5G ZORRO

Grant Agreement 871533

H2020 Call identifier: H2020-ICT-2019-2
Topic: ICT-20-2019-2020 - 5G Long Term Evolution

D4.3: Final prototype of Zero Touch Service Management with Security and Trust

Dissemination Level		
☒	PU	Public
☐	PP	Restricted to other programme participants (including the Commission Services)
☐	RE	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO	Confidential, only for members of the consortium (including the Commission Services)

Grant Agreement no:	Project Acronym:	Project title:
871533	**5GZORRO**	**Zero-touch security and trust for ubiquitous computing and connectivity in 5G networks**

Lead Beneficiary:	Document version:
ALB	**V1.1**

Work package:
WP4 - Zero Touch Automation with Trust, Security and AI

Deliverable title:
D4.3: Final prototype of Zero Touch Service Management with Security and Trust

Start date of the project:	Contractual delivery date:	Actual delivery date:
01/11/2019 **(duration 30 months)** **(extended to 36 months)**	**30/Jun/2022**	**08/07/2022**

Editor(s)
André Gomes (ALB)

List of Contributors

Participant	Short Name	Contributor
Nextworks	NXW	Pietro G. Giardina, Juan Brenes, Michael De Angelis, Elena Bucchianeri, Giacomo Bernini
i2CAT Foundation	i2CAT	Adriana Fernández-Fernández, Carlos Herranz Claveras, Javier Fernandez Hidalgo, Muhammad Shuaib Siddiqui
Universidad de Murcia	UMU	José María Jorquera Valero, Pedro Miguel Sánchez Sánchez, Manuel Gil Pérez, Gregorio Martínez Pérez
Atos Spain	ATOS	Guillermo Gómez Chavez
IBM Israel Science and Technology	IBM	David Breitgand, Kathrine Barabash
Altice Labs	ALB	André Gomes, Bruno Santos
Intracom	ICOM	Alberto Erspamer, Dimitrios Laskaratos, Vasileios Theodorou
Ubiwhere	UW	Filipa Martins, Pedro Teixeira
Fondazione Bruno Kessler	FBK	Rasoul Behravesh, Cristina Costa

List of Reviewers

Participant	Short Name	Contributor
Intracom	ICOM	Marinela Mertiri
Atos Spain	ATOS	Aurora Ramos
Nextworks	NXW	Giacomo Bernini
i2CAT Foundation	I2CAT	Muhammad Shuaib Siddiqui

Change History

Version	Date	Partners	Description/Comments
0.0	27 May 2022	ALB	Initial version release
0.1	13 Jun 2022	IBM, ICOM, UMU, UW	Contributions on sections 2 and 3
0.2	20 Jun 2022	UMU, UW	Review of content provided in section 2
0.3	24 Jun 2022	NXW, ATOS, ALB, UMU, i2CAT, UW	Contributions & review on sections 2, 3, 4 and 5
0.4	28 Jun 2022	ATOS, UW, ALB, FBK	Contributions & review on sections 2, 4, 5 and 6
0.5	30 Jun 2022	ATOS, ICOM, ALB	Internal Review
0.6	7 Jul 2022	NXW, ATOS, ALB, UMU, UW, IBM, ICOM, FBK	Contributions & review on sections 2, 3, 4, 5 and 6
1.0	8 Jul 2022	ALB, NXW, i2CAT	Fixed comments and suggestions for final version
1.1	22 Feb 2023	ALB	URLs and typos fixed, . Final version for public release.

DISCLAIMER OF WARRANTIES

5GZORRO has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 871533. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

Table of Contents

List of Tables

List of Figures

Executive Summary

This document reports on the final prototypes of the zero-touch slice and service management solutions, including security and trust orchestration, that have been developed as part of the 5GZORRO platform. These have been carried out in the context of Work Package (WP) 4, and this document describes the essential features of all trusted and secure zero-touch slice and service management software components. The implemented prototypes are aligned to the design guidelines reported in previous deliverables (i.e., D4.1 and D4.4) and are the result of continuous development and integration efforts performed by multiple software teams, taking also into account validation feedback collected from WP5 as part of the project use cases integration and KPI assessment.

In summary, this document is organized to describe:

- The software components that have been implemented inside each one of the three WP4 functional pillars: Security and Trust Orchestration, Intelligent and Automated (zero-touch) Slice and Service Management, and MANO and slicing enhancements.
- The main functionalities and software prototype implementation details for each developed component, including the results of functional tests to validate the integration of the various components within the 5GZORRO platform.
- References to the major prototype components and the respective source code repositories hosted in the 5GZORRO GitHub available at: https://github.com/5gzorro
- Results of validation tests, including functional tests for the various modules and integration tests, to demonstrate the proper functionality of the involved components and their interrelation.
- A proposed approach for automated Installation instructions for the presented prototypes, with the identification of deployment profiles and dependencies among the various required 5GZORRO components and modules

1. Introduction

5GZORRO has been tackling the issue of network services orchestration in a multi-party environment, implementing a new state-of-art platform for distributed orchestration, security, and trust solutions. This platform is capable of multi-domain/multi-stakeholder smart resource selection and trading, with greatly reduced manual intervention (zero-touch automation) and increased secure end-to-end chains across domains.

To achieve resource & service management automation, the zero-touch service management and orchestration platform of 5GZORRO provides continuous data transformation, including a proactive scaling mechanism, that adjusts the infrastructure's capacity to host resources published in the marketplace, allocating resources according to demand. For the usage of external resources in a trusted manner, the platform relies on a Security & Trust framework that secures communications between third partied, at both intra and inter-domain environments.

The set of software modules designed and developed in 5GZORRO, to achieve what was mentioned beforehand, have been continuously integrated and developed, aiming to reflect the defined architecture of the 5GZORRO platform that evolved up until the final stages of the project.

Building and evolving on top of what was reported in D4.2 [1] for the intermediate software prototypes of the 5GZORRO zero-touch service and slice management functionalities, this document describes the final prototypes resulting from the development work done to incorporate the security and trust framework developed across WP4's tasks 4.1, 4.2 and 4.3 into the 5GZORRO platform, giving support to zero touch service management.

Specifically, this document presents the final release of software prototypes corresponding to the consolidation of WP4 outcomes. These are maintained in the 5GZORRO GitHub organization available at:

https://github.com/5gzorro

and tagged as *5gzorro-full-1.0-pre-final*. However, some adjustments can still be realized in the different prototypes until the end of the project based on feedback received from the use case validation trials performed in WP5. These will be directly reported in the software documentation material included in the available repositories (and tagged as *final*).

1.1. Document outline

This document is structured in two main parts.

In the first part, the status of the modules is reported in terms of main functionalities, developments of the prototypes, and functional tests.

Section 2 focuses on the modules that implement the concepts of security and trust into the 5GZORRO platform, these being the 5G-enabled Trust and Reputation Management Framework (5G-TRMF), the Trusted Execution Environment (TEE) Security Management, the Security Analysis Services (SAS), and the VPN-as-a-Service.

Section 3 describes the automated management of secured cross-domain slices and services, which is implemented by the Intelligent and Automated Slice & Service Management (ISSM) and its sub-modules.

Section 4 reports on the set of components that build the 5GZORRO orchestration stack of slices, and the enhancements of the MANO stack in terms of resource management, orchestration, and licensing. These are the Any Resource Manager (xRM), Network Slice and Service Orchestrator (NSSO), Network Service Mesh Manager (NSMM), and the e-Licensing Manager.

In the second part, which maps into Section 5, the final integration work is reported against generic scenarios which will be then specialised in WP5 in the context of more specific use case validation scenarios. The section is organised to cover three different integration scenarios: zero-touch network slice orchestration, E-licensing control, and data-driven actuation.

Section 6 provides the instructions necessary to install the developed prototypes, including the mapping of modules required according to the stakeholder role.

Section 7 presents some concluding remarks.

Some specific theoretical background at the base of the 5G-enabled Trust and Reputation Management Framework implementation has been reported in the Appendix I as complement to the core prototype description information presented in section 2.1. The content of this Appendix extends the information already provided with D4.2 [1], with new information concerning community context factors, and general rewards and penalty equations.

2. Security and Trust Orchestration

The 5GZORRO Security and Trust layer introduces the security and trust functionalities required to ensure the selection of trusted third parties, protect a tenant service or application running in a computing node, and secure the communications between 5GZORRO components. In this vein, Security and Trust Orchestration plays an important role not only for the correct 5GZORRO platform functioning but also for bringing a high-level trustworthiness. Therefore, the Security and Trust layer can be interpreted as an enabler both at intra-domain and inter-domain environments.

The 5GZORRO Security and Trust Orchestration sub-system is split into four modules:

- 5G-enabled Trust and Reputation Management Framework (5G-TRMF), which is responsible for determining stakeholders' trust score and establishing reliable relationships. The 5G-TRMF is in turn interconnected with the Security Analysis Service (SAS) as well as other external components to the Security and Trust layer such as the Data Lake, the Smart Resource and Service Discovery, etc.
- Trusted Execution Environment Security Management, which enables critical workloads in multi-tenant and multi-stakeholder scenarios. In this case, the TEE is not directly interconnected with other Security and Trust components, but it interacts with essential components such as SLA Monitoring, Intelligent SLA Breach Predictor and Monitoring Data Aggregation.
- Security Analysis Service, which offers internal security services such as detecting and mitigating feasible attacks and threats, being such information consumed by the 5G-TRMF to update trust scores.
- Inter-domain Security module, which guarantees a secure end-to-end communication through an on-demand VPN as a Service (VPNaaS). Moreover, this module interacts with the Identity and Permissions Management to verify the identity of another gateway. Note that the Identity and Permissions Management component is described in D3.2 [49] since it is a cross-work package module.

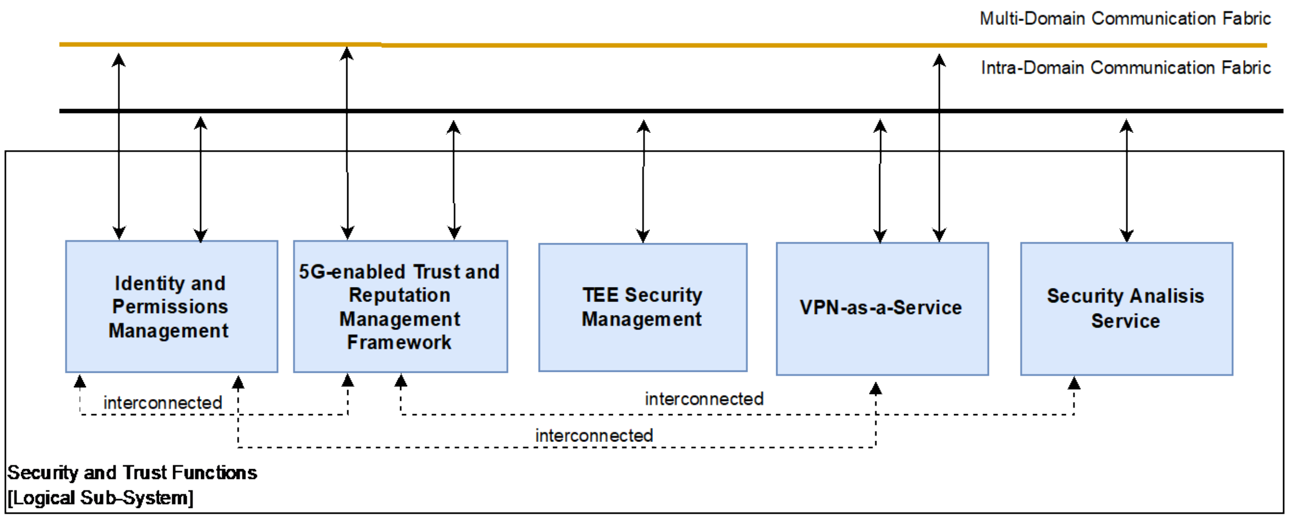

Multi-Domain Communication Fabric
Intra-Domain Communication Fabric
Identity and Permissions Management
5G-enabled Trust and Reputation Management Framework
TEE Security Management
VPN-as-a-Service
Security Analisis Service
interconnected
interconnected
interconnected
Security and Trust Functions
[Logical Sub-System]

Figure 2-1: Security and Trust components

Figure 2-1 displays the modules that are part of the Security & Trust logical sub-system of 5GZORRO platform.

2.1. 5G-enabled Trust and Reputation Management Framework

5GZORRO ecosystem brings several novelties, for instance, enabling secure, flexible, and automated business establishments that boost new compositions of resources and services in 5G networks by means of multi-stakeholder combinations. 5GZORRO introduces the 5G-enabled Trust and Reputation Management Framework (5G-TRMF) that is the module in charge of guaranteeing the establishment of a trustworthy end-to-end chain across domains. Therefore, in a situation where a stakeholder needs to forecast trust levels of multiple stakeholders to identify the most appropriate entity, the 5G-TRMF will provide the required functionality to assess stakeholders' reputation, and in consequence, to ensure a reliable selection based on past experiences and recommendations.

2.1.1. Main Functionalities

The scope of the 5G-TRMF is not only to evaluate stakeholders' trust score, but also to manage the whole lifecycle of the trust computation process. The main functionalities of a trust lifecycle management are:

- Data collection process from available storage sources (Data Lake, Resource and Service Offer Catalogue and a dedicated trust database).
- Assessment of trust information from own history and recommendations.
- Decision-making based on a trust score.
- Trust information storage for future interactions.
- Continuous evaluation of trust in an established relationship using security and SLA events.

Novel considerations related to the interactions of the 5G-TRMF with other 5GZORRO components have emerged since the release of the related intermediate prototype in deliverables D4.2 [1]. Therefore, the prior workflow has been readjusted to better describe the final interactions of the 5G-TRMF with other key modules and the triggered actions. As it can be observed in Figure 2-2, the Smart Resource and Service Discovery (SRSD) application module is the one in charge of launching the trust computation process for an initial list of product offers pre-classified by the SRSD employing mechanisms such as imperative constraints and considerations provided by the consumer. This capability is started through the *requestTrustScore* method (ref. step 3 in Figure 2-2).

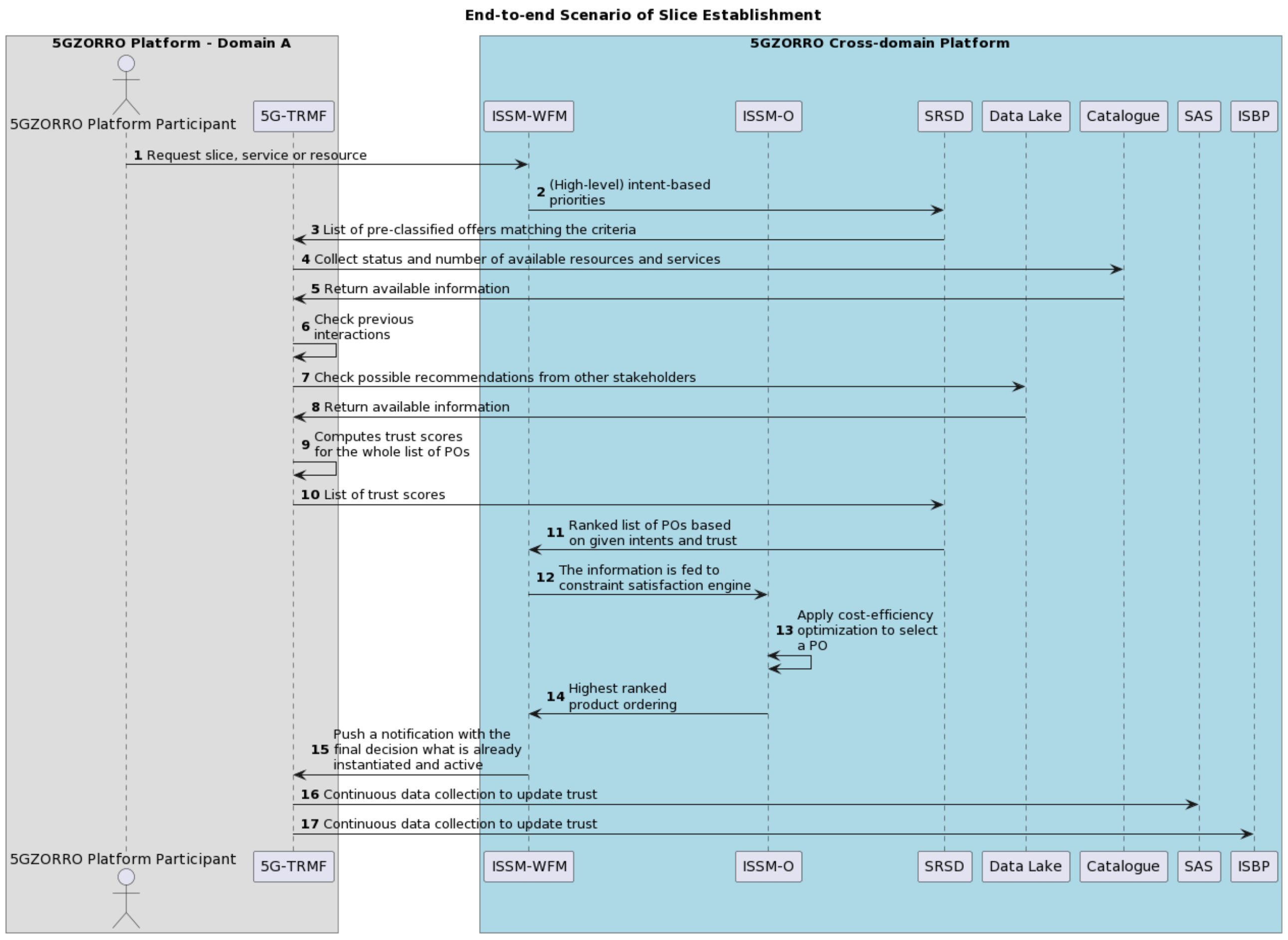

End-to-end Scenario of Slice Establishment
5GZORRO Platform - Domain A
5GZORRO Cross-domain Platform
5GZORRO Platform Participant
5G-TRMF
ISSM-WFM
ISSM-O
SRSD
Data Lake
Catalogue
SAS
ISBP
1 Request slice, service or resource
2 (High-level) intent-based priorities
3 List of pre-classified offers matching the criteria
4 Collect status and number of available resources and services
5 Return available information
6 Check previous interactions
7 Check possible recommendations from other stakeholders
8 Return available information
9 Computes trust scores for the whole list of POs
10 List of trust scores
11 Ranked list of POs based on given intents and trust
12 The information is fed to constraint satisfaction engine
13 Apply cost-efficiency optimization to select a PO
14 Highest ranked product ordering
15 Push a notification with the final decision what is already instantiated and active
16 Continuous data collection to update trust
17 Continuous data collection to update trust

Figure 2-2: 5G-TRMF interactions with SRSD, Resource and Service Offer Catalogue and Data Lake

Once the 5G-TRMF receives a set of offers to be evaluated, it will launch the *gatherInformation* method to retrieve trust information from three main sources (ref. step 4 in Figure 2-2). Firstly, the *gatherInformation* method collects information with respect to the status and number of resource and service offers published in the Catalogue. Secondly, it also gathers trust information from its dedicated trust database which contains historical information from previous trust service chains. Lastly, the method gets recommendations from the previous interactions published in the Data Lake by other 5GZORRO Platform Participants (ref. step 7 in Figure 2-2). Thus, this information will be utilised to perform a final trust score that will be forwarded to the SRSD.

After receiving all trust assessments from the list of offers, the SRSD application will generate a ranked list of offers based on certain intent priorities as well as the trust scores. Then, such a list of offers will be sent to the Intelligent Slice and Service Manager (ISSM). After that, the ISSM Optimizer (ISSM-O) will determine a cost-efficient allocation of resources and services available at the marketplace to implement the required slice or service. Hence, the ISSM-O should notify the final list of highest ranked product ordering to the ISSM Workflow Manager (ISSM-WFM). At this point, the ISSM-WFM will carry out multiple tasks, with the aid of other modules like the Network Slice and Service Orchestrator (NSSO), to configure and instantiate a slice. Once the slice is instantiated and active, the ISSM-WFM will notify the final decision to the 5G-TRMF (ref. step 15 in Figure 2-2). As a final step, the 5G-TRMF will launch a continuous data collection for the selected slice or service to start the entire trust lifecycle and update the trust score, in the case of some defined events or triggers are activated. As steps 16 and 17 depict, the Security Analysis Service (SAS) and the Intelligent SLA Monitoring and Breach Predictor (ISBP) are the two principal information sources to be considered. In this regard, the 5G-TRMF will launch threats to monitor in parallel different set of events.

In relation to the workflow updates, it has been required to review the previous 5G-TRMF Information Model presented in deliverables D4.1 [53] and D4.2 [1]. In particular, the updated information model contemplates new trust parameters to compute and update the trust score of services and resource providers and their product offers.

More specifically, to correctly manage the new trust parameters defined in the 5G-TRMF Information Model, multiple equations have been defined which are used to compute the 5GZORRO stakeholder trust score (see Appendix I). In particular, the 5G-TRMF leverages the well-known reputation model called *PeerTrust* [2] which is envisaged as a basis from which to design equations in line with the 5GZORRO ecosystem. Thus, these equations will be employed by one of the modules that make up the architecture of the 5G-TRMF presented in deliverables D4.1 [53] and D4.4 [19], the Trust Computation module. In this regard, all trust parameters, as well as equations contemplated for calculating trust scores, can be found in Appendix I. Additionally, new functionalities are described in Appendix I (sections 9.2.4 and 9.2.5) where a resilience mechanism against trust attack and two novel reward and punishment mechanisms have been respectively developed.

2.1.2. Prototype implementation

The 5G-TRMF designed in deliverables D4.1 [53] and D4.4 [19] included various modules: Information Gathering and Sharing, Trust Computation, Trust Storage, and Continuous Update. This final prototype covers, in terms of modules and interfaces, the whole 5G-TRMF.

Regarding the communication paradigm employed by the 5G-TRMF, it follows both a request/response paradigm and a publish/subscribe paradigm. The former is utilised by the 5GZORRO modules that need to launch the trust lifecycle, for instance, the SRSD, to retrieve historical trust information, recommendations as well as to send trust information about a stakeholder. The latter is used by the 5G-TRMF itself to, process new events or triggers. In the case of publish/subscribe communication paradigm, such a framework will utilise the common Kafka instance which will be shared with other 5GZORRO modules (intra- and cross-domain communication fabric).

The definition of the 5G-TRMF interfaces is available at the following GitHub page*:*

https://github.com/5GZORRO/5G-TRMF.

Concerning the virtualization technology used for the software package, this module is released as a Docker container to facilitate orchestration and delivery together with other 5GZORRO modules.

2.1.3. Functional tests

The following table describes several functional features which allow checking the proper behaviour of different modules and activities carried out by the 5G-TRMF. It should be noted that all the tests described in Table 2-1 are related both to the integration of the 5G-TRMF with other 5GZORRO modules and internal to the 5G-TRMF. For instance, the *Request Trust Score* test involves the SRSD which interacts with the 5G-TRMF to assess Product Offers (POs), the Start Data Collection test. Therefore, Table 2-1 covers the interaction between SRSD and 5G-TRMF to assess POs via the *Request Trust Score* test, the information retrieval by the 5G-TRMF (the *Start Data Collection* test*),* the information storage in the Data Lake and private database (the *Storage Trust Information* test) or the updating of trust through the SAS and ISBP components (the *Trigger Security Events* and *Trigger Breach Predictions* tests), among others.

Table 2-1: 5G-TRMF functional tests

Name	Description	Passed (Yes/No/Partially)
Trust Parameters	Verify whether the dictionary received as parameter complies with the format of the described trust information model	Yes
Start data collection	Verify that the 5G-TRMF can retrieve information from Data Lake, Catalogue, and its dedicated database	Yes

Request Trust Score	Verify that the 5G-TRMF can process a list of Product Offers sent by the SRSD and return back a list of trust scores.	Yes
Compute Trust Score	Verify that the 5G-TRMF can perform all steps necessary to assess the Pos.	Yes
Storage Trust Information	Verify that the 5G-TRMF can properly interact with its dedicated database and the Kafka topic.	Yes
Notify Final Selection	Verify that the 5G-TRMF can start the continuous update process about a Product Offer previously analysed.	Yes
Trigger security events	Verify that the 5G-TRMF can create a given threat to contact with the SAS and manage the received security network events.	Yes
Trigger Breach Predictions	Verify that the 5G-TRMF can create a given threat to contact with the ISBP and manage the messages related to breach predictions published in the Kafka Topic.	Yes

2.2. Trusted Execution Environment Security Management

The Trusted Execution Environment Security Management module focuses on the development of functionalities that allow 5GZORRO to protect their tenant service or application running in a computing node against a stakeholder with malicious intentions.

For that purpose, this module integrates commercial Trusted Execution Environments (TEEs) for the software components execution to enhance the security and trust of the software. In this regard, a hardware-based TEE approach has been adopted, by making use of Intel's SGX (Software Guard Extensions).

As previously mentioned in D4.1 [53] and D4.2 [1], Secure Linux Containers (SCONE) [3], a framework built on top of Intel's TEE solution, was the chosen technology to abstract specific implementation details of the secure enclave and to allow the deployment of critical services on SGX-enabled [4] nodes.

2.2.1. Main Functionalities

With the architecture definition of the 5GZORRO platform and integration and development of its components, the overall design and consequent functionalities of the TEE module have undergone some relevant changes. One core aspect was the integration of SCONE framework modules with the orchestration services, allowing the deployment of critical services on SGX-enabled nodes present in the marketplace.

Since 5GZORRO project mainly expects to support the integration of zero trust hardware platforms (TEE) as a root of trust for the monitoring of information and the establishment of end-to-end secure communications enabling critical workloads to go across different stakeholders, it was decided to concentrate efforts in achieving this specific goal.

This goal was achieved by focusing on converting existing critical components, avoiding a great restructuring of the actual architecture, while keeping the monitoring of the critical data as secure as possible. In specific, some workloads (such as Prometheus) required complex changes in their build processes to ensure the final binary was compliant with the SCONE framework requirements (specifically, ensuring the binary is a Position Independent Executable, PIE, by changing the build process). Therefore, the main functionality of the TEE module is the enhancement of the security and trustworthiness of the system by providing the execution of 5GZORRO services and components related to the monitoring of critical information in a TEE. SLA Monitoring, Intelligent SLA Breach Predictor (ISBP) and Monitoring Data Aggregation (MDA) have been primarily selected to be running under a TEE, to assure the aggregation, processing/computation integrity of SLA monitoring data and to guarantee that the detection and prediction of SLA violations occur in a safe environment.

Another functionality is the exposure, at a higher level, of the capabilities offered by the TEE platform. This functionality is provided by the TEE Management service. However, it's important to note that since most of the core functionalities needed to manage TEEs are provided by SCONE set of APIs and are not needed externally by other 5Gzorro components, they are not being directly exposed by the TEE Management service. In this regard, the TEE Management service mostly acts as a source of truth by exposing the capability to attest the components running under a TEE.

2.2.2. Prototype implementation

Several hardware solutions were explored to provide the needed SGX support, by considering the interconnectivity between the testbeds, practicality and cost. At this level, a solution that involved running the workloads directly on bare metal was favoured over the integration of VM's with SGX support (from OpenStack or a Cloud Service Provider). In this regard, Intel NUC was the selected platform to be used as an environment for TEE. This platform allows 5GZORRO to access a pool of compute resources whose underlying CPUs have built-in native SGX support.

As mentioned before, SCONE framework is used to enable TEE capabilities by allowing the deployment of application services on the SGX-enabled hardware.

The adopted SCONE framework for TEE is based on four main components that allow deploying an application in a secure environment:

- **Session:** Entails all security-relevant details of a SCONE application. It includes every aspect of the container ecosystem such as commands to be executed on images, secrets, volumes, and environment variables. Everything is then attested with the unique signature of the enclave which is authorized to retrieve the secrets from the CAS.
- **LAS (Local Attestation Service):** Service that runs locally, alongside the enclave, and the application that wants to access the secrets. This service is responsible to complete the application attestation and ensure that the hardware is SGX-enabled and share that information with CAS.
- **CAS (Configuration and Attestation Service):** Remote service which stores things like configurations, secrets and filesystem keys needed by the application in runtime, that were provided by the session. This service ensures that all secrets are protected from being visible by humans and are only visible inside the TEEs. These secrets and configurations will then be put in transit and shared with the application once the attestation takes place (when the application proves its integrity and authenticity).
- **Docker container:** The trusted application intended to run in an enclave that once attested by the LAS it will receive the configurations and secrets that are stored in the CAS.

Using these main four components, a stakeholder that wants to run an application in a secure environment can easily achieve that with a few steps:

- Update the application to run on top of SCONE cross-compiled docker images and mount the source code inside the container.
- Launch a new dedicated CAS or use any existing one.
- Launch LAS alongside the enclave.
- Get the enclave SCONE hash.
- Upload the session to CAS with the enclave SCONE hash and all security details.
- Run the secure application in an enclave.

As described in D4.2 [1], before the integration of TEE and SCONE with the selected 5GZORRO components, a proof of concept was prepared to validate this environment and tools. For this, a Docker container was deployed in a virtual machine with SGX-enabled capabilities in a Kubernetes cluster with two nodes, one with SGX-enabled capabilities from the cloud provider Azure [5] and another without those capabilities placed in an OpenStack instance. In this scenario (see Figure 2-3), it was validated that an application can run securely in a non-trusted shared environment (i.e., OpenStack, Cloud, on-prem), since only the application itself (with the hashes generated by CAS) can see its own security-relevant and runtime information. Every other service or other tenant will see encrypted information at runtime or at rest (for more details on the proof-of-concept refer to D4.2). The functional tests performed for this validation are presented in Table 2-2 in the following section.

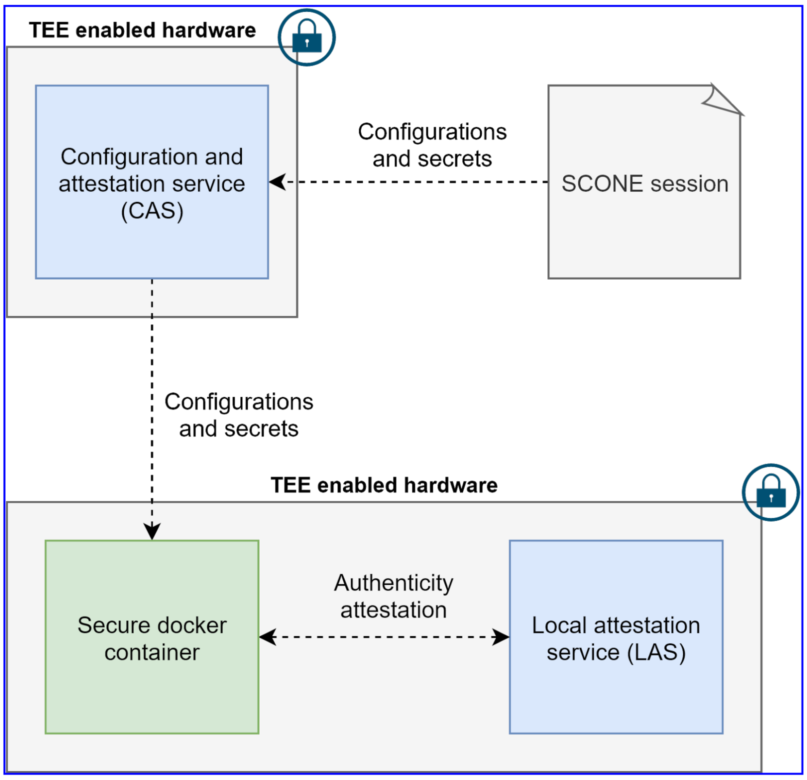

TEE enabled hardware
Configuration and attestation service (CAS)
Configurations and secrets
SCONE session
Configurations and secrets
TEE enabled hardware
Secure docker container
Authenticity attestation
Local attestation service (LAS)

Figure 2-3: TEE Proof of concept diagram

After this preliminary proof-of-concept, the implementation efforts have been focused in integrating three critical elements in 5GZORRO platform with TEE capabilities to improve their execution security: the Monitoring Data Aggregation (MDA), the SLA Monitoring and the Intelligent SLA Breach Prediction (ISBP).

For the MDA, as it is responsible for processing and aggregating data, doing those operations in a TEE provides the means to establish a high level of trust in that process, as SCONE can make the process tamper resistant. With respect to SLA Monitoring and Intelligent SLA Breach Prediction, SCONE's attestation mechanisms can ensure that the sensitive SLA monitoring and breach predictor computations or authenticity proofs for smart contracts can run inside a tamper proof environment where the private information will never be exposed to outside actors.

Additionally, the TEE Manager is also integrated in a TEE, so that a complete chain of trust for the metrics is established.

2.2.3. Functional tests

The steps and functional tests that have been carried out for validating the implementation of the proof-of-concept and to validate the correct integration and implementation of the TEEs for each of the selected 5Gzorro components are described in Table 2-2. Generally, the process involves the "*sconification*" of each component individually using SCONE framework, creating a CAS session, deploying the component in a local environment and attesting that it's running in SGX, integrating it with the testbed NUC, and finally testing the interaction of each deployed module with other components to validate and evaluate performance (Figure 2-4).

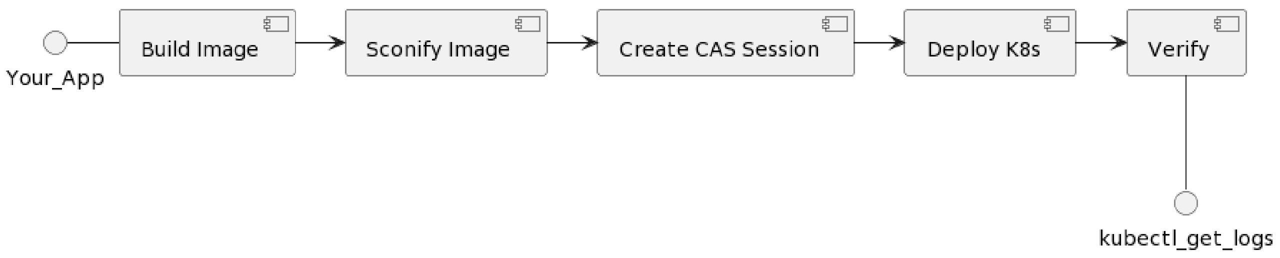


Figure 2-4: TEE functional testing flow

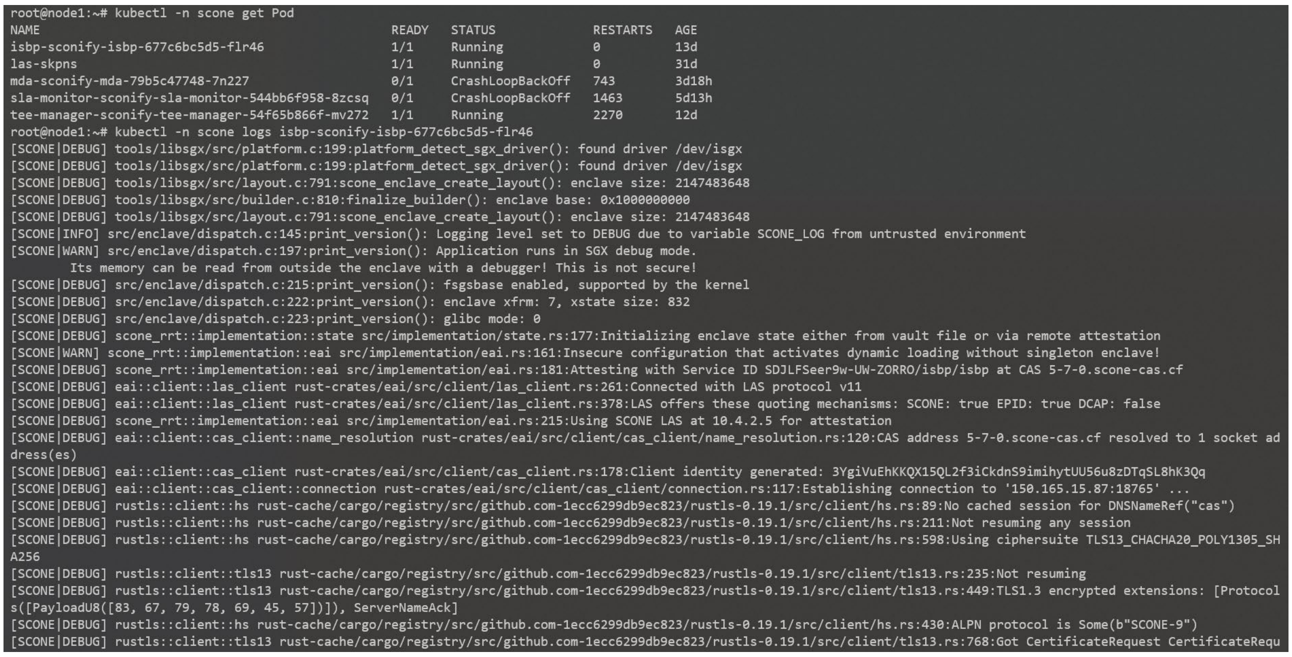
Note that the CAS used for the tests was provided publicly by SCONE framework. The tests performed specifically for the TEE Manager (API) are also displayed.

Table 2-2: Trusted execution environment security management functional tests

Name	Description	Passed (Yes/No/Partially)
Proof-of-concept: Create docker image (Sconification)	Create docker image on top of SCONE cross-compiler docker images	Yes
Proof-of-concept: Setup trusted node	Setup a node with SGX-enabled capabilities	Yes
Proof-of-concept: Instantiate LAS	Setup LAS service alongside the enclave	Yes
Proof-of-concept: Upload SCONE session	Upload SCONE session to CAS with the secrets that are intended to run on the docker container	Yes
Proof-of-concept: Instantiate docker container	Instantiate docker container in the trusted node	Yes
Proof-of-concept: Retrieve secrets	Retrieve secrets from the docker container that are passed by CAS once LAS attests its authenticity	Yes
Proof-of-concept: Validate docker container security	Check if the secrets that should be passed by CAS were not in the container environment and the running code was encrypted	Yes
Setup trusted node (locally)	Setup a local node with SGX-enabled capabilities	Yes
Setup trusted node (testbed NUC)	Setup testbed NUC with SGX-enabled capabilities	Yes
Instantiate LAS	Setup LAS service alongside the enclave (local environment and NUC)	Yes
Upload SCONE sessions	Upload SCONE sessions to CAS with the secrets that are intended to run on the docker container	Yes
MDA: Sconification	Create docker image on top of SCONE cross-compiler docker images	Yes
MDA: Instantiate docker container	Instantiate docker container in the trusted node (local environment and NUC)	Yes
SLA Monitoring: Sconification	Create docker image on top of SCONE cross-compiler docker images	Yes
SLA Monitoring: Instantiate docker container	Instantiate docker container in the trusted node (local environment and NUC)	Yes
ISBP: Sconification	Create docker image on top of SCONE cross-compiler docker images	Yes

ISBP: Instantiate docker container	Instantiate docker container in the trusted node (local environment and NUC)	Yes
TEE Manager: Sconification	Create docker image on top of SCONE cross-compiler docker images	Yes
TEE Manager: Instantiate docker container	Instantiate docker container in the trusted node (local environment and NUC)	Yes
TEE Manager: Get attestation information	Retrieve the attestation information of a component instantiated on a TEE	Yes

In Figure 2-5 we can see the TEE Manager, SLA Monitoring, MDA and ISBP modules deployed on the NUC. It can also be noticed the LAS service instantiated. Additionally, an example of logging on the ISBP running on the Intel SGX Based TEE is presented. The original application was changed to run on the Intel SGX using the SCONE framework.



```
root@node1:~# kubectl -n scone get Pod
NAME                                            READY   STATUS             RESTARTS   AGE
isbp-sconify-isbp-677c6bc5d5-flr46              1/1     Running            0          13d
las-skpns                                       1/1     Running            0          31d
mda-sconify-mda-79b5c47748-7n227                0/1     CrashLoopBackOff   743        3d18h
sla-monitor-sconify-sla-monitor-544bb6f958-8zcsq   0/1     CrashLoopBackOff   1463       5d13h
tee-manager-sconify-tee-manager-54f65b866f-mv272   1/1     Running            2270       12d
root@node1:~# kubectl -n scone logs isbp-sconify-isbp-677c6bc5d5-flr46
[SCONE|DEBUG] tools/libsgx/src/platform.c:199:platform_detect_sgx_driver(): found driver /dev/isgx
[SCONE|DEBUG] tools/libsgx/src/platform.c:199:platform_detect_sgx_driver(): found driver /dev/isgx
[SCONE|DEBUG] tools/libsgx/src/layout.c:791:scone_enclave_create_layout(): enclave size: 2147483648
[SCONE|DEBUG] tools/libsgx/src/builder.c:810:finalize_builder(): enclave base: 0x1000000000
[SCONE|DEBUG] tools/libsgx/src/layout.c:791:scone_enclave_create_layout(): enclave size: 2147483648
[SCONE|INFO] src/enclave/dispatch.c:145:print_version(): Logging level set to DEBUG due to variable SCONE_LOG from untrusted environment
[SCONE|WARN] src/enclave/dispatch.c:197:print_version(): Application runs in SGX debug mode.
        Its memory can be read from outside the enclave with a debugger! This is not secure!
[SCONE|DEBUG] src/enclave/dispatch.c:215:print_version(): fsgsbase enabled, supported by the kernel
[SCONE|DEBUG] src/enclave/dispatch.c:222:print_version(): enclave xfrm: 7, xstate size: 832
[SCONE|DEBUG] src/enclave/dispatch.c:223:print_version(): glibc mode: 0
[SCONE|DEBUG] scone_rrt::implementation::state src/implementation/state.rs:177:Initializing enclave state either from vault file or via remote attestation
[SCONE|WARN] scone_rrt::implementation::eai src/implementation/eai.rs:161:Insecure configuration that activates dynamic loading without singleton enclave!
[SCONE|DEBUG] scone_rrt::implementation::eai src/implementation/eai.rs:181:Attesting with Service ID SDJLFSeer9w-UW-ZORRO/isbp/isbp at CAS 5-7-0.scone-cas.cf
[SCONE|DEBUG] eai::client::las_client rust-crates/eai/src/client/las_client.rs:261:Connected with LAS protocol v11
[SCONE|DEBUG] eai::client::las_client rust-crates/eai/src/client/las_client.rs:378:LAS offers these quoting mechanisms: SCONE: true EPID: true DCAP: false
[SCONE|DEBUG] scone_rrt::implementation::eai src/implementation/eai.rs:215:Using SCONE LAS at 10.4.2.5 for attestation
[SCONE|DEBUG] eai::client::cas_client::name_resolution rust-crates/eai/src/client/cas_client/name_resolution.rs:120:CAS address 5-7-0.scone-cas.cf resolved to 1 socket ad
dress(es)
[SCONE|DEBUG] eai::client::cas_client rust-crates/eai/src/client/cas_client.rs:178:Client identity generated: 3YgiVuEhKKQX15QL2f3iCkdnS9imihytUU56u8zDTqSL8hK3Qq
[SCONE|DEBUG] eai::client::cas_client::connection rust-crates/eai/src/client/cas_client/connection.rs:117:Establishing connection to '150.165.15.87:18765' ...
[SCONE|DEBUG] rustls::client::hs rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/hs.rs:89:No cached session for DNSNameRef("cas")
[SCONE|DEBUG] rustls::client::hs rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/hs.rs:211:Not resuming any session
[SCONE|DEBUG] rustls::client::hs rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/hs.rs:598:Using ciphersuite TLS13_CHACHA20_POLY1305_SH
A256
[SCONE|DEBUG] rustls::client::tls13 rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/tls13.rs:235:Not resuming
[SCONE|DEBUG] rustls::client::tls13 rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/tls13.rs:449:TLS1.3 encrypted extensions: [Protocol
s([PayloadU8([83, 67, 79, 78, 69, 45, 57])]), ServerNameAck]
[SCONE|DEBUG] rustls::client::hs rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/hs.rs:430:ALPN protocol is Some(b"SCONE-9")
[SCONE|DEBUG] rustls::client::tls13 rust-cache/cargo/registry/src/github.com-1ecc6299db9ec823/rustls-0.19.1/src/client/tls13.rs:768:Got CertificateRequest CertificateRequ
```

Figure 2-5: Monitoring modules and LAS service running on the Intel SGX Based TEE

2.3. Security Analysis Service (SAS)

The Security Analysis Service, previously named Intra-domain Security, is the module that provide the security services in charge of performing network diagnostics in order not only to detect possible network vulnerabilities, attacks, or threats for Network Services inside each domain, but also to apply the required countermeasures for the mitigation of these adverse events.

2.3.1. Main Functionalities

The SAS applies security analysis mechanisms for the collection and monitoring of network service metrics to efficiently analyse the network traffic and detect possible network vulnerabilities or malicious behaviour.

The main functionality of the module is the mirroring of the network traffic to the Security Analytics VNF component for further security analysis, through a virtual Tap (vTAP) VNF. In fact, vTAP VNF, which is responsible for connecting the different Network Services of a Network Slice and also mirror the respective traffic, uses a virtual TAP configuration [4] to capture a copy of the data flowing between the deployed Network Services of a certain domain. This happens, for example, through the presence of a virtual switch on the virtual links between the respective services. In addition, the VNFs (Security Analytics, vTAP) are deployed alongside with the Network Slice requested from the stakeholders, as they are responsible for the Network Services hosted on the specific Slice. In addition, if security threats or malicious activity have been detected from the network analysis performed inside the Security Analytics VNF Security Analysis Service, and especially the Security Analytics VNF module, can apply the necessary countermeasures and mitigation procedures to the Network Slice. These actions are mainly a set of traffic rules, configurations of firewall policies or even the entire isolation of the evicted user’s VNF by blocking its communication link with the other components of the Network Slice.

To this need, the Security Analysis Service can be deployed as a set of Security VNFs (vTAP, Security Analytics & ELK) in conjunction with the Network Services VNFs requested from 5GZORRO platform consumers in the same Network Slice Description according to MANO specification. In addition, the Security Analytics VNF can be activated and start analysing the network traffic of the Network Services while further integrated with the ELK VNF for storing the diagnostics, via automated and on demand mechanisms. Note that the vTAP and Security Analytics VNFs are deployed per slices, however, the ELK VNF is set up per domain, so the ELK VNF will store all the metrics provided by multiple networks slice under the same domain. For the deployment of Network Services, the ETSI OSM MANO orchestration framework [56] has been used as it also provides the necessary tools for performing the respective automated and on demand configurations (day-1 or day-2 configurations). The high-level architectural diagram of this module is shown in Figure 2-6.

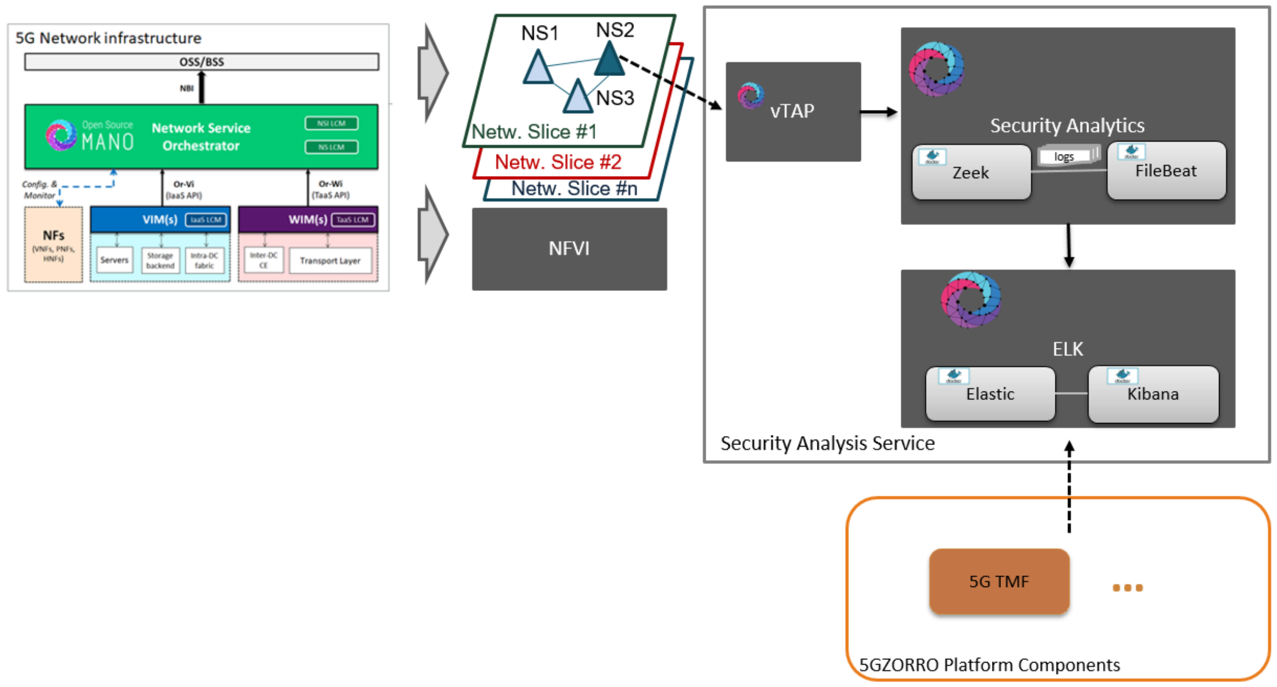


Figure 2-6: Security Analysis Service module architecture

2.3.2. Prototype implementation

In the final implementation, the Security Analytics VNF component encapsulates the functionality and the support of the Zeek platform [6] integrated with a Filebeat [7] instance for data collection. In addition, this data can be further sent to the ELK VNF component that encapsulates both an Elasticsearch [8] for storing the data and a Kibana for the visualization of the statistics [9]. Zeek is used as the network security monitoring tool which is responsible for the analysis of network events that will be passed to Filebeat for the data transformation and then stored/analysed within Elasticsearch platform. Finally, data is sent to Kibana platform for visualization. Stored network data analysis is supported as well.

To deliver the JSON text based Zeek logs to Elasticsearch, Filebeat is used. It reads the Zeek log files and delivers them to Elasticsearch. When providing data to Elasticsearch, a pipeline is specified for all events that are inserted inside the Elasticsearch index. After that, all the Zeek logs can be accessed through Kibana dashboard.

Elasticsearch is a distributed, RESTful search and analytics engine capable of storing data and searching it in near real time. Kibana is a browser-based analytics and search dashboard for Elasticsearch. All the software modules encapsulated inside the Security Analytics VNF as well as the instances of Elasticsearch and Kibana inside ELK VNF are deployed as Docker containers in a well-defined Docker network configuration, while the respective Security Analytics, ELK and vTap VNFs use the appropriate Openstack images.

Finally, these services are implemented as a set of on demand and automated functionalities to perform a variety of actions upon the Security Analysis Service module. These actions can be applied remotely by the user from the orchestration frameworks as on demand configurations to enable the SAS functionalities and customize their behaviour.

The implementation follows blacktop/docker-zeek project as published in:

https://github.com/blacktop/docker-zeek.

The final implementation of the SAS module can be found in the 5GZORRO GitHub repository at:

https://github.com/5GZORRO/intrasecurity .

2.3.3. Functional tests

To validate SAS functionalities, we will describe the communication flow and how the respective components interact.

At the Security Analytics VNF, the Zeek container starts analysing the network traffic while the obtained statistics in the form of log files are stored to a specific directory of the Security Analytics VNF. This directory is shared also to the Filebeat container which can retrieve the log files and transforming them. In addition, the Filebeat container has a direct link with the Elasticsearch container that is placed inside the ELK VNF for the transformed statistics to be stored. For this reason, the Elasticsearch container can be found at the IP of the ELK VNF and at the exposed port 9200. Finally, the Kibana container inside the ELK VNF has also a link with Elasticsearch container by applying only the name “elasticsearch” as the two containers are included in the same VNF, namely the ELK.

Our software tests, Table 2-3, refer to the successful deployment of the various containers needed for the module and the successful data sharing communication between them.

Table 2-3: Intra-domain functional tests

Name	Description	Passed (Yes/No/Partially)
1. Zeek service	Start traffic analysis and obtains statistics. Links with Filebeat by sharing the same directory	YES
2.Filebeat service	Transformation of Zeek’s statistics. Links with Elasticsearch which is listening at the IP_ELK_VF:9200	YES
3.Elastisearch service	Statistics from Filebeat stored.	YES
4.Kibana service	Visualization of statistics. Links with Elasticsearch by applying the respective container’s name	YES

2.4. **VPN-as-a-Service**

The VPN-as-a-Service (VPNaaS), previously called Inter-domain Security module, oversees the establishment of secure and reliable connections between different domains within 5GZORRO. Thus, this module offers on-demand VPN-as-a-Service, so that the interconnection between domains is done automatically without any previous configuration. For this purpose, it integrates the key derivation necessary to establish the secure connection with the Identity & Permissions Management module (see D3.2 [49]), obtaining the cryptographic keys and information from it. In addition, the VPNaaS interacts with the Network Service Mesh Manager (NSMM) which enables the stitching of services in different domains, creating the desired resources in the operator to the exposed services to the outside world and securing the communication between them through the VPNaaS modules.

2.4.1. Main Functionalities

The VPNaaS offers to other 5GZORRO Platform modules a specific interface to interact with the underlying gateway to manage the lifecycle of an on-demand tunnel establishment. Thus, the main functionalities of the VPNaaS are reported below:

- Configuration and deployment in the network gateways of each domain, so that they act as an intermediary between the end entities and the external domain.
- Exposing capabilities to deal with encrypted payload forwarded by the Identity & Permissions Management module.
- Securing only the IP of the Virtualized Infrastructure Manager to redirect the network traffic, avoiding protecting all traffic passing for the NSMM.
- Verifying the identity of the other gateway through sharing its unique identifier named Decentralized Identifier (DID) [51], public key and timestamp with the Identity & Permissions Management module.

2.4.2. Prototype implementation

In the final prototype, a full version of the entire set of interfaces, defined in D4.4 [19], has been implemented to verify its suitability and to test the desired functionalities. The functionalities which have been implemented for this release are a) Client-to-gateway and gateway-to-gateway authentication; b) Integration with the Identity and Permission manager module for key management and identification; c) Integration with the NSMM for the configuring of the gateway and deployment.

The prototype is implemented following a request/response setup based on a Restful API. The definition of the interfaces is available as Swagger file at:

https://5gzorro.github.io/VPNaaS/.

The prototype has been implemented using Python 3, with the following additional packet requirements:

- *flask* and *flask_restful*, for the Restful API setup [10]
- *Gevent*, for networking management [11]
- *Werkzeug*, for the WSGI server [12]

For the VPN service setup and management, WireGuard VPN [13] has been leveraged. WireGuard is an open-source software implementing VPNs with state-of-the-art cryptography and an easy setup. This tool aims to provide faster connections than previous solutions such as IPSec [14] or OpenVPN [15]. WireGuard works in a client-server setup in which VPN connections are added using new network interfaces. This configuration enables to have different VPN connections to different domains in the same client. The only configuration needed is to define which IP range is redirected to each WireGuard interface.

To enable the automated installation and configuration of WireGuard, the module includes an interface named "launch" which is tasked to deploy WireGuard in the target machine and to configure the required network properties to enable traffic forwarding. Additional packets installed during setup are linux-kernel-headers and openresolv.

The 5GZORRO GitHub repository of the module is available at:

https://github.com/5GZORRO/VPNaaS.

The current prototype is released in the form of a Virtual Machine (VM) as the module is intended to be deployed in gateways (which are rarely deployed as containers). However, it might be possible to deploy the module in containers if the required dependencies are fulfilled.

2.4.3. Functional tests

The functional tests (see Table 2-4) carried out for this module are mainly related to the verification of the correct functioning of the implemented methods. In particular, Table 2-4 showcases the interaction between the NSMM and the VPNaaS to establish or terminate a secure connection (the *Connection establishment* and *Connection deletion* tests) or the interaction between the VPNaaS and the Id&P modules to verify the public key and the DID.

Table 2-4: VPNaaS functional tests

Name	Description	Passed (Yes/No/Partially)
Module setup	The module can be automatically installed and configured.	Yes
Client addition	A new client is connected to the VPN, obtaining a client IP in the VPN.	Yes
Client removal	A client is disconnected, removing its IP from the known list.	Yes
Connection establishment	The automated connection establishment between two VMs running in different subnets is tested.	Yes
Connection deletion	In the current design we use the public key to search for the client to be removed.	Yes
Key verification	The gateway checks the identity of the other gateway through sharing its DID, pubkey and timestamp	Yes

3. Intelligent and Automated Slice & Service Management

The 5GZORRO Intelligent and Automated Slice and Service Management (ISSM) sub-system focuses on automated management of secured cross-domain slices and services within them. The ISSM is thus responsible for enforcing business transactions both at the system level by interacting with the Network Slice and Service Orchestrator (NSSO) and with alternative slicing technologies that might be developed in the future, as well as by managing business transaction contexts across the entire 5GZORRO platform allowing a principled, repeatable, auditable, and trustworthy interaction among the multiple components of the platform to realize a specific business flow.

The 5GZORRO Intelligent and Automated Slice & Service Management sub-system is comprised of three modules:

- ISSM Workflow Manager: responsible for executing the orchestration workflow
- ISSM-MEC: handles deployment of 5G slices natively on Kubernetes (K8s) [55] in a multi-cluster environment
- ISSM-Optimizer: optimizes cost-efficiency and cost-trustworthiness trade-offs of network services and slices

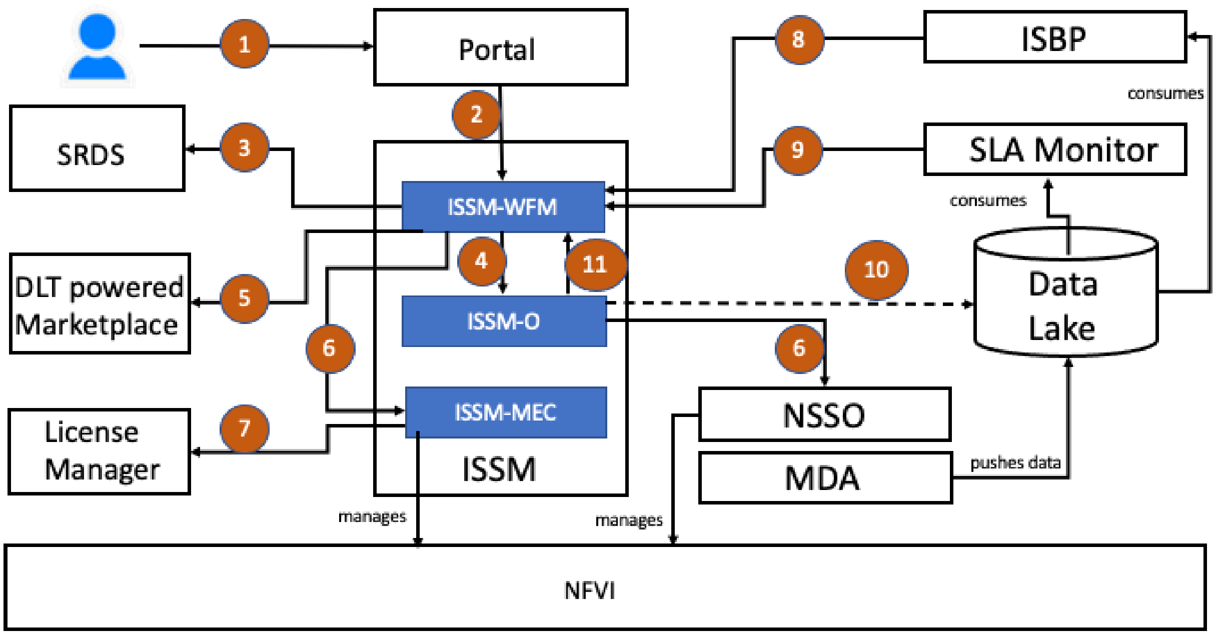


Figure 3-1: ISSM interaction with the rest of the 5GZORRO platform components

Figure 3-1 provides a high-level description of how ISSM interacts with the rest of the 5GZORRO platform. The numbers on the arrows signifying the interactions do not necessarily correspond to sequential steps, even though they are suggestive of a typical business flow.

1. An administrator persona interacts with ISSM through the 5GZORRO portal. Our implementation supports selection of single and multiple Product Offers in the context of a single instantiate and scale-out operations for VNF, edge resources, slices, and services as well as higher level intents that

only indicate desired properties for slices and services, such as geography, cost, and QoS letting ISSM to figure out the rest of the lower-level details and perform optimization along the way.

2. The portal calls REST API that we implemented as NBI for ISSM-WFM to pass it an array of product offers and/or a high-level intent that should be used to instantiate or scale-out a slice or a service.
3. If necessary, for example in case of a high-level intent that indicates that a customer is looking for a Content Distribution Network (CDN) cache service with specific QoS and geography coverage, ISSM-WFM makes a REST call to SRSD to identify suitable product offer candidates. We do not show how SRSD interacts with the Catalogue and other components, since this is out of scope for describing ISSM interactions with the platform.
4. ISSM-WFM performs a REST call on ISSM-O (Optimizer) to optimize the product offers selection after the search space has been narrowed down by SRDS.
5. After ISSM-O determines the final selection of the product offers, ISSM-WFM interacts with the Marketplace to acquire these resources.
6. Once the resources required for the service instantiation or scale-out have been acquired, ISSM-WFM instantiates or scales out either via NSSO or ISSM-MEC. Both serve as the technical level orchestrators. More technical level orchestrators can be added to the ISSM ecosystem in the future.
7. ISSM-MEC contacts License Manager to validate instantiation.
8. At any asynchronous point in time, SLA Monitor might send notifications about an actual SLA breach, to ISSM-WFM via the interdomain communication fabric (Kafka) to trigger a preinstalled mitigation flow. Currently, scale-out mitigation flow is implemented.
9. At any asynchronous point in time, ISBP might send notifications about an actual SLA breach, to ISSM-WFM via the interdomain communication fabric (Kafka) to trigger a preinstalled mitigation flow. Currently, scale-out mitigation flow is implemented. For both interactions 9 and 10, ISSM-WFM might run the sequence explained above in interactions 3 – 7 to optimize its scale-out step.
10. ISSM-O can continuously pull data from the operational data lake to populate the internal mathematical optimization models
11. ISSM-O can call on ISSM-WFM requiring re-optimization of the current resource allocation and product orders.

It should be noted that while Figure 3-1 depicts ISSM components as being centralized, they are only logically centralized for ISSM-WFM and ISSM-MEC and are, in fact, distributed and symmetric. ISSM-O is a centralized platform component, because of its role as an inter-domain planner.

3.1. ISSM Workflow Manager (ISSM-WFM)

The ISSM Workflow Manager (ISSM-WFM) executes orchestration workflows in a context of a business transaction, such as extending a slice across a second domain in cooperation with the Network Slice and Service Orchestration.

3.1.1. Main Functionalities

In deliverable D2.4 [16], updated orchestration workflows have been presented which better capture the intended behaviours in scenarios such as cross-domain slice establishment, scale-out and optimization. Furthermore, in D2.4 updates have been provided about how ISSM-WFM integrates with the rest of the functional 5GZORRO architecture. The new design of ISSM-WFM is fully distributed. This is shown in Figure 3-2. In this distributed architecture, each domain locally installs ISSM API server (implemented as K8s Service), Event Bus (implemented by Kafka), Event Mediator (implemented as Argo Sensor), and a set of Workflow

Repository (implemented as Argo Workflow Templates). Initially, the set of workflow templates contains only one generic template, "Orchestrate". This template implements a generic Argo flow that interacts with NBI of a technical orchestrator in that domain. Currently supported technical orchestrators are "Dummy", "NSSO", and "ISSM-MEC". The template can be extended to support additional technical orchestrators in the future.

Other workflow templates are being added dynamically when Product Offers are acquired on the marketplace.

A typical ISSM-WFM workflow would start in a specific domain (for the sake of discussion and without losing generality, let us assume that a workflow starts in Domain A triggered by a Requestor), which can be either Portal or some automated functionality, such as Intelligent SLA Breach Predictor, SLA Monitor, ISSM-O or any other future manual or automated component being part of zero touch slice management and orchestration cycle. The request is performed via ISSM-WFM API server.

The ISSM-WFM API server pre-processes the request, creates a business transaction specification, and publishes it on the Event Bus. An Event Mediator receives it from the bus and triggers an appropriate workflow that was previously onboarded to the Workflow Repository of the domain as explained in the next subsection. The local orchestration workflow starts executing. The steps can span multiple components of the 5GZORRO platform. At some step of the workflow running in Domain A, an orchestration (or any other logic) sub-workflow might be required to be executed in Domain B. To that end, the workflow step in Domain A publishes a message (containing parameters and an entry point for a workflow in Domain B). Event Mediator of Domain B is subscribed on the topic dedicated to this domain in the cross-domain Event Bus. It receives the message and triggers an appropriate workflow in Domain B. At some point in the workflow of Domain B, the control might have to be passed back to Domain A. Possibly also status and variables must be passed back. To that end, the workflow of Domain B publishes a message on the cross-domain Event Bus. This might trigger an optional "stitching workflow" in the cross-domain Workflow Engine, after which control will be passed to Domain A by publishing a message on a dedicated topic for Domain A. The Event Mediator of Domain A receives the message from the cross-domain bus and continues with the Domain A workflow. This way, control can be passed back and forth arbitrary number of times across different per-domain ISSM-WFM components with support for "stitching" and synchronization by the cross-domain ISSM-WFM component and arbitrarily complex business level orchestration flows can be developed by the 5GZORRO platform developer.

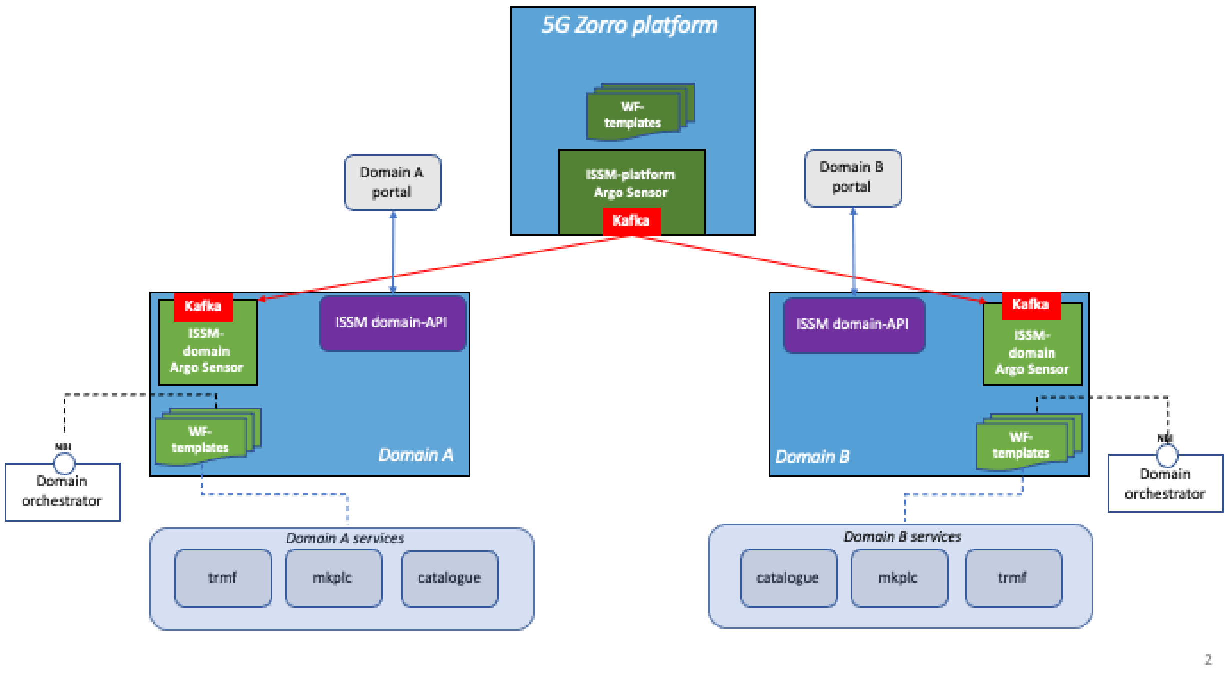


Figure 3-2: ISSM-WFM Distributed Architecture

3.1.2. Prototype implementation

The final ISSM-WFM prototype can be found on the 5GZORRO GitHub at:

https://github.com/5GZORRO/issm/.

The final prototype supports all three use cases and provides a methodology for developing new flows.

We now highlight the ISSM-WFM implementation with respect to the lifecycle of orchestrated service, VNF, slice, or any other resource supported by the 5GZORRO information model. The GitHub repository linked above can be used to retrieve further details.

The 5GZORRO methodology assumes that complex custom services can be dynamically created out of multiple Product Offers. These custom services require custom orchestration flows. In general while many steps of orchestration flow are recurring across flows (e.g., product offers discovery, product offers optimization, product offers acquisition, passing parameters to a technical orchestrator, polling technical orchestrator, sending/receiving SLA breach event for previously instantiated services, etc.), there are typically some service-specific steps, dependencies, and parameters that are idiosyncratic to lifecycle management operations of a service, such as “instantiate” and “scale-out”.

Indeed, “instantiate” and “scale-out” have different semantics across different use cases, because the services considered in the use case can be radically different.

To handle this complexity in a generic way, we have implemented the following two step methodology:

- Implement service-specific orchestrator plugin: Product Offer developer should implement an Argo workflow template that captures the Product Offer specific orchestration. This template captures lifecycle operations for this Product Offer. Implementation-wise, this template is a YAML file. Examples corresponding to the 5GZORRO use cases can be found on the 5GZORRO GitHub repository at: https://github.com/5GZORRO/issm/tree/master/snfvo.
- Connect service-specific orchestrator plugin to Domain Event Sensor (i.e., to the Argo Kafka sensor of the domain): in the current prototype implementation, this is done by applying the

https://github.com/5GZORRO/issm/blob/master/apply-domain.sh script. We refer to this *operation as workflow onboarding.*

From this point on, ISSM-WFM is enriched to orchestrate Product Orders that correspond to this Product Offer. This way ISSM-WFM is extensible and evolvable to future product offers.

3.1.3. Functional tests

The functional tests to validate ISSM-WFM API with respect to the Workflow 3-6 reported in D2.3 [17] are presented in Table 3-1. The tests have been performed using mock-up APIs for all involved components. The ISSM-WFM APIs have also been tested for the workflows involving the actual components, such as NSSO, and Data Lake, as described in section 5.2.

Table 3-1: ISSM-WFM functional tests

Name	Description	Passed (Yes/No/Partially)
Create flow	Create a new business flow	Yes
Execute flow	Execute a new business flow	Yes
Get progress	Get progress of a business flow using web hook	Yes
Delete flow	Remove a business flow from ISSM-WFM	Yes
Continuous progress	Observe a progress of a flow with GUI	Yes

3.2. ISSM-MEC

The ISSM-MEC Manager is a cross-domain component that realizes control plane of the Cloud-Native MEC Platform (CNMP). Architecturally, using the terminology of ETSI MEC [18], the ISSM MEC Manager belongs to the *MEC System*, i.e., the control plane of MEC, and CNMP represents *MEC Hosts*.

The key idea behind ISSM MEC Manager is to use Kubernetes (k8s) [55] as the orchestrator for CNFs that are hosted by MEC to extend the cloud native experience also to the control plane itself rather than treat only as NFVI that should be externally orchestrated.

The Cloud Native MEC Platform (CNMP) is a per-domain component. External to ISSM, it was designed to represent and experiment with the emerging cloud native MEC environment, to be managed/federated under 5GZORRO. This architectural component was conceived during the first year of the project, following the industry trend for cloud native transformation. In 5GZORRO its role is twofold: first, to demonstrate that 5GZORRO platform can integrate with cloud native k8s based MEC and, second, to extend 5GZORRO use cases validation to this emerging type of platform.

Final design updates for both ISSM-WFM and ISSM-MEC are described in D4.4 [19].

3.2.1. Prototype Implementation

Figure 3-3 depicts the final implementation of ISSM-MEC and MEC CNMP with application to free5GC open source 5G project [20]. The details of the implementation and operational prototype can be found on the 5GZORRO GitHub at:

https://github.com/5GZORRO/issm-mec-cnmp.

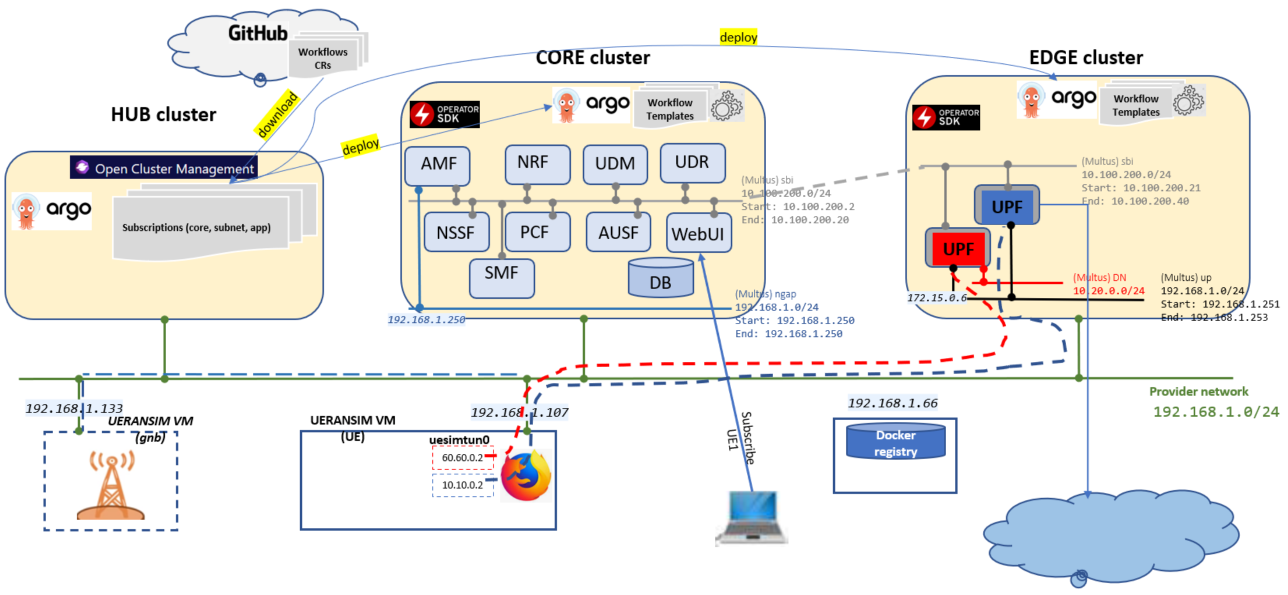


Figure 3-3: Reference Implementation of ISSM-MEC Manager and MEC Cloud Native Platform with support for slice deployment for fully kubernetized free5GC Core and UPF.

3.3. ISSM Optimizer (ISSM-O)

The ISSM Optimizer (ISSM-O) is a cross domain component that optimizes cost-efficiency and cost-trustworthiness trade-offs of network services and slices required to be created in a context of a specific business transaction subject to constraints such as security and service area. ISSM-O continuously optimizes services and slices that have been already set up in previous transaction flow executions.

The main idea behind ISSM-O is to optimize and select the best resource offers from the Marketplace, given a specific objective. ISSM-O closely works with the ISSM-WFM, where it gets the physical resource offers acquired from the interactions between ISSM-WFM and SRSD components, together with slice topology received from the portal. After that, the ISSM-O components performs a set of optimization processes to reach a solution to embed the service requested onto the physical resource offers.

The final design of the ISSM-O can be found in D4.4 [19].

3.3.1. Main Functionalities

The final updates regarding optimization flows were provided in D2.4 [16], which describes scenarios such as optimization for the establishment of a slice request and optimization for the scenario of scaling of an already existing slice.

As presented in D2.4 [16], ISSM-O deals with NP-hard problems, which is to embed logical networks on top of shared substrate network across all the technological domains. In this regard, we have studied the problem of service embedding and resource allocation, and how to implement it in the ISSM-O component of 5GZORRO. The main issue with this joint problem of service embedding and resource allocation is that it becomes interactable when the problem size grows, meaning more nodes are added to the network or more services are issued to be deployed in the network. The state-of-the-art works in the domain mainly seek individual embedding of service instances (i.e., virtual networks) with every Virtual Network Function (VNF) instance and every individual forwarding path defined when a user session occurs. On the contrary, our proposed model is Fluid. Our heuristic solution comprises two phases: (1) slice capacity planning using Linear Programming (LP) and (2) greedy Protocol Data Unit (PDU) session allocation when sessions are initiated by the slice/service consumer. Using this technique, we can now scale to extremely big networks with thousands of users making service requests.

Concerning the first phase of our proposed method, which is to plan the capacity in the network using LP techniques, we use mathematical solvers to reach an optimal solution to the problem. There are many solvers out there than can be employed for solving such a problem, but each have their own characteristics and a set of limitations. After evaluating several possibilities for open-source tools, we have selected PuLP as a tool to generate general Linear Program implementations that can then be given as input to different backend solvers. Our main goal in the ISSM-O is to make it compatible with different optimization solvers, where, based on the requests from the customer and the time required to solve a model, the backend solver can change.

3.3.2. Prototype implementation

The final prototype implementation of the ISSM-O captures the entire interfaces needs to interact with the ISSM-WFM. ISSM-O's interaction with the ISSM-WFM happens through an Apache Kafka message broker, where it listens to a specific topic to receive the requests, perform the optimization process, and then publish the results on a specific topic. The functionality of the ISSM-O has been tested for the scenarios of optimization for slice establishment and slice scaling.

The prototype implementation of the ISSM-O component is done using Python 3, with the below packages employed in the implementation:

- PuLP for LP modelling in Python
- COIN-OR
- Numpy
- Pandas

As mentioned earlier, we have used PuLP as the LP modeler to generate optimization models that can used by different solvers. The main appealing feature of PuLP is that it is essentially capable of generating Mathematical Programming System (MPS) or LP files and call GLPK [23], COIN-OR CLP/CBC [24], CPLEX, GUROBI, MOSEK [25], XPRESS [26], CHOCO [27], MIPCL [28], SCIP [29] to solve linear programming problems. Our implementation of ISSM-O can be easily tuned to employ any of the mentioned solvers to be used for the optimization.

The 5GZORRO GitHub repository of the module is available at:

https://github.com/5GZORRO/issm-optimizer.

4. MANO and Slicing Enhancements

This section describes the sub-system of the 5GZORRO Platform devoted to the Management and orchestration of services and slices. From an architectural point of view, the set of services offered by the orchestration sub-system is mainly oriented to the internal consumption by other components in the platform, rather than by the 5GZORRO stakeholders themselves. In this sense, such set of modules offers functionalities related to resource management and monitoring, e-licensing management as well as to control and network slice and service orchestration within a stakeholder's domain. These functionalities are available towards the upper and more abstracted layers of the 5GZORRO platform, such as the Intelligent Slice and Service Management, which exploits them for building service and slices based on an intelligent composition of the available resources. With respect to what already reported in D4.2 [1], the set of modules composing the MANO enhancements introduces a new component, the Network Service Mash Manager (NSMM), that provides the capability of properly configuring the virtual infrastructure networks and specific service (see VPNaaS in section 2.4) to establish secure connections in a context of cross-domain end-to-end network slices.

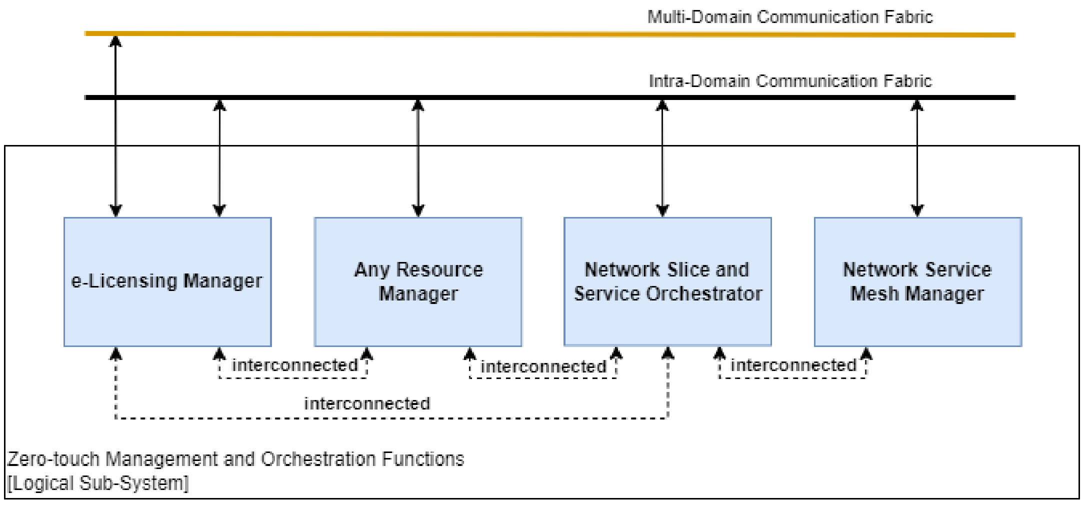

Multi-Domain Communication Fabric
Intra-Domain Communication Fabric
e-Licensing Manager
Any Resource Manager
Network Slice and Service Orchestrator
Network Service Mesh Manager
interconnected
interconnected
interconnected
interconnected
Zero-touch Management and Orchestration Functions [Logical Sub-System]

Figure 4-1: Mano and Slicing enhancements modules in 5GZORRO Platform

Figure 4-1 shows the modules part of the MANO that belongs to the Zero-touch Management and Orchestration logical sub-system of 5GZORRO platform. The dashed arrows indicate the explicit interactions between the modules.

4.1. Any Resource Manager

The Any Resource Manager (xRM) is a module in the 5GZORRO Platform, available in each stakeholder domain, that directly interacts with the underlying 5G Virtualized Platform, offering, towards the upper layer applications, a set of services related to the resources monitoring and management, including a direct support to the 5GZORRO Resource and Service Offering Catalogue (RSOC) through a translation service for the on-demand translation of technical descriptors (virtual network function resources, network services, radio resources, spectrum resource, edge resources, cloud resources, slice services) into proper resource and service models defined by the TM Forum [30].

4.1.1. Main Functionalities

The xRM is designed to be a multi-container application, with the aim to provide the high level of flexibility required to immediately tackle any changes or updates in both resource composition and in the underlying 5G virtualized infrastructure.
The main functionalities the xRM offers are:

- Proxy Interface for the underlying 5GZORRO Virtualized Platform leveraging an authentication system based on API key, e.g., Monitoring Data Aggregator (MDA) and E-Licensing modules exploit the xRM interface to interact with components of the 5GZORRO Virtualization Platform like NFV Orchestrator (e.g., ETSI OSM), VIM, Radio, and Network controllers.
- On-Demand translation of technical resource and service descriptors into TM Forum resource and service models and onboarding of these assets on the RSOC
- Proxy Interface to retrieve VNF resources, network services, radio resources, spectrum resource, edge resources, cloud resources, slice services from the underlying components of the 5GZORRO Virtualization Platform (i.e., 5G-Catalogue for VNFD and NSD, Radio Controller for Radio and Spectrum, Slice Manager for Edge, Cloud and Slice).

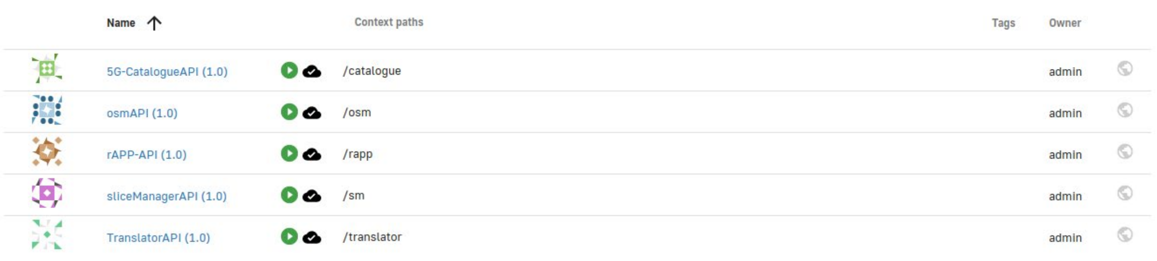
The final architectural design of the xRM was reported in D4.4 [19].

4.1.2. Prototype implementation

xRM is a module constituted by a set of components; each of these components expose a REST Interface to access the capabilities of the module itself, in details:

- 5G-Catalogue: Retrieves the VNF and NS descriptors from the underlying NFVO to make them available for the translation into TM Forum resource and service models and to enable the 5GZORRO Portal to display these kinds of resources. The 5G-Catalogue is a Java Spring Boot Application.
- Apache Kafka: Message Broker used internally by the 5G-Catalogue services; it is not exposed externally.
- Resource Definition Translator: A Java Spring Boot Application to enable the translation of assets, previously specified, from the 5GZORRO Portal. The Resource Definition Translator exposes a REST Northbound Interface with a set of POST requests to perform the translation of the specified assets. The Translator fetches the asset to be translated from the 5GZORRO components that hold that specific resource (e.g., 5G-Catalogue for VNFD and NSD, Radio Controller for Radio and Spectrum, ecc…); these are the same components backed by the xRM.

To aggregate and compose the REST interfaces of the components that compose the xRM, listed above, and, at the same time, to create new REST endpoints, a tool called Gravitee [31] was used. Gravitee is an API Gateway that permits to manage, create, and secure APIs. Gravitee was used to secure the REST API exposed by the 5G-Catalogue and the Resource Definition Translator: two sets of APIs were created using Gravitee to wrap the API exposed by the Spring Boot Applications. A module that wants to consume the API defined by Gravitee needs to subscribe to the API Application created within Gravitee itself and then use the released API Key in each subsequent HTTP REST request to access the capabilities of the wrapped modules. The same approach is used to proxy also other components of the 5GZORRO Virtualization Platform, in this case, APIs, for the NFV Orchestrator (ETSI OSM), Radio Controller and Slice Manager were defined in order to enable the 5GZORRO modules to exploit the underlying infrastructure (i.e., with the 5GZORRO Virtualized Platform) without directly interacting with it. Figure 4-2 depicts the API defined with Gravitee.



Name		Context paths	Tags	Owner
5G-CatalogueAPI (1.0)		/catalogue		admin
osmAPI (1.0)		/osm		admin
rAPP-API (1.0)		/rapp		admin
sliceManagerAPI (1.0)		/sm		admin
TranslatorAPI (1.0)		/translator		admin

Figure 4-2: Gravitee API

Figure 4-3 depicts how the defined APIs are mapped to the components that form the xRM.

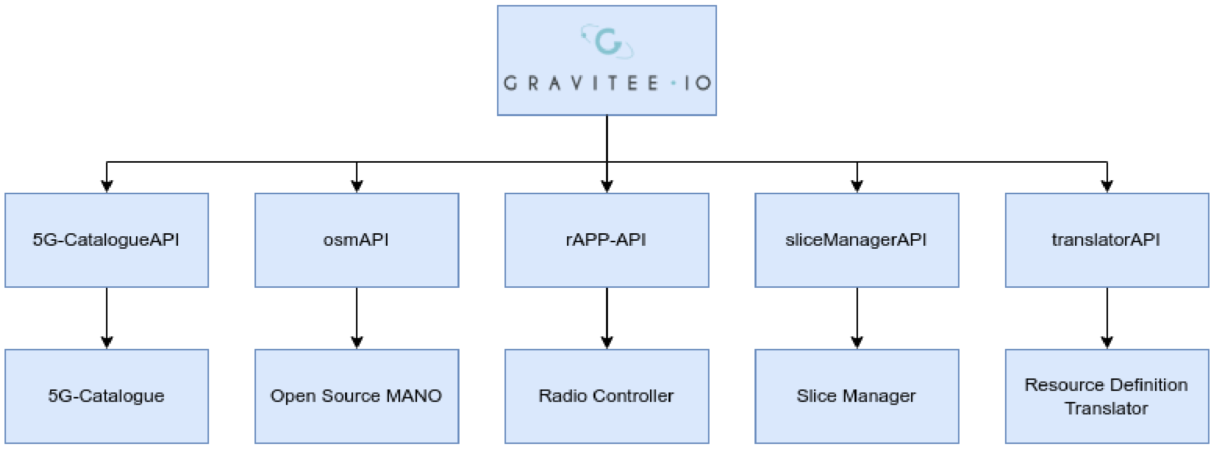
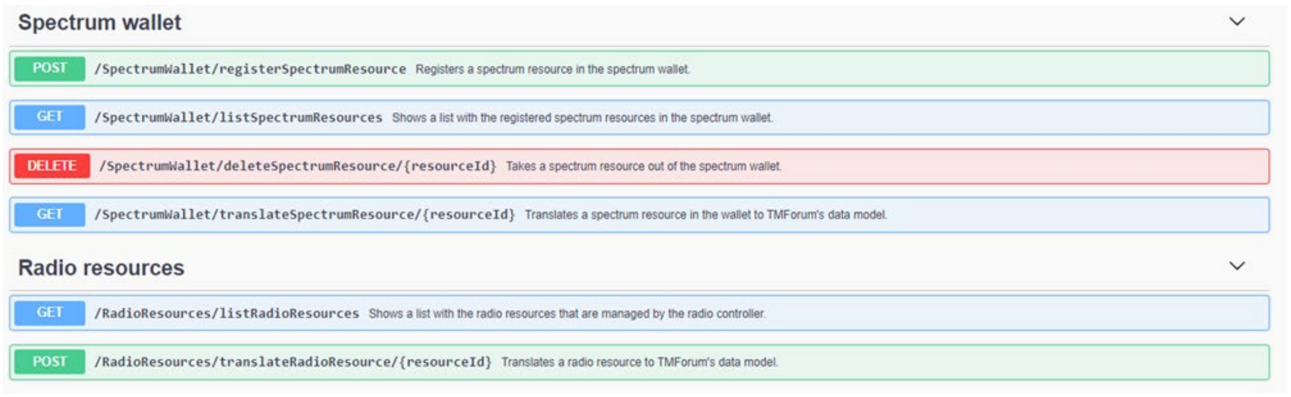


Figure 4-3: Gravitee Gateway and the defined APIs and the wrapped components

Each Gravitee API works as a proxy to the northbound interface of the backed 5GZORRO Platform component so that, as mentioned above, the upper layer components do not have to directly interact with underlying components of the 5GZORRO Virtualization Platform (i.e., directly interact with the correspondent northbound interface), and implement an authentication mechanism for the underlying component.

The rAPP is a containerized application (rAPP) that serves as a proxy to i2CAT's RAN Controller for managing spectrum resources and exposing spectrum and radio resources to the xRM. To this aim, the rAPP exposes a RESTful API, which is shown in Figure 4-4. In terms of deployment, the Radio Controller is deployed in virtual machines orchestrated by OpenStack, with its software components, including the rAPP, running as containers managed by Docker.



Spectrum wallet

POST /SpectrumWallet/registerSpectrumResource Registers a spectrum resource in the spectrum wallet.

GET /SpectrumWallet/listSpectrumResources Shows a list with the registered spectrum resources in the spectrum wallet.

DELETE /SpectrumWallet/deleteSpectrumResource/{resourceId} Takes a spectrum resource out of the spectrum wallet.

GET /SpectrumWallet/translateSpectrumResource/{resourceId} Translates a spectrum resource in the wallet to TMForum's data model.

Radio resources

GET /RadioResources/listRadioResources Shows a list with the radio resources that are managed by the radio controller.

POST /RadioResources/translateRadioResource/{resourceId} Translates a radio resource to TMForum's data model.

Figure 4-4: rAPP API

The 5GZORRO GitHub repository of the module is available at:

https://github.com/5GZORRO/xRM.

4.1.3. Functional Tests

This section reports the set of the functional tests performed during the implementation of the xRM. As described above, the xRM interacts with several elements of the 5GZORRO platform and the 5G Virtualized infrastructure to perform its own tasks so, the set of tests reported in the tables below also includes the integration with the dependency modules.
Table 4-1 describes tests for the retrieval of different NFV Descriptors (NSD, VNFD, etc.) to be used as input for the translation process towards the TM Forum models. The descriptors are retrieved by the 5G-Catalogue components of the xRM that collected them directly from the NFVO.

Table 4-1 : 5G-Catalogue Functional Tests

Name	Description	Passed (Yes/No/Partially)
Descriptors Retrieval	Retrieval of the on-boarded VNFD, PNFD, NSD (previously retrieved from the NFVO and already present in 5G-Catalogue)	Yes
NFVO Descriptors Retrieval	Retrieval of the VNFD, PNFD, NSD descriptors from the NFVO	Yes

Table 4-2 shows tests related to the xRM interface configured to extract metrics to be collected by the MDA, directly for the API exposed by OSM.

Table 4-2: OSM API Functional Tests

Name	Description	Passed (Yes/No/Partially)
Collecting values of the monitored metrics	The module proxy the collection of the metrics from OSM (API wrapping)	Yes
Proxy OSM NBI	Wrap and secure with Gravitee the Northbound Interface of OSM	Yes

In Table 4-3 the tests have been conducted against the interface exposed by the radio controller that, when required, retrieves the list of on-boarded spectrum and radio resources, and enforces specific configuration in the RAN infrastructure.

Table 4-3: Radio Controller API Functional Tests

Name	Description	Passed (Yes/No/Partially)
Radio and Spectrum Retrieval	Retrieval of the on-boarded Radio and Spectrum resources from the Radio Controller	Yes
Spectrum Resource Creation	Registration of a Spectrum Resource	Yes
Radio Resource Registration	Registration of a Radio Resource	Yes

Radio Resource Configuration	Configuration of the radio parameters of a Radio Resource	Yes
Discovery of Radio and Spectrum Resources	Discovery of registered Radio and Spectrum Resources	Yes
Service Configuration	Configuration of a service on a set of Radio Resources	Yes

Table 4-4 shows tests for resource and service retrieval to feed the xRM translation module. In this case, the target module is the I2CAT Slice Manager (see section 4.2).

Table 4-4: Slice Manager API Functional Tests

Name	Description	Passed (Yes/No/Partially)
Edge, Cloud, Slice Retrieval	Retrieval of the on-boarded Edge, Cloud and Slice from the Slice Manager	Yes

Finally, Table 4-5, shows tests related to the translation of services and resources collected by different modules: 5G-Catalogue/NFVO, Radio Controller and Slice Manager. The table also includes a test for the final step of the translation process: the storage of the translated information in the Resource and Service Offer Catalogue (RSOC, reported in D3.2 [49]).

Table 4-5: Translator API Functional Tests

Name	Description	Passed (Yes/No/Partially)
VNFD, PNFD, NSD Translation	Translation of VNFD, PNFD, NSD from ETSI SOL-006 to TMF 633, 634	Yes
Radio and Spectrum Translation	Translation of Radio and Spectrum resources to TMF 633	Yes
Edge, Cloud, Slice Translation	Translation of Edge, Cloud and Slice resources to TMF 633, 634	Yes
RSOC POST	Invocation of RSOC to store the translated resources and services	Yes

4.2. Network Slice and Service Orchestrator

The Network Slice and Service Orchestrator (NSSO) is responsible for processing communication service level lifecycle management actions, namely instantiate, terminate, and scale, received from the Intelligent and Automated Slice & Service Management (ISSM). With this scope, the NSSO performs the mapping of the communication service into network slices, within an administrative domain, and interacts with the 5G virtualized infrastructure for the lifecycle management of the resources associated with the network slices. This mapping is performed based on rules and algorithms which can be updated dynamically as part of the management control loops. Moreover, the NSSO is responsible for configuring the monitoring metrics associated to the service, the correspondent e-Licenses to be attached to the service instances and the virtual infrastructure networking, when the local network slice is part of an e2e one across multiple domains.

4.2.1. Main Functionalities

The main scope of NSSO is to manage the lifecycle of service and slices within an administrative domain. With respect to the initial prototype reported in D4.2 [1], the current version of the NSSO has been integrated completely in the 5GZORRO platform, as depicted in Figure 4-5 showing the interaction with the modules involved in the orchestration process.

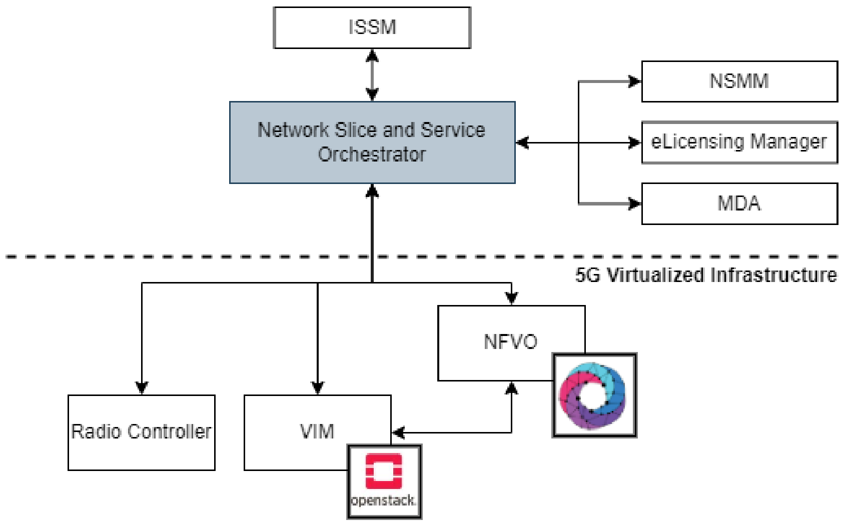

ISSM
Network Slice and Service Orchestrator
NSMM
eLicensing Manager
MDA
5G Virtualized Infrastructure
NFVO
Radio Controller
VIM
openstack

Figure 4-5: Network Slice and Service Orchestrator interactions

The NSSO exposes a REST interface towards the ISSM, which represents the 5GZORRO *business-oriented* orchestrator that acts as a bridge between the 5GZORRO Marketplace and orchestration stack, triggering orchestration commands based on business events such as new product orders, service extensions with new resources acquired from marketplace, etc. During its orchestration operations, the NSSO interacts with other components of the 5GZORRO platform to prepare the context for the deployment of a new Network Service/Slice before requesting its explicit provision to the modules of the 5G Virtualized infrastructure. In particular, the NSSO consumes services from the following modules:

- ***NSMM*** (See section 4.3): NSSO queries the NSMM in the case of cross-domain Network Slice/Service that requires the 5GZORRO VPN cross-domain service (VPNaaS) to secure the connectivity between the involved domains.
- ***eLicensing Manager*** (eLM, see section 4.4): NSSO requests eLM to check license terms of the 5G product before proceeding with the orchestration operations.
- ***MDA*** (Reported in D3.2 [49]): NSSO instructs the Monitoring Data Aggregator to collects metrics related to the target Network Service/Slice to be deployed.

With respect to the 5G virtual infrastructure, NSSO interacts directly with NFV Orchestrator, VIM and Radio Controller for Network service and slice, provisioning, resource allocation and radio configuration.

4.2.2. Prototype implementation

The NSSO is a multi-module component built by the integration of two different orchestration platforms: the Nextworks Vertical Slicer (VS) and the I2CAT Slice Manager (SM). The VS represent the software baseline and, as depicted in Figure 4-6, consists of two main software components, the Vertical and the network service management functions (V/NSMF) with the respective repositories for maintaining technical descriptors (V/NS Catalogues) and Service/Slicer Records (V/NS Record).

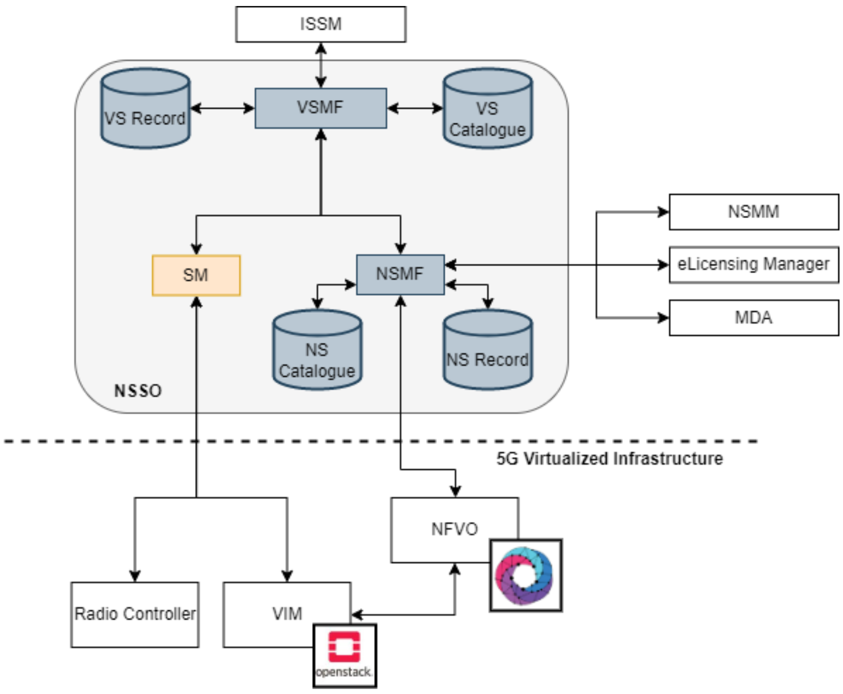


Figure 4-6 : NSSO software architecture

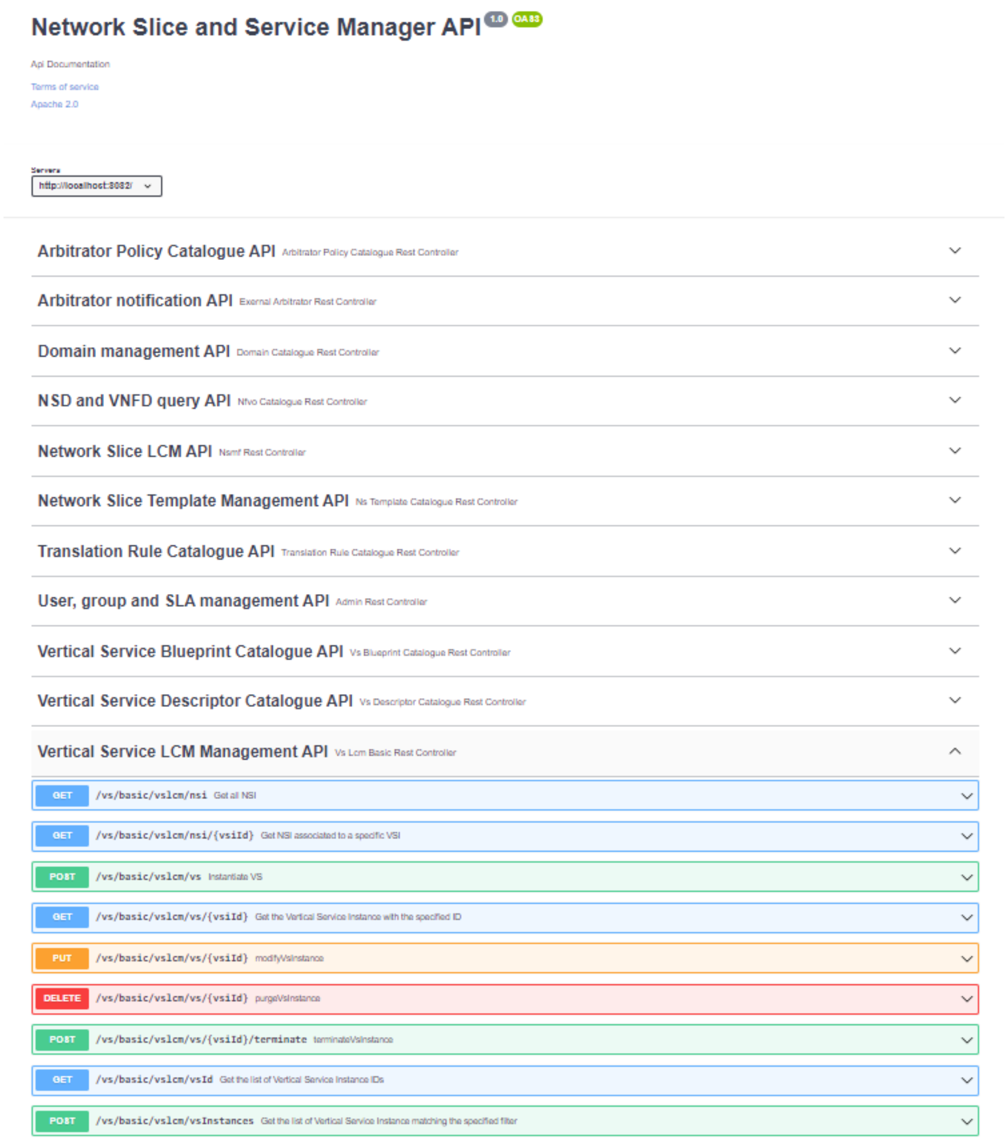
The VSMF is responsible for processing the service lifecycle management requests, while the NSMF contains the logic to translate network slices into resource and network service allocations in the 5G Virtualized infrastructure. In particular, the VS NSMF offers standardized network slice management interfaces, and implements a driver-based approach to interact with the underlying infrastructure and MANO platforms. Several ETSI NFV information models are supported: IFA-014 [32], SOL-006 [33].

In terms of implementation, both VSMF and NSMF are Java Spring Boot applications, while the catalogues and the record databases are maintained by a PostgreSQL instance. The interaction between the two modules happens through specific interfaces that implements both request/response (REST API) and publish/subscribe communication paradigms, for enabling both synchronous and asynchronous message exchanging. The VSMF exposes a REST-based North-bound interface (NBI) that is consumed by the ISSM to make requests to the NSSO.

The SM is evolved from the 5GCity project [34] and inside the NSSO acts as another NSMF providing the capabilities to interact with the radio controller and allocate resource in the VIM. SM has been extended in the context of 5GZORRO to meet the requirements of a NEST-based slice provisioning. To do so, the following functionalities have been added in the current prototype:

- *Management of Generic Slice Template (GST):* Define the slice attributes together with the set of supported values for each of them in the managed domain. The specific attributes that we are currently using are:
    - Isolation Level (e.g., Logical Isolation)
    - User Data Access (e.g., Direct internet access)
    - Compute Availability Zone (e.g., nexusEdge)
    - Access Technology (e.g., "5G", "4G", "Wi-Fi")
    - Area of Service
    - Radio Spectrum (WiFi channel, 4G/5G band, duplex mode)
    - Maximum Downlink Throughput
    - Maximum Downlink Throughput per UE
    - Maximum Uplink Throughput
    - Maximum Uplink Throughput per UE

- *Management of Network Slice Type (NEST):* Define an instance of the GST with required values for the desired attributes. Note that to be feasible, selected values must be a subset of supported values. Together with the registration of every NEST, a network slice blueprint [35] (i.e., required physical and logical resources) is also generated in the SM, which references the resources to be used by this NEST. In the current prototype, this can be done in two different ways:
    - Imperative approach: The resources to be used by the slice are also passed in the payload provided in the NEST creation request.
    - Declarative approach: The internal logic of the SM (as NSMF) and the RAN Controller (as NSSMF) is used to determine the resources to be included in the slice blueprint, based on the requested attributes' values.
- *NEST-based Network Slice Instance (NSI):* Define a network slice instance following the specifications of the blueprint associated with the selected NEST. Additionally, some configurable parameters are also expected at this step for the required slice/service (e.g., PLMN, SSID, etc.)


Network Slice and Service Manager API 1.0 OAS3

Api Documentation

Terms of service

Apache 2.0

Servers: http://localhost:8082/

Arbitrator Policy Catalogue API Arbitrator Policy Catalogue Rest Controller

Arbitrator notification API Exernal Arbitrator Rest Controller

Domain management API Domain Catalogue Rest Controller

NSD and VNFD query API Nfvo Catalogue Rest Controller

Network Slice LCM API Nsmf Rest Controller

Network Slice Template Management API Ns Template Catalogue Rest Controller

Translation Rule Catalogue API Translation Rule Catalogue Rest Controller

User, group and SLA management API Admin Rest Controller

Vertical Service Blueprint Catalogue API Vs Blueprint Catalogue Rest Controller

Vertical Service Descriptor Catalogue API Vs Descriptor Catalogue Rest Controller

Vertical Service LCM Management API Vs Lcm Basic Rest Controller

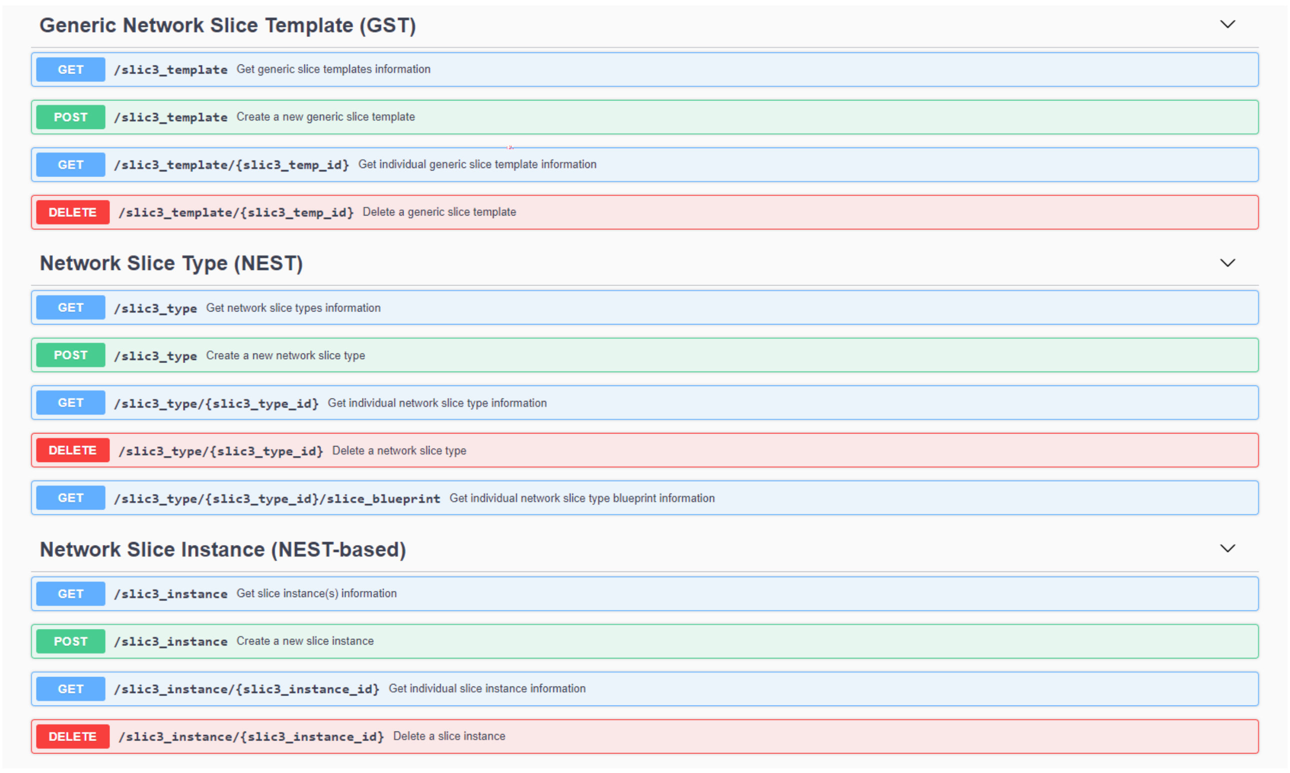
GET	/vs/basic/vslcm/nsi	Get all NSI
GET	/vs/basic/vslcm/nsi/{vsiId}	Get NSI associated to a specific VSI
POST	/vs/basic/vslcm/vs	Instantiate VS
GET	/vs/basic/vslcm/vs/{vsiId}	Get the Vertical Service Instance with the specified ID
PUT	/vs/basic/vslcm/vs/{vsiId}	modifyVsInstance
DELETE	/vs/basic/vslcm/vs/{vsiId}	purgeVsInstance
POST	/vs/basic/vslcm/vs/{vsiId}/terminate	terminateVsInstance
GET	/vs/basic/vslcm/vsId	Get the list of Vertical Service Instance IDs
POST	/vs/basic/vslcm/vsInstances	Get the list of Vertical Service Instance matching the specified filter

Figure 4-7: NSSO OpenAPI specification

Both VS and SM are currently deployed as container applications maintained in a Kubernetes environment:

- *VSMF*: Containerized deployment of the vertical service management function available in [36]. This module contains the driver to interact with NSMF and SM. Available in [36].

- *NSMF (VS)*: Containerized deployment of the network slice management function available in [36]. This module contains the drivers to interact with the MDA, e-Licensing Manager and NSMM, using the APIs established in [37][38]. It also onboards a driver to interact directly with the NFVO (ETSI OSM [39])
- *NSMF (SM)*: Containerized deployment of the evolved Slice Manager, with services exposed via the APIs established in [40]. This module contains the driver to interact with the i2CAT's RAN Controller. In terms of deployment, the i2CAT SM is deployed in the 5GBarcelona testbed as a software container orchestrated by Kubernetes. In Figure 4-8, the OpenAPI showcasing the extended functionalities can be checked.


Generic Network Slice Template (GST)

GET /slic3_template Get generic slice templates information
POST /slic3_template Create a new generic slice template
GET /slic3_template/{slic3_temp_id} Get individual generic slice template information
DELETE /slic3_template/{slic3_temp_id} Delete a generic slice template

Network Slice Type (NEST)

GET /slic3_type Get network slice types information
POST /slic3_type Create a new network slice type
GET /slic3_type/{slic3_type_id} Get individual network slice type information
DELETE /slic3_type/{slic3_type_id} Delete a network slice type
GET /slic3_type/{slic3_type_id}/slice_blueprint Get individual network slice type blueprint information

Network Slice Instance (NEST-based)

GET /slic3_instance Get slice instance(s) information
POST /slic3_instance Create a new slice instance
GET /slic3_instance/{slic3_instance_id} Get individual slice instance information
DELETE /slic3_instance/{slic3_instance_id} Delete a slice instance

Figure 4-8: SM OpenAPI

The 5GZORRO GitHub repository of NSSO is available at:

https://github.com/5GZORRO/nsso.

4.2.3. Functional tests

The set of functional tests conducted so far on the NSSO are reported in Table 4-6. In the description of each test is mentioned any dependency and interaction with other 5GZORRO modules, if required to demonstrate the target functionality.

Table 4-6: Network Slice and Service Orchestrator functional tests

Name	Description	Passed (Yes/No/Partially)
Service instantiation	A simple service/slice instantiation is requested from the ISSM. The NSSO provisions a network slice with one network service relying on the local domain functionality. This would include several interactions with different modules: • **MDA -** The NSSO requests the configuration of the metrics to the MDA	Yes

	• **eLicensing Manager -** The NSSO establishes the e-Licenses associated with a particular service instance in the e-License Manager • **NFVO (5G Virtualized Infrastructure) –** Instantiation of service components in the virtual infrastructure In a context of cross-domain e2e service: • **NSMM -** Allocation of secure networking channels for services spanning across different administrative domains	
Creation of GST	A GST is created at the SM specifying the set of supported values for each one of the considered attributes.	Yes
Creation of NEST (and network slice blueprint)	A NEST and associated network slice blueprint are created at the SM	Yes
NEST-based instantiation of RAN slices	A slice instance is created at the SM based on the provided NEST (and associated network slice blueprint). This would include several interactions with underlying systems such as the VIM and the RAN Controller.	Yes
Instantiation of vertical services using radio and edge resources	Instantiate a service using a previously allocated slice including radio access and edge resources (allocated through the SM module). This would include several interactions with underlying systems such as the NFVO.	Yes
Onboarding of SOL006 based descriptors	Onboard SOL006 based NSD packages through the NBI of the NSSO into the NFVO catalogue. This would include several interactions with underlying systems such as the NFVO.	Yes

4.3. Network Service Mesh Manager

The aim of the Network Service Mesh Manager (NSMM) is to secure the communication of an end-to-end service deployed across different administrative domains. Such an operation is performed at orchestration time and is explicitly requested by the NSSO which is also the solely consumer of the NSMM interface. In this regard, the NSMM can be considered as an extension of the NSSO and fully part of 5GZORRO Orchestration stack.

4.3.1. Main Functionalities

To secure communications of an end-to-end cross-domain service, the NSMM needs to interact with different modules belonging to the 5GZORRO platform and directly with the VIM (OpenStack), as depicted in Figure 4-9.

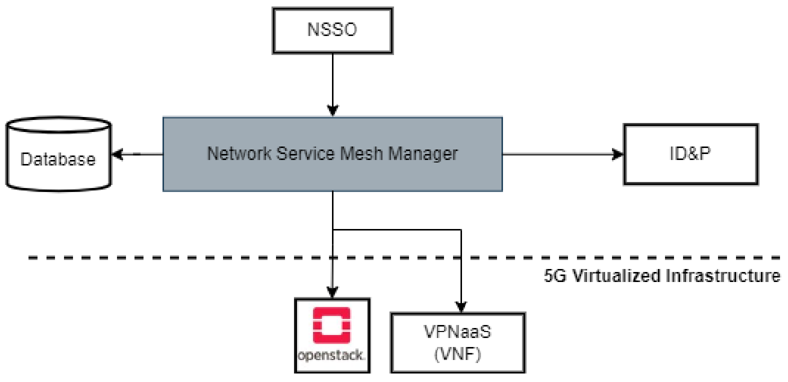


Figure 4-9: Network Service Mesh Manager interactions

The NSMM receives commands directly from the NSSO and is invoked only when a cross-domain connection needs to be established. The securing operation is realized by properly configuring an instance of VPNaaS module (see section 2.4) instantiated as a VNF at orchestration time. The VPNaaS instances are used as communication gateways in each side of the VPN tunnel as depicted in Figure 4-10 while, the tunnel itself, is established by using 5GZORRO stakeholder information and keys provided by the ID&P Manager.

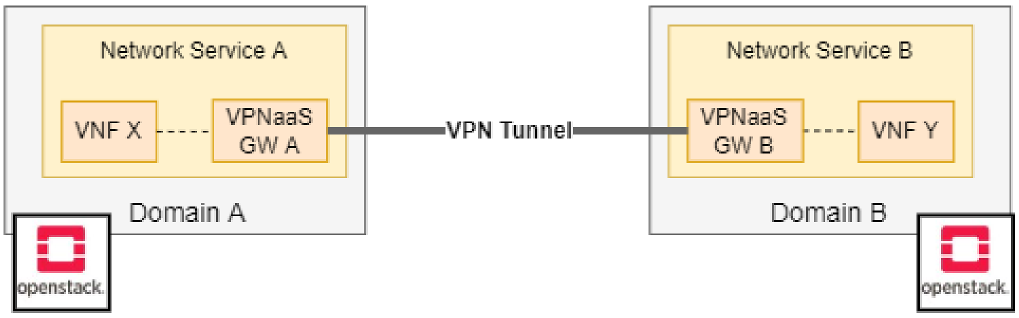


Figure 4-10: VPN Secure link across two domains

The network service in Domain A connects with the network service in Domain B through a VPN tunnel where the respective instances of the VPNaaS works as gateway for the cross-domain network traffic. This implies that the NSMM must offer a set of functionalities that allows the enforcement of specific configurations enforced in both the VIM and VPNaaS instances.

In particular, the NSMM offers two main functionalities:

- **OpenStack networking configuration**
  - Creation of the necessary OpenStack network resources to connect the Network Functions in the Network Service. These resources are *networks* and *subnets*.
  - Creation of OpenStack network resources to expose the gateway Network Function (VPNaaS instance) to the outside world. This step basically consists of creating a *virtual router* connected to the gateway and the floating network (assumed as the operator external network), to be able to associate a Floating IP (i.e., Public IP) to that virtual machine.
- **Cross-Domain connection securing**
  - Interaction with the ID&P to retrieve keys and DID, identification of the network and services to be exposed to other domains, and VPNaaS instance configuration

When the NSSO needs to orchestrate a service which is part of an e2e cross-domain one, first requests the NSMM to prepare the networking in the VIM, then instantiates the service along with the VPNaaS-based gateway. At this point, the NSSO can request the NSMM to secure the cross-domain connection.

4.3.2. Prototype Implementation

The NSMM prototype implements all the modules described in deliverable D4.4 [19] and both functionalities previously mentioned to stich and secure services across multiple domains. The 5GZORRO GitHub repository of the module is available at:

https://github.com/5GZORRO/network-service-mesh-manager.

The NSMM implements a request/response communication paradigm based on REST API and the NBI is described and available at:

https://github.com/5GZORRO/network-service-mesh-manager/tree/main/api.

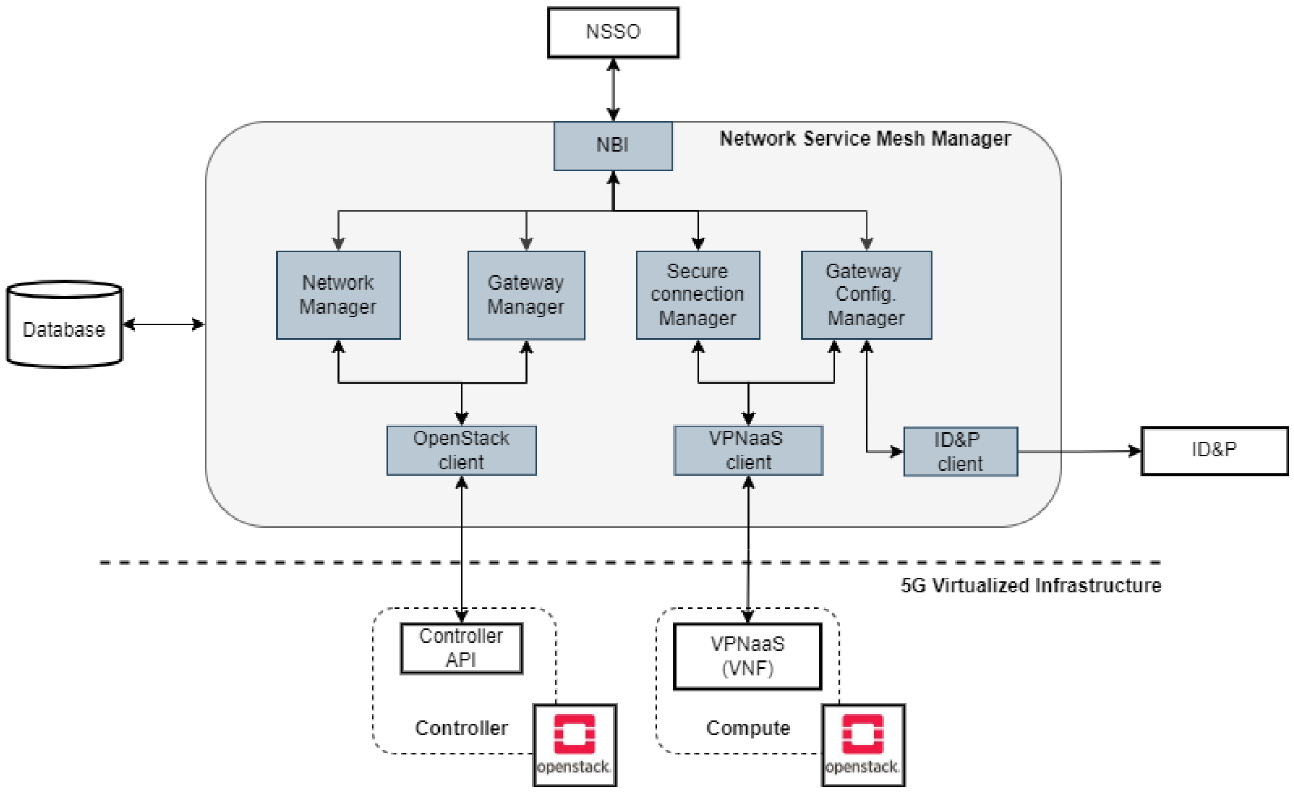
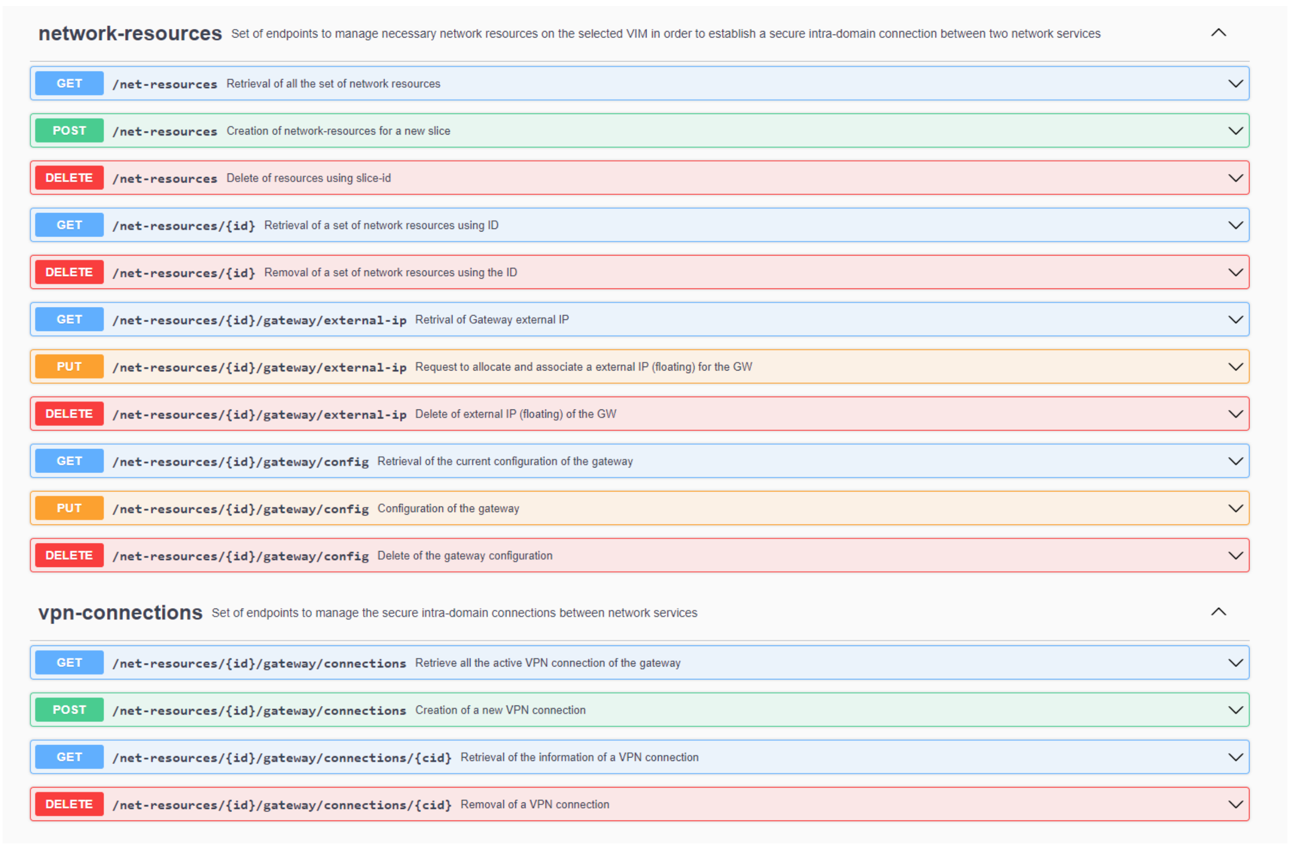


Figure 4-11: NSMM Software architecture

As depicted in Figure 4-11, which describes the actual NSMM software architecture, the final prototype interacts through a set of dedicated clients with the VIM, the ID&P, and the local instance of the VPNaaS gateway. The Network Manager (NM) and the Gateway manager (GM) are responsible for the preparation of the network in OpenStack while the Gateway Config Manager (GCM) and the Secure Connection Manager (SCM) are responsible for the establishment of the secure VPN-based cross-domain connection. In particular, the NM creates the VIM networks and the dedicated subnets for the internal communication, the GM creates and configures the VIM virtual router for the cross-domain connectivity, the GCM retrieves Keys and DID and identify networks and services to be exposed to the other domains, and, finally, the SCM configures the VPNaaS instance to establish the VPN tunnel.

The four managers interact with the Database (an instance of PostgreSQL) to guarantee the persistence of the data (explicit arrows are omitted in the figure for the sake of readability). The NBI is specified in OpenAPI format, as depicted in Figure 4-12.



network-resources Set of endpoints to manage necessary network resources on the selected VIM in order to establish a secure intra-domain connection between two network services

GET `/net-resources` Retrieval of all the set of network resources

POST `/net-resources` Creation of network-resources for a new slice

DELETE `/net-resources` Delete of resources using slice-id

GET `/net-resources/{id}` Retrieval of a set of network resources using ID

DELETE `/net-resources/{id}` Removal of a set of network resources using the ID

GET `/net-resources/{id}/gateway/external-ip` Retrival of Gateway external IP

PUT `/net-resources/{id}/gateway/external-ip` Request to allocate and associate a external IP (floating) for the GW

DELETE `/net-resources/{id}/gateway/external-ip` Delete of external IP (floating) of the GW

GET `/net-resources/{id}/gateway/config` Retrieval of the current configuration of the gateway

PUT `/net-resources/{id}/gateway/config` Configuration of the gateway

DELETE `/net-resources/{id}/gateway/config` Delete of the gateway configuration

vpn-connections Set of endpoints to manage the secure intra-domain connections between network services

GET `/net-resources/{id}/gateway/connections` Retrieve all the active VPN connection of the gateway

POST `/net-resources/{id}/gateway/connections` Creation of a new VPN connection

GET `/net-resources/{id}/gateway/connections/{cid}` Retrieval of the information of a VPN connection

DELETE `/net-resources/{id}/gateway/connections/{cid}` Removal of a VPN connection

Figure 4-12: NSMM OpenAPI specification

The NSMM prototype has been implemented in Go [43] (Golang 1.17) using the main following libraries:

- gin-gonic/gin [44] v1.7.4 and deepmap/oapi-codegen [45] v1.9.0, to automatically generate the server from the OpenAPI NSMM interface definition.
- Gorm [46] v1.22.5, ORM library for Golang.
- Postgres driver [47] v1.2.3 as database driver.
- gophercloud/gophercloud [48] v0.23.0, an OpenStack Go SDK.

The current prototype can be deployed on

- Kubernetes (using Helm)
- Docker (with docker-compose)

Both these deployments automatically start the NSMM and the database. The NSMM module can also be manually compiled and executed on VM installing all the necessary dependencies and the Postgres Database. More information can be found on the NSMM GitHub repository linked above.

4.3.3. Functional Tests

The Table 4-7 shows the Network Service Mesh Manager functional tests. The tests have been conducted in both local machines and 5GZORRO testbed. For those functionalities that requires the interaction with external modules i.e., ID&P, the related tests also include the integration tests with those modules.

Table 4-7: Network Service Mesh Manager functional tests

Name	Description	Passed (Yes/No/Partially)
Network resource creation	The NSMM creates all the network resources for a Network Service, these are basic internal networks of the Network Services	Yes

	and network resources for the Gateway to enable external exposure and connectivity.	
Floating IP allocation	The NSMM allocates a floating IP to the Gateway VNF to enable external exposure and connectivity.	Yes
VPNaaS configuration	The NSMM configures the VPNaaS modules. This would imply the request of information (DID and Keys) to the ID&P	Yes
VPN connection creation	The NSMM creates the VPN connection towards another domain which expose a Gateway with a running and configured instance of the VPNaaS module.	Yes
VPN connection removal	The NSMM deletes the VPN connection toward another domain.	Yes
Floating IP release	The NSMM releases the floating IP to the Gateway VNF to make it no more reachable and exposed.	Yes
Network resources removal	All the network resources of a Network Services created by the NSMM are deleted.	Yes

4.4. E-Licensing Manager

The objective of the e-Licensing Manager (eLM) is to enable a trustworthy tracking of the usage of purchased software components/xNFs, in real-time, providing prove of usage so the vendors/owners can materialise revenues associated to licensing costs. It is designed to seamlessly integrate stakeholders to the 5GZORRO ecosystem who want to either offer and consume xNFs or want to offer their virtualized infrastructure for the orchestration and deployment of xNFs, resulting on a production-grade telecommunication framework which will monitor the utilization of xNFs commercialized through the 5GZORRO Marketplace.

4.4.1. Main functionalities

Given the final design of the eLM, which was described in D4.4 [19], this section includes the main functionalities of the eLM from the perspective of the software vendor as well as from the MANO enhancements in a dynamic and multi-stakeholder ecosystem.

From the software vendor point of view, the eLM allows the commercialization of software components/xNF with a wide range of business plans, including flat rates, pay-as-you-grow rates, and subscription rates. Enhanced by the capability of the eLM to monitor the usage and thus the compliance of the rates based on system metrics, such as uptime, or custom application-level metrics thanks to the eLM interaction with the Data Lake and Cross-domain Analytics services of the 5GZORRO Platform. This has been further elaborated on D3.3 [51] and D4.4 [19].

From the MANO point of view, the eLM offers a way to validate the compliance of a software component/xNF respect to a Smart Contract that clearly states what are the conditions and limitation of its use. Before the xNF is instantiated, the eLM will verify that the license is valid for a given administrative domain and that the limits will not be surpassed considering previous and active uses from other participating stakeholders on the 5GZORRO ecosystem. Then, during the lifetime of the xNF, the eLM will monitor in real time its usage based on the configured metrics as to offer a prove of usage on the smart contract.

4.4.2. Prototype implementation

This section shows the implemented prototype and technologies used to realize the design shown in D4.4 [19]. The eLM has been updated in the latest phase to become a decentralized component which relies in the Integration Fabric of the Platform to synchronize instances of the eLM in different administrative domains. Each of these instances is referred to as an eLM Agent (eLMA) deployed as a cloud-native microservice-based application using Helm. This contributes to the flexibility of the 5GZORRO Platform, the profile of actors and installation procedures, further explained in section 6.

All internal microservices of eLMA have been developed using Python 3.8 and standard libraries such as *flask, connexion[swagger], pika, request, kafka_python* and *schedule* while the business logic and core operations are kept in an independent library developed externally as part of the asset's development program of ATOS.

Some key characteristic considered during the implementation include a small footprint, cross-stakeholder monitoring of licenses usage, and a dynamic plugin-driven configuration. This last feature is achieved by customizing environment variables in the helm chart which then installs the appropriate plugin and additionally ensures connectivity to the different components that the eLM needs to interact with as to provide its functionalities. This applies to the interface with the underlaying infrastructure and the NFV orchestration of the 5GZORRO Platform, the eLM has been tested on a fully cloud-native infrastructure using Kubernetes and the ISSM-MEC component of 5GZORRO and on an infrastructure based on Openstack, the ETSI Open-Source MANO (OSM) and the NSSO component of 5GZORRO.

Furthermore, the eLM has been designed to best reflect the business plan of xNF vendors by allowing a highly customizable drafting of the licenses in terms of granularity of configuration and monitoring. This design has been reported in D3.2 [49] and D3.3 [51].

D4.4 [19] included the definition of the functional blocks of the eLM (see also Figure 4-13) as well as the main interfaces with other components of the 5GZORRO Platform.

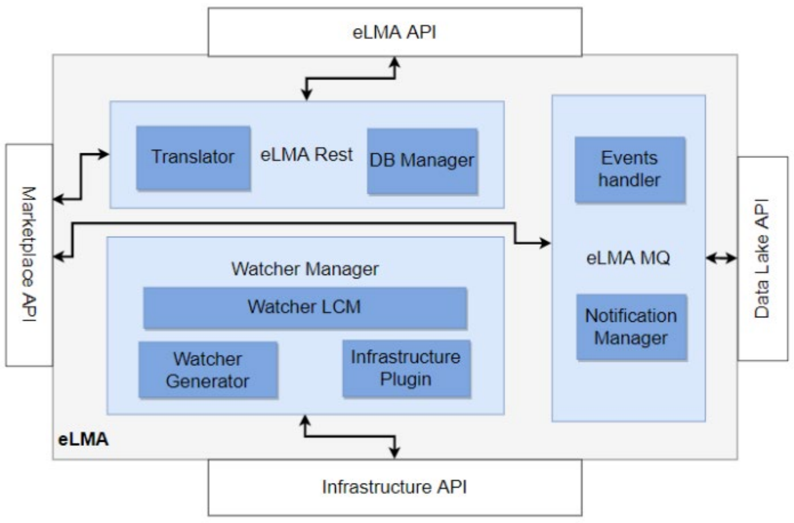


Figure 4-13: eLM microservices and functional blocks

The current prototype of the eLM complements the previous information with details on the technologies used:

- Each eLMA is deployed in the infrastructure of a 5GZORRO participant as a collection of loosely coupled microservices (light blue boxes in Figure 4-13) preferable in a Cloud-Native environment such as Kubernetes using Helm, although it has been tested using docker-compose in a single VM too.
- In terms of communications, it follows a request/response paradigm (based on REST) for synchronous internal workflows and for the interaction with most of the additional components of the 5GZORRO Platform. It additionally relies on a publish/subscribe paradigm (AMQP) for the handling of asynchronous events related to the monitoring and usage of the xNFs and their associated licenses. Events are handled via a RabbitMQ broker that is shared between several components of the 5GZORRO Platform, and this broker is used to notify the Marketplace Portal about a license violation that has just happened and that is being processed by the Platform.

- The source code, as well as the complete list of requirements and the installation scripts and documentation is available within the 5GZORRO GitHub at:

  https://github.com/5GZORRO/elicensing-manager-agent.

  In terms of storage and resources, the eLMA is designed to be light and stateless for all its internal microservices while it relies on a NoSQL database to persist the monitored xNFs and the metrics associated to the compliance of the licenses. For this, a MongoDB has been chosen.
- The eLM makes use of the OpenAPI specification for the REST endpoints which are accessible within the cluster or through the exposed service under *<eLMA-URL>:<eLMA-PORT>/ui,* where eLMA-URL and eLMA-PORT are fully configurable via its deployment scripts.

4.4.3. Functional tests

In this section the tests that have been performed to the e-Licensing Manager components are presented These tests are further demonstrated in section 5.3.1, where the internal call flows of an end-to-end scenario are shown.

While the first four tests shown in Table 4-8 can run in isolation from the rest of the 5GZORRO Platform using *pytest* and mocking the external dependencies, the remaining have been carried out at the 5GZORRO Barcelona Testbed. In said testbed, the components that are required for each test are described in the corresponding description.

Table 4-8: E-Licensing manager functional tests

Name	Description	Passed (Yes/No/Partially)
Test_registration_simple_PO	Test for the registration of a Product Offering with a single PO Price in the ELMA.	Yes
Test_registration_bundled_PO	Test for the registration of a bundled PO where each internal PO has an additional POP	Yes
Test_registration_PO_error	Registration test of faulty POs which include non-existent references in the marketplace, missing POPs, invalid xNF references in the POPs	Yes
Test_watcher	Test for the watcher creation/deletion/update	Yes
Test_osm_interface	Test for the communication between the eLMA and the MANO via the xRM component. It includes: • Login • Get_descriptors • Get_instance_status • Describe_instances • Create_subscription • Remove_subscription	Yes
Test_cross_domain_interface	Test that the cross-domain communication can stop an instantiation if the limits on the license usage are surpassed	Yes
Test_interfaces	Integration test with the NSSO, Marketplace catalogue, Marketplace Portal, Data Lake, xRM, SCLCM. These tests are further elaborated in Section 5.3.1	Yes

5. Module integration for zero-touch service management prototype

This section presents the integration strategy implemented for the realization of a comprehensive 5GZORRO Zero-touch service management prototype with security and trust features, based on the various module prototypes described in previous sections.

As introduced in D4.2 [1], the integration strategy adopted by 5GZORRO leveraged on the organization of the whole software development and integration work in teams, each in charge of specific areas of the 5GZORRO Platform. Such strategy facilitated the implementation and integration of the different parts building the 5GZORRO Platform by fostering cross-WP interactions.

To further improve the defined methodology, the initially established teams have been refurbished, and now are as listed below:

- **TEAM #A – TELECOM MARKETPLAC**E: Mainly encompasses modules from Work Package 3; includes the e-Licensing Manager.
- **TEAM #B – ZERO-TOUCH SLICING:** This team tackles modules such as ISSM Workflow Manager, ISSM-MEC, ISSM-O, 5G-TRMF, NSSO and xRM.
- **TEAM #C – SLA-DATALAKE:** Shares the same modules from TEAM #B, excluding the 5G-TRMF.
- **TEAM #D – E2E SECURITY:** This team focuses on security-centric modules such as NSMM, VPN-as-a-Service, SAS, TEE and 5G-TRMF.
- **TEAM #Z – TESTBED**: Provides & maintains the required infrastructure to host the previously mentioned software components of 5GZORRO.

Continuing the work of team #6 described in D4.2 [1], team #Z will continue the work on the deployment's front. Furthermore, Teams #B, #C & #D focused on continuing the integration efforts of the previous Teams #3, #4 and #5, respectively, which are in scope with elements described in this deliverable, whereas Team #A encapsulated the work being made on the previous Teams #1 &#2.

5.1. Integration scenario overview

As introduced in D4.2 [1], part of the integration strategy was to create a scenario to be followed by the various development teams, and use said scenario as a storyline to drive the integration of all software components that constitute the 5GZORRO platform. Since there were no issues found between the established integration scenario in D4.2, and all the work performed thus far, the phases remain unchanged.

Likewise, the scenario carries over from D4.2 [1] following these steps:

- Creation of a resource/service offer by an authorized stakeholder.
- Publication of the offer in the marketplace.
- Post-deployment optimizations performed in a Zero-touch manner.

The high-level steps of the integration scenario followed by the software development teams can be checked in Figure 5-1.

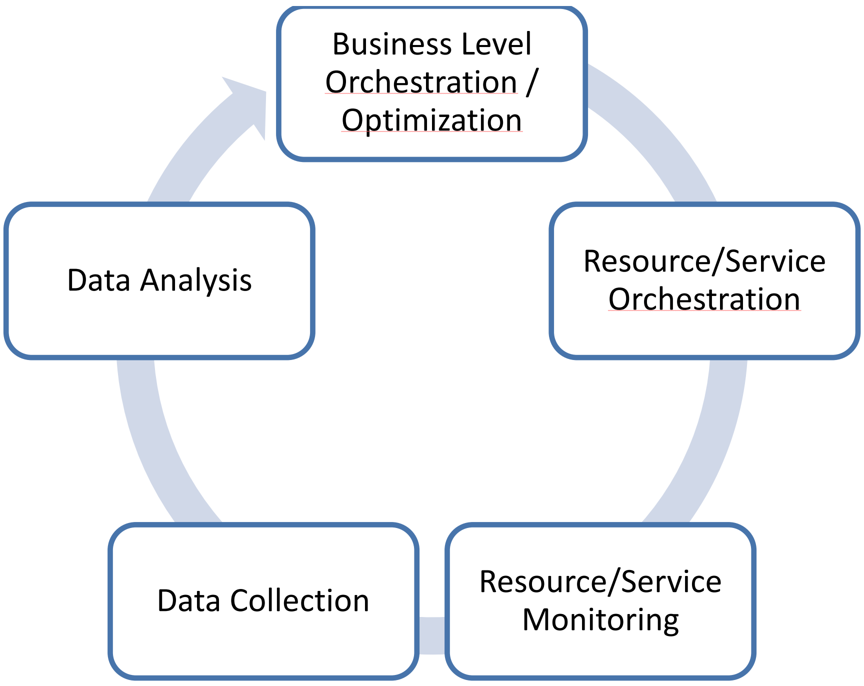

Business Level Orchestration / Optimization
Resource/Service Orchestration
Resource/Service Monitoring
Data Collection
Data Analysis

Figure 5-1: High-level steps followed by the Teams for the integration scenario

5.2. Zero-touch network slice orchestration

The overall technical approach of the zero-touch network slice orchestration integration scenario has remained the same as was previously reported in D2.3 [17] with workflows 3-6, 3-7, 3-10 guiding our efforts on the end-to-end integration. One major addition to the integration activities is involved with adding extra flexibility to the end-to-end operation of the components. More specifically, rather than always having fully automated flows always starting from high-level intents for a slice or a service, we found it useful to trade full automation for extra flexibility through the manual process if needed. For example, Product Offers can be acquired asynchronously by a 5GZORRO persona and then at a later stage, when Product Orders that were obtained by acquiring the corresponding product offers need to be combined and instantiated or instantiated and scaled out. Thus, a long management flow of ISSM might need to be split into shorter ones, e.g., we need to provide a flow that starts with the product order (that was created already) and instantiate or scale-out them. To that end, we have implemented the corresponding ISSM flows, integrated them with the 5GZORRO Portal and successfully demonstrated them for all three Use Cases in the context of the project demos showcased at EUCNC'22.

Another major integration effort included developing and implementing a methodology that allows ISSM to seamlessly expand its orchestration flows to include the Product Offer specific orchestration for instantiation and scale-out. More specifically, when Product Orders are being mixed and matched from the portal to create composite services, we require one service to be the main product order and the rest to be subordinate Product Orders. Furthermore, the Product Offer corresponding to the main Product Order contains a reference to the service specific NFVO that contains the logic for instantiate and scale-out entry points. This way, when a service being created through the portal from Product Orders, the service orchestration is automatically added to ISSM.

Finally, since ISSM-WFM design became fully distributed and symmetric to better fit the 5GZORRO approach, we performed regression tests to ensure quality after the changes to ISSM-WFM have been implemented.

5.2.1. Integration tests and results

The prototypes that were previously validated under own premises testbeds, such as for integration between ISSM and NSSO to enable slice instantiation per request, have been fully ported to the 5GZORRO's testbed infrastructure.

Also, as part of the advancements made since the writing of D4.2 [1], MDA was integrated with the orchestration-centric software components across all of 5GZORRO's testbed environments, being able to receive configurations from the NSSO component (REST API) in its designated Operator domain, fetch metric data from ETSI OSM accordingly, and push data to the testbed's Data Lake.

Table 5-1 list the set of tests performed to validate the set of functionalities implemented by the ISSM & the NSSO tackling some integration steps, whilst Table 5-2 details MDA's functional tests related to the integration steps:

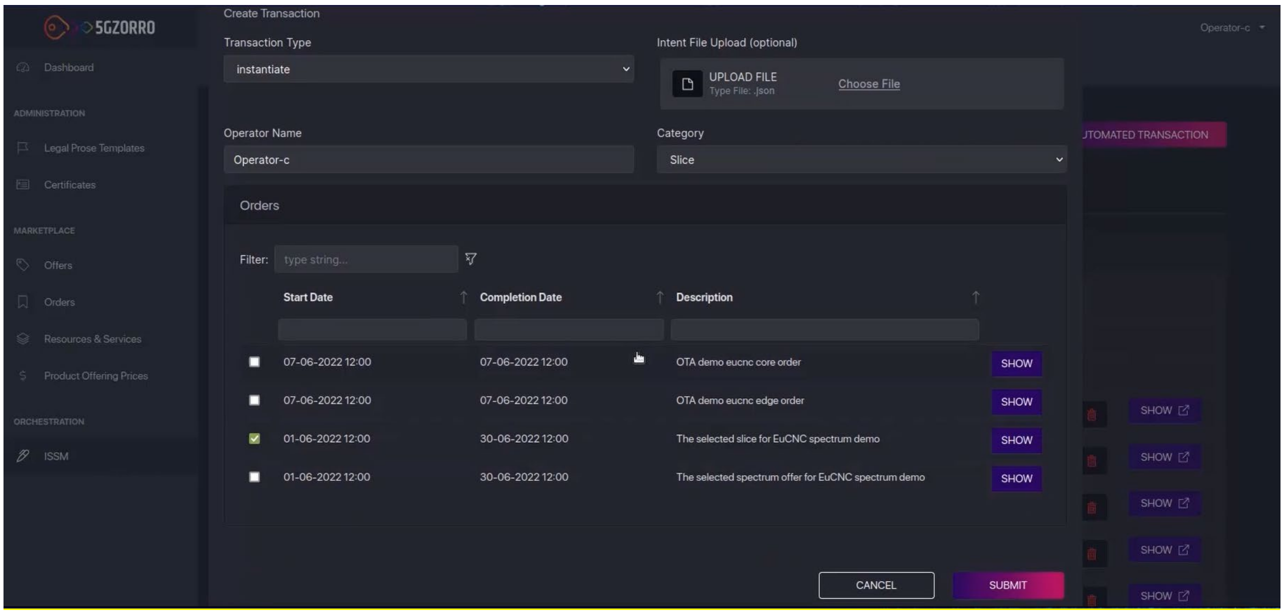
Table 5-1: ISSM-NSSO functional test set

Name	**Description**	**Passed (Yes/No/Partially)**
Process slice request	ISSM-WFM receives a "slice instantiation" request	Yes
Request resource instantiation	ISSM -WFM requests resource instantiation from NSSO	Yes
Generate instantiation	NSSO runs instantiation	Yes
Generate instantiation	NSSO completes instantiation	Yes
Configure Vertical Slicer virtual resources	Slice setup with gNodeB connected to 5GCore	Yes

Table 5-2: MDA functional test set

Name	**Description**	**Passed (Yes/No/Partially)**
Start Aggregate Metric	NSSO propagates configuration with dynamic variables to MDA	Yes
Read Metric Data	MDA fetches metric values from OSM	Yes
Aggregate Monitoring Data	Aggregation of metric data by the MDA component	Yes
Signing Data	Hash/signing data with PK from the Operator with SHA256 and RSA algorithm	Yes
Post Monitoring Data	Post data into Data Lake (DL Kafka topics / REST API)	Yes

Now integrated with the Portal, a user can request an instantiation of a Network Slice by accessing the 'Orchestration > ISSM' tab, as shown in Figure 5-2. Figure 5-3 depicts the submitted information, triggered by the portal, required for an instantiation of a Slice.



Create Transaction

Transaction Type: instantiate

Intent File Upload (optional): UPLOAD FILE Type File: .json Choose File

Operator Name: Operator-c

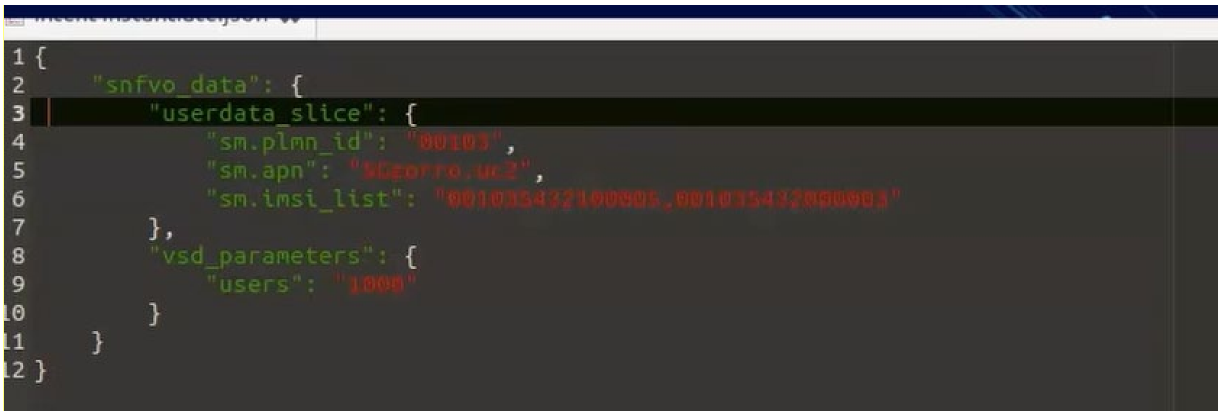
Category: Slice

Orders

Filter: type string...

	Start Date	Completion Date	Description	
☐	07-06-2022 12:00	07-06-2022 12:00	OTA demo eucnc core order	SHOW
☐	07-06-2022 12:00	07-06-2022 12:00	OTA demo eucnc edge order	SHOW
✓	01-06-2022 12:00	30-06-2022 12:00	The selected slice for EuCNC spectrum demo	SHOW
☐	01-06-2022 12:00	30-06-2022 12:00	The selected spectrum offer for EuCNC spectrum demo	SHOW

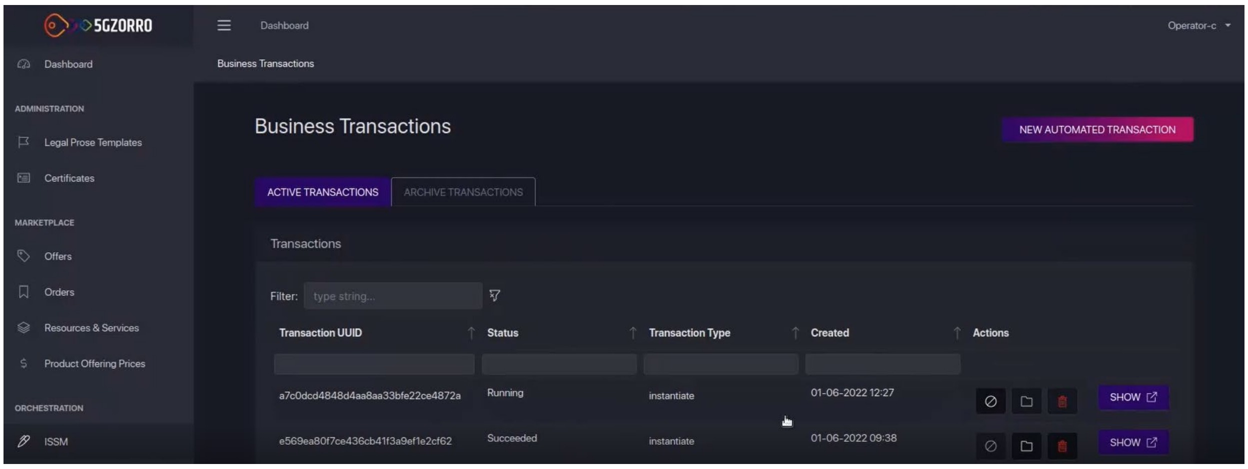
Figure 5-2: Slice instantiation submission on the Consumer's Portal



```
{
    "snfvo_data": {
        "userdata_slice": {
            "sm.plmn_id": [redacted],
            "sm.apn": [redacted],
            "sm.imsi_list": [redacted]
        },
        "vsd_parameters": {
            "users": [redacted]
        }
    }
}
```

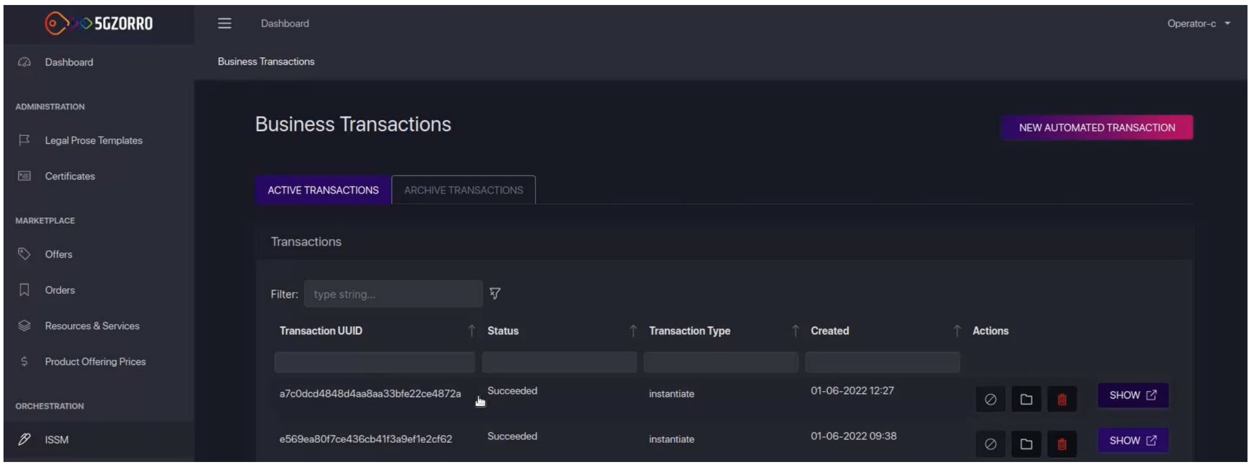
Figure 5-3: 5G Slice submission info

While background operations run to setup the Network Slice, the Portal will be updating and displaying in real-time the status of the instantiation, shifting from ‘Running’ to ‘Successful’ whenever it is completed (Figure 5-4 and Figure 5-5):



Transaction UUID	Status	Transaction Type	Created	Actions
a7c0dcd4848d4aa8aa33bfe22ce4872a	Running	instantiate	01-06-2022 12:27	SHOW
e569ea80f7ce436cb41f3a9ef1e2cf62	Succeeded	instantiate	01-06-2022 09:38	SHOW

Figure 5-4: Slice instantiation running



Transaction UUID	Status	Transaction Type	Created	Actions
a7c0dcd4848d4aa8aa33bfe22ce4872a	Succeeded	instantiate	01-06-2022 12:27	SHOW
e569ea80f7ce436cb41f3a9ef1e2cf62	Succeeded	instantiate	01-06-2022 09:38	SHOW

Figure 5-5: Slice instantiation complete

Upon completion of the instantiation & setup of the slice, the data monitoring workflow can now start. This requires that the NSSO, alerted about a slice instantiation by ISSM, sends a configuration to the MDA. To aggregate metrics, this config needs some required fields for each metric, such as *step*, *step_aggregation* and *aggregation_method*, Figure 5-6*.*

```
POST 'http://172.28.3.15:30040/settings'

{
    "transaction_id": "ab51f3e1-7b61-4f9d-85a4-9e9f366b593b",
    "instance_id": "05cce067-4818-4afc-b0f8-0e4a1babf753",
    "product_id": "10",
    "tenant_id": "domain-operator-a",
    "context_ids": [
        {
            "resource_id": "10",
            "network_slice_id": "05cce067-4818-4afc-b0f8-0e4a1babf753",
            "parent_id": "10"
        }
    ],
    "topic": "operator-a-in-0",
    "monitoring_endpoint": "http://172.28.3.15:30036/osm/api/v1/query",
    "data_source_type": "OSM",
    "metrics": [
        {
            "metric_name": "osm_requests",
            "metric_type": "numerical",
            "step": "1m",
            "aggregation_method": "AVG",
            "step_aggregation": "2m"
        }
    ]
}
```

Figure 5-6: Creation of monitoring configuration

Whenever the MDA receives a new configuration, the component triggers a process to read metric values from ETSI OSM (Figure 5-7), or other source, and this process relies on the *Step* key, which represents a metric reading interval, saved into a database temporarily.

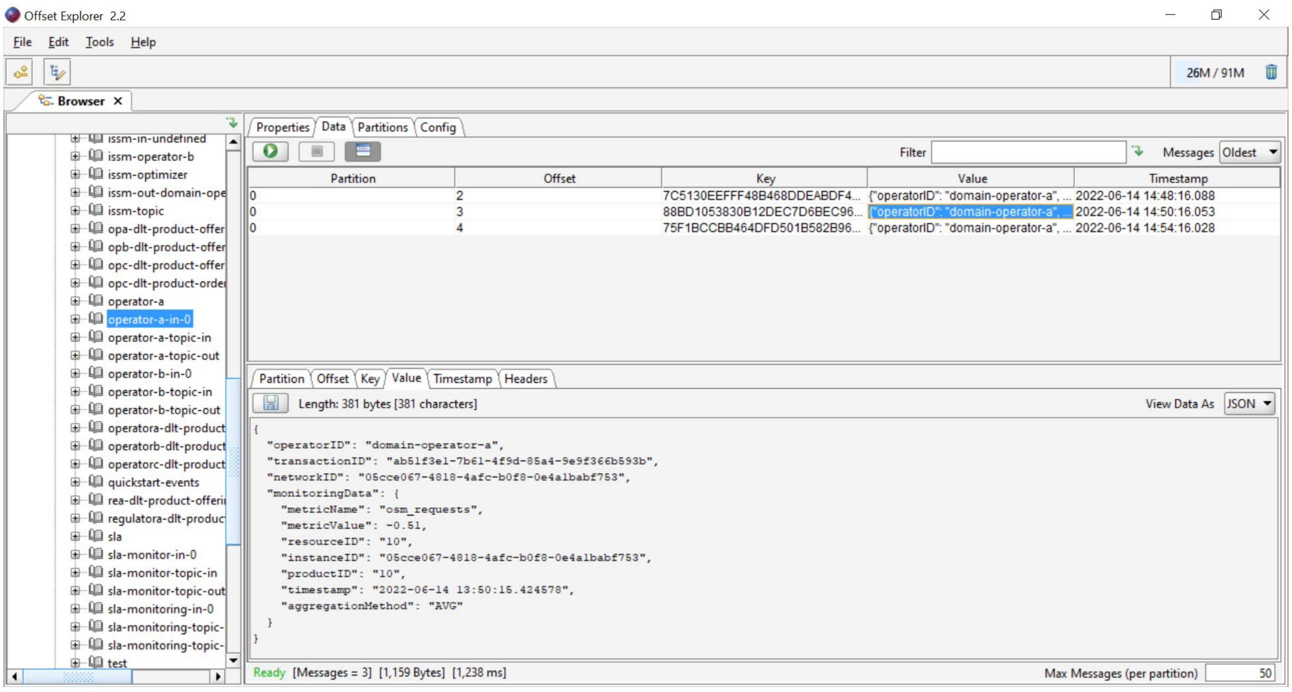
```
GET 'http://172.28.3.15:30036/osm/api/v1/query?query=osm_requests&time=2022-06-14T15:22:33.564731Z'

{
    "status": "success",
    "data": {
        "resultType": "matrix",
        "result": [
            {
                "metric": {
                    "__name__": "osm_requests",
                    "job": "mon_exporter",
                    "instance": "mon:8000",
                    "ns_id": "05cce067-4818-4afc-b0f8-0e4a1babf753",
                    "ns_name": "test0_instance",
                    "project_id": "0b0d7475-319f-48ce-a216-92b85f7800a8",
                    "vdu_name": "test-0-mgmtVM-0",
                    "vnf_member_index": "0"
                },
                "value": [
                    1636046553.564731,
                    "0.43"
                ]
            }
        ]
    }
}
```

Figure 5-7: Requesting of configured metric and its value

The other automatic process performed together with the data collection phase is the application of aggregations rules by using the *step_aggregation* field, where the read values are aggregated according to the *aggregation_method.* This aggregation method must include a valid operation, and to do so, one of these methods needs to be provided: *sum, average, count, max, min* and *stddev* (*standard deviation*).

When ending the aggregation process, the final phase of the workflow corresponds to sending the aggregated data to the Data Lake, where for that step there is the requirement of specifying the topic to send the data, and two fields which contain the stored data, *key* and *value*. The key-value method is used to enable the signature process, meaning that, included in the information sent, there is also an encrypted hash of provided monitoring data, determined before in the MDA, using a key of the Resource Consumer.

Figure 5-8 showcases the Kafka topic, which was received in the configuration, with the stored aggregated and processed data. The figure displays a list with entries structured with the previously described key and value method, and with the timestamp during which the record arrived.



Partition	Offset	Key	Value	Timestamp
0	2	7C5130EEFFF48B468DDEABDF4...	{"operatorID": "domain-operator-a", ...	2022-06-14 14:48:16.088
0	3	88BD1053830B12DEC7D6BEC96...	{"operatorID": "domain-operator-a", ...	2022-06-14 14:50:16.053
0	4	75F1BCCBB464DFD501B582B96...	{"operatorID": "domain-operator-a", ...	2022-06-14 14:54:16.028

Length: 381 bytes [381 characters]

```
{
  "operatorID": "domain-operator-a",
  "transactionID": "ab51f3e1-7b61-4f9d-85a4-9e9f366b593b",
  "networkID": "05cce067-4818-4afc-b0f8-0e4a1babf753",
  "monitoringData": {
    "metricName": "osm_requests",
    "metricValue": -0.51,
    "resourceID": "10",
    "instanceID": "05cce067-4818-4afc-b0f8-0e4a1babf753",
    "productID": "10",
    "timestamp": "2022-06-14 13:50:15.424578",
    "aggregationMethod": "AVG"
  }
}
```

Ready [Messages = 3] [1,159 Bytes] [1,238 ms]

Figure 5-8: Storage of data on the Data Lake

5.3. E-Licensing control

Every software component or xNF onboarded in the 5GZORRO Marketplace as a resource or service offer may include the tied licensing agreements associated, and with them, the definition of the costs derived from the use of this software. This integration work aims to demonstrate the process that begins after the software acquisition by any customer. From this moment, the xNF is ready to be onboarded, deployed, and managed by the platform.

The e-Licensing control configuration is triggered with the purpose of obtaining the relevant information about the use of the software component, in accordance with the agreements that have been signed. Stamping this usage in the blockchain.

Figure 5-9 depicts the complete workflow of the license configuration from the registration to the scheduled monitoring showing the different components involved.

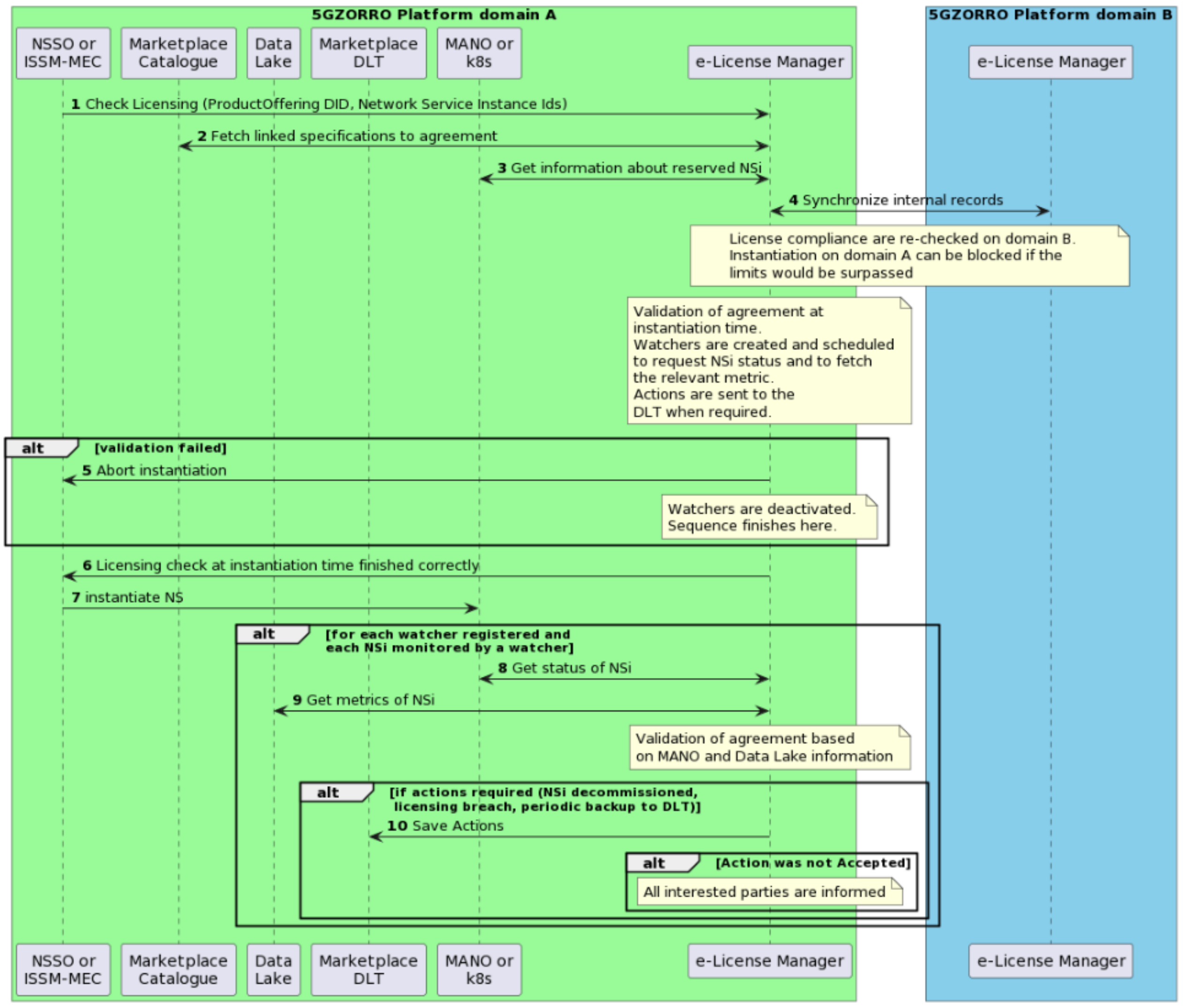

5GZORRO Platform domain A
5GZORRO Platform domain B
NSSO or ISSM-MEC
Marketplace Catalogue
Data Lake
Marketplace DLT
MANO or k8s
e-License Manager
e-License Manager
1 Check Licensing (ProductOffering DID, Network Service Instance Ids)
2 Fetch linked specifications to agreement
3 Get information about reserved NSi
4 Synchronize internal records
License compliance are re-checked on domain B. Instantiation on domain A can be blocked if the limits would be surpassed
Validation of agreement at instantiation time. Watchers are created and scheduled to request NSi status and to fetch the relevant metric. Actions are sent to the DLT when required.
alt [validation failed]
5 Abort instantiation
Watchers are deactivated. Sequence finishes here.
6 Licensing check at instantiation time finished correctly
7 instantiate NS
alt [for each watcher registered and each NSi monitored by a watcher]
8 Get status of NSi
9 Get metrics of NSi
Validation of agreement based on MANO and Data Lake information
alt [if actions required (NSi decommissioned, licensing breach, periodic backup to DLT)]
10 Save Actions
alt [Action was not Accepted]
All interested parties are informed

Figure 5-9: E-License configuration workflow

Before the steps shown in Figure 5-9, the software component/xNF vendor, consumer, the ISSM and Marketplace have already onboarded and configured a valid Order that links Product Offerings (POs), e-licenses and PO Prices:

1. Either the NSSO, in the case the orchestration uses an underlying NFV orchestrator, or the ISSM-MEC, in the case the orchestration is fully Cloud-Native, informs the eLM about a new xNF instance that is in the process of being instantiated with a PO. The endpoint for this step and the response on steps 5 and 6 are defined in an OpenAPI specification in GitHub [38].

2-4. The eLM performs validations at instantiation time which includes fetching the licenses and specifications from the Marketplace, the synchronization with neighbouring instances of the eLM and the creation of the watcher processes that will be monitoring the instance and any internal service that include a license.

5. In the event of steps 2-4 showing that the xNF cannot be instantiated, the NSSO or ISSM-MEC is informed to abort the instantiation. The workflow would stop here.

6-7 The instantiation is accepted and is completed by the NSSO or ISSM-MEC.

At this point, the eLMA (eLM Agent in each specific domain) has created and configured all the relevant watcher processes to continuously monitor the usage of the instances.

8. Each watcher starts its monitoring sequence by requesting the last known status of an instance to check if an event has changed it since the last check.
9. Metrics related to the license are obtained. Tests have been carried out for the case of uptime, number of instances and number of active users (application metric). This is used to validate the limits stablished on the license to ensure that the instance is still compliant.
10. Either by a license violation or as a scheduled update of the DLT, an action representing the relationship between the instance, a PO, a POP, and the current status of the associated metrics is sent out. In the first case or if the action is not correctly persisted in the DLT, the eLMA will alert all the interested parties and request the downstream workflow for such event in the SCLCM.

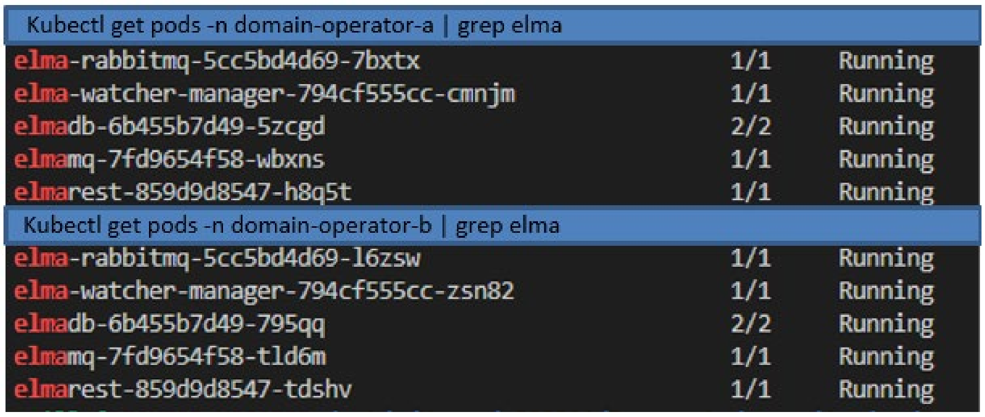
5.3.1. Integration tests and results

The eLM has been integrated with additional components from the 5GZORRO platform with respect to what was described in the previous version of this deliverable (D4.2 [1]). New interactions with the Marketplace catalogue and portal, the ISSM-MEC and the Data Lake have been implemented, which is reported in Table 5-3.

Table 5-3: e-License Manager integration tests

Name	Description	Passed (Yes/No/Partially)
Kubernetes deployment	Helm based deployment of the eLM in the BCN testbed	Yes
Multi-domain communication of eLM	Communication of eLM instances on different administrative domains	Yes
Registration of bundled product offering	Registration of the licenses for an order composed of a bundled product offering from different resource vendors	Yes
Extension of NS to a second domain	License check to allow the instantiation of a second NS using the same licenses	Yes
License restriction breach	Attempt to instantiate the same NS more times than the ones allowed by the licenses	Yes
License expiration	Detection and notification of the expiration of a license	Yes
License for a VNF	Integration with the NSSO to monitor the usage of a VNF in OSM and Openstack	Yes
License for a CNF	Integration with the ISSM-MEC to monitor the usage of a CNF in Kubernetes	Yes

In addition, the eLM has been improved to be deployed in a cloud-native environment using Helm and Kubernetes. Figure 5-10 shows the running pods that build up an instance of the eLM Agent on each administrative domain which correspond to both namespaces of domain-operator-a and domain-operator-b in the Figure. This is consistent with what was introduced by Figure 4-13 with the addition of the internal database (elmadb) and the Rabbitmq broker (elma-rabbitmq) which is shared with other modules of the platform. Being the elmarest pod the entry point to the eLM on each domain, it is exposed using a nodeport service with a fixed port 30880 for domain-operator-a and 31880 for domain-operator-b. This will help understand the logs shown in this section.

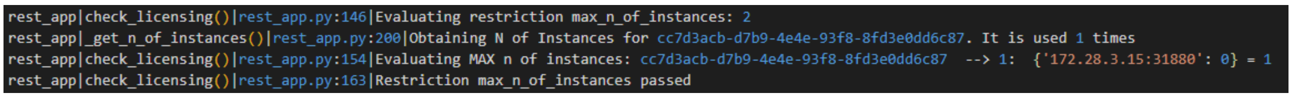


```
Kubectl get pods -n domain-operator-a | grep elma
elma-rabbitmq-5cc5bd4d69-7bxtx              1/1     Running
elma-watcher-manager-794cf555cc-cmnjm       1/1     Running
elmadb-6b455b7d49-5zcgd                     2/2     Running
elmamq-7fd9654f58-wbxns                     1/1     Running
elmarest-859d9d8547-h8q5t                   1/1     Running
Kubectl get pods -n domain-operator-b | grep elma
elma-rabbitmq-5cc5bd4d69-l6zsw              1/1     Running
elma-watcher-manager-794cf555cc-zsn82       1/1     Running
elmadb-6b455b7d49-795qq                     2/2     Running
elmamq-7fd9654f58-tld6m                     1/1     Running
elmarest-859d9d8547-tdshv                   1/1     Running
```

Figure 5-10: Pods of the e-License Manager

On the orchestration part, the interface between eLM and NSSO has not changed and is available in D4.2 [1], while the internal models and technologies of the eLM have been updated during the migration to a fully decentralized architecture and have been reported in D4.4 [19] and D3.3 [51]. The interface with the ISSM-MEC to support a fully cloud-native infrastructure uses the same endpoints and call flows as the NSSO.

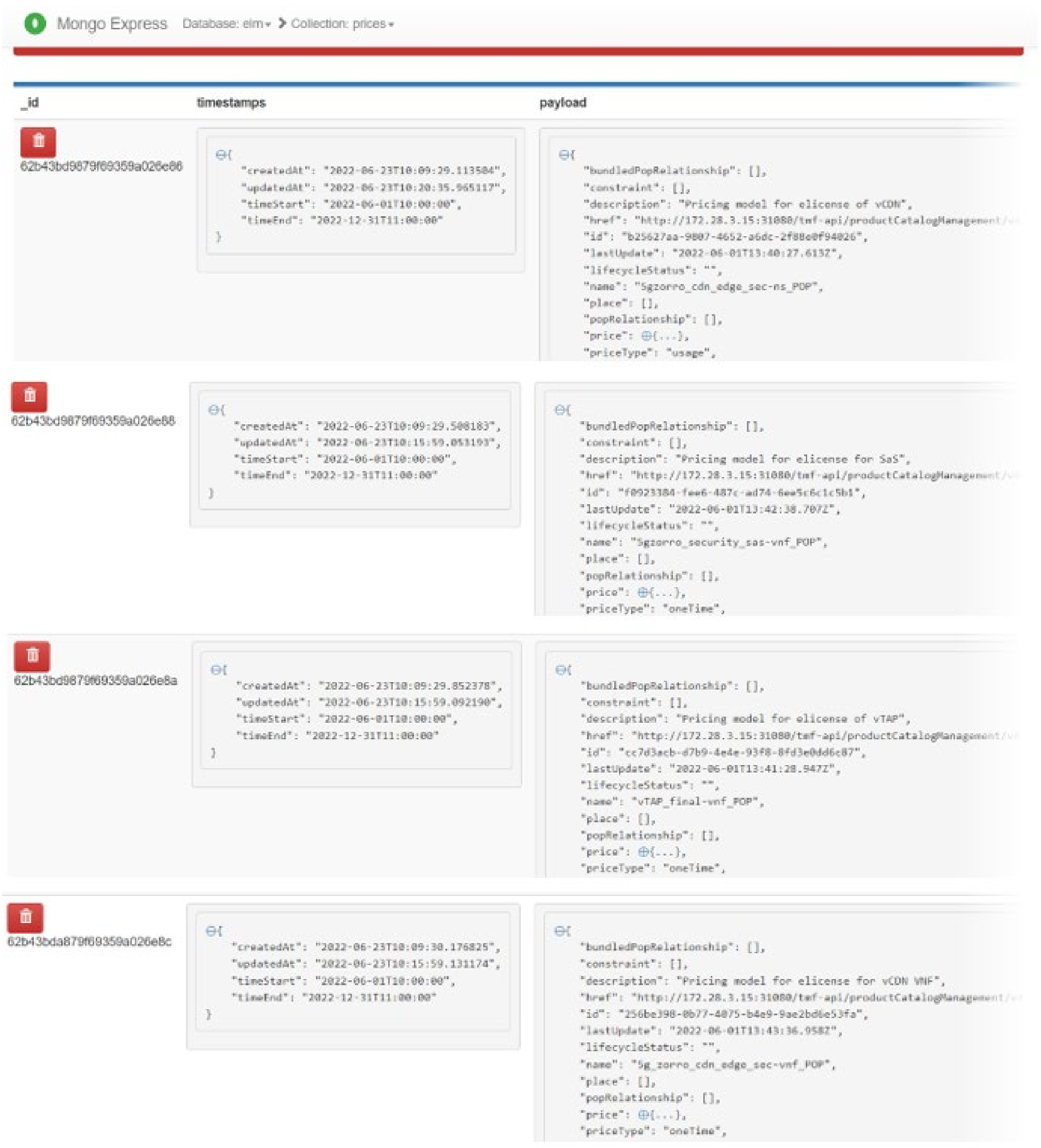
During the registration of the offerings and prices in the eLMA of domain-operator-a, it will contact other eLMAs in neighboring administrative domains through the integration fabric. Figure 5-11 shows the interaction between two eLMAs which confirm to each other that the services and resources comply with the licensing limits, terms, and conditions in an aggregated way. In this example, the e-licenses included a limit on the number of instances or the service that are allowed, this is configured using the *MaxNumberInstances* advanced option in the portal as explained in D3.3 [51].



```
rest_app|check_licensing()|rest_app.py:146|Evaluating restriction max_n_of_instances: 2
rest_app|_get_n_of_instances()|rest_app.py:200|Obtaining N of Instances for cc7d3acb-d7b9-4e4e-93f8-8fd3e0dd6c87. It is used 1 times
rest_app|check_licensing()|rest_app.py:154|Evaluating MAX n of instances: cc7d3acb-d7b9-4e4e-93f8-8fd3e0dd6c87  --> 1:  {'172.28.3.15:31880': 0} = 1
rest_app|check_licensing()|rest_app.py:163|Restriction max_n_of_instances passed
```

Figure 5-11: eLM restriction check success

The interface between the Marketplace catalogue and the eLM has been extended to support bundled products offers inside an order. This feature has been explained in detail in D3.2 [49] and results in the registration of all the licenses that are linked to the bundled offer and thus the creation of individual watchers for each one of them by iterating with the catalogue's API. Figure 5-12 show the internal database of the eLMA after the registration of a bundled order of a NS (5gzorro_cdn_edge_sec-ns) composed of 3 different VNFs (5gzorro_security_sas-vnf, vTAP_final-vnf and 5g_zorro_cdn_edge_sec-vnf) each one with its own pricing model (name of the product offering price appends a "_POP" to the name of the corresponding service or resource).

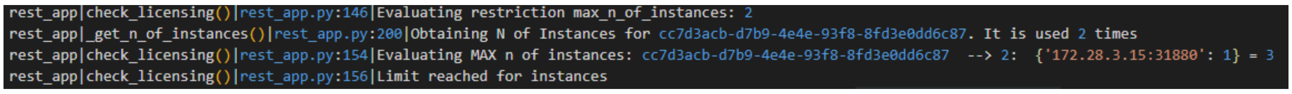


```
_id                        timestamps                                   payload

62b43bd9879f69359a026e86   ⊖{                                           ⊖{
                              "createdAt": "2022-06-23T10:09:29.113584",    "bundledPopRelationship": [],
                              "updatedAt": "2022-06-23T10:20:35.965117",    "constraint": [],
                              "timeStart": "2022-06-01T10:00:00",           "description": "Pricing model for elicense of vCDN",
                              "timeEnd": "2022-12-31T11:00:00"              "href": "http://172.28.3.15:31080/tmf-api/productCatalogManagement/
                           }                                                "id": "b25627aa-9807-4652-a6dc-2f88e0f94026",
                                                                            "lastUpdate": "2022-06-01T13:40:27.613Z",
                                                                            "lifecycleStatus": "",
                                                                            "name": "5gzorro_cdn_edge_sec-ns_POP",
                                                                            "place": [],
                                                                            "popRelationship": [],
                                                                            "price": ⊕{...},
                                                                            "priceType": "usage",

62b43bd9879f69359a026e88   ⊖{                                           ⊖{
                              "createdAt": "2022-06-23T10:09:29.508183",    "bundledPopRelationship": [],
                              "updatedAt": "2022-06-23T10:15:59.053193",    "constraint": [],
                              "timeStart": "2022-06-01T10:00:00",           "description": "Pricing model for elicense for SaS",
                              "timeEnd": "2022-12-31T11:00:00"              "href": "http://172.28.3.15:31080/tmf-api/productCatalogManagement/
                           }                                                "id": "f0923384-fee6-487c-ad74-6ee5c6c1c5b1",
                                                                            "lastUpdate": "2022-06-01T13:42:38.707Z",
                                                                            "lifecycleStatus": "",
                                                                            "name": "5gzorro_security_sas-vnf_POP",
                                                                            "place": [],
                                                                            "popRelationship": [],
                                                                            "price": ⊕{...},
                                                                            "priceType": "oneTime",

62b43bd9879f69359a026e8a   ⊖{                                           ⊖{
                              "createdAt": "2022-06-23T10:09:29.852378",    "bundledPopRelationship": [],
                              "updatedAt": "2022-06-23T10:15:59.092190",    "constraint": [],
                              "timeStart": "2022-06-01T10:00:00",           "description": "Pricing model for elicense of vTAP",
                              "timeEnd": "2022-12-31T11:00:00"              "href": "http://172.28.3.15:31080/tmf-api/productCatalogManagement/
                           }                                                "id": "cc7d3acb-d7b9-4e4e-93f8-8fd3e0dd6c87",
                                                                            "lastUpdate": "2022-06-01T13:41:28.947Z",
                                                                            "lifecycleStatus": "",
                                                                            "name": "vTAP_final-vnf_POP",
                                                                            "place": [],
                                                                            "popRelationship": [],
                                                                            "price": ⊕{...},
                                                                            "priceType": "oneTime",

62b43bda879f69359a026e8c   ⊖{                                           ⊖{
                              "createdAt": "2022-06-23T10:09:30.176825",    "bundledPopRelationship": [],
                              "updatedAt": "2022-06-23T10:15:59.131174",    "constraint": [],
                              "timeStart": "2022-06-01T10:00:00",           "description": "Pricing model for elicense for vCDN VNF",
                              "timeEnd": "2022-12-31T11:00:00"              "href": "http://172.28.3.15:31080/tmf-api/productCatalogManagement/
                           }                                                "id": "256be398-0b77-4075-b4e9-9ae2bd6e53fa",
                                                                            "lastUpdate": "2022-06-01T13:43:36.958Z",
                                                                            "lifecycleStatus": "",
                                                                            "name": "5g_zorro_cdn_edge_sec-vnf_POP",
                                                                            "place": [],
                                                                            "popRelationship": [],
                                                                            "price": ⊕{...},
                                                                            "priceType": "oneTime",
```

Figure 5-12: Bundled offering registration on the eLM

With the objective of extending the number of vendor's business plans compatible with the eLM, e-licenses can be configured to base its monitoring and pricing on application-level metrics in the same way as they can be used to drive the SLA monitoring framework of the 5GZORRO ecosystem.

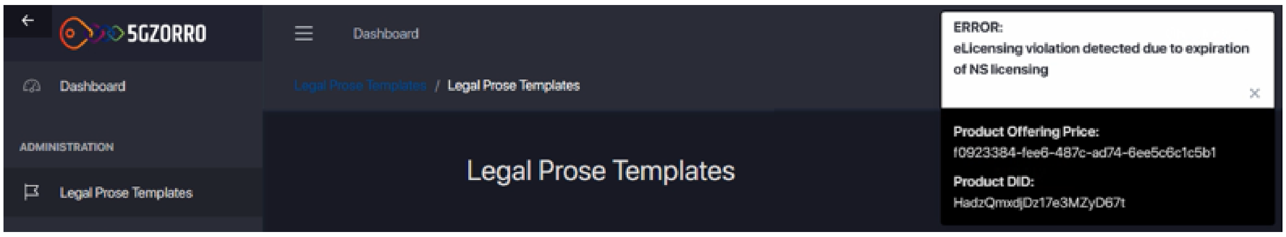
The test continued by extending the coverage of the service on the domain-operator-b which was still permitted under the restrictions configured in the licenses (only 2 instances allowed). Then a new instantiation of the service was requested in the original domain to evaluate the response of the eLM in that scenario, Figure 5-13 show that the eLM finds out that if the new instances is not allowed in its own domain (having 2 then). Then, the aggregated compliance with the restriction would be breached and thus sends back a negative response to the NSSO so that it does not continue with the instantiation of the new service. This would correspond to the call flow number 5 from Figure 5-9.



```
rest_app|check_licensing()|rest_app.py:146|Evaluating restriction max_n_of_instances: 2
rest_app|_get_n_of_instances()|rest_app.py:200|Obtaining N of Instances for cc7d3acb-d7b9-4e4e-93f8-8fd3e0dd6c87. It is used 2 times
rest_app|check_licensing()|rest_app.py:154|Evaluating MAX n of instances: cc7d3acb-d7b9-4e4e-93f8-8fd3e0dd6c87  --> 2:  {'172.28.3.15:31880': 1} = 3
rest_app|check_licensing()|rest_app.py:156|Limit reached for instances
```

Figure 5-13: eLM restriction check failure

An additional interface of the eLM allows the publication of e-license breaches to interested actors, in this case, the portal can show a pop-up on the UI in the event of one of the licenses having expired. For demonstration purposes one of the licenses was set manually to expire shortly after the instantiation of the NS, the feedback received at the Marketplace Portal is shown in Figure 5-14.

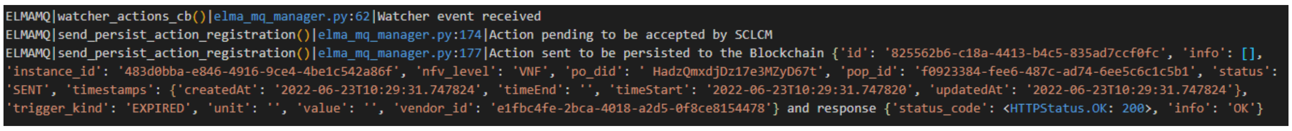


ERROR:
eLicensing violation detected due to expiration of NS licensing

Product Offering Price:
f0923384-fee6-487c-ad74-6ee5c6c1c5b1
Product DID:
HadzQmxdjDz17e3MZyD67t

Figure 5-14: e-License expiration in the portal

The expiration of a license also triggers the delivery of an action to be recorded on the DLT. This is shown on the logs of the elmamq pod from Figure 5-15 which is the one handling all events raised by the watchers.



```
ELMAMQ|watcher_actions_cb()|elma_mq_manager.py:62|Watcher event received
ELMAMQ|send_persist_action_registration()|elma_mq_manager.py:174|Action pending to be accepted by SCLCM
ELMAMQ|send_persist_action_registration()|elma_mq_manager.py:177|Action sent to be persisted to the Blockchain {'id': '825562b6-c18a-4413-b4c5-835ad7ccf0fc', 'info': [],
'instance_id': '483d0bba-e846-4916-9ce4-4be1c542a86f', 'nfv_level': 'VNF', 'po_did': ' HadzQmxdjDz17e3MZyD67t', 'pop_id': 'f0923384-fee6-487c-ad74-6ee5c6c1c5b1', 'status':
'SENT', 'timestamps': {'createdAt': '2022-06-23T10:29:31.747824', 'timeEnd': '', 'timeStart': '2022-06-23T10:29:31.747820', 'updatedAt': '2022-06-23T10:29:31.747824'},
'trigger_kind': 'EXPIRED', 'unit': '', 'value': '', 'vendor_id': 'e1fbc4fe-2bca-4018-a2d5-0f8ce8154478'} and response {'status_code': <HTTPStatus.OK: 200>, 'info': 'OK'}
```

Figure 5-15: eLM action after expiration of license

5.4. Data-driven actuation

The provisioning of a product offer (i.e., a network slice) to an operator through 5GZORRO components, creates a Service Level Agreement (SLA) contract. The SLA contains the quantifiable metrics of the virtual resource that have been agreed upon, called Service Level Objectives (SLO). The creation of the SLA initiates a workflow whose purpose is to trigger a data pipeline that predicts the future values of those SLOs throughout the lifecycle of the SLA, using Machine Learning algorithms. To that end, the following 5GZORRO components are involved:

- Intelligent Slice & Service Manager (ISSM).
- Monitoring Data Aggregator (MDA).
- Smart Contract Lifecycle Manager (SCLCM).
- Data Lake.
- Intelligent SLA Breach Prediction module (ISBP).

The workflow is visible in Figure 5-16.

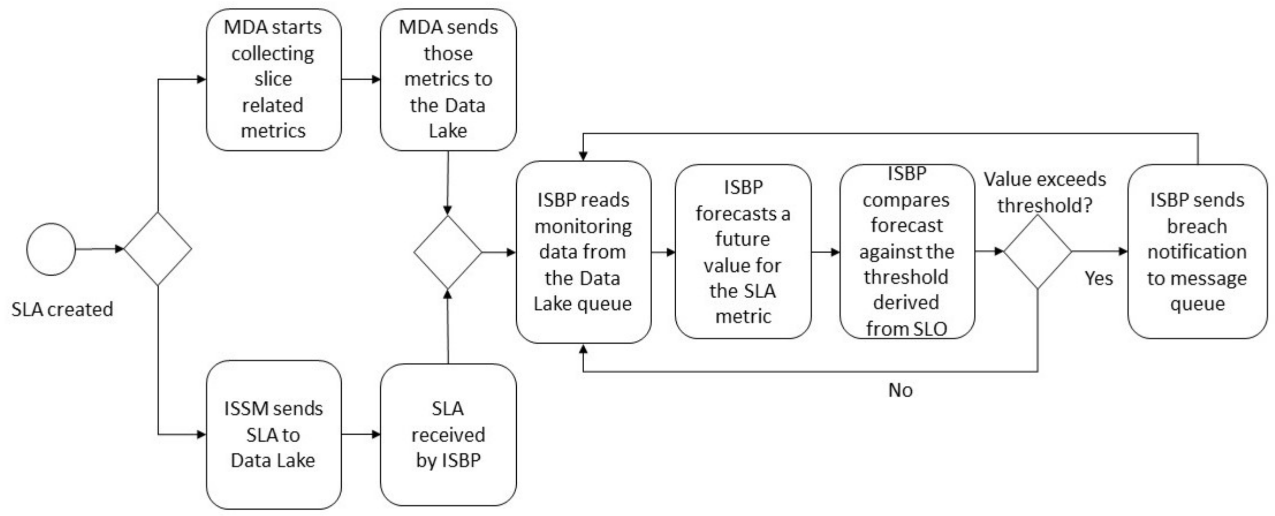

SLA created
MDA starts collecting slice related metrics
MDA sends those metrics to the Data Lake
ISSM sends SLA to Data Lake
SLA received by ISBP
ISBP reads monitoring data from the Data Lake queue
ISBP forecasts a future value for the SLA metric
ISBP compares forecast against the threshold derived from SLO
Value exceeds threshold?
Yes
No
ISBP sends breach notification to message queue

Figure 5-16: SLA Breach prediction workflow

Once the slice is provisioned, the MDA starts collecting monitoring data, and sends them to the Storage Service located in the Data Lake of 5GZORRO. Concurrently, the ISSM sends the SLA to a different message queue also located in the Data Lake. The ISBP, which interacts with the Data Lake and awaits messages from its message queues, parses the message containing the SLA and creates a pipeline that generates future predictions of the SLO metric, based on the monitoring data of the slice. The predictions are generated by an AI model that has been trained on data with similar behaviour to the ones it is predicting. The ISBP also includes functionality that enables the re-training of the AI model if the accuracy of its predictions falls under a given threshold. Finally, if the generated prediction is above the threshold dictated by the SLA, it is packaged in a notification message and pushed to a message queue. This notification can be then picked up by the Intelligent Slice and Service Manager component of 5GZORRO that can then take action to prevent contract breach.

The architecture of the components mentioned above can be seen in Figure 5-17. The MDA and the ISSM, are not part of the Data Lake and interface with the Storage Services and a message queue to push monitoring data and the SLA events respectively. Accordingly, the Data Lake maintains several message queues to implement a stream-based service. As can be seen, each queue stores different kind of data.

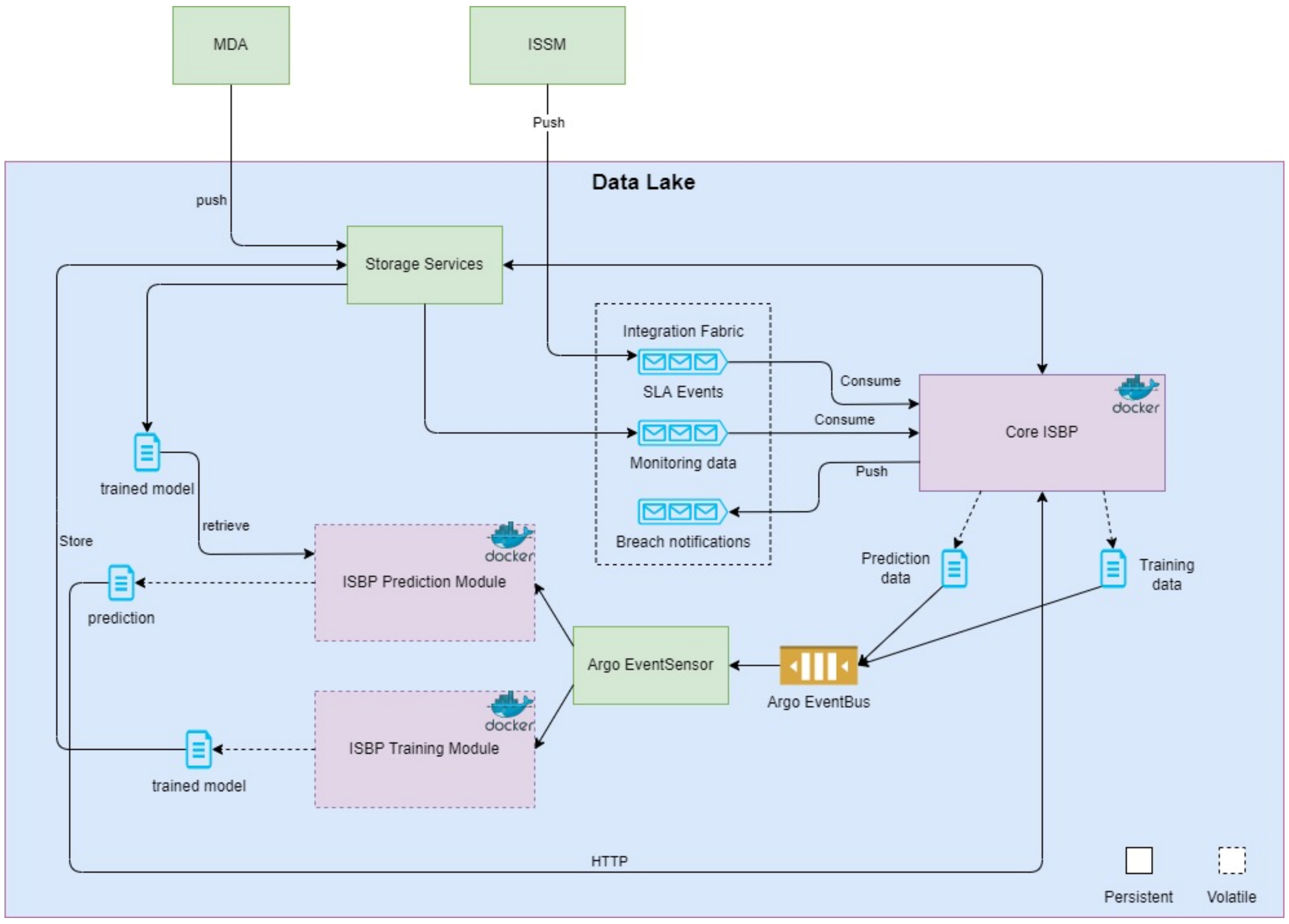


Figure 5-17: ISBP Architecture

The ISBP is located within a Docker container and has connections to the message queues, but also exposes a REST API, implementing a limited number of functions. The Prediction and Training modules are ephemeral containers whose lifecycle depends only on the duration of the task they must complete. The launching of these containers is triggered by events fired by the creation of the files containing the data for the prediction and model training respectively. Upon completion, the Prediction module sends the generated prediction to the ISBP using a simple HTTP request, whereas the Training module stores the new model to the Data Lake storage.

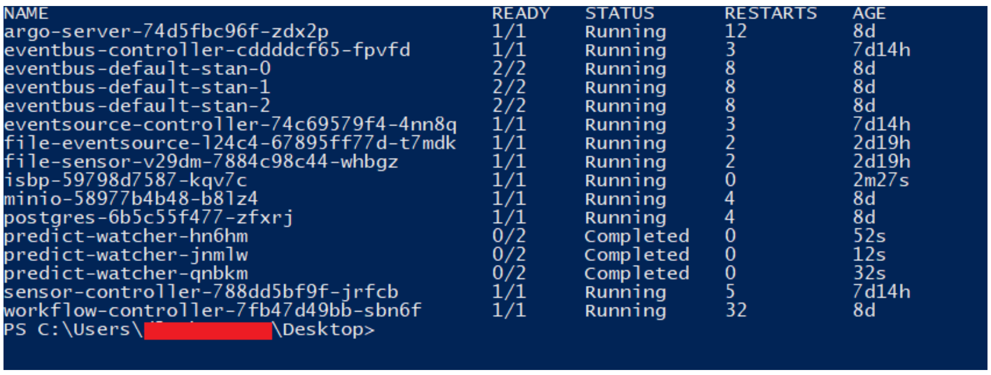
5.4.1. Integration tests and results

The following tests have been conducted in the 5GBarcelona testbed both individually and during end-to-end trials, as listed in Table 5-4.

Table 5-4: Intelligent SLA Breach Predictor functional test set

Name	**Description**	**Passed (Yes/No/Partially)**
Start ML pipeline	Trigger a ML pipeline upon new SLA activation	Yes
Generate predictions	Generate predictions continuously on the incoming data stream	Yes
Notify upon SLA violation prediction	Trigger a notification message and dispatch it to the relevant components when SLA violation is predicted	Yes
Terminate ML pipeline	Terminate pipeline when SLA is concluded/terminated	Yes

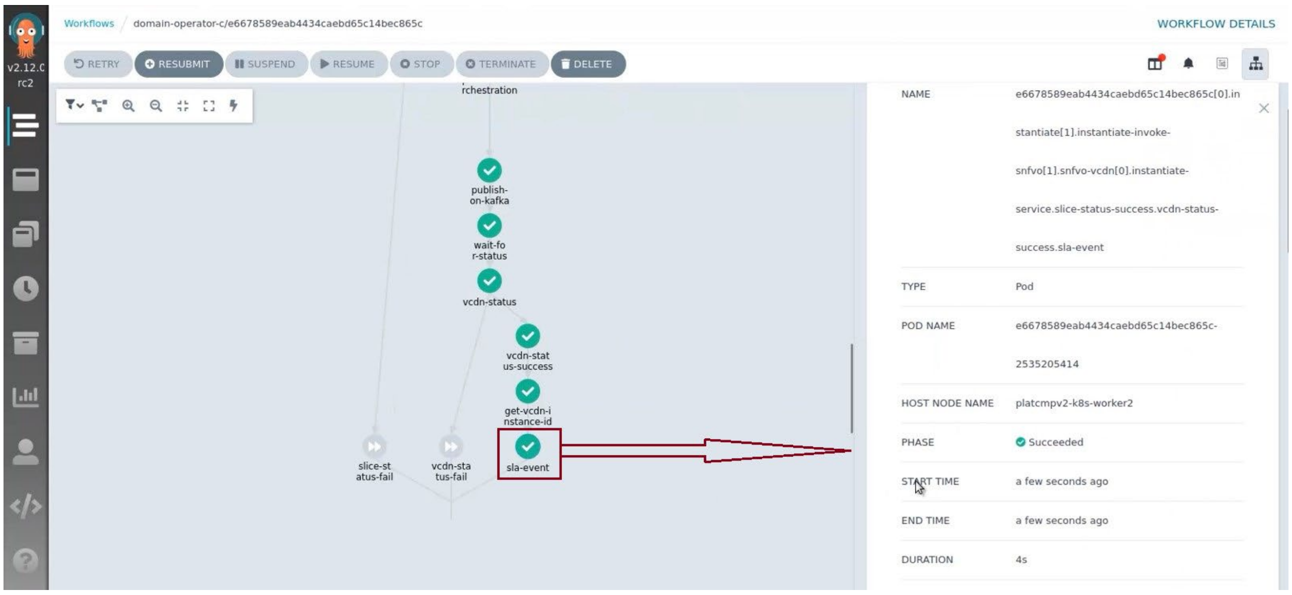
The ISBP component and Prediction/Training modules have been developed and tested in local ICOM premises with a local installation of Apache Kafka as the message queue ISBP connects to. The tests involve an existing dataset consisting of metrics acquired from one of ICOM's servers. These metrics include the server bandwidth, cache hits and request per minute. The data is pushed to Kafka which simulates the workflow discussed above. The results with the ephemeral containers mentioned previously can be seen in Figure 5-18.



```
NAME                                      READY   STATUS      RESTARTS   AGE
argo-server-74d5fbc96f-zdx2p              1/1     Running     12         8d
eventbus-controller-cddddcf65-fpvfd       1/1     Running     3          7d14h
eventbus-default-stan-0                   2/2     Running     8          8d
eventbus-default-stan-1                   2/2     Running     8          8d
eventbus-default-stan-2                   2/2     Running     8          8d
eventsource-controller-74c69579f4-4nn8q   1/1     Running     3          7d14h
file-eventsource-124c4-67895ff77d-t7mdk   1/1     Running     2          2d19h
file-sensor-v29dm-7884c98c44-whbgz        1/1     Running     2          2d19h
isbp-59798d7587-kqv7c                     1/1     Running     0          2m27s
minio-58977b4b48-b8lz4                    1/1     Running     4          8d
postgres-6b5c55f477-zfxrj                 1/1     Running     4          8d
predict-watcher-hn6hm                     0/2     Completed   0          52s
predict-watcher-jnmlw                     0/2     Completed   0          12s
predict-watcher-qnbkm                     0/2     Completed   0          32s
sensor-controller-788dd5bf9f-jrfcb        1/1     Running     5          7d14h
workflow-controller-7fb47d49bb-sbn6f      1/1     Running     32         8d
PS C:\Users\[redacted]\Desktop>
```

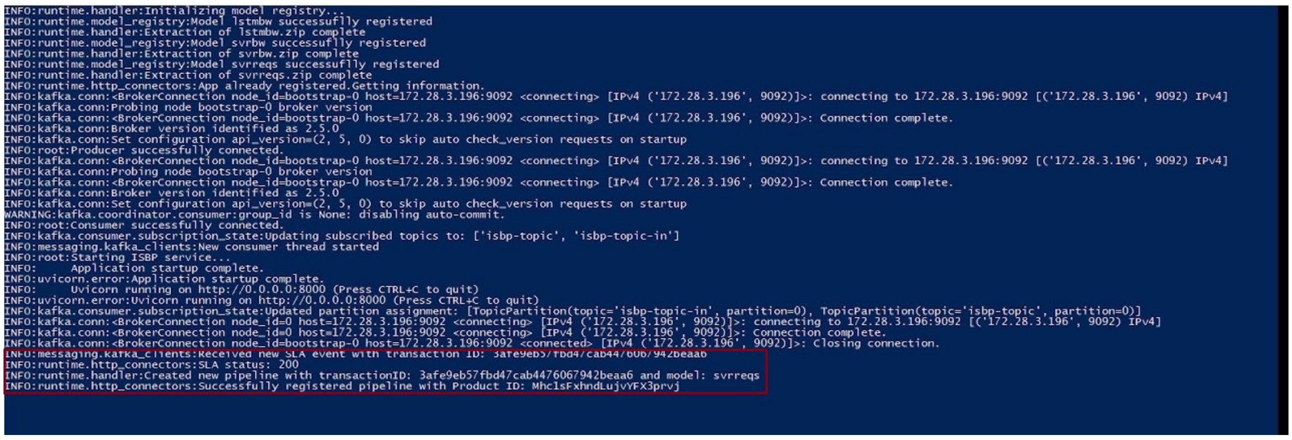
Figure 5-18: ISBP containers

The integration of these components with the rest of the components of the Data Lake, i.e., the message queues and other aspects of the Storage Services is completed. Another component that interacts indirectly with ISBP is the Intelligent Slice and Service Manager (ISSM), as indicated in Figure 5-17. More specifically, when a new SLA is activated and the respective service is instantiated, the ISSM sends an SLA event to the Integration Fabric (see Figure 5-19). This event is captured by the ISBP (Figure 5-20), which then starts gathering data and generate predictions (Figure 5-21). In the example shown in the figures, a CDN slice is monitored, and the metric used is the number of requests, as defined in the SLA.



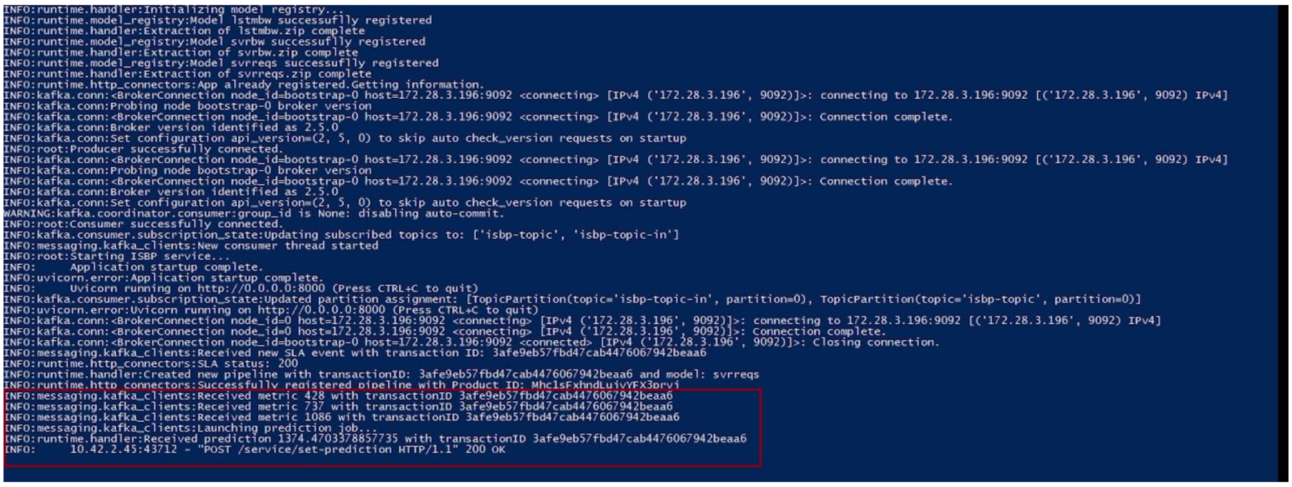
NAME	e6678589eab4434caebd65c14bec865c[0].instantiate[1].instantiate-invoke-snfvo[1].snfvo-vcdn[0].instantiate-service.slice-status-success.vcdn-status-success.sla-event
TYPE	Pod
POD NAME	e6678589eab4434caebd65c14bec865c-2535205414
HOST NODE NAME	platcmpv2-k8s-worker2
PHASE	✓ Succeeded
START TIME	a few seconds ago
END TIME	a few seconds ago
DURATION	4s

Figure 5-19: ISSM creates an SLA event and pushes it to integration fabric



```
INFO:runtime.handler:Initializing model registry...
INFO:runtime.model_registry:Model lstmbw successuflly registered
INFO:runtime.handler:Extraction of lstmbw.zip complete
INFO:runtime.model_registry:Model svrbw successuflly registered
INFO:runtime.handler:Extraction of svrbw.zip complete
INFO:runtime.model_registry:Model svrreqs successuflly registered
INFO:runtime.handler:Extraction of svrreqs.zip complete
INFO:runtime.http_connectors:App already registered.Getting information.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
INFO:root:Producer successfully connected.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
WARNING:kafka.coordinator.consumer:group_id is None: disabling auto-commit.
INFO:root:Consumer successfully connected.
INFO:kafka.consumer.subscription_state:Updating subscribed topics to: ['isbp-topic', 'isbp-topic-in']
INFO:messaging.kafka_clients:New consumer thread started
INFO:root:Starting ISBP service...
INFO:     Application startup complete.
INFO:uvicorn.error:Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:uvicorn.error:Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:kafka.consumer.subscription_state:Updated partition assignment: [TopicPartition(topic='isbp-topic-in', partition=0), TopicPartition(topic='isbp-topic', partition=0)]
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connected> [IPv4 ('172.28.3.196', 9092)]>: Closing connection.
INFO:messaging.kafka_clients:Received new SLA event with transaction ID: 3afe9eb57fbd47cab4476067942beaa6
INFO:runtime.http_connectors:SLA status: 200
INFO:runtime.handler:Created new pipeline with transactionID: 3afe9eb57fbd47cab4476067942beaa6 and model: svrreqs
INFO:runtime.http_connectors:Successfully registered pipeline with Product ID: MhclsFxhndLujvYFX3prvj
```

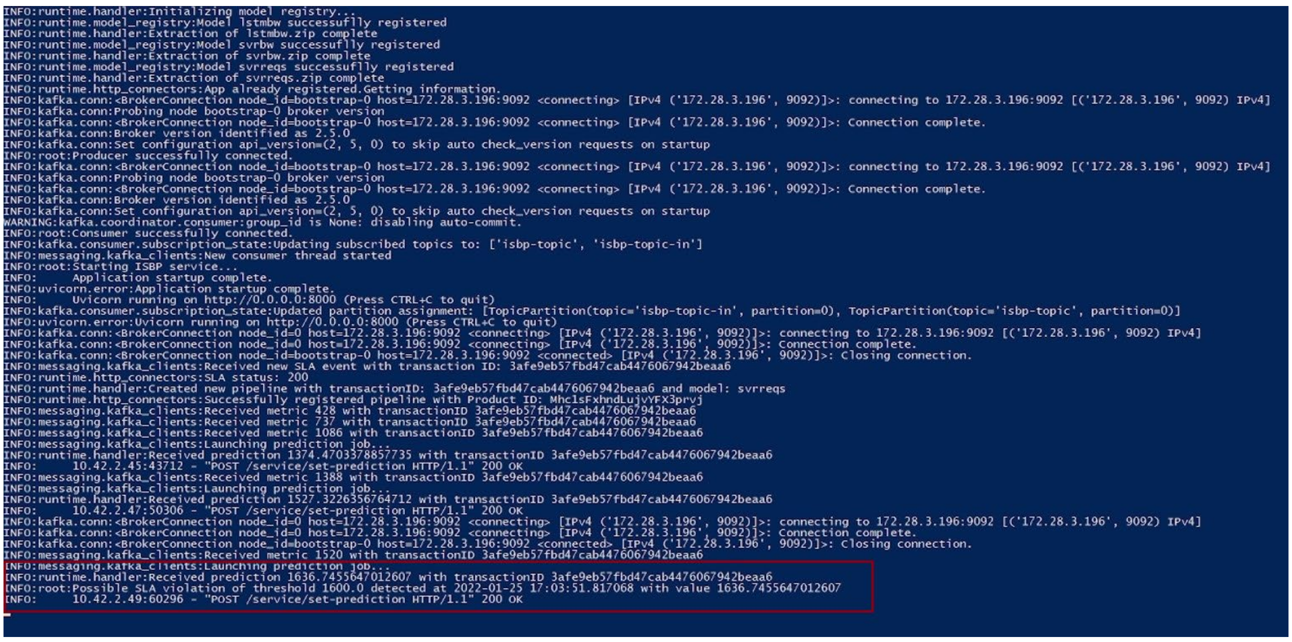
Figure 5-20: ISBP receives the new SLA event



```
INFO:runtime.handler:Initializing model registry...
INFO:runtime.model_registry:Model lstmbw successuflly registered
INFO:runtime.handler:Extraction of lstmbw.zip complete
INFO:runtime.model_registry:Model svrbw successuflly registered
INFO:runtime.handler:Extraction of svrbw.zip complete
INFO:runtime.model_registry:Model svrreqs successuflly registered
INFO:runtime.handler:Extraction of svrreqs.zip complete
INFO:runtime.http_connectors:App already registered.Getting information.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
INFO:root:Producer successfully connected.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
WARNING:kafka.coordinator.consumer:group_id is None: disabling auto-commit.
INFO:root:Consumer successfully connected.
INFO:kafka.consumer.subscription_state:Updating subscribed topics to: ['isbp-topic', 'isbp-topic-in']
INFO:messaging.kafka_clients:New consumer thread started
INFO:root:Starting ISBP service...
INFO:     Application startup complete.
INFO:uvicorn.error:Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:kafka.consumer.subscription_state:Updated partition assignment: [TopicPartition(topic='isbp-topic-in', partition=0), TopicPartition(topic='isbp-topic', partition=0)]
INFO:uvicorn.error:Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connected> [IPv4 ('172.28.3.196', 9092)]>: Closing connection.
INFO:messaging.kafka_clients:Received new SLA event with transaction ID: 3afe9eb57fbd47cab4476067942beaa6
INFO:runtime.http_connectors:SLA status: 200
INFO:runtime.handler:Created new pipeline with transactionID: 3afe9eb57fbd47cab4476067942beaa6 and model: svrreqs
INFO:runtime.http_connectors:Successfully registered pipeline with Product ID: MhclsFxhndLujvYFX3prvj
INFO:messaging.kafka_clients:Received metric 428 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Received metric 737 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Received metric 1086 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Launching prediction job...
INFO:runtime.handler:Received prediction 1374.4703378857735 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:     10.42.2.45:43712 - "POST /service/set-prediction HTTP/1.1" 200 OK
```

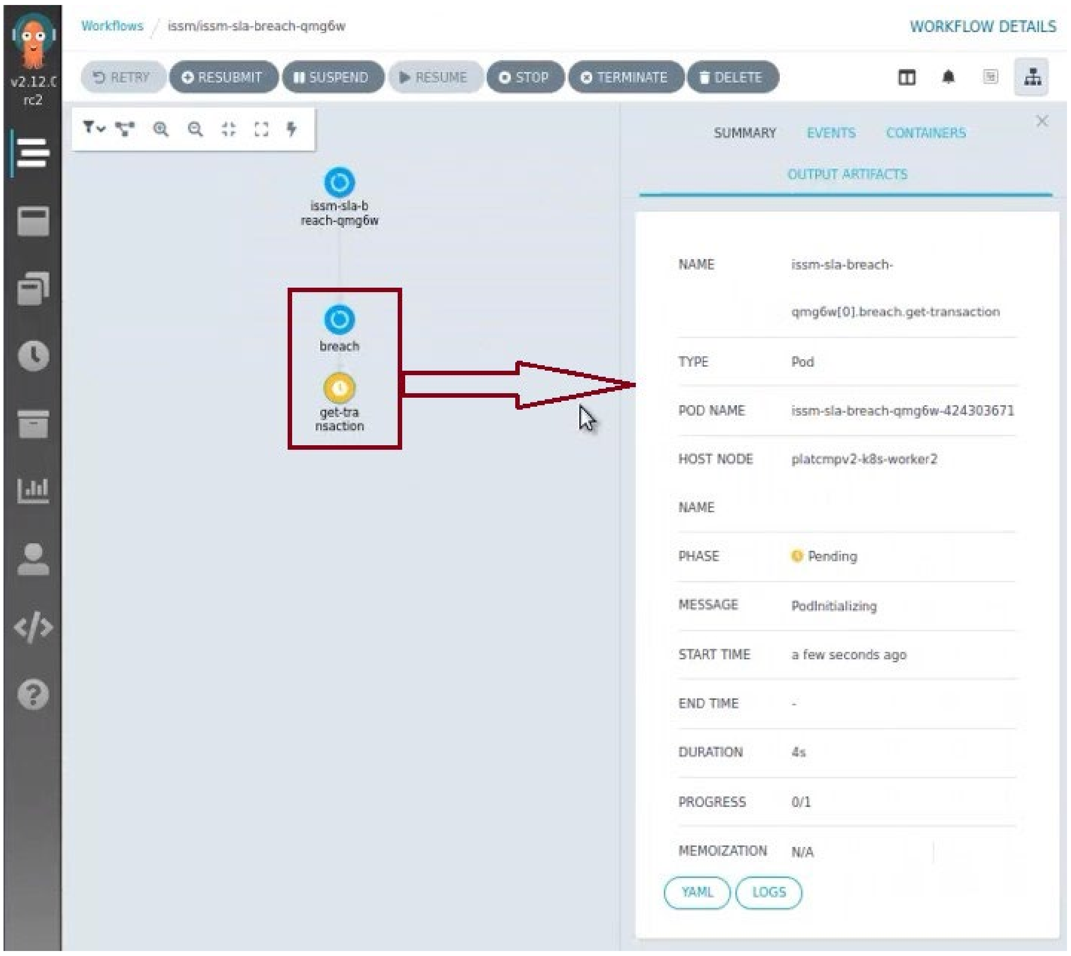
Figure 5-21: ISBP gets monitoring data and generates predictions

When the ISBP predicts that the monitored metric will exceed the defined threshold (Figure 5-22), it generates a breach prediction notification and pushes it to the Integration Fabric, so that the ISSM will receive it (Figure 5-23). After that, the ISSM initiates the processes that will allow the avoidance of the SLA breach. For example, it may trigger the extension of the service and/or the network slice.



```
INFO:runtime.handler:Initializing model registry...
INFO:runtime.model_registry:Model lstmbw successuflly registered
INFO:runtime.handler:Extraction of lstmbw.zip complete
INFO:runtime.model_registry:Model svrbw successuflly registered
INFO:runtime.handler:Extraction of svrbw.zip complete
INFO:runtime.model_registry:Model svrreqs successuflly registered
INFO:runtime.handler:Extraction of svrreqs.zip complete
INFO:runtime.http_connectors:App already registered.Getting information.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
INFO:root:Producer successfully connected.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
WARNING:kafka.coordinator.consumer:group_id is None: disabling auto-commit.
INFO:root:Consumer successfully connected.
INFO:kafka.consumer.subscription_state:Updating subscribed topics to: ['isbp-topic', 'isbp-topic-in']
INFO:messaging.kafka_clients:New consumer thread started
INFO:root:Starting ISBP service...
INFO:     Application startup complete.
INFO:uvicorn.error:Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:kafka.consumer.subscription_state:Updated partition assignment: [TopicPartition(topic='isbp-topic-in', partition=0), TopicPartition(topic='isbp-topic', partition=0)]
INFO:uvicorn.error:Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connected> [IPv4 ('172.28.3.196', 9092)]>: Closing connection.
INFO:messaging.kafka_clients:Received new SLA event with transaction ID: 3afe9eb57fbd47cab4476067942beaa6
INFO:runtime.http_connectors:SLA status: 200
INFO:runtime.handler:Created new pipeline with transactionID: 3afe9eb57fbd47cab4476067942beaa6 and model: svrreqs
INFO:runtime.http_connectors:Successfully registered pipeline with Product ID: Mhc1sFxhndLujvYFX3prvj
INFO:messaging.kafka_clients:Received metric 428 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Received metric 737 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Received metric 1086 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Launching prediction job...
INFO:runtime.handler:Received prediction 1374.4703378857735 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:     10.42.2.45:43712 - "POST /service/set-prediction HTTP/1.1" 200 OK
INFO:messaging.kafka_clients:Received metric 1388 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Launching prediction job...
INFO:runtime.handler:Received prediction 1527.3226356764712 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:     10.42.2.47:50306 - "POST /service/set-prediction HTTP/1.1" 200 OK
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: connecting to 172.28.3.196:9092 [('172.28.3.196', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=0 host=172.28.3.196:9092 <connecting> [IPv4 ('172.28.3.196', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=172.28.3.196:9092 <connected> [IPv4 ('172.28.3.196', 9092)]>: Closing connection.
INFO:messaging.kafka_clients:Received metric 1520 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:messaging.kafka_clients:Launching prediction job...
INFO:runtime.handler:Received prediction 1636.7455647012607 with transactionID 3afe9eb57fbd47cab4476067942beaa6
INFO:root:Possible SLA violation of threshold 1600.0 detected at 2022-01-25 17:03:51.817068 with value 1636.7455647012607
INFO:     10.42.2.49:60296 - "POST /service/set-prediction HTTP/1.1" 200 OK
```

Figure 5-22: ISBP predicts an SLA breach



NAME	issm-sla-breach-qmg6w[0].breach.get-transaction
TYPE	Pod
POD NAME	issm-sla-breach-qmg6w-424303671
HOST NODE NAME	platcmpv2-k8s-worker2
PHASE	Pending
MESSAGE	PodInitializing
START TIME	a few seconds ago
END TIME	-
DURATION	4s
PROGRESS	0/1
MEMOIZATION	N/A

Figure 5-23: ISSM gets the SLA event

6. Installation procedures

As the development phases of the 5GZORRO software components kept increasing, the need to standardize some modules' installation and deployment complexities, due to partners building frameworks in-house, has surged as a key attention point. Since the established project development teams have been working on a continuous delivery approach, consolidating, and aligning installation and deployment steps for all different elements of the platform is crucial to ensure reduction on deployment effort and time.

To serve as basis for the infrastructure of the platform, achieving a high-level of scalability, the Kubernetes system was chosen to enable cluster building of several parts of the platform. It also provides monitoring mechanisms which allow the overview of requirements of applications and resources, and the orchestration of data storage.

According to the installation procedures described in Deliverable D3.2 [49], the components are organized into profiles depending on their specific platform usage scenarios. These are:

- Cross-domain platform profile: Assigned to software components that enable common workflows in 5GZORRO.
- Administrator profile: Dedicated to components used for governance-centric operations and other tasks related to governance activities.
- Trader profile: A profile which represents a common Stakeholder participating in the 5GZORRO ecosystem. It uses components that enable resource & service provisioning, as well as consumption activities.
- Regulator profile: Corresponds to all software components used by a Stakeholder of the platform with the regulator role.

For the software components and prototypes reported in this document, the Cross-domain and the Trader profiles are the ones to be applied to them during the installation phase. More information regarding deployment approach and profile establishment can be found in the deliverable D3.2 [49].

6.1. Cross-domain profile

This profile includes the components of the platform that provide zero-touch automation services and should be deployed as a pre-requisite dependency to all the other profiles. The profile includes, in particular:

- ISSM: Responsible for executing orchestration workflows in a context of a business transaction, such as extending a slice across a second domain, in cooperation with the Network Slice and Service Orchestration.
- ETSI OSM: Orchestration stack that enables Virtualized Network Function deployments by configuring associated information models.

6.2. Trader profile

The Trader profile comprises the components required for slice & service management, to allow the zero-touch automation services to acquire security and trust capabilities in multi-tenant and multi-stakeholder environments. This profile includes, in particular:

- 5G-enabled Trust & Reputation Management Framework: Manages the computation of trust values among different stakeholders based on a trust-chain established with entities.

- VPN-as-a-Service: Establishes secure and trusted connections between different domains in the 5GZORRO environment.
- NSSO: Responsible for the end-to-end vertical service management, handling the decomposition and mapping of the service into network slices and network slice instances across multiple domains.
- NSMM: Enables secure connection establishment between slices/network services.
- xRM: Interacts with the underlying 5G Virtualized Platform to enable resource monitoring and management
- eLicensing Manager: Provides operators and software vendors/consumers the mechanisms to control the usage of the vendors' software products.

6.3. Components mapping

Table 6-1 displays the deployment configurations of the different profiles with respect to the components described in this document.

Table 6-1: Zero-touch service management components required per stakeholder role

	Cross-domain	**Admin**	**Regulator**	**Trader**
ISSM	X			
5G-enabled Trust & Reputation Management Framework (5G-TRMF)				X
VPN-as-a-Service				X
NSSO				X (NSSO-VS)
NSMM				X
xRM				X
eLM	X (core)			X (agent)
ETSI OSM	X			

7. Conclusions

This deliverable reports on the final activities related to implementation and integration works performed with the modules that constitute the 5GZORRO platform in support of the zero-touch service management capabilities with security and trust. The document has taken basis from the final design of the related platform components reported in D4.4 [19], as well as from the preliminary prototypes delivered with D4.2 [1]. On top of these, and reports the final evolutions and updates implemented to provide full functioning zero-touch service management components integrated in the 5GZORRO platform. In practice, the document collects supported functionalities and prototype implementation details of the software produced in WP4, with links to available documentation and code repositories related to the various components released on GitHub.

The latest integration work amongst the partners has resulted in new validation tests which can be found structured across the sections of this document tackling zero-touch orchestration, slice instantiation and data-driven operation automation, security and trust orchestration, e-licensing control, and management.

As a result of the progress shown from the implemented integration methodology, the previous setup of the development teams has been further improved, with the goal of aiding and supporting the integration activities among the involved parties, improving focus, and bringing them more closely to the overall scope of the integrations.

In summary, the software prototypes described in this document corresponds to the final release consolidated in the context of WP4. Any fixes and adaptations on the various modules derived from ongoing and subsequent integration and validation activities carried out in WP5 (*Validation through Use Cases*), will be applied when required as updates to the software code and documentation material included in corresponding GitHub repositories.

References

[1] 5GZORRO Consortium, Deliverable D4.2 - Intermediate prototype of Zero Touch Service Mgmt with Security and Trust.

[2] Xiong, L., & Liu, L. (2004) - Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. IEEE transactions on Knowledge and Data Engineering, 16(7), 843-857.

[3] SCONE – A secure container environment - https://scontain.com/index.html?lang=en

[4] Intel Software Guard Extensions – https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

[5] Azure Cloud – SGX powered Servers - https://azure.microsoft.com/en-us/blog/dcsv2series-vm-now-generally-available-from-azure-confidential-computing/ Accessed 27 January 2021.

[6] Zeek network security monitor - https://zeek.org/

[7] Filebeat log shipper - https://www.elastic.co/beats/filebeat

[8] Elasticsearch search and analytics engine - https://www.elastic.co/elasticsearch/

[9] Kibana - https://www.elastic.co/kibana.

[10] Flask - https://flask.palletsprojects.com/en/1.1.x

[11] Gevent Python networking library - http://www.gevent.org/

[12] Werkzeug WSGI server - https://werkzeug.palletsprojects.com/en/1.0.x/

[13] Haga, S., Esmaeily, A., Kralevska, K., & Gligoroski, D. (2020, November) - 5G Network Slice Isolation with WireGuard and Open Source MANO: A VPNaaS Proof-of-Concept. In 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) (pp. 181-187). IEEE.

[14] Bollapragada, V., Khalid, M., & Wainner, S. (2005) - IPSec VPN Design. Cisco Press.

[15] Feilner, M. (2006) - OpenVPN: Building and integrating virtual private networks. Packt Publishing Ltd.

[16] 5GZORRO Consortium, Deliverable D2.4 – Final design of the 5GZORRO Platform for Security & Trust.

[17] 5GZORRO Consortium, Deliverable D2.3 – Update Design of the 5GZORRO Platform for Security & Trust.

[18] ETSI MEC - https://www.etsi.org/technologies/multi-access-edge-computing

[19] 5GZORRO Consortium, Deliverable D4.4 - Final Design of Zero Touch Service Management with Security and Trust Solutions.

[20] free5GC - https://www.free5gc.org/

[21] CPLEX - https://www.ibm.com/products/ilog-cplex-optimization-studio

[22] GUROBI - http://www.gurobi.com/

[23] GLPK - http://www.gnu.org/software/glpk/glpk.html

[24] CBC - https://github.com/coin-or/Cbc

[25] MOSEK - https://www.mosek.com/

[26] XPRESS - https://www.fico.com/es/products/fico-xpress-solver

[27] CHOCO - https://choco-solver.org/

[28] MIPCL - https://github.com/ajenter/mipcl

[29] SCIP - https://www.scipopt.org/

[30] TM Forum - https://www.tmforum.org/

[31] Gravitee - https://www.gravitee.io/

[32] GS NFV-IFA 014 - https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-IFA%20014v4.2.1%20-%20GS%20-%20Network%20Service%20Templates%20Spec.pdf

[33] ETSI GS NFV-SOL 006 V3.3.1 (2020-08): Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on YANG Specification.

[34] 5GCity – A distributed cloud & radio platform for 5G Neutral Hosts - https://www.5gcity.eu/

[35] Network Slice Blueprint Definition - https://www.ngmn.org/wp-content/uploads/160113_NGMN_Network_Slicing_v1_0 page 6.

[36] NXW slicer, 5GZORRO-core-1.0-alfa release - https://github.com/nextworks-it/slicer/tree/5gzorro-core-1.0-rc

[37] MDA OpenAPI specification - https://github.com/5GZORRO/mda/blob/main/doc/openapi.json

[38] E-Licensing OpenAPI specification - https://github.com/5GZORRO/elicensing-manager-agent/blob/master/elicensemanageragent/swagger.yaml

[39] OSM - https://osm.etsi.org/

[40] Slice Manager GitHub space - https://github.com/5GZORRO/slice-manager

[41] NSMM GitHub space - https://github.com/5GZORRO/network-service-mesh-manager

[42] NSMM NBI - https://github.com/5GZORRO/network-service-mesh-manager/tree/main/api

[43] Go - https://go.dev/

[44] Gin Gonic - https://gin-gonic.com/

[45] Codegen - https://pkg.go.dev/github.com/deepmap/oapi-codegen/pkg/codegen

[46] GORM - https://gorm.io/index.html

[47] PostgreSQL JDBC Driver - https://jdbc.postgresql.org/

[48] Gophercloud - http://gophercloud.io/

[49] 5GZORRO Consortium, Deliverable D3.2 - Prototypes of evolved 5G Service layer solutions.

[50] 5GZORRO GitHub space - https://github.com/5GZORRO

[51] 5GZORRO Consortium, Deliverable D3.3 - Final design of the evolved 5G Service layer solutions.

[52] eLM OpenAPI - https://github.com/5GZORRO/elicensing-manager-agent/blob/master/elicensemanageragent/swagger.yaml

[53] 5GZORRO Consortium, Deliverable D4.1 - Design of Zero Touch Service Management with Security & Trust Solutions.

[54] 5GZORRO Consortium, Deliverable D3.1 - Design of the evolved 5G Service layer solutions.

[55] Kubernetes - https://kubernetes.io/

[56] ETSI OSM MANO orchestration framework - https://osm.etsi.org/

8. Abbreviations and Definitions

8.1. Abbreviations

API	Application Programming Interface
CLI	Command Line Interface
DID	Distributed Identifier
GST	Generic Slice Template
MANO	Management and Orchestration
MNO	Mobile Network Operator
NBI	North Bound Interface
NEST	Network Slice Type
NFVO	Networks Function Virtualization Orchestrator
NSI	Network Service Instance
OSM	Open Source MANO
PLMN	Public Land Mobile Network
RAN	Radio Access Network
REST	Representational State Transfer
SBI	South Bound Interface
SSID	Service Set Identifier
TAP	Terminal Access Point
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
VPN	Virtual Private Network
WP	Work Package
WSGI	Web Server Gateway Interface

9. Appendix I – Trust Management Framework

9.1. 5G-TRMF Information Model

The redesigned information model of the 5G-TRMF is presented in Figure 9-1. It presents the set of characteristics that are interpreted by the PeerTrust reputation model to generate a final score for each trust computation request.

The Table 9-1, Table 9-2 and Table 9-3 present the related Information Models.

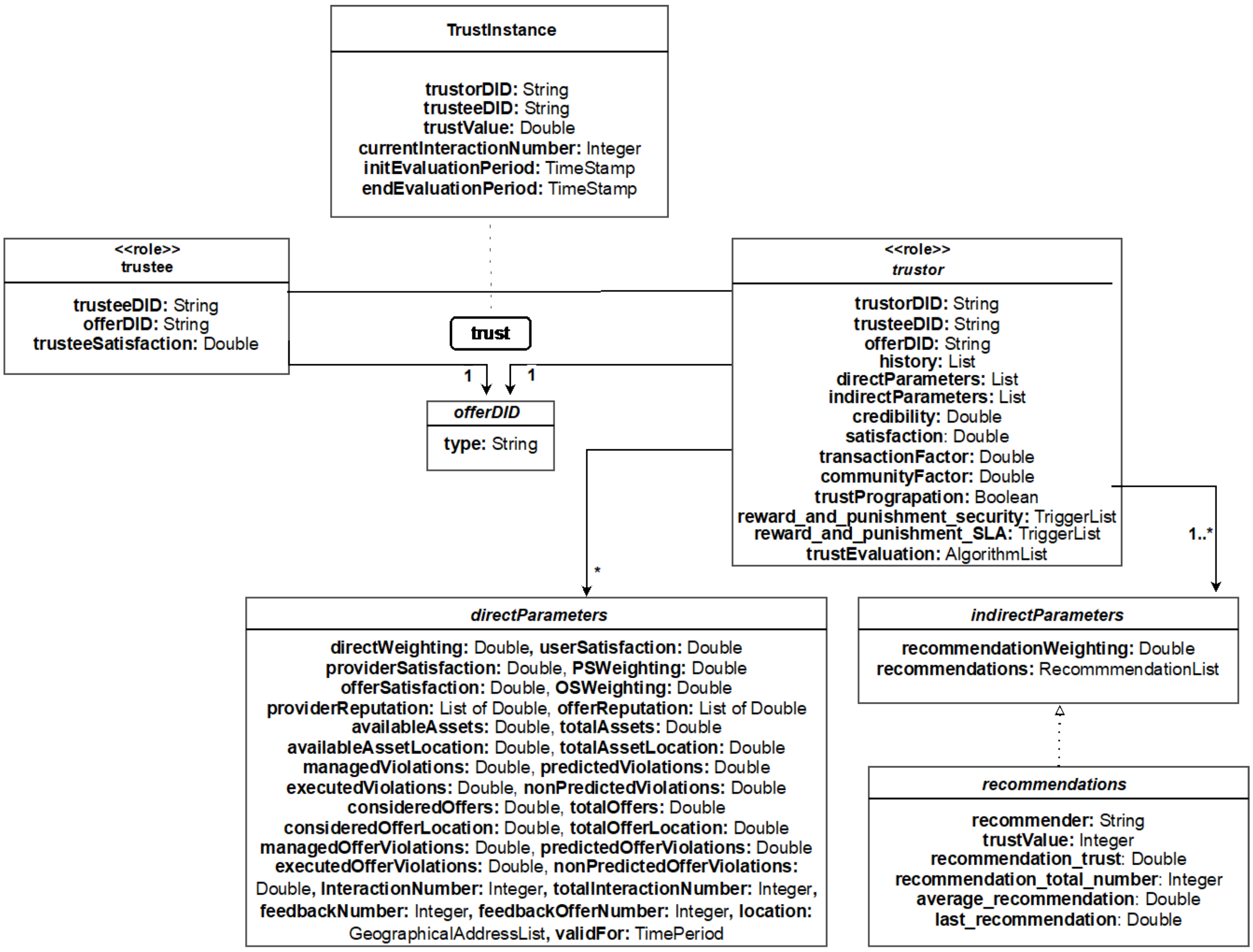

TrustInstance
trustorDID: String
trusteeDID: String
trustValue: Double
currentInteractionNumber: Integer
initEvaluationPeriod: TimeStamp
endEvaluationPeriod: TimeStamp
<<role>>
trustee
trusteeDID: String
offerDID: String
trusteeSatisfaction: Double
trust
1
1
offerDID
type: String
<<role>>
trustor
trustorDID: String
trusteeDID: String
offerDID: String
history: List
directParameters: List
indirectParameters: List
credibility: Double
satisfaction: Double
transactionFactor: Double
communityFactor: Double
trustPrograpation: Boolean
reward_and_punishment_security: TriggerList
reward_and_punishment_SLA: TriggerList
trustEvaluation: AlgorithmList
*
1..*
directParameters
directWeighting: Double, userSatisfaction: Double
providerSatisfaction: Double, PSWeighting: Double
offerSatisfaction: Double, OSWeighting: Double
providerReputation: List of Double, offerReputation: List of Double
availableAssets: Double, totalAssets: Double
availableAssetLocation: Double, totalAssetLocation: Double
managedViolations: Double, predictedViolations: Double
executedViolations: Double, nonPredictedViolations: Double
consideredOffers: Double, totalOffers: Double
consideredOfferLocation: Double, totalOfferLocation: Double
managedOfferViolations: Double, predictedOfferViolations: Double
executedOfferViolations: Double, nonPredictedOfferViolations:
Double, InteractionNumber: Integer, totalInteractionNumber: Integer,
feedbackNumber: Integer, feedbackOfferNumber: Integer, location:
GeographicalAddressList, validFor: TimePeriod
indirectParameters
recommendationWeighting: Double
recommendations: RecommmendationList
recommendations
recommender: String
trustValue: Integer
recommendation_trust: Double
recommendation_total_number: Integer
average_recommendation: Double
last_recommendation: Double

Figure 9-1 : UML diagram of 5G-TRMF

Table 9-1: 5G-TRMF Instance Information Model

Parameter	Type	Description
trustorDID	String	Unique identifier for a resource or service consumer.
trusteeDID	String	Unique identifier for a resource or service provider.
trustValue	Double	Current trust value assigned.
currentInteractionNumber	Int	Number of interactions between the trustor and the trustee.
initEvaluationPeriod	TimeStamp	The time when trust value was generated.
endEvaluationPeriod	TimeStamp	The time when trust value will be over and has to be reassigned if required.

Table 9-2: Trustee Entity Information Model

Parameter	Type	Description
trusteeDID	String	Unique identifier for a resource or service provider.
offerDID	String	Unique identifier for a particular product offer of the provider.
type	String	Kind of offer (RAN, spectrum, VNF/CNF, slice, or edge)
trusteeSatisfaction	Double	Truestee's satisfaction after x interactions with other providers.

Table 9-3: Trustor Entity Information Model

Parameter	Type	Description
trustorDID	String	Unique identifier for a resource or service consumer.
trusteeDID	String	Unique identifier for a resource or service provider.
offerDID	String	Unique identifier for a particular product offer.
type	String	Kind of offer (RAN, spectrum, VNF/CNF, slice, or Edge)
history	List (double)	Set of trust evaluations about an entity.
directParameters	List of key-value features	Dictionary with direct trust data to calculate trust level.
directWeighting	Double	Direct weighting parameter.
userSatisfaction	Double	Internal assessment of the service or resource provided by a stakeholder (trustor).
***provider**Satisfaction*	Double	Trustor satisfaction in a third-party provider (trustee).
PSWeighting	Double	Weighting factor $\in$ [0,1], PS + OS = 1
offerSatisfaction	Double	Trustor satisfaction in a particular kind of offer of a third-party provider.
OSWeighting	Double	Weighting factor $\in$ [0,1], PS + OS = 1
providerReputation	List (double)	Set of previous trust evaluations about a provider.
offerReputation	List (double)	Set of previous trust evaluations about a specific kind of offer of a provider.
availableAssets	Integer	The available assets (services and resources) of the trusteeDID when the trustor is determining the reputation.
totalAssets	Integer	The total assets of the trusteeDID when the trustor is determining the reputation (active and inactive).
availableAssetLocation	Integer	The available assets of the trusteeDID, at a specific location, when the trustor is determining the reputation.

totalAssetLocation	Integer	The total assets of the trusteeDID, at a specific location, when the trustor is determining the reputation (active and inactive).
managedViolations	Integer	The total number of predicted SLA violations that were finally managed successful, associated with the trusteeDID.
Parameter	Type	Description
predictedViolations	Integer	The total number of predicted SLA violations associated with the trusteeDID.
executedViolations	Integer	The total number of predicted SLA violations that were finally managed unsuccessful (violation), associated with the trusteeDID.
nonpredictedViolations	Integer	The number of SLA violations that were not predicted and turned out to be SLA violations, associated with the trusteeDID.
consideredOffers	Integer	The number of offers considered by the Smart Resource and Service Discovery (SRSD), for a particular type of offer, from the trusteeDID when trustor is determining the reputation.
consideredOfferLocation	Integer	The available number of a particular offer type from the trusteeDID when trustor is determining the reputation.
totalOfferLocation	Integer	The number of offers considered by the Smart Resource and Service Discovery (SRSD) for a particular type of offer from the trusteeDID, at a specific location, when trustor is determining the reputation.
managedOfferViolations	Integer	The available number of a particular offer type from the trusteeDID, at a specific location, when trustor is determining the reputation.
predictedOfferViolations	Integer	The total offer number (for a particular kind of offer) of predicted SLA violations that were finally managed successful, associated with the trusteeDID.
executedOfferViolations	Integer	The total offer number (for a particular kind of offer) of predicted SLA violations associated with the trusteeDID.
nonpredictedOfferViolations	Integer	The total offer number (for a particular kind of offer) of predicted SLA violations that were finally managed unsuccessful (violation), associated with the trusteeDID.
interactionNumber	Integer	The total offer number (for a particular kind of offer) of SLA violations that were not predicted and turned out to be SLA violations, associated with the trusteeDID.
totalInteractionNumber	Integer	Number of interactions carried out by the trusteeDID with the other domains.
feedbackNumber	Integer	Number of feedbacks made by other providers about the trusteeDID.
feedbackOfferNumber	Integer	Trustor satisfaction in a third-party provider (trustee).
location	List of GeographicalAddress objects	It constitutes a group of GeographicalAddress.
validFor	TimePeriod	The period for which this resource or service is valid.

indirectParameters	List of key-value features	Dictionary with indirect trust data to calculate trust level.
recommendationWeighting	Double	Recommender's weighting parameter(s).
recommendations	List of recommendations	Set of recommendation about a third entity from one or more external entities.
Parameter	Type	Description
recommender	String	Unique identifier for a recommender.
trust_value	Int	Final trust score ranged from 0.0 to 1.0.
recommendation_trust	Double	Trust level of the trustor in the recommendation.
recommendation_total_number	Int	Number of recommendations of a given trustee.
average_recommendation	Double	Average of all recommendations.
last_recommendation	Double	The last recommendation.
credibility	Double	Factor that determines how accurate the recommendations are.
satisfaction	Double	Trustor satisfaction in a specific trustee.
transactionFactor	Double	Intra or inter-domain trust score and parameterTuple propagation (0 means intra, 1 means inter)
communityFactor	Double	It indicates the triggers to recompute trust score.
trustPropagation	Double	It identifies different evaluation algorithms.
reward_and_punishment_security	List of triggers	Increase or decrease score based on current events such as security, time-decay, etc.
reward_and_punishment_SLA	List of triggers	Increase or decrease score based on current events such as breach predictions and detections, time-decay, etc.
trust_evaluation	List of algorithms	It identifies different evaluation algorithms such as PeerTrust reputation model.

9.2. 5G-enabled Trust and Reputation Management Framework Equations

9.2.1. General PeerTrust equation

The PeerTrust model is composed of a main equation that allows representing how a *domain v* can evaluate the trust score of a *domain u*. In particular, the principal equation is as follow:

$$T(u) = \alpha * \left(\frac{\sum_{i=1}^{I(u)} S(u,i) * Cr\big(p(u,i)\big) * TF(u,i)}{I(u)}\right) + \beta * CF(u)$$

- α , β depict weighting factors to be considered by the *domain v*. $\alpha, \beta \in [0,1]$and $\alpha + \beta = 1$. It is advisable that α has a higher value than β.
- $I(u)$ is the total interaction number of *domain u* with the rest of domains.
- $p(u,i)$ denotes the rest of domains participating in the i-th interaction with the *domain u*.
- $S(u,i)$ represents the normalized value of *satisfaction* that the *domain u* obtains from $p(u,i)$ in its i-th interaction.
- $Cr(v)$ references the *domain v*'s credibility has in the *domain u*'s opinion.
- $TF(u,i)$ is the context *factor* adapted on the i-th *transaction* of *domain u*.

- $CF(u)$ indicates the context *factor* adapted on the *community* of entities to which the *domain u* belongs.

9.2.1.1. Satisfaction equation

Once the principal equation has been introduced (see section above), each of the parts that make up the main equation will be split into sub-equations to explain in more detail how each of the parameters is evaluated. The next equation defines the *domain u*'s satisfaction on a product offer published by a particular DPn the i-th interaction.

$$S(u,i) = \gamma * PS(u,i) + \varphi * OS(u,i)$$

- γ, φ depict weighting factors to be considered by the *domain u*. $\gamma, \varphi \in [0,1]$ and $\gamma + \varphi = 1$.
- $PS(u,i)$ is the *satisfaction* that the *domain u* has on the i-th *domain* (*provider*).
- $OS(u,i)$ is the *satisfaction* that the *domain u* has on the i-th *domain*'s *offer*.
- It should be pointed out that $PS(u,i) + OS(u,i) = 1$.

The provider's satisfaction of the *domain u* on the *i-th* interaction will be computed about the *domain j* stakeholder.

$$PS(u,j) = Rep(u,j) * \bigoplus_{x=1}^{n} Rec(x,j) * T^{(t-1)}(u,x)$$

- $\oplus$ is an aggregation operation such as Minimum value, Maximum value, Arithmetic mean, or Harmonic mean. n denotes the rest of domains participating in the x-th interaction with the *domain j*.
- $Rec(x,j)$ is the recommendation of the x-th domain with which the *domain j* has a trust relationship. In other words, $T(x,j)$.
- $T^{(t-1)}(u,x)$ is the last trust score that the *domain u* has on the *domain x*.

The $Rep(u,j)$ is the average reputation that the *domain u* has on the *provider j* based on all assets (service and resources). This reputation contemplates features such as available assets, assets in a particular location, and multiple time windows to compute these features along the *provider j* lifecycle.

$$Rep(u,j) = \sum_{k=1}^{n} \varepsilon(k) * \frac{\left(\frac{AA(j)}{IA(j)} + \frac{AAL(j)}{IAL(j)} + 2 * \frac{MV(j)}{PV(j)} - 2 * \frac{EV(j)+NPV(j)}{PV(j)}\right) + 2}{6}$$

- $\sum_{k=1}^{n}$ represents the n time windows established by the *domain u*.
- $\varepsilon(k)$ depicts weighting factor to be considered by the *domain u* for each the time window k. $\varepsilon(k) \in [0,1]$ and $\varepsilon_1 + \cdots + \varepsilon_n = 1$.
- $AA(j)$ means the *available assets* of *provider j* when the *domain u* determined the reputation on *provider j*.
- $IA(j)$ depicts the *total assets* of the *provider j* when the *domain u* determined the reputation on *provider j*.
- $AAL(j)$ means the *available assets* of the *provider j,* at a particular *location,* when the *domain u* determined the reputation on *provider j*.
- $IAL(j)$ depicts the *total assets* of the *provider j,* at a particular *location,* when the *domain u* determined the reputation on *provider j*.
- $MV(j)$ represents the total number of predicted SLA *violations* that were finally *managed* successful.
- $PV(j)$ represents the total number of *predicted* SLA *violations*.
- $EV(j)$ represents the total number of predicted SLA *violations* that were finally managed unsuccessful (*executed*).

- $NPV(j)$ represents the number of SLA *violations* that were *not predicted* and turned out to be SLA violations.

The provider's satisfaction of the *domain u* on the *i-th* interaction will be computed about a particular offer of the *domain j* stakeholder.

$$OS(u, o_j) \; = \; Rep(u, o_j) \; * \; \bigoplus_{x=1}^{n} Rec(x, o_j) \; * \; T^{(t-1)}(u, x)$$

- $\oplus$ is a aggregation operation such as Minimum value, Maximum value, Arithmetic mean, or Harmonic mean. n denotes the rest of domains participating in the x-th interaction with the *domain j*.
- $Rec(x, o_j)$ is the recommendation of the x-th domain with which the *domain j* has a trust relationship about a specific kind of offer (RAN, Spectrum, Edge, Slice, or VNF/CNF). In other words, $T(x, o_j)$.
- $T^{(t-1)}(u, x)$ is the last trust score that the *domain u* has on a kind of specific offer of the *domain x*.

The $Rep(u, o_j)$ is the average reputation that the *domain u* has on a kind of specific offer of the *provider j*. This reputation contemplates features such as available offers, offer in a particular location, and multiple time windows to compute these features along the *provider j's* offer lifecycle.

$$Rep\big(u, o_j\big) = \sum_{k=1}^{n} \varepsilon(k) * \frac{\left(\frac{CO(j)}{IO(j)} + \frac{COL(j)}{IOL(j)} + 2 * \frac{MOV(j)}{POV(j)} - 2 * \frac{EOV(j)+NPOV(j)}{POV(j)}\right) + 2}{6}$$

- $\sum_{k=1}^{n}$ represents the n time windows established by the *domain u*.
- $\varepsilon(k)$ depicts weighting factor to be considered by the *domain u* for each the time window k. $\varepsilon(k) \in$ [0,1] and $\varepsilon_1 + \cdots + \varepsilon_n = 1$.
- $CO(j)$ means the number of *offers considered* by the Smart Resource and Service Discovery (SRSD), for a particular type of offer, from the *provider j* when the *domain u* determined the reputation on *provider j*.
- $IO(j)$ depicts the available number of a particular *offer* type from the *provider j* when the *domain u* determined the reputation on *provider j*.
- $COL(j)$ means the number of *offers considered* by the Smart Resource and Service Discovery (SRSD), for a particular type of offer from *provider j,* at a particular *location,* when the *domain u* determined the reputation on *provider j*.
- $IOL(j)$ depicts the *available number* of a particular *offer* type from the *provider j,* at a particular *location,* when the *domain u* determined the reputation on *provider j*.
- $MOV(j)$ represents the total *offer* number (for a particular kind of offer) of predicted SLA *violations* that were finally *managed* successful.
- $POV(j)$ represents the total offer number (for a particular kind of offer) of *predicted* SLA *violations*.
- $EOV(j)$ represents the total offer number (for a particular kind of offer) of predicted SLA *violations* that were finally managed unsuccessful (*executed*).
- $NPOV(j)$ represents the offer number of SLA *violations* that were *not predicted* and turned out to be SLA violations.

It is worth mentioning that $IA, IAL, PV, IO, IOL, and\ POV$ are parameters whose minimum value is 1. In other case, if these parameters were initialized to 0, such equation parts should be omitted due to the division of 0 by 0 not being allowed.

9.2.2. Feedback Credibility equation

In this first iteration, we have contemplated using a general credibility metric that can be applied to multiple contexts. Specifically, the personalized similarity metric (PSM) is the one selected. The objective of this formula is to determine how similar *v and w domains* are when evaluating the same *domain u*.

$$Cr\big(p(u,i)\big) = \frac{Sim(\,p(u,i),w)}{\sum_{j=1}^{I(u)} Sim(p(u,j),w)} \equiv \frac{Sim(v,w)}{\sum_{j=1}^{I(u)} Sim(v,w)}$$

$$Sim(v,w) \; = \; 1 \; - \sqrt{\frac{\sum_{x\,\in IJS(v,w)} \left(\frac{\sum_{i=1}^{I(x,v)} S(x,i)}{I(x,v)} - \frac{\sum_{i=1}^{I(x,w)} S(x,i)}{I(x,w)}\right)^2}{\mid IJS(v,w) \mid}}$$

- $I(x,v)$ depicts the total number of interactions that have been carried out by the *domain x* with the *domain v*.
- $I(x,w)$ depicts the total number of interactions that have been carried out by the *domain x* with the *domain w*.
- $\mid IJS(v,w) \mid$ is the set of domains that are interacted both with *domain v* and *domain w*.

9.2.3. Transaction Context Factor equation

The purpose of this equation is to calculate a final value associated with the current transaction type (product offer and provider) from the number of feedbacks provided in different time windows established. A higher number of feedbacks in the different time windows will indicate that both the type of offer and the provider are currently being used by other domains, and therefore, there will be a higher number of recommenders to be contemplated for finally determining a stable reputation.

$$TF(u,i) = \frac{\sum_{j=1}^{n} \varepsilon(j) * \left(\frac{\frac{FO(u,i)}{TOI(u,i)} + \frac{R(u,i)}{TI(u,i)}}{2}\right)}{n}$$

- $\sum_{j=1}^{n}$ represents the number of time windows established by the *domain u*.
- $\varepsilon(j)$ depicts weighting factor to be considered by the *domain u* for each the time window j. $\varepsilon(j) \in [0,1]$ and $\varepsilon_1 + \cdots + \varepsilon_n = 1$.
- $FO(u,i)$ is the total number of *feedbacks* of a particular type of *offer* that have been published about the *domain u* in the DLT.
- $R(u,i)$ means the total number of recommendations made by other domains about the *domain u* and published in the DLT.
- $TOI(u,i)$ is the total number of offer interactions recorded in the i-th interaction of *the domain u*.
- $TI(u,i)$ is the total number of provider interactions recorded in the i-th interaction of *the domain u*.

9.2.4. Community Context Factor equation

The purpose of the $CF(u)$ is to obtain the feedacks about a *domain u*. For this purpose, the interaction number that the *domain u* had in the community through the contribution of services or resources with other domains are evaluated. In addition, a dynamic list of trustworthy recommenders is contemplated to ask for *domain u*'s feedbacks. This new CF equation tries avoiding the bad-mouthing attack since each recommendation is thoroughly analysed. On the one hand, the *CF* considers the action trust that is the trust score that the *domain v* has on the *domain j.* On the other hand, the new *CF* contemplates the recommendation trust that is the trust on *domain j* making recommendations RT(v, j).

$$CF(u) = \frac{\frac{R(u)}{TI(u)} + \frac{\oplus_{j=1}^{n}\big(\alpha * T(v,j) + (1-\alpha) * (RT(v,j) * Rec(j,u))\big) * Inf(v,j)}{n}}{2}$$

- $Rec(j,u)$ the recommendation of the j-th domain with which the *domain u* has a trust relationship and it is in our list of trustworthy recommenders. In other words, $T(j,u)$.
- $RT(v,j)$ is the recommendation trust that the *domain v* has on the *recommender j* through previous recommendations. Being *RT(v,j)* ≥ 0.3
- $Inf(v,j)$is the recommender's influence over all recomenders contemplated. The higher *RT(v,j)*, the greater influence.

$$Inf(u) = \frac{RT(v,j)}{\overline{RT}}$$

- $RT(v,j)$ represents the arithmetic mean of all recommendation trust ≥ 0.2.

The RT parameter should be updated after each new interaction between the trustor with a given trustee. In this regard, the RT will leverage the following equation:

$$RT(v,j)^t = \begin{cases} (1 + (\overline{S(v,j)} - S(v,j)) -* \frac{\left(\overline{Rec(j,u)} - Rec(j,u)\right)}{10} + RT^{t-1}, & if\ S(v,j), Rec(v,j) < 0\ OR\ S(v,j), Rec(v,j) > 0 \\ RT^{t-1} - (1 - (\overline{S(v,j)} - S(v,j)) * \frac{\left(\overline{Rec(j,u)} - Rec(j,u)\right)}{10}, & if\ S(v,j) < 0\ AND\ Rec(v,j) > 0 \\ RT^{t-1} - (1 + (\overline{S(v,j)} - S(v,j)) * \frac{\left(\overline{Rec(j,u)} - Rec(j,u)\right)}{10}, & if\ S(v,j) > 0\ AND\ Rec(v,j) < 0 \\ RT^{t-1}, & if\ S(v,j), Rec(v,j) = 0 \end{cases}$$

- $S(v,j)$ is the current satisfaction and $\overline{S(v,j)}$ is the average satisfaction.

9.2.5. General reward and punishment equations

In this section, we are going to explain the general equation to determine a reward or punishment score from certain collected events. Such an equation allows us to employ multiple types of reward and security mechanisms, for instance, time-driven, event-driven, etc. Therefore, the type of reward and punishment mechanism will be defined through the parameter *RP(v,u)*.

9.2.5.1. Reward and punishment mechanism based on security events

In this regard, the general equation to update a previous trust score after computing a reward or punishment value is the following one:

$$N_{ts}(v,u) = \begin{cases} O_{ts}(v,u) + (RP(v,u) - 0.5) * (\frac{1 - O_{ts}(v,u)}{10}), & if\ RP(v,u) \geq 0.5 \\ O_{ts}(v,u) - \left(0.5 - RP(v,u)\right) * (\frac{1 - O_{ts}(v,u)}{10}), & if\ RP(v,u) < 0.5 \end{cases}$$

- $N_{ts}(v,u)$ is the new trust score after applying the reward and punishment of the *domain v* on the *domain u*.
- $O_{ts}(v,u)$ is the last trust score that the *domain v* has in the *domain u*.
- $RP(v,u)$ is the reward or punishment score to be applied over the last trust score.

In this case, we are going to contemplate a set of security events gathered from Zeek and Filebeat through the Security Analysis Service. Especially, this mechanism will be a time-driven type so after a pre-established time window (i.e., 30 minutes) the previous trust score will be updated using the network monitoring events. The reward and punishment equation is as follows:

$$RP_{total}(v,u) = \delta * RP_{total} + (1 - \delta) * RP_{current}$$

$$RP_{current}(v,u) = \alpha * Conn(v,u) + \beta * Notice(v,u) + \psi * Weird\ (v,u) + \phi * Stat(v,u)$$

- δ depicts a weighting factor with respect to the forgetting factor to be considered by *domain v*. $\delta \in$ [0,1].
- $\alpha, \beta, \psi, \phi$ depict weighting factors per dimension to be considered by *domain v*. $\alpha, \beta, \psi, \phi \in$ [0,1] and $\alpha + \beta + \psi + \phi$ = 1.

In the case of $Conn(v,u)$, it gathers the tracking/logging of general information regarding TCP, UDP, and ICMP traffic. Hence, we will formulate the following equation:

$$Conn(v,u) = \rho * \frac{TCP_{resp_{pkts}}}{TCP_{orig_{pkts}}} + \mu * \frac{UDP_{resp_{pkts}}}{UDP_{orig_{pkts}}} + \omega * \frac{ICMP_{resp_{pkts}}}{ICMP_{orig_{pkts}}}$$

- ρ, μ, ω depict weighting factors to be considered by *domain v*. $\rho, \mu, \omega \in$ [0,1] and $\rho + \mu + \omega$ = 1. It is advisable that μ has a higher value than ρ, ω.

When it comes to $Notice(v,u)$, it collects using Zeek likely monitoring events which are odd or potentially bad. Thus, we will formulate the following equation:

$$Notice(v,u) = 1 - \frac{\left(\frac{N_{actual_{events}}}{N_{actual_{events}} + N_{events_{n-1}}} + \frac{N_{actual_{events}}}{N_{actual_{events}} + N_{event\_last\_5\_window}}\right)}{2}$$

where monitoring events may be *CaptureLoss::Too_Much_Loss, Weird::Activity, PacketFilter::Dropped_Packets, Software::Vulnerable_Version, Scan::Port_Scan, and HTTP::SQL_Injection_Attacker*, to name but a few.

Regarding the $Weird(v,u)$, it provides a default set of actions to take as unusual or exceptional activity that can indicate malformed connections, malfunctioning or misconfigured hardware, or even an attacker attempting to avoid/confuse a sensor. In this case, we will formulate the following equation:

$$Weird(v,u) = 1 - \frac{\left(\frac{N_{actual_{weird\_events}}}{N_{actual_{weird\_events}} + N_{weird\_events_{n-1}}} + \frac{N_{actual_{weird\_events}}}{N_{actual_{events}} + N_{weird\_event\_last\_5\_window}}\right)}{2}$$

where weird events may be *DNS_UNMTATCHED_RELY or ACTIVE_CONNECTION_REUSE*, among others.

Finally, $Stats(v,u)$ gets log memory/packet/lag statistics from Zeek. In this case, we want to determine the percentage of packets analyzed by Zeek with respect to those issued. The equation is the following one:

$$Stats(v,u) = \rho * \frac{TCP_{pkts\_analysed_{Zeek}}}{TCP_{orig_{pkts}}} + \mu * \frac{UDP_{pkts\_analysed_{Zeek}}}{UDP_{orig_{pkts}}} + \omega * \frac{ICMP_{pkts\_analysed_{Zeek}}}{ICMP_{orig_{pkts}}}$$

- ρ, μ, ω depict weighting factors to be considered by *domain v*. $\rho, \mu, \omega \in$ [0,1] and $\rho + \mu + \omega$ = 1. It is advisable that μ has a higher value than ρ, ω.

9.2.5.2. Reward and punishment mechanism based on SLA Breach Predictions and violations

In this regard, the general equation to update a previous trust score after computing a reward or punishment value is the following one:

$$N_{ts}(v,u) = \begin{cases} O_{ts}(v,u) - Pu(u,v,m) * (\dfrac{1 - O_{ts}(v,u)}{5}), & \mathit{if\ not\ new\ violation} \\ O_{ts}(v,u) + Re(u,v,m) * (\dfrac{1 - O_{ts}(v,u)}{5}), & \mathit{otherwise} \end{cases}$$

- $N_{ts}(v,u)$ is the new trust score after applying the reward and punishment of the *domain v* on the *domain u*.
- $O_{ts}(v,u)$ is the last trust score that the *domain v* has in the *domain u*.
- $Re(v,u,m)$ is the reward score to be applied over the last trust score about the metric *m*.
- $Pu(v,u,m)$ is the punishment score to be applied over the last trust score about the metric *m*.

In the case of punishment, it considers three main dimensions:

$$Pu(v,u) = \sum_{m=1}^{n} \frac{BPRate(u,m) + ITrust(v,u) * SLVRate(u,m)}{2}$$

In the case of $BPRate(u,m)$, it computes the probability of suffering breach violation after predictions if the sakeholder *u* continues with the current behaviour. $BPRate(u,m)$ is subdivided into:

$$BPRate(u,m) = \frac{SLOBP(u,m)}{\sum_{k=1}^{n} SLOBP(u,k)} * CertaintyBP(u,m)$$

- $SLOBP(u,m)$ is the number of breach predictions in a given time window for the metric *m*.
- $CertaintyBP(u,m)$ is the accuracy of the algorithm used by the ISBP module to predict breach predictions.

With respect to the $ITrust(v,u)$, it measures how trust affects over the SLA events appearing in real time.

$$ITrust(v,u) = \left(1 - \frac{1 - T(v,u)}{1 + T(v,u)}\right) * \mu_{trus}(v,u)$$

- $T(v,u)$ is the current trust score between domain *v* and domain *u.*
- $\mu_{trus}(v,u)$ is a fuzzy set to determine the impact level of trust over the SLA events.

Finally, the punishment mechanism considers the SLA violation rate:

$$SLAVRate(u,m) = \alpha * \ SLAVRate^{t-1}(u,m) \ + \ (1 - \alpha) * \ Increment(u,m) * \ \mu_{vio}(u,m)$$

- $Increment(u,m)$ measures the growth of violations over the historical value.
- $\mu_{vio}(u,m)$ is another fuzzy set to find out the impact of new SLA violations over the new rate.

To reward stakeholders when they do not have SLA Violations, this mechanism also contemplates a reward function. This function is applied when the number of SLA Violations in the current time windows is 0.

$$Re(v,u) = \min((\sum_{m=1}^{n} SLAVRate^{t-1}(u,m) - SLAVRate^{t}(u,m)), 1)$$

<END OF DOCUMENT>