Programmable edge-to-cloud virtualization fabric for the 5G Media industry

D4.1 - 5G-MEDIA Catalogue APIs and Network Apps

Work Package: WP4 - 5G-MEDIA Open Repository of Network Apps
Lead partner: UPM

Author(s): Javier Serrano, David Jiménez, Federico Álvarez [UPM], Francesco Iadanza, Daniele Pavia [ENG], Francesca Moscatelli, Gino Carrozzo, Elian Kraja [NXW], Madeleine Keltsch, Gordana Macher, Sebastian Prokesch [IRT], Igor Fritzsch [BIT], Rocío Ortiz, Alberto Florez [TID], Panagiotis Athanasoulis, Stamatia Rizou [SILO], Petros Drakoulis, Panagiotis Dimou, Stamatis Samaras, Nikolaos Zioulis, Alexandros Doumanoglou, Dimitros Zarpalas [CERTH]

Delivery date (DoA): 31/8/2018
Actual delivery date: 31/8/2018
Dissemination level: Public
Version number: 1.0
Status: Final

Grant Agreement N°: 761699
Project Acronym: 5G-MEDIA
Project Title: Programmable edge-to-cloud virtualization fabric for the 5G Media industry
Instrument: IA
Call identifier: H2020-ICT-2016-2
Topic: ICT-08-2017, 5G PPP Convergent Technologies, Strand 2: Flexible network applications
Start date of the project: June 1st, 2017
Duration: 30 months

Project co-funded by the European Commission under the Horizon 2020 Programme.
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Who</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>June, 20(^{th}) 2018</td>
<td>UPM</td>
<td>Template creation and assignment</td>
</tr>
<tr>
<td>0.2</td>
<td>June, 26(^{th}) 2018</td>
<td>UPM, NXW, ENG</td>
<td>Initial contributions and list of applications</td>
</tr>
<tr>
<td>0.3</td>
<td>July, 2(^{nd}) 2018</td>
<td>NXW, ENG, IRT, CERTH, TID</td>
<td>Security framework, interfaces and applications contributions</td>
</tr>
<tr>
<td>0.4</td>
<td>July, 19(^{th}) 2018</td>
<td>UPM, NXW</td>
<td>Catalogue architecture contributions</td>
</tr>
<tr>
<td>0.5</td>
<td>July, 26(^{th}) 2018</td>
<td>IRT, BIT</td>
<td>Media applications contributions</td>
</tr>
<tr>
<td>0.6</td>
<td>August, 1(^{st}) 2018</td>
<td>UPM, NXW, ENG</td>
<td>Information model and security framework consolidation</td>
</tr>
<tr>
<td>0.7</td>
<td>August, 13(^{th}) 2018</td>
<td>UPM, NXW</td>
<td>Document consolidation</td>
</tr>
<tr>
<td>0.8</td>
<td>August, 21(^{st}) 2018</td>
<td>UPM</td>
<td>Comments and reviews incorporated</td>
</tr>
<tr>
<td>0.9</td>
<td>August, 27(^{th}) 2018</td>
<td>UPM, NXW</td>
<td>Implementation contributions and document consolidation</td>
</tr>
<tr>
<td>0.10</td>
<td>August, 30(^{th}) 2018</td>
<td>UPM, NXW, ENG, IRT, CERTH</td>
<td>Final restructuration, comments and reviews incorporated</td>
</tr>
<tr>
<td>0.11</td>
<td>August, 31(^{st}) 2018</td>
<td>UPM</td>
<td>Final document consolidation</td>
</tr>
<tr>
<td>1.0</td>
<td>August, 31(^{st}) 2018</td>
<td>UPM</td>
<td>Finalized document for submission</td>
</tr>
</tbody>
</table>
## Quality Control

<table>
<thead>
<tr>
<th>Role</th>
<th>Date</th>
<th>Who</th>
<th>Approved/Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer</td>
<td>August, 30th</td>
<td>Ugur Acar [NETAS]</td>
<td>Approved</td>
</tr>
<tr>
<td></td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interim Reviewer</td>
<td>August, 30th</td>
<td>Gordana Macher [IRT]</td>
<td>Approved</td>
</tr>
<tr>
<td></td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interim Reviewer</td>
<td>August, 30th</td>
<td>Francesca Moscatelli [NXW]</td>
<td>Comments and suggestions for improvement</td>
</tr>
<tr>
<td></td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 17th</td>
<td>Ugur Acar, Fatih Ustok, Selçuk Keskin [NETAS]</td>
<td>Comments and suggestions for improvement</td>
</tr>
<tr>
<td></td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 14th</td>
<td>Igor Fritsch [BIT]</td>
<td>Comments and suggestions for improvement</td>
</tr>
<tr>
<td></td>
<td>2018</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Disclaimer

This document may contain material that is copyright of certain 5G-MEDIA project beneficiaries, and may not be reproduced or copied without permission. The commercial use of any information contained in this document may require a license from the proprietor of that information. The 5G-MEDIA project is part of the European Community's Horizon 2020 Program for research and development and is as such funded by the European Commission. 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 with respect to this document, which is merely representing the authors’ view.

The 5G-MEDIA Consortium is the following:

<table>
<thead>
<tr>
<th>Participant number</th>
<th>Participant organisation name</th>
<th>Short name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>ENGINEERING – INGEGNERIA INFORMATICA SPA</td>
<td>ENG</td>
<td>Italy</td>
</tr>
<tr>
<td>02</td>
<td>IBM ISRAEL - SCIENCE AND TECHNOLOGY LTD</td>
<td>IBM</td>
<td>Israel</td>
</tr>
<tr>
<td>03</td>
<td>SINGULARLOGIC ANONYMI ETAIREIA PLIOFORIAKON SYSTIMATON KAI EFARMOGON PLIOFORIKIS</td>
<td>SILO</td>
<td>Greece</td>
</tr>
<tr>
<td>04</td>
<td>HELLENIC TELECOMMUNICATIONS ORGANIZATION S.A. - OTE AE (ORGANISMS TILEPIKOINION TIS ELLADOS OTE AE)</td>
<td>OTE</td>
<td>Greece</td>
</tr>
<tr>
<td>05</td>
<td>CORPORACION DE RADIO Y TELEVISION ESPANOLA SA</td>
<td>RTVE</td>
<td>Spain</td>
</tr>
<tr>
<td>06</td>
<td>UNIVERSITY COLLEGE LONDON</td>
<td>UCL</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>07</td>
<td>TELEFONICA INVESTIGACION Y DESARROLLO SA</td>
<td>TID</td>
<td>Spain</td>
</tr>
<tr>
<td>08</td>
<td>UNIVERSIDAD POLITECNICA DE MADRID</td>
<td>UPM</td>
<td>Spain</td>
</tr>
<tr>
<td>09</td>
<td>INSTITUT FUER RUNDFUNKTECHNIK GMB</td>
<td>IRT</td>
<td>Germany</td>
</tr>
<tr>
<td>10</td>
<td>NEXTWORKS</td>
<td>NXW</td>
<td>Italy</td>
</tr>
<tr>
<td>11</td>
<td>ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS</td>
<td>CERTH</td>
<td>Greece</td>
</tr>
<tr>
<td>12</td>
<td>NETAS TELEKOMUNIKASYON ANONIM SIRKETI</td>
<td>NET</td>
<td>Turkey</td>
</tr>
<tr>
<td>13</td>
<td>INTERINNOV SAS</td>
<td>IINV</td>
<td>France</td>
</tr>
<tr>
<td>14</td>
<td>BITTUBES GMB</td>
<td>BIT</td>
<td>Germany</td>
</tr>
<tr>
<td>15</td>
<td>NATIONAL CENTER FOR SCIENTIFIC RESEARCH - DEMOKRITOS</td>
<td>NCSRD</td>
<td>Greece</td>
</tr>
</tbody>
</table>
# Table of Contents

Executive summary...........................................................................................................................................11

1. Introduction ....................................................................................................................................................12
   1.1. Scope of the Deliverable..............................................................................................................................12

2. Overall 5G-MEDIA Architecture..................................................................................................................13

3. Information model specifications ....................................................................................................................16
   3.1. Specification for generalized Network Service and VNF Descriptors based on TOSCA..................16
   3.2. TOSCA CSAR Packaging Format.............................................................................................................21

4. 5G App and Service Catalogue design ..........................................................................................................27
   4.2. 5G App and Service Catalogue architecture............................................................................................28
      4.2.2. MANO Plugin......................................................................................................................................32
      4.2.3. SDN Plugin..........................................................................................................................................33

5. Security Framework .......................................................................................................................................35
   5.2. AAA Service Requirements ......................................................................................................................37
   5.3. ATN/ATZ User requirements for the Catalogue.......................................................................................38
      5.3.2. User Roles definition............................................................................................................................39
      5.3.3. User Groups definition..........................................................................................................................39
      5.3.4. AAA User Requirements....................................................................................................................39
   5.4. AAA Service Architecture .......................................................................................................................40
      5.4.2. Policy Engine....................................................................................................................................44
      5.4.3. Resource Registry................................................................................................................................47
   5.5. ATN/ATZ MANO Integration......................................................................................................................49
      5.5.2. Plain Credentials..................................................................................................................................49
      5.5.3. Bearer Token.......................................................................................................................................49
      5.5.4. Centralized ATN/ATZ Management....................................................................................................49
      5.5.5. Delegated ATN/ATZ Management.......................................................................................................50
      5.5.6. OSM R3 Integration............................................................................................................................50
      5.5.7. OSM R4 Integration............................................................................................................................52
   5.6. Accounting ..............................................................................................................................................52
      5.6.2. Events and metrics collector for the Accounting service.................................................................54

6. Interfaces specification ...................................................................................................................................58
   6.2. 5G App and Service Catalogue NBI..........................................................................................................58
   6.3. 5G App and Service Catalogue Admin interface.......................................................................................62
      6.3.2. AAA Admin UI....................................................................................................................................62
   6.4. 5G App and Service Catalogue SBI..........................................................................................................63
      6.4.2. Notification service...............................................................................................................................63
6.4.3. Open Source MANO Plugin ................................................................. 64

7. Media and Network functions repository ............................................. 68
   7.2. Generic Network Functions.............................................................. 68
       7.2.2. Generic VNFs and PNFs descriptions ...................................... 69
   7.3. Media Network Functions............................................................... 73
       7.3.2. Media VNFs and PNFs descriptions ....................................... 76

Conclusions ................................................................................................. 94

References .................................................................................................. 95

Annex I – ETSI NFV TST008 v.2.5.1 monitoring data YAML ...................... 96

Annex II .................................................................................................... 101

Annex III – ETSI NFV SOL005 v.2.4.1 NSD Management Interface YAML .... 102
List of Figures

Figure 1 – Architecture of the 5G-MEDIA platform .........................................................14
Figure 2 – NSD high level design from ETSI NFV IFA014 v2.4.1 [1] .................................17
Figure 3 – VNFD high level design from ETSI GS IFA011 v2.4.1 [2] .................................18
Figure 4 – DescriptorTemplate class diagram implemented in the 5G App and Service
Catalogue ..........................................................................................................................19
Figure 5 – TopologyTemplate class diagram with three node types, NS, Sap and NsVirtualLink .........................................................................................................................20
Figure 6 – CSAR v1.1 with TOSCA.meta ........................................................................26
Figure 7 – CSAR v1.1 with root YAML definitions file ......................................................26
Figure 8 – 5G App and Service Catalogue functional high-level design ............................29
Figure 9 – 5G App and Service Catalogue simplified software architecture ....................31
Figure 10 – NSD/VNF Package on-boarding sequence diagram ..................................32
Figure 11 – Interactions between the AAA Services, the Catalogue and the external
resources ..........................................................................................................................35
Figure 12 – AAA Service Architecture ............................................................................41
Figure 13 – AAA Service Interfaces ................................................................................42
Figure 14 – Sequence diagram of the developer using the SDK to login on the Catalogue
using AAA ........................................................................................................................45
Figure 15 – AAA class diagram for authentication and authorization ............................48
Figure 16 – AAA class diagram for the accounting .........................................................53
Figure 17 – Sequence diagram with the Monitoring pushing data to the Accounting
service ...............................................................................................................................54
Figure 18 – Metrics & events collection for Accounting ...................................................55
Figure 19 – AAA users management .............................................................................62
Figure 20 – AAA user permissions ...............................................................................62
Figure 21 – AAA user audit ...........................................................................................63
Figure 22 – 5G App and Service Catalogue MANO Plugin class diagram ....................65
Figure 23 – 5G App and Service Catalogue OSM Plugin class diagram .........................66
Figure 24 – Image Import API payload example, web-download case .............................67
Figure 25 – simple audio video workflow ....................................................................79
Figure 26 – Speech-to-Text Engine: Design approach ....................................................83
Figure 27 – Relative objective comparisons between the codecs for a given geometry
distortion level (3D Hausdorff distance). Performance is compared for the resulting
bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to
produce corresponding Hausdorff) (top right), decoding and encoding time (bottom
left and bottom right respectively). This analysis is done for the geometry only coding
case. .................................................................................................................................89
Figure 28 – Relative objective comparisons between the codecs for a given geometry
distortion level (3D Hausdorff distance). Performance is compared for the resulting
bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to
produce corresponding Hausdorff) (top right), decoding and encoding time (bottom
left and bottom right respectively). This analysis is done for the case of coding the geometry and the normals.

Figure 29 – Relative objective comparisons between the codecs for a given geometry distortion level (3D Hausdorff distance). Performance is compared for the resulting bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to produce corresponding Hausdorff) (top right), decoding and encoding time (bottom left and bottom right respectively). This analysis is done for the case of coding the geometry and custom attributes.

Figure 30 – Tandberg/Ericsson Video Processor.

Figure 31 – Nevion Virtuoso software-defined media node.

**List of Tables**

Table 1 – Example of TOSTA metadata .................................................................22
Table 2 – Example of CSAR YAML template .........................................................22
Table 3 – NFV-SOL004 options for Package Integrity ............................................24
Table 4 – Example of CSAR manifest ..................................................................24
Table 5 – 5G App and Service Catalogue software requirements ............................30
Table 6 – AAA service requirements ..................................................................37
Table 7 – User requirements ..............................................................................39
Table 8 – ATN Service API ..................................................................................44
Table 9 – Example of AAA policies for the Catalogue API ......................................47
Table 10 – NSD Management Interface ...............................................................58
Table 11 – VNF Package Management Interface ..................................................59
Table 12 – App Package Management Interface ....................................................61
Table 13 – SOL005 Notification Message types and corresponding topics ..............63
Table 14 – Nsd Notification Consumer Interface operations ....................................64
Table 15 – VNF Package Notification Consumer Interface operations ....................64
Table 16 – Generic Network Apps technical requirements .......................................68
Table 17 – vIDS REST API .................................................................................71
Table 18 – vDNS REST API ..................................................................................73
Table 19 – Media specific apps technical requirements ..........................................73
Table 20 – vCache REST APIs .............................................................................77
Table 21 – Example SDP File ...............................................................................79
Table 22 – Example Console Input/FFmpeg Call .....................................................80
Table 23 – MPE processing core control commands ...............................................81
Table 24 – Speech-to-text API ..............................................................................85
Table 25 – Networking simulation results for a TI end-to-end pipeline assessing the performance of different 3D codecs .........................................................91
<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Authentication, Authorization and Accounting</td>
</tr>
<tr>
<td>ACC</td>
<td>Accounting</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATN</td>
<td>Authentication</td>
</tr>
<tr>
<td>ATS</td>
<td>Apache Traffic Server</td>
</tr>
<tr>
<td>ATZ</td>
<td>Authorization</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network</td>
</tr>
<tr>
<td>CNO</td>
<td>Cognitive Network Optimization</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update and Delete</td>
</tr>
<tr>
<td>CSAR</td>
<td>Cloud Service ARchive</td>
</tr>
<tr>
<td>DevOps</td>
<td>Development and Operations</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>ExtCPs</td>
<td>External Connection Points</td>
</tr>
<tr>
<td>FaaS</td>
<td>Function as a Service</td>
</tr>
<tr>
<td>IDP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>JWT</td>
<td>JSON Web Token</td>
</tr>
<tr>
<td>MAPE</td>
<td>Monitor, Analysis, Policy, Execution</td>
</tr>
<tr>
<td>MANO</td>
<td>Management and Organization</td>
</tr>
<tr>
<td>MEC</td>
<td>Multi-access Edge Computing</td>
</tr>
<tr>
<td>NBI</td>
<td>North-Bound Interface</td>
</tr>
<tr>
<td>NF</td>
<td>Network Function</td>
</tr>
<tr>
<td>NFP</td>
<td>Network Forwarding Path</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Function Virtualization</td>
</tr>
<tr>
<td>NFVI</td>
<td>Network Function Virtualization Infrastructure</td>
</tr>
<tr>
<td>NFVO</td>
<td>Network Function Virtualization Orchestrator</td>
</tr>
<tr>
<td>NS</td>
<td>Network Service</td>
</tr>
<tr>
<td>NSD</td>
<td>Network Service Descriptor</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>OSM</td>
<td>Open Source MANO</td>
</tr>
<tr>
<td>OSS</td>
<td>Operation Support System</td>
</tr>
<tr>
<td>PAM</td>
<td>Pluggable Authentication Module</td>
</tr>
<tr>
<td>PNF</td>
<td>Physical Network Function</td>
</tr>
<tr>
<td>PNFD</td>
<td>Physical Network Function Descriptor</td>
</tr>
<tr>
<td>PoP</td>
<td>Point of Presence</td>
</tr>
<tr>
<td>QoE</td>
<td>Quality of Experience</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RO</td>
<td>Resources Orchestrator</td>
</tr>
<tr>
<td>SBI</td>
<td>South-Bound Interface</td>
</tr>
<tr>
<td>SDK</td>
<td>Service Development Kit</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Networking</td>
</tr>
<tr>
<td>SDNC</td>
<td>SDN Controller</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreements</td>
</tr>
<tr>
<td>SVP</td>
<td>Service Virtualization Platform</td>
</tr>
<tr>
<td>TOSCA</td>
<td>Topology and Orchestration Specification for Cloud Applications</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>VCA</td>
<td>VNF Configuration and Abstraction</td>
</tr>
<tr>
<td>VDU</td>
<td>Virtual Deployment Unit</td>
</tr>
<tr>
<td>VIM</td>
<td>Virtualized Infrastructure Manager</td>
</tr>
<tr>
<td>VL</td>
<td>Virtual Link</td>
</tr>
<tr>
<td>VLD</td>
<td>Virtual Link Descriptor</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VNF</td>
<td>Virtual Network Function</td>
</tr>
<tr>
<td>VNFD</td>
<td>Virtual Network Function Descriptor</td>
</tr>
<tr>
<td>VNFFG</td>
<td>VNF Forwarding Graph</td>
</tr>
<tr>
<td>VNFM</td>
<td>VNF Manager</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WIM</td>
<td>WAN Infrastructure Manager</td>
</tr>
</tbody>
</table>
Executive summary

The programmable edge-to-cloud fabric for dynamic hosting of 5G Virtual Network Functions for the Media Industry (5G-MEDIA) project proposes a solution for the problem of how media applications and the underlying 5G networks should be coupled and interwork, in order to ensure the applications allocate the resources they need to deliver high quality of experience and so that the network is not overwhelmed by media traffic.

The 5G-MEDIA project will extend the ongoing 5G-PPP work on NFV and SDN key technologies for offering advanced managed environment for network services and media-related applications that links their online lifecycle management with user experience and decisions in resources and infrastructures usage optimization. In order to design and create the media services over the platform, developers and users could either use existing applications, or design, create and upload new ones. A key feature of the 5G-MEDIA architecture is the catalogue where the descriptors of available applications and NSs are stored. This catalogue is present in the SDK as a private catalogue, allowing developers to design and validate applications, and in the core of the SVP as a public catalogue, storing all available applications and NSs descriptors for all platform users.

This document provides the design of the 5G-MEDIA public catalogue and its interfaces, as well as the details of a first set of network applications development.

The first chapter introduces this deliverable, its purpose and its relationship with the rest of the deliverables of the project.

The second chapter presents the overall architecture of the 5G-MEDIA platform, introducing the main components.

The third chapter provides the specifications of the information models used by the catalogue.

The fourth chapter depicts the architecture of the public catalogue, defining the main components of it.

The fifth chapter is related to the security framework of the catalogue.

The sixth chapter contains the specifications of the interfaces used by the catalogue.

The seventh chapter defines the development status of an initial set of 5G-MEDIA network applications.

Finally, the final chapter summarizes and concludes the deliverable.
1. Introduction

The goal of this deliverable is to provide the initial design of the 5G-MEDIA Applications and Services Catalogue, its interfaces and a first set of network applications development status. The public Catalogue is a key part of the Service Virtualization Platform (SVP) depicted in D3.1. It allows to store available applications and network services.

The main innovation carried out is the proposition of a generic catalogue, compatible with multiple 5G solutions, trying to leverage the progress made by the H2020 5G-PPP Phase 1 Project SELFNET. This SELFNET core, together with the designs presented in this deliverable, allows the platform to work with different kind of SDN controllers, VIMs and MANOs. This document presents also the mechanisms of Authentication, Authorization and Accounting (AAA) that are used as security framework for each kind of user, as well as the northbound (NB) and southbound (SB) interfaces of the catalogue.

Finally, the document presents the current status of an initial set of generic VNFs and media specific VNFs and PNFs; providing a detailed description of its status, features and behaviour.

1.1. Scope of the Deliverable

This deliverable is focused on the three tasks of WP4 "5G-MEDIA Open Repository of Network Apps": T4.1 “Catalogue Setup and Operation”, T4.2 “Generic Network Apps and Functions” and T4.3 “Media Network Apps and Tools”. T4.1 is in charge of the design of the catalogue and its interfaces while tasks T4.2 and T4.3 are in charge of the set of applications provided by the project. Future D4.2 “5G-MEDIA Catalogue Portal and Network Apps” will contain the final implementation of the catalogue, together with a portal design, which will act as a GUI, and the final set of applications.

1 https://5g-ppp.eu/selfnet/
2. Overall 5G-MEDIA Architecture

The 5G-MEDIA approach delivers an integrated programmable service platform for the development, design and operation of media applications in 5G networks. Figure 1 shows the high-level architecture of the 5G-MEDIA operations and configuration platform. The main building blocks comprising the 5G-MEDIA architecture are:

- An Application/Service Development Kit (SDK) that enables access to media applications development tools, such as editor, validator, emulator, private catalogue, etc.
- A SVP that hosts the components related to the ETSI MANO framework\(^2\), the Applications and Services Catalogue, the Media Service MAPE and the corresponding Virtualized Infrastructure Manager (VIM) and WAN Infrastructure Manager (WIM) plugins enabling the integration to different NFVI platforms.
- Different Network Function Virtualization Infrastructures (NFVIS) comprising the “Physical Layer” that provide computing resources by different operators and supporting different cloud technologies to host generic and media-specific VNFs depicted at the “Virtualized Resource Layer”.

The 5G-MEDIA SDK provides a set of open source tools to support the rapid development of media applications using a DevOps approach. In particular, these tools allow the definition of media service forwarding graphs (also using already existing VNFs, stored in the 5G-MEDIA private VNFD/NSD Catalogue), to proof and package the various functions, as well as to emulate behaviours of the virtualized infrastructure, to accelerate application development and provide a testing environment to be utilized prior to service deployment in the runtime SVP. In addition, the 5G-MEDIA SDK tools enable the use of the innovative concept of the Function-as-a-Service (FaaS). Particularly, 5G-MEDIA pioneers in applying FaaS to VNF management, complementing traditional VM based VNFs with FaaS based media specific functions, aiming at dramatically reducing development cycles and slashing operational costs to 5G-MEDIA users. In this perspective, developers do not have to care about the low-level details related to the virtual computing and storage infrastructure (e.g. virtual server profiling in terms of CPU, RAM, etc.), thus drastically contributing to reduce the service creation time cycle and maintenance effort. In this line, the service developers will be able to create the so-called FaaS VNFs, i.e., VNFs that are instantiated upon the detection of specific events or VNFs whose events trigger specific actions on other components. The combination of the FaaS approach with the VNF packaging and the enablement of inserting FaaS VNFs in a typical VNF forwarding graph is one of the main innovation aspects of the proposed 5G-MEDIA approach.

The 5G-MEDIA SDK interacts with the 5G-MEDIA SVP, which hosts the components related to the ETSI MANO framework (NFV Orchestrator, VNF Manager(s), Infrastructure Manager(s), the public VNF/NS Catalogue and Repositories etc.), as well as unique components that are used to support 5G-MEDIA internal services and deliver its applications to the external stakeholders (e.g. the Media Service MAPE – Monitor, Analysis, Policy, Execution).

Figure 1 – Architecture of the 5G-MEDIA platform
A major innovation of the project is the development of the Media Service MAPE component, which is composed of the Monitoring system, the Cognitive Network Optimization (CNO) engine and the Policy Manager. The development of MAPE will be based, where applicable, to the design, utilization of tools and architectural diagram being part of the outcomes of CogNet 5G-PPP phase 1 project, while achieving integration to OSM components, including Monitoring plugin and Service Orchestrator. The monitoring services aims to monitor the performance/behaviour of the instantiated Network Services (NSs), the integrated NFVIs, the interconnecting networks and the applications themselves. The measured performance metrics are directly used by the CNO engine which comprises mechanisms that take advantage of Machine Learning (ML) techniques and optimization policies to trigger the dynamic instantiation and update of VNF Forwarding Graphs (VNFFGs) or scaling options according to the capabilities provided by OSM (i.e. Scaling Groups construct in the NS descriptors) over the different NFVIs. Alternatively, after the proper execution of the policy classification procedure, 5G-MEDIA SVP will be also able to take advantage of the benefits provided by the VCA Engine (and in particular the Juju adapter provided by OSM) to execute commands on the VNFs themselves, according to the specific use case scenario needs. As it becomes clear, the CNO is able to respond on dynamic changes of the environment (e.g., location change of end users, varying QoS demands) and in cooperation with the Policy Manager to make suggestions to the MANO NFVO and VNFM how to manage/update VNFFGs, perform scaling options, execute commands via Juju charms or even start new components enforced by the FaaS VNFs in order to meet expected QoS/SLA requirements in the most effective way. Last but not least, the QoS/QoE monitoring modules are able to provide both NSs individual and aggregated monitoring views and graphical user interfaces for metrics specified by the developer per NS/VNF, which are part of the generic monitoring system of the SVP.

In the context of 5G-MEDIA approach, as depicted in Figure 1 at the “Virtualized Resource Layer”, each media service comprises a chain of media-specific and network-specific atomic services, all interconnected to deliver an expected output to the end user (media consumer). The resulting graph (i.e., a graph where nodes refer to computing tasks and edges refer to network communication links) is referred as Media Service Forwarding Graph. During the design, development and testing phases, the 5G-MEDIA platform provides appropriate programming tools to abstract the details of the underlying 5G infrastructure and allow developers to focus on the functionality of the services. Once the media service is deployed in the virtualized infrastructure, the 5G-MEDIA platform provides mechanisms to flexibly adapt service operations to dynamic conditions and react upon events (e.g. to transparently accommodate auto-scaling of resources, VNF re-placement, etc.).

Finally, in terms of the supporting physical infrastructure (Physical Layer), the proposed architecture considers that several cloud-based NFVIs will be connected to the SVP via the appropriate VIM and WIM components, enabling the use of computing and networking resources in the respective NFVI.

---

3 [http://www.cognet.5g-ppp.eu/](http://www.cognet.5g-ppp.eu/)
3. Information model specifications

The specification of the descriptors for Network Services, Virtual Network Functions and Multi-access Edge Computing (MEC) applications in 5G-MEDIA has the twofold objective of aligning with the latest standards available in ETSI NFV and ETSI MEC ISGs, while integrating novel extensions and parameters to capture additional requirements of multimedia applications. This approach maximizes the compatibility and the interoperability of 5G-MEDIA architectural components with external NFV products, thus facilitating its acceptance and adoption in the NFV community. Where needed, extensions to the standard information and data models are proposed following the best practices commonly adopted in the languages and protocols used at the NFV MANO and MEC interfaces (e.g. TOSCA CSAR packages or HTTP messages for REST APIs) for augmenting information elements and primitives. In the following, we describe the descriptors adopted in 5G-MEDIA for the definition of Network Services and MEC applications (section 3.1) as well as Virtual Network Functions (section 3.2).

3.1. Specification for generalized Network Service and VNF Descriptors based on TOSCA

The definition of Network Service Descriptors (NSD) and VNF Descriptors (VNFD) in ETSI NFV ISG follows a two-step approach. The first phase involves the specification of language-independent information models that identify the content of the descriptors through a structured tree of inter-correlated and inter-dependent information elements. The second phase translates these information models into data models based on specific languages and package formats, for example the TOSCA YAML Cloud Service Archive (CSAR) or the YANG language.

The abstract information models are documented in a set of “ETSI GS NFV-IFA” reports for both descriptors and NFV MANO components’ interfaces. In detail, the Network Service Template is specified in the ETSI GS NFV-IFA 014 [1] and the Virtual Network Function Descriptor and Packaging format is specified in the ETSI GS NFV-IFA 011 [2]. The most recent version of these reports, v3.1.1, has been released in August 2018. However, the information models for 5G-MEDIA have been defined based on the previous version (v2.4.1) and a possible update based on v3.1.1 specifications will be evaluated during the next months of the project.

The interfaces’ protocols and data models are specified in the “ETSI GS NFV-SOL” reports. The most advanced work is on the TOSCA YAML Cloud Service Archive (CSAR) format, with an official release of the VNF Package specification (ETSI GS NFV-SOL 004 v2.4.1 [3], dated February 2018). The specification of the VNFD and the NSD TOSCA models is still in progress, with the latest draft version of the ETSI GS NFV-SOL 001 [4] released in June 2018, but quite mature especially for the VNFD part. On the other hand, the definition of the YANG models for VNFD and NSD is still at a very early stage in the ETSI GS NFV-SOL 006 [5]. The adoption of TOSCA models is actually quite common in existing NFV frameworks; for example, Cloudify⁴, OpenStack Tacker⁵, OpenBaton⁶ and

---

⁴ https://cloudify.co/
⁵ https://wiki.openstack.org/wiki/Tacker
⁶ https://openbaton.github.io/
ONAP\textsuperscript{7} support TOSCA-based descriptors. TOSCA has been also selected for the reference format in 5G-MEDIA descriptors.

Since at the time of writing this deliverable the ETSI GS NFV SOL001 specification for NFV descriptors based on TOSCA is still work in progress, with its release expected in November 2018, the proposed specification for a generic 5G-MEDIA NSD is based on the combined analysis of Draft ETSI GS NFV-SOL 001 v0.9.0 (June 2018) and ETSI GS NFV-IFA 014 v2.4.1. In particular, IFA 014 was taken into consideration as starting point to define a TOSCA-based NSDs extended with additional information elements that are currently missing in Draft SOL 001. Although the VNFD specification in Draft SOL 001 is more mature than the NSD model, for completeness also TOSCA Simple Profile YAML v1.1\textsuperscript{8} and ETSI GS NFV-IFA 011 v2.4.1 were taken into consideration. However, since the types definitions in TOSCA Simple Profile for Network Functions Virtualization v1.0\textsuperscript{9} (draft 0.4, released on 11th May 2017) are already included in the Draft ETSI GS NFV SOL001, we are considering this last one as main reference.

According to IFA 014, an NSD is a deployment template containing several information used at the Network Function Virtualization Orchestrator (NFVO) for handling the Network Service (NS) lifecycle management. In particular, a NS is a composition of Network Functions (NFs), virtual or physical ones, that could have a specified connectivity between them in the form of one or more forwarding graphs. The NSD should include or refer to the descriptors of its constituent objects as depicted in Figure 2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{nsd.png}
\caption{NSD high level design from ETSI NFV IFA014 v2.4.1 [1]}
\end{figure}

In details, an NSD should:

- Refer to zero, one or more Virtualized Network Function Descriptors (VNFD),
- Refer to zero, one or more Physical Network Function Descriptors (PNFD), where information contained in such descriptor are limited to connectivity requirements for

\textsuperscript{7} \url{https://www.onap.org/}
\textsuperscript{8} \url{http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/TOSCA-Simple-Profile-YAML-v1.1.pdf}
\textsuperscript{9} \url{http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.pdf}
determining how to connect Physical Network Function (PNF) to Virtual Links (VLs) and then integrate them with the NS,

• refer zero, one or more nested NSD, where at least either one VNFD or one nested NSD must be referenced,
• include zero, one or more Virtual Link Descriptors (VLDs),
• include zero, one or more Virtual Network Function Forwarding Graph Descriptors (VNFFGDs), where an NSD can contain different VNFFGDs, each referring to subsets of VLDs, VNFDs and PNFDs for describing the topology of the NS and, eventually, including one or more Network Forwarding Path (NFP).

As stated in IFA011, the VNF Package should contain all the files and metadata descriptors required by a Service Provider for handling VNF lifecycle. In particular, a VNFD should:

• describe the Network Function Virtualisation Infrastructure (NFVI) requirements in terms of needed resources for instantiating a VNF,
• describe dependencies related to the VNF components, for instance connectivity constraints in terms of Virtual Deployments Units (VDUs) and External Connection Points (ExtCPs), in order to allow a proper deployment and configuration,
• describe VNF allowed operations and lifecycle events (e.g. scaling).

The Draft SOL001 specification mentioned at the beginning of this section specifies an NFV data model using the TOSCA language. This data model is intended to be used as a deployment template for VNFs and NSs, defining the NFV specific types to fulfil the NFV requirements and the operational behaviour of such VNFs/NSs. Indeed, the SOL001 assumes a number of node types (e.g. VNF, VDU, VLD etc.), data types etc., according to the TOSCA language specification, for describing VNFs composing a service topology and the NS itself.
In TOSCA Simple Profile YAML, a Service Template (ST) is intended to specify the topology and the orchestration of IT services. Usually a ST contains the Topology Template, which defines the structure of the service and consists of a set of Node Template and Relationship Template definitions.

**Figure 4** presents the class diagram of the implemented information model for a TOSCA descriptor template for representing both NSs and VNFs, while **Figure 5** depicts the class diagram representing three node types in the NS score, i.e. NSNode, SapNode and NsVirtualLinkNode. The members of the presented Java classes with the related methods implement the information model specification produced by ETSI. A detailed explanation of the various fields and semantics is available in [4].
The presented information model specification has been implemented in the first release of the 5G App and Service Catalogue as Java libraries imported in the Catalogue project as Maven dependencies. The two libraries are named *NfvManoLibsSol001Common* and *NfvManoLibsSol001Descriptors*, which contains respectively the SOL 001 base data type and the information model elements. Future implementation includes the following extensions:

- Node types, data types and relationship types definition for representing MEC Application Descriptors (AppDs), implementing the ETSI specifications under ETSI GS MEC 010-2 [7].
- Node types, data types and relationships types definition for representing SDN application descriptors, leveraging on the SELFNET project background available in [8].
- Node types, data types and relationships types for representing monitoring descriptors including NFVI and application-specific metrics. ETSI GS NFV-TST 008 [6] is currently under study for defining a consistent set of compute and networks metrics, a draft nodes and datatypes specification is available in the Annex I – ETSI NFV TST008 v.2.5.1 monitoring data YAML.
3.2. TOSCA CSAR Packaging Format

Within the context 5G-MEDIA, “packaging format” refers to the format employed to package in a single file everything (bar external resources) that needs to be distributed with a VNF or a NS for onboarding purposes. As far as desirable qualities for packaging go, such format is supposed to be compliant with an open and interoperable standard, be capable at describing the content of the package itself, have the capabilities to include or reference software artefacts, ensure its integrity and maintain confidentiality of its most sensible contents. The ETSI CSAR (Cloud Service ARchive) open standard is compliant with each of the requirements mentioned above, therefore a perfect match for 5G-MEDIA.

The CSAR specification has been defined and improved upon in several OASIS and ETSI specifications. The following documents were used for reference: TOSCA Simple Profile in YAML Version 1.0¹⁰, TOSCA Simple Profile in YAML Version 1.1¹¹, TOSCA v1.0 OS¹² and ETSI GS NFV-SOL004¹³. Please note that the TOSCA-specified file formats for Service Templates, Nodes, Relationships and other details of the TOSCA specification are outside of the scope of this document. However, examples pertaining the CSAR format will be provided later on.

3.2.1. Basic file structure and contents

A CSAR package essentially consists of a zip archive that contains a specific file structure. As per TOSCA specification, there are two valid versions of the CSAR format:

1. the metadata directory file structure, based on the presence of a TOSCA-metadata directory at the root of the archive that contains a TOSCA.meta declaration file. The TOSCA.meta file contains Entry-Definitions keywords that are used to reference every other file within the archive, including metadata;
2. the simplified file structure, a YAML definition file at the root of the archive that contains a TOSCA definition template must be present (for example, a VNFD file).

In both options TOSCA definition documents, distinguished by the “.tosca” extension and containing Service Template definitions, are used. In CSAR v1.1 the definition files are usually placed inside a Definitions directory, although, since TOSCA SimpleProfile YAML v1.1, definition files may be placed in any directory. It is worth noting that TOSCA definitions files may be referenced and reused by other CSAR Service Templates, for example by defining Node Types and Relationship Types in one CSAR archive and then referencing them in a Service Template contained inside another CSAR archive.

When using the TOSCA-Metadata archive structure the TOSCA.meta file, which, as previously stated, provides entry information to the parser about the archive and its contents, it must contain at least a block_0 entry which holds information about the archive itself. All the

¹⁰http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd01/TOSCA-Simple-Profile-YAML-v1.0-csprd01.pdf
¹¹http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd01/TOSCA-Simple-Profile-YAML-v1.1-csprd01.pdf
¹²http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf
following blocks shall begin with a name/value pair and will hold metadata information about the files contained or referenced within the archive (for example URLs to the files/artefacts). Such Name/value pairs are constituted by the path to the file (name) and the relative mime type.

An example of TOSCA.meta file content is shown in the Table 1:

| TOSCA-Meta-File-Version: 1.0 |
| CSAR-Version: 1.0 |
| Created-By: OASIS TOSCA TC |
| Entry-Definitions: Definitions/myapp.yaml |
| Entry-Manifest: myapp.mf |
| Entry-Change-Log: changelog.txt |

When using a YAML TOSCA definition template at the root of the archive instead, it must define a metadata section where template_name and template_version are present. It must also be the only YAML file present at the root of the archive. The CSAR specification does not restrict the possible filenames for such file. Other definition files may be present elsewhere in the package.

An exemplifying YAML definition template for demonstration purposes is shown in the Table 2:

```
tosca_definitions_version: tosca_simple_yaml_1_1
metadata:
template_name: MainServiceTemplate
template_author: Onboarding portal
template_version: 1.0
```

### 3.2.2. VNF/NS specific packaging requirements

ETSI GS NFV-SOL004 states that “A VNF Package shall contain the VNFD as the main TOSCA definitions YAML file”, which means that the VNFD will be referenced as the first definitions entry in the TOSCA.meta file using the Entry-Definitions keyword, or used as the root definition template according to the CSAR file structure of choice. Please note that since no reference to Network Services and NSDs is made within any stable CSAR specification, it will be assumed that the same logic and file structures can be applied when packaging Network Services. This principle will be applied through this document from here on. Therefore, in this case, when packaging a Network Service, the NSD will be used as the main definitions YAML file.
Another requirement when packaging VNFs/NSs is the inclusion of a manifest file either at the root of the archive or where referenced by the TOSCA.meta with the keyword Entry-Manifest, depending on the file structure of choice. The manifest file must have the same name of the entry definition file but with an “.mf” extension.

The manifest will contain metadata regarding the VNF and, as explained more in-depth in the next paragraph, an entry for each file contained or referenced within the archive together with its digest and the relative hashing algorithm.

A change log text file is also expected to be included either at the root of the archive or referenced inside the TOSCA.meta through the Entry-Change-Log keyword. This is just a simple text file that should be used to track any changes between different releases of the same package.

A reference to the licence of the archive and its contents is also required, as always by referencing the licence files with the Entry-Licenses keyword within the TOSCA.meta or using a Licences directory located at the root of the archive. If different artefacts that are present or referenced within the archive are distributed under different licences, the relative licence file can be included separately from the primary licence file.

Finally, in order to optionally enable package validation, a Tests directory inside the root of the archive or an Entry-Tests keyword referenced path shall contain definitions of the test required to validate the package when package onboarding occurs. No specifications on the format of the test files seems to be available at the moment.

3.2.3. Package Integrity and Security

ETSI GS NFV-SOL 004 offers two different options to enable integrity checks and security measures within the specification. However, since Option 2 is considered valid only if the CSAR package contains every artefact referenced within the included definition files, Option 1 appears to be the most appropriate choice for 5G-MEDIA. In fact, Option 1 allows the usage of “thin” packages where artefacts (for e.g. such as VM images, software packages, etc.) can be either included or referenced by pointing to external resources. However, artefact inclusion within the package in Option 1 is not mandatory, therefore it offers more flexibility in terms of packaging. That is why 5G-MEDIA has chosen Option 1 as the reference security specification to adopt for its CSAR packaging needs.

The Table 3, taken from the ETSI GS NFV-SOL004, recaps the different characteristics of both Options:
When using Option 1, the manifest file explained in the previous paragraph must contain a digest entry for each file included or referenced within the package by specifying its source, the algorithm used to generate the digest and the hash itself.

An example of a manifest file with the expected header and a single artefact is shown in the Table 4:

Once the manifest file is written the following step is to sign it with the private key of the VNF/NS provider. Such private key is generated using the CA that’s installed at “onboarder” (NFVO by specification) level. Since package validation in 5G-MEDIA happens at Catalogue level, the CA may be stored and accessed here instead. Once the digest is created it is included at the end of the manifest file. When the manifest file is ready, the public certificate of the VNF/NS provider is then included in the archive. In a CSAR file without metadata directory the “.cert” file is expected to have the same name as the main definitions YAML file and placed at the root of the archive, while, as usual, if the TOSCA-Metadata directory is present, the public certificate file location will be indicated using the Entry-Certificate keyword.

This process ensures that no tampering has occurred with the manifest file. Since the manifest file contains digests for all the files included within the archive, this certifies that all the contents of the CSAR archive have maintained their integrity since the manifest file signing has occurred.

An example of file signing with openssl is provided in the Annex II.
Aside from the manifest signing technique, the CSAR standard optionally includes the possibility to sign artefacts individually. When an artefact is signed, the relative signature file and public certificate will share the same name and path of the artefact, with the exception of the relative file extensions, for example:

- **JARS/artifact1.jar** - artifact
- **JARS/artifact1.sha256** - signature
- **JARS/artifact1.cert** - public key

When it comes to the security of artefacts that contain sensitive data, the CSAR specification includes the possibility to encrypt them individually by using both symmetric and asymmetric (private-public) mechanisms. It is worth noting that, when encrypting artefacts contained within a CSAR file, the digest of the file that will be encrypted will be included in the manifest file before the file encryption occurs.

When symmetric artefact encryption is used, the party that will be in charge of onboarding the package (NFVO/Catalogue) will use a public key to encrypt another key generated by the VNF/NS provider. Such encrypted key will then be used to encrypt the artefact and will be delivered within the package using the CMS file format.

When asymmetric (private/public) artefact encryption is used, the party that will be in charge of onboarding the package (Catalogue) will give the VNF/NS provider a public key. This public key will then be used by the package provider to encrypt sensitive artefacts. When the package will be onboarded, the private key available at the onboarding point will be used to decrypt the encrypted artefacts. Since the CSAR specification does not clarify how to distinguish an encrypted file from an unencrypted one, adopting some sort of naming convention such as using a specific file extension (for e.g. “.enc”) or including an encrypted files list at the root of the archive may prove useful to avoid any ambiguity when onboarding occurs.
3.2.4. CSAR structure examples

A couple of examples for both valid CSAR file structures are included for reference and shown in the Figure 6 and Figure 7.

**Figure 6 – CSAR v1.1 with TOSCA.meta**

```
/                      
/TOSCA-metadata/       
   TOSCA.meta         
/Artifacts/           
   myvnf.jar          
/Definitions/         
   myvnf.yml          
   other_definitions.yml
/Licences/            
   GPL2.0.txt         
/Tests/validation_tests.yml
myvnf.cert            
myvnf.mf
```

**Figure 7 – CSAR v1.1 with root YAML definition file**

```
/                      
/Artifacts/            
   myvnf.jar           
/Definitions/          
   other_definitions.yml
/Licences/             
   GPL2.0.txt          
/Tests/validation_tests.yml
myvnf.cert            
myvnf.mf               
myvnf.yml
```
4. 5G App and Service Catalogue design

The 5G App and Service Catalogue is the public catalogue within the 5G-MEDIA SVP and it provides the APIs and functionalities needed for all the management operations on NSDs and App/VNF Packages, like on-boarding, removal, queries, enabling and disabling of specific descriptors or packages. In particular, the Catalogue implements a generalised descriptors information model based on ETSI GS NFV-SOL 001 (see Section 0), which is NFVO-agnostic in terms of descriptor/package syntax and format.

The adoption of the generic information model for representing NSDs and VNF Packages, along with their VNFDs, allows NFV developers and service providers to produce a portable offering of their virtual applications and services, without facing the high fragmentation in the way NS descriptors and VNF packages are represented across the different MANO frameworks. Several formats (e.g. JavaScript Object Notation (JSON) vs YAML Ain't Markup Language (YAML)) are currently used for representing NSDs and VNFDs in NFV environments. Application/VNF Packages are structured differently and can include different contents (e.g. monitoring parameters, software images, etc.) depending on the specific NFV platform. Moreover, different NFVOs and VNFM s usually adopt different procedures for specifying and configuring service’s components (e.g. JuJu charms, cloud-init, other Day1/Day2 configuration recipes, etc.).

Available ETSI NFV MANO frameworks (e.g. ETSI OSM, OpenBaton, Tacker, Rift.Ware, etc.) implement proprietary information models, interfaces and mechanisms for configuring services. Usually, implementations are ETSI standards inspired, but to date there is a proliferation of NFVO-specific descriptors and packages, where framework-specific solutions are used to encode information about monitoring parameters, produced outputs, lifecycle events behaviour (e.g. auto-scaling rules) and NFVI-specific extensions (e.g. SR-IOV, DPDK, containers instead of VMs, etc.).

In future end-to-end 5G networks, composed of multiple virtualized infrastructures distributed across different administrative domains, a unified 5G Apps and Service Catalogue, able to provide an NFV MANO platform agnostic solution, will be needed to address the issues raised today by the fragmentation in NFV MANO solutions, in particular:

- Allowing the composition of end-to-end services combining and stitching elements (e.g. NSs or VNFs/Apps) from different providers;
- Enabling mechanisms for NS orchestration and delivery not limited by the characteristics and capabilities of the provider’s NFVI (e.g. its geographical coverage, its technologies and the quantity of resources available at the edge and at the core of the network);
- Complementing a domain’s catalogue with Virtual Functions from federated MANO systems,
- Facilitate the onboarding of virtual applications designed and made available by several vertical application developers, acting as customers of the NFV Network Operator.
4.2. 5G App and Service Catalogue architecture

The functional high-level design of the 5G App and Service Catalogue is depicted in Figure 8, where the main Catalogue’s building blocks are highlighted:

- **North-Bound Interface (NBI):** With the purpose of being standard-aligned also in terms of interfaces specification and implementation, the NBI of the Catalogue is designed and developed following the REST API defined in ETSI GS NFV SOL005 v2.4.1 [9] specification for the Os-Ma-Nfvo reference point. In particular, implementing the interfaces for NSD Management and VNF Package Management operations (e.g. upload, fetch, update, delete and query), see section [6.2].

- **Admin Interface and Policy Engine:** The Admin Interface is intended to be used by the Catalogue owner for the management of the user’s subscription and policies configuration. In the current implementation, the Admin Interface is implemented in the 5G-MEDIA AAA component within the SVP, see section [6.3] for implementation details.

- **Dispatch Engine:** The Dispatch Engine is the functional module representing the Catalogue’s inner logic for implementing the different workflows related to NSDs and App/VNF Packages management, for instance coordinating the NSD validation, persistency and translation.

- **Notification Dispatch Interface:** The Notification Interface maps to the Catalogue’s message bus handler used for notifying events or distribute specific NSD or App/VNF Package information to external consumers (e.g. information from monitoring descriptors), but also for the intra-Catalogue communication between the different sub-modules.

- **MANO Plugins:** A MANO Plugin can be considered as a connector to a specific underlying MANO framework. In particular, a MANO Plugin is responsible for the translation of descriptors in the MANO-specific format, for the uploading of the package’s images and for actuating the NSD and VNF Package CRUD operation on top of the MANO SO.

- **SDN Plugins:** An SDN Plugin is a component capable of deploying SDN Applications in a specific SDN Controller type (e.g. OpenDaylight14, ONOS15, Floodlight16 etc).

---

14 [https://www.opendaylight.org/](https://www.opendaylight.org/)
15 [https://onosproject.org/](https://onosproject.org/)
16 [http://www.projectfloodlight.org/floodlight/](http://www.projectfloodlight.org/floodlight/)
In terms of software implementation, the 5G App and Service Catalogue is developed as a Java Spring Boot application\(^{17}\) and adopts Maven\(^{18}\) as build automation tool, PostgreSQL\(^{19}\) as SQL database (DB) and Apache Kafka\(^{20}\) as message broker. The Catalogue main software requirements are summarized in Table 5.

---

\(^{17}\) [https://spring.io/](https://spring.io/)

\(^{18}\) [https://maven.apache.org/](https://maven.apache.org/)

\(^{19}\) [https://www.postgresql.org/](https://www.postgresql.org/)

\(^{20}\) [https://kafka.apache.org/](https://kafka.apache.org/)
### Table 5 – 5G App and Service Catalogue software requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle-Java8&lt;sup&gt;21&lt;/sup&gt;</td>
<td>The Java Development Kit, version 8 is preferred.</td>
</tr>
<tr>
<td>Maven</td>
<td>Apache Maven is a tool for software project management that is based on the concept of project object model (POM).</td>
</tr>
<tr>
<td>PostgresSQL</td>
<td>PostgreSQL is a SQL-based open source object-relational database system.</td>
</tr>
<tr>
<td>Apache Kafka</td>
<td>Apache Kafka is a distributed streaming platform that allows to publish and subscribe to streams of records.</td>
</tr>
<tr>
<td>NfvManoLibsSol001Common</td>
<td>Java library developed in the context of the 5G-MEDIA project and implementing the SOL 001 base datatypes, see Section 3.1.</td>
</tr>
<tr>
<td>NfvManoLibsSol001Descriptors</td>
<td>Java library developed in the context of the 5G-MEDIA project and implementing the SOL 001 NSD and VNFD information elements, see Section 3.1.</td>
</tr>
<tr>
<td>J-OSMClient&lt;sup&gt;22&lt;/sup&gt;</td>
<td>Open source Java library implementing a client for operating on top of the OSM Service Orchestrator (SO) NBI.</td>
</tr>
<tr>
<td>OpenStack4j&lt;sup&gt;23&lt;/sup&gt;</td>
<td>Open source Java library implementing a client which allows provisioning and control of an OpenStack instance.</td>
</tr>
</tbody>
</table>

The following picture (<strong>Figure 9</strong>) depicts a simplified software architecture representing the main software components of the catalogue:

---

<sup>21</sup> [https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html](https://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html)
<sup>22</sup> [https://github.com/girtel/J-OSMClient](https://github.com/girtel/J-OSMClient)
<sup>23</sup> [http://www.openstack4j.com/](http://www.openstack4j.com/)
In particular, with reference to the functional design, the Catalogue API Handler comprises the implementation of the NSD Management and VNF Package Management REST controllers plus the Dispatch Engine. Once a request for onboarding an App/VNF Package or an NSD is received, the REST controller invokes initially the policy engine and, then, the proper parser on the base of the request content type. The Descriptor Parser and the Archive Parser are services in charge of transforming the request’s payload in information model objects ready to be stored in the catalogue DB. The DB is based on PostgresSQL, its access and modification are mediated through the Java Persistence API (JPA) and, in particular, the tools made available in the Spring Data JPA library. The validation of the received archive(descriptor) is performed by the Parsers, verifying the format of the package, the syntax of the descriptor and the presence of all the mandatory attributes. Once the information model elements are built, the REST controller saves them in the DB, where information is then accessible through the Repository Interfaces. After saving the descriptor’s elements in DB, the REST Controller notifies the successful on-boarding via the Kafka Message Bus. The MANO Plugins, as well as the SDN Plugins, are listening on specific topics, based on SOL 001 notification messages in

24 http://www.postgresql.org/
the case of the MANO Plugin type. When a notification about the storing of a new NSD/VNF Package is received, the MANO Plugin retrieves the content from the DB, performs the translation in the MANO-specific format and then on-boards the translated NSD/VNF Package in the target MANO framework inner catalogue. Figure 10 presents the on-boarding sequence diagram for an NSD or VNF package.

![Figure 10 – NSD/VNF Package on-boarding sequence diagram](image)

The 5G App and Service Catalogue was designed following the principles of modularity and adaptability, implementing the communication between the core of the Catalogue (API handler) and the Plugins via message bus allows to evolve and maintain each Plugin independently from rest of the software architecture, except for the maintenance of the interface to the bus. In the following sections, we give a more detailed description of the concepts behind the adoption of SDN and MANO Plugins, while, as mentioned above, further implementation details are reported in Section 0.

4.2.2. MANO Plugin

One of the main characteristics/functionalities of the 5G App and Service Catalogue is the capability of interfacing itself with multiple MANO platforms. In particular, the modular design of the catalogue foresees the integration of several MANO Plugins, one for each supported
MANO type (e.g. Open Source MANO, OpenStack Taker\(^{25}\), etc.), in order to allow the onboarding and management of NSDs and VNF Packages in different MANO’s inner catalogues.

A MANO plugin can be considered on one hand as an adaptation layer, where generalized NSDs and VNF Packages are translated in MANO-specific descriptors, while, on the other hand, it implements the catalogue SBI to operate on the underlying NFVO or VIM and to receive notification about services lifecycle events or capabilities updates, e.g. the integration of a new VIM/NFVI PoP in the domain administrated by the MANO platform. Concerning VIMs, another important functionality integrated in the MANO Plugin is the possibility of uploading application images on VIM instances operating in the MANO domain. In summary, a MANO Plugin will contain three main components:

- **Translation Module**: this component is responsible for translating generic NSDs/App Packages into the specific format expected at the underlying MANO Service Orchestrator (e.g. in the case of the 5G-MEDIA project, packages compatible with the OSM information model specification),
- a set of **VIM plugins**: a VIM Plugin will be integrated for each VIM that administrates an NFVI-PoP in the MANO platform administrative domain, the main scope of this component is the uploading of application/function images in the target VIM images’ storage,
- **MANO agent**: for sending commands to on-board, enable, disable and remove descriptors, collecting feedbacks about onboarding and instantiation operations as well as for notification about, for instance, new VIM instances or new capabilities supported by the MANO platform.

In the specific case of the 5G-MEDIA project, the targeted MANO Plugin integrated within the 5G App and Service Catalogue component will be the OSM plugin, while VIM Plugins will include OpenStack\(^{26}\), OpenWhisk\(^{27}\) and OpenNebula\(^{28}\).

4.2.3. **SDN Plugin**

The general solution proposed through the 5G App and Service Catalogue for being compatible with multiple underlying platforms, comprises also the possibility of interfacing different SDN Controllers for SDN Applications deployments. Like in the case of the MANO Plugins, leveraging on the achievement of the SELFNET Project and in particular on the outcomes from the NFV and SDN Repository implementation [8], we propose the integration of SDN Plugins.

As stated by Open Networking Foundation (ONF) and reported in [8], an **SDN Application** is a “software program designed to perform a task in a software-defined networking (SDN) environment. SDN applications can replace and expand upon functions that are implemented

\(^{25}\) https://wiki.openstack.org/wiki/Tacker

\(^{26}\) https://www.openstack.org/

\(^{27}\) https://openwhisk.apache.org/

\(^{28}\) https://opennebula.org/
through firmware in the hardware devices of a conventional network.” In particular, an SDN Application, in SELFNET, is intended as a Virtualized Control Network Function, then an application that, once deployed in an SDN Controller, is capable of configuring and managing devices in the domain network data-plane.

In the 5G-MEDIA project next stage, we envision the use of SDN Applications in correlation with the instantiation of an end-to-end NSs across multiple NFVI-PoPs, then for establishing the inter-VIMs network connectivity in order to allow the communication between NS components deployed in different VIM domains (see [10] and [11] for more details). For instance, the use of an SDN Application capable of provisioning and configuring network paths will be needed for connecting FaaS-based functions in an OpenWhisk domain with standard VNFs deployed using the OpenStack VIM.
5. Security Framework

The Catalogue security framework developed for 5G-MEDIA, also identified as Authentication, Authorization and Accounting (AAA) is in charge of managing the security aspects of user interaction with the Catalogue and the related resources it interfaces with. The AAA provides tokens to the catalogue API users, basically the SDK and the Catalogue itself; it is also in charge of the provisioning of access tokens to the Monitoring for its accounting services and to the SDK to access Monitoring data.

Its main features are the storing of the underlying services credentials, bearer token generation, when supported, in order to limit the sharing of plain credentials, the authorization policies enforcing for the Catalogue users on resources and user resource consumption tracking at NFVI level for the accounting.

The security framework provides two main services: the Authentication and Authorization service (ATN/ATZ) and the Accounting service (ACC). The basic principle on which the ATN/ATZ is implemented is HTTP Filtering: calls to the Catalogue API are intercepted by a filter and then authorized only if compliant to the defined security policies, which are either predefined according to the default Catalogue user roles or customized via policy definition in order to support extendibility. The same filtering principle is applied to all the other resources involved even if not strictly based on HTTP protocol, like in the case of Kafka bus used to manage monitoring data.

The diagram shown on Figure 11 gives an overall outlook on the Security Framework and its interactions with the rest of the 5G-MEDIA platform such as external users (SDK, UI), NFVIs, MANOs, SDN Controllers and the Catalogue itself.

![Figure 11 – Interactions between the AAA Services, the Catalogue and the external resources](image-url)
The ATN/ATZ service on one hand intercepts (with an HTTP Filter) the requests coming from the SDK and other users that interact via the exposed Catalogue API interfaces, and on the other hand with the underlying resources (MANOs, NFVIs and other SDNC appliances) in order to enable users to exploit the Catalogue functionalities, for example, VNF/NS/PNF onboarding. Authentication and Authorization interaction between the ATN/ATZ and the SDK or any other external service happens via the exposed ATN Service. Once the requesting service authenticates, it provides a token that can be employed by the authenticated party (the Catalogue) to request access to the desired protected resource through the specific APIs (MANO API, NFVI POP API, etc.).

Such token contains the relevant information about the authenticated user (for example, the role and some claims later used to get access to protected resources) that is signed by AAA and then verified by AAA when the Catalogue adds it to each request sent to AAA to receive tokens to access protected resources using the AAA Catalogue Service.

In this scenario the Catalogue first authenticates on the ATN Service with a dedicated user, then adds the token provided by the SDK to the call and invokes the AAA Catalogue Service API and request a token for each of the resources it needs to access. Depending on the available support to tokens of the involved resources, the response will be either a token or plain (encrypted) credentials. For example, it could be a valid Keystone token for OpenStack or plain credentials for a SDN Controller.

In principle, whenever the underneath entities support it, AAA will provide tokens to the Catalogue services. This is obviously strictly dependent on the technologies supported and by the security policies imposed by the respective providers. For this reason, the AAA Services rely on a plug-in architecture to support extendibility and manage possible different approaches such as in the case of OSM R3 and R4.

The amount of resources that each user may access depends on the configured roles (e.g. a VNF developer) and the specific policies that enables the user to access some resources, for example the permission to have access only to specific instances of OpenStack. Roles and policies will be described later in the section dedicated to the Authorization.

The Accounting (ACC) service interfaces with the Monitoring service to track computing resources consumption made by 5G-MEDIA users. Data collected by the Monitoring component on the NFVI is sent to the Accounting module via a set of specific APIs. The Accounting service offers reports to review user resource usage and its aggregated data can be employed to support the billing models later defined in the WP7.

To summarize, the key features of the ATN/ATZ/ACC include:

- Centralized user Authentication and Authorization management across different underlying technologies.
- Modular, plugin-based architecture to extend the ATN/ATZ capabilities in order to support different NFVI POPs, MANOs, SDN Controllers and other NFV entities.
Employs temporary bearer tokens that are generated after a successful user login in order to minimize risk of credentials user data exposure. Bearer tokens are employed for user authentication across 5G-MEDIA Catalogue interactions as much as possible, as long as the underlying technologies support them.

- Configurable, policy-based user roles that are expressed in a readable YAML format.
- User resource consumption at NFVI level gets tracked for both FaaS and not FaaS VNFs.

The Authentication, Authorization and Authentication architecture components are further explained in detail in the following paragraphs, starting from the Service and User requirements that have been used as a foundation for the definition of the service capabilities.

### 5.2. AAA Service Requirements

A comprehensive list of requirements regarding the AAA service requirements is shown in the Table 6. The following represent high-level requirements that are envisioned for the AAA to properly support the Catalogue needs, including user and MANO/NFVI/SDNC interaction and management and Accounting facilities.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>AAA offers RESTful APIs for ATN/ATZ</td>
<td>Interact with the ATN service in a programmatic way</td>
</tr>
<tr>
<td>A2</td>
<td>AAA offers a web ATN/ATZ UI</td>
<td>Interact with the AAA service in an interactive way</td>
</tr>
<tr>
<td>A3</td>
<td>AAA hosts user ATN data</td>
<td>host sensitive user data such as username/password</td>
</tr>
<tr>
<td>A4</td>
<td>AAA supports Multi-tenancy</td>
<td>Each User may belong to any number of tenants, each tenant may own any number of resources across multiple NFVIs</td>
</tr>
<tr>
<td>A5</td>
<td>AAA Integrates with MANOs</td>
<td>Integrate ATN/ATZ with underlying MANO for a centralized approach to user management/AAA</td>
</tr>
<tr>
<td>A6</td>
<td>AAA Integrates with NFVIs</td>
<td>Integrate ATN/ATZ with underlying NFVI for a centralized approach to user management/AAA</td>
</tr>
<tr>
<td>A7</td>
<td>AAA Integrates with SDN Controllers</td>
<td>Integrate ATN/ATZ with underlying NFVI for a centralized approach to user management/AAA</td>
</tr>
<tr>
<td>A8</td>
<td>AAA Hosts Service Provider data</td>
<td>Host user-related data needed to interface with the underlying services such as endpoints, administrative accounts, etc.</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Reason</td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>A9</td>
<td>AAA supports Policy management</td>
<td>Offer facilities to host configured ATZ policies</td>
</tr>
<tr>
<td>A10</td>
<td>AAA uses a human readable Policy syntax</td>
<td>Offer an intuitive method for user ATZ policy definition</td>
</tr>
<tr>
<td>A11</td>
<td>AAA enforces ATZ Policies</td>
<td>Offer a service to enforce configured ATZ policies</td>
</tr>
<tr>
<td>A12</td>
<td>AAA Hosts Accounting Data</td>
<td>Offer facilities to host data user-related computing resources consumption</td>
</tr>
<tr>
<td>A13</td>
<td>AAA Aggregates Accounting Data</td>
<td>Aggregate harvested monitoring data given specific rules</td>
</tr>
<tr>
<td>A14</td>
<td>AAA Provides Accounting Reports</td>
<td>Offer to generate human-readable reports of aggregated, user-related computing resources consumption data</td>
</tr>
<tr>
<td>A15</td>
<td>AAA Offers an ACC RESTful API</td>
<td>Offer an API to be employed by the monitoring service to provide user-related computing resources consumption</td>
</tr>
<tr>
<td>A16</td>
<td>AAA offers Simple user provisioning</td>
<td>Let SVPs provision users autonomously and then let the Catalogue administrators provision users that mirror those created at MANO/NFVI level</td>
</tr>
<tr>
<td>A17</td>
<td>AAA offers Centralized user provisioning</td>
<td>Offer to provision users from the AAA service. Such users will then be propagated to the related underlying services</td>
</tr>
<tr>
<td>A18</td>
<td>AAA supports Delegated user ATN/ATZ</td>
<td>Let the underlying services delegate user ATN/ATZ to the AAA service.</td>
</tr>
<tr>
<td>A19</td>
<td>AAA supports Catalogue bearer token authentication</td>
<td>Offer bearer token authentication for interaction with the Catalogue service by the SDK</td>
</tr>
<tr>
<td>A20</td>
<td>AAA supports Resources bearer token management</td>
<td>Offer bearer token management for Catalogue service interaction with the underlying resources</td>
</tr>
</tbody>
</table>

5.3. ATN/ATZ User requirements for the Catalogue

This section describes the Authentication and Authorization requirements for each identified user role and group. User roles hereby presented, while treated as “plain” user roles by the Catalogue, are defined and managed by the ATN/ATZ as a set of policies using the format described in the Policy Format paragraph. Using such customizable policies enables fine grained authorization and provides extensibility. All users are actually stored within a database, known as Resource Registry, that is part of the AAA service.
5.3.2. User Roles definition

Here is a rundown of the User Roles and Groups defined within the AAA Service:

- **Admin**: administrative role. As the name implies it is used to perform administrative tasks. It is the only user that has all permissions granted within the Policy Engine.
- **Developer**: 5G-MEDIA developer role, also known as SDK Developer. Can onboard and interact in various ways with the Catalogue when it comes to VNFs. Developers can also access to NSs and PNFs in read-only mode.
- **Service composer**: The Service Composer is capable of building NSs from available VNFs and define PNFs, therefore he can onboard and interact with them. It can also access VNFs in read-only mode.
- **Internal Catalogue User**: this special user is employed internally by Catalogue to interact with the AAA Internal Service to retrieve stored user, MANO/NFVI/SDNC user data and other AAA related data.

5.3.3. User Groups definition

In certain cases, user role permissions overlap, that’s why some user groups were defined:

- **Service developer**: includes developers and service composers.
- **Authenticated user**: a superset that includes every catalogue user that can authenticate against ATN/ATZ Service.

Since the Admin role has all permissions granted by definition, no Admin group has been specified. While some permissions are available only to Admins, it’s assumed that every requirement enumerated below is either associated to a specific user role or to a group is granted to Admins by default.

5.3.4. AAA User Requirements

User requirements have been written using an agile user story, which means that every requirement should be read as “**As a [user], I [want to], [so that]**” highlighting the users involved, their actions and the expected outcomes.

A comprehensive list of all user requirements for the AAA service is shown in the Table 7. These user requirements will be further clarified in the Policy Format paragraph, where policies derived from such user requirements will be mapped to user roles and Catalogue endpoints.

<table>
<thead>
<tr>
<th>ID</th>
<th>User/Group</th>
<th>Action</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>Internal catalogue user</td>
<td>Be authorized and get tokens</td>
<td>use the NFVIs resources without managing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to access NFVIs</td>
<td>plain username/password</td>
</tr>
<tr>
<td>ID</td>
<td>User/Group</td>
<td>Action</td>
<td>Outcome</td>
</tr>
<tr>
<td>----</td>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>U2</td>
<td>Internal catalogue user</td>
<td>Be authorized and configure authentication/authorization services on the NFVIs</td>
<td>use the NFVIs resources without managing the local users and permissions</td>
</tr>
<tr>
<td>U3</td>
<td>Internal Accounting user</td>
<td>be authorized and access the Accounting services</td>
<td>push accounting data from the monitoring</td>
</tr>
<tr>
<td>U4</td>
<td>Service Composer</td>
<td>be authorized to interact with the Catalogue REST API except for the VNFD creation</td>
<td>get token to access the Catalogue REST API, without the option to create VNFDs</td>
</tr>
<tr>
<td>U5</td>
<td>Service Developer</td>
<td>be authorized to interact with the Catalogue REST API</td>
<td>get token to access the Catalogue REST API</td>
</tr>
<tr>
<td>U6</td>
<td>Service Developer</td>
<td>use a tool to manage standard packaging format</td>
<td>employ a standard, interoperable format for artefact packaging</td>
</tr>
<tr>
<td>U7</td>
<td>Service Developer</td>
<td>support the package artefact digests and encryption</td>
<td>encrypt package artefacts and use artefact digests to ensure that no tampering has happened</td>
</tr>
<tr>
<td>U8</td>
<td>Service Developer</td>
<td>have access at my accounting reports</td>
<td>check the accounting reports about the resources usage on the NFVI</td>
</tr>
<tr>
<td>U9</td>
<td>Authenticated User</td>
<td>manage the provisioned tokens</td>
<td>the tokens provided are visible and manageable by the AAA admin</td>
</tr>
<tr>
<td>U10</td>
<td>Admin</td>
<td>manage users</td>
<td>provision new users, specifying username and password. Modify users by assigning different roles and tenants. Remove users.</td>
</tr>
<tr>
<td>U11</td>
<td>Admin</td>
<td>configure and enforce fine grained permissions for users on NFVI</td>
<td>have a fine grained access control over the NFVI</td>
</tr>
<tr>
<td>U12</td>
<td>Admin</td>
<td>configure authentication/authorization services on the NFVIs</td>
<td>provision service to configure authentication/authorization configuration on the NFVIs</td>
</tr>
</tbody>
</table>

5.4. **AAA Service Architecture**

The architecture of the AAA service is shown in the *Figure 12*. 
The main AAA Service components are:

- The (Policy) Engine applies policy-based access control to the Catalogue resources.
- The PolicyDB stores role policies in YAML format. Policies express who can perform an action exposed by the Catalogue APIs using a specific syntax that is explained in greater detail under the paragraph Policy Format.
- The Resource Registry holds data about users, tenants and underlying MANO/NFVI/SDNC.
- A plugin architecture to interface to different MANOs, NFVIs and resources in general to manage the authentication/authorization through tokens, if supported.

The main interfaces to the SDK, the Catalogue and the Monitoring for the accounting are shown in the Figure 13.
The **CatalogueService** provides an API that allows the Catalogue to know which resources are available for each user and to get the credentials. The **AccountingService** provides an API that allows the Monitoring to open/close tracking sessions related to NS, VNF, VDU and the resource consumptions. The ATN Service provides authentication services with respect to the SDK, the monitoring and the other potential users of the catalogue and supports different protocols to support future integrations of the Catalogue with other services.

Apart from a custom bearer token protocol implementation, **OpenIDConnect v1.0** with an embedded **IdentityProvider** (IDP) is going to be supported, relying on the opensource implementation Keycloak. OpenID Connect is a software framework identity layer built on top of the OAuth 2.0 protocol. Both OpenID Connect and OAuth 2.0 are industry standards in authentication and authorization and are employed by industry leaders such as Google, Microsoft and Valve to give user access to their web APIs. Both standards are typically employed to authenticate and authorize user access to web services on common computing and mobile platforms via browsers, software clients and mobile applications. While OAuth 2.0 can manage user authentication by employing a pseudo-authentication mechanism, its main
focus is on authorization. OpenID Connect adds a simplified workflow and a more robust authentication mechanism to OAuth 2.0. OpenID Connect offers a decentralized approach to authentication and authorization since it is possible to use any number of Identity Providers run by any external entity for user authentication.

The choice to support OpenID Connect is related to the possible integration with OSM R3 that already supports it and embeds an OpenID Connect IDP, although no specification is currently available about authentication delegation to any external IdP. On the other hand, OSM R4 still does not support this protocol, but the integration of such protocol should facilitate the interoperability with 5G-MEDIA and be a valuable contribution to the opensource community.

The custom bearer token ATN protocol is based on JSON Web Token (JWT) and its Java JWT Project implementation. The JWT authentication protocol uses a stateless security approach that improves the scalability of the system and avoids the single point of failure problem.

JWT enables the transmission of information (claims) between two parties in a compact, verifiable form by verifying its signature.

To secure the usage of JWT, a few additional precautions have been taken:

- The signature is verified before trusting any information carried by the token.
- Sensitive data exchange is limited to the internal Catalogue services and is based on a mutual certificate HTTPS channel to guarantee confidentiality.

JWT tokens are of two types: access tokens and refresh tokens. Access tokens contain the information needed to know if a user can access a resource, and have short validity period; refresh tokens are used to get a new access token when they expire.

The high level custom ATN protocol is:

- the client sends to AAA user/pass and gets an access token (with short expiration, for instance 1h) and a refresh token (with longer expiration, for instance 1d)
- the client uses (with bearer authentication) the access token to access protected resources on the catalogue
- when the client gets an error due to expired access token, the client sends to AAA the refresh token (with bearer authentication) and gets a new access token and a new refresh token

The API provided for authentication consists in the HTTP methods shown in the Table 8:

---

29 [https://github.com/jwtk/jjwt](https://github.com/jwtk/jjwt)
**Table 8 – ATN Service API**

<table>
<thead>
<tr>
<th>#</th>
<th>Operation</th>
<th>HTTP method</th>
<th>URI</th>
<th>Body</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Login with user/pass and get access and refresh tokens pair</td>
<td>POST</td>
<td>/api/authenticate</td>
<td>Body with {&quot;username&quot;:&quot;USER&quot;,&quot;password&quot;:&quot;PASS&quot;}</td>
<td>{&quot;id_token&quot;:&quot;ACCESS_TOKEN&quot;,&quot;refresh_token&quot;:&quot;REFRESH_TOKEN&quot;}</td>
</tr>
<tr>
<td>2</td>
<td>Get new access and refresh tokens pair</td>
<td>POST</td>
<td>/api/refresh</td>
<td>Body with {&quot;refresh_token&quot;:&quot;REFRESH_TOKEN&quot;}</td>
<td>{&quot;id_token&quot;:&quot;ACCESS_TOKEN&quot;,&quot;refresh_token&quot;:&quot;REFRESH_TOKEN&quot;}</td>
</tr>
<tr>
<td>3</td>
<td>Access protected resources</td>
<td>ALL</td>
<td>ALL</td>
<td>Header with Authorization: Bearer ACCESS_TOKEN</td>
<td>the protected resource</td>
</tr>
</tbody>
</table>

ACCESS_TOKEN and REFRESH_TOKEN are the actual values of the access and refresh tokens provided by the authentication methods (#1 and #2). Access token allows access to protected resources (see #3), while REFRESH_TOKEN is meant to allow the retrieval of a new access and refresh token without doing login again.

Session logout is managed on the client side, therefore no explicit methods to invalidate tokens is provided by the server.

### 5.4.2. Policy Engine

The Policy Engine is the component in charge of filtering incoming requests to the Catalogue endpoints. When a Catalogue endpoint is requested by a client, the ATN/ATZ Service employs a simple HTTP Filtering technique to limit, deny, or authorize access. Once the requesting user authenticates, the Policy Engine will retrieve the user roles related to his account from the Resource Registry and it will apply an URL based filter depending on the policies, stored within the PolicyDB, that apply to that user. Finally, if the user is authorized to access the requested endpoint, he will be redirected to his desired URL, otherwise the client will receive an HTTP Access Denied response from the ATN/ATZ Service.

The Figure 14 shows the developer interaction with the ATN Service, the SDK and the Catalogue to access resource. Please note how the Policy Engine intervenes via HTTP Filtering. The flow starts (A) with the SDK login to AAA to get a token, then the SDK (B) invokes invoking some methods on Catalogue API adding the token to the request with an Intercepting filter checking the validity of the token and the request is forwarded to the Catalogue application logic, then (C) the Catalogue asks the AAA which resources the user can access and (D) gets the credentials for each of them to later access them and conclude the activities.
5.4.2.1. PolicyDB & Policies Format

In the ATN/ATZ Service, the Policy Engine policies are used to drive a policy-based approach to user authorization within the catalogue by determining which user can access which resource exposed by the Catalogue. Before choosing a custom approach to policy expression, TOSCA standards were investigated to understand whether a TOSCA compliant approach could be maintained for policy expression as with other components of the project. However, TOSCA specs leave to the interested parties to use their own standards for access control policies\(^{30}\), therefore a custom format inspired by the way OpenStack expresses its policies\(^{31}\) has been devised. Due to this, OpenStack Policy YAML file documentation can be roughly used as a reference.

Within the Policy Engine, policies are expressed in a human readable YAML format and are represented by one line \"target:action\" \"rule\" statements. Targets are exposed by the Catalogue and are subjects to actions, for e.g. \"vnf:onboard\". Targets:Actions can also map to specific Catalogue API REST endpoints such as, for example, \"/vnf/onboard/\". Meanwhile rules

\(^{30}\) See paragraph 12.3.1 in [http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd01/TOSCA-Simple-Profile-YAML-v1.1-csprd01.pdf](http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd01/TOSCA-Simple-Profile-YAML-v1.1-csprd01.pdf)

\(^{31}\) see: [https://docs.openstack.org/ocata/config-reference/policy-yaml-file.html](https://docs.openstack.org/ocata/config-reference/policy-yaml-file.html)
express which user role or group may perform a specific action on an object that is manipulated by the target. Rules often involve a specific user performing an action on a specific object, such as, for example, an admin removing a user.

Special rules exist for particular use cases:

- """ - means no rule filtering is applied. Every authenticated user can perform the requested action
- "!" - means never. No user can actually perform that action
- "role:userrole" - is used to match a specific user role
- "group:usergroup" - is used to match a specific user group
- "is_admin" or "role:admin" - means that, in order to perform the requested action, the requesting user must have an Admin user role

Other rules conventions used to achieve more complex rule definition include:

- the usage of the logical operators and, or, not
- referring to specific users with the "%(user_id)" syntax
- referring to specific user groups with the "%(group_id)" syntax
- the "rule:owner" syntax that’s used to check whether the calling user is owner of the object that it is trying to manipulate

When a client calls a Catalogue’s endpoint, the Policy Engine retrieves the calling user’s own roles and groups to check whether the user is actually authorized to perform the requested action. It then applies filtering rules to the requested endpoint according to the policies specified within the PolicyDB. It is worth noting that, despite its name, the PolicyDB is actually a YAML file that contains all the defined policies.

A few examples on policy writing within the Policy Engine:

- VNF onboarding:
  - “vnf:onboard”: “is_admin or %(role_id):developer”
- NS removal:
  - “nsd:remove”: “is_admin or rule:owner or %(role_id):servicecomposer”
- User creation:
  - “identity:user”: “is_admin”

The Table 9 shows an example of Targets and Actions for the Catalogue API. Service endpoints, users that can access them and HTTP methods are included. This table also shows a set of default policies that will be employed by the Policy Engine which are derived from the AAA Service Requirements. Please refer to the ATN/ATZ Requirements for the Catalogue paragraph for more information.
Table 9 – example of AAA policies for the Catalogue API

<table>
<thead>
<tr>
<th>ID</th>
<th>User</th>
<th>Target</th>
<th>Actions</th>
<th>Method</th>
<th>Endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Developer</td>
<td>vnf</td>
<td>Create</td>
<td>POST</td>
<td>/nfvo/vnfd_management/vnf_packages/*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Upload</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Upload_from_uri</td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Remove</td>
<td>PATCH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Service</td>
<td>vnf</td>
<td>Query</td>
<td>GET</td>
<td>/nfvo/vnfd_management/vnf_packages/*</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td></td>
<td>Fetch_vnfd</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fetch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Authenticated</td>
<td>vnf</td>
<td>Subscribe</td>
<td>POST</td>
<td>/nfvo/vnfd_management/subscriptions/*</td>
</tr>
<tr>
<td></td>
<td>Users</td>
<td></td>
<td></td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Service</td>
<td>nsd</td>
<td>Create</td>
<td>POST</td>
<td>/nfvo/nsd_management/ns_descriptors/*</td>
</tr>
<tr>
<td></td>
<td>Composer</td>
<td></td>
<td>Upload</td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Remove</td>
<td>PATCH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>Service</td>
<td>nsd</td>
<td>Query</td>
<td>GET</td>
<td>/nfvo/nsd_management/ns_descriptors/*</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>Service</td>
<td>pnfd</td>
<td>Create</td>
<td>POST</td>
<td>/nfvo/nsd_management/pnf_descriptors/*</td>
</tr>
<tr>
<td></td>
<td>Composer</td>
<td></td>
<td>Upload</td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Remove</td>
<td>PATCH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update</td>
<td>PUT</td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>Service</td>
<td>pnfd</td>
<td>Query</td>
<td>GET</td>
<td>/nfvo/nsd_management/pnf_descriptors/*</td>
</tr>
<tr>
<td></td>
<td>Developer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>Authenticated</td>
<td>nsd</td>
<td>Subscribe</td>
<td>POST</td>
<td>/nfvo/nsd_management/subscriptions/*</td>
</tr>
<tr>
<td></td>
<td>user</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.4.3. Resource Registry

The Resource Registry acts as central repository for data that is used by the ATN/ATZ Service. The Resource Registry is a database that stores user data such as username and passwords, tokens, registered MANO/NFVI/SDNC endpoints and user accounting data retrieved from the monitoring system. The Resource Registry is typically accessed by the Policy Engine to retrieve said data when performing its authentication & authorization tasks.

Data that is stored within the Resource Registry:

- User data: username, password, roles
- Catalogue Endpoint: name, URL, method and properties
- Tenant: Catalogue tenants
- Resources: id, name, type, url, tokens and authentication data required (username, password, pubkey) to access each resource
- Accounting data: VNF and FaaS (VDU) ids, names and usage data (timestamps, CPU cycles, RAM, disk space occupied)

The AAA class diagram for the management of authentication and authorization is shown in the Figure 15.

![Figure 15 – AAA class diagram for authentication and authorization](image)

Basically, each catalogue user (CatalogUser) as assigned a set of credentials for each of the NFVI resource (Resource). Such credentials can be specific for the user (ResourceUserLogin) or admin-like for the remote configuration (ResourceAdminLogin). Users are then grouped per tenant (CatalogTenant) that in turn may have policies (Policy) to control user access over resources. Tokens are then managed for both catalogue users (CatalogueToken) and resource
users (ResourceToken). Finally, VDU resource consumptions for accounting (AccVduResourceConsumption) is tracked using a hierarchical structure from NSs, VNFs, VDUs.

5.5. **ATN/ATZ MANO Integration**

The ATN/ATZ Service offers different possible integrations with the MANO instances, the NFVI POPs and with other different resources that are configured through its services that require the Catalogue to exploit its authentication and authorization capabilities. Apart from offering basic services to store credentials (Resource Registry) and policies (PolicyDB) that describes “who” can access “what”, and the software component that enforces such policies (Policy Engine), the ATN/ATZ Service aims to centralize user and permission management for all the underlying resources, depending on the possible integration that each type of resource offers. The main goal is of the ATN/ATZ is to integrate, from the security perspective, with MANO frameworks and, in particular, OSM with its two main releases currently available: OSM R3 and OSM R4. The plugin architecture enables extensibility beyond the 5G-MEDIA project requirements.

The following paragraphs explore several integration scenarios that have been envisioned depending on the underlying technologies the ATN/ATZ Service interfaces/integrates with.

5.5.2. **Plain Credentials**

This integration basically sees AAA as a plain credentials provider to the Catalogue and it is used in all the cases where resources do not allow any other feature to delegate ATN/ATZ. This category typically involves hardware devices, SDN controllers and so on. This kind of integration is employed when there’s a need to provide to the Catalogue, through a secure API or a local software component, user credentials that are required to interface with this kind of device or service. It lets the Catalogue manage both the ATN flow and the communication with the resources. To reduce security issues, communication between the Catalogues and the AAA services is either at API level in case of a common application server (e.g. the same Spring Boot instance), or using mutual certificate authentication over HTTPS channel.

5.5.3. **Bearer Token**

This integration technique sees the AAA Service as a token provider to the Catalogue and is possible whenever the underlying resource supports token as a way to limit the plain username/password exchange. If the request is authorized by AAA, plain authentication only occurs once at the beginning of the user interaction with the system, then the Catalogue receives a valid Bearer token that can be used for any further interaction between the Catalogue itself and the underlying resource, as long as the token itself is not expired.

A typical use case sees the Catalogue requesting a token to the ATN/ATZ Service for a specific resource. The AAA Service invokes the specific resource endpoint (with the previously defined credentials stored in AAA) to get the token and give it back to the Catalogue. From that moment the Catalogue adds it to its requests toward the resource.
The obvious advantage is that most of the communication from the Catalogue to the resource is done using temporary credentials (tokens). Both OSM R3 and OSM R4 lightweight support this kind of integration although with different token support implementation.

5.5.4. Centralized ATN/ATZ Management

This integration scenario considers the ATN/ATZ Service as a centralized authentication and authorization service for both the Catalogue and the resources that integrate with the ATN/ATZ Service. The downside of this approach is that AAA requires an administrative account for the underlying resource to be managed, so this makes sense only if they share the same security domain (e.g. are managed by the same organization).

A typical use case sees AAA on which a new user is created along with its relative permissions, then AAA provisions such user directly into the managed resource using an admin account on such resource. The advantage of this approach is to make user provisioning easier as everything can be configured from a single portal. The disadvantage is the need for an administrative account, something that not every SVP may agree with, especially among different security domains. Both OSM R3 and OSM R4 lightweight support this kind of integration, but with completely different NBI interfaces.

5.5.5. Delegated ATN/ATZ Management

Differently from the centralized ATN/ATZ Management, this integration scenario requires the resources to support delegation either by having a local repository with their users/permissions - while still accepting external users as authenticated and authorized on its platform (for example Shibboleth\(^32\), which includes Single Sign-On features), or by relying completely on a remote repository for ATN/ATZ. This scenario has strong dependencies on the technologies supported by the managed resource itself. In this case the ATN/ATZ offers an identity database to the managed resource, but it’s the resource itself that queries the ATN/ATZ Service when a user tries to perform an action that requires an authentication.

OSM R3 supports this scenario by using a PAM Module which allows both local and remote authentication at the same time. When a user tries to authenticate, PAM can perform the authentication both locally and/or remotely with different priorities depending on its configuration. However, OSM R3 still requires the ATZ part to be setup through its own API, so that a remote call from the ATN/ATZ Service is needed to complete the process.

OSM R4 has not such feature yet, so a possible integration would require a specific component to be added to its own ATN/ATZ application logic.

5.5.6. OSM R3 Integration

As stated in the previous chapters, OSM is the main MANO framework used in the 5G-MEDIA project, in particular OSM R3 release is the MANO used for the first implementation. From the

\(^{32}\) [https://www.shibboleth.net/](https://www.shibboleth.net/)
perspective of security integration OSM R4 is completely different to the previous release and will be described in the next section.

OSM R3 includes an experimental extension to support multi tenancy that relies on PAM (Pluggable Authentication Module)\(^{33}\). PAM is generally used in Linux OSs to enable authentication for both users defined on the operating system and from remote sources. PAM supports multiple modules that can be configured in pipeline with a “control flag”. Each module may have different priorities to establish whether the allowed/denied result module execution should be considered necessary (“required”) or enough (“sufficient”) to consider a user authenticated.

Each module may support a different authentication mechanism, such as users defined in the local operating system (e.g. in “/etc/passwd”), reference a local/remote server using a standard protocol (LDAP, RDBMS), or even use a customized protocol (e.g. a remote HTTP server using REST calls). When employing a PAM module that supports remote authentication, remote OSM R3 user authentication can achieved, either by making it remotely managed (see the “Centralized ATN/ATZ Management” paragraph), delegated (see the “Delegated ATN/ATZ Management” paragraph) or even a local/remote hybrid.

Since OSM R3 supports PAM for ATN, it may be configured to execute any of the modules mentioned above (many of which available from the open source community) or even a custom one. However, this would be only applied to the authentication process. In fact, once a user is authenticated, OSM R3 still requires a further configuration step for the authorization on OSM “projects”. This second step either requires a manual configuration through the OSM UI or an application callback to the OSM NBI from the AAA Service, which is exactly what is provided by the OSM R3 plugin.

The integration described in this section aims to provide complete ATN/ATZ management over OSM R3 from the AAA Service, in order to overcome the manual configurations that should be otherwise necessary to obtain the same goal. The integration of the OSM R3 ATN services in 5G-MEDIA is achieved configuring a PAM HTTP module\(^{34}\) that delegates ATN to the AAA Service. Such module forwards the encrypted credentials to the AAA Service and gets back an allowed/denied result according to the PAM protocol. This implies a trust relation between the OSM R3 instance and the AAA Service that is obtained with mutual, certificate-based HTTPS authentication.

Another option involves using OSM R3 with PAM MySQL\(^{35}\) (or any other supported RDBMS) and a custom module that the A.A.A. Service interfaces with to manage users; in specific cases within the same organization and with secured point to point communication, OSM and the AAA Service could potentially share the same RDBMS and avoid such customization.

All the mentioned options share the PAM ATN delegation, they only differ in protocol used to communicate with the AAA Service (HTTP, MySQL, custom).

---

\(^{33}\) [https://en.wikipedia.org/wiki/Pluggable_authentication_module](https://en.wikipedia.org/wiki/Pluggable_authentication_module)

\(^{34}\) [https://github.com/john-sharratt/pam-http](https://github.com/john-sharratt/pam-http)

The integration of OSM R3 ATZ services in 5G-MEDIA is achieved with a callback from the AAA Service to the OSM NBI, with two main options available:

- Through HTTP calls to the OSM SO that are conformant to the ones used by the OSM UI, by relying on an “HTTP-aware client library” such as HttpClient\(^\text{36}\) to execute them; this does not require any extension in OSM;
- Through HTTP calls to an ad-hoc OSM SO tasklet\(^\text{37}\) (according to the OSM SO integration guidelines) that translates the request into the internal OSM R3 format and passes it to the internal bus (DTS) to be elaborated.

5.5.7. OSM R4 Integration

OSM R4 has been released in June 2018 and provides a different NBI wrt OSM R3 to manage ATN/ATZ services. A first evaluation shows that bearer token ATN is supported, yet no standard protocols (such as OpenID Connect) are supported. A possible extension to integrate with AAA Service will be necessary. A complete evaluation of the OSM R4 MBI interface has been postponed to allow implementation and testing on OSM R3 for the mid-term review. The plan is to support R4 by the end of the project, updated specifications will be provided in the next deliverables.

5.6. Accounting

The service provided by the Accounting service is meant as a basic tracking service to enable different billing models to be plugged in and used on top of the data retrieved by the monitoring and stored by AAA Service.

The main interactions are with the monitoring, to receive the NFVI resource usage for each user/tenant, and with the reports provided to the users consuming of such data. As a starting point, data is presented by using historical reports in order to quickly identify resource usages about the common NFVI resources such as CPU cycles, memory and disk space usage. OpenNebula\(^\text{38}\) approach to reporting has been used a reference. A noteworthy difference from the typical virtual machine-based usage monitoring is the tracking of resources related to FaaS, where the underlying resources are containers running in Kubernetes instead of plain virtual machines, and where monitoring data is retrieved by dedicated monitoring software such as Kubelets\(^\text{39}\), although many other options are available.

The AAA class diagram for the management of accounting is shown in the Figure 16, where each Catalogue user has usage sessions organized hierarchically: from NS, to VNF and finally to VDU consumptions for the main resources (CPU, MEMORY, DISK).

\(^{38}\) [https://opennebula.org/](https://opennebula.org/)
The basic interactions with the Monitoring are shown in the Figure 17.
The above class diagram summarizes the interaction flow between the Monitoring Service, available MANOs and the Accounting Service. In short, the Monitoring Service authenticates with the AAA Service and retrieves a bearer token. Once the Monitoring service is authenticated, the Monitoring Service retrieves a list of all the available resources per any given user, then it opens/closes a monitoring session (per NS, VNF, VDU) for each user sending the associated resource usage data to the Accounting Service.

5.6.2. Events and metrics collector for the Accounting service

This component is an application developed for purposes of accounting the operational states and relations of the Network Service (NS) instances, Virtual Network Functions (VNFs) and Virtual Deployment Units (VDUs) of an OSM tenant, while receiving metrics concerning the running VDUs. The aforementioned actions are conducted for billing purposes.
The leading role of this component is to record the relations of a tenant with an NS instance and its underlying VDUs, while also recording every alteration of state for each VDU and NS of every tenant. This component can be decomposed into:

- the NFVI Events Handler,
- the database with the exposed API,
- the Accounting client,
- the OSM RO client and
- the NFVI Metrics Collector

These components have been integrated, successfully tested and validated with OpenStack and DevStack, yet more types of NFVIS will be gradually introduced.

These components have been developed in Python and is deployed as a fully dockerized application. For the database & API, Django and Django REST Framework have been employed, while PostgreSQL has been selected as the database and the application is served through Apache server and the WSGI module. The documentation of the RESTful API is available online through the Swagger framework. The Accounting Service client and OSM RO client are implemented as simple HTTP clients. Moreover, the NFVI Metrics Collector consumes messages from specific topics in the 5G-MEDIA Kafka broker while the Celery has been employed for purposes of periodically forwarding the collected metrics to the Accounting service. The Figure 18 shows the whole architecture.

![Figure 18 – Metrics & events collection for Accounting](image-url)
NFVI Events Handler

The NFVI Events Handler is the component that connects to a broker or any other interface offered by a NFVI (e.g. RabbitMQ for OpenStack) and receives the notifications it produces. The general idea behind this component is that it receives notifications about events that occur concerning any of the NFVI’s VDUs. The events might include, but are not limited to, creation, pause, suspension, powering off or deletion of a VDU. The VIM notification handler processes each of the notifications about an event accordingly.

Create Notifications

On creation of a VDU, the NFVI Events Handler receives a relevant notification in JSON format including the UUID of the newly created VDU. Before creating a VDU object in the database, the tenant and NS instance that the VDU belongs to should be retrieved. Unfortunately, no direct reverse matching between a VDU and a tenant is offered by the OSM API. Therefore, in order to retrieve the tenant and NS instance of a specific VDU the following operations should be executed:

- Retrieval of all OSM tenants employing the OSM RO-API
- For every tenant:
  - Creation of a new tenant object if it does not exist in the database
  - Retrieval of all NS instances for current tenant via the OSM RO-API
  - For each NS instance:
    - Creation of a new NS instance object if it does not exist in the database
    - Retrieval of all VNFs for current NS instance via the OSM RO-API
    - For each VNF:
      - Access the VMs in the VNF until the UUID of the newly created VM is found
  - When the tenant and NS instance of the VDU is found create a new VDU object under this tenant and NS instance and set state to “active”

Pause, Suspend, Shut Down Notifications

For each notification that concerns a state transition of a VDU, or subsequently an NS instance, a new action object is created. The JSON schema of the action object is presented below.

```json
{
    "id": "id",
    "vdu_uuid": "vdu_uuid",
    "state": "state",
    "start_datetime": "start_datetime",
    "end_datetime": "end_datetime"
}
```

On pause, suspension or shut down messages, the state of the VDU object is updated accordingly, along with the state of the NS instance it belongs to. The end datetime is set on
the latest action object and a new one is created denoting the beginning of this state for the VDU and NS instance.

**Delete Notifications**

Upon deletion of an NS instance, separate delete notifications will be received for each of the underlying VDUs. The state of the instance and the VDUs will be set to “deleted”, and no notifications will be received henceforth regarding them. However, these objects will not be deleted from the database as they will be required in case of a requested instance or VDU accounting for a specific tenant.

**Database & API**

The implemented API includes endpoints responsible for fetching NS instances by tenant, VDUs by NS instance or tenant etc. The recording of all state transitions of NS instances and VDUs by the NFVI Events Handler enables this component to provide an API including endpoints able to retrieve information referring to a specific time range about the overall hours of a VDU being active, paused, suspended and shut down, or the overall hours of an NS instance being up or down depending on the state of its underlying VDUs etc. The aforementioned operations concerning NS instances or VDU accounting are available per tenant.

**Accounting Service Client**

The Accounting client is an HTTP client developed for purposes of interfacing with the API offered by the Accounting service. The client serves as a wrapper for making requests to the accounting service, opening and closing NS, VNF and VDU sessions, while also forwarding selected metrics for each VDU.

**OSM RO Client**

The OSM RO client is another HTTP client developed that invokes web services of the OSM RO API. This client is an essential element as it interfaces with OSM and retrieves information about (OSM) tenants, data centers, NS instances, VNFs and their relations. The OSM RO client plays a significant role as the relations between NS instances, VNFs, VDUs and tenants must be constructed, leveraging on the notifications received and handled by the NFVI Events Handler.

**NFVI Metrics Collector**

The NFVI Metrics Collector component is responsible for collecting resources’ consumption from the Monitoring services regarding the running VDUs. For purposes of billing, the metrics that are collected are the CPU utilization, memory usage and disk usage. The NFVI Metrics Collector is a consumer on the Kafka broker where monitoring is pushing the collected metrics received from NFVIs (e.g. ceilometer in case of OpenStack). Periodically, the NFVI Metrics Collector aggregates the collected metrics per VDU and, publishes them to the Accounting service.
6. Interfaces specification

6.2. 5G App and Service Catalogue NBI

The 5G App and Service Catalogue is a component that operates on top of different MANO Platforms. With the purpose of implementing a neutral North-Bound Interface, independent from any current implementation of MANO’s inner catalogues, the NBI design leverages on latest ETSI NFV standards, in particular following the specifications for the NSD Management and VNF Package Management interfaces provided in the ETSI NFV SOL005 v2.4.1 [9] (released in February 2018) specification. In the following tables are listed all the operations available at the 5G App and Service Catalogue NBI for the onboarding and management of NSDs, PNFDs and VNF Packages. In particular, the Java Spring REST Controller skeleton for NSDs and VNF Packages, along with the SOL 005 information model elements, was generated through the Swagger Codegen\(^\text{40}\) tool. As inputs for the Swagger Codegen we used the NSD Management YAML from ETSI OpenAPIs\(^\text{41}\), extended with the information model elements missing in the ETSI available version, a draft version is available in Annex III – ETSI NFV SOL005 v2.4.1 NSD Management Interface YAML, while the YAML for the VNF Package Management was defined from scratch.

Following tables present the set of REST operations available at the Catalogue NBI.

\(\text{Table 10 – NSD Management Interface}\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create NSD resource</td>
<td>POST</td>
<td>/nsd/v1/ns_descriptors</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query NSDs</td>
<td>GET</td>
<td>/nsd/v1/ns_descriptors</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query NSD</td>
<td>GET</td>
<td>/nsd/v1/ns_descriptors/{nsdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Delete NSD</td>
<td>DELETE</td>
<td>/nsd/v1/ns_descriptors/{nsdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Modify NSD</td>
<td>PATCH</td>
<td>/nsd/v1/ns_descriptors/{nsdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Fetch NSD content</td>
<td>GET</td>
<td>/nsd/v1/ns_descriptors/{nsdInfoId}/nsd_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
</tbody>
</table>

\(^\text{40}\) https://swagger.io/tools/swagger-codegen/
\(^\text{41}\) https://nfvwiki.etsi.org/index.php?title=API_specifications
### VNF Package Management Interface

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload NSD content</td>
<td>PUT</td>
<td>/nsd/v1/ns_descriptors/{nsdInfoId}/nsd_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Create PNFD resource</td>
<td>POST</td>
<td>/nsd/v1/pnf_descriptors</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query PNFDs</td>
<td>GET</td>
<td>/nsd/v1/pnf_descriptors</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query PNFD</td>
<td>GET</td>
<td>/nsd/v1/pnf_descriptors/{pnfdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Delete PNFD</td>
<td>DELETE</td>
<td>/nsd/v1/pnf_descriptors/{pnfdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Modify PNFD</td>
<td>PATCH</td>
<td>/nsd/v1/pnf_descriptors/{pnfdInfoId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Fetch PNFD content</td>
<td>GET</td>
<td>/nsd/v1/pnf_descriptors/{pnfdInfoId}/pnfd_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Upload PNFD content</td>
<td>PUT</td>
<td>/nsd/v1/pnf_descriptors/{pnfdInfoId}/pnfd_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Subscribe</td>
<td>POST</td>
<td>/nsd/v1/subscriptions</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query subscriptions</td>
<td>GET</td>
<td>/nsd/v1/subscriptions</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Query individual subscriptions</td>
<td>GET</td>
<td>/nsd/v1/subscriptions/{subscriptionId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
<tr>
<td>Delete individual subscription</td>
<td>DELETE</td>
<td>/nsd/v1/subscriptions/{subscriptionId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 5.4 for resource description</td>
</tr>
</tbody>
</table>

### Table 11 – VNF Package Management Interface

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create VNF Package resource</td>
<td>POST</td>
<td>/vnfpkgm/v1/vnf_packages</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Query VNF Packages</td>
<td>GET</td>
<td>/vnfpkgm/v1/vnf_packages</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Operation</td>
<td>HTTP Method</td>
<td>URI</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Query VNF Package</td>
<td>GET</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Delete VNF Package</td>
<td>DELETE</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Update VNF Package</td>
<td>PATCH</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Query VNFD</td>
<td>GET</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}/vnfd</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Fetch VNF Package content</td>
<td>GET</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}/package_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Upload VNF Package content</td>
<td>PUT</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}/package_content</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Upload VNF Package from URI</td>
<td>POST</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}/package_content/upload_from_uri</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Fetch VNF Package artifact</td>
<td>GET</td>
<td>/vnfpkgm/v1/vnf_packages/{vnfPkgId}/artifacts/{artifactPath}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Subscribe</td>
<td>POST</td>
<td>/vnfpkgm/v1/subscriptions</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Query subscriptions</td>
<td>GET</td>
<td>/vnfpkgm/v1/subscriptions</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
</tbody>
</table>
Moreover, the 5G App and Service Catalogue will support also the onboarding and management of Multi-access Edge Computing (MEC) applications. The list of operations that will be available in the catalogue’s next release for App Packages management is reported in the following table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query individual subscription</td>
<td>GET</td>
<td>/vnfpkgm/v1/{subscriptionId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
<tr>
<td>Delete individual subscription</td>
<td>DELETE</td>
<td>/vnfpkgm/v1/{subscriptionId}</td>
<td>Cf. ETSI NFV SOL005 v2.4.1 Sec. 9.4 for resource description</td>
</tr>
</tbody>
</table>

Moreover, the 5G App and Service Catalogue will support also the onboarding and management of Multi-access Edge Computing (MEC) applications. The list of operations that will be available in the catalogue’s next release for App Packages management is reported in the following table:

Table 12 – App Package Management Interface

<table>
<thead>
<tr>
<th>Operation</th>
<th>HTTP Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create App Package resource</td>
<td>POST</td>
<td>/apppkgm/v1/app_packages</td>
<td></td>
</tr>
<tr>
<td>Query App Packages</td>
<td>GET</td>
<td>/apppkgm/v1/app_packages</td>
<td></td>
</tr>
<tr>
<td>Query App Package</td>
<td>GET</td>
<td>/apppkgm/v1/app_packages/{appPkgId}</td>
<td></td>
</tr>
<tr>
<td>Delete App Package</td>
<td>DELETE</td>
<td>/apppkgm/v1/app_packages/{appPkgId}</td>
<td></td>
</tr>
<tr>
<td>Query AppD</td>
<td>GET</td>
<td>/apppkgm/v1/app_packages/{appPkgId}/appd</td>
<td></td>
</tr>
<tr>
<td>Fetch App Package content</td>
<td>GET</td>
<td>/apppkgm/v1/app_packages/{appPkgId}/package_content</td>
<td></td>
</tr>
<tr>
<td>Upload App Package content</td>
<td>PUT</td>
<td>/apppkgm/v1/app_packages/{appPkgId}/package_content</td>
<td></td>
</tr>
<tr>
<td>Upload App Package from URI</td>
<td>POST</td>
<td>/apppkgm/v1/app_packages/{appPkgId}/package_content/upload_from_uri</td>
<td></td>
</tr>
<tr>
<td>Subscribe</td>
<td>POST</td>
<td>/apppkgm/v1/subscriptions</td>
<td></td>
</tr>
<tr>
<td>Query subscriptions</td>
<td>GET</td>
<td>/apppkgm/v1/subscriptions</td>
<td></td>
</tr>
<tr>
<td>Query individual subscription</td>
<td>GET</td>
<td>/apppkgm/v1/{subscriptionId}</td>
<td></td>
</tr>
<tr>
<td>Delete individual subscription</td>
<td>DELETE</td>
<td>/apppkgm/v1/{subscriptionId}</td>
<td></td>
</tr>
</tbody>
</table>
6.3. 5G App and Service Catalogue Admin interface

6.3.2. AAA Admin UI

The AAA provides an admin web portal that grant access to four different type of users with different privileges:

- plain users: can register on the portal and request access to specific resources;
- admins: configure the resources, check the users (Figure 19) and authorize with specific permissions (Figure 20), check the platform health status, check the token provisioning and validity, can manage the log level for the platform and verify the access audit (Figure 21), have access to the accounting reports;
- accounting and catalogue internal users: have access to the Swagger API and the Postman examples.

Among the main features provided to admins, the ability to configure a user on a remote SVP and, depending on the integration provided, the remote configuration of a user/tenant.

Some examples of the portal are reported below.

Figure 19 – AAA users management

Figure 20 – AAA user permissions
6.4. 5G App and Service Catalogue SBI

6.4.2. Notification service

In order to implement the notification service, the 5G App and Service Catalogue integrates the use of a message bus, in particular, for this purpose we select the Apache Kafka message bus for being compliant with the technology adopted for the implementation of the 5G-MEDIA MAPE in the SVP [12]. The notification service’s scope is to notify Catalogue’s internal and external consumers about NSDs and App/VNF Packages management events.

For the Catalogue’s inner events, the message bus is configured with NSD and VNF Packages specific topics, derived from the notification message types specified in SOL005 v2.4.1:

Table 13 – SOL005 Notification Message types and corresponding topics

<table>
<thead>
<tr>
<th>Notification Message Type</th>
<th>Message Bus Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NsdOnboardingNotification</td>
<td>onboard</td>
</tr>
<tr>
<td>NsdOnboardingFailureNotification</td>
<td>onboard</td>
</tr>
<tr>
<td>NsdChangeNotification</td>
<td>change</td>
</tr>
<tr>
<td>NsdDeletionNotification</td>
<td>delete</td>
</tr>
<tr>
<td>PnfdOnboardingNotification</td>
<td>onboard</td>
</tr>
<tr>
<td>PnfdOnboardingFailureNotification</td>
<td>onboard</td>
</tr>
<tr>
<td>PnfdDeletionNotification</td>
<td>delete</td>
</tr>
<tr>
<td>VnfPackageOnboardingNotification</td>
<td>onboard</td>
</tr>
<tr>
<td>VnfPackageChangeNotification</td>
<td>change</td>
</tr>
</tbody>
</table>

SB MANO Plugins are Catalogue’s inner consumers on the above listed topics, for instance, if a new NSD is onboarded, MANO Plugins consume a NsdOnboardingNotification, which includes the needed information for retrieving the NSD’s elements from the DB and translating them in the proper format expected at the targeted NFVO.

Table 10 and Table 11 reports respectively the operations available at the MANO NsdNotificationsConsumerInterface and VnfPackageNotificationsConsumerInterface implemented within each MANO Plugin:
5G-MEDIA - Grant Agreement number: 761699
D4.1 – 5G-MEDIA Catalogue APIs and Network Apps

Table 14 – Nsd Notification Consumer Interface operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptNsdOnBoardingNotification</td>
<td>This method is called when a new NSD is on-boarded, after all the steps are done.</td>
</tr>
<tr>
<td>acceptNsdChangeNotification</td>
<td>This method is called when there is a change of state in an on-boarded NSD. Only a change in operational state will be reported. A change in usage state will not be reported.</td>
</tr>
<tr>
<td>acceptNsdDeletionNotification</td>
<td>This method is called when an on-boarded NSD is deleted.</td>
</tr>
<tr>
<td>acceptPnfdOnBoardingNotification</td>
<td>This method is called when a new PNFD is on-boarded, after all the on-boarding steps are done.</td>
</tr>
<tr>
<td>acceptPnfdDeletionNotification</td>
<td>This method is called when an on-boarded PNFD is deleted.</td>
</tr>
</tbody>
</table>

Table 15 – VNF Package Notification Consumer Interface operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptVnfPackageOnBoardingNotification</td>
<td>This method is called when a new VNF Package is on-boarded, after all the steps are done.</td>
</tr>
<tr>
<td>acceptVnfPackageChangeNotification</td>
<td>This method is called when there is a change of state in an on-boarded VNF Package. Only a change in operational state will be reported and deletion pending attribute. A change in usage state will not be reported.</td>
</tr>
</tbody>
</table>

Notifications’ consumer can be also external with respect to the Catalogue framework. In the Catalogue’s next release further notification types will be implemented, in particular, the notification mechanism will be extended also to dispatch information about application level monitoring in NSs as well as in Apps and VNFs. App/VNF Packages, leveraging on the SELFNET package structure, will contain also Monitoring Descriptors that detail which kind of application-specific metrics can be monitored. The 5G-MEDIA MAPE will be the consumer of such information, in order to start monitoring application’s metrics along with NFVI’s metrics once the service is instantiated.

6.4.3. Open Source MANO Plugin

Figure 22 depicts the MANO Plugin class diagram. In the 5G App and Service Catalogue implementation, a MANOPlugin is an abstract class containing basic MANO-related information, such as the manoType (e.g. OSM, Taker etc.) and the KafkaConnector, which allows the plugin to act as a consumer and producer on the catalogue message bus. The MANOPlugin contains also a MANO abstract object that includes the manoid and is intended to be extended with MANO-specific information to be persisted in the MANOREpository. The reason for persisting MANO information is related first to the adaptability and modularity of the Catalogue’s architecture, where MANO Plugins will be instantiated and terminated.
dynamically, then to the resiliency of the system: store the MANO information allows to re-
 instantiate the plugins, for instance, in case of failures. The MANOPlugin implements the
 NsdNotificationsConsumerInterface and the VnfPackageNotificationsConsumerInterface as
 specified in section [6.4.2]

The OpenSourceMANOPlugin, represented in Figure 23, extends the MANOPlugin class
 providing a concrete implementation of the notifications interfaces. Moreover, the
 OpenSourceMANOPlugin includes an OSMMano field, which is an extension of the above-
 mentioned MANO. The OSMMano contains all the needed access information for operating
 on top of the OSM SO via NB ReST APIs. In particular, the onboarding in the OSM inner
catalogue is performed through the use of the J-OSMClient library. J-OSMClient is a java
library, released under the BSD 2-Clause License, which is integrated in the Catalogue as a
Maven application dependency and implements a Java client capable of handling NSD and VNF
Package management operations on top of OSM R3.3. The OpenSourceMANOPlugin initial

Figure 22 – 5G App and Service Catalogue MANO Plugin class diagram

Figure 23 – 5G App and Service Catalogue MANO Plugin class diagram

\[\text{https://github.com/girtel/J-OSMClient} \]
implementation includes also a first OSMModelTranslator prototype acting as a Spring Boot Service and capable of translating a SOL001 TOSCA NSD/VNFD in the OSM expected format. The implementation of VIM Plugins (i.e. targeting OpenStack, OpenNebula and OpenWhisk) is foreseen for the Catalogue next release, the OpenStack Plugin is under development and its design principles are detailed in the following section.

Figure 23 – 5G App and Service Catalogue OSM Plugin class diagram

6.4.3.1. Open Stack Plugin

Inside a MANOPlugin, the VIM Plugin is the module in charge of uploading to the targeted VIM’s repository App/VNF’s images, if they are present between the package’s artifacts or if they are referred via URL in the VNFD VDU’s properties (i.e. the swImage element). In the first case, images included in the package received at Catalogue NBI will be stored in a dedicated image repository along with all the correlated information, while, in the second case, images are already stored in a repository functionally bounded with the Catalogue.
The OpenStackVIMPlugin will be implemented as an extension of an abstract VIMPlugin, adding all the needed information for operating via ReST APIs on top of the OS Glance images repository. Referring to the OpenStack APIs documentation v2\(^3\) (see D6.1 for further details about OpenStack deployment in 5G-MEDIA testbeds), two different mechanisms exist to upload images: *glance-direct* and *web-download*.

In both cases, before uploading the image, an image record has to be created using the **Create Image API**. This request returns an image unique *id* to be used in each of the following operations related to this image, moreover, in a cloud with the interoperable image import enabled, the response will contain also the available import methods.

- **glance-direct**, in this case the workflow comprises three steps:
  1. create an image record
  2. use the **Image Stage API** for uploading the image to a stage area
  3. use the **Image Import API** to make the image accessible, specifying the use of the glance-direct method

- **web-download**:
  1. create an image record
  2. use the **Image Import API** specifying the use of the web-download method. In particular, in this case the request payload will include an URI field containing the location of the image (see *Figure 24*

```json
{
   "method": {
      "name": "web-download",
      "url": "https://download.cirros-cloud.net/0.4.0/cirros-0.4.0-ppc64le-disk.img"
   }
}
```

*Figure 24 – Image Import API payload example, web-download case*

In the case of the OpenStackVIMPlugin we select the **web-download** workflow, more suitable for the Catalogue’s purpose, since, as mentioned before, the App/VNF’s images will be stored in dedicated repositories. In terms of implementation, the OpenStack4j\(^4\) Java library, released under the Apache 2.0 License, will be use as a client for the OS Glance repository, in particular using the v2 of the implemented APIs in order to be compliant with the OpenStack versions deployed in the 5G-MEDIA testbeds.

---

\(^3\) [https://developer.openstack.org/api-ref/image/v2/index.html#image-data](https://developer.openstack.org/api-ref/image/v2/index.html#image-data)

\(^4\) [http://www.openstack4j.com/learn/image-v2](http://www.openstack4j.com/learn/image-v2)
7. Media and Network functions repository

This section provides the repository of applications that will be stored in the 5G App and Service Catalogue. The Generic Network Functions are those that are not strictly media related, and they will be used for composing and deploying media services on top of the 5G-MEDIA SVP. The Media Network Functions are specific media packages that facilitates media applications within the 5G-MEDIA framework.

All these applications could be implemented and then deployed as VNF on an NFVI, in order to bring flexibility in terms of deployment, upgradability and functionality and allows for more dynamic media applications and scenarios. Other option is that the applications are developed as PNFs, which can be either hardware solutions providing specific functionality (i.e. broadcaster’s transmission equipment like en-/decoders and gateways) or they can be functions that are virtual but not dynamically provisioned as the VNFs mentioned before, but already running in the network.

7.2. Generic Network Functions

The following table reports the initial set of Generic Network Functions along with their description and requirements in terms of Virtual Deployment Units (VDUs).

<table>
<thead>
<tr>
<th>VNF/App</th>
<th>DETAILS</th>
</tr>
</thead>
</table>
| vFirewall | NF name: NETW_APP_001_FW  
Short description of functionalities: Security Front/Back End VNFs to protect users and service providers data  
VNF component name: NETW_APP_001_FW_01  
• Memory: 1 GB  
• CPU: 1 vCPU  
• Storage: 1 GB  
• Image: vFirewall-v01.qcow2  
Connectivity specifications:  
• VLs: mgmt_net, ext_net, data_net  
Provider: NXW  
Status: READY |
| IDS/DPI | VNF name: NETW_APP_002_IDS_DPI  
Short description of functionalities: Security Front/Back End VNFs to protect users and service providers data  
VNF component name: NETW_APP_002_IDS_DPI_01  
• Memory: 8 GB  
• CPU: 4 vCPUs  
• Storage: 20 GB  
• Image: vIDS-v01.qcow2 |
### 7.2.2. Generic VNFs and PNFs descriptions

#### 7.2.2.1. vFirewall

The vFirewall network function is based on VyOS, an open source GNU/Linux-based operating system extended with network routing and firewall software suitable for being deployed in VNFs in the form of Virtual Machines (VMs). VyOS VM can be installed in various hypervisors such as KVM, Xen in HVM mode, VMware, Microsoft Hyper-V, and VirtualBox and has a small footprint in terms of disk and memory usage (i.e. typically 1GB of memory and 1GB of disk).

In more details, VyOS provides several network functionalities among which it is worth mentioning:

- **Static and dynamic routing**: BGP for IPv4 and IPv6, OSPFv2, RIP, RIPng, policy-based routing, equal cost multi-path,
- **NAT**: source NAT, destination NAT (port forwarding), one to one, one to many, and many to many translations,
- **Firewall**: assignment of Firewall rule sets for IPv4 and IPv6 to specific interfaces, zone-based firewall, address/network/port groups for IPv4

VyOS comes with a command line interface that can be used for actuating configurations of the different functionalities. Configuration commands do not change the running

---

**NOTE**: [https://vyos.io/](https://vyos.io/)
configuration immediately: in fact, changes are at first staged and then committed, thus allowing to track users’ modifications for router maintenance and configuration rollback. Configurations are visualized in a simple human-readable format and can be committed in the confirmed mode. Staged modification which are not committed are automatically reverted after a specified amount of time. After every commit the previous configuration version is archived and versioned.

Typically, the vFirewall VNF is deployed with 1 interface (mapped as eth0) and used for management purposes plus as many interfaces as needed on the base of the service topology. In the current VNF implementation, the day0 configuration is done via the `start-config/config/config.boot`, which is executed after `successfullmboot` of the instance, while the day1 configuration is done via script.

The configuration script is embedded in the VNF base image and is executed for configuring the Firewall functionality when the day0 configuration is terminated. In the case of the OSM NFVO, the script can be executed via `cloud-init` and/or `proxy-charms`.

7.2.2.2. vIDS/DPI

The vIDS/DPI network function is based on the SELFNET virtual Intrusion Detection System (vIDS) VNF, which analyses network traffic in real-time and reports alerts when packets content matches a specific detection rule (called signature). This VNF has been developed in the SELFNET 5G-PPP Phase 1 EU project as part of the Self-Protection Use Case activities.

The SELFNET vIDS VNF [8] acts as a Deep Packet Inspection tool, and for this it encapsulates Snort, a leading open source network-based IDS that offers a detection engine based on regular expressions and aims at finding matches between network packets and signatures configured as rule sets. In addition, to content matching at the level of network packets (headers and payloads), this VNF can also make protocol analysis and inspections at the TCP/IP headers level to detect a wide selection of cyber-attacks and probes, such as buffer overflows, stealth port scans and TCP floods.

The SELFNET vIDS VNF produces alerts whenever a matching between a packet content (mainly its payload) and one of the detection rules set in the DPI engine occurs. The SELFNET vIDS VNF internally stores the alerts in dedicated files created with the following pattern:

```
/var/log/snort/snort.log.xxx
```

These alerts are stored in `tcpdump` format, so that their content can be shown by executing the “`tcpdump -r`” command. Moreover, a textual representation of the alerts produced by the DPI engine is also stored in the file:

```
/var/log/snort/alert
```

---

46 [https://5g-ppp.eu/selfnet/](https://5g-ppp.eu/selfnet/)
47 [https://www.snort.org/](https://www.snort.org/)
which collects the whole set of events and anomalies detected by the DPI engine in a human readable format.

The SELFNET vIDS VNF has also the capability to export the produced alerts by sending them over a Kafka bus. In particular, the alerts internally stored by the VNF (following the approach described above) are translated into the Intrusion Detection Message Exchange Format (IDMEF) [13] and then posted on a previously configured Kafka message bus. IDMEF is the standard format mostly used by IDS to exchange alerts and information with other network management tools, as it allows interoperability, flexibility, and independence of the communication protocol through the use of XML formats. The Kafka server endpoint, and the topics to be used for posting alerts messages, are assumed to be configured in the SELFNET vIDS VNF instance during the instantiation, e.g. at Day0/1 configuration stage via cloud-init.

The runtime configuration (i.e. Day-2) of the SELFNET vIDS VNF allows managing the detection rules of its DPI engine. For this, the VNF embeds a REST server to receive direct configuration requests from external management entities. This allows adding new detection rules to the DPI engine directly, e.g. coming from any NFV MANO tool or component.

In Table 17 the REST API exposed by the VNF is specified. Two major operations are offered: the configuration of new detection rules and the deletion of existing rules.

<table>
<thead>
<tr>
<th>HTTP Method &amp; REST Endpoint</th>
<th>Request body</th>
<th>Description</th>
</tr>
</thead>
</table>
| POST /vids/snort/signatures | `{ "sourceIP": <src-ip>, "destinationIP": <d-ip>, "destinationPort": <d-port>, "rule": <rule-params>, "sid": <sid> }` | This POST operation is used to apply a new detection rule (signature) in the Snort DPI engine. The request body include attributes as follows:
  - `sourceIP`: the source IP address of the network flow to inspect
  - `destinationIP`: the IP address of the network flow to inspect
  - `destinationPort`: the port used at the destination server for communications the given network flow under inspection
  - `rule`: the expression of the low-level detection pattern that the DPI engine has to match in the packets’ payloads.
  - `sid`: unique identifier of the detection rule (i.e. signature), assigned by the caller. |

| DELETE /vids/snort/signatures/<sid> | None | This DELETE operation is used to remove a previously set detection rule in the DPI engine. The request include attributes and parameters as follows:
  - `sid`: identifier of the detection rule to remove |
7.2.2.3. vDNS

The vDNS network function is based on the BIND\textsuperscript{48} software, an open source DNS software which implements the DNS protocol on GNU/Linux-based operating system.

BIND is the most used software for implementing DNS systems in the Internet. It is functionally composed of

- A \textit{Domain Name Resolver}, which is responsible to resolve URL translations to IP and viceversa, by querying a local name cache or forwarding the request to configured Name Servers
- A \textit{Domain Name Authority server}, which answers in authoritative mode requests from resolvers

The configuration of the DNS service based on BIND follows the common best practices for configuring name servers (cf. The BIND 9 Administrator Reference Manuals, available at \url{https://www.isc.org/downloads/bind/doc/}). More specifically, in Ubuntu 16.04 Linux distributions the main configuration elements of the service are available at

- `/etc/bind` for general service and application configuration parameters contained in the configuration files:
  - `named.conf`
  - `named.conf.default-zones`
  - `named.conf.local`
  - `named.conf.options`

- `/var/lib/bind` in which the specific zones files are available
  - `ZONE-X-NAME.zone`, for forward translations from URL to IP
  - `ZONE-X-NAME.rev.zone`, for reverse translations from IP to URL

In 5G-MEDIA, the vDNS is a Physical Network Function (PNF) which is pre-instantiated in the NFVI infrastructure and made reachable on the networks used by the VNFs to communicate among them for application floes (i.e. no DNS is configured for the control traffic, though the configuration could be possible).

To allow a dynamic configuration and update of the records in the vDNS PNF a REST interface is exposed by the VNF. An initial version of this API is described in \textbf{Table 18}

```bash
# curl -X DELETE -H 'Content-Type: application/json' -H 'X-API-Key: secret' -d '{ "hostname": "host.example.com"}' \url{http://localhost:9999/dns}
```

\textsuperscript{48} \url{https://www.isc.org/downloads/bind/}
Table 18 – vDNS REST API

<table>
<thead>
<tr>
<th>HTTP Method &amp; REST Endpoint</th>
<th>Request body</th>
<th>Description</th>
</tr>
</thead>
</table>
| POST /dns                   | {            | This POST operation is used to apply a new A record to the zone file for a given DOMAIN. The request body include attributes as follows:
|                             | "hostname": | • hostname: the FQDN of the host to be mapped
|                             | <host.domain>, | • ip: the IP address to map to the URL
|                             | "ip": <IP address>> |          |
| DELETE /dns                | {            | This DELETE operation is used to remove a previously set FQDN for a host. The request body include attributes as follows:
|                             | "hostname": | • hostname: the FQDN of the host to be removed
|                             | <host.domain> | |

7.3. Media Network Functions

The following table reports the initial list of Media Network Functions along with their description and requirements in terms of Virtual Deployment Units (VDUs).

Table 19 – Media specific apps technical requirements

<table>
<thead>
<tr>
<th>VNF/App</th>
<th>DETAILS</th>
</tr>
</thead>
</table>
| vCache        | VNF name: MEDIA_APP_001_CACHE  
Short description of functionalities: Cache server where the user is connected to, can be deployed according to a hierarchy of mid/edge caches orchestrated/managed by the vCDN. In order to serve the media content near to the end user, reducing latency and offering a better QoS/QoE.  
VNF component name: MEDIAGEN_APP_001_CACHE_01  
• Memory: 8 GB  
• CPU: 4 vCPUs  
• Storage: 40 GB  
• Image: vCache.qcow2  
Connectivity specifications:  
• VLS: mgmt_net, ext_net, data_net  
Provider: NXW  
Status: READY |
| vCompression Engine | VNF name: GEN_FUN_002_COMPRESSION  
Short description of functionalities: Virtual media function for decompression/encoding and encoding/decoding of audio-visual media content.  
VNF component name: GEN_FUN_002_COMPRESSION_01  
• Memory: 8 GB |
### Media Process Engine

**VNF name:** MEDIA_APP_003_MPE  
**Short description of functionalities:** A media function that performs the switching and mixing of audio/video signals. It serves as a virtual broadcast video/vision switcher.  
**VNF component name:** MEDIA_APP_003_MPE_01  
- **Memory:** 8 GB  
- **CPU:** 8 vCPUs  
- **Storage:** 10 GB  
- **Image:** MPEv1.qcow2  
**Connectivity specifications:**  
- **VLs:** mgmt_net, ext_net, data_net  
**Provider:** IRT  
**Status:** WORK IN PROGRESS

### Speech-to-Text Engine

**VNF name:** MEDIA_FUN_004_SPEECH  
**Short description of functionalities:** A media function that allows for the recognition and analysis of the audio/video material’s audio signal which will be decoded into text.  
**VNF component name:** MEDIA_FUN_004_SPEECH_01  
- **Memory:** 2 GB  
- **CPU:** 2 vCPUs  
- **Storage:** 2 GB  
- **Image:** vspeech.img (qcow2)  
**Connectivity specifications:**  
- **VLs:** mgmt_net, ext_net, data_net  
**Provider:** BIT  
**Status:** WORK IN PROGRESS

### Demonstrator

**VNF name:** MEDIA_FUN_005_Demonstrator  
**Short description of functionalities:** A media function that receives an A/V stream and displays that stream with minimal latency in a browser-based user interface. In addition, it can interact with the Speech-to-Text Engine in such a way as to provide a subtitle overlay to the A/V output stream.  
**VNF component name:** MEDIA_FUN_005_Demonstrator_01  
- **Memory:** 2 GB  
- **CPU:** 2 vCPUs  
- **Storage:** 2 GB  
- **Image:** vplay.img (qcow2)  
**Connectivity specifications:**  
- **VLs:** mgmt_net, ext_net, data_net  
**Provider:** BIT  
**Status:** READY
<table>
<thead>
<tr>
<th>VNF/App</th>
<th>DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>vTranscoder3D</td>
<td><strong>VNF name:</strong> MEDIA_APP_006_TRANSCODER3D</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Performs on-the-fly transcoding of a 3D media stream comprising of time-varying geometry and texture information. Both types of content (2D &amp; 3D) are simultaneously re-encoded.</td>
</tr>
<tr>
<td></td>
<td><strong>VNF component name:</strong> Media_APP_006_TRANSCODER3D_01</td>
</tr>
<tr>
<td></td>
<td>• Memory: 4 GB</td>
</tr>
<tr>
<td></td>
<td>• CPU: 1 vCPU</td>
</tr>
<tr>
<td></td>
<td>• GPU: 1 x Nvidia 1080Ti (actual hw) (for gpu image)</td>
</tr>
<tr>
<td></td>
<td>• Storage: 10 GB</td>
</tr>
<tr>
<td></td>
<td>• Image: vTranscoder3D-cpu, vTranscoder3D-gpu</td>
</tr>
<tr>
<td></td>
<td><strong>Connectivity specifications:</strong></td>
</tr>
<tr>
<td></td>
<td>• VLS: mgmt_net, ext_net, data_net</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> CERTH</td>
</tr>
<tr>
<td></td>
<td><strong>Status:</strong> WORK IN PROGRESS</td>
</tr>
<tr>
<td>5G-MEDIA Gateway</td>
<td><strong>PNF name:</strong> MEDIA_PNF_01_GATEWAY</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Convert the incoming SDI Signals to IP or vice versa</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> IRT (Vendor Solution)</td>
</tr>
<tr>
<td></td>
<td><strong>Status:</strong> READY (Procurement of ST 2110 equipment in progress)</td>
</tr>
<tr>
<td>UHD Streaming Server</td>
<td><strong>PNF name:</strong> MEDIA_PNF_02_UHDSS</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Transmits VoD from media libraries according to user profiling and preferences. Can be eventually replicated and serves as the root for vCache/vTranscoder hierarchies</td>
</tr>
<tr>
<td></td>
<td><strong>VNF component name:</strong> MEDIA_APP_02_UHDSS_01</td>
</tr>
<tr>
<td></td>
<td>• Memory: 8 GB</td>
</tr>
<tr>
<td></td>
<td>• CPU: 4 vCPUs</td>
</tr>
<tr>
<td></td>
<td>• Storage: 40 GB</td>
</tr>
<tr>
<td></td>
<td>• Image: vCache.qcow2</td>
</tr>
<tr>
<td></td>
<td><strong>Connectivity specifications:</strong></td>
</tr>
<tr>
<td></td>
<td>• VLS: mgmt_net, ext_net, data_net</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> NXW</td>
</tr>
<tr>
<td></td>
<td><strong>Status:</strong> READY</td>
</tr>
<tr>
<td>vBuffer</td>
<td><strong>VNF Name:</strong> MEDIA_APP_07_BUFFER</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Sliding window buffer to be used in combination to the vReplay VNF</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> CERTH</td>
</tr>
<tr>
<td></td>
<td><strong>Status:</strong> PLANNED</td>
</tr>
<tr>
<td>Image/Face Recognition</td>
<td><strong>VNF Name:</strong> MEDIA_APP_08_IMRECOGNITION</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Detection of objects within the A/V material with a context-aware text-based output</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> BIT</td>
</tr>
<tr>
<td></td>
<td><strong>Status:</strong> PLANNED</td>
</tr>
<tr>
<td>vReplay</td>
<td><strong>VNF Name:</strong> MEDIA_APP_09_REPLAY</td>
</tr>
<tr>
<td></td>
<td><strong>Short description of functionalities:</strong> Temporal aggregation of recent 3D media frames into a 3D replay clip</td>
</tr>
<tr>
<td></td>
<td><strong>Provider:</strong> CERTH</td>
</tr>
</tbody>
</table>
### VNF/App Details

<table>
<thead>
<tr>
<th>VNF/App</th>
<th>DETAILS</th>
</tr>
</thead>
</table>
| vTranscoder | VNF Name: MEDIA_APP_010_TRANSCODER  
Short description of functionalities: Transcoding unit for media content, transcodes the media content according to the QoS/QoE offered to the end user, can be centralized or deployed at the edge for reducing latency and improving QoS/QoE  
Provider: UPM and NXW  
Status: PLANNED |

#### 7.3.2. Media VNFs and PNFs descriptions

#### 7.3.2.1. vCache

The vCache VNF is responsible to bring the media content closer to the 5G-MEDIA user by caching frequently-accessed information at the edge of the network, then improving the response time. The vCache application is based on the Apache Traffic Server\(^49\) (ATS). ATS is a high-performance web proxy cache that improves network efficiency and performance. It may be configured to run as forward and reverse proxy. The Apache Traffic Server software is released under Apache 2.0 license\(^50\), after the donation made by Yahoo! to the Apache Foundation.

There are different types of configuration for the vCache. The very basic configuration consists in a transparent forward proxy that allows the network administrator to monitor or filter the traffic. The other type of configuration, chosen for the vCache VNF in the 5G-MEDIA project, is the transparent and reverse proxy that forwards all the requests to a single origin server. The origin server may be the real distributor of the media content, but in certain cases, it may be another vCache server. In this last case, a hierarchical level of vCaches will be configured. There is no limit to the number of levels in the hierarchy of vCaches. Furthermore, when more than one level of vCaches are present, the first level must use as origin server a vCache on the second level, called parent. The vCache must be aware that its origin server is not a real origin media server, but a parent cache. These settings will be provided during the configuration phase.

ATS offers also the possibility to inspect the cached media on the current node. This is feasible by configuring the Cache Inspector plugin. This allows the network administrator to inspect and eventually destroy the cache content, by destroying all or a portion of it. The access of the Cache Inspector plugin is limited to a pool of IP addresses, configurable dynamically via REST API.

---


\(^50\) Apache license 2.0: [http://www.apache.org/licenses/LICENSE-2.0](http://www.apache.org/licenses/LICENSE-2.0)
On top of the vCache, a configuration agent was implemented in order to perform runtime a set of configurations. The agent was designed as web service exposing REST APIs and running on startup on each vCache instance. The following list exposes the set of configuration operations available on the vCache with examples of payloads:

### Table 20 – vCache REST APIs

<table>
<thead>
<tr>
<th>HTTP Method &amp; REST Endpoint</th>
<th>Request body</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /</td>
<td></td>
<td>Lists all the available APIs.</td>
</tr>
<tr>
<td>PATCH /vnfconfig/v1/cache_configuration</td>
<td>’anfrom': ‘172.16.0.2’, ‘to’: ‘172.16.0.253’</td>
<td>Modifies the configuration of the vCache inspector plugin, setting up the IP range for accessing the cache content.</td>
</tr>
<tr>
<td>PATCH /vnfconfig/v1/cache_origin_configuration</td>
<td>‘ip’: ‘10.0.8.39’, ‘port’: 32400</td>
<td>Modifies the configuration of the cache related to the origin server setting up its IP address and port. This configuration is not applicable to hierarchical vCache levels or the first level of the vCache.</td>
</tr>
<tr>
<td>PATCH /vnfconfig/v1/cache_edge_origin_configuration</td>
<td>‘ip’: ‘10.0.4.250’, ‘port’: 8080</td>
<td>Modifies the configuration of the caches from the second level in the hierarchy, i.e. configures the vCaches with parent vCache IP address and port.</td>
</tr>
</tbody>
</table>

### 7.3.2.2. vCompression Engine

The vCompression Engine is responsible for compression/decompression and encoding/decoding of the audio/video content before and after the WAN transfer. It is based on open source encoding techniques and aims at using the latest video standards such as SMPTE 2110 and H.264/H.265. Depending on the coding parameters it requires high computational power and is an ambitious software-only approach. Typically, hardware encoders/decoders are used today to fulfil this task. So, a virtual software-only approach will boost flexibility and agility in remote production.
Design Basis

As the software basis for the vCompression Engine FFmpeg is chosen. FFmpeg is a very popular multimedia framework which allows for encoding, decoding, transcoding, multiplexing, demultiplexing, streaming, filtering and playing of almost all media contents. FFmpeg includes libavcodec, an audio/video codec library used by many commercial and free software products, libavformat (Lavf), an audio/video container mux and demux library, and the core FFmpeg command line program for transcoding multimedia files. FFmpeg is published under the GNU Lesser General Public License 2.1+ or GNU General Public License 2+.

FFmpeg ST 2110 Extension

The Canadian Broadcasting Corporation CBC which is Canada’s national public broadcaster joined forces with Savoir-faire Linux to enable FFmpeg to process high-bitrate media data like uncompressed HD streams. The aim is to support the SMPTE standard ST 2110 both for sending and receiving streams. The former is yet to be done, the latter has already been achieved by CBC in a controlled lab environment with artificial sources (e.i. GStreamer).

For the vCompression Engine to work correctly the exchange of an SDP (Session Description Protocol, RFC 4566) file from the sender is indispensable. In the SDP file essential information/parameters about the stream/session are listed. The receiver uses these information to correctly interpret the stream/session.

Supported Standards:

- RFC 4175 for YCbCr-4:2:2, 8-bit and 10-bit
- AES67

Transcoding dependencies:

- x264
- faac
- yasm

---

51 https://ffmpeg.org
52 https://github.com/cbcrc/FFmpeg
53 https://gstreamer.freedesktop.org/
Simple audio video workflow

![Diagram of a simple audio video workflow]

Figure 25 – simple audio video workflow

Example SDP File

Table 21 – Example SDP File

```plaintext
# session
v=0
o=- 123456 1 IN IP4 X.X.X.X
s=A simple SMPTE ST 2110 session
i=basic streams for audio video
a=recvonly

# timing: unbounded and permanent session
t=0 0

# audio description first for better results
m=audio 10857 RTP/AVP 97
c=IN IP4 224.0.1.129/4
a=rtpmap:97 L24/48000/2

# video description
m=video 10855 RTP/AVP 96
c=IN IP4 224.0.1.129/4
a=rtpmap:96 raw/90000
a=fmtp:96 sampling=YCbCr-4:2:2; width=1920; height=1080;
depth=10; colorimetry=BT.709
```
Example Console Input/FFmpeg Call

**Table 22 – Example Console Input/FFmpeg Call**

```
ffmpeg -strict experimental -buffer_size 300000 -i
test.sdp -fifo_size 10000000 -c:a libfdk_aac -c:v libx264
-pass 1 -f mpegts udp://127.0.0.1:5000
```

Goals

The aim is to receive an uncompressed media stream according to ST 2110 from the media gateways, inside the vCompression Engine decapsulate the audio video data and apply a video encoding standard such as H.264 to limit the required network bandwidth while keeping the best possible video and audio quality. At the receiving side it is intended to implement another vCompression Engine to decode/decompress the signals back to uncompressed signals and encapsulate them to ST 2110 again for further usage inside the broadcaster’s facilities. These processing tasks are done in an agile virtual environment within the 5G-MEDIA network.

IRTs specific goals are:

- adapt the solution to real world scenarios and get it working with real audio video/broadcast equipment
- adapt it for the 5G-MEDIA platform/network (implementation as a VNF
- develop and implement the encoder for ST 2110

7.3.2.3. Media Process Engine

The Media Process Engine is a media-specific function, which acts, in this first version, as a mere video signal switcher, based on the OpenSource tools Voctomix and GStreamer. Concretely, the MPE VNF is composed by voctocore, the processing part of the Voctomix solution. This processing core has python-based code that allows to switch between different input streamings, and to compose a new video streaming, by composing the input together with a background.

As the voctocore is python based, python libraries have been installed in the VM of the VNF. Apart from this, FFmpeg is also installed, as well as the desired libraries, for taking the video in its correct format, in our case H264 library. Finally, GStreamer libraries are also installed.

The running voctocore stays listening for input signals, as well as control commands. The ports specification is as follows:

55 https://github.com/voc/voctomix
56 https://gstreamer.freedesktop.org/
Ports that will accept Raw UYVY Frames and Raw S16LE Audio in a Matroska container:

- 10000, 10001, … – Main Video-Sources, depending on the number of configured Sources

Ports that will accept Raw UYVY Frames without Audio in a Matroska container:

- 16000 Mixer – Background Loop

Ports that will accept Raw S16LE Audio without Video in a Matroska container:

- 18000 – Stream-Blanker Audio-Input

Ports that will provide Raw UYVY Frames and Raw S16LE Audio in a Matroska container:

- 13000, 13001, … – Main Video-Source Mirrors, depending on the number of configured Sources
- 11000 – Main Mixer Output

Ports that will provide JPEG Frames and Raw S16LE Audio in a Matroska container – only when [previews] enabled=true is configured in a configuration file:

- 14000, 14001, … – Main Video-Source Mirrors, depending on the number of configured Sources
- 12000 – Main Mixer Output

Port 9999 will Accept Control Protocol Connections.

Over this 9999 port, TCP line-based commands are transported to control the core with simple set and get commands. These commands are listed in Table 23:

Table 23 – MPE processing core control commands

<table>
<thead>
<tr>
<th>#</th>
<th>method</th>
<th>header</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET</td>
<td>get_video(self)</td>
<td>Gets the current video-status, consisting of the name of video-source A and video-source B</td>
</tr>
<tr>
<td>2</td>
<td>SET</td>
<td>set_video_a(self, src_name)</td>
<td>Sets the video-source A to the supplied source-name or source-id, swapping A and B if the supplied source is currently used as video-source B</td>
</tr>
<tr>
<td>3</td>
<td>SET</td>
<td>set_video_b(self, src_name)</td>
<td>Sets the video-source B to the supplied source-name or source-id, swapping A and B if the supplied source is currently used as video-source A</td>
</tr>
<tr>
<td>4</td>
<td>GET</td>
<td>get_audio(self)</td>
<td>Gets the current volumes of the audio-sources</td>
</tr>
<tr>
<td>#</td>
<td>method</td>
<td>header</td>
<td>description</td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>5</td>
<td>SET</td>
<td>set_audio(self, src_name)</td>
<td>Sets the audio-source to the supplied source-name or source-id</td>
</tr>
<tr>
<td>6</td>
<td>SET</td>
<td>set_audio_volume(self, src_name, volume)</td>
<td>Sets the volume of the supplied source-name or source-id</td>
</tr>
<tr>
<td>7</td>
<td>GET</td>
<td>get_composite_mode(self)</td>
<td>Gets the name of the current composite mode</td>
</tr>
<tr>
<td>8</td>
<td>GET</td>
<td>get_composite_modes(self)</td>
<td>Lists the names of all available composite-modes</td>
</tr>
<tr>
<td>9</td>
<td>GET</td>
<td>get_composite_mode_and_video_status(self)</td>
<td>Retrieves the composite-mode and the video-status</td>
</tr>
<tr>
<td>10</td>
<td>SET</td>
<td>set_composite_mode(self, mode_name)</td>
<td>Sets the name of the id of the composite-mode</td>
</tr>
<tr>
<td>11</td>
<td>SET</td>
<td>set_videos_and_composite(self, src_a_name, src_b_name, mode_name)</td>
<td>Sets the A- and the B-source synchronously with the composition-mode</td>
</tr>
<tr>
<td>12</td>
<td>GET</td>
<td>get_config(self)</td>
<td>Returns the parsed server-config</td>
</tr>
<tr>
<td>13</td>
<td>GET</td>
<td>get_config_option(self, section, key)</td>
<td>Returns a single value from the server-config</td>
</tr>
<tr>
<td>14</td>
<td>POST</td>
<td>restart_source(self, src_name)</td>
<td>Restarts the specified source</td>
</tr>
</tbody>
</table>

When a Command was invalid or had invalid Parameters, the Server responds with error followed by a Human Readable error message.

### 7.3.2.4. Speech-to-Text Engine

The Speech-to-Text Engine is a media-specific function that recognizes speech within the audio signal of any audio video material which is streamed via RTP, RTMP or RTSP protocol. As a result, this service provides a text derived from the recognized speech. In the following the basic design and its parts will be represented.
Design

The Speech-to-Text function is built up into 6 different parts, which are chained together and where each fulfills a particular task (see [PICTURE]). These are:

- **LS**: Describes the input source, which can be an RTP, RTMP or RTSP Stream. Both UDP and TCP sockets are supported by his service.
- **AX**: The Audio Extractor extracts the audio channel from the video stream.
- **AC**: The audio channel has to be encoded by the Audio Converter to fit the needs of the next sub process.
- **IT**: The Inference Task is the main task, which is responsible for converting the speech into text.
- **OI**: To access the result of the Inference Task 3 different interfaces will be exposed by this service, which based on TCP sockets. By name these are:
  - WebSockets: For real-time transmission.
  - HTTP-REST-API: Stateless CRUD-API to deliver results, configuration, and monitoring information.
  - UI: A user interface especially for demonstration purposes and easy configuration, which uses the REST-API
  - Webhooks: A real-time capable interface, which defines an URI endpoint where to push the results.
- **MON**: The monitor allows for reading the current status of the service and can be accessed by the OI.

Configuration

To configure the function it needs at least the URL of the input stream. Supported streams are RTP, RTMP and RTSP. Optional parameters concerns the service itself such as the port where the service should run, the runtime mode (development, production) or the language which will be used by the Inference Task. For the Google Speech API based Speech-to-Text function additionally the google application credentials are mandatory for proper execution. The following list shows all relevant parameters:
• **SOURCE_STREAM_URL**: URL of the RTP/RTMP/RTSP video live stream  
• **WEBHOOKS_URL**: URL of the remote host or service which is intended to receive the real-time result of the Speech-to-Text Engine  
• **PORT**: port on which the web service is running (default: 3000)  
• **LANGUAGE**: language of the video live stream (default: en-US)  
• **VALIDATION_DURATION**: duration in ms how long the credentials will be valid (default: 60.000)  
• **GOOGLE_APPLICATION_CREDENTIALS**: google application credentials  
• **NODE_ENV**: runtime environment (default: dev)

The Speech-to-Text function can be either configured via HTTP-REST-API or during instantiation via environmental variable injection. That means to configure the function by defining a specific file named .env or by using build-in commands which are provided by the packaging technology (i.e. --e in case of Docker). Another option is to access the function via the user interface, which facilitates the configuration via REST-API.

### Monitoring

The function’s state machine presupposes to have states defined in the function, which will in particular be consumed by the monitoring part of the function itself or by other functions or services, since the monitoring is exposed to the interfaces Websockets, Webhooks and as one of the resources of the REST-API.

- **stop**: Indicates the transition of one state into the state where the Speech-to-Text Inference Task is stopped (value: 1)  
- **start**: Indicates the transition of one state into the state where the Speech-to-Text Inference Task is started (value: 2)  
- **started**: State in which the Speech-to-Text Inference Task is started (value: 3)  
- **stopped**: State in which the Speech-to-Text Inference Task is started (value 4)  
- **timeout**: Either no stream is recognized or the duration of the credentials timed out (value: 5)  
- **restart**: If no stream was detected or any other error has occurred the Speech-to-Text service will be restarted (value: 6)

### Interfaces

The REST-API, WebSocket and Webhooks interfaces are provided to configure, monitor and access the results of the Speech-to-Text inference. While the Webhooks and REST-API are based on the Hypertext Transfer Protocol (HTTP), WebSocket uses the TCP network protocol to establish bidirectional real-time data transmission between the Speech-to-Text Engine and other enhanced media services. As another option the Webhooks can be used as a real-time push interface. Once the URL of a remote host or service is configured on the Speech-to-Text Engine, the results as well as the monitoring data will be transferred remotely.
Table 24 – Speech-to-text API

<table>
<thead>
<tr>
<th>#</th>
<th>HTTP method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET</td>
<td>/api</td>
<td>Finds all data</td>
</tr>
<tr>
<td>2</td>
<td>GET</td>
<td>/api/transcription</td>
<td>Finds transcription</td>
</tr>
<tr>
<td>3</td>
<td>POST</td>
<td>/api/transcription/start</td>
<td>Starts transcription</td>
</tr>
<tr>
<td>4</td>
<td>POST</td>
<td>/api/transcription/stop</td>
<td>Stops transcription</td>
</tr>
<tr>
<td>5</td>
<td>GET</td>
<td>/api/transcription/state</td>
<td>Finds state of transcription</td>
</tr>
<tr>
<td>6</td>
<td>GET</td>
<td>/api/words</td>
<td>Finds words</td>
</tr>
<tr>
<td>7</td>
<td>GET</td>
<td>/api/statistics</td>
<td>Finds statistics</td>
</tr>
<tr>
<td>8</td>
<td>GET</td>
<td>/api/configuration</td>
<td>Finds configuration</td>
</tr>
<tr>
<td>9</td>
<td>POST</td>
<td>/api/configuration</td>
<td>Updates configuration</td>
</tr>
</tbody>
</table>

Packaging

The Speech-to-Text Engine is packaged to the following three formats:

- Docker: Build script included
- Virtual Machine Image: Start/Stop scripts included

Unikernel: The Speech-to-Text Engine was split in two parts, one low level container/vm with FFmpeg dependencies and one higher level vm packaged by unikernel
7.3.2.5. Demonstrator

The Demonstrator is a self-contained media VNF function that receives an A/V stream regardless of the transport protocol (UDP, TCP, RTP, etc.) and displays that stream with minimal latency in a browser-based user interface. In addition, it can interact with the Speech-to-Text Engine in such a way as to provide a subtitle overlay to the A/V output stream. In this context, all data is passed from the origin Speech-to-Text function to the Demonstrator via WebSocket. Conversely, the Demonstrator relays all requests to the origin Speech-to-Text function essentially via HTTP based RESTful API.

Since the Demonstrator is based on a browser-based user interface, the output stream must be adapted for a reliable presentation. This is mainly because it cannot be assumed that the output stream matches or is compatible with the HTML media formats supported by the browsers, which are H.264, VP8, VP9 Theora or Streaming WebM. For this reason, a design consisting of FFmpeg and FFserver was chosen for the presentation of the A/V stream. Accordingly, the A/V stream is received by FFmpeg and finally forwarded directly to the streaming server FFserver without any encoding or filtering. FFserver acts as an HTTP server, accepting POST requests from FFmpeg to acquire the stream to publish and serving HTTP clients GET requests with the data of the A/V output stream. The configuration of the server is done via a configuration file, which is read out once when the streaming server is started. The resulting live feed can subsequently be played back via web-based video player. Looking closer, you can see that the live feed corresponds to a flash stream. This output format has been deliberately chosen, as it can partly bypass the internal buffer or cache strategies of the browsers, but also to ensure the widest possible support of browsers.

7.3.2.6. vTranscoder 3D

The requirements in terms of networking for 3D media streaming applications are challenging as the bandwidth that is required for a single 3D media stream is typically very high (~15-25 Mbps). In addition, the latency requirements are varying according to each application, but nonetheless, most of the envisaged applications are expected to also require interaction – in order to capitalize on the 3D nature of the media – therefore being further restricted by low latency requirements. These are the result of:

i) a lack of standardization for 3D codecs (making the 3D payload inefficient),
ii) the amount of content that needs to be transmitted (geometry, connectivity, attributes and at least 4 textures),
iii) the time-varying nature of the media (rendering inter-frame 3D codec schemes is computationally inefficient), and
iv) the low processing timing constraints to achieve interactivity (further imposing computational restrictions to the coding of the textures – i.e. resorting to image instead of video coding).

A 3D media transcoding function will greatly help in delivering real-time 3D media streaming services as it will:
i) exploit the computational resources of the network / cloud instead of the end-users’ ones,

ii) adapt the content to the presentation / display capabilities (i.e. resolution, type - screen, VR or AR headset, device type – pc, mobile or tablet, preferences, etc.) of each consumer,

iii) re-encode content in multiple quality levels to allow for adaptive streaming and ensure acceptable QoE levels for the service consumers, and

iv) minimize traffic flow as a result of its (re-)placement within the network.

Full live tele-immersive 3D media transcoding is a novel concept that has not been explored in either academic literature or actual business. The main reason for this is mainly that the technology behind real-time 3D acquisition has also recently emerged, but following the overall mixed reality related developments, is expected to start reaching the consumer base in the following years. While adaptive streaming solutions for stereoscopic 3D video, multi-view content and on demand 3D content have been explored, there are little developments for the real-time, full 3D setting, which is a lot more complex. This complexity arises from:

i) its multimedia nature (combining both 3D & 2D media),

ii) the unexplored underlying QoE model (comprising of 3D & 2D production and encoding settings),

iii) the lack of real-time oriented 3D codecs suited for attribute accompanied 3D content that also exploit temporal information redundancy,

iv) the high bandwidth – low latency contradiction (interactive 3D media applications), and

v) the stringent processing requirements for this type of multimedia content.

**Design:**

In terms of the transcoder’s design, there are two main design choices that need to be taken. First, it is necessary to select the codecs that will be used which in the 3D media case involve two different codec types, 2D (i.e. image/video) and 3D (i.e. geometry and attributes). Second, the qualities / profiles that the transcoder will output need to be defined. The latter choice is dependent on the underlying QoE model. Given that tele-immersive media are relatively unexplored in terms of QoE, two studies were conducted:

1. A preliminary exploration of the complete 3D media streaming QoE model from the perspective of a spectating user (i.e. a client that only receives live content and does not participate in any live interactions).
2. A study to assess the relative importance of the two sub-media types (2D & 3D) in the overall QoE.

For both of these studies, two production parameters are also evaluated, the texture and geometry resolution. Given the live nature of the media, these are important as they influence both the final visual quality (i.e. fidelity of the rendering), as well as the streaming capacity (combination of bandwidth and processing time) of each 3D media production station. The first study showcased the complexity of the 3D media domain and the difficulty of disambiguating between the different parameters (production and networking), while the second one led to identifying the golden section between the texture and geometry resolution.
parameters. These results were published in Transactions on Broadcasting and the International Conference on Quality of Multimedia Experience respectively, with the details available online at [http://www.5gmedia.eu/outcomes/publication/](http://www.5gmedia.eu/outcomes/publication/) in entries “Quality of Experience for 3-D Immersive Media Streaming”, and “Subjective quality assessment of textured human full-body 3D-reconstructions” respectively.

Additionally, given the lack of standardization with respect to 3D codecs, the performance of a set of open source codecs was benchmarked and evaluated. This performance assessment was realized in the context of tele-immersive media streaming. More specifically, MPEG’s O3DGC[^57], Google’s Draco[^58] and an open source compression library oriented for speed as a product of academic research[^59]. The baseline used was OpenCTM[^60] that was modified to use a faster entropy codec (LZ4[^61] instead of LZMA[^62]). The latter study compared the performance of all the aforementioned 3D mesh codecs in rate-distortion terms as well as their runtime complexity that affect end-to-end latency. Since some codecs (or codec configurations) are optimized either for speed or output bitrate, the optimum codec choice depends on actual network conditions. A network streaming case study revealed the right choices for some common network scenarios. This work is currently under review with the relative performance comparison results presented in Figure 27, Figure 28 and Figure 29 for the cases of coding only the geometry, the geometry and the normals and the geometry and the custom attributes respectively. Given the difficulty in drawing conclusions from these comparisons since it is not easily possible to model both compression rate performance as well as coding performance efficiency simultaneously, we decided to run a network simulation that will jointly assess both processing and compression performance. To that end, we used a realistic TI scenario using vertices and triangles average values from the existing TI pipeline. In addition, we jointly coded geometry, connectivity, normals and attributes to closely resemble the actual scenario that will be deployed within 5G-Media. We also varied the networking conditions used. The results are presented in Table 25. As it can be seen, the more recent codecs (Corto & Draco) are better versions of the older ones (OpenCTM & O3dgc) respectively as the first offer better encoding/decoding performance at the cost of coding efficiency, while the latter produce lower bit-rates at higher encoding/decoding times. Therefore, the more recent codecs are a natural choice for the implementation of the vTranscoder3D. At the same time, Draco it performs better in degraded networking conditions compared to Corto whose worse rate performance is diminished as the networking conditions are improved. These considerations will better help in optimizing the vTranscoder3D VNF for different scenarios. It should be noted that the simulation, being a theoretic one, considers both best and worst case scenarios for latency, while it also models throughput (i.e. frame-rate) by considering all stages of the pipeline (compression, transmission and decompression).

[^57]: [https://github.com/amd/rest3d/tree/master/server/o3dgc](https://github.com/amd/rest3d/tree/master/server/o3dgc)
[^58]: [https://github.com/google/draco](https://github.com/google/draco)
[^59]: [https://github.com/cnr-isti-vclab/corto](https://github.com/cnr-isti-vclab/corto)
[^60]: [http://openctm.sourceforge.net/?page=about](http://openctm.sourceforge.net/?page=about)
[^61]: [https://github.com/lz4/lz4](https://github.com/lz4/lz4)
[^62]: [https://www.7-zip.org/sdk.html](https://www.7-zip.org/sdk.html)
Figure 27 – Relative objective comparisons between the codecs for a given geometry distortion level (3D Hausdorff distance). Performance is compared for the resulting bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to produce corresponding Hausdorff) (top right), decoding and encoding time (bottom left and bottom right respectively). This analysis is done for the geometry only coding case.
Figure 28 – Relative objective comparisons between the codecs for a given geometry distortion level (3D Hausdorff distance). Performance is compared for the resulting bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to produce corresponding Hausdorff) (top right), decoding and encoding time (bottom left and bottom right respectively). This analysis is done for the case of coding the geometry and the normals.

Figure 29 – Relative objective comparisons between the codecs for a given geometry distortion level (3D Hausdorff distance). Performance is compared for the resulting bit rate (top left), the Hausdorff distance (since OpenCTM does not manage to produce corresponding Hausdorff) (top right), decoding and encoding time (bottom left and bottom right respectively). This analysis is done for the case of coding the geometry and custom attributes.
Table 25 – Networking simulation results for a TI end-to-end pipeline assessing the performance of different 3D codecs.

<table>
<thead>
<tr>
<th>Codec</th>
<th>Latency Lower Bound (msec)</th>
<th>Latency Upper Bound (msec)</th>
<th>Frame-rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corto</td>
<td>37.06</td>
<td>53.51</td>
<td>49.69</td>
</tr>
<tr>
<td>Draco</td>
<td>41.42</td>
<td>64.44</td>
<td>55.84</td>
</tr>
<tr>
<td>O3dgc</td>
<td>50.53</td>
<td>80.87</td>
<td>50.78</td>
</tr>
<tr>
<td>OpenCTM</td>
<td>54.23</td>
<td>78.78</td>
<td>34.27</td>
</tr>
</tbody>
</table>

**Line Bandwidth: 1Gbps, RTT: 1ms**

<table>
<thead>
<tr>
<th>Codec</th>
<th>Latency Lower Bound (msec)</th>
<th>Latency Upper Bound (msec)</th>
<th>Frame-rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corto</td>
<td>20.79</td>
<td>29.44</td>
<td>85.82</td>
</tr>
<tr>
<td>Draco</td>
<td>26.94</td>
<td>37.82</td>
<td>64.19</td>
</tr>
<tr>
<td>O3dgc</td>
<td>35.25</td>
<td>50.32</td>
<td>50.78</td>
</tr>
<tr>
<td>OpenCTM</td>
<td>30.64</td>
<td>43.46</td>
<td>57.73</td>
</tr>
</tbody>
</table>

**Line Bandwidth: 100Mbps, RTT: 20ms**

<table>
<thead>
<tr>
<th>Codec</th>
<th>Latency Lower Bound (msec)</th>
<th>Latency Upper Bound (msec)</th>
<th>Frame-rate (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corto</td>
<td>103.49</td>
<td>119.93</td>
<td>12.81</td>
</tr>
<tr>
<td>Draco</td>
<td>101.58</td>
<td>124.60</td>
<td>14.58</td>
</tr>
<tr>
<td>O3dgc</td>
<td>113.47</td>
<td>144.61</td>
<td>13.82</td>
</tr>
<tr>
<td>OpenCTM</td>
<td>146.25</td>
<td>170.81</td>
<td>8.95</td>
</tr>
</tbody>
</table>

The previously mentioned research drove the implementation of vTranscoder, adopting different compression algorithms and parameterizations that perform optimal under specific network conditions. Further, the QoE studies helped us to conclude on the quality levels (profiles) that the transcoder would output, optimizing for the best end-user QoE which depends on visual quality and end-to-end latency given a fixed budget of processing power and network resources.

7.3.2.7. 5G-MEDIA Gateway

The purpose of the 5G-MEDIA Gateway PNF is to convert incoming SDI signals from cameras or video servers to IP to make use of them inside the network or vice versa converting those signals back to SDI. Gateways are typically hardware appliances so IRT will make use of vendor solutions here as well.

**Current Solution: Tandberg/Ericsson Video Processor EN8190 HD**

The current setup includes a Tandberg/Ericsson Video Processor which is high density, multifunctional, video processing platform. The Video Processor has a compact 1RU form factor with up to six hot swap option slots with a single PSU or up to four option slots with a dual PSU option, making it an ideal solution for the whole spectrum of high resilience to high density requirements. The Video Processor chassis supports a comprehensive range of video processing options, including SD MPEG-2 and HD MPEG-4 AVC encoding on hot swap option modules. This modular design allows for an easy upgrade of functionality and the addition of new services incrementally.
In our scenario the video processor serves as an encoder that is able to receive uncompressed audio and video signals via SDI and encode them as H.264. The output data is an MPEG transport stream (MPEG-TS) over IP (1 GbE interface) that requires much less bandwidth in comparison to the uncompressed SDI stream. This solution is preliminary due to the unavailability of SDI-to-IP gateways with ST 2110 support.

**Future Solution: Nevion Virtuoso**

The Nevion Virtuoso IP production Platform is a software defined media node platform designed to meet the challenges of an IP-based live broadcast production environment. The platform runs virtualized media functions dependent of the user’s needs. The functions reach from encoding/decoding and transport protection to monitoring, and signal processing.

In our specific scenario the platform is used to packetize incoming SDI signals without any encoding and transmit them with the uncompressed bitrate of about 1,5 gbps per signal stream to the vCompression Engine where the compression/encoding for the WAN transfer is carried out (Section 7.3.2.2).

Its key features are:

- Software-based approach with license options ensures easy and future-proof upgrade path
- Multi-format video and audio compression
- Adaptation of SDI, ASI and audio signals
- Fully standards-compliant IP media transport
- Advanced audio/video processing
- Content protection and encryption
- High availability device and network redundancy
- Simple integration in any IT/IP network infrastructure
- Powerful monitoring and alarm handling
- Virtuoso OS provides easy-to-use user interface for monitoring & control
Unfortunately, up to this point the Nevion platform does not support the new ST 2110 standard. An early update is expected in August 2018. As soon as this is available this solution supporting signals according to ST 2110 is be preferred and will be implemented in the 5G-MEDIA testbed for use case 2.

7.3.2.8. UHD Streaming Server

The UHD Streaming Server PNF, also called Origin Server is the server where all media contents are stored. The application is based on the media server Plex\(^{63}\).

Plex offers the possibility to stream media contents anywhere. It’s available for various devices and operating systems. It also offers the possibility to share its own library with other users. It supports MP4, MKV, MOV and other video formats. Regarding audio format, MP3, M4A, FLAC and other formats are supported. Plex supports also photos in JPEG, PNG and RAW format.

Plex server may be installed on Windows, Linux or OS X machines. But it can be easily installed also on NAS devices, like Synology\(^{64}\), Netgear\(^{65}\) or Drobo\(^{66}\).

On the other hand, media can be easily played over a large number of browsers and devices. Plex supports the majority of the browsers: Firefox, Chrome, Edge, Safari, etc. For mobile phones and tablet, Plex app is available for IOS, Windows and Android devices. The application may be used also with Smart TVs, Streaming devices (like Amazon Fire TV, Apple TV etc) as well as on Gaming Consoles (Playstation, Xbox). Symphony\(^{67}\), the Nextworks’ platform for smart builds, also integrates Plex on its player.

With Plex the user can stop playing a media stream on one device and resume it on another device, properly configured. The user can also transfer the stream from the current device to another listed device (i.e. from phone app to Smart TV).

Plex offers the possibility to automatically adjust the streaming quality, thanks to the transcoding feature. When this feature is on, the quality of the stream will be adapted to the connection speed. Media will start playing on the default quality rate and will be adapted during playback.

---

\(^{63}\) Media Server Plex: [http://www.plex.tv](http://www.plex.tv)

\(^{64}\) Synology: [https://www.synology.com](https://www.synology.com)

\(^{65}\) Netgear: [https://www.netgear.it](https://www.netgear.it)

\(^{66}\) Drobo: [https://www.drobo.com/](https://www.drobo.com/)

Conclusions

In this deliverable, the design of the Public Catalogue has been presented, placing it inside the general architecture of the 5G-MEDIA platform. This design contains the main components of the catalogue, the information models used by it, such as the packaging format, the app descriptors and the NS descriptors.

It has been also presented the interfaces needed to communicate to and from the catalogue towards other components of the platform. Another important topic presented is the policies of Authentication, Authorization and Accounting, which represent a security framework based on the role of each user.

Finally, it is mentioned as a main innovation, the ability of the catalogue to work with different solutions, basing the design on an ETSI compliant way, and proposing it as a general solution, compatible with multiple MANOs, VIMs and SDN controllers.

Finally, an initial set of generic network applications and media specific applications are presented, providing detailed description of its features and behaviours.
References


Annex I – ETSI NFV TST008 v.2.5.1 monitoring data YAML

The following table presents a draft for the TST 008 v2.5.1 YAML depicting a preliminary template for monitoring data.

```yaml
tosca_definitions_version: tosca_simple_yaml_1_0
# ETSI GS NFV TST 008 version 2.5.1

description: template for monitoring data
data_types:
tosca.datatypes.monitoring.ETSI.computing.parameters:
derived_from: tosca.datatypes.Root
properties:
tick_interval:
  type: float
  description: the period of timed interrupts when the processors execution context can be recorded.
  required: true
set_execution_context:
  type: map
  description: desired set of processor states with reported utilization
  entry_schema:
    type: float
    required: true
end_time:
  type: timestamp
  description: the termination of the measurement interval
  required: true
measurement_interval:
  type: float
  description: the duration of the observation by the measurement system
  required: true
tosca.datatypes.monitoring.ETSI.computing.metrics:
derived_from: tosca.datatypes.Root
properties:
processor_usage:
  type: integer
  description: total time that one or more compute resources are used.
  required: false
processor_utilization:
  type: float
  description: ration of processor_usage over total time measured
  required: false
```
tosca.datatypes.monitoring.ETSI.network.parameters:
  derived_from: tosca.datatypes.Root
  properties:
    net_interface:
      type: string
      description: interface identifier where communication metrics are monitored
      required: true
    measurement_time:
      type: timestamp
      description: the point in time when the counter were read
      required: true
    interface_speed:
      type: integer
      description: nominal frequency of the physical interface bit clock in bits per second.
      required: false
    interface_status:
      type: boolean
      description: operational state of the interface, if it is ready for use
      required: true

tosca.datatypes.monitoring.ETSI.network.metrics:
  derived_from: tosca.datatypes.Root
  properties:
    packet_count:
      type: integer
      description: successfully transmitted packages
      required: false
    octet_count:
      type: integer
      description: number of Bytes of the successfully transmitted packages
      required: false
    drop_packet_count:
      type: integer
      description: number of discarded packages
      required: false
    error_packet_count:
      type: integer
      description: in TX number of transmission attempts with errors, in RX number of packages with errors detected
      required: false

tosca.datatypes.monitoring.ETSI.memory.parameters:
  derived_from: tosca.datatypes.Root
properties:
  measurement_time:
    type: timestamp
    description: the point in time when the values were read
    required: true
  mem_ram:
    type: integer
    description: RAM available to the compute node measured in MB
    required: true
  swap_space:
    type: integer
    description: configured memory available for processes to share through swapping
    required: true

tosca.datatypes.monitoring.ETSI.memory.metrics:
  derived_from: tosca.datatypes.Root
  properties:
    memory_buffered:
      type: integer
      description: amount of temporary storage for raw disk blocks
      required: false
    memory_cached:
      type: integer
      description: amount of RAM used as cache memory
      required: false
    memory_free:
      type: integer
      description: amount of RAM unused
      required: false
    memory_slab:
      type: integer
      description: amount of memory used as a data structure cache by the kernel
      required: false
    memory_total:
      type: integer
      description: amount of usable RAM
      required: false
    memory_used:
      type: integer
      description: amount of memory used by user processes
      required: false

tosca.datatypes.monitoring.resources:
  derived_from: tosca.datatypes.Root
properties:
  computing_parameters:
    type: tosca.datatypes.monitoring.ETSI.computing.parameters
    required: true
  computing_metrics:
    type: tosca.datatypes.monitoring.ETSI.computing.metrics
    required: false
  memory_parameters:
    type: tosca.datatypes.monitoring.ETSI.memory.parameters
    required: true
  memory_metrics:
    type: tosca.datatypes.monitoring.ETSI.memory.metrics
    required: false
  storage:
    type: integer
    description: available storage in GB
    required: true
  storage_usage:
    type: float
    description: percentage of storage used
    required: true

tosca.datatypes.monitoring.networking:
  derived_from: tosca.datatypes.Root
  properties:
    network_parameters:
      type: tosca.datatypes.monitoring.ETSI.network.parameters
      required: true
    network_metrics:
      type: tosca.datatypes.monitoring.ETSI.network.metrics
      required: false

tosca.datatypes.monitoring.application:
  derived_from: tosca.datatypes.Root
  properties:
    app_params:
      type: null
      description: to define in future versions

node_types:
  tosca.nodes.monitoring:
    derived_from: tosca.nodes.Root
    properties:
      resources:
type: tosca.datatypes.monitoring.resources
required: true

networking:
  type: tosca.datatypes.monitoring.networking
  required: true

application:
  type: tosca.datatypes.monitoring.application
  required: true
Annex II

Here is an example of a manifest file signing on Linux using the openssl tool:

Private certificate generation. We are using a self-signed certificate for exemplifying purposes, but in a real world scenario the certificate should be generated with the CA available at NFVO (or Catalogue in 5G-MEDIA) level instead:

```bash
# openssl genrsa -aes128 -passout pass:test -out private.pem 2048
```

Public certificate generation. This certificate will be distributed within the CSAR file:

```bash
# openssl rsa -in private.pem -passin pass:test -pubout -out public.cert
```

Manifest file signing with the private certificate:

```bash
# openssl dgst -sha256 -sign private.pem -out /tmp/test.mf.sha256 test.mf
```

A base64 encoded signature is generate in order to have it included in the manifest file:

```bash
# openssl base64 -in /tmp/test.mf.sha256 -out signature.sha256
```

Let’s add the base64 encoded signature. Please note that this echo command is not enough, the encoded signature should be wrapped in the correct textual encoding as per IETF RFC7468, in this case using the PKCS#7 format

```bash
# echo signature.sha256 >> test.mf
```

On the receiving end (Catalogue or NFVO’s) the manifest is parsed and the signature digest extracted. Here is how the signature gets decoded:

```bash
# openssl base64 -d -in signature.sha256 -out /tmp/test_v.sha256
```

Finally the public.cert is use in concert with the signature file to verify the integrity of the manifest file:

```bash
openssl dgst -sha256 -verify public.cert -signature test_v.sha256 test.mf
```

A valid response should be: **Verified OK**

---

Annex III – ETSI NFV SOL005 v.2.4.1 NSD Management Interface YAML

The following table presents a draft for the SOL 005 NSD Management Interface YAML, which extends the NSDManagement.yaml available in ETSI OpenAPIs. This YAML is suitable for generating clients and servers’ stubs using the Swagger Codegen tool (Swagger v0.2).

```yaml
swagger: '2.0'

info:
  description: |
    DRAFT - SOL005 - NSD Management Interface IMPORTANT: Please note that this file might be not aligned to the current version of the ETSI Group Specification it refers to and has not been approved by the ETSI NFV ISG. In case of discrepancies the published ETSI Group Specification takes precedence. Please report bugs to https://forge.etsi.org/bugzilla/buglist.cgi?component=Nfv-Openapis version: 2.4.1
title: DRAFT - SOL005 - NSD Management Interface
contact:
  name: NFV-SOL WG
license:
  name: ETSI Forge copyright notice
  url: 'https://forge.etsi.org/etsi-forge-copyright-notice.txt'

produces:
- application/json
- application/yaml

paths:

'/nsd/v1/ns_descriptors':
  get:
    summary: Query NSD Info
    operationId: getNSDsInfo
description: >-
      The GET method queries information about multiple NS descriptor resources. This method shall follow the provisions specified in the Tables 5.4.2.3.2-1 and 5.4.2.3.2-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.
    parameters: []
    responses:
      '200':
        description: >-
          Information about zero or more NS descriptors.
```

Page 102/143
The response body shall contain a representation of zero or more NS descriptors, as defined in clause 5.5.2.2 of GS NFV-SOL 005.

**schema:**
- **type:** array
- **items:**
  - **type:** object
  - $ref: '#/definitions/NsdInfo'

'400':
- **description:** There are two possible scenarios listed below.
  - **Error:** Invalid attribute-based filtering parameters.
  - The response body shall contain a ProblemDetails structure, in which the "detail" attribute should convey more information about the error.
  - **Error:** Invalid attribute selector. The response body shall contain a ProblemDetails structure, in which the "detail" attribute should convey more information about the error.

**schema:**
- **type:** object
  - $ref: '#/definitions/ProblemDetails'

'500':
- **description:** Status 500
- **schema:**
  - **type:** object
  - $ref: '#/definitions/ProblemDetails'

**post:**
- **summary:** Create NSD Info
- **operationId:** createNsdInfo
- **description:** The POST method is used to create a new NS descriptor resource. This method shall follow the provisions specified in the Tables 5.4.2.3.1-1 and 5.4.2.3.1-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.

**consumes:**
- application/json

**parameters:**
- **name:** body
  - **in:** body
  - **required:** true
  - **schema:**
    - **type:** object
$ref: '#/definitions/CreateNsdInfoRequest'
  description: >-
The CreateNsdInfoRequest contains parameters for creating an NS descriptor resource, as defined in clause 5.5.2.4 of GS NFV-SOL 005.

responses:
'201':
  description: Status 201
  schema:
    type: object
    $ref: '#/definitions/NsdInfo'
  description: >-
    An NS descriptor resource was created successfully, as a new NS descriptor resource. The response body shall contain a representation of the new NS descriptor resource, as defined in clause 5.5.2.2 of GS NFV-SOL 005.
  headers:
    Location:
      type: string
      description: >-
        The HTTP response shall include a "Location" HTTP header that contains the resource URI of the new NS descriptor resource.

'400':
  description: Status 400
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'500':
  description: Status 500
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'/nsd/v1/ns_descriptors/{nsdInfoId}':
  get:
    summary: Query NSD Info
    operationId: getNSDInfo
    description: >-
The GET method reads information about an individual NS descriptor. This method shall follow the provisions specified in GS NFV-SOL 005 Tables 5.4.3.3.2-1 and 5.4.3.3.2-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.
    parameters: []
    responses:
      '200':
description: >-
Information about the individual NS descriptor. The response body shall contain a representation of the individual NS descriptor, as defined in clause 5.5.2.2 of GS NFV-SOL 005.
schema:
  type: object
  $ref: '#/definitions/NsdInfo'
'400':
description: Status 400
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
'404':
description: Status 404
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
'409':
description: Status 409
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
'500':
description: Status 500
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
delete:
  summary: Delete NSD
  operationId: deleteNSDInfo
description: >-
The DELETE method deletes an individual NS descriptor resource. An individual NS descriptor resource can only be deleted when there is no NS instance using it (i.e. usageState = NOT_IN_USE) and has been disabled already (i.e. operationalState = DISABLED). Otherwise, the DELETE method shall fail. This method shall follow the provisions specified in the Tables 5.4.3.3.5-1 and 5.4.3.3.5-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.
parameters: []
responses:
  '204':
description: >-
The operation has completed successfully. The response body shall be
empty.

'404':
description: Status 404
schema:
type: object
$ref: '#/definitions/ProblemDetails'

'409':
description: Status 409
schema:
type: object
$ref: '#/definitions/ProblemDetails'
description: >-
Error: The operation cannot be executed currently, due to a conflict with the state of the resource. Typically, this is due to the fact the NS descriptor resource is in the enabled operational state (i.e. operationalState = ENABLED) or there are running NS instances using the concerned individual NS descriptor resource (i.e. usageState = IN_USE). The response body shall contain a ProblemDetails structure, in which the "detail" attribute shall convey more information about the error.

'500':
description: Status 500
schema:
type: object
$ref: '#/definitions/ProblemDetails'

patch:
summary: Update NSD Info
operationId: updateNSDInfo
description: >-
The PATCH method modifies the operational state and/or user defined data of an individual NS descriptor resource. This method can be used to:

1) Enable a previously disabled individual NS descriptor resource, allowing again its use for instantiation of new network service with this descriptor. The usage state (i.e. "IN_USE/NOT_IN_USE") shall not change as a result.

2) Disable a previously enabled individual NS descriptor resource, preventing any further use for instantiation of new network service(s) with this descriptor.
The usage state (i.e. "IN_USE/NOT_IN_USE") shall not change as a result.

3) Modify the user defined data of an individual NS descriptor resource.

This method shall follow the provisions specified in the Tables.
5.4.3.4.1 and 5.4.3.4.2 for URI query parameters, request and response data structures, and response codes.

consumes:
- application/json

parameters:
- name: body
  in: body
  required: true

schema:
  type: object
  $ref: '#/definitions/NsdInfoModifications'


description:
The operation was completed successfully. The response body shall contain attribute modifications for an 'Individual NS Descriptor' resource (see clause 5.5.2.6 of GS NFV SOL-005).

responses:
'200':
  description: Status 200
  schema:
    type: object
    $ref: '#/definitions/NsdInfoModifications'

description:
This type represents attribute modifications for an individual NS descriptor resource based on the "NsdInfo" data type. The attributes of "NsdInfo" that can be modified are included in the "NsdInfoModifications" data type.

NOTE: At least one of the attributes - nsdOperationalState and userDefinedData - shall be present.

'400':
  description: Status 400
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'404':
  description: Status 404
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'409':
  description: Status 409
  schema:
    type: object
/nsd/v1/ns_descriptors/{nsdInfoId}/nsd_content':
get:
schema:
parameters:
- name: nsdInfoId
  in: path
  required: true
  type: string

The GET method fetches the content of the NSD.

The NSD can be implemented as a single file or as a collection of multiple files. If the NSD is implemented in the form of multiple files, a ZIP file embedding these files shall be returned. If the NSD is implemented as a single file, either that file or a ZIP file embedding that file shall be returned.

The selection of the format is controlled by the "Accept" HTTP header passed in the GET request:

- If the "Accept" header contains only "text/plain" and the NSD is implemented as a single file, the file shall be returned; otherwise, an error message shall be returned.

- If the "Accept" header contains only "application/zip", the single file or the multiple files that make up the NSD shall be returned embedded in a ZIP file.

- If the "Accept" header contains both "text/plain" and "application/zip", it is up to the NFVO to choose the format to return for a single-file NSD; for a multi-file NSD, a ZIP file shall be
NOTE: The structure of the NSD zip file is outside the scope of the present document.

produces:
- application/json
- application/yaml
- application/zip

parameters:
# - name: Accept
#   in: header
#   required: true
#   type: string
#   description: >-
#     The request shall contain the appropriate entries in the "Accept"
#     HTTP header as defined above.
- name: Range
  in: header
  required: false
  type: string
  description: >-
    The request may contain a "Range" HTTP header to obtain single range of bytes from the NSD file. This can be used to continue an aborted transmission.

If the NFVO does not support range requests, the NFVO shall ignore the 'Range' header, process the GET request, and return the whole NSD file with a 200 OK response (rather than returning a 4xx error status code).

responses:
'200':
  description: >-
    On success, the content of the NSD is returned. The payload body shall contain a copy of the file representing the NSD or a ZIP file that contains the file or multiple files representing the NSD, as specified above. The "Content-Type" HTTP header shall be set according to the format of the returned file, i.e. to "text/plain" for a YAML file or to "application/zip" for a ZIP file.

schema:
  type: object
headers:
  Content-Type:
    type: string
'206':
  description: >-
    On success, if the NFVO supports range requests, a single consecutive byte range from the content of the NSD file is returned.

    The response body shall contain the requested part of the NSD file.

    The "Content-Range" HTTP header shall be provided according to IETF RFC 7233 [23].

    The "Content-Type" HTTP header shall be set as defined above for the "200 OK" response.

headers:
  Content-Range:
    type: string
  Content-Type:
    type: string

'404':
  description: Status 404
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'406':
  description: Status 406
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'
  description: >-
    If the "Accept" header does not contain at least one name of a content type for which the NFVO can provide a representation of the NSD, the NFVO shall respond with this response code. The "ProblemDetails" structure may be included with the "detail" attribute providing more information about the error.

'409':
  description: Status 409
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'
  description: >-
    Error: The operation cannot be executed currently, due to a conflict with the state of the resource. Typically, this is due to
the fact "nsdOnboardingState" has a value different from ONBOARDED. The response body shall contain a ProblemDetails structure, in which the "detail" attribute shall convey more information about the error.

'416':
description: Status 416
schema:
type: object
$ref: '#/definitions/ProblemDetails'
description: The byte range passed in the "Range" header did not match any available byte range in the NSD file (e.g. "access after end of file"). The response body may contain a ProblemDetails structure.

'500':
description: Status 500
schema:
type: object
$ref: '#/definitions/ProblemDetails'

put:
summary: Upload NSD
operationId: uploadNSD
description: The PUT method is used to upload the content of a NSD. The NSD to be uploaded can be implemented as a single file or as a collection of multiple files, as defined in clause 5.4.4.3.2 of GS NFV-SOL 005. If the NSD is implemented in the form of multiple files, a ZIP file embedding these files shall be uploaded. If the NSD is implemented as a single file, either that file or a ZIP file embedding that file shall be uploaded. The "Content-Type" HTTP header in the PUT request shall be set accordingly based on the format selection of the NSD. If the NSD to be uploaded is a text file, the "Content-Type" header is set to "text/plain". If the NSD to be uploaded is a zip file, the "Content-Type" header is set to "application/zip". This method shall follow the provisions specified in the Tables 5.4.4.3.3-1 and 5.4.4.3.3-2 of GS-NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.
consumes:
- application/json
- application/yaml
- application/zip
parameters:
- name: Content-Type
in: header
required: false
type: string
description: >-
The payload body contains a copy of the file representing the NSD or a ZIP file that contains the file or multiple files representing the NSD, as specified above. The request shall set the "Content-Type" HTTP header as defined above.

- name: body
  in: body
  required: true
  schema:
    type: object

responses:
'202':
description: Status 202
schema:
  type: object
description: >-
The NSD content was accepted for uploading, but the processing has not been completed. It is expected to take some time for processing (asynchronous mode). The response body shall be empty. See note.

'204':
description: >-
The NSD content was successfully uploaded and validated (synchronous mode). The response body shall be empty.

'400':
description: Status 400
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'404':
description: Status 404
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'409':
description: >-
Error: The operation cannot be executed currently, due to a conflict with the state of the resource. Typically, this is due to the fact that the NsdOnboardingState has a value other than CREATED. The response body shall contain a ProblemDetails structure, in which the "detail" attribute shall convey more information about the error.
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'500':
  description: Status 500
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

parameters:
- name: nsdInfoId
  in: path
  required: true
  type: string

'/nsd/v1/pnf_descriptors':
  get:
    summary: Query PFNDs Info
    operationId: getPNFDsInfo
    description: 
      The GET method queries information about multiple PNF descriptor resources.
    parameters:
    - name: exclude_default
      in: query
      required: false
      type: string
      description: Indicate to exclude the following complex attributes from the response.
      See clause 4.3.3 for details.
      
      The NFVO shall support this parameter.
      
      The following attributes shall be excluded from the PnfInfo structure in the response body if this parameter is provided, or none of the parameters "all_fields," "fields", "exclude_fields", "exclude_default" are provided: userDefinedData.
    - name: all_fields
      in: query
      required: false
      type: string
      description: 

Include all complex attributes in the response. See clause 4.3.3 for details. The NFVO shall support this parameter.

responses:

'200':

description: Status 200

schema:

type: array

items:

type: object

$ref: '#/definitions/PnfdInfo'

description: >-

Information about zero or more PNF descriptors. The response body shall contain a representation of zero or more PNF descriptors, as defined in clause 5.5.2.2.

'400':

description: Status 400

schema:

type: object

$ref: '#/definitions/ProblemDetails'

'500':

description: Status 500

schema:

type: object

$ref: '#/definitions/ProblemDetails'

post:

summary: Create PNFD Info

operationId: createPNFDInfo

description: The POST method is used to create a new PNF descriptor resource.

consumes:

- application/json

parameters:

- name: body

  in: body

  required: true

  schema:

    type: object

    $ref: '#/definitions/PnfdInfo'


description: |

Parameters of creating a PNF descriptor resource, as defined in clause 5.5.2.6 of GS NFV-SOL 005.

responses:

'201':

description: Status 201

schema:
type: object
 $ref: '#/definitions/PnfdInfo'

description: >-
A PNF descriptor resource was created successfully, as a new PNF descriptor resource. The response body shall contain a representation of the new PNF descriptor resource, as defined in clause 5.5.2.5. The HTTP response shall include a "Location" HTTP header that contains the resource URI of the new PNF descriptor resource.

'400':
description: Status 400
schema:
 type: object
 $ref: '#/definitions/ProblemDetails'

'500':
description: Status 500
schema:
 type: object
 $ref: '#/definitions/ProblemDetails'

'/nsd/v1/pnf_descriptors/{pnfdInfoId}':
get:
 summary: Query PNFD Info
operationId: getPNFDInfo
description: >-
The GET method reads information about an individual PNF descriptor. This method shall follow the provisions specified in the Tables 5.4.6.3.2-1 and 5.4.6.3.2-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.

parameters: []
responses:
 '200':
description: >-
Information about the individual PNFD descriptor. The response body shall contain a representation of the individual PNF descriptor, as defined in clause 5.5.2.5 of GS NFV-SOL 005.

schema:
 type: object
 $ref: '#/definitions/PnfdInfo'

'400':
description: Status 400
schema:
 type: object
 $ref: '#/definitions/ProblemDetails'
'404':
  description: Status 404
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'409':
  description: Status 409
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'500':
  description: Status 500
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

delete:
  summary: Delete PNFD
  operationId: deletePNFDInfo
  description: >-
    The DELETE method deletes an individual PNF descriptor resource. An
    individual PNF descriptor resource can only be deleted when there is no
    NS instance using it or there is NSD referencing it. To delete all PNFD
    versions identified by a particular value of the "pnfdInvariantId"
    attribute, the procedure is to first use the GET method with filter
    "pnfdInvariantId" towards the PNF descriptors resource to find all
    versions of the PNFD. Then, the client uses the DELETE method described
    in this clause to delete each PNFD version individually. This method
    shall follow the provisions specified in the Tables 5.4.6.3.5-1 and
    5.4.6.3.5-2 of GS NFV-SOL 005 for URI query parameters, request and
    response data structures, and response codes.

parameters: []

responses:
  '204':
    description: >-
      The operation has completed successfully. The response body shall be
      empty.
  '400':
    description: Status 400
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
  '404':
    description: Status 404
    schema:
type: object

'schema:

description: Status 500

'type: object

'schema:

description: Status 500

'patch:

summary: Update PNFD Info

operationId: updatePNFDInfo

description: The PATCH method modifies the user defined data of an individual PNF descriptor resource. This method shall follow the provisions specified in the Tables 5.4.6.3.4-1 and 5.4.6.3.4-2 for URI query parameters, request and response data structures, and response codes.

consumes:
- application/json

parameters:
- name: body
  in: body
  required: true

'schema:

type: object

'schema:

description: This type represents attribute modifications for an individual PNF descriptor resource based on the "PnfdInfo" data type. The attributes of "PnfdInfo" that can be modified are included in the "PnfdInfoModifications" data type.

responses:

'200':

description: Status 200

'schema:

type: object

'schema:

description: Status 200

'400':

description: Status 400

'schema:

type: object
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>Status 404</td>
<td>object</td>
</tr>
<tr>
<td>409</td>
<td>Status 409</td>
<td>object</td>
</tr>
<tr>
<td>412</td>
<td>Status 412</td>
<td>object</td>
</tr>
<tr>
<td>500</td>
<td>Status 500</td>
<td>object</td>
</tr>
</tbody>
</table>

**Parameters**

- `pnfdInfoId`
  - `in`: path
  - `required`: true
  - `type`: string

**Get**

```plaintext
/\nsd/v1/pnf_descriptors/{pnfdInfoId}/pnfd_content'
```

- `summary`: Get PNFD Content
- `operationId`: getPNFD
- `description`: The GET method fetches the content of the PNFD.
- `produces`:
  - `application/json`
  - `application/yaml`
  - `application/zip`
- `responses`:
  - `200`
    - `schema`
    - `type`: object
    - `description`: On success, the content of the PNFD is returned. The payload body shall contain a copy of the file representing the PNFD. The
"Content-Type" HTTP header shall be set to "text/plain".

'400':
  description: Status 400
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'409':
  description: Status 409
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'404':
  description: Status 404
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

'500':
  description: Status 500
  schema:
    type: object
    $ref: '#/definitions/ProblemDetails'

put:
  summary: Upload PNFD
  operationId: uploadPNFD
  description: >-
    The PUT method is used to upload the content of a PNFD. This method shall follow the provisions specified in the Tables 5.4.7.3.3-1 and 5.4.7.3.3-2 of GS NFV-SOL 005 for URI query parameters, request and response data structures, and response codes.
  consumes:
    - application/json
    - application/yaml
  parameters:
    - name: Content-Type
      in: header
      required: false
      type: string
      description: >-
        The request shall set the "Content-Type" HTTP header to "text/plain".
    - name: body
      in: body
      required: true
      schema:
type: object
description: The payload body contains a copy of the file representing the PNFD.

responses:
'204':
description: The PNFD content was successfully uploaded and validated. The response body shall be empty.

'400':
description: Status 400
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'404':
description: Status 404
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'409':
description: Status 409
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'500':
description: Status 500
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

parameters:
- name: pnfdInfoId
  in: path
  required: true
  type: string

'/nsd/v1/subscriptions':
get:
  summary: Query Subscription Information
  operationId: getSubscriptions
  description: The GET method queries the list of active subscriptions of the functional block that invokes the method. It can be used e.g. for resynchronization after error situations. This method shall support the URI query parameters, request and response data structures, and response
codes, as specified in the Tables 5.4.8.3.2-1 and 5.4.8.3.2-2 of GS
NFV-SOL 005.
parameters: []
responses:
  '200':
    description: Status 200
    schema:
      type: array
      description: >-
        The list of subscriptions was queried successfully. The response
        body shall contain the representations of all active subscriptions
        of the functional block that invokes the method.
      items:
        type: object
        $ref: '#/definitions/NsdmSubscription'
        description: >-
          This type represents a subscription related to notifications
          about NSD management.
  #  '303':
  #    description: >-
  #      A subscription with the same callbackURI and the same filter already
  #      exits and the policy of the NFVO is to not create redundant
  #      subscriptions. The response body shall be empty.
  #    headers:
  #      Location:
  #        type: string
  #        description: >-
  #          The HTTP response shall include a "Location" HTTP header that
  #          contains the resource URI of the existing subscription resource.
  '400':
    description: Status 400
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
  '500':
    description: Status 500
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
post:
  summary: Subscribe
  operationId: createSubscription
  description: >-
    The POST method creates a new subscription. This method shall support
the URI query parameters, request and response data structures, and response codes, as specified in the Tables 5.4.8.3.1-1 and 5.4.8.3.1-2 of GS-NFV SOL 005. Creation of two subscription resources with the same callbackURI and the same filter can result in performance degradation and will provide duplicates of notifications to the OSS, and might make sense only in very rare use cases. Consequently, the NFVO may either allow creating a subscription resource if another subscription resource with the same filter and callbackUri already exists (in which case it shall return the "201 Created" response code), or may decide to not create a duplicate subscription resource (in which case it shall return a "303 See Other" response code referencing the existing subscription resource with the same filter and callbackUri).

consumes: []
parameters:
- name: body
  in: body
  required: true
  schema:
    type: object
    $ref: '#/definitions/NsdmSubscriptionRequest'
    description: >-
This type represents a subscription request related to notifications about NSD management.

responses:
  '201':
    description: Status 201
    schema:
      type: object
      $ref: '#/definitions/NsdmSubscription'
      description: >-
This type represents a subscription related to notifications about NSD management.
    headers:
      Location:
        type: string
        description: |
          The HTTP response shall include a "Location:" HTTP header that points to the created subscription resource.

  '303':
    description: >-
A subscription with the same callbackURI and the same filter already exists and the policy of the NFVO is to not create redundant subscriptions. The response body shall be empty.
    headers:
Location:
  type: string
description: The HTTP response shall include a "Location" HTTP header that contains the resource URI of the existing subscription resource.
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
'400':
description: Status 400
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
'500':
description: Status 500
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'

'/nsd/v1/subscriptions/{subscriptionId}':
get:
summary: Query Subscription Information
operationId: getSubscription
description: The GET method retrieves information about a subscription by reading an individual subscription resource. This method shall support the URI query parameters, request and response data structures, and response codes, as specified in the Tables 5.4.9.3.2-1 and 5.4.9.3.2-2.
parameters: []
responses:
  '200':
description: Status 200
schema:
  type: object
  $ref: '#/definitions/NsdmSubscription'
description: This type represents a subscription related to notifications about NSD management.
  '400':
description: Status 400
schema:
  type: object
  $ref: '#/definitions/ProblemDetails'
  '404':
delete:
summary: Terminate Subscription
operationId: deleteSubscription
description: >-
The DELETE method terminates an individual subscription. This method shall support the URI query parameters, request and response data structures, and response codes, as specified in the Tables 5.4.9.3.5-1 and 5.4.9.3.3-2 of GS NFV-SOL 005.
parameters: []
responses:
  '204':
    description: >-
The subscription resource was deleted successfully. The response body shall be empty.
  '400':
    description: Status 400
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
  '404':
    description: Status 404
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
  '500':
    description: 500
    schema:
      type: object
      $ref: '#/definitions/ProblemDetails'
parameters:
- name: subscriptionId
  in: path
  required: true
  type: string
definitions:

# Identifier:
#    type: object
#
# Uri:
#    type: object
#
# Link:
#    type: object

SubscriptionAuthentication:
    type: object

ProblemDetails:
    type: object
    properties:
        type:
            type: string
            format: uri
        title:
            type: string
        status:
            type: integer
        detail:
            type: string
        instance:
            type: string
            format: uri
    additionalProperties: true
    required:
        - status
        - detail

KeyValuePairs:
    type: object
    additionalProperties:
        type: string

NsdOperationalStateType:
    type: string
    description: The enumeration NsdOperationalStateType shall comply with the provisions
defined in Table 5.5.4.3-1 of GS NFV_SOL 005. It indicates the operational state of the resource.

ENABLED = The operational state of the resource is enabled.
DISABLED = The operational state of the resource is disabled.

enum:
- ENABLED
- DISABLED

NsdOnboardingStateType:
type: string
description: >-
The enumeration NsdOnboardingStateType shall comply with the provisions defined in Table 5.5.4.5-1 of GS NFV-SOL 005. It indicates the onboarding state of the NSD.

CREATED = The NSD information object is created. UPLOADING = The associated NSD content is being uploaded. PROCESSING = The associated NSD content is being processed, e.g. validation. ONBOARDED = The associated NSD content is on-boarded.

enum:
- CREATED
- UPLOADING
- PROCESSING
- ONBOARDED

NsdUsageStateType:
type: string
description: >-
The enumeration NsdUsageStateType shall comply with the provisions defined in Table 5.5.4.4-1 of GS NFV-SOL 005. It indicates the usage state of the resource.

IN_USE = The resource is in use.
NOT_IN_USE = The resource is not-in-use.

enum:
- IN_USE
- NOT_IN_USE

NsdLinksType:
type: object
description: Links to resources related to this resource.
properties:
  self:
    type: string
    format: uri
  nsd_content:
    type: string
    format: uri

PnfdOnboardingStateType:
  type: string
  description: >-
    The enumeration PnfdOnboardingStateType shall comply with the provisions defined in Table 5.5.4.6-1 of GS-NFV SOL005. It indicates the onboarding state of the individual PNF descriptor resource.
    
    CREATED = The PNF descriptor resource is created. UPLOADING = The associated PNFD content is being uploaded. PROCESSING = The associated PNFD content is being processed, e.g. validation. ONBOARDED = The associated PNFD content is on-boarded.
  enum:
    - CREATED
    - UPLOADING
    - PROCESSING
    - ONBOARDING

PnfdUsageStateType:
  type: string
  description: >-
    The enumeration PnfdUsageStateType shall comply with the provisions defined in Table 5.5.4.7-1 of GS NFV-SOL005. It indicates the usage state of the resource.
    
    IN-USE = The resource is in use.
    NOT_IN_USE = The resource is not-in-use.
  enum:
    - IN_USE
    - NOT_IN_USE

PnfdLinksType:
  type: object
  description: Links to resources related to this resource.
properties:
  self:
    type: string
    format: uri
  pnfd_content:
    type: string
    format: uri

NsdInfoModifications:
  type: object
  properties:
    nsdOperationalState:
      type: string
      $ref: '#/definitions/NsdOperationalStateType'
    userDefinedData:
      type: object
      $ref: '#/definitions/KeyValuePairs'

NsdInfo:
  type: object
  required:
    - _links
    - id
    - nsdOnboardingState
    - nsdOperationalState
    - nsdUsageState
  properties:
    id:
      type: string
      format: uuid
    nsdId:
      type: string
      format: uuid
    nsdName:
      type: string
      description: >-
        Name of the onboarded NSD. This information is copied from the NSD content and shall be present after the NSD content is on-boarded.
    nsdVersion:
      type: string
    nsdDesigner:
      type: string
      description: >-
Designer of the on-boarded NSD. This information is copied from the NSD content and shall be present after the NSD content is on-boarded.

nsdInvariantId:
  type: string
  format: uuid

vnfPkgIds:
  type: array
  items:
    type: string
    format: uuid

pnfdInfolds:
  type: array
  items:
    type: string
    format: uuid

nestedNsdInfolds:
  type: array
  items:
    type: string
    format: uuid

nsdOnboardingState:
  type: string
  $ref: '#/definitions/NsdOnboardingStateType'

onboardingFailureDetails:
  type: string
  $ref: '#/definitions/ProblemDetails'

nsdOperationalState:
  type: string
  $ref: '#/definitions/NsdOperationalStateType'

nsdUsageState:
  type: string
  $ref: '#/definitions/NsdUsageStateType'

userDefinedData:
  type: object
  $ref: '#/definitions/KeyValuePairs'

  _links:
    type: object
    description: Links to resources related to this resource.
    $ref: '#/definitions/NsdLinksType'

description: This type represents a response for the query NSD operation.
userDefinedData:
  type: object
  $ref: '#/definitions/KeyValuePairs'
description: This type creates a completely new NS descriptor resource.

PnfdInfoModifications:
  type: object
  required:
    - userDefinedData
  properties:
    userDefinedData:
      type: object
      $ref: '#/definitions/KeyValuePairs'
      description: >-
        This type represents attribute modifications for an individual PNF descriptor resource based on the "PnfdInfo" data type. The attributes of "PnfdInfo" that can be modified are included in the "PnfdInfoModifications" data type.

PnfdInfo:
  type: object
  required:
    - _links
    - id
    - pnfdOnboardingState
    - pnfdUsageState
  properties:
    id:
      type: string
      format: uuid
    pnfdId:
      type: string
      format: uuid
    pnfdName:
      type: string
      description: >-
        Name of the on boarded PNFD. This information is copied from the PNFD content and shall be present after the PNFD content is on-boarded.
    pnfdVersion:
      type: string
    pnfdProvider:
      type: string
      description: >-
        Provider of the on boarded PNFD. This information is copied from the
PNFD content and shall be present after the PNFD content is onboarded.

- **pnfdInvariantId:**
  - type: string
  - format: uuid

- **pnfdOnboardingState:**
  - type: string
  - $ref: '#/definitions/PnfdOnboardingStateType'

- **onboardingFailureDetails:**
  - type: string
  - $ref: '#/definitions/ProblemDetails'

- **pnfdUsageState:**
  - type: string
  - $ref: '#/definitions/PnfdUsageStateType'

- **userDefinedData:**
  - type: object
  - $ref: '#/definitions/KeyValuePairs'

- **_links:**
  - type: object
    - description: Links to resources related to this resource.
    - $ref: '#/definitions/PnfdLinksType'

**NotificationType:**

- type: string
  - description: Match particular notification types. Permitted values:
    - NsdOnBoardingNotification, NsdOnboardingFailureNotification,
    - NsdChangeNotification, NsdDeletionNotification

    PnfdOnBoardingNotification, PnfdOnBoardingFailureNotification,
    PnfdDeletionNotification.

The permitted values of the "notificationTypes" attribute are spelled exactly as the names of the notification types to facilitate automated code generation systems.

- enum:
  - NsdOnBoardingNotification
  - NsdOnboardingFailureNotification
  - NsdChangeNotification
  - NsdDeletionNotification
  - PnfdOnBoardingNotification
  - PnfdOnBoardingFailureNotification
  - PnfdDeletionNotification
NsdmLinksType:
   type: object
   description: Links to resources related to this resource.
   properties:
     self:
       type: string
       format: uri

NsdmNotificationsFilter:
   type: object
   properties:
     notificationTypes:
       type: array
       items:
         type: string
         $ref: '#/definitions/NotificationType'
     nsdInfoId:
       type: string
       format: uuid
     nsdId:
       type: string
       format: uuid
     nsdName:
       type: string
       description: Match the name of the onboarded NSD.
     nsdVersion:
       type: string
     nsdDesigner:
       type: string
       description: Match the NSD designer of the on-boarded NSD.
     nsdInvariantId:
       type: string
       format: uuid
     vnfPkgIds:
       type: array
       items:
         type: string
         format: uuid
     pnfdInfolds:
       type: array
       items:
         type: string
         format: uuid
nestedNsdInInfoIds:
  type: array
  items:
    type: string
    format: uuid

nsdOnboardingState:
  type: string
  $ref: '#/definitions/NsdOnboardingStateType'

nsdOperationalState:
  type: string
  $ref: '#/definitions/NsdOperationalStateType'

nsdUsageState:
  type: string
  $ref: '#/definitions/NsdUsageStateType'

pnfdId:
  type: string
  format: uuid

pnfdName:
  type: string
  description: Match the name of the on-boarded PNFD.

pnfdVersion:
  type: string

pnfdProvider:
  type: string
  description: Match the provider of the on-boarded PNFD.

pnfdInvariantId:
  type: string
  format: uuid

pnfdOnboardingState:
  type: string
  $ref: '#/definitions/PnfdOnboardingStateType'

pnfdUsageState:
  type: string
  $ref: '#/definitions/PnfdUsageStateType'

description: >-
  This type represents a subscription filter related to notifications about
  NSD management. It shall comply with the
  provisions defined in Table 5.5.3.2-1 of GS NFV-SOL 005. At a particular
  nesting level in the filter structure, the following applies: All
  attributes shall match in order for the filter to match (logical "and"
  between different filter attributes). If an attribute is an array, the
  attribute shall match if at least one of the values in the array matches
  (logical "or" between the values of one filter attribute).
NsdmSubscription:
  type: object
  required:
  - _links
  - callbackUri
  - id
  properties:
    id:
      type: string
      format: uuid
    filter:
      type: object
      $ref: '#/definitions/NsdmNotificationsFilter'
      description: This type represents a subscription filter related to notifications about NSD management. It shall comply with the provisions defined in Table 5.5.3.2-1 of GS NFV-SOL 005. At a particular nesting level in the filter structure, the following applies: All attributes shall match in order for the filter to match (logical "and" between different filter attributes). If an attribute is an array, the attribute shall match if at least one of the values in the array matches (logical "or" between the values of one filter attribute).
    callbackUri:
      type: string
      format: uri
    _links:
      type: object
      description: Links to resources related to this resource.
      $ref: '#/definitions/NsdmLinksType'
      description: This type represents a subscription related to notifications about NSD management.

NsdmSubscriptionRequest:
  type: object
  required:
  - callbackUri
  properties:
    filter:
      type: object
      $ref: '#/definitions/NsdmNotificationsFilter'
```json

callbackUri:
  type: string
  format: uri
  description: The URI of the endpoint to send the notification to.

authentication:
  type: object
  $ref: '#/definitions/SubscriptionAuthentication'
  description: This type represents a subscription request related to notifications about NSD management.

CreatePnfdInfoRequest:
  type: object
  properties:
    userDefinedData:
      type: object
      $ref: '#/definitions/KeyValuePairs'
      description: This type creates a new PNF descriptor resource.

NsdmLinks:
  type: object
  required:
  - nsdInfo
  - subscription
  properties:
    nsdInfo:
      type: string
      format: uri
    subscription:
      type: string
      format: uri
    description: This type represents the links to resources that an NSD management notification can contain.

NsdOnboardingNotification:
  type: object
  required:
  - _links
  - id
  - notificationType
  - nsdlId
  - nsdInfoId
  - timeStamp
```
properties:
  id:
    type: string
    format: uuid
  notificationType:
    type: string
    $ref: '#/definitions/NotificationType'
    description: Discriminator for the different notification types. Shall be set to "NsdOnboardingNotification" for this notification type.
  subscriptionId:
    type: string
    format: uuid
  timeStamp:
    type: string
    format: date-time
    description: Date-time of the generation of the notification.
  nsdInfoId:
    type: string
    format: uuid
  nsdId:
    type: string
    format: uuid
  _links:
    type: object
    $ref: '#/definitions/NsdmLinks'
    description: This type represents the links to resources that an NSD management notification can contain.
  description: This type represents an NSD management notification, which informs the receiver of the successful on-boarding of an NSD. It shall comply with the provisions defined in Table 5.5.2.9-1. The support of this notification is mandatory. The notification shall be triggered by the NFVO when the "nsdOnboardingState" attribute of a new NSD has changed to "ONBOARDED".

NsdOnboardingFailureNotification:
  type: object
  required:
    - _links
    - id
    - notificationType
    - nsdInfoId
    - onboardingFailureDetails
- `timeStamp` properties:
  - `id`:
    - type: string
    - format: uuid
  - `notificationType`:
    - type: string
    - $ref: '#/definitions/NotificationType'
    - description: Discriminator for the different notification types. Shall be set to "NsdOnboardingFailureNotification" for this notification type.
  - `subscriptionId`:
    - type: string
    - format: uuid
  - `timeStamp`:
    - type: string
    - format: date-time
    - description: Date-time of the generation of the notification.
  - `nsdInfoId`:
    - type: string
    - format: uuid
  - `nsdId`:
    - type: string
    - format: uuid
  - `onboardingFailureDetails`:
    - type: object
    - $ref: '#/definitions/ProblemDetails'
    - _links:
      - type: object
      - $ref: '#/definitions/NsdmLinks'
      - description: This type represents the links to resources that an NSD management notification can contain.
      - description: This type represents an NSD management notification, which informs the receiver of the failure of on-boarding an NSD. It shall comply with the provisions defined in Table 5.5.2.10-1. The support of this notification is mandatory. The notification shall be triggered by the NFVO when the on-boarding of an NSD has failed.

NsdChangeNotification:
- type: object
- required:
  - _links
- id
- notificationType
- nsdId
- nsdInfoId
- nsdOperationalState
- timeStamp

properties:
  id:
    type: string
    format: uuid
  notificationType:
    type: string
    $ref: '#/definitions/NotificationType'
    description: Discriminator for the different notification types. Shall be set to "NsdChangeNotification" for this notification type.
  subscriptionId:
    type: string
    format: uuid
  timeStamp:
    type: string
    format: date-time
    description: Date-time of the generation of the notification.
  nsdInfoId:
    type: string
    format: uuid
  nsdId:
    type: string
    format: uuid
  nsdOperationalState:
    type: string
    $ref: '#/definitions/NsdOperationalStateType'
    description: The enumeration NsdOperationalStateType shall comply with the provisions defined in Table 5.5.4.3-1 of GS NFV_SOL 005. It indicates the operational state of the resource.

  ENABLED = The operational state of the resource is enabled. DISABLED = The operational state of the resource is disabled.

  _links:
    type: object
    $ref: '#/definitions/NsdmLinks'
    description: >-


This type represents the links to resources that an NSD management notification can contain.

description: >-

This type represents an NSD management notification, which informs the receiver of a change of the "nsdOperationalState" attribute of an on-boarded NSD. Changes in the value of the "nsdUsageState" and "nsdOnboardingState" attributes are not reported. The notification shall comply with the provisions defined in Table 5.5.2.11-1. The support of this notification is mandatory. The notification shall be triggered by the NFVO when the value of the "nsdOperationalState" attribute has changed, and the "nsdOperationalState" attribute has the value "ONBOARDED".

**NsdDeletionNotification:**

type: object

required:

- _links
- id
- notificationType
- nsdlId
- nslInfoId
- timeStam

properties:

id:

type: string

format: uuid

notificationType:

type: string

$ref: '#/definitions/NotificationType'

description: >-

Discriminator for the different notification types. Shall be set to "NsdDeletionNotification " for this notification type.

subscriptionId:

type: string

format: uuid

timeStam:

type: string

format: date-time

description: Date-time of the generation of the notification.

nsdlInfoId:

type: string

format: uuid

nsdlId:

type: string

format: uuid
_links:
  type: object
  $ref: '#/definitions/NsdmLinks'
  description: >-
    This type represents the links to resources that an NSD management notifications can contain.

description: >-
  This type represents an NSD management notification, which informs the receiver of the deletion of an on-boarded NSD. The notification shall comply with the provisions defined in Table 5.5.2.12-1. The support of this notification is mandatory. The notification shall be triggered by the NFVO when it has deleted an on-boarded NSD.

PnfdmLinks:
  type: object
  required:
    - pnfdInfo
    - subscription
  properties:
    pnfdInfo:
      type: string
      format: uri
    subscription:
      type: string
      format: uri
  description: >-
    This type represents the links to resources that a PNFD management notification can contain.

PnfdOnboardingNotification:
  type: object
  required:
    - _links
    - id
    - notificationType
    - pnfdId
    - pnfdInfold
    - time_stamp
  properties:
    id:
      type: string
      format: uri
    notificationType:
      type: string
$ref: '#/definitions/NotificationType'
description: >-
   Discriminator for the different notification types. Shall be set to
   "PnfdOnboardingNotification" for this notification type.
subscriptionId:
type: string
format: uuid
timeStamp:
type: string
format: date-time
description: Date-time of the generation of the notification.
pnfdInfoId:
type: string
format: uuid
pnfdId:
type: string
format: uuid
_links:
type: object
$ref: '#/definitions/PnfdmLinks'
description: >-
   This type represents the links to resources that a PNFD management
   notification can contain.
description: >-
   This type represents a PNFD management notification, which informs the
   receiver of the successful on-boarding of a
PNFD. It shall comply with the provisions defined in Table 5.5.2.13-1. The
support of this notification is mandatory. The notification is triggered
when a new PNFD is on-boarded.

PnfdOnboardingFailureNotification:
type: object
required:
  - _links
  - id
  - notificationType
  - onboardingFailureDetails
  - pnfdInfoId
  - timeStamp
properties:
id:
type: string
format: uuid
notificationType:
  type: string
  $ref: '#/definitions/NotificationType'
  description: >-
    Discriminator for the different notification types. Shall be set to
    "PnfdOnboardingFailureNotification" for this notification type.
subscriptionId:
  type: string
  format: uuid
timeStamp:
  type: string
  format: date-time
  description: Date-time of the generation of the notification.
pnfdInfold:
  type: string
  format: uuid
pnfdId:
  type: string
  format: uuid
onboardingFailureDetails:
  type: object
  $ref: '#/definitions/ProblemDetails'
_links:
  type: object
  $ref: '#/definitions/PnfdmLinks'
  description: >-
    This type represents the links to resources that a PNFD management
    notification can contain.
description: >-
  This type represents a PNFD management notification, which informs the
  receiver of the failure of on-boarding a
  PNFD. It shall comply with the provisions defined in Table 5.5.2.14-1. The support of this
  notification is mandatory. The notification is triggered when the on-boarding of a PNFD fails.

PnfdDeletionNotification:
  type: object
  required:
    - _links
    - id
    - notificationType
    - pnfdId
    - pnfdInfold
    - timeStamp
  properties:
id:
  type: string
  description: >-
    Identifier of this notification. If a notification is sent multiple
times due to multiple subscriptions, the "id" attribute of all these
notifications shall have the same value.
notificationType:
  type: string
  $ref: '#/definitions/NotificationType'
  description: >-
    Discriminator for the different notification types. Shall be set to
"PnfdDeletionNotification " for this notification type.
subscriptionId:
  type: string
  format: uuid
timeStamp:
  type: string
  format: date-time
  description: Date-time of the generation of the notification.
pnfdInfoId:
  type: string
  format: uuid
pnfId:
  type: string
  format: uuid
  _links:
    type: object
    $ref: '#/definitions/PnfdmLinks'
    description: >-
      This type represents the links to resources that a PNFD management
      notification can contain.
description: >-
  This type represents a PNFD management notification, which informs the
receiver of the deletion of an on-boarded PNFD. The notification shall
comply with the provisions defined in Table 5.5.2.15-1. The support of
this notification is mandatory. The notification is triggered when an
on-boarded PNFD is deleted.