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

D5.1 - 5G-MEDIA Programming Tools

Work Package: WP5 - 5G-MEDIA APIs and SDK Tools
Lead partner: NET
Author(s): Uğur Acar[NET], Refik Fatih Ustok[NET], Selcuk Keskin[NET], Avi Weit[IBM], David Breitagd[IBM], Francesco Iadanza[ENG], Javier Serrano[UPM]
Delivery date (DoA): 31/08/2018
Actual delivery date: 19/12/2018
Dissemination level: Public
Version number: 1.3 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>March, 21st 2018</td>
<td>NET</td>
<td>Document structure</td>
</tr>
<tr>
<td>0.2</td>
<td>June, 21st 2018</td>
<td>NET</td>
<td>First consolidated version</td>
</tr>
<tr>
<td>0.3</td>
<td>July, 31st 2018</td>
<td>NET</td>
<td>First interim version</td>
</tr>
<tr>
<td>0.4</td>
<td>August, 7th 2018</td>
<td>NET</td>
<td>Revised based on comments from ENG</td>
</tr>
<tr>
<td>0.5</td>
<td>August, 15th 2018</td>
<td>NET</td>
<td>Revised based on comments from IBM</td>
</tr>
<tr>
<td>1.0</td>
<td>August, 27th 2018</td>
<td>NET</td>
<td>Finalized document for submission</td>
</tr>
<tr>
<td>1.1</td>
<td>December, 5th 2018</td>
<td>NET</td>
<td>Revised based on comments at the review report</td>
</tr>
<tr>
<td>1.2</td>
<td>December, 6th 2018</td>
<td>NET</td>
<td>Revised based on comments from ENG</td>
</tr>
<tr>
<td>1.3</td>
<td>December, 18th 2018</td>
<td>NET</td>
<td>Revised based on comments from IBM</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>December, 13th 2018</td>
<td>David Breitgand [IBM]</td>
<td>Comments and suggestions for improvements</td>
</tr>
<tr>
<td>Reviewer</td>
<td>December, 5th 2018</td>
<td>Francesco Iadanza [ENG]</td>
<td>Comments and suggestions for improvements</td>
</tr>
<tr>
<td>Reviewer</td>
<td>August 29th, 2018</td>
<td>David Breitgand [IBM]</td>
<td>Approved</td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 20th 2018</td>
<td>Francesco Iadanza [ENG]</td>
<td>Approved</td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 13th 2018</td>
<td>Gordana Macher [IRT]</td>
<td>Comments and suggestions for improvement</td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 12th 2018</td>
<td>David Breitgand [IBM]</td>
<td>Comments and suggestions for improvement</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Reviewer</td>
<td>August, 3rd 2018</td>
<td>Francesco Iadanza [ENG]</td>
<td>Comments and suggestions for improvement</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 PLIROFORIAKON SYSTMATON KAI EFARMOGON PLIROFORIKIS</td>
<td>SILO</td>
<td>Greece</td>
</tr>
<tr>
<td>04</td>
<td>HELLENIC TELECOMMUNICATIONS ORGANIZATION S.A. - OTE AE (ORGANISMOS TILEPIKIONION 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 GMBH</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 GMBH</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** ............................................................................................................................................................................ 12

**1. INTRODUCTION** ............................................................................................................................................................................. 13

1.1. SCOPE OF THE DELIVERABLE ....................................................................................................................................................... 14

1.2. INNOVATIVE ASPECTS OF 5G-MEDIA SDK ................................................................................................................................... 14

**2. USE CASE 1 WORKFLOW WITH SDK** .............................................................................................................................................. 15

**3. OVERALL 5G-MEDIA SDK** .............................................................................................................................................................. 16

**4. SPECIFICATION FOR 5G-MEDIA SDK** ........................................................................................................................................... 17

4.1. SPECIFICATIONS OF 5G-MEDIA ALL IN ONE USER INTERFACE ........................................................................................................... 17

4.1.1. User Interface ........................................................................................................................................................................... 17

4.1.2. Architecture Design ............................................................................................................................................................... 18

4.1.3. Authentication for User Interface ..................................................................................................................................... 19

4.2. SPECIFICATIONS OF THE 5G-MEDIA EDITOR ............................................................................................................................ 21

4.2.1. Specifications of FaaS UI .................................................................................................................................................... 21

4.2.2. Editor Design .................................................................................................................................................................... 22

4.3. SPECIFICATIONS OF 5G-MEDIA VALIDATOR ........................................................................................................................... 24

4.3.1. Validator Use Cases (NSD - VNFD) ..................................................................................................................................... 24

4.3.2. Validator Design ................................................................................................................................................................ 24

4.3.3. Validator Web UI ............................................................................................................................................................. 25

4.3.4. Validator CLI .................................................................................................................................................................. 26

4.4. SPECIFICATIONS OF 5G-MEDIA EMULATOR .......................................................................................................................... 26

4.4.1. Scenarios ........................................................................................................................................................................ 26

4.4.2. Solution Architecture ................................................................................................................................................... 27

4.4.3. FaaS VNF Emulation ................................................................................................................................................ 28

4.5. SPECIFICATIONS OF 5G-MEDIA CATALOGUE ........................................................................................................................... 28

4.5.1. UI .................................................................................................................................................................................... 29

4.6. SPECIFICATIONS OF 5G-MEDIA SERVICE MONITORING .................................................................................................... 31

4.7. SPECIFICATIONS OF 5G-MEDIA SERVICE PROFILING ........................................................................................................... 32

4.7.1. Functionalities of Profiler Tool ..................................................................................................................................... 33

4.8. END-TO-END WORKFLOW OF UNIKERNELS .......................................................................................................................... 34

**5. CONCLUSIONS** .................................................................................................................................................................................. 39

**6. REFERENCES** .................................................................................................................................................................................. 40

**ANNEX I – ALL-IN-ONE UI INSTALLATION** ........................................................................................................................................ 42

**ANNEX II – TUTORIALS AND USER SCREENS** ........................................................................................................................................... 42

---

Page 5 / 56
ANNEX III – 5G-MEDIA VALIDATOR COMMANDS .................................................................51
ANNEX IV – LEAN OW INSTALLATION ............................................................................53
# List of Figures

Figure 1 – 5G-MEDIA Overall SDK Interactions ............................................................... 13
Figure 2 – Demonstration of a serverless VNF (vTranscoder) emulation ....................... 15
Figure 3 – All in One User Interface ........................................................................... 17
Figure 4 – Command Line Interface ........................................................................... 18
Figure 5 – Sample user interface for FaaS.................................................................... 21
Figure 6 – 5G-MEDIA Editor Architecture .................................................................. 22
Figure 7 – Technology Stack: OSM UI R3 vs R4 ......................................................... 23
Figure 8 – UI Deployment: OSM UI R3 vs R4 .............................................................. 23
Figure 9 – 5G-MEDIA Validator Architecture ............................................................... 25
Figure 10 – 5G-MEDIA Emulator ETSI NFV Reference Architecture Mapping ............... 26
Figure 11 – 5G-MEDIA Emulator Main Scenarios ....................................................... 27
Figure 12 – 5G-MEDIA Emulator High Level Design ................................................... 27
Figure 13 – Catalogue UIs ............................................................................................ 29
Figure 14– Private and Public Catalogue Interaction ..................................................... 30
Figure 15- Service Monitoring Tools Logical Component Diagram .............................. 31
Figure 16- Service Monitoring Tools High Level Design .............................................. 32
Figure 17 - Profiler logical component diagram ............................................................ 33
Figure 18- Traffic generation with Service Profiling ..................................................... 34
Figure 19 - Unikernel image building flow ................................................................... 37
Figure 20- All-In-One Setup ....................................................................................... 42
Figure 21- Login screen of the all-in-one UI ............................................................... 43
Figure 22- Configuration Screen ................................................................................. 44
Figure 23- Dashboard .................................................................................................. 44
Figure 24- Tutorials Screen .......................................................................................... 45
Figure 25- Interactive Lean OW CLI Tutorial ............................................................. 46
Figure 26- Interactive OSM Client Tutorial ................................................................. 47
Figure 27- Validator: Example of valid descriptor ....................................................... 48
Figure 28- Validator: Example of invalid descriptor .................................................... 48
Figure 29- Editor ......................................................................................................... 49
Figure 30- Instantiating the network services ............................................................... 49
Figure 31- Service Profiling................................................................. 50
Figure 32- Service Monitoring........................................................... 50
Figure 33- Packaging Tool................................................................. 51
List of Tables

Table 1 – Authentication API ........................................................................................................ 19
Table 2 – Response of authentication server ............................................................................. 20
Table 3 – Decoded token .............................................................................................................. 20
Table 4 – Status codes of validator .............................................................................................. 25
Table 5 – Example configuration files .......................................................................................... 35
Table 6 – Another example configuration file ................................................................................ 36
Table 7 – Template of API call ..................................................................................................... 37
Table 8 – Command for help ........................................................................................................ 51
Table 9 – Schema validation ......................................................................................................... 51
Table 10 – NSD validation with VNFD path given ......................................................................... 52
Table 11 – NSD validation without VNFD path given .................................................................... 52
Table 12 – VNFD validation ........................................................................................................... 52
Table 13 – VNF package validation ............................................................................................... 53
## Definitions and acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Authentication, Authorization and Accounting</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>E2E</td>
<td>End-to-End</td>
</tr>
<tr>
<td>EMU</td>
<td>Emulator</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standard Institute</td>
</tr>
<tr>
<td>FaaS</td>
<td>Function as a Service</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HTTP Secure</td>
</tr>
<tr>
<td>K8S</td>
<td>Kubernetes</td>
</tr>
<tr>
<td>KVM</td>
<td>Kernel-based Virtual Machines</td>
</tr>
<tr>
<td>LXD</td>
<td>Linux Container Hypervisor</td>
</tr>
<tr>
<td>MANO</td>
<td>Management and Orchestration</td>
</tr>
<tr>
<td>MAPE</td>
<td>Monitor Analyse Plan Execute</td>
</tr>
<tr>
<td>MEC</td>
<td>Mobile Edge Computing</td>
</tr>
<tr>
<td>NBI</td>
<td>North Bound Interface</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>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSM</td>
<td>Open Source MANO</td>
</tr>
<tr>
<td>OSM IM</td>
<td>Open Source MANO Information Model</td>
</tr>
<tr>
<td>OW</td>
<td>OpenWhisk</td>
</tr>
<tr>
<td>PED</td>
<td>Profiling Experiment Descriptor</td>
</tr>
<tr>
<td>PoP</td>
<td>Point of Presence</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RO</td>
<td>Resource Orchestrator</td>
</tr>
<tr>
<td>SDK</td>
<td>Service Development Kit</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Networking</td>
</tr>
<tr>
<td>SO</td>
<td>Service Orchestrator</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SVP</td>
<td>Service Virtualization Platform</td>
</tr>
<tr>
<td>TSDB</td>
<td>Time Series Data Base</td>
</tr>
<tr>
<td>UC</td>
<td>Use Case</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>VIM</td>
<td>Virtualized Infrastructure Manager</td>
</tr>
<tr>
<td>VLD</td>
<td>Virtual Link Descriptors</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>VNF-FG</td>
<td>VNF Forwarding Graph</td>
</tr>
</tbody>
</table>
Executive summary

The 5G-MEDIA project focuses on building an integrated programmable service platform that facilitates the design, development and deployment of media services, exploiting the advancements of 5G technology. To achieve this goal and improve developers’ time efficiencies, a service development kit (SDK) has been developed to provide a set of open-source tools that support the rapid development of media applications using a DevOps approach.

The scope of this document is to provide a comprehensive overview of programming tools and the specifications for the SDK, discussing the first version of the workflows of SDK considering the use case 1 (UC1) of deliverable D2.2 - 5G-MEDIA Requirements and Use Case Refinement and end-to-end workflow of unikernels’ packaging. Specifications of 5G-MEDIA all-in-one user interface (UI) are also presented in this document. In addition to all-in-one UI, UIs for specific tools such as private catalogue are also demonstrated. Furthermore, implementations of Function as a Service (FaaS) Emulation and FaaS Command Line Interface (CLI) Tools are given. The programming tools and open-source frameworks that are used in 5G-MEDIA project are further discussed in this document. Finally, the installation steps of the all-in-one UI and the screenshots for each programming tool embedded in the all-in-one UI are provided.
1. Introduction

The goal of this document is to provide an initial design of the 5G-MEDIA SDK, which constitutes the interface to application developers/platform operators/infrastructure owners allowing the creation of applications, network services (NSs) and functions, as well as the set of tools that enable the developers in the implementation, packaging, deploying, monitoring or analysing of artefacts. The aim of the 5G-MEDIA SDK is to speed up the application development process using the 5G-MEDIA tools with an all-in-one UI. In this document, each tool is discussed in detail.

This document is organized as follows. In Section 2, the workflow of use case 1 is described from the perspective of the SDK users. In Section 3, the overall design of the 5G-MEDIA SDK is discussed giving high-level functionality of each tool. Specifications of 5G-MEDIA SDK are explained in detail in Section 4, such as all-in-one user interface (UI), architecture design, editor, validator, emulator and specifications of a public catalogue UI and REST API, monitoring and profiling. Furthermore, FaaS Virtual Network Function (VNF) UI specifications are also discussed in Section 4.2. Following the editor specs, the design of the validator and the CLI tools are discussed in Section 4.3. Section 4.4 presents the emulator including the scenarios and the solution architecture. Private catalogue design with UI and the REST API is explained in Section 4.5. Service monitoring tool and its high-level design is explained in Section 4.6. Following the monitoring tool, the profiling tool is presented in Section 4.7. Then an end-to-end workflow of unikernels, managed the SDK, is presented in Section 4.8. We conclude this deliverable in Section 5.

![Figure 1 – 5G-MEDIA Overall SDK Interactions](image)

5G-MEDIA overall SDK architecture is given in Figure 1 where the main steps are reported: the validation of the descriptors via the Validator, the import (if valid) of the descriptors onto the private catalogue, the transfer to the public catalogue of the Service Virtualization Platform (SVP) and the instantiation of the related VNF/NS through Open Source MANO (OSM) UI.
1.1. **Scope of the Deliverable**

This deliverable provides an initial design of the 5G-MEDIA SDK components (All-in-One User Interface, Editor, Validator, Private Catalogue, Emulation Toolkit, Service Monitoring and Profiling). The “Packaging and Integration Tools” is not part of this deliverable since it will be discussed in deliverable *D5.4 - Packaging and Integration Tools evaluation and setup.*

1.2. **Innovative Aspects of 5G-MEDIA SDK**

5G-MEDIA SDK consists of many components that bring innovation compared to the existing platforms. These innovations can be listed as follows:

- **5G-MEDIA SDK includes an all-in-one UI** which enables developers’ access all SDK tools from a single interface. This improves developer’s efficiency and provides a smooth user experience.

- **FaaS Emulation implemented using Lean OpenWhisk (Lean OW) and FaaS CLI Tools** allow media application developers (NS developers) to leverage the FaaS programming model and quickly develop and evaluate value added code while relieving them from the infrastructure management concerns.

- **CLI tools for unikernels**, enable unikernel development, which provides many benefits compared to a traditional Operating System (OS), including improved security, smaller footprint and consequent faster boot time.

- **VNF/NS Emulation toolkit including service profiling and monitoring tools** allows doing load testing on a media application (profiling) and provides visualization of pre-defined performance metrics (monitoring) in an emulated multi - Virtualized Infrastructure Manager (VIM) environment including OpenStack and OpenWhisk (OW) emulators. This allows media application developers to test and verify their applications functionality, debug and fine-tune them before deploying to a production environment.
2. Use Case 1 Workflow with SDK

Figure 2 shows a typical SDK workflow dealing with a FaaS VNF. The workflow comprises four steps as outlined below. Steps 1—4 demonstrate emulation of the vTranscoder serverless VNF which is part of the UC1 VNF Forwarding Graph (VNF-FG) (see deliverable D4.1 - 5G-MEDIA Catalogue APIs and Network App). Furthermore, in this document we outline the additional monitoring and profiling steps and provide a more detailed description of them in the deliverable D6.1 - 5G-MEDIA Use Case Scenarios and Testbed.

1 - Pre-onboarding

A developer uses wskdeploy against Lean OW to create OW actions, whose code attributes point to the relevant Docker containers that embody the VNF logic and are stored in some repository(ies) (e.g., in DockerHub). The meta-data created in the Lean OW database for the actions and the container images, comprise the VNFs’ “images”. Pushing the implementation Docker containers into repository(ies) and creating the corresponding actions in the Lean OW is referred to as “pre-onboarding”. This process in SDK is API-wise identical to the “pre-onboarding” onto a production system (see deliverable D3.2 - Specification of the 5G-MEDIA Serverless Computing Framework, Section 3.6).

2 - Onboarding

The developer creates the Virtual Network Function Descriptors (VNFDs) and a Network Service Descriptor (NSD) for the actions and then exports them for validation. The VNFDs and the NSD are onboarded to the public catalogue only after the validation succeeds.

3 – Instantiation in Emulator

The developer instantiates vTranscoder by using 5G-MEDIA Editor with help of configuration microservice (also referred as Conf MS in Figure 2).
4 - Emulation (Simulation)

After the instantiation of the vTranscoder in the emulated environment is completed, a 3D traffic simulator is run to test the action on the Lean OW machine [Lean]. If there is no error, the emulation of the vTranscoder in the developer environment is finished and the serverless VNF is a candidate for deployment to the production environment.

5 - Profiling

This optional step is required whenever a developer wants to do some load testing on the emulated VNF. The developer creates one or more test container/s pre-configured with the IP address, the required port numbers to send and receive traffic, a media input file and, the duration of the testing process over SDK’s Profiling Tool UI. The media input file is used for configuring various media traffic generation types. Thus, after the creation of one or more test container/s, pre-configured media traffic is forwarded to the emulated VNF or eventually NS so that developers can evaluate its performance under different traffic load conditions.

6 - Monitoring

During the profiling phase, monitoring tools are available for media application developers to gather and centralize monitored metrics into a local database, so developers can analyse them through a web-based dashboard. As a result, they can optimize or fine-tune the performance of their applications.

3. Overall 5G-MEDIA SDK

The all-in-one UI consists of the links to the tools which are web or command line interfaces which give smooth user experience while switching from one tool to another:

- **the editor** is a web-based application whose frontend is running in the media application developer’s browser. Its main purpose is to assist with creating and editing of NS projects and their descriptors. This is done by simplifying repetitive and complicated tasks. E.g., the NSDs are visualized as a graph of VNFs instead of using a textual representation.

- **the validator** is the tool which is responsible for validation of the SDK projects, packages, services and functions. The validator is provided in the form of both the CLI tools and a web UI.

- **the emulator** allows media application developers to locally prototype and test complete NS chains in realistic end-to-end multi-Point of Presence (PoP) scenarios in multi-vim environments. It allows the execution of real network functions, packaged as Docker containers, in emulated network topologies running locally on the media application developer's machine.

- **the profiler** allows developers to deploy NSs on the emulation platform and perform load testing under different resource constraints. During these tests a variety of metrics can be monitored which allows service developers to find bugs, investigate problems or detect bottlenecks in their services. The **service monitoring tool** guides the media application developer by providing performance data of the emulated service and its components in a quantitative manner during profiling.
4. **Specification for 5G-MEDIA SDK**

5G-MEDIA SDK provides a set of tools that helps developers easily implement and deploy new media related network applications to the SVP. The SDK provides a programming model for application developers by offering several functionalities such as private NSD/VNFD catalogue, editor, validator, emulator, service monitoring and profiling tools, which allow defining complex media services consisting of multiple VNFs.

4.1. **Specifications of 5G-MEDIA All in One User Interface**

Since 5G-MEDIA SDK has a set of tools with several different functionalities, integrating all of them into a single interface is very helpful as it increases the productivity of the developers by empowering them with a quick and convenient control. The interface consists of the set of navigation items which give fast access to each SDK tool.

4.1.1. **User Interface**

The proposed interface is a web-based UI embedded into a desktop container (a.k.a desktop application) which can be used by the developers on their personal computers. Figure 3 shows the layout of the interface implemented in the current iteration. The interface has links which open each SDK tool. Hereby, the developer can easily use any tool of the SDK (i.e., OSM CLI, Lean OW CLI) through the interface without searching for a separate entry point of each tool.

![Figure 3 – All in One User Interface](image-url)
Since a CLI might be needed for some use cases, the navigators of the interface can execute an external command line interface application, e.g. PuTTY, Git, etc. The overview of the concept of opening an external application is presented in Figure 4.

![Figure 4 – Command Line Interface](image)

Furthermore, the SDK comprises tutorials about how to use programming tools based on open source tools such as the Xterm.js [XTERM] that provides fully featured terminals to the users on a web UI. Therefore, the users could execute the sample commands on an embedded terminal screen using the respective tool described in the tutorial. The screenshots of the tutorials are given in Annex II.

Installing of SDK All-In-One UI is described in detail in Annex I.

### 4.1.2. Architecture Design

The proposed interface consists of two platforms Electron [Electron] and React [React]. React, which is one of the Facebook’s first open source projects, is a JavaScript library for building UIs. First, encapsulated components that manage their own state are built, and then they are composed to make complex UIs. Electron is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. The Electron framework lets the developer write cross-platform desktop applications. The proposed interface is covered by the Electron framework to bring it an ability for calling an external application without losing the response speed.

In order to implement all-in-one interface, a single page is created with the React framework and then it is split in two pages that are the sidebar page to implement navigation links for the tools and the content page. Navigation items are used at the sidebar page for each tool.
included in the all-in-one UI. A navigation item makes a connection between the related tool and the content page based on the route information parameter which uses an identical value for each page of the tool. A webview component of Electron is used at the content page of which presents the content using Uniform Resource Locators (URLs) given in the configuration file.

### 4.1.3. Authentication for User Interface

Authentication based on tokens is used to access 5G-MEDIA Platform services from the SDK. The SDK developers invoke the Authentication, Authorization and Accounting (AAA) Service API to login and gets the token to be added as a specific HTTP header to all subsequent calls to the Catalogue and some other 5G-MEDIA components such as Monitoring to gather metrics. The API request and response are given in Table 1.

<table>
<thead>
<tr>
<th>Table 1 – Authentication API</th>
</tr>
</thead>
</table>

**LOGIN**

**Request:**

POST /api/authenticate HTTP/1.1  
Host: aaa-server  
Content-Type: application/json  
{"username":"user1","password":"pass1"}

**Response:**

{
  "id_token": "my access token",
  "refresh_token": "my refresh token"
}

**ACCESS TO PROTECTED RESOURCE**

**Request:**

GET /something  
Host: catalogue-server  
Content-Type: application/json  
Authorization: Bearer my access token
For a comprehensive description of all the options supported (such as refresh tokens) and protocols supported (e.g., OpenID Connect [OpenID]), please refer to the section about security in the deliverable D4.1 - 5G-MEDIA Catalogue APIs and Network Apps.

An important feature of the AAA tokens is that they contain information about the logged in user and adopt JWT as first implementation. For example, a given answer from the AAA is shown in Table 2.

<table>
<thead>
<tr>
<th>Table 2 – Response of authentication server</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;alg&quot;: &quot;HS256&quot;,</td>
</tr>
<tr>
<td>&quot;typ&quot;: &quot;JWT&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;id_token&quot;:</td>
</tr>
<tr>
<td>&quot;eyJhbGciOiJIUzUzUxMiJ9.eyJzdWIiOiJkZXI</td>
</tr>
<tr>
<td>xIw1YXV0aCI6I1JPTfV0VSvK1DRV9ERVZf</td>
</tr>
<tr>
<td>TE9QRIiLCJleHAiOjE1MzE4NjAzMDN9.k</td>
</tr>
<tr>
<td>xDNUC3-Auq6jM2_k_CaqtuXRQsfzc7prb86r</td>
</tr>
<tr>
<td>w9WjnGzApgs9qV3S4gkapc9C5BDikbMQCGJ</td>
</tr>
<tr>
<td>5nJFEgqnmBw&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

The decoded HS256 user info contained in the token body is illustrated in Table 3.

<table>
<thead>
<tr>
<th>Table 3 – Decoded token</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;sub&quot;: &quot;developer1&quot;,</td>
</tr>
<tr>
<td>&quot;auth&quot;: &quot;ROLE_SERVICE_DEVELOPER&quot;,</td>
</tr>
<tr>
<td>&quot;exp&quot;: 1531860303</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

This method is used by the SDK to customize the UI depending on the role of the logged in user. For example, developers use the private catalogue and CLIs to generate VNF descriptors while operators use the public catalogue to upload generated descriptors to the real world environment.
4.2. Specifications of the 5G-MEDIA Editor

The 5G-MEDIA Editor is a web-based application whose frontend is running in the browser of the media application developer. In fact, the editor runs in the developer’s local OSM environment so that the developer can manage emulation environments such as vim-emu [ETSIvim] and Lean OW. It is the main UI for adding new data centres/VIMs, designing, validating, onboarding media applications (VNFs or NSs) to the private or public catalogue by building VNF-FG and instantiating or shutting down an NS deployed in the emulated data centres. This UI also visualizes topological dependencies or interconnection of involved VNFs of VNF-FG and descriptions of individual VNFs. This interface can also be used to conveniently manage lifecycle operations on VNFs and NSs such as their instantiation and termination. Authentication will be required for onboarding media applications from a private catalogue to the public catalogue. Two user roles have been identified: NS composer and media application developer. An NS composer can only onboard NS descriptors retrieving the VNF descriptors from the public catalogue via the SDK. On the other hand, a media application developer can implement custom VNFs and have the rights for onboarding both NSDs and VNFDs to private and public catalogues.

4.2.1. Specifications of FaaS UI

Figure 5 shows an OSM UI design for FaaS featuring a FaaS button that is considered to be added to the composer in the next release of the 5G-MEDIA SDK. The developer can list the FaaS VNFs by pressing this button and can edit them using the editor. The FaaS VNFs are then updated if the developer presses the update button, then uploaded to the OW using OW API (this can be either Lean OW or a full OW installation). The UI will also include a button for the
FaaS accounts under the account sections like the Software-Defined Networking (SDN) or the VIM accounts. Hence, the developers will be able to store access credentials of the OW through this menu. These access credentials will be used for authorization in any OW API calls related to the FaaS specific configurations.

It should be noted that since the project will migrate to OSM r4 and eventually there will also be a possible migration to OSM r5, diversions from the proposed design are likely to occur. As the project moves forward we will adapt and re-evaluate this design in an agile fashion to address the changes in OSM.

4.2.2. Editor Design

In this section, editor solution architecture and open source tools that are used in the design are given.

4.2.2.1. Solution Architecture

The architecture of the 5G-MEDIA Editor is shown in Figure 6. The 5G-MEDIA Editor is split into a backend and a frontend part that can be executed by different servers to improve the scalability of the 5G-MEDIA Editor. Figure 6 shows this modularization and how the components can be split between the backend and frontend modules. In this setup, the application logic is encapsulated in the backend module and exposed via a RESTful interface to the frontend parts. The backend module also interfaces with the OSM system via the Northbound REST API based on ETSI NFV-SOL-005 [ETSIoslo5].

![Figure 6 – 5G-MEDIA Editor Architecture](image)

The editor and its components could either run on the developer’s local computer where they can be configured locally or it could be hosted by a central editor server as a part of the 5G-MEDIA SDK. The key point for deploying the editor on either a local computer or a hosted server is to configure access from the application server to the OSM North Bound Interface (NBI).

4.2.2.2. Open Source Tools

5G-MEDIA SDK leverages the OSM UI tool [OSMr3], [OSMr4] to meet the requirements of the Network Function Virtualization (NFV)/SDN network aligned with the ETSI NFV model.
Recently, OSM released a newer version (a.k.a Release Four (R4)) of the UI component called “light-ui”, which is completely different from the previous releases in terms of used programming languages and technology stack [OSMr4]. The UI tool in OSM R3 is implemented in the Javascript programming language and uses React and Flux [Flux] frameworks on the front end and a Nodejs [Node]s framework on the backend implementation, whereas UI in OSM R4 is implemented in the Python programming language and uses Django [Django] web development framework (Figure 7).

Another major difference between OSM R3 and OSM R4 relates to the installation process and requirements of the UI tool. As shown in Figure 7, the UI tool in R3 is installed on the same LXD container with the Service Orchestrator (SO) module. However, as shown in Figure 8, the UI in R4, is not installed on the LXD container as it is dockerized as the light-UI. It can be installed independently to either any platform supporting Docker containers or to an application server which supports Python applications. Therefore, development process and extending functionalities of the UI are simplified in terms of code complexity (e.g. easy to use open source web framework in R4) and development environment setup time.

OSM R3 UI supports the following functionalities [OSMr3]:

- Controlling and adding VIM accounts;
- Onboarding packages, VNFs and network services;
- An interface for NS composition and visualization of VNF-FG and its elements;
- Instantiation of a NS or a media application.
OSM R4 UI supports the following functionalities [OSMr4]:

- Has a smaller footprint (2CPUs, 4 GB RAM)
- Drag and drop descriptors interface is supported to add new services;
- Controlling and adding VIM accounts;
- Onboarding packages, VNFs and NSs to OSM catalogue;
- An interface for NS composition;
- Instantiation of a NS or a media application.

4.3. Specifications of 5G-MEDIA Validator

4.3.1. Validator Use Cases (NSD - VNFD)

Topology and Orchestration Specification for Cloud Applications (TOSCA) language [TOSCA2018] is created by The Organization for the Advancement of Structured Information Standards (OASIS) to describe a topology of cloud-based web services, their components, and relationships in a uniform and vendor-independent way [SONATAD3].

Most of the template languages can be considered as candidates to describe VNFs as they can be used to describe web services and compute requirements. However, they may fail to describe complete NS as they usually lack a detailed description of network functionality and requirements.

ETSI is currently working on descriptors that cover virtual network functions as well as NSs, however this is still in a draft stage. [ETSiDep].

In 5G-MEDIA, media applications (they are just specialized NS), are chains of one or more VNFs that, once orchestrated, realize the desirable end-to-end functionality. Each chain is constructed and maintained by the SO using the information which is included in the corresponding NSD. The characteristics of media applications are explained in detail in deliverable D3.1 - Initial Design of the 5G-MEDIA Operations and Configuration Platform.

In addition to an NSD and a set of VNFDs, a 5G-MEDIA service is also characterised by a package descriptor for the overall service. The package descriptor is generally a compressed file which includes the VNFD and NSD files. Each of these descriptors follows a YAML schema language format. The 5G-MEDIA Validation tools have been developed to check if given descriptors are compliant with the 5G-MEDIA descriptors which are aligned with ETSI MANO [ETSINFV]. These tools can be used across various modules of this project.

4.3.2. Validator Design

In this section, architecture and open source libraries that are used in the 5G-MEDIA validator are given (Figure 9). The inputs of the validator can be specified as follows

- NSD/VNFD YAML files
  - NSD/VNFD files can be validated with a path where referred files are located.
- JSON Schema File
- The user can upload the schema file of their choice against which they want to validate their descriptor files.

![Diagram of Validator Architecture](image)

**Figure 9 – 5G-MEDIA Validator Architecture**

The validator first converts the input file into a JSON object and then validates the parsed JSON object against a schema. The validator returns success or failure result, also specifying a reason and an error code. The validator uses both the TOSCA [TOSCA2018] and the OSM Information Model (IM) [ETSIOSM] as schema definitions to validate NSD and VNFD files. Also, an open-source tool, called Tv4, have been used to validate the JSON formatted VNFD and NSD files against the input schema file [Tv4web].

### 4.3.3. Validator Web UI

The validator tools are also presented as a web UI which could improve the user experience. As Tv4 open-source tool [Tv4web] supports all modern browsers, it could be seamlessly applied in the UI. In this UI, the developer only needs to select the descriptor file to be validated and the schema file to be used for validation. The developer could then have the descriptor file (YAML or JSON) in an editable text box of the UI and make any necessary changes. At the bottom of the UI, the developer could view the validation results with the code given in Table 4 and also information about how to make the descriptor file valid according to the given schema. The developer could also export their validated descriptors as a compressed package in tar.gz format. The screenshots of the validator web UI are provided in Figure 27 and Figure 28 of Annex II.

**Table 4 – Status codes of validator**

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success Codes</strong></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>JSON Schema File is valid</td>
</tr>
<tr>
<td>201</td>
<td>NSD or VNFD File is valid.</td>
</tr>
<tr>
<td><strong>Failure Codes</strong></td>
<td></td>
</tr>
<tr>
<td>501</td>
<td>Validator cannot locate the file to be validated.</td>
</tr>
<tr>
<td>502</td>
<td>The schema file is not valid.</td>
</tr>
<tr>
<td>503</td>
<td>NSD file is not valid</td>
</tr>
<tr>
<td>504</td>
<td>VNFD file is not valid</td>
</tr>
</tbody>
</table>
4.3.4. Validator CLI

The 5G-MEDIA SDK also includes CLI tools to validate the descriptors. An open source tool of Tv4-cmd [Tv4] which uses Node.js as a CL parser has been considered to develop the Validator CLI tools. Similar to Validator web UI, Validator CLI tool also return the codes given in Table 4. More information about the usage of the Validator CLI tools can be found in Annex III.

4.4. Specifications of 5G-MEDIA Emulator

The 5G-MEDIA Emulator facilitates local prototyping and testing of NSs in realistic end-to-end multi-PoP scenarios in multi-vim environments. This platform allows execution of real network functions packaged as Docker containers in emulated network topologies running locally on the developer's machine. The emulation platform not only offers Openstack-like APIs for each emulated PoP but also provides OpenWhisk APIs via Lean OW, which is a low footprint OW and which can therefore be installed to developer’s personal computer with minimal resources [OWMed]. 5G-MEDIA Emulator also integrates with management and orchestration (MANO) system, which is responsible to deploy and manage the NSs tested in the emulated environment.

Figure 10 shows the scope of our solution and its mapping to the ETSI NFV reference architecture in which it replaces the network function virtualization infrastructure (NFVI).

Figure 10 – 5G-MEDIA Emulator ETSI NFV Reference Architecture Mapping

4.4.1. Scenarios

5G-MEDIA Emulator supports Docker based containers in the emulated network. The FaaS based VNFs are not images in the regular sense. They are artefacts such as actions, triggers, rules and packages that together comprise the FaaS based VNF image. FaaS VNFs can also be Docker container images accompanied by some metadata. Figure 11 depicts the three
different VNF-FG scenarios supported, including only VNFs (b), only FaaS VNFs (c) or a mixture of them (a).

![Diagram](image)

*Figure 11 – 5G-MEDIA Emulator Main Scenarios*

### 4.4.2. Solution Architecture

5G-MEDIA Emulator leverages vim emulator (a.k.a, vim-emu/son-emu) and Lean OW to provide an emulated network in the developers’ environment. The design of vim-emu is based on Containernet [Containernet] which extends the Mininet emulation framework [Mininet]. Furthermore, Containernet 1.0 supports using standard Docker containers as VNFs within the emulated network. Also, Containernet 2.0 release (March 2018) adds experimental support for KVM based VMs into the emulated networks. It is further planned to have a mixed environment with containers and the VMs in an emulated environment. The Emulator mimics FaaS framework of the SVP, which comprises Apache OW and K8s. The emulated environment uses Lean OW uses Minikube [Mini] which is a Kubernetes distribution used as a standard development environment on a single machine. Minikube runs a single node Kubernetes cluster inside a VM on a developers’ laptop and supports only Docker containers. Figure 12 shows a high-level design of 5G-MEDIA Emulator. Furthermore, it also allows adding and removing containers from the emulated network at runtime while supporting the use of the emulator like a cloud infrastructure in which we can start and stop compute resources (in the form of containers) at any point.

![Diagram](image)

*Figure 12 – 5G-MEDIA Emulator High Level Design*
4.4.3. FaaS VNF Emulation

The FaaS emulator is an all-in-one development environment comprising Lean OW and Minikube. Furthermore, the FaaS emulator is not fundamentally different from the SVP, which uses a full clustered installation of Apache OpenWhisk and of a K8s cluster. A FaaS VNF that is to be emulated, should be pre-onboarded into Lean OW in a regular way using a wskdeploy tool. This tool allows to define the OpenWhisk action that implements this VNF. As a next step, a VNFD should be defined for the VNF and onboarded to the catalogue via OSM. Again, this process is not different from that of the regular onboarding. Finally, the VNFD should be instantiated using OSM. The VNF instance and its status will be shown in the OSM UI. The instance can be terminated from OSM on the user’s discretion as if it was a regular VNF instance.

More information about the internal design of FaaS VIM can be found in deliverable D3.2 - Specification of the 5G-MEDIA Serverless Computing Framework. Installation and usage of Lean OW with OSM and Minikube are described in Annex IV. Please refer to deliverable D3.2 - Specification of the 5G-MEDIA Serverless Computing Framework (Sections 4.1 and 4.2) for the details of configuring a VNF at the time of instantiation (Day 0) and dynamic configuration when the VNF starts executing (Day 1), respectively.

4.5. Specifications of 5G-MEDIA Catalogue

The catalogue, also named as 5G App and Service Catalogue is a fundamental part of the 5G-MEDIA SVP and the 5G MANO architecture in general. In fact, it provides all the functionalities needed for CRUD operations on NS and VNF Packages, i.e., onboard, update, delete and information fetch as APIs. The catalogue design is based on the concept of a novel generalized and extensible format for describing NSs along with the VNFs composing them. This catalogue leverages the design and development background of the SELFNET Catalogue [Selfnet] and the latest ETSI NFV MANO specifications [ETSIr2], [ETSIos] and [ETSIosI5]. The catalogue allows to handle also the onboarding and management of other types of descriptors/packages in the IT domain, like Mobile Edge Computing (MEC) applications, SDN applications and functions developed through the FaaS paradigm.

Two types of catalogues are considered: public and private. The public catalogue is meant for the SVP users and requires authentication. A private catalogue is used by the developer on his/her local personal computer without the need to authenticate through the 5G-MEDIA SDK. The public catalogue contains the available VNFDs, NSDs and repository of VNF images on the SVP while the private catalogue consists of those just for development purposes.

In the following section, the catalogue UI is given.

1 [https://github.com/apache/incubator-openwhisk-wskdeploy/blob/master/README.md].
4.5.1. **UI**

The UI of the 5G-MEDIA private catalogue is same as that of the public one. The 5G-MEDIA public catalogue is not part of the SDK *per se*, however we include a brief description of it to make this deliverable self-contained. The UI of the catalogue is developed as part of the WP4 and given in Figure 13-a and Figure 13-b. The details of the public catalogue and REST API can be found in the deliverable of D4.1 – 5G-MEDIA Catalogue APIs and Network Apps.

(a)  
(b)  

*Figure 13 – Catalogue UIs*

4.5.2 **High-level Design**

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

- Mano plugin: It can be considered as a connector to a specific underlying MANO framework. It contains the following components:
  - API handler
  - Translator
  - vim-emu agent
  - vim plugin

- Policy engine: It is intended to be used by the Catalogue owner for the management of the user’s subscription and policies configuration.

Figure 14 shows catalogue positioning in the 5G-MEDIA platform. Developers use Applications and Services Private Catalogue inside SDK as a local repository of VNF/NS descriptors and container images.

In 5G-MEDIA, developers can onboard their NSDs or VNFDs to the public catalogue after the validation and then validated descriptors can be processed by the dispatch engine. The dispatch engine is also connected to the MANO catalogue for which the OSM catalogue is considered in 5G-MEDIA. The interaction point with SDK is the NBI, which is based on ETSI-NFV-SOL 005 standards [ETSiSol5].

Details of the main components can be found in deliverable D4.1 – 5G-MEDIA Catalogue APIs and Network Apps.
4.6. Specifications of 5G-MEDIA Service Monitoring

In the 5G network context, SDN and NFV offer new ways of managing the network and the connected traffic processing nodes. This results in a programmable way to design VNF forwarding graphs to implement a media application or an NS, effectively automating the deployment and management of such applications or services. Virtualisation techniques such as VMs or containers encounter a significantly growing interest as telecom services which are previously managed by hardware applications now increasingly experience the softwarization. However, these containerization and virtualization techniques require different kind of customizations and configurations. In order to avoid these bugs and decrease the time required between the development process and the operations side of a developed media application, 5G-MEDIA SDK provides monitoring tools that allow rapid testing and verification of any modified parameter.

The 5G-MEDIA SDK has a set of monitoring tools available for media application developers that can gather and centralize monitored metrics into a local database. Metrics can be queried from either the VNFs deployed in the emulator or the Media Service Monitor Analyse Plan Execute (MAPE) of SVP. After the metric data is stored in the local database, further analysis can take place to debug or optimize the performance of the monitored VNF or of a service. Figure 15 below describes logical component diagram of this service monitoring tools.

![Logical Component Diagram](image)

**Figure 15: Service Monitoring Tools Logical Component Diagram**

Figure 16 below shows the detailed flow of the monitoring tools. The container monitoring tool in the emulator (i.e, cAdvisor) gathers metrics related to the compute, storage or network and these metrics are exported by starting a query scheduling. This scheduling involves an SDN controller which uses the OpenFlow protocol to inquire the packet and byte counters of the virtual network interfaces of the emulated service. In addition to the interface counters, some particular flow counters can be installed so that a better, fine-grained network traffic monitoring can be provided. Monitoring metrics of the emulator are then pushed to a Metrics Gateway (i.e Prometheus Push Gateway) from where the metrics will be pulled by the external Metrics Database (Prometheus database) in the SDK. The emulator has a REST API to control the export of metrics to the Metrics Gateway.
The 5G-MEDIA SVP has a monitoring framework, as a module of the Media Service MAPE component, to collect and store metrics about resources and running NSs and media applications (See deliverable D3.3 - Specification of the 5G-MEDIA QoS Control and Management Tools). In addition, the SVP can be instructed to export several metrics from the monitoring database of the MAPE component. The media application developer using the SDK can also request monitoring data from this database. In OSM R4, monitoring capabilities have been extended further by integrating a Monitoring Module to collect telemetry related to VMs and VNFs, enabling alarms and binding them to the orchestrated NSs [OSMr4]. The Monitoring Module uses plugins enabling the interaction with external monitoring tools and drive metrics and configuration updates to them. This way, the tools in the SDK can help further analyse or debug the service performance. The gathered metrics can be visualized using the visualization tool (i.e Grafana UI [Grafana]) which visualizes the inquired metrics from the Metrics Database using a web-based UI.

The FaaS VNF monitoring architecture is described in detail in deliverable D3.2 - Specification of the 5G-MEDIA Serverless Computing Framework. In SDK, the FaaS VNFS monitoring is fully aligned with the rest of the architecture. In SDK, instantiation of the FaaS VNFS via the OSM FaaS Plugin causes VNF to be offloaded to Minikube (an all-in-one Kubernetes) by Lean OW. Similarly to the monitoring architecture of SVP, in SDK, we leverage the Prometheus monitoring framework that obtains metrics from cAdvisor (the native metric producer of the Kubernetes) and stores them in the Prometheus Time Series Data Base (TSDB). Grafana dashboards are used to query Prometheus TSDB to obtain compound metrics related to the FaaS VNFS execution.

4.7 Specifications of 5G-MEDIA Service Profiling

The 5G-Media Profiling tool provides load testing under various resource constraints on NSs which are deployed on the emulation platform. A variety of metrics can be monitored during these tests and these monitoring metrics can help service developers find bugs, detect
bottlenecks or investigate problems in their media applications. The 5G-Media profiling tool aims to automate big parts of this workflow and thus support service developers as much as possible. To achieve this aim, the 5G-Media profiling tool creates one or more test containers to send different types of media traffics according to the use case. For example, a 3D media file of a Virtual Reality (VR) game can be used to test Tele-Immersive (TI) applications. These containers can be automatically deployed on the 5G-Media emulator. The logical component diagram of the 5G-Media Profiling tool is presented in Figure 17.

![Profiler logical component diagram](image)

As the profiler creates descriptors for experiment purposes, which results in instantiating VNFs under specific resource allocations and measuring results, for FaaS the developer needs to keep the same sorts of metrics as for any other VNF; the difference is that one can visualise the resource consumption metrics through K8s. The profiler gets the data from the time series database available in the 5G-MEDIA (Prometheus) and examines the association among the resources consumption and the overall traffic load served. In addition to the analysis results, simple visualisations are produced based on the Grafana tool.

### 4.7.1. Functionalities of Profiler Tool

In Figure 17, the 5G-Media Profiling tool with profiling workflow is presented. First, the user creates a media file to make an offline testing of network service on the emulation platform. Then, the network service is instantiated on the emulation platform to get the required ports for traffic testing. These are the configuration ports to make handshaking between the test VNFs and the VNFs’ input port to receive the data and output port to send the response. The
The purpose of the profiler is to create stress on the network service under different load conditions. The profiler creates a container on the emulation platform and sends the media traffic data from the test container to the NS. The instance count in the pre-configuration parameters initiates multiple copies of the test container. Thus, the instantiated test containers send the media traffic data simultaneously from multiple sources. In addition, the duration of the load testing can be configured in the Profiler Tool UI. The developer sets the frame counts to determine how many frames will be transferred during the media traffic simulation. The parameter set and the beginning of the testing process can be observed in Figure 18.

![Figure 18- Traffic generation with Service Profiling](image)

The performance data during a profiling round is collected in two ways. First, service-internal performance metrics are monitored. Second, the platform metrics, such as the packet counters on the virtualized interfaces, are gathered through the platform's monitoring APIs. Finally, all measured data which are gathered from various sources, is aggregated and stored in the monitoring tools database and visualised on the dashboard.

4.8. End-To-End Workflow of Unikernels

Supporting unikernels in 5G-MEDIA does not require specific configurations from the perspective of the SDK, as they share the same approach the OSM uses (up to the releases R3 and R4) for plain VNFS’ configuration, which is based on Juju charms, with the only difference that they are based on HTTP calls instead of SSH, as explained later. To enable some DevOps automations, though, a few specific tools are provided to the developer to facilitate e2e
development. For this reason, a use-case is described to highlight the different steps and the configuration at different levels.

In a typical e2e unikernel development life cycle, the DevOps main steps are:

- project setup and source sharing on source control management tool,
- automatic building tool setup to get the unikernel VDU image and do the automatic tests,
- unikernel VDU image deployment on the VIMs for user acceptance tests, with possible configuration, and final mark as production-ready.

Due to the intrinsic constraint of unikernels and to the building tools used, the setup of such tools is usually manual and consists in the creation of an ad-hoc build file. In the 5G-MEDIA project, the integration of Mikelangelo [Mikelangelo] building tools overcomes this limitation and provides some DevOps automation.

The 5G-MEDIA DevOps pipeline for unikernels consists of the source code shared on Gitlab along with the build file for the project binaries and the build file for the unikernel image, and the setup of a Jenkins task for the automatic image building. While there is still the complexity to write the build file for unikernel, the advantage of Mikelangelo is its support to higher level programming languages such as Java, Javascript and Python, where the usual build file has many parts in common and can rely on a template.

An example is provided below for these types of projects, with a “meta” folder containing “package.yaml” and “run.yaml” configuration files, the first with the dependencies from the Mikelangelo packages for the correct build, the other one for the runtime configuration. In the case of a Javascript based project, such files will be like in Table 5.

<table>
<thead>
<tr>
<th>package.yaml</th>
<th>run.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: vspeech-gsa-unikernel</td>
<td>runtime: node</td>
</tr>
<tr>
<td>title: vspeech-gsa-unikernel</td>
<td>config_set:</td>
</tr>
<tr>
<td>author: eng</td>
<td>main_conf:</td>
</tr>
<tr>
<td>require:</td>
<td>config_set_default:</td>
</tr>
<tr>
<td>- osv.cli</td>
<td>main_conf</td>
</tr>
</tbody>
</table>

The most important part of package.yaml is the “require” field with the list of the components needed to be added to the build; for the run.yaml, the runtime used to launch the process and the entry point with references to the binaries. Another example of package.yaml and run.yaml, for Java programming language, is given in Table 6.
Table 6 - Another example configuration file

<table>
<thead>
<tr>
<th>package.yaml</th>
<th>run.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: vspeech-gsa-unikernel</td>
<td>runtime: java</td>
</tr>
<tr>
<td>title: vspeech-gsa-unikernel</td>
<td>config_set:</td>
</tr>
<tr>
<td>author: eng</td>
<td>conf:</td>
</tr>
<tr>
<td>created: 2018-06-23T22:47:24Z</td>
<td>main: /5G-unikernel-iperf-0.0.1-SNAPSHOT.jar</td>
</tr>
<tr>
<td>require:</td>
<td>args:</td>
</tr>
<tr>
<td>- openjdk8-zulu-full</td>
<td>-</td>
</tr>
</tbody>
</table>

For such higher-level programming languages, an SDK tool is provided to the developers to create all the necessary files for unikernel image building, a simple Python tool ("configure_unikernel.py") that accepts the project name and the programming language name as parameters. The resulting files contains default values that cover most of the typical use cases and can be customized for specific needs.

To speed up the project setup configuration, another Python tool ("setup_ci_cd.py") is available in the SDK to create with one command the project on Gitlab and to generate an issue on Jira to request the configuration of Jenkins for such project.

Once Jenkins is setup, each commit will produce a unikernel image available to the developers from the image repository for the tests; the marking as production-ready of the project is done simply pushing the code on the Gitlab production branch and having another Jenkins task configured to build the resulting image (Figure 19).

The screen for the profiler tool is provided in Figure 33 in Annex II. The user can create and test the unikernel images and then push them to an external repository by using the UI.

Details of the progress can be found in deliverable D5.4 – Packaging and Integration Tools evaluation and setup.
A dedicated unikernel repository is available for 5G-MEDIA, from where the developers can download the ISO for testing purposes, and the catalogue has access to push such images on the NFVI PoPs images repositories with just one API call, for example in OpenStack:

```
openstack image create --disk-format qcow2 --os-image-api-version 1 --location <unikernel qemu image URL> <image name>
```

Once unikernel images are deployed on the NFVI PoPs and the VNF are instantiated, the VNF configuration is done with the usual OSM approach, but with the difference that it relies on a http proxy charm instead of the ssh proxy charm.

The rationale is the lack of support for multiple processes that would be required using ssh. As a consequence, the VNF descriptors remain the same and the UI will produce automatically all the fields declared, while on the other hand the developer has to take care of correct Juju charm configuration based on rest calls.

For the perspective of the developer, the main differences with reference to the official documentation about Juju charms in OSM [ETSIOSM] are:

- in the dependencies described in "layer.yaml", that now only includes "layer:basic" and no more "layer:vnfproxy" that is based on ssh
- the code of the Python scripts inside reactive folder makes use of http calls instead of ssh command execution.
The data exchanged will be application specific and most commonly json formatted data managed by OSM UI with the automatic creation of the related fields. To distinguish unikernel VDUs, it is used the VNF descriptor field "vdu:hypervisor_type" with value "unikernel"; this enables possible UI customisations (e.g. dedicated icons) and the management of the deployment constraints that are going to be explored along the project timespan (e.g. Openstack is supported, OpenNebula is in progress).

Updated guidelines for the developers about the setup of CI/CD for unikernels are also available on Gitlab along with example projects to cover the most common programming languages supported by Mikelangelo [Mikelangelo].

Unikernel testing is possible on the developer machine using DevStack [Devstack], a Openstack all in one deployment on a single virtual machine that can run, for example, in VirtualBox [VirtualBox]. In this environment, unikernels are treated as plain VMs running on KVM.

The next step will be to evaluate a hybrid emulator supporting both plain VMs (for unikernels) and Docker (for the other VNFs, as happens with OSM vim-emu [ETSIvim], based on Containernet); the announced integration of Containernet 2.0\(^2\) in vim-emu could be an option.

\(^{2}\) Containernet 2.0 integration, [https://github.com/sonata-nfv/son-emu/issues/273](https://github.com/sonata-nfv/son-emu/issues/273)
5. Conclusions

In this deliverable, we have presented an overview of the programming tools to efficiently develop NFV-based media applications in 5G networks. The parts of 5G-MEDIA SDK have been presented discussing various open-source frameworks and libraries that are considered for the project. The architecture and usage of the 5G-MEDIA SDK tools such as the all-in-one UI, the validator, the editor, the emulator, the service monitoring tools and the private catalogue have been explained in detail and demonstrated with screenshots. We have presented the workflow of the UC1 of 5G-MEDIA project using the SDK. Implementation of the serverless FaaS approach to NFV technology has been presented and the development of the appropriate programming tools to accommodate the FaaS paradigm have been given. We finally presented the end-to-end workflow of the unikernels development from the perspective of the developer using the SDK.
6. References

[Containernet] Containernet https://containernet.github.io/

[Devstack] Devstack https://docs.openstack.org/devstack/latest/


[ETSIMslol5] ETSI Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nvfo Reference Point, Available online: http://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.04.01_60/gs_NFV-SOL005v020401p.pdf


[Electron] Electron, Build cross platform desktop apps with JavaScript, HTML, and CSS: https://electronjs.org/

[Flux] Flux, Application architecture for building user interfaces: https://facebook.github.io/flux/


[Hackfest] https://osm.etsi.org/wikipub/index.php/1st_OSM_Hackfest

[Lean] Lean OW https://github.com/kpavel/incubator-openwhisk/tree/lean


[Mininet] Mininet An Instant Virtual Network on your Laptop (or other PC): http://mininet.org/


[React] React, A JavaScript library for building user interfaces: https://reactjs.org/


[Sonata] Selfnet 5G https://github.com/Selfnet-5G


[VirtualBox] Virtual Box https://www.virtualbox.org/wiki/Downloads

Annex I – All-in-one UI Installation

The all-In-One UI installation has been simplified by creating a single setup file which allows you to select the path where you want to install the application as given in Figure 20.

![All-In-One Setup](image)

*Figure 20- All-In-One Setup*

Annex II – Tutorials and user screens

This section has been added as per the request of our project reviewers. In this section, we give the screens that are provided to the 5G-MEDIA developers. The first screen that the developer encounters in the all-in-one UI is the login screen (see Figure 21) where the user is expected to login with their user credentials. Furthermore, there is a button navigating to the configuration page.
In the configuration page which is given in Figure 22, the developer can configure the all-in-one UI using their development environment details such as the IPs of the editor, catalogue, monitoring, and lean OW. Figure 23 includes the dashboard screen which is presented to the developers after the login screen. The dashboard screen consists of navigation panels for the programming tools of the SDK and tutorials explaining how to use each of these tools. The dashboard also includes a map showing each consortium partner.
Figure 22 - Configuration Screen

Figure 23 - Dashboard
Clicking on each tutorial title, the developer is equipped with extensive information on the programming tool and how to use it. Tutorial screens are provided in Figure 24, Figure 25 and Figure 26.
In Figure 25, an interactive Lean OW CLI tutorial has been prepared in order for the users to execute the commands by simply pressing a button and observing the output in real-time through an embedded terminal, such as generating a package, creating an action, checking the current status of the system and listing all the actions.
An interactive OSM client tutorial has also been demonstrated in Figure 26. In this tutorial the user could execute OSM commands such as creating a vnfd, nsd, instantiating a service and listing all the descriptors that are created.

The screens for the validator are provided in Figure 27 and Figure 28. As seen in Figure 27, the developer can upload the descriptor file to be validated and the file can be a type of JSON, YAML or a compressed package in tar.gz format. The developer then uploads the schema file against which the descriptor file is validated. The validator then returns the result, i.e. if the descriptor is valid, it would return 201 code. In Figure 28, another example where the descriptor is invalid is given. As a result, Code 504 returns and the validator also points out where the error occurs and how to fix it according to the given schema.

In Figure 29, the screens of the editor are demonstrated. As mentioned earlier in the Section 4.2.2., OSM has been embedded as an editor tool into the all-in-one SDK. The developers can edit the VNFDs and NSDs and then instantiate network services as provided in Figure 30.

In Figure 31, we present the screen where the network simulator is carried out to profile the network service according to uploaded media file. The screen for service monitoring is demonstrated in Figure 32 where many monitoring parameters can be observed while the simulator is running.
Figure 27 - Validator: Example of valid descriptor

Figure 28 - Validator: Example of invalid descriptor
Figure 29- Editor

Figure 30- Instantiating the network services
**Figure 31 - Service Profiling**

**Figure 32 - Service Monitoring**
Annex III – 5G-MEDIA Validator Commands

Table 8 – Command for help

```
user$ 5gmediavalidate -h
```

Usage: 5gmediavalidate <options>
Where options are:
- -s, --schema <File>
- -v, --vnfd Virtual Network Function Descriptor <File>
- -vp, --vnfpackage Virtual Network Function Package <tar.gz.file>
- -n, --nsd Network Service <File>
- -np, --nspackage Network Service Package <tar.gz.file>
- -p, --vnfdpath Path for VNFD files that are referred in NSD <Path>

Table 9 – Schema validation

```
user$ 5gmediavalidate -s sample_schema.json
```
Table 10 – NSD validation with VNFD path given

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user$ 5gmediavalidate -n ./nsd/sample_nsd.yaml -p ./vnfd/ -s sample_schema.json</code></td>
<td>Validates NSD with VNFD paths provided.</td>
</tr>
</tbody>
</table>

This Schema is valid (Code: 200)

This YAML is valid (Code: 201)

sample_nsd.yaml: YAML is valid (Code: 201)

Table 11 – NSD validation without VNFD path given

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user$ 5gmediavalidate -n ./nsd/sample_nsd.yaml -s sample_schema.json</code></td>
<td>Validates NSD without VNFD paths.</td>
</tr>
</tbody>
</table>

There is no path given to validate the VNFDs that are referred in the NSD file

This Schema is valid (Code: 200)

sample_nsd.yaml: YAML is valid (Code: 201)

Table 12 – VNFD validation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user$ 5gmediavalidate -v ./vnfd/samplevnf.yaml -p ./vnfd/ -s sample_schema.json</code></td>
<td>Validates VNFD with paths provided.</td>
</tr>
</tbody>
</table>

This Schema is valid (Code: 200)

sample_vnf.yaml: YAML is valid (Code: 201)
Table 13 – VNF package validation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user$ 5gmediavalidate -np vnf-complete.tar.gz -s sample_schema.json</td>
<td>This Schema is valid (Code: 200)</td>
</tr>
<tr>
<td>vnf-complete/sample_nsd.yaml: YAML is valid (Code: 201)</td>
<td></td>
</tr>
</tbody>
</table>

Annex IV – Lean OW Installation

Create a Linux Ubuntu 16.04 VM with 1vCPU, 4GM memory and 16GB disk. This VM will serve as a local development sandbox for your FaaS VNFs.

As a pre-requisite, docker engine should be available in the local environment. Follow instructions³ to install docker (version 17.09.1~ce-0~ubuntu) on Ubuntu 16.04.

-Installation of Minikube

Install kubectl v1.12.2

```
user$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Install minikube v0.30.0

```
user$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Install socat

```
user$ sudo apt-get install -y socat
```
Start minikube. Pay attention to `--vm-driver=none` you will need to supply it later on.

```
user$ sudo minikube start --vm-driver=none
```

Put the docker network in promiscuous mode

```
user$ sudo ip link set docker0 promisc on
```

Ensure minikube properly works

```
user$ sudo minikube status
```

Create the roles

```
user$ sudo kubectl create -f roles.yml
```

Create the offload service

```
user$ sudo kubectl create -f offload.yml
```

Ensure offload-service POD responds by retrieving the ipaddress and port and pasting it in curl

```
user$ sudo minikube ip
user$ sudo kubectl describe service ow-offloadservice | grep http-api | grep NodePort | awk '{print $3}' | cut -d'/-' -f1
user$ curl <MINIKUBE IP>:<OFFLOAD PORT>/hello
```

-Installation of Lean OW

Install helm v2.11.0

```
user$ curl -LO https://storage.googleapis.com/kubernetes-helm/helm-v2.11.0-linux-amd64.tar.gz
user$ tar xzvf helm-v2.11.0-linux-amd64.tar.gz
user$ sudo ~/linux-amd64/helm init
```

To see if Helm is ready, use the command below and make sure the tiller-deploy pod is in the Running state

```
user$ kubectl get pods -n kube-system
```
Grant the necessary privilege

```
user$ kubectl create clusterrolebinding tiller-cluster-admin --clusterrole=cluster-admin --serviceaccount=kube-system:default
```

Pull lean OW image

```
user$ sudo docker pull docker5gmedia/repo:efx-controller_v2
```

Set namespace and label the node

```
user$ kubectl create namespace openwhisk
user$ kubectl label nodes --all openwhisk-role=invoker
```

Create ~/mycluster.yaml to describe your lean OW installation (Replace minikube-ip with output of sudo minikube ip) label the node

```
whisk:
  ingress:
   type: NodePort
   apiHostName: minikube-ip
   apiHostPort: 31001

controller:
  lean: true
  image: docker5gmedia/repo:efx-controller_v2
  imagePullPolicy: IfNotPresent

nginx:
  httpsNodePort: 31001
```

Install lean OW via helm

```
user$ git clone -b lean https://github.com/kpavel/incubator-openwhisk-deploy-kube.git
user$ cd incubator-openwhisk-deploy-kube
user$ sudo ~/linux-amd64/helm install ./helm/openwhisk --name=owdev -f ~/mycluster.yaml
```
Wait until openwhisk PODs are either in Completed or Running state (takes ~15 minutes)

```
user$ sudo kubectl get pod -n openwhisk
```

**wsk CLI installation**

Download [OpenWhisk_CLI-latest-linux-amd64.tgz]({#})

Unzip the file

```
user$ tar xzvf OpenWhisk_CLI-latest-linux-amd64.tgz
```

Learn the minikube IP address

```
user$ sudo minikube ip
```

Create and edit ~/.wskprops with the below replacing minikube-ip with output of above command

```
APIHOST= https://minikube-ip:31001
NAMESPACE=guest
AUTH=23bc46b1-71f6-4ed5-8c54-816aa4f8c502:123zO3xZCLrMN6v2BKK1dXYFpX1PkccOFqm12CdAsMgRU4VrNZ9lyGVCGuMDGIwP
```

---