



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

[H2020 - Grant Agreement No. 815178]

Deliverable D5.3

Documentation and supporting material for 5G stakeholders (Release A)

Editor 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 June 30th, 2020

Distribution PUBLIC (PU)



List of Authors

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

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 © 2020 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 D5.3	A. Díaz Zayas (UMA)	26/06/2020

LIST OF ACRONYMS

Acronym	Meaning
3GPP	3 rd 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 ^o 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	20
Figure 2 Deployed 5GENESIS platforms	24
Figure 3 The 5GENESIS Experimentation Workflow	28
Figure 4 Network Service Onboarding flow diagram.....	33
Figure 5 Experimentation via de Portal	37
Figure 6 5GENESIS Portal registration screen	38
Figure 7 5GENESIS Portal sign in screen.....	38
Figure 8 5GENESIS Portal info page	39
Figure 9 Experiment creation screen	40
Figure 10 User's dashboard	40
Figure 11 Experiment executions screen	41
Figure 12 Experiment execution log screen	42
Figure 13 Experiment result visualization screen.....	42
Figure 14 Generated PDF results report.....	43
Figure 15 Experimentation via the Open APIs	44
Figure 16 Integration of new components to the 5GENESIS Architecture	47
Figure 17 Prometheus instrument	49
Figure 18 Slice Manager Building Blocks.....	57
Figure 19 Slice Manager Source Code Directory.....	60
Figure 20 5GENESIS related OpenTap instruments for NFV MANO (OSM)	73
Figure 21 OSM Instrument	74
Figure 22 VIM Instrument.....	75
Figure 23 OSM Test Steps	77
Figure 24 Class hierarchy of the OpenTap OSM Test Steps.....	78
Figure 25 Class hierarchy of the generated OSM API accessors from its OpenAPI definition.....	78
Figure 26 InfluxDB result listener configuration.....	80
Figure 27 Ping application interface	82
Figure 28 iPerf application interface	83
Figure 29 Resource Agent application interface	85
Figure 30 Remote PC agents TAP instrument and configuration	87
Figure 31 Remote PC agents TAP steps	87

List of Tables

Table 1 Document interdependencies.....	16
Table 2 5GENESIS Release A. Summary of functionalities	21
Table 3 5GENESIS Experimentation Roles and Stakeholders	22
Table 4 5GENESIS Athens Platform	24
Table 5 5GENESIS Málaga Platform	25
Table 6 5GENESIS Limassol Platform	26
Table 7 5GENESIS Surrey Platform	26
Table 8 5GENESIS Berlin Platform	27
Table 9 Example of the report for the MCPTT test case	31
Table 10 List of test cases available in Release A	36
Table 11 Operations between Slice Manager and southbound components	57

Executive Summary

Overall, deliverable D5.3 is a self-contained document serving as a general handbook for all the features supported by Release A of the 5GENESIS Experimentation Framework, which has been also opensourced 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 and its deployment.

This documentation includes the kind of interactions an external entity needs to establish to run experiments over a 5GENESIS platform and how 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 provided a manual for testbed operators interested on adopting the 5GENESIS experimentation framework.

Table of Contents

LIST OF ACRONYMS	6
1 INTRODUCTION	15
1.1 Purpose of the Document	15
1.2 Structure of the Document	17
1.3 Target Audience	17
2 THE 5GENESIS ARCHITECTURE	19
2.1 Concepts and Architecture	19
2.2 Main features of Release A	21
2.3 Involved stakeholders and experimentation roles	22
2.4 The 5GENESIS platforms	24
2.5 The 5GENESIS experimentation workflow	28
2.5.1 Experiment consultation phase	28
2.5.2 Experiment provisioning phase	29
2.5.3 Experiment execution phase	30
2.5.4 Experiment decommissioning phase	31
2.6 What kind of experiment can be executed?	32
2.6.1 Testing network services	32
2.6.2 Testing user equipment	33
3 DOCUMENTATION FOR EXPERIMENTERS.....	35
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	40
3.2.4 Visualization of results	41
3.3 Experimentation via the Open APIs	43
3.3.1 Description of the Experiment Descriptor Template	44
3.3.2 Sending execution requests to the ELCM	45
3.3.3 Accessing results and logs	45
3.3.3.1 Grafana dashboards	46
3.3.3.2 Experiment logs.....	46

4	DOCUMENTATION FOR TECHNOLOGY PROVIDERS	47
4.1	Introduction to the development of 5GENESIS plugins	48
4.1.1	Description of the southbound interface of ELCM	48
4.1.1.1	TAP plugins for the ELCM	49
4.1.1.1.1	Examples: TAP plugins example	49
4.1.1.2	Python extensions for the ELCM	55
4.1.1.2.1	Example: CompressFiles task	56
4.1.2	Description of the southbound interface of the Slice Manager	57
4.1.2.1	Python plugins for the slice manager	58
4.1.2.1.1	Example: Amarisoft plugin	60
5	DOCUMENTATION FOR PLATFORM OPERATORS.....	63
5.1	Coordination Layer deployment	63
5.1.1	Portal	63
5.1.1.1	Installation.....	63
5.1.1.2	Deployment.....	64
5.1.1.3	Configuration.....	64
5.1.2	ELCM.....	66
5.1.2.1	Installation.....	66
5.1.2.2	Deployment.....	67
5.1.2.3	Configuration.....	67
5.1.2.4	Facility Configuration.....	69
5.1.2.5	PDF Report generation	72
5.2	OSM Plugin.....	72
5.2.1	OpenTap OSM Instruments	73
1.1.1.1	OSM Instrument	73
1.1.1.2	VIM Instrument.....	75
5.2.2	OpenTap OSM Test Steps	76
5.3	Monitoring and Analytics Deployment	79
5.3.1	Result management.....	79
5.3.1.1	InfluxDB	79
5.3.1.2	Result listener.....	80
5.3.2	Prometheus Infrastructure Monitoring	80
5.3.3	Performance Monitoring	81
5.3.3.1	ADB Monitoring agents	81
5.3.3.1.1	Ping agent	81

5.3.3.1.2	iPerf agent.....	83
5.3.3.1.3	Resource agent	84
5.3.3.1.4	Remote PC Agents	85
5.3.3.2	MonroeVN.....	87
5.3.4	Analytics.....	90
5.4	Slice Manager Deployment.....	91
REFERENCES.....		94
ANNEX 1 – EXAMPLE TEMPLATE FOR GRAFANA REPORTER.....		96
ANNEX 2 – PROMETHEUS TAP PLUGIN SOURCE CODE.....		97
ANNEX 3 – ANDROID ADB AGENTS TESTPLAN EXAMPLE		103
ANNEX 4 – JSON SCHEMA OF THE SLICE MANAGER SOUTHBOUND MESSAGES.....		104
	WIM Data JSON Schema	104
	EMS Data JSON Schema.....	107
ANNEX 5 – SOUTHBOUND COMPONENTS CONFIGURATION FILES SCHEMAS		112
	VIM	112
	NFVO	113
	WIM	114
	EMS	114

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.

For the Release A, the consortium took the strategic decision to open source the 5GENESIS experimentation layer, launching the “Open 5GENESIS Suite” open source project at GitHub (Link), where all the components of the 5GENESIS experimentation layers (Rel. A) have been opensourced under Apache license.

This document considers the first release of the 5GENESIS platforms (Release A)/[Open 5GENESIS Suite](#) and will be upgraded by the final deliverable D5.4 “Documentation and supporting material for 5G stakeholders (Release B)” due in the third and last year of the project lifetime. The whole development towards the Rel. B will be performed online at the public GitHub repository. The latter document shall contain the full and final documentation with enhanced functionalities and improved automation, as will be targeted for the Release B of the platforms. 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.

To provide an 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.
D2.2 [3]	5GENESIS Overall Facility Design and Specifications	Presents the 5GENESIS Facility architecture and lists the functional components to be deployed in each testbed.
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.
D3.1 [5]	Management and orchestration (Release A)	Describes the implementation of the MANO solutions that are integrated in the infrastructure, together with the relevant interfaces and deployment options.
D3.3 [6]	Slice management WP3 (Release A)	Describes the implementation of the Slice Manager solution, and its interfaces towards the MANO and NMS components.
D3.5 [7]	Monitoring and WP3 analytics (Release A)	Describes the implementation of Infrastructure and Performance Monitoring components, as well as of Analytics tools, including the interfaces with infrastructure elements (Release A).
D3.9 [8]	5G Core Network WP3 Functions (Release A)	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.
D4.2 [11]	The Athens Platform	Details the Athens platform in 5GENESIS Release A (3/2020).
D4.5 [12]	The Malaga Platform	Details the Malaga platform in 5GENESIS Release A (3/2020).
D4.8 [13]	The Limassol Platform	Details the Limassol platform in 5GENESIS Release A (3/2020).
D4.11 [14]	The Surrey Platform	Details the Surrey platform in 5GENESIS Release A (3/2020).
D4.14 [15]	The Berlin Platform	Details the Berlin platform in 5GENESIS Release A (3/2020).

1.2 Structure of the Document

The topics of deliverable D5.3 are presented with the following structure:

- **Section 1** ‘Introduction’ introduces the document and presents the purpose, the structure and the target audience of D5.3.
- **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. Interactions through the OpenAPI, which offers a systemic and automated interface unlike 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 3rd 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.
- **Section 6** concludes the document.
- 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 that will be appreciated by engineering teams that consider engaging with 5G technologies from various perspectives. The content presented is mainly addressed to:

- Project 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 A of the 5GENESIS components and platforms (WP3 and WP4 related work, respectively).

In this section, we briefly introduce the 5GENESIS architecture, described in D2.2 [3], to provide a better understanding on how the architecture reflects the interaction with the respective actors: Experimenters and related stakeholders. The architecture is being updated as the project progresses and its later version is discussed in deliverable D2.4 “Final report on facility design and experimentation planning”, due at the end of the second year of the project lifetime. Although no fundamental changes are expected, the improvements introduced will be also documented in deliverable D5.4 “Documentation and supporting material for 5G stakeholders (Release B)”.

The 5GENESIS architecture, depicted in Figure 2, 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, the 5GENESIS Portal and the Open APIs. Via the Portal, the Experimenter can be authenticated, on-board 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 experiment lifecycle stages sequencing, by maintaining the experiment status and providing feedbacks.

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 the 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 exploited 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.

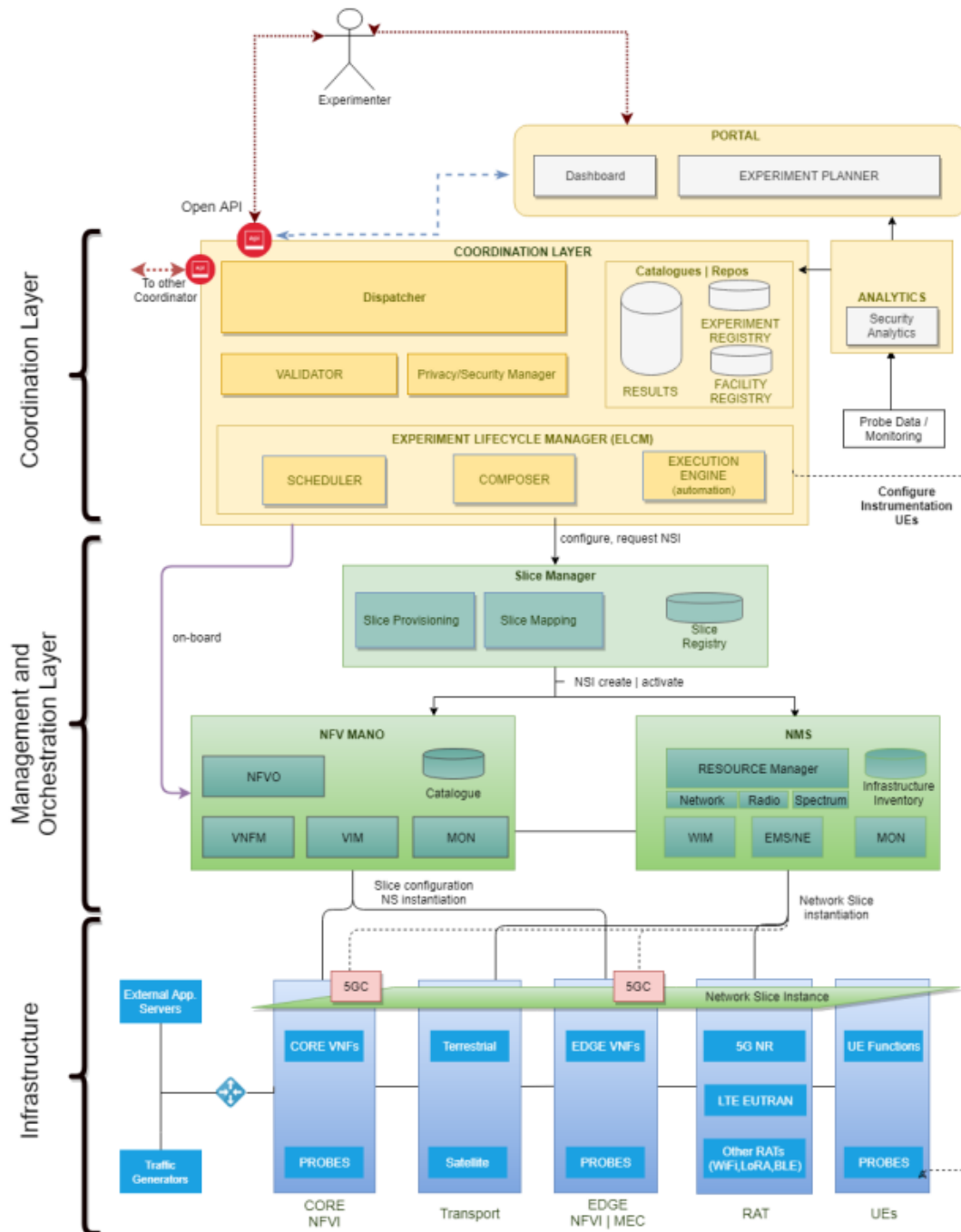


Figure 1 5GENESIS architecture

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 Layer¹. 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.

¹ Additional information about the Infrastructure Layer is included in WP4 deliverables.

2.2 Main features of Release A

The Release A of the 5GENESIS Coordination Layer and that of the platforms form altogether the “[Open 5GENESIS Suite](#)” and provide some baseline functionalities in order to allow initial integration at the platforms, thus enabling the first phase of experimentation. Release A provides the support for the execution of Tests Cases and the collection of results. In the forthcoming releases (B and C) the whole automation cycle will be further automated and additional features will enhance the experimentation capabilities and KPI validations for the vertical Experimenters. In a nutshell, the prominent features available after the installation of the Release A codebase are summarised in Table 2.

Table 2 5GENESIS Release A. Summary of functionalities

Component Name	Functionalities	Reference
Portal	<ul style="list-style-type: none"> • User registration • Experiment Creation • Experiment Initiation • Experiment logs and raw results visualization 	Release A Repository
ELCM	<ul style="list-style-type: none"> • Start/Stop/Runtime status of the experiment • Automatic handling and storage of execution logs • Orchestration of external components via Test Automation Platform (TAP) Test Plans or script execution • Extensible via plugins for various infrastructure elements and components • Baseline tests for platform validation defined • Core functionality tested by implementing several test cases (for validation of a number of KPIs) 	Release A Repository
Monitoring and Analytics	<ul style="list-style-type: none"> • Statistical processing and report on selected KPIs (depending on the available test cases) • Monitoring data collection from various probes • Monitoring data collection from infrastructure monitoring platforms (i.e. Prometheus) • Performance Monitoring (PM) • Infrastructure Monitoring (PM) 	Release A Repository
Slice Manager	<ul style="list-style-type: none"> • Start/Stop/Inspect end-to-end network slices • Open APIs supported by swagger-io tool • Lightweight web user interface • Integrated Command-Line Interface (CLI) tool • Modular architecture supporting different infrastructure technologies • Slice deployment and configuration time measurement 	Release A Repository

NMS	<ul style="list-style-type: none"> • WAN Infrastructure Manager (WIM) • Software Defined Network (SDN) Control • Monitoring • TAP plugins for controlling Android devices and computers through Secure Shell (SSH) • Performance monitoring tools for Android: resource usage, latency and throughput probes • Remotely controlled latency and throughput probes for Personal Computer (PC) 	Release A Repository / OPENTAP repository
-----	---	--

2.3 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 3 below:

Table 3 5GENESIS Experimentation Roles and Stakeholders

5GENESIS Experimentation Roles	
Experimenter	
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 through a direct use of the API for Experimenters, developed in the project.	
Potential Stakeholders:	<ul style="list-style-type: none"> • <u>Vertical</u> representatives: keen to test a 5G infrastructure and experience promised services & 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 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). • <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.
Platform Technology Provider	

5GENESIS Experimentation Roles	
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.	
Potential Stakeholders:	<ul style="list-style-type: none"> • <u>Technology Providers</u> (5G Vendors, Software Integrators) bringing in products • <u>Research Institutions and Academia</u>: Bringing in prototypes and deployment configuration recommendations
Platform Operator 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).	
Potential Stakeholders	<ul style="list-style-type: none"> • <u>5GENESIS Platforms</u> (Athens, Málaga, Limassol, Surrey, Berlin) • <u>Technology Providers</u> • <u>Research Institutions and Academia</u> <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&D projects under the 5G-PPP umbrella, the Platform Operators are primarily the partners of the 5GENESIS consortium.</p>
Testers and End Users 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.	
Potential Stakeholders	<ul style="list-style-type: none"> • Vertical representatives • Platform Operators • Technology providers • End-Users <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 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.

2.4 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 4 (Athens), Table 5 (Málaga), Table 6 (Limassol) Table 7 (Surrey) and Table 8 (Berlin):

Table 4 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.

5GENESIS Athens-GREECE Platform		
Sites	<ol style="list-style-type: none"> 1. NCSR Campus@Ayia Paraskevi 2. COSMOTE@Marousi 3. Stadium@Egaleo 	
Deployed 5G Technologies	2020	5G Non-Stand-Alone (NSA) Core Network: Athonet, Amarisoft, Eurecom Radio Access: Nokia 5G, Nokia LTE, Amarisoft, Eurecom, Runel UE: Commercial, Eurecom, Amarisoft
	2021	Upgrade to 5G Stand-Alone (SA)
Use Cases	<ol style="list-style-type: none"> 1. Big Event in a soccer stadium 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) 3. Security-as-a-Service 	
Contact	athens@5genesis.eu	

Table 5 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"> 1. Ada Byron Research@UMA (indoor & outdoor) 2. Málaga city centre 3. Málaga Police Emergency Centre 4. Telefonica I+D lab in Málaga/Madrid 	
Deployed 5G Technologies	2020	5G NSA Core Network: Athonet, Polaris Radio Access: Nokia 5G, Nokia LTE, Amarisoft, Eurecom, Runel UE: Commercial, OAI, Amarisoft
	2021	Upgrade to 5G SA
Use Cases	<ol style="list-style-type: none"> 1. Wireless video in large scale event 2. Multimedia Mission Critical Services 3. Edge-based Mission critical services 	

5GENESIS Málaga -SPAIN Platform	
Contact	malaga@5genesis.eu

Table 6 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"> 1. Primetel@Limassol 2. Avanti@Makarios 3. Portable Hotspots 	
Deployed 5G Technologies	2020	5G NSA Core Network: Athonet, Amarisoft, Eurecom Radio Access: Amarisoft, Eurecom UE: Commercial, Eurecom
	2021	Upgrade to 5G SA
Use Cases	<ol style="list-style-type: none"> 1. 5G maritime communications 2. 5G Capacity-on-demand and IoT in rural areas 	
Contact	limassol@5genesis.eu	

Table 7 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 Core Network: 5GIC 5GC NSA (in-house) Radio Access: Huawei UE: Commercial
	2021	Upgrade to 5G SA

5GENESIS Surrey-UK Platform	
Use Cases	1. Massive IoT for large-scale public events
Contact	surrey@5genesis.eu

Table 8 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	1. Fraunhofer FOKUS @West Berlin 2. IHP@Frankfurt (Oder) 3. Humboldt University@Berlin center	
Deployed 5G Technologies	2020	5G SA Core Network: Open5GCore (in-house) Radio Access: Huawei UE: Commercial
	2021	Further upgrades and new deployments of 5G SA Radio
Use Cases	1. Festival of Lights	
Contact	berlin@5genesis.eu	

- The 5GENESIS Architecture 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. Interactions through the OpenAPI, which offers a systemic and automated interface unlike 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 3rd 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.

- **Section 6** concludes the document.
- Finally, in the **Annex** sections, the technical and configuration parameters are documented as referenced in each section.

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.

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.

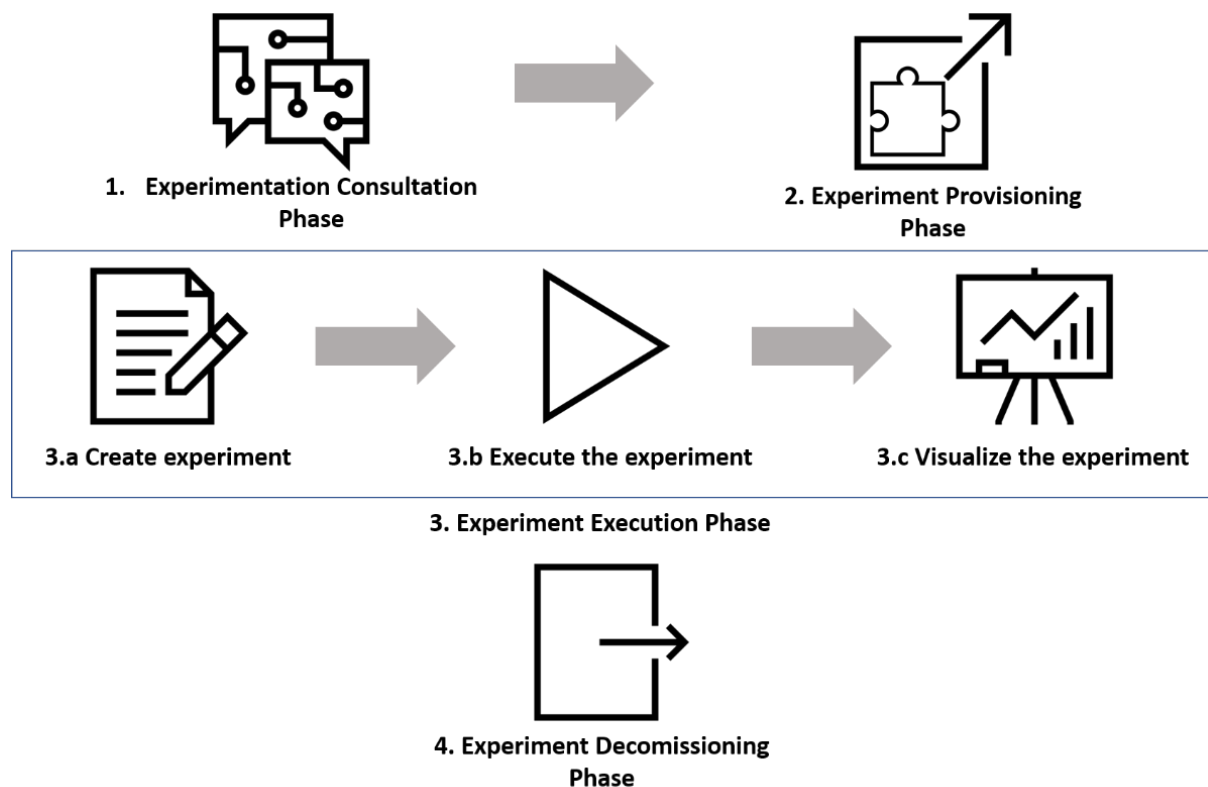


Figure 3 The 5GENESIS Experimentation Workflow

- The Experimenter shall detail the equipment and systems to be on-boarded for the experiment and will be instructed on 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 understood, the Experimenter may need to introduce the necessary adaptations into the systems to be on boarded, either Network Service (NS), applications or equipment, so the explained methodologies are supported, and 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 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 requested 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 will 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)	<table><tr><td colspan="3">MCPTT Access time</td></tr><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			MCPTT access time [ms]			Mean	95% confidence interval for Mean		Lower bound	Upper bound	55,192	50,114	60,271
MCPTT Access time																	
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 by the Experimenter are decommissioned from the 5GENESIS platform and any 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.

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 provides 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, described in Section 2.4.1, to understand the required resources to onboard 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 onboarding.

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).

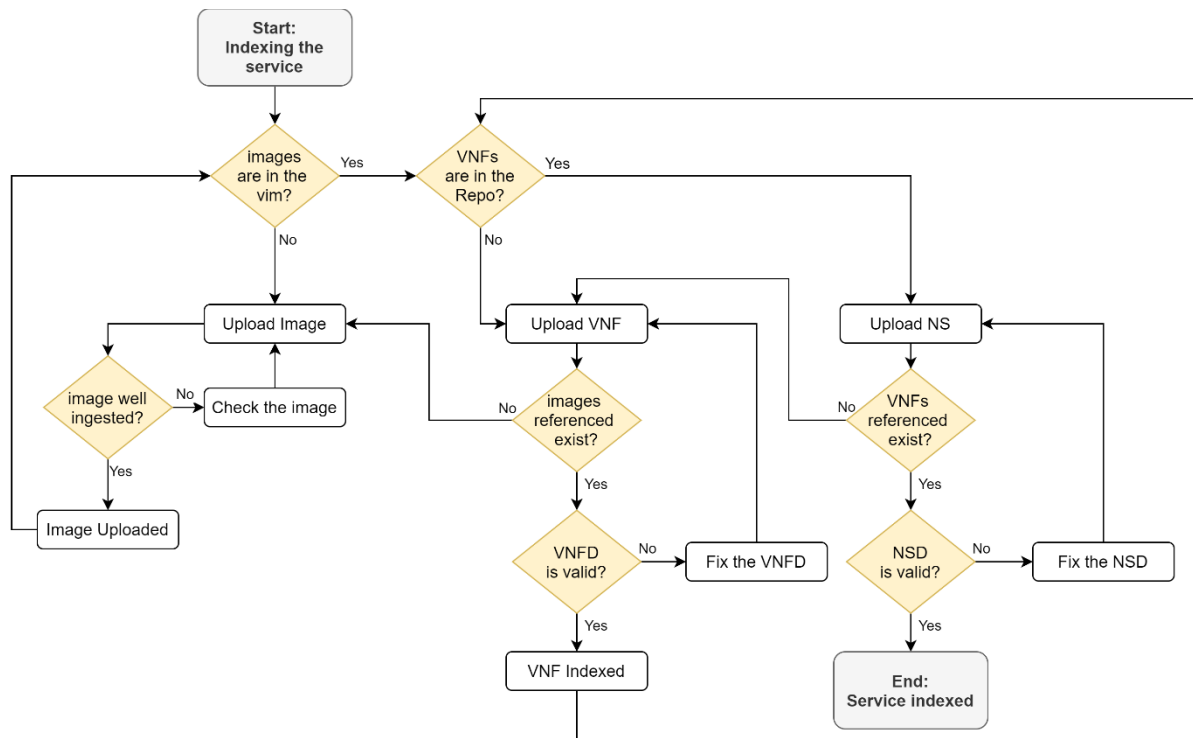


Figure 4 Network Service Onboarding flow diagram

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 have to provide the images and the descriptors to the platform operator for the onboarding operation.

In Release B 5GENESIS will offer 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 fields 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. In Release A, the Open APIs provide a basic set of functions that enable the automation of the experiments. Defining the experiments via the Portal is probably the preferred option for most of the Experimenters, as they will be able to run experiments in a controlled environment all the times they want, and, if needed, automate the execution via the Open API. 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 specified please refer to D6.1 [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 test cases that will subsequently be listed in the 5GENESIS Portal.

The test cases define the target KPI and how to execute the test, but it is also required to specify 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 realistic test conditions).

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 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 A

Test case	Target	Output	Platforms
Round-trip-time	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
Throughput	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
Streaming	KPIs related to streaming service such as jitter and adaptation	Average, maximum, minimum, 95% confidence interval, 5% confidence interval	Athens
Service creation time	The time needed to deploy virtual machines (VMs) on compute systems	Average, maximum, minimum, 95% confidence interval, 5% confidence interval	Málaga, Berlin
MCPTT	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.

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 that declare parameters such as the level of the transmission power in a base station, the mobility of the end devices, the 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.

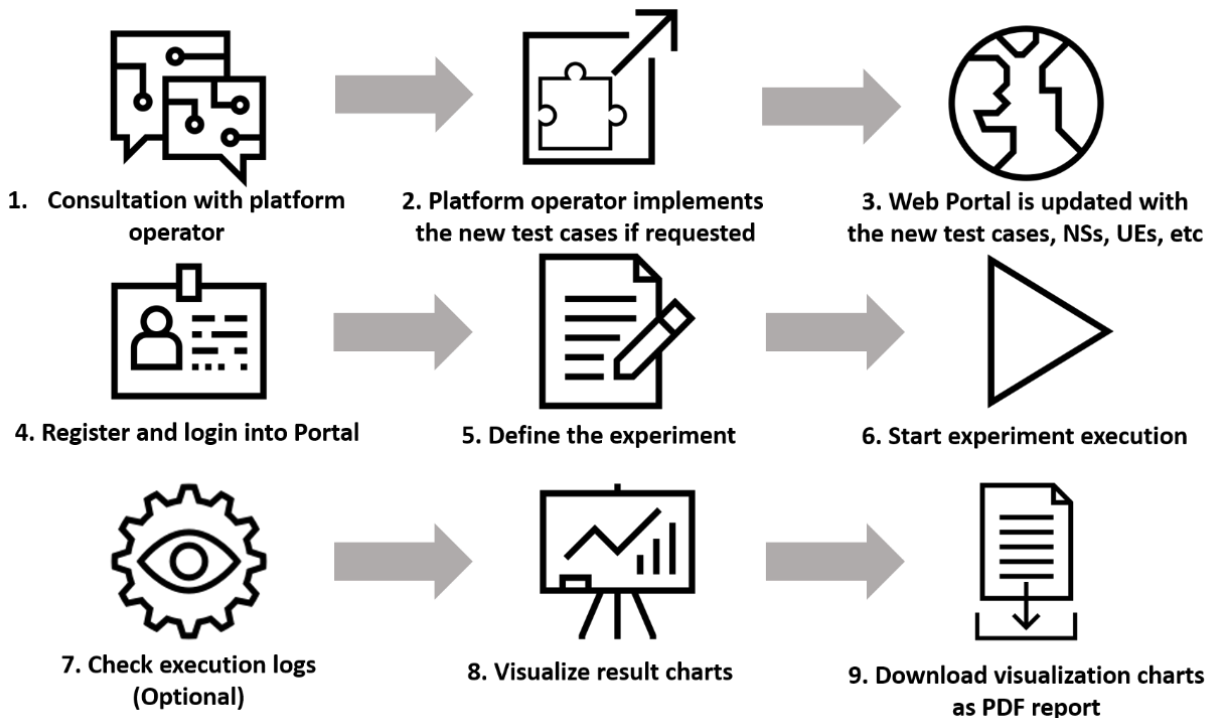


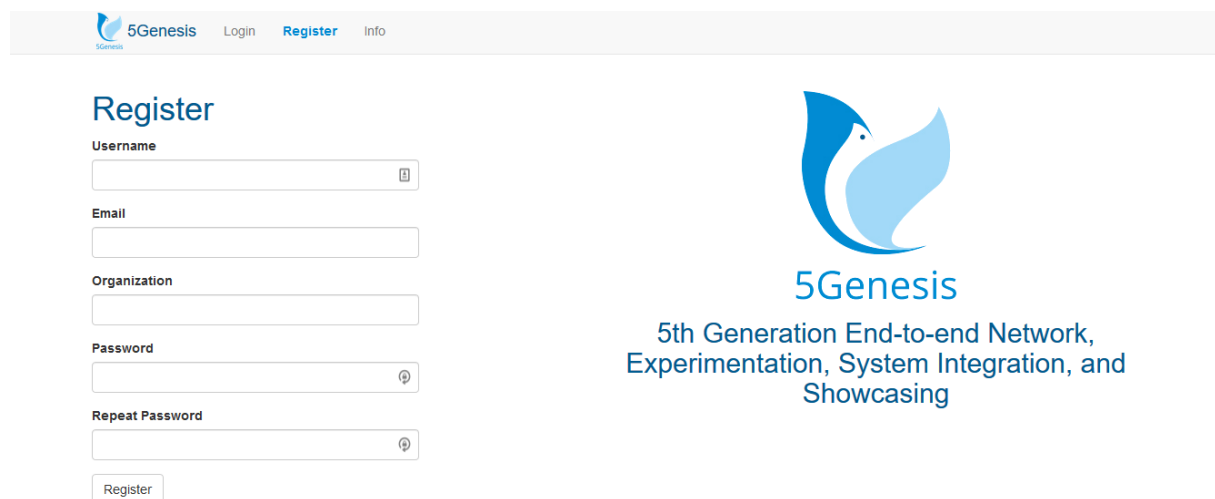
Figure 5 Experimentation via de Portal

3.2.1 Login

Each platform will deploy 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.

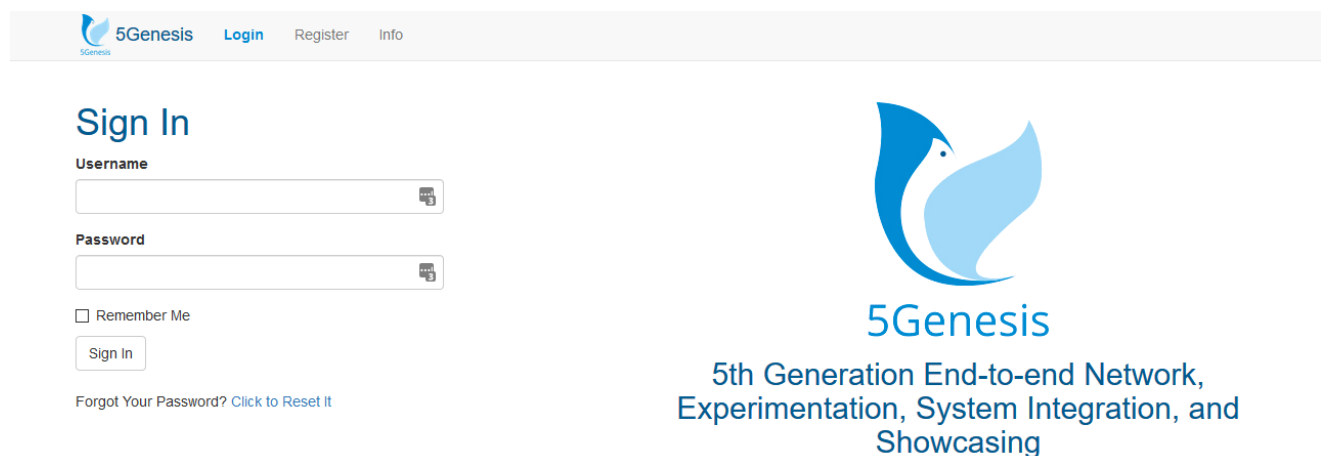
To proceed with the registration, the user must click on the “Register” tab in the upper part of the Portal website. All the required information must be filled in, and then the user must click the “Register” button (as depicted in Figure 6). All the information will be automatically validated, and the new account will be successfully created if the information is correct. In future releases of the Portal, the accounts will need validation by a system administrator before their activation.

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.



The registration screen features a header with the 5Genesis logo and navigation links for Login, Register, and Info. The main content area is titled 'Register' and contains a form with the following fields: Username, Email, Organization, Password, and Repeat Password. Each field has a corresponding icon (e.g., a person icon for Username, an envelope for Email, a building for Organization, and a key for Password). A 'Register' button is located at the bottom of the form. To the right of the form is a large 5Genesis logo and the text '5th Generation End-to-end Network, Experimentation, System Integration, and Showcasing'.

Figure 6 5GENESIS Portal registration screen



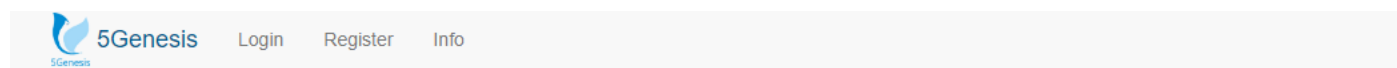
The sign in screen features a header with the 5Genesis logo and navigation links for Login, Register, and Info. The main content area is titled 'Sign In' and contains a form with the following fields: Username and Password. Each field has a corresponding icon (e.g., a person icon for Username and a key for Password). Below the Password field is a checkbox labeled 'Remember Me' and a 'Sign In' button. At the bottom of the form is a link that says 'Forgot Your Password? Click to Reset It'. To the right of the form is a large 5Genesis logo and the text '5th Generation End-to-end Network, Experimentation, System Integration, and Showcasing'.

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).

It is important to note that, during Release A, if an experiment requires a NS, this service must be available in the testbed or must be uploaded manually by the testbed operator. Additionally, any other functionality or feature not directly supported through the Portal can be discussed with the testbed operator, to check for its feasibility. In Release B, the NS onboarding will be added in Portal to allow Experimenters to upload the required NS artefacts by themselves.



MALAGA PLATFORM DESCRIPTION

Available devices

- **Samsung Galaxy S9** (March 2018):
OS: Android 8.0 - Samsung Experience Vision 9.0
Exynos 9810 (10 nm) - 4GB RAM - 1440 x 2960 pixels, 18.5:9 ratio
IP address: 172.23.2.189
Attached to scenario LTE pedestrian
- **Samsung Galaxy Note 10+ 5G** (August 2019):
OS: Android 10.0 - OneUI 2.0
Exynos 9825 (7 nm) - 12GB RAM - 1440 x 3040 pixels, 19:9 ratio
IP address: 172.23.4.151
Attached to scenario 5G NR Non-standalone

Scenarios

- **LTE Pedestrian:**
This setup is based on an eNodeB emulator configured with a EPA5 channel model, signal to noise ratio of 20dB, adaptive modulation, single carrier and 20MHz bandwidth and all the resource blocks assigned to the UE.
- **5G NR Non-standalone**
One eNodeB and one gNodeB, 40MHz bandwidth, RSSI values within a range of between -64dBm and -60dBm and a signal to noise ratio of 20dB.

Network services

- **Nemergent MCS server:**
The Nemergent MCS Server provides the application-level components required to deploy the full 3GPP Rel13 Mission Critical Push-To-Talk (MCPTT) service, together with a selection of the main features in the 3GPP Rel14 Mission Critical Video (MCVideo) and Mission Critical Data (MCData) services.
- **DASH Video streaming server:**
Custom made VNF that integrates the GPAC streaming server. GPAC is an implementation of the MPEG-4 Systems standard written in ANSI C. GPAC provides tools for media playback, vector graphics and 3D rendering, MPEG-4 authoring and distribution.

Figure 8 5GENESIS Portal info page

3.2.2 Definition of experiments

The “Create Experiment” tab is available when the user has logged in. This tab allows the user to define the parameters for the 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 can be selected from a list depending on the availability for 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, this way fulfilling the Experimenter needs.

Figure 9 Experiment creation screen

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 10.

Experiment ID	Name	Type	Action
6	Video Performance	Standard	Run Experiment Executions
5	Streaming test case	Standard	Run Experiment Executions
2	Maximum User Data Rate Test	Standard	Run Experiment Executions
1	Round Trip Time - Ideal	Standard	Run Experiment Executions

ACTIONS

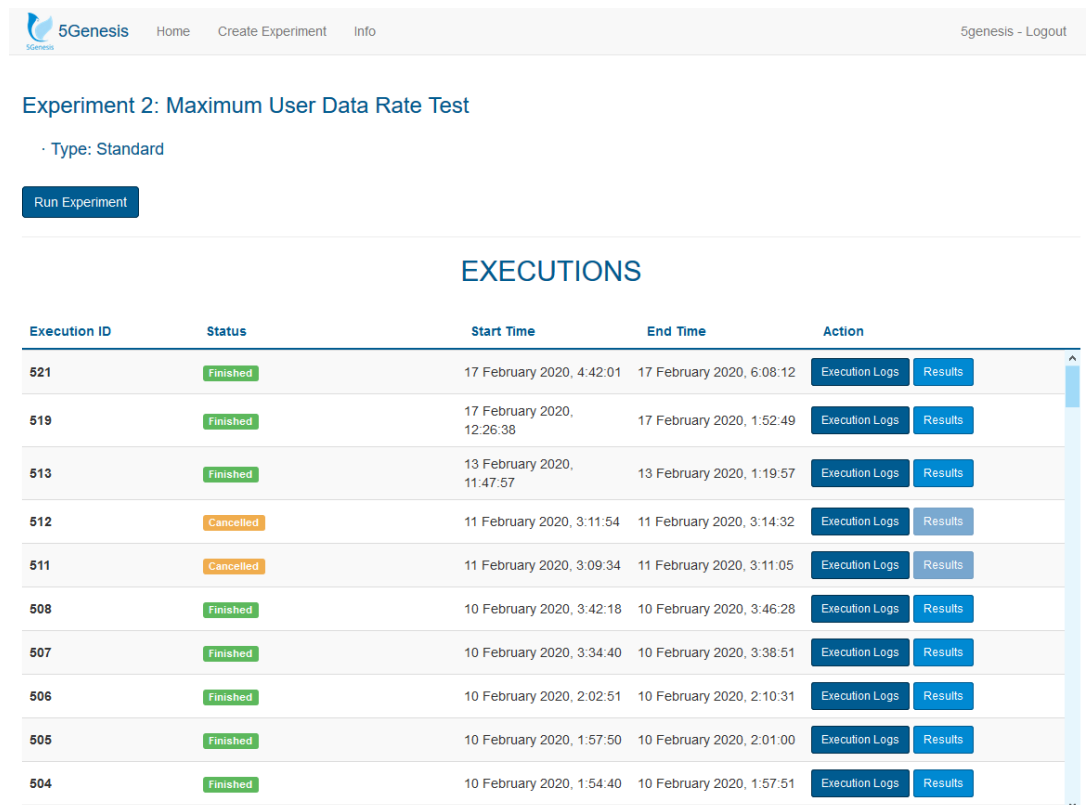
- 17 February 2020, 4:41:53
Ran experiment: Maximum User Data Rate Test
- 17 February 2020, 2:19:52
Ran experiment: Round Trip Time - Ideal
- 17 February 2020, 12:26:30
Ran experiment: Maximum User Data Rate Test
- 16 February 2020, 9:07:44
Created experiment: Streaming test case

Figure 10 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 11.

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 finished

successfully. Additionally, the “Run Experiment” button in the upper left part serves as a shortcut to execute (re-run) the experiment again.



Execution ID	Status	Start Time	End Time	Action
521	Finished	17 February 2020, 4:42:01	17 February 2020, 6:08:12	Execution Logs Results
519	Finished	17 February 2020, 12:26:38	17 February 2020, 1:52:49	Execution Logs Results
513	Finished	13 February 2020, 11:47:57	13 February 2020, 1:19:57	Execution Logs Results
512	Cancelled	11 February 2020, 3:11:54	11 February 2020, 3:14:32	Execution Logs Results
511	Cancelled	11 February 2020, 3:09:34	11 February 2020, 3:11:05	Execution Logs Results
508	Finished	10 February 2020, 3:42:18	10 February 2020, 3:46:28	Execution Logs Results
507	Finished	10 February 2020, 3:34:40	10 February 2020, 3:38:51	Execution Logs Results
506	Finished	10 February 2020, 2:02:51	10 February 2020, 2:10:31	Execution Logs Results
505	Finished	10 February 2020, 1:57:50	10 February 2020, 2:01:00	Execution Logs Results
504	Finished	10 February 2020, 1:54:40	10 February 2020, 1:57:51	Execution Logs Results

Figure 11 Experiment executions screen

The “Execution Logs” button leads to a screen, depicted in Figure 12, 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. It also shows a “Results” button with the same functionality as the one in the “Executions” screen.

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

3.2.4 Visualization of results

The visualization of the results of the experiment is based on Grafana. The Grafana dashboard is accessible through the “Results” button shown in Figure 11. 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.

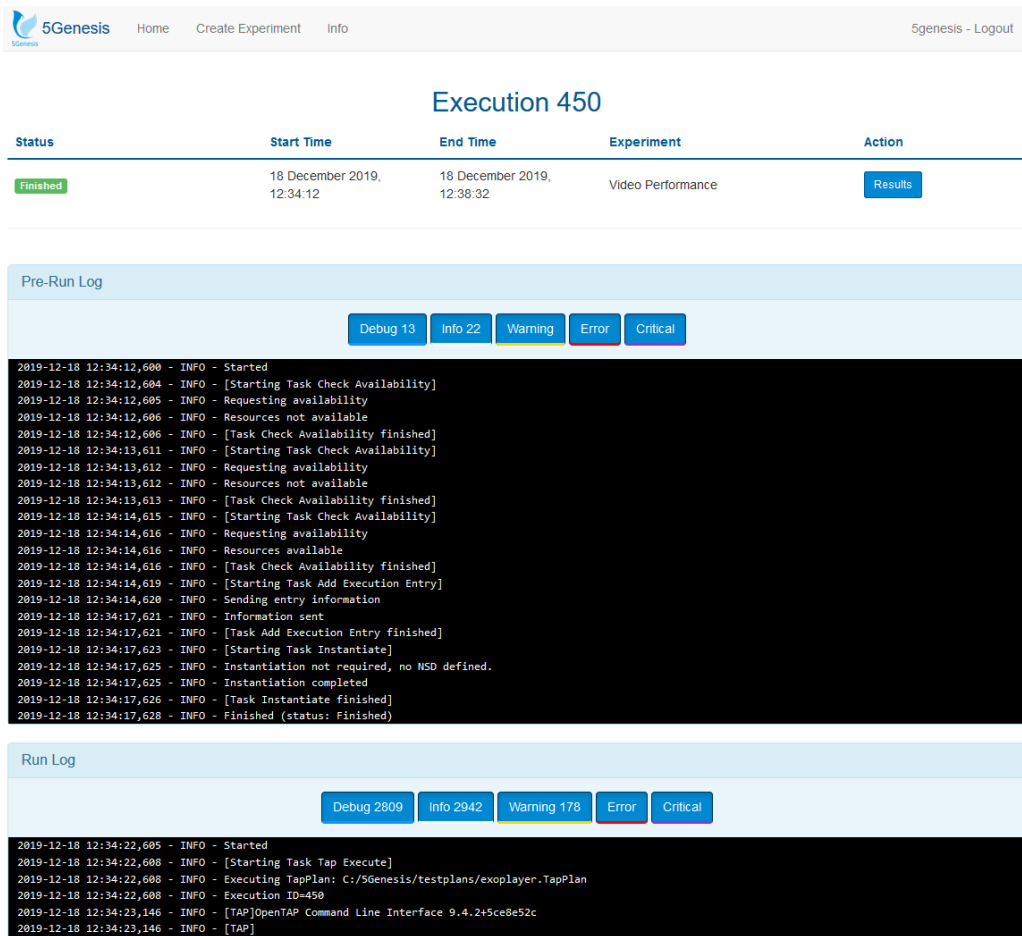


Figure 12 Experiment execution log screen



Figure 13 Experiment result visualization screen

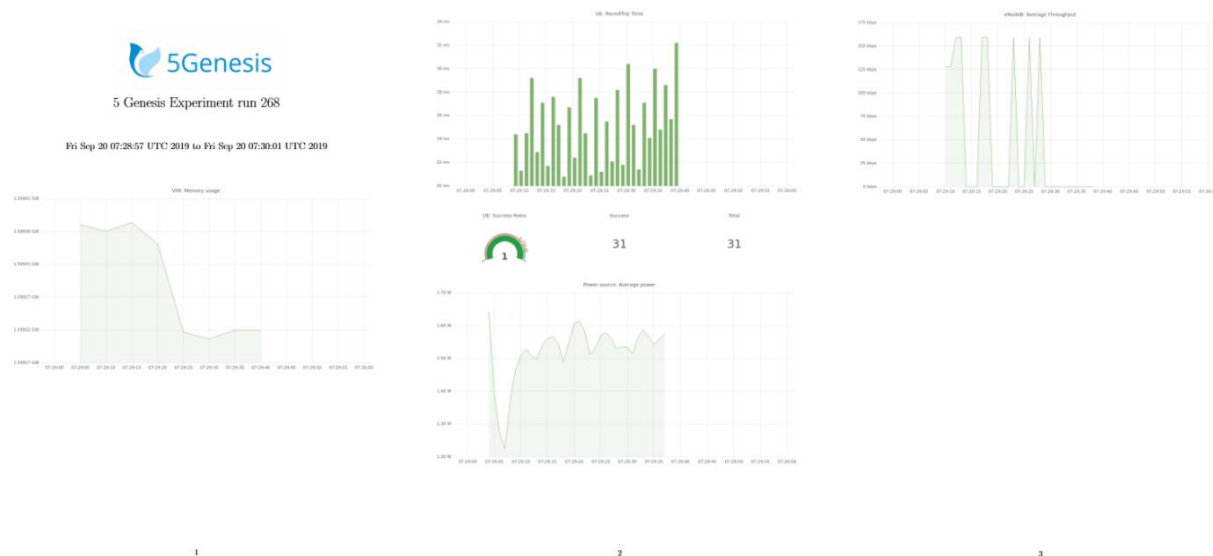


Figure 14 Generated PDF results report

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 must be 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 below in Figure 14.

It is important to note that the users will be asked for credentials the first time they access to the visualization of the results of an experiment. To obtain the username and password, the user must contact the platform administrators. In Release B the Experimenter will be able to create on its own the account that will be used, given that it will be authorized by the platform operator the first time the Experimenter logged in.

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

3.3 Experimentation via the Open APIs

During Release A, there is a minimum 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, but with a more limited interface than that planned for Release B (refer to deliverable D3.7 for the extended version of the Open API).

Figure 15 shows the experimentation steps when using the Open APIs.

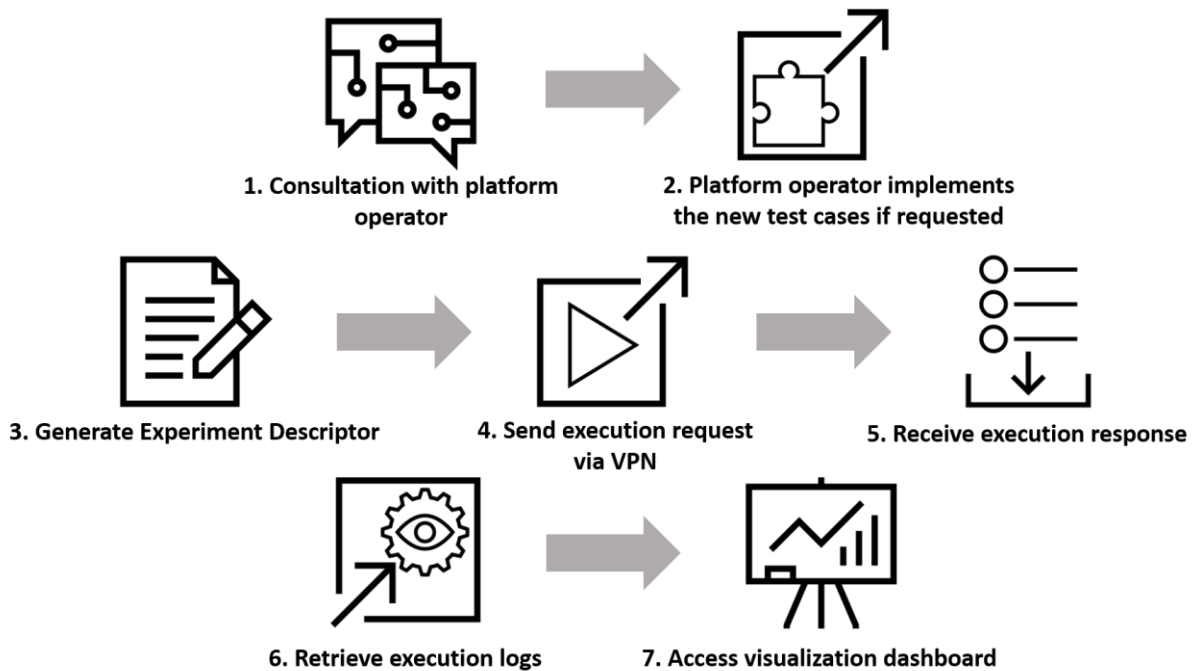


Figure 15 Experimentation via the Open APIs

3.3.1 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:

```

{
  'Id': <Integer>,
  'Name': <String>,
  'User': {
    'Id': <Integer>,
    'UserName': <String>,
    'Email': <String>,
    'Organization': <String>
  },
  'Executions': <List[Integer]>,
  'Platform': <String>,
  'TestCases': <List[String]>,
  'UEs': <See below>,
  'Slice': <String>,
  'NSD': <String>
}
  
```

The expected contents of each field are:

- **'Id'**: Numeric ID of the experiment. This value may match an existing experiment ID in the Portal. However, when using the Open APIs directly, the execution will not be registered in the Portal.
- **'Name'**: Name of the experiment.

- *'User'*: The *User* dictionary contains information about the Experimenter requesting the execution. It is used internally by the ELCM to display some tracing information, but will not perform any check about the validity of these values. They are, however, mandatory.
- *'Executions'*: The list of previous execution IDs of the same experiment. It is possible to provide an empty list if these values are not known.
- *'TestCases'*: The list of test cases (by name) that are to be run during the experiment execution. The list of available test cases varies for different platforms and can be requested to the platform administrators if necessary.
- *'UEs'*: Information about the UEs used during the experiment. This field is a Dictionary of type (*<String>*, *<UE_Info>*), where the key corresponds to the identifier of the UE in the Platform, and *UE_Info* is in itself another dictionary that contains extra information about the UE. In practice, it is possible to supply an empty dictionary as *UE_Info*, while the list of available UE identifiers can be requested to the platform administrators.
- *'Slice'*: The ID of the Network Slice to use during the experiment. During Release A, this value is a placeholder, and can be set to any string value.
- *'NSD'*: The NS Descriptor file to use during the *deployment* phase of the experiment execution. Onboarding and uploading of NSD files can only be performed by platform administrators in Release A. For this reason, this functionality is not available through the Open APIs and this field must be set to *None*.

3.3.2 Sending execution requests to the ELCM

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 [17] 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.

3.3.3 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.

3.3.3.1 Grafana dashboards

Once the experiment execution finishes, the ELCM will perform the creation of a customized Grafana dashboard that contains the most relevant results generated by the experiment. The URL for this dashboard is generated following a known pattern. For an experiment execution with ID `<ExecutionId>` the URL will be:

```
<Grafana_Host>:<Grafana_Port>/d/Run<ExecutionId>/experiment-run-  
<ExecutionId>
```

The details about the host and port where Grafana listens for connections can be requested to the platform administrators.

3.3.3.2 Experiment logs

The experiment logs are accessible by sending a GET request to the following endpoint on the ELCM `/execution/<ExecutionId>/logs`. For example, using cURL:

```
curl -X GET "<ELCM_host>:<ELCM_port>/execution/<ExecutionId>/logs"
```

The ELCM will return a JSON object with the following structure:

```
{  
  'Status': '<'Success' or 'Not found'>,  
  'PreRun': <LogInfo or None>,  
  'Executor': <LogInfo or None>,  
  'PostRun': <LogInfo or None>  
}
```

Where `<LogInfo>` are dictionaries with the following content:

```
{  
  'Count': {  
    'Debug': <Integer>,  
    'Info': <Integer>,  
    'Warning': <Integer>,  
    'Error': <Integer>,  
    'Critical': <Integer>  
  }  
  'Log': <List[Tuple[String, String]]>  
}
```

- 'Count' is a dictionary that contains, for each severity level, the number of messages of that kind.
- 'Log' is a list of all the messages contained, where each entry is a pair of Strings: The first string corresponds to the severity level of the message, while the second string is the message itself.

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 in Figure 16, the plugins implement all the functions required to communicate with the ELCM, the Slice Manager and the monitoring system.

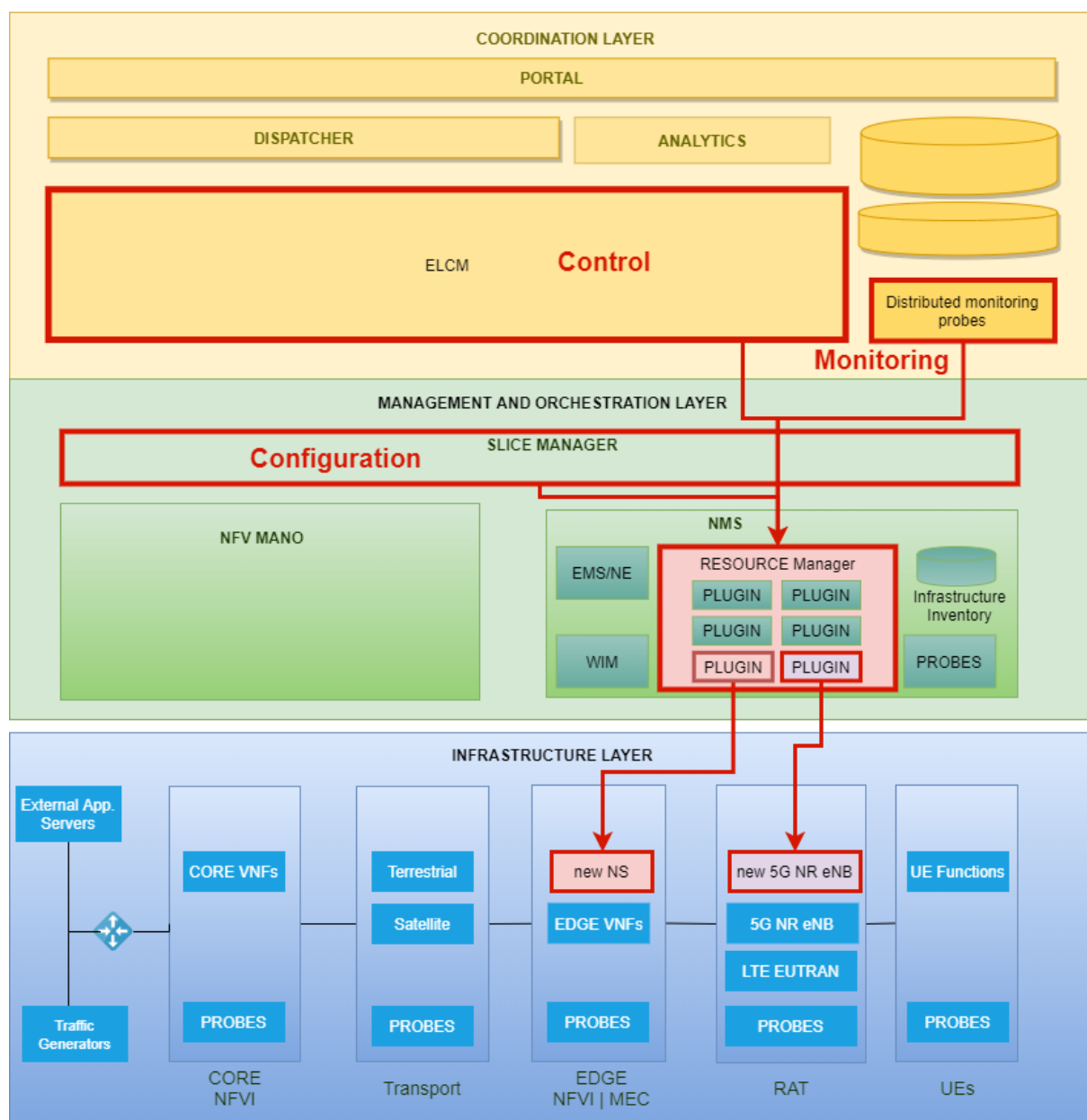


Figure 16 Integration of new components to the 5GENESIS Architecture

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 [17], 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 [18] 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 [19]. 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 17. The *Display* decorator defines how these settings should be presented to the end user.

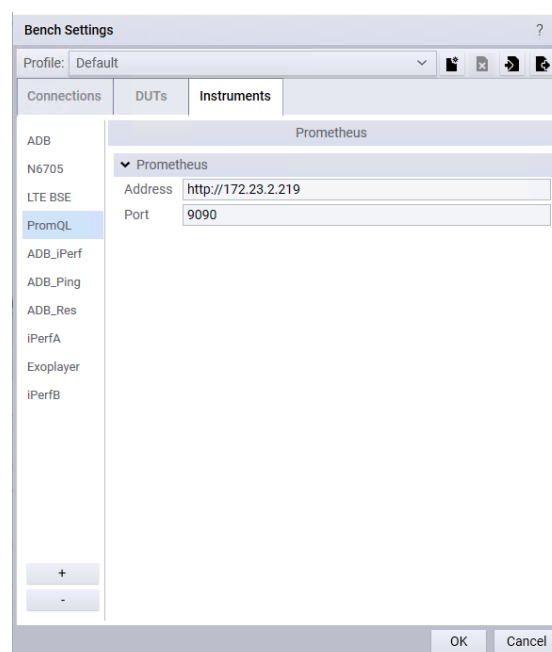


Figure 17 Prometheus instrument

```
[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*th 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, *toConvertible* is a helper method that tries to convert a given string to the most suitable data type possible.

```
private IConvertible toConvertible(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, params):
        super().__init__("Compress Files", 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.params: Contains a dictionary with the parameters of the Task. This dictionary will be filled by the ELCM using the information contained in the Facility Registry (please refer to Deliverable D3.15, section 3.2.1).
 - 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
```

At this point, the new Task is available and can be used 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, params):
        super().__init__("Compress Files", 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 18 depicts an overview of the building blocks of the 5GENESIS Slice Manager.

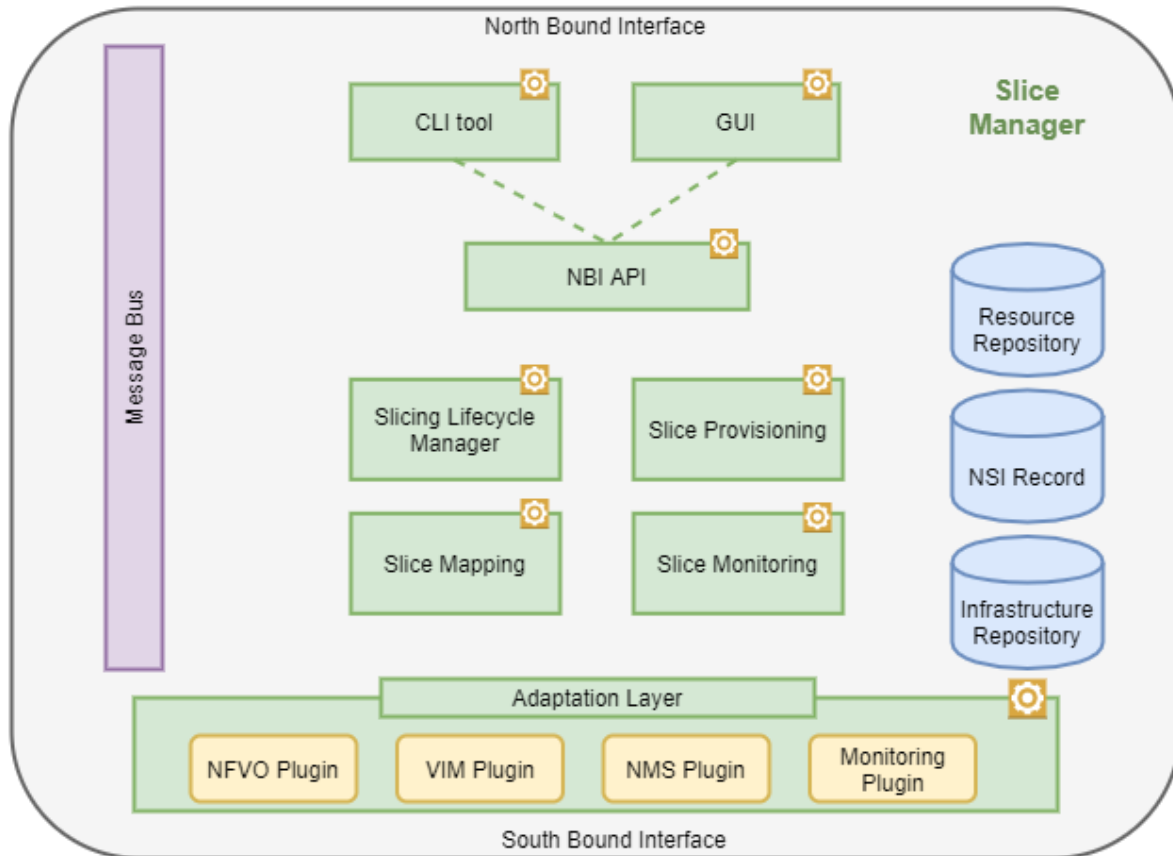


Figure 18 Slice Manager Building Blocks

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

Component	Operation	Phase
NFVO	Read NSDs and VNFDs	Slice Creation – Placement
	Add a new VIM account (VIM Tenant)	Slice Creation – Resource Provisioning
	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
WIM	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
EMS	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
MON	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 19.

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)
```

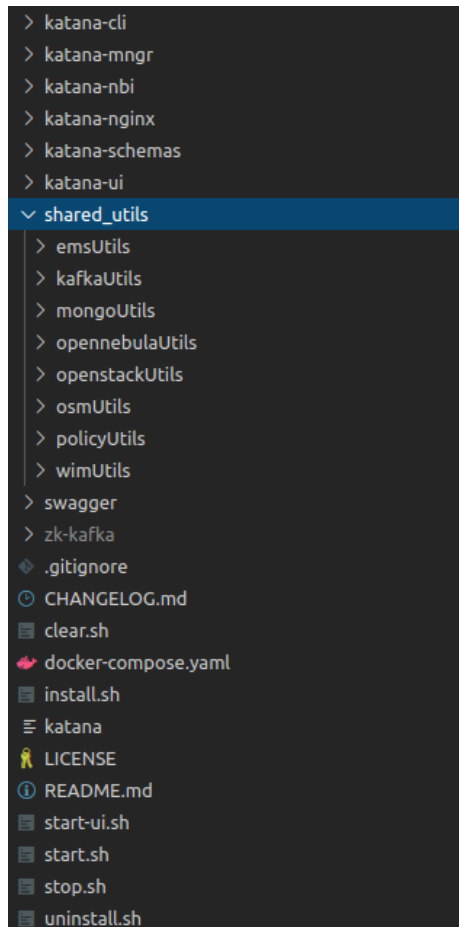


Figure 19 Slice Manager Source Code Directory

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 A. These components are described in deliverables D3.1, D3.3, D3.5, D3.7, D3.9 and D3.11.

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 [21].
- Virtualenv [44].
- [Optional] Grafana [22] (tested on version 5.4).
- [Optional] Grafana reporter [23] (tested on version 2.1.0, commit 41b38a0, see ELCM readme for more information).

5.1.1.1 Installation

The installation process for the development of the Portal is composed of the following steps:

1. Clone the Portal repository to a known folder, e.g. in `c:\5gPortal`.
2. Enter the folder:

```
cd c:/5gPortal
```

3. Create a new Python virtualenv:

```
pip install virtualenv
virtualenv --python=python3.7 venv
```

4. Activate the virtual environment:

- For Windows:

```
venv\Scripts\Activate.bat
```

- For Linux:

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

5. On some Linux distributions it may be necessary to install additional packages. For example the following packages need to be installed through apt on Ubuntu: 'gcc', 'python3-dev'
6. Install Python dependencies:

```
pip install -r requirements.txt
```

6. Upgrade (initialize) the database to the latest version:

```
flask db upgrade
```

7. Start the development server:

```
flask run
```

It is recommended, but not required, to run the Portal in a Python virtual environment [24]. If virtual environments are not used, the reader can skip steps 3 and 4.

The Portal is configured for creating an SQLite database automatically (`app.db`) if no other database backend is configured. If the deployment will use a different backend it might be wise to set it before running step 6. More information on this is provided below in the configuration subsection.

After executing the last step, the app will generate a default configuration file (`config.yml`) and start listening for requests on port 5000. The development server can be stopped pressing `Control+C`.

5.1.1.2 Deployment

The deployment instructions to setup a functional Portal are easy to follow, since the Portal repository includes a `Vagrantfile` that can be used to automatically deploy a VM that includes the Portal instance (running under `Gunicorn`) and an `nginx`. This file can also be used as an example of how to deploy the Portal on an existing Linux machine or using Docker containers, since most of the executed commands are valid in many other environments.

In order to deploy using `Vagrant`:

1. Install Vagrant [25] and Virtualbox [26].
2. Navigate to the Portal folder.
3. Create the VM:

```
vagrant up
```

This will create and start a VM named `5genesis-portal` and bind port 80 of the host machine to the Portal instance. If you cannot bind the Portal to port 80 you can use a different port by setting other value in the `Vagrantfile` (`config.vm.network "forwarded_port"`).

The default deployment does not use https. To enable it one needs to provide the necessary certificates and customize the `nginx` configuration. This repository includes an example configuration file (`Vagrant/nginx_ssl.conf`) that can be used as a base.

5.1.1.3 Configuration

The Portal instance can be configured by setting environment variables and by editing the `config.yml` file. The Portal uses `python-dotenv`, so it is possible to save the environment variables in the `.flaskenv` file.

For a basic configuration of the Portal one can follow these steps:

1. Navigate to the Portal installation folder and edit the `config.yml` file.
2. The file should already contain the default values. Most of them can be customized later, but for the initial tests just the `Dispatcher` and `TestCases` sections will be modified.
3. Modify the `Host` and `Port` values in the `Dispatcher` section so that they refer to the IP and port where the ELCM will be listening (during Release A the `Dispatcher` does not exist as a separate entity, which is the reason why these values point to the ELCM).

4. Include a new `TEST` value (or modify the existing `RTT` one) in the `TestCases` list. Save the file. The `TEST` test case will be used in order to ensure that the basic functionality of the Portal and ELCM is correctly configured.
5. Start the Portal using the commands previously explained and navigate on a web browser to the address and port configured. You should see the Portal's login screen. Navigate to `Register` and create a user.
6. Login with the newly created user's credentials and click on `Create Experiment`.
7. Give a name to the experiment and select `TEST` in the `TestCases` list. All other values can be left as default. Then press `Add Experiment`.
8. The new experiment will appear on the user's Home screen. Do not click on the `Run Experiment` button yet (If you decide to press the button you will just see an error notice, and clicking on `Executions` will perform similarly).

The environment variables that can be set for the configuration of the Portal are:

- **SECRET_KEY:** Set this value to a **RANDOM** string (the default value is not random enough) [27].
- **FLASK_RUN_PORT:** Port where the portal will listen (5000 by default)
- **SQLALCHEMY_DATABASE_URI:** Database instance that will be used by the Portal. Depending on the backend it is possible that additional Python packages will need to be installed, for example, MySQL requires `pymysql`. For more information you can read about Dialects [28].
- **UPLOAD_FOLDER:** Folder path where the uploaded files will be stored.
- **MAIL_SERVER** (currently unused): Mail server location (localhost by default).
- **MAIL_PORT** (currently unused): Mail server port (8025 by default).

The values that can be configured on `config.yml` are explained in the following list:

- **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 assume for an authentication token to expire, which should be slightly shorter (30/60 seconds) than the real expiration time 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).
- **Platform:** Platform name/location.
- **TestCases:** List of test cases supported by the platform.
- **UEs:** Dictionary that contains information about the UEs available in the platform. Each element key defines the unique ID of the UE, while the value contains a dictionary with extra data about the UE (currently the operating system).
- **Slices:** List of available Network Slices.
- **Grafana URL:** Base URL of Grafana dashboard to display the Execution results.
- **Description:** Description of the platform.

- **PlatformDescriptionPage**: HTML file that contains the description of the platform. The HTML written in this file will be inserted in the 'Info' page of the Portal.
- **Logging**: Parameters for storing application logs.

It is important to note that direct communication with the ELCM is still needed. The list of test cases and UEs selected for each experiment will be sent to the Dispatcher (and ELCM) on every execution request. The ELCM uses these values in order to customize the campaign execution (via the Composer and the Facility Registry). The information on the slices is not currently used by the ELCM (as of 30/03/2020).

It is also possible to display system-wide notices in the Portal by including the file named `notices.yml` in the root folder. The list may include as many notices as necessary, and the format for the `notices.yml` file is as follows:

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

5.1.2 ELCM

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

- Compatible with Windows or Linux (Windows recommended).
- Python 3.7.x [21].
- Virtualenv [44].
- [Optional] Grafana [22] (tested on version 5.4).
- [Optional] Grafana reporter [23] (tested on version 2.1.0, commit 41b38a0).
- [Optional] InfluxDb [29] (tested on version 1.7.6).

5.1.2.1 Installation

The required steps for the installation and development of the ELCM are the following:

1. Clone the repository to a known folder, e.g. in `c:\ELCM`
2. Enter the folder

```
cd c:/ELCM
```

3. Create a new Python virtualenv:

```
pip install virtualenv
virtualenv --python=python3.7 venv
```

4. Activate the virtual environment:

- For Windows:
`venv\Scripts\Activate.bat`

- For Linux:
`source venv/bin/activate`

5. Install Python dependencies:

```
pip install -r requirements.txt
```

6. Start the development server:

```
flask run
```

It is recommended, but not required, to run the Portal in a Python virtual environment [24]. If you are not using virtual environments, skip steps 3 and 4.

The app will generate a default configuration file (`config.yml`) and start listening for requests on port 5001. Refer to the Configuration section for information about customizing the default values. Press `Control+C` to stop the development server.

5.1.2.2 Deployment

Since the ELCM should not be exposed to the Internet (the administration interface should only be accessed on the host machine or from inside a trusted network), and it will not receive a large number of concurrent requests, it is possible to run the ELCM using the included development server.

It is, however, also possible to run the ELCM using a production Web Server Gateway Interface (WSGI) server such as Waitress [30] following these instructions:

1. Install Waitress on the ELCM Python environment:

```
pip install waitress
```

2. Start the server:

```
waitress-serve --listen=*:5001 app:app
```

5.1.2.3 Configuration

Similarly, to the Portal, the ELCM instance can be configured by setting environment variables and by editing the `config.yml` file. The ELCM also uses `python-dotenv`, so it is possible to save the environment variables in the `.flaskenv` file.

A basic configuration of the ELCM can be achieved performing the following steps:

1. Navigate to the ELCM installation folder and edit the `config.yml` file.
2. The file should already contain the default values. You will need to customize many of these values later, but for now it is necessary to edit only the `Dispatcher` section. Modify the `Host` and `Port` values so that they refer to the IP and port where the Portal is listening and save the file.
3. Create a new YAML file in the `TestCases` subfolder, for example `test.yml` (the name of the file is not important, but the extension must be `yml`). This file defines the set of actions to perform when an experiment execution received through the Portal includes the `TEST` test case. It is important that the name of the dictionary on this file matches the name of the test case configured in the Portal. Copy or write the following content in it and save the file.

```
TEST:
  - Order: 10
    Task: Run.Message
    Config: { Severity: INFO, Message: "This is a TEST message" }
```

4. Start the ELCM using the commands explained previously and, on a web browser, navigate to the address and port configured. You should see the main dashboard. In the Diagnostics section review the `Facility Log`. Ensure that the following messages

are shown (there will be additional messages from debug to warning, but no error or critical messages):

```
1 TestCases defined on the facility: TEST.  
No UEs defined on the facility.  
No DashBoards defined on the facility.
```

There may be an error on the `Configuration Log`: This is normal and easy to solve by editing the `config.yml` file. For this basic configuration it is only important that the `Dispatcher` section correctly points to the Portal.

The environment variables that can be set for the ELCM configuration are:

- `FLASK_RUN_PORT`: Port where the ELCM will listen (5000 by default)
- `SECRET_KEY`: A random string.

Since the ELCM should not be exposed to the Internet, it is not so important to set a random `SECRET_KEY`. However, the value is still needed and it's recommended to follow the same approach as with the 5GENESIS Portal.

The values that can be configured on the ELCM's `config.yml` file are:

- **TempFolder**: Root folder where the temporal files for the Executors can be created.
- **Logging**:
 - `Folder`: Root folder where the different log files will be saved.
 - `AppLevel`: Minimum log level that will be displayed in the console.
 - `LogLevel`: Minimum log level that will be recorded in the log files.
- **Dispatcher**. The `Dispatcher` does not currently exist as a separate entity, so this information refers to the Portal during Release A:
 - `Host`: Location of the machine where the `Dispatcher` is running (localhost by default).
 - `Port`: Port where the `Dispatcher` is listening for connections (5000 by default).
- **Tap**. These values (Tap) will be used by the `Run.TapExecute` task:
 - `Enabled`: Whether or not to use TAP, if set to `False` the settings below will be ignored
 - `OpenTap`: `True` if using OpenTap (TAP 9), `False` if using TAP 8
 - `Exe`: TAP CLI executable
 - `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. Recommended to set to `True`, but increases execution time by roughly 15 seconds.
- **Grafana**. These values will be used when generating a dashboard for the results generated by an experiment:
 - `Enabled`
 - `Host`
 - `Port`
 - `Bearer`: Grafana API key without the 'Bearer ' prefix
 - `ReportGenerator`: URL where the `Grafana reporter` instance can be reached, if any

- **SliceManager.** These values will be used to communicate with the Katana Slice Manager when deploying/decommissioning slices and when using the `Run.SliceCreationTime` task:
 - Host
 - Port
- **InfluxDb.** These values will be used for sending results to an InfluxDb instance. In particular, they will be used by the `Run.SliceCreationTime` task:
 - Enabled: If set to False the settings below will be ignored
 - Host
 - Port
 - User: InfluxDb instance user
 - Password: InfluxDb user password
 - Database: InfluxDb instance database
- **Metadata.** Additional ELCM instance metadata, currently used for values sent to InfluxDb:
 - HostIp: IP address of the machine where the ELCM is running
 - Facility: Facility name (or platform)

5.1.2.4 Facility Configuration

Starting with version 1.1, the facility is no longer configured by a single `facility.yml` file. UEs must now be defined using separate files in the `UEs` sub-folder, while TestCases (and their dashboards) are defined in the `TestCases` sub-folder.

The available values for each element have not changed, so it is possible to move each key on the `facility.yml` file to their respective folders and have the same configuration as before the update.

The contents of the `UEs` and `TestCases` sub-folders describe the behavior of the 5GENESIS platform when an Experiment execution request is received. These folders will be automatically generated if they do not exist. The ELCM will load the contents of every `yml` 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 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 a `Config` dictionary, which is different for every task. More information about the composition process can be found in section 3.2 of Deliverable D3.15 [31], but please note that this example uses the old `facility.yml` file, but the behaviour is the same. The following is an example of a *yaml* file that configures an UE:

```
TestUE:
  - Order: 1
```

```

Task: Run.Dummy
Config:
  Message: This is a dummy entity initialization
- Order: 10
Task: Run.Dummy
Config:
  Message: This is a dummy entity closure

```

- **TestCases and Dashboards.** Similarly to the UEs, the files in the 'TestCases' folder define the actions required in order to execute a certain TestCase included in the Experiment descriptor. The first key, which defines the name of the TestCase, follows the same format as in the UEs section, however, these files can contain an additional Dashboard key that define the list of panels that will be generated as part of the Grafana dashboard that corresponds to the TestCase.

The following values can be set for each panel:

- [Mandatory] Type: 'singlestat' (gauges or single value) or 'graph' (time series)
- [Optional] Name: Panel name, '{Measurement}: {Field}' if not set
- [Mandatory] Measurement: Measurement (table) name
- [Mandatory] Field: Field (column) name
- [Optional] Unit: Field unit
- [Mandatory] Size: (As list) [,]
- [Mandatory] Position: (As list) [,]
- [Optional] Color: Graph or text color(s). For Gauges this is a list of 3 colors, otherwise a single value. Each color can be defined using these formats: "#rrggbb" or "rgba(rrr, ggg, bbb, a.aa)"

Additional values for the 'graph' type:

- [Mandatory] Lines: True to display as lines, False to display as bars
- [Mandatory] Percentage: Whether the field is a percentage or not
- [Optional] Interval: Time interval of the graph, default \$__interval if not set
- [Optional] Dots: Display dots along with the graph or bar

Additional values for the 'singlestat' type:

- [Mandatory] Gauge: True to display as a gauge, false to display as numeric value
- [Optional] MaxValue: Max expected value of the gauge, 100 if not set
- [Optional] MinValue: Min expected value of the gauge, 0 if not set

The following is an example TestCase file:

```

Slice Creation:
- Order: 5
Task: Run.SliceCreationTime
Config:
  ExperimentId: "@{ExperimentId}"
  WaitForRunning: True
  Timeout: 60
  SliceId: "@{SliceId}"
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]
```

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.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 (DEBUG, INFO, WARNING, ERROR, CRITICAL)
 - 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 variable expansion `@[key]`. 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 this case 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: @[Publish2];
3: @[Publish3]; 4: @[Publish4:NoProblem]" }
```

Will produce this message in the log:

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

- **Run.SliceCreationTime.** Sends the Slice Creation Time reported by the Slice Manager to InfluxDb. Configuration values:

- `ExecutionId`: Id of the execution (can be dynamically 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 dynamically expanded from `@{SliceId}`)
- **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 or not 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
```

It is 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.
- `@{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`)

5.1.2.5 PDF Report generation

It is possible to integrate an instance of Grafana reporter [23] 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. An example of a custom template for the PDF report generation can be seen in Annex 1 – Example Template for Grafana reporter.

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*.

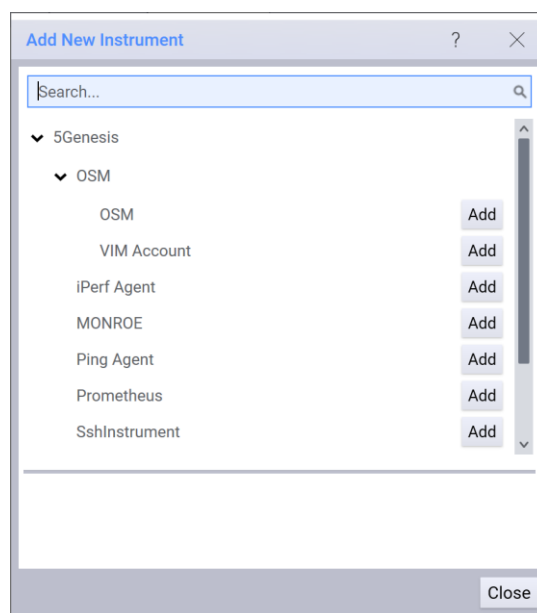


Figure 20 5GENESIS related OpenTap instruments for NFV MANO (OSM)

1.1.1.1. 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.

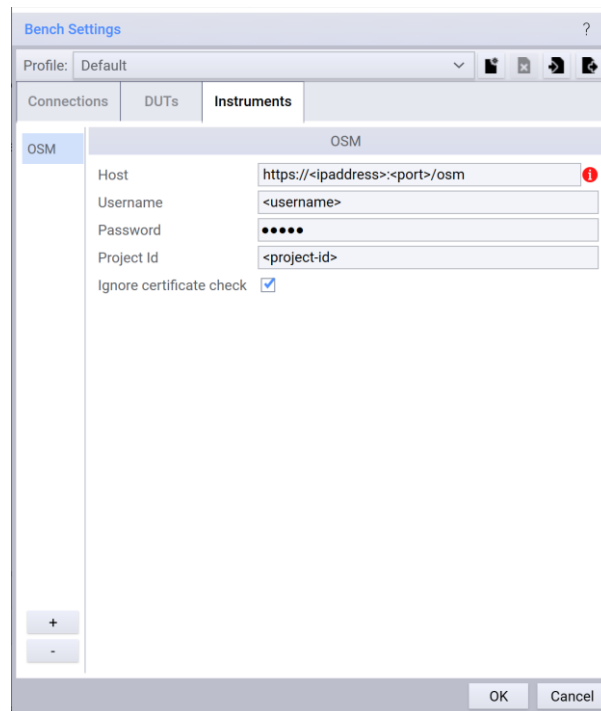


Figure 21 OSM Instrument

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; }
```

1.1.1.2. 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.

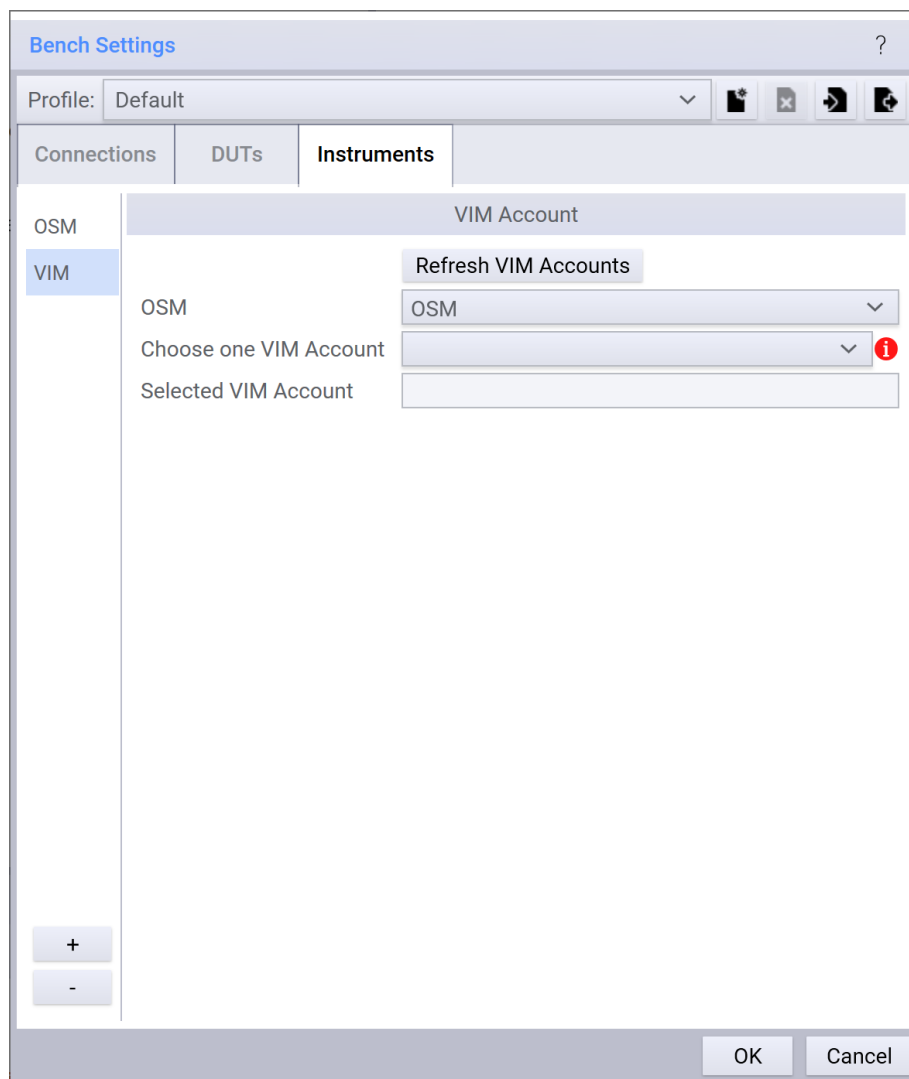


Figure 22 VIM Instrument

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<VimInfo> AvailableVimInfos { 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. This is illustrated in Figure 23.

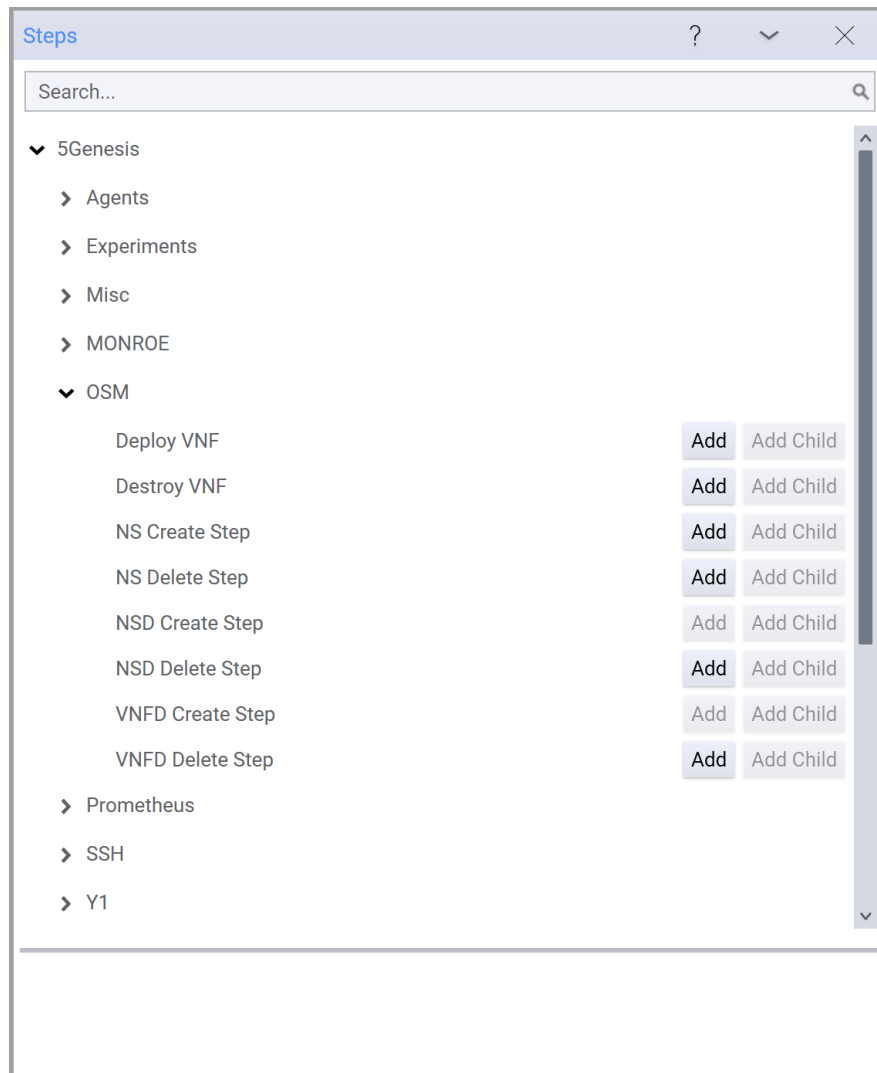


Figure 23 OSM Test Steps

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 24.

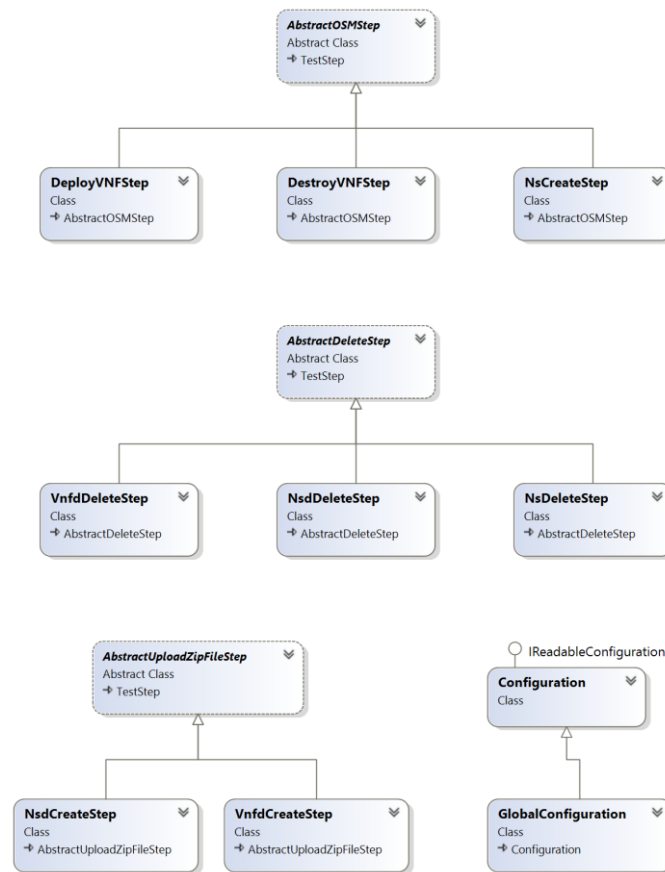


Figure 24 Class hierarchy of the OpenTap OSM Test Steps

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.

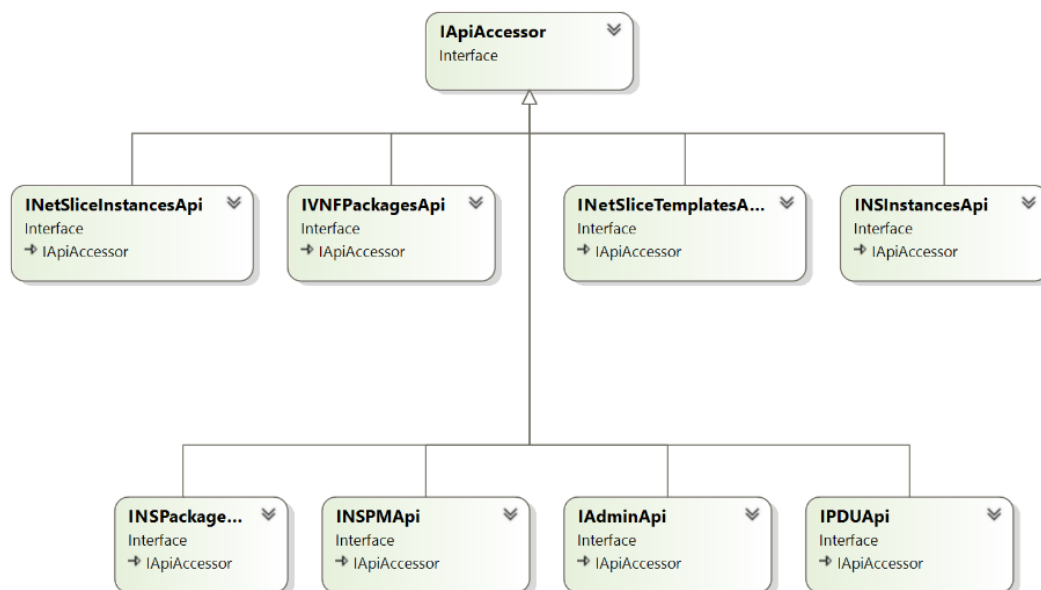


Figure 25 Class hierarchy of the generated OSM API accessors from its OpenAPI definition

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 A” of the 5GENESIS M&A framework has been thus designed and developed, as documented in the project deliverable D3.5 [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 [45] and MONROE [46], 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 [29] 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 [32].

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 [33].

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 26.

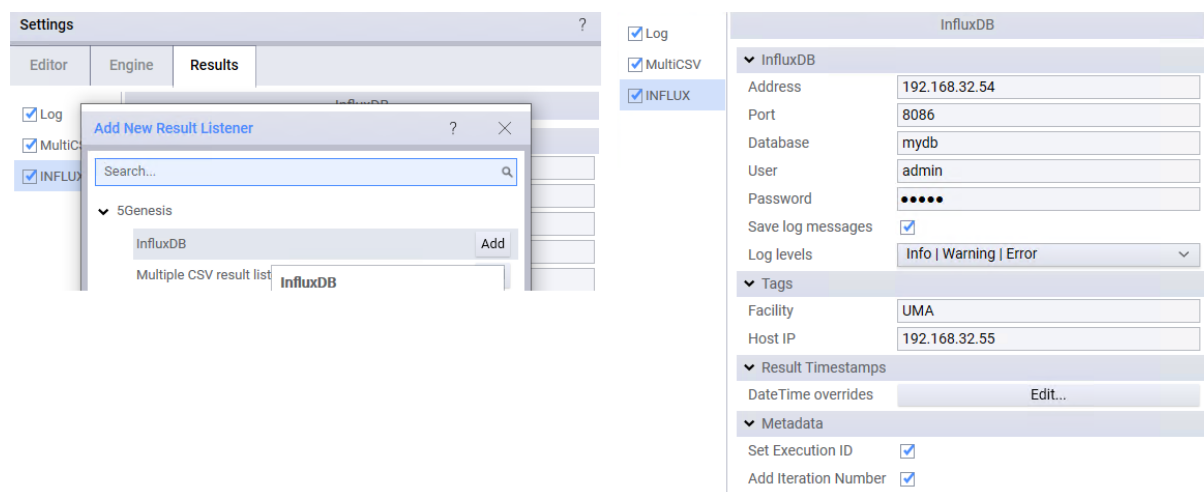


Figure 26 InfluxDB result listener configuration

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 [31]. 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 [38].

5.3.2 Prometheus Infrastructure Monitoring

Prometheus [19] 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 .yml 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 [34].

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 [35]. 5GENESIS will make use of node-exporters to monitor its infrastructure, as for example with the SNMP Exporter [36] for network monitoring.

Additionally, an OpenTAP plugin for Prometheus (Figure 17 on Section 4.1.1.1.1) 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 retriever 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 27, 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:

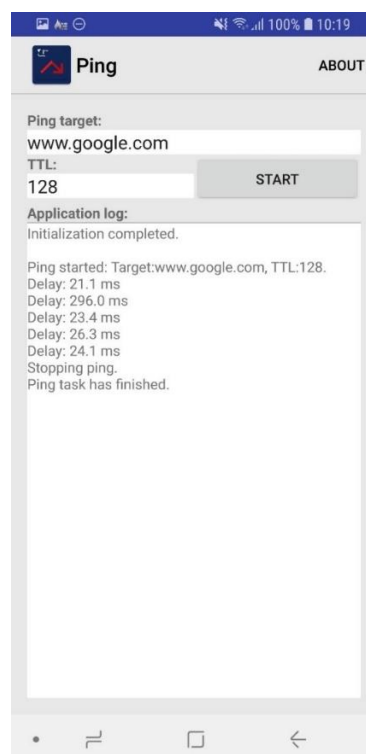


Figure 27 Ping application interface

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.

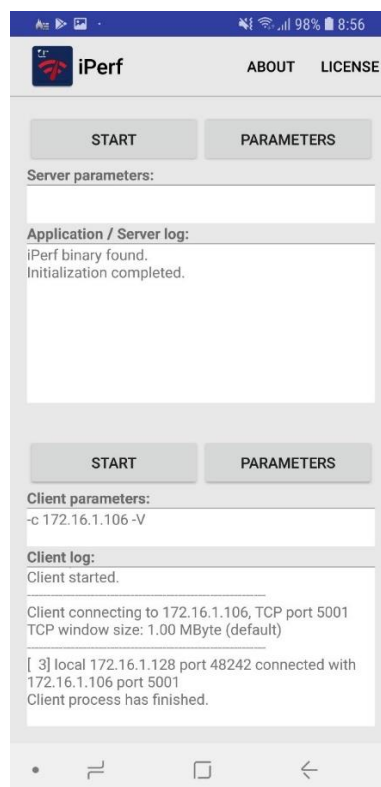


Figure 28 iPerf application interface

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. The user interface can be seen in Figure 28.

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

Parameters for the iPerf binary are passed as a comma separated string (with no spaces) using the intent extra `-e com.uma.iperf.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.iperf/.iPerfService -a
com.uma.iperf.CLIENTSTART -e com.uma.iperf.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.iperf/.iPerfService -a
com.uma.iperf.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 29, 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.

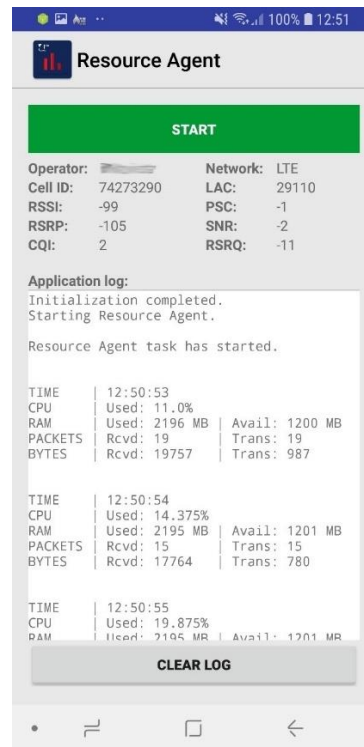


Figure 29 Resource Agent application interface

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:

The image shows two parts of the 5Genesis interface. On the left, the 'Add New Instrument' dialog is open, showing a search bar and a list of instruments under the '5Genesis' category. 'iPerf Agent' and 'Ping Agent' are highlighted with red boxes, each with an 'Add' button. On the right, there are two configuration forms. The top form is for the 'iPerf Agent', showing fields for 'IP' (127.0.0.1) and 'Agent Port' (5000). The bottom form is for the 'Ping Agent', showing fields for 'IP' (127.0.0.1) and 'Agent Port' (5555).

Figure 30 Remote PC agents TAP instrument and configuration

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

The image shows two configuration panels for agents. The left panel is for the 'iPerf Agent' (Agent: iPerfA, Action: Measure). It has sections for 'Parameters' (Role: Client, Host: 127.0.0.1, Port: 5001, Max Run Time: 99999 s, Extra Parameters: empty), 'Measurement' (Wait Mode: Time, Time: 4 s), 'Errors' (Verdict on error: Not Set), and 'Common' (Enabled: checked, Step Name: iPerf Agent). The right panel is for the 'Ping Agent' (Agent: PingA, Action: Measure). It has sections for 'Parameters' (Target: 8.8.8.8, Packet Size: 0), 'Measurement' (Wait Mode: Time, Time: 4 s), 'Errors' (Verdict on error: Not Set), and 'Common' (Enabled: checked, Step Name: Ping Agent).

Figure 31 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 [46]):

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.

Ping Container (monroe/ping:virt)

The Ping container² 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/',
```

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


```
'nodeid': 'fake.nodeid',
'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 container³ 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:

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

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

In its Release A, 5GENESIS Analytics is based on Python. 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://gitlab.fokus.fraunhofer.de/5genesis/analytics> (branch "Release A" for the Release A components).

Regarding the statistical analysis, 5GENESIS Analytics currently includes two scripts, working with KPI data stored either in csv files or in a platform-specific InfluxDB instance. Examples of usage are embedded in the scripts.

As regards the ML-oriented analyses, 5GENESIS Analytics currently includes scripts for: time series management (synchronization), outlier detection (via Z-score and Median Absolute Deviation), linear correlation, linear and Support Vector Regression (SVR), Random Forest, and feature selection (several options available). For Release A, a “main.py” script is shared and includes several examples of usage for the above functionalities. In Release B, a full chain ML Analytics usage would include:

- Connect to a platform specific InfluxDB instance.
- Retrieve the heterogeneous data (multiple time series) collected during an experiment.
- Sync the time series in time.
- Discard possible outliers in the series.
- Perform the required analysis, e.g., evaluate linear correlation between each possible pair of time series.

The connection to a platform-specific InfluxDB instance is achieved through the use of a pre-existing InfluxDB-Python client, for which code and documentation can be found at <https://github.com/influxdata/influxdb-python>. The client basically reproduces in Python the InfluxDB native querying language (InfluxQL), thus allowing both read/write connections to remote InfluxDB instances from within Python.

5.4 Slice Manager Deployment

Katana is a Slice Manager component developed for the 5GENESIS project. Katana Slice Manager is a central software component responsible for controlling all the devices comprising the network, providing an interface for creating, modifying, monitoring and deleting slices. Through the NBI, Katana interacts with a Coordination Layer or directly with the network operator. It receives the Network Slice Template (NEST) for creating network slices and provides the API for managing and monitoring them. Through the SBI, it talks to the components of the MANO Layer, namely the NFVO, the VIM, the EMS and the WIM, in order to manage the functions in the network and perform CRUD (Create, Read, Update and Delete) operations on end-to-end network slices.

The requirements for the Katana Slice Manager used in 5GENESIS are the following:

- docker version \geq 18.09.6
- docker-compose version \geq 1.17.1
- Hardware resources: 2 vCPUs, 4GB RAM, 40GB Disk

The necessary steps for the installation and execution of this service are the following:

1. Create a VM with the recommended resources and install Docker [39] and Docker Compose [40]
2. Get the source code from the Gitlab repository

```
git clone https://gitlab.fokus.fraunhofer.de/5genesis/slice-manager.git
```

```
cd slice-manager
```

3. Checkout to the Release_A Branch

```
git checkout Release_A
```

4. Run the Installation Script

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

5. Start katana Slice Manager

```
./start.sh
```

Other additional commands used with Katana Slice Manager are:

- Start Katana Slice Manager service and the web UI module, run:

```
./start-ui.sh
```

- Stop Katana service, but keep the databases with any associated data:

```
./stop.sh
```

- Stop Katana service, and clean any associated data:

```
./clear.sh
```

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

1. Refer to the example configuration files [41] and fill them with the proper values depending on each platform

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

```
katana vim add -f <vim_conf.json>
```

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

```
katana nfvo add -f <nfvo_conf.json>
```

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

```
katana wim add -f <wim_conf.json>
```

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

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

6. Create a slice giving the Network Slice Template:

```
katana slice add -f <slice_conf.json>
```

7. You can check the configured components/slices:

```
katana <component/slice> ls
```

8. You can inspect an added component/slices:

```
katana <component/slice> inspect <component_id/slice_id>
```

9. You can check the deployment time of an instantiated slice:

```
katana slice deployment_time <slice_id>
```

10. Delete a component/slice:

```
katana <<component/slice> rm <component_id/slice_id>
```

Note that WIM and EMS are optional configurations for the Release A. In addition to that, creation of VIM is tested with OpenStack (Rocky Release). Subsequent releases will be supported in later versions of the Slice Manager. More information on the Katana Slice Manager and the Slice Manager service in 5GENESIS can be found in the Wiki of the Katana repository [42] and in the corresponding deliverable D3.3 [43].

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
- [3] 5GENESIS, Deliverable D2.2 "Initial overall facility design and specifications" [Online], https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS_D2.2_v1.0.pdf
- [4] 5GENESIS, Deliverable D2.3 "Initial planning of tests and experimentation" [Online], https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS_D2.3_v1.0.pdf
- [5] 5GENESIS Consortium, "D3.1 Management and orchestration (Release A)," [Online]. Available: https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.1_v1.0.pdf.
- [6] 5GENESIS Consortium, "D3.3 Slice management WP3 (Release A)," [Online]. Available: https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.3_v1.0.pdf.
- [7] 5GENESIS, Deliverable D3.5 "Monitoring and Analytics (Release A)" [Online], https://5genesis.eu/wp-content/uploads/2019/10/5GENESIS_D3.5_v1.0.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.
- [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.
- [10] 5GENESIS Deliverable D3.13 "Security Framework – Release A", 2019
- [11] 5GENESIS Deliverable D4.2 "The Athens Platform," [Online]. Available: 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
- [13] 5GENESIS Deliverable D4.8 "The Limassol Platform," [Online]. Available: 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
- [15] 5GENESIS Deliverable D4.14 "The Berlin Platform," [Online]. Available: https://5genesis.eu/wp-content/uploads/2020/02/5GENESIS_D4.14_v1.0.pdf
- [16] 5GENESIS, Deliverable D6.1 "Trials and experimentation -cycle 1" [Online], https://5genesis.eu/wp-content/uploads/2019/12/5GENESIS_D6.1_v2.00.pdf
- [17] Curl, <https://curl.haxx.se/>
- [18] Open TAP, : <https://doc.opentap.io/Developer%20Guide/Introduction/>
- [19] Prometheus, <https://prometheus.io/>
- [20] Restsharp: <http://restsharp.org/>
- [21] Python, <https://www.python.org/>
- [22] Grafana, <https://grafana.com/>
- [23] Grafana Reporter, <https://github.com/IzakMarais/reporter>
- [24] Python virtualenv, <https://virtualenv.pypa.io/en/stable/>

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

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"
    },
    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"
    }
  },
  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"
]
}
```