



**5TH GENERATION END-TO-END NETWORK, EXPERIMENTATION,  
SYSTEM INTEGRATION, AND SHOWCASING**

[H2020 - Grant Agreement No. 815178]

Deliverable D5.4

# Documentation and supporting material for 5G stakeholders (Release B)

**Editor** E. Atxutegi (NEM), A. Díaz (UMA)

**Contributors** UMA (UNIVERSIDAD DE MALAGA), COS (COSMOTE),  
NEM (NEMERGENT), NCSRD (NATIONAL CENTER FOR  
SCIENTIFIC RESEARCH “DEMOKRITOS”), ATH  
(ATHONET SRL), ECM (EURECOM), UPV (UNIVERSIDAD  
POLITÉCNICA DE VALENCIA), KAU (KARLSTADS  
UNIVERSITET), SRL (SIMULA METROPOLITAN CENTER  
FOR DIGITAL ENGINEERING), ATOS (ATOS SPAIN SLC),  
SHC (SPACE HELLAS (Cyprus) Ltd.), FhG (FOKUS-  
FRAUNHOFER GESELLSCHAFT E.V., INSTITUTE FOR  
OPEN COMMUNICATION SYSTEMS), INF (INFOLYSIS  
P.C.), INT (INTEL), IHP (IHP GmBH)

**Version** 1.0

**Date** August 16<sup>th</sup>, 2021

**Distribution** PUBLIC (PU)



## List of Authors

<b>UMA</b>	<b>UNIVERSITY OF MÁLAGA</b>
A. Díaz Zayas, B. García, I. González, P. Merino	
<b>COS</b>	<b>COSMOTE KINITES TILEPIKOINONIES AE</b>
F. Setaki	
<b>NCSR</b>	<b>NATIONAL CENTER FOR SCIENTIFIC RESEARCH “DEMOKRITOS”</b>
G. Xilouris, T. Anagnostopoulos, T. Sarlas, H. Koumaras	
<b>KAU</b>	<b>KARLSTADS UNIVERSITET</b>
Brunstrom, M. Raziullah, J. Karlsson	
<b>SRL</b>	<b>SIMULA METROPOLITAN CENTER FOR DIGITAL ENGINEERING</b>
Ö. Alay	
<b>NEM</b>	<b>NEMERGENT</b>
E. Atxutegi, J. O. Fajardo	
<b>FhG</b>	<b>FOKUS-Fraunhofer Gesellschaft e.V., Institute for Open Communication Systems</b>
A. Prakash, O. Keil, M. Emmelmann, F. Eichhorn, S. K. Rajaguru	
<b>UPV</b>	<b>UNIVERSIDAD POLITÉCNICA DE VALENCIA</b>
A. Fornés	
<b>ATOS</b>	<b>ATOS SPAIN SA</b>
E. Jimeno	
<b>SHC</b>	<b>Space Hellas (Cyprus) Ltd.</b>
D. Lioprasitis, G. Gardikis	
<b>ATH</b>	<b>ATHONET SRL</b>
D. Munaretto, N. Menon	
<b>ECM</b>	<b>EURECOM</b>
F. Kaltenberger, P. Matzakos	
<b>INF</b>	<b>INFOLYSIS P.C.</b>
V. Koumaras, A. Papaioannou	
<b>INT</b>	<b>Intel Deutschland GmbH</b>
V. Frasca	
<b>IHP</b>	<b>IHP GMBH</b>
J. Gutiérrez	
<b>FOG</b>	<b>FOGUS</b>
Tsolkas, Passas, D. Xenakis, D. Chouliaras	
<b>LMI</b>	<b>LM Ericsson Ireland</b>
Bosneag, A-M	

## Disclaimer

---

The information, documentation and figures available in this deliverable are written by the 5GENESIS Consortium partners under EC co-financing (project H2020-ICT-815178) and do not necessarily reflect the view of the European Commission.

The 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 reader uses the information at his/her sole risk and liability.

## Copyright

---

Copyright © 2021 the 5GENESIS Consortium. All rights reserved.

The 5GENESIS Consortium consists of:

NATIONAL CENTER FOR SCIENTIFIC RESEARCH “DEMOKRITOS”	Greece
AIRBUS DS SLC	France
ATHONET SRL	Italy
ATOS SPAIN SA	Spain
AVANTI HYLAS 2 CYPRUS LIMITED	Cyprus
AYUNTAMIENTO DE MALAGA	Spain
COSMOTE KINITES TILEPIKOINONIES AE	Greece
EURECOM	France
FOGUS INNOVATIONS & SERVICES P.C.	Greece
FON TECHNOLOGY SL	Spain
FRAUNHOFER GESELLSCHAFT ZUR FOERDERUNG DER ANGEWANDTEN FORSCHUNG E.V.	Germany
IHP GMBH – INNOVATIONS FOR HIGH PERFORMANCE MICROELECTRONICS/LEIBNIZ-INSTITUT FUER INNOVATIVE MIKROELEKTRONIK	Germany
INFOLYSIS P.C.	Greece
INSTITUTO DE TELECOMUNICACOES	Portugal
INTEL DEUTSCHLAND GMBH	Germany
KARLSTADS UNIVERSITET	Sweden
L.M. ERICSSON LIMITED	Ireland
MARAN (UK) LIMITED	UK
MUNICIPALITY OF EGALEO	Greece
NEMERGENT SOLUTIONS S.L.	Spain
ONEACCESS	France
PRIMETEL PLC	Cyprus
RUNEL NGMT LTD	Israel
SIMULA RESEARCH LABORATORY AS	Norway
SPACE HELLAS (CYPRUS) LTD	Cyprus
TELEFONICA INVESTIGACION Y DESARROLLO SA	Spain
UNIVERSIDAD DE MALAGA	Spain
UNIVERSITAT POLITECNICA DE VALENCIA	Spain
UNIVERSITY OF SURREY	UK

This document may not be copied, reproduced or modified in whole or in part for any purpose without written permission from the 5GENESIS Consortium. In addition to such written permission to copy, reproduce or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

## Version History

Rev. N	Description	Author	Date
1.0	Release of D5.4	NEM	16/08/2021

## List of Acronyms

Acronym	Meaning
3GPP	3 <sup>rd</sup> Generation Partnership Project
5G-PPP	5G Public-Private Partnership
AAA	Authentication, Authorization and Accounting
ADB	Android Debug Bridge
API	Application Programming Interface
CA	Consortium Agreement
CLI	Command-Line Interface
CQI	Channel Quality Indicator
CRUD	Create, Read, Update, Delete
Dx.y	Deliverable N <sup>o</sup> y of Work Package x
EC	European Commission
ELCM	Experiment Life Cycle Manager
eMBB	Enhanced Mobile Broadband
EMS	Element Management System
GA	Grant Agreement
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IM	Infrastructure Monitoring
IoT	Internet of Things
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LTE	Long Term Evolution
M&A	Monitoring and Analytics
MANO	Management & Orchestration
MCPTT	Mission Critical Push-To-Talk
MCS	Mission Critical Services
ML	Machine Learning
mMTC	Massive Machine Type Communications
NBI	Northbound Interface
NEST	Network Slice Template
NFVO	Network Function Virtualization Orchestrator
NMS	Network Management System
NS	Network Service
NSA	Non-Stand-Alone
NSD	Network Service Descriptor
NSR	Network Service Record
OAI	OpenAirInterface
PC	Personal Computer
PM	Performance Monitoring
POSIX	Portable Operating System Interface for X

PSC	Primary Synchronization Code
RAN	Radio Access Network
REST	Representational State Transfer
RRM	Radio Resource Management
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSSI	Received Signal Strength Indicator
RTT	Round-Trip-Time
SA	Stand-Alone
SBI	Southbound Interface
SDN	Software Defined Network
SLM	Slicing Lifecycle Manager
SNR	Signal to Noise Ratio
SSH	Secure Shell
TAP	Test Automation Platform
TS	Technical Specification
UE	User Equipment
UI	User Interface
URL	Uniform Resource Locator
URLLC	Ultra-Reliable Low Latency Communication
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VNFR	Virtual Network Function Record
WIM	WAN Infrastructure Manager
WP	Work Package
WSGI	Web Server Gateway Interface

## List of Figures

Figure 1 5GENESIS architecture.....	19
Figure 2 Deployed 5GENESIS platforms .....	22
Figure 3 The 5GENESIS Experimentation Workflow.....	28
Figure 4 Network Service Onboarding flow diagram .....	33
Figure 5 Experimentation via the Portal .....	37
Figure 6 5GENESIS Portal registration screen .....	38
Figure 7 5GENESIS Portal sign in screen .....	38
Figure 8 Example of a 5GENESIS Portal info page .....	39
Figure 9 Experiment creation screen .....	40
Figure 10: Network services on boarding.....	41
Figure 11 User's dashboard .....	42
Figure 12 Experiment executions screen.....	42
Figure 13 Experiment execution log screen .....	44
Figure 14 Experiment result visualization screen .....	44
Figure 15 Generated PDF results report.....	45
Figure 16 Analytics dashboard interface.....	46
Figure 17: OPEN APIs interfaces .....	47
Figure 18 Experimentation via the Open APIs .....	47
Figure 19: Authorization Open APIs interfaces .....	48
Figure 20: Open APIs Mano interfaces.....	49
Figure 21: Experiment Descriptor Template.....	50
Figure 22: ELCM Open APIs interfaces .....	51
Figure 23: Result catalog Open APIs interfaces.....	52
Figure 24 Integration of new components to the 5GENESIS Architecture .....	53
Figure 25 Prometheus instrument .....	55
Figure 26 Slice Manager Building Blocks.....	63
Figure 27 Slice Manager Source Code Directory .....	66
Figure 28 Certificate exception .....	73
Figure 29 Dispatcher Execution Test log .....	74
Figure 30 5GENESIS related OpenTap instruments for NFV MANO (OSM).....	86
Figure 31 OSM Instrument .....	86
Figure 32 VIM Instrument.....	88
Figure 33 OSM Test Steps.....	90
Figure 34 Class hierarchy of the OpenTap OSM Test Steps .....	91
Figure 35 Class hierarchy of the generated OSM API accessors from its OpenAPI definition .....	91
Figure 36 – Typical OSM Plugin usage for testing and development.....	92
Figure 37 InfluxDB result listener configuration.....	94



Figure 38 Ping application interface.....	96
Figure 39 iPerf application interface .....	97
Figure 40 Resource Agent application interface.....	99
Figure 41 Remote PC agents TAP instrument and configuration .....	101
Figure 42 Remote PC agents TAP steps .....	101
Figure 43 Time series overview: graphical representation of the time series for the KPIs .....	106
Figure 44 Statistical analysis: main stats for the KPIs .....	107
Figure 45 KPI correlation: correlation matrices / coefficients for different KPIs in the same / different experiments .....	107
Figure 46 Feature selection: elimination of redundant features .....	108
Figure 47 KPI prediction: prediction of KPIs based on historical information and relations to other KPIs .....	108
Figure 48 KPI prediction: prediction of KPIs based on historical information and relations to other KPIs (details).....	109
Figure 49 DataHandler API and example response .....	109
Figure 50 StatisticalAnalysis API & response example .....	110
Figure 51 Correlation API call & response example .....	111
Figure 52 Feature selection API & response example.....	111
Figure 53 Predict API and example response .....	112
Figure 54: Katana Slice Manager Containers .....	114

## List of Tables

Table 1 Document interdependencies .....	15
Table 2 5GENESIS Release B. Summary of functionalities .....	20
Table 3 5GENESIS Athens Platform .....	22
Table 4 5GENESIS Málaga Platform.....	23
Table 5 5GENESIS Limassol Platform .....	23
Table 6 5GENESIS Surrey Platform.....	24
Table 7 5GENESIS Berlin Platform .....	24
Table 8 5GENESIS Experimentation Roles and Stakeholders .....	25
Table 9 Example of the report for the MCPTT test case.....	31
Table 10 List of test cases available in Release B .....	36
Table 11 Operations between Slice Manager and southbound components.....	63

## Executive Summary

Overall, deliverable D5.4 is a self-contained document serving as a general handbook for all the features supported by Release B of the 5GENESIS Experimentation Framework, which has been also open sourced as “Open 5GENESIS Suite” at GitHub (<https://github.com/5genesis>). The focus of this deliverable is to provide a clear understanding of the features provided by 5GENESIS for experimentation, the kind of experiments that can be executed, and how new components can be integrated and managed by the 5GENESIS Experimentation Framework.

This documentation includes the kind of interactions an external entity needs to establish in order to run experiments over a 5GENESIS platform and the mechanisms to retrieve the measurements’ results. Moreover, it describes the way the Experimenters benefit from the configurability of the underlying infrastructure. The document also provides a detailed development guide for 5G technology providers willing to integrate their solutions into the testbed. Finally, the document stands as a manual for testbed operators interested in adopting the 5GENESIS experimentation Framework.

This document builds upon the previous work presented in D5.3 [16] that focused on the initial release (Release A) of the 5GENESIS framework, and provides a revised and holistic view of the documentation for the 5G stakeholders. As the work of Open 5GENESIS Suite is progressing in the open source community, adaptations and support material shall be updated here-after on-line in the Open 5GENESIS GitHub repository as necessary.

# Table of Contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>15</b>
1.1	Purpose of the Document.....	15
1.2	Structure of the Document.....	16
1.3	Target Audience .....	17
<b>2</b>	<b>THE 5GENESIS ARCHITECTURE.....</b>	<b>18</b>
2.1	Concepts and Architecture .....	18
2.2	Main features of Release B.....	20
2.3	The 5GENESIS platforms .....	21
2.4	Involved stakeholders and experimentation roles .....	25
2.5	The 5GENESIS experimentation workflow .....	27
2.5.1	Experiment consultation phase .....	28
2.5.2	Experiment provisioning phase.....	29
2.5.3	Experiment execution phase.....	29
2.5.4	Experiment decommissioning phase.....	32
2.6	What kind of experiment can be executed?.....	32
2.6.1	Testing network services.....	32
2.6.2	Testing user equipment .....	33
<b>3</b>	<b>DOCUMENTATION FOR EXPERIMENTERS.....</b>	<b>35</b>
3.1	The 5GENESIS Experimentation Methodology .....	35
3.2	Experimentation via the Portal.....	37
3.2.1	Login .....	37
3.2.2	Definition of experiments .....	39
3.2.3	Execution of experiments .....	41
3.2.4	Visualization of results.....	43
3.2.4.1	Grafana dashboard .....	43
3.2.4.2	Analytics Dashboard .....	45
3.3	Experimentation via the Open APIs.....	46
3.3.1	Authorization .....	47
3.3.2	Provisioning.....	48
3.3.3	Description of the Experiment Descriptor Template.....	49
3.3.4	Sending execution requests to the ELCM .....	51
3.3.5	Accessing results and logs.....	52
<b>4</b>	<b>DOCUMENTATION FOR TECHNOLOGY PROVIDERS.....</b>	<b>53</b>

4.1	Introduction to the development of 5GENESIS plugins .....	54
4.1.1	Description of the southbound interface of ELCM .....	54
4.1.1.1	TAP plugins for the ELCM .....	54
4.1.1.1.1	Examples: TAP plugins example.....	55
4.1.1.2	Python extensions for the ELCM .....	61
4.1.1.2.1	Example: CompressFiles task .....	62
4.1.2	Description of the southbound interface of the Slice Manager .....	62
4.1.2.1	Python plugins for the slice manager .....	64
4.1.2.1.1	Example: Amarisoft plugin .....	66
<b>5</b>	<b>DOCUMENTATION FOR PLATFORM OPERATORS .....</b>	<b>69</b>
5.1	Coordination Layer deployment.....	69
5.1.1	Portal .....	69
5.1.1.1	Deployment.....	69
5.1.1.1.1	Pre-requisites.....	69
5.1.1.1.2	Installation procedure .....	69
5.1.1.1.3	Starting the Portal .....	70
5.1.1.1.4	Minimal integration tests.....	70
5.1.1.2	Configuration.....	70
5.1.2	Dispatcher .....	71
5.1.2.1	Installation.....	72
5.1.2.2	Configuration.....	72
5.1.2.3	Deployment.....	72
5.1.2.4	Testing .....	73
5.1.3	ELCM.....	74
5.1.3.1	Deployment.....	75
5.1.3.1.1	Pre-requisites.....	75
5.1.3.1.2	Installation procedure .....	75
5.1.3.1.3	Starting the Portal .....	75
5.1.3.2	Configuration.....	76
5.1.3.2.1	Facility Configuration (Platform registry) .....	77
5.1.3.3	Available Tasks .....	80
5.1.3.3.1	Variable expansión .....	82
5.1.3.4	MONROE experiments.....	83
5.1.3.5	Distributed experiments.....	84
5.2	OSM Plugin .....	85

5.2.1	OpenTap OSM Instruments.....	85
5.2.1.1	OSM Instrument.....	86
5.2.1.2	VIM Instrument.....	87
5.2.2	OpenTap OSM Test Steps.....	89
5.3	Monitoring and Analytics Deployment .....	92
5.3.1	Result management .....	92
5.3.1.1	InfluxDB .....	92
5.3.1.2	Result listener.....	93
5.3.2	Prometheus Infrastructure Monitoring.....	94
5.3.3	Performance Monitoring.....	95
5.3.3.1	ADB Monitoring agents .....	95
5.3.3.1.1	Ping agent .....	95
5.3.3.1.2	iPerf agent.....	97
5.3.3.1.3	Resource agent.....	98
5.3.3.1.4	Remote PC Agents.....	99
5.3.3.2	MonroeVN.....	101
5.3.3.2.1	Ping Container (monroe/ping:virt) .....	102
5.3.4	Analytics .....	104
5.4	Slice Manager Deployment.....	112
5.4.1	Deployment .....	113
5.4.2	Configuration .....	114
5.4.3	Slice Instantiation .....	115
<b>REFERENCES .....</b>		<b>116</b>
<b>ANNEX 1 – EXAMPLE TEMPLATE FOR GRAFANA REPORTER .....</b>		<b>118</b>
<b>ANNEX 2 – PROMETHEUS TAP PLUGIN SOURCE CODE .....</b>		<b>119</b>
<b>ANNEX 3 – ANDROID ADB AGENTS TESTPLAN EXAMPLE.....</b>		<b>125</b>
<b>ANNEX 4 – JSON SCHEMA OF THE SLICE MANAGER SOUTHBOUND MESSAGES .....</b>		<b>126</b>
WIM Data JSON Schema .....		126
EMS Data JSON Schema .....		129
<b>ANNEX 5 – SOUTHBOUND COMPONENTS CONFIGURATION FILES SCHEMAS.....</b>		<b>134</b>
VIM 134		
NFVO 135		
WIM 136		
EMS 136		
<b>ANNEX 6 –NEST JSON SCHEMA .....</b>		<b>138</b>

# 1 INTRODUCTION

## 1.1 Purpose of the Document

5GENESIS is one of the three 5G PPP Phase-3 projects [1] that are chartered to provide large-scale end-to-end 5G network experimentation infrastructures, with the main target to facilitate the vertical industries, SMEs, and all other players and stakeholders of the 5G ecosystem, in the course of validating 5G deployments for their business mandates. Complying with this fundamental project objective, the 5GENESIS Experimentation Framework is built to offer the adequate level of openness, in terms of specification and implementation, to orchestrate the on boarding of industry specific systems and software, and to manage the 3rd parties' interactions effectively during the experimentation life-cycle.

Towards this objective, this deliverable collects in a single, self-contained document, all necessary information, conceptual and technical, to effectively support the 5G stakeholders that wish to engage with the 5GENESIS platforms. It summarises the basic definitions and design principles of the 5GENESIS Experimentation Framework, and sets out to practically describe - in the narrative of user support documentation (manual) - the technical interactions and parameters that need to be exchanged when interacting with the 5GENESIS Facility.

Both in Release A and B, the consortium took the strategic decision to open source the 5GENESIS experimentation layer, launching the “Open 5GENESIS Suite” open source project at GitHub [50], where all the components of the 5GENESIS experimentation layers have been open sourced or are in the process or doing so under Apache License 2.0 or GPL 3.0. The aim of opening the 5GENESIS Experimentation layer and releasing the Open 5GENESIS Suite is to achieve sustainability of the solution beyond the project lifetime, allowing also the customization of the components to the different needs of the various vertical industries.

This document considers the second release of the 5GENESIS platforms (Release B)/[Open 5GENESIS Suite](#) and supersedes Deliverable D5.3 [16] that presents the relevant material for Release A. To provide a comprehensive vision of the 5G Experimentation Framework offered by 5GENESIS, this deliverable synthesizes extensive results as part of the work performed in WP2 for the high-level requirements and architecture specification, in WP3 for the implementation of specific directives, and in WP4 for platform-related technical capabilities and features.

Table 1 presents the list of main deliverables detailing this work as the source for more analytic studies.

**Table 1 Document interdependencies**

Deliverable Number	Document Title	Relevance
D2.1 [2]	Requirements of the Facility	Defines the initial requirements and guiding principles that support the 5GENESIS developments.
D2.4 [3]	Final report on facility design and	Presents the 5GENESIS Facility architecture and lists the functional components to be deployed in each testbed.

Deliverable Number	Document Title	Relevance
	experimentation planning	
D2.3 [4]	Initial planning of tests and experimentation	Provides the testing and experimentation methodology and processes that rule the testbed definition, operation and maintenance.
D3.2 [5]	Management and orchestration (Release B)	Describes the implementation of the MANO solutions that are integrated in the infrastructure, together with the relevant interfaces and deployment options.
D3.4 [6]	Slice management WP3 (Release B)	Describes the implementation of the Slice Manager solution, and its interfaces towards the MANO and NMS components.
D3.6 [7]	Monitoring and WP3 analytics (Release B)	Describes the implementation of Infrastructure and Performance Monitoring components, as well as of Analytics tools, including the interfaces with infrastructure elements (Release B).
D3.10 [8]	5G Core Network WP3 Functions (Release B)	Describes the 5G Core network functions and provides input on their integration with the infrastructure and management components.
D3.11 [9]	5G Access Components and User Equipment (Release A)	Describes the 5G Radio Access components and UE devices. Final Release B is due the end of project.
D4.2 [11]	The Athens Platform	Details the Athens platform in 5GENESIS Release B (Jan/2020).
D4.5 [12]	The Malaga Platform	Details the Malaga platform in 5GENESIS Release B (Jan/2020).
D4.8 [13]	The Limassol Platform	Details the Limassol platform in 5GENESIS Release B (Jan/2020).
D4.11 [14]	The Surrey Platform	Details the Surrey platform in 5GENESIS Release B (Jan/2020).
D4.14 [15]	The Berlin Platform	Details the Berlin platform in 5GENESIS Release B (Jan/2020).
D5.3 [16]	Documentation and supporting material for 5G stakeholders (Release A)	Details the documentation of the platform and 5GENESIS suite for Release A.

## 1.2 Structure of the Document

The topics of deliverable D5.4 are presented with the following structure:



- **Section 1** ‘Introduction’ introduces the document and presents the purpose, the structure and the target audience of D5.4.
- **Section 2** ‘The 5GENESIS Facility Overview’ provides the synopsis of the 5GENESIS concepts, architecture, components and interfaces, and the nomenclature that guides the more technical descriptions that follow in the next sections. **Section 2** also presents the 5GENESIS experimentation workflows, identifying potential stakeholders, their roles, and explaining the workflow of interacting when engaging with any of the 5GENESIS platforms.
- **Section 3** ‘Documentation for experimenters’ starts with extensive technical description of the interactions considered primarily for the representatives of the vertical industries, and focuses on the usage of the Portal, which is considered the main entry point for the experimenters. Direct interactions through the OpenAPI, which offers a systemic interface similar to the graphical one, is also described in this section.
- **Section 4** ‘Documentation for technology providers’ delves into technical descriptions and configuration capabilities of the 5GENESIS components and it is directed towards technical development teams with specific configuration requirements and constraints for integrating 3<sup>rd</sup> party systems and software as part of any 5GENESIS platform.
- **Section 5** ‘Documentation for platform operators’ dives into the platform internal configurations necessary to implement the 5G Facility functionalities, either within the project as an internal handbook or for new testbeds that are interested in deploying the 5GENESIS experimentation framework developed as part of the 5GENESIS project.
- Finally, in the **Annex** sections, the technical and configuration parameters are documented as referenced in each section.

## 1.3 Target Audience

This deliverable is a public document with an extensively technical, hands-on insight on 5GENESIS developments to be appreciated by engineering teams that consider engaging with 5G technologies from various perspectives. The content presented is mainly addressed to:

- Projects that consider pilots or trials with Vertical industries, e.g. ICT-19 projects, which are interested to validate their industry-related use cases in the 5G experimentation platforms offered by 5GENESIS in Athens, Málaga, Limassol, Surrey and Berlin, either by validating specific Key Performance Indicator (KPI) targets or testing their systems and software in advanced 5G deployments.
- Development and engineering teams, 5G equipment vendors, SMEs, technology providers that consider integrating their products and services in the 5GENESIS Facility.
- The Project Consortium, as a handbook of the end-to-end project developments for internal support and basis for further enhancements.
- The Research Community and funding European Commission (EC) Organisation, as the detailed synopsis of the technical orientation and achievements targeted by the project.
- The general public, as a testimony of the technical scope, and software orientation used by the 5GENESIS project.

Finally, the content of this deliverable is in-line with the guidelines of Deliverables D1.2/D1.3 “Legal aspects and data management (Release A/B)”.

## 2 THE 5GENESIS ARCHITECTURE

---

### 2.1 Concepts and Architecture

All the content in this deliverable refers to the Release B of the 5GENESIS components and platforms (WP3 and WP4 related work, respectively).

In this section, we briefly introduce the 5GENESIS architecture, described in D2.4 [3], to provide a better understanding on how the architecture reflects the interaction with the respective actors: Experimenters and related stakeholders.

The 5GENESIS architecture, depicted in Figure 1, is structured in three main blocks: **Coordination Layer** (yellow), **Management and Orchestration (MANO) Layer** (green) and **Infrastructure Layer** (blue).

Following a top-down description of the 5GENESIS architecture, we start with the **Coordination layer**. This layer offers all the components of the 5GENESIS experimentation framework relevant to experiments, as well as the experiment facing Application Programming Interfaces (APIs) and User Interfaces (UIs). In detail, the Coordination Layer provides Northbound Interfaces (NBI) for the Experimenter, namely the 5GENESIS Portal and the Open APIs (more information available in D3.8 [19]). Via the Portal, the Experimenter can be authenticated, onboard vertical application components, submit experiment requests and, after the execution of the experiment, acquire measurements (either raw or processed). During the experimentation, the Experiment Lifecycle Manager (ELCM) is responsible for the sequencing of experiment lifecycle stages by maintaining the experiment status and providing feedback to the experimenter.

The analytics component of the Coordination Layer is responsible for the complete collection and analysis of the heterogeneous monitoring data produced during the usage of the 5GENESIS experimentation. In order to collect the monitoring information from all elements of each 5GENESIS platform, the analytics component retrieves the measurements from the probes deployed in each platform. This component ingests either in-real time or after the end of each experiment session, the measurements in a unified database for post-processing and long-term storage. To this end, the monitoring framework also collects and stores information from the testing probes that are deployed during the experiment.

In light of state-of-the-art network monitoring and analytics functionalities, the 5GENESIS Monitor and Analytics (M&A) framework positions itself as a key enabler for a complete validation of 5G KPIs.

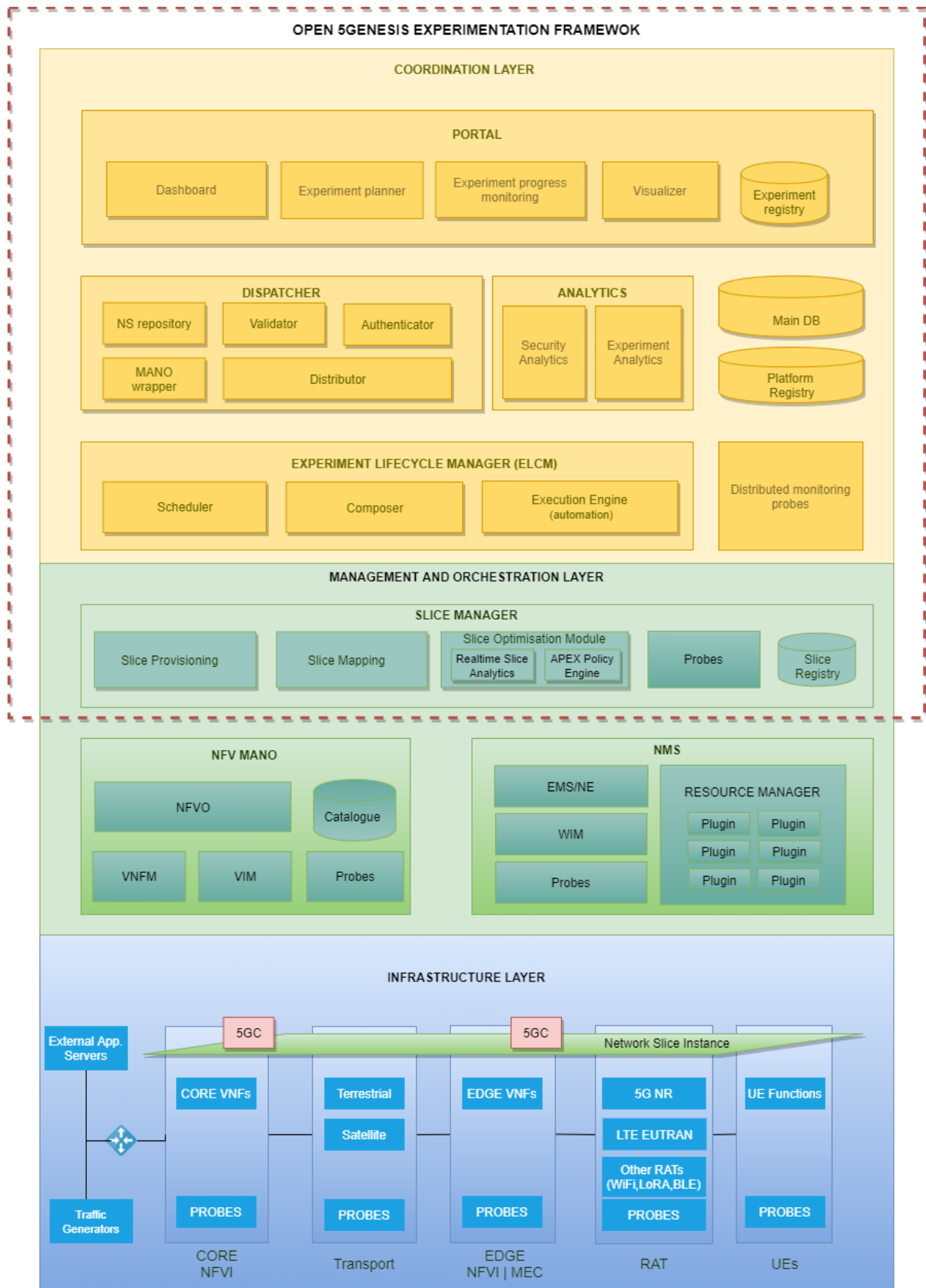


Figure 1 5GENESIS architecture

In the **Management and Orchestration Layer**, the Slice Manager is in charge of the configuration and deployment of the slices. This also implies the access to the components available at the Infrastructure Layer<sup>1</sup>. Section 4.1.2 “Description of the slice manager southbound interface” describes the development of the plugins required for establishing the communication with the Slice Manager. These plugins are part of the Network Management System (NMS) shown in the architecture. More information about infrastructure commonalities could be found in the final architecture design in D2.4 [3].

## 2.2 Main features of Release B

The Release B of the 5GENESIS Coordination Layer and that of the platforms form altogether the “[Open 5GENESIS Suite](#)” and provide enhanced experimentation capabilities and KPI validations for the vertical Experimenters compared to the functionalities included in the Release A of the components, providing the support for the execution of Tests Cases and the collection of results. In a nutshell, the prominent features available after the installation of the Release B codebase are summarised in Table 2.

**Table 2 5GENESIS Release B. Summary of functionalities**

Component Name	Functionalities	Reference
Portal	<ul style="list-style-type: none"> <li>• Integrated Platform-wide authentication</li> <li>• Experiment Creation</li> <li>• Experiment Initiation</li> <li>• Experiment logs and raw results visualization</li> <li>• Vertical application components on boarding</li> <li>• Support for distributed experiment’s execution</li> </ul>	<a href="#">Release B Repository</a>
Dispatcher	<ul style="list-style-type: none"> <li>• Expose the directory of the NS and VNF artefacts.</li> <li>• Authorized user requests</li> <li>• Validation of NS and Experiment descriptor</li> <li>• Retrieve stored experiments from the database</li> <li>• Launch experiment (create)</li> <li>• Cancel execution</li> <li>• Get execution log</li> </ul>	<a href="#">Release B Repository</a>
ELCM	<ul style="list-style-type: none"> <li>• Initiation, runtime status of the experiment</li> <li>• Automatic handling and storage of execution logs</li> <li>• Orchestration of external components via Test Automation Platform (TAP) Test Plans or script execution</li> <li>• Extensible via plugins for various infrastructure elements and components</li> <li>• Testbed resources management</li> <li>• Support for distributed experiments</li> </ul>	<a href="#">Release B Repository</a>

<sup>1</sup> Additional information about the Infrastructure Layer is included in WP4 deliverables.

	<ul style="list-style-type: none"> <li>• Baseline tests for platform validation defined</li> <li>• Core functionality tested by implementing several test cases (for validation of a number of KPIs)</li> </ul>	
<b>Monitoring and Analytics</b>	<ul style="list-style-type: none"> <li>• Statistical processing and report on selected KPIs (depending on the available test cases)</li> <li>• Monitoring data collection from various probes</li> <li>• Monitoring data collection from infrastructure monitoring platforms (i.e. Prometheus)</li> <li>• Performance Monitoring (PM)</li> <li>• Infrastructure Monitoring (PM)</li> <li>• Analytics dashboard: <ul style="list-style-type: none"> <li>○ Data visualization</li> <li>○ Prediction</li> <li>○ Correlation</li> <li>○ Feature selection</li> <li>○ Statistical analysis</li> </ul> </li> </ul>	<a href="#">Release B Repository</a>
<b>Slice Manager</b>	<ul style="list-style-type: none"> <li>• Start/Stop/Inspect end-to-end network slices</li> <li>• Open APIs supported by swagger-io tool</li> <li>• Lightweight web user interface</li> <li>• Integrated Command-Line Interface (CLI) tool</li> <li>• Modular architecture supporting different infrastructure technologies</li> <li>• Slice deployment and configuration time measurement</li> <li>• Prometheus Node exporter</li> <li>• Message Broker</li> <li>• Slice Optimization Module</li> </ul>	<a href="#">Release B Repository</a>
<b>NMS</b>	<ul style="list-style-type: none"> <li>• WAN Infrastructure Manager (WIM)</li> <li>• Software Defined Network (SDN) Control</li> <li>• Monitoring</li> <li>• TAP plugins for controlling Android devices and computers through Secure Shell (SSH)</li> <li>• Performance monitoring tools for Android: resource usage, latency and throughput probes</li> <li>• Remotely controlled latency and throughput probes for Personal Computer (PC)</li> </ul>	<a href="#">Release B Repository</a>  <a href="#">OPENTAP repository</a>

## 2.3 The 5GENESIS platforms

The concepts and architecture of 5GENESIS are implemented in five platforms across Europe, ready to serve as end-to-end 5G testbeds to support experimentation for SMEs, industries, and projects that may consider trials involving Vertical industries.



Figure 2 Deployed 5GENESIS platforms

These platforms are located in Athens [11], Málaga [12], Limassol [13], Surrey [14] and Berlin [15], and each one of them has some distinct features and orientation, as summarised in Table 3 (Athens), Table 4 (Málaga), Table 5 (Limassol) Table 6 (Surrey) and Table 7 (Berlin):

Table 3 5GENESIS Athens Platform

5GENESIS Athens-GREECE Platform		
An edge-computing-enabled shared radio infrastructure (gNBs and small cells), with different ranges and overlapping coverage that are supported by an SDN/NFV enabled core, to showcase secure content delivery and low latency applications in large public events.		
Sites	<ol style="list-style-type: none"> <li>1. NCSR Campus@Ayia Paraskevi</li> <li>2. COSMOTE@Marousi</li> <li>3. Stadium@Egaleo</li> </ol>	
Deployed 5G Technologies	2020	5G Non-Stand-Alone (NSA)  <b>Core Network:</b> Athonet, Amarisoft, Eurecom <b>Radio Access:</b> Nokia 5G, Nokia LTE, Amarisoft, Eurecom, Runel  <b>UE:</b> Commercial, Eurecom, Amarisoft
	2021	Upgrade to 5G Stand-Alone (SA)
Use Cases	<ol style="list-style-type: none"> <li>1. Big Event in a soccer stadium</li> <li>2. “Eye in the sky” applications (Control the drone over a low-latency 5G slice and transmit HD and 4K real-time video to the ground control station)</li> <li>3. Security-as-a-Service</li> </ol>	
Contact	athens@5genesis.eu	

Table 4 5GENESIS Málaga Platform

5GENESIS Málaga -SPAIN Platform		
Automated orchestration and management of different network slices over multiple domains, on top of the 5G New Radio (NR) and fully virtualised core network to showcase mission critical services in the lab and in outdoor deployments.		
Sites	<ol style="list-style-type: none"> <li>1. Ada Byron Research@UMA (indoor &amp; outdoor)</li> <li>2. Málaga city centre</li> <li>3. Málaga Police Emergency Centre</li> <li>4. Telefonica I+D lab in Málaga/Madrid</li> </ol>	
Deployed 5G Technologies	2020	5G NSA  <b>Core Network:</b> Athonet, Polaris <b>Radio Access:</b> Nokia 5G, Nokia LTE, Amarisoft, Eurecom, Runel  <b>UE:</b> Commercial, OAI, Amarisoft
	2021	Upgrade to 5G SA
Use Cases	<ol style="list-style-type: none"> <li>1. Wireless video in large scale event</li> <li>2. Multimedia Mission Critical Services</li> <li>3. Edge-based Mission critical services</li> </ol>	
Contact	malaga@5genesis.eu	

Table 5 5GENESIS Limassol Platform

5GENESIS Limassol-CYPRUS Platform		
Radio interfaces of different characteristics and capabilities, combining terrestrial and satellite communications, integrated to showcase service continuity and ubiquitous access in underserved areas		
Sites	<ol style="list-style-type: none"> <li>1. Primetel@Limassol</li> <li>2. Avanti@Makarios</li> <li>3. Portable Hotspots</li> </ol>	
Deployed 5G Technologies	2020	5G NSA  <b>Core Network:</b> Athonet, Amarisoft, Eurecom <b>Radio Access:</b> Amarisoft, Eurecom  <b>UE:</b> Commercial, Eurecom

5GENESIS Limassol-CYPRUS Platform		
	2021	Upgrade to 5G SA
Use Cases	<ol style="list-style-type: none"> <li>1. 5G maritime communications</li> <li>2. 5G Capacity-on-demand and IoT in rural areas</li> </ol>	
Contact	limassol@5genesis.eu	

Table 6 5GENESIS Surrey Platform

5GENESIS Surrey-UK Platform		
Multiple radio access technologies that can support massive Machine Type Communications (mMTC), including 5G NR and NB-IoT, combined under a flexible Radio Resource Management (RRM) and spectrum sharing platform to showcase massive IoT services		
Access Sites	1. 5G Innovation Centre @University of Surrey Campus	
Deployed 5G Technologies	2020	5G NSA  <b>Core Network:</b> 5GIC 5GC NSA (in-house) <b>Radio Access:</b> Huawei  <b>UE:</b> Commercial
	2021	Upgrade to 5G SA
Use Cases	1. Massive IoT for large-scale public events	
Contact	surrey@5genesis.eu	

Table 7 5GENESIS Berlin Platform

5GENESIS Berlin-GERMANY Platform		
Ultra-dense areas covered by various network deployments, ranging from indoor nodes to nomadic outdoor clusters, coordinated via advanced backhauling technologies to showcase immersive service provisioning		
Sites	<ol style="list-style-type: none"> <li>1. Fraunhofer FOKUS @Berlin</li> <li>2. IHP@Frankfurt (Oder)</li> <li>3. Humboldt University@Berlin center</li> </ol>	



5GENESIS Berlin-GERMANY Platform		
Deployed 5G Technologies	2020	5G SA  <b>Core Network:</b> Open5GCore (in-house) <b>Radio Access:</b> Huawei & Nokia  <b>UE:</b> Commercial
	2021	Further upgrades and new deployments of 5G SA Radio
Use Cases	1. Festival of Lights	
Contact	berlin@5genesis.eu	

## 2.4 Involved stakeholders and experimentation roles

As identified during the analysis study in deliverable D2.1 [2], the main stakeholders to realise the successful adoption of the evolving 5G business concepts, applications and technology are the (i) Business Verticals, the (ii) Connectivity Providers/Operators, the (iii) Technology Providers and the (iv) End Users.

5GENESIS brings the experimentation perspective in the picture as an important enabler to give interested entities the capability of early 5G adoption, allowing the assessment of 5G deployments and vertical industries' services offered by the 5GENESIS platforms. Such trial deployment, execution and validation, is called an *Experiment* in the context of the project and introduces specific *roles* for the involved stakeholders interacting with the 5GENESIS Platforms. These roles are important to clarify the type and complexity of interactions with the platforms and denote specific capabilities that are assumed. It is worth mentioning that there is no fixed association between the experimentation roles and the involved stakeholders, rather a variety of combinations is expected; a stakeholder can undertake multiple roles during an experiment, as explained in Table below:

Table 8 5GENESIS Experimentation Roles and Stakeholders

5GENESIS Experimentation Roles	
<b>Experimenter</b>  An external user that wants to use the 5GENESIS experimentation framework. The Experimenter can set experiments and obtain results through either the web interface after registration (5GENESIS portal), or a direct use of the API for Experimenters developed in the project.	
Potential Stakeholders:	<ul style="list-style-type: none"> <li><u>Vertical</u> representatives: keen to test a 5G infrastructure and experience promised services &amp; KPIs. Verticals may optionally bring their industry specific systems and software to be loosely integrated with the 5GENESIS platforms. These can be equipment or appliances using SIM cards, or applications, physical or virtual, to be activated in the edge</li> </ul>

## 5GENESIS Experimentation Roles

	<p>infrastructure. As part of the experiments, verticals can request measuring specific KPIs (Test Cases) either standard and predetermined per platform, or especially described (through a custom test case) in well-defined network setups (Scenarios).</p> <ul style="list-style-type: none"> <li>• <u>Technology Providers</u>: Vendors of system and software that build products around the 5G ecosystem and need to have early validation results using end-to-end 5G deployment setups. They are certainly bringing in their products for integration, still they must also follow the experimentation life cycle to get measurable validation results.</li> </ul>
<b>Platform Technology Provider</b> <p>Stakeholders that provide software and hardware components and deployment configurations to any of the 5GENESIS platforms. Their interest is to assure smooth integration of their deliveries within the target platforms. The interactions of concern in this case are in principle disjoint from the experiments' execution life cycle, as they focus on the preparation and pre-provisioning activities to ensure the incorporation of the solutions in the platforms.</p>	
Potential Stakeholders:	<ul style="list-style-type: none"> <li>• <u>Technology Providers</u> (5G Vendors, Software Integrators) bringing in products</li> <li>• <u>Research Institutions and Academia</u>: Bringing in prototypes and deployment configuration recommendations</li> </ul>
<b>Platform Operator</b> <p>Hosts, manages and operates the platform's software and infrastructure, including the network infrastructure and main/edge data centres, as well as the 5GENESIS experimentation framework for coordination, management, orchestration and monitoring. To comply with the 5GENESIS requirements, internal development is necessary to expose specific interfaces (implemented through plugins) and support project specific developments (such as the Coordination Layer or the Slice Manager).</p>	
Potential Stakeholders	<ul style="list-style-type: none"> <li>• <u>5GENESIS Platforms</u> (Athens, Málaga, Limassol, Surrey, Berlin)</li> <li>• Technology Providers</li> <li>• Research Institutions and Academia</li> </ul> <p>A Platform Operator can be any entity that has the 5G competency to operate an end-to-end 5G infrastructure. While traditionally, the role is assumed by telecommunication providers, other business opportunities seem to emerge. Nevertheless, for the life cycle of the R&amp;D projects under the 5G-PPP umbrella, the Platform Operators are primarily the partners of the 5GENESIS consortium.</p>
<b>Testers and End Users</b> <p>These are the users of the services deployed in the 5GENESIS platforms to support the execution of an experiment, carrying out specific interactions and utilising specific equipment as necessary per case. They can be either individuals or corporate end users.</p>	

## 5GENESIS Experimentation Roles

<b>Potential Stakeholders</b>	<ul style="list-style-type: none"> <li>• Vertical representatives</li> <li>• Platform Operators</li> <li>• Technology providers</li> <li>• End-Users</li> </ul> <p>All stakeholders are potential end-users and can bring their own groups, either employees, customers or randomly selected users.</p>
-------------------------------	---

The following example presents the involvement of several stakeholders around a 5GENESIS platform for a realistic 5G end-to-end demonstration. This example will be used as a reference in the next sections.

**Demo Case:** The municipality of Málaga is interested to test a new Mission Critical Service (MCS) to be deployed for managing large-scale outdoor demonstrations in the city of Málaga.

**Experimenter:** The ICT department of the Municipality of Málaga (MoM), who shall discuss the requirements, collect the results and communicate to the authorities the benefits identified. MoM shall be assisted with the documentation provided in Section 3.

**Platform Technology Provider:** The company NEM is in charge of building the Mission Critical Service for the ICT department of MoM, and will provide the relevant software to be installed at the 5G infrastructure edge site. NEM will also make sure that the User Equipment (UE) brought in for the trials shall have the respective MCS application installed and shall support MoM in all actions necessary to execute the MCS trial. NEM shall be assisted with Documentation of Section 4.

**Platform Operator:** UMA has the administrator role for the Málaga 5GENESIS Platform and undertakes any communication necessary with MoM for the proper execution of the experiment, with the support of the 5GENESIS consortium. UMA shall make sure that the 5G infrastructure necessary to execute the experiment shall be made available at the site and period agreed with MoM. UMA shall be assisted with the Documentation of Section 5.

**End Users/Testers:** Policemen from the MoM, equipped with the UE containing the proper MCS application as well as 5GENESIS probes, shall gather the relevant measurements.

## 2.5 The 5GENESIS experimentation workflow

Depending on the nature of the experiment, a close cooperation between the Experimenters and the platform operators may be necessary. As graphically depicted in Figure 3 and explained below, the following phases are considered as part of the experimentation workflow.

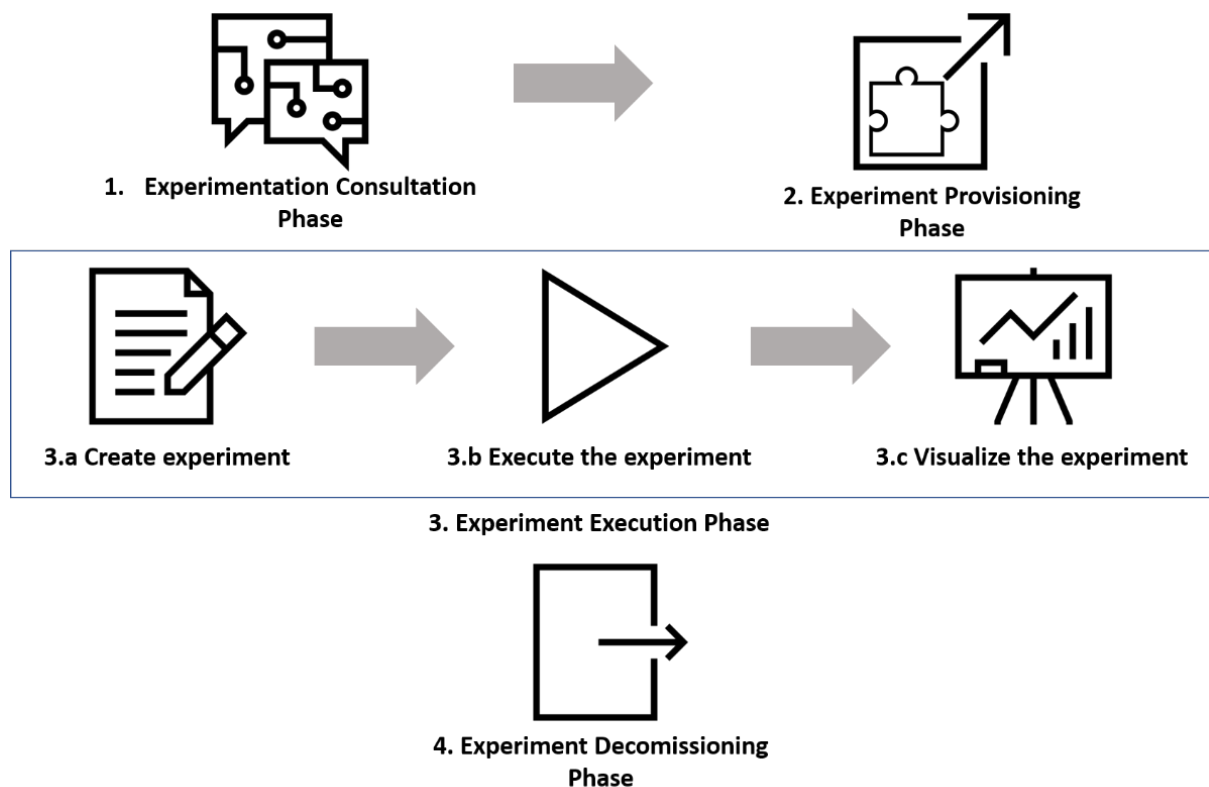


Figure 3 The 5GENESIS Experimentation Workflow

### 2.5.1 Experiment consultation phase

The experimentation procedure requires a close interaction between the Experimenter and the Platform Operator. The initial stage of the experiment includes a consulting work that will enable the Platform Operator to collect and understand the requirements of the Experimenter and perform the feasibility study that shall set the action plan necessary. During the consultancy phase:

- The Platform Operator will identify the type of experiment that must be executed and the needed measurements to validate the system, service or product under test, and will also guide the Experimenter to understand the experiment execution life cycle and needed interactions.
- The Experimenter shall define the macroscopic requirements so as to decide the equipment and systems to be onboarded for the experiment (later on the Experiment Execution Phase will elaborate the details). Therefore, there will exist an instruction phase to clarify what is necessary to be supported for the proper integration in the 5GENESIS platforms. The Platform Operator shall also instruct the Experimenter about the existing mechanisms to handle and expose measurements, and to ensure that the platform probes can effectively collect the required measurements and, appropriately, integrate them into the reporting and analytics components of the platform.

In the demo example of the MoM MCS Demo, during the consultation phase, the following is agreed:

MoM requires to use an eMBB 5G Service, as the MCS application supports video streaming communication between policemen. An URLLC service is also necessary for MCS alerting. The KPIs that are of interest are Latency and Throughput.

The experiment shall be executed by 5 policemen equipped with 5G phones and it will be run in the area close to the Ada Byron building at the University Campus of Málaga. The chosen test cases will be executed in a scenario under dense traffic conditions. The 5G phones shall be provided by MoM as they use confidential MoM information (contacts and applications).

### 2.5.2 Experiment provisioning phase

At this phase all preparatory actions prior to the experiment execution are considered:

- Once the measurement platform procedures are clarified, the Experimenter may need to introduce the necessary adaptations into the systems to be onboarded (either Network Service (NS), applications or equipment) since it should finally provide to the Platform Operator all the components of the solution under test. e.g. a mobile device, a mobile app, a NS, etc. In order to use the NS as part of an experiment, Experimenters must first provide all the necessary configuration artefacts such as the Virtual Network Function Descriptor (VNFD) packages, the Virtualized Infrastructure Manager (VIM) images required for the VNFDs and the Network Service Descriptor (NSD) that would manage the Virtual Network Functions (VNFs).
- The Platform Operator will deploy the solution in the platform and will produce, if needed, new test cases to cover the experimentation features requested by the Experimenter. Furthermore, at this stage, the Platform Operator shall proceed with the proper resources reservation and network provisioning to guarantee that the requested experiment can be executed with the required capabilities.

For the MoM MCS demo, at this phase:

NEM shall prepare the MCS service to be deployed in the Málaga cloud platform following the guidelines provided to ensure proper integration. NEM shall also provide the proper MCS application for the UEs. The Platform Operator will equip the UEs with the proper probes to collect the measurements required for the agreed test cases.

UMA needs to prepare the 5G infrastructure as necessary, and to make sure that network resources are available and properly configured. The URLLC and eMBB slices shall be configured as necessary. UMA will provide the SIM cards to be used during the execution of the experiments.

### 2.5.3 Experiment execution phase

Once the provisioning and experiment setup are confirmed, the experimentation execution can take place, including the steps of Experiment Creation/Definition, Experiment Execution, and Experiment Results Visualization. These are the steps that are systematically supported through the 5GENESIS Coordination Layer, as documented in detail in Section 3.

- **Create the Experiment:** The process of creating an experiment refers to the definition of the:
  - **Test Case** to be executed: Test cases define the KPIs to be measured, together with specific preconditions, and calculation methods. By default, the Experimenter will

have a list of standard test cases that cover technologic KPIs such as throughput and round-trip time, and custom test cases as agreed during the consultancy phase and implemented by the Platform Operator during the provisioning phase.

- **Test Scenario:** The scenario refers to the network and environment configuration where the experiment shall be executed.
- **Network Slice:** the list of predefined slices that shall be involved during the service execution, selected from the available ones offered by the platform.

The Experimenter can include standard test cases already provided by the Málaga Platform or new test cases, and thus a custom test case needs to be created by UMA. Since the experiment will be performed manually by 5 policemen, there is no need to create a custom automation sequence for the devices. However, it is necessary to initialize and close the different probes, and to collect the agreed set of measurements.

The measurements generated by the components located within the Málaga Platform premises can already be retrieved automatically. However, the devices are not directly connected to the testbed and a new method for retrieving the generated logs needs to be implemented. During the consultation phase it is agreed that the easiest way to accomplish this is to instruct the policemen on how to send these logs from their devices to a central repository, from where they can be automatically retrieved at the end of the experiment.

The functionality for retrieving and parsing the logs from the mobile devices to generate the necessary measurements needs to be developed ad-hoc for this experiment, following the methodology presented in Section 4.

- **Execute the Experiment:** Upon the experiment creation, the Experimenter can request the execution of the experiment and collect the respective results. Logs of previous experiment executions are also available.
- **Visualize the Experiment:** The Experimenter has access to a Grafana dashboard that provides live details of the experiment's execution, entailing diagrams of the collected measurements.

As part of the custom test case definition, a new dashboard template will be generated by UMA using the methodology presented in Section 5.1.2.4. This dashboard will include raw results, such as the instantaneous evolution of throughput and latency, memory and CPU usage on the phones and the different services deployed as VNFs.

The raw results can be also post-processed, and the output measurements specified in the test cases will be provided to the Experimenter. Table 9 provides an example of the report provided for the standard Mission Critical Push to Talk (MCPTT) access time test case.

Table 9 Example of the report for the MCPTT test case

Test Case ID	TC-MCPTT- 001, TC-MCPTT-002													
General description of the test	MCPTT Access time test. This test assesses the time between when an MCPTT User requests to speak and when this user gets a signal to start speaking. It does not include the MCPTT call establishment time, since it measures the time previously defined when the request to speak is done during an ongoing call.													
Purpose	Measure time from request to speak to permission granted in a MCPTT call. The MCPTT access time calibration tests aim at assessing the measurement capabilities of the measurement system employed for further MCPTT access time tests.													
Executed by	Partner:	UMA	Date: 09.07.2019											
Involved Partner(s)	UMA, NEM, ATOS													
Scenario	Athonet 4G Core with Nokia small cell with -17 dBm power and LTE band 7. The measurements are taken at the application level, in the NEM MCS application.													
Slicing configuration	VNF deployed at the compute node													
Components involved (e.g. HW & SW components)	NEM MCS applications and MCS server VNF, Nokia small cell eNB, Athonet 4G EPC, NEM (SONIM) UEs													
Metric(s) under study (see Section 4)	MCPTT													
Additional tools involved	Logcat Android log command-line tool													
Primary measurement results (those included in the test case definition)	<div>MCPTT Access time</div> <table><tr><td colspan="3">MCPTT access time [ms]</td></tr><tr><td rowspan="2">Mean</td><td colspan="2">95% confidence interval for Mean</td></tr><tr><td>Lower bound</td><td>Upper bound</td></tr><tr><td>55,192</td><td>50,114</td><td>60,271</td></tr></table>			MCPTT access time [ms]			Mean	95% confidence interval for Mean		Lower bound	Upper bound	55,192	50,114	60,271
MCPTT access time [ms]														
Mean	95% confidence interval for Mean													
	Lower bound	Upper bound												
55,192	50,114	60,271												
Complementary measurement results	n/a													

### 2.5.4 Experiment decommissioning phase

Upon the completion of the trial, the systems provided to the Experimenter are decommissioned from the 5GENESIS platform and all provided equipment is returned. The created users and experiment results are nevertheless maintained for some months after the decommission takes place, so to better support the evaluation processes of the Experimenter (the specific clauses are discussed and defined during the Consultancy phase).

## 2.6 What kind of experiment can be executed?

The experimentation methodology adopted in 5GENESIS is very flexible and is open to run a wide range of experiments, always subject to agreement with the platforms. While the section 2.4 includes an example of a field test experiment, this section introduces two new experiments that try to illustrate different types of tests that can be executed in the 5GENESIS platforms.

### 2.6.1 Testing network services

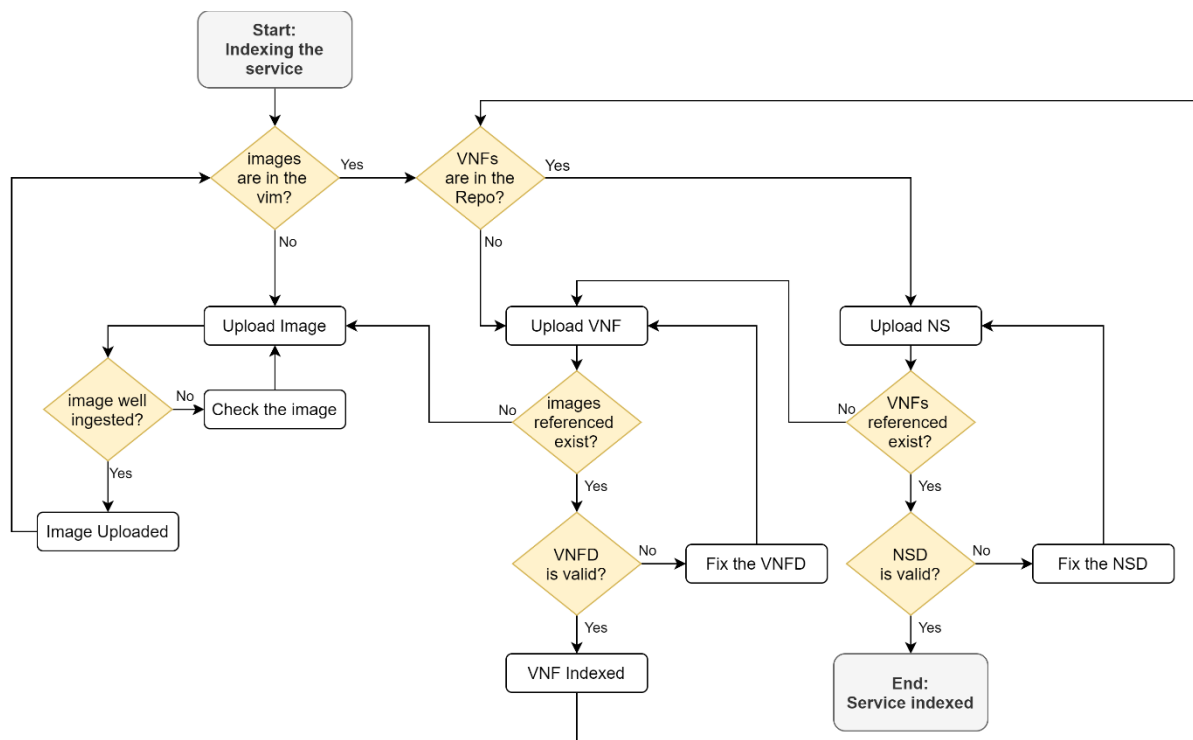
For the Experimenters in the group of Technology Providers and, more specifically, the network service developers that would require to evaluate the performance of a NS, the experimentation procedure is described as follows.

The experiment starts with a consultancy phase, as described in Section 2.4.1, to understand the required resources to on board and to instantiate as well as, the parameters that need to be monitored to test the performance of the NS.

The platforms integrate a monitoring framework that includes infrastructure monitoring probes. Service-specific measurements need to be provided by the Experimenter. The Platform Operator will provide the specific probes and instructions to equip the NS with specific probes. Once the required probes are in place, the NS is ready for the on boarding.

The flow diagram shown in Figure 4 explains in detail the flow for network service onboarding. The required artefacts are images, VNFs and NS packages. Each VNF has dependencies with the images onboarded in the VIM and each NS has dependencies with the VNFs onboarded in the NFVO (NFV Orchestrator).





**Figure 4 Network Service Onboarding flow diagram**

The flow has a clear start and ending points (grey boxes). There are several split decisions (orange diamonds) and actions to be committed (white boxes).

In Release A of the 5GENESIS platforms, the verticals had to provide the binaries/images and the descriptors to the platform operator for the onboarding operation.

In Release B, 5GENESIS offers an appropriate interface to facilitate the onboarding process. Upon onboarding of each separate component, the Experimenter receives a unique identifier that can be used for referencing the onboarded component during the definition of the experiment.

The platform will implement the test cases required by the Experimenter to test the NS performance. The experiment can define different experiments combining, for example, the test cases and different versions of the NS.

## 2.6.2 Testing user equipment

In this case, the Experimenter is a UE manufacturer that needs to assess the throughput reached by the device. For this experiment, the steps to run are the following ones:

1. The Experimenter will contact the platform operator to check the experimentation features of the platform and to detect incompatibility problems, if any.
2. If the equipment manufacturer wants to run automated experiments remotely, a plugin for the control and configuration of the equipment must be developed (see Section 4). The Platform Operator will develop the required plugin in close cooperation with the equipment manufacturer. If the equipment manufacturer wants to run field tests by themselves, the development of plugins for automating the control and configuration of the UEs is not required.

3. As in this case the Experimenter is interested in executing throughput tests, the Platform Operator can install, in the UE under test, one of the iPerf agents developed by the project, assuming that they are compatible. If the operating system is not compatible with the available agents, or if the Experimenter is interested on custom measurements, the Platform Operator will instruct the Experimenter on how to develop/modify the custom monitoring agents to inject the collected data into the 5GENESIS monitoring framework. The degree of integration of the monitoring capabilities of the UE into the platform will depend on the Experimenter needs.
4. The Experimenter will travel, or send the devices, to the Platform Operator. Once the devices have been received and plugged into the 5G network, the equipment manufacturer can start the execution of the tests.

## 3 DOCUMENTATION FOR EXPERIMENTERS

---

This section illustrates how Experimenters can use the 5GENESIS platforms to carry out their experiments to validate or demonstrate the performance of their products, applications, or services.

The Experimenter can set experiments and get results through the 5GENESIS Portal, as well as directly execute the experiments via the Open APIs. Due to its simplicity, defining the experiments via the Portal is probably the preferred option for most of the Experimenters that are getting familiar with the framework or do not have a technical background to cope directly with the Open API. The Portal offers a way to run experiments in a controlled environment every time the Experimenters want. Instead, the Open API provides the interface to handle experiments and experiment runs in a more flexible way, being able to automate the execution of it easily. Supporting documentation for both alternatives is provided in the following subsections.

### 3.1 The 5GENESIS Experimentation Methodology

The experimentation methodology in 5GENESIS, introduced in Deliverable D2.3 [4] is driven by the execution of test cases. In a nutshell, a test case defines the targeted KPI, the required measurements and the test conditions. For more details about the test cases specification please refer to D6.1 that contains practical examples of test case executions [16]. With this in mind, the platforms provide a list of standard test cases that cover generic technological KPIs defined for 5G. Specific experimentation requirements coming from different verticals can be covered with the definition and implementation of custom service-specific test cases that will be subsequently listed in the 5GENESIS Portal.

The test cases define the target KPI specific metric and how to execute the test, but it is also required to specify the network configuration. In the 5GENESIS Experimentation Methodology, the network configuration is defined via the deployed slice (resources explicitly assigned) and the scenario (parameters that are configured to reproduce different test conditions based on standardized network fluctuations).

A test case includes information related to the configurations of the experimentation platform needed for receiving the measurement(s). The KPI definition, the measurements methodology and the information for the equipment preparation are added in this field. More precisely, a test case provides the following information:

- **Target KPI.** Each test case targets a single KPI. Secondary/complementary KPIs could also be defined as complementary measurements (see below). The definition of the main target KPI specializes the related target metric, i.e., the definition of the main KPI declares at least the reference points from which the measurement(s) will be performed, the underlying system, and the reference protocol stack level.
- **Complementary measurements.** A secondary list of useful KPIs to interpret the values of the target KPI. Getting these measurements is not mandatory for the test case. However, they allow to provide additional set of results besides the target measurement, an additional and useful context data for the analysis, and an

interpretation of the obtained main KPIs (e.g. several additional KPIs of the same metric give more data information and distribution of results as in the examples where the main KPI is the mean score but it is also fed by percentile KPIs).

- **Applicability.** A list of features and capabilities that are required by the system in order to guarantee the feasibility of the test.

The list of test cases that have been defined by the 5GENESIS project is available in deliverable D6.1 [16].

**Table 10 List of test cases available in Release B**

Test case	Target	Output	Platforms
<b>Round-trip-time</b>	The RTT measured from a client to a server over a mobile network.	Average, maximum, minimum, 95% confidence interval, 5% confidence interval	Athens, Málaga, Limassol, Berlin, Surrey
<b>Latency</b>	The delay between a client and a server at application layer	Average (expected mean) E2E Application Layer	Limassol
<b>Throughput</b>	Throughput measurements between a client and a server over a mobile network	Average, maximum, minimum, 95% confidence interval, 5% confidence interval	Malaga, Athens, Limassol, Berlin, Surrey
<b>Streaming</b>	KPIs related to streaming service such as jitter and adaptation	Average	Athens, Malaga
<b>Service creation time</b>	The time needed to deploy virtual machines (VMs) on compute systems	Average, 95% confidence interval	Malaga, Berlin
<b>MCPTT</b>	KPIs on standardized MCPTT services such as access time	Average, maximum, minimum, 95% confidence interval, 5% confidence interval	Malaga

The test cases listed in deliverable D6.1 are the *standard ones* defined by the consortium in the project framework. However, it is also possible for the Experimenter to define new test cases to cover specific measurements, or testing requirements requested by the verticals. In the same way, it is also possible for Experimenters to work on ways to expose additional test cases in the Open API or Portal or expose test cases currently not available for automation (e.g. capacity, density of users, ubiquity, etc.).

Each 5GENESIS platform will also provide a predefined list of end-to-end slices covering radio resources configuration, mobile core network, transport network and resources allocated in the virtualized infrastructures offered by the platforms. This list will be available in the Portal for setting up an experiment.

Finally, each platform will offer a list of scenarios that will depend on the technologies and supported releases. Each scenario includes information related to network, service and environment configurations and it is specific to the selected technologies and the target system. From the performance perspective, the scenario reproduces network conditions that impact the values of the KPIs to be measured. More precisely, a test case that targets a specific measurement can be set for different scenarios and conditions that declare parameters such as the level of the transmission power in a base station, the emulated mobility of the end devices, the background traffic load in the system, etc.

## 3.2 Experimentation via the Portal

Figure 5 depicts, in more detail, the steps to run an experiment via the 5GENESIS Portal. Each step is explained in the following subsections.

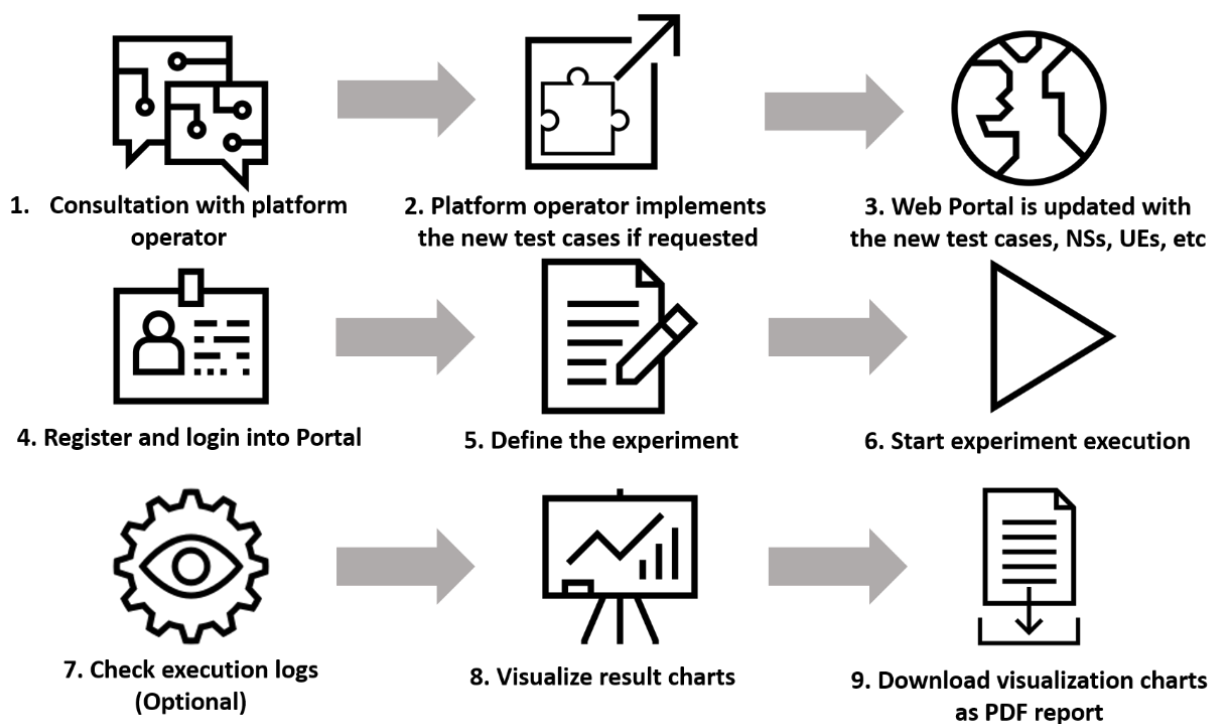


Figure 5 Experimentation via the Portal

### 3.2.1 Login

Each platform has deployed its own instance of the Portal. The 5GENESIS Portal allows external users to perform experiments in the 5GENESIS platforms, as well as to retrieve the results produced by the experiments. Prior to that, the first necessary steps to perform experimentation through the 5GENESIS Portal are the Experimenter registration and login.

To proceed with the registration, the user must click on the “Register” tab in the upper part of the Portal website. All the required information must be filled in, and then the user must click the “Register” button (as depicted in Figure 6). All the information will be automatically validated, and the new account will be successfully created if the information is correct. The authentication and authorization procedure are done using the so-called Authenticator (described in D3.8 [19]) that manages the authentication, confidentiality, integrity and non-

repudiation security features of the platform. Experimenters are registered to it and get assigned an access token. The access token can be claimed using Basic Authentication (username + password). It is a short-lived token, so it is renewed before its expiration date using a refresh token procedure. It is also possible to authenticate every request using Basic Auth with a user already registered and validated by the Platform Administrator.

Once the user has an active account, the login is straightforward. The user must click in the “Login” tab in the upper part of the Portal website, then fill in username and password, and click on the “Sign In” button, as shown in Figure 7.

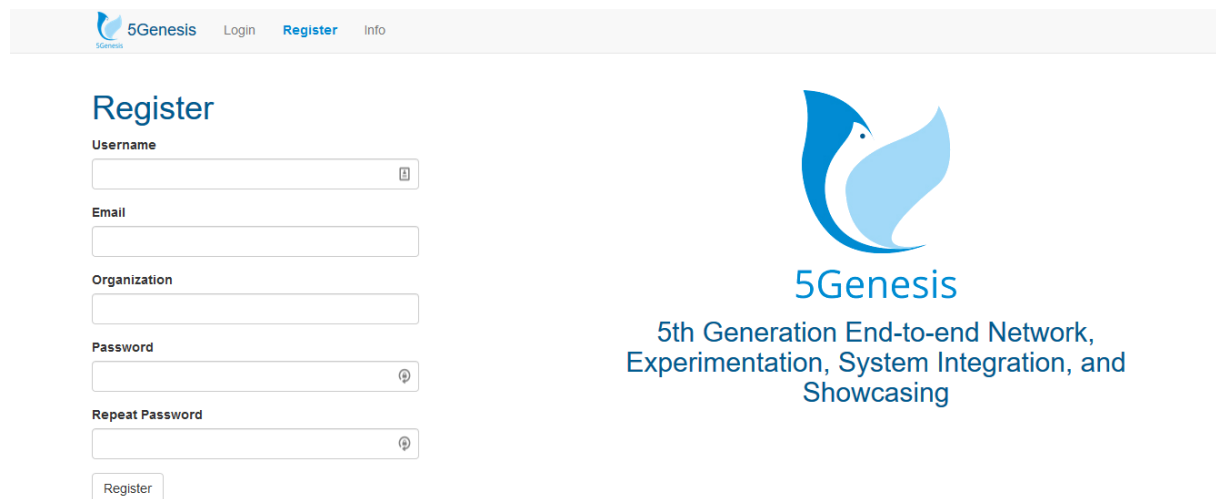


Figure 6 5GENESIS Portal registration screen

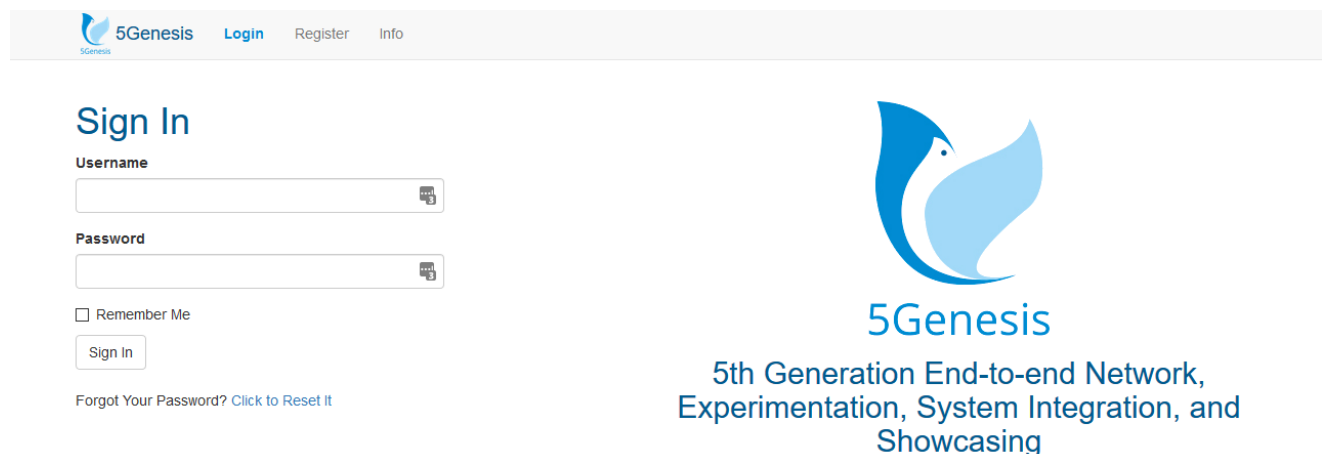



Figure 7 5GENESIS Portal sign in screen

If the Experimenter has an active account and has successfully logged in, then it is possible to continue with the next steps: the creation of experiments, their execution and, finally, the visualization and retrieval of the experiments results.

The “Info” tab (see Figure 8) in the upper part of the Portal interface, which is present independently of the user being logged in or not, offers information about the available testbed devices, scenarios and network functions (NFs).

 5Genesis

[Login](#) [Register](#) [Info](#)

Figure 8 Example of a 5GENESIS Portal info page

### 3.2.2 Definition of experiments

The “Create Experiment” tab is available when the user is logged in. This tab allows the user to define the parameters for an experiment. The parameters that can be set are the name for the experiment, its type, the test cases to execute, the devices used for the experiment, and the network slice.

As Figure 9 shows, the test cases, devices and slice that can be selected from a list depending on the availability of the specific platform. Once all the parameters have been appropriately set for the experiment, the user must click on the “Add Experiment” button, which will make the experiment available in the experiments list in the user’s dashboard.

As previously mentioned, if the Experimenter needs different test cases or devices from the ones available in the testbed at the moment, this can be discussed with the testbed operator to further study the requested additions so that the Experimenter needs are fulfilled.

## CREATE EXPERIMENT

**Name**

**Type**

☐ Avoid running other experiments at the same time

**Test Cases**

☐ MCPTT Test  
☐ Maximum User Data Rate  
☐ Round Trip Time  
☐ Service Creation Time  
☐ Simple Test Case

**UEs**

☐ Galaxy Note S10  
☐ Galaxy S9

☒ **Network slicing**

**Slice**

**Scenario**

**Network Services**

Figure 9 Experiment creation screen

It is important to note that if an experiment requires a NS, either the NS will already be available, or the Experimenters have uploaded the required NS artefacts themselves. The NS with its dependencies are on boarded onto the NFVO only after analysing the Experiment Descriptor. Once the validation is done, exclusively the required components of the NS will be on boarded instead of the whole set of components within the NS, making the process more effective and optimizing the resources in the platform. The network service on boarding interface can be seen in Figure 10.




### Basic Information


Name

Test NS

Location

 Main

Visibility

 Public


Description

A test network service

Update

✖ Network service not ready

### Virtualized Infrastructure Manager

Vim Image: ubuntu\_16.04 

### VNFD Packages

No VNFD packages defined

Available VNFDs:

ExampleVnfd\_1

Add

Add VNFD package

Browse

Pre-load

### Network Service Descriptor

Available NSDs:

ExampleNsd

Select

Add NSD file

Browse

Pre-load

Figure 10: Network services on boarding

All network services available to the experimenter (which includes public network services as well as those defined by the experimenters themselves) are visible in the network services dashboard. The user can continue editing previously created network services and easily see which ones can be used while defining a new experiment, since only those that are marked as ready are selectable in the experiment definition interface.

### 3.2.3 Execution of experiments

The user's dashboard, which can be accessed clicking in the "Home" tab while being logged in, shows all the experiments previously created by the user. This dashboard allows the user to start the execution of an experiment or to view the history of both previous and current executions of a specific experiment, using the buttons "Run experiment" and "Executions" respectively, as seen in Figure 11.

The screenshot shows the 5Genesis User's dashboard. At the top, there is a navigation bar with the 5Genesis logo, a 'Home' link, and buttons for 'Create Experiment', 'Create Distributed Experiment', 'Network Services', and 'Info'. A user profile 'm - Logout' is in the top right. The main section is titled 'EXPERIMENTS' and contains a table with columns: ID, Name, Type, and Actions. The table lists three experiments: ID 4 (test, Standard), ID 3 (Remote, Distributed), and ID 1 (1, Distributed). Each experiment has associated test cases, UEs, and a remote location. Action buttons like 'Run', 'History', 'Descriptor', and 'Configure' are provided for each experiment. On the right, an 'ACTIONS' sidebar shows a list of recent experiment runs with timestamps and the number of runs.

ID	Name	Type	Actions
4	test	Standard	TestCases: Test2 Run History Descriptor
3	Remote	Distributed	TestCases: MalagaSide UEs: MalagaUE Remote: Atenas Configure History Descriptor
1	1	Distributed	TestCases: MalagaSide UEs: MalagaUE Remote: Atenas TestCases: AtenasSide UEs: AtenasUE Run History Descriptor

Figure 11 User's dashboard

The experiment execution history section, accessible through the “Executions” button, shows the status of all the experiments’ executions and information such as their start and end times, as shown in Figure 12.

For every execution of the experiment, there are two buttons available: (a) “Execution Logs” and (b) “Results”. The “Results” button becomes available only if the experiment has finished successfully. Additionally, the “Run Experiment” button in the upper left part serves as a shortcut to execute (re-run) the experiment again.

#### Experiment 2: Video

Type: Standard

Run Experiment View descriptor

EXECUTIONS				
Execution ID	Status	Start Time	End Time	Action
20008	Finished	07 May 2021, 11:36:49	07 May 2021, 11:38:19	   
20007	Finished	07 May 2021, 11:34:39	07 May 2021, 11:36:09	   
20006	Finished	07 May 2021, 11:33:08	07 May 2021, 11:34:38	   
20005	Finished	07 May 2021, 11:31:08	07 May 2021, 11:32:38	   
20004	Finished	07 May 2021, 11:25:58	07 May 2021, 11:26:58	   
20003	Finished	07 May 2021, 11:22:08	07 May 2021, 11:22:38	   
20002	Finished	07 May 2021, 11:17:18	07 May 2021, 11:18:18	   
20001	Finished	07 May 2021, 11:11:58	07 May 2021, 11:12:28	   

Figure 12 Experiment executions screen

The first button leads to a screen, depicted in Figure 13, where the user can check the log messages generated during the experiment execution, divided into three different execution stages (Pre-Run, Run and Post-Run). The log messages can also be filtered by severity. Besides, this screen also shows the information present in the “Executions” screen for this specific execution: the status, start and end time, as well as the experiment executed. The second and

third buttons lead to the result visualization interfaces described in Section 3.2.4. Finally, the fourth button provides access to a compressed file that contains all the logs and results in CSV format.

The Release B brings an additional validation step while running or launching an experiment due to the presence of the Experiment Distributor. This module validates the target platform(s) of the experiment so that the experiment is distributed to a single platform or more than one if a cross-platform experiment is launched.

The “Results” buttons lead to an interactive visualization of the most important results generated by the experiment.

### 3.2.4 Visualization of results

The Portal provides access to two different interfaces for reviewing the results of the experiments: a set of raw results can be visualized by using customized Grafana dashboards, while the full set of obtained data as well as extended statistical analysis capabilities are available via the Analytics Dashboard.

#### 3.2.4.1 Grafana dashboard

The Grafana dashboard is accessible through the “Results” button shown before. By pressing this button, the user will be redirected to the corresponding dashboard in the Grafana dashboard, which is customized for each kind of test case. The Experimenter can zoom in any of the included graphs in order to see a detailed view of the selected periods of time. An example result dashboard can be seen in Figure 13.

Platform administrators configure the dashboards so that they display the measurements specified in the test cases and any additional measurement collected by the probes available at the platforms. If an Experimenter requires a custom dashboard, they can contact the platform administrator. It is also common to define new dashboard templates as part of the creation of custom experiments.

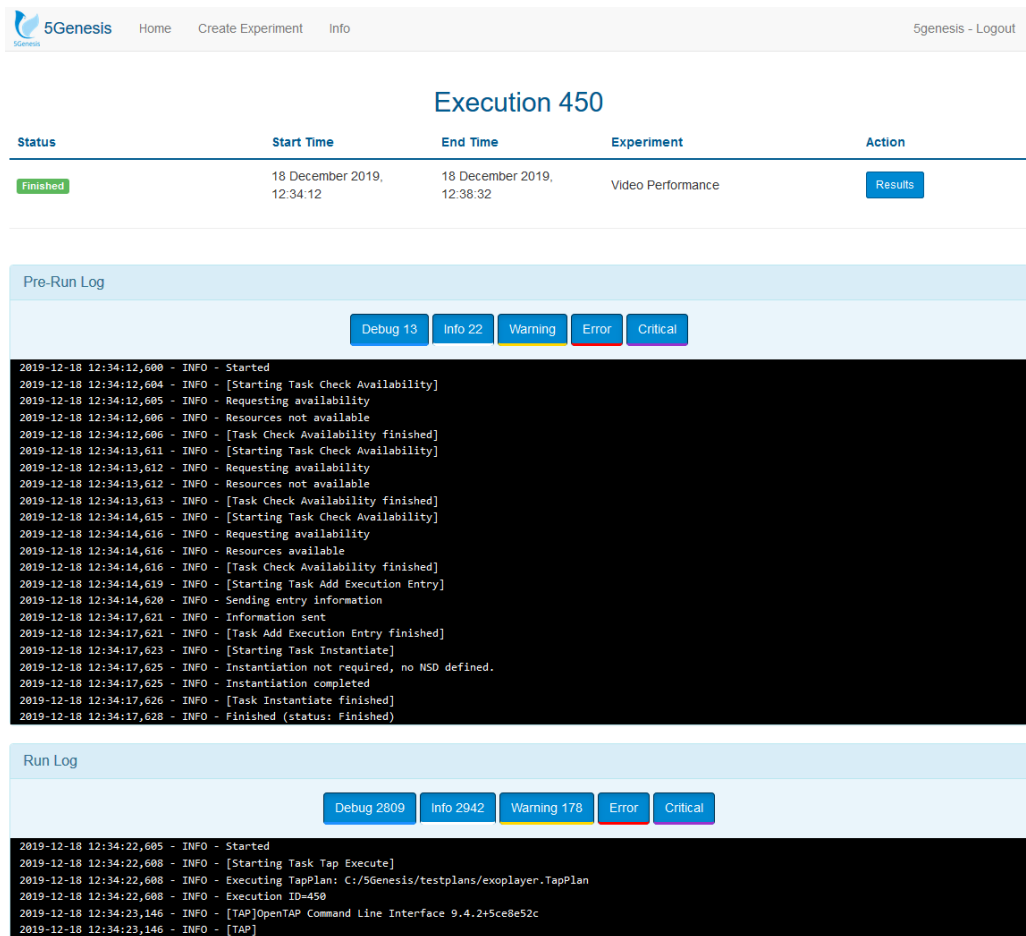


Figure 13 Experiment execution log screen



Figure 14 Experiment result visualization screen



Figure 15 Generated PDF results report

The different available visualization options of a dashboard can be seen in Section 5.1.2.4. However, it should be noted that the definition of these dashboards is performed by a platform administrator.

The “PDF Report” button, present at the top-right corner of the dashboard, allows the user to download all the result graphs available in the dashboard as a PDF file. An example PDF file with extracted dashboards is shown in Figure 15.

If the Experimenter requires different graphs or visualization options for an experiment, this can be requested from the platform administrators for checking how they can better fulfil the experiment’s necessities.

### 3.2.4.2 Analytics Dashboard

The Analytics Dashboard provides access to all the results generated by the experiment, as well as additional statistical analysis capabilities such as prediction, correlation and feature selection. The interface of the “Statistical Analysis” screen of the Analytics Dashboard can be seen in Figure 16. Since the Analytics Dashboard is fully integrated with the authorization framework of 5Genesis, experimenters do not need to log into this interface or keep track of additional user credentials.

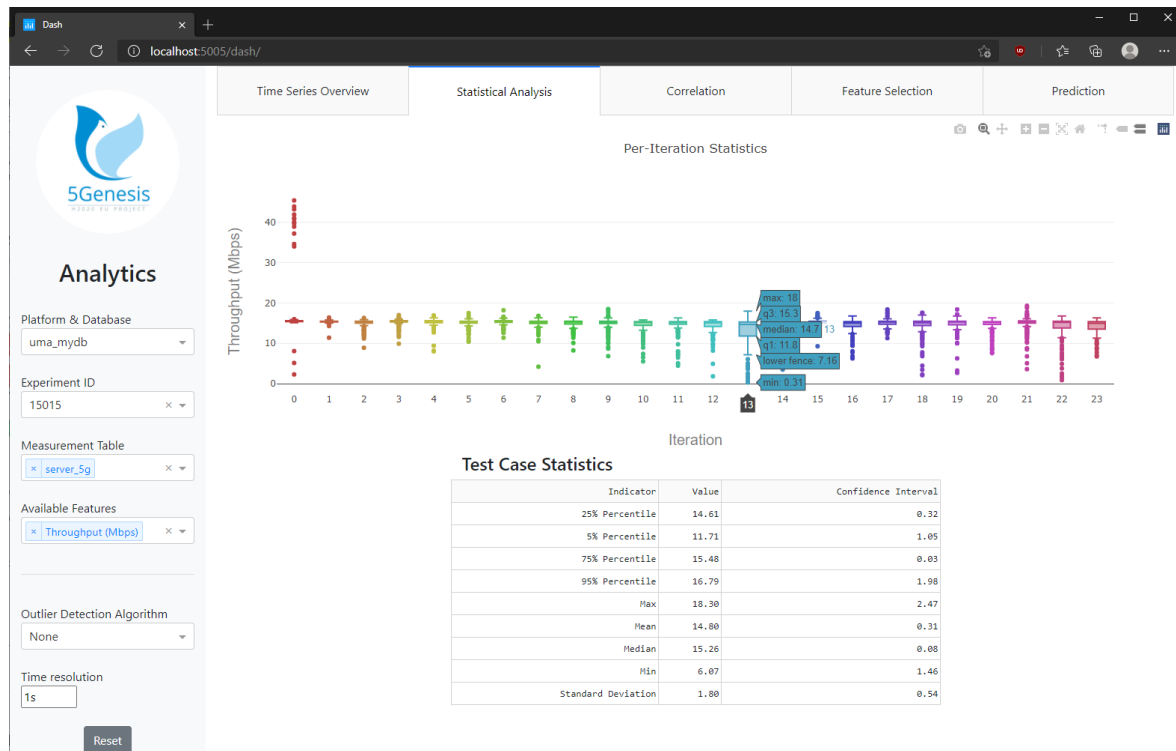


Figure 16 Analytics dashboard interface

Additionally, the Analytics Dashboard allows the creation of different kinds of graphs based on the experiment's results, which experimenters can export for use outside of the Analytics Dashboard.

More information about the Analytics Dashboard can be seen in Deliverable D3.6 [7].

### 3.3 Experimentation via the Open APIs

Release B offers a set of Open APIs that are accessible for the communication between the Portal and the ELCM. However, it is possible for the Experimenter to manually use these same endpoints. This way, Experimenters may request access to this internal communication channel and perform actions on the testbed (available Open API grouped in Figure 17).

Figure 18 shows the experimentation steps when using the Open APIs.

## 5GENESIS Dispatcher Open APIs <sup>2.7.</sup>

[ Base URL: DISPATCHER:8082/ ]

Open APIs specification for 5GENESIS Dispatcher

[Terms of service](#)

[Contact the developer](#)

[Apache 2.0](#)

[Find out more about Swagger](#)

Schemes	HTTPS	Authorize
<b>MANO</b>	MANO OSM Repository and VIM operations	>
<b>ELCM</b>	ELCM interfaces, Experiment Descriptor Validator, Creator & Distributor	>
<b>Result Catalog</b>	Operations to retrieve the results data from experiments execution	>
<b>Auth</b>	Operations for users in order to access to the different microservices	>
<b>Auth: Admin Functions</b>	Operations for Admin to manage the user Auth	>

Figure 17: OPEN APIs interfaces

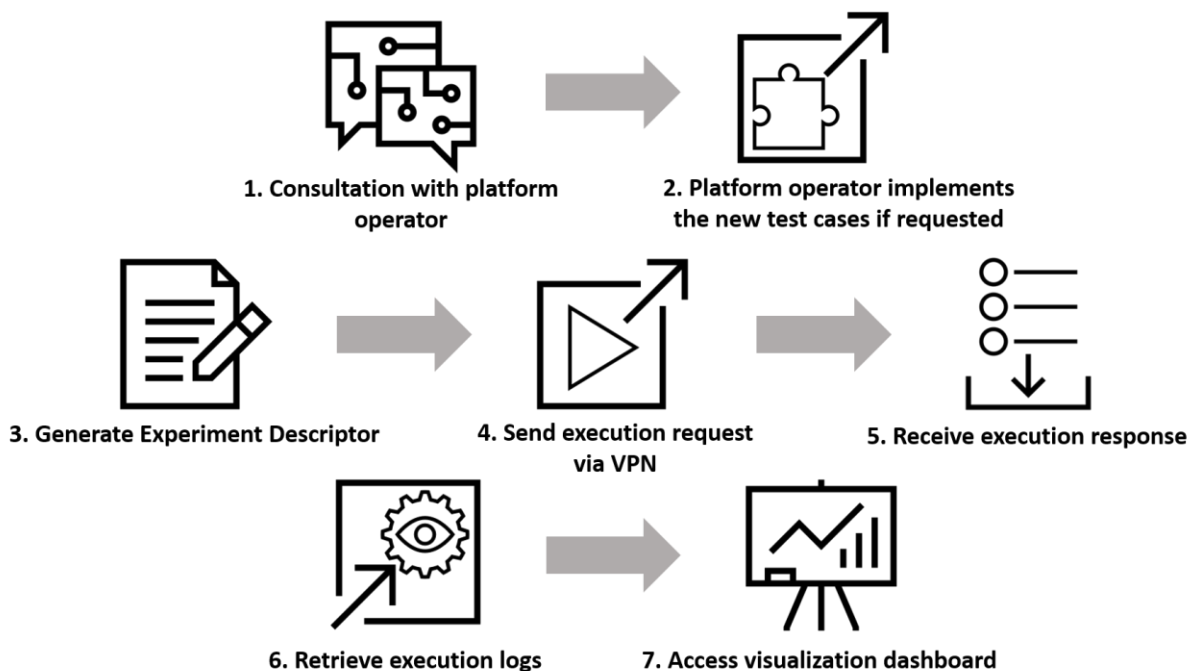


Figure 18 Experimentation via the Open APIs

### 3.3.1 Authorization

As explained in the Portal definition in Section 3.2, the system provides and requires authorization in its Release B. For that, the corresponding Open API definition is listed. The following swagger, lists all the different calls from the experimenters perspective and from the perspective of the platform administrator of the system.

Auth Operations for users in order to access to the different microservices			▼
GET	/auth/get_token	Get token by Basic Auth	🔒
PUT	/auth/change_password	Change Password	🔒
POST	/auth/register	Register User in the platform (But not activated)	
PUT	/auth/recover_password	Recover Password by email	
Auth: Admin Functions Operations for Admin for managing the user Auth			▼
GET	/auth/show_users	Show all the users in the platform	🔒
PUT	/auth/validate_user/{username}	Validate registered users	🔒
DELETE	/auth/drop_db	Drop Users DB	🔒
DELETE	/auth/delete_user/{username}	Delete a specific User	🔒
POST	/auth/register_platform_in_platform	Register this current Platform in another platform	🔒
GET	/auth/show_platforms	Show all the current platforms	🔒
PUT	/auth/validate_platform/{platform_name}	Validate Platform for distributing experiments	🔒
DELETE	/delete_platform/{platformName}	Delete Platform	🔒

Figure 19: Authorization Open APIs interfaces

In short the available Open API calls are:

- Oriented for end-user or Experimenter
  - User Registration
  - Password change
  - Get Token
  - Recover Password
- Oriented for administrator(s)
  - Show registered users in the system
  - Drop user database
  - Delete a single user from the database
  - Activate a single user from the database
  - Register platform in platform (In a Dispatcher environment)
  - Validate platform
  - Show platform
  - Delete platform

### 3.3.2 Provisioning

The definition of the network service together with its descriptors and images can be completed step by step at different times, while the Portal keeps track of the current status. For the selection of each component the Portal provides a list of existing artifacts that are



available, or allows the user to upload a new one. Figure 20 shows the endpoints used during the onboarding process:

MANO MANO OSM Repository and VIM operations			▼
POST	/mano/vnfd	Add a VNFD or new VNFD version to the repository	🔒
GET	/mano/vnfd	List VNFDs located in the repository	🔒
POST	/mano/nsd	Add a NSD or new NSD version to the repository	🔒
GET	/mano/nsd	List NSDs located in the repository	🔒
POST	/mano/image	Upload and register an image file in the VIM	🔒
GET	/mano/image	Get the list of the images in the VIMs	🔒
GET	/mano/vims	Retrieves the list of registered VIMs in the mano.conf file	🔒
POST	/mano/onboard	Onboard one NS in OSM	🔒
DELETE	/mano/nsd/{nsdId}	Deletes a NS	🔒

Figure 20: Open APIs Mano interfaces

These artifacts redirect the calls to the MANO Wrapper component to finalize the onboarding in the system:

- Retrieve a list of available VIMs
- Retrieve the list of available images in each VIM
- Onboard a new VIM image
- Onboard a new NS
- Delete a specific NS
- Retrieve the list of available VNFDs
- Onboard a new VNFD
- Retrieve the list of available NSDs
- Onboard a new NSD

### 3.3.3 Description of the Experiment Descriptor Template

The details about the creation of an Experiment Descriptor are abstracted by the Portal. The Portal uses the information provided by the Experimenter during the experiment definition (see Section 3.2) in order to create a JSON object that describes the experiment to run. This object is then sent to the ELCM to start the execution.

When using the Open APIs directly, the Experimenter must generate this Experiment Descriptor manually. The Experiment Descriptor adheres to the following template:

```
{
  "Application": Optional[str],
  "Automated": bool,
  "ExclusiveExecution": bool,
  "ExperimentType": str,
  "Extra": Dict[str, str] (may be empty),
  "NSs": List[Tuple[str, str]] ((nsd id, vim location) may be empty),
  "Parameters": Dict[str, str] (may be empty),
  "Remote": Optional[str],
  "RemoteDescriptor": Optional[<Remote descriptor>],
  "ReservationTime": Optional[int],
  "Scenario": Optional[str],
  "Slice": Optional[str],
  "TestCases": List[str] (may be empty),
  "UEs": List[str] (may be empty),
  "Version": str
}
```

Figure 21: Experiment Descriptor Template

The expected contents of each field are:

- **Application:** For MONROE experiments, the name of the container to execute in the MONROE node.
- **Automated:** Whether or not to execute Test Cases during the experiment. For non-automated experiments, the validation framework will deploy the requested network slice, and keep it available for the **ReservationTime** specified.
- **ExclusiveExecution:** Whether or not the experiment can be executed alongside other experiments in the testbed.
- **ExperimentType:** Type of the experiment. Accepted values are 'Standard', 'Custom', 'MONROE' and 'Distributed'.
- **Extra:** A generic dictionary of values reserved for usage on third party applications or future extension of the Experiment Descriptor.
- **NSs:** Contains a list of tuples, where each entry selects the network service to deploy, and the VIM where the NS will be deployed.
- **Parameters:** Contains a dictionary of customized values for use on Custom experiments, or the configuration of the MONROE node on MONROE experiments.
- **Remote:** For use on distributed experiments. Define the name of the remote testbed to use during the experiment.
- **RemoteDescriptor:** For use on distributed experiments. The contents are the same as on a normal Experiment Descriptor, except for the **RemoteDescriptor** key, which is removed. This descriptor contains the definition of the experiment for the **Remote** side.
- **ReservationTime:** For use during non-automated experiments, sets the number of minutes that the experiment will be kept running.
- **Scenario:** The name of the scenario to use for customizing the values of the used **Slice**.
- **Slice:** The name of the base slice descriptor to use during the experiment.
- **TestCases:** The list of test cases that are to be run during the experiment execution.

- **UEs:** The list of UEs to be used during the experiment.
- **Version:** Version of the experiment descriptor format used ('2.1.0' at the time of writing).

### 3.3.4 Sending execution requests to the ELCM

The ELCM component has multiple operations described available in the swagger service, that will be authorized by the dispatcher. The Open API endpoints involve validation of the experiment descriptor, ELCM functionalities (both CRUD experiments and the consult of the ELCM resources).

ELCM ELCM interfaces, Experiment Descriptor Validator, Creator & Distributor			
POST	/elcm/validate/ed	Validate Experiment descriptor	🔒
POST	/elcm/api/v0/run	Run an Experiment Descriptor	🔒
GET	/elcm/execution{id}	Execution information	🔒
GET	/elcm/execution{id}/cancel	Cancel execution	🔒
GET	/elcm/execution{id}/delete	Delete execution	🔒
GET	/elcm/execution{id}/json	Retrieve JSON data from a experiment	🔒
GET	/elcm/execution{id}/logs	Logs of the experiment execution	🔒
GET	/elcm/execution{id}/results	Execution results	🔒
GET	/elcm/execution{id}/descriptor	Retrieve the Execution Experiment Descriptor	🔒
GET	/elcm/facility/testcases	Available testcases	🔒
GET	/elcm/facility/ues	Available UEs	🔒
GET	/elcm/facility/scenarios	Current scenarios	🔒
GET	/elcm/facility/baseSliceDescriptors	Current slices	🔒

Figure 22: ELCM Open APIs interfaces

In order to execute certain experiment, the ELCM listens and waits for execution requests via its REST API. The specific endpoint for this is located at `/api/v0/run`, and accepts POST requests that include the Experiment Descriptor as payload.

These requests can be sent, for example, by using the cURL [20] application. The following command is an example of such requests, assuming that the Experiment Descriptor is saved to a file called `ed.json`:

```
curl -X POST -H "Content-type: application/json" \
-d @ed.json "<ELCM_host>:<ELCM_port>/api/v0/run"
```

The ELCM will reply with a JSON object that contains the following information:

```
{
  "Success": <Boolean>,
  "ExecutionId": <Integer>,
  "Message": <String>
}
```

- **Success:** A Boolean value indicating if the request has been correctly processed.

- **ExecutionId**: The unique ID assigned to the experiment execution, or *None* in case of error.
- **Message**: A descriptive message. In case of error it may contain information about the issues detected.

### 3.3.5 Accessing results and logs

Since the executed experiments through the Open APIs are not registered in the Portal, it is not possible to access to the Grafana visualization of the results or view the execution logs in the Portal interface. However, the logs can be retrieved from the ELCM, and the Grafana dashboard is created and can be accessed using the ExecutionId returned by the ELCM when sending an execution request.

The request to obtain the refined data comes from the statistical container from the analytics module. In the same way, there is also a way to request raw data results in order to obtain the values used by the analytics components to plot graphs and calculate statistical operations. The returned values will be different depending on the applied filters in the request.

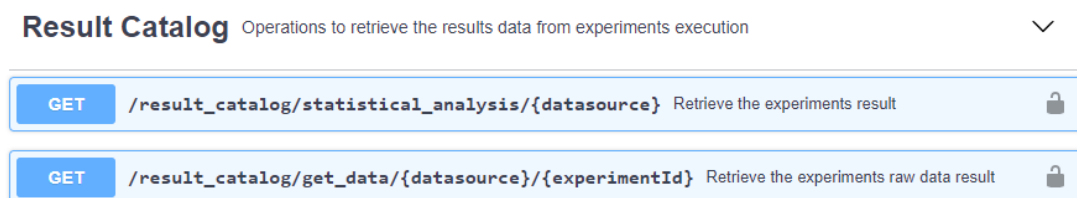


Figure 23: Result catalog Open APIs interfaces

## 4 DOCUMENTATION FOR TECHNOLOGY PROVIDERS

This section is oriented towards technology providers that are willing to incorporate their products into a 5GENESIS experimentation platform. To this end, technology providers need to know the APIs available to interconnect their systems with the control, configuration and monitoring planes available in a 5GENESIS platform. The integration is done through the development of plugins which will enable the new component to communicate with the key components of the 5GENESIS Experimentation Framework (i.e. the [Open 5GENESIS Suite](#)). As shown before, the plugins implement all the functions required to communicate with the ELCM, the Slice Manager and the monitoring system.

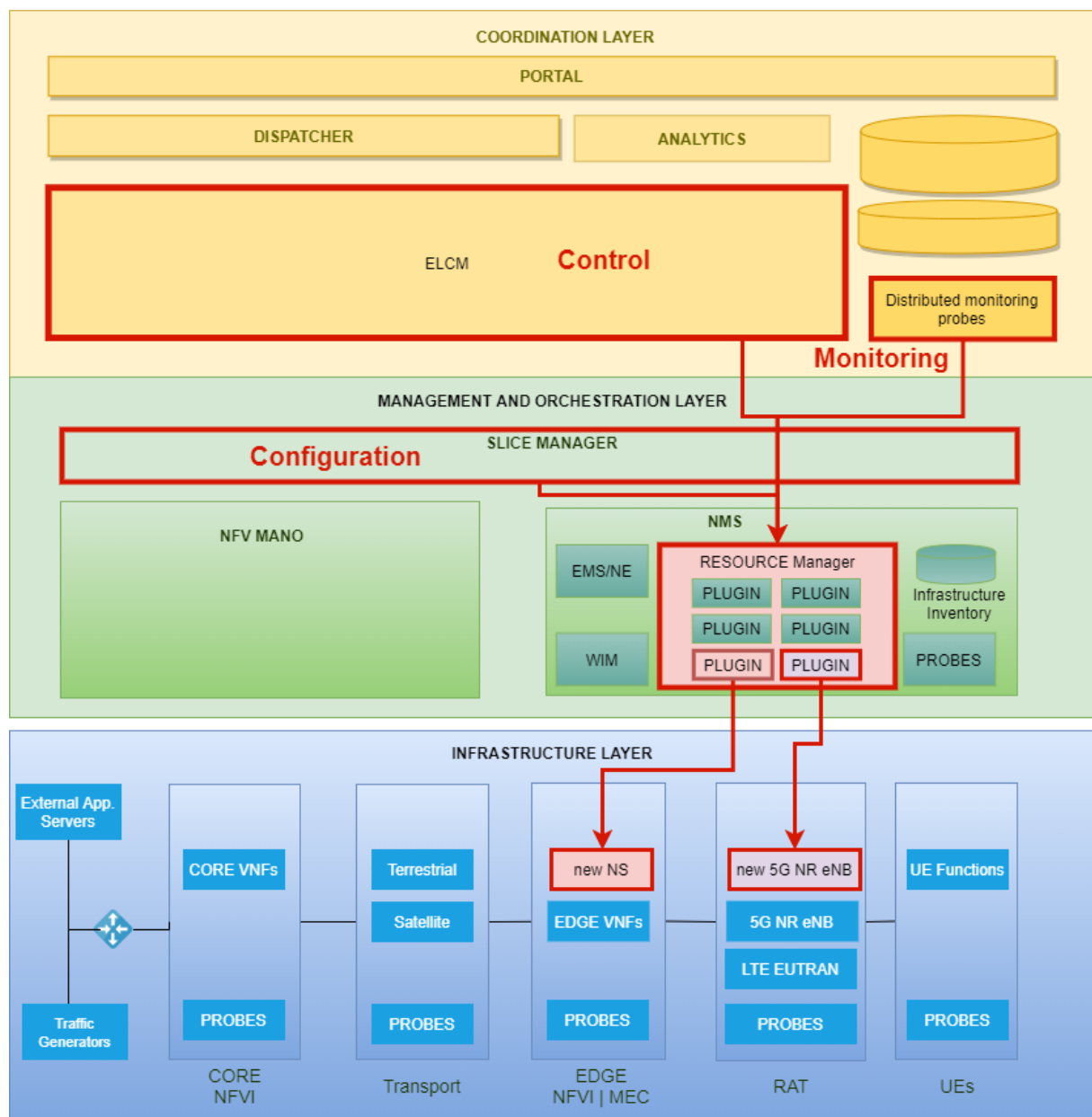


Figure 24 Integration of new components to the 5GENESIS Architecture

The ELCM is in charge of the experiment control and execution. Section 4.1.1 describes in detail the southbound interface offered by the ELCM. The Slice Manager monitors the slices and its configuration. The southbound APIs provided by the Slice Manager are described in Section 4.1.2. These sections also include examples of plugins developed by the consortium to integrate existing products such as Prometheus, an open source system for monitoring. The integration of the measurements exposed by the new products is presented in Section 5. Specifically, Section 5.3.1.2 describes the mechanisms (more concretely, results listeners, in the terminology used in 5GENESIS) available in the 5GENESIS Experimentation Framework that act as sinks for the measurement collected from all the probes.

## 4.1 Introduction to the development of 5GENESIS plugins

To ensure platform sustainability, it will be necessary either to extend the provided functionality or to incorporate new functionality with the addition of new components. These components need to be controlled using the available interfaces. The development of new plugins could be also necessary when the target of an experiment is to test a component. In this case, during the testing, it may also be necessary to control the component to reproduce real working conditions. For example, when testing a UE, we need to power on the device, attach it to the mobile network and, once the data connection is established, execute an application.

These new plugins are usually detected during the consultation phase prior to the execution of the experiment or the integration of the new components. During this phase, the platform administrators should give advice on which of the different methods of development described in the following sections is more suitable for the specific requirements, and how to perform the actual development.

### 4.1.1 Description of the southbound interface of ELCM

The ELCM is able to control elements of the testbed:

- Executing native ELCM Tasks: Tasks are the building blocks that can be used as part of an experiment execution. For example, the execution of external applications is a kind of Task. Several types of Tasks are available and, if necessary, new Tasks can be created and added to the ELCM.
- Delegating the execution of certain actions to external applications: This option is mainly used for starting the execution of OpenTAP test plans, but can also be used for executing external other applications that are available in the same environment. Using OpenTAP as a secondary orchestrator has several advantages: several TAP plugins have been developed in the context of the 5GENESIS project that are tailored to the requirements of the 5GENESIS platforms. For example, a correctly configured OpenTAP instance will automatically handle the collection of experiment results compatibly with the other elements of the 5GENESIS platforms.

#### 4.1.1.1 TAP plugins for the ELCM

The ELCM has no limitation or specific requirements for using TAP plugins. For this reason, the general guidelines, detailed in the official OpenTAP Developer Guide [21], apply when developing plugins are used in the 5GENESIS platforms.

Existing TAP plugins can be used in the definition of test plans that will be executed by the ELCM. However, there are recommendations about the format of results generated by a test step, that, when followed, they ease the integration of the plugin with the rest of the components in a 5GENESIS platform.

#### 4.1.1.1.1 Examples: TAP plugins example

We present an example of TAP plugins developed in the context of the 5GENESIS project: the Prometheus TAP plugin. This plugin includes basic functionalities for retrieving any kind of results from a running Prometheus [22] instance, and for publishing them in a format that can be automatically handled by the custom 5GENESIS results listener. This plugin makes use of the RestSharp library [23]. The source code of this plugin can be seen in Annex 2 – Prometheus TAP plugin source code.

##### 4.1.1.1.1.1 The Prometheus TAP Instrument

In TAP, Instruments are the entities that encapsulate the configuration settings and basic usage of an external entity. The Prometheus Instrument includes the connection settings (IP address and port where the instance is listening for connection), along with a method (*GetResults*) for retrieving results based on a configurable query. The Instrument also include three other methods that are mandatory for all Instruments: a constructor, where the default configuration values are set, and two methods, *Open* and *Close*, that are automatically executed at the start and at the end of a testplan, which configure and release the *RestClient* object that handles the connection with the Prometheus instance.

Instrument settings are defined as public properties of the Instrument class as shown in Figure 25. The *Display* decorator defines how these settings should be presented to the end user.

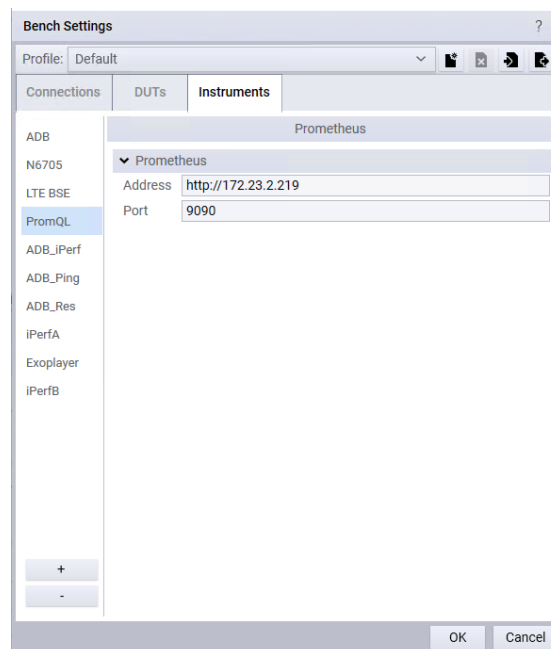


Figure 25 Prometheus instrument

```
[Display("Address", Group: "Prometheus", Order: 2.1,
Description: "Prometheus HTTP API address")]
public string Host { get; set; }

[Display("Port", Group: "Prometheus", Order: 2.2,
Description: "Prometheus HTTP API port")]
```

```
public int Port { get; set; }
```

The constructor of the Instrument is shown below. The default values of the Instrument are defined here, along with some rules that are used to verify that the user input is valid.

```
public PrometheusInstrument()
{
    Name = "PromQL";

    Host = String.Empty;
    Port = 80;

    Rules.Add(() => (!string.IsNullOrEmpty(Host)),
        "Please select an Address", "Host");
    Rules.Add(() => (Port > 0), "Please select a valid port number", "Port");
}
```

The *Open* and *Close* methods are defined below. In this case we create a *RestClient* object and save it on a private variable for use at any time during the test plan execution. When the test plan ends, we release this instance.

```
public override void Open()
{
    base.Open();
    this.client = new RestClient($"{Host}:{Port}/");
}

public override void Close()
{
    this.client = null;
    base.Close();
}
```

Finally, we define the *GetResults* method. This method makes use of the previously created *RestClient* object for communicating with the Prometheus instance.

```
public PrometheusReply GetResults(
    string query, DateTime start, DateTime end, double step)
{
    RestRequest request = new RestRequest("/api/v1/query_range",
    Method.GET, DataFormat.Json);

    request.AddParameter("query", query);
    request.AddParameter("start", start.ToString(TimeFormat));
    request.AddParameter("end", end.ToString(TimeFormat));
    request.AddParameter("step", $"{step.ToString(cultureInfo)}s");

    IRestResponse reply = client.Execute(request, Method.GET);

    PrometheusReply result = new PrometheusReply() {
        Status = reply.StatusCode,
        StatusDescription = reply.StatusDescription,
        Content = reply.Content
    };

    return result;
}
```

This method takes as parameters the query used for retrieving the results from Prometheus, along with the desired time range and resolution (in seconds) of the request. The method will return a *PrometheusReply* instance, which is the entity that encapsulates all the parsing logic required to obtain results from the raw JSON response sent by the Prometheus.



#### 4.1.1.1.1.2 The *PrometheusReply* class

The *PrometheusReply* class is the most complex part of the Prometheus plugin. However, this complexity is a consequence of the logic required for parsing the JSON object returned by Prometheus (in order to extract useful results from the raw data), and not from requirements imposed by TAP.

The response sent by Prometheus is a JSON object that follows the template presented below:

```
public HttpStatusCode Status { get; set; }

{
    "status": "success" | "error",

    // In case of error or warning
    "errorType": <string>,
    "error": <string>,
    "warnings": List<string>,

    // Returned measurements
    "data": {
        "resultType": "matrix" | "vector" | "scalar" | "string",
        // The format of "result" varies depending on "resultType", but
        // we are interested in "matrix" measurements
        "result": [
            {
                "metric": <Result metadata, including name>,
                "values": <List of pairs (timestamp, value)>
            },
            ...
        ]
    }
}
```

The *PrometheusReply* class extracts the information contained in the returned JSON and presents it in a format that is more convenient for further processing (in particular, for publishing as standard TAP results). Each property traverses a particular set of fields from the JSON, generating the required output.

```
public HttpStatusCode Status { get; set; }

public string StatusDescription { get; set; }

public string Content { get; set; }

public bool Success
{
    get { return ((int)Status >= 200) && ((int)Status <= 299); }
}
```

A *PrometheusReply* instance comprises several public properties, along with two private methods that handle the most complex processing. The *Status*, *StatusDescription* and *Success* properties are related to the outcome of the HTTP request, while *Content* stores the complete JSON reply (as a string) received from Prometheus. The following properties operate by extracting information from *Content*.

```
public string Message
{
    get
    {
        if (!string.IsNullOrEmpty(Content))
```

```

    {
        dynamic json = JsonConvert.DeserializeObject(Content);
        string status = json["status"].ToString();
        if (status == "error")
        {
            string errorType = json["errorType"];
            string message = json["error"];

            return $"Error: {errorType} - {message}";
        }
        else { return status; }
    }
    else { return "<Reply has no Content>"; }
}

```

The *Message* property tries to extract a descriptive status message from the reply. This message appears in the *status* field of the JSON object, however, if this field is equal to “error”, then additional information is extracted from the *errorType* and *error* fields.

```

public IEnumerable<ResultTable> Results
{
    get
    {
        if (Success)
        {
            dynamic json = JsonConvert.DeserializeObject(Content);
            dynamic data = json["data"];
            dynamic resultsList = data["result"];
            foreach (dynamic result in resultsList)
            {
                yield return getResultTable(result);
            }
        }
    }
}

```

*Results* returns all the available results contained in the received JSON response. Since a query may return multiple kinds of results, this property returns a collection of *ResultTable*. A *ResultTable* is a data structure used within TAP for sharing and handling results in TAP. It consists of an arbitrary number of columns (*ResultColumn*), each with a different *Name* and a sorted collection of values. These columns effectively create a table, where a row in position *n* is formed by the *n*<sup>th</sup> value in each column. The results generated by the Prometheus plugin contain a row of results for each timestamp in the returned time series.

Internally, the *Results* property delegates the creation of each independent *ResultTable* to a private method: *getResultTable*.

```

private ResultTable getResultTable(dynamic result)
{
    // Extract the available metadata from the "metric" dictionary
    Dictionary<string, string> metadata = new Dictionary<string, string>();
    foreach (var entry in result["metric"])
    {
        metadata[entry.Name] = entry.Value.ToString();
    }

    // Extract "values".
    List<double> timestamps = new List<double>();
    List<string> datetimes = new List<string>();
}

```

```

List<IConvertible> values = new List<IConvertible>();

foreach (var point in result["values"])
{
    double timestamp = double.Parse(point.First.ToString());
    DateTime datetime = DateTimeOffset.FromUnixTimeMilliseconds(
        (long) (timestamp * 1000)).DateTime;

    timestamps.Add(timestamp);
    datetimes.Add(datetime.ToString(PrometheusInstrument.TimeFormat));
    values.Add(this.toIConvertible(point.Last.ToString()));
}

// Create columns for UNIX timestamp, local datetime and value
string name = metadata.ContainsKey("__name__") ?
    metadata["__name__"] : "Prometheus result";

ResultColumn timestampColumn =
    new ResultColumn("Timestamp", timestamps.ToArray());

ResultColumn datetimesColumn =
    new ResultColumn("DateTime", datetimes.ToArray());

ResultColumn valuesColumn = new ResultColumn(name, values.ToArray());

// Create a column for each metadata value, repeated for every row
List<ResultColumn> resultColumns = new List<ResultColumn>();
foreach (var item in metadata)
{
    ResultColumn column = new ResultColumn(item.Key,
        Enumerable.Repeat(item.Value, timestamps.Count).ToArray());

    resultColumns.Add(column);
}

resultColumns.AddRange(new ResultColumn[] {
    timestampColumn, datetimesColumn, valuesColumn });

return new ResultTable(name, resultColumns.ToArray());
}

```

The *getResultTable* method iterates through all the points in the time series, where each point contains a timestamp and a value. The method generates three columns, one with the timestamp as a POSIX timestamp, which is the default format accepted by the 5GENESIS result listeners, another with the timestamp in a human readable format, and a third one with the value. Then, an additional column is created for every value returned by Prometheus as metadata.

Finally, *toIConvertible* is a helper method that tries to convert a given string to the most suitable data type possible.

```

private IConvertible toIConvertible(string value)
{
    if (long.TryParse(value, out long parsedLong))
    { return parsedLong; }

    if (double.TryParse(value, out double parsedDouble))
    { return parsedDouble; }

    if (bool.TryParse(value, out bool parsedBool))

```

```
{ return parsedBool; }

    return value;
}
```

#### 4.1.1.1.1.3 The Publish Prometheus results test step

The last component of the Prometheus TAP plugin is the *PublishStep* class, which defines the Prometheus test step. In general, a test step is composed by a collection of settings, which are defined as public properties, a constructor that defines the default values, a mandatory *Run* method, where the required actions are performed and, optionally, two methods (*PrePlanRun* and *PostPlanRun*) that are executed at the start and end of a testplan execution and may be used for initializing and releasing any necessary resources.

The settings and constructor of the *PublishStep* class are defined in a similar way as those of the Prometheus Instrument, and thus are omitted for simplicity. The complete code of this class is available as part of Annex 2 – Prometheus TAP plugin source code.

```
public override void Run()
{
    DateTime start = (PeriodMode == PeriodEnum.Absolute) ?
        Start : DateTime.UtcNow - RelativePeriod;
    DateTime end = (PeriodMode == PeriodEnum.Absolute) ?
        End : DateTime.UtcNow;

    PrometheusReply reply = Instrument.GetResults(Query, start, end, Step);

    if (reply.Success)
    {
        bool hasResults = false;

        foreach (ResultTable resultTable in reply.Results)
        {
            resultTable.PublishToSource(Results);

            long numResults = resultTable.Columns.First().Data.LongLength;
            Log.Info($"Published {numResults} results" +
                $" of type {resultTable.Name}");

            if (numResults > 0) { hasResults = true; }
        }

        if (!hasResults) { Log.Warning("No results have been retrieved."); }
    }
    else
    {
        Log.Error($"Request to Prometheus failed:" +
            $" {reply.StatusDescription} ({reply.Status})");
        Log.Error($" {reply.Message}");
        if (VerdictOnError.IsEnabled)
        {
            UpgradeVerdict(VerdictOnError.Value);
        }
    }
}
```

The *Run* method makes use of the Prometheus Instrument defined previously. The call to *GetResults* is customized by using the values selected by the user in the Prometheus test step settings, including the specific query to perform and the time period requested using absolute or relative timing.

in the Prometheus test step settings then checks the received reply. If there is an error the user is notified via the TAP application log, and, optionally, a verdict (that can be used for cancelling the execution of the remaining steps in a test plan) is generated.

If the request is successful, all *ResultTables* generated from the reply are published, so that they can be retrieved by all configured result listeners in the TAP instance. The Prometheus test step also generates a warning message for the user if the request is successful but without any result.

#### 4.1.1.2 Python extensions for the ELCM

To create new Tasks it is necessary to edit the source code of the ELCM. As the ELCM is designed to be extensible, it is only necessary to make changes to one of the existing files, while the code of the new Task resides in a new file that is saved in a specific folder (along with the rest of the Tasks). The basic steps for defining a new Task are:

- Create a new Python file with a descriptive name, for example 'compress\_files.py'. This file must be saved in the /Executor/Tasks/Run subfolder of the ELCM.
- In this file, define a new class, in our example 'CompressFiles', which inherits from 'Task'.
- The constructor of this class must have the signature displayed below:

```
class CompressFiles(Task):
def __init__(self, logMethod, parent, params):
    super().__init__("Compress Files", parent, params, logMethod, None)
```

- In general, the parameters received in the constructor will be sent directly to the superclass, along with a Task name (in the first parameter). The last parameter is an optional "Condition" method. If the callable passed in this parameter evaluates to False, the execution of the Task will be skipped.
- Override the Run method of the Task class. If this method is not overridden, a NotImplementedError will be raised at runtime. The following methods and fields are available:
  - self.name: Contains the name of the Task.
  - self.parent: Contains a reference to the Executor that called the task. Using this reference a Task can gain access to additional information about the Experiment
  - self.params: Contains a dictionary with the parameters of the Task. This dictionary will be filled by the ELCM using the information contained in the Platform Registry (for more information, please refer to Deliverable D3.16, section 3.4.2).
  - self.Log(level, message): Creates a new log message with the selected severity level. The ELCM will automatically route this message to the correct log file.
  - self.Publish(key, value): Saves a value under the key identifier. This value can be retrieved by Tasks that are executed later in the same experiment execution (via parameter expansion as part of their own self.params dictionary).
- In order to make the new Task available for use during an experiment execution, it is necessary to add a new *import* directive to /Executor/Tasks/Run/\_\_init\_\_.py. In our example:

```
from .compress_files import CompressFiles
```

After restarting the ELCM, the new Task will be available for use when defining new experiments. The Task identifier, in this example, is "Run.CompressFiles".

#### 4.1.1.2.1 Example: CompressFiles task

In the following we report the full source code of the CompressFiles Task, along with comments that explain its actions.

```
from Task import Task
from Helper import Level
from os.path import abspath

class CompressFiles(Task):
    def __init__(self, logMethod, parent, params):
        super().__init__("Compress Files", parent, params, logMethod, None)

    def Run(self):
        from Helper import Compress, IO
        # Compress and IO are utility classes for creating Zip
        # files and managing files and folders.

        files = [abspath(f) for f in self.params.get("Files", [])]
        folders = [abspath(f) for f in self.params.get("Folders", [])]
        output = self.params.get("Output", "")

        # First we extract all the necessary parameters from
        # the params dictionary. We always work with the
        # absolute path of the files and folders.
        # 'Output' is the name of the file to create, which
        # is always saved to the temporal folder of the
        # experiment execution

        self.Log(Level.INFO, f"Compressing files to output: {output}")

        for folder in folders:
            files.extend(IO.GetAllFiles(folder))

        # Recursively add all the files from the selected folders

        self.Log(Level.DEBUG, f"Files to compress: {files}")

        try:
            Compress.Zip(files, output)
            self.Log(Level.INFO, "File created")
        except Exception as e:
            self.Log(Level.ERROR, f"Exception while creating zip file: {e}")
```

### 4.1.2 Description of the southbound interface of the Slice Manager

The 5GENESIS Slice Manager is based on a highly modular architecture, built as a group of microservices, each running in a docker container. Figure 26 depicts an overview of the building blocks of the 5GENESIS Slice Manager.

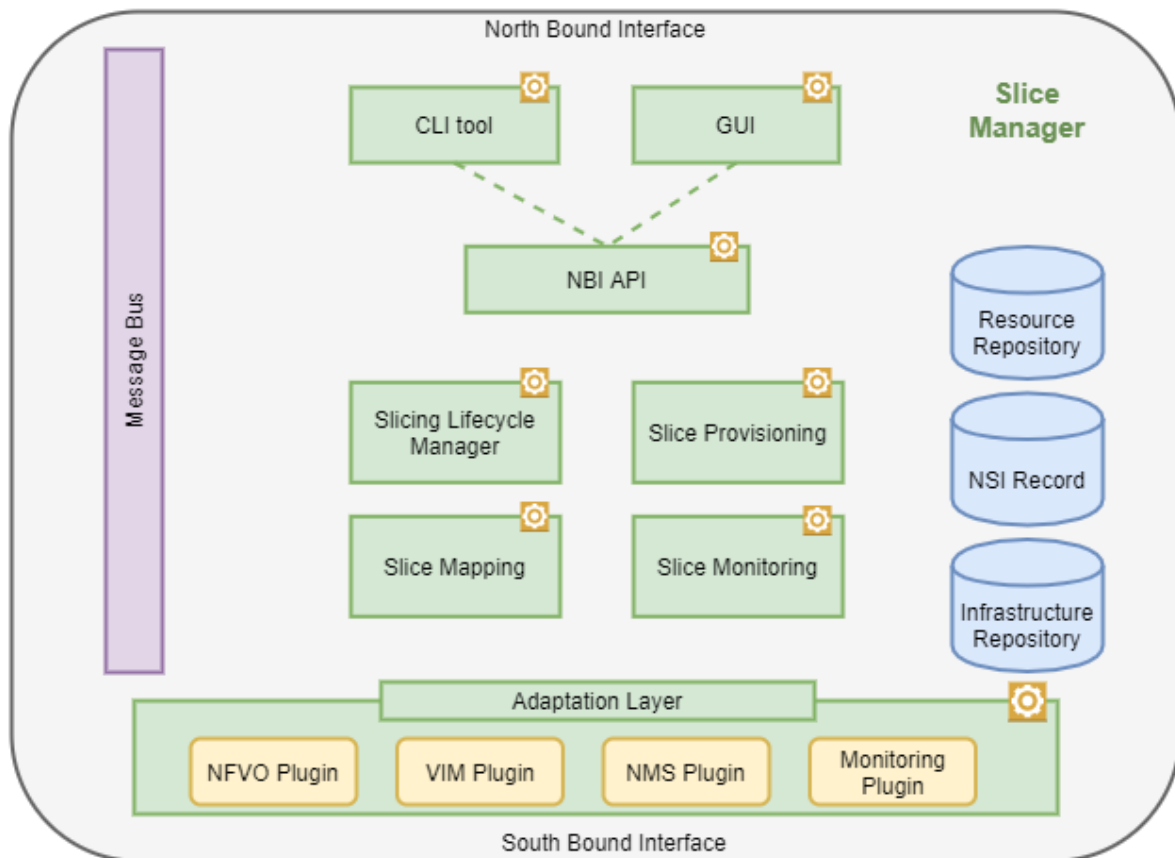


Figure 26 Slice Manager Building Blocks

The southbound components that interact with the Slice Manager are: VIMs, NFVOs and NMSs, which include the WIM, the Element Management System (EMS) and Monitoring Framework. The Adaptation Layer module provides a level of abstraction regarding the lower layer technology, making it feasible for the Slice Manager to operate over any MANO layer component without any modifications to its core functionality, as long as the proper plugin is loaded. The plugins translate the Slice Manager messages and actions that must be performed to type-specific messages and API calls for the south bound components. Table 11 presents the actions performed by the plugins for every southbound component.

Table 11 Operations between Slice Manager and southbound components

Component	Operation	Phase
VIM	Create a new Tenant	Slice Creation – Resource Provisioning
	Get information about VIM available resources	Slice Creation – Placement
	Delete a Tenant	Slice Termination
NFVO	Read NSDs and VNFDs	Slice Creation – Placement
	Add a new VIM account (VIM Tenant)	Slice Creation – Resource Provisioning

Component	Operation	Phase
	Instantiate a new NS	Slice Creation – Activation
	Read NS Records (NSRs) and VNF Records (VNFRs)	Slice Creation – Activation
	Delete an instantiated NS	Slice Termination
	Delete a VIM account (VIM Tenant)	Slice Termination
<b>WIM</b>	Create the transport network graph	Slice Creation – Resource Provisioning
	Activate the network traffic steering for a network slice	Slice Creation – Activation
	Delete the transport network graph	Slice Termination
<b>EMS</b>	Reserve RAN components	Slice Creation – Resource Provisioning
	Configure and start RAN services	Slice Creation – Activation
	Terminate RAN services	Slice Termination
	Release RAN components	Slice Termination
<b>MON</b>	Get information about Platform available resources	Slice Creation – Placement

Most of these actions can be performed with simple API calls to the endpoints that are available by the southbound components. During the slice creation phase the Slice Manager generates data that will be consumed by the EMS and the WIM in order to instantiate and configure the parts of the slice for which they are responsible. These data is in the form of JSON files, which are defined with the use of JSON Schemas. The JSON Schema is a vocabulary that allows to annotate and validate JSON documents. These JSON Schemas are presented in Annex 4 – JSON schema of the Slice Manager Southbound messages. Each plugin must be able to read and translate the data to component-specific messages.

Finally, the plugins must be able to handle the registration of a new southbound component to the Slice Manager. The registration of new components is realized with the use of JSON configuration files, the JSON Schemas of which are presented in Annex 5 – Southbound components configuration files schemas. The plugin must be able to create a new component object based on these files and store it in the Infrastructure repository.

#### 4.1.2.1 Python plugins for the slice manager

Python modules can be integrated directly within the Slice Manager service stack in order to act as wrapper plugins for any underlying components, in order for them to communicate with



the Slice Manager Southbound Interface (SBI). These modules must introduce methods and functions that will enable the Slice Manager to support the following features:

- Communicate with the underlying component through any available APIs (e.g. REST, ssh, message bus, etc.), and perform the actions that are part of the slice management procedures.
- Receive the messages that the Slice Manager produces for the underlying components and translate them to component-specific messages.
- Create objects for every new component based on the configuration files.

There are two available options for the communication between the python plugins and the slice lifecycle manager: (i) Use direct calls to functions and methods defined and imported by the plugin modules and (ii) create producers and consumer objects that will use the Kafka message bus which is part of the Slice Manager software stack. In both cases, the python module must be saved in the directory “katana-slice\_manager/shared\_utils/” with a descriptive name following the “<component>Utils.py” format, for example openstackUtils.py, as depicted in Figure 27.

The plugins define python classes with methods and functions that implement the enlisted features. These classes are imported in the NBI module, as presented below.

```
from katana.shared_utils.osmUtils import osmUtils
from katana.shared_utils.tango5gUtils import tango5gUtils
from katana.shared_utils.mongoUtils import mongoUtils
```

The NBI module creates a new python object from the imported class for every new southbound component that is registered to the Slice Manager and stores it in the Infrastructure repository. Any time that the plugin must be used for the slice management procedures, the object will be restored from the repository and used by Slicing Lifecycle Manager (SLM), which calls its methods in order to perform the necessary actions.

If the plugin uses the Kafka message bus for the communication with the other Slice Manager modules, then it should also create the Kafka producers and consumers, as well as the Kafka topics that will be used for the message exchange procedures. These objects are created using the kafkaUtils module, as shown in the example below.

- 1) Create Kafka topic and Kafka consumer that listens for new messages:

```
# Create Kafka topic
kafkaUtils.create_topic()

# Create the Kafka Consumer
consumer = kafkaUtils.create_consumer()

# Check for new messages
for message in consumer:
    logger.info("--- New Message ---")
    logger.info(
        "Topic: {0} | Partition: {1} | Offset: {2}".format(
            message.topic, message.partition, message.offset
        )
    )
    # Commit the latest received message
    consumer.commit()
    action = message.value["action"]
    payload = message.value["message"]
```

## 2) Create Kafka Producer and use it to send messages:

```
# Send the message to katana-mngr
producer = kafkaUtils.create_producer()
slice_message = {"action": "add", "message": nest}
producer.send("slice", value=slice_message)
```

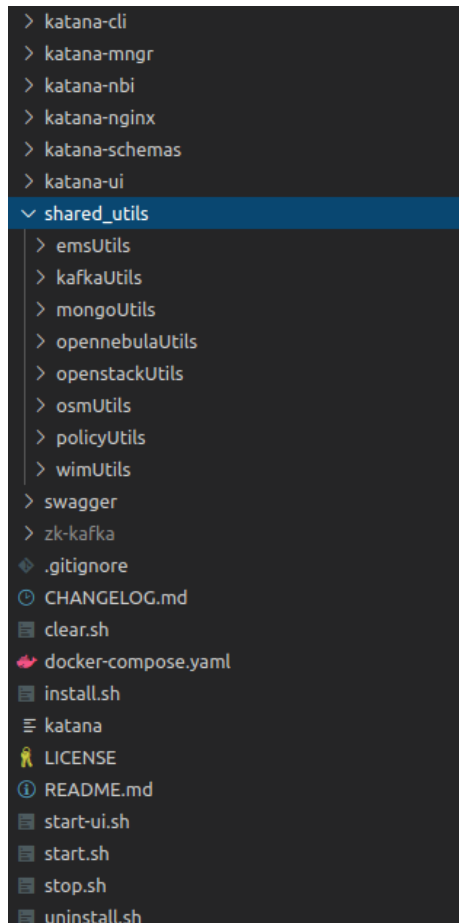


Figure 27 Slice Manager Source Code Directory

### 4.1.2.1.1 Example: Amarisoft plugin

This section describes the implementation and integration of the Amarisoft plugin, a python module that is used by the Slice Manager in order to communicate with the Amarisoft EMS. The Amarisoft EMS is responsible for the configuration and management of the Amarisoft radio components.

This plugin is implemented as a python module that creates a class that contains all the necessary methods that enable the communication with the Amarisoft EMS. The module is stored at the directory “katana-slice\_manager/sharedUtils/ems/Utils/” and it is named “amar\_emsUtils.py”. The class is imported to the NBI module. Every time a new Amarisoft EMS is registered to the Slice Manager, a new object is created and stored in the infrastructure repository, as presented below.

```
from katana.shared_utils.emsUtils import amar_emsUtils, test_emsUtils

[...]
def post(self):
```

```

"""
Add a new EMS. The request must provide the ems details.
used by: `katana ems add -f [yaml file]`
"""

new_uuid = str(uuid.uuid4())
# Create the object and store it in the object collection
try:
    ems_id = request.json["id"]
    if request.json["type"] == "amarisoft-ems":
        ems = amar_emsUtils.Ems(request.json['url'])
    elif request.json["type"] == "test-ems":
        ems = test_emsUtils.Ems(request.json['url'])
    else:
        return "Error: Not supported EMS type", 400
except KeyError:
    return f"Error: Required fields: {self.req_fields}", 400
thebytes = pickle.dumps(ems)
obj_json = {"_id": new_uuid, "id": request.json["id"],
            "obj": Binary(thebytes)}
request.json['_id'] = new_uuid
request.json['created_at'] = time.time() # unix epoch
try:
    new_uuid = mongoUtils.add('ems', request.json)
except pymongo.errors.DuplicateKeyError:
    return f"EMS with id {ems_id} already exists", 400
mongoUtils.add('ems_obj', obj_json)
return f"Created {new_uuid}", 201

```

[...]

The object is restored from the SLM module whenever necessary during the slice management procedures. The complete source code of the Amarisoft plugin, where the class and its methods are defined, is presented below.

```

import requests
import json
import logging

# Logging Parameters
logger = logging.getLogger(__name__)
file_handler = logging.handlers.RotatingFileHandler(
    'katana.log', maxBytes=10000, backupCount=5)
stream_handler = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s %(name)s %(levelname)s'
    '%(message)s')
stream_formatter = logging.Formatter(
    '%(asctime)s %(name)s %(levelname)s %(message)s')
file_handler.setFormatter(formatter)
stream_handler.setFormatter(stream_formatter)
logger.setLevel(logging.DEBUG)
logger.addHandler(file_handler)
logger.addHandler(stream_handler)

class Ems():
    """
    Class implementing the communication API with EMS
    """

    def __init__(self, url):
        """

```

```
Initialize an object of the class
"""
self.url = url

def conf_radio(self, slice_data):
    """
    Configure radio components for the newly created slice
    """
    ems_url = self.url
    api_prefix = '/deploy'
    url = ems_url + api_prefix
    headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json'
    }
    data = slice_data
    r = None
    try:
        r = requests.post(url, json=json.loads(json.dumps(data)),
                          timeout=360, headers=headers)
        logger.info(r.json())
        r.raise_for_status()
    except requests.exceptions.HTTPError as errh:
        logger.exception("Http Error:", errh)
    except requests.exceptions.ConnectionError as errc:
        logger.exception("Error Connecting:", errc)
    except requests.exceptions.Timeout as errt:
        logger.exception("Timeout Error:", errt)
    except requests.exceptions.RequestException as err:
        logger.exception("Error:", err)

def del_slice(self, slice_data):
    """
    Delete a configured radio slice
    """
    logger.info("Deleting Radio Slice Configuration")
    ems_url = self.url
    api_prefix = '/deploy'
    url = ems_url + api_prefix
    headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json'
    }
    data = slice_data
    r = None
    try:
        r = requests.post(url, json=json.loads(json.dumps(data)),
                          timeout=360, headers=headers)
        logger.info(r.json())
        r.raise_for_status()
    except requests.exceptions.HTTPError as errh:
        logger.exception("Http Error:", errh)
    except requests.exceptions.ConnectionError as errc:
        logger.exception("Error Connecting:", errc)
    except requests.exceptions.Timeout as errt:
        logger.exception("Timeout Error:", errt)
    except requests.exceptions.RequestException as err:
        logger.exception("Error:", err)
```

## 5 DOCUMENTATION FOR PLATFORM OPERATORS

---

This section includes the manuals to deploy the components of the 5GENESIS Experimentation Framework developed as part of Release B. These components are described in deliverables D3.2, D3.4, D3.6, D3.8, D3.10 and D3.12.

### 5.1 Coordination Layer deployment

#### 5.1.1 Portal

The requirements for the development and installation of the 5GENESIS Portal include the following:

- Compatible with Windows or Linux.
- Python 3.7.x [24].
- ELCM.
- Dispatcher
- [Optional] Grafana [25] (tested on version 5.4).
- [Optional] Analytics Dashboard

##### 5.1.1.1 Deployment

###### 5.1.1.1.1 Pre-requisites

The Portal requires connectivity with running instances of the Dispatcher (user authentication and NS onboarding) and ELCM (platform registry and experiment execution).

Additional dependencies may be needed depending on your environment. For example, older Windows version may require certain Visual C++ redistributables to be installed, and the following packages are known to be required on many Ubuntu distributions: *gcc*, *python3.7*, *python3.7-venv*, *python3.7-dev*. Fixes for specific issues are usually easy to find on Internet.

###### 5.1.1.1.2 Installation procedure

This repository includes two sets of scripts for use on Linux (*.sh*) and Windows (*.ps1*) machines. In general, these scripts should be able to perform most of the actions required for instantiating the Portal, however, depending on the deployment environment some actions may fail or require additional tweaking. The contents of the scripts can be used as a guide for manual installation, and a description of the actions performed by the scripts is included below for use as reference.

1. Ensure that Python 3.7.x is installed. For environments with multiple Python versions please note the correct alias. For example, older Ubuntu distributions refer to Python 2.x by default when invoking *python*, and reference Python 3.7 as *python3* or *python3.7*. Use the *--version* parameter to check the version number.
2. Clone [the repository](#) to a known folder.
3. Run *install.sh* <python\_alias> or *install.ps1* <python\_alias> (depending on your OS). The script will:
  - a. Display the Python version in use (ensure that this is 3.7.x)
  - b. Create a Python virtual environment [27] for exclusive use of the Portal.

- c. Install the required Python packages (using pip). Most issues occur during this step, since it is highly dependent on the environment. In case of error, note the name of the package that could not be installed, the error message and your OS distribution. Performing an Internet search with this information usually yields a solution. Once solved you may re-run the script (delete the *venv* folder that was created by the script if necessary) until all packages are correctly installed.
  - d. Initialize the Portal database
4. Run *start.sh* or *start.ps1* (depending on your OS). This will create an empty configuration file (*config.yml*). If necessary, press ctrl+c (or your OS equivalent) in order to close the server.
5. Ensure that the *config.yml* is available in the Portal folder and customize its contents. The Portal needs information about how to connect with the Dispatcher and ELCM components (more information about all the possible configuration values can be found below).
6. Customize the *.flaskenv* file. Replace the `__REPLACEWITHSECRETKEY__` label with a random string

#### 5.1.1.1.3 Starting the Portal

Once configured, the Portal can be started by running *start.sh* *<port\_number>* or *start.ps1* *<port\_number>*. If not specified, the server will listen on port 5000. In order to stop the server, press ctrl+c (or your OS equivalent) in the terminal where the server is running.

#### 5.1.1.1.4 Minimal integration tests

In order to test that the connections with the Dispatcher and ELCM are working properly, perform the following actions:

1. Check the log messages (in the log file or console output) that appear when starting the Portal. The Portal tries to retrieve the facility configuration from the ELCM when starting, and displays the number of registered test cases, UEs, scenarios and slice descriptors. If these correspond to the ones configured in the ELCM, then the connection is working properly. If you do not see any messages, check the Logging section of the configuration file (*config.yml*). Ensure that the levels are set to *DEBUG* or *INFO*
2. Open the Portal using a web browser.
3. Register a new user (top right, *Register* tab). If no errors are reported after pressing the *Register* button at the bottom then the connection with the Dispatcher is working properly. Note that newly registered users are not "active", and cannot log in to the Portal until their registration has been validated by the platform administrator(s). For information about the user activation procedure, refer to the Dispatcher (Authenticator) documentation.

#### 5.1.1.2 Configuration

The Portal instance can be configured by editing the *config.yml* file:

- Logging:
  - **Folder:** Folder where the log files will be saved. Defaults to *./Logs*.
  - **AppLevel:** Minimum message level to display in the terminal output (one of *CRITICAL*, *ERROR*, *WARNING*, *INFO*, *DEBUG*). Defaults to *INFO*.
  - **LogLevel:** Minimum message level to write in the log files. Defaults to *DEBUG*.

- **Dispatcher:**
  - **Host:** Location of the machine where the Dispatcher is running (localhost by default).
  - **Port:** Port where the Dispatcher is listening for connections (5001 by default).
  - **TokenExpiry:** Time (in seconds) to consider that an authentication token has expired, should be slightly shorter (30/60 seconds) than the real expiration time configured on the Dispatcher.
- **ELCM:**
  - **Host:** Location of the machine where the ELCM is running (localhost by default).
  - **Port:** Port where the ELCM is listening for connections (5001 by default).
- **Grafana URL:** Base URL of Grafana Dashboard to display Execution results.
- **Platform:** Platform name. Will be displayed in the Portal and will identify the platform during distributed experiments.
- **Description:** Short textual description of the platform. Will be displayed in the Portal.
- **PlatformDescriptionPage:** HTML file that contains a more detailed description of the platform. The HTML written in this file will be inserted in the *Info* page of the Portal. Defaults to *platform.html*. This file is included in this repository, and can be customized or used as reference.
- **EastWest:** Configuration for distributed experiments.
  - **Enabled:** Boolean value indicating if the East/West interfaces are available. Defaults to False.
  - **Remotes:** Dictionary containing the connection configuration for each remote platform's Portal, with each key containing **Host** and **Port** values in the same format as in the **ELCM** section. Defaults to an empty dictionary (`{}``).
- **Analytics:**
  - **Enabled:** Boolean value indicating if the Analytics Dashboard is available. Defaults to False.
  - **URL:** External URL of the Analytics Dashboard
  - **Secret:** Secret key shared with the Analytics Dashboard, used in order to create secure URLs

Portal notices:

It is possible to display system-wide notices in the Portal by creating a file named *notices.yml* in the root folder of the Portal. The format of this file shall be as follows:

```
Notices:
- Example notice 1
- Example notice 2
```

The list may include as many notices as necessary.

### 5.1.2 Dispatcher

The Dispatcher component is hosted in the 5Genesis GitHub repository ([Release B Repository](#)). All the instructions as well as the examples can be found, as well as identified issues that might help during the installation and set-up of the environment. A more detailed information on the flow and designed components can be found in **D3.2 Management and Orchestration (Release B)** [5] and **D3.8 Open APIs, service level functions and interfaces for verticals (Release B)** [19].

### 5.1.2.1 Installation

The instructions for the installation process, will download a copy of the Dispatcher and MANO component, up and running on a local machine for development and testing purposes after the clone of the repository.

Before installing the component, some pre-requisites are needed for running the 5Genesis Dispatcher, the requirements are the following:

- docker version >= 18.09.6
- docker-compose version >= 1.17.1
- Configuration files correctly filled up:
- MANO Wrapper module config: configuration file (mano.conf) inside the mano folder.
- NFVO + VIM

### 5.1.2.2 Configuration

The Dispatcher needs to be configured properly before the containers are built. For that, a simplified configuration file is offered: dispatcher.conf, which will have to be edited and adapted. The file should contain information of all the modules the Dispatcher forwards information to (validator, mano, elcm, result\_catalog, etc.) and how to do it. For each module, a new enabler will be added in the Dispatcher. It uses the following format:

[module\_name]

PROTOCOL=[http|https]

HOST=x.x.x.x -> IP or DNS name of the host component

PORT=xxxx -> Port where the app API is available

PATH=/ -> Base path of the application ("/" by default)

The config file template already includes the validator and the mano modules as they are included within the Dispatcher. They are already configured and should not be touched.

During the installation process you will be asked for a couple of simple questions: One of them is whether you want the Authentication module installed as well or not, so it is completely optional.

Once edited properly, the configuration will be applied and the containers built, based on the config file we have just created (dispatcher.conf):

```
$ ./install.sh
```

### 5.1.2.3 Deployment

The start script will deploy and run the Dispatcher container, the Validator, the MANO Wrapper and a Swagger environment to test the available features:

```
$ ./start.sh
```

Dispatcher will be accessible through port 8082 through a SSL certificate, so HTTPS is required.

Swagger environment will be accessible through port 5002.

Installation of the SSL certificate



The SSL certificate is self-signed. It is not supervised by a Certification Authority because in this current moment is not possibly know where the platforms will be hosted.

For installing the certificate is required to open the internet navigator in `https://<IP_OF_DISPATCHER>:8082`

Something like this will appear, the web page format depends of your browser: Installation of the certificate

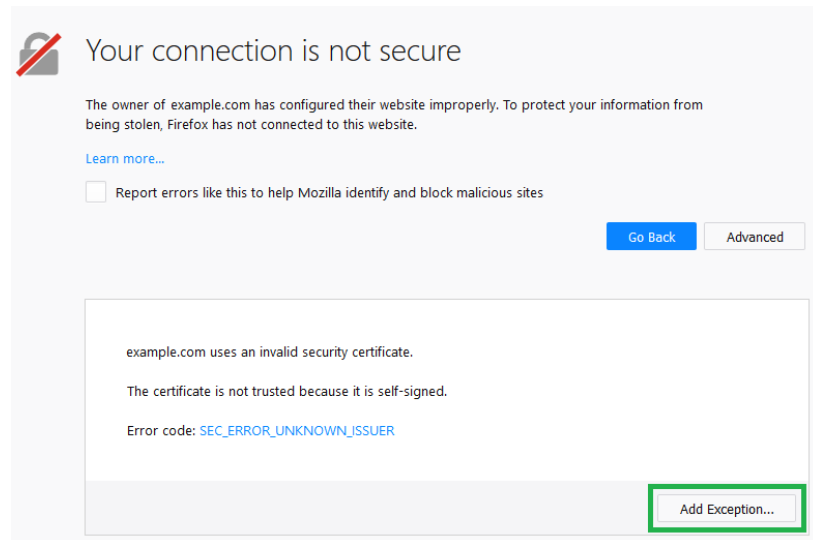


Figure 28 Certificate exception

It is required to add the exception/believe in the certificate, as shown in the figure. After this step, it is able to use the dispatcher through SSL certification.

To stop the Dispatcher service just run the following:

```
$ ./stop.sh
```

#### 5.1.2.4 Testing

The testing environment needs to be configured before starting the container. The environment, in the config folder, file needs to be edited to adapt it to the local environment:

# Test variables

TEST\_EMAIL=<user e-mail address>

API\_URL=<Dispatcher API URL. Example: https://192.168.33.116:8082>

VIM\_NAME=<VIM name. Example: malagacore>

This file is created on installation time based on the information provided by the user

The previous set of tests can be executed by running a script:

```
$ ./runtests.sh
```

This test starts the web server to show the reports after the testing and executes all the tests. The report is accessible via web browser on port 8200. The figure shows the Test Log after the execution.

## Dispatcher Test Log

### REPORT

Generated  
20200716 14:23:06 UTC+01:00  
17 hours 5 minutes ago

### Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	21	21	0	00:02:35	<div></div>
All Tests	21	21	0	00:02:35	<div></div>

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					<div></div>

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
<a href="#">Dispatcher Test</a>	21	21	0	00:02:38	<div></div>

### Test Execution Log

<b>SUITE</b> Dispatcher Test	00:02:38.087
Full Name:	Dispatcher Test
Source:	<a href="#">/robotTesting/testsuite/dispatcher_test.robot</a>
Start / End / Elapsed:	20200716 14:20:26.710 / 20200716 14:23:04.797 / 00:02:38.087
Status:	21 critical test, 21 passed, 0 failed 21 test total, 21 passed, 0 failed

<b>TEST</b> Register New User	00:00:02.483
<b>TEST</b> Validate User	00:00:02.467
<b>TEST</b> Show Users (Admin Basic Auth)	00:00:00.468
<b>TEST</b> Get User Token (User Basic Auth)	00:00:05.626
<b>TEST</b> List VIMs (Token Auth)	00:00:05.626
<b>TEST</b> Upload Image VIM (Token Auth)	00:00:54.629
<b>TEST</b> Register VIM Image (Admin Basic Auth)	00:00:05.520
<b>TEST</b> Get Image List (Token Auth)	00:00:05.790
<b>TEST</b> Index Faulty VNFD (Token Auth)	00:00:07.406
<b>TEST</b> Index VNFD (Token Auth)	00:00:07.125
<b>TEST</b> Get VNFD list (Token Auth)	00:00:05.839
<b>TEST</b> Get VNFD list (Basic Auth)	00:00:05.538

Figure 29 Dispatcher Execution Test log

### 5.1.3 ELCM

The requirements for the development and installation of the ELCM include the following:

- Compatible with Windows or Linux.
- Python 3.7.x [24].
- Optional Integrations:
  - 5Genesis Portal (Version 2.4.0 or later)
  - Dispatcher (Version 2.8.2 or later)
  - Katana slice Manager (Version 2.2.6 or later)
  - Grafana (tested with version 5.4).
  - InfluxDB (tested with version 1.7.6 )
  - OpenTAP (tested with version 9.9 (Windows))

### 5.1.3.1 Deployment

#### 5.1.3.1.1 Pre-requisites

Additional dependencies may be needed depending on your environment. For example, older Windows version may require certain Visual C++ redistributables to be installed, and the following packages are known to be required on many Ubuntu distributions: *gcc*, *python3.7*, *python3.7-venv*, *python3.7-dev*. Fixes for specific issues are usually easy to find on Internet.

#### 5.1.3.1.2 Installation procedure

This repository includes two sets of scripts for use on Linux (*.sh*) and Windows (*.ps1*) machines. In general, these scripts should be able to perform most of the actions required for instantiating the ELCM, however, depending on the deployment environment some actions may fail or require additional tweaking. The contents of the scripts can be used as a guide for manual installation, and a description of the actions performed by the scripts is included below for use as reference.

7. Ensure that Python 3.7.x is installed. For environments with multiple Python versions please note the correct alias. For example, older Ubuntu distributions refer to Python 2.x by default when invoking *python*, and reference Python 3.7 as *python3* or *python3.7*. Use the *--version* parameter to check the version number.
8. Clone [the repository](#) to a known folder.
9. Run *install.sh* <python\_alias> or *install.ps1* <python\_alias> (depending on your OS). The script will:
  - a. Display the Python version in use (ensure that this is 3.7.x)
  - b. Create a Python virtual environment [27] for exclusive use of the Portal.
  - c. Install the required Python packages (using pip). Most issues occur during this step, since it is highly dependent on the environment. In case of error, note the name of the package that could not be installed, the error message and your OS distribution. Performing an Internet search with this information usually yields a solution. Once solved you may re-run the script (delete the *venv* folder that was created by the script if necessary) until all packages are correctly installed.
10. Run *start.sh* or *start.ps1* (depending on your OS). This will create an empty configuration file (*config.yml*). If necessary, press ctrl+c (or your OS equivalent) in order to close the server.
11. Ensure that the *config.yml* is available in the ELCM folder and customize its contents. Information about all the possible configuration values can be found below.

#### 5.1.3.1.3 Starting the Portal

Once configured, the ELCM can be started by running *start.sh* <port\_number> or *start.ps1* <port\_number>. If not specified, the server will listen on port 5001. In order to stop the server, press ctrl+c (or your OS equivalent) in the terminal where the server is running.

Before using the scripts for starting a production ELCM instance consider changing the ``<YOUR SECRET KEY HERE>`` value to a random string. This is particularly important if the ELCM port is exposed to the Internet (in this case also consider using ``waitress``, as can be seen in the Portal scripts).

Please note that it is not recommended exposing the ELCM to the open Internet, regardless of these tips.

### 5.1.3.2 Configuration

The ELCM instance can be configured by editing the *config.yml* file:

- **TempFolder:** Root folder where the temporal files for the Executors can be created.
- **ResultsFolder:** Root folder where the files generated by each experiment execution will be saved.
- **Logging:**
  - **Folder:** Folder where the log files will be saved. Defaults to *./Logs*.
  - **AppLevel:** Minimum message level to display in the terminal output (one of *CRITICAL, ERROR, WARNING, INFO, DEBUG*). Defaults to *INFO*.
  - **LogLevel:** Minimum message level to write in the log files. Defaults to *DEBUG*.
- **Portal:**
  - **Host:** Location of the machine where the Portal is running (localhost by default).
  - **Port:** Port where the Portal is listening for connections (5000 by default).
- **SliceManager:**
  - **Host:** Location of the machine where the Katana Slice Manager is running
  - **Port:** Port where the Slice Manager is listening
- **Tap:**
  - **Enabled:** Whether to use TAP or not, if set to *False* the settings below will be ignored
  - **OpenTap:** True if using OpenTap (TAP 9.0 or later), False if using TAP 8 (legacy option)
  - **Exe:** TAP CLI executable name
  - **Folder:** TAP installation folder
  - **Results:** TAP results folder
  - **EnsureClosed:** Performs an additional check on test plan completion, to ensure that all child processes are correctly closed. Defaults to *True*.
  - **EnsureAdbClosed:** Forcibly close any adb process at the end of the execution. This will close adb instances started by any other means, and may cause issues with other running experiment executions. Defaults to *False*, set to *True* only if the TAP executions hang at the end frequently due to adb not closing.
- **Grafana:**
  - **Enabled:** If set to *False* the settings below will be ignored
  - **Host:** Location of the machine where the Grafana instance is running
  - **Port:** Port where the Grafana API is listening
  - **Bearer:** Grafana API key without the 'Bearer ' prefix
  - **ReportGenerator:** URL where the 'Grafana reporter' instance can be reached, if any
- **InfluxDb:**
  - **Enabled:** If set to *False* the settings below will be ignored
  - **Host:** Location of the machine where the InfluxDb instance is running
  - **Port:** Port where InfluxDB is listening
  - **User:** InfluxDb instance user
  - **Password:** InfluxDb user password
  - **Database:** InfluxDb instance database
- **Metadata:**

- **HostIp:** IP address of the machine where the ELCM is running
- **Facility:** Facility name (or platform)
- **EastWest:** Configuration for distributed experiments
  - **Enabled:** Boolean value indicating if the East/West interfaces are available. Defaults to *False*.
  - **Timeout:** Timeout for any communication with the remote side of the experiment execution. Defaults to 120 seconds.
  - **Remotes:** Dictionary containing the connection configuration for each remote platform's ELCM, with each key containing *Host* and *Port* values in the same format as in the *Portal* or *SliceManager* sections. Defaults to two (invalid) example entries.

#### 5.1.3.2.1 Facility Configuration (Platform registry)

The exposed capabilities and functionality of the facility are defined by a set of files distributed in 4 folders inside the root of the ELCM instance. These are:

- **TestCases:** Contains information about the available test cases that can be run by the facility.
- **UEs:** Contains specific actions required for using and releasing specific equipment of the facility during the execution of test cases.
- **Resources:** Contains the definition of certain equipment that can only be used by one experiment at a time.
- **Scenarios:** Additional configuration values that can be set during the deployment of a network slice.

##### 5.1.3.2.1.1 TestCases and UEs

The contents of the *UEs* and *TestCases* sub-folder describe the behavior of the 5Genesis Platform when an Experiment execution request is received. These folders will be automatically generated (empty) if they do not exist. The ELCM will load the contents of every *yaml* file contained in these folders on startup and whenever the 'Reload facility' button on the web dashboard is pressed. The dashboard will also display a validation log ('Facility log') which can be used in order to detect errors on a TestCase or UE configuration.

#### UEs:

The files on the *UEs* folder describe the actions to perform when a certain UE is included in the 'Experiment descriptor' received as part of the request (for example, initializing or configuring the UE). The 'Composer' will add the actions defined for every UE to the Tasks list. The following is an example of a *yaml* file that configures an UE:

```
TestUE:
- Order: 1
  Task: Run.Dummy
  Requirements: [UE1]
  Config:
    Message: This is a dummy entity initialization
- Order: 10
  Task: Run.Dummy
  Config:
    Message: This is a dummy entity closure
```

The name of the UE will be extracted from the initial key on the dictionary (not the name of the file). This key contains a list of every action to perform, described by the relative *Order* in which

to run, the *Task* to perform (which correspond to the different Tasks defined in the *Executor.Tasks* package) and *Config* dictionary, which is different for every task and optionally a list of *Requirements*. These requirements corresponds to the resources defined for the facility. (See "Facility resources" below). Additional information about the available tasks can be seen in the *Available Tasks* section below.

### TestCases:

Similarly to the UEs, the files in the *TestCases* folder define the actions required in order to execute a certain test case. The following is an example TestCase file:

```
Slice Creation:
- Order: 5
  Task: Run.SingleSliceCreationTime
  Config:
    ExperimentId: "@{ExperimentId}"
    WaitForRunning: True
    Timeout: 60
    SliceId: "@{SliceId}"
Standard: True
Distributed: False
Dashboard: {}
```

### Standard and Custom experiment - TestCase parameters:

In order to control how each TestCase is handled by the 5GENESIS Portal and when using the Open APIs, several keys can be added to the *yaml* description. These keys are:

- **Standard:** Boolean. Indicates whether the TestCase is selectable from the list of Standard test cases. If not specified, this value defaults to *False* if the *Custom* key is defined, *True* otherwise.
- **Custom:** List of strings. Indicates that the TestCase is a Custom test case and may accept parameters. If this value is set to an empty list ([]) the test case is considered public and will appear on the list of Custom experiments for all users of the Portal. If the list contains one or more email addresses, the test case will be visible only to the users with matching emails.
- **Parameters:** Dictionary of dictionaries, where each entry is defined as follows:

```
"<Parameter Name>":
  Type: "String, used to guide the user as to what is the expected format"
  Description: "String, textual description of the parameter"
```

Parameters can be used to customize the execution of test cases. For example, a Test Case may be implemented using a TAP test plan, that accepts an external parameter called 'Interval'. Using variable expansion the value of this external parameter can be linked with the value of an 'Interval' (or a different name) parameter contained in the experiment descriptor.

It is also possible to define default values during variable expansion, which means that a Test Case can be defined as 'Standard', where it will use the default values for all parameters, and 'Custom', where some values can be replaced by the experimenter.

### TestCase dashboards:

If a Grafana instance is available and configured, the ELCM can automatically create a Dashboard for displaying some of the most important raw results generated during an experiment execution. In order to use this functionality, the test case definition must include a

collection of Grafana panel definitions. For each experiment execution, the panels defined by all of the test cases selected will be aggregated in a single dashboard. An example of dashboard definition with a single panel can be seen below.

```
Dashboard:
- Name: "Slice Deployment Time"
  Measurement: Slice_Creation_Time
  Field: Slice_Deployment_Time
  Unit: "s"
  Type: Singlestat
  Percentage: False
  Size: [8, 8]
  Position: [0, 0]
  Gauge: True
  Color: ["#299c46", "rgba(237, 129, 40, 0.89)", "#d44a3a"]
  Thresholds: [0, 15, 25, 30]
```

It is possible to integrate an instance of Grafana reporter [26] in order to generate PDF reports from the Grafana dashboards of the experiments. This feature will appear as a button on the top-right of the dashboard.

For using this feature in the ELCM you only need to specify the URL where `Grafana reporter` is reachable. Please refer to the reporter documentation for the configuration of the reporter itself.

#### Dashboard auto-generation:

The ELCM is able to generate additional panels if certain values appear on the names of the generated TAP results. For this feature to work an additional result listener (AutoGraph) must be enabled in TAP. This result listener does not require any configuration and, once enabled, the auto-generation of panels will work for any subsequently executed experiment.

This feature works as follows:

1. During the experiment execution within TAP, the AutoGraph result listener inspects the generated results for names that include information about panel generation.
2. At testplan end, the result listener generates a message describing the panel to generate.
3. If the test case includes a dashboard definition the ELCM will generate the panels described in it first.
4. The logs generated during the experiment execution will be parsed, looking for messages generated by the AutoGraph result listener.
5. For each message detected, a new panel will be generated after the ones described in the test case.

#### Facility resources:

It is possible to define a set of available local resources. These resources can be specified as requirements for the execution of each kind of task inside a test case.

Resources are defined by including a YAML file in the *Resources* folder. The contents of these files are as follows:

- **Id:** Resource ID. This Id must be unique to the facility and will be used to identify the resource on the test cases.
- **Name:** Name of the resource (visible on the ELCM dashboard).

- **Icon:** Resource icon (visible on the ELCM dashboard).

Required resources are configured per task. When an experiment execution is received, the ELCM will generate a list of all the required resources. When an experiment starts, all these resources will be locked and the execution of other experiments with common requirements will be blocked until the running experiment finishes and their resources are released.

### Scenarios and Network Slice deployment:

A scenario is a collection of configuration values that are used to further customize the behavior of a deployed slice. These values are defined as YAML files contained in the *Scenarios* folder, where each file contains a dictionary with a single key (that defines the name of the Scenario). The value for this key is a second dictionary that contains the collection of values that are to be customized by the Scenario.

When the experiment requests the deployment of a Network Slice the ELCM will create a NEST description. The NEST created by the ELCM has 3 main parts:

- A reference to a base slice descriptor, which must be available in the Katana Slice Manager.
- A collection of values that are to be overridden from the base slice descriptor, taken from the selected Scenario.
- A possibly empty list of references to Network Services that are to be included as part of the Network Slice.

#### 5.1.3.3 Available Tasks

The following is a list of the tasks that can be defined as part of a TestCase or UE list of actions, as well as their configuration values:

##### Run.CliExecute

Executes a script or command through the command line. Configuration values:

- **Parameters:** Parameters to pass to the command line (i.e. the line to write on the CLI)
- **CWD:** Working directory where the command will run

##### Run.CompressFiles

Generates a Zip file that contains all the specified files. Configuration values:

- **Files:** List of (single) files to add to the Zip file
- **Folders:** List of folders to search files from. All the files contained within these folders and their sub-folders will be added to the Zip file
- **Output:** Name of the Zip file to generate.

##### Run.CsvToInflux

Uploads the contents of a CSV file to InfluxDb. The file must contain a header row that specifies the names of each column, and must contain a column that specifies the timestamp value of the row as a POSIX timestamp (seconds from the epoch as float, and UTC timezone). Configuration values:

- **ExecutionId:** Id of the execution (can be dinamically expanded from '{@ExecutionId}')
- **CSV:** Path of the CSV file to upload



- **Measurement:** Measurement (table) where the results will be saved
- **Delimiter:** CSV separator, defaults to `','`.
- **Timestamp:** Name of the column that contains the row timestamp, defaults to `"Timestamp"`.
- **Convert:** If True, try to convert the values to a suitable format (int, float, bool, str). Only 'True' and 'False' with any capitalization are converted to bool. If False, send all values as string. Defaults to True.

#### Run.Delay

Adds a configurable time wait to an experiment execution. Has a single configuration value:

- **Time:** Time to wait in seconds.

#### Run.Dummy

Dummy action, will only display the values on the *Config* dictionary on the log

#### Run.Message

Displays a message on the log, with the configured severity. Configuration values:

- **Severity:** Severity level
- **Message:** Text of the message

#### Run.Publish

Saves a value (identified with a name) for use in another task that runs later. The value can be retrieved using the `@[key]` or `@[Publish.key]` variable expansion. If the key is not defined at the time of expansion it will be replaced by the string `<<UNDEFINED>>` unless another default is defined using `@[key:default]`.

In the case of this Task the *Config* dictionary contains the keys and values that will be published. For example, the following tasks:

```
- Order: 5
  Task: Run.Publish
  Config: { Publish1: "Text", Publish2: 1 }
- Order: 10
  Task: Run.Message
  Config:
    Severity: INFO
    Message: "1: @[Publish1]; 2: @[Publish.Publish2]; 3: @[Publish3]; 4:
@[Publish.Publish4:NoProblem]"
```

Will produce this message in the log:

```
- INFO - 1: Text; 2: 1; 3: <<UNDEFINED>>; 4: NoProblem
```

#### Run.PublishFromFile / Run.PublishFromPreviousTaskLog

Reads the contents of a file / the log of the previous task and looks for lines that match the specified regular expression pattern, publishing the groups found. Configuration values:

- **Pattern:** Regular expression to try to match
- **Keys:** List of (index, key) pairs, where index refers to the regex group, and key is the identifier to use when publishing.
- **Path** (only for Run.PublishFromFile): Path of the file to read

### Run.SingleSliceCreationTime

Sends the Slice Creation Time reported by the Slice Manager to InfluxDb. This task will not perform any deployment by itself, and will only read the values for a slice deployed during the experiment pre-run stage. Configuration values:

- **ExecutionId:** Id of the execution (can be dinamically expanded from '{@ExecutionId}')
- **WaitForRunning:** Boolean, wait until the Slice Manager reports that the slice is running, or retrieve results immediately
- **Timeout:** 'WaitForRunning' timeout in (aprox) seconds
- **SliceId:** Slice ID to check (can be dinamically expanded from '{@SliceId}')

### Run.SliceCreationTime

Repeats a cycle of slice creation and deletion for a configured number of times, obtaining the Slice Creation Time on each iteration and sending the values to the configured InfluxDb database. This task does not take into account the slices deployed during the experiment's pre-run stage (if any). This task uses a local NEST file to describe the slice to be deployed. Configuration values:

- **ExecutionId:** Id of the execution (can be dinamically expanded from '{@ExecutionId}')
- **NEST:** Absolute path of the NEST file to use
- **Iterations:** Number of iterations. Defaults to 25
- **Timeout:** Timeout in (aprox) seconds to wait until the slice is running or deleted before skipping the iteration. If not specified or set to None the task will continue indefinitely.
- **CSV:** If set, save the generated results to a CSV file in the specified path. In case of error while sending the results to InfluxDb a CSV file will be forcibly created on '{@TempFolder}/SliceCreationTime.csv' (only if not set, otherwise the file will be created as configured).

### Run.TapExecute

Executes a TAP TestPlan, with the possibility of configuring external parameters. Configuration values:

- **TestPlan:** Path (absolute) of the testplan file.
- **GatherResults:** Indicates whether to compress the generated CSV files to a Zip file (see below)
- **External:** Dictionary of external parameters

If selected, the task will attempt to retrieve all the results generated by the testplan, saving them to a Zip file that will be included along with the logs once the execution finishes. The task will look for the files in the TAP Results folder, inside a sub-folder that corresponds with the experiment's execution ID, for this reason, it is necessary to add a MultiCSV result listener to TAP that has the following (recommended) 'File Path' configuration:

```
Results\{Identifier}\{Date}-{ResultType}-{Identifier}.csv
```

#### 5.1.3.3.1 Variable expansión

It's possible to expand the value of some variables enclosed by '{@ }'. (Use quotes where required in order to generate valid YAML format). Available values are:

- **@{ExecutionId}:** Experiment execution ID (unique identifier)
- **@{SliceId}:** ID of the slice deployed by the Slice Manager during the PreRun stage

- **@{TempFolder}**: Temporal folder exclusive to the current executor, it's deleted when the experiment finishes.
- **@{Application}**: The *Application* field from the Experiment Descriptor
- **@{JSONParameters}**: The *Parameters* dictionary from the Experiment Descriptor, in JSON format (a single line string)
- **@{ReservationTime}**: The *ReservationTime* field of the Experiment Descriptor (minutes), or 0 if not defined
- **@{ReservationTimeSeconds}**: Same as above, but converted to seconds.
- **@{TapFolder}**: Folder where the (Open)TAP executable is located (as configured in *config.yml*)
- **@{TapResults}**: Folder where the (Open)TAP results are saved (as configured in *config.yml*)

Separate values from the *Parameters* dictionary can also be expanded using the following expressions:

- **@[Params.key]**: The value of 'key' in the dictionary, or '<<UNDEFINED>>' if not found
- **@[Params.key:default]**: The value of 'key' in the dictionary, or 'default' if not found

#### 5.1.3.4 MONROE experiments

The ELCM is able to handle the execution of experiments using a MONROE node [49]. This functionality requires:

- A physical or virtual MONROE node that is prepared to be controlled by the TAP agent
- An OpenTAP instance configured with the required TAP instrument and steps for controlling the MONROE TAP agent (available as part of the 5Genesis TAP plugins, and has network connectivity with the MONROE node)

This repository includes the files required for handling the execution of MONROE experiments, however, a small preparation is needed before they can be used:

- The *MONROE\_Base.TapPlan* file is a pre-configured TAP testplan that contains the required steps and external parameters required for controlling the MONROE TAP agent. Open this file using OpenTAP and confirm that no issues appear in the log. In particular:
  - Check that all steps were loaded successfully (should be 9 in total)
  - Check that the necessary result listeners are enabled in the 'Set execution metadata' step
  - Check that your MONROE instrument is selected in all MONROE steps ('Start/List/Stop experiment')
- Save the test plan and, if necessary, move it to another location. Note the absolute path of the file.
- Edit the *TestCases/MONROE\_Base.yml* file. This is a special TestCase definition that is used for handling the execution of MONROE experiments and will not appear as a normal test case for experimenters. Change the '<<Replace with the location of your MONROE\_Base testplan.>>' placeholder with the absolute path of *MONROE\_Base.TapPlan*.

### 5.1.3.5 Distributed experiments

When correctly configured, two 5Genesis platforms can perform the execution of a distributed experiment, in which both platforms execute tasks in a coordinated manner and exchange information with each other. In order to use this functionality, the following conditions must be met:

- On each platform, a test case that defines the set of actions (including any necessary coordination) of that side exists.
- The East/West interface of the ELCM in both sides is enabled, there is connectivity between the two instances and connection details for the remote side's ELCM are defined.
- The remote platforms are registered in the Dispatcher of both sides (see the Dispatcher documentation).

Optionally, in order to ease the creation of a valid experiment descriptor:

- The East/West interface of the Portal in both sides is enabled, there is connectivity between the two instances and connection details for the remote side's Portal are defined.

The creation of a distributed experiment is a collaborative activity between the two platforms involved in the execution of the experiment. Each platform is responsible for the definition of their set of actions, as only they have the required knowledge on the usage of their equipment, but must agree with the other platform's administrators about any necessary coordination and information exchange that is required in order to successfully execute the test case.

The actual definition of the test case is very similar to that of a normal (non-distributed) experiment, but with the following differences:

- The test case definition yaml must include an additional key: *Distributed: True*
- A distributed experiment cannot be *Custom* (i.e. cannot define *Parameters*)
- Additional task types are available (for coordination and information exchange)

The general workflow during a distributed experiment is as follows:

- The Dispatcher of one of the platforms (the `Main` platform) receives a distributed experiment execution request, either from the Portal or through the Open APIs.
- The Dispatcher performs the initial coordination, contacting with the ELCM of its own platform and the Dispatcher of the remote platform (the `Secondary` platform).
- Once the initial coordination is completed, the ELCM on both sides communicate directly for the rest of the experiment execution.
- Each side performs the execution of their tasks as normal, unless they reach a point where they must coordinate:
  - If one of the platforms must wait until the remote side has performed some actions:
    - The waiting platform can use the *Remote.WaitForMilestone* task.
    - The other platform can indicate that the actions have been performed using the *Run.AddMilestone* task.
  - If one of the platforms requires certain information from the remote side:
    - The querying platform can use the *Remote.GetValue* task.

- The other platform can set the value requested using any of the *Run.Publish*, *Run.PublishFromFile* and *Run.PublishFromPreviousTaskLog* tasks.
- Once both platforms execute all their tasks, the `Main` platform requests all the generated files and results to the `Secondary` platform, so that they are saved along with the ones generated by the `Main` and available to the experimenter.

The following tasks are available during the execution of a distributed experiment:

*Remote.WaitForMilestone*

Halts the execution of additional tasks until the remote side specifies that a certain milestone has been reached (using the *Run.AddMilestone* task). Configuration values:

- **Milestone:** Name of the milestone to wait for.
- **Timeout:** Custom timeout for this particular request. If not specified, the value configured in the East/West section of the configuration is used.

*Remote.GetValue*

Halts the execution of additional tasks until a certain value can be obtained from the remote side (using any of the *Run.Publish*, *Run.PublishFromFile* and *Run.PublishFromPreviousTaskLog* tasks). When received, the value will be published internally and available for variable expansion. Configuration values:

- **Value:** Name of the value to request.
- **PublishName:** Name to use when publishing the value. If not specified the same *Value* name will be used.
- **Timeout:** Custom timeout for this particular request. If not specified, the value configured in the East/West section of the configuration is used.

## 5.2 OSM Plugin

In order to facilitate the quick development, release and testing of the platforms, it is imperative that developers and technology providers have the necessary tools to do so. One way to develop and test the viability and practicality of NFV MANO's VNFs and NSDs (which in turn define a NS and partially define a platform) is to deploy and test them using the MANO's native tools (command line or UI Dashboard). However, while feasible for a few quick testing and development iterations, such a process becomes unproductive for large scale development activities and automated testing. As OpenTap is used as the test automation and sequence tool, it is axiomatic that we use this technology to also develop and test the VNFs and NSD that compose our platform. Thus, we provide OpenTap plugins to deploy and destroy VNFs and NSDs/NS in NFV MANO (OSM).

### 5.2.1 OpenTap OSM Instruments

As part of the 5GENESIS package for OpenTap, we provide two OpenTap instruments to connect and manage VNFs and NSD in OSM. These are the *OSM Instrument* and the *VIM Account Instrument*.

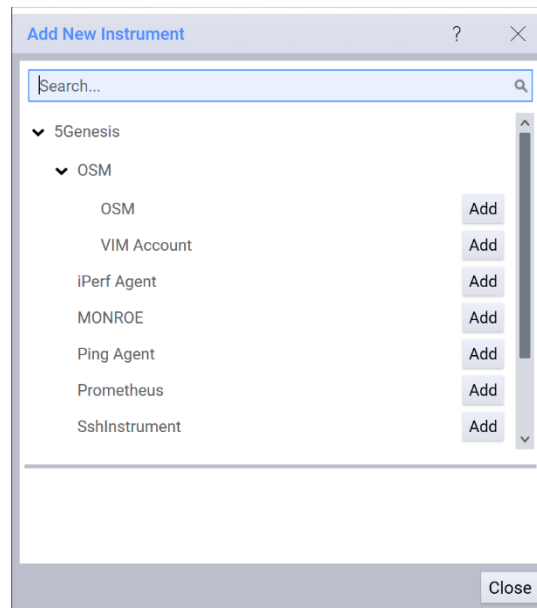


Figure 30 5GENESIS related OpenTap instruments for NFV MANO (OSM)

#### 5.2.1.1 OSM Instrument

This instrument facilitates the developer (tester) to provide information about the OSM instance that would be used for the management and orchestration of VNFs and NS. The required fields are the *Host*, *Username* and *Password*. If no *Project Id* is provided, then the VNFs and NSDs would be deployed and created in the first available project in OSM. The information provided through this instrument is used for accessing the NBI API of NFV MANO in the various OSM related test steps. This instrument allows the configuration of OSM REST endpoint used by the test steps. This instrument also takes care of the *AccessToken* generation.

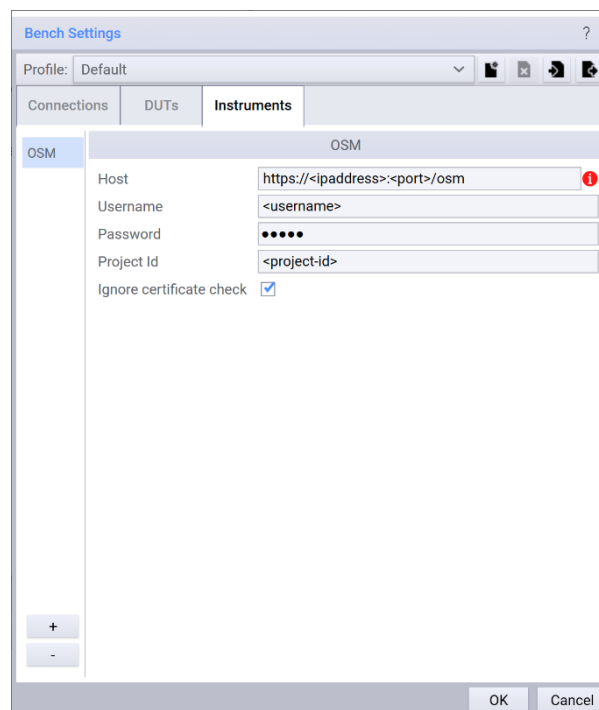


Figure 31 OSM Instrument

The settings of the OSM Instrument are defined as public properties of the Instrument class. The *Display* decorator defines how these settings should be presented to the end user.

```
namespace Tap.Plugins._5Genesis.OSM.Instruments
{
    [Display("OSM", Groups: new string[] { "5Genesis", "OSM" }, Description:
"OSM Instrument")]
    public class OSMInstrument : Instrument
    {
        ...
    }
}
```

```
[Display(Name: "Host", Order: 1.1, Description: "The address of the OSM REST
service endpoint, e.g., http://localhost:9999/osm")]
public string Host { get; set; }

[Display(Name: "Username", Order: 1.2, Description: "The username for
generating the AccessToken")]
public string Username { get; set; }

[Display(Name: "Password", Order: 1.3, Description: "The password for
generating the AccessToken")]
public SecureString Password { get; set; }

[Display(Name: "Project Id", Order: 1.4, Description: "This is optional. If
not provided, the first project available for this user is assigned.")]
public string ProjectId { get; set; }

[Display(Name: "Ignore certificate check", Order: 1.5, Description: "Ignore
the certificate check when requesting a HTTPS resource on the OSM NBI")]
public bool DisableSSLCheck { get; set; }

[Browsable(false)]
public TokenInfo AccessToken { get; private set; }
```

#### 5.2.1.2 VIM Instrument

The NFV MANO can be configured to work with multiple VIM such as OpenStack, OpenVIM, VMware, etc. Thus, it is imperative that the user (developer/tester) has the choice of the VIM to which the VNFs and NSDs/NSs are deployed and created. This instrument provides such a facility. The user first needs to choose the OSM instance and then choose a VIM account registered in this OSM instance and available to this user (based on the OSM credentials provided in the OSM Instrument). The selected VIM account is then used by the various OSM related test steps.

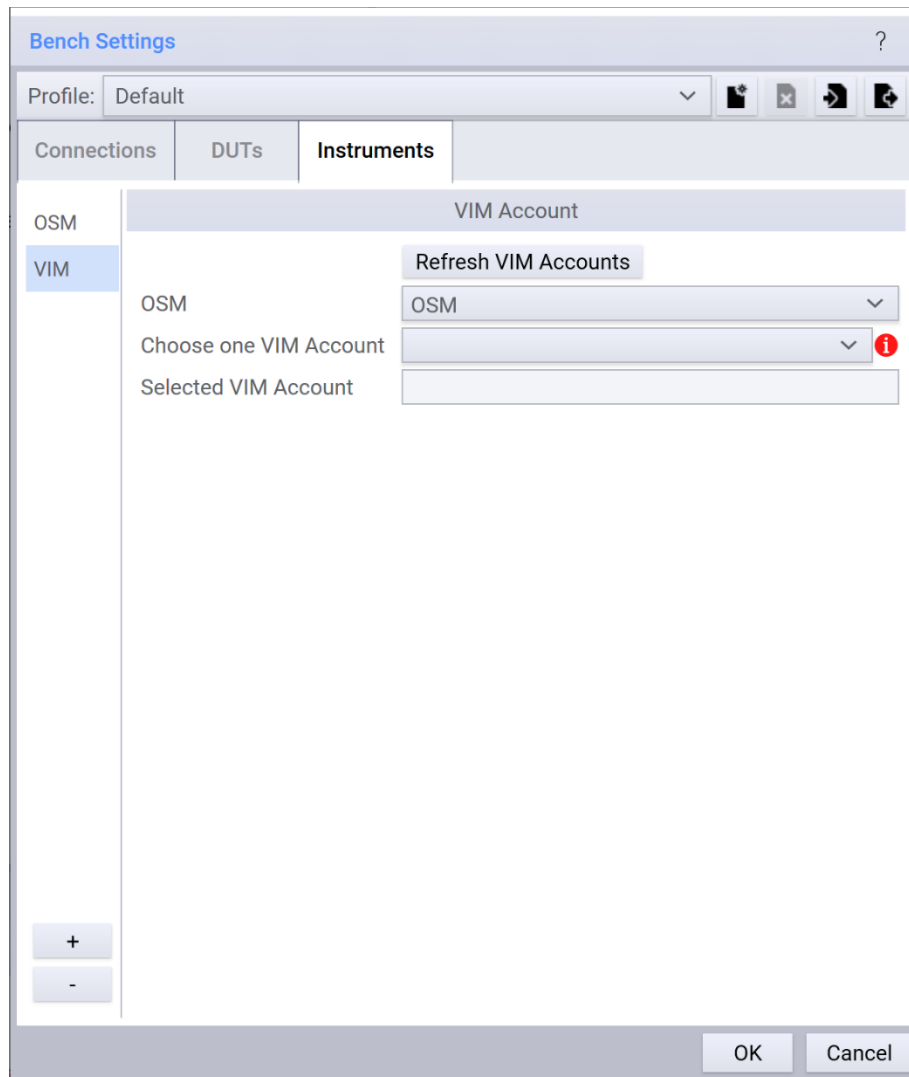


Figure 32 VIM Instrument

Similar to the OSM Instrument, the settings of the VIM Instrument are defined as public properties of the Instrument class. The *Display* decorator defines how these settings should be presented to the end user.

```
namespace Tap.Plugins._5Genesis.OSM.Instruments
{
    [Display("VIM Account", Groups: new string[] { "5Genesis", "OSM" },
    Description: "VIM Account Instrument")]
    public class VIMInstrument : Instrument
    {
        ...
    }
}
```

```
[Display(Name: "OSM", Order: 1.1, Description: "The OSM instance to use")]
public OSMInstrument OSMInstrument { get; set; }

[Browsable(false)]
[XmlIgnore]
public List<VmInfo> AvailableVmInfos { get; private set; }
```



```
private VimInfo _SelectedVimAccount;

[Display(Name: "Choose one VIM Account", Order: 1.2, Description: "Choose the
VIM account available in the chosen OSM")]
[AvailableValues("AvailableVimInfos")]
[Browsable(true)]
[XmlIgnore]
public VimInfo SelectedVimAccount
{
    get
    {
        return _SelectedVimAccount;
    }
    set
    {
        _SelectedVimAccount = value;
        if (_SelectedVimAccount != null)
        {
            SelectedVimAccountName = _SelectedVimAccount.name;
            SelectedVimAccountId = _SelectedVimAccount._id;
        }
        else
        {
            SelectedVimAccountName = null;
            SelectedVimAccountId = new Guid();
        }
    }
}

[Display(Name: "Selected VIM Account", Order: 1.3, Description: "Selected VIM
account available in the chosen OSM")]
public string SelectedVimAccountName { get; set; }

[Display(Name: "Selected VIM Account Id", Order: 1.4, Description: "Selected
VIM account available in the chosen OSM")]
[Browsable(false)]
public Guid SelectedVimAccountId { get; set; }
```

## 5.2.2 OpenTap OSM Test Steps

In order to have control over the complete lifecycle of a VNF and NSD/NS, we provide OpenTap steps that facilitate the creation/deployment and destruction of VNF and NSD/NS in OSM.

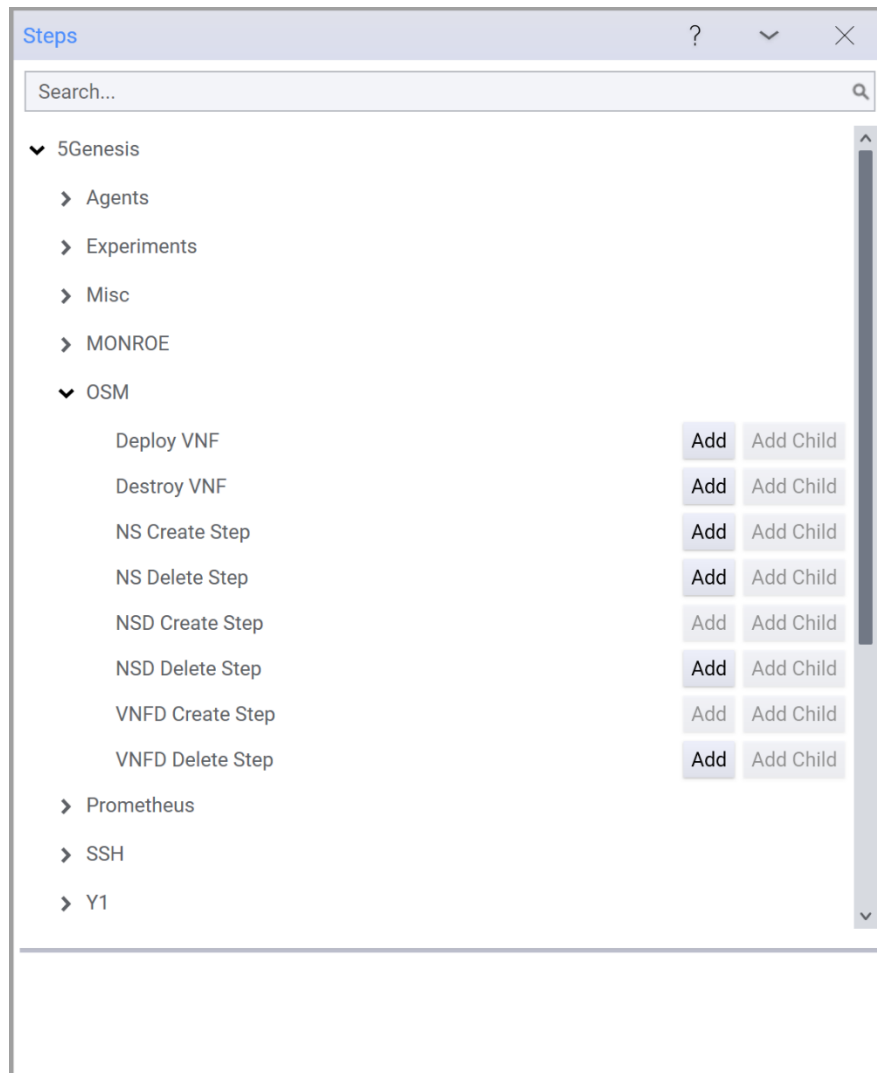


Figure 33 OSM Test Steps

The test steps are categorized as follows:

- Parent Steps
  - Deploy VNF
  - Destroy VNF
- Parent/Child Steps
  - NS Create
  - NS Destroy
- Child Steps
  - VNFD Create
  - VNFD Destroy
  - NSD Create
  - NSD Destroy

All the test classes are extensions of the *OpenTap.TestStep* class. Similar to the instruments, the settings are defined as public properties of the TestStep class. The *Display* decorator defines how these settings are presented to the end user. The class hierarchy of the test steps is illustrated in Figure 34 together with the OSM and VIM instruments.

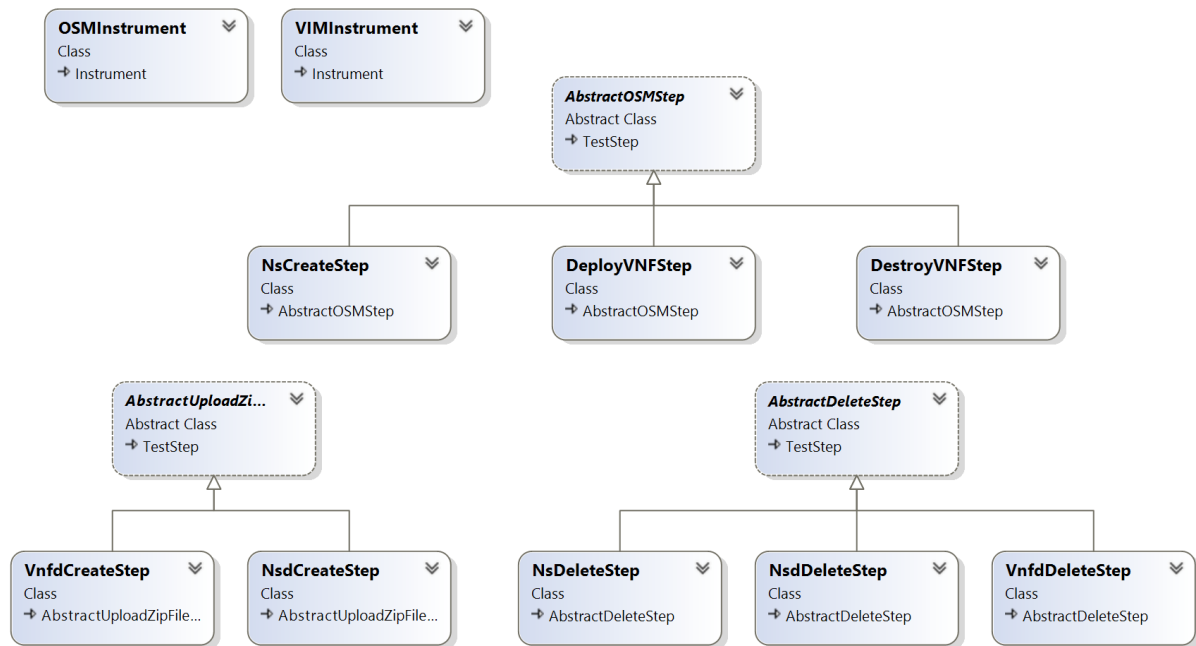


Figure 34 Class hierarchy of the OpenTap OSM Test Steps

The *TestSteps* use a REST client (RestSharp) to access the REST API of the OSM and VIM instances. As OSM provides an OpenAPI definition to its REST API, the API accessor interfaces are auto-generated. The figure below illustrates the class structure of the generated OSM REST APIs.

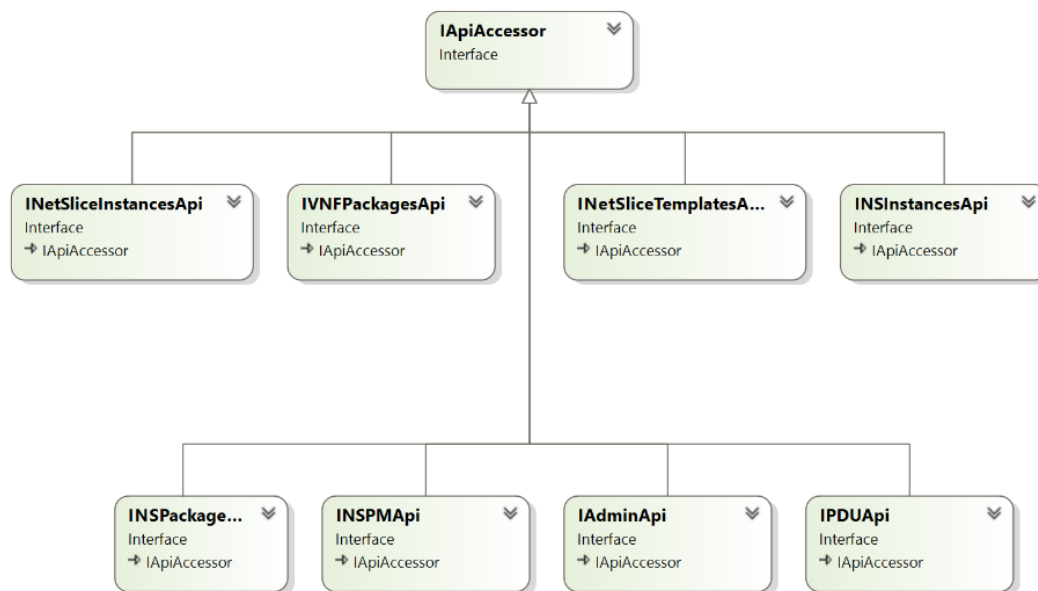


Figure 35 Class hierarchy of the generated OSM API accessors from its OpenAPI definition

Usage of the OSM Plugin for OpenTAP is pretty straightforward. As a prerequisite, ensure that your OSM instance is configured correctly. Either configure your OSM instance with the *emu-vim* VIM emulator for local OSM based testing and development, or with OpenStack / OpenVIM when using OSM deployed in a *staging* / team environment.

Step Name	Verdict	Duration	Flow	Step Type
Deploy VNF - PingPong	Pass	1.33 s		5Genesis \ OSM \ Deploy VNF
VNFD Create Step - Ping	Pass	490 ms		5Genesis \ OSM \ VNFD Create Step
VNFD Create Step - Pong	Pass	428 ms		5Genesis \ OSM \ VNFD Create Step
NSD Create Step - PingPong	Pass	349 ms		5Genesis \ OSM \ NSD Create Step
NS Create Step - PingPong	Pass	50.4 ms		5Genesis \ OSM \ NS Create Step
Destroy VNF - PingPong	Pass	493 ms		5Genesis \ OSM \ Destroy VNF
NS Delete Step - Pingpong	Pass	170 ms		5Genesis \ OSM \ NS Delete Step
NSD Delete Step - PingPong	Pass	178 ms		5Genesis \ OSM \ NSD Delete Step
VNFD Delete Step - Ping	Pass	89.0 ms		5Genesis \ OSM \ VNFD Delete Step
VNFD Delete Step - Pong	Pass	54.5 ms		5Genesis \ OSM \ VNFD Delete Step

Figure 36 – Typical OSM Plugin usage for testing and development

As the first step, create instruments that reflect your OSM and VIM environment. Thereafter add test steps to create VNFDs and NSD, followed by the test step to instantiate the NS. Destroy the VNFDs and NSDs in the reverse order of the creation. A snapshot of the typical OSM Plugin usage is shown above.

## 5.3 Monitoring and Analytics Deployment

The 5GENESIS consortium targets the realization of a full-chain M&A framework, for a complete collection and analysis of the heterogeneous data produced during the usage of the 5GENESIS experimentation framework. The framework ultimately aims to the verification of the infrastructure status during the experiments for the validation of 5G KPIs.

The “Release B” of the 5GENESIS M&A framework has been thus designed and developed, as documented in the project deliverable D3.6 [7]. It includes advanced Monitoring tools and Machine Learning (ML)-based Analytics, divided in three main functional blocks, that is, Infrastructure Monitoring (IM), Performance Monitoring (PM), and Storage/Analytics.

In light of the state-of-the-art in network monitoring and analytics functionalities, the 5GENESIS M&A framework positions itself as a key enabler for a complete validation of 5G KPIs. Its design, which takes roots from former EU H2020 Projects TRIANGLE [48] and MONROE [49], along with its development, which is based on widespread programming languages (e.g., Python and ML libraries for the Analytics component) and results in open-access software components, allow a lightweight integration, use, and exploitation within heterogeneous hardware/software platforms.

### 5.3.1 Result management

#### 5.3.1.1 InfluxDB

InfluxDB [32] is the solution used as storage for the monitoring and analytics framework of 5GENESIS.

It can be installed from a downloaded .deb or .rpm file from the InfluxDB download page, and it can also be installed through the `influxdb` package using the corresponding package manager (depending on the OS being used) after adding the corresponding repository to it. You can check the exact commands for your distribution in the InfluxDB installation documentation [35].

As an example, Ubuntu users must execute the following commands to add the repository:

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,}
${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list
```

Then, to install and start the InfluxDB service in Ubuntu, the commands to execute are:

```
sudo apt-get update && sudo apt-get install influxdb
sudo service influxdb start
```

InfluxDB configuration is stored in a local configuration file that can be found in `/etc/influxdb/influxdb.conf`. Not all the settings must be configured in the file, since all the commented-out settings will use the default values, which can be checked with the command `influxd config`. To launch InfluxDB with a specific configuration file, the `-config` option must be used, as for example:

```
influxd -config /etc/influxdb/influxdb.conf
```

The configuration file to use can also be determined through the environmental variable `INFLUXDB_CONFIG_PATH`.

For details on the configuration of InfluxDB, please refer to its configuration documentation [36].

### 5.3.1.2 Result listener

The 5GENESIS TapPackage for OpenTap includes result listeners that are tailored for use in 5GENESIS. The most important one is the InfluxDB result listener, which is able to send correctly tagged results to InfluxDB (provided that certain steps are included in the executed testplan).

To use the result listener in a testplan, create a new result listener of the InfluxDB kind and modify the configuration so that it points to your instance, as can be seen in Figure 37.

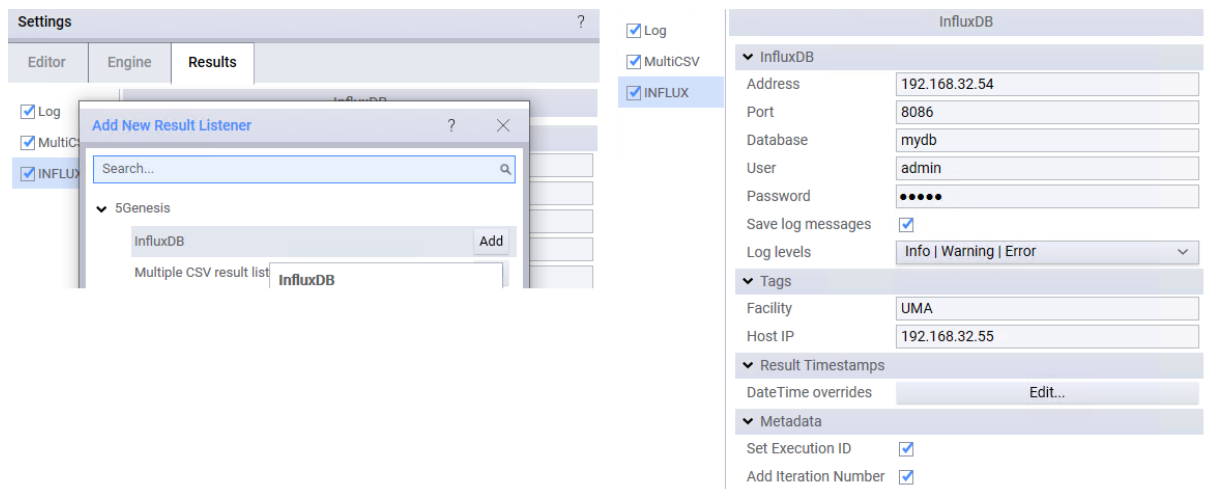


Figure 37 InfluxDB result listener configuration

For the result listener to work effectively the "Set Execution ID" (or "Set Execution Metadata") and "Mark Start of Iteration" steps need to be included in the testplan. An example of a testplan configured for working with the 5GENESIS result listeners can be found at [40]. The general rules are:

- At the start of the testplan there should be a "Set Execution ID" or "Set Execution Metadata" step. This step will set the execution ID for the rest of the testplan execution, so that all results are correctly tagged.
- At the start of the main loop (the loop that defines the N iterations of a test), a "Mark Start of Iteration" should be included. This step will automatically increase the iteration number so that results generated are tagged with the correct `_iteration_` value.

More information about the result listener can be seen in the TapPlugin readme file [41].

### 5.3.2 Prometheus Infrastructure Monitoring

Prometheus [22] is the open-source system monitoring platform chosen for the Infrastructure Monitoring in 5GENESIS.

Prometheus does not need to be installed, but just downloaded and extracted. Once done, to start Prometheus you must simply move to the directory containing the Prometheus binary file and execute:

```
./prometheus --config.file=prometheus.yml
```

To check whether Prometheus has started correctly, you can browse <http://localhost:9090> to see a status page about Prometheus itself.

Its configuration, similar to the Portal and the ELCM, uses YAML. This means the configuration must be provided in a .yaml file, and the file `prometheus.yml` present in the Prometheus directory can be used as a baseline configuration. For details on the configuration of Prometheus check its configuration documentation [37].

Prometheus supports a long list of third-party exporters which extend its functionality allowing to export existing metrics from third-party systems as Prometheus metrics. The complete list of Prometheus exporters can be checked in its documentation [38]. 5GENESIS will make use of

node-exporters to monitor its infrastructure, as for example with the SNMP Exporter [39] for network monitoring.

Additionally, an OpenTAP plugin for Prometheus has been developed to be used in 5GENESIS. It provides an Instrument and a step for retrieving results from a Prometheus instance. The step can be configured for performing any query using PromQL, and time range can be specified either as an absolute start/end or relative to the current time. The retrieved results will be registered by the active result listeners, including the InfluxDb result listener on a correctly configured TAP instance.

## 5.3.3 Performance Monitoring

### 5.3.3.1 ADB Monitoring agents

Along the performance monitoring tools, 5GENESIS makes use of different monitoring agents for Android that can be used through adb, allowing its execution remotely. Those agents are the ping agent, the iPerf agent, and the resource agent.

#### 5.3.3.1.1 Ping agent

This agent acts as a wrapper for the Ping application on Android devices and is useful for network performance testing, being used as a traffic generator. It can be used directly on the device through the application user interface, or through adb commands, the latter allowing remote execution and hence automatic testing.

The user interface, seen in Figure 38, is very simple: it contains two text boxes to introduce the hostname or IP of the target and the IP time to live, a button to start and stop the execution, and a log space where results of the ping transmission will appear.

The agent can also be used via the `startservice` command of adb, sending intents to the agent's service. The command would be the following:

```
adb shell am startservice -n com.uma.ping/.PingService
```

The intents, indicated with the `-a` option, that the application accepts are:

```
com.uma.ping.START (Requires parameters)
```

```
com.uma.ping.STOP
```

The parameters needed for the START intent need to be passed, with the intent extra `-e com.uma.ping.PARAMETERS`, as a comma separated list of (key)=(value) pairs. The exact syntax can be seen in the example below. The parameters that need to be specified are the following:

- `target`: Hostname or IP of the target machine.
- `ttl`: IP time to live

Additionally, it is recommended to use the flag `--user 0` to specify the user of the command, since it may be necessary under some circumstances.

Examples of the usage of the two available intents can be seen below:

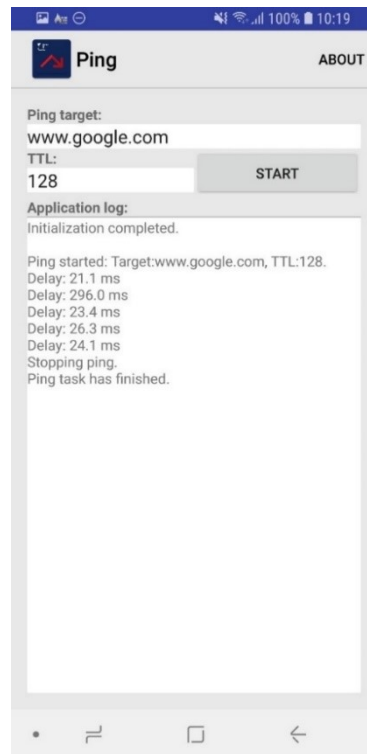


Figure 38 Ping application interface

The agent can also be used via the `startservice` command of `adb`, sending intents to the agent's service. The command would be the following:

```
adb shell am startservice -n com.uma.ping/.PingService
```

The intents, indicated with the `-a` option, that the application accepts are:

```
com.uma.ping.START (Requires parameters)
```

```
com.uma.ping.STOP
```

The parameters needed for the `START` intent need to be passed, with the intent extra `-e com.uma.ping.PARAMETERS`, as a comma separated list of (key)=(value) pairs. The exact syntax can be seen in the example below. The parameters that need to be specified are the following:

- `target`: Hostname or IP of the target machine.
- `ttl`: IP time to live

Additionally, it is recommended to use the flag `--user 0` to specify the user of the command, since it may be necessary under some circumstances.

Examples of the usage of the two available intents can be seen below:

```
adb shell am startservice -n com.uma.ping/.PingService -a com.uma.ping.START  
-e com.uma.ping.PARAMETERS \"target=127.0.0.1,ttl=128\" --user 0
```

```
adb shell am startservice -n com.uma.ping/.PingService -a com.uma.ping.STOP
```

A plugin has been developed to allow the use of the ping agent with OpenTap. An example of an OpenTap testplan including the use of the ping agent can be seen in Annex 3 – Android ADB agents testplan example.



### 5.3.3.1.2 iPerf agent

The iPerf agent acts as a wrapper for the iPerf network measuring tool, allowing the remote use of iPerf on Android terminals and the possibility of performing automated network performance testing.

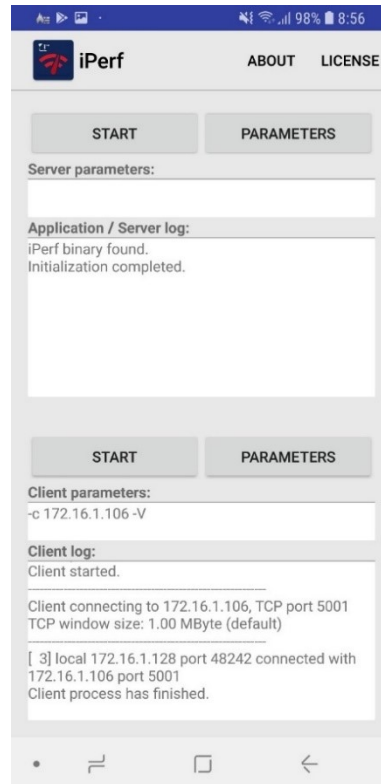


Figure 39 iPerf application interface

Its usage is very similar to that of the ping agent. The user interface is very similar too: it contains one part for server mode and a separate part for client mode, and both of them have a button that allows to configure the parameters, and another to start and stop the execution, along with their corresponding log windows.

For its execution with adb, the startservice command is:

```
adb shell am startservice -n com.uma.ipperf/.iPerfService
```

The accepted intents for this agent are the following:

```
com.uma.ipperf.CLIENTSTART (Requires parameters)
com.uma.ipperf.SERVERSTART (Requires parameters)
com.uma.ipperf.CLIENTSTOP
com.uma.ipperf.SERVERSTOP
```

Parameters for the iPerf binary are passed as a comma separated string (with no spaces) using the intent extra `-e com.uma.ipperf.PARAMETERS`. It is also recommended to specify the flag `-user 0`. Examples of the use of two iPerf agent intents can be seen below:

```
adb shell am startservice -n com.uma.ipperf/.iPerfService -a
com.uma.ipperf.CLIENTSTART -e com.uma.ipperf.PARAMETERS "-c,127.0.0.1,-t,35,-
w,4000K,-p,5002,-l,1470,-f,m,-P,1,-i,1" --user 0
```

```
adb shell am startservice -n com.uma.ipperf/.iPerfService -a com.uma.ipperf.CLIENTSTOP
```

An OpenTap plugin has also been developed to allow the use of the iPerf with it.

#### 5.3.3.1.3 Resource agent

The resource agent acts as a device and network monitoring application for Android devices. Its usage is very similar to that of the ping and iPerf agents. This agent can also be used through a user interface, seen in Figure 40, or executing adb commands. The upper part of the user interface monitors the network interface, offering data about Operator, Network, Cell ID, RSSI, PSC, RSRP, SNR, CQI and RSRQ. The lower part show information about the device monitoring, such as CPU and RAM usage, and packets and bytes both received and transmitted. The network section is updated on network information changes, while the device information will start updating once the user presses the Start button, and then will update every one to three seconds, depending on the device.

For its usage with adb, the startservice command is:

```
adb shell am startservice -n com.uma.resourceAgent/.ResourceAgentService
```

The accepted intents for this agent are the following:

```
com.uma.resourceAgent.START  
com.uma.resourceAgent.STOP
```

Examples of the execution of both intents through adb with the resource agent can be seen below:

```
adb shell am startservice -n com.uma.resourceAgent/.ResourceAgentService -a com.uma.resourceAgent.START
```

```
adb shell am startservice -n com.uma.resourceAgent/.ResourceAgentService -a com.uma.resourceAgent.STOP
```

A plugin has also been developed to allow the use of the resource agent with OpenTap. An example of an OpenTap testplan including the use of the resource agent can be seen in Annex 3 – Android ADB agents testplan example.

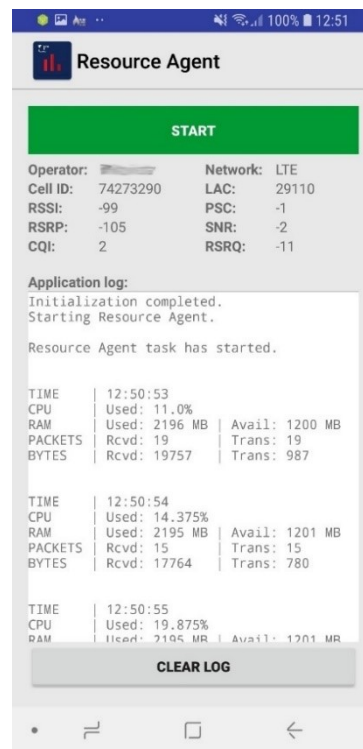


Figure 40 Resource Agent application interface

#### 5.3.3.1.4 Remote PC Agents

Two agents have been developed as part of the performance monitoring tools to be used remotely in any PC: ping remote PC agent and iPerf remote PC agent.

The process for the installation of both agents is very similar. The steps to perform are the following:

1. Clone the repository to a known folder, e.g. in C:\remote-pc-agents
2. Enter the folder iPerfAgent or pingAgent depending on the agent being installed.
3. Create a new Python virtualenv:

```
Pip install virtualenv
virtualenv venv
```

4. Activate the virtual environment:

```
source venv/bin/activate
```

5. Install Python dependencies:

```
Pip install -r requirements.txt
```

6. If installing the iPerf remote PC agent, you need to configure the location of the iPerf executable by editing the config.yml option IPERF\_PATH and indicating the location of the iPerf executable.

7. Start the server:

```
Flask run
```

To stop the server it is necessary to press Control+C. Both agents will start listening for requests on port 5000, but it is possible to change the listening port of the agent defining the environmental variable FLASK\_RUN\_PORT. It is recommended to run a script similar to the following one, in order to set the value of this variable:

```
source ./venv/Scripts/activate
```

```
FLASK_RUN_PORT=8888 flask.exe run
```

Both of the remote PC agents expose a REST API with the following endpoints:

For ping:

- `/Ping/<address>`: Executes ping process to the given address. Returns a JSON reporting the success of the execution and a message.
- `/Ping/<address>/Size/<packetSize>`: Executes ping process to the given address with the specific number of data bytes to be sent. Returns a JSON reporting the success of the command execution and a message.
- `/Close`: Closes ping process. Returns a JSON reporting the success of the process closure and a message.
- `/LastJsonResult`: Retrieve the result of the previous executions. Returns a JSON reporting the success of the retrieval, a message and a list of Results (dictionary with parsed results).
- `/StartDateTime`: Retrieve the date and time of the last execution. Returns a JSON reporting the success of the retrieval and a message informing of the date.
- `/IsRunning`: Check if there is another execution running. Returns a JSON reporting the success of the check and a message informing if is running or not.

For iPerf:

- `/Iperf (POST)`: Executes iPerf process with the specific parameters in the request body **in JSON format**. Returns a JSON reporting the success of the execution and a message.
- `/Iperf/<pathParameters>`: Executes iPerf process with the specific parameters. Returns a JSON reporting the success of the command execution and a message.
- `/Close`: Closes iPerf process. Returns a JSON reporting the success of the process closure and a message.
- `/LastRawResult`: Retrieve the results of the previous execution. Returns a JSON reporting the success of the retrieval, a message and a list of Results (full line).
- `/LastJsonResult`: Retrieve the results of the previous execution. Returns a JSON reporting the success of the retrieval, a message and a list of Results (dictionary with parsed results).
- `/LastError`: Retrieve the errors of the last execution. Returns a JSON reporting the success of the retrieval and a message informing of the error.
- `/StartDateTime`: Retrieve the date and time of the last execution. Returns a JSON reporting the success of the retrieval, a message and a list of Errors.
- `/IsRunning`: Check if there is another execution running. Returns a JSON reporting the success of the check and a message informing if is running or not.

The official 5GENESIS TapPackage includes the instruments and steps necessary for using the iPerf and Ping remote agents. To use these agents with TAP:

1. Create a new iPerf or Ping Agent instrument and specify the address and port where they are running:

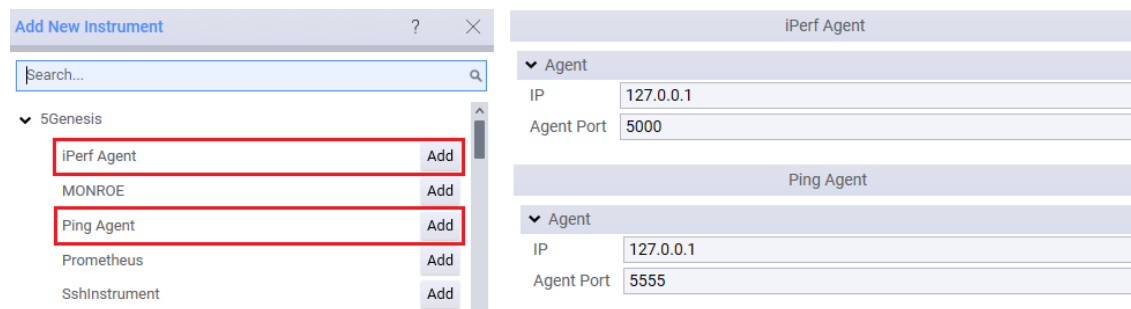


Figure 41 Remote PC agents TAP instrument and configuration

2. You can use the "iPerf Agent" and "Ping Agent" steps (in 5Genesis/Agents):

iPerf Agent Configuration		Ping Agent Configuration	
Agent	iPerfA	Agent	PingA
Action	Measure	Action	Measure
Parameters		Parameters	
Role	Client	Target	8.8.8.8
Host	127.0.0.1	Packet Size	0
Port	5001	Measurement	
Max Run Time	99999 s	Wait Mode	Time
Extra Parameters		Time	4 s
Measurement		Errors	
Wait Mode	Time	Verdict on error	Not Set
Time	4 s	Common	
Errors		Enabled	<input checked="" type="checkbox"/>
Verdict on error	Not Set	Step Name	Ping Agent
Common			
Enabled	<input checked="" type="checkbox"/>		
Step Name	iPerf Agent		

Figure 42 Remote PC agents TAP steps

### 5.3.3.2 MonroeVN

MONROE is a mobile broadband hardware and software measurement platform, supporting deployment of a variety of measurement probes in the form of containers. MONROE VN is a complete virtualization of a MONROE node developed as part of 5GENESIS. Several modifications have been actuated on the MONROE core SW, in order to integrate it within the 5GENESIS experimentation framework. To simplify the installation of MONROE VN on heterogeneous HW platforms, the client-side MONROE SW packages have been modularized and are made available as an apt/deb package (<https://github.com/MONROE-PROJECT/apt-repo>). A scheduling component (i.e., a TAP agent) is integrated as part of the MONROE VN and can be used to manage the execution of measurement probes.

Following are the steps to install a MONROE VN (for further details see [49]):

1. Install a fresh debian stretch (with defaults)
2. (as root or via cloud-init/other mgmt. tool): `apt install -y curl && curl -fsSL https://raw.githubusercontent.com/MONROE-PROJECT/monroe-experiment-core/ReleaseA/get-monroe-release-a.sh -o get-monroe-release-a.sh && sh get-monroe-release-a.sh`

After the installations are done, either the Monroe TAP plugin or curl can be used to test an experiment (for details see <https://github.com/5genesis/monroe-experiment-core/tree/ReleaseA/schedulers/tap-agent>).

Using curl:

1. To deploy and start an experiment, e.g.:
  - o `curl --insecure -H 'x-api-key: $3cr3t_Pa$$w0rd!' -d '{ "script": "monroe/ping:5genesis-rela"}' -H "Content-Type: application/json" -X POST`
2. To stop and retrieve the results (as a zip file), e.g.:
  - o `curl --insecure -H 'x-api-key: $3cr3t_Pa$$w0rd!' -X POST https://<URL>:8080/api/v1.0/experiment/test1/stop -o test1.zip`

Any number of input parameters can be sent during the experiment deployment. For example, as shown in the deployment step above, only the script name is sent ("script": "monroe/ping:5genesis-rela"), other relevant parameters regarding the experiment can also be sent. The MONROE TAP agent receives them as a json string.

Currently installation, deploying and running an experiment require Internet access from the Monroe VN. While curl can be used to test the installation, the 5GENESIS Monroe TAP plugin is the preferred way of integration with the MONROE VN. To carry out the experiments defined in 5GENESIS, two MONROE measurement probes have been designed so far, the *ping* and *throughput containers* detailed below. The MONROE TAP agent offers the interface used to start the probes.

#### 5.3.3.2.1 Ping Container (monroe/ping:virt)

The Ping container<sup>1</sup> measures IP RTT by continuously sending ping packets to a configurable server (default 8.8.8.8, Google public DNS). In 5GENESIS, this container measures latency between where the MONROE VN is deployed and any other end point. The container sends one Echo Request (ICMP type 8) packet per second to a server (end point) over a specified interface until aborted. The RTT is measured as the time between the Echo request and the Echo reply (ICMP type 0) is received from the server. The container is designed to run as a Docker container and does not attempt to do any active network configuration. If the interface selected for the experiment does not exist (i.e., it is switched off) when the experiment starts, the process will immediately exits. The source of the container can be found here: <https://github.com/5genesis/monroe-experiments/tree/ReleaseA/ping>

The default input values are:

```
{
  'dataid': 'MONROE.EXP.PING',
  'interfacename': 'eth0',
  'zmqport': 'tcp://172.17.0.1:5556', 'dataversion': 2,
  'meta_grace': 120,
  'flatten_delimiter': '.',
  'modem_metadata_topic': 'MONROE.META.DEVICE.MODEM',
  'interval': 1000,
  'size': 56,
  'resultdir': '/monroe/results/',
  'nodeid': 'fake.nodeid',
```

<sup>2</sup> <https://github.com/MONROE-PROJECT/Experiments/tree/master/experiments/ping>

```
'server': '8.8.8.8',
'interfaces_without_metadata': 'eth0,wlan0',
'modeminterfacename': 'InternalInterface',
'export_interval': 5.0,
'guid': 'no.guid.in.config.file',
'ifup_interval_check': 5,
'verbosity': 2
}
```

All debug/error information is printed on *stdout* depending on the “verbosity” variable. The container executes a statement similar to running *fping* like the following:

```
fping -I eth0 -D -p 1000 -l 8.8.8.8
```

The container produces a single line JSON object similar to the following (pretty printed and added comments here for readability)

Successful reply

```
{
  "Guid": "313.123213.123123.123123", # exp_config['guid']
  "Timestamp": 23123.1212, # time.time()
  "Iccid": 2332323, # meta_info["ICCID"]
  "Operator": "Telia", # meta_info["Operator"]
  "NodeId" : "9", # exp_config['nodeid']
  "DataId": "MONROE.EXP.PING",
  "DataVersion": 2,
  "SequenceNumber": 70,
  "Rtt": 6.47,
  "Bytes": 84,
  "Host": "8.8.8.8",
}
```

No reply (lost interface or network issues)

```
{
  "Guid": "313.123213.123123.123123", # exp_config['guid']
  "Timestamp": 23123.1212, # time.time()
  "Iccid": 2332323, # meta_info["ICCID"]
  "Operator": "Telia", # meta_info["Operator"]
  "NodeId": "9", # exp_config['nodeid']
  "DataId": "MONROE.EXP.PING",
  "DataVersion": 2,
  "SequenceNumber": 70,
  "Host": "8.8.8.8",
}
```

### *Throughput Container (monroe/iperf:virt)*

The throughput container<sup>3</sup> uses the iPerf tool for active measurements of the maximum achievable bandwidth between two endpoints on IP networks. This container is designed to run on MONROE VN. The container can use either TCP or UDP as the transport protocol. The source of the container can be found here: <https://github.com/5genesis/monroe-experiments/tree/ReleaseA/iperf>

The default input values are:

```
{
  'dataid': '5GENESIS.EXP.IPERF',
  'protocol': 'tcp',
}
```

<sup>3</sup> <https://github.com/MONROE-PROJECT/Experiments/tree/monroe-virtual/experiments/iperf>

```

    'resultdir': '/monroe/results/',
    'flatten_delimiter': '.',
    'interfaces': 'eth0',
    'nodeid': 'virtual',
    'bandwidth': 0,
    'duration': 10,
    'iperfversion': 3,
    'guid': 'fake.guid',
    'metadata_topic': 'MONROE.META',
    'zmqport': 'tcp://172.17.0.1:5556',
    'verbosity': 2,
    'server': '130.243.27.222'
}

```

The iPerf container produces a JSON object (file) per interface (and IP) configured in input "interfaces". An example of produced output for iPerf2 is illustrated below.

```

{
  "DataId": "5GENESIS.EXP.IPERF",
  "Protocol": "tcp",
  "DataVersion": 2,
  "Interface": "eth0",
  "Timestamp": 1551446332.883225,
  "Guid":
"sha256:a4b55ff5a8893c2e267394fd6481a7908e0a7dd9a48d6a29458104b411712ff9.test-iperf2.7.1",
  "NodeId": "7",
  "Results.transferID": "3",
  "Results.transferred_bytes": "1012662272",
  "Results.source_port": "52977",
  "Results.timestamp": "20190301131852.883",
  "Results.destination_address": "192.168.100.13",
  "Results.interval": "0.0-10.0",
  "Results.source_address": "172.18.3.2",
  "Results.destination_port": "5001",
  "Results.bits_per_second": "809884422"
}

```

### 5.3.4 Analytics

The main goal of 5GENESIS Analytics is to enable a full and reliable assessment of 5G KPIs. It thus provides:

- Statistical analysis of the KPIs, as defined in Deliverable D6.1.

Machine Learning (ML) analysis of KPIs and other parameters monitored during the experiment executions, aiming to find correlations and causalities, pinpoint issues leading to performance losses, and ultimately trigger improved configurations during next experiments. The 5Genesis Analytics module provides methods for analysing and offline learning on the data that is provided by the platform monitoring. Full description of requirements and installation procedure are found at <https://github.com/5genesis>:



## Requirements

Docker >= 18.06.0

## Installation

1. Install and start Docker.
2. Activate Docker Swarm (or replace this step with alternative deployment options, such as Kubernetes):

i. Type

```
docker system info
```

ii. Look for "Swarm: active" or "Swarm: inactive"

iii. If "inactive", type the following to set up Docker Swarm:

```
docker swarm init
```

3. Create a Docker Swarm secret by entering the connection details and credentials for each database connection:

```
docker secret create analytics_connections - << END
platform_name:      # replace platform_name, e.g. uma
  host: ip           # replace ip, e.g. 192.168.0.1
  port: p            # replace p with the port, e.g. 8080
  user: u            # replace u with the user name, e.g. user1
  password: pw       # replace pw with the password
  databases:
    - db_name1       # replace db_name1 with your database name(s)
    - db_name2       # replace or delete if there is only one database
END
```

4. Create a Docker Swarm secret for user authentication. Users can only access experiment IDs that are encrypted with this secret. A master password access is also granted with this secret. Replace "secret123" with the actual secret string.

```
echo secret123 | docker secret create analytics_secret -
```

5. Clone this repository.

6. The Analytics Module can be configured to cache the requested data for faster future use. This is disabled by default for compatibility, but can be enabled by setting `ENABLE_CACHE` to `true` in the stack file `analytics-stack.yaml`:

```
services:
  data_handler:
    ...
    environment:
      ENABLE_CACHE: "true"
    ...
```

7. Build and deploy containers with

```
./Analytics/install.sh
```

Once the containers are installed and running, a graphical user interface will be available at <http://localhost:5005/dash>. Note that a user authentication token is required to access experiment results. The token contains the valid experiment IDs encrypted by the 5Genesis Portal. A secret is shared between the Portal and the Analytics to facilitate the exchange of the encrypted authenticated experiment IDs. This is fully described in the installation instructions.

The individual containers are accessible through the different tabs of the visualization service, and also through REST APIs as described below.

The tabs in the visualization service are:

- (1) time series overview: graphical representation of the time series for the KPIs,

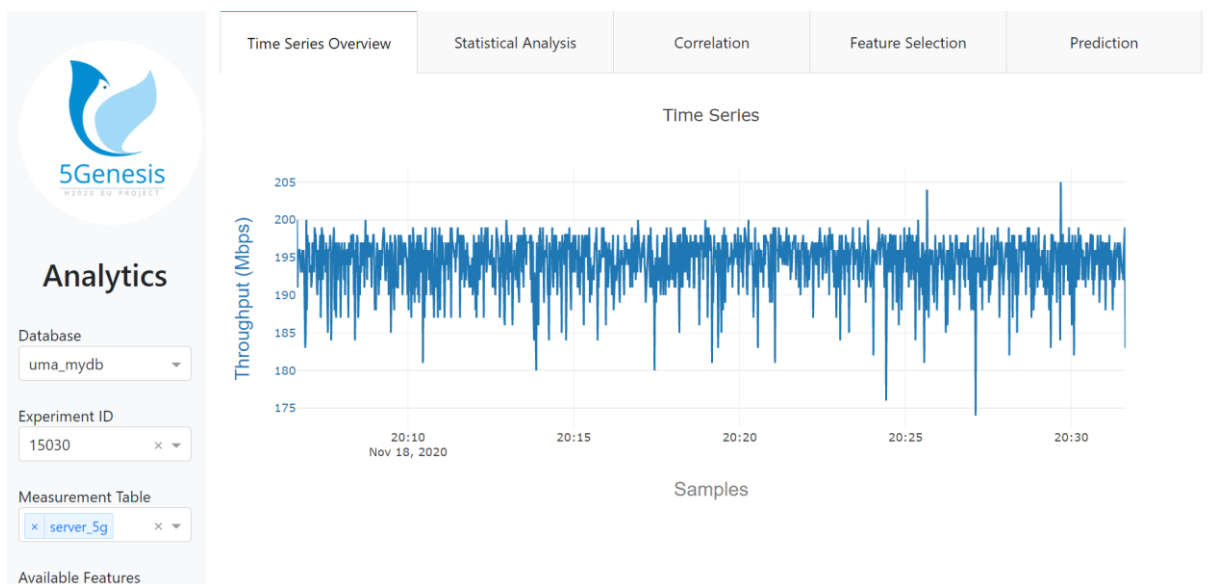


Figure 43 Time series overview: graphical representation of the time series for the KPIs

- (2) statistical analysis: main stats for the KPIs,

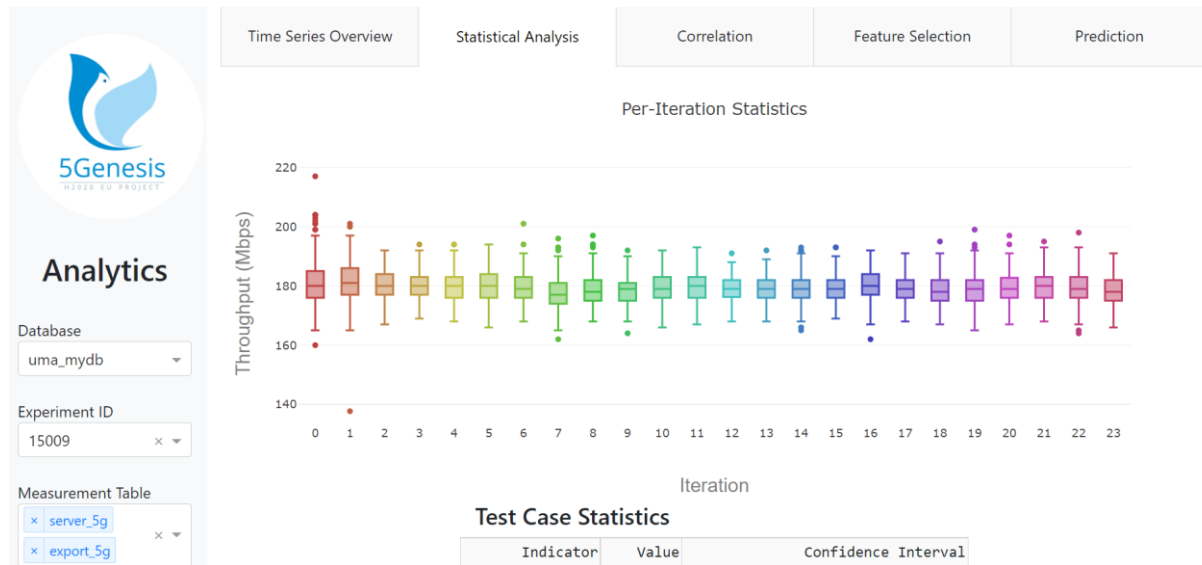


Figure 44 Statistical analysis: main stats for the KPIs

(3) KPI correlation: correlation matrices / coefficients for different KPIs in the same / different experiments,

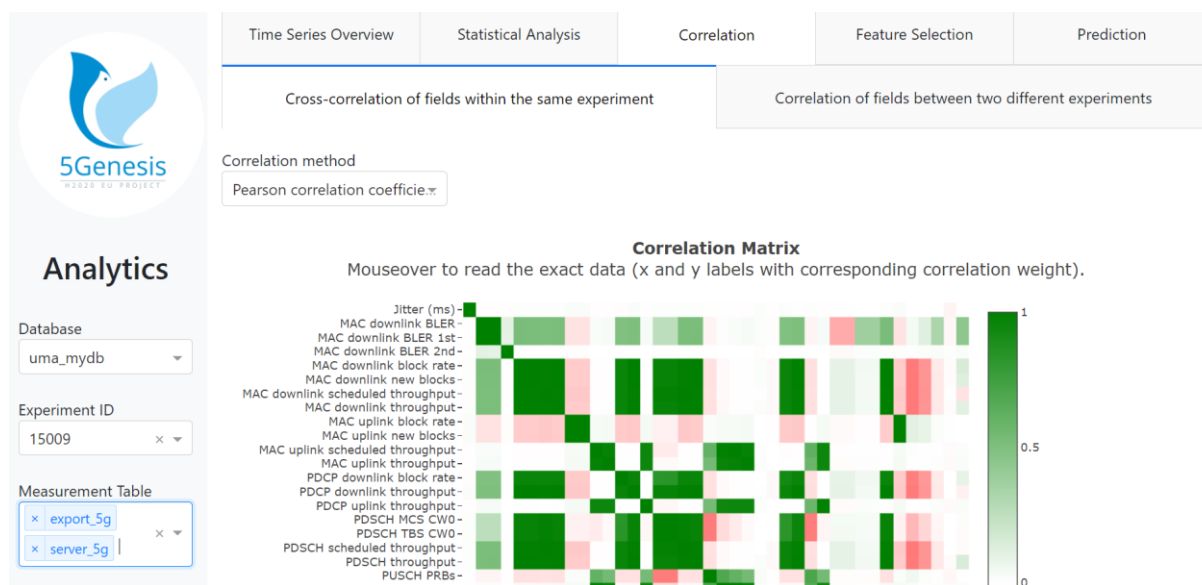


Figure 45 KPI correlation: correlation matrices / coefficients for different KPIs in the same / different experiments

(4) feature selection: elimination of redundant features,

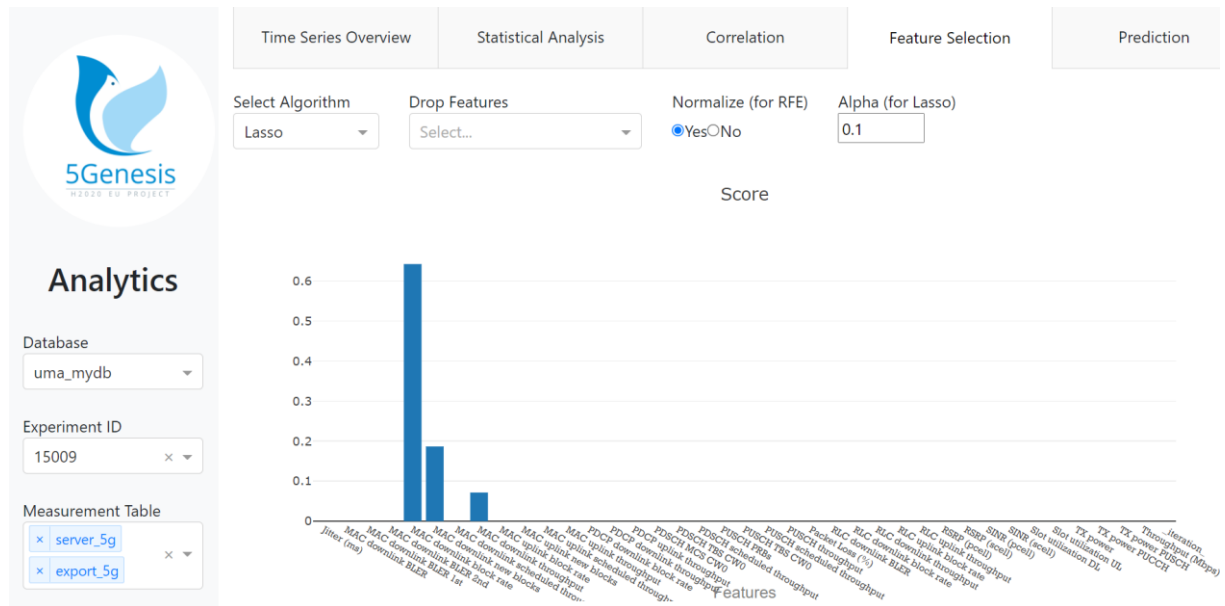


Figure 46 Feature selection: elimination of redundant features

(5) KPI prediction: prediction of KPIs based on historical information and relations to other KPIs.

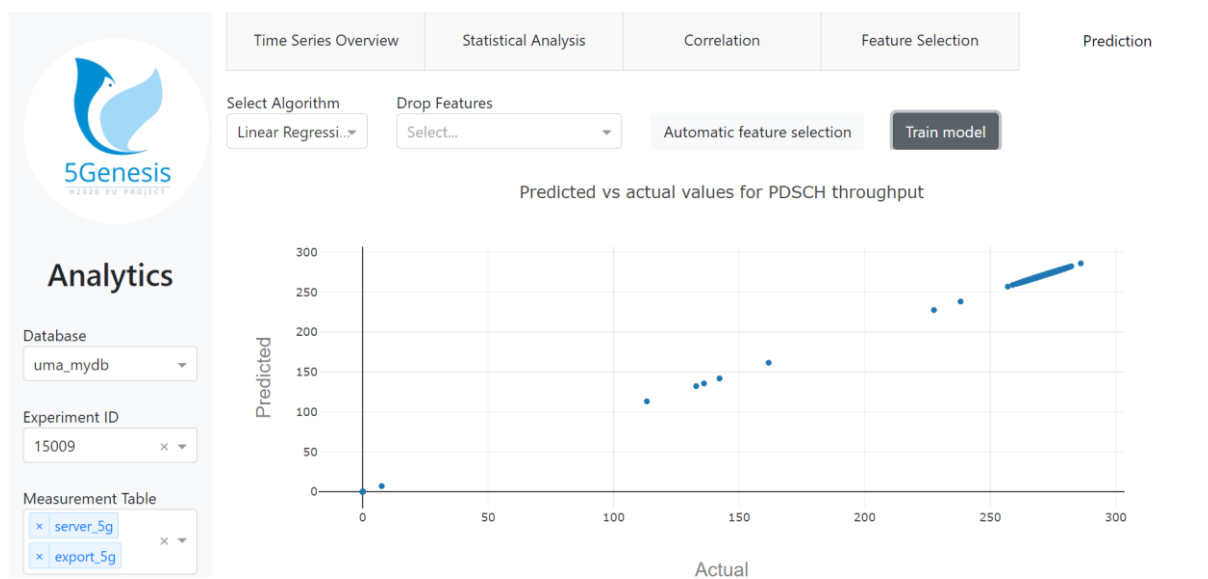


Figure 47 KPI prediction: prediction of KPIs based on historical information and relations to other KPIs

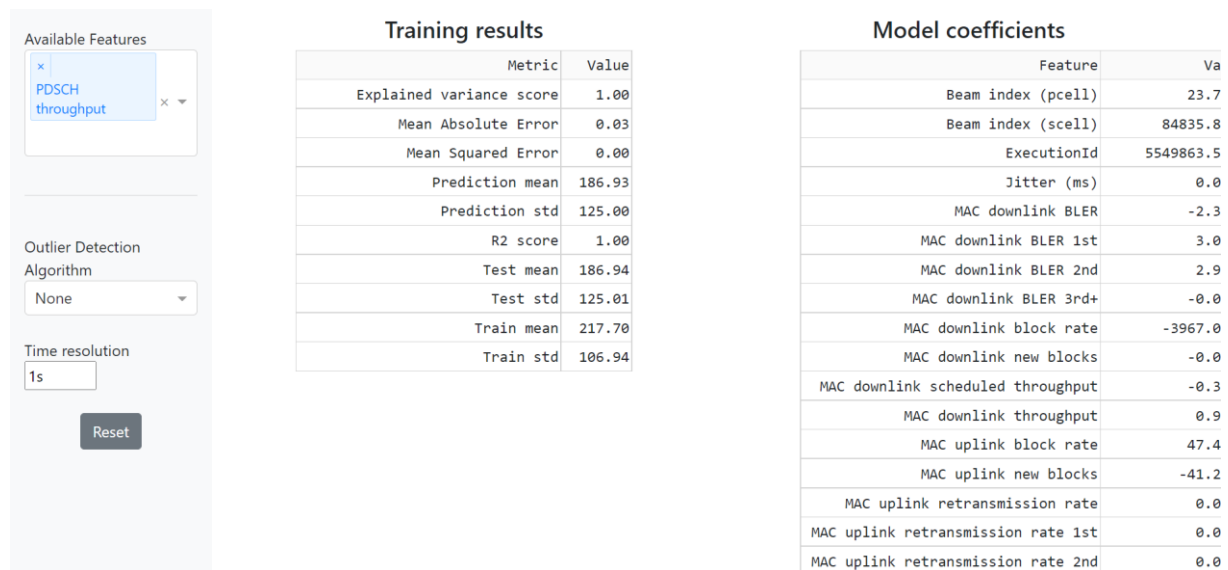


Figure 48 KPI prediction: prediction of KPIs based on historical information and relations to other KPIs (details)

Apart from the visualization service and the ones listed above, there is a DataHandler service that can be consumed through their respective REST APIs.

**DataHandler** is available for retrieving data and for data preprocessing (e.g., synchronization of KPI time series). A call to the DataHandler will return a collection of KPIs and their values (as exemplified in Fig xx below), where each data point is UNIX-timestamped. The data handler service also provides other endpoints, e.g., listing all available datasources, all available experiments, available experiments for a given measurement and available measurements for a given experiment.

`data-handler-URL/get_data/datasource/experimentId`

Response from /get\_data:

```
{
  "PDCP downlink throughput": {
    "1600811115000": 163.4,
    "1600811116000": 160.8,
    ...
  },
  "CellDistance": {
    "1600811115000": 44.9959961221,
    "1600811117000": 47.3855153703,
    ...
  },
  ...
}
```

Figure 49 DataHandler API and example response

**Statistical Analysis:** The call to the statistical analysis endpoint returns test statistics about the experiment data. Experiment id, KPI and measurement parameters must be specified, as shown in figure below. The result returns statistics metrics such as percentiles, min, max and mean for each individual experiment iteration (0, 1, 2, ...), as well as for the whole experiment run, averaged over all iterations.

```
stat-analysis-URL/statistical_analysis/database?
experimentid=123&kpi=Throughput&measurement=
throughput_measures
```

Response from /statistical\_analysis:

```
{
  "Throughput": {
    "Iteration Statistics": {
      0: {
        "5% Percentile": 167,
        "25% Percentile": 176,
        ...
      },
      1: {
        "5% Percentile": 171.8,
        "25% Percentile": 177,
        ...
      },
      ...
    },
    "Test Case Statistics": {
      "5% Percentile": {
        "Value": 171.43958333333333,
        "Confidence Interval":
          0.5810154812485544
      },
      "25% Percentile": {
        "Value": 175.890625,
        "Confidence Interval":
          0.3156676239473986
      },
      ...
    }
  }
}
```

Figure 50 StatisticalAnalysis API & response example

**Linear Correlation Analysis:** The most common call to the linear correlation service is to request the correlation matrix for all fields/ KPIs in an experiment. An optional parameter `removeoutliers` can also be specified, to exclude outliers from the data used for the correlation. The exact call and results are exemplified in Fig zzz below. The result shows the pairwise correlation values for all fields. The correlation service also offers another endpoint (`/correlate/experiments`) that allows the user to retrieve the correlation values for KPIs across different experiments.

```
/correlate/fields/datasource/experimentId?
remove_outliers=zscore
```

Response from /correlate/fields:

```
{
  "CellDistance": {
    "CellDistance": 1,
    "PDCP downlink throughput": -0.382707002,
    "SINR": -0.336993597,
    ...
  },
  "PDCP downlink throughput": {
    "CellDistance": -0.382707002,
    "PDCP downlink throughput": 1,
    "SINR": 0.825896797,
    ...
  },
  "SINR": {
    "CellDistance": -0.336993597,
    "PDCP downlink throughput": 0.825896797,
    "SINR": 1,
    ...
  },
  ...
}
```

Figure 51 Correlation API call & response example

**Feature Selection:** The feature selection service offers one endpoint (selection), which requires the experiment id and measurement parameters. The result contains three parts. "Features - Original" contains the full unfiltered list of features in the data, "Features - Selected" contains the subset of features that were chosen by the feature selection algorithm and "Score" contains the scores that the algorithm assigned to each feature for more information about the selected features.

```
feature-selection-URL/selection/datasource/
algorithm/target?experimentid=123&measurement
=throughput_measures
```

Response from /selection:

```
{
  "Features - Original": {
    0: "MAC downlink BLER",
    1: "MAC downlink BLER 1st",
    2: "MAC downlink BLER 2nd",
    3: "PDCP downlink block rate",
    4: "RLC downlink throughput",
    ...
  },
  "Features - Selected": {
    0: "PDCP downlink block rate",
    1: "RLC downlink throughput"
  },
  "Score": {
    "MAC downlink BLER": 0,
    "MAC downlink BLER 1st": 0,
    "MAC downlink BLER 2nd": 0,
    "PDCP downlink block rate": 0.848355061,
    "RLC downlink throughput": 0.0458724611,
    ...
  }
}
```

Figure 52 Feature selection API & response example

**Prediction:** The prediction service offers an endpoint to train an ML model, where the desired algorithm and target KPI are specified in the path. At least one experiment id parameter must be specified as well, and we may also ask the prediction service to remove outliers by specifying the outlier removal method. The result shows the co-efficient of the trained model (feature importance), the real and predicted values of the test set, and the results of the prediction on the test set, including R2 and Mean Squared Error (MSE) (exemplified in Fig uuu). A second endpoint (/model) is also available that allows to download the last trained ML model for future use.

```
prediction-URL/train/datasource/algorithm/target?
experimentid=123&remove_outliers=zscore
```

Response from /train:

```
{
  "coefficients": {
    "MAC downlink throughput": 0.4974099716,
    "PDCP downlink block rate": 0.2324643848,
    "RLC downlink throughput": 0.1372971483,
    "MAC downlink scheduled throughput":
      0.0665096093,
    "PDSCH throughput": 0.0533150278,
    ...
  },
  "real_predicted_values": {
    "y_pred": {
      0: 109.8001233468,
      1: 141.7893004874,
      ...
    },
    "y_test": {
      0: 110.3,
      1: 130.7,
      ...
    }
  },
  "results": {
    "R2 score": 0.9956903516,
    "Mean Squared Error": 19.393652019,
    ...
  }
}
```

Figure 53 Predict API and example response

ML methods in 5GENESIS Analytics are based on standard libraries. Once the data needed for specific analyses are retrieved, they are managed via well-known and well-documented statistical and ML libraries, such as pandas and scikit-learn, among others. Source code and documentation of Analytics algorithms are accessible at <https://github.com/5genesis/Analytics>.

## 5.4 Slice Manager Deployment

Katana Slice Manager is a centralized component of the 5GENESIS architecture that operates on top of the several MANO layer components. It is responsible for creating multiple dedicated virtual networks using a shared physical infrastructure. Each virtual network slice is composed of independent and isolated logical network functions, as described in D3.4 [6]. Each network slice is optimized to serve the required resources and quality of service (QoS) regarding latency, throughput, capacity, coverage. Katana offers an interface, the NorthBound Interface, implemented as a set of RESTful APIs that enables managing and monitoring the platform's underlying physical and virtual resources, as well as the instantiated network slices. Through



the NBI, Katana interacts with the Coordination Layer components or directly with the network operator and receives the Network Slice Template (NEST) that triggers the network slice creation process. Through the SouthBound Interface, Slice Manager communicates with the various components of the MANO Layer, namely the NFVO, the VIM, the EMS, and the WIM, to manage the virtual and physical network functions comprising instantiated end-to-end network slices. This section describes the steps that the Slice Manager administration must execute to install and configure the Slice Manager and deploy network slices. Note that this guide presents both the `katana-cli` tool and the respective RESTful API request for each step.

### 5.4.1 Deployment

Katana Slice Manager is implemented as a mesh of integrated microservices working collectively to offer slice management services. Each microservice is a module running in a dedicated Docker container, forming the Slice Manager's software stack. For this purpose, the deployment environment of Katana Slice Manager must fulfill the following requirements:

- docker version  $\geq$  18.09.6
- docker-compose version  $\geq$  1.17.1
- Hardware resources: 2 vCPUs, 4GB RAM, 40GB Disk
- One network interface with external connectivity

To deploy Katana Slice Manager on a physical or virtual server, the following steps must be executed:

1. Install Docker [42] and Docker Compose [43] on the deployment server.
2. Download the source code from the Gitlab repository:

```
git clone https://github.com/medianetlab/katana-slice_manager.git
cd slice-manager
```

3. Checkout to the Release\_B Branch:

```
git checkout Release_B
```

4. Run the installation script. This script builds the necessary Docker images that will be used for running the various microservices of the Slice Manager software stack. Furthermore, it installs on the server environment the `katana-cli` tool that can be used for interacting with the Slice Manager services.

```
sudo ./install.sh
```

5. Run the deployment script. This script deploys the Docker containers that comprise the Slice Manager software stack from the already built Docker images.

```
./start.sh [-m] [-p]
```

The deployed services running in Docker containers are depicted in the figure below:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS		NAMES		
c2d140f2f9df	mnlab/katana-swagger:latest	"/docker-entrypoint..."	31 hours ago	Up 31 hours
80/tcp, 0.0.0.0:8001->8080/tcp		katana-swagger		
7e362b7358f0	mnlab/katana-nfv_mon:latest	"python katana/expor..."	31 hours ago	Up 31 hours
0.0.0.0:8002->8002/tcp		katana-nfv_mon		
5cda317c120e	mnlab/katana-apex:latest	"/bin/sh -c 'sh -c \"..."	31 hours ago	Up 31 hours
12345/tcp		katana-apex		
33ba5c54dc37	mnlab/katana-mngr:latest	"python3 katana/kata..."	31 hours ago	Up 31 hours
		katana-mngr		
d9ac3f64e705	mnlab/katana-cli:latest	"/bin/bash"	31 hours ago	Up 31 hours
		katana-cli		
4988cdd45c79	mnlab/katana-nbi:latest	"gunicorn -b 0.0.0.0..."	31 hours ago	Up 31 hours
0.0.0.0:8000->8000/tcp		katana-nbi		
a8bc2a88d2e7	mnlab/katana-grafana:latest	"/run.sh"	31 hours ago	Up 31 hours
0.0.0.0:3000->3000/tcp		katana-grafana		
7f51671131e8	confluentinc/cp-enterprise-kafka:5.4.2	"/etc/confluent/dock..."	31 hours ago	Up 31 hours
0.0.0.0:9092->9092/tcp		katana-kafka		
dd73007134c4	confluentinc/cp-zookeeper:5.4.2	"/etc/confluent/dock..."	31 hours ago	Up 31 hours
2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp		katana-zookeeper		
3a163f09c885	mongo:4.0.5	"docker-entrypoint.s..."	31 hours ago	Up 31 hours
27017/tcp		katana-mongo		
8784f5cf5e0f	mnlab/katana-prometheus:latest	"/bin/prometheus --c..."	31 hours ago	Up 31 hours
0.0.0.0:9090->9090/tcp		katana-prometheus		

Figure 54: Katana Slice Manager Containers

The deployment script accepts two optional flags, “-m” and “-p”. When using the “-m” option, three additional services will be instantiated to enable the Slice Manager monitoring module. On the other hand, the “-p” option will force the katana Kafka message broker to be accessible by external Kafka producer and consumer services. For this purpose, the system administrator must define the external IP address or domain name of the Slice Manager.

## 5.4.2 Configuration

A basic configuration of the Katana Slice Manager can be done following these steps:

1. Create the configuration files for the underlying components of the platform. More specifically, the configuration files include details for the VIM, NFVO, WIM, and EMS employed to support the end-to-end slice deployment. The JSON schema of the said configuration files are presented in Annex 5 – Southbound components configuration files schemas.

2. Add the VIMs that are part of the slice deployment:

```
katana vim add -f <vim_conf.json>
curl -XPOST \
  -H "Content-type: Application/Json" \
  -d "@<vim_conf.json>" \
  http://<KatanaAddress>:8080/api/vim
```

3. Add the NFVO that will be the orchestrator for the VIMs:

```
katana nfvo add -f <nfvo_conf.json>
curl -XPOST \
  -H "Content-type: Application/Json" \
  -d "@<nfvo_conf.json>" \
  http://<KatanaAddress>:8080/api/nfvo
```

4. Add the WIM that will manage the transport network of the platform:

```
katana wim add -f <wim_conf.json>
curl -XPOST \
  -H "Content-type: Application/Json" \
  -d "@<wim_conf.json>" \
  http://<KatanaAddress>:8080/api/wim
```

5. Add the EMS that will manage the radio components of the platform:

```
katana ems add -f <ems_conf.json>
```

```
curl -XPOST \
  -H "Content-type: Application/Json" \
  -d "@<ems_conf.json>" \
  http://<KatanaAddress>:8080/api/ems
```

6. Get a list with the configured components:

```
katana <component> ls
curl -XGET http://<KatanaAddress>:8080/api/<component>
```

7. You can inspect an added component:

```
katana <component> inspect <component_id>
curl -XGET http://<KatanaAddress>:8080/api/<component>/<component_id>
```

8. Delete an added component:

```
katana <component> rm <component_id>
curl -XDELETE \
  http://<KatanaAddress>:8080/api/<component>/<component_id>
```

9. Update a component:

```
katana <component> update <component_id> -f <new_component_conf.json>
curl -XPUT \
  -H "Content-type: Application/Json" \
  -d "@<new_component_conf.json>" \
  http://<KatanaAddress>:8080/api/<component>/<component_id>
```

### 5.4.3 Slice Instantiation

The Network Slice Template (NEST) is used for describing the parameters of a network slice. A NEST must complement a slice creation request that a user sends to the Slice Manager to deploy a new network slice. Annex 6 –NEST presents the JSON of the NEST, while D3.4 includes a full specification of the template. The NEST file is included in a slice creation request:

```
katana slice add -f <NEST>
curl -XPOST \
  -H "Content-type: Application/Json" \
  -d "@<NEST>" \
  http://<KatanaAddress>:8080/api/slice
```

This command will trigger a new slice creation. The following two commands return a list with all the active slices and their status and extended details of a specific slice, respectively:

```
katana slice ls
curl -XGET http://<KatanaAddress>:8080/api/slice
katana slice inspect <slice_id>
curl -XGET http://<KatanaAddress>:8080/api/slice/<slice_id>
```

Moreover, the Slice Manager administrator can check the deployment time of any instantiated slice:

```
katana slice deployment_time <slice_id>
curl -XGET http://<KatanaAddress>:8080/api/slice/<slice_id>/time
```

Finally, the following command will terminate an active slice:

```
katana slice rm <slice_id>
curl -XDELETE http://<KatanaAddress>:8080/api/slice/<slice_id>
```

## REFERENCES

---

- [1] 5G PPP Phase 3 projects [Online]. Available: <https://5g-ppp.eu/5g-ppp-phase-3-projects/>
- [2] 5GENESIS, Deliverable D2.1 “Requirements of the Facility” [Online], [https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS\\_D2.1\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS_D2.1_v1.0.pdf)
- [3] 5GENESIS, Deliverable D2.4 “Final report on facility design and experimentation planning” [Online], [https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS\\_D2.4\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS_D2.4_v1.0.pdf)
- [4] 5GENESIS, Deliverable D2.3 “Initial planning of tests and experimentation” [Online], [https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS\\_D2.4\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS_D2.4_v1.0.pdf)
- [5] 5GENESIS Consortium, "D3.2 Management and orchestration (Release B)," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2021/03/5GENESIS\\_D3.2-v1.0.pdf](https://5genesis.eu/wp-content/uploads/2021/03/5GENESIS_D3.2-v1.0.pdf).
- [6] 5GENESIS Consortium, "D3.4 Slice management WP3 (Release B)," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2021/05/5GENESIS\\_D3.4\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2021/05/5GENESIS_D3.4_v1.0.pdf).
- [7] 5GENESIS, Deliverable D3.6 “Monitoring and Analytics (Release B)” [Online], [https://5genesis.eu/wp-content/uploads/2021/05/5GENESIS\\_D3.6\\_v1.0\\_FINAL.pdf](https://5genesis.eu/wp-content/uploads/2021/05/5GENESIS_D3.6_v1.0_FINAL.pdf)
- [8] 5GENESIS Deliverable D3.9 “5G Core Network WP3 Functions (Release A)," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS\\_D3.9\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.9_v1.0.pdf).
- [9] 5GENESIS Deliverable D3.11 “Access Components and User Equipment," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS\\_D3.11\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.11_v1.0.pdf).
- [10] 5GENESIS Deliverable D3.14 “Security Framework – Release B”, 2021, [Online] Available: [https://5genesis.eu/wp-content/uploads/2021/03/5GENESIS\\_D3.14\\_v.1.0.pdf](https://5genesis.eu/wp-content/uploads/2021/03/5GENESIS_D3.14_v.1.0.pdf)
- [11] 5GENESIS Deliverable D4.2 “The Athens Platform," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS\\_D4.2\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.2_v1.0.pdf)
- [12] 5GENESIS Deliverable D4.5 “The Malaga Platform," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS\\_D4.5\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.5_v1.0.pdf)
- [13] 5GENESIS Deliverable D4.8 “The Limassol Platform," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS\\_D4.8\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.8_v1.0.pdf)
- [14] 5GENESIS Deliverable D4.11 “The Surrey Platform," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS\\_D4.11\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.11_v1.0.pdf)
- [15] 5GENESIS Deliverable D4.14 “The Berlin Platform," [Online]. Available: [https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS\\_D4.14\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.14_v1.0.pdf)
- [16] 5GENESIS Deliverable D5.3 “Documentation and supporting material for 5G stakeholders (Release A)". Available: [https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS-D5.3\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2020/07/5GENESIS-D5.3_v1.0.pdf)
- [17] 5GENESIS, Deliverable D6.1 “Trials and experimentation -cycle 1” [Online], [https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS\\_D6.1\\_v2.00.pdf](https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS_D6.1_v2.00.pdf)
- [18] 5GENESIS, Deliverable D6.2 “Trials and experimentation -cycle 2” [Online], [https://5genesis.eu/wp-content/uploads/2020/08/5GENESIS\\_D6.2\\_v1.0\\_FINAL.pdf](https://5genesis.eu/wp-content/uploads/2020/08/5GENESIS_D6.2_v1.0_FINAL.pdf)
- [19] 5GENESIS, Deliverable D3.8 “Open APIs, service level functions and interfaces for verticals (Release B)", [Online] Available: [https://5genesis.eu/wp-content/uploads/2021/04/5GENESIS\\_D3.8\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2021/04/5GENESIS_D3.8_v1.0.pdf)
- [20] Curl, <https://curl.haxx.se/>
- [21] Open TAP, : <https://doc.opentap.io/Developer%20Guide/Introduction/>
- [22] Prometheus, <https://prometheus.io/>
- [23] Restsharp: <http://restsharp.org/>

- [24] Python, <https://www.python.org/>
- [25] Grafana, <https://grafana.com/>
- [26] Grafana Reporter, <https://github.com/IzakMarais/reporter>
- [27] Python virtualenv, <https://virtualenv.pypa.io/en/stable/>
- [28] Vagrant, <https://www.vagrantup.com/downloads.html>
- [29] Virtualbox, <https://www.virtualbox.org/wiki/Downloads>
- [30] StackOverflow “Demystify Flask app.secret\_key” ,  
<https://stackoverflow.com/a/22463969>
- [31] SQL dialects, <https://docs.sqlalchemy.org/en/latest/dialects/index.html>
- [32] InfluxDB, <https://www.influxdata.com/>
- [33] Waitress, <https://github.com/Pylons/waitress>
- [34] 5Genesis Consortium, “D3.15 Experiment and Lifecycle Manager”, [Online]. Available:  
[http://5genesis.eu/wp-content/uploads/2019/10/5GENESIS\\_D3.15\\_v1.0.pdf](http://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.15_v1.0.pdf)
- [35] InfluxDB installation documentation,  
<https://docs.influxdata.com/influxdb/v1.7/introduction/installation/#installing-influxdb-oss>
- [36] InfluxDB configuration documentation,  
<https://docs.influxdata.com/influxdb/v1.7/administration/config/>
- [37] Prometheus configuration documentation,  
<https://prometheus.io/docs/prometheus/latest/configuration/configuration/>
- [38] Prometheus exporters, <https://prometheus.io/docs/instrumenting/exporters/>
- [39] Prometheus SNMP Exporter, [https://github.com/prometheus/snmp\\_exporter](https://github.com/prometheus/snmp_exporter)
- [40] Testplan example with 5Genesis result listeners,  
[https://gitlab.fokus.fraunhofer.de/Georgios.Xylouris/5genesis-integration/wikis/uploads/552ade83c9bf3c23bfa5db790ba13d0f/3 - External parameters.TapPlan](https://gitlab.fokus.fraunhofer.de/Georgios.Xylouris/5genesis-integration/wikis/uploads/552ade83c9bf3c23bfa5db790ba13d0f/3_-_External_parameters.TapPlan)
- [41] TapPlugin readme file, <https://gitlab.fokus.fraunhofer.de/5genesis/tap-plugins/blob/develop/README.md#tapplugins5genesisinfluxdb>
- [42] Docker, <https://docs.docker.com/install/>
- [43] Docker Compose, <https://docs.docker.com/compose/install/>
- [44] Slice Manager configuration files examples,  
[https://gitlab.fokus.fraunhofer.de/5genesis/slice-manager/tree/Release\\_A/Config-files\\_examples](https://gitlab.fokus.fraunhofer.de/5genesis/slice-manager/tree/Release_A/Config-files_examples)
- [45] Katana Slice Manager, [https://github.com/medianetlab/katana-slice\\_manager/wiki](https://github.com/medianetlab/katana-slice_manager/wiki)
- [46] 5Genesis Consortium, “D3.3 Slice Manager”, [Online]. Available:  
[https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS\\_D3.3\\_v1.0.pdf](https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.3_v1.0.pdf)
- [47] Virtualenv, <https://virtualenv.pypa.io/en/latest/>
- [48] TRIANGLE H2020 project, <https://www.triangle-project.eu/>
- [49] MONROE, <https://github.com/MONROE-PROJECT/monroe-experiment-core/tree/ReleaseA>
- [50] Open 5GENESIS Suite, <https://github.com/5genesis>

## ANNEX 1 – EXAMPLE TEMPLATE FOR GRAFANA REPORTER

The following is an example of a custom template for the Grafana reporter, which includes the 5GENESIS branding. Note that you need to specify the correct path for the 5GENESIS logo.

```
%use square brackets as goolang text templating delimiters
\documentclass{article}
\usepackage{graphicx}
\usepackage[margin=1in]{geometry}
\graphicspath{ {images/} }

\begin{document}
\title{
\includegraphics[scale=1.0]{<<PATH TO 5GENESIS LOGO>>}~\\
5 Genesis [[.Title]] [[if .VariableValues]] \\ \large [[.VariableValues]]
[[end]] [[if .Description]]
%\small [[.Description]] [[end]]}
\date{[[.FromFormatted]] to [[.ToFormatted]]}
\maketitle
\begin{center}
[[range .Panels]] [[if .IsSingleStat]] \begin{minipage}{0.3\textwidth}
\includegraphics[width=\textwidth]{image[[.Id]]}
\end{minipage}
[[else]] \par
\vspace{0.5cm}
\includegraphics[width=\textwidth]{image[[.Id]]}
\par
\vspace{0.5cm}
[[end]] [[end]]
\end{center}
\end{document}
```

## ANNEX 2 – PROMETHEUS TAP PLUGIN SOURCE CODE

---

### PrometheusInstrument.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;
using OpenTap;

using RestSharp;
using RestSharp.Extensions;
using System.Net;
using System.Security;

using System.IO;
using System.IO.Compression;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

namespace Tap.Plugins._5Genesis.Prometheus.Instruments
{
    [Display("Prometheus", Group: "5Genesis",
        Description: "Prometheus Instrument")]
    public class PrometheusInstrument : Instrument
    {
        private RestClient client = null;

        public static string TimeFormat = "yyyy-MM-ddTHH:mm:ss.fffZ";
        private static IFormatProvider cultureInfo =
            System.Globalization.CultureInfo.InvariantCulture;

        #region Settings

        [Display("Address", Group: "Prometheus", Order: 2.1,
            Description: "Prometheus HTTP API address")]
        public string Host { get; set; }

        [Display("Port", Group: "Prometheus", Order: 2.2,
            Description: "Prometheus HTTP API port")]
        public int Port { get; set; }

        #endregion

        public PrometheusInstrument()
        {
            Name = "PromQL";

            Host = "http://promgenesis.medianetlab.eu";
            Port = 80;

            Rules.Add(() => (!string.IsNullOrEmpty(Host)),
                "Please select an Address", "Host");
            Rules.Add(() => (Port > 0),
                "Please select a valid port number", "Port");
        }
    }
}
```

```
public override void Open()
{
    base.Open();

    this.client = new RestClient($"{Host}:{Port}/");
}

public override void Close()
{
    this.client = null;
    base.Close();
}

public PrometheusReply GetResults(string query,
                                   DateTime start, DateTime end,
                                   double step)
{
    RestRequest request =
        new RestRequest("/api/v1/query_range",
                        Method.GET, DataFormat.Json);
    request.AddParameter("query", query);
    request.AddParameter("start", start.ToString(TimeFormat));
    request.AddParameter("end", end.ToString(TimeFormat));
    request.AddParameter("step", $"{step.ToString(cultureInfo)}s");

    IRestResponse reply = client.Execute(request, Method.GET);

    PrometheusReply result = new PrometheusReply() {
        Status = reply.StatusCode,
        StatusDescription = reply.StatusDescription,
        Content = reply.Content
    };

    return result;
}
}
```

### PrometheusReply.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.IO.Compression;
using Newtonsoft.Json.Linq;
using OpenTap;
using Newtonsoft.Json;

namespace Tap.Plugins._5Genesis.Prometheus.Instruments
{
    public class PrometheusReply
    {
        public HttpStatusCode Status { get; set; }

        public string StatusDescription { get; set; }
    }
}
```



```
public string Message
{
    get
    {
        if (!string.IsNullOrEmpty(Content))
        {
            dynamic json =
                JsonConvert.DeserializeObject(Content);
            string status = json["status"].ToString();
            if (status == "error")
            {
                string errorType = json["errorType"];
                string message = json["error"];

                return $"Error: {errorType} - {message}";
            }
            else { return status; }
        }
        else { return "<Reply has no Content>"; }
    }
}

public string Content { get; set; }

public bool Success
{
    get { return ((int)Status >= 200) && ((int)Status <= 299); }
}

public IEnumerable<ResultTable> Results
{
    get
    {
        if (Success)
        {
            dynamic json =
                JsonConvert.DeserializeObject(Content);
            dynamic data = json["data"];
            dynamic resultsList = data["result"];
            foreach (dynamic result in resultsList)
            {
                yield return getResultTable(result);
            }
        }
    }
}

private ResultTable getResultTable(dynamic result)
{
    // Extract the available metadata from the "metric"
    // dictionary
    Dictionary<string, string> metadata =
        new Dictionary<string, string>();

    foreach (var entry in result["metric"])
    {
        metadata[entry.Name] = entry.Value.ToString();
    }

    // Extract "values".
    List<double> timestamps = new List<double>();
}
```

```

List<string> datetimes = new List<string>();
List<IConvertible> values = new List<IConvertible>();

foreach (var point in result["values"])
{
    double timestamp = double.Parse(point.First.ToString());
    DateTime datetime =
        DateTimeOffset.FromUnixTimeMilliseconds(
            (long) (timestamp * 1000)).DateTime;
    timestamps.Add(timestamp);
    datetimes.Add(
        datetime.ToString(PrometheusInstrument.TimeFormat));
    values.Add(this.toIConvertible(point.Last.ToString()));
}

// Create columns for UNIX timestamp, local datetime
// and value
string name = metadata.ContainsKey("__name__") ?
    metadata["__name__"] : "Prometheus result";
ResultColumn timestampColumn =
    new ResultColumn("Timestamp", timestamps.ToArray());
ResultColumn datetimesColumn =
    new ResultColumn("DateTime", datetimes.ToArray());
ResultColumn valuesColumn =
    new ResultColumn(name, values.ToArray());

// Create a column for each metadata value, repeated for
// every row
List<ResultColumn> resultColumns = new List<ResultColumn>();
foreach (var item in metadata)
{
    ResultColumn column =
        new ResultColumn(item.Key,
            Enumerable.Repeat(
                item.Value, timestamps.Count).ToArray());
    resultColumns.Add(column);
}

resultColumns.AddRange(new ResultColumn[] {
    timestampColumn, datetimesColumn, valuesColumn });

return new ResultTable(name, resultColumns.ToArray());
}

private IConvertible toIConvertible(string value)
{
    if (long.TryParse(value, out long parsedLong)) {
        return parsedLong; }
    if (double.TryParse(value, out double parsedDouble)) {
        return parsedDouble; }
    if (bool.TryParse(value, out bool parsedBool)) {
        return parsedBool; }
    return value;
}
}

```

PrometheusStep.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;

using OpenTap;
using Tap.Plugins._5Genesis.Prometheus.Instruments;
using Tap.Plugins._5Genesis.Misc.Extensions;

namespace Tap.Plugins._5Genesis.Prometheus.Steps
{
    [Display("Publish Prometheus results",
        Groups: new string[] { "5Genesis", "Prometheus" })]
    public class PublishStep : TestStep
    {
        public enum PeriodEnum { Relative, Absolute }

        #region Settings

        [Display("Instrument", Group: "Instrument", Order: 1.0)]
        public PrometheusInstrument Instrument { get; set; }

        [Display("Query", Group: "Request", Order: 2.0)]
        public string Query { get; set; }

        [Display("Period Mode", Group: "Request", Order: 2.1)]
        public PeriodEnum PeriodMode { get; set; }

        [Display("Past", Group: "Request", Order: 2.2)]
        [EnabledIf("PeriodMode", PeriodEnum.Relative,
            HideIfDisabled = true)]
        public TimeSpan RelativePeriod { get; set; }

        [Display("Start", Group: "Request", Order: 2.2)]
        [EnabledIf("PeriodMode", PeriodEnum.Absolute,
            HideIfDisabled = true)]
        public DateTime Start { get; set; }

        [Display("End", Group: "Request", Order: 2.3)]
        [EnabledIf("PeriodMode", PeriodEnum.Absolute,
            HideIfDisabled = true)]
        public DateTime End { get; set; }

        [Unit("s")]
        [Display("Step", Group: "Request", Order: 2.4)]
        public double Step { get; set; }

        [Display("Set Verdict on Error", Group: "Verdict", Order: 99.0,
            Description: "Set step verdict to the selected " +
                "value if Prometheus reply does not " +
                "indicate a success (2xx status code)")]
        public Enabled<Verdict> VerdictOnError { get; set; }

        #endregion

        public PublishStep()
        {
            Query = "collectd_enb_cpu_vcpu{enb_cpu=\"cpu\", \" +
                \"exported_instance=\"10.2.1.10\"}";
            PeriodMode = PeriodEnum.Relative;
            RelativePeriod = new TimeSpan(0, 15, 0);
        }
    }
}
```

```
        Start = DateTime.UtcNow.AddMinutes(-15);
        End = DateTime.UtcNow;
        Step = 5.0;
        VerdictOnError = new Enabled<Verdict>() {
            IsEnabled = false, Value = Verdict.Error };
    }

    public override void Run()
    {
        DateTime start = (PeriodMode == PeriodEnum.Absolute) ?
            Start : DateTime.UtcNow - RelativePeriod;
        DateTime end = (PeriodMode == PeriodEnum.Absolute) ?
            End : DateTime.UtcNow;

        PrometheusReply reply =
            Instrument.GetResults(Query, start, end, Step);

        if (reply.Success)
        {
            bool hasResults = false;

            foreach (ResultTable resultTable in reply.Results)
            {
                resultTable.PublishToSource(Results);

                long numResults =
                    resultTable.Columns.First().Data.LongLength;
                Log.Info($"Published {numResults} results of" +
                    $" type {resultTable.Name}");

                if (numResults > 0) { hasResults = true; }
            }

            if (!hasResults) {
                Log.Warning("No results have been retrieved."); }
        }
        else
        {
            Log.Error($"Request to Prometheus failed: " +
                $"{reply.StatusDescription} ({reply.Status})");
            Log.Error($" {reply.Message}");
            if (VerdictOnError.IsEnabled)
            {
                UpgradeVerdict(VerdictOnError.Value);
            }
        }
    }
}
```

## ANNEX 3 – ANDROID ADB AGENTS TESTPLAN EXAMPLE

```
<?xml version="1.0" encoding="utf-8"?>
<TestPlan type="OpenTap.TestPlan" Locked="false">
  <Steps>
    <TestStep type="Tap.Plugins.UMA.AdbAgents.Steps.AdbResourceAgentStep"
Version="1.0.0" Id="11023f09-6ab8-46fc-a2a6-2d910e37f495">
      <Instrument />
      <Action>Measure</Action>
      <LogcatThreshold>15</LogcatThreshold>
      <MeasurementMode>Children</MeasurementMode>
      <MeasurementTime>10</MeasurementTime>
      <Enabled>true</Enabled>
      <Name>Adb Resource Agent</Name>
      <ChildTestSteps>
        <TestStep type="Tap.Plugins.UMA.AdbAgents.Steps.AdbPingAgentStep"
Version="1.0.0" Id="1418ba76-ca0e-44f5-a89a-215003a5c5c2">
          <Instrument
Source="OpenTap.InstrumentSettings">ADB_Ping</Instrument>
          <Target>www.google.com</Target>
          <Ttl>128</Ttl>
          <Action>Measure</Action>
          <LogcatThreshold>15</LogcatThreshold>
          <MeasurementMode>Time</MeasurementMode>
          <MeasurementTime>10</MeasurementTime>
          <Enabled>true</Enabled>
          <Name>Adb Ping Agent</Name>
          <ChildTestSteps />
        </TestStep>
      </ChildTestSteps>
    </TestStep>
  </Steps>
  <Package.Dependencies>
    <Package Name="OpenTAP" Version="9.3.0+907ad9be" />
    <Package Name="SDK" Version="9.3.0+907ad9be" />
    <Package Name="UMA.AdbAgents" Version="1.1.2" />
  </Package.Dependencies>
</TestPlan>
```

## ANNEX 4 – JSON SCHEMA OF THE SLICE MANAGER SOUTHBOUND MESSAGES

This section presents the JSON Schemas of the data that the Slice Manager generates for the EMS and the WIM during the Slice Creation phase. These data describe the slice parameters. Each plugin must be able to translate the data to component-specific messages.

### WIM Data JSON Schema

```
{
  $schema: "http://json-schema.org/draft-07/schema#",
  type: "object",
  description: "Schema of the message from Katana to WIM",
  properties: {
    slice_sla: {
      type: "object",
      description: "Slice parameteres as defiend in NEST",
      properties: {
        network_DL_throughput: {
          type: "object",
          description: "The achievable data rate in downlink for the whole network slice (and not per user).",
          properties: {
            guaranteed: {
              type: "number",
              description: "kbps"
            },
            maximum: {
              type: "number",
              description: "kbps"
            }
          },
        },
        network_UL_throughput: {
          type: "object",
          description: "The achievable data rate in uplink for the whole network slice (and not per user).",
          properties: {
            guaranteed: {
              type: "number",
              description: "kbps"
            },
            maximum: {
              type: "number",
              description: "kbps"
            }
          },
        },
        mtu: {
          type: "number",
          description: "Bytes"
        }
      },
    },
  },
}
```

```
core_connections: {
  type: "array",
  description: "List of connections that are part of the slice and must be
  implemented by the WIM",
  items: {
    type: "object",
    description: "The endpoints of the connections",
    properties: {
      core: {
        type: "object",
        description: "The core part of the radio connection",
        properties: {
          ns: {
            type: "array",
            description: "A list of VIMs where the NSs have been instantiated",
            items: {
              type: "object",
              description: "A VIM hosting NSs",
              properties: {
                location: {
                  type: "string",
                  description: "The location of the VIM"
                },
                vim: {
                  type: "string",
                  description: "The ID of the VIM"
                }
              }
            },
          },
          pnf: {
            type: "array",
            description: "A list of the PNFs that are part of the slice",
            items: {
              type: "object",
              description: "A Physical Network Service",
              properties: {
                pnf-id: {
                  type: "string",
                  description: "A Unique ID of the pnf"
                },
                pnf-name: {
                  type: "string",
                  description: "The name of the PNF"
                },
                description: {
                  type: "string"
                },
                ip: {
                  type: "string",
                  description: "The management IP of the PNF"
                },
                location: {
                  type: "string",
                  description: "The location of the PNF"
                },
                optional: {
                  type: "boolean"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
}
}
},
radio: {
  type: "object",
  description: "The core part of the radio connection",
  properties: {
    ns: {
      type: "array",
      description: "A list of VIMs where the NSs have been instantiated",
      items: {
        type: "object",
        description: "A VIM hosting NSs",
        properties: {
          location: {
            type: "string",
            description: "The location of the VIM"
          },
          vim: {
            type: "string",
            description: "The ID of the VIM"
          }
        }
      },
    },
    pnf: {
      type: "array",
      description: "A list of the PNFs that are part of the slice",
      items: {
        type: "object",
        description: "A Physical Network Service",
        properties: {
          pnf-id: {
            type: "string",
            description: "A Unique ID of the pnf"
          },
          pnf-name: {
            type: "string",
            description: "The name of the PNF"
          },
          description: {
            type: "string"
          },
          ip: {
            type: "string",
            description: "The management IP of the PNF"
          },
          location: {
            type: "string",
            description: "The location of the PNF"
          },
          optional: {
            type: "boolean"
          }
        }
      },
    },
  },
},
},
},
},
},
}
```



```
},
extra_ns: {
  type: "array",
  description: "A list of VIMs where the NSs that are not part of the core
slice have been instantiated",
  items: {
    type: "object",
    description: "A VIM hosting NSs",
    properties: {
      location: {
        type: "string",
        description: "The location of the VIM"
      },
    },
    vim: {
      type: "string",
      description: "The ID of the VIM"
    }
  }
}
}
```

## EMS Data JSON Schema

```
{
  $schema: "http://json-schema.org/draft-07/schema#",
  definitions: {
    ns_list: {
      type: "array",
      description: "A list of the NSs on which the EMS must run D1 & D2
configuration",
      items: {
        type: "object",
        description: "A NS",
        properties: {
          name: {
            type: "string",
            description: "The name of the Network Service"
          },
          location: {
            type: "string",
            description: "The location of the Network Service"
          },
        },
        vnf_list: {
          type: "array",
          description: "A list of VNFs that compose the NS",
          items: {
            type: "object",
            description: "A VNF",
            properties: {
              vnf_name: {
                type: "string",
                description: "The name of the VNF"
              },
            },
            mgmt_ip: {
              type: "string",
              description: "The management IP of the VNF of the VNF"
            },
          },
        },
      },
    },
  },
}
```

```
vdu_IP_list: {
  type: "array",
  description: "The list of the VDUs that compose the VNF",
  items: {
    type: "string",
    description: "A VDU IP"
  }
}
},
pnf_list: {
  type: "array",
  description: "A list of the PNFs on which the EMS must run D1 & D2 configuration",
  items: {
    type: "object",
    description: "A Physical Network Service",
    properties: {
      pnf-id: {
        type: "string",
        description: "A Unique ID of the pnf"
      },
      pnf-name: {
        type: "string",
        description: "The name of the PNF"
      },
      description: {
        type: "string"
      },
      ip: {
        type: "string",
        description: "The management IP of the PNF"
      },
      location: {
        type: "string",
        description: "The location of the PNF"
      },
      optional: {
        type: "boolean"
      }
    }
  },
  slice_sla: {
    type: "object",
    description: "Slice Parameters for NEST",
    properties: {
      ue_DL_throughput: {
        type: "object",
        description: "This attribute describes the guaranteed data rate supported by the network slice per UE in downlink",
        properties: {
          guaranteed: {
            type: "number",
            description: "kbps"
          },
          maximum: {
```

```
type: "number",
description: "kbps"
}
},
ue_UL_throughput: {
type: "object",
description: "This attribute describes the guaranteed data rate supported by
the network slice per UE in uplink",
properties: {
guaranteed: {
type: "number",
description: "kbps"
},
maximum: {
type: "number",
description: "kbps"
}
}
},
group_communication_support: {
type: "number",
enum: [
0,
1,
2,
3
],
description: "0: not available 1: Single Cell Point to Multipoint (SCPTM) 2:
Broadcast/Multicast 3: Broadcast/Multicast + SC-PTM"
},
number_of_terminals: {
type: "number",
description: "This attribute describes the maximum number of concurrent
terminals supported by the network slice."
},
positional_support: {
type: "object",
description: "This attribute describes if the network slice provides geo-
localization methods or supporting methods.",
properties: {
availability: {
type: "array",
description: "Describes if this attribute is provided by the network slice
and contains a list of positioning methods provided by the slice.",
items: {
type: "number",
enum: [
1,
2,
3,
4,
5,
6,
7
],
description: "1: CID 2: E-CID (LTE and NR) 3: OTDOA (LTE and NR) 4: RF
fingerprinting 5: AECID 6: Hybrid positioning 7: NET-RTK"
}
}
},
frequency: {
```

```
type: "number",
description: "Seconds"
},
accuracy: {
type: "number",
description: "Meters"
}
},
radio_spectrum: {
type: "array",
description: "Defines the radio spectrum supported by the network slice.",
items: {
type: "string",
description: "This attribute simply tells which frequencies can be used to
access the network slice. Example: n1, n77, n38"
}
},
device_velocity: {
type: "number",
enum: [
1,
2,
3,
4
],
description: "1: Stationary: 0 km/h 2: Pedestrian: 0 km/h to 10 km/h 3:
Vehicular: 10 km/h to 120 km/h 4: High speed vehicular: 120 km/h to 500 km/h"
},
terminal_density: {
type: "number",
description: "Maximum number of connected and/or accessible devices per unit
area (per km2) supported by the network slice [Number/km^2]"
}
}
},
type: "object",
description: "Schema of the message from Katana to EMS",
properties: {
core: {
type: "object",
description: "The Core part of the Radio Service",
properties: {
ns: {
$ref: "#/definitions/ns_list"
},
pnf: {
$ref: "#/definitions/pnf_list"
}
},
radio: {
type: "object",
description: "The Radio part of the Radio Service",
properties: {
ns: {
$ref: "#/definitions/ns_list"
},
pnf: {
$ref: "#/definitions/pnf_list"
}
```

```
}  
}  
},  
slice_sla: {  
$ref: "#/definitions/slice_sla"  
}  
}  
}
```

# ANNEX 5 — SOUTHBOUND COMPONENTS

## CONFIGURATION FILES SCHEMAS

---

### VIM

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "description": "A new VIM",
  "properties": {
    "id": {
      "type": "string",
      "description": "Unique id"
    },
    "name": {
      "type": "string",
      "description": "The name for the new VIM"
    },
    "auth_url": {
      "type": "string",
      "description": "VIM's authentication URL - example:
http://10.200.64.2:5000/v3/"
    },
    "username": {
      "type": "string",
      "description": "The admin username"
    },
    "password": {
      "type": "string",
      "description": "The admin password"
    },
    "admin_project_name": {
      "type": "string",
      "description": "The admin project"
    },
    "location": {
      "type": "string",
      "description": "VIM's location"
    },
    "type": {
      "type": "string",
      "description": "VIM's type"
    },
    "version": {
      "type": "string",
      "description": "The version of the VIM's OS"
    },
    "description": {
      "type": "string",
      "description": "A description for the VIM"
    },
    "infrastructure_monitoring": {
      "type": "string",
      "description": "Optional - The URL of the Prometheus system that is
responsible for monitoring the VIM"
    }
  }
}
```

```

    "config": {
      "type": "object",
      "description": "Optional parameters regarding the VIM operation -
example: Security group"
    }
  },
  "required": [
    "id",
    "auth_url",
    "username",
    "password",
    "admin_project_name"
  ]
}

```

## NFVO

```

{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "description": "A new NFVO",
  "properties": {
    "id": {
      "type": "string",
      "description": "Unique id"
    },
    "name": {
      "type": "string",
      "description": "The name for the new NFVO"
    },
    "nfvoip": {
      "type": "string",
      "description": "NFVO's authentication URL - example:
http://10.200.64.2:5000/v3/"
    },
    "nfvousername": {
      "type": "string",
      "description": "The admin username"
    },
    "nfvopassword": {
      "type": "string",
      "description": "The admin password"
    },
    "tenantname": {
      "type": "string",
      "description": "NFVO's Tenant name"
    },
    "type": {
      "type": "string",
      "description": "NFVO's type"
    },
    "version": {
      "type": "string",
      "description": "The version of the NFVO's OS"
    },
    "description": {
      "type": "string",
      "description": "A description for the NFVO"
    }
  }
}

```

```

    },
    "config": {
      "type": "object",
      "description": "Optional parameters regarding the NFVO operation -
example: network: flat"
    }
  },
  "required": ["id", "nfvusername", "nfvopassword", "nfvoip",
"tenantname"]
}

```

## WIM

```

{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "description": "A new WIM",
  "properties": {
    "id": {
      "type": "string",
      "description": "Unique id"
    },
    "name": {
      "type": "string",
      "description": "The name for the new WIM"
    },
    "description": {
      "type": "string",
      "description": "A description for the WIM"
    },
    "url": {
      "type": "string",
      "description": "WIM's authentication URL - example:
http://10.200.64.2:5000/"
    },
    "type": {
      "type": "string",
      "description": "WIM's type"
    },
    "monitoring-url": {
      "type": "string",
      "description": "If set, katana-prometheus will scrape the target URL"
    }
  },
  "required": [
    "id",
    "url",
    "type"
  ]
}

```

## EMS

```

{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "description": "A new EMS",

```



```
"properties": {
  "id": {
    "type": "string",
    "description": "Unique id"
  },
  "name": {
    "type": "string",
    "description": "The name for the new EMS"
  },
  "description": {
    "type": "string",
    "description": "A description for the EMS"
  },
  "url": {
    "type": "string",
    "description": "EMS' authentication URL - example:
http://10.200.64.2:5000/"
  },
  "type": {
    "type": "string",
    "description": "EMS' type"
  }
},
"required": ["id", "url", "type"]
}
```

## ANNEX 6 –NEST JSON SCHEMA

```
{  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "base_slice_descriptor":{
      "type": "object",
      "description": "This is the schema for the core \
part of the new slice",
      "properties": {
        "base_slice_des_id": {
          "type": "string",
          "description": "Id of the slice descriptor \
which will be used as base for the gst"
        },
        "base_slice_des_ref": {
          "type": "string",
          "description": "Optional - Reference to an added \
slice descriptor which will be used as base for the gst"
        },
        "coverage": {
          "type": "array",
          "description": "A list with all the locations that \
are part of the slice",
          "items":{
            "type": "string",
            "description": " The location name for each site in the slice"
          }
        },
        "delay_tolerance": {
          "type": "boolean",
          "description": "Supported or not supported"
        },
        "deterministic_communication":{
          "type": "object",
          "description": "This attribute defines if the \
network slice supports deterministic communication \
for periodic user traffic. Periodic traffic refers \
to the type of traffic with periodic transmissions.",
          "properties":{
            "availability": {
              "type": "boolean",
              "description": "This parameter describes if the \
network slice supports deterministic communication."
            },
            "periodicity": {
              "type": "array",
              "description": "This parameter provides a \
list of periodicities supported by the network slice.",
              "items": {
                "type": "number",
                "description": "Seconds"
              }
            }
          }
        },
        "network_DL_throughput": {
          "type": "object",

```

```
"description": "The achievable data rate in downlink \
  for the whole network slice (and not per user).",
"properties": {
  "guaranteed": {
    "type": "number",
    "description": "kbps"
  },
  "maximum": {
    "type": "number",
    "description": "kbps"
  }
},
},
"ue_DL_throughput": {
  "type": "object",
  "description": "This attribute describes the \
    guaranteed data rate supported by the network slice \
    per UE in downlink",
  "properties": {
    "guaranteed": {
      "type": "number",
      "description": "kbps"
    },
    "maximum": {
      "type": "number",
      "description": "kbps"
    }
  }
},
},
"network_UL_throughput": {
  "type": "object",
  "description": "The achievable data rate in uplink \
    for the whole network slice (and not per user).",
  "properties": {
    "guaranteed": {
      "type": "number",
      "description": "kbps"
    },
    "maximum": {
      "type": "number",
      "description": "kbps"
    }
  }
},
},
"ue_UL_throughput": {
  "type": "object",
  "description": "This attribute describes the \
    guaranteed data rate supported by the network slice \
    per UE in uplink",
  "properties": {
    "guaranteed": {
      "type": "number",
      "description": "kbps"
    },
    "maximum": {
      "type": "number",
      "description": "kbps"
    }
  }
},
},
"group_communication_support": {
```

```

    "type": "number",
    "enum": [0, 1, 2, 3],
    "description": "0: not available 1: Single Cell \
    Point to Multipoint (SCPTM) 2: Broadcast/Multicast \
    3: Broadcast/Multicast + SC-PTM"
  },
  "isolation_level": {
    "type": "object",
    "description": "A network slice instance may be fully or \
    partly, logically and/or physically, isolated from \
    another network slice instance",
    "properties": {
      "isolation": {
        "type": "number",
        "enum": [0, 1, 2, 3],
        "description": "0: No Isolation 1: Physical Isolation \
        2: Logical Isolation 3: Both Isolation"
      }
    }
  },
  "mtu": {
    "type": "number",
    "description": "Bytes"
  },
  "mission_critical_support": {
    "type": "object",
    "description": "Mission-critical (MC) leads to a priority of \
    the network slice relative to others, for C-plane and \
    U-plane decisions.",
    "properties": {
      "availability": {
        "type": "boolean"
      },
      "mc_service": {
        "type": "array",
        "description": "This attribute specifies whether or \
        not the network slice supports MC push-to-talk, MC data, \
        MC video, Isolated E-UTRAN Operation for Public Safety \
        or MC interworking.",
        "items": {
          "type": "number",
          "enum": [1, 2, 3, 4, 5],
          "description": "1: MCPTT 2: MCData 3: MCVideo 4: IOPS \
          5: MC interworking"
        }
      }
    }
  },
  "mmtel_support": {
    "type": "boolean",
    "description": "This attribute describes whether the \
    network slice supports IP Multimedia Subsystem (IMS) \
    and Multimedia Telephony Service MMTel."
  },
  "nb_iiot": {
    "type": "boolean",
    "description": "This parameter describes whether NB-IoT \
    is supported in the network slice."
  },
  "number_of_connections": {
    "type": "number",

```

```

    "description": "This attribute describes the maximum number \
    of concurrent sessions supported by the network slice."
  },
  "number_of_terminals": {
    "type": "number",
    "description": "This attribute describes the maximum number \
    of concurrent terminals supported by the network slice."
  },
  "positional_support": {
    "type": "object",
    "description": "This attribute describes if the network \
    slice provides geo-localization methods or supporting methods.",
    "properties": {
      "availability": {
        "type": "array",
        "description": "Describes if this attribute is provided \
        by the network slice and contains a list of \
        positioning methods provided by the slice.",
        "items": {
          "type": "number",
          "enum": [1, 2, 3, 4, 5, 6, 7],
          "description": "1: CID 2: E-CID (LTE and NR) 3: OTDOA \
          (LTE and NR) 4: RF fingerprinting 5: AECID 6: \
          Hybrid positioning 7: NET-RTK"
        }
      },
      "frequency": {
        "type": "number",
        "description": "Seconds"
      },
      "accuracy": {
        "type": "number",
        "description": "Meters"
      }
    }
  },
  "radio_spectrum": {
    "type": "array",
    "description": "Defines the radio spectrum supported \
    by the network slice.",
    "items": {
      "type": "string",
      "description": "This attribute simply tells which \
      frequencies can be used to access the network slice. \
      Example: n1, n77, n38"
    }
  },
  "simultaneous_nsi": {
    "type": "number",
    "enum": [0, 1, 2, 3, 4, 5],
    "description": "0: Can be used with any network slice \
    1: Can be used with network slices with same SST value \
    2: Can be used with any network slice with same SD value \
    3: Cannot be used with another network slice \
    4-15: operator defined class"
  },
  "qos": {
    "type": "array",
    "description": "This attribute defines all the QoS \
    relevant parameters supported by the network slice, based \
    on 3GPP defined standard values (5QIs)",

```

```

        "items": {
            "type": "object",
            "description": "Refers to 5QI defined in \
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144",
            "properties": {
                "qi": {
                    "type": "number",
                    "description": "Based on the table of defined 5QI by 3GPP"
                },
                "resource_type": {
                    "type": "number",
                    "enum": [0, 1, 2],
                    "description": "0: GBR (Mission Critical Video user \
plane) 1: Delay critical GBR (Intelligent \
Transport Systems) 2: Non-GBR (Voice, AR)"
                },
                "priority_level": {
                    "type": "number",
                    "description": "Associated with 5G QoS \
characteristics indicates a priority in \
scheduling resources among QoS Flows."
                },
                "packet_delay_budget": {
                    "type": "number",
                    "description": "The Packet Delay Budget (PDB) defines \
an upper bound for the time that a packet may be \
delayed between the UE and the UPF [Seconds]."
                },
                "packet_error_rate": {
                    "type": "number",
                    "description": "The Packet Error Rate (PER) defines \
an upper bound for the rate of packets that are \
not successfully delivered by the corresponding \
receiver [percentage]."
                },
                "jitter": {
                    "type": "number",
                    "description": "Jitter is defined as a variation in \
the delay of received packets [Seconds]."
                },
                "max_packet_loss_rate": {
                    "type": "number",
                    "description": " the maximum rate for lost packets of \
the QoS flow that can be tolerated in the uplink (UL) \
and downlink (DL) direction [percentage]."
                }
            }
        },
        "nonIP_traffic": {
            "type": "boolean"
        },
        "device_velocity": {
            "type": "number",
            "enum": [1, 2, 3, 4],
            "description": "1: Stationary: 0 km/h 2: Pedestrian: 0 km/h to \
10 km/h 3: Vehicular: 10 km/h to 120 km/h 4: High \
speed vehicular: 120 km/h to 500 km/h"
        },
        "terminal_density": {

```

```

        "type": "number",
        "description": "maximum number of connected and/or \
        accessible devices per unit area (per km2) supported by \
        the network slice [Number/km^2]"
    }
}
},
"service_descriptor":{
    "type": "object",
    "description": "This is the schema for the Service Descriptor part \
    of slice IM",
    "properties": {
        "ns_list": {
            "type": "array",
            "description": "List of the NSD to be instantiated alongside \
            the slice",
            "items": {
                "type": "object",
                "description": "A NS",
                "properties":{
                    "nfvo-id":{
                        "type": "string",
                        "description": "The NFVO that will manage the life \
                        cycle of the NS"
                    },
                    "nsd-id": {
                        "type": "string",
                        "description": "The NSD id as defined on the NFVO"
                    },
                    "ns-name": {
                        "type": "string",
                        "description": "The name of the NS"
                    },
                    "placement": {
                        "type": "number",
                        "enum": [0, 1],
                        "description": "1: Core, 2: Edge"
                    }
                }
            }
        }
    }
},
"test_descriptor":{
    "type": "object",
    "description": "This is the schema for the Test Descriptor \
    part of slice IM",
    "properties": {
        "probe_list": {
            "type": "array",
            "description": "A list of probe ids to be included in the slice",
            "items": {
                "type": "string"
            }
        }
    },
    "performance_monitoring": {
        "type": "object",
        "description": "This attribute provides the capability to \
        monitor KQIs and KPIs.",
        "properties": {
            "availability": {

```

```

        "type": "array",
        "description": "List of KQIs and KPIs available \
for monitoring",
        "items": {
            "type": "number",
            "enum": [1, 2, 3],
            "description": "1: Throughput 2: Latency 3: \
Service Request Success Rate"
        }
    },
    "frequency": {
        "type": "number",
        "description": "Seconds"
    }
},
"performance_prediction": {
    "type": "object",
    "description": "This attribute provides the capability to \
predict KQIs and KPIs.",
    "properties": {
        "availability": {
            "type": "array",
            "description": "List of KQIs and KPIs available \
for monitoring",
            "items": {
                "type": "number",
                "enum": [1, 2, 3],
                "description": "1: Throughput 2: Latency 3: Service Request
Success Rate"
            }
        },
        "frequency": {
            "type": "number",
            "description": "Seconds"
        }
    }
}
},
{
    "type": "object",
    "properties": {
        "base_slice_descriptor": { "$ref": "#/definitions/base_slice_descriptor" },
        "service_descriptor": { "$ref": "#/definitions/service_descriptor" },
        "test_descriptor": { "$ref": "#/definitions/test_descriptor" }
    },
    "required": ["base_slice_descriptor"]
}

```