# D5.1 Service Platform Operational First Release

<table>
<thead>
<tr>
<th>Project Acronym</th>
<th>5GTANGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title</td>
<td>5G Development and Validation Platform for Global Industry-Specific Network Services and Apps</td>
</tr>
<tr>
<td>Project Number</td>
<td>761493 (co-funded by the European Commission through Horizon 2020)</td>
</tr>
<tr>
<td>Instrument</td>
<td>Collaborative Innovation Action</td>
</tr>
<tr>
<td>Start Date</td>
<td>01/06/2017</td>
</tr>
<tr>
<td>Duration</td>
<td>30 months</td>
</tr>
<tr>
<td>Deliverable</td>
<td>D5.1 Service Platform Operational First Release</td>
</tr>
<tr>
<td>Workpackage</td>
<td>WP5</td>
</tr>
<tr>
<td>Due Date</td>
<td>M10</td>
</tr>
<tr>
<td>Submission Date</td>
<td>11/4/2018</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Status</td>
<td>To be approved by EC</td>
</tr>
<tr>
<td>Editor</td>
<td>Carlos Parada (ALB)</td>
</tr>
<tr>
<td>Contributors</td>
<td>Carlos Parada, José Bonnet (ALB), Thomas Soenen (IMEC), Eleni Fotopoulos, Anastasios Zafeiropoulos (UBITECH), Stefan Schneider, Manuel Peuster (UPB), Evgenia Kapassa, Marios Touloupou, Panagiotis Stavrianos, Michael Filippakis (UPRC), Ana Pol, Anton Roman (QUO), Ricard Vilalta, Raul Muñoz, Ramon Casellas, Ricardo Martínez (CTTC), Felipe Vicens, Sonia Castro (ATOS), Peer Hasselmeyer (NEC), Theodore Zahariadis, Panos Trakadas, Panos Karkazis (SYN), Stavros Kolometos (NCSRD)</td>
</tr>
<tr>
<td>Reviewer(s)</td>
<td>José Bonnet (ALB), Dimosthenis Kyriazis (UPRC), Laurent Roulet (NBLF), Daniel Behnke, Patrick-Benjamin Bökö, Marcel Müller (WEID), Sonia Castro (ATOS)</td>
</tr>
</tbody>
</table>

**Keywords:**

NFV, MANO, Prototype, Service Platform, Slicing, Policy, SLA, Monitoring, APIs, Catalogue, Open Source
## Deliverable Type

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Document</td>
</tr>
<tr>
<td>DEM</td>
<td>Demonstrator, pilot, prototype</td>
</tr>
<tr>
<td>DEC</td>
<td>Websites, patent filings, videos, etc.</td>
</tr>
<tr>
<td>OTHER</td>
<td></td>
</tr>
</tbody>
</table>

## Dissemination Level

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
<td>Public, only for members of the consortium (including the Commission Services)</td>
</tr>
<tr>
<td>CO</td>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
</tr>
</tbody>
</table>

### Disclaimer:

This document has been produced in the context of the 5GTANGO Project. The research leading to these results has received funding from the European Community’s 5G-PPP under grant agreement n° 761493.

All information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors’ view.
Executive Summary:

This Deliverable D5.1 describes the initial design and development of the 5GTANGO Service Platform, for the Release v0 (M10, March 2018). After the delivery of D2.1 [19] (M3, August 2017), which outlines the prototype requirements, and D2.2 [20] (M6, December 2017), which presents the 5GTANGO high level architecture design, D5.1 digs deeper into the Service Platforms details, describing the internal components and interfaces, as well as the interactions with external systems and entities. In parallel, D3.2 [27] and D4.1 [28] describe similar aspects for the Verification and Validation (V&V) platform and the Service Development Kit (SDK), respectively.

The 5GTANGO project is not creating the Service Platform from scratch. Instead, it relies and extends the excellent work performed by the 5G-PPP phase I project SONATA [1]. Some of the Service Platform components started its development in the SONATA project, and will be extended in their features or refactored within the scope of 5GTANGO, in order to cope with the new requirements. Some other components will be developed by 5GTANGO from scratch, namely regarding new features such as the support of Service Level Agreements (SLAs), the policy-driven approach, or the network slicing, just to name the most relevant.

This document specifies the Service Platform by tackling the internal components and describing their design and implementation details. The developments inherited from SONATA project are briefly described, providing references to the SONATA’s documentation. But major spotlights are turned to the new 5GTANGO components, as well as to the 5GTANGO extensions to the existing SONATA components. The description of the interfaces among the internal components complement the understanding of the design and implementation details, capturing the full picture of the Service Platform. The interactions between the Service Platform and external systems or entities, provides a comprehensive view of how the Service Platform integrates in an external (and real) environment.

Security aspects are important topic addressed by the 5GTANGO Service Platform. In multi-tenant environments, it is fundamental to assure that only authenticated and authorized entities can access to the platform and use his resources. Security needs to be addressed in a holistic way, covering all interaction channels of the different players with the systems, namely securing portals, protecting APIs, and guaranteeing the integrity of the software packaging.

The specification of the Service Platform reflected in this Deliverable intends to document the initial design and implementation details. However, considering that the project is using an agile development process - Continuous Integration and Continuous Delivery (CI/CD) - the design here documented is suitable to be changed during the project lifetime, which is the normal way of moving forward using modern software development methodologies. In roughly one year (M21, February 2019), the project will release the Deliverable D5.2, which will document the final prototype design, updating the set of features and implementation details.

The 5GTANGO Service Platform will be delivered as open source under the SONATA trademark, as SONATA is today a reference on the ETSI NFV Management and Orchestration landscape. With 5GTANGO contributions, it is expected to improve substantially, increasing the SONATA’s recognition. The major 5GTANGO Service Platform objective is to make this software popular, lasting for many years, far beyond the project lifetime.
Contents

List of Figures vii

List of Tables ix

1 Introduction 1
  1.1 Document Scope ............................................. 1
  1.2 Overview ...................................................... 1
  1.3 Prototype Features ........................................... 2
  1.4 Document Structure .......................................... 3

2 Components 4
  2.1 Gatekeeper ..................................................... 4
    2.1.1 Overview ................................................ 4
    2.1.2 Descriptors .............................................. 5
    2.1.3 APIs ....................................................... 5
    2.1.4 Design and Implementation ................................. 6
    2.1.5 Flow Diagrams ............................................ 6
  2.2 Catalogues ...................................................... 9
    2.2.1 Overview ................................................ 9
    2.2.2 Architecture ............................................ 9
    2.2.3 APIs ....................................................... 10
    2.2.4 Search function of the Catalogues .......................... 10
    2.2.5 Flow Diagrams ............................................ 11
  2.3 Repositories .................................................... 12
    2.3.1 Overview ................................................ 12
    2.3.2 APIs ....................................................... 13
  2.4 MANO Framework ............................................... 15
    2.4.1 Overview ................................................ 15
    2.4.2 Design ..................................................... 15
    2.4.3 APIs ....................................................... 17
    2.4.4 Policy-based Placement ................................... 20
  2.5 Infrastructure Abstraction .................................... 21
    2.5.1 Overview ................................................ 21
    2.5.2 APIs ....................................................... 22
  2.6 VIMs/WIMs ...................................................... 22
    2.6.1 VIMs ....................................................... 22
    2.6.2 WIMs ....................................................... 22
  2.7 Monitoring Manager ............................................ 23
    2.7.1 Overview ................................................ 23
    2.7.2 Design and Implementation ................................ 25
    2.7.3 APIs ....................................................... 25
2.8 Slice Manager ................................. 26
 2.8.1 Overview .................................. 26
 2.8.2 Slice Manager Design ..................... 26
 2.8.3 Descriptors ............................... 27
 2.8.4 APIs ..................................... 27
 2.8.5 Design and Implementation .............. 28
 2.8.6 Flow Diagrams ............................. 28
2.9 Policy Manager ............................... 30
 2.9.1 Overview .................................. 30
 2.9.2 Runtime Policies Facets ................. 31
 2.9.3 Runtime Policies Descriptor .......... 33
 2.9.4 Sections of the Runtime Policy Descriptor . 33
 2.9.5 APIs ..................................... 35
 2.9.6 Implementation Status .................. 35
2.10 SLA Manager ................................. 36
 2.10.1 Overview .................................. 36
 2.10.2 5GTANGO SLA Manager Design .......... 36
 2.10.3 Descriptors ............................... 37
 2.10.4 APIs ..................................... 39
 2.10.5 Design and Implementation ............ 39
 2.10.6 Flow Diagrams ............................. 40
2.11 Portal ...................................... 42
3 Interfaces .................................. 44
 3.1 SP Internal Interfaces ...................... 44
    3.1.1 Gatekeeper / Catalogues ................. 44
    3.1.2 Gatekeeper / MANO Framework .......... 45
    3.1.3 Gatekeeper / Slice Manager ............ 45
    3.1.4 Gatekeeper / Policy Manager ............ 45
    3.1.5 Gatekeeper / SLA Manager ............... 46
    3.1.6 Slice Manager / MANO Framework ....... 46
    3.1.7 Slice Manager / Repositories .......... 47
    3.1.8 Slice Manager / Catalogues ............. 47
    3.1.9 MANO Framework / Infrastructure Abstraction . 47
    3.1.10 MANO Framework / Monitoring Manager .. 48
    3.1.11 MANO Framework / Repositories ........ 49
    3.1.12 Policy Manager / Monitoring Manager .. 49
    3.1.13 SLA Manager / Catalogues .............. 49
 3.2 SP External Interfaces ...................... 51
    3.2.1 SP / SDK .................................. 51
    3.2.2 SP / V&V .................................. 51
    3.2.3 SP / VIMs/WIMs .......................... 51
    3.2.4 SP / Portal .................................. 52
    3.2.5 SP / Service Provider .................. 53
    3.2.6 SP / Developer ............................ 54
    3.2.7 SP / Customer ............................. 55
4 Source Code ................................. 56
5 Security Aspects

5.1 Infrastructure security .................................................. 58
5.2 Portal security .............................................................. 58
  5.2.1 Secure access from the user’s browsers ........................... 58
  5.2.2 User authentication ................................................... 58
5.3 APIs Security ............................................................... 58
5.4 Packages Security ......................................................... 59

6 Prototype Roadmap .......................................................... 60

6.1 ETSI-Compliant APIs ...................................................... 60
  6.1.1 Os-Ma-nfvo Interfaces (OSS-NFVO) ............................... 60
  6.1.2 Or-Vnfm Interfaces (NFVO-VNFM) ................................ 61
  6.1.3 Ve-Vnfm Interfaces (VNFM-VNF/EM) ............................. 61
6.2 Slicing .......................................................... 61
6.3 Runtime Policy ......................................................... 62
6.4 SLA .......................................................... 62
6.5 MANO Framework ...................................................... 63
6.6 Monitoring ............................................................ 63
6.7 Catalogues .......................................................... 64
6.8 Repositories .......................................................... 64
6.9 Portal .......................................................... 65

7 Conclusions ................................................................. 66

A Appendices ................................................................. 67

A.1 Policy Descriptor Example ............................................ 67
A.2 Drools Rules Example .................................................. 68
A.3 NST Example ............................................................ 69
A.4 NSI Record Example ..................................................... 70
A.5 SLA Template Descriptor Example .................................. 71

B Bibliography .............................................................. 73
List of Figures

1.1 Overview of main components of the 5GTANGO v0 release. 3
2.1 SONATA API Gateway. 4
2.2 Gatekeeper Positioning in 5GTANGO. 5
2.3 5GTANGO Gatekeeper Architecture. 7
2.4 5GTANGO Package Uploading (end-to-end). 8
2.5 5GTANGO Catalogues Architecture. 9
2.6 5GTANGO Catalogues Flow Diagram. 12
2.7 5GTANGO Repositories Design. 13
2.8 ETSI-compliant MANO Architecture. 16
2.9 5GTANGO Monitoring Manager Diagram. 23
2.10 Slice Manager’s internal architecture. 27
2.11 Network Slice Template Creation. 29
2.12 Network Slice instantiation. 30
2.13 Policies Conditions. 32
2.14 Policies Actions. 32
2.15 SLA Manager Components. 37
2.16 SLA Template Generation Flow. 41
2.17 SLA Template Modification Flow. 41
2.18 SLA Template State Status Flow. 42
2.19 Portal Dashboard shapshot. 43
2.20 Portal Requests shapshot. 43
3.1 Policy enforcement for a network service. 49
3.2 Retrieval of NSD and PD and SLA Tempalte storage on the Catalogue. 50
3.3 SP and V&V Relashionship. 52
3.4 Retrieve the list of NSs. 53
List of Tables

2.1 APIs exposed by the Gatekeeper for the Release v0. ......................... 5
2.2 APIs exposed by Catalogues for the Release v0. ............................ 10
2.3 Syntax of the URI string for searching the Catalogues. .................. 11
2.4 Available operators for searching the Catalogues. ........................ 11
2.5 API for handling NS Records. ................................................. 13
2.6 API for handling VNF records. ............................................. 14
2.7 API for handling Slice Instance records. ................................ 14
2.8 API for handling SLA information. ....................................... 14
2.9 API for handling Policy information. .................................. 14
2.10 MANO Framework communication topics. .................................. 17
2.11 IA communication topics. .................................................... 22
2.12 Monitoring extensions to VNF-level monitoring. ......................... 26
2.13 Monitoring extensions to MoonGen. ....................................... 26
2.14 Network Slice Template APIs. ............................................. 27
2.15 Network Slice Instance APIs. ............................................ 28
2.16 APIs for Policy Management. .............................................. 35
2.17 APIs for SLA Template Management. .................................... 39
2.18 APIs for SLA Management. ................................................. 39

3.1 Monitoring Manager API. ...................................................... 48
3.2 REST endpoint overview for SP and SDK interaction. .................... 51
1 Introduction

This section introduces the reader to the basics of the 5GTANGO Service Platform, to their motivations and provides some introductory information.

1.1 Document Scope

The objective of this deliverable is to describe the design and implementation details of the first release of the 5GTANGO prototype due in project month 10 (March 2018), focusing on the major components of the Service Platform. The deliverable provides a broad overview of the internal components as well as the interfaces among them. It also defines the interactions of the Service Platform with external entities and players, contributing to a clear understanding of the overall picture.

The description contained in this document will be updated in project month 21 (February 2019), when the second and final version of the 5GTANGO Service Platform is released. Considering the agile methodologies extensively used in the project, it is expected that some design and implementation decisions will be updated and extended over the course of the project. Following these updates as well as the planned enriched functionality in the service platform, Deliverable D5.2 will describe the final 5GTANGO Service Platform design and implementation.

1.2 Overview

This 5GTANGO deliverable D5.1, describes the components and features of the first release of 5GTANGO’s Service Platform (SP) for Network Function Virtualization (NFV). The development of the SP has started in the SONATA project [1]. Now, 5GTANGO will continue the work of SONATA, by improving some features and adding others, some of them even beyond the ETSI NFV scope.

Some SONATA features improved in 5GTANGO include the monitoring extensions to support a Virtual Network Functions (VNFs), Network Services (NSs) and even Slice-level views, providing a holistic perspective of the aggregated monitoring at different abstraction layers. Also the support of additional datacenter infrastructures, others than Openstack, is a target, especially considering the increasing trend on Container technologies. The alignment with ETSI RESTful APIs is also a target, since ETSI NFV compliance is of paramount importance. Catalogues and Repositories need also to be improved to support new features.

Regarding new features, in particular, 5GTANGO focuses on creating a policy-driven environment, where the new Policy Manager component is responsible to manage, store and enforce all the policies associated to the platform. That includes policies related to service placement, scaling, updating or healing, among others. This is inline with similar NFV solutions and is also aligned with the emerging trends regarding to zero-touch management and network automation.

5GTANGO pays particular attention to QoS (Quality of Service) and SLAs (Service Level Agreements), a fundamental tool to ensure that the level of QoS promised to the customer can be properly contracted and committed. The new SLA Manager component is responsible to create dynamically
SLA templates and agreements with the customer, taking into consideration the available resources and the capability to ensure certain service quality.

5GTANGO addresses another hot topic on 5G networks, which is network slicing. The ability to create multiple virtual networks on top of a common infrastructure is an outstanding asset when it comes to cope with different types of terminals and requirements, from IoT (Internet of Things) sensors, to video consumers, or vehicles. In such a heterogeneous environment, it is fundamental to be able to create different virtual networks (slices) for different use case requirements, providing the best-of-breed network for each of them. The new Slice Manager component is responsible to manage the entire lifecycle of the slices, from the templates on-boarding to the slice instantiation, including scaling and termination operations.

1.3 Prototype Features

This section lists the features that the first version of the 5GTANGO Service Platform (including the service development kit, the service platform, the V&V component) will support.

These features are:

1. Developers may use the SDK tools to build a package containing a NS, the VNFs that comprise the NS and Tests to validate the NS or VNFs;
2. Developers may upload VNF/NS packages;
3. Developers may query the status of the package un-packaging and validation process;
4. Packages are un-packaged and their content (including the file containing the package) is stored in the Catalogue;
5. The package’s developer is notified of the successful package un-packaging and validation process;
6. The V&V is notified about the existence of a new package;
7. Tests and NSs are automatically associated through a tagging mechanism built into their descriptors;
8. The V&V automatically selects NSs to be tested:
   1. NS instantiation is requested to the Service Platform;
   2. The NS instance data is read from the Repositories;
   3. The NS instance is used in a pre-defined way;
   4. The results of the execution of the NS instance are collected;
   5. The NS instance is terminated;
9. The following features are ready and available via APIs, although not yet via the Portal (see Fig. 1.1):
   1. Network Slices comprising a number of NS can be defined;
   2. Policies can be defined;
   3. SLAs can be defined;

Details on all these features can be seen in this document, as well as deliverables D3.2 [27] and D4.1 [28]. The high level architecture supporting these features is shown in Fig. 1.1.
1.4 Document Structure

The remainder of this deliverable is structured as follows. Sec. 2 describes the internal components that comprise the SP, detailing their design and implementation. Sec. 3 describes the interfaces among the internal components of the SP, as well as the external interaction with other entities and stakeholders, such as developers, operators and customers. Sec. 4 describes the open source software modules that comprise the SP. Sec. 5 approaches the most relevant security aspects of the Service Platform, while Sec. 6 highlights the most relevant features to be developed until the end of the project. Finally, Sec. 7 presents the conclusions of the work done so far.
2 Components

This section describes the internal components of the 5GTANGO Service Platform prototype.

2.1 Gatekeeper

This section describes the features that the Gatekeeper [20] has for this first version of the Service Platform. For more details, please refer to the related GitHub repository [15].

2.1.1 Overview

The Gatekeeper in the 5GTANGO Service Platform (SP) continues to play the protector role it has been given in the SONATA SP [41]. It is the component responsible to act as API gateway, exposing the APIs to the outside. The Gatekeeper takes care of API complexities, like authentication and authorization mechanisms or compliance with Standards Developing Organizations (SDOs), e.g. ETSI, translating these calls into the more simplified version on the internal components. This contributes for an efficient and centralized control of the exposed APIs, detaching the APIs exposed to the outside from the ones exposed by the internal components, making the APIs evolution more flexible.

The Gatekeeper’s microservices that result from SONATA’s v3.1 [45] is shown in Fig. 2.1. As it can be seen, the whole external access goes through a security gateway (son-sec-gtw), which handles the requests to the gatekeeper API (son-gtkapi). This component plays the role of an API Gateway, validating the requests and users, and collecting KPIs. Valid requests are then forwarded to the adequate component for processing.

The complexity of the component grew with the implemented features, which led to a bloated component, hard to maintain an evolve, with each microservice in one container. Now considering also the V&V, which shares some of the features of the SP, we have reorganized the Gatekeeper into the following containers:

Other containers coming from the SONATA v3.1 version, like User Manager, the Rate limiter, the KPIs Manager and the Licence Manager will be reused as they are in this version.

![Figure 2.1: SONATA API Gateway.](image-url)
In 5GTANGO, the Gatekeeper have been extended in order to incorporate the APIs and cope
with the new components, namely, the Slice Manager, the Policy Manager and the SLA Manager.
The new Portal uses the APIs exposed by the Gatekeeper to build the graphical user interface. The
Fig. 2.2 depicts the positioning of the Gatekeeper in 5GTANGO (Connections between the other
components within the SP are not represented).

As represented above, every external access to the SP goes through the Gatekeeper. This single
point of access allows the fine control for a number of features, like user authentication, rate limiting,
KPIs collection, etc.

2.1.2 Descriptors

The Gatekeeper components has no particular descriptor associated to them; however, they can
provide any type of requested descriptors, as a result of API invocations, namely: Packages, NSs,
VNFs, Tests, Slices, Policies and SLAs.

2.1.3 APIs

The Gatekeeper APIs are described in the Gatekeeper’s API Gateway [15]. They are mostly in
REST, with the exception of the one with the MANO (MANagement and Orchestration), which is
message-based (RabbitMQ). Some of these APIs are shared between the SP and the V&V [28].

Tbl. 2.1 shows the set of APIs exposed by the Gatekeeper for the Release v0. This includes
APIs coming from the SONATA project, along with others created by 5GTANGO from scratch.
This explains some misalignment in the APIs format. As 5GTANGO is deeply committed to
fully align with ETSI, new APIs follow the ETSI model even for APIs not specified by ETSI
(like for the Slicing features), while the others follow the SONATA approach. For example,
the /api/nst/v1/descriptors follows the $PATH/<interface>/<version>/<resource>/. . .
ETSI format, while the /api/v3/services follows the SONATA format, which is
$PATH/<version>/<interface>/<resource>/. . .
For the Release v1, the APIs format will be fully aligned with the ETSI model.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload a package</td>
<td>POST</td>
<td>/api/v3/packages</td>
</tr>
</tbody>
</table>
**2.1.4 Design and Implementation**

The re-organization, mentioned above, of putting more than one microservice in one container made the whole process of generating and uploading a container a little slower, but with big gains in simplification of the Gatekeeper and reduction in the overall component size.

Components **Security Gateway, Router, User Manager, Licence Manager, KPIs Manager** and **Gatekeeper Common**, which are common to the V&V Platform, are described in [27].

The **Gatekeeper SP Specific** component concentrates all the microservices that are SP-specific features, i.e., Services’ Lifecycle Management (and eventually others in future versions).

The API listed in Tbl. 2.1 is specified with a simple configuration file such as the one shown below.

```yaml
base_path: /api/v3
paths:
  /packages(/?|/\*):
    site: http://tng-gtk-common:5200/packages
    verbs: [ get, post ]
  /slices(/?|/\*):
    site: http://tng-slices
    verbs: [ get, post, delete ]
  /policies(/?|/\*):
    site: http://tng-policies
  /slas(/?|/\*):
    site: http://sla-generator:8080
```

Please note that the /api/v3 is the evolution of the SONATA external API (/api/v2). This new version includes the new APIs specifically for 5GTANGO, like Policies Manager or SLAs Manager. These components may have a different API (e.g., /api/v1), which nevertheless must be mapped in this file. For further details, please look into the components repository [15].

**2.1.5 Flow Diagrams**

This section illustrates the most significant flow diagrams of the Gatekeeper.
The Package uploading flow, shown in Fig. 2.4, is a significant evolution from the one implemented for SONATA: the unpackaging and validation of the package is now made in a single component, which enhances evolution and reuse.

Details of this workflow, also mentioned in [27], are the following:

1. **ENV lcm callback**: the common Packages module retrieves the V&V LCM (LifeCycle Management) callback URL from the ENV;

2. **on-board package(user callback)**: the user (through the SDK) uploads a new package, including his/her own callback;

3. **Security Gateway (SGTW) to Router**: the on-boarding request goes through the Security Gateway and then...

4. **Router to Packages**: ...through the Router, hitting the Packages module;

5. **stores user callback**: the Packages module stores the user callback (for later use, see step 11) and generates an internal callback;

6. **on-board package (internal callback)**: the new package is uploaded to the Unpackager;

7. **package being validated**: the user is notified about the success of the package reception;

8. **processes/validates**: the Unpackager module validates and...

9. **stores**: ...stores the new package in the Catalogues and...

10. **notifies(internal callback)**: ...notifies the Packages module about the success of the operation, using the internal callback;
Figure 2.4: 5GTANGO Package Uploading (end-to-end).

11. **fetches user callback**: ... fetches the previously stored user callback (in step 5) and...

12. **package stored (user callback)**: ...notifies the user of the successfully on-boarded package;

13. **package stored (lcm callback)**: the Packages module calls the V&V LCM callback (fetched in step 1), notifying it of a new package successfully on-boarded and...

14. **fetches package**: the V&V LCM can fetch all needed assets from the Catalogues and proceed work.

We need these three callbacks because:

1. the user might not send a callback, opting for searching his/her package later (a callback implies a ‘server’ available to be called, which might not be practical) – so, an internal callback must be generated, to cover the cases in which the user does not send a callback;

2. only the V&V LCM callback is not enough, since we want to share the feature between the V&V and the SP.

Other apparently shorter/faster/better solutions, such as directly uploading the package from the nginx to the tng-package wouldn’t allow the swap of the callbacks, and therefore would not allow the implementation of the features described above.

Please note as well that the SDK provided callback allows the integration of other tools to automate the process of uploading a package.
2.2 Catalogues

This section briefly describes the architecture of the first version of the 5GTANGO Catalogues [20], referenced as tng-cat, as well as its APIs. Further details and related work are attached in the Appendix and in the corresponding GitHub repository [21].

2.2.1 Overview

The 5GTANGO Catalogues persist in constituting an instrumental component in the 5GTANGO environment, being present at various parts of it (such as the V&V and Service Platforms main building blocks). The extension of the SONATA Catalogues [40] has the aim of hosting the new information and type of metadata regarding the V&V framework and the new components presented in 5GTANGO’s Service Platform, namely the SLA Manager, the Slice Manager and the Policy Manager. The corresponding descriptors can be imported and fetched both from 5GTANGO components and from the developer. Also, the 5GTANGO Catalogues will reuse the implemented file storage and retrieve for packages. In the first 5GTANGO software release, the main objective of the Catalogues’ API is to support the storage, the update, the retrieval and the deletion of the new information (i.e. stored datasets) and the metadata produced by the various components of the 5GTANGO overall environment or/and by the developers.

2.2.2 Architecture

Along with the already-available storage functionality in the SONATA Catalogues’ components [42] for storing Package Descriptors (PDs), NS Descriptors (NSDs), VNF Descriptors (VNFDs) etc., each new component (i.e. VNFs and NSs) introduced in the 5GTANGO ecosystem will be characterized by the corresponding associated information. The descriptors provided from each framework will be included in different groupings (collections) inside the database, as it is depicted in Fig. 2.5:

- SLA Descriptors (SLADs): Collection including the information of low-level requirements in the respective policies and QoS definitions.
- Test Descriptors (TESTDs): Collection including the information of test qualifications and execution.
- Network Slice Templates (NSTs): Collection including the relevant information for the Network Slice Instantiation.
- Policy Descriptors (PLDs): Collection including the information for the policies formulation process.
In the 5GTANGO infrastructure, the main difference between the Catalogues and the Repositories is that the former includes the descriptions of the services while the latter includes the information about the instantiated services and functions. The interaction with the Repositories allows the Catalogue to check if a service has instances running. The Catalogues’ API enables access for the upload, retrieval, update and deletion of the stored descriptors through the RESTful API.

Given that the aforementioned new mechanisms (e.g. slice management, policy management, SLA management) in the 5GTANGO architecture have a great impact on the NSs/VNFs functionality and the respective quality provisioning, the Catalogues will enable storage of the corresponding descriptors (e.g. SLAD, TESTD, NST and PLD) in order to contain information about the respective stored VNFs/NSs. In this perspective, we designed and implemented a searching mechanism of descriptors of the Catalogues. In this way, not only the information perused is provided as mentioned above, but also the stored descriptors are scanned. Via the searching interface, Catalogues deliver the descriptors matching the query criteria given while the matching of NSs/VNFs with the corresponding SLADs, TESTDs, NST and PLDs is available.

### 2.2.3 APIs

This section describes the REST API endpoints that are introduced in the catalogues. The 5GTANGO Catalogues’ API is described in the 5GTANGO SP Catalogues API repository [21]. A description of the Catalogues’ API is given in Tbl. 2.2:

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>List all the available descriptors</td>
<td>GET</td>
<td>/api/catalogues/v2/{collection}</td>
</tr>
<tr>
<td>List all descriptors matching a specific filter(s)</td>
<td>GET</td>
<td>/api/catalogues/v2/{collection}?{attributeName}={value}</td>
</tr>
<tr>
<td>List only the last version for all descriptors</td>
<td>GET</td>
<td>/api/catalogues/v2/{collection}?version=last</td>
</tr>
<tr>
<td>List a descriptor using the UUID</td>
<td>GET</td>
<td>/api/catalogues/v2/{collection}/{id}</td>
</tr>
<tr>
<td>Store a descriptor in the Catalogue</td>
<td>POST</td>
<td>/api/catalogues/v2/{collection}</td>
</tr>
<tr>
<td>Update a descriptor using the naming triplet, i.e., name, vendor &amp; version</td>
<td>PUT</td>
<td>/api/catalogues/v2/{collection}</td>
</tr>
<tr>
<td>Update a descriptor using the UUID</td>
<td>PUT</td>
<td>/api/catalogues/v2/{collection}/{id}</td>
</tr>
<tr>
<td>Set status of a descriptor using the UUID</td>
<td>PUT</td>
<td>/api/catalogues/v2/{collection}/{id}</td>
</tr>
<tr>
<td>Delete a descriptor using the naming triplet, i.e., name, vendor &amp; version</td>
<td>DELETE</td>
<td>/api/catalogues/v2/{collection}</td>
</tr>
<tr>
<td>Delete a descriptor using the UUID</td>
<td>DELETE</td>
<td>/api/catalogues/v2/{collection}/{id}</td>
</tr>
</tbody>
</table>

Where \{collection\} is one of network-services, vnfs, packages, slas/template-descriptors, tests, nsts, policies, denoting the MongoDB collection of each grouping.

### 2.2.4 Search function of the Catalogues

The ability to search the catalogue is a vital prerequisite for several services on the grounds that it reduces the number of objects returned by a query operation by filtering. The matching criteria could be from plain queries, i.e fetching a descriptor with the unique naming triplet or its current status, to more complex queries, i.e applying comparison query operators. A key-based filtering is applied to a GET method with the aim of returning objects that match the particular filter. The key-based filtering will execute test on a scalar attribute against a constant value in terms of greater or smaller than, inequality etc. Key-based filtering will be available by setting appropriately a set
of URI query parameters.

2.2.4.1 Mapping content of the Catalogue

The access of any desirable attribute/value included in deep hierarchically-structured data is necessary. In order to optimize the process of searching the Catalogues, an index data structure is storing mappings of content to the corresponding document, called inverted file, which is stored in the database. More specifically, the inverted file hosts the desirable pairs of attribute/value (defined by the developer) along with a list of unique IDs of descriptors for the corresponding pair. The inverted file is updated for the PUT, POST and DELETE HTTP methods, where a document is added to (or deleted from) the database, and correspondingly a specific value and/or a list of IDs is updated.

2.2.4.2 Syntax

The paramount issue of the search function is the ability to explore attributes in the hierarchical structure of the data and applying filters, even to lower levels in the hierarchy. The set of filters will be a part of the URI query string implying the method of operation. According to ETSI template, the filter expression will be described in Tbl. 2.3:

<table>
<thead>
<tr>
<th>Name</th>
<th>URI string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter_Expression</td>
<td>attributeName[&quot;.&quot;attributeName].op=value[^&quot;value]</td>
</tr>
<tr>
<td>Combination of expressions</td>
<td>Filter_Expression[^&amp;Filter_Expression]</td>
</tr>
</tbody>
</table>

where [.] denotes optional parameters given in the URI string, value a scalar value and op the comparison operator representing one of the \{neq, eq, gt, gte, lt, lte, in, nin\}. The role of the dot ("." ) operator serves as the key component in concatenating the attributeName entries in the URI string, where the filter of deeper attributes is necessary, in a structured document. Thus, the comparison operator is applied to the last attribute mentioned. Below in Tbl. 2.4, the available comparison operators are listed:

<table>
<thead>
<tr>
<th>Comparison Operator (op)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeName.neq=value</td>
<td>Rightmost attribute is not equal to the value</td>
</tr>
<tr>
<td>attributeName=value</td>
<td>Rightmost attribute is equal to the value</td>
</tr>
<tr>
<td>attributeName.eq=value</td>
<td>Alternative form of equality</td>
</tr>
<tr>
<td>attributeName.gt=value</td>
<td>Rightmost attribute is greater than the value</td>
</tr>
<tr>
<td>attributeName.lt=value</td>
<td>Rightmost attribute is less than the value</td>
</tr>
<tr>
<td>attributeName.gte=value</td>
<td>Rightmost attribute is greater than or equal to the value</td>
</tr>
<tr>
<td>attributeName.lte=value</td>
<td>Rightmost attribute is less than or equal to the value</td>
</tr>
<tr>
<td>attributeName.in=value,[value]</td>
<td>Rightmost attribute contains at least one value of the list</td>
</tr>
<tr>
<td>attributeName.nin=value,[value]</td>
<td>Rightmost attribute doesn’t contain any value of the list</td>
</tr>
</tbody>
</table>

2.2.5 Flow Diagrams

In this section, a general flow diagram of the Catalogues’ interaction with an HTTP method through the RESTful API is presented in Fig. 2.6.

Workflow details:

1. Request HTTP method: A HTTP request arrives at the Catalogues’ API.
2. **Before deleting a service/function descriptor, check for instances:** In case of DELETE HTTP request, the Catalogues’ API fetches instantiated services from the Repositories in order not to delete instantiated services or relative correlated information.

3. **Instances check returned:** The information about instantiated services or functions is returned to the Catalogues’ API.

4. **Serve corresponding HTTP method:** The Catalogues’ API serves the HTTP method requested by uploading/retrieving/deleting the corresponding descriptors.

5. **Confirmation:** The Database confirms the normal operation of the interaction.

6. **Return Code/ID/Descriptors:** Finally, the Catalogues’ API returns to the user/framework the response status code of the HTTP message. Also, if a descriptor is uploaded or retrieved, the ID and the entire descriptor is returned.

### 2.3 Repositories

This section covers the architecture of the repositories, as well as the overview of the APIs and the provided features.

#### 2.3.1 Overview

5GTANGO repositories are composed by an API that handles the fetch, store and delete operations of the records in the database. From SONATA Service platform [40], 5GTANGO Service Platform will reuse the NS Repositories and VNF Repositories, extracting them from son-catalogue-repos [51] components and creating a specific component for repositories. 5GTANGO repositories will have in addition the Slice Records Repository, the SLA Records Repository and Policy Records Repository.

It is important to note that the design incorporates a place for the V&V Test Results Repositories that is described in-depth in 5GTANGO deliverable D3.2 [27].
The design of the 5GTANGO Repositories represented in Fig. 2.7 is based on microservice architecture, a docker image is built with the APIs developed in Ruby. The container is deployed by Ansible scripts and it is listening on port 4012 ready to serve the Repositories. As a dependency, the 5GTANGO Repositories require a MongoDB. A collection is created in the Mongo database per API.

2.3.2 APIs

2.3.2.1 NS Repositories

Tbl. 2.5 shows the API endpoints responsible for handling the runtime information of NSs that 5GTANGO MANO deploys. The schema designed for the NS record is available on GitHub in the sonata-nfv/tng-schema repository [18]. NS records are used by 5GTANGO MANO to track the Service Information and also to configure the service via Function-Specific Managers (FSMs) and (Service-Specific Managers) SSMs.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create NS Instance Record</td>
<td>POST</td>
<td>/records/nsr/ns-instances</td>
</tr>
<tr>
<td>List NS Instances Record</td>
<td>GET</td>
<td>/records/nsr/ns-instances</td>
</tr>
<tr>
<td>Fetch NS Instance Record</td>
<td>GET</td>
<td>/records/nsr/ns-instances/&lt;service_instance_uuid&gt;</td>
</tr>
<tr>
<td>Update NS Instance Record</td>
<td>PUT</td>
<td>/records/nsr/ns-instances/&lt;service_instance_uuid&gt;</td>
</tr>
<tr>
<td>Delete NS Instance Record</td>
<td>DELETE</td>
<td>/records/nsr/ns-instances/&lt;service_instance_uuid&gt;</td>
</tr>
</tbody>
</table>

2.3.2.2 VNF Repositories

Tbl. 2.6 shows the API endpoints responsible for handling the runtime information of VNFs that 5GTANGO MANO deploys. The schema designed for the VNF records is available on GitHub in the sonata-nfv/tng-schema repository [18]. VNFs records are used by MANO to track the VNF information and also to configure the service via FSMs and SSMs.
Table 2.6: API for handling VNF records.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create VNF Instance Record</td>
<td>POST</td>
<td>/records/vnfr/vnf-instances</td>
</tr>
<tr>
<td>List VNF Instances Record</td>
<td>GET</td>
<td>/records/vnfr/vnf-instances</td>
</tr>
<tr>
<td>Fetch VNF Instance Record</td>
<td>GET</td>
<td>/records/vnfr/vnf-instances/&lt;vnf_instance_uid&gt;</td>
</tr>
<tr>
<td>Update VNF Instance Record</td>
<td>PUT</td>
<td>/records/vnfr/vnf-instances/&lt;vnf_instance_uid&gt;</td>
</tr>
<tr>
<td>Delete VNF Instance Record</td>
<td>DELETE</td>
<td>/records/vnfr/vnf-instances/&lt;vnf_instance_uid&gt;</td>
</tr>
</tbody>
</table>

2.3.2.3 Slice Records Repositories

Tbl. 2.7 shows the API endpoints responsible for handling the runtime information of Slices that 5GTANGO Slicing Manager deploys. The schema designed for the NS record is available on GitHub in the sonata-nfv/tng-schema repository [18]. Slice Records are used by 5GTANGO Slice Manager to track the information of slices deployed.

Table 2.7: API for handling Slice Instance records.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Network Slice Instance Record</td>
<td>POST</td>
<td>/records/nsir/ns-instances</td>
</tr>
<tr>
<td>List Network Slice Instances Record</td>
<td>GET</td>
<td>/records/nsir/ns-instances</td>
</tr>
<tr>
<td>Fetch Network Slice Instance Record</td>
<td>GET</td>
<td>/records/nsir/ns-instances/&lt;slice_instance_uid&gt;</td>
</tr>
<tr>
<td>Update Network Slice Instance Record</td>
<td>PUT</td>
<td>/records/nsir/ns-instances/&lt;slice_instance_uid&gt;</td>
</tr>
<tr>
<td>Delete Network Slice Instance Record</td>
<td>DELETE</td>
<td>/records/nsir/ns-instances/&lt;slice_instance_uid&gt;</td>
</tr>
</tbody>
</table>

2.3.2.4 SLA Mapping Repositories

Tbl. 2.8 contains the API endpoint information related to the operation of the SLA runtime information. This repository contains the correlations between the high-level and the low-level requirements.

Table 2.8: API for handling SLA information.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create SLA Instance Record</td>
<td>POST</td>
<td>/records/sla/sla-instances</td>
</tr>
<tr>
<td>List SLAs Instances Record</td>
<td>GET</td>
<td>/records/sla/sla-instances</td>
</tr>
<tr>
<td>Fetch SLA Instance Record</td>
<td>GET</td>
<td>/records/sla/sla-instances/&lt;sla_instance_uid&gt;</td>
</tr>
<tr>
<td>Update SLAs Instance Record</td>
<td>PUT</td>
<td>/records/sla/sla-instances/&lt;sla_instance_uid&gt;</td>
</tr>
<tr>
<td>Delete SLA Instance Record</td>
<td>DELETE</td>
<td>/records/sla/sla-instances/&lt;sla_instance_uid&gt;</td>
</tr>
</tbody>
</table>

2.3.2.5 Policies Repositories

Tbl. 2.9 represents the endpoints required for the policies operations. In this repository, policies runtime information will be stored to be enforced by 5GTANGO MANO.

Table 2.9: API for handling Policy information.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Policy Instance Record</td>
<td>POST</td>
<td>/records/policy/policy-instances</td>
</tr>
<tr>
<td>List Policy Instances Record</td>
<td>GET</td>
<td>/records/policy/policy-instances</td>
</tr>
<tr>
<td>Fetch Policy Instance Record</td>
<td>GET</td>
<td>/records/policy/policy-instances/&lt;policy_instance_uid&gt;</td>
</tr>
<tr>
<td>Update Policy Instance Record</td>
<td>PUT</td>
<td>/records/policy/policy-instances/&lt;policy_instance_uid&gt;</td>
</tr>
<tr>
<td>Delete Policy Instance Record</td>
<td>DELETE</td>
<td>/records/policy/policy-instances/&lt;policy_instance_uid&gt;</td>
</tr>
</tbody>
</table>

2.4 MANO Framework

The MANO Framework is one of the core components of the 5GTANGO Service Platform. It manages the life cycle of all NS requests and instances by orchestrating the available infrastructure. This section sets forth the MANO Framework prototype, which is an evolution of the one developed in SONATA.

2.4.1 Overview

We start by giving a short elaboration on the SONATA MANO Framework design. Next, we describe the key feature that was added for this prototype release: policy-based placement. Multiple actors desire to control how the VNFs of a network service are placed on the infrastructure. The NS developer wants to control the placement to optimise the QoS for the end-user, as only he or she knows which location (e.g. close to the end-user or close to the core) results in an optimal performance. The operator wants to control the placement to make optimal use of its infrastructure (e.g. minimise energy consumption) and the customer – the one that requests the instantiation of the NS – might want to prevent the use of infrastructure in certain regions due to privacy concerns. With our policy-based placement functionality, we introduce a mechanism that allows each of these actors to influence the placement result. To the best of our knowledge, such a mechanism is not yet available in the domain, but it is our believe that it is an indispensable component for a carrier-grade NFV platform.

2.4.2 Design

The MANO Framework, shown in Fig. 2.8, has a flexible plug-in based design. The different components communicate by using a topic-based message broker. This allows the SP owner to easily replace one of the components to change its functionality on the fly, without the need to restart the framework. A more detailed description of the MANO Framework and its APIs can be found in the GitHub documentation [47], and in the SONATA deliverables D4.1[43], D4.2[44] and D4.3[45]. All the components described in this section are always present when the MANO Framework is operational, except for the specific managers (more details below).

The Plugin Manager (PM) is the bookkeeper of the MANO Framework. All other components register with the PM when they are started. When a component needs information on the available components, the PM is contacted.

The Service Lifecycle Manager (SLM) is the central component of the MANO Framework. It manages and orchestrates all events on the level of a NS. It is the contact point for the Gatekeeper, which converts customer requests into instructions for the SLM (e.g. instantiate, configure, terminate, update, scale a NS). The SLM goes through these workflows by involving the other MANO Framework components when needed.

The Function Lifecycle Manager (FLM) manages and orchestrates all events on the level of a network function. It gets instructions from the SLM. For example, when the SLM receives a request to instantiate a new NS, at some point the SLM will invoke the FLM to instantiate a network function (or multiple times if there are more than one VNF). The FLM will then, based on the VNF descriptor, contact the Infrastructure Adaptor, the component that streamlines and consumes the VIM (Virtualized Infrastructure Manager) APIs, and request the instantiation of the image. Other workflows that the current FLM implementation can perform are configuring and terminating VNFs. In future releases, the FLM can be extended with additional workflows, e.g. to scale a VNF.
The Placement Plugin (PP) encapsulates the placement logic. Every NS that requires instantiation requires a placement process. During this process, it is calculated which VNF image will be deployed on which VIM. The PP receives requests to calculate the placement from the SLM, and this request includes all the required data: i) topology of the infrastructure, ii) NS and VNF descriptors and iii) additional constraints and policies to be considered during the calculation (e.g. the exclusion of certain regions). This component was extended to contain the policy-based placement functionality. A detailed description of this extension, and how the policy-based placement mechanism works can be found in Sec. 2.4.4.

One of the main features of the MANO Framework is the concept of Service and Function Specific Managers, which allow developers to customise the MANO behaviour for a specific NS or VNF. The SP allows developers to add them to the descriptors of a NS or VNF, as shown in the NSD snippet below.

```json
name: vfoward-8-2-test
service_specific_managers:
  - description: placement SSM.
    id: sonssmvforwardplacement1
    image: 5gtango/vforward-placement
    options:
      - key: type
        value: placement
    vendor: eu.5gtango-nfv
    version: 0.8.2
```
These specific managers are processes, coded by the developer and embedded in docker containers, that incorporate service or function specific orchestration behaviour. The **Specific Manager Registry** (SMR) on-boards these docker containers, instantiates them and reserves an isolated message broker channel for them to use. This channel allows them to communicate with the MANO Framework, but prevents them from eavesdropping any of the communication that is not intended for them. Service Specific Managers (SSMs) customise orchestration behaviour for NS life cycle events such as NS instantiation, termination, scaling or healing, while Function Specific Managers (FSMs) customise function orchestration events such as starting, stopping, scaling or (re)configuring a VNF. In practice, When the SLM needs to execute a NS life cycle event, it will first check whether one or more SSMs are associated with this NS and workflow. If that is the case (e.g. the code snippet above shows how an SSM for workflow placement is made available), the SLM will request this SSM to execute the workflow, instead of using the generic workflow available in the MANO Framework. A similar behaviour can be expected for VNF life cycle events and FSMs. Through these mechanisms, a developer can customise the behaviour of the Service Platform with respect to their NS or VNF. This is a requirement for any realistic MANO Framework, as any non-trivial NS requires customised orchestration steps.

Other components, outside of the MANO Framework, can also achieve connection to the message broker. One of those is the **Monitoring Framework**. For example, at the latest stage of a NS instantiation workflow, the SLM uses the Monitoring Framework REST API to instruct it to start monitoring the new NS and VNFs. Once monitoring information is available for this NS, the monitoring framework publishes it on the message broker. This enables the MANO Framework to dynamically trigger new life cycle events such as reconfiguration, termination, migration, healing or scaling. The same goes for the Policy Manager, it can also post on the message broker to trigger new workflows in the MANO Framework.

The MANO Framework follows the ETSI design, as shown in Fig. 2.8, but the ETSI APIs (e.g. SOL002 [36] and SOL003 [37]) are not yet supported.

### 2.4.3 APIs

The MANO Framework uses a RabbitMQ message bus for its communication. Components communicate by subscribing to the same topics and publishing messages on those topics. It allows for one-to-one communication, as well as one-to-many. Tbl. 2.10 shows the different topics that are currently being consumed by the MANO Framework. A detailed overview of the payload formatting of the messages that are being published on these topics can be found in the GitHub wiki [47].

<table>
<thead>
<tr>
<th>Action</th>
<th>Consumer</th>
<th>Message Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy a VNF</td>
<td>FLM</td>
<td>mano.function.deploy</td>
</tr>
<tr>
<td>Start a VNF</td>
<td>FLM</td>
<td>mano.function.start</td>
</tr>
<tr>
<td>Stop a VNF</td>
<td>FLM</td>
<td>mano.function.stop</td>
</tr>
<tr>
<td>Configure a VNF</td>
<td>FLM</td>
<td>mano.function.configure</td>
</tr>
<tr>
<td>Request placement calc.</td>
<td>PP</td>
<td>mano.service.place</td>
</tr>
<tr>
<td>Register plugin</td>
<td>PM</td>
<td>platform.management.plugin.register</td>
</tr>
<tr>
<td>Deregister plugin</td>
<td>PM</td>
<td>platform.management.plugin.deregister</td>
</tr>
<tr>
<td>Request status of plugin</td>
<td>PM</td>
<td>platform.management.plugin.status</td>
</tr>
<tr>
<td>Send heartbeat</td>
<td>PM</td>
<td>platform.management.plugin.&lt;Plugin UUID&gt;.heartbeat</td>
</tr>
<tr>
<td>Instantiate NS</td>
<td>SLM</td>
<td>service.instances.create</td>
</tr>
<tr>
<td>Terminate NS</td>
<td>SLM</td>
<td>service.instance.terminate</td>
</tr>
<tr>
<td>Instruct FSM</td>
<td>FSM</td>
<td>generic.fsm.&lt;VNF Instance UUID&gt;</td>
</tr>
<tr>
<td>Instruct SSM</td>
<td>SLM</td>
<td>generic.ssm.&lt;NS Instance UUID&gt;</td>
</tr>
<tr>
<td>Reconfigure NS by SSM</td>
<td>SLM</td>
<td>monitor.ssm.&lt;NS Instance UUID&gt;</td>
</tr>
</tbody>
</table>
In order to support the Policy-based Placement feature, the API between the Placement Plugin and the SLM was extended. The SLM still sends a placement request on the `mano.service.place` topic, with the characteristics described next. Future API extensions will include topics to support updating, scaling, migrating and healing capabilities for NS and VNFs.

### 2.4.3.1 Request

To request a mapping, The SLM sends a request to the Placement Plugin. This message requires the following header fields:

- **app_id**: to indicate the sender of the message;
- **correlation_id**: a correlation id for the message;
- **reply_to**: the topic on which the sender expects a response, in this case `mano.service.place`.

The payload of the request is a YAML encoded dictionary that should include the following fields:

- **nsd**: the NS descriptor;
- **functions**: a list with the VNF descriptors;
- **topology**: a dictionary with the topology of the infrastructure, as received from the Infrastructure Adaptor (IA);
- **serv_id**: the service instance id of the service that is being placed;
- **nap**: a dictionary with information on the ip and the location of the sources and destinations of the service;
- **operator_policies**: A dictionary describing the operator policies. This dictionary has the following keys
  - **policy**: load balanced, fill first or priority
  - **policy_list** (optional): an ordered list that indicates priority of VIMs (referenced by their name). Default is empty.
  - **weights** (optional): a dictionary with operator and developer as key. There values should be a float between 0 and 1 and their sum should be 1. These values weigh the influence of both actors on the placement decision. They are set by the operator. Default for operator is 1 and developer 0.
• **customer_policies**: A dictionary describing the customer policies. This dictionary has a key `blacklist` with a list of VIM names as value.

The following snippet shows an example of a request payload.

```yaml
nsd: <nsd>
functions:
- <vnfd 1>
- <vnfd 2>
topology: <topology as exposed by IA>
serv_id: "12345678-1234-1234-123456789012"
nap:
- ingress:
  location: "Athens"
  nap: "192.168.1.1"
- egress:
  location: "Aveiro"
  nap: "172.101.1.1"
operator_policy: "priority"
policy_list:
- "Athens-200"
- "Dublin-East"
customer_policy:
- blacklist:
  - "Athens-200"
```

### 2.4.3.2 Response

The response of the Placement Plugin will have the following headers:

- **app_id**: To indicate which MANO Framework plugin responded;
- **correlation_id**: The same correlation id as in the request.

The payload of the message is a YAML encoded dictionary with the following fields:

- **mapping**: either None, when a mapping was not possible, or a dictionary. This dictionary has an element for each VNF Virtual Deployment Unit in the service. The id of the VNF instance is the key, the value is again a dictionary. This dictionary has `vim` as key, which has the id of that VIM as value.

An example of the response payload could be

```yaml
<vnf 1 uuid>:
  vim : <uuid of selected vim>
<vnf 2 uuid>:
  vim : <uuid of selected vim>
```

The topology payload in the requests contains a mapping between VIM names and VIM uuids.
2.4.4 Policy-based Placement

Different actors may want to control the placement process of a new NS, e.g.:

- A developer has specific knowledge that certain placements will result in a better QoS;
- The operator wants to orchestrate its infrastructure looking at the big picture, e.g. reducing its energy cost or ensuring that it can accommodate as much future requests as possible;
- A customer might want to prevent that certain VIMs are used for its NS, due to privacy considerations.

All three actors want to set constraints for the placement process. Some of them are hard (e.g. a VNF requires 2 CPUs), some of them are soft (e.g. Place a VNF as close as possible to the NS destination).

In 5GTANGO, we extend the MANO Framework with Policy-based placement capabilities. This section describes how we realize this. A more technical description, including evaluations of the mechanism, will be targeting academic publication in the near future.

When multiple actors set hard and soft constraints for the placement process, some of them might be conflicting. We need to model the placement problem and calculate the optimal solution. The following constraints / policies are supported by this MANO prototype:

- Developer constraints:
  - Request proximity between the source or destination of the NS and one of the VNFs (soft);
  - Request affinity of two VNFs, to ensure that they are colocated on the same VIM (soft);
  - Resource constraints, indicating how many cores and how much memory and storage are needed by a VNF (hard).

- Customer constraints:
  - Blacklist certain VIMs from usage for a NS (hard).

- Operator constraints:
  - Balance the load over all resources (soft);
  - Use as few VIMs as possible by filling those already used first (soft);
  - Prioritize certain VIM to be used first (soft).

All these constraints are collected by the SLM. The customer constraints are collected from the Customer by the Gatekeeper, and included in the NS instantiation request from the Gatekeeper to the SLM. The SLM can find it in this payload, under the blacklist key. The value is a list of vim_names that are not allowed to be used. The SLM will collect the operator policies by sending a message to the Policy Manager on the policy.embedding topic with an empty payload. The SLM will expect a response on the same topic. The payload should be a dictionary, with the operator_policy key. The value of this key should be either load balanced, fill first or priority. If the value is priority, another key needs to be in the payload: priority_list. The value of this is a list with VIM names. The order of this list indicates the priority of each VIM. The Developer constraints are embedded by the developer in the NS descriptor and VNF descriptors as follows:
resource_requirements:
  cpu:
    vcpus: 1
  memory:
    size: 2
    size_unit: "GB"
  storage:
    size: 20
    size_unit: "GB"

and

soft_constraints:
  - objective: "proximity"
    involved_actors:
      - "source"
      - "generic_function_1"
  - objective: "affinity"
    involved_actors:
      - "generic_function_1"
      - "generic_function_2"

For the soft constraints, the objective should either be proximity or affinity. The value of the field involved_actors should be a list of two elements out of source, destination or any of the vnf_id of the involved VNF.

Once the SLM has collected all this information, it collects the topology from the Infrastructure Adaptor. This payload describes all the available VIMs with their capacity and usage. This, together with the collected constraints is sent to the Placement Plugin. The SLM can also add weights to this message for the operator and developer constraints, to tweak the importance of each actor’s preference. The Placement Plugin translates the available resources and constraints into a Linear Program using the PuLP python library. Next, this Linear Program is solved using the Open Source GLPK [13] linear solver. The solution, a mapping of VNF images onto VIMs, is forwarded to the SLM, which carries on with the NS instantiation. In the request to the Placement Plugin, the SLM can set some weights to favor the policies coming from the developer or operator. These weights can be set by the operator, and enter the SLM through the SLM - Policy Manager interaction.

With this feature, the MANO Framework becomes very dynamic in terms of placement as all the involved actors can have a word in it.

### 2.5 Infrastructure Abstraction

This section addresses the Infrastructure Abstraction, a legacy component of SONATA that consumes the heterogeneous virtual infrastructure (both compute and networking resources) APIs.

#### 2.5.1 Overview

The Infrastructure Abstraction — also referenced as Infrastructure Adapter — is the southbound interface of the Service Platform. It lays just on top of the various Virtual/WAN Infrastructure Managers (VIMs/WIMs), either compute, storage or networking resources, that are being
orchestrated by the Service Platform. Examples of such VIMs are OpenStack, Kubernetes, OpenDayLight, AWS, etc. The objective of the Infrastructure Adapter is to streamline the APIs of the heterogeneous VIM market into one API for the MANO Framework to use. This allows the MANO Framework to be agnostic to the wide variety of VIM APIs. A first prototype of the Infrastructure Adapter was developed in the scope of the SONATA project. The Infrastructure Adapter of the 5GTANGO project will build further on that one. This first 5GTANGO prototype does not yet contain new features.

Detailed descriptions of the Infrastructure Adapter architecture can be found in SONATA Deliverables D4.1[43], D4.2[44] and D4.3[45].

### 2.5.2 APIs

The Infrastructure Adapter communicates by using a RabbitMQ message bus. The topics it consumes are listed in Tbl. 2.11. Detailed descriptions of the expected payload for each topic can be found in the Infrastructure Adapter GitHub wiki[50].

![Table 2.11: IA communication topics.](image)

<table>
<thead>
<tr>
<th>Action</th>
<th>Message Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a new compute VIM</td>
<td>infrastructure.management.compute.add</td>
</tr>
<tr>
<td>Add a new network VIM</td>
<td>infrastructure.management.network.add</td>
</tr>
<tr>
<td>List available compute VIMs</td>
<td>infrastructure.management.compute.list</td>
</tr>
<tr>
<td>List available network VIMs</td>
<td>infrastructure.management.network.list</td>
</tr>
<tr>
<td>Remove a compute VIM</td>
<td>infrastructure.management.compute.remove</td>
</tr>
<tr>
<td>Remove a network VIM</td>
<td>infrastructure.management.network.remove</td>
</tr>
<tr>
<td>Create a new stack</td>
<td>infrastructure.service.prepare</td>
</tr>
<tr>
<td>Deploy a VNF in a stack</td>
<td>infrastructure.function.deploy</td>
</tr>
<tr>
<td>Chain a set of VNFs</td>
<td>infrastructure.chain.configure</td>
</tr>
<tr>
<td>Unchain a set of VNFs</td>
<td>infrastructure.chain.deconfigure</td>
</tr>
<tr>
<td>Delete a stack</td>
<td>infrastructure.service.remove</td>
</tr>
<tr>
<td>Add a new WAN</td>
<td>infrastructure.management.wan.add</td>
</tr>
<tr>
<td>List available WANs</td>
<td>infrastructure.management.wan.list</td>
</tr>
<tr>
<td>Attach VIM to a WAN</td>
<td>infrastructure.management.wan.attach</td>
</tr>
<tr>
<td>Construct WAN for a NS</td>
<td>infrastructure.service.wan.configure</td>
</tr>
<tr>
<td>Destruct WAN for a NS</td>
<td>infrastructure.service.wan.deconfigure</td>
</tr>
</tbody>
</table>

### 2.6 VIMs/WIMs

This section describes the Virtual Infrastructure Managers (VIMs) and WAN Infrastructure Managers (WIMs).

#### 2.6.1 VIMs

For the Service Platform to initiate NSs, it will have to communicate with specific VIMs, which includes platforms as OpenStack, Kubernetes etc. Each VIM has its API providing distinct ways to connect the IA with it, accessing network and hardware resources. Each of these VIMs will have a Service Function Chaining (SFC) for the NSs that will allow the platform to deploy the service and adequately accommodate the wanted networking graph.

#### 2.6.2 WIMs

To control the WAN (Wide Area Network), VTN (Virtual Tenant Network) Manager is used, a component of OpenDaylight (ODL), that allows managing the network using SDN (Software-Defined Networks) technologies. With the VTN it is possible to set traffic flow rules, controlling
the network traffic, allowing, prohibiting or redirecting the packets that match the conditions applied. A RESTful API is provided that is integrated with the already existent VTN REST API and acts as a wrapper for the IA component, combining multiple VTN functions in a single call. The WIM also consists of a database that provides information about the hardware configuration of the infrastructure owner. The owner needs an SDN switch that they will have configured and registered to be controlled by ODL+VTN. This information can only be provided by the owner, and as a result, when the WIM is installed, the owner also has to populate the database with the required data. This data includes a particular segment of their deployment, the port id and virtual bridge name of where this segment is attached to the SDN switch, as well as an approximate physical location. More information on the WIM can be found in the SONATA Deliverable D4.3 [45].

2.7 Monitoring Manager

This section presents the extensions that have been foreseen for the first release (v0) of the 5GTANGO Service Platform, being in compliance with the requirements set in [30]. The starting point that fueled the decision for the implementation of the extensions mentioned below was primarily the requirements that have been identified from the use cases within 5GTANGO.

2.7.1 Overview

The Monitoring Framework, as developed and integrated within the Service Platform of SONATA, includes several components and provides a wealth of functionalities to fulfill the requirements of all involved stakeholders. A high-level architecture diagram is shown in Fig. 2.9.
As already discussed, there are three extensions that have been implemented during the first release of 5GTANGO Service Platform, as analyzed in the sections below, followed by the reasoning behind each extension decision.

### 2.7.1.1 Infrastructure-based monitoring collector

The decision in SONATA was to implement probes that would be installed in each VM comprising a VNF in a NS, so that generic as well as application-specific metrics could be measured and sent to the monitoring server for storage and further processing. The advantage was the ease of inclusion of such probe in the VM and the accuracy of the measurements.

However, during the 2nd ETSI NFV Plugtests [39], in Sophia Antipolis, France, the SONATA and 5GTANGO partners had the opportunity to discuss with several vendors, being interest to instantiate their VNF and NS in the 5GTANGO Service Platform. After discussing with them, it was evident that 5GTANGO should be able to provide a second monitoring “probe”, based on infrastructure monitoring, such as Ceilometer in OpenStack. This would provide the advantages that there is no need for “injecting” software in the VNFs. However, the inclusion of such monitoring tools on infrastructure level come with a couple of drawbacks: first, they are consuming large amount of memory in order to store data that might not be needed by the users or the infrastructure administrator, and second they are not appropriate for the VNFs that are closed source, as they might need specific metrics.

Consequently, these infrastructure-based monitoring solutions provide a constant set of metrics without the appropriate tools for extending them according to the developers needs.

Thus, in 5GTANGO, infrastructure-based monitoring solutions will be integrated in the existing monitoring framework architecture, while the solution that has been implemented and tested in SONATA will be kept active in order to cover the application-specific needs of the developers. So, developers will be able either to use infrastructure-level monitoring tools or provide their own tools to measure service-specific metrics. This is a mandatory approach not only because of the limitations of the developers with respect to the willingness to install probes in their network functions but also because there is a true need for flexibility to the monitoring framework in order to be able to easily expand the (application-specific) metrics to be monitored through custom tools.

Technically speaking, this will be realized by collecting monitoring data from those tools and integrate them to the Monitoring Server.

### 2.7.1.2 Provide monitoring on Service level

Another aspect of SONATA Monitoring Framework that has been revisited for extensions in 5GTANGO is the development of the capability to support monitoring not only on VM and VNF level but also (and most importantly for the majority of the scenarios) on NS level.

In particular, in the first release, the Monitoring Framework supports the case where two (or more) VMs comprise one VNF. This is shown in the Fig. 2.9, where on the left part, two VMs comprise one VNF. In addition the definition of rules and metrics on service level must also be supported, providing the ability to activate alerts on service level.

The implications on the implementation part are affecting the API, requiring extensions on the methods as will be described in sections below.

### 2.7.1.3 Support active monitoring measurements

In SONATA it became evident that Monitoring Framework must be enhanced in order to provide an effective way of measuring active networking data, such as latency, bandwidth, etc. on real-time
and upon request of the developer or service provider.

To accomplish this requirement, the Monitoring Framework is integrated with a Packet Generator tool, named MoonGen [8], offering API methods in order to POST the data to the Monitoring Server and GET methods so that the interested user is able to retrieve data. This integration will be used not only for testing purposes, in the light of V&V [30], but also for specific measurements related to the production environment of NSs.

2.7.2 Design and Implementation

This section provides insight on the design and implementation details with respect to the three extensions implemented in the first release of 5GTANGO Service Platform Monitoring Framework.

2.7.2.1 Infrastructure-based monitoring collector

In order to integrate the OpenStack Ceilometer measurements with Prometheus Server, a new service has been developed that gathers the monitoring data from Ceilometer publisher. Ceilometer publishers are components that make it possible to save the data into a persistent storage through the message bus, or to send it to one or more external consumers. This feature is taken advantage of in 5GTANGO in order to facilitate the transmission of the collected data from Ceilometer to Prometheus server and the respective time-series database.

The integration of tools such as Ceilometer have resulted in refactoring on the way that Prometheus deals with these new metrics, their representation on Ceilometer (e.g. UUID on SONATA implementation must be replaced by the triplet \{resource type, resource ID, metric ID\} on 5GTANGO implementation) and the relationships on the definition of NSs and slices that are kept in the monitoring database.

2.7.2.2 Provide monitoring on Service level

As mentioned earlier, this release is extended with respect to the cases where multiple VDUs (Virtualization Deployment Unit) comprise a single VNF.

In this first release of 5GTANGO, it is possible either to retrieve monitoring data on VDU level (collect all VDU-related metrics) or on VNF level (returning an array including the VDUs comprising the VNF under question and their respective set of monitored metrics).

2.7.2.3 Support active monitoring measurements

The data collected by MoonGen are parsed in a JSON format, including all the monitoring metrics and their values in a configurable time scale and with configurable measurement frequency. This makes the set of measurements quite large in size and difficult to be processed by the Monitoring Manager, as the exact processing of data relies solely on the developer demands.

Thus, in this release, the Monitoring Framework gathers the MoonGen data and store them as objects in a database supporting JSON objects.

2.7.3 APIs

The Monitoring Framework APIs are described in [2]. The following five API methods (Tbl. 2.12), are related to the extension of the VNF level monitoring:
Table 2.12: Monitoring extensions to VNF-level monitoring.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>List VNF Metrics</td>
<td>GET</td>
<td>/api/v1/prometheus/vnf/{vnf_id}/metrics/list</td>
</tr>
<tr>
<td>List VDU Metrics</td>
<td>GET</td>
<td>/api/v1/prometheus/vnf/{vnf_id}/vdus/{vdu_id}/metrics/list</td>
</tr>
<tr>
<td>VNF Metric Description</td>
<td>GET</td>
<td>/api/v1/prometheus/vnf/{vnf_id}/metrics/name/{metric_name}</td>
</tr>
<tr>
<td>Request VDU Metric Data</td>
<td>POST</td>
<td>/api/v1/prometheus/vnf/{vnf_id}/vdus/{vdu_id}/metrics/data/</td>
</tr>
<tr>
<td>Request VNF Metric Data</td>
<td>POST</td>
<td>/api/v1/prometheus/vnf/{vnf_id}/metrics/data/</td>
</tr>
</tbody>
</table>

The three API methods shown below (Tbl. 2.13) are related to the extension with respect to the integration with the MoonGen:

Table 2.13: Monitoring extensions to MoonGen.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Service Test List</td>
<td>GET</td>
<td>/api/v1/active/service/{service_id}/test/list</td>
</tr>
<tr>
<td>Get Service Test Data</td>
<td>GET</td>
<td>/api/v1/active/service/{service_id}/test/{test_id}</td>
</tr>
<tr>
<td>Add Service Test Data</td>
<td>POST</td>
<td>/api/v1/active/service/{service_id}/test/{test_id}</td>
</tr>
</tbody>
</table>

2.8 Slice Manager

This section describes the main functionalities and features that are supported by the Slice Manager [30] in the first version of the Service Platform prototype.

2.8.1 Overview

A Network Slice Instance (NSI) is defined in [35] as a set of network functions and the resources for these network functions which are arranged and configured, forming a complete logical network, which meets certain characteristics in terms of available bandwidth, latency, or QoS, among others described in 5QI (5G QoS Indicator).

3GPP has proposed a data model [14], which consists of a list of Network Slice Subnetworks instances (NSSI), which contain a set of network functions and the resources for these network functions which are arranged and configured to form a logical network. ETSI NFV and 5G Americas are the entities that directly map a NSI with NSs saying basically that the sub-network instances are the same as NSs.

The NSI contains NSSI, which in turn contain Network Functions (NFs) (e.g., belonging to Access Network (AN) and Core Network (CN)) as well as all information relevant to the interconnections between these NFs like topology of connections and individual link requirements (e.g. QoS attributes). The NSI is created by using a Network Slice Template (NST).

ETSI ISG NFV is actually highlighting the relationship between NSs and Slices / SubnetSlices [34]. This is important since the SP is familiar and supports the NS (and VNF) constructions. Our Slice Manager makes extensive use of SP internal components in order to deploy network slices.

2.8.2 Slice Manager Design

The proposed Network Slice Manager internal architecture is shown in Fig. 2.10. The Network Slice Manager is a Service Platform (SP) functional block that interacts with OSS (Operations Support Systems) and is responsible to access the SP MANO APIs for NS control.

The Network Slice Manager is a consumer of the REST APIs exposed by the MANO framework (reference point Os-Ma-nfvo [33]) and the Infrastructure Abstraction (IA). Within a Network Slice Manager, one can identify two main functions:
The first function is responsible for assigning services and applications to network slices and for managing the lifecycle of these slices. We will call it the *Slice Lifecycle Manager*.

The second function is responsible for mapping network slices to NSs. We will call it the *Slice2NS Mapper*.

### 2.8.3 Descriptors

The proposed YAML Schemas are based on ISO/IEC DIS 19086-2.

A Network Slice Template (NST) refers to the descriptor that details the Network Slice descriptor, which incorporates the requested NSs to be deployed for a network slice instance, and the relationship among them.

A Network Slice Instance (NSI) Record (NSIR) refers to the descriptor that includes the information of an instance of a network slice instance, which will use a network slice template reference for its deployment. The NSI will include references to the deployed NS Records (NSR).

At the Appendix can be seen an example of a Network Slice Template descriptor, in Sec. A.3 and the equivalent Network Slice Instance record in Sec. A.4.

### 2.8.4 APIs

#### 2.8.4.1 Network Slice Template APIs

The Tbl. 2.14 shows the API primitives to manage NSTs.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE NST</td>
<td>POST</td>
<td>/api/nst/v1/descriptors</td>
</tr>
<tr>
<td>GET ALL NST</td>
<td>GET</td>
<td>/api/nst/v1/descriptors</td>
</tr>
<tr>
<td>GET SPECIFIC NST</td>
<td>GET</td>
<td>/api/nst/v1/descriptors/{uuid}</td>
</tr>
<tr>
<td>DELETE NST</td>
<td>DELETE</td>
<td>/api/nst/v1/descriptors/{uuid}</td>
</tr>
</tbody>
</table>

#### 2.8.4.2 Network Slice Instance APIs

The Tbl. 2.15 shows the API primitives to manage NSIs. Note that the instantiation of a Network Slice is achieved with the first endpoint of this API, /api/nsilcm/v1/nsi.
Table 2.15: Network Slice Instance APIs.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create instance NSI</td>
<td>POST</td>
<td>/api/nsilcm/v1/nsi</td>
</tr>
<tr>
<td>Terminate NSI</td>
<td>POST</td>
<td>/api/nsilcm/v1/nsi/{nsiId}/terminate</td>
</tr>
<tr>
<td>GET all NSIs</td>
<td>GET</td>
<td>/api/nsilcm/v1/nsi</td>
</tr>
<tr>
<td>GET NSI</td>
<td>GET</td>
<td>/api/nsilcm/v1/nsi/{nsiId}</td>
</tr>
</tbody>
</table>

2.8.5 Design and Implementation

The first release of the **Slice Manager** has been programmed using **Python2**. Several libraries are used, such as **Flask**, **flask-restful**, **python-dateutil**, **python-uuid**. **Flask** library is extensively used for the **Slice Manager** North Bound Interface (NBI) API programming, as each API call results in a flask route.

Moreover, connectors to Repositories and MANO framework have been introduced, in order to properly interact with other SP components.

**Slice Lifecycle Manager**

The **Slice Lifecycle Manager** is responsible for the entire lifecycle management of the created network slice instance, until it is terminated. Subsequent lifecycle events are likely to have an impact on the lifecycle of the underlying NSs, but not systematically.

The Slice Lifecycle Manager function is responsible for the definition and update of Network Slice Templates (NST). If a customer facing service needs to be assigned to an existing NST, it can request a NST update. If not, a novel NST, might be created.

NSTs are then converted by the Slice2NS mapper into NSDs and flavours. Once the NSD has been on-boarded, a slice instantiation request can be triggered, resulting in an NS instantiation request with the appropriate flavour identifier being sent to the NSO.

**SLM** is responsible for interacting with SP Repositories and Catalogues.

**Slice2NS Mapper**

The **Slice2NS Mapper** functionality has to maintain an association between NST and NSDs identifiers, as well as an association between slice identifiers and NS instance identifiers. In next releases, it might also deal with the required combination/integration(concatenation of NSs.

This behaviour might involve automatic generation of a new NSD, which might be explored in future releases.

**Slice2NS Mapper** is responsible for interacting with SP MANO Framework.

2.8.6 Flow Diagrams

This section summarise the main flow diagrams involving the interaction with the **Slice Manager** (SM).

2.8.6.1 Creation of a NST

The SM is the service platform new component, responsible to manage the life cycle of Network Slices. The on-boarded NSTs are stored in the Catalogue after previous validation. The Slice template on-boarding workflow is depicted in Fig. 2.11. Depending on the validation level to be performed, the Catalogue could be also involved, to verify whether the involved NFV building blocks (NSs) are available.
2.8.6.2 Instantiation of a NSI

Once a Network Slice Template (NST) is on-boarded, multiple Network Slice Instances (NSI) can be created using this template. The Network Slice instantiation triggers the instantiation of the list of NSs defined in the NST, which are used as building blocks to compose the NSI. The multiple NSs may need to be combined to form a meaningful network/service.

OSS is the entity that usually triggers the Network Slice instantiation, although it may also be triggered automatically (autonomous behaviour). The NSI creation workflow is depicted in Fig. 2.12.

Workflow description:

1. OSS requests the NSI Creation to the Gatekeeper;
2. The Gatekeeper relies the NSI Creation to the Slice Manager, translating to the internal API;
3. The Slice Manager (Slice2NSMapping component) finds on the NST the underlying artifacts (NSs) to be deployed for this NSI;
4. For every NS to be deployed, request the MANO Framework to instantiate it;
5. If resources are not available, notify the Slice Manager with the appropriate Error;
6. In turn, the Slice Manager notifies the Gatekeeper;
7. In turn, the Gatekeeper notifies the OSSs;
8. If resources are available, respond Instantiation OK;
9. In turn, the Slice Manager responds OK to the Gatekeeper;
10. In turn, the Gatekeeper responds OK to OSSs.

### 2.9 Policy Manager

This section describes the main functionalities and features that are supported by the **Runtime Policies Manager** [30] in the first version of the Service Platform. It should be noted that the Placement Policy features are implemented as part of the MANO Framework component Sec. 2.4.

#### 2.9.1 Overview

As detailed in [30], the Runtime Policies Manager is responsible for the enforcement of such policies over an instantiated NS. Towards the design and enforcement of a runtime policy, the following workflow is followed.

- Runtime policies definition is realised through a Policies Editor by the Service Provider, taking into account - optionally - the set of defined rules already available in the NSD and VNFDs. The final set of rules is based on the selections, insertions and updates on behalf of the Service Provider, avoiding in this way any potential contradictory rules. The outcome of this process is a Policy Descriptor associated with a NS that follows a specific policy description schema.
• The created policy descriptor is validated. Actually, during the policy creation, validation mechanisms are applied within the policies editor, leading to a YAML-valid policies description.

• The validated policy descriptor is translated to Drools (rule based management system with specific syntax for rules description) [4] that regards the engine being used for realizing real-time inference. The outcome of this process is a Drools file containing the set of expressions included in the policy.

• Upon the instantiation of a NS, the Drools file associated with the NS policy is loaded to the Policy Manager components (Production Memory) responsible for keeping track of the set of defined rules.

• Policies enforcement is realized during the overall lifecycle of the NS instance. Upon deployment of a NS, the selected policy is activated, leading to consumption of data and alerts made available in the Message Broker. Real time inference is taking place, taking into account the denoted set of rules, leading to the publication of suggested actions to the Message Broker in order to be consumed by the interested orchestration components. Upon termination of the NS instance, the associated rules are de-activated.

Following, we provide specifications with regards to the runtime policies descriptor, along with an example with the produced Drools file is provided and the set of defined APIs for creating and managing policies within the 5GTANGO Service Platform.

2.9.2 Runtime Policies Facets

Regarding the set of metrics to be included in the conditions part, it can be associated with configuration options of each VNF (component), resources usage metrics, VNF (component) profiling information, NS metrics focusing mainly on virtual links QoS characteristics as well as overall resources usage metrics. * Component configuration metrics are related to metrics denoted on behalf of the application developer as component specific metrics and made available through the VNF descriptor. * Resource usage metrics regard the average/min/max consumption of resources from the deployed component image over the virtualized infrastructure. * Component characterization metrics regard the profiling result of a component and its mapping with one or more characteristics. * With regards to network link monitoring metrics, a set of QoS characteristics are considered, as they can be mainly provided through the monitoring infrastructure of communication service providers.

The mindmap Fig. 2.13 depicts the main conditions categories.

With regards to the actions part of a policy definition Fig. 2.14, a set of actions are defined based on the targeted entity to apply them, namely VNFs (component) and NS. Prior to detailing the set of potential actions, it should be noted that no specific binding between conditions and actions exist, however some combinations may not be available, taking into account that there is no related business logic. * Component actions may be associated with an orchestration functionality (e.g. spawn or deprovision a number of VMs), a change in the configuration of custom metrics of a component (e.g. change in the transcoding quality level), a change in the resource allocated to the component (e.g. migration to a different VM that may be requested upon mobility of the end user in an edge computing application scenario), etc. * NS actions regard activation or altering of configuration of network monitoring and management mechanisms (e.g. monitoring of end-to-end delay among two connection endpoints, deployment of a network function or service, establishment of a VPN, etc.).
Figure 2.13: Policies Conditions.

Figure 2.14: Policies Actions.
2.9.3 Runtime Policies Descriptor

A NS Runtime Policy Descriptor (RPD) is a deployment template attached optionally to a NS referencing a set of enforcement rules, upon which certain actions are taken in order to meet some objectives described by specific SLAs. Policies themselves are formulated as event-condition-actions tuples with the semantics of “on event(s), if condition(s), do action(s).”

Our NS policy descriptor schema specifies the runtime policy to be enforced upon an NSD instantiation. It is based on concepts defined by state-of-the-art policy models such as SUPA [6], DEN-ng Policy Model [9], XACML [5] and The Policy Continuum [10]. It is, however, adapted and extended to meet the 5GTANGO specific needs.

The schema is written in YAML and can be easily translated to JSON [7], e.g. by using a json-to-yaml translator [11].

2.9.4 Sections of the Runtime Policy Descriptor

Following, we discuss the various sections of a policy descriptor. The general descriptor section contains some of the mandatory fields that have to be present in each and every policy descriptor. All other sections might be optional.

2.9.4.1 General Descriptor Section

At the root level, we first have the mandatory fields, that describe and identify the policy in a unique way.

- **descriptor_schema** referenced to the corresponding schema of a descriptor (e.g., URL or local path)

- **$schema** (optional) provides a link to the schema that is used to describe the policy and can be used to validate the policy descriptor file. This is related to the original JSON schema specification.

Moreover, the policy signature, i.e the **name**, is of great importance as it identifies the policy uniquely.

- **name** is the name of the policy. It can be created with any name written in lower letters and no strange symbols.

- **network_service** is the metadata of the network service this policy refers to. A network_service is defined in an unique way by the following fields (**vendor, name, version**). A network_service may have more than one policy descriptors, but a policy descriptor always refers only to one network_service.

The general descriptor section also contains optional components as outlined below.

- **author** (optional) describes the author of the policy descriptor.

- **description** (optional) provides an arbitrary description of the policy.
2.9.4.2 Policy Rules Section

A policy has to include a list with at least one policy rule. For each policy rules, the following three fields are mandatory:

- **name** is the unique name of the policy rule. It can be created with any name written in lower letters and no strange symbols.

- **conditions** is a set of expressions that bind with logical operators the conditions and thresholds that should be satisfied in order to trigger the actions list. It is detailed at the Conditions section.

- **actions** is a set of actions that can be enforced upon the satisfaction of the Conditions section. It is detailed at the Actions section.

Each policy rule also contains optional components as outlined below.

- **salience** (optional) Each rule has an integer priority attribute which defaults to zero and can be negative or positive. Salience is a form of priority where rules with higher priority values are given higher priority when ordered in the Activation queue of the inference engine.

- **inertia** (optional) indicates the period of time (minutes, hours, days) the policy rule stays inactive after being triggered.

- **duration** (optional) and **aggregation** (optional) indicate the specific time window and aggregation function (min,max,average). Sliding time windows allow the enablement of rules that will only match events occurring in the last X time units.

2.9.4.3 Conditions Section

The conditions section contains all the information regarding the events and thresholds that should be satisfied in order to trigger the actions list. The section is mandatory and starts with:

- **condition** represents the logical operator to join with the rule conditions presented by the field “rules”.

- **rules** can be a list of expressions as defined at expression section OR can have the form of a nested recursive conditions, e.g. contain a condition and rules fields.

2.9.4.4 Expression Section

The expression section contains all the information regarding how a threshold can be matched to a specific event. The section is mandatory and contains the following fields:

- **id** the id of the event.

- **field** the field as shown to the policy editor.

- **type** the type of the event value. This can be string, double or integer.

- **operator** the operator of the threshold. This is defined at the operator enumeration and for the time being can be equal, less or greater.

- **value** the value of the threshold.
2.9.4.5 Actions Section

The actions section contains all the information regarding the actions that can be enforced upon the satisfaction of the conditions section. The section is mandatory and starts with:

- **actions** contains all the actions that can be enforced by the various orchestration 5GTANGO mechanisms.

This section has to have at least one item with the following information:

- **action_type** represents the type of the action. Action types come from a declared enumeration. For the time being, the following types are supported: *Infrastructure, Orchestration, LifecycleManagement, NetworkMechanism, AlterConfiguration, Profile & Log*.

- **name** represents the name of the action. Action names are also an enumeration and depend on the action types. For example, the *LifecycleManagement* type can get the name of start, stop or restart.

- **target** refers to the NS / VNF / VDU instance where the action is going to be enforced.

Each action also contains optional components as outlined below.

- **stability_period** indicates the period of time (minutes, hours, days) this action should not be enforced again upon being triggered.

In the Appendix it can be seen an example of a policy descriptor Sec. A.1 and the equivalent representation of the policy rules as Drool Sec. A.2.

2.9.5 APIs

Tbl. 2.16 shows the API endpoints of the Policy Manager.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a Policy</td>
<td>POST</td>
<td>/api/policymngr/v1/policy-descriptor</td>
</tr>
<tr>
<td>Delete a Policy</td>
<td>DELETE</td>
<td>/api/policymngr/v1/policy-descriptor/{policyDescriptorUuid}</td>
</tr>
<tr>
<td>View existing Policies</td>
<td>GET</td>
<td>/api/policymngr/v1/policy-descriptor/list</td>
</tr>
<tr>
<td>Enforce Policy</td>
<td>POST</td>
<td>/api/policymngr/v1/policy-descriptor/activate</td>
</tr>
<tr>
<td>Deactivate Policy</td>
<td>POST</td>
<td>/api/policymngr/v1/policy-descriptor/deactivate</td>
</tr>
</tbody>
</table>

2.9.6 Implementation Status

At the current phase of the project, the first version of the Policy Manager is implemented and integrated in the first operational prototype of the Service Platform. Focus is given on the interaction with the Message Broker and the consumption and publication of messages in the predefined policy-oriented topics. Based on the provided messages through the Monitoring Manager, real-time inference is taking place in the deployed Drools rules-based management system and the triggered suggestions are made available to the Message Broker in order to be consumed by various orchestration components.
The main technologies used for the development of the various components include Java 8, which has been used as the programming language for coding the tng-policy-mngr component. Since within java exist many application frameworks, it’s worth to mention that the policy manager makes use of the Spring Framework, which is an application framework and inversion of control container for the Java platform. The Policy Manager component is built and deployed with the Apache Maven build automation tool. Maven addresses two aspects of building software: first, it describes how software is built and second, it describes its dependencies. Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories, such as the Maven 2 Central Repository, and stores them in a local cache.

The Policy Manager has been developed making use of the Drools technology. Drools is a Business Logic integration Platform (BLiP). It is written in Java and is an open source project [4]. Drools give the mechanics to use a powerful rule engine that allows to define “What to Do” and not “How to do it.” Unlike codes, Rules are written in less complex language which means that service platform end user can easily create and verify a set of rules. With the help of Drools, policy enforcement becomes very scalable. Synchronous communication between the Policy Manager and the rest of the components is done via REST APIs, while asynchronous communication is done via the use of a very well known and open source pub/sub framework, RabbitMQ.

2.10 SLA Manager

This sub-section describes the features that will be implemented for the Service Level Agreements (SLAs) Management Framework, in order to support the life-cycle of SLAs.

2.10.1 Overview

The 5GTANGO SLA Framework is a component of the Service Platform, that allows to manage the whole life-cycle of service level agreements - from template creation to violation detection. Considering that the Service Platform aims at supporting the interaction between a distributed set of users and resources, the implementation of an SLA Management Framework aims at governing this interaction. Since each component potentially affects the overall behavior of the platform, any high-level parameter (e.g. performance, availability) specified for the service in an SLA, will be linked to low-level requirements (e.g. CPU, RAM) – encapsulated in the respective policies. This mapping will be based in an Artificial Neural Network and the approach is briefly described in the SLA Manager Prototype Roadmap Sec. 6.4.

The SLA Manager should consider two phases: Pre-deployment (Template Management) and after deployment (Agreement Management). It is worth mentioning, that SLA Templates, are advertisements from the service provider to the customer regarding an available Network Service. Initial Objectives (SLOs – High Level Requirements) about the Network Service are included in the Template, and can be changed if desired. On the contrary, SLA Agreements are contracts that define the level of service agreed by both stakeholders, and cannot be changed without penalties.

The prototype of the SLA Management Framework included in the first 5GTANGO Software release is a new component, added to the SONATA Service Platform, that focuses mainly on the Template Management.

2.10.2 5GTANGO SLA Manager Design

As detailed in [30], the SLA Manager is responsible for Template and Agreements generation and management, efficient high-to-low level requirement mapping, as well as QoS optimization through
mechanisms that take place after the successful deployment of a NS. Towards the design of the SLA Manager, the following mechanisms were considered:

- **SLA Template Generator**: creates an initial and tailored SLA template for the service provider, and the final SLA itself;
- **SLA Mapping Mechanism**: mapping between the high-level requirements described by the end-user and the low-level requirements described by the service provider;
- **SLA Parameter Analyzer**: decide whether the process of the mapping mechanism should be done or not, as described in 5gtango D2.2 [30];
- **SLA Mapping Repository**: store the correlations between the high-level and the low-level requirements (mapping mechanism output);
- **SLA Monitor Analyzer**: Compare the QoS parameters from the Mapping Repository, with the computed monitoring measurements and check if there is any violation.

The SLA Manager’s components and how they relate to each other are shown in Fig. 2.15.

In the first software release of 5GTANGO, we focused mainly on the SLA Template Generator, in order to create efficient templates (i.e. advertisements) that will be available to customers, taking in mind the Service Provider’s preferences, as well as the NS requirements and the corresponding policy rules.

### 2.10.3 Descriptors

An SLA Template Descriptor, refers to the core SLA Template document that incorporates metrics, as specific objectives or quality attributes, parameters as well as expressions (i.e. rules) between parameters. The proposed YAML Schema, based on ISO/IEC DIS 19086-2, aims to specify the
main “building blocks” of an SLA template and also present an expression (i.e. function) that allows a Service Provider to specify any metric included in a template (e.g. availability, response time, etc.) Therefore, the main SLA Templates building block of the reference model are the following:

- root block with general descriptions;
- sla_template with details regarding the SLA Template and its objectives.

Examples of SLA Templates as well as the schema, are available at the 5GTANGO Schema Repository [18], as well as at the Appendix Sec. A.5. Each SLA Template descriptor is comprised by three main sections. The general description, the SLA Template and the NS section.

2.10.3.1 General Descriptor Section
At the root block, we first have the mandatory fields, that describe and identify the SLA template descriptor in a unique way.

- schema (optional) provides a link to the schema that is used to describe the SLA template and can be used to validate the SLA template descriptor file. This is related to the original JSON schema specification;
- name is the name of the SLA template;
- version is the version of the SLA template.

The general descriptor block also contains some optional components as outlined below.

- author (optional) describes the author of the SLA template descriptor;
- description (optional) provides an arbitrary description of the SLA template.

2.10.3.2 SLA Template Section
An SLA template descriptor has to include an sla_template block. The following two fields are mandatory:

- offered_date is the creation date of the template;
- valid_until is the expiration date of the template.

2.10.3.3 NS Section
An SLA template descriptor has to include at least one NS block as child of sla_template block. The following two fields are mandatory:

- objectives that describes the High Level Service Level Objective (SLO) (e.g. NS availability). The NS block contains at least one objective.
  - metrics that corresponds to how to measure the objective. Each objective contains at least one metric.
  - rate that describes the specific time window.
*expression* that describes a function under which the specific metric of the SLA should obey. Each metric contains only one expression.

- parameters “links” the metric with a set of parameters that need to be accompanied with the metrics and included into the expression (expressing in detail each metric). An expression contains at least one parameter.

### 2.10.4 APIs

One of the key points in the SLA Management Framework, is the provision of an NS with the corresponding quality requirements tailored to it. In the 5GTANGO Service Platform, we focus on the management of NS SLAs through REST-based APIs.

The SLA Manager provides mechanisms to support service level agreements management. The capabilities that the SLA Manager of the 5GTANGO Service Platform are:

- Definition and advertisement of the capabilities of an NS provided by a service providers in SLA Template format;
- Creation of SLA Agreements based on the templates;
- One-shot negotiation process. *(i.e. takes place between the customer and the Service Provider only one time and focus on customer’s high-level demands)*.

In the first 5GTANGO Software release, the capabilities implemented refer to the first mentioned point, and include the SLA Template Management. Most of the APIs are described in the 5GTANGO Catalogue in the SLA APIs section. Briefly, the ones related to the SLA Manager and implemented from the 5GTANGO Catalogue are inTbl. 2.17.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload an SLA Template to Catalogue</td>
<td>POST</td>
<td>api/catalogues/v2/slas/template-descriptors</td>
</tr>
<tr>
<td>List of all available SLA Templates</td>
<td>GET</td>
<td>/api/catalogues/v2/slas/template-descriptors</td>
</tr>
<tr>
<td>List of all published SLA Templates</td>
<td>GET</td>
<td>/api/catalogues/v2/slas/template-descriptors?published=true</td>
</tr>
<tr>
<td>List SLA Templates per UUID</td>
<td>GET</td>
<td>/api/catalogues/v2/slas/template-descriptors/{uid}</td>
</tr>
<tr>
<td>Delete an SLA Template</td>
<td>DELETE</td>
<td>/api/catalogues/v2/slas/template-descriptors/{uid}</td>
</tr>
</tbody>
</table>

In addition to those, the SLA Manager component supports two additional APIs in its own core. These are described in the Tbl. 2.18.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a new template</td>
<td>POST</td>
<td>/api/slas/v1/templates/{nsd_uuid}</td>
</tr>
<tr>
<td>Edit an existing template</td>
<td>PUT</td>
<td>/api/slas/v1/templates/{sla_uuid}</td>
</tr>
<tr>
<td>Customize an existing template</td>
<td>PUT</td>
<td>/api/slas/v1/templates/customize /{sla_uuid}</td>
</tr>
</tbody>
</table>

### 2.10.5 Design and Implementation

The SLA Template Management enters the flow prior to the NS instantiation, and focuses on automated templates generation, through the Template Generator Mechanism. The Generator is able to obtain a set of policy rules for the specific NS by the Policy Framework, NS information,
such as NS description and the corresponding monitoring parameters from the NS Descriptor. In the future, the mechanism tends to obtain as well historical monitoring data from the Monitoring Manager (i.e. NS performance data, resource parameters preferences). The mechanism analyzes and correlates those information, in a way that can formulate a generalized SLA Template with initial objectives about a specific NS. Great emphasis has been placed on the customization of the Templates as well, in order to be tailored to the Service Provider’s needs. After the Template is created, the Provider is able to add more explicit metrics in order to enrich it. In next version of the mechanism, the analysis is planned to use Artificial Intelligence (AI). Firstly, clustering algorithms should be applied, to assign a set of observations (i.e. monitoring data, historical data, previous SLA template parameters) into subsets so that observations in the same subset are similar in some sense. Then, a classification model should be developed, in order to come up to a conclusion from the observed subsets, and use this information to include better QoS parameters into the templates.

Java has been used as programming language for coding the tng-sla-mgmt so far, and especially the template generator plugin. It’s worth mentioning that the SLA Generator makes use of the Jersey Framework. Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services in Java.

2.10.6 Flow Diagrams

The SLA Template Generation flow, is an important evolution to the current SLA Management Frameworks, as it defines automatically the advertisement of the capabilities of a NS provided by a service provider in SLA Template format. In more detail the Generator have access both to the NS-Descriptors and its associated set of policies in order to correlate this information. The aim of the analysis is to formulate a generalized SLA Template, applicable to other services. At the same time, it could be adjusted to the corresponding needs, by adding more explicit parameters and generate a tailored SLA Template to each Service Provider.

2.10.6.1 SLA Template Generation

The SLA Template Generation flow is shown in Fig. 2.16.

1. The Service Provider requests through the Portal, all the available NSs;

2. The Service Provider can check the provided NSs that the template terms can apply to. This information is retrieved from the 5GTANGO Catalogue through the Gatekeeper;

3. Once the Service Provider has selected a NS, he/she requests the generation of an SLA Template;

4. Then, the SLA Manager, retrieves automatically the appropriate information (NS Descriptors and Policy Rules) from both the 5GTANGO Catalogue and the Policy Manager in order to create the template.

2.10.6.2 SLA Template Customization

The SLA Template Customization flow is shown in Fig. 2.17.

Workflows description (high-level):

- Once the template is generated, it is available to the Service Provider through the Portal in order to a) publish it to the customers and b) customize it.
Figure 2.16: SLA Template Generation Flow.

Figure 2.17: SLA Template Modification Flow.
The Service Provider can add additional business values to the template.

The information is sent to the SLA Manager through the Gatekeeper.

The SLA Manager is responsible to edit the current SLA template and send it back to the Catalogue.

The Catalogue updates the current template, make its state published.

Finally, the list of available templates is updated and available to the Portal.

2.10.6.3 Published and Active SLA Templates

When an SLA Template is uploaded to the 5GTANGO Catalogue, some additional fields are attached to the SLA Template Descriptor (SLAD). Those fields refer to the state and status of a specific SLA Template. The status can be either active or inactive, and the state published or unpublished. It is worth mentioning that the state is needed in order the SLA Manager to know when an SLA Template is available to the customers. If the template is published, that means the Provider has no more power over it and cannot modify it, only delete it. Also, the deletion of a template means that it is unpublished and at the same time inactive. Those actions are presented in detail in Fig. 2.18.

2.11 Portal

5GTANGO portal is a Web User Interface (WUI) developed within the project in order to be able to manage different elements of the 5GTANGO architecture.

It is being developed in Angular (v5.2.9) following Material Design guidelines. It interacts with the rest of the system using the REST API interfaces exposed by the Gatekeeper which routes the request to the right service. The Portal does not require any back-end and it relies completely on the API exposed by the different platform microservices through the 5GTANGO Gatekeeper.

Nginx [3] is being used as the web server for the WUI files. The Gatekeeper API is consumed directly from the browser by the Portal web application.

It has a modular internal architecture and the manager can enable and disabled depending on the elements available in the system.

The Portal can be deployed as a container which consists of an Nginx instance serving the Portal files.

The Portal is divided in these modules which can be enabled or disabled on a per user basis depending on the user permissions which can be set from the portal:

- **Dashboard screen**: it is the first screen (Fig. 2.19) shown after the login and gives a general idea of the status of the platform. The elements and values which are shown there can be customized.
• Users section: this section allows to create and edit the users which can access the different sections of the WUI.

• Validation and Verification section: this sections manages the V&V service developed within the scope of 5GTANGO project. It will allow to manage NS and VNF tests and check their results and the rest of verification procedures (see [27]).

• Service platform section: this section allows listing available services and service and functions’ records, as well as the following menus:
  – SLA: from this sub-menu it is possible to manage SLA agreements and templates.
  – Policies: from this sub-menu it is possible to manage runtime and placement policies.
  – Slices: from this sub-menu it is possible to manage NSTs.

• Service management section: from this section Operator users will be able to manage user and service licensing, list the available NS and instantiate/manage NS. Fig. 2.20 shows the screen to check pending instantiation requests.

The Portal will evolve with the whole platform, with features being added as they are developed.
3 Interfaces

This section describes the interfaces of the 5GTANGO Service Platform prototype. In particular, are described the interfaces among the internal components as well as the interactions of the 5GTANGO SP with external entities and stakeholders.

3.1 SP Internal Interfaces

This section describes the interfaces among the SP internal components.

3.1.1 Gatekeeper / Catalogues

For the first 5GTANGO prototype, the interactions between the Gatekeeper and the Catalogues are based on the APIs exposed by the Catalogues. The interface between the Gatekeeper and the 5GTANGO Catalogue should support the following interactions:

1. Storing VNF, NS, NST, Policy, SLA template and Test descriptors along with package files in the catalogue;
2. Retrieving VNF, NS, NST, Policy, SLA template and Test descriptors along with package files from the catalogue;
3. Deleting VNF, NS, NST, Policy, SLA template and Test descriptors along with package files from the catalogue;
4. Updating VNF, NS, NST, Policy, SLA template and Test descriptors along with package files in the catalogue;
5. Searching the Catalogue for the available VNF, NS, NST, Policy, SLA template and Test descriptors.

As the above list suggests, the interactions are asynchronous between the 5GTANGO Catalogues and the Gatekeeper.

For writing in the 5GTANGO Catalogue, there is a store function. This will put a 5GTANGO package in the catalogue (which contains the binary data of the VNF(s)/NS and their descriptors) or just descriptors to an existing item (VNF(s), NS, packages etc.). The descriptors will be stored in the database and when storing a package, the binary data will be stored in the filesystem.

The retrieval of hosted objects from the 5GTANGO catalogue will fetch from the catalogue their package or descriptors.

The deletion of the hosted objects in the 5GTANGO catalogue is available. In the case where either dependency is present between the instances and the objects under deletion or the VNF(s)/NS selected for deletion are currently instantiated, a mechanism first checks any conflicts before proceeding with the deletion.

In addition, the update function will be used in the 5GTANGO Catalogues. By selecting an item (VNF/NS/package etc.) to be updated, a newer version can be stored while the older version can still be accessed.
In order to discover the objects (VNFs/NS/packages etc.) that are in the 5GTANGO catalogue, the search function can be used. A list can be obtained with the available VNFs/NS/packages and some search parameters can be specified to limit the number of objects in the corresponding list.

### 3.1.2 Gatekeeper / MANO Framework

For this prototype, the interface between the Gatekeeper and the MANO Framework exists out of two APIs. Both are exposed by the MANO Framework and consumed by the Gatekeeper.

1. The GK can request the MANO Framework to instantiate a new NS, by posting a message on the message bus, on the `service.instances.create` topic. The details of this request and response are described here in the MANO Framework GitHub [47].

2. The GK can request the MANO Framework to terminate a running NS instance, by posting a message on the message bus, on the `service.instances.terminate` topic. The details of this request and response are described here in the MANO Framework GitHub [47].

Compared to the latest SONATA prototype, the NS instantiate API was extended so that the payload contains a `blacklist` key. The value of this key is a list of VIM names. The content of this list resembles those VIMs that the customer who requested the new NS does not wish to be used.

### 3.1.3 Gatekeeper / Slice Manager

Interactions between the Gatekeeper and the Slice Manager are basically Managing slices: the whole slice lifecycle will have to be supported by the Gatekeeper, namely to give the GUI access to it.

In particular, the Gatekeeper will be the front-end for APIs related to the management of NSTs, as well as the life cycle management of NSIs:

- On-board NST, to load Network Slice templates into the platform;
- Update NST, to update the templates on the platform;
- Enable/Disable NST, to enable/disable temporarily NSTs on the platform;
- Delete NST, to delete permanently NSTs on the platform;
- Instantiate NSI, to instantiate a Network Slice instance, based on a NST;
- Update/Scale/Heal NST, to modify a running NSI at different levels;
- Terminate NSI, to terminate a running NSI.

### 3.1.4 Gatekeeper / Policy Manager

The interface between the Gatekeeper and the Policy Manager currently supports the following interactions:

1. Request the enforcement of a policy accompanying the instantiation of a new service or the dynamic management of a NSI; During the instantiation of a new service and in case that the NS is associated with a policy, policy enforcement mechanisms are also activated.
2. Report on the inference results by the Policy Manager and publication of the relevant suggestions and alerts in the Message Broker; Based on the inference results through the examination of the set of rules over the collected data, alerts are produced and made available through the Message Broker.

3. Request the termination of a policy’s enforcement and the collection of relevant monitoring data, upon the termination of the relevant running service instance or the disposal of the created NSI.

### 3.1.5 Gatekeeper / SLA Manager

Interactions between the **Gatekeeper** and the **SLA Manager** are those which will allow the **Service Platform Owner** to manage the different SLAs created in the Service Platform.

The interface for the first software release of 5GTANGO supports the following interactions:

1. Request the generation of SLA Templates. The GK can request the generation of a SLA Template for a specific NS. The input needed by the GK is limited to the NS uuid as the generation follows an automatic process.

2. Manage all available SLA Templates
   - The GK can request the editing of specific fields in the generated template. The editable fields include the name and description of an SLA Template, definitions of all the three major building blocks (i.e. SLO, metric and parameter definition), as well as both parameter and SLO value. This editing allows the Service Provider to get a generalized SLA template in an automated way, and if desired to make it tailored to his/hers needs.
   - The GK can request the addition on new objectives inside the generated template.

### 3.1.6 Slice Manager / MANO Framework

To create slice instances, the Slice Manager instructs the MANO Framework to deploy the required NSs, one by one. Therefore, the Slice Manager uses the internal API to invoke the MANO Framework.

For this prototype, the interactions between the Slice Manager and the MANO Framework uses mainly two APIs, both are exposed by the MANO Framework and consumed by Slice Manager:

1. The Slice Manager can request the MANO Framework to instantiate a new NS, by posting a message on the Message Bus, on the `service.instances.create` topic. The details of this request and response are described in the MANO Framework GitHub [47].

2. The Slice Manager can request the MANO Framework to terminate a running NS instance, by posting a message on the Message Bus, on the `service.instances.terminate` topic. The details of this request and response are described in the MANO Framework GitHub [47].

3. In latter versions of the SP prototype, the Slice Manager could request the MANO Framework other LifeCycle Management operations, such as Update, Scale or Heal, among others, in order to Update, Scale or Heal a NSI.
3.1.7 Slice Manager / Repositories

The Slice Manager interacts with the Repositories in order to store Network Slice Instances Records (NSIR) on the repositories. In particular, the following are the possible actions:

- The Slice Manager can create a record in the Repositories that should be stored
- The Slice Manager can update an existing record that the Repositories should be adapted
- The Slice Manager can request a record from the Repositories
- The Slice Manager can delete a record from the Repositories

The Repositories expose a REST API to provide these actions in Sec. 2.3.

3.1.8 Slice Manager / Catalogues

The interface between the Slice Manager and the 5GTANGO Catalogue supports the following interactions:

1. Storing NSTs in the Catalogue;
2. Retrieving NSTs from the Catalogue;
3. Deleting NSTs from the Catalogue;
4. Updating NSTs in the Catalogue;
5. Searching in the Catalogue for the available NSTs.

For writing in the 5GTANGO Catalogue, there is an upload function. After the examination of every descriptor itself and the rejection of any possible duplicate storage, the NST will be stored in the database.

When NSTs or NSD need to be retrieved from the 5GTANGO Catalogue, the retrieve function will fetch from the catalogue the appropriate descriptors.

When NSTs or NSD need to be deleted from the 5GTANGO Catalogue, the delete function can be used. Because there might be other services that depend on the items to be deleted or the NSTs selected for deletion are currently instantiated a mechanism first checks any conflicts before proceeding with the deletion.

The update function will be used to put a NST updated to the 5GTANGO Catalogue. By selecting a NST or NSD to be updated, a newer version of that item is stored while the older version is still accessible. This works with the 5GTANGO Catalogues’ revision and version control mechanism tracking the possible changes.

To retrieve the NST that exists inside the 5GTANGO Catalogue, the search function can be used where a list can be obtained with the available NSTs and some search parameters can be specified to limit and filter the items in that list.

3.1.9 MANO Framework / Infrastructure Abstraction

The Infrastructure Adapter exposes a range of APIs that are consumed by the MANO Framework. All these APIs are realised by using the RabbitMQ message bus. The MANO Framework is publishing on the following topics, which are consumed by the IA:
1. `infrastructure.function.deploy`: Instantiate a new VNF instance

2. `infrastructure.service.remove`: Terminate a running NS

3. `infrastructure.chain.configure`: Configure a chain between VNFs on the same PoP (Point of Presence)

4. `infrastructure.chain.deconfigure`: Deconfigure a chain between VNFs on the same PoP

5. `infrastructure.service.wan.configure`: Configure a WAN network between VNFs on different PoPs

6. `infrastructure.service.wan.deconfigure`: Deconfigure a WAN network between VNFs on different PoPs

7. `infrastructure.management.compute.list`: Request detailed information on available compute infrastructure

For a detailed description of the payload of each request and response, please check the Infrastructure Adapter GitHub wiki [50].

### 3.1.10 MANO Framework / Monitoring Manager

The interaction between the MANO framework and the Monitoring Manager happens during the instantiation of a new service and during the triggering process of an alert.

In the first occasion, MANO framework informs the Monitoring Manager that a new service has been successfully instantiated. This information is passed through the Monitoring Manager REST API, as described in [2]. In particular, there are two APIs related to this functionality, as described in Tbl. 3.1. The POST API is passing all information related to the service monitoring, such as VNFs that the NS consists of, specific metrics to be monitored, thresholds per metric as well as information on the VNFs comprising the NS. The DELETE API deletes the service monitoring, as dictated by the MANO framework through the information contained in the service instantiation record.

<table>
<thead>
<tr>
<th>Action</th>
<th>HTTP Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new NS instance monitoring</td>
<td>POST</td>
<td>/api/v1/service/new</td>
</tr>
<tr>
<td>Delete NS instance monitoring</td>
<td>DELETE</td>
<td>/api/v1/services/{srvid}</td>
</tr>
</tbody>
</table>

In the case that an alert is triggered due to the violation of a rule, as monitored in accordance to the method described above, the MANO Framework can be notified through a pub/sub message bus, where each entity, such as the MANO Framework can subscribe to. For the purposes of monitoring related subscriptions, MANO collects the respective information from the `son.monitoring` topic. Within the specific topic, interesting fields of information are provided, such as the id of the service that has been triggered, the uuid of the VM whose performance has been violated, the time that the event has occurred and the name of the alert. The alerts are pushed to the message bus through a component of the Monitoring Manager that is called Alert Manager which is responsible for sending notifications about firing alerts to the subscribed entities.
3.1.11 MANO Framework / Repositories

The MANO Framework interacts with the Repositories in order to store Network Service Records (NSR) and VNF Records (VNFR) on the repositories. In particular, the following are the possible actions:

- The MANO Framework `create` a record in the Repositories that should be stored
- The MANO Framework `update` an existing record that the Repositories should be adapted
- The MANO Framework `request` a record from the Repositories
- The MANO Framework `delete` a record from the Repositories

The Repositories expose a REST API to provide these actions in Sec. 2.3.

3.1.12 Policy Manager / Monitoring Manager

The interaction between the Policy Manager and the Monitoring Framework is depicted in the Fig. 3.1. Upon the instantiation of a network service and the enforcement of a policy, a set of monitoring metrics are collected and processed according to a set of expressions defined in the policies. The metrics and the corresponding expressions may regard VNF-specific metrics, NS-specific metrics or resources usage metrics. Processing of the collected data is realised within the Monitoring Framework, leading to the triggering of alerts that leads to the publishing to the 5GTANGO Message Broker at the topic `son.monitoring.policy`. Such alerts are consumed by the Policy Manager for realising inference over the defined set of rules. The suggested actions from the inference results are then published to the Message Broker at the topic with name `eu.tng.policy.runtime.actions` in order to be consumed by orchestration mechanisms.

3.1.13 SLA Manager / Catalogues

For the first 5GTANGO prototype, the interactions between the SLA Manager and the Catalogue are based on RESTful APIs, exposed by the Catalogue and consumed by the SLA Manager.

The currently existing interactions are the following:
Figure 3.2: Retrieval of NSD and PD and SLA Template storage on the Catalogue.

1. Retrieving NS descriptors (NSDs) from the catalogue;

2. Storing SLA Template descriptors (SLADs) in the catalogue;

3. Retrieving SLA Template descriptors from the catalogue;

4. Deleting SLA Template descriptors from the catalogue;

5. Updating SLA Template descriptors in the catalogue.

The SLA Manager retrieves from the Catalogue a NSD and a Policy Descriptor (PD) for a specific NS (given a uuid). The descriptors will be handled by the SLA Manager internally in order to generate the SLAD.

Once the SLAD is generated, the 5GTANGO Catalogue store function takes place, in order to store the SLAD in the database and make it available to other components (e.g. the Gatekeeper).

When VNF(s)/NS/packages or any descriptors need to be retrieved from the 5GTANGO catalogue, the retrieve function will fetch them from the catalogue.

When SLA Templates need to be retrieved from the 5GTANGO catalogue - either by the SLA Manager or other components - the retrieve function will fetch from the catalogue their descriptors.

When SLA Templates need to be deleted from the 5GTANGO catalogue, the delete function can be used. Because this template may be published to costumers, in order to avoid complications, there is a need for business logic from the SLA Manager. Therefore, there is a need to check if the SLAD is unpublished and inactive, before proceeding with the deletion.

To put an updated SLA Template into the 5GTANGO catalogue, the update function will be used. By selecting a template to be updated, a newer version of that item is stored, with the additional inputs.

As an example, Fig. 3.2 shows the interactions for the retrieval of a NSD and PD in order to store the generated SLA Template descriptor to the Catalogue.
3.2 SP External Interfaces

This section describes the interactions of the Service Platform with external entities and stakeholders.

3.2.1 SP / SDK

5GTANGO’s SDK [28] is a set of tools that supports a service developer to create, test, and package VNFs and complex NSs. In addition, the SDK provides tools to interact with the Service Platform (or V&V) and thus acts as a client of some APIs offered by the service platform. The main interactions between the SDK and the service platform are authentication, package on-boarding, package file retrieval, package search, package, service, function meta-data retrieval, record retrieval, service instantiation, and monitoring data retrieval. They have been described in D2.2 [20].

The rest of this section gives a high-level overview of the involved API endpoints. More details, data models, and API details can be found in the online documentation of the Gatekeeper modules [46, 17]. Many of these endpoints are compatible to the endpoints implemented by the last SONATA service platform releases [44, 45], for example, the package on-boarding endpoint. Even though the used package format has changed, the uploading mechanism has remained the same, which allows to re-use as much SONATA outcomes as possible and reduce development times.

Tbl. 3.2 lists all endpoints involved into the interactions between service platform and SDK. For sake of clarity, the table only shows the root endpoints. Each endpoint might provide additional sub-endpoints which which are fully documented online [46].

<table>
<thead>
<tr>
<th>Action</th>
<th>Method(s)</th>
<th>Endpoint</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication</td>
<td></td>
<td>/api/v2/users/*</td>
<td>user management</td>
</tr>
<tr>
<td>authentication</td>
<td></td>
<td>/api/v2/sessions/*</td>
<td>session management</td>
</tr>
<tr>
<td>packages</td>
<td>GET</td>
<td>/api/v2/packages/*</td>
<td>package on-boarding and management</td>
</tr>
<tr>
<td>services</td>
<td>GET</td>
<td>/api/v2/services/*</td>
<td>service information retrieval</td>
</tr>
<tr>
<td>functions</td>
<td>GET</td>
<td>/api/v2/functions/*</td>
<td>function information retrieval</td>
</tr>
<tr>
<td>records</td>
<td>GET</td>
<td>/api/v2/records/*</td>
<td>record information retrieval</td>
</tr>
<tr>
<td>instantiation (LCM)</td>
<td>GET</td>
<td>/api/v2/requests/*</td>
<td>LCM, e.g., instantiation</td>
</tr>
<tr>
<td>vim</td>
<td></td>
<td>/api/v2/vims/*</td>
<td>VIM management</td>
</tr>
<tr>
<td>wim</td>
<td></td>
<td>/api/v2/wims/*</td>
<td>WIM management</td>
</tr>
<tr>
<td>kpi</td>
<td></td>
<td>/api/v2/kpis/*</td>
<td>KPI management</td>
</tr>
<tr>
<td>license</td>
<td>GET</td>
<td>/api/v2/licenses/*</td>
<td>license information retrieval</td>
</tr>
</tbody>
</table>

3.2.2 SP / V&V

The V&V is a plug-able environment to the Service Platform. It will be able to work with multiple vendors of Service Platforms, as far as those vendors follow the specification of the Service Platform. For 5GTANGO project, we had integrated with SONATA’s Service Platform. V&V external API is documented in [27]. Fig. 3.3 depicts the relationships among the two components.

3.2.3 SP / VIMs/WIMs

The Service Platform (SP) interacts with VIMs/WIMs in order to request and manage infrastructure resources, both Datacenter resources (VIMs - Virtualized Infrastructure Manager) and WAN resources WIMs (WAN Infrastructure Manager). The Infrastructure Abstraction (IA), located in the lower layer architecture of 5GTANGO Service Platform, has a southbound interface that implements the APIs needed to communicate with the VIMs and WIMs.
In the case of VIMs, the IA generates HEAT templates to orchestrate the resources in OpenStack. Additionally, the IA uses neutron to handle the datacenter networking, as well as the keystone for authentication among other OpenStack APIs.

The 5GTANGO Service Platform can enforce the Service Function Chain (SFC) in the Network Service that is deployed, making use of an SFC agent installed in the OpenStack node. The IA has an interface to that SFC agent for that purpose.

Moreover, the IA also implements an interface to the WIM in case the Service is deployed across the network. This interface is in charge of enforcing the end-to-end service connectivity between PoPs. The WIM makes uses of OpenDaylight VTN (Virtual Tenant Network) to build that WAN connection.

### 3.2.4 SP / Portal

The interactions between the Portal and the SP are solely based on the RESTful APIs provided by the SP (Gatekeeper).

The currently existing or planned interactions (these will most probably evolve as development proceeds):

- **Packages**: manages packages (note: package on-boarding is mostly a developer’s task, uploading the package through the Service Platform’s API, see [28]; an exception will be the ability for the Service Platform Manager to be able to ‘pull’ packages from a given V&V platform), including package file download and package deletion;

- **Services**: lists services’s meta-data (note: services are on-boarded within a package and are therefore deleted only on package deletion); services can also be instantiated, updated and terminated;

- **Functions**: lists functions’s meta-data (note: as services, functions are on-boarded within a package and are therefore deleted only on package deletion)

- **Records**: lists services’ and functions’ records (note: records are created upon a successful service instantiation request; they’re not deleted, but may have to be updated);

- **Monitoring**: manages requests for monitoring data, both synchronous and asynchronous;

- **Slices**: manages slices, being able to assign services to slices, create slices, etc.;
Figure 3.4: Retrieve the list of NSs.

- **Policies/SLAs**: manages policies and SLAs, its rules and applicabilities;

- **VIMs/WIMs**: manages VIMs/WIMs on which resources will be reserved/allocated.

As an example, Fig. 3.4 shows the interactions for the representation of the list of services and packages on the GUI, by utilizing the appropriate Gatekeeper API methods.

### 3.2.5 SP / Service Provider

The Service Provider (or Operator) is usually the owner of the Service Platform and interacts with it mainly through two channels: Portal and APIs. The interaction with the Portal is performed by a human, the Service Provider Manager, which accesses to the GUI in order to perform a set of required actions like, deploy services, consult monitoring data, assign policies, or verify SLAs, among many others. The interaction with the APIs is not performed by humans, but by Service Provider systems, the so-called OSS (Operational Support Systems), which invoke the APIs. OSSs are usually systems that integrate APIs from multiple sub-systems (e.g. Service Platform, or traditional network elements), in order to manage end-to-end networks and services.

#### 3.2.5.1 Portal Interactions

The 5GTANGO Portal is a unified GUI used by different stakeholders/roles: Service Providers, Developers and Customers. It provides the appropriate views and functionality for each role, based on authentication and authorization mechanisms. Within the same role, multiple users/managers are also supported. The user and role management can also be done in the Portal by a global administrator. In the left menu, the Portal has a Service Platform tab, where the manager can have access to all SP features, namely:
• View a dashboard, with the most relevant information;

• Manage VNF/NS Packages (and Slice Templates in the future), on-boarding, updating and removing VNF and NS artifacts;

• Manage VNF/NS Lifecycle (and Slice Instances in the future), instantiating, scaling and terminating VNFs/NSs;

• Manage NS Policies (not fully implemented yet), assigning policies to particular Services, regarding both, placement and runtime;

• Manage NS SLAs (not fully implemented yet), associating SLAs to particular Services and Customers;

• View Monitoring data (not fully implemented yet), viewing and getting monitoring data;

3.2.5.2 API Interactions

The 5GTANGO APIs can be used by authorized entities to access to SP features. These parties can be internal, like the Portal, or external lie OSS. APIs use authentication and authorization mechanisms to ensure the appropriate access to the a full set of features, namely:

• Manage VNF/NS Packages, on-boarding, updating and removing VNF and NS packages;

• Manage Slice Templates, on-boarding, updating and removing NSTs;

• Manage VNF/NS Lifecycle, instantiating, scaling and terminating a VNF/NS;

• Manage Slice Instances Lifecycle, instantiating and terminating a Slice instance;

• Manage Catalogue, getting and setting new catalogue items (descriptors, templates);

• Manage Repository, getting and setting new repository records;

• Manage NS Policies, assigning policies to particular Services, both placement and runtime;

• Manage NS SLAs, associating SLAs to particular Services and Customers;

• Get Monitoring data, getting alarms, and aggregated data and raw data streams;

3.2.6 SP / Developer

NS and VNF developers can use the 5GTANGO Software Development Kit (SDK) for their interactions with the Service Platform. More details about the interaction of the SP with SDK can bee in Sec. 3.2.1. For a more detailed information about the SDK, please see D4.1 [28].

Furthermore, Developers can use the 5GTANGO Portal to perform the interaction with the V&V to validate the developed functions. The Portal will also allow to create and manage SLAs and policies associated to the new services and functions and instantiate them in order to test them in a production-like environment. The user used by the developer to login in the Portal must have the right permissions to perform such actions.
3.2.7 SP / Customer

The Portal is expected to be the main management element of the 5GTANGO platform for external customers. The customer will be able to manage and operate the whole lifecycle of functions and services. Namely, the main actions the customer will be able to carry out from the Portal are included below:

- A dashboard with the main information regarding customer’s functions and services status and relevant metrics will be showed in the first screen. The customer will be able to check the status of all the elements her permissions allow to access at a glance.
- Check and completely manage the existing functions and services through the Catalogue.
- Check and select the SLA to be applied to NSs and VNFs.
- Check, modify and enable/disable the runtime and placement policies to be applied to functions and services.
- Check and completely manage the existing NSTs on the Catalogue and order their instantiation and termination. It will also be possible to manage the list of NSs defined in the NST and whose instantiation will be triggered when the NST is deployed.
4 Source Code

Source code repositories supporting this version are listed below, by component.

- **Gatekeeper**:
  - `tng-api-gtw` [15]: including the **Security Gateway** and the **Router**;
  - `tng-gtk-common` [16]: holding those microservices that are shared between the V&V Platform and the Service Platform. i.e., **Packages, Services** and **Functions**;
  - `tng-gtk-sp` [17]: holding those microservices that are specific to the Service Platform, e.g., **Services Lifecycle Management**, **VIM/WIM Management**, etc.;

- **SLA Manager**:
  - `tng-sla-mgmt` [26]: including the **SLA Template Generator** mechanisms for the Templates Management;
  - `tng-schema` [18]: holding the SLA Template schema, among with some SLA Template examples;
  - `tng-cat` [21]: holding the generated SLA Templates and provide APIs in order to manage them (store, update and delete them accordingly);

- **Slice Manager**:
  - `tng-slice-mngr`: include the **Slice Manager** main functions;
  - `tng-schema`: holding the NST and NSI schemas, among with some NSI and NST examples;

- **Policy Manager**:
  - `tng-policy-mngr` [24]: include the **Runtime Policy Manager**;
  - `tng-schema` [18]: holding the runtime policy descriptor (RPD) schema, among with some policy examples for specific NS instances;

- **Catalogues**:
  - `tng-cat` [21]: include the **Catalogues**;

- **Repositories**
  - `tng-rep` [21]: include the **Repositories**;

- **MANO Framework**
  - `son-mano-framework` [47]: Include the different plugins of the **MANO Framework**;

- **Infrastructure Adaptor**
  - `son-sp-infrabstract` [50]: Includes both **VIM-adaptor** and **WIM-adaptor**;
• Monitoring
  – son-monitor [49]: including core components of monitoring framework;
  – son-monitor-probe [48]: including probes for generic and specific monitoring;
  – tng-monitor-infra [22]: provide infrastructure monitoring solution;

• Portal
  – tng-portal [25]: contains all the code which need to be served by a Web server to get an operative portal which need to connect to the platform services through the Gatekeeper.
5 Security Aspects

This section describes the security mechanisms used in the scope of the 5GTANGO SP. Security is a crucial aspect of any publically available API or web application. Adding security as an afterthought in whatever software system is costly, so we are building on top of SONATA’s SP, which already provided this. In the following sub-sections we summarise both the existing and the newly introduced security aspects of the 5GTANGO SP.

5.1 Infrastructure security

5GTANGO’s testbeds are accessible through VPN (see [29] for a more detailed analysis of this).

5.2 Portal security

The Portal will be accessed by three kinds of users:

1. Service Platform administrators: who need it to check the overall configuration and performance of the SP;

2. Developers: those developing services, functions and tests must be known to the SP, in order to be authorized to upload and download packages, verify available and re-usable services, functions or tests, etc.;

3. Customers: those users wanting to be able to buy (i.e., instantiating) services should register themselves as playing this role. Customers will be able to control their services’ (and only theirs) life cycle, being able to update or stop them.

5.2.1 Secure access from the user’s browsers

Access from the user’s browser should be done with security turned on, i.e., by using HTTPS (TLS). This is controlled by the SP’s Security Gateway of the API Gateway [15].

5.2.2 User authentication

Users will have to provide their username and password (as well as a valid email) upon registration. In the future, we plan to also use OAuth2.0 to authenticate users, not storing passwords at all and using the most common Social Networks for doing it.

5.3 APIs Security

External tools and systems can access the SP APIs by using valid tokens, or API keys. These tokens will be generated for users with the Developer role. Optionaly, developers may upload their public key, which can then be used to authenticate them.
5.4 Packages Security

Packages are signed and immutable to ensure integrity as described in D4.1 [28]. Moreover, packages can references external artifacts which may be stored in remote locations. The ensure the integrity of these referenced artifacts, a checksumming mechanism is used. More details about package format and features are described in the publicly available 5GTANGO package format specification [23].
6 Prototype Roadmap

This section describes the major features in the Service Platform roadmap, which will be available for the v1 and final version of the prototype.

6.1 ETSI-Compliant APIs

For the final version of the 5GTANGO prototype, it is planned the support of ETSI-compliant interfaces, in particular for the following reference points / interfaces. The Router and the Gatekeeper will be responsible to accommodate these changes by mapping the ETSI-compliant APIs to the currently in use internal APIs. These following interfaces will be adopted along the roadmap of the 5GTANGO SP:

- Ve-Vnfm Interfaces (VNFM-VNF/EM)
- Or-Vnfm Interfaces (NFVO-VNF)
- Ve-Vnfm Interfaces (VNFM-VNF/EM)
- Vi-Vnfm Interfaces (VNFM/VIM)

The last mentioned interface between the VNFM and the VIM will be particularly challenging to adopt. We’ll address it after we have a solid understanding of it.

In addition to the standards ETSI Interfaces, 5GTANGO has designed the Slice Manager (SM) interfaces towards the OSSs using a similar philosophy as ETSI NFV Os-Ma-nfvo reference point, with which has many similarities. Although this reference point is not yet standardized by any SDO (even though 5G Americas has provided some ideas in [12], the project believes this is a good approach to follow for consistency reasons.

- Slice Manager Interfaces (OSS/SM)

6.1.1 Os-Ma-nfvo Interfaces (OSS-NFVO)

For the Os-Ma-nfvo reference point, between the NFVO and the OSSs, the following interfaces will be supported (see [33] and [38] for more details):

- NSD Management interface
- NS Lifecycle Management interface
- NS Performance Management interface
- NS Fault Management interface
6.1.2 Or-Vnfm Interfaces (NFVO-VNFM)

For the Or-Vnfm reference point, between the NFVO and the VNFM, the following interfaces will be supported (see [31] and [37] for more details):

- VNF Package Management interface
- VNF Lifecycle Management interface
- VNF Performance Management interface
- VNF Fault Management interface
- VNF Indicator interface

6.1.3 Ve-Vnfm Interfaces (VNFM-VNF/EM)

For the Ve-Vnfm reference point, between the VNFM and the VNF/EM, the following interfaces will be supported (see [32] and [36] for more details):

- VNF Lifecycle Management interface
- VNF Performance Management interface
- VNF Fault Management interface
- VNF Indicator interface
- VNF Configuration interface

6.2 Slicing

For the final version of the 5GTANGO prototype the Slice Manager will support advanced features. In particular, the following NST lifecycle management operations will be supported:

- Enable / Disable of NSTs, enabling/disabling NSTs
- Update of NSTs, updating NST descriptors

The following NST lifecycle management operations will also be supported:

- NSI Scaling, scaling in/out NSIs
- NSI Update, updating running NSIs (configurations, resource migrations, etc.)
- NSI Healing, fixing problems occurred to running NSIs

The Slice Manager (SM) interfaces towards the OSSs are not yet standardized by any SDO. However, 5GTANGO will align them with ETSI NFVO API due to their functional similarities. They Slice Manager will expose the following interfaces:

- NST Management interface
- NSI Lifecycle Management interface
- NSI Performance Management interface
- NSI Fault Management interface
6.3 Runtime Policy

Towards the final implementation and release of the runtime policy manager, a set of extensions are going to take place regarding the support of the following features:

- CRUD operation with regards to policies stored in the Catalogue. At the current version, the policy manager plugin stores policy descriptors internally. However, in the final release, CRUD operations regarding policies descriptors are going to be realised through interaction with the Catalogue.

- Implementation of a Policies Editor for editing and updating policies descriptions. The outcome of the Policies Editor is the policies descriptor that is going to be validated by the Policy Manager and made available in the Catalogue.

- Support of inertia feature. The “inertia” feature is a rule property that indicates the period of time (minutes, hours, days) the policy rule stays inactive after being triggered.

- The set of supported policy enforcement actions will be further extended so as to generate more action types, according with the policy yaml schema.

- Permit the consumption of aggregated data streams, supported by the monitoring engine. Policy engine is now consuming only alerts from the monitoring engine. A specific limited set of aggregated data streams is going also to be consumed, supporting inference based on such data.

- Request the collection of set of monitoring data based on the set of rules defined in the policies; The activation of policies enforcement mechanisms implies the activation of set of monitoring probes upon request.

6.4 SLA

For the final version of the 5GTANGO prototype the SLA Manager is going to support the whole life-cycle of service level agreements. The following mechanisms will be adopted along the roadmap of the 5GTANGO Service Platform:

- SLA Mapping Mechanism: mapping between the high-level requirements described by the end-user and the low-level requirements described by the service provider.

- SLA Parameter Analyzer: decide whether the process of the mapping mechanism should be done or not, by checking previous historical data.

- SLA Monitor Analyzer: Compare the QoS parameters from the SLA Repository, with the computed monitoring measurements and check if there is any violation. This mechanism also introduces the continuous optimization of the output of the Mapping Mechanism in order to produce better results, and as a result better provisioning of quality parameters that need to be added in the SLA Agreements.

A very ambitious part of the SLA Manager component refers to the SLA Mapping Mechanism. In order to create efficient SLA Agreements, a one-shot negotiation process will be considered, through this mechanism. The purpose, is to map high-level requirements expressed by the costumers, to low-level recourse parameters, taken by the SLA Template in form of policy rules and historical
monitoring data. The mechanism decomposes the service level objectives to associated policies for better QoS enforcement. More specifically, the Mapping mechanism obtains a set of policies from the Policy Manager, a set of low-level requirements from the service provider and a set of high-level requirements from the costumers, based on the NS being selected. The data-sets are used to compose, aggregate, or convert the low-level metrics to high-level SLA parameters. As a result, the outputs of the mapping mechanism are explicit SLA parameters and metrics. In addition, monitoring data are going to be used and adopted by the SLA Manager, towards better quality assurance.

The Mapping Mechanism will be based on unsupervised learning, using an Artificial Neural Network (ANN). ANNs can be applied to solve the translation problem, through directly mapping the service specific SLOs, to resource attributes. ANNs are good candidates for mapping, as the influence of high-level network service requirements (i.e. workload parameters from the end-user), and the changes in low-level requirements are reflecting at the changes in the ANN output.

### 6.5 MANO Framework

For the final version of the 5GTANGO Service Platform, the MANO Framework will be further evolved to support following functionality:

- **Policy based placement:** The current policy based placement functionality solves the embedding problem as an Integer Linear Problem (ILP). Once the substrate or NSs grow larger, solving this ILP might become unfeasible due to time constraints. Therefore, the functionality will be evolved to solve the embedding problem with time-efficient heuristics.

- **Policy based re-optimisation:** Although newly instantiated VNFs are optimally placed on the substrate with respect to developer, customer and operator policies, the addition of new VNFs or termination of existing VNFs might allow for an even better placement at a later stage in the VNF life cycle. Therefore, we will extend the MANO Framework to calculate optimal embeddings for all deployed VNFs at various moments in their life cycle, and compare the gain that can be made from migrating them to a different location with respect to the migration cost.

- **Migration:** The MANO Framework will extend its functionality to support both stateless and stateful VNF migrations with minimal service interruption.

- **Scaling:** The MANO Framework will support scaling out and scaling in.

- **Self healing:** The MANO Framework should extend its functionality so that it can use the various detection mechanisms (Monitoring Framework, Policy Manager and FSMs) to diagnose VNF or NS corruption and repair the issue.

### 6.6 Monitoring

The final version of the 5GTANGO monitoring manager will develop new as well as adapt and improve existing mechanisms along the roadmap of the 5GTANGO Service Platform:

- **Support of active network monitoring:** Support active monitoring, integrated with the framework implemented so far, along with the proper collection and processing of data in accordance with the use cases requirements to offer VNF level and NS level monitoring.
• **Support requirements from 5GTANGO components:** As part of the use case requirements, one could consider support for internal 5GTANGO components, such as SLA Manager and Policy Manager. Such requirements will include data availability for supporting policy decisions, the ability to frequently modify rules and thresholds during runtime.

• **Automatic instantiation of developer-specific metrics:** 5GTANGO will extend SONATA implementation by developing the mechanism to easily include developer-specific metrics in order to insert new metrics to be monitored.

• **Follow ETSI-based NSI PM specifications:** The aim is to provide a framework as close as possible to the specifications provided by ETSI, including collaboration with slice manager to address specifications related to Performance Monitoring.

• **Default VNF metrics:** Following the experience of ETSI plugfest, we will provide solution for collecting monitoring data from VNF and NS without the need for probes within NS images, based on the integration of infrastructure monitoring solutions already available.

### 6.7 Catalogues

The final version of the 5GTANGO catalogues will improve and adopt the following mechanisms along the roadmap of the 5GTANGO Service Platform:

- The extended availability of the Catalogues for complicated queries.
- The efficient search and mapping of different types of metadata and descriptors.
- The Decision Support mechanism, enabling the filtering, prioritizing and delivering information to the user.
- The Continuous Optimisation mechanism, introducing the amendment in the V&V process by implementing a feedback loop which monitors the generated data from the actual deployed information and the test results.

The 5GTANGO Catalogues will introduce an enhanced multi-sided data storage. Not only, are the Catalogues to support an active role in the overall infrastructure with the Decision Support and Continuous Optimisation mechanisms but also it will enable the efficient exploration of its hosted descriptors with its searching utilities.

### 6.8 Repositories

The final version of the repositories will support the storage of VNFs, NSs, and NSIs records, as well as all the related information, like SLA or Policy. In particular, the following will be added:

- Slices Instances records
- Policies Instances records
- SLA Instances Mapping
- Smart deployment for Service Platform or VnV
6.9 Portal

In the final version of the Portal it will be possible to manage all the elements developed within the scope of 5GTango project in an visual, simple and intuitive way. The WUI will be designed with User Experience as a main priority and it will be validated in an interactive process during the development phase.

All the sections will be fully operative in the final version of the Portal:

- The **Dashboard screen** will contain customizable panels which give relevant status information of the platform.
- The **Users section** will allow a complete management of users.
- The **Validation and Verification section** will enable the management of all the features supported by the V&V server. It will also be possible to check the result of validation tests.
- The **Service platform section** will enable the creation, modification, control and deletion of NSs and VNFs.
- **Service management section**: will allow the management of user and service licensing. A list with all the available NS and instantiation and management of NS will be also possible from this section.
7 Conclusions

This Deliverable D5.1 provided a detailed description of the 5GTANGO Service Platform, approaching both the design and the implementation details.

The 5GTANGO Service Platform is an evolution of the SONATA Service Platform, extending some existing features and creating new others. The 5GTANGO Service Platform is composed by multiple internal components interfacing with each others, following a microservices approach. Some of these components come from the SONATA project [1], while some others were created from scratch. In particular, the SONATA inherited components are the Gatekeeper, the Catalogues, the Repositories, the MANO Framework, the Infrastructure Abstraction, the VIMs/WIMs and the Monitoring. In this first version of 5GTANGO, we introduce four new components: the Slice Manager, the Policy Manager, the SLA Manager and the Portal.

Many of the SONATA inherited components, like the Catalogues or the MANO Framework, have been extended in order to support new features. Some others, like the Gatekeeper, needed a refactoring work to accommodate the new requirements, namely due to the cohabitation with the V&V framework.

The 5GTANGO new components are the result of the emerging features that the project has proposed to tackle. The ability to create multiple virtual networks on top of a common cloud infrastructure is particularly important on 5G networks, so Slicing is one of these features, being the Slice Manager the component in charge of it. Today, it is also of paramount importance for the customer to guarantee the contracted service quality so the definition of Service Level Agreements is another hot feature we wanted to consider; this is handled by the SLA Manager. The Policy Manager is the component that makes possible the policy-driven operation, managing the service lifecycle according to a certain rules. Finally, the Portal is a new component that intends to unify the existing two graphical interfaces on SONATA, one devoted for Customers and the other for Operators.

As this is the first version of the 5GTANGO Service Platform, at this stage, most of the components are still in early stages of development, and even the design may be suitable to changes in the next months. This is actually something usual when agile development methodologies are in used on software development processes. The description of the final version of the Service Platform prototype will be provided in Deliverable D5.2 (M21, May 2019). In May, the prototype developments are expected to be concluded and the integration work with verticals will start.
A Appendices

A.1 Policy Descriptor Example

The following runtime policy descriptor instance describes a set rules applied upon deployment of a NS with name “default-nsd”, created by the “tango” vendor and having a version of “0.9”. The described rules are the following two:

1. A rule with name ‘actionUponAlert’ and high priority (salience 1). If policy manager consumes an LogMertic object from son-broker related with vnf1 and having a value of ‘mon_rule_vm_cpu_perc’ then the ‘actionUponAlert’ will be triggered leading to an InfrastructureType Action that applies the flavour 3 at the vnf1.

2. Similarly the rule with name ‘highTranscodingRateRule’ checks whether the average values of the vnf1 CPUload and the vnf2 RAM are respectively more than 70 and less than 8 during a 10 minutes sliding window. If so, an InfrastructureType Action is generated that applies the flavour 3 at the vnf1.


name: "samplepolicydemo"
network_service:
  vendor: "tango"
  name: "default-nsd"
  version: "0.9"
policyRules:
  - name: "actionUponAlert"
    salience: 1
    inertia:
      value: 30
      duration_unit: "m"
    conditions:
      condition: AND
      rules:
        - id: vnf1.LogMetric
          field: vnf1.LogMetric
          type: string
          input: text
          operator: equal
          value: "mon_rule_vm_cpu_perc"
    actions:
      - action_object: "ComponentResourceAllocationAction"
A.2 Drools Rules Example

The above runtime policy descriptor instance is translated at the following drool rules that consist of the production memory of the Policy manager inference engine.

```drools
rule "actionUponAlert"
when
    LogMetric(componentid== "vnf1" && value== "mon_rule_vm_cpu_perc") \n    from entry-point "MonitoringStream"
then
    insertLogical( new ComponentResourceAllocationAction("samplepolicydemognsid","vnf1",\n                 InfrastructureType.ApplyFlavour,"3"));
```

```drools
action_type: "InfrastructureType"
name: "ApplyFlavour"
value: "3"
target: "vnf1"
- name: "highTranscodingRateRule"
salience: 1
inertia:
    value: 30
duration_unit: "m"
duration:
    value: 10
duration_unit: "m"
aggregation: "avg"
conditions:
    condition: AND
rules:
    - id: vnf1.CPULoad
      field: vnf1.CPULoad
type: double
input: number
operator: greater
value: '70'
    - id: vnf2.RAM
      field: vnf2.RAM
type: integer
input: select
operator: less
value: '8'
actions:
    - action_object: "ComponentResourceAllocationAction"
      action_type: "InfrastructureType"
      name: "ApplyFlavour"
      value: "3"
      target: "vnf1"
```

```drools
rule "actionUponAlert"
when
    LogMetric(componentid== "vnf1" && value== "mon_rule_vm_cpu_perc") \n    from entry-point "MonitoringStream"
then
    insertLogical( new ComponentResourceAllocationAction("samplepolicydemognsid","vnf1",\n                 InfrastructureType.ApplyFlavour,"3"));
```
end

rule "highTranscodingRateRule"
when
  (  
      $tot0 := java.lang.Double( $tot0 >70.0 ) from accumulate(  
          $m0 := MonitoredComponent( name== "vnf1" && metric== "CPULoad" ) \  
              over window:time(70s)from entry-point "MonitoringStream",  
              average( $m0.getValue() ) ) and  
      $tot1 := java.lang.Double( $tot1 <8 ) from accumulate(  
          $m1 := MonitoredComponent( name== "vnf2" && metric== "RAM" ) over \  
              window:time(70s) from entry-point "MonitoringStream",  
              average( $m1.getValue() ) )  
  )
then
insertLogical( new ComponentResourceAllocationAction("samplepolicydemognsid","vnf1",  
      InfrastructureType.ApplyFlavour,"3"));
end

A.3 NST Example

A NST refers to the descriptor that details the slice template, which includes references to the  
underlying NSs to be deployed for NSI, and the relationship among them.

s_nssai:
    sst: 1 # eMBB
    sd: 1

slice-descriptor/nst-schema.yml"

vendor: "eu.5gtango.slice-descriptor"
name: "slice"
version: "0.1"
author: "Ricard Vilalta, CTTC"
description: >  
    "Example Slice."

# The NSs reused by this service (recursively)

network_services:
  - ns_id: "ns_sonata"
    ns_vendor: "eu.5gtango.slice-descriptor"
    ns_name: "5gtango-demo"
    ns_version: "0.1"

##
## The Slice service interface points to the 
## outside world.
## service_interface_points:
- id: "mgmt"
  interface: "ipv4"
  type: "management"
- id: "input"
  interface: "ipv4"
  type: "external"
- id: "output"
  interface: "ipv4"
  type: "external"

## The Slice 5QI

### qos:
- 5qi_value: "6"

## A.4 NSI Record Example

A NSI Record refers to the descriptor that includes the information of an instance of a NSI, which will use a NST reference for its deployment. The NSI will include references to the deployed NS Records (NSR).

```plaintext
id: "00000000-0000-0000-0000-000000000000"

s_nssai:
  sst: 1 # eMBB
  sd: 1

descriptor_version: "0.9"

vendor: "eu.5gtango.slice-descriptor"
name: "slice"
version: "0.1"
author: "Ricard Vilalta, CTTC"
description: >
  "Example Slice."

# The NSs reused by this service (recursively)

network_services:
- ns_id: "ns_sonata"
  ns_vendor: "eu.5gtango.slice-descriptor"
  ns_name: "5gtango-demo"
  ns_version: "0.1"
  nsr_id: "00000000-0000-0000-0000-000000000000"
```
The Slice 5QI

A.5 SLA Template Descriptor Example

The SLA template definition process aims at generating and refining the SLA templates. The service providers analyze their business objectives through a business modelling process in order to optimize their offerings to the customers.

The following SLA Template example, states a service commitment of a monthly availability percentage of at least 95% for the selected NS. This percentage is calculated through an expression as the monthly up-time of the NS, divided with the total time.


vendor: "tango-sla-template"
name: "example-sla-template"
version: "0.1"
author: "Evgenia Kapassa, Marios Touloupou, UPRC"
description: "A default SLA Template with initial SLOs that are expressed through\metrics,parameters and an expression"

sla_template:

# general details regarding the SLA Template
offered_date: '2018-01-24 10:00:00'
valid_until: '2019-01-24 10:00:00'
service_provider_id: sp001

# involved NS
ns:
  uuid: ns_uuid_01
description: a NS description

# high level SLOs
objectives:
  - slo_id: req001
    slo_name: NetworkServiceAvailability
    slo_definition: Indicator that the NS is up and running.
slo_unit: "/\%
slo_value: '95'

# corresponds to how to measure the objective
metric:
- metric_id: mtr001
  metric_definition: Total time the NS is alive

  rate:
  parameterWindow: 1 month
  sampleInterval: sample rate every 1 hour

  # a function under which the specific metric of the SLA should obey
  expression:
  expression_statement: prmtr001/prmtr002
  expression_language: ISO80000
  expression_unit: hour

  # expressing in detail each parameter inside the expression
  parameters:
  - parameter_id: prmtr001
    parameter_name: NetworkServiceUptime
    parameter_definition: NS Uptime refers to the amount of time that the NS has been powered on and working properly.
    parameter_unit: hour
    parameter_value: '700'
  - parameter_id: prmtr002
    parameter_name: Total_Time
    metric_definition: Total Time refers to one 1 month of the NS deployment
    parameter_unit: hour
    parameter_value: '720'
B Bibliography

[34] NFVISG ETSI. Nvf-eve 012 report on network slicing support with etsi nfv architecture framework, 2017.


