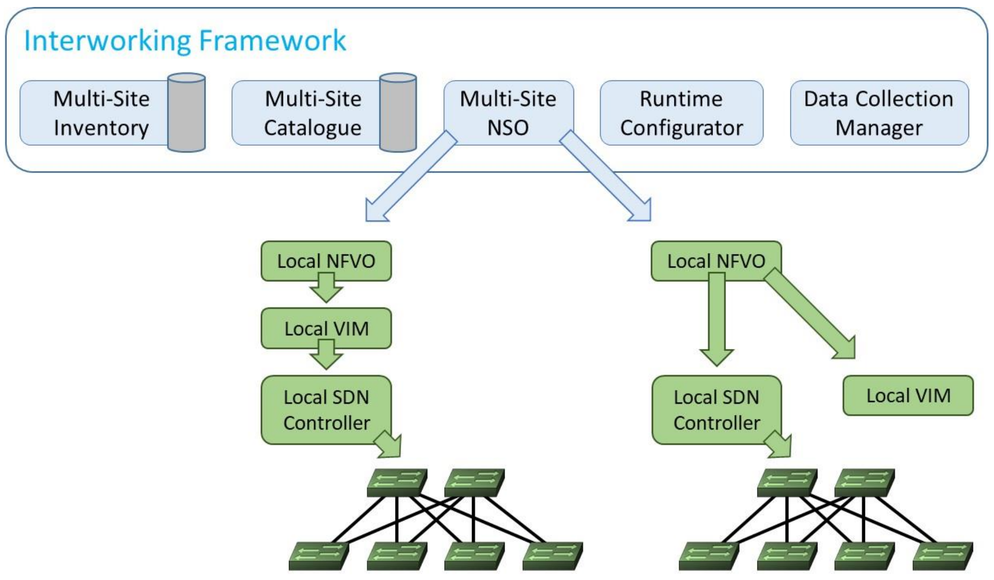

Figure 2: Typical SDN Control scenarios in 5G EVE

These configurations are not only typical in 5G EVE sites, but also in most of the Cloud Service Providers' datacentres. However, this is not necessarily true for telecom operators. For many reasons, one of these being the interaction with the transport network in Data Centre Interconnection (DCI) scenarios, it may happen that SDN Controllers at the operators' datacentres are not governed by local VIMs or NFVOs, but by other hierarchical SDN components (Figure 3).

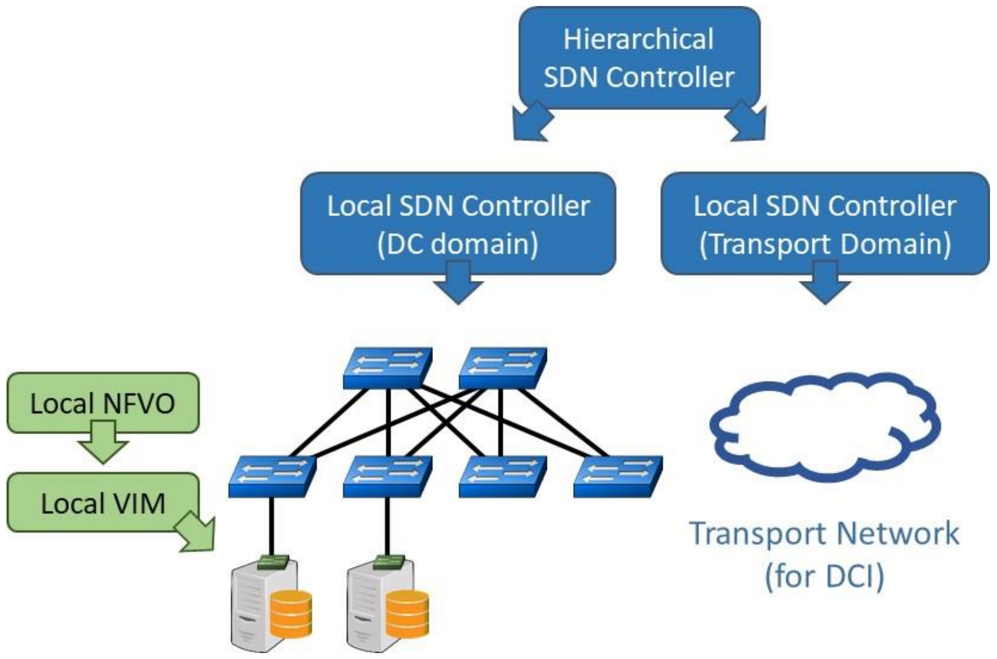
Hierarchical SDN Controller
Local SDN Controller (DC domain)
Local SDN Controller (Transport Domain)
Local NFVO
Local VIM
Transport Network (for DCI)

Figure 3: Potential SDN Control scenario in telecom operators

In this sense, the project has taken advantage of the inclusion of a new Use Case ("high quality multi-site gaming experience", described in D2.6 ([7]) – *Participating vertical industries planning*) to face this challenge, and demonstrate scenarios in which the SDN Controllers (either the local or even the hierarchical one) may be directly exposed to the Multi-Site Network Service Orchestrator (MS-NSO).

In the gaming Use Case, estimations of the Quality of Experience (QoE) are generated, which can trigger corrective actions when needed. Initially, the actions would target the gaming engine and media. For example, in case too many users joined a game, and that affected the overall quality (example: packet losses generating artefacts), the QoE Estimation module could address the transcoders, and reduce the video rates. The reduction in bandwidth would reduce the packet losses, resulting in zero artefacts and better QoE, even if the transmitted video quality is lower.

The capability of the QoE Estimation module to impact on the network as well as on the gaming engine would certainly improve the gaming system. The module would have more flexibility to decide which component should implement the corrective actions, resulting in better QoE. Indeed, imagine the QoE Estimation module in the example case above could somehow detect that the packet losses are all happening in a single network bottleneck, and could request to the Orchestrator additional resources at that network segment. Under these circumstances the system would not need to reduce the transmitted video rate, thus resulting in improved experienced quality.

2.1.1.1 New requirements to MS-NSO

When considering that the Network Optimiser module will be able to trigger network related actions at the MS-NSO, and that the latter should progress these directly to an SDN Controller, two new requirements compared to D3.2 appear:

- The NBI interface of the MS-NSO now must be opened to support requests from components outside the Interworking Layer (IWL) or the 5G EVE Portal. Furthermore, these components are in fact external to 5G EVE development, coming as they are from a Vertical. Apart from the update of the NBI itself, this results in very important operational concerns, including of course security.
- The SBI interface of the MS-NSO now must implement a new type of communication, towards an SDN Controller.

2.1.1.2 Implementation approach

The approach for the support of the new requirements in 5G EVE will be to try to reuse as much as possible what is already available. Two reference points (Or-Or and Os-Ma-Nfvo) were designed for the MS-NSO in D3.2, being currently supported by the implementation that 5G EVE has made of ETSI NFV SOL 005 ([5]).

For the NBI, the better positioned option which is being evaluated is to use the POST request available to update a Network Service. This request includes a field called “vnfConfigurableProperties”, which is a list of key-value pairs, where we could include the proposed operation. For example, we could code “newBandwidth = 5M”, or more open actions like “bandwidthOperation = increase”.

The big benefit of this approach is that it would be quick to implement and would not increase the complexity of the MS-NSO. Probably the biggest drawback is that the original intention of the operation is not exactly this one, resulting in a solution a little bit against the spirit of the specification. The alternative will be to implement a more SDN-like operation in the MS-NSO NBI.

Regarding the SBI, there are also two options. One of them is implementing at the MS-NSO a WIM-like interface, as defined for OSM. The biggest issue about the current specification of this interface is that it is more focused on the provisioning of VPN-like services, which limits quite a lot its potential usage.

The second option, probably more realistic, is to make a proxy out of the MS-NSO for this specific operation and let the Adaptation Layer build a translator between NFV SOL 005 and an API which SDN Controllers can understand. That later API will be typically based on some IETF YANG models over a Netconf interface. The capability to adapt such approach to the specific needs in our use case is very high, since we could use any YANG model agreed with the Controller vendor, in case the IETF ones were not enough.

Anyhow, the implementation of these features has not started yet. As mentioned above, the Use Case has been recently included in 5G EVE, and current focus is placed on integrating it with the available infrastructure on WP2 and the on-boarding processes of WP4. Resulting modifications in the IWL with regards to the gaming Use Case will be included in Deliverable D3.5.

2.1.2 Central repository for Site-related information

During the integration of components in the Interworking Framework and the development of new features, the requirement for a single repository to store information about 5G EVE sites and their features arose. Indeed, each component was relying on an internal, local storage to keep both its status and some contextual information, like for example the list of NFV Orchestrators registered with the 5G EVE infrastructure. While this solution kept each component as an independent and fully functional micro-service, data inconsistencies started to happen. Each component would need to implement multiple checks in the logic to ensure its local information is in sync with other components and subsequently map external data entities to local entities to ensure consistent behaviour.

Considering the issues mentioned above, the developers discussed and agreed on having a unique component to store shared information, while maintaining component-related, dynamic status in local storage elements. The new component, named IWF Repository, stores and provide CRUD operations for the following information elements:

- Sites
- NFV Orchestrators
- RAN Orchestrators
- Orchestrator credentials
- VIM accounts available at NFV Orchestrators

- VIM network names (management and external networks, statically configured in each site)
- VIM availability zones (to distinguish between resources available at cloud and at edge)
- RAN zones (to represent coverage areas and antenna capabilities)
- Subscriptions to notifications about Network Service lifecycle
- Data shippers

More information on the data model and implementation details can be found in Section 3.7.

2.1.3 RAN Orchestration

As we introduce above, a new kind of orchestrator is supported in the IWL. The goal is to configure and/or orchestrate the Radio Access Network stage of the 5G EVE sites, based on the requirements received from the 5G EVE Portal: Network Slice Type (eMBB, mMTC, URLLC), Technology (LTE, 5G mid-bad, 5G mmWaves, NB IoT, CAT-M) and Coverage zone (e.g. location, room).

The RAN orchestration shall adapt or deploy the RAN according to the received parameters and configure the required Connection Points with the 5GCore.

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 is the IWL component that is responsible for storing all of the Networks Service Descriptors (NSDs) and VNF Descriptors (VNFDs) that model single-site and multi-site vertical experiments. It is accessed by the 5G EVE Portal to retrieve the capabilities of each site facility in terms of available Network Services and VNFs to properly compose experiments according to the specific vertical requirements and constraints. Indeed, the 5G EVE Portal is allowed to onboard new NSDs in the Multi-site Catalogue whenever a vertical experiment is made of the composition of existing NSDs and VNFDs in one or more site facility. It is important to recall that according to the 5G EVE principles, VNFs are not allowed to be onboarded in the various site facilities from the I/W framework, or from the 5G EVE Portal. The VNF onboarding (and removal) is performed directly on the site catalogues (upon a related request issued on the 5G EVE Portal ticketing system) and through a human driven process by each site manager. For this reason, the Multi-site Catalogue implements specific catalogues synchronization mechanisms through which it is able to periodically retrieve the available VNFs in each site facility catalogue.

The Multi-site Catalogue follows the ETSI NFV SOL specifications, and in particular the data model for NSDs and VNFDs is aligned with the ETSI NFV SOL001 TOSCA models [8], while the northbound APIs are compliant with the ETSI NFV SOL005 v2.4.1 interfaces [5]. As the various local site catalogues (where site-specific NSDs and VNFDs are stored) implement their own data models (e.g. based on ETSI OSM or ONAP descriptors models), the Multi-site Catalogue takes care (in both I/W framework to local site catalogue and local site catalogue to I/W framework directions) to translate the NSDs and VNFDs from ETSI NFV SOL001 TOSCA model into the site-specific data model. Indeed, the Multi-site Catalogue embeds part of the I/W framework adaptation layer features with the aim of exposing at its northbound a common and unified NSD and VNFD data model for the 5G EVE Portal use.

In practice, the Multi-site Catalogue is conceived to provide the following Network Service and VNF on-boarding related procedures:

- *Catalogues synchronization*, to let the Multi-site Catalogue automatically retrieve, translate and store available NSDs and VNFDs from each site facility.
- *VNF onboarding*, to allow dynamic and automated retrieval, translation and storage 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 triggered by the 5G EVE Portal*, to allow the 5G EVE Portal to onboard in the I/W framework new NSDs for single- and multi-site vertical experiments. The descriptors are then translated and forwarded to the proper site catalogues into the specific data model format.
- *Network Service Descriptor onboarding triggered by the local site catalogues*, to allow dynamic and automated retrieval, translation and storage of new NSDs that are on-boarded directly in the 5G EVE sites at runtime.
- *Network Service Descriptor removal triggered by the 5G EVE Portal*, 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.

- *Network Service Descriptor removal triggered by the local site catalogues*, to allow dynamic and automated removal of existing NSDs in the Multi-site Catalogue when the related NSs are no longer available in the 5G EVE sites.

3.1.1 Software architecture

This section describes the second software release of the Multi-site Catalogue, as an incremental drop on top the first one delivered with deliverable D3.3 [3]. In terms of high-level software design (reported as part of the deliverable D4.1 [9] and referenced in D3.3), the main principles and functionalities are still valid. In particular, the Multi-site Catalogue is developed in Java, as Maven projects, and adopt PostgreSQL as backend database. It is an extension of the 5G Apps and Services Catalogue developed in the context of the 5G-MEDIA project[1] . As depicted in Figure 4, the Multi-site Catalogue exposes at its northbound a set of REST APIs, which are compliant with the ETSI NFV SOL005 interfaces defined for NSD Management and VNF Package Management. The core engine is implemented in the Catalogue Service, which provides the logic for Multi-site Catalogue workflows described above and coordinates the interactions with the underlying local site catalogues. For this, site-dependant NFVO drivers are integrated with a Kafka message bus, to take care, following an asynchronous approach, of the bi-directional translation of ETSI NFV SOL001 TOSCA data model into the specific local site catalogue data model (e.g. ETSI OSM or ONAP ones) for each of the NSD or VNFD that is either retrieved from the site facilities, or is onboarded from the 5G EVE Portal.

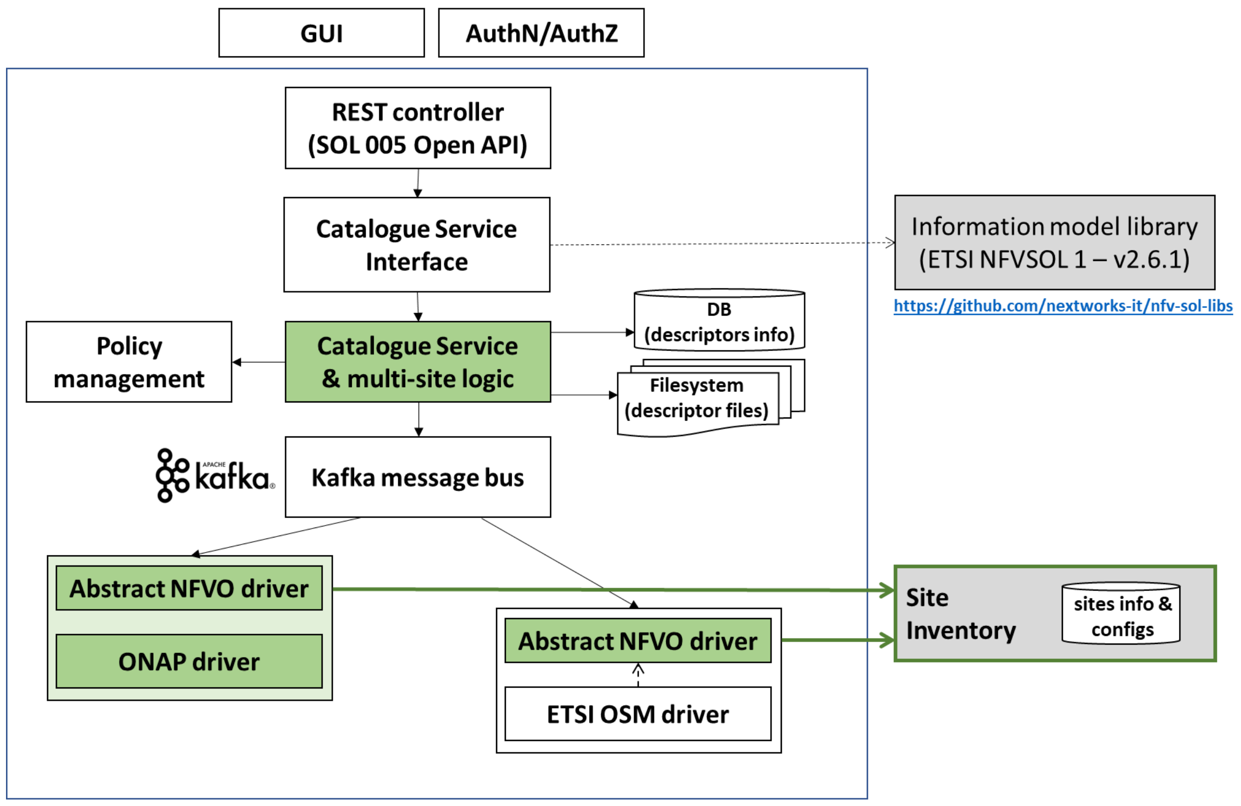
GUI
AuthN/AuthZ
REST controller (SOL 005 Open API)
Catalogue Service Interface
Information model library (ETSI NFVSOL 1 – v2.6.1)
https://github.com/nextworks-it/nfv-sol-libs
DB (descriptors info)
Policy management
Catalogue Service & multi-site logic
Filesystem (descriptor files)
APACHE kafka
Kafka message bus
Abstract NFVO driver
ONAP driver
Site Inventory
sites info & configs
Abstract NFVO driver
ETSI OSM driver

Figure 4: Multi-site Catalogue high level software design – second release version for D3.4

The Multi-site Catalogue stores the files included in the NSD and VNF packages (according to the ETSI NFV SOL004 [10] CSAR format) in the local filesystem, while summary information for each of the onboarded descriptor (including identifiers, mapping with local site catalogue identifiers, sites where a descriptor is onboarded) is stored into a backend database based on PostgreSQL.

[1] http://www.5gmedia.eu/

With respect to the first software release described in deliverable D3.3, some new features have been implemented in the Multi-site Catalogue for this second release. These additional functionalities are highlighted in green in Figure 4 and can be summarized as follows:

- New Catalogue Service logics for the support of (with reference to workflows listed in section 3.1 above):
    - VNF onboarding workflow.
    - VNF removal workflow.
    - Network Service Descriptor onboarding triggered by the local site catalogues workflow.
    - Network Service Descriptor removal triggered by the local site catalogues workflow.
- New NFVO driver for the integration with ONAP local site orchestrator.
- Integration with the I/W framework IWF Repository for automated NFVO drivers configuration.

3.1.2 Open API description

The Multi-site Catalogue northbound APIs are compliant with the ETSI NFV SOL005 v2.4.1 specification [5] and are implemented as REST APIs that provide the operations related to NSD Management and VNF Package Management.

Table 1: Services provided by Multi-Site Catalogue

Service	Path	Method	Input	Output	Description
Multi-Site Catalogue	/ns_descriptors	POST	Create NsdInfo Request	NsdInfo Instance	Create a new NSD resource
Multi-Site Catalogue	/ns_descriptors	GET	-	Array of NsdInfo	Returns the information of all NSD resources
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}	GET	-	NsdInfo	Returns the information of individual NSD resource
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}	DELETE	-	-	Delete individual NSD
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}/nsd_content	PUT	NSD file	-	Onboard new NSD content (TOSCA format)
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}/nsd_content	GET	-	NSD file	Returns the individual NSD content (TOSCA format)
Multi-Site Catalogue	/vnf_packages	GET	-	Array of VNFPckgInfo	Returns the information of all VNF Package resources
Multi-Site Catalogue	/vnf_packages/{vnfPkgId}/	GET	-	VNFPckgInfo	Returns the individual VNF Package resource
Multi-Site Catalogue	/vnf_packages/{vnfPkgId}/vnfd	GET	-	VNFD file	Returns the individual VNFD file (TOSCA format)

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

No updates have been implemented with respect to the APIs and related endpoints detailed in deliverable D4.1 ([9]) and referenced in D3.3 ([3]), as part of the Multi-site Catalogue software design and first prototype release documentation. However, for the sake of completeness of this document, we report in the Table 1 a summary of the APIs and related endpoints implemented and exposed by the Multi-site Catalogue.

3.1.3 Service description

This section reports the second release of the Multi-site Catalogue, which advances the first drop provided with D3.3. As depicted in Figure 4, new features have been implemented with the aim of supporting the full set of Multi-site Catalogue functionalities defined in D3.2. In summary, this new release of the Multi-site Catalogue now implements the full set of onboarding workflows described in section 3.1.1:

- Catalogues synchronization
- VNF onboarding
- VNF removal
- NSD onboarding triggered by the 5G EVE Portal
- NSD onboarding triggered by the local site catalogues
- NSD removal triggered by the 5G EVE Portal
- NSD removal triggered by the local site catalogues

As described in the previous section, this new version of the Multi-site Catalogue provides also a new NFVO driver to interact with ONAP local site orchestrators, beyond the one already existing for ETSI OSM local site orchestrators. This is a new feature that has been implemented from scratch to support the integration of the Multi-site Catalogue with the French site facility, where indeed ONAP is the centralized orchestrator responsible for the lifecycle management of Network Services and VNFs that implement the French site vertical experiments. In particular, this new Multi-site Catalogue ONAP driver is in charge to communicate with the Translation component described in section 3.8.1, as shown in Figure 5, to retrieve the available VNFs and experiment services available in the local ONAP catalogue, which are modelled following a TOSCA CSAR model [15] not compliant with the ETSI NFV SOL001 and SOL004 one used in the Multi-site Catalogue. With this software release, the ONAP driver supports the following workflows:

- Catalogue synchronization, to retrieve from ONAP (at start-up) all of the existing VNFDs and NSDs available in the local catalogue in the ONAP TOSCA CSAR format, and translate them into the Multi-site Catalogue ETSI NFV SOL001 TOSCA format.
- VNF onboarding, to automatically detect new VNFs onboarded on ONAP at runtime, translate them into the standard TOSCA format and store them in the local Catalogue DB.
- VNF removal, to automatically detect VNFs that are removed from the ONAP local catalogue at runtime, and delete them in turn from the Multi-site Catalogue.
- NSD onboarding triggered by the local site catalogues, to automatically detect new NSDs onboarded on ONAP at runtime, translate them into the standard TOSCA format and store them in the local Catalogue DB.
- NSD removal triggered by the local site catalogues, to automatically detect NSDs that are removed from the ONAP local catalogue at runtime, and delete them in turn in the Multi-site Catalogue

Moreover, a new integration with the IWF Repository has been also implemented in this release. This allows, for each new per-site NFVO driver instance automatically created when the Multi-site Catalogue application starts, to retrieve the proper information to access the given local site catalogue (e.g. in terms of credentials, tenants, URLs, endpoints) and automatically configure the NFVO driver itself. This allows to keep such information centralized in the IWF Repository and accessible by any of the I/W framework component, and avoid to have local Multi-site Catalogue configuration files (as it was for the previous software release described in D3.3).

The Multi-site Catalogue delivered with this second software release is available as opensource code on the Nextworks GitHub at:

https://github.com/nextworks-it/5g-catalogue/tree/v4.0.0

As an additional Multi-site Catalogue automated deployment feature, a Docker Compose script has been made available to have a containerized deployment of the Multi-site Catalogue as the combination of two Docker containers: i) one for the Multi-site Catalogue GUI front-end, ii) one for the Multi-site Catalogue backend application (that is the full set of software components within the green box in Figure 5). The Docker Compose script is available at:

https://github.com/nextworks-it/5g-catalogue/tree/v4.0.0/deployments/docker

As said, this is not to be considered as the final release of the Multi-site Catalogue software prototype. Additional features will be implemented in the coming months to fulfil the 5G EVE platform requirements for the lifecycle management of 5G vertical experiments.

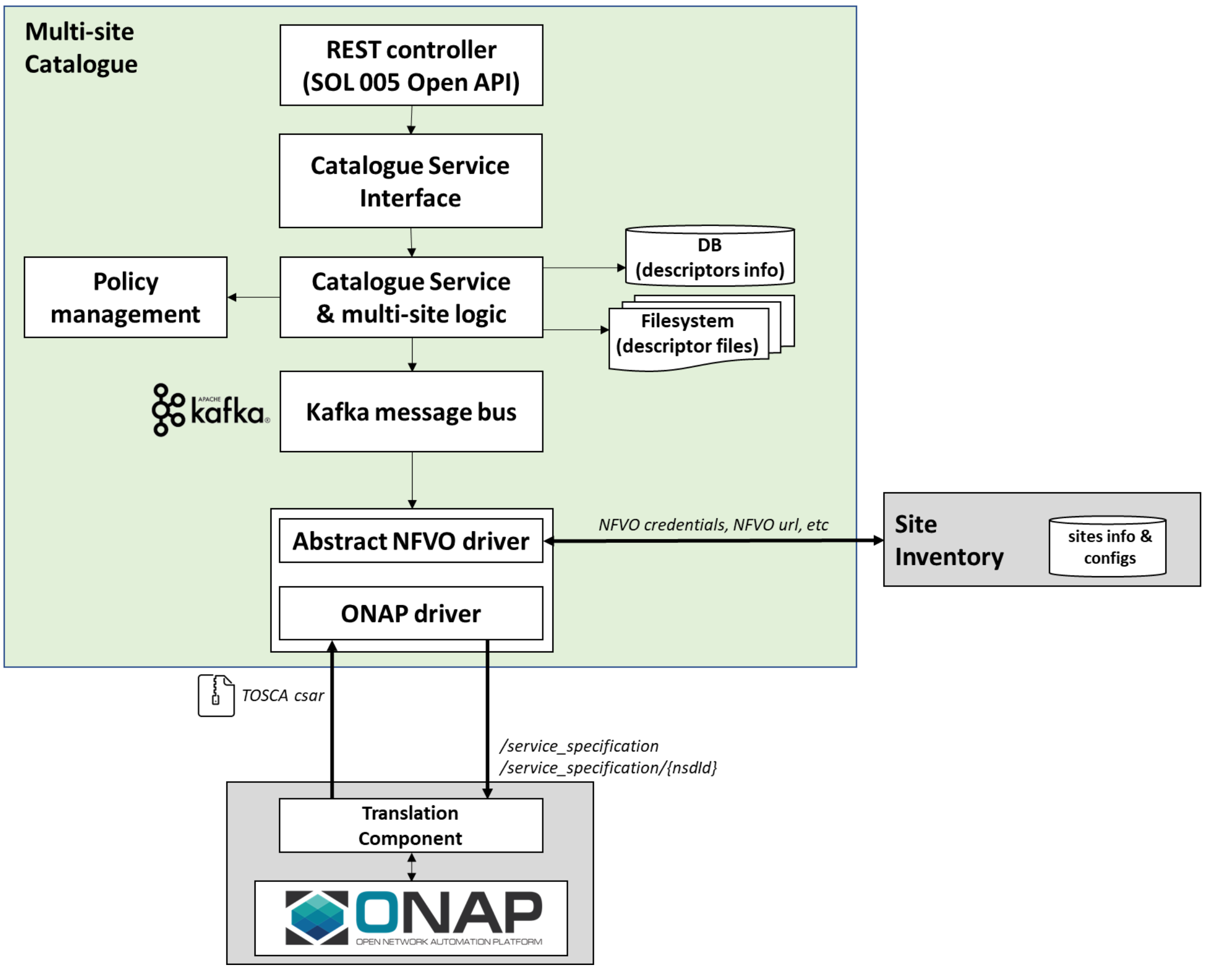
Multi-site Catalogue
REST controller (SOL 005 Open API)
Catalogue Service Interface
Policy management
Catalogue Service & multi-site logic
DB (descriptors info)
Filesystem (descriptor files)
kafka
Kafka message bus
Abstract NFVO driver
ONAP driver
NFVO credentials, NFVO url, etc
Site Inventory
sites info & configs
TOSCA csar
/service_specification
/service_specification/{nsdId}
Translation Component
ONAP
OPEN NETWORK AUTOMATION PLATFORM

Figure 5: Multi-site Catalogue ONAP driver

3.2 Multi-Site Inventory

The Multi-Site Inventory is the component of the Inter-Working Layer that provides information about the Network Services deployed in the 5G EVE sites.

3.2.1 Software architecture

The Multi-Site Inventory is integrated with the Multi-Site Network Orchestrator, as described in section 3.3.1.

3.2.2 Open API description

The north-bound interface of the Multi-Site Inventory is available at:

https://github.com/5GEVE/OpenAPI/tree/master/MSNO

It is based on the ETSI NFV SOL 005 interface and implements a subset of the Network Service Life Cycle Management.

3.2.3 Service description

The Multi-Site Inventory services are detailed in Table 2 and described in section 3.3.3.

Table 2: Multi-Site Inventory services

Resource Name	Resource URI	HTTP Method	Description
NS Instances	/ns_instances	GET	Returns the list of onboarded NS information
NS Instances	/ns_instances	POST	Creates a new NS into the MSNO
Individual NS Instances	/ns_instances/{nsInstanceId}	GET	Returns the information of a NS

3.3 Multi-Site Network Orchestrator

The Multi-Site Network Orchestrator (MSNO) is one of the key components for deploying Network Services in the 5G EVE sites. As described in previous deliverables (please refer to [1], [2] and [3]), the MSNO offers a unique and standard API for managing the lifecycle of the Network Services in all the 5G EVE sites.

The main mission of the MSNO is to unify the management of the Network Services, together with the Adaptation Layer, which provides a protocol conversion to per-site NFV-O APIs. It also interacts with other key IWL components like the Multi-Site Catalogue, from where MSNO obtains the NSD content and the NSD Information and with the IWL repository, which contains per-site information.

We describe here the delta implementation from D3.3 ([3]), which includes two software drops for MS8 and MS9. As we detail in the workflows, MSNO is fully integrated with the rest of the components of the IWL and provides all the features required for deploying experiments at MS9 status and supports ICT-19 projects.

3.3.1 Software architecture

The software architecture of the MSNO is based on a micro service architecture (Figure 6). We select this approach basically due to the flexibility that it provides during the development, especially with a small development team, as it allows to develop and to enhance each service individually.

The main micro-service is the Multi-Site Network Orchestrator service, which implements the northbound interface that provides service to the 5G EVE Portal. We develop an auxiliary micro-service for the notifications part of the northbound interface, for decoupling the on-demand service to the Portal from the asynchronous operations. MSNO service is also in charge of stablishing the communications with the other IWL components.

All the micro-services use a storage service for the persistent storage, called *storage* service. The persistency is guarantee by using an open source reliable data base, called MongoDB[2]

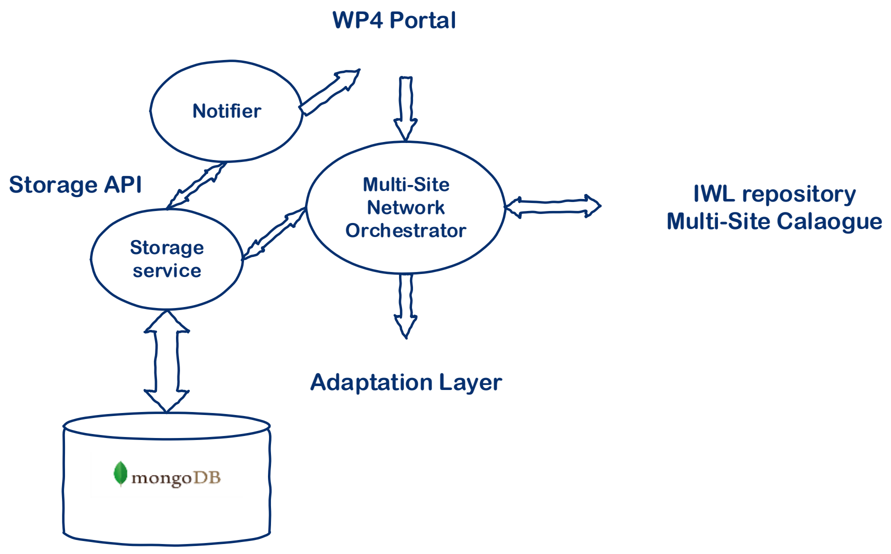

Figure 6: MSNO micro-service architecture

All the micro services share the same internal architecture, described in Figure 7. The development language selected for these processes is Go[3] and we use standard Go libraries for implementing the HTTP (net. HTTP), Service routing (gorilla.Mux) and YAML/JSON support.

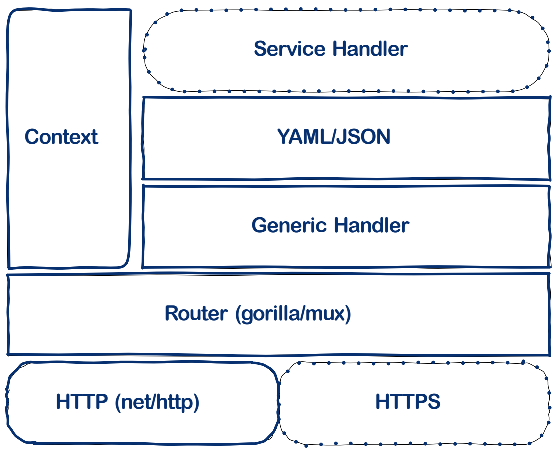

Figure 7: MSNO micro-service internal architecture

With this, we implemented a YAML parser in Python for automatically generating the basic code of each process from the OpenAPI specifications, which are described in the next section. This approach automatizes the common code of each micro-service and allows us to concentrate the effort in the development of the service logic. For the deployment of the micro services we selected the Docker[4] containers technology.

[2] https://www.mongodb.com

[3] https://golang.org/

[4] https://www.docker.com/

3.3.2 Open API description

The north-bound interface of the MSNO is available at:

https://github.com/5GEVE/OpenAPI/tree/master/MSNO

It is based on the ETSI NFV SOL 005 ([5]) interface and implements a subset of the Network Service Life Cycle Management.

3.3.3 Service description

As commented, the service provided by the Multi-Site Network Orchestrator (MSNO) is an implementation of the ETSI NFV SOL 005 interface related to the Network Service Life Cycle Management.

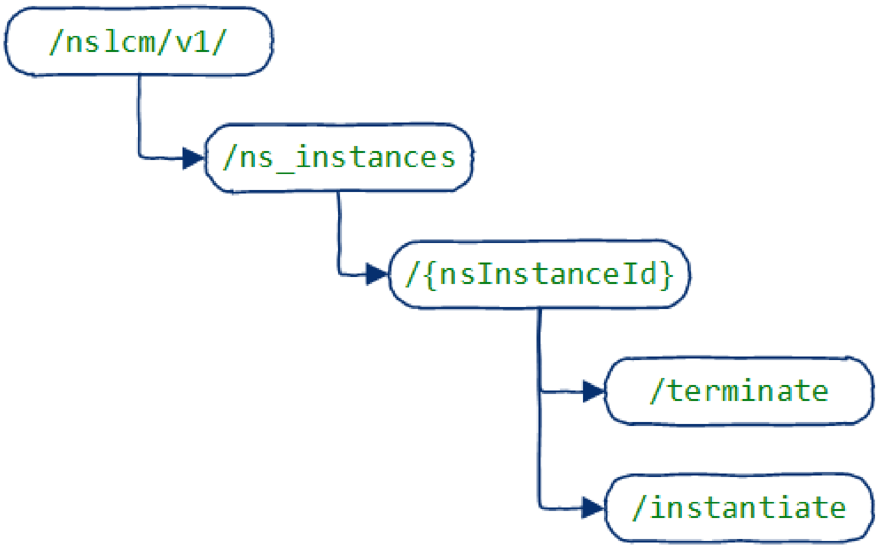

Figure 8: Multi-Site Network Orchestrator URI structure

Table 3: Service description

Resource Name	Resource URI	HTTP Method	Description
NS Instances	/ns_instances	GET	Returns the list of onboarded NS information
NS Instances	/ns_instances	POST	Creates a new NS into the MSNO
Individual NS Instances	/ns_instances/{nsInstanceId}	GET	Returns the information of a NS
Individual NS Instances	/ns_instances/{nsInstanceId}	DELETE	Deletes a NS from MSNO
Instantiate NS	/ns_instances/{nsInstanceId}/instantiate	POST	Instantiates a NS into target site
Terminate NS	/ns_instances/{nsInstanceId}/terminate	POST	Terminates an instantiated NS

3.3.3.1 Create Network Service Request

This request creates (onboard) a Network Service in the MSNO. It is the first step for deploying a Network Service into the Inter-Working Layer.

The MSNO does not raise any operation towards the 5G EVE sites as the target site is not known yet, so the NS remains in NOT_INSTANTIATE state in the MSNO:

The resource URI is: **{apiRoot}/nslcm/v1/ns_instances**, and the resource method is *POST*.

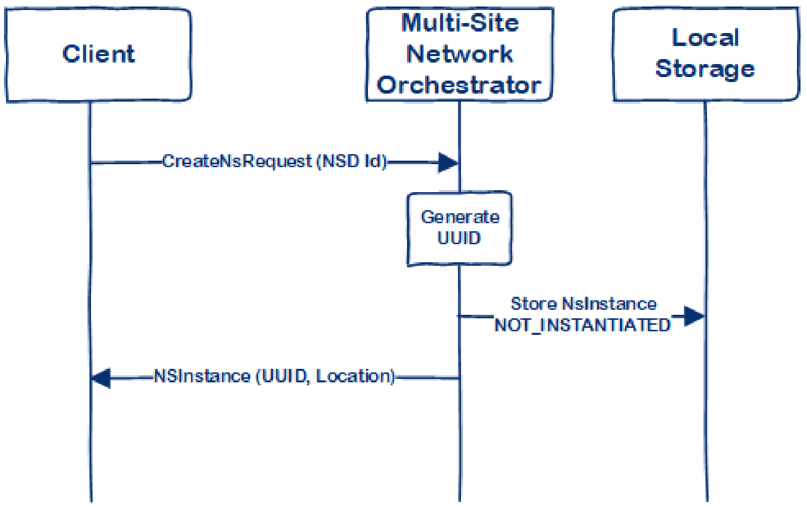
Client
Multi-Site Network Orchestrator
Local Storage
CreateNsRequest (NSD Id)
Generate UUID
Store NsInstance NOT_INSTANTIATED
NSInstance (UUID, Location)

Figure 9: Create Network Service Workflow

The workflow is as following:

1. Client sends a CreateNSRequest with the NSD ID of the Network Service to be deployed
2. MSNO generates a unique UUID for the Network Service Instance
3. MSNO stores the NS in the local storage with the state of NOT_INSTANTIATED
4. Response includes the UUID for the NS and the link to the resource

3.3.3.2 Query NS Instances

The Query Network Service Instances service returns the list of the Network Services that are onboarded in the Inter-Working Layer and its local status.[5]

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances**, and the resource method is *GET*.

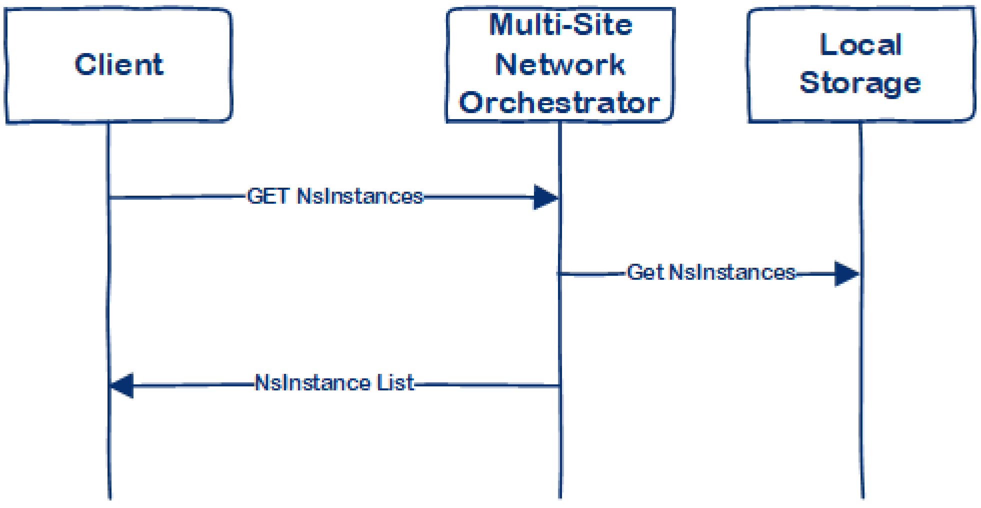

Figure 10: Workflow for the NS instances query

[5] As the end-2-end notification system is not ready, the local status could be outdated

3.3.3.3 Query NS Instance

This service allows to retrieve the status of a single Network Service Instance in the platform. MSNO will check with the site NFV-O the last status of the NS before informing the client.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}**, and the resource method is *GET* .

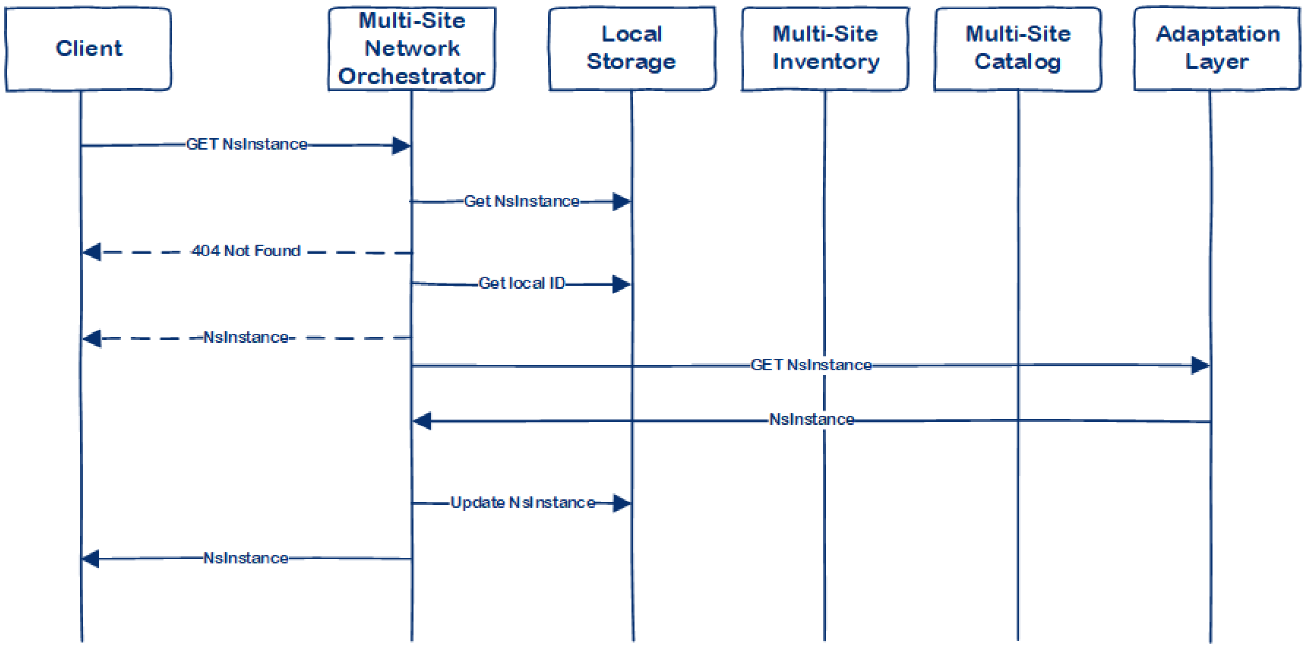

Figure 11: Workflow for Query

The workflow is as follows:

1. Client sends a GET request including NS instance ID
2. MSNO retrieves NS Instance local information for checking if the NS is onboarded
3. MSNO retrieves local NS ID from the mapping database.
4. In case that NS has not instantiated yet, MSNO returns local information
5. In case that NS has been instantiated, MSNO retrieves NS information from the local site
6. Local NS information is updated
7. Updated NS information is sent to the client

3.3.3.4 Instantiate NS

This service allows to instantiate a Network Service that is previously onboarded into the IWL. This is the second step for deploying a NS into 5G EVE as the result is the Network Service deploying in the target site selected by the Experimenter.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}/instantiate**, and the resource method is *POST*.

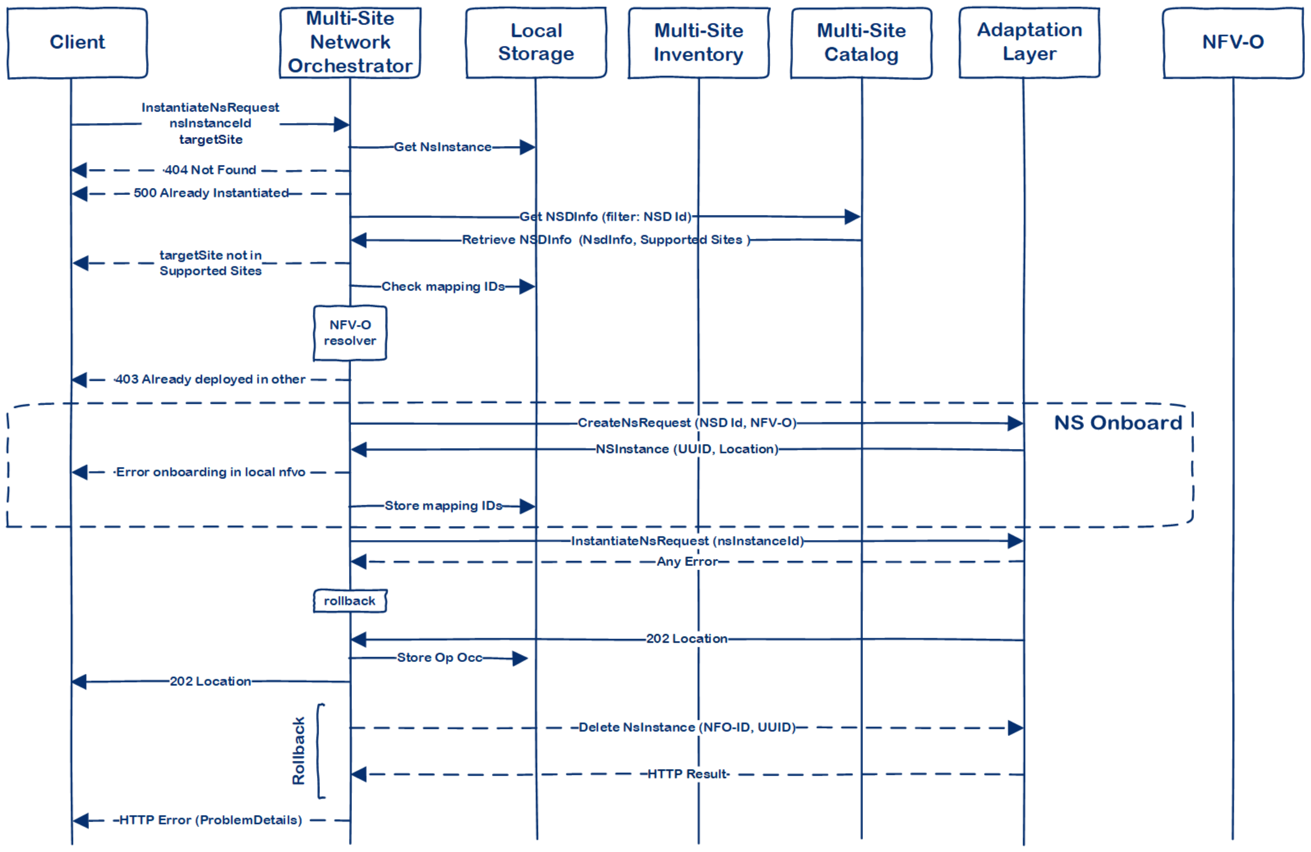
Client
Multi-Site Network Orchestrator
Local Storage
Multi-Site Inventory
Multi-Site Catalog
Adaptation Layer
NFV-O
InstantiateNsRequest nsInstanceId targetSite
Get NsInstance
404 Not Found
500 Already Instantiated
Get NSDInfo (filter: NSD Id)
Retrieve NSDInfo (NsdInfo, Supported Sites)
targetSite not in Supported Sites
Check mapping IDs
NFV-O resolver
403 Already deployed in other
NS Onboard
CreateNsRequest (NSD Id, NFV-O)
NSInstance (UUID, Location)
Error onboarding in local nfvo
Store mapping IDs
InstantiateNsRequest (nsInstanceId)
Any Error
rollback
202 Location
Store Op Occ
202 Location
Rollback
Delete NsInstance (NFO-ID, UUID)
HTTP Result
HTTP Error (ProblemDetails)

Figure 12: Workflow for the Instantiate Request

The workflow is:

1. The client sends an Instantiate request with the NS Instance ID and the target 5G EVE site
2. MSNO retrieves NS information from local storage
3. In case that the NS instance does not exists, or it has been instantiated, MSNO generates the correspondent error
4. MSNO retrieves the NSD Info information from Multi-Site Catalogue, which includes the supported sites for the NS instance
5. In case that target 5G EVE site is not in the supported sites list, the request is rejected
6. MSNO determines which site NFV-O will use for deploying the NS
7. MSNO checks if the NS has been deployed in a site.
8. In case that the target site is different to the site that has onboarded the NS, MSNO sends a 403 Forbidden error. If the local site is the same, MSNO skips onboarding phase (steps 9, 10) and sends an Instantiation request (step 11)
9. MSNO generates a CreateNsRequest towards the selected NFV-O, through the Adaptation Layer
10. In case of success, MSNO stores the mapping between NS Instance ID and the UUID provided by the site NFV-O
11. MSNO instantiates the NS in the local NFV-O
12. A positive response is generated

In case of error during the workflow, the MSNO raises the rollback procedure, which is in charge of deleting any NS created in a local NFV-O.

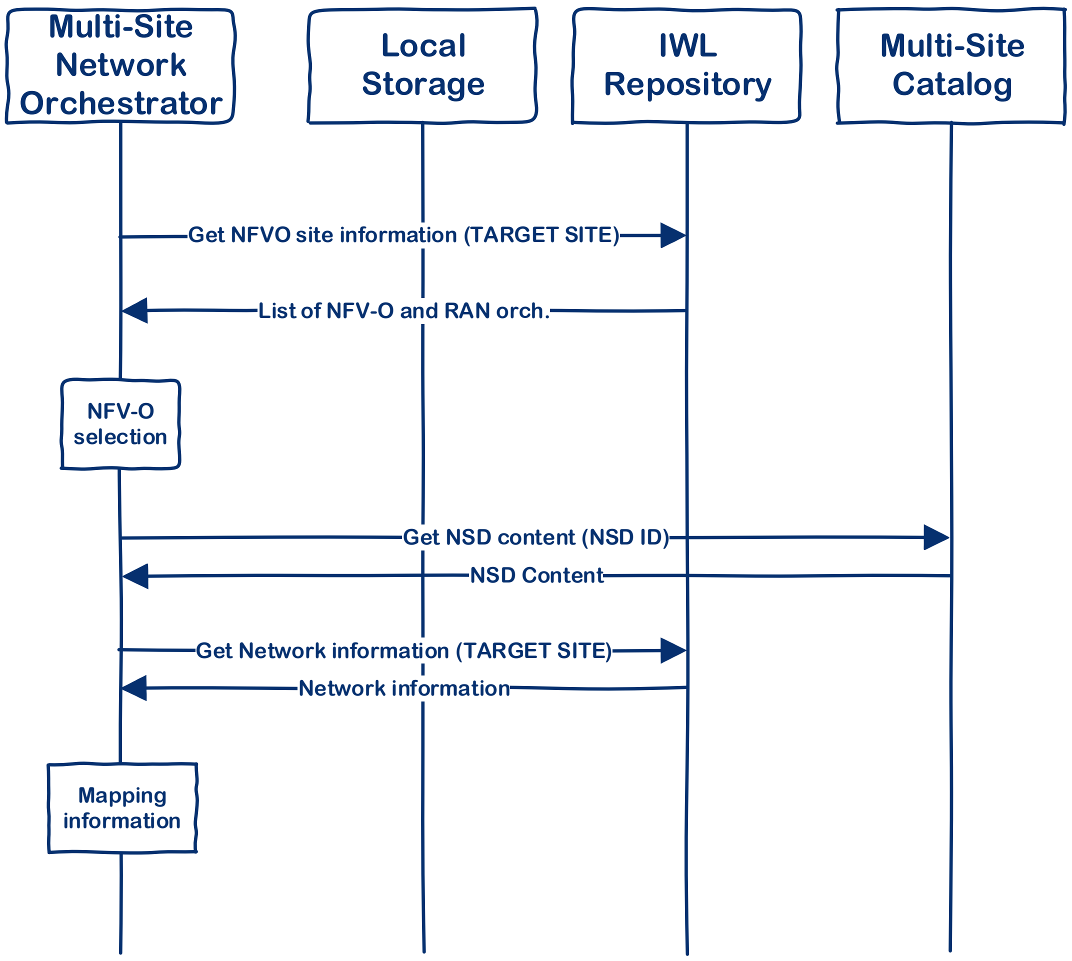
Multi-Site Network Orchestrator
Local Storage
IWL Repository
Multi-Site Catalog
Get NFVO site information (TARGET SITE)
List of NFV-O and RAN orch.
NFV-O selection
Get NSD content (NSD ID)
NSD Content
Get Network information (TARGET SITE)
Network information
Mapping information

Figure 13: NFV-O resolution

In the MS9 scope, IWL supports a multi orchestration scenario, with one orchestrator for RAN and another one for NFV management. For that, it implements the selection of the NFV-O as follows:

1. MSNO retrieves from IWL repository the list of orchestrators.
2. MSNO selects the appropriate NFV-O for the NS management
3. MSNO retrieves the NSD content from the MSC and determines if it is required a mapping between the network names and the site implementation of public networking
4. MSNO retrieves the network mapping information from IWL repository
5. MSNO includes the mapping information in the Instantiation Request body

3.3.3.5 Terminate NS

This service allows to terminate a Network Service

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}/terminate**, and the resource method is *POST*.

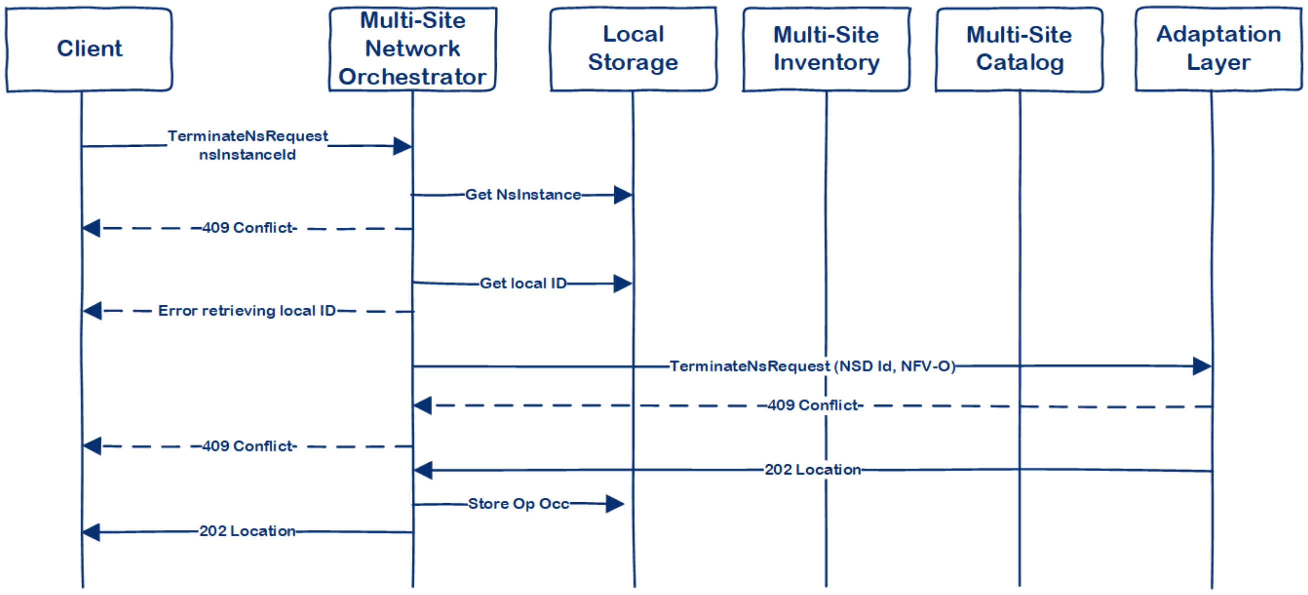

Figure 14: Terminate Network Service Workflow

The workflow is:

1. Client sends a Terminate request with the NS Instance ID
2. MSNO retrieves local NS instance information
3. In case of NS state is not INSTANTIATED, MSNO responds with a 409 Conflict (Note: as the local status in this drop is not synced with local NFVO, this check is skipped)
4. MSNO retrieves local ID for the NS instance
5. MSNO sends the Terminate request to the local NFV-O
6. MSNO forwards the response to the client

3.3.3.6 Delete NS

Deletes a Network Service from the 5G EVE platform. All resources are cleaned at IWL level as well as site NFV-O level.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}**, and the resource method is *DELETE.*

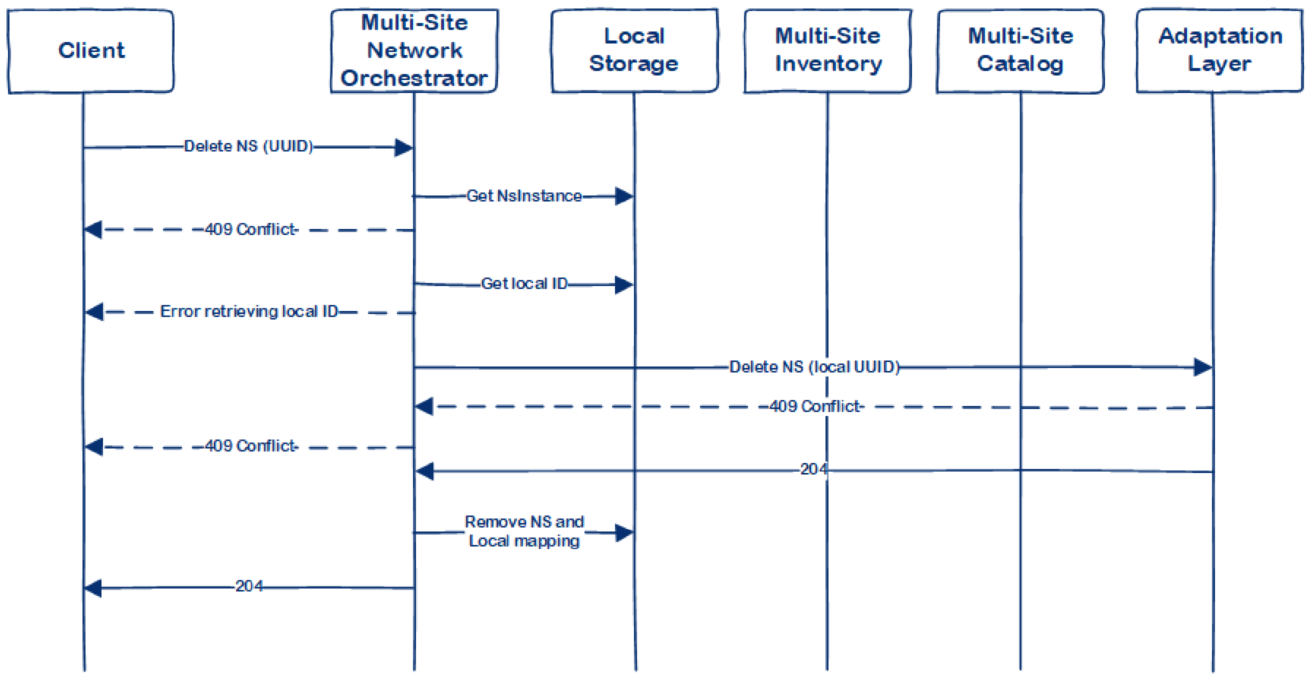

Figure 15: Delete NS workflow

The workflow is:

1. Client sends a delete request with the NS instance ID
2. MSNO retrieves local NS information
3. In case of NS state is not NOT_INSTANTIATED, MSNO responds with a 409 Conflict (Note: as the local status in this drop is not synced with local NFVO, this check is skipped)
4. MSNO retrieves local ID for the NS instance
5. MSNO sends the Delete NS request to the local NFV-O
6. MSNO deletes NS and local ID from storage
7. MSNO forwards the response to the client

3.4 Data Collection Manager

As summarised in the deliverable D3.3 ([3]), the Data Collection Manager is the component responsible for the collection, persistence and delivery of all the network and vertical performance metrics, and also KPI values, which are required to be gathered during the execution of experiments, with two main objectives: monitor the experiment (thanks to the Monitoring and Result Collection tools provided within WP4 scope) and validate the targeted KPIs (thanks to the KPI Validation Framework proposed and developed within WP5 scope).

For this purpose, the publish-subscribe paradigm is used in order to fully implement this workflow. In that way, there are different components subscribed to specific topics, which play the role of communication channel to successfully deliver the monitored data published from one entity to the components subscribed to that topic.

Next, the different improvements integrated in the Data Collection Manager, based on Apache Kafka[6] and Apache ZooKeeper[7], will be presented.

3.4.1 Software architecture

In the current implementation of the DCM, the architecture selected to build and deploy it has evolved into a simpler approach, compared to the software architecture presented in D3.3, based on the integration of the Kafka Confluent[8] platform to provide value-added features on top of the Apache Kafka and Apache ZooKeeper tools.

The architecture in which the DCM has been implemented is presented in the next Figure 16.

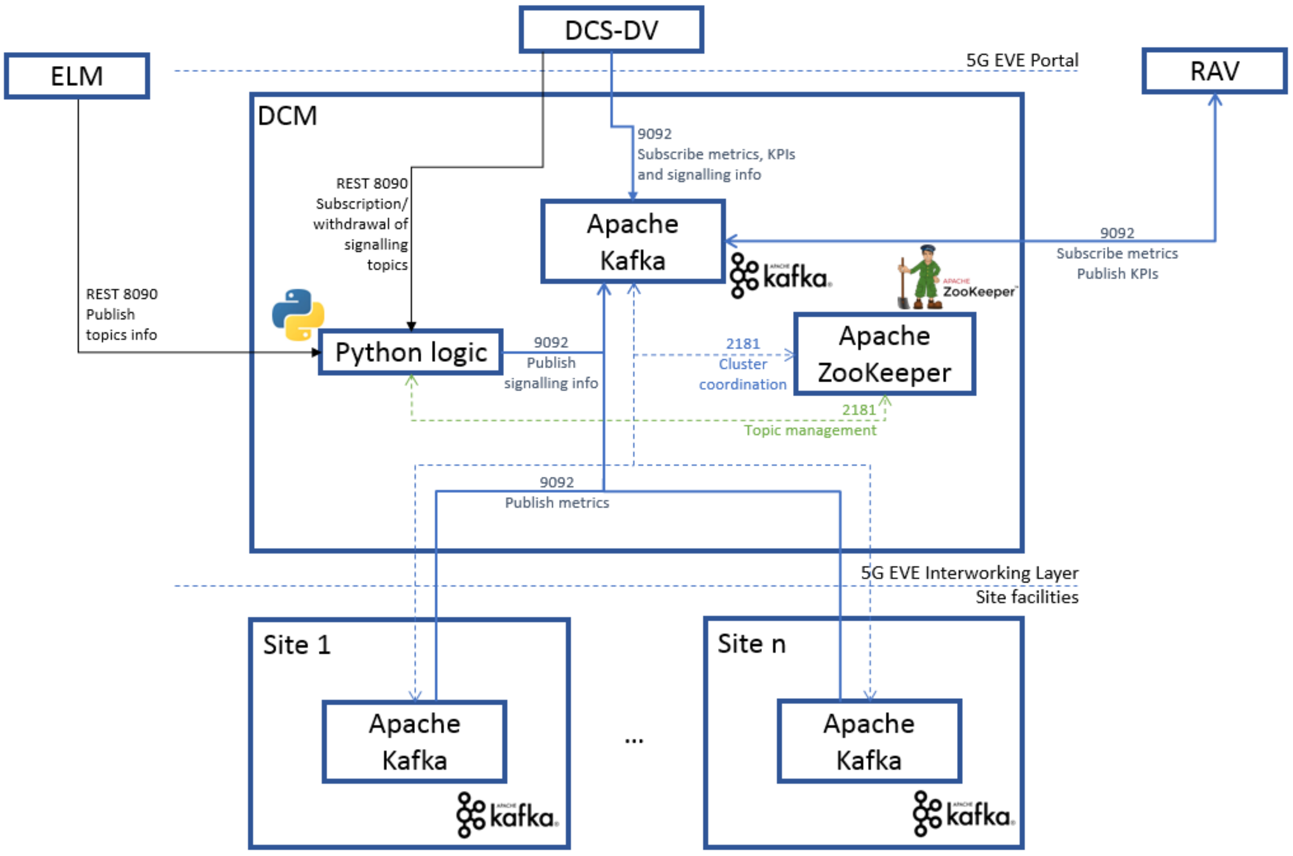

Figure 16: Data Collection Manager final architecture

In the architecture, Apache Kafka and Apache ZooKeeper are present. The first one is used for implementing the publish-subscribe paradigm to coordinate the operation between the different modules interested in the delivery of metrics' (both application and infrastructure) and KPIs' values, while the second one is in charge of controlling the cluster configuration built among all the Apache Kafka components present in the architecture.

Moreover, to automate the process and to include some value-added features, a lightweight Python logic has been also implemented, which mainly takes care of the topic management in coordination with the ELM (for the topics related to the experiments) and with the DCS-DV (for the signalling topics). This piece of software intends to take the role played originally by Kafka Confluent, as all the features provided by that tool are not really needed in the context of 5G EVE. This Python logic can be found here: https://github.com/5GEVE/5geve-wp3-dcm-handler.

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

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

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

Regarding site facilities, note that Apache ZooKeeper acts as the central component that coordinates all the cluster activity, so it is the responsible for correctly configure all the topics handled by the Monitoring platform and also the replication mode to be followed by each topic. Regarding this last fact, what it is intended to do is to replicate the data from each site facility to the DCM, so that the DCM will be able to handle all the topics related to the experiments handled in all site facilities, but site facility will only work with the topics related to experiments deployed in their own site. This ensures that data that come from a single site facility is not replicated towards another site facility, which is a security requirement that must be fulfilled. This particular architecture will be described with more detail in the next deliverable D3.5.

The architecture in Figure 16 also reflects the interactions between the different modules, also including the ports used in that communication:

- Port 2181: standard ZooKeeper port, used for coordinating the Kafka cluster from ZooKeeper, and also used as destination port by the Python logic to manage the lifecycle of the topics provided by the ELM.
- Port 8090: Python's logic listening port which enables the operations exposed in its OpenAPI, as discussed in chapter 3.4.2.
- Port 9092: standard Kafka port, used for handling the delivery of data between the publishers and the subscribers. This interaction is fully described in D3.3 – chapter 3.4.3.2, but now the interaction between the Python logic and the DCM and DCS-DV is also included for handling the signalling topic's information, previous to the execution of the experiments.

Note that persistence configuration, in the case of the Kafka servers placed in each site facility, will depend on the particular requirements of each site. In the case of the Kafka server located in the DCM, the retention time defined will be enough to cover the whole experiment lifecycle, but the data persistence service will be offered by the DCS-DV instead, as the possibilities that it provides (data pre-processing, filtering, indexing, etc.) allows the Monitoring platform to better handle and manage the monitored data generated. In any case, a disk size theoretical estimation based on the worst case (i.e. all the use cases performing experiments at the same time, managing 20 topics per experiment, and with a retention time of two weeks) results in a value slightly below to 100 TB ([14]), this being a hard upper limit to have in mind. However, according to the tests already done and the overview of the different use cases (i.e. they will not be executed simultaneously, and the topics handled are around 5), the amount of disk needed could be around 100 GB with that assumptions.

Finally, the implementation of the DCM architecture is thought to be deployed in a single, Linux server, physical or virtual (VM). The current Linux distribution used to deploy the DCM is Ubuntu Server 16.04 LTS, and the deployment steps can be found in the following repository: https://github.com/5GEVE/5geve-wp3-dcm-deployment. There, a specific Ansible playbook provides all the commands, files and configurations needed to setup a DCM instance from scratch or to update it to the latest version.

3.4.2 Open API description

According to the software architecture presented in chapter 3.4.1, the OpenAPI specification that corresponds to the REST API offered by the DCM Python logic can be found in the following link: https://github.com/5GEVE/OpenAPI/tree/master/DataCollectionManager. This specification fully replaces the Kafka Confluent's REST Proxy OpenAPI definition presented in deliverable D3.3.

Then, the operations exposed by the DCM through the Python API, which are aligned with the three main operations detected in D3.3 (i.e. subscribe, unsubscribe and publish), are the following:

Table 4: Data Collection Manager operations from its Open API specification

Service	Path	Method	Input	Output	Description
Data Collection Manager	/dcm/subs-cribe	POST	expId = internal topic = signalling topic name	201 – accepted 400 - error	Subscribe[9] to the signalling topic provided as input in the body request.
Data Collection Manager	/dcm/un-subscribe	DELETE	expId = internal topic = signalling topic name	201 – accepted 400 - error	Unsubscribe to the signalling topic provided as input in the body request.
Data Collection Manager	/dcm/pu-blish/<to-pic>	POST	topic = signalling topic[10]. Array of records[11] containing values with the information related to the topics to be subscribed/un-subscribed.	201 – accepted 400 – error	Publish the information related to the topics that will be created during the experiment (and deleted afterwards) in the signalling topics, so that the DCM triggers all the mechanisms to create the topics in Kafka and to distribute them to the proper entities.

3.4.3 Service description

In the DCM service description, the topic framework proposal already presented in the D3.3 will be updated with the last changes included to adapt it to the new features enabled in the platform. Then, the workflows presented in D3.3 – Annex A.3 will be updated with the current architecture specification showed in chapter 3.4.1. Finally, the information model to be used for publishing metrics' and KPIs' values in the Monitoring platform will be also introduced.

3.4.3.1 Topic framework proposal update

As a reminder, the topic framework proposal intends to model the data handled by the Data Collection Manager in terms of different topics to be used, depending on the traffic expected by the Data Collection Manager.

In this case, the distinction between signalling topics and data topics are still maintained. However, there are several changes in both that must be commented:

- **Signalling topics:** in this case, it has been adapted to cover both infrastructure and application metrics, using different signalling topics for each of them. Moreover, results' data will no longer be handled, so its signalling topic is directly removed from the proposal. Then, the signalling topic used by the DCM are:
    - **signalling.metric.infrastructure:** signalling topic for infrastructure metric's data.
    - **signalling.metric.application:** signalling topic for application metric's data.

[9] Note that the subscribe and unsubscribe operations for the topics related to metrics and KPIs do not need to expose an external API, as it is directly handled by the Python logic with direct interactions with Apache ZooKeeper.

[10] For the moment, the data managed by this endpoint is exclusively related to the messages sent by the ELM to subscribe/unsubscribe to the topics related to the experiment. However, this endpoint can also be used to publish data in Kafka in another topic. The considerations to allow this operation will be described in D3.5 if it is finally needed.

[11] Examples with the format of this array of records can be found in deliverable D4.4 – Annex A.

 - **signalling.kpi:** signalling topic for KPI's data.
- **Data topics:** as results are no longer monitored by the platform, they are also removed from the definition of the data topics. Moreover, a new field, called <*uc*> (related to the use case) is also included for enabling some functionalities in the DCS-DV in the 5G EVE Portal. Then, the current format to be followed, updating the last one presented in D4.3, is the following*:* ***<uc>.<exp>.<site>.[application_metric|infrastructure_metric|kpi].<value>***, where:
  - *<uc>:* it identifies the use case related to the topic. It will be used to handle user permissions in the DCS-DV.
  - *<exp>:* is the experiment ID, uniquely identifying the experiment itself. This value is generated and assigned by the ELM.
  - *<site>:* identifies the site facility where the data is extracted (e.g. Spain, Italy, etc.).
  - *[application_metric|infrastructure_metric|kpi]:* it is a key word that indicates the type of information to which belongs the next parameter.
  - *<value>:* set the name of the parameter to be monitored (e.g. latency), identifying the metric/KPI.

3.4.3.2 Updated workflows

Regarding the workflows, the three main phases detected in D3.3 – Annex A.3 have been maintained, but also updating them with the new features enabled in the Monitoring platform. Then, the workflow would be the following:

- **Subscription phase (**Figure 17**)**: in the updated workflow presented in the following figure, note that the workflow followed by the Result Analysis and Validation component is independent from the workflow followed between the ELM and the DCM to exchange the topic names, as the provision of the topics is directly handled by the Experiment Execution Manager in that case. Moreover, in the subscribers installed in the DCS, they will be created two consumers for each data topic: one is the Logstash pipeline that will ingest all the monitored data that comes from the corresponding publishers, and the other one is a single Kafka consumer that will detect when the first message arrives to the DCS, in order to generate the corresponding dashboards related to the monitored metric/KPI.
- **Monitoring and data collection phase (**Figure 18**):** this workflow is practically the same than the one presented in D3.3, but also including some considerations related to the visualization of metrics and KPIs in the Portal GUI. As commented in the subscription phase, when the first message arrives at the DCS, it triggers the creation of the corresponding dashboards, and then the Kafka consumer related to that process is closed. At the end of the workflow, which can be seen in the following figure, it is also presented the interaction between the user and the Portal GUI, performed during all the experiment execution. When the user enters in the Experiment Metrics Dashboards' page in the 5G EVE Portal GUI, a request to the DCS is performed in order to obtain the URLs of the dashboards to present the metrics and KPIs in an on-line fashion. The components and the architecture implemented in the Portal GUI for this purpose are fully described in the deliverable D4.4 ("Report on benchmarking of new features", which will be deliver in parallel to this deliverable).
- **Withdrawal phase (**Figure 19**):** again, it is the opposite operation to the subscription phase. In figure below, it is also presented, at the end of the picture, the withdrawal process of the signalling topics, which is performed when no more experiments are expected during a consider period of time, just to stop the Monitoring platform.

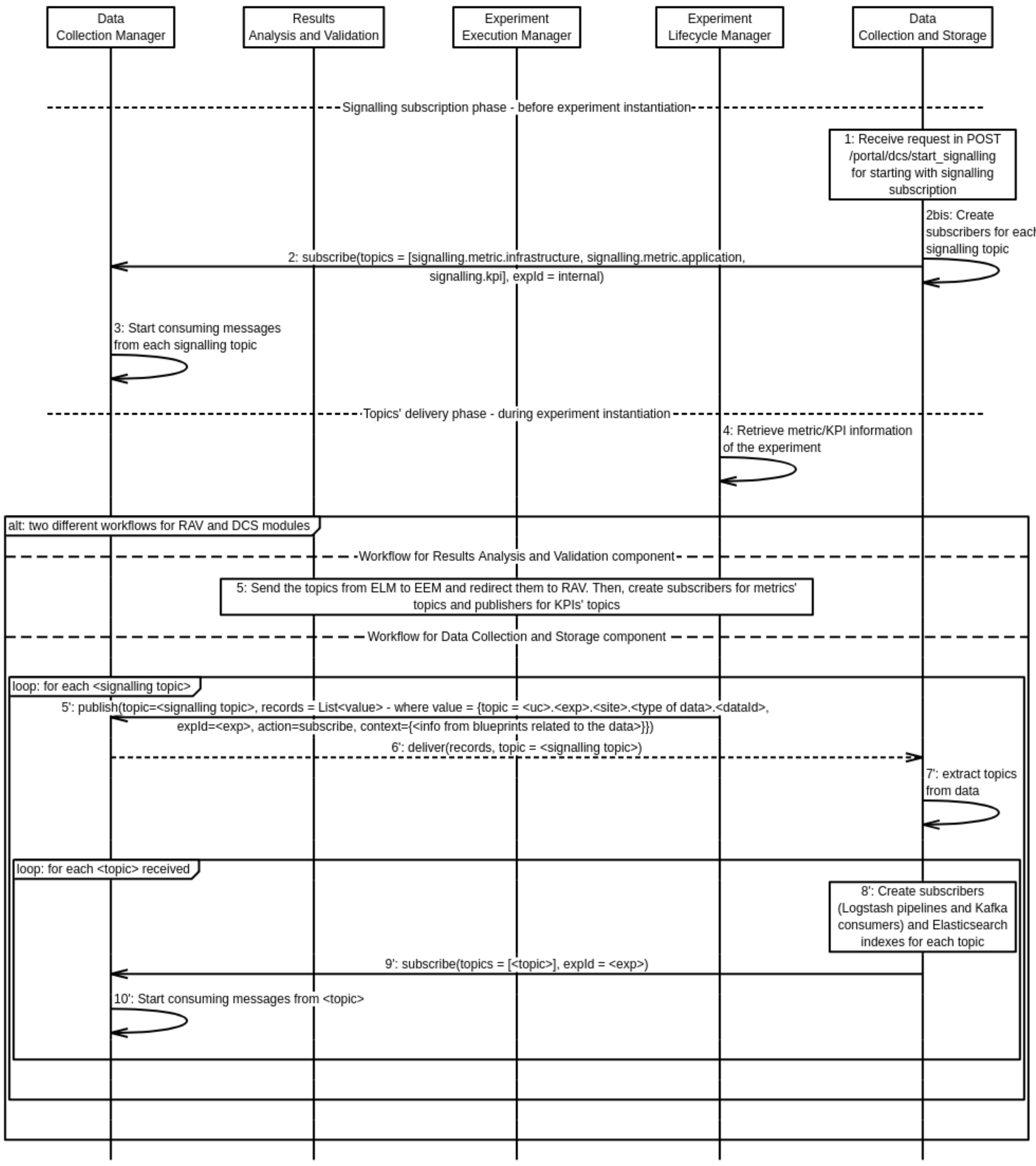
Data Collection Manager
Results Analysis and Validation
Experiment Execution Manager
Experiment Lifecycle Manager
Data Collection and Storage
Signalling subscription phase - before experiment instantiation
1: Receive request in POST /portal/dcs/start_signalling for starting with signalling subscription
2bis: Create subscribers for each signalling topic
2: subscribe(topics = [signalling.metric.infrastructure, signalling.metric.application, signalling.kpi], expId = internal)
3: Start consuming messages from each signalling topic
Topics' delivery phase - during experiment instantiation
4: Retrieve metric/KPI information of the experiment
alt: two different workflows for RAV and DCS modules
Workflow for Results Analysis and Validation component
5: Send the topics from ELM to EEM and redirect them to RAV. Then, create subscribers for metrics' topics and publishers for KPIs' topics
Workflow for Data Collection and Storage component
loop: for each <signalling topic>
5': publish(topic=<signalling topic>, records = List<value> - where value = {topic = <uc>.<exp>.<site>.<type of data>.<dataId>, expId=<exp>, action=subscribe, context={<info from blueprints related to the data>}})
6': deliver(records, topic = <signalling topic>)
7': extract topics from data
loop: for each <topic> received
8': Create subscribers (Logstash pipelines and Kafka consumers) and Elasticsearch indexes for each topic
9': subscribe(topics = [<topic>], expId = <exp>)
10': Start consuming messages from <topic>

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

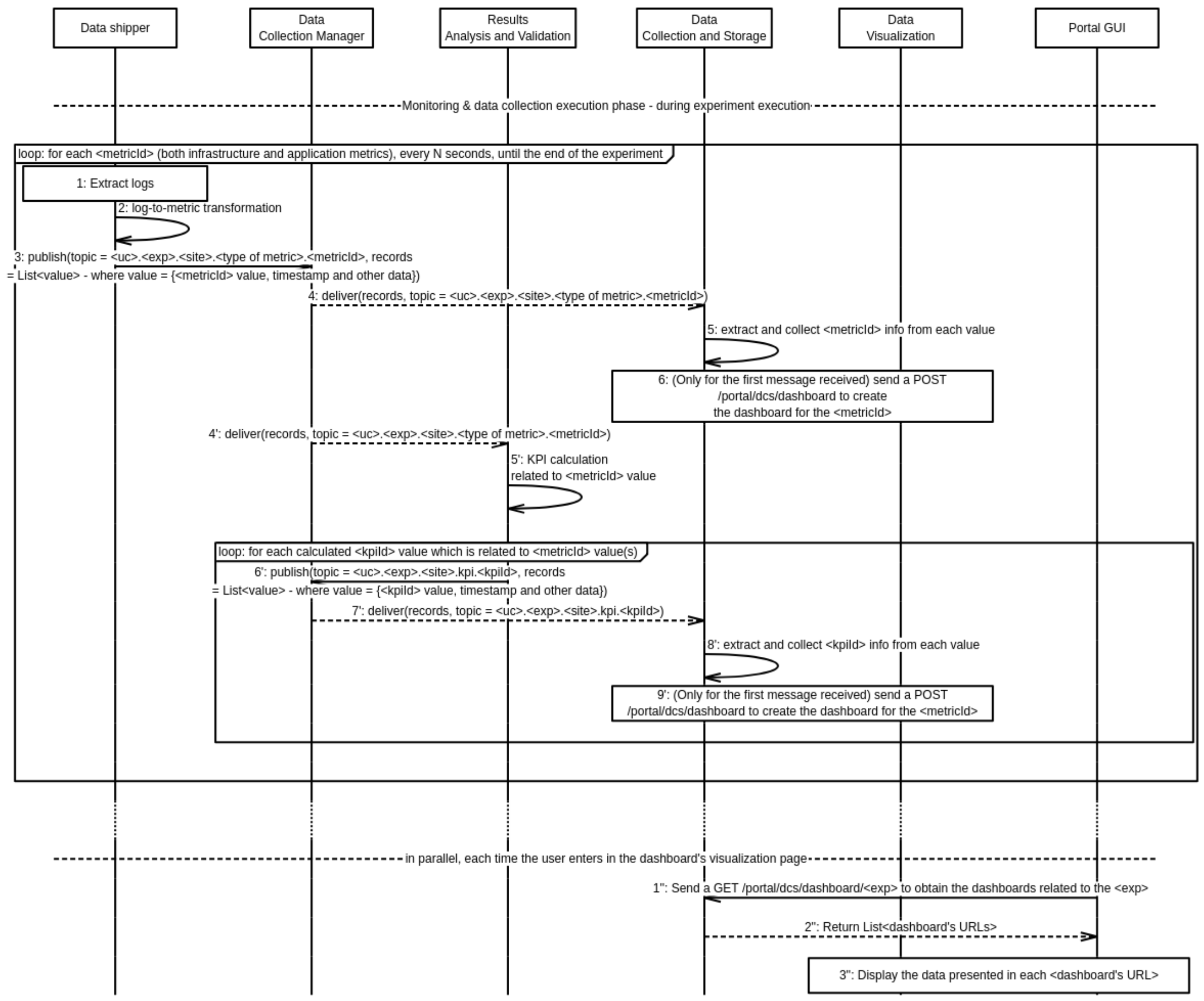
Data shipper
Data Collection Manager
Results Analysis and Validation
Data Collection and Storage
Data Visualization
Portal GUI
Monitoring & data collection execution phase - during experiment execution
loop: for each <metricId> (both infrastructure and application metrics), every N seconds, until the end of the experiment
1: Extract logs
2: log-to-metric transformation
3: publish(topic = <uc>.<exp>.<site>.<type of metric>.<metricId>, records = List<value> - where value = {<metricId> value, timestamp and other data})
4: deliver(records, topic = <uc>.<exp>.<site>.<type of metric>.<metricId>)
5: extract and collect <metricId> info from each value
6: (Only for the first message received) send a POST /portal/dcs/dashboard to create the dashboard for the <metricId>
4': deliver(records, topic = <uc>.<exp>.<site>.<type of metric>.<metricId>)
5': KPI calculation related to <metricId> value
loop: for each calculated <kpiId> value which is related to <metricId> value(s)
6': publish(topic = <uc>.<exp>.<site>.kpi.<kpiId>, records = List<value> - where value = {<kpiId> value, timestamp and other data})
7': deliver(records, topic = <uc>.<exp>.<site>.kpi.<kpiId>)
8': extract and collect <kpiId> info from each value
9': (Only for the first message received) send a POST /portal/dcs/dashboard to create the dashboard for the <metricId>
in parallel, each time the user enters in the dashboard's visualization page
1'': Send a GET /portal/dcs/dashboard/<exp> to obtain the dashboards related to the <exp>
2'': Return List<dashboard's URLs>
3'': Display the data presented in each <dashboard's URL>

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

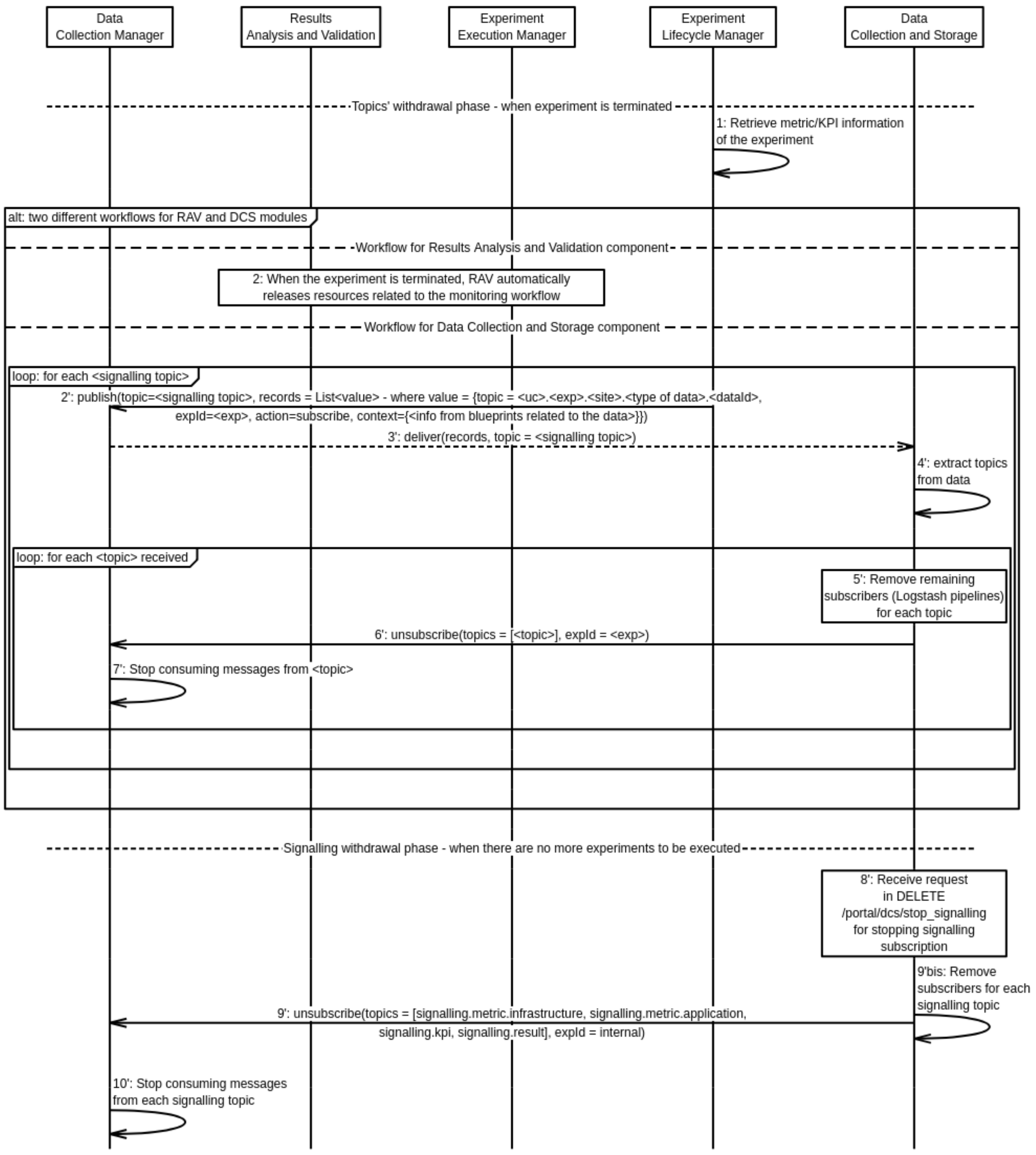
Data Collection Manager
Results Analysis and Validation
Experiment Execution Manager
Experiment Lifecycle Manager
Data Collection and Storage
Topics' withdrawal phase - when experiment is terminated
1: Retrieve metric/KPI information of the experiment
alt: two different workflows for RAV and DCS modules
Workflow for Results Analysis and Validation component
2: When the experiment is terminated, RAV automatically releases resources related to the monitoring workflow
Workflow for Data Collection and Storage component
loop: for each <signalling topic>
2': publish(topic=<signalling topic>, records = List<value> - where value = {topic = <uc>.<exp>.<site>.<type of data>.<dataId>, expId=<exp>, action=subscribe, context={<info from blueprints related to the data>}})
3': deliver(records, topic = <signalling topic>)
4': extract topics from data
loop: for each <topic> received
5': Remove remaining subscribers (Logstash pipelines) for each topic
6': unsubscribe(topics = [<topic>], expId = <exp>)
7': Stop consuming messages from <topic>
Signalling withdrawal phase - when there are no more experiments to be executed
8': Receive request in DELETE /portal/dcs/stop_signalling for stopping signalling subscription
9'bis: Remove subscribers for each signalling topic
9': unsubscribe(topics = [signalling.metric.infrastructure, signalling.metric.application, signalling.kpi, signalling.result], expId = internal)
10': Stop consuming messages from each signalling topic

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

3.4.3.3 Information model to publish monitored data in the Monitoring platform

In order to publish the data related to the metrics' and KPIs' values to be handled by the Monitoring platform in a unified way, an information model to define the specific fields that the message containing that data must have is presented here.

In this proposed information model, the publication in Kafka implies the building of a specific JSON string, in which the data needed by upper layers is included. This JSON string must be built by the Data shipper (i.e. light pieces of software that collect the data related to monitored metrics in each VNF/PNF/specific component related to a given experiment and publish them in the corresponding Kafka broker) and the publishers used in other components (e.g. the RAV), including the values that correspond to the expected fields in the message.

For publishing a message in Kafka, the publisher will need in advance the IP address of the targeted Kafka broker (the port is 9092, a well-known one) and the topic to publish. In the case of the RAV, these two parameters are well-known beforehand, but in the case of the Data shippers for infrastructure and application metrics, this information must be provided in the Day-2 configuration process performed by the Runtime Configurator, as presented in the whole section 3.5. Regarding the Kafka IP address to be used by each Data shipper, it will be the one related to the site in which that Data shipper is operating, then the data is replicated to the DCM as commented before.

Having said that, the proposed JSON string for the information model, with the explanation of each field to be included, is the following:

```
{
    'records': [{
        'value': {
            '[metric_value|kpi_value]': <value captured from the execution>,
            'timestamp': <time in which value has been captured, in Unix epoch format>,
            'unit': <unit used in metric_value>,
            'device_id': [ID of the device, to be used in upper layers if needed. Set to ‘nil’ if
not needed],
            'context': [param1=value1 param2=value2 ...] <include here other information that may
be needed in upper layers – customized parameter. Set to ‘nil’ if not needed>
        }
    }]
}
```

In this case, there are some mandatory parameters, which are the *metric_value* captured for the specific metric, with its *unit*, and the related *timestamp*. It can be also included the *device_id* that identifies the device in which it has been captured, to be used in upper layers if needed, and in case of needing to publish more information, it can be included in the optional *context* field. In the case of KPIs, the only modification to be considered is to change *metric_value* for *kpi_value*, as presented above.

The reason of using the *records* and *value* fields is to be compatible with the format used for the provision of the topics from the ELM to the DCM. Also, in this way, a message batch can be sent, containing several, consecutive metrics’ values of the same metric, if required.

The way of publishing this JSON depends on the implementation followed in the Kafka publisher; for example, using Kafka libraries in well-known programming languages such as Python, Java, Go, etc., or by using specific tools such as Beats[12].

Specifically, in the 5G EVE context, it is already defined a Day-2 configuration process that allows users to install Filebeat[13] in specific VNFs and to configure them in a standardised way, as proposed in the Runtime Configuration description in section 3.5. In case of following that approach, the idea is to monitor, with Filebeat, a CSV file, where each row represents one value of a monitored metric, with all the fields commented before. The configuration included in that Filebeat module allows it to transform each new row in that CSV file in JSON automatically, following the information model presented before, and to send the final JSON string automatically to Kafka. Then, the information model to be followed to build each CSV row will be simplified in this way:

[12] https://www.elastic.co/beats/

[13] https://www.elastic.co/beats/filebeat

```
<metric_value>,<timestamp>,<unit>,[<device_id>],[<context>]
```

In this case, however, the batch option is not possible. Note that, again, *device_id* and *context* are optional fields, which must be set to 'nil' if they are not used. In the case of the *context* field, it must contain the values in the following format: *param1=value1 param2=value2*, etc. (i.e. param=value items separated by spaces).

This process of configuring a Data Shipper based on Filebeat and monitoring CSV files will be explained with more detail in the next D3.5.

3.5 Runtime Configurator

The Runtime Configurator (RC) is the component in charge of supporting the experiment configuration, execution and termination through the interaction with the Experiment Execution Manager, handling the commands to be executed in the different phases of an experiment, differentiating between the Day-2 configuration of the targeted servers, both VNFs/PNFs or infrastructure components, and the experiment execution itself, executing the commands related to each step of the experiment in the corresponding servers.

3.5.1 Software architecture

The proposed architecture for the RC in D3.3 was based, mainly, on Ansible Core[14], using SSH in the interaction between the EEM and the RC as a first approach, but with the intention of evolving it towards a RESTful interaction by integrating the AWX project[15] in the RC. However, this approach was not enough for fulfilling the requirements imposed to the RC due to the evolution of the 5G EVE end-to-end workflow, related to the automation of the whole workflow.

For this reason, the Runtime Configurator software architecture has evolved to the one presented in the following figure, including other internal components to achieve the full automation in the operations executed between the EEM and the RC:

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

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

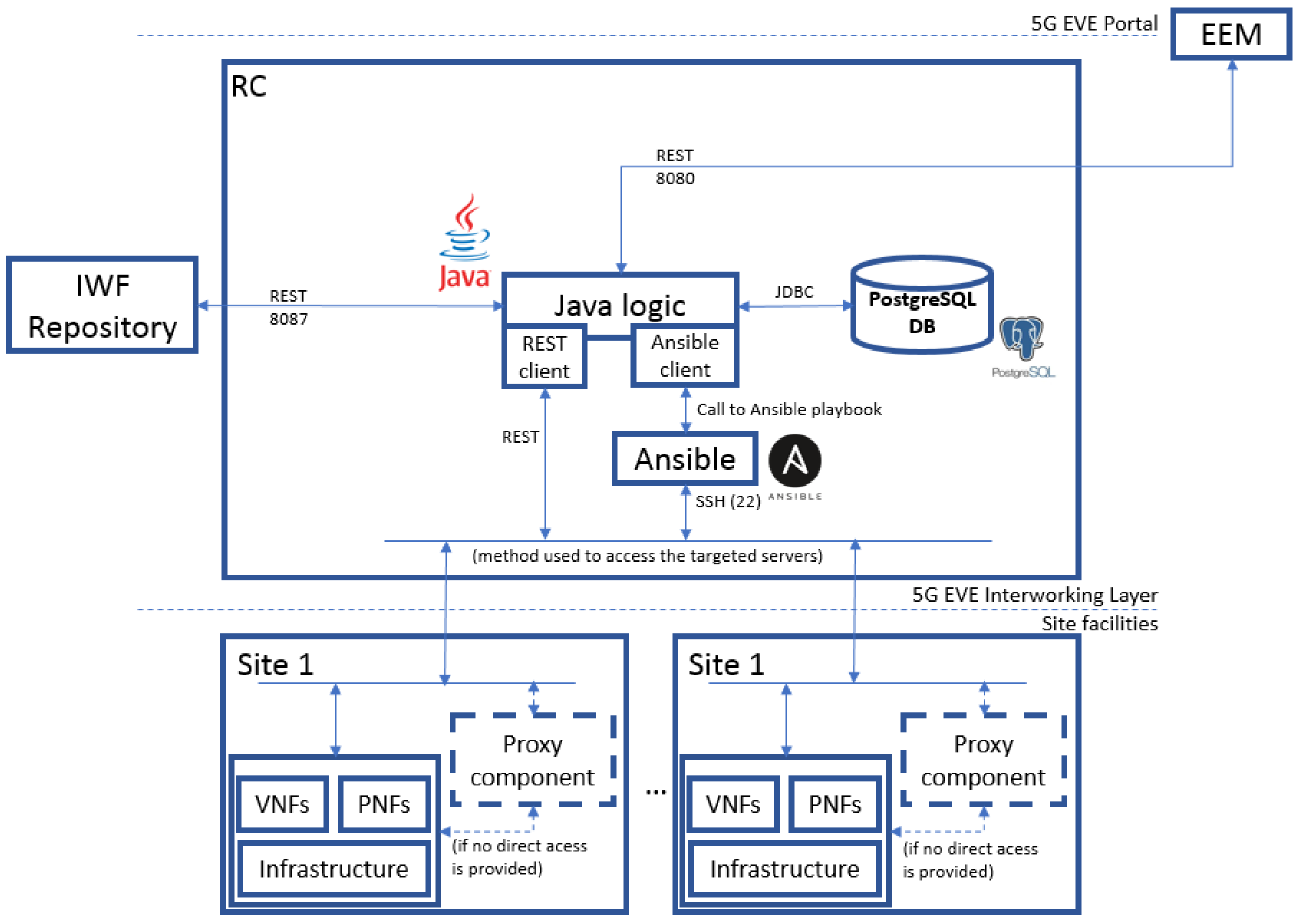

Figure 20: Runtime Configurator final architecture

Now, all the interactions between the EEM and the RC are managed by a Java logic running within the RC server, based on Spring Framework, playing the role of the AXW project aforementioned, which will not be installed in the RC then. This Java logic can be found here: https://github.com/5GEVE/5geve-wp3-rc-handler.

This module also handles the interaction with the IWF Repository, which will provide all the scripts related to the configuration of infrastructure metrics, and with the targeted servers to be configured through specific clients, counting with an Ansible client (which interacts with Ansible) and also with a REST client (in case a specific server exposes a REST API to be accessed, instead of SSH). All the data handled by the Java logic is saved in an internal PostgreSQL database, according to the data model presented in the following figure.

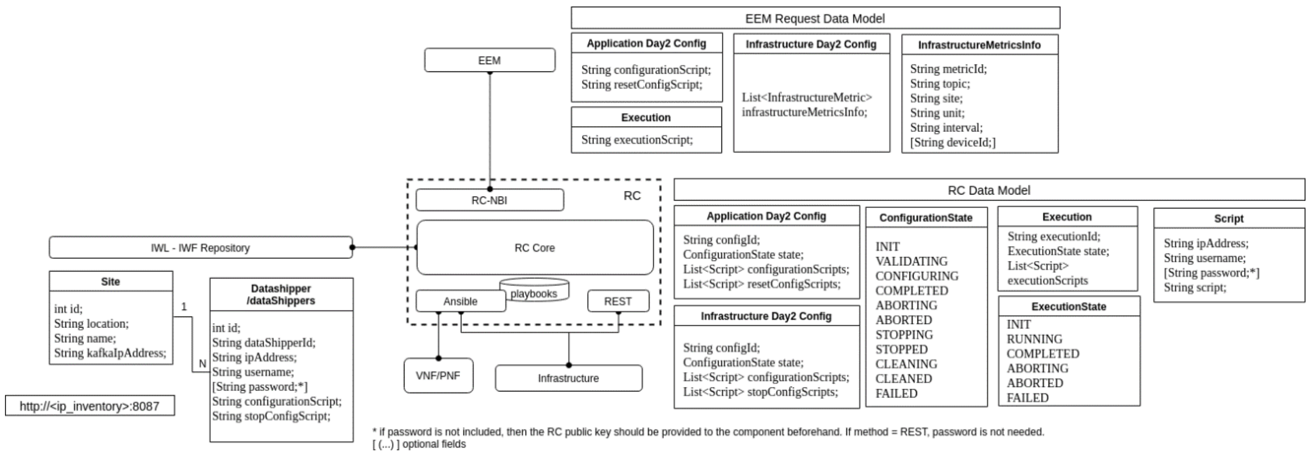

Figure 21: Runtime Configurator data model

Basically, there are three types of request that are handled between the EEM and the RC:

- Application Day-2 configuration: it handles the provisioning and execution of the scripts related to the initialization and resetting of the configuration of applications.
- Infrastructure Day-2 configuration: it is directly received the information related to the different infrastructure metrics to be monitored in the experiment, according to its definition in the corresponding Experiment Blueprints (ExpB), and also including some other fields like the topic or the deviceId, which are provided to the EEM by the ELM. Then, through this interface, the start and stop operations for the monitoring of each infrastructure metric are handled. To obtain the scripts for these operations, a previous interaction between the RC and the IWF Repository is needed to extract the scripts according to the requested Data Shipper, whose identifier is based on the information provided in the EEM request.
- Execution: in the same way that the application Day-2 configuration, it handles the provisioning and execution of all the scripts related to the experiment execution of the particular test case.

What it is saved in the RC internal database is all these scripts, also providing a unique identifier for each provisioning operation requested by the EEM and the current status of the operation. This will be further extended in chapter 3.5.3.

Regarding the access to the targeted servers, the main alternative offered by the RC is still the OpenSSH protocol by using specific Ansible playbooks, which are triggered by the Java logic with the information provided by the EEM (and the IWF Repository in the case of infrastructure metrics). However, as commented before, a REST client is also provided for particular cases where the interaction between the RC and the targeted server must be based on RESTful operations. For obtaining the IP addresses, they will be provided by the EEM, which would previously obtain these IP addresses from the MSNO, according to the references included in the Test Cases Blueprints (TCBs), except infrastructure components, whose access configuration is fairly fixed and declared in the IWF Repository. Apart from that, the considerations related to the Proxy Component already mentioned in D3.3 are still present in this scenario.

Finally, in the same way that the DCM, the RC is thought to be also deployed in a single Linux server, physical or virtual (VM). The current Linux distribution used to deploy the RC is Ubuntu Server 16.04 LTS, and the deployment steps can be found in the following repository: https://github.com/5GEVE/5geve-wp3-rc-deployment. There, a specific Ansible playbook provides all the commands, files and configurations needed to setup a RC instance from scratch or to update it to the latest version.

3.5.2 Open API description

The OpenAPI specification exposed by the Java logic in the RC can be found in the following link: https://github.com/5GEVE/OpenAPI/tree/master/RuntimeConfigurator. This specification fully replaces the AWX's OpenAPI definition presented in deliverable D3.3.

As a result, the high-level operations provided by the RC need to be updated from what it was reported in D3.3 – Annex B, according to the previous OpenAPI definition. These are collected in tables below.

Table 5: Runtime Configurator updated NBI

Runtime Configurator NBI		
Application Day-2 Configuration		
Description	Operations related to start or reset the application Day-2 configuration in the targeted VNFs or PNFs.	
Reference Standards	REST API interface.	
Operations Exposed	***Information Exchanged***	***Information Model***

Load commands	• Configuration scripts. • Reset configuration scripts. It will return the configId to be used in further operations and also the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Get status of an operation	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Start configuration	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Reset configuration	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Abort configuration	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Infrastructure Day-2 Configuration		
Description	Operations related to start or stop the configuration of infrastructure components.	
Reference Standards	REST API interface.	
Operations Exposed	***Information Exchanged***	***Information Model***
Load infrastructure information	• List<information related to each infrastructure metric> It will return the configId to be used in further operations and also the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Get status of an operation	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Start configuration	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.

Stop configuration	• configId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Experiment Execution		
Description	Operations related to the execution of commands during the Experiment Execution phase.	
Reference Standards	REST API interface.	
Operations Exposed	***Information Exchanged***	***Information Model***
Load commands	• Commands to be executed. It will return the execId to be used in further operations and also the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Get status of an operation	• execId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Start execution	• execId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.
Abort execution	• execId It will return the current status of the operation.	See the Runtime Configurator data model presented in section 3.5.1, the OpenAPI specification and the service description presented in section 3.5.3.

Table 6: Runtime Configurator updated SBI

Runtime Configurator SBI		
Execution of commands through SSH[16]		
Description	Execution of the commands for all the operations exposed in the NBI in the targeted servers.	
Reference Standards	Ansible (OpenSSH protocol).	
Operations Exposed	***Information Exchanged***	***Information Model***

[16] Note that the REST SBI interface is not exposed, as this interface is only used in specific component from specific use cases that really need that way of interacting.

Execute command	• Data to access the targeted servers (IP address, credentials, etc.). • A list with the commands to be executed in the servers. It will return the result of the commands execution.	Specific command to be executed in the server, managed by Ansible playbooks.

With this information, the detailed operations exposed by the RC to the EEM, according to its OpenAPI specification, can be checked in the following table:

Table 7: Runtime Configurator operations from its Open API specification.

Service	Path	Method	Input	Output[17]	Description
Runtime Configurator	/rc/nbi	GET	-	200 - OK	Get RC version and check that the service is running.
Runtime Configurator	/rc/nbi/application/day2/configuration	POST	configurationScript = scripts executed during start application Day-2 configuration operation. resetConfigScript = scripts executed during reset application Day-2 configuration operation.	200 - OK	Provide the scripts for application Day-2 configuration to the RC. It returns the configId and the current status of the operation.
Runtime Configurator	/rc/nbi/application/day2/configuration/<configId>	GET	configId	200 - OK	It returns the current status of the operation.
Runtime Configurator	/rc/nbi/application/day2/configuration/<configId>/start	POST	configId	200 - OK	Execute the configurationScript commands. It returns the current status of the operation.
Runtime Configurator	/rc/nbi/application/day2/configuration/<configId>/reset	POST	configId	200 - OK	Execute the resetConfigScript commands. It returns the current status of the operation.
Runtime Configurator	/rc/nbi/application/day2/configuration/<configId>/abort	DELETE	configId	200 - OK	Abort the current operation. It returns the current status of the operation.

[17] The output is always 200 OK, but the operation status is managed thanks to the operation status attribute which is always returned in all requests.

Runtime Configurator	/rc/nbi/infrastructure/day2/configuration	POST	List<infrastructureMetricsInfo> = list containing the data related to each infrastructure metric to be configured.	200 - OK	Provide the information related to each infrastructure metric for the Day-2 configuration process. It returns the configId and the current status of the operation.
Runtime Configurator	/rc/nbi/infrastructure/day2/configuration/<configId>	GET	configId	200 - OK	It returns the current status of the operation.
Runtime Configurator	/rc/nbi/infrastructure/day2/configuration/<configId>/start	POST	configId	200 - OK	Execute the configurationScript commands (extracted from the IWF Repository by the RC). It returns the current status of the operation.
Runtime Configurator	/rc/nbi/infrastructure/day2/configuration/<configId>/stop	POST	configId	200 - OK	Execute the stopConfigScript commands (extracted from the IWF Repository by the RC). It returns the current status of the operation.
Runtime Configurator	/rc/nbi/execution	POST	execScript = commands to be executed during Experiment Execution phase.	200 - OK	Provide the scripts for experiment execution to the RC. It return the execId and the current status of the operation.
Runtime Configurator	/rc/nbi/execution/<execId>	GET	execId	200 - OK	It returns the current status of the operation.
Runtime Configurator	/rc/nbi/execution/<execId>/start	POST	execId	200 - OK	Execute the execScript commands. It returns the current status of the operation.
Runtime Configurator	/rc/nbi/execution/<execId>/abort	DELETE	execId	200 - OK	Abort the current operation. It returns the current status of the operation.

3.5.3 Service description

In order to describe the service offered by the RC, it will be firstly explained the current information model used to build the scripts to be placed in both the Test Cases Blueprints and the IWF Repository. Then, it will be introduced the updated specific-purpose workflows followed in a normal execution of test cases that have both infrastructure and application Day-2 configuration and experiment execution commands.

3.5.3.1 Information model to build the scripts

To achieve the execution of the different commands through Ansible playbooks, in this first full definition of the Runtime Configurator, these commands must be provided in a specific format in both the TCBs (for application configuration and experiment execution) and in the IWF Repository (for infrastructure metrics). Below, it is presented a template to be used in the TCBs for the definition of the commands:

```
testCaseBlueprint:
  description: Example for integration with the RC
  name: Example TCB
  configurationScript: [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=<VNF_IP_REFERENCE> ansible_user=$$user ansible_ssh_pass=$$password
ansible_become_pass=$$password\" | tee -a hosts && export ANSIBLE_HOST_KEY_CHECKING=False &&
ansible-playbook -i hosts execute_script.yml -e 'script=\"<SCRIPT_OR_COMMAND_TO_BE_EXECUTED>\"';
  executionScript: sleep $$sleep_time;
  resetConfigScript: (...)
  userParameters:
    user: $$user
    password: $$password
    sleep_time: $$sleep_time
    (...)
  infrastructureParameters:
    <VNF_IP_REFERENCE>
    (...)

  version: '2.0'
```

In this particular case, note that the commands always follow the same structure:

- They begin with the "reserved words" *`[ ! -e hosts ] || rm hosts && touch hosts &&,`* which removes the hosts file managed by Ansible, in which it is contained the information to access to the targeted servers. Then, this hosts file is always clean in each command execution.
- Then, it follows by *`echo \"server ansible_host=<VNF_IP_REFERENCE> ansible_user=$$user ansible_ssh_pass=$$password ansible_become_pass=$$password\" | tee -a hosts &&,`* command that creates the hosts file with particular data referenced in the *userParameters* field in the TCB. By this way, Ansible can know the information to access to the server(s), by knowing the IP address, username and password. If password is not provided, it is assumed that the RC has provided their public keys to the targeted servers in an off-line process in order to access them without requiring authentication.
- Again, the previous set of commands are followed by some "reserved words" which are *`export ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e,`* which prevents Ansible from checking host key verification and calls to the execute_script.yml playbook with the information included in the hosts file already created.
- Finally, the execute_script.yml playbook accepts one single parameter, which is the script to be executed: *`'script=\"<SCRIPT_OR_COMMAND_TO_BE_EXECUTED>\"';`* in which it must be included the command to be executed in the targeted server as if it were running in a single terminal.

Moreover, some observations must be taken into consideration:

- Several commands can be concatenated in a single "script" field by separating them with a ";". But, in case of needing to include several commands concatenated in the *`<SCRIPT_OR_COMMAND_TO_BE_EXECUTED>`* field, they must be separated with "&&", as the RC logic uses the ";" to separate the commands in order to execute them sequentially.
- Note that, in the *executionScript* field of this template, it is provided the following command: `sleep $sleep_time;` which is used to wait a certain amount of time in case of needing to do some actions off-line within the scope of the experiment. This command does not need to include all the reserved words and structure aforementioned.
- Currently, execute_script.yml is the only playbook that can be used in this first RC implementation; but, in the near future, other playbooks will be also available to be used. These will be reported in the next D3.5. Some of these playbooks that have been already developed (but not integrated in this logic),

apart from the execute_script.yml one, are available in the following Github repository: https://github.com/5GEVE/5geve-rc.

- The commands included in the IWF Repository to start and stop the infrastructure components must follow that format for either the execution of single commands or the sleep operation.

An example of a TCB following this approach can be checked in Annex A. Note that it includes, as *infrastructureParameters*, some parameters related to application metrics that may be needed to properly configure the Data Shippers related to them, e.g. making a reference to the topic that the Data Shipper must use to publish the data, which is injected by the EEM after receiving that information from the ELM.

However, this way of building the scripts in the TCB and in the IWF Repository are not definitive, as they are not efficient because they need to include several reserved words that are useless to the experimenters. In that way, a new information model to define these scripts in a more elegant way is under evaluation and will be reported with detail in the next D3.5 with all the instructions needed to correctly make use of the different functionalities offered.

3.5.3.2 Updated specific-purpose workflows

The workflows related to the operation of the RC have completely changed due to the modifications in the architecture. All the steps related to the RC workflow are always triggered by the Experiment Execution Manager, following this approach:

- First of all, the EEM provides the information related to the corresponding operation (i.e. application configuration, infrastructure operation and experiment execution) to the RC, which automatically generates an identifier to be used by the EEM in subsequent requests.
- When the EEM is about to trigger a script execution in whatever type of operation (start, stop, reset, etc.), it uses the identifier aforementioned to reference the scripts that have to be executed, and then the RC starts executing them asynchronously. Thanks to that, the EEM does not need to wait until the scripts have been finished. In order to know it, the EEM performs a polling to the RC, checking the status of the operation (again, using the identifier) until it changes to the expected status.

Then, the complete workflow in a full experiment execution with all the possible operations present is the following:

1. First of all, there are some preliminary interactions between several components implied in the E2E workflow, e.g. the provision of the topics from the ELM to the DCM or the proper configuration of the EEM based on the information provided by the ELM.

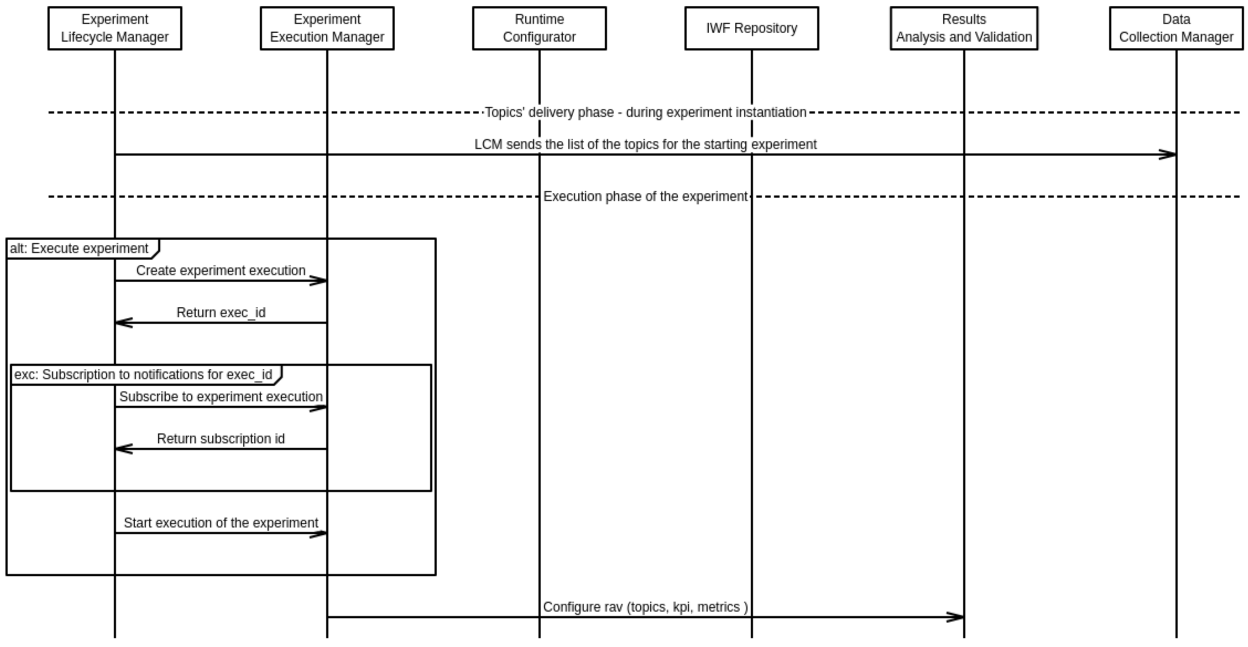
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Results Analysis and Validation
Data Collection Manager
Topics' delivery phase - during experiment instantiation
LCM sends the list of the topics for the starting experiment
Execution phase of the experiment
alt: Execute experiment
Create experiment execution
Return exec_id
exc: Subscription to notifications for exec_id
Subscribe to experiment execution
Return subscription id
Start execution of the experiment
Configure rav (topics, kpi, metrics)

Figure 22: Preliminary steps before starting with the experiment execution

2. Then, the first scripts that are requested to be executed by the EEM are the ones related to the applications' configuration in the so-called application Day-2 configuration phase. But first, all the scripts related to the application configuration (i.e. the configuration scripts and the reset configuration scripts) are loaded in the RC by the EEM, as commented before. Then, after doing this, the start operation is triggered by the EEM, which performs a polling process until the configuration has finished (status changes to COMPLETED). During that time, the configuration scripts are executed in the targeted VNFs and PNFs by using specific Ansible playbooks. Note that the commands executed in this step are the ones included in the *configurationScript* field in the TCBs.

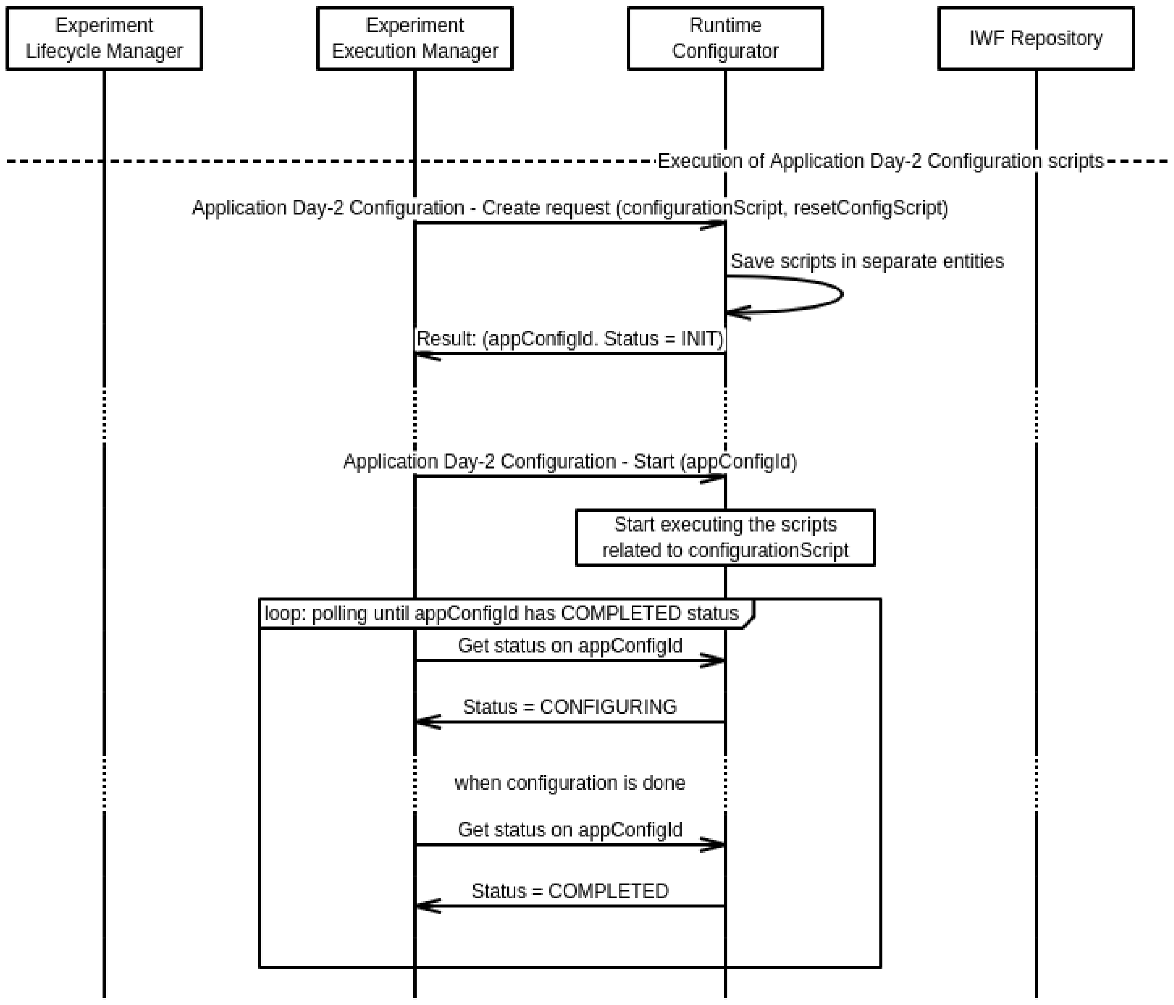
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Execution of Application Day-2 Configuration scripts
Application Day-2 Configuration - Create request (configurationScript, resetConfigScript)
Save scripts in separate entities
Result: (appConfigId. Status = INIT)
Application Day-2 Configuration - Start (appConfigId)
Start executing the scripts related to configurationScript
loop: polling until appConfigId has COMPLETED status
Get status on appConfigId
Status = CONFIGURING
when configuration is done
Get status on appConfigId
Status = COMPLETED

Figure 23: Execution of application Day-2 configuration scripts

3. When the application Day-2 configuration process is finished, then starts the infrastructure Day-2 configuration process, following the same approach, but instead of provisioning directly the scripts, the EEM provides the information related to each infrastructure metric. As already commented, this information can be used to make specific requests to the IWF Repository to extract the scripts and the information to successfully connect to the infrastructure components. After this, the workflow is the same: when the EEM triggers the start operation, the commands are executed and the EEM, in the meanwhile, remains doing a polling until the status changes to COMPLETED. Note that the commands executed in this step are the ones included in the *configurationScript* field for the selected Data Shipper in the IWL Repository.

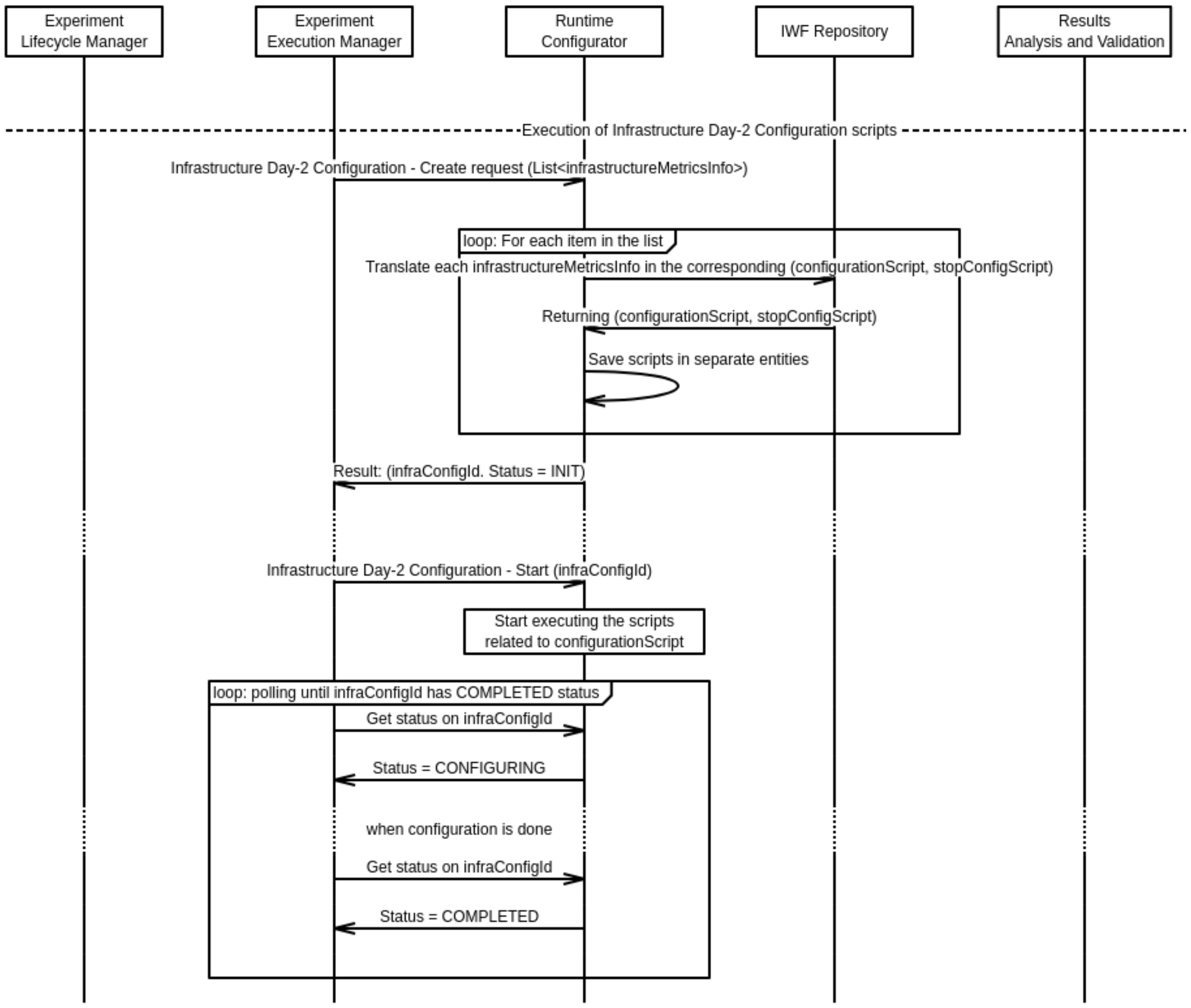
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Results Analysis and Validation
Execution of Infrastructure Day-2 Configuration scripts
Infrastructure Day-2 Configuration - Create request (List<infrastructureMetricsInfo>)
loop: For each item in the list
Translate each infrastructureMetricsInfo in the corresponding (configurationScript, stopConfigScript)
Returning (configurationScript, stopConfigScript)
Save scripts in separate entities
Result: (infraConfigId. Status = INIT)
Infrastructure Day-2 Configuration - Start (infraConfigId)
Start executing the scripts related to configurationScript
loop: polling until infraConfigId has COMPLETED status
Get status on infraConfigId
Status = CONFIGURING
when configuration is done
Get status on infraConfigId
Status = COMPLETED

Figure 24: Execution of infrastructure Day-2 configuration scripts

4. When the infrastructure Day-2 configuration process is completed, it starts the execution of the test cases, i.e. the experiment execution scripts. Again, the same process presented before is repeated: commands are loaded by the EEM and when the start operation is triggered, they are executed in the targeted server, and the EEM remains in the polling process until the status changes to COMPLETED. In the workflow presented, there are also some interactions with the RAV that are out of the scope of the RC operation. Note that the commands executed in this step are the ones included in the *executionScript* field in the TCBs.

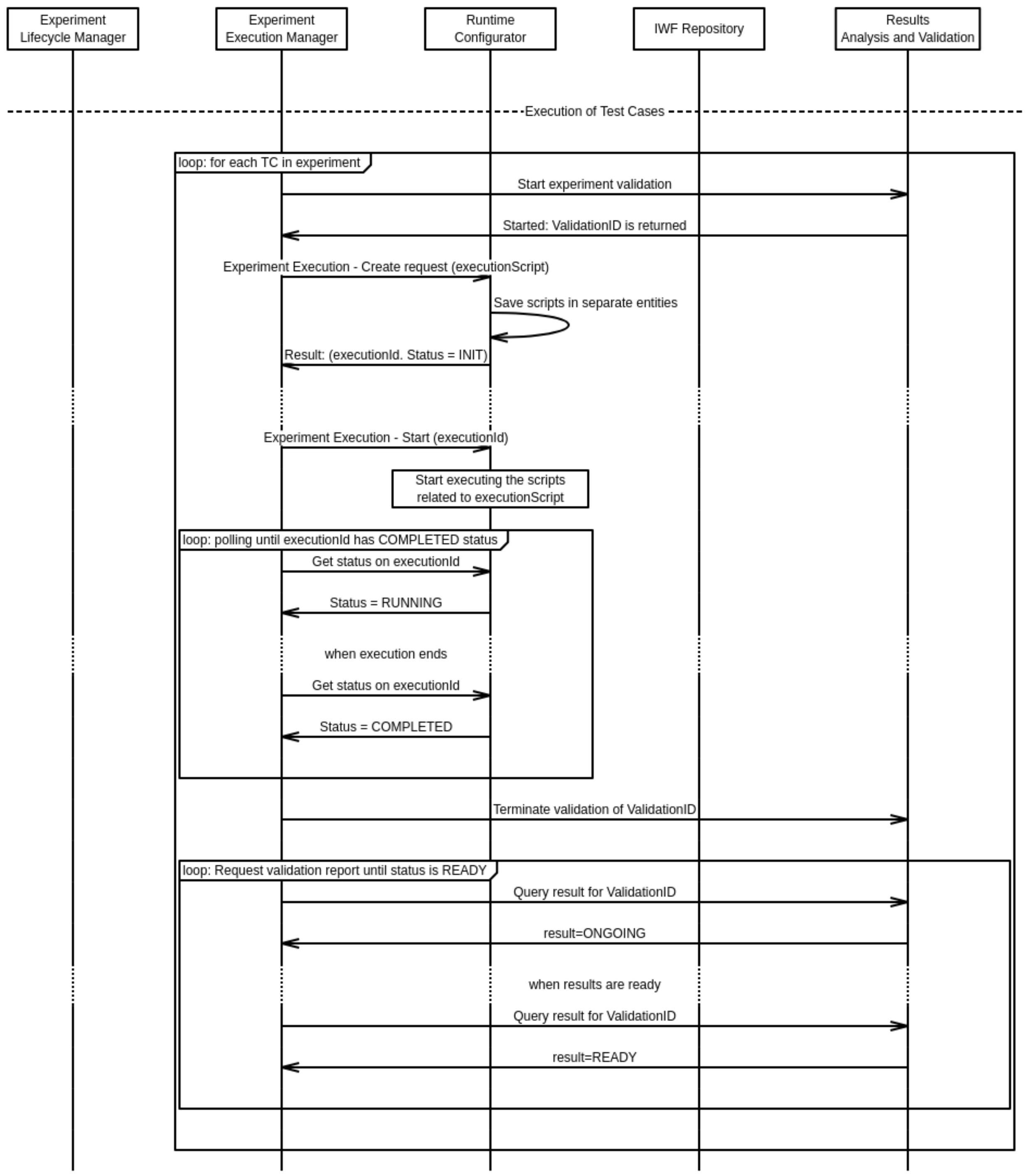
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Results Analysis and Validation
Execution of Test Cases
loop: for each TC in experiment
Start experiment validation
Started: ValidationID is returned
Experiment Execution - Create request (executionScript)
Save scripts in separate entities
Result: (executionId. Status = INIT)
Experiment Execution - Start (executionId)
Start executing the scripts related to executionScript
loop: polling until executionId has COMPLETED status
Get status on executionId
Status = RUNNING
when execution ends
Get status on executionId
Status = COMPLETED
Terminate validation of ValidationID
loop: Request validation report until status is READY
Query result for ValidationID
result=ONGOING
when results are ready
Query result for ValidationID
result=READY

Figure 25: Execution of experiment scripts

5. When the experiment execution phase terminates, it is time to stop the infrastructure scripts by triggering the stop operation in the EEM. Then, the RC executes the scripts related to the stop operation, following the same approach explained in the previous phases. Now, the expected status to complete this phase is STOPPED. Note that the commands executed in this step are the ones included in the *stopConfigScript* field for the selected Data Shipper in the IWL Repository.

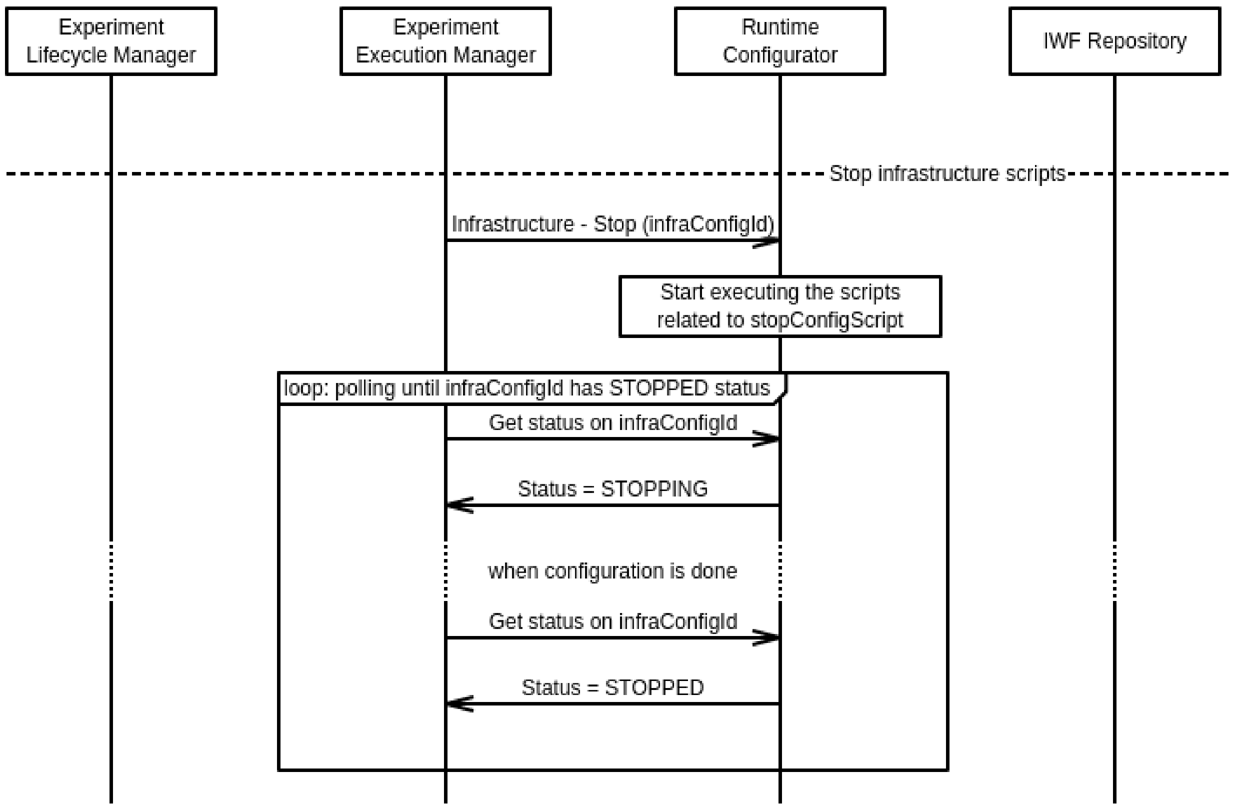
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Stop infrastructure scripts
Infrastructure - Stop (infraConfigId)
Start executing the scripts related to stopConfigScript
loop: polling until infraConfigId has STOPPED status
Get status on infraConfigId
Status = STOPPING
when configuration is done
Get status on infraConfigId
Status = STOPPED

Figure 26: Stop infrastructure scripts

6. And finally, it happens the same for the case of the reset application scripts: they are requested by the EEM, they are then executed by the RC and the EEM waits until receiving a CLEANED status in the polling process. Note that the commands executed in this step are the ones included in the *resetConfigScript* field in the TCBs. After this, other resources (related to the RAV, DCM and other related components) are released.

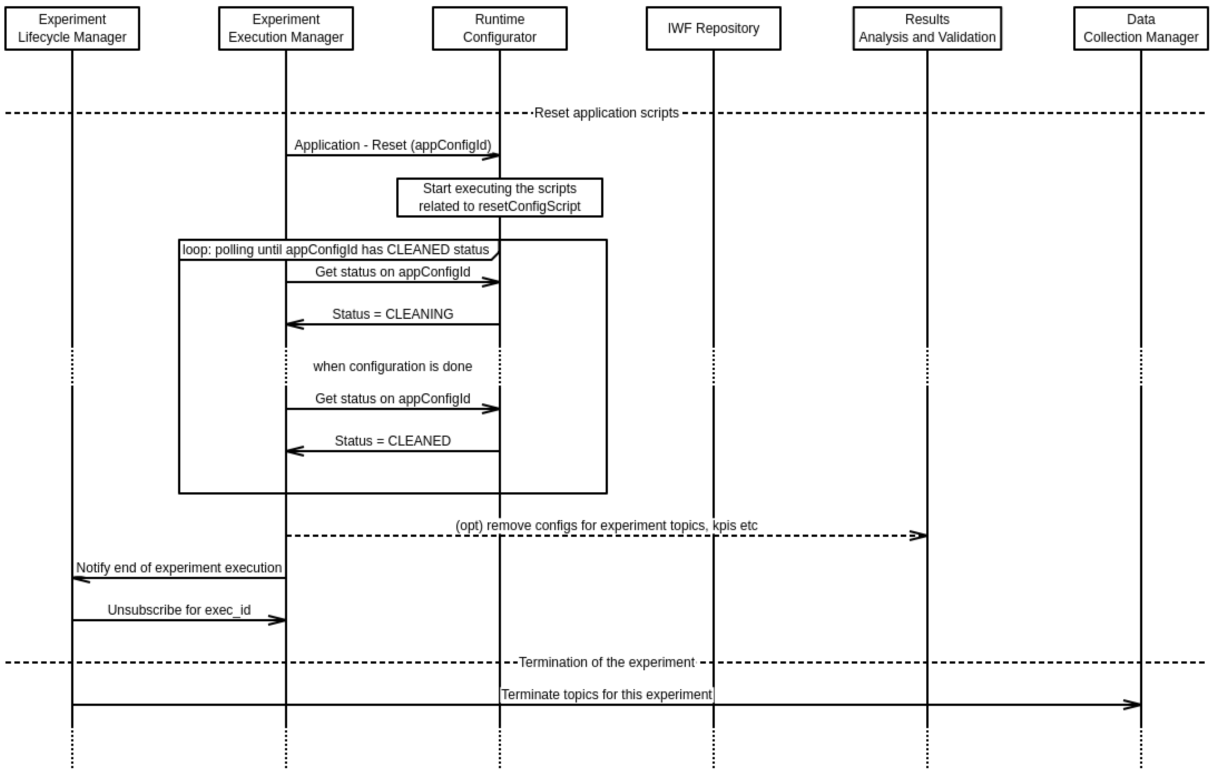
Experiment Lifecycle Manager
Experiment Execution Manager
Runtime Configurator
IWF Repository
Results Analysis and Validation
Data Collection Manager
Reset application scripts
Application - Reset (appConfigId)
Start executing the scripts related to resetConfigScript
loop: polling until appConfigId has CLEANED status
Get status on appConfigId
Status = CLEANING
when configuration is done
Get status on appConfigId
Status = CLEANED
(opt) remove configs for experiment topics, kpis etc
Notify end of experiment execution
Unsubscribe for exec_id
Termination of the experiment
Terminate topics for this experiment

Figure 27: Reset application scripts and termination of the experiment

3.6 Adaptation Layer

The Adaptation Layer at the South-Bound Interface (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 [8] 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.

In 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, Spanish and Greek site facilities (i.e. all of them make use of OSM as site orchestrator). In the current release, the Multi-Site Catalogue includes a driver to integrate the ONAP NFV Orchestrators. More details can be found in Section 3.1.

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:

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

3.6.2.1 Application architecture

The MSO-LO is a web service that exposes an 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 communications between MSO and the MSO-LO backend.

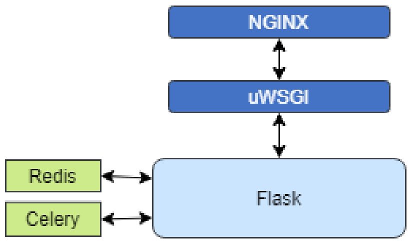

Figure 28: Architecture of the application

As shown in Figure 28, the architecture of the application is composed by the following components:

- Reverse proxy: we use the popular NGINX[18].
- WSGI server: we use uWSGI[19] typically used for running Python web applications.
- Webservice: we use the lightweight micro-framework Flask[20] written in python, that is designed for a quick and easy "getting started", with the ability to scale up to complex applications.
- In-memory data store: we use Redis[21], an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker.
- Periodic and background Tasks manager: we use Celery[22] , an open source asynchronous task queue or job queue which is based on distributed message passing.

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 through the stack.

Regarding persistent storage, with respect to the previous version of the software described in D3.3, we do not use the storage solution based on SQLite in the production environment but, as described in Section 3.7, we use the IWF Repository component. Furthermore, we use a Redis instance as an-in-memory storage database to

[18] https://www.nginx.com/

[19] https://uwsgi-docs.readthedocs.io/en/latest/

[20] https://flask.palletsprojects.com/en/1.1.x/

[21] https://redis.io/

[22] https://docs.celeryproject.org/en/stable/

support the background task scheduler (celery), the tracking of LCM operation status and shared information between multiple processes of the MSO-LO.

3.6.2.2 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 29

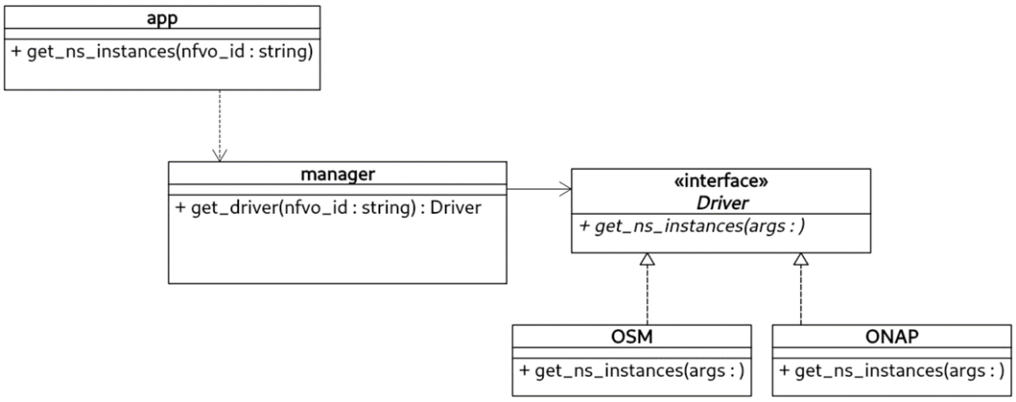
app
+ get_ns_instances(nfvo_id : string)
manager
+ get_driver(nfvo_id : string) : Driver
«interface»
Driver
+ get_ns_instances(args :)
OSM
+ get_ns_instances(args :)
ONAP
+ get_ns_instances(args :)

Figure 29: UML class diagram with an example method of the interface.

Figure 29 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 [13]. The pattern is very effective for our solution as it separates the creation of driver instances from the *app* module that actually uses 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[23]). 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 30 shows the interactions needed to obtain the list of NS instances from the catalogue of an OSM NFVO identified with ‘osm_1’.

[23] https://docs.python.org/3/library/abc.html

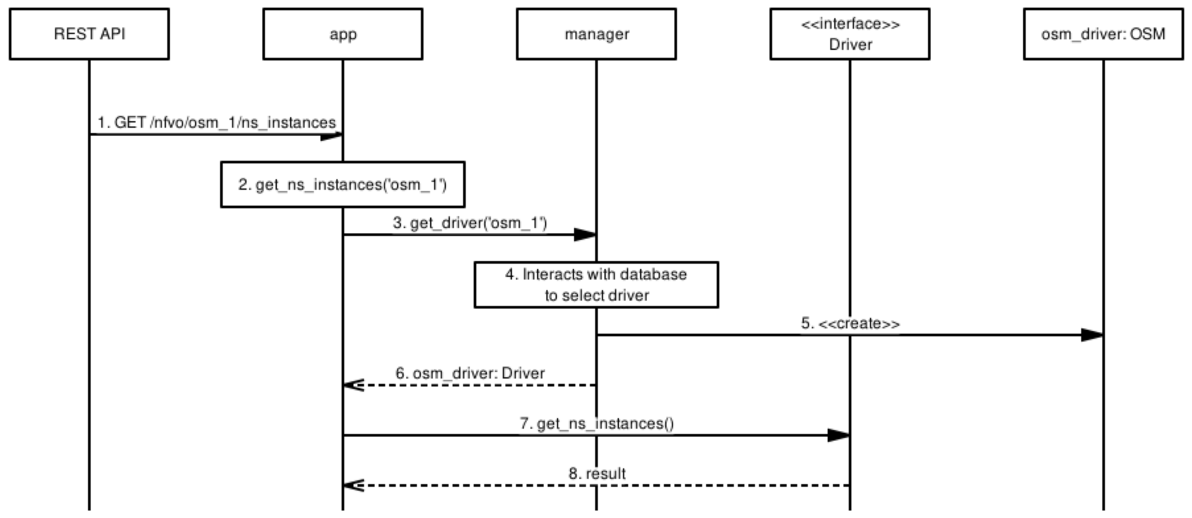

Figure 30: Message sequence chart to show modules and objects interactions.

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.

Regarding the NS Lifecycle Management Notification, the MSO-LO provides an API interface compliant to the ETSI NFV SOL005 specification, supporting three types of notifications:

- NsLcmOperationOccurrenceNotification: Inform subscribed clients about an event in the lifecycle of an NS. The notification includes NS id, OpOcc id, StateType (PROCESSING, COMPLETED, etc.) and affected components.
- NsIdentifierCreationNotification: Informs subscribed clients about NS id creation.
- NsIdentifierDeletionNotification: Informs subscribed clients about NS id deletion.

The solution provides a generic subscription system to allow clients to filter out notifications and receive only the ones about NS instances they are interested about. Subscriptions are stored in the IWF Repository and the MSO-LO northbound interface provides CRUD methods for their management. More details on supported REST operations can be found in Section 3.6.2.4. A crucial part of the subscription information element is the "uri". This element must be a valid HTTP endpoint enabled by the subscribing client to receive notifications. MSO-LO sends notifications via a POST request targeting the aforementioned "uri".

The notifications management is delegated to specific components, responsible for checking the status of operations applied to NS instances for each type of NFVO. To better understand how the specific components works, we use a message sequence chart for each of them to show the interaction of the MSO-LO with the other components of the 5G EVE ecosystem.

Figure 31 shows the use case of the ONAP NFVO located in the French Site. At first step, the MSNO sends a request to instantiate an NS Instance, and the MSO-LO takes care to handle the request forwarding it to the specific ONAP Translation Component. At step 2 the MSNO sends a request in order to create a subscription for a specific NS Instance to the MSO-LO (the one that has been just created) and, if the request is valid, the subscription is stored inside the IWF Repository.

As soon as the ONAP Translation Component has a change of state for an operation to be notified, it sends a POST on the SBI of the MSO-LO, specifically on the endpoint "/nfvo/{nfvoId}/notifications". At this point, the MSO-LO takes care of the notification and checks whether there are any subscriptions related to the NS Instance involved and it forwards the notification on the URI that was defined by the MSNO during the creation of the subscription.

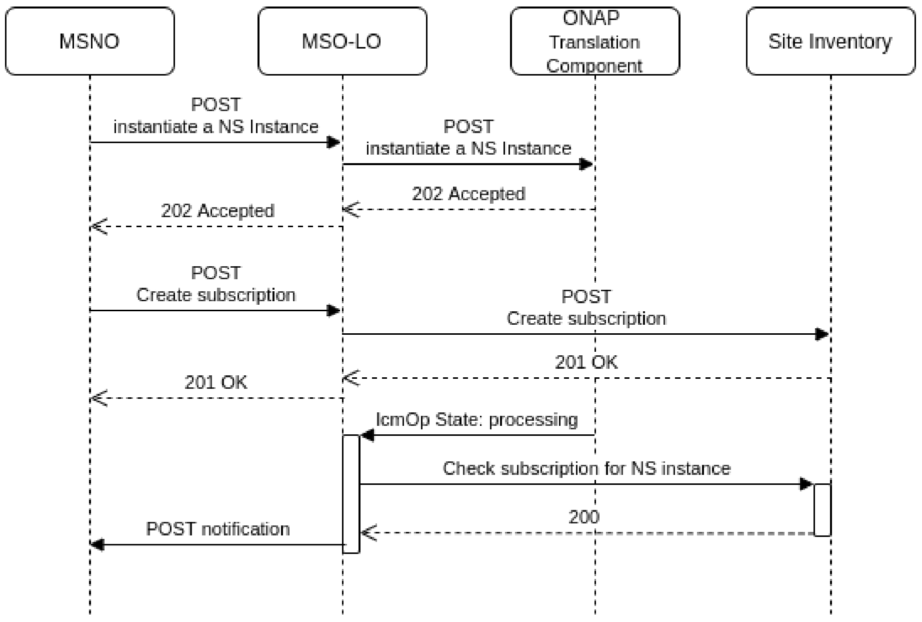

Figure 31: Subscriptions and notifications management for ONAP

Figure 32 shows the use case of the OSM NFVO. The first two steps, the instance creation and the subscription creation, are similar to the use case of the ONAP NFVO described above. Contrary to the previous case, the OSM NFVO is not able to independently notify a change of state happening on one of its managed resources. Then, a background process is activated within the MSO-LO that periodically check for any operations happening in the OSM NFVO. The last state of a specific operation is stored in an in-memory storage implemented with Redis, a key-value database. If a change in the operation's state is detected, the MSO-LO takes care of forwarding the notification to the subscribed client.

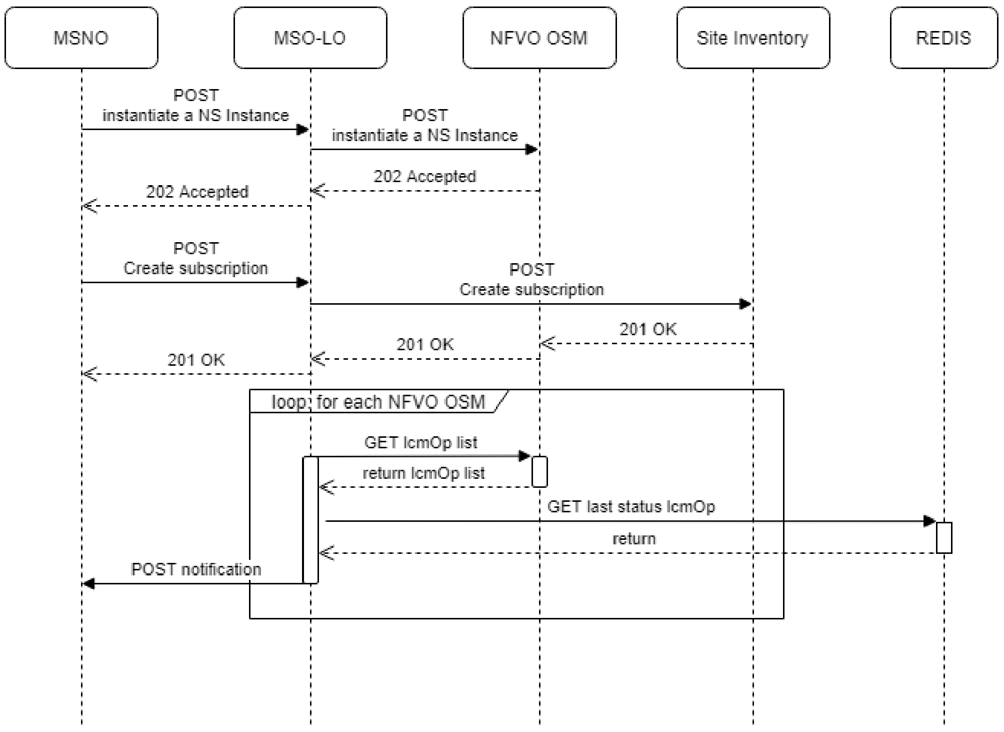

Figure 32: Subscriptions and notifications management for OSM

3.6.2.3 OSM specific features

The sites hosting the OSM NFVO have shown the need to support additional parameters during the instantiation phase. To meet this request the MSO-LO, in accordance with the ETSI NFV-SOL 005 ([5]) standard, supports the "additionalParamsForNs" field in the body of requests to the endpoint "/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate". The information included in this field is only processed by the OSM driver, while other drivers ignore it.

The following instantiation parameters are supported:

- vim network name: this allows the association of a VLD to an existing network in the targeted site, useful when there is a static management network.
- vim account: this allows the MSO to specify in which VIM a particular VNF must be instantiated, useful in case there are multiple VIMs within a site.

3.6.2.4 Open API description

For the implementation we adopt and extend the proposed implementation described in ETSI NFV-SOL 005 ([5]). 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[24], an online tool to describe API adhering to the OpenAPI[25] Specification. The API is available as a YAML file in the 5G EVE public organization hosted on GitHub[26]. Table 8 summarizes the API methods offered by MSO-LO interface.

Table 8: MSO-LO API 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.

[24] https://swagger.io/

[25] https://www.openapis.org/

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

MSO-LO	/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate	POST	nsFlavourId	-	Instantiate a 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/ns_lcm_op_occs	GET	-	Array of NS LCM operation	Query multiple NS LCM operation
MSO-LO	/nfvo/{nfvoId/ns_lcm_op_occs/{nsLcmOpOccId}	GET	nsLcmOpOccId	NS LCM operation	Read an individual NS LCM operation occurrence resource.
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.
MSO-LO	/nfvo/{nfvoId}/notification	POST	nsInstanceId operation operationState	-	Notify a status change of a operation state to MSO-LO

3.7 IWF Repository

The 5G EVE IWF Repository is the component in charge of storing shared information about sites and their features. It acts as a centralized storage element, exposing its data to other components of the Interworking Framework via a REST HTTP interface. The source code can be found in GitHub[27].

3.7.1 Software architecture

The IWF Repository is a REST HTTP Java application using a PostgreSQL database for persistent storage. To enforce best practices and speed up the development process, we use the Spring Framework[28] and the Spring

[27] https://github.com/5GEVE/iwf-repository

[28] https://spring.io/

Boot configuration convention. Figure 33 shows the IWF components that make use of the IWF Repository in their operational workflows.

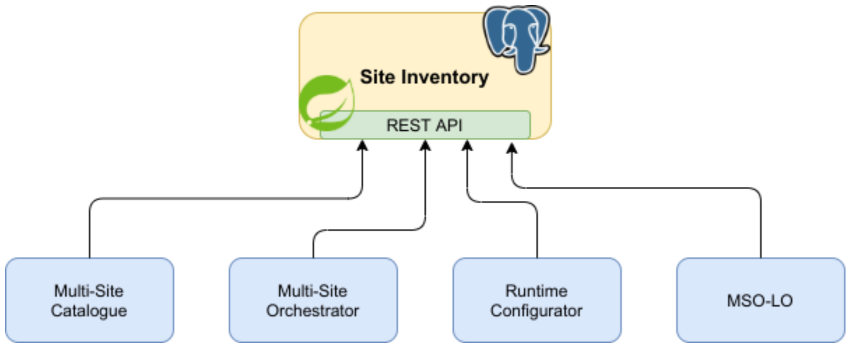

Figure 33: IWL components that use the IWF Repository

Information elements listed in Section 2.1.2 are encoded as Java classes and annotated with '@Entity' following the standard defined by the Java Persistence API. The Spring Framework, thanks to Hibernate[29], uses the annotations to define the data structures in PostgreSQL using the proper Data Definition Language. No direct interaction with the database is required from the developer. The approach made it possible to create a complex database schema in a short time. Figure 34 shows Entity-Relationship diagram of the generated database.

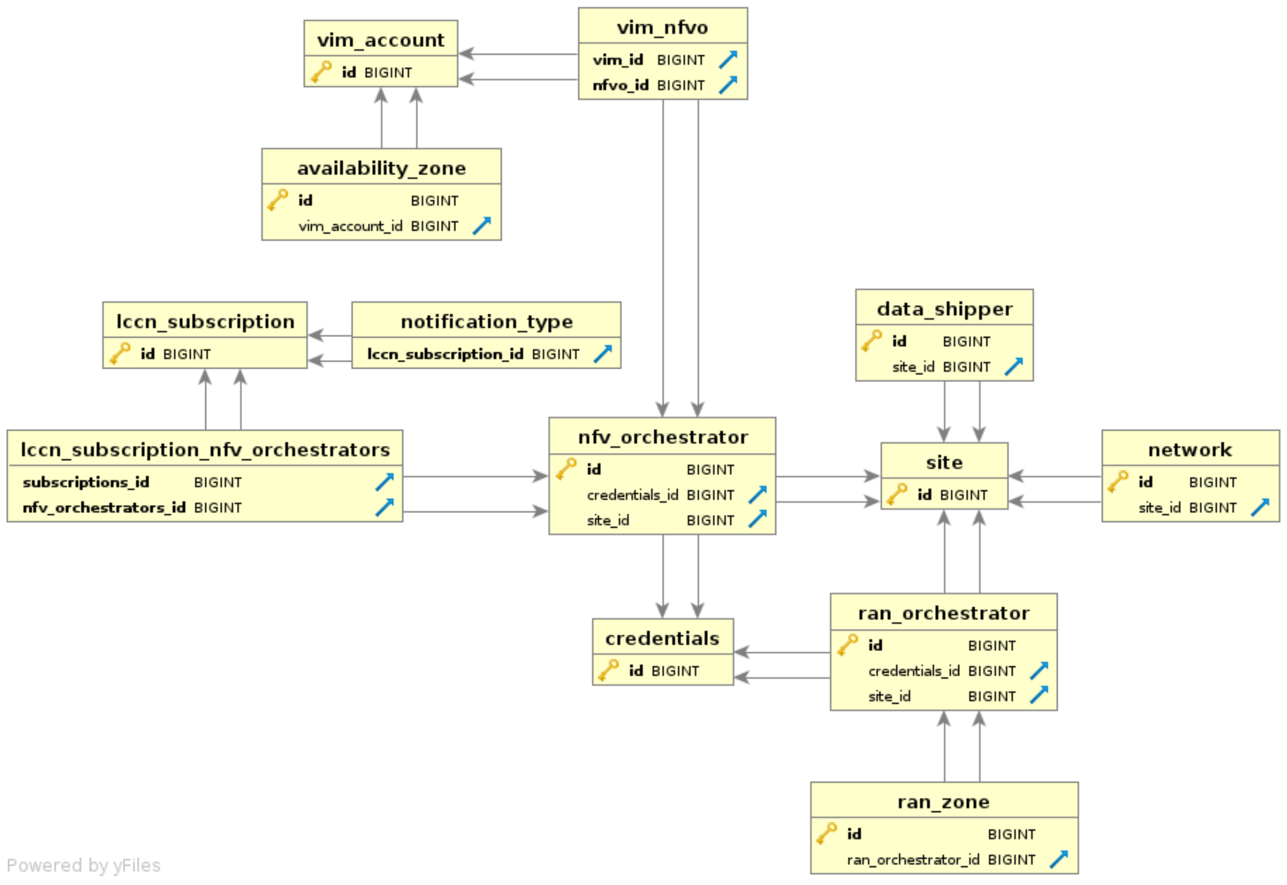

Figure 34: Entity-Relationship diagram for IWF Repository database

To expose the entities and related operation on an HTTP REST interface, we use the module 'spring-data-rest'. This Spring module automatically exposes a discoverable REST APIs by discovering the entities defined in the

[29] https://hibernate.org/

project. The API supports CRUD operations in HTTP format, such as POST, GET, PATCH/PUT, DELETE and generates sub-paths to navigate the associations. The amount of code and development effort to generate all this is extremely limited.

For deployment, we use Dockerfile + docker-compose. The Dockerfile downloads all necessary dependencies and builds the application. The docker-compose creates a multi-service environment including both the IWF Repository application and a PostgreSQL database instance. The application is extremely portable and it can be deployed on any environment with zero configuration.

3.7.2 Open API description

The OpenAPI documentation for IWF Repository interface can be found in GitHub[30]. Since it is auto-generated, the documentation is very verbose.

Table 9: IWF Repository main REST API paths

Service	Path	Method	Input	Output	Description
site-inventory	/availabilityZones	GET	-	Array of Availability Zone entities	Retrieve list of Availability Zones entities.
site-inventory	/availabilityZones	POST	Availability Zone entity	The newly created entity	Create new Availability Zone entity.
site-inventory	/availabilityZones/{id}	GET	-	Availability Zone entity	Retrieve a single Availability Zone entity
site-inventory	/availabilityZones/{id}	PUT	Availability Zone entity with updated values	Availability Zone entity	Update an existing Availability Zone entity.
site-inventory	/availabilityZones/{id}	DELETE	-	-	Delete an existing Availability Zone entity.
site-inventory	/availabilityZones/{id}	PATCH	Fields to be updated	Availability Zone entity	Update an existing Availability Zone entity.
site-inventory	/dataShippers	GET	-	Array of Data Shipper entities	Retrieve list of Data Shipper entities.
site-inventory	/dataShippers	POST	Data Shipper entity	The newly created entity	Create new Data Shipper entity.
site-inventory	/dataShippers/{id}	GET	-	Data Shipper entity	Retrieve a single Data Shipper entity

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

site-inventory	/dataShippers/{id}	PUT	Data Shipper entity with updated values	Data Shipper entity	Update an existing Data Shipper entity.
site-inventory	/dataShippers/{id}	DELETE	-	-	Delete an existing Data Shipper entity.
site-inventory	/dataShippers/{id}	PATCH	Fields to be updated	Data Shipper entity	Update an existing Data Shipper entity.
site-inventory	/subscriptions	GET	-	Array of LccnSubscription entities	Retrieve list of LccnSubscription entities.
site-inventory	/subscriptions	POST	LccnSubscription entity	The newly created entity	Create new LccnSubscription entity.
site-inventory	/subscriptions/{id}	GET	-	LccnSubscription entity	Retrieve a single LccnSubscription entity
site-inventory	/subscriptions/{id}	PUT	LccnSubscription entity with updated values	LccnSubscription entity	Update an existing LccnSubscription entity.
site-inventory	/subscriptions/{id}	DELETE	-	-	Delete an existing LccnSubscription entity.
site-inventory	/subscriptions/{id}	PATCH	Fields to be updated	LccnSubscription entity	Update an existing LccnSubscription entity.
site-inventory	/networks	GET	-	Array of Network entities	Retrieve list of Network entities.
site-inventory	/networks	POST	Network entity	The newly created entity	Create new Network entity.
site-inventory	/networks/{id}	GET	-	Network entity	Retrieve a single Network entity
site-inventory	/networks/{id}	PUT	Network entity with updated values	Network entity	Update an existing Network entity.
site-inventory	/networks/{id}	DELETE	-	-	Delete an existing Network entity.

site-inventory	/networks/{id}	PATCH	Fields to be updated	Network entity	Update an existing Network entity.
site-inventory	/nfvOrchestrators	GET	-	Array of NfvOrchestrator entities	Retrieve list of NfvOrchestrator entities.
site-inventory	/nfvOrchestrators	POST	NfvOrchestrator entity	The newly created entity	Create new NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	GET	-	NfvOrchestrator entity	Retrieve a single NfvOrchestrator entity
site-inventory	/nfvOrchestrators/{id}	PUT	NfvOrchestrator entity with updated values	NfvOrchestrator entity	Update an existing NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	DELETE	-	-	Delete an existing NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	PATCH	Fields to be updated	NfvOrchestrator entity	Update an existing NfvOrchestrator entity.
site-inventory	/ranOrchestrators	GET	-	Array of RanOrchestrator entities	Retrieve list of RanOrchestrator entities.
site-inventory	/ranOrchestrators	POST	RanOrchestrator entity	The newly created entity	Create new RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	GET	-	RanOrchestrator entity	Retrieve a single RanOrchestrator entity
site-inventory	/ranOrchestrators/{id}	PUT	RanOrchestrator entity with updated values	RanOrchestrator entity	Update an existing RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	DELETE	-	-	Delete an existing RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	PATCH	Fields to be updated	RanOrchestrator entity	Update an existing RanOrchestrator entity.

site-inventory	/ranZones	GET	-	Array of RanZone entities	Retrieve list of RanZone entities.
site-inventory	/ranZones	POST	RanZone entity	The newly created entity	Create new RanZone entity.
site-inventory	/ranZones/{id}	GET	-	RanZone entity	Retrieve a single RanZone entity
site-inventory	/ranZones/{id}	PUT	RanZone entity with updated values	RanZone entity	Update an existing RanZone entity.
site-inventory	/ranZones/{id}	DELETE	-	-	Delete an existing RanZone entity.
site-inventory	/ranZones/{id}	PATCH	Fields to be updated	RanZone entity	Update an existing RanZone entity.
site-inventory	/sites	GET	-	Array of Site entities	Retrieve list of Site entities.
site-inventory	/sites	POST	Site entity	The newly created entity	Create new Site entity.
site-inventory	/sites/{id}	GET	-	Site entity	Retrieve a single Site entity
site-inventory	/sites/{id}	PUT	Site entity with updated values	Site entity	Update an existing Site entity.
site-inventory	/sites/{id}	DELETE	-	-	Delete an existing Site entity.
site-inventory	/sites/{id}	PATCH	Fields to be updated	Site entity	Update an existing Site entity.
site-inventory	/vimAccounts	GET	-	Array of Vim Account entities	Retrieve list of Vim Account entities.
site-inventory	/vimAccounts	POST	Vim Account entity	The newly created entity	Create new Vim Account entity.
site-inventory	/vimAccounts/{id}	GET	-	Vim Account entity	Retrieve a single Vim Account entity
site-inventory	/vimAccounts/{id}	PUT	Vim Account entity with updated values	Vim Account entity	Update an existing Vim Account entity.

site-inventory	/vimAccounts/{id}	DELETE	-	-	Delete an existing Vim Account entity.
site-inventory	/vimAccounts/{id}	PATCH	Fields to be updated	Vim Account entity	Update an existing Vim Account entity.

3.7.3 Service description

The IWF Repository is accessed by other components of the Interworking Framework by means of its REST API in their operations workflows. The API access is preferred for a number of reasons with respect to a direct access to a Database Management system. Indeed, the API performs some validation checks on data entities when they are created e.g., `not null` constraints, pattern matching for UUIDs and IP addresses. Furthermore, having a REST API enables to limit the client operations on the data entities (e.g., for read-only entities) and it keeps open the option to add security mechanisms in the future (at the moment, Site-Inventory interface listens on a local subnet with restricted access through a VPN server). Spring offers some well-established modules in order to support the developer in adding security features to its application.

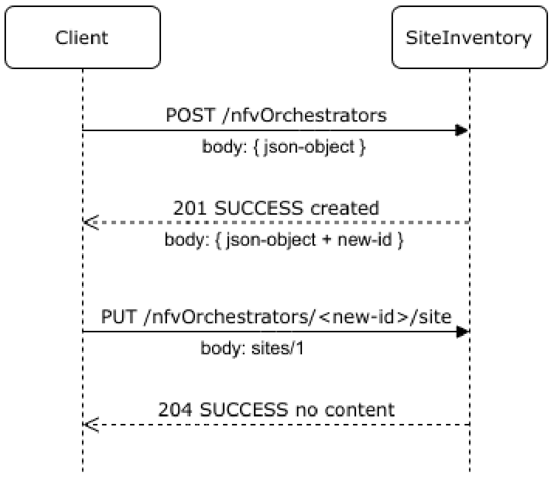

Figure 35: Example HTTP requests to create new entities in IWF Repository

The creation of new entities in the IWF Repository through the REST API must take also an additional step to create the associations. For example, let us consider the creation of a new Nfv Orchestrator entity as showed in Figure 35. We can create the new database record in the proper table by performing a POST request to /nfvOrchestrators (first message in the figure). The IWF Repository returns a 201 status code if this operation is successful (second message in the figure). Now, to associate the newly created entity with a Site entity, an additional PUT request is needed (third message in the figure). The request path, in our example, is '/nfvOrchestrators/{new-id}/site'. The headers must contain a Content Type header with value 'text/uri-list'. Finally, the request body must contain the path of the Site entity we want to associate: /sites/{id-of-desired-site}. If the association has been created successfully, the IWF Repository returns a 204 status code (fourth message in the figure).

3.8 Site adaptations

All the commands executed by Adaptation Layer's components need to be translated to a form understandable by local orchestrators. Each Site in the project requires a special driver that allows communication between the MSO-LO interface endpoints. 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.

3.8.1 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 interconnection with the I/W Framework. It creates a kind of 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 a full range of life-cycle management operations that can be steered by the Runtime Configurator of I/W Framework. To fully integrate ONAP and the I/W Framework, a special ONAP adapter needs to be implemented. The adapter will translate requests made by I/W Framework components to the form understandable by ONAP using the set of ONAP external APIs.

In D3.4 we are providing implementation that allows to integrate ONAP with I/W Framework services, mainly Service Orchestration and Multi-Site Catalogue. The second delivery of the MSO-LO interface (that provides operations for the management of NS and VNF instances) will focus on Publish & Subscribe methods that allows to notify user about services status changes (Simple VNF onboarding and initialization methods has been already implemented in first release - D3.3). The second release of IWL supports as well basic ONAP catalog - IWL catalog synchronization. More details are listed below. Architectural integration of IWL with ONAP as a single point of contact is depicted in next figure.

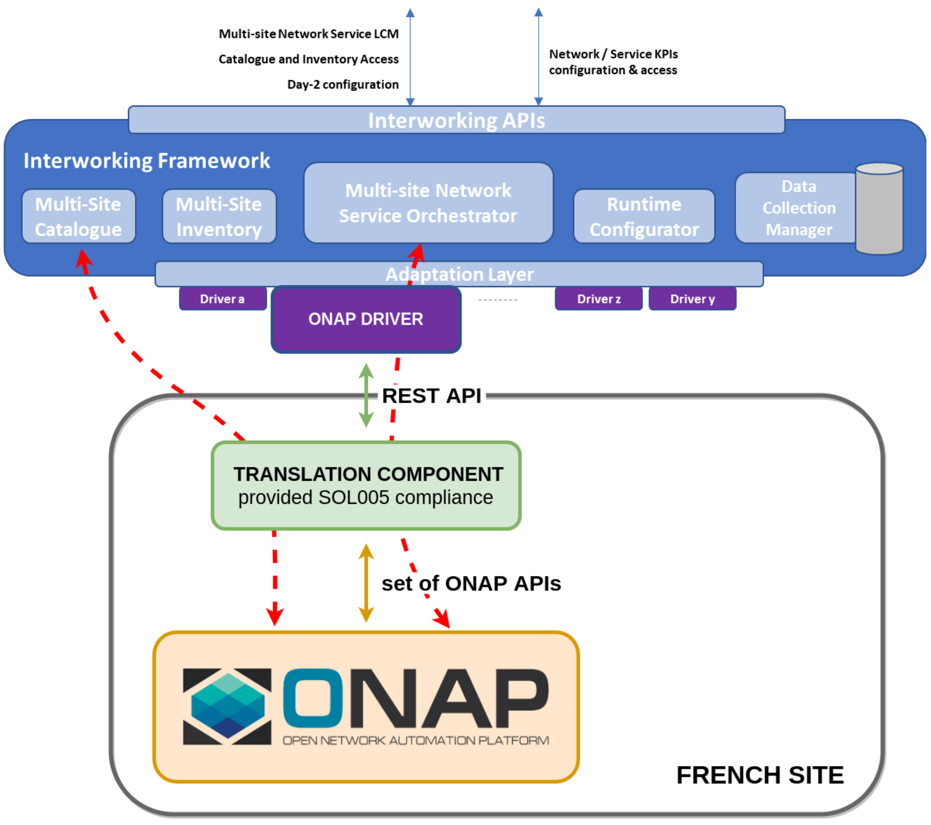

Figure 36: The overview architecture of French Site

A new additional component has been implemented between ONAP driver and adaptation layer. Main responsibility of Translation Component is to provide communication compliance with ETSI NFV SOL005 standard. As far as MSNO is communicating using SOL005 standard and ONAP uses its own custom interface - we introduced component that is translating SOL005 requests and calls ONAP external APIs. To be more precise - when any requests comes from IWL to French Sites - in first step it is passed to Translation Component that has defined REST APIs and may be treated as a "external APIs of French Site". Depending on the type of the request, Translation Component triggers action using directly ONAP internal or external API. That kind of solution has been introduced to allow translate all SOL005 based requests and trigger appropriate action on ONAP in both direction - to and from ONAP.

In this D3.4 delivery the following network service operations are implemented:

- create, instantiate, terminate and delete network service instances;
- retrieve a list of network services and life-cycle management operation occurrences;
- retrieve an information about selected network service or life-cycle management operation occurrence resource.

New functionalities supported in current release of French Site are related to notifications about life-cycle management operation occurrences of network services.

Table 10 describes in detail the API methods offered by Translation Component provided SOL005 compliance for ONAP.

Table 10: API provided by Translation Component as a "external API of French Site"

Service	Path	Method	Input	Output	Description
Translation Component	/instances	GET	-	Array of NS instances	Retrieve a list of NS instances
Translation Component	/instances/ {nsInstanceId}	GET	-	NS instance	Read an individual NS instance resource.
Translation Component	/create	POST	nsdId, nsName, nsDescription	NS identifier	Creates and returns a Network Service identifier (nsId) in a Nfvo
Translation Component	/instantiate/ {nsInstanceId}	POST	-	-	Instantiate a NS instance.
Translation Component	/terminate/ {nsInstanceId}	POST	terminationTime	-	Terminate a NS instance
Translation Component	/delete/ {nsInstanceId}	DELETE	-	-	Delete a NS instance
Translation Component	/ns_lcm_op_occs	GET	-	Array of LCM	Retrieve a list of NS LCM operation occurrences.

Translation Component	/ns_lcm_op_occs/ {nsLcmOpOccId}	GET	-	LCM Info	Retrieve an individual NS LCM operation occurrence resource.
Translation Component	/ns_lcm_op_occs/ ns_id/ {nsInstanceId}	GET	-	Array of LCM for NS instance	Retrieve a list of NS LCM operation occurrences for NS instance
Translation Component	/service_specification	GET	-	Array of NSD	Retrieve a list of NSD
Translation Component	/service_specification/{nsdId}	GET	-	csar archive for NSD	Download csar archive with service tosca model

Current D3.4 delivery focuses as well on integration of ONAP and MS-Catalogue. The main goal was to synchronize ONAP services descriptors with I/W Layer catalog. To achieve that, dedicated API to retrieve service specification from French Site was provided in Translation Component. It allows MS-Catalogue access to ONAP and retrieve (download) selected service specification as a TOSCA CSAR archive dedicated ONAP service description format. A dedicated tool has been also implemented inside MS-Catalog to translate ONAP service model to the service description format required by I/W Layer. It allows to receive and store specific information about services like topology, resource requirements, services management interfaces. More detailed information regarding this custom tool will be provided in MS-Catalog description. Information retrieved in synchronization ONAP-IWL catalog will be utilized by other components, like the 5G EVE Portal - to present detailed information about use cases in French site, or Runtime Configurator - to access configurable interfaces in VNFs.

3.8.2 Greek site

Several experiments for different use cases are planned to be executed at the Greek site. The key component for the successful execution of these experiments is the Orchestrator. This component will be the connecting link between the IWL and the Greek site and it will be responsible for translating the commands from the upper layers into actions in the Greek site infrastructure (Figure 37). After carefully studying the available options, OSM was selected for the role of the Greek site Orchestrator.

OSM is an ETSI-hosted open source community that delivers a production-quality MANO stack for NFV, capable of consuming openly published information models, available to everyone, suitable for all VNFs, operationally significant and VIM-independent. OSM is aligned to NFV ISG information models while providing first-hand feedback based on its implementation experience. As far as the deployment is concerned, it can either be distributed, derived from a set of pre-built downloadable containers (Docker) that can be easily deployed in minutes or as a single pre-configured vagrant box (pre-set VM). The deployment environments that have been tested with the OSM deployment are Ubuntu 18.04 or 16.04 (64-bit variant required).

OSM provides a unified northbound interface (NBI), based on NFV SOL005, which enables the full operation of system, Network Services and Network Slices under its control. In fact, OSM's NBI offers functionalities for managing the lifecycle of Network Services (NS) and Network Slices Instances (NSI), providing as a service all the necessary abstractions to allow the complete control, operation and supervision of the NS/NSI lifecycle by client systems, avoiding the exposure of unnecessary details of its constituent elements. It admits both

YAML/JSON formats and by default serves https (auto-signed certificate) on port 9999 and bearer authentication (with token) is used. Basic authentication or no authentication is also possible by changing the 'nbi.cfg' file.

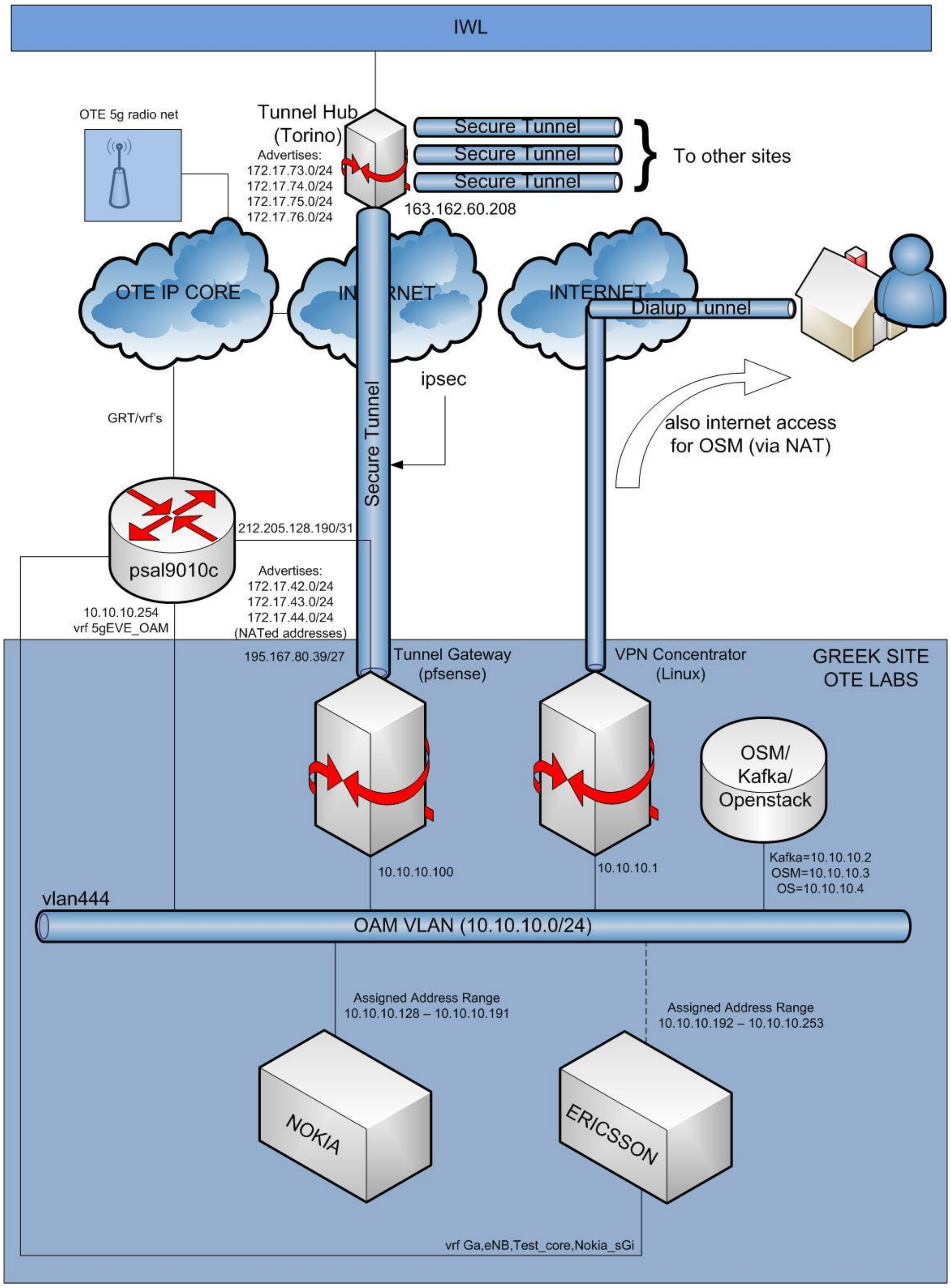
IWL
OTE 5g radio net
Tunnel Hub (Torino)
Advertises: 172.17.73.0/24 172.17.74.0/24 172.17.75.0/24 172.17.76.0/24
Secure Tunnel
Secure Tunnel
Secure Tunnel
To other sites
163.162.60.208
OTE IP CORE
INTERNET
INTERNET
Dialup Tunnel
ipsec
Secure Tunnel
also internet access for OSM (via NAT)
GRT/vrf's
psal9010c
212.205.128.190/31
Advertises: 172.17.42.0/24 172.17.43.0/24 172.17.44.0/24 (NATed addresses)
10.10.10.254 vrf 5gEVE_OAM
195.167.80.39/27
Tunnel Gateway (pfsense)
VPN Concentrator (Linux)
GREEK SITE OTE LABS
OSM/ Kafka/ Openstack
10.10.10.100
10.10.10.1
Kafka=10.10.10.2 OSM=10.10.10.3 OS=10.10.10.4
vlan444
OAM VLAN (10.10.10.0/24)
Assigned Address Range 10.10.10.128 – 10.10.10.191
Assigned Address Range 10.10.10.192 – 10.10.10.253
NOKIA
ERICSSON
vrf Ga,eNB,Test_core,Nokia_sGi

Figure 37: The overview architecture of Greek Site

The API implementation (Table 11) 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.

Finally, OSM provides an intuitive web user interface that can be exploited for operating the system locally or remotely. It can also be operated by using the OSM client that offers a Python client library and a command-line tool to interact with OSM.

Table 11 OSM API definition for Greek site

Network Service Descriptor Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve NSD	GET /nsd/v1/ns_descriptors/<nsd_id>/nsd_content
Onboard NSD	POST /nsd/v1/ns_descriptors
Modify NSD	PUT /nsd/v1/ns_descriptors/<nsd_id>/nsd_content
Delete NSD	DELETE /nsd/v1/ns_descriptors/<nsd_id>/
Retrieve list of NSDs	GET /nsd/v1/ns_descriptors
VNF Descriptor Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve VNFD	GET /vnfpkgm/v1/vnf_packages/<vnfPkgId>/ package_content
Retrieve list of VNFD	GET /vnfpkgm/v1/vnf_packages
PNF Descriptor Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve PNFD	GET /nsd/v1/pnf_descriptors/<pnfdId>/pnfd_content
Retrieve list of PNFD	GET /nsd/v1/pnf_descriptors
Network Service Instance Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve NSI	GET /nslcm/v1/ns_instances /<nsInstanceId>
Create NSI	POST /nslcm/v1/ns_instances
Instantiate NSI	POST /nslcm/v1/ns_instances/<nsInstanceId>/instantiate
Delete NSI	DELETE /nslcm/v1/ns_instances /<nsInstanceId>
Terminate NSI	POST /nslcm/v1/ns_instances /<nsInstanceId>/terminate
Scale NSI	POST /nslcm/v1/ns_instances/<nsInstanceId>/scale
Retrieve list of NSI	GET /nslcm/v1/ns_instances
VNF Instances Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve VNFI	GET /nslcm/v1/vnf_instances/<vnfInstanceId>
Retrieve list of VNFI	GET /nslcm/v1/vnf_instances

PNF Instances Management	
Adaptation Layer Operation	***OSM equivalent Operation***
Retrieve PNFI	GET /nslcm/v1/vnf_instances/<vnfInstanceId>
Retrieve list of PNFI	GET /nslcm/v1/vnf_instances

3.8.3 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.

As shown in the following figure, the NFV orchestration in the Italian site facility is performed by two solution that coexists: an open source OSM orchestrator, and an Ericsson orchestrator called EVER. In particular, the OSM orchestrator is mostly dedicated to the coordination of the vertical VNFs (i.e. those providing the specific Use Case applications), while EVER orchestrator coordinates 4G/5G network related VNFs and functionalities (e.g. EPC, 5G Core, RAN).

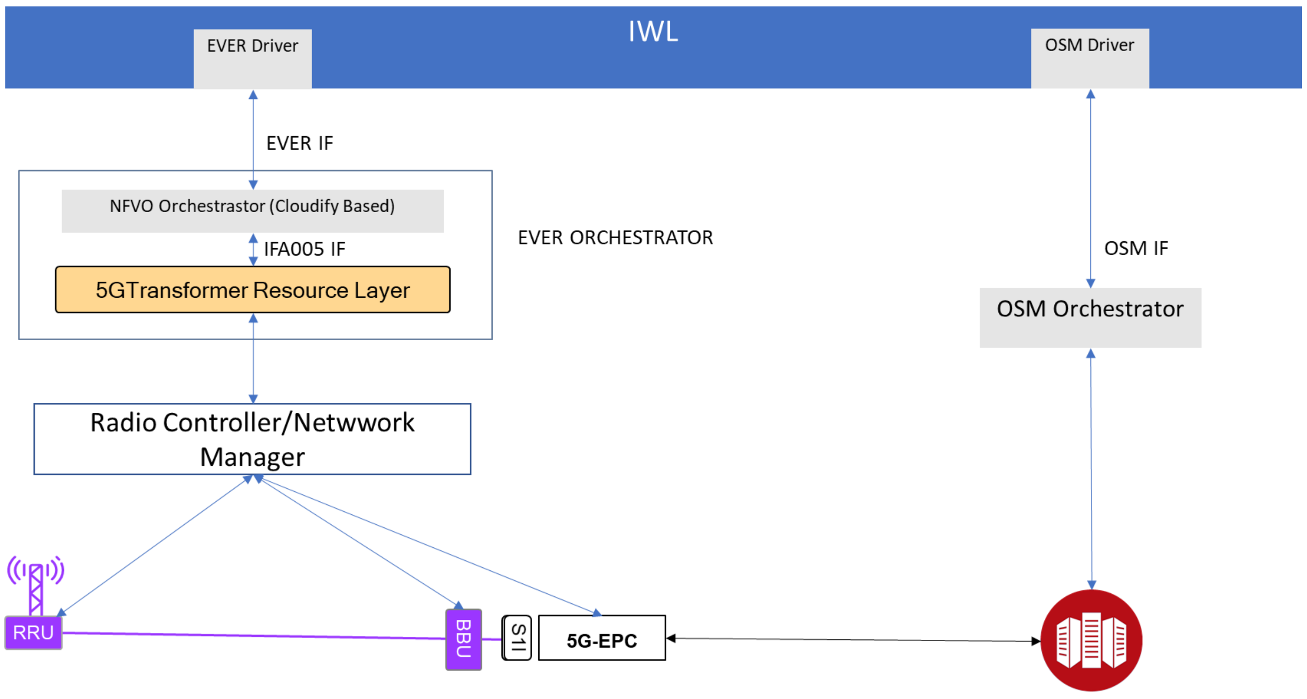
IWL
EVER Driver
OSM Driver
EVER IF
NFVO Orchestrastor (Cloudify Based)
IFA005 IF
5GTransformer Resource Layer
EVER ORCHESTRATOR
OSM IF
OSM Orchestrator
Radio Controller/Netwwork Manager
RRU
BBU
S1I
5G-EPC

Figure 38: NFVO Orchestrators at the Italian site

In D3.4 we provide a first implementation of EVER and related driver for the Adaptation layer (called "EVER Driver"). The EVER orchestrator is formed by an NFV orchestrator based on Cloudify[31] that handles the 4G/5G network NFV and by Resource Layer framework released in GitHub[32] and developed in EU H2020 5GTransformer[33] project that is used to handle the infrastructure resources. The following operations are handled by EVER orchestrator:

[31] https://cloudify.co/

[32] https://github.com/5g-transformer/5gt-mtp

[33] http://5g-transformer.eu/

- create, instantiate, terminate and delete 4G/5G network service instances (e.g. network services lifecycle management or LCM);
- retrieve a list of network services and life-cycle management operation occurrences;
- retrieve information about selected network service or life-cycle management operation occurrence resource.

The retrieve operations are implemented and tested, while the LCM is partially implemented and will be finalized in the next Deliverable D3.5. The reason is the LCM in EVER orchestrator is handled according the 3GPP RAN slice model described in [10] and [12]. This model requires extending the current IWL implementation by adding "3GPP RAN Slice" specific NS parameter for RAN orchestration. Such extension will be provided and described in final D3.5, so a final version of EVER orchestrator with the complete integration tests will be described in that deliverable.

Regarding the EVER driver, it is included in the adaptation layer repository ("mso-lo" project) and related tests are details in D3.7.

Table 12 describes in detail the API methods offered by EVER orchestration that are used by the driver.

Table 12: API provided by EVER orchestrator to EVER driver

Service	Path	Method	Input	Output	Description
Retrieve NS list	/instances	GET	-	Array of NS instances	Retrieve a list of NS instances
Retrieve NS by id	/instances/ {nsInstanceId}	GET	-	NS instance	Retrieve an individual NS instance resource.
LCM Network Service	/create	POST	nsdId, nsName, nsDescription	NS identifier	Creates and returns a Network Service identifier (nsdId) in a NFVO
LCM Network Service	/instantiate/ {nsInstanceId}	POST	-	-	Instantiate a NS instance identified by the nsInstanceId
LCM Network Service	/terminate/ {nsInstanceId}	POST	terminationTime	-	Terminate a NS instance identified by the nsInstanceId
LCM Network Service	/delete/ {nsInstanceId}	DELETE	-	-	Delete a NS instance identified by the nsInstanceId
Retrieve LCM list	/ns_lcm_op_occs	GET	-	Array of LCM	Retrieve a list of NS LCM operation occurrences.
Retrieve LCM by Id	/ns_lcm_op_occs/ {nsLcmOpOccId}	GET	-	LCM Info	Retrieve an individual NS LCM operation occurrence resource.

3.8.4 Spanish site

In order to carry out the experimentation planned to be performed at the Spanish site, here we present an NFV system mainly constituted by a software MANO stack and two standalone private cloud platforms as depicted in Figure 39. The cloud platforms comprise the NFV infrastructure that enables the deployment of VNFs both at the cloud and/or at the edge sides, depending on the experimentation procedure. Related with the resource orchestration, ETSI OSM is in charge of providing the MANO implementation and, in particular, this platform utilizes OSM Release SEVEN [6]. To facilitate the installation and the deployment of the NFVO and VNFM stack, ETSI OSM is executed as a virtual machine within the 5TONIC laboratory and in compliance with the requirements established by the OSM community.

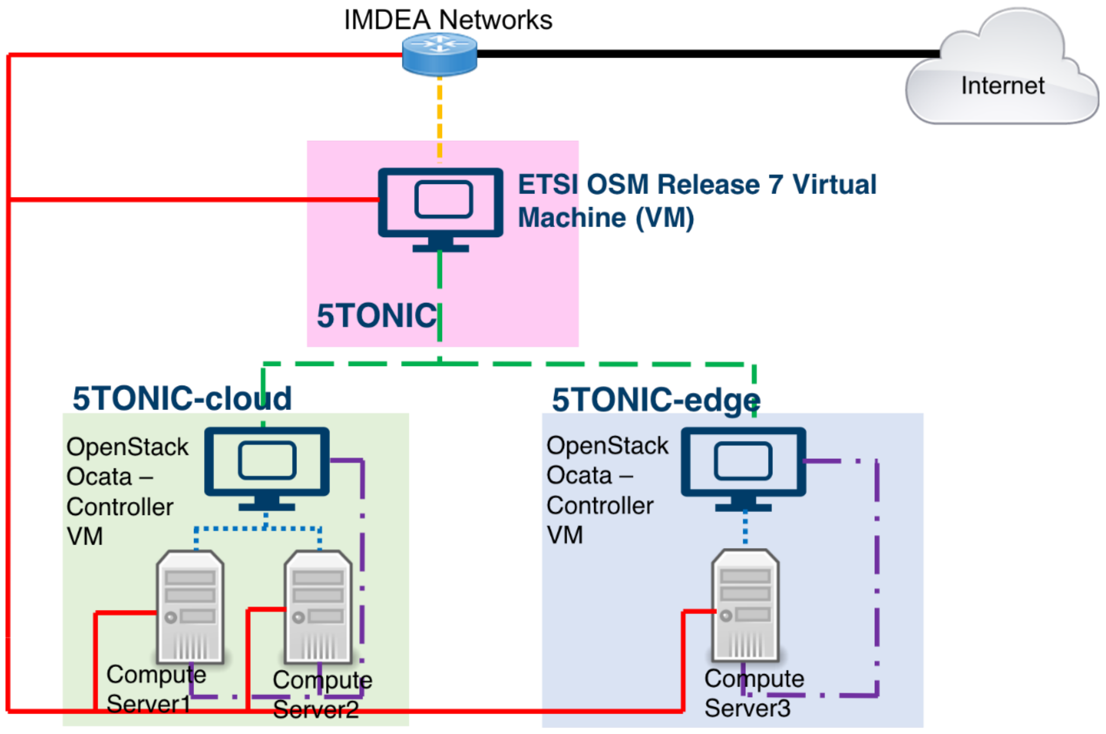
IMDEA Networks
Internet
ETSI OSM Release 7 Virtual Machine (VM)
5TONIC
5TONIC-cloud
OpenStack Ocata – Controller VM
Compute Server1
Compute Server2
5TONIC-edge
OpenStack Ocata – Controller VM
Compute Server3

Figure 39: MANO at the Spanish site

Besides the NFVO and VNFM stack, the 5TONIC laboratory hosts the two aforementioned cloud platforms, both based on OpenStack release Ocata. For the first platform, two high-profile servers provide a total amount of computational capability of 48 vCPUs, 256 GB of RAM and 8 TB for storage.

To manage this amount of resources that cover a portion of the NFVI, the OpenStack node that fulfils the VIM operations (i.e., the OpenStack controller) runs in a virtual machine with 4 vCPUS, 16 GB RAM and 180 GB of storage. This cloud platform is represented as 5TONIC-cloud in Figure 39.

The second cloud platform included in the 5TONIC laboratory has one server computer with 4 vCPUs, 8 GB RAM and 256 GB of storage as computation capabilities.

As in the previous platform, the VIM is based on an OpenStack controller, which runs in a virtual machine with 4 vCPUs and 12 GB RAM, 600 GB of storage.

In reference to this second cloud platform is indicated as 5TONIC-edge in Figure 39.

Even though these two platforms are independent and managed by their own VIM, both are compliant with ETSI OSM Release SEVEN and integrate the OpenStack Telemetry service, supporting the performance metrics collection of the allocated VNFs. Thus, the experimentation procedure can benefit from the auto-scaling functionality provided by ETSI OSM. Moreover, both infrastructures implement the OpenStack networking service, where both the pre-created and tenant networks options have been configured.

On the other hand, it is worth mentioning that the testbed also integrates the required networking that enables the proper operation of the complete NFV system, supporting the orchestration and deployment of multi-site network services over the three cloud platforms comprising the NFVI. These networks can be summarized as follows:

- Infrastructure management networks (i.e., the blue round dot line in Figure 39): the aim of these types of networks is to allow each VIM to be capable of managing the computational resources of the cloud platform under its control. Therefore, these networks are present independently in all the parts of the NFVI testbed.
- Orchestrator-to-VIM networks (i.e., the green long dash line in Figure 39): these networks are in charge of supporting the reservation and allocation of the necessary resources for enabling the subsequent network services deployment. In addition, these networks are in charge of the lifecycle management of the stated network services and slices. In this context, the ETSI OSM is able to interact with each of the VIMs included in the testbed and handle the designed deployments in the experimentation procedure.
- Orchestrator-to-VNF networks (i.e., the red solid line in Figure 39): the objective in this case is to allow the ETSI OSM platform to control and monitor the VNFs lifecycle (i.e., get the VNFs state information and support the scaling operations based on this information), as well as the VNF configuration. For this reason, ETSI OSM requires IP connectivity with each of the VNFs hosted by their respective platform. This network is exposed outside the infrastructure to allow other 5G EVE components, like the Runtime Configurator, to manage all VNFs.
- Service-oriented networks (i.e., the purple long dash dot line in Figure 39): this latter encompasses the networks among the different VNFs comprising a network service (regardless of the platform on which they are executed) to ensure the proper functioning of the implemented service.
- IWL network (i.e., the orange dotted line in Figure 39): this network is used to provide access to the Spanish OSM to the IWL components hosted by TIM at the Turin site.

It is important to highlight that 5TONIC provides specific network services to support all the networks mentioned above, even with the other three sites of the 5G EVE infrastructure to enable the user data plane, further away from the 5TONIC laboratory (located in Madrid, Spain). Similarly, the communication between the MSNO and the site adaptors available in Turin can access the OSM deployed in 5TONIC, where a dedicated project is available to the 5G EVE project. To this end, we leverage on the NAT box provided by IMDEA Networks to connect with the VPN tunnel established with TIM, through the Internet. On the one hand, the traffic dissemination in the case of 5TONIC is delivered through RedIRIS, which is the high-performance national research and education network in Spain.

4 Inter-site connectivity status

As reported in D3.3 [3], we selected a star architecture for connecting the 5G EVE sites, being Turin's site the centre of the connectivity. Using this, we selected a distributed deployment for the 5G EVE Framework, with the 5G EVE Portal in 5TONIC and the IWL in Turin's site.

As we planned, we interconnect three networks:

- Orchestration and Management network, required for connecting 5G EVE components and for the IWL-site connectivity.
- Control plane network, for connecting 5G Core elements.
- User plane network, for connecting elements in the data path of the 5G users.

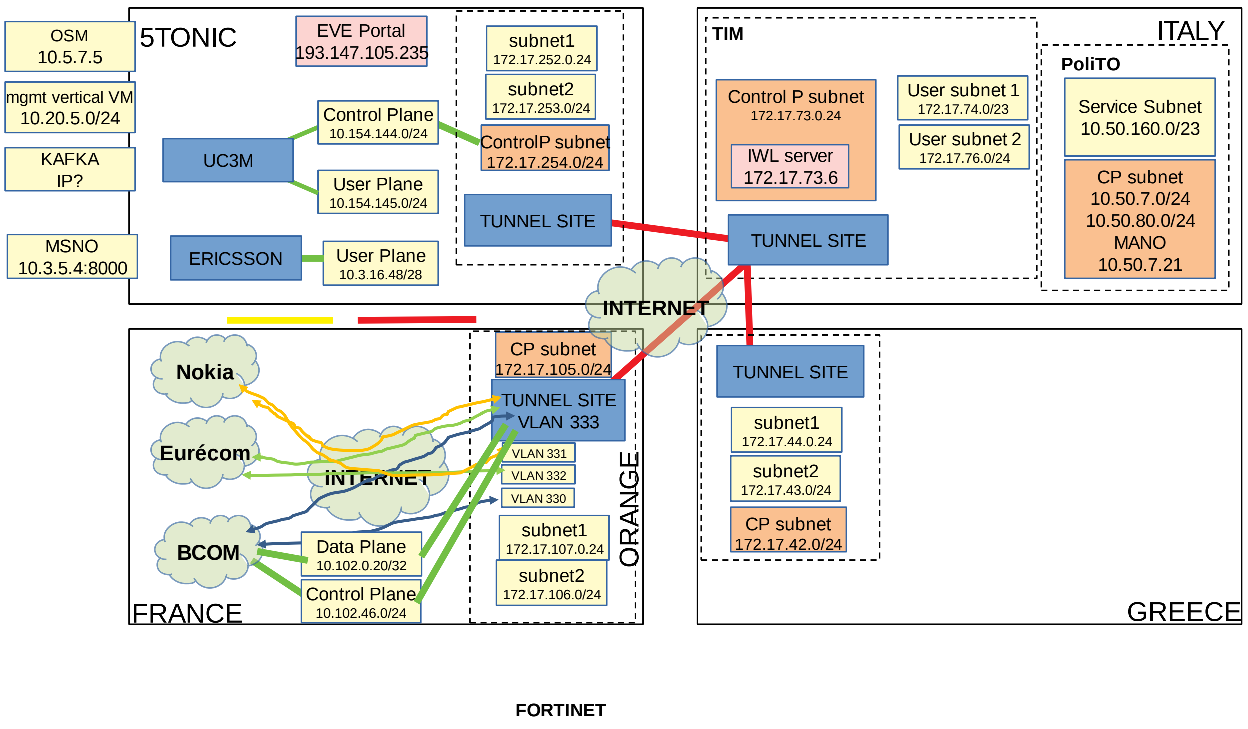

Figure 40: 5G EVE site integration

One issue we faced is the IP addressing overlapping among the 5G EVE sites. Since this is not an issue easy to solve, as it involves parts of the networks that are not related with 5G EVE project, we decided to use an IP translation system, which allows us to define common subnets for all the 5G EVE sites.

We describe in the Figure 40 the integration of the 5G EVE lab with TIM's lab.

The performance measurements of the inter-site connectivity are described in the Deliverable 3.7, which will be published at the same time of this deliverable.

5 Updated roadmap

According to the proposal (Figure 41), the 5G EVE project has two official deliveries of the I/W framework: the previous D3.3 and the delivery described in this document , with the full I/W framework.

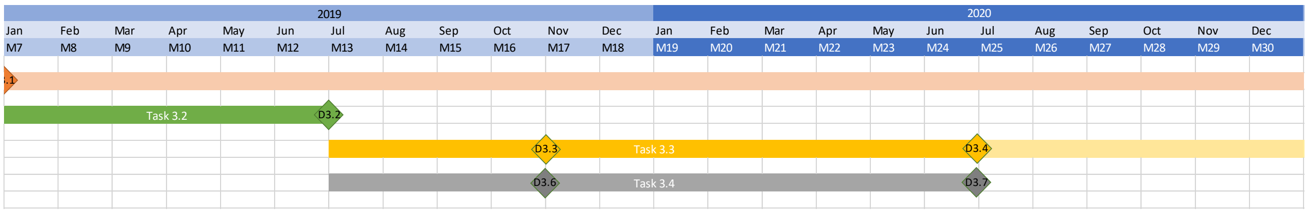

Figure 41: Task plan for WP3 for the 5G EVE second year

In Table 13 we show the plan for supporting each of the features included in D3.2, based on WP3 plan and the 5G EVE project roadmap published in WP1. We decided to concentrate both Drop 2 and D3.4 in the same software drop, and for this reason we merged the roadmap as described in the table below.

One of the impacts of the COVID-19 situation is in the effort required for integrating all the components of the 5G EVE project, concretely those components with a hard dependency with the sites. For that reason, we re-planned the development of the IWL for postponing those features that are not essential for the MS9, basically related to the support of Multi-site experiments.

The multi-site experiments are supported as we have in place the inter-site connectivity. What we delayed is the automatic deployment of Network Services in more than one 5G EVE site. We assume that those kinds of deployments can be done using several single site deployments.

For the next delivery (D3.5, M35) of the IWL we plan to include:

- Support of multiple site NFV-O, by selecting the appropriate NFV-O depending on the availability of the VNF.
- Enhance the support of RAN controllers.
- Implement the deployment of multi-site network services using a single experiment.

Table 13: 5G EVE Interworking Framework Roadmap

Key Features	Category	Brief Description	D3.3 (M16)	Drop 1 (M18)	D3.4 (M24)
Orchestration Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for orchestration traffic	Italian and French sites	Fully provided	Fully 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)	Fully 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)	Fully 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)	Fully provided
Control Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for control traffic	-	-	Fully provided
Data Plane Interworking	Connectivity	Secure connectivity among sites for user traffic. Low bandwidth	-	-	Fully provided

		performance experiments will employ best effort connectivity. High bandwidth performance experiments will employ a parallel high bandwidth low latency network, which will be available at least between two sites.			
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.	-	-	Fully 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.	-	-	Support of multi orchestration scenarios for single sites
Vertical Slicing	Slicing	End-to-end slice that satisfies the requirement of the Vertical	-	Selected UC	Fully provided
Multi-Site Slicing	Slicing	Vertical Slice that spans across multiple sites	-	-	Supported for connectivity

6 Conclusions

We present in this document the result of the work of WP3 during the second year of the project.

The focus of the Inter Working Layer development was in the integration of the 5G EVE components. For this reason, we released two software deliverables for the Milestone 8 and the Milestone 9, with all required features for executing the vertical use cases with the scope defined in each Milestone.

This report shows how IWL solves the integration of heterogenous sites by providing a unique interface for the orchestration and management of experiments on the sites. We successfully integrate the 5G EVE sites and now a vertical is able to deploy and execute an experiment regardless the underlying technology implemented in the site.

Also, we demonstrate that the modular and open approach incorporates easily new requirements (Telefónica gaming use case) and component (RAN orchestrator, IWL repository) and we think that this is a good baseline for future integration with new sites, e.g. coming from ICT-19 projects.

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] 5G EVE D3.3: First implementation of the interworking reference model, available at https://doi.org/10.5281/zenodo.3628179

[4] 5G EVE D3.6: Interworking test suites, available at https://doi.org/10.5281/zenodo.3628189

[5] ETSI GS NFV-SOL 005 V2.6.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point, available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.06.01_60/gs_nfv-sol005v020601p.pdf

[6] OSM, "Open Source MANO: ETSI-hosted project to develop MANO software stack aligned with ETSI NFV", Accessed on: Jun. 1, 2020.[Online]. Available: https://osm.etsi.org/

[7] 5G EVE D.2.6: Participating vertical industries planning. Available at https://doi.org/10.5281/zenodo.3628261

[8] ETSI GS NFV-SOL 001 V2.6.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification. Available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/001/02.06.01_60/gs_NFV-SOL001v020601p.pdf

[9] 5G EVE D4.1: Experimentation tools and VNF repository. Available at https://doi.org/10.5281/zenodo.3628201

[10] ETSI GS NFV-SOL 004 v2.4.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; VNF Package specification .Available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/004/02.05.01_60/gs_nfv-sol004v020501p.pdf

[11] 3GPP TS28.530, V15.1.0, Telecommunication management; Management of 5G networks and network slicing; Concepts, use cases and requirements, 2018

[12] 3GPP TS28.540, V15.1.0 Management and orchestration; 5G Network Resource Model (NRM); Stage 1

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

[14] Ramon Perez, Jaime Garcia-Reinoso, Aitor Zabala, Pablo Serrano, Albert Banchs, “A Monitoring Framework for Multi-Site 5G Platforms”, 2020 European Conference on Networks and Communications (EuCNC): Operational \& Experimental Insights (OPE), pp. 52-56, 2020.

[15] ONAP TOSCA model: available at https://wiki.onap.org/display/DW/4.+ONAP+Internal+TOSCA+Modeling+and+Data+Model

Annex A. Example of TCB based on the information model proposed for the scripts

Related to the Runtime Configurator described at section 3.5, we include an example of Test Case Blueprint:

```
testCaseBlueprint:
  description: Example for integration with the RC
  name: Example TCB
  configurationScript: [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress ansible_user=$$userB
ansible_ssh_pass=$$passwordB ansible_become_pass=$$passwordB\" | tee -a hosts && export
ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e
'script=\"/bin/bash /home/eve/install_datashipper.sh delay_iface $$metric.topic.delay_iface
$$metric.site.delay_iface $$metric.unit.delay_iface $$metric.interval.delay_iface
$$metric.deviceId.delay_iface\"'; [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress ansible_user=$$userA
ansible_ssh_pass=$$passwordA ansible_become_pass=$$passwordA\" | tee -a hosts && export
ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e 'script=\"sudo
apt-get install python3\"'
  executionScript: [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress ansible_user=$$userA
ansible_ssh_pass=$$passwordBA ansible_become_pass=$$passwordA\" | tee -a hosts && export
ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e 'script=\"ping
-c5 vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress\"'; sleep $$sleep_time; [ ! -e hosts ] || rm hosts
&& touch hosts && echo \"server ansible_host=vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress
ansible_user=$$userB ansible_ssh_pass=$$passwordB ansible_become_pass=$$passwordB\" | tee -a hosts
&& export ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e
'script=\"ping -c5 vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress\"'
  resetConfigScript: [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress ansible_user=$$userA
ansible_ssh_pass=$$passwordA ansible_become_pass=$$passwordA\" | tee -a hosts && export
ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e 'script=\"sudo
systemctl stop filebeat\"'; [ ! -e hosts ] || rm hosts && touch hosts && echo \"server
ansible_host=vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress ansible_user=$$userB
ansible_ssh_pass=$$passwordB ansible_become_pass=$$passwordB\" | tee -a hosts && export
ANSIBLE_HOST_KEY_CHECKING=False && ansible-playbook -i hosts execute_script.yml -e
'script=\"/bin/bash /home/eve/stop_datashipper.sh\"'
  userParameters:
    userA: $$userA
    passwordA: $$passwordA
    userB: $$userB
    passwordB: $$passwordB
    sleep_time: $$sleep_time
  infrastructureParameters:
    $$metric.topic.delay_iface
    $$metric.site.delay_iface
    $$metric.unit.delay_iface
    $$metric.interval.delay_iface
    $$metric.deviceId.delay_iface
    vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress
    vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress

  version: '2.0'
```