



**EXPERIMENTATION AND VALIDATION OPENNESS FOR LONGTERM
EVOLUTION OF VERTICAL INDUSTRIES IN 5G ERA AND BEYOND**

[H2020 - Grant Agreement No.101016608]

Deliverable D3.3

Implementations and integrations towards EVOLVED-5G framework realisation (final)

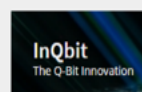
Editor R. Marco Alaez (ATOS)

Contributors ATOS, NCSRD, UMA, FOG, MAG, TID, UPV, INTRA,
LNV

Version 1.0

Date April 31st, 2023

Distribution PUBLIC (PU)



DISCLAIMER

This document contains confidential information reserved for the partners of the EVOLVED-5G ("Experimentation and Validation Openness for Longterm evolution of VErtical inDUstries in 5G era and beyond) Consortium and is subject to the confidentiality obligations set out in the Grant Agreement 101016608 and to the EVOLVED-5G Consortium Agreement.

Neither this document nor the information contained herein shall be used, copied, duplicated, reproduced, modified, or communicated by any means to any third party, in whole or in parts, except with prior written consent of the EVOLVED-5G Consortium. In such case, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. In the event of infringement, the consortium reserves the right to take any legal action it deems appropriate.

This document reflects only the authors' view and does not necessarily reflect the view of the European Commission. Neither the EVOLVED-5G Consortium as a whole, nor a certain party of the EVOLVED-5G Consortium warrant that the information contained in this document is suitable for use, nor that the use of the information is accurate or free from risk and accepts no liability for loss or damage suffered by any person using this information.

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 user thereof uses the information at its sole risk and liability.

REVISION HISTORY

Revision	Date	Responsible	Comment
0.1	March 13 th 2023	R. Marco (ATOS)	ToC
0.2	April 14 th 2023	R. Marco (ATOS)	Finished contributions
0.3	April 19 th 2023	R. Marco (ATOS)	Finished internal review
0.4	April 25 th 2023	R. Marco (ATOS)	Internal review comments addressed
0.6	April 28 th 2023	R. Marco (ATOS)	SC review
0.7	May 10 th 2023	R. Marco (ATOS)	Final review
0.8	May 11 th 2023	R. Marco (ATOS)	Final version

LIST OF AUTHORS

<i>Partner</i>	<i>Partner</i>	<i>Name & Surname</i>
<i>ATOS</i>	<i>ATOS IT SOLUTIONS AND SERVICES IBERIA SL</i>	<i>Ricardo Marco Sonia Castro</i>
<i>MAG</i>	<i>MAGGIOLI SPA</i>	<i>Yannis Karadimas, Alexandros Tzoumas</i>
<i>UMA</i>	<i>UNIVERSIDAD DE MÁLAGA</i>	<i>Bruno García García Francisco Luque Schempp Jorge Márquez Ortega M^a del Mar Moreno M^a del Mar Gallardo Laura Panizo Jaime Pedro Merino Gomez</i>
<i>NCSRD</i>	<i>NATIONAL CENTER FOR SCIENTIFIC RESEARCH "DEMOKRITOS"</i>	<i>Dimitrios Fragkos, George Makropoulos, Harilaos Koumaras, Anastasios Gogos</i>
<i>UPV</i>	<i>UNIVERSITAT POLITECNICA DE VALENCIA</i>	<i>Regel González Usach</i>
<i>FOGUS</i>	<i>FOGUS INNOVATIONS & SERVICES P.C.</i>	<i>Dimitris Tsolkas</i>
<i>TID</i>	<i>TELEFONICA INVESTIGACION Y DESARROLLO SA</i>	<i>David Artuñedo Javier Garcia</i>
<i>INTRA</i>	<i>INTRASOFT INTERNATIONAL SA</i>	<i>Angela Dimitriou</i>

GLOSSARY

<i>Abbreviations/Acronym</i>	<i>Description</i>
3GPP	<i>Third Generation Partnership Project</i>
5G-NPN	<i>5G Non-Public Network</i>
5GS	<i>5G System</i>
AEF	<i>API Exposing Function</i>
AF	<i>Application Function</i>
AMF	<i>Mobility Management Function</i>
API	<i>Application Programming Interface</i>
AUSF	<i>Authentication Server Function</i>
BBU	<i>BaseBand Unit</i>
CAPIF	<i>Common API Framework</i>
CCF	<i>CAPIF Core Function</i>
CI/CD	<i>Continuous Integration and Continuous Deployment</i>
CLI	<i>Command Line Interface</i>
CNC	<i>Centralized Network Configuration</i>
CUC	<i>Centralized User Configuration</i>
ELCM	<i>Experiment Life-Cycle Manager</i>
FOF	<i>Factories of the Future</i>
HTTPS	<i>Hypertext Transfer Protocol Secure</i>
KPIs	<i>Key Performance Indicator</i>
LTE	<i>Long Term Evolution</i>
MITM	<i>Man in the Middle</i>
NEF	<i>Network Exposure Function</i>
Network App	<i>Network Application</i>
NSA	<i>Non-StandAlone</i>
PKI	<i>Public Key Infrastructure</i>
PSK	<i>Pre Shared Key</i>
RAN	<i>Radio Access Network</i>
RRU	<i>Remote Radio Unit</i>
SA	<i>StandAlone</i>
SDK	<i>Software Development Kit</i>
SMF	<i>Session Management Function</i>
SQL	<i>Structured Query Language</i>
SSH	<i>Secure Shell</i>
SUPI	<i>Subscription Permanent Identifier</i>
TSN	<i>Time Sensitive Network</i>
UDM	<i>User Data Management</i>
UE	<i>User Equipment</i>
VNFs	<i>Virtual Network Functions</i>
XSS	<i>Cross Site Scripting</i>

EXECUTIVE SUMMARY

This document contains information, which is proprietary to the EVOLVED-5G ("Experimentation and Validation Openness for Longterm evolution of VErtical inDustries in 5G era and beyond) Consortium that is subject to the rights and obligations and to the terms and conditions applicable to the Grant Agreement number: 101016608. The action of the EVOLVED-5G Consortium is funded by the European Commission.

Neither this document nor the information contained herein shall be used, copied, duplicated, reproduced, modified, or communicated by any means to any third party, in whole or in parts, except with prior written consent of the EVOLVED-5G Consortium. In such case, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. In the event of infringement, the consortium reserves the right to take any legal action it deems appropriate.

This document reflects only the authors' view and does not necessarily reflect the view of the European Commission. Neither the EVOLVED-5G Consortium as a whole, nor a certain party of the EVOLVED-5G Consortium warrant that the information contained in this document is suitable for use, nor that the use of the information is accurate or free from risk and accepts no liability for loss or damage suffered by any person using this information.

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 user thereof uses the information at its sole risk and liability.

TABLE OF CONTENTS

TABLE OF CONTENTS	6
1 INTRODUCTION	1
1.1 Purpose	1
1.2 Target audience	1
1.3 Structure	1
2 The EVOLVED-5G Facility	2
2.1 General description.....	2
2.2 Network Apps and implementation Evolution.....	3
2.3 Architecture Evolution	4
3 Community	5
3.1 EVOLVED-5G Wiki	6
3.1.1 Component Description	8
3.1.1.1 Wiki Frontend	8
3.1.1.2 Wiki Backend.....	10
3.1.2 Wiki Implementation	11
3.1.3 Wiki Security	11
4 Workspace Environment	13
4.1 Component and Workflow description.....	13
4.1.1 Development tools.....	15
4.1.1.1 Network App Template	15
4.1.1.2 SDK.....	15
4.1.1.3 Open repositories	16
4.1.1.4 Dummy Network Application	16
4.1.2 Verification tools – CI/CD Pipelines.....	16
4.1.2.1 SonarQube	16
4.1.2.2 Nmap (Network App open ports).....	17
4.1.2.3 Trivy	17
4.1.2.4 Build, deploy, destroy	17
4.1.2.5 CAPIF	18
4.1.2.6 NEF.....	18
4.1.2.7 TSN.....	18

4.1.2.8	Robot Framework	19
4.2	Workspace Implementation	19
4.2.1	Development Phase	20
4.2.1.1	Network App template	20
4.2.1.2	SDK.....	21
4.2.1.3	Dummy Network Application.....	26
4.2.2	Verification Phase	27
4.2.2.1	Network App-CAPIF Interaction	30
4.2.2.2	Network App-NEF Emulator Interaction	31
4.2.2.3	Network App-TSN APIs Interaction	32
4.3	Security	34
4.3.1	Development Phase	34
4.3.2	Verification Phase	34
4.3.2.1	Security in Jenkins.....	35
4.3.2.2	Security in Open Repository.....	35
4.3.2.3	Security in OpenShift	35
4.3.2.4	Security in NEF	36
4.3.2.5	Security in CAPIF	37
4.3.2.6	Security in TSN	38
5	Validation environment	38
5.1	Component description	39
5.2	Implementation	40
5.2.1	Validation Phase.....	40
5.2.1.1	ELCM	43
5.2.1.2	CI/CD services - Validation environment Interaction	44
5.3	Security	46
6	EVOLVED-5G Infrastructure Evolution	46
6.1	Component Description.....	46
6.1.1	Athens platform evolution	46
6.1.1.1	Core Network.....	47
6.1.1.2	5G RAN.....	47
6.1.1.3	Kubernetes cluster implementation	48
6.1.2	Málaga platform evolution	49



D3.3 Implementations and integrations towards EVOLVED-5G framework realisation (final)

GA Number 101016608

6.1.2.1	Kubernetes cluster implementation	49
6.1.2.2	Time-Sensitive Networking (TSN) over 5G	49
6.1.3	CAPIF Core Function Tool evolution.....	51
6.1.3.1	CAPIF Core Function Tool implementation	52
6.1.4	TSN FrontEnd	54
6.1.4.1	TSN FrontEnd implementation	54
6.1.5	NEF Emulator evolution	56
6.1.5.1	Companion application	56
6.1.5.2	NEF Emulator Implementation	57
7	Conclusion	59
REFERENCES.....		60

LIST OF FIGURES

Figure 1 EVOLVED-5G reference architecture	5
Figure 2 EVOLVED-5G Wiki.....	7
Figure 3 Topics and structure of contents of the EVOLVED-5G Wiki	8
Figure 4 EVOLVED-5G Wiki public-facing front-end view.....	9
Figure 5 Main public-facing front-end elements of the EVOLVED-5G Wiki	10
Figure 6 EVOLVED-5G Wiki administration dashboard	10
Figure 7 Workspace functional blocks	13
Figure 8 Workspace workflow	14
Figure 9 Fully centralized TSN network architecture - IEEE 802.1Qcc.....	19
Figure 10 Old Network App structure vs New Network App structure	20
Figure 11 Old Inputs vs New inputs to create a Network App.....	21
Figure 12 Location Subscriber - Subscribe to monitor location changes.....	23
Figure 13 Location Subscriber - Request location for a given device.....	23
Figure 14 ConnectionMonitor- Subscribe monitor network connectivity.....	24
Figure 15 QoS Awareness- Subscribe to monitor changes to the QoS thresholds.....	25
Figure 16 List Dummy Network App files.....	27
Figure 17 NEF verification pipeline (part I).....	28
Figure 18 NEF verification pipeline (part II)	28
Figure 19 TSN verification pipeline.....	29
Figure 20 Parameterization of the NEF verification pipeline.....	29
Figure 21 CAPIF invocation in the verification pipelines.....	31
Figure 22 NEF invocation in the verification pipeline	32
Figure 23 Implementation of the TSN testing plan in the TSN verification pipeline	33
Figure 24 Jenkins Market Share numbers	35
Figure 25 OpenShift console showing User Identity (upper right corner) and project permissions	36
Figure 26 Validation Environment architecture	39
Figure 27 Validation workflow	42
Figure 28 Ericsson 5G RAN.....	48
Figure 29 Athens Kubernetes Cluster.....	49
Figure 30 TSN over 5G testbed at the Málaga platform	50
Figure 31 FT980-WW, One Plus 11 5G and Google Pixel 7 5G	51
Figure 32 NGINX implements TLS mutual authentication and routes API requests to CAPIF Core Function Services	53
Figure 33 Companion App	57
Figure 34 NEF Emulator	58

LIST OF TABLES

Table 1 SDK versions mapped with features	3
Table 2 NEF Security features.....	36
Table 3 Open APIs experiment management endpoints	45
Table 4 Open APIs result retrieval endpoints.....	45
Table 5 TSN FrontEnd API endpoints	54
Table 6 TSN AF API endpoints	55

1 INTRODUCTION

1.1 PURPOSE

This deliverable is the third of a series of four reports to be delivered by WP3. The main goal of this third report, titled “*D3.3 Implementations and integrations towards EVOLVED-5G framework realization (final)*” is threefold:

- To provide a comprehensive view on the final version of the EVOLVED-5G facility, focusing on the final releases of the components and tools implemented during the lifetime of the project within WP3 (30 months), including TSN capabilities.
- To describe final implementation details of the development, verification, and validation environments related to the lifecycle of the Network Apps in the EVOLVED-5G facility, including tools and technologies used as well as an updated version of the architecture.
- Detail the evolution of the infrastructure of both EVOLVED-5G sites, Athens and Málaga.

D3.3 describes final outcomes for three out of the four different tasks that compose WP3:

- “T3.1 Production of the Workspace and the Related Verification Tools”, focused on the SDK tools developed for the creation and verification of Network Apps.
- “T3.2 Development of Network App Validation Tools and Open Repository”, related to the store and validation of Network Apps
- “T3.3 Overall Framework Integration and 5G Infrastructure Evolution”, devoted, as indicated by its name, to the integration of all environments that compose the EVOLVED-5G facility on top of the available 5G infrastructure.

And it complements previous work delivered in D3.1. “*D3.1 Implementations and integrations towards EVOLVED-5G framework realization (intermediate)*”, submitted in M12.

1.2 TARGET AUDIENCE

As its counterpart D3.1, this deliverable has been conceived to be public, as its intention is to provide information about the EVOLVED-5G facility to a broad variety of research individuals and communities. Hence, the target audience is the same as described in D3.1: Project Consortium; the funding European Commission organisation; the Industry 4.0 vertical and, in particular, Industry 4.0 developers and Factories of the Future (FoF) vertical groups; any other vertical industries and groups that may benefit from 5G technologies; the scientific audience; and the general public.

1.3 STRUCTURE

The deliverable is organized as follows:

- **Section 1. Introduction.** It presents the deliverable, providing information about its target audience, purpose, and structure.
- **Section 2. The EVOLVED-5G Facility.** It provides a general view of the EVOLVED-5G facility, presents the latest version of its architecture, and describes the evolution of the conceptualization and implementation of the Network Apps.
- **Section 3. Community.** It provides information about the different tools that have been developed by the project with the goal of creating a community around the EVOLVED-5G technical vision and outcomes: the Wiki, the Forum, and the Library.
- **Section 4. Workspace.** It is dedicated to the description of the workspace environment, that integrates two different phases of the Network App lifecycle: development and verification. For each phase, a description of the tools that compose it, their implementation and security aspects are provided.
- **Section 5. Validation.** It is focused on the next phase of the Network App lifecycle: validation. As in the previous section, it describes its tools, their implementation, and security-related aspects.
- **Section 6. 5G Infrastructure evolution.** It provides an overview about how the 5G infrastructure that supports the EVOLVED-5G facility, both in Málaga and Athens, has evolved since the submission of deliverable D3.1. This includes a description about the implementation of the Kubernetes clusters in both sites, the improvement of the CAPIF Core Function Tool and the NEF Emulator, the TSN Application Function.
- **Section 7. Conclusion.** It closes the report with final impressions, main conclusions and future work.

2 THE EVOLVED-5G FACILITY

2.1 GENERAL DESCRIPTION

The final architectural view (including all the relations between environments, components and tools) that realizes the EVOLVED-5G facility was previously depicted in Deliverable D2.3 “*Overall framework for NetApp development and evaluation*” [46], which was successfully submitted in M21 (October 2022). Thus, in this deliverable the architectural upgrades only focus on the delta created after D2.3 submission, including adaptations, extensions of some additional functionalities, and the addition of **TSN capabilities**. These updates are indicated in Section 2.3, by providing the most recent and complete architectural diagram of the EVOLVED-5G approach (Figure 1). In line with the previous versions of the architecture, EVOLVED-5G has been keeping active all the design principles previously followed in WP2 for the conceptualization, creation and refinement of the architecture:

- System components are arranged at different levels of abstraction, that create the compositional and structural logic of the EVOLVED-5G Architecture. From abstract to concrete, we find:

- Tier 1 – Environments (Red): They refer to larger bundles of components and tools that either support a specific phase of the Network App lifecycle or give support to other environments.
- Tier 2 – Functional Blocks (Green): Encapsulate related functionalities that are part of an environments.
- Tier 3 – Tools and Functionalities (Brown): Are components that support de implementation of the different capabilities.
- The environments are associated with the different phases of the Network App lifecycle. These are the **Workspace** (development and verification), **Validation Environment** (validation), **Certification Environment** (certification), and **Marketplace**. The **5G-NPN** is an additional environment that supports both the Validation and Certification environments.
- All the environments are interconnected with three central elements: The CI/CD services, the Open Repository, and the Community.

Further details and specifics about the conceptualization of the architecture and relations between the three tiers can be found in Deliverable D2.3, Section 2. This includes the description of each environment and component, the indicative usage scenario and a description of the integration design. This information is still relevant and up-to-date, with the exception of the small changes described in the present deliverable (Section 2.3.)

2.2 NETWORK APPS AND IMPLEMENTATION EVOLUTION

Previous deliverables D2.1 [57], D2.2 [51], and D3.1 [15] described the concept of the Network App, which has since undergone updates to ensure that it aligns with all the components of the EVOLVED-5G ecosystem. The main goal of the Network App is to communicate securely and efficiently with the 5G Core, specifically with the NEF Emulator and more recently, with the TSN Frontend, both through the CAPIF in a secured and standardised way. To achieve this, the Network App utilises the SDK libraries provided in the project to establish a communication link with the components. In addition to communication, the Network App developers also use a CLI tool to navigate through the different stages of the CI/CD framework, including verification, validation, and certification. The final objective is to publish the certified Network Apps in the EVOLVED-5G Marketplace, which is a platform for sharing and distributing 5G applications and will be further described in the last deliverable of WP3, D3.4.

It is important to note that the SDK developed in EVOLVED-5G is not a single version, but rather a collection of different versions of the libraries that have been updated and upgraded throughout the lifetime of the project. Indeed, it is the software development conceived as an iterative process that involves continuous improvements and updates based on new requirements, feedback, and bug fixes. The different versions of the SDK libraries used by the Network App developers are mapped with specific features and functionalities as presented in Table 1.

Table 1 SDK versions mapped with features

SDK versions	Features Involved
SDK 0.6.0	<ul style="list-style-type: none"> • Network App template • CLI

SDK v0.8.9	<ul style="list-style-type: none"> • CAPIF v2.1 • NEF v1.6.2
SDK v1.0.0	<ul style="list-style-type: none"> • CAPIF v3.0 • NEF v2.0 • TSN API v1.0
SDK v1.1	<ul style="list-style-type: none"> • CAPIF 3.1 • NEF v2.1 • TSN API v.1.0 • Companion application v1.0

As it can be seen from Table 1 there has been different releases of the SDK from the first one (v0.6.0) included in D3.1 [15] up to the actual v1.1. Table 1 highlights the main releases that have been used by developers. In between, different versions have been released [61] as well, fixing bugs, including different features gathered from WP4 and WP5 requirements. The initial version just included the creation of the Network App repository in GitHub (CLI + Network App template) while the following releases have included more features such as, SDK Libraries compatible with first NEF (v1.6.2) and CAPIF (v2.1) versions, Network App verification tests for NEF and CAPIF, reaching finally the latest version of the SDK v1.1 which provides compatibility with latest version of CAPIF, NEF and companion application, also new libraries for TSN. The progress is showcasing the delta from an initial and basic version, until the maturity level reached integrating all the tools developed in WP3 and consequently extending their utilisation not only to Development and Verification phase (see Section 4) but also to Validation phase (see Section 5).

The evolution of the development tools is presented in the following sections, which includes the SDK libraries, the dummy Network App, and CLI tools in Section 4.1.1. The evolution of the components can be found in Section 6.1.3 for CAPIF, Section 6.1.5 for NEF Emulator, and Section 6.1.4 for TSN FrontEnd. The companion application is presented in Section 6.1.5.1. Finally, Section 4.2.2 covers the verification phase that presents the interaction between the Network App and CAPIF, NEF, and TSN.

2.3 ARCHITECTURE EVOLUTION

This section gives an overview on those changes undertaken in the final architecture of the EVOLVED-5G described in WP2. These changes respond to small variations introduced in the 5G-NPN environment (i.e., the exposure of TSN capabilities and refinements in Exposure Services).

The Exposure Services are arranged with the CAPIF Services in the front of the inbound API communication, in order to emphasize the role of CAPIF as frontend for both the NEF Services and the TSN FrontEnd.

The **TSN FrontEnd** is a new component that is part of the TSN infrastructure inside the 5G-NPN. This component acts as a standardized entry point for the configuration of the TSN capabilities. The TSN FrontEnd makes use of an internal API to communicate with the TSN AF (not depicted in Figure 1, but implemented as part of the *TSN over 5G* element), which is optional and platform-specific. More information about this separation and the implementation of the TSN communication can be seen in Section 6.1.4.

Finally, the radio access network has been separated in two different paths, one based on the standard 5G NR, without deterministic communication, and another using TSN over 5G, if available.

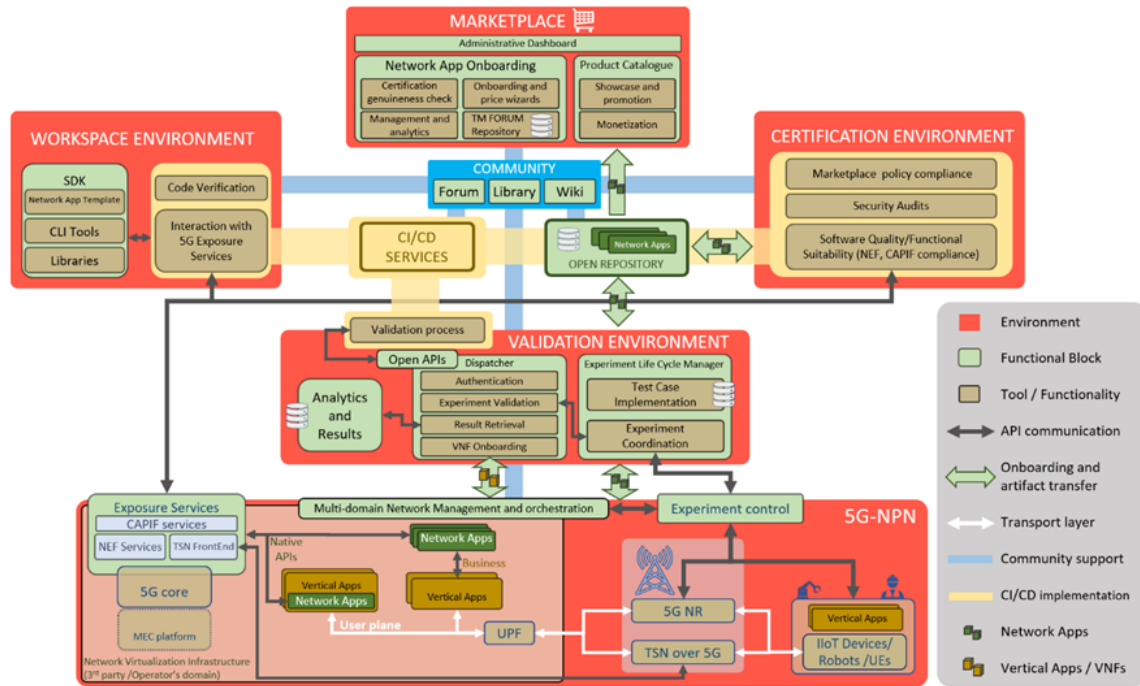


Figure 1 EVOLVED-5G reference architecture

3 COMMUNITY

Throughout technological and business history, it has been repeatedly observed that the generation of innovative advancements and notable technical developments is not sufficient for their success. It is essential to reach potential users and make them familiar with the technology product developed, the innovation involved and the product's usage to promote adoption. Moreover, support provision is another key factor to favour user uptake.

In this regard, in the context of software development, the creation of a community around it, provides multiple advantages such as significantly increased visibility, popularity and adoption, contribution to further development and technical support for and by their members in forums, open code repositories or other online spaces. The construction of a community around a technical project can enable its long-term success, impact and further continuation after the project ends by the hand of its members or key stakeholders attracted via this community.

The EVOLVED-5G project pays attention to these facts and has a strong commitment towards community building. In this regard, the Community block of the EVOLVED-5G architecture is fully devoted to fostering the creation of a community of developers, academia and other stakeholders out of the boundaries of the project.

This particular block is designed to facilitate collaboration, knowledge sharing, and communication among project members and external stakeholders, to ultimately foster a strong sense of community inside and outside the project and promote the continued

development and adoption of EVOLVED-5G technical innovations, in line with objectives in WP6.

Three main components harmonize the entire community block, each responding to different purposes but all at once supporting the creation of a community around the technical outcomes of the project. Namely, it encompasses:

- EVOLVED-5G Wiki [1]: official centralized technical documentation site of the EVOLVED-5G project.
- EVOLVED-5G Forum [2]: public online discussion platform where community users can post inquiries, ideas, and opinions on various topics regarding the EVOLVED-5G project, interacting with other members in the topic threads. Support for technical inquiries regarding EVOLVED-5G technical topics (e.g., Network App development) is provided by the EVOLVED-5G community, that includes project's developers.
- EVOLVED-5G Library [3]: set of items related to EVOLVED-5G publicly accessible, such as short articles, former presentations, links to some news or the EVOLVED-5G online training courses.

The Wiki, Forum and Library refer to EVOLVED-5G technical resources and provide spaces for describing their deployment, usage, technical aspects, give technical support to developers and end-users, and discuss and share knowledge, best practices and expertise on those technical artifacts. They complement the action in the direction of the EVOLVED-5G GitHub organization and the Marketplace. Together, these elements facilitate the creation of a vibrant and collaborative ecosystem that promotes innovation and accelerates the development and adoption of EVOLVED-5G technical outcomes.

Section 3.1 below describes the technical aspects of the EVOLVED-5G Wiki, one of the three entities that compose the Community architectural block. The other two components, the EVOLVED-5G Forum and the EVOLVED-5G Library, have been initially described in deliverable D6.1 [62] and all the implementation details will be given in the last deliverable of WP3 (D3.4) and the next WP6 deliverables. for the Forum and the Library respectively.

3.1 EVOLVED-5G WIKI

The EVOLVED-5G Wiki is the official centralized source of technical documentation of the EVOLVED-5G project, containing how-to, materials for project developers and generic public who may be interested in EVOLVED-5G as project. Indeed, it plays a relevant role in the community building strategy of the project as it not only acts as a centralized platform for sharing technical knowledge with developers but also for providing key but concise information regarding the project vision and objectives.

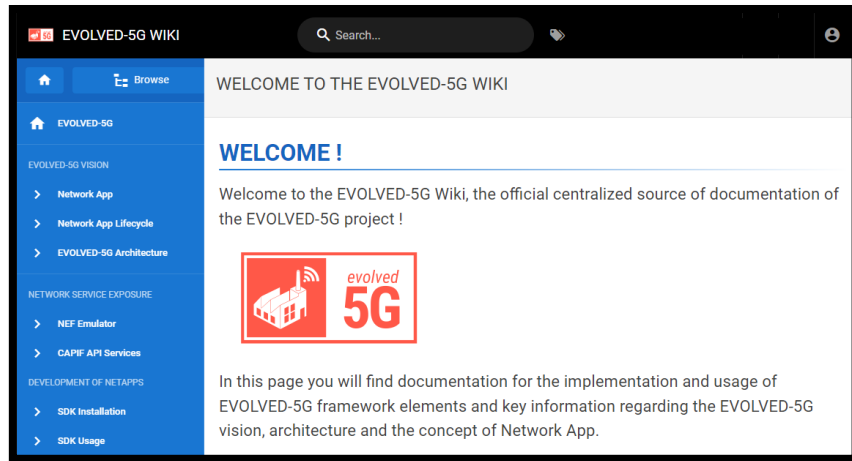


Figure 2 EVOLVED-5G Wiki

The key technical documentation is organized around different areas:

- EVOLVED-5G Vision (Network App Vision, Network App lifecycle & EVOLVED-5G Architecture).
- Network Service Exposure (NEF and CAPIF).
- Network App Development.
- Network App Verification.
- Network App Validation.
- Network App Certification.
- Network App Publication in the EVOLVED-5G Marketplace.

Those areas are tightly related to the NetApp lifecycle phases and the different EVOLVED-5G framework environments.

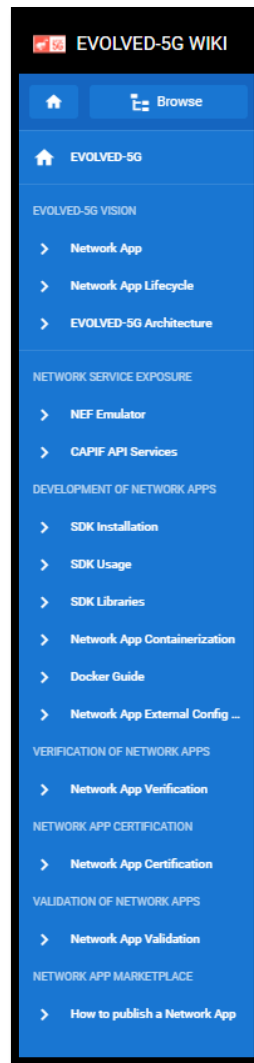


Figure 3 Topics and structure of contents of the EVOLVED-5G Wiki

3.1.1.1 Component Description

The EVOLVED-5G Wiki is an open-source component of the EVOLVED-5G framework that extends wiki.js engine. As a web application, the EVOLVED-5G Wiki is composed by two main functional blocks: frontend and backend. These blocks combine to create a powerful and versatile wiki platform that is highly usable, scalable, and performant. The front-end component is responsible for managing the graphical user interface displayed on the client side –on the user’s browser-, while the backend manages the server-side operations. In following sections, a more detailed explanation of these blocks is provided.

3.1.1.1.1 Wiki Frontend

The different frontend components are constructed using Vue.js [4] a JavaScript framework that provides a reactive and dynamic user interface that facilitates interaction, access to information from the wiki, as well as content creation and update.

The wiki frontend provides a graphical interface, with features for high usability to provide a good user experience, and a responsive design to ensure users can properly visualize the web platform and access contents from any internet-enabled device. Layout

and website elements are automatically and dynamically adjusted, providing mobile friendliness.

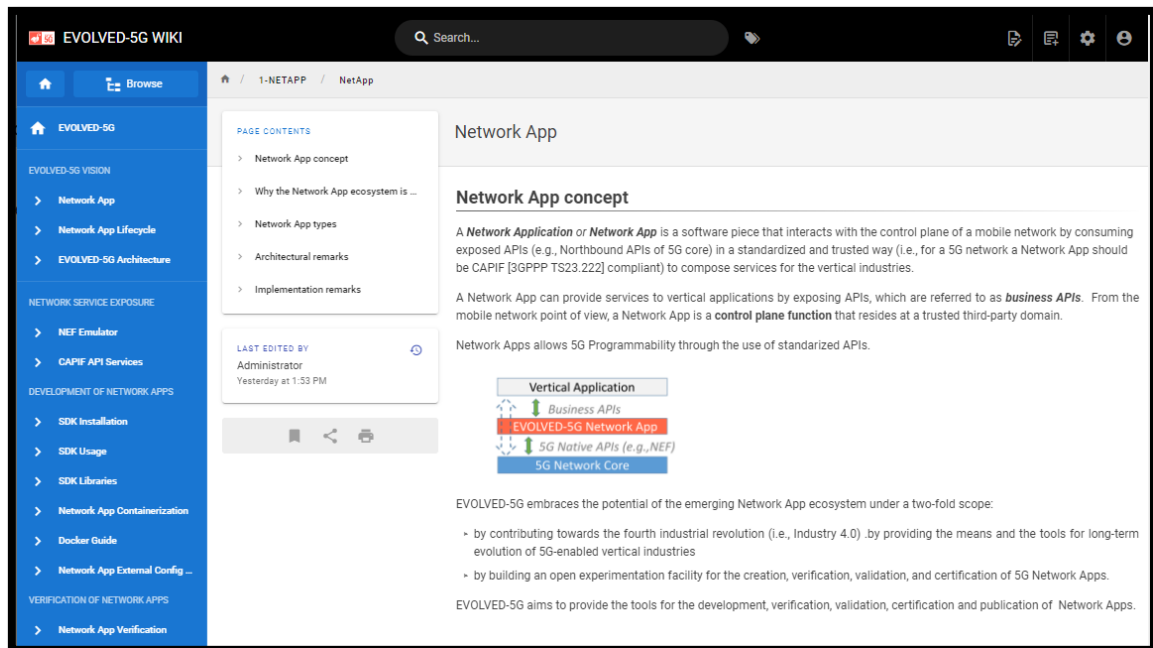


Figure 4 EVOLVED-5G Wiki public-facing front-end view

Main elements of the wiki public-facing interface are:

- **Content navigation menu:** displaying the tree structure of contents of the wiki, allowing for easy navigation. This menu is always visible and accessible on the left side of the screen, providing users a clear overview of the wiki's organization and structure.
- **Search functionality:** the web interface includes a search bar that allows users to search for specific content across all pages in the wiki.
- **Main panel:** displays contents regarding one specific topic selected from the navigation menu.
- **Shortcut content menu:** a small panel that shows the tree structure of the topic contents displayed on the main panel, providing a shortcut for each section and subsection. This feature provides higher usability, and improved content findability and access.
- **Last edition date module:** to provide information regarding the timeliness of the wiki content and its last updates.
- Functionalities of page printing, sharing on user social media and bookmarking.

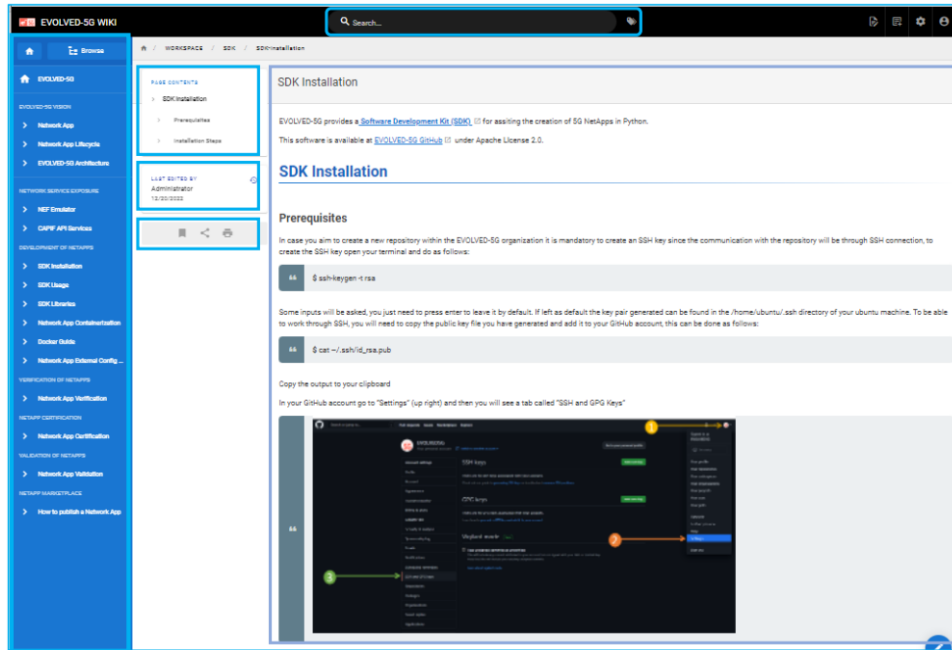


Figure 5 Main public-facing front-end elements of the EVOLVED-5G Wiki

Moreover, in addition to the user interface, presenting wiki topics and contents online, a dashboard for the administration of the platform provides access to specific functionalities, regarding site configuration, monitoring, analytics and security.

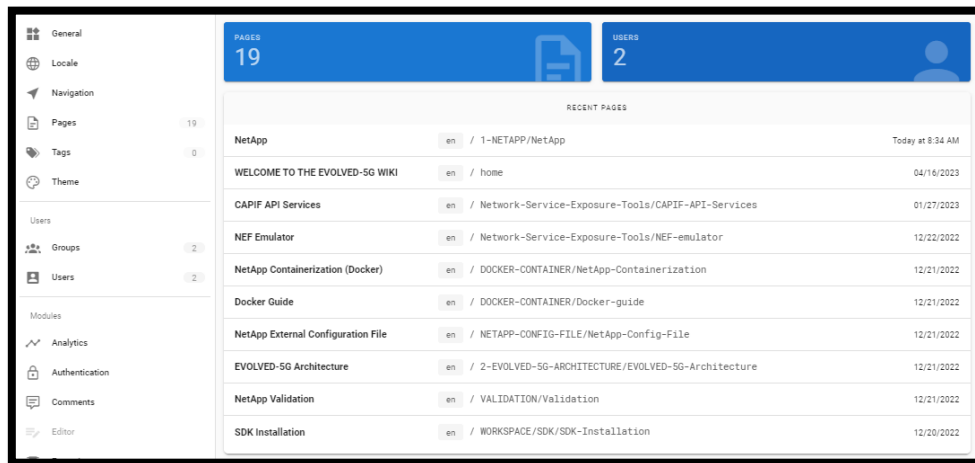


Figure 6 EVOLVED-5G Wiki administration dashboard

3.1.1.2 Wiki Backend

The EVOLVED-5G Wiki backend is responsible for the server-side operations and it is built on top of Node.js [5], an open-source, cross-platform JavaScript runtime employed for building scalable and performant Network Apps.

This backend encompasses a PostgreSQL [6] database to store, manage and retrieve wiki content, user data, and configuration settings, and it is capable to support other different relational database systems (MySQL [7], MariaDB [8], and SQLite [9]).

The EVOLVED-5G Wiki exposes a REST API that allows to interact with the platform programmatically employing GraphQL [10] queries. This API can be used to create, update, and delete pages, retrieve key information, as well as to perform other operations.

Some important functionalities provided by the backend are:

- Content Management (creation, edition, categorization and metadata addition). The EVOLVED-5G wiki integrates a Markdown editor.
- Git-based Version Control: the backend stores a history of all page revisions, allowing a very advanced management of contents. It is based on Git [11], the most important distributed version control system. Git provides version control functionality, allowing to track changes applied to pages and roll back to previous versions.
- User Management: for the management of users, roles, and permissions. In this regard, the EVOLVED-5G wiki supports multiple authentication providers.
- Search engine: enabling powerful content search across the entire wiki or specific sections.
- Integration with third-party tools such as Google Analytics [12], Slack [13], and Trello [14].
- Wiki configuration and customization: In addition to allow for the management of advanced configuration features, it provides a wide variety of customization options, including themes, branding, and plugins.
- Security: the backend has built-in security features such as regular or two-factor authentication, authorization, password policies, and IP restrictions, to ensure the safety and confidentiality of the wiki data.

3.1.2 Wiki Implementation

The EVOLVED-5G wiki engine has been implemented upon Node.js, and the whole component has been virtualized employing Docker technology, allowing for immediate deployment on any OS regardless of the underlying environment and technologies. Another remarkable implementation feature is the openness of code, as an open-source component, following the philosophy of making software available to the community for use, modification, and distribution.

An instance has been deployed on a dedicated server, running on the domain of **wiki.evolved-5g.eu**. In addition to the wiki framework deployment, security certificates and additional analytic functionality have been incorporated into the server to enhance the overall system and provide improved global security and raise awareness.

This wiki instance has been properly configured and populated with technical contents related to the EVOLVED-5G vision and technical development. Periodic content updates are performed according to feedback from developers and project technical progress and evolution.

3.1.3 Wiki Security

The EVOLVED-5G Wiki implements various security mechanisms to ensure confidentiality, integrity, and availability of the data stored in the platform and shared to the client side accessing the wiki, as well as protection against unauthorized access, theft, misuse of data and malicious attacks to the platform and recovery features.

These mechanisms include encrypted connections (such as HTTPS), strong authentication and authorization procedures, regular vulnerability assessments and security audits, intrusion detection and prevention systems, data backup and recovery plans. Additionally, the server hosting the Wiki makes use of firewalls and other advanced security technologies to prevent and mitigate potential security threats. In more detail, the security mechanisms implemented are:

- **Authentication and authorization:** EVOLVED-5G Wiki has a robust authentication and authorization system that allows for platform content management. Several administrative management actions can only be performed directly on the server, which minimizes the risk of external malicious attacks.
- **Support for 2FA authentication** for enabling user access to private areas.
- **Third-party authentication integration:** the EVOLVED-5G Wiki engine integrates third-party authentication systems such as Google, Facebook, and GitHub. This feature facilitates user management and reduces the risk of use of weak or compromised passwords.
- **Data encryption:** EVOLVED-5G Wiki employs SSL/TLS encryption to protect the transmission of data between the server and the clients accessing the web application, ensuring that information is kept secure in transit.
- **Secure connection via HTTPS & security certificates:** EVOLVED-5G Wiki implements the HTTPS protocol that provides an additional layer of security to protect the confidentiality and integrity of the data stored on the platform, enabling secure connection between client and server. HTTPS encrypts all data transmitted between clients and the server, ensuring that the data is protected against interception and eavesdropping by attackers.
- **Git-based version control:** EVOLVED-5G Wiki stores historical version information for each content displayed online and website configuration, allowing to revert to a previous version in case an unwanted modification occurs.
- **Semi-automated backup:** the EVOLVED-5G wiki allows for regular backups of the information stored on the platform, ensuring data protection, portability and data recovery.
- **Audit logging:** allowing for activity monitoring and the identification of suspicious activities.
- **Protection against malicious code injection** through attacks such as cross-site scripting (XSS) or SQL injection.
- **Web app uptime monitoring system:** notifying in case the server is found to be down or unresponsive.
- **Firewall protection** at two levels: i) at LAN level and ii) at server level, blocking all external access attempts to unauthorised ports.
- **Security configuration:** the wiki platform, server and LAN access are adequately configured for enhancing security and minimizing threats.

The previous security features protect server-client communication from security risks of data interception, man-in-the-middle attacks (MITM), identity spoofing, data alteration, data theft, and malicious content injection attacks such as XSS or SQL injection and facilitates data regulatory compliance. Moreover, these measures allow for a quick recovery of the wiki platform, stored contents and configuration.

4 WORKSPACE ENVIRONMENT

4.1 COMPONENT AND WORKFLOW DESCRIPTION

The Workspace environment is responsible for the first phases in the lifecycle of a Network App, encompassing the development phase and the verification phase. Different functional blocks are involved, mainly represented in Figure 7.

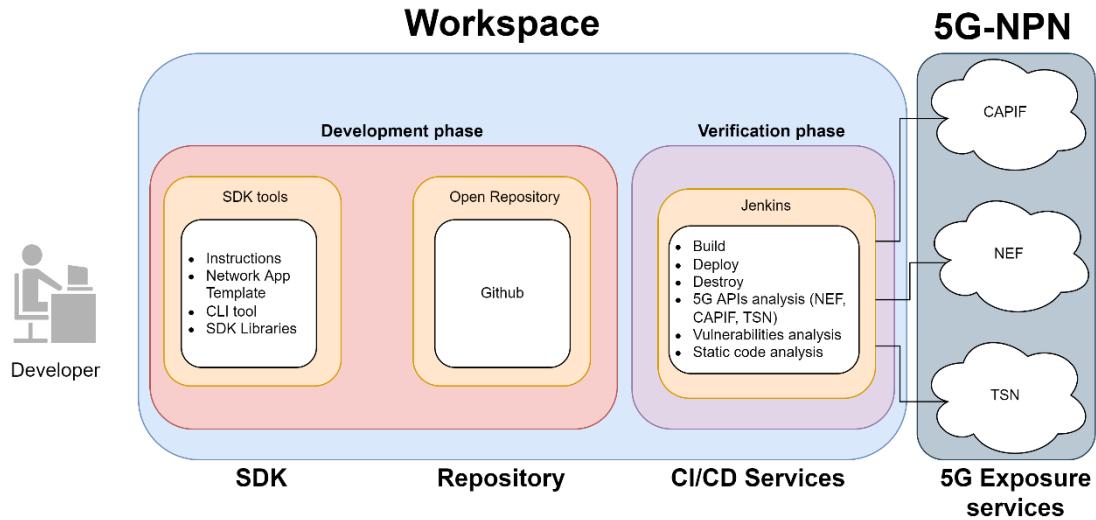


Figure 7 Workspace functional blocks

All the functional blocks are explained in detail in Deliverable D3.1 [15] although, new additions and improvements have been implemented such as (i) new SDK libraries, (ii) new and improved verification tests for instance, a new verification test has been provided for developers to verify the proper Network App functionality regarding the new TSN functionality. Hereafter, in Figure 8 it is explained in detail the workflow from the point of view of a developer from creation of a Network App until it is verified in the EVOLVED-5G framework.

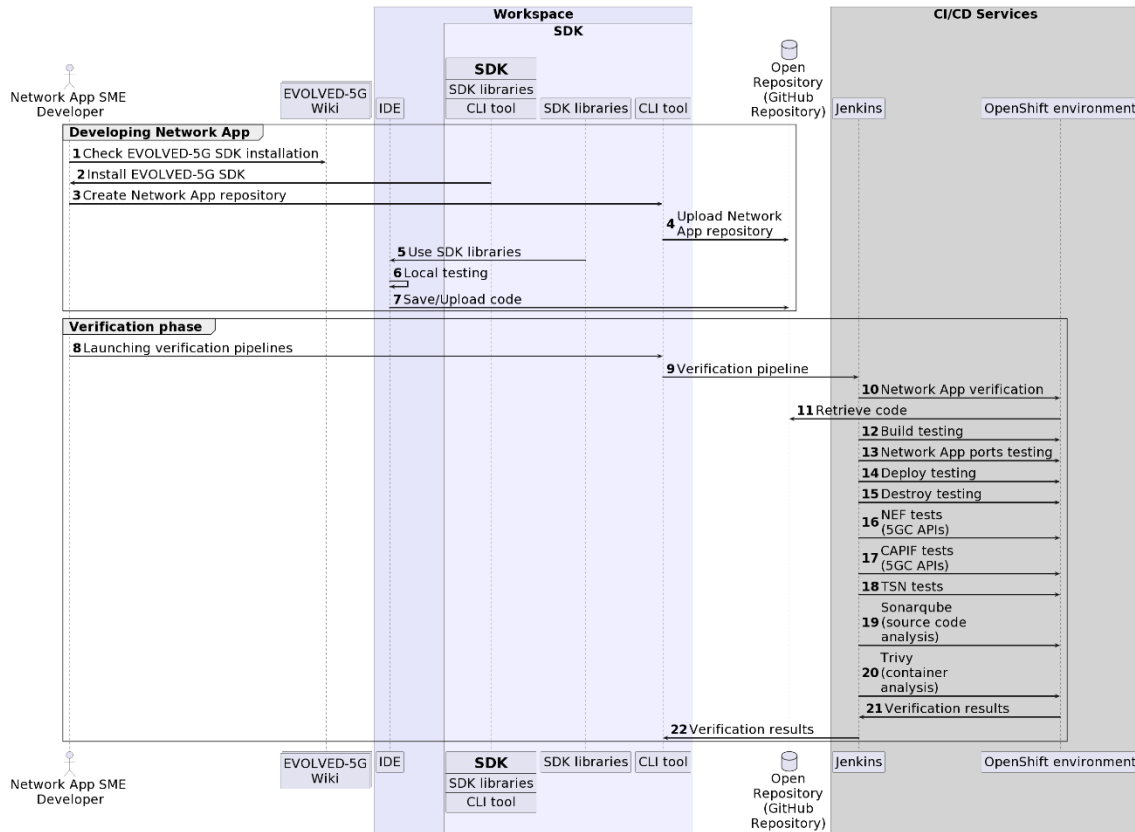


Figure 8 Workspace workflow

As it can be seen from Figure 8 steps of the workspace workflow are as follows. The development phase is orchestrated around 5 main stages while the verification phase comprises steps 8 to 21 (both included).

Development phase:

1. Checking the SDK installation: The first step (step 1) during the Development phase refers to checking the SDK installation in the EVOLVED-5G Wiki (as presented in Section 3) to actually understand how the SDK should be installed in their local environment. As explained in Deliverable D3.1 the SDK is composed by CLI tool and the SDK libraries, to sum it up, with the CLI tool the developer is able to create the Network App repository in GitHub as well as launching all the commands for verifying the Network App as it will be explained in Section 4.1.1.2.1. Also, the validation pipeline can be launched through the CLI tool (further details a will be provided in Deliverable D5.5). On the other hand, the SDK libraries allow developers' Network App to interact with 5G APIs providing abstraction towards the API.
2. SDK local installation: Once the developer understands how to install the SDK tool, then next step (step 2) is to locally install the SDK tool in order to use i) the CLI tool for creating the Network App and ii) to use the SDK libraries in its Network App.
3. Creation of the Network App repository: The next step (step 3) is to create the Network App repository. This step has changed in order to ease the developers' task and is explained in detail in Section 4.2.1.

4. Upload of the Network App to the repository: the Network App will be uploaded to the EVOLVED-5G GitHub repository [16] (step 4).
5. Final tests: the next stage in the workflow comprises steps 5, 6 and 7 which relate to use the SDK libraries, do the necessary tests to check that the Network App works properly locally and once everything is checked up then upload the new code to the GitHub repository.

With this last step, the development phase is ended, and the Network App developer can start with the verification phase to check that the Network App is fully functional within the EVOLVED-5G framework and its considerations.

Verification phase:

The Verification phase starts with step 8. Indeed, the first step of the verification phase is to select and execute one of the available verification tests from the CLI tool (more details in Section 4.1.2). Then a verification pipeline is started in OpenShift environment (step 9 and 10), OpenShift will retrieve the code (step 11) and will start executing one of various verification tests selected from the developer (step from 12 to 20). After the pipeline has finished it will return a result to the developer informing either a success or a failure (step 20 and 21).

4.1.1 Development tools

Each functional block within the Workspace is composed by different tools that are used in development and verification phase during the Network App lifecycle. Following subsections provide information about each one of them.

4.1.1.1 Network App Template

The aim of the Network App Template [58] tool is to assist developers, whether they are part of the consortium or not, by providing an example of a Network App in EVOLVED-5G while also outlines the required folder structure. The Network App Template complements the creation of the Network App at the development phase, as it provides an example of how a Network App should be designed, providing folder and file structure. As a result of the Network App Template, the developer is properly guided and able to adequately follow the process for creating Network Apps described in the Wiki, to a real implementation.

4.1.1.2 SDK

As mentioned in Section 4.1.1, the SDK is composed by two main tools described in the following subsections.

4.1.1.2.1 CLI tool

It allows creating a repository in GitHub as well as locally, performing the Network App' configuration, i.e., file and folder structure, by means of the Network App Template. Also, it provides different commands to start and launch the different verification tests available. Finally, it provides a command to check the status of each verification test launched.

4.1.1.2.2 SDK libraries

The SDK libraries are a set of Python classes that provide abstraction towards the 5G APIs and enhance Network Apps with 5G capabilities, which in following phases such as, verification, validation and certification will be tested.

4.1.1.3 Open repositories

EVOLVED-5G is composed by two open repositories as explained in Deliverable D3.1 [15] GitHub, as a code repository for Network Apps while, the artifact repository is storing the Network App images generated during validation and certification phases.

4.1.1.4 Dummy Network Application

The Dummy Network Application is a non-functional piece of software designed to reproduce behavioural functionalities an actual Network App must have in the EVOLVED-5G project. It has connectivity with all the exposure services (CAPIF, NEF and TSN) as well as it is guaranteed to successfully pass all the verification tests available.

4.1.2 Verification tools – CI/CD Pipelines

The verification of the Network Apps that are developed leveraging the EVOLVED-5G framework is implemented through a collection of tools that are appropriately configured by the EVOLVED-5G framework. The aim is to verify the proper development of the Network App and its effective communication with the 5G network. These tools are seamlessly incorporated in the EVOLVED-5G verification pipeline, which can be invoked through the SDK by means of the CLI tool within the EVOLVED-5G workspace in an intuitive manner. Each Network App developer, regardless of the particularities and specific functionalities of the under-development Network App, may invoke the verification pipeline as many times as needed to ensure that the Network App satisfies all requirements for the next phase of the Network App lifecycle, i.e., the validation phase.

The functionalities of the Network App that are verified during the verification phase, can be grouped in three main categories: (i) syntax analysis of the code of the Network App, (ii) quality of the Network App docker container and (iii) proper communication capabilities of the Network App with the 5G network. For the third category of tests, the integration capabilities of the Network App with the NEF, CAPIF and TSN tools developed for the purposes of the project are necessarily involved in the process. This alleviates the developer from the requirement of an existing 5G network to execute the tests. The details about the tools employed in the verification pipeline are described in the following sections.

4.1.2.1 SonarQube

SonarQube tool has been described in Deliverable 5.3 [17] section 2.1.2. This tool has been integrated into EVOLVED-5G SDK to enable developers using this tool during development phase to get results before submitting their Network Apps for Validation. In order to use the tool from the SDK, a pipeline has been created in the CI/CD environment. To launch this pipeline, the following command has to be used:

```
evolved5g run-verification-tests --mode code_analysis --repo  
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

Results of this command can be gathered using the command:

```
evolved5g      check-job      --id      YOUR_ID      --user  
DEVELOPER_PIPELINE_USERNAME      --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

4.1.2.2 Nmap (Network App open ports)

The Nmap is a free and open-source tool for network discovery and security auditing. In EVOLVED-5G it has been utilised to check whether the ports declared in the dockerfile of the Network App are being used while the Network App is deployed. It has been integrated during building the verification tests.

4.1.2.3 Trivy

Trivy tool has been described in Deliverable 5.3 [17] section 2.1.1. This tool has been integrated into EVOLVED-5G SDK to enable developers using this tool during development process to get results before submitting their Network Apps to Validation. In order to use the tool from the SDK, a pipeline has been created in the CI/CD environment. To launch this pipeline, the following command has to be used:

```
evolved5g run-verification-tests --mode security_scan --repo  
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

The commands to gather the results are described in Section 4.1.2.1.

4.1.2.4 Build, deploy, destroy

While developing their Network Apps, Developers might want to test whether their code can be deployed as a container, considering that a container platform (Kubernetes) will be used during Validation and Certification phases.

The first step Developers need to take is to build their Network Apps. This process generates an image of the Network App contained in the Developers GitHub repository. The output of this process is a containerized image that is suitable to be deployed in a Container platform. To launch this process, the following command has to be used:

```
evolved5g run-verification-tests --mode build --repo  
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

The build pipeline also offers the version network app (--version NETWORK_APP_VERSION) as an optional parameter to be provided to the previous command.

Once the image is built, developers might want to test deploying their Network Apps in the Container Platform. For this purpose, the OpenShift container platform in the CICD environment will be used. To launch this process, the following command has to be used:

```
evolved5g run-verification-tests --mode deploy --repo  
REPOSITORY_NAME
```

Once the image is deployed, Developers will need to clean the environment before repeating the Deployment test. For this purpose, the Network App needs to be destroyed

from the Container platform, so the resources can be liberated. To launch this process, the following command has to be used:

```
evolved5g run-verification-tests --mode destroy --repo  
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

Alike the build pipeline, the destroy pipeline, also offers the version network app (--version NETWORK_APP_VERSION) as an optional parameter to be provided to the previous command.

The commands to gather the results are described in Section 4.1.2.1.

4.1.2.5 CAPIF

CAPIF Core Function tool is an implementation of CAPIF APIs for the CAPIF Core Function entity as defined in 3GPP TS 29.222 [18]. This tool implements a REST API server that accepts API requests from CAPIF entities defined as API Invokers, API Exposing Function, API Publishing Function and API Management function. It has been developed and implemented in the context of EVOLVED-5G for the Verification, Validation and Certification process.

To launch this process, the following command has to be used:

```
evolved5g run-verification-tests --mode capif_nef --repo  
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd  
DEVELOPER_PIPELINE_PASSWORD
```

The commands to gather the results are described in Section 4.1.2.1.

4.1.2.6 NEF

NEF is a software component that emulates the Northbound APIs of 3GPP's Network Exposure Function (NEF) as defined in 3GPP TS 29.522 [19]. It follows the RESTful paradigm and provides Network Apps with some of the internal services and capabilities that the 5G System (5GS) offers. It has been developed and implemented in the context of EVOLVED-5G for the Verification, Validation and Certification process.

To launch this process and gather the results, the commands described in Section 4.1.2.5 can be used.

4.1.2.7 TSN

The IEEE 802.1 Working Group has developed Time-Sensitive Networking (TSN) [20], which is a set of standards with the intention to provide deterministic connectivity (e.g., bounded latency) over Ethernet networks. The time synchronization and traffic shaping are decisive features (among others) to meet the strict requirements of the applications such as Industrial Automation and Automotive Communications. Figure 9 presents the fully centralized TSN network architecture, where the TSN endpoints are connected using TSN Bridges. Furthermore, the TSN endpoints communicates the QoS requirements in advance to the Centralized User Configuration (CUC). Then, CUC transfers this information to the Centralized Network Configuration (CNC) which is in charge of the TSN Bridges configuration in support of the TSN endpoints.

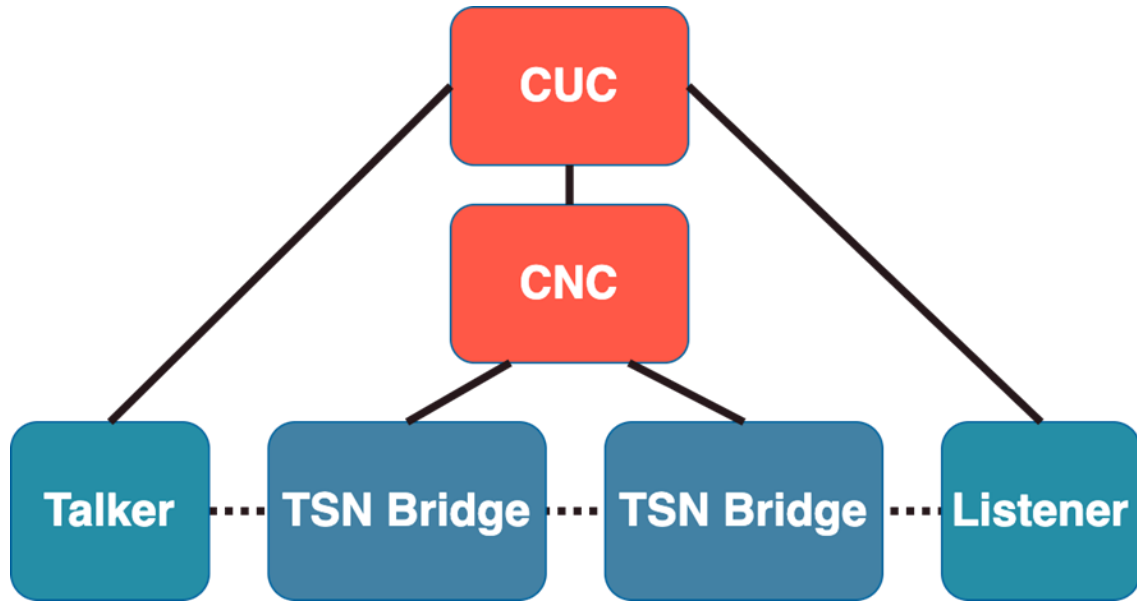


Figure 9 Fully centralized TSN network architecture - IEEE 802.1Qcc

5G supports the possibility to expand the TSN wired networks toward wireless networks. In addition, the combination of TSN over 5G networks has benefits like mobility and replacing cables in the industry. The recent 3GPP standards (Rel.16) [21] (Rel.17) [22] introduce the entities and procedures for this integration. For instance, the time synchronization management, and the role of the 5G network components, especially the TSN Application Function (AF), which is a key component to ensure the quality through the end-to-end connectivity that enables the 5G network to act as a transparent TSN Bridge[23].

To launch this process, the following command has to be used:

```
evolved5g run-verification-tests --mode capif_tsn --repo
REPOSITORY_NAME --user DEVELOPER_PIPELINE_USERNAME --passwd
DEVELOPER_PIPELINE_PASSWORD
```

The commands to gather the results are described in Section 4.1.2.1.

4.1.2.8 Robot Framework

Robot Framework [63] is an open-source automation tool with numerous capabilities of being integrated with several external tools and, in practice, any programming language. In a human readable manner, its syntax supports the definition of any testing plan of a software component implemented in any language. In the workspace environment of EVOLVED-5G, Robot Framework is employed for testing the communication of the Network App with the 5G capabilities, i.e., NEF, CAPIF and TSN explained in the previous sections.

4.2 WORKSPACE IMPLEMENTATION

This section describes how different tools implemented under the Workspace environment have been carried out.

4.2.1 Development Phase

This sub-section is devoted to describing the implementation that has been followed in the Workspace environment to support the creation of a Network App from scratch.

4.2.1.1 Network App template

As mentioned in Section 4.1.1.1, the Network App Template is providing a file and folder structure when the Network App is created. In Figure 10 it can be seen the different files and folder provided to the developer.

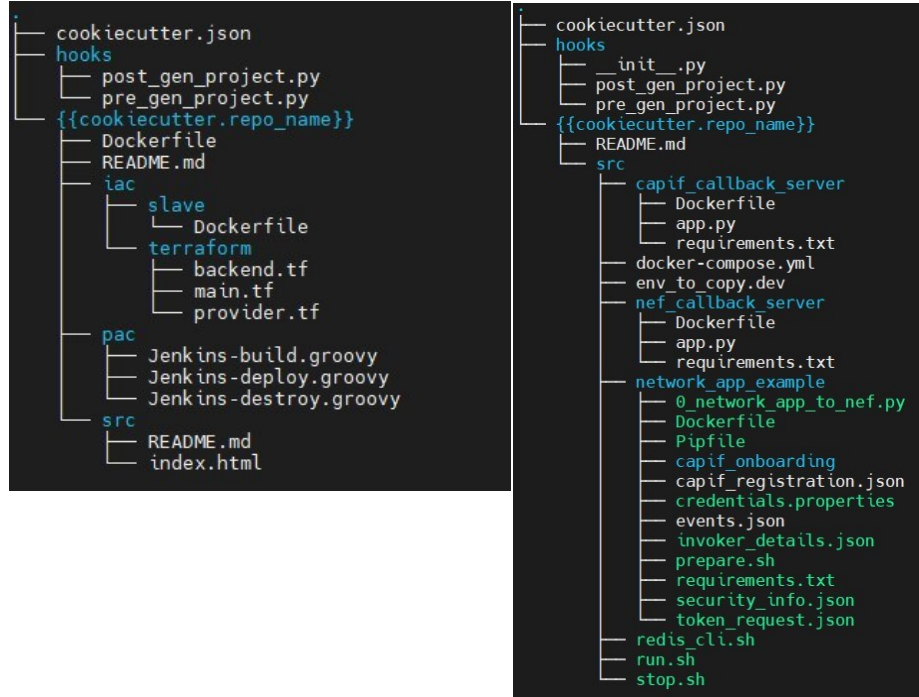


Figure 10 Old Network App structure vs New Network App structure

Figure 10 has been already described in Deliverable D3.1, however, there are some upgrades that are worth mentioning. Folders “*pac*” and “*iac*” have been removed. The reason behind removing “*pac*” and “*iac*” folders is that Terraform [24] is not used anymore and, instead, Helm [26] has been adopted. Now the “*src*” folder contains different files and folder to provide a functional example to developers, the main files and folders includes are the following:

- Example Network App container, which contains the example of a Network App, including:
 - Dockerfile: It stores all the commands to call on the command line to build the Network App.
 - Capif registration json file: It stores all the information related to CAPIF such as, folder to store certificates, http and https ports among others.
 - Capif onboarding folder, to store CAPIF certificates.
 - A python file, simple example of the Network App using the SDK libraries to communicate with CAPIF.
 - A bash file, to register the Network App into CAPIF.
- Dockercompose file: It contains all the commands to build and deploy all the necessary containers to start a Network App.
- README file: It explains the purpose of the repository.

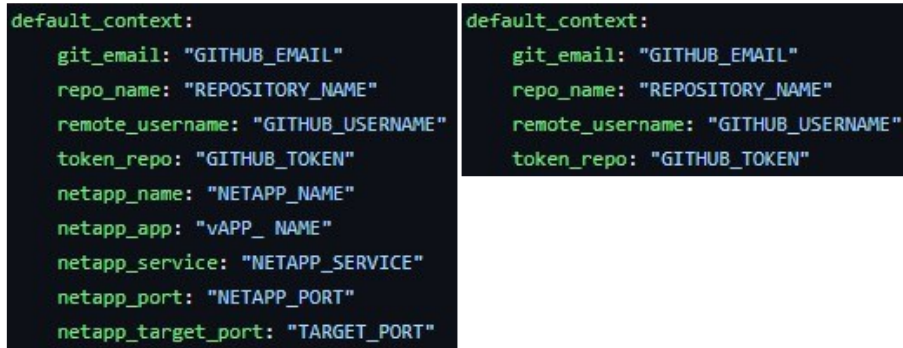
4.2.1.2 SDK

This subsection explains updates and improvements implemented in the CLI tool and the SDK libraries.

4.2.1.2.1 CLI tool

The SDK CLI tool purpose is twofold, i) to create the Network App both, locally and remotely (GitHub) and ii) to launch all the verification tests through the different pipelines implemented in EVOLVED-5G, as it has been described in Section 4.1.2.

Regarding the first function, the updated approach is for the developer to compile a configuration file rather than providing all the inputs manually via terminal. Also, the inputs have been reduced significantly as it can be seen from Figure 11. Now only four basic inputs are required from the developer, their email, the repository name, the GitHub username and the GitHub token.



Old Inputs	New Inputs
default_context:	default_context:
git_email: "GITHUB_EMAIL"	git_email: "GITHUB_EMAIL"
repo_name: "REPOSITORY_NAME"	repo_name: "REPOSITORY_NAME"
remote_username: "GITHUB_USERNAME"	remote_username: "GITHUB_USERNAME"
token_repo: "GITHUB_TOKEN"	token_repo: "GITHUB_TOKEN"
netapp_name: "NETAPP_NAME"	
netapp_app: "vAPP_NAME"	
netapp_service: "NETAPP_SERVICE"	
netapp_port: "NETAPP_PORT"	
netapp_target_port: "TARGET_PORT"	

Figure 11 Old Inputs vs New inputs to create a Network App

The new command to create the Network App repository is:

```
evolved5g generate --config-file <path to the user configuration file>
```

4.2.1.2.2 SDK libraries

The SDK libraries are a set of python classes that a) allows Network app developers to register and authenticate their Network Apps to the CAPIF tool, b) speeds up development for specific scenarios (ex. making calls to the 5G-API to track user devices) c) allow “Providers” (that is API creators that want to make their APIs available to the Network Apps) to register their resources to CAPIF (ex. the 5G-APIs exposed by the NEF Emulator, or the TSN FrontEnd).

These libraries can be found in the following repository [27]. In order to use them, the developers have to:

- Import the evolved5G.sdk python module in their code.
- Make sure they have access to a running instance of the CAPIF tool, since the SDK interacts with the CAPIF APIs

The SDK Libraries are composed by the following classes:

CAPIFInvokerConnector: This Python class facilitates a Network App to register and onboard itself to the CAPIF tool. The developer does not need to know the details of the CAPIF APIs and, which calls to make to which endpoint, since this is covered by the class itself.

In the examples folder of the SDK repository a Python script [28] is provided to demonstrate its usage. The developer only needs to provide parameters (in a configuration file) that specify:

- A folder to store the SSL certificates required for Network App - CAPIF communication
- The URL of the CAPIF service (port, host and protocol)
- The Network App credentials that will be used during registration
- A set of fields related with the creation of the SSL certificates.

The registration and onboarding need to happen once. Once this registration is complete, the Network App is recognized by the CAPIF tool and has access to discover and use the available 5G-APIs.

ServiceDiscoverer: This python class allows the Network App developer to

- a) Discover services (API endpoints) exposed by CAPIF
- b) Retrieve access tokens from CAPIF in order to use these APIs

Once again, the developer does not need to know the details of the CAPIF APIs, which calls to make to which endpoint, since this is covered by the class itself. In the examples folder of the SDK repository a python script [29] is provided to demonstrate its usage. ServiceDiscoverer is used internally by the SDK classes: LocationSubscriber, ConnectionMonitoring, QosAwareness and TSNManager that are explained below.

LocationSubscriber: This Python class can be used to monitor device locations, allowing the Network App developer to receive a notification every time the device being monitored, via the Network App, connects to a different cell. A high-level demonstration can be seen in Figure 12 and Figure 13 below. In the examples folder of the SDK repository a python script [30] is provided to demonstrate its usage. Using the LocationSubscriber library in the Network App, the developer must follow the subsequent steps:

- As a first step, it is necessary to start the NEF emulator to simulate an environment where a user device is moving between different network cells. NEF emulator already provides a scenario with three different user devices and four 5G cells, simulating the movement of the user devices.
- The second step is to initialize a web server, which can either reside in the Network App or in an external web server, to receive Event notifications from the NEF Emulator (or the 5G API) every time the device that is monitored changes its location (moves to another cell).

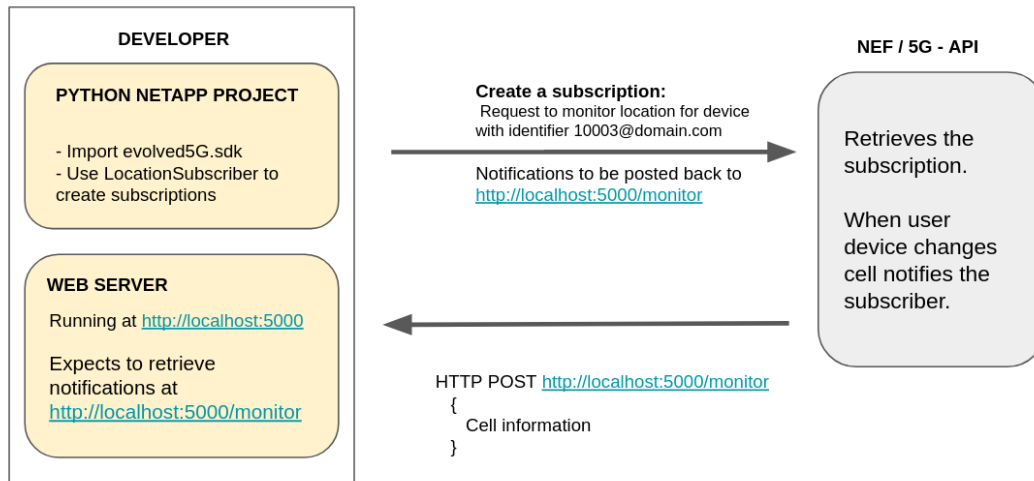


Figure 12 Location Subscriber - Subscribe to monitor location changes

There is also the option to request the Location of a given device, at any given time.

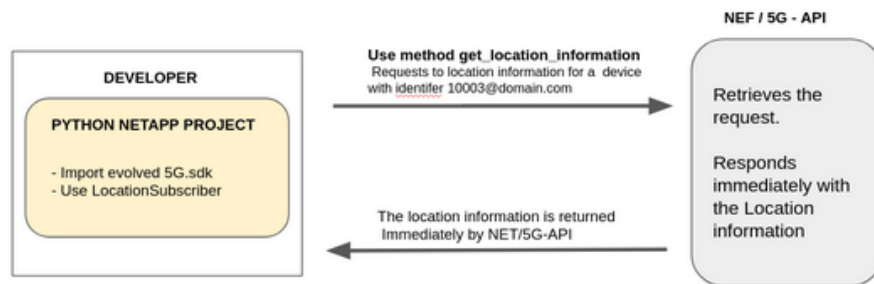


Figure 13 Location Subscriber - Request location for a given device

LocationSubscriber interacts with the CAPIF tool in order to retrieve 5G-APIs related to location monitoring, retrieve access tokens for this APIs and make the relevant calls. All these details are transparent for the developers, speeding up their development.

ConnectionMonitor: This Python class can be used to monitor the network connectivity of devices. It allows a Network App developer to retrieve a notification every time the monitored device connects (or disconnects to the network). Consider a scenario where a Network App wants to monitor 100 devices in the 5G Network. The Network App wants to track, at any given time how many devices are connected to the 5G Network and how many devices are disconnected. Using ConnectionMonitor the Network Apps can retrieve notifications by the 5G Network for individual devices in the following situations:

- Connection is lost (for example the user device has not been connected to the 5G network for the past 10 seconds)
- Connection is alive (for example the user device has been connected to the 5G network for the past 10 seconds)

A high-level demonstration can be seen in Figure 14 the Figure below. In the examples folder of the SDK repository a python script [31] is provided to demonstrate its usage.

To use the ConnectionMonitor library in the Network App, the developer must follow the subsequent steps:

- Start the NEF emulator to simulate an environment where a user device is moving between different network Cells. When a user device is out of range the NEF emulator sends notifications.
- In order to retrieve notification about the location of the device, the developer must initialize a web server to retrieve notifications from the NEF Emulator (or the 5G-API)

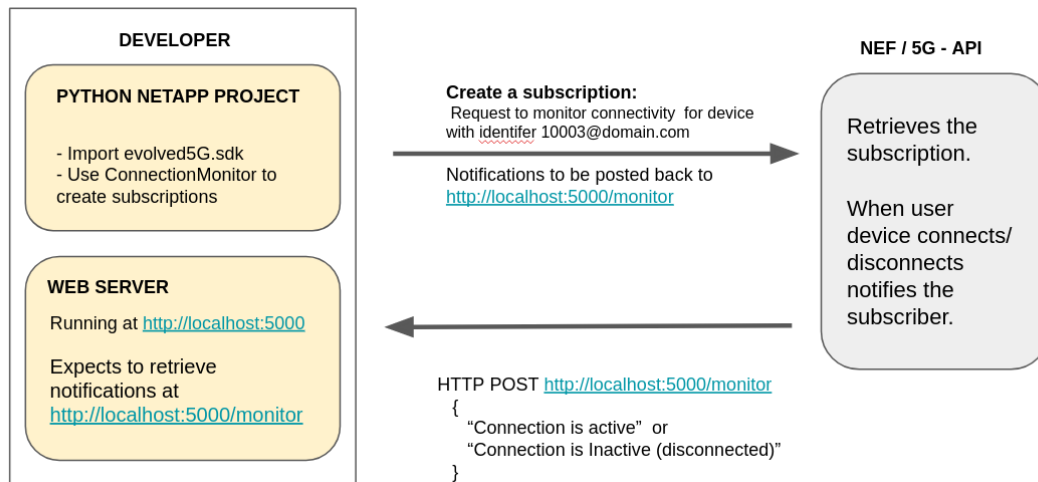


Figure 14 ConnectionMonitor- Subscribe monitor network connectivity

ConnectionMonitor interacts with the CAPIF tool in order to retrieve 5G-APIs related to network monitoring, retrieve access tokens for this APIs and make the relevant calls. All these details are transparent for the developers, speeding up their development.

QoSAwareness: This python class can be used to control network traffic connectivity and monitoring, allowing the Network App developer to select between Non-Guaranteed Bit Rate (Non-GBR) and Guaranteed Bit Rate (GBR) quality flow indicators (GFIs) and receive notifications in the Network App if the network conditions are not met. This can result in better service experience for the Network App consumers for scenarios like Live Streaming or Conversational Voice among others. A developer can use QoSAwareness to:

- Establish a Guaranteed Bit Rate (GBR) QoS Flow in the Network App. In this scenario the developer has to specify a) the QoS Flow Indicator (QFI) of interest: Conversational Voice or Conversational Video or Discrete automation b) the minimum allowed delay of data packages, during uplink or downlink or roundtrip (e.g., minimum 20ms for uplink) c) any data thresholds on volume (e.g., up to 5 GB for uplink). Once the QoS subscription is established, the Network App developer receives notifications when a) thresholds (minimum delay) cannot be achieved b) thresholds (minimum delay) are restored or c) usage thresholds (data volume) are exceeded.
- Establish a Non-Guaranteed Bit Rates (Non-GBR) QoS in the Network App. In this scenario the developer has to specify a) the QoS quality indicator of interest: Live Streaming or TCP Base, b) the data thresholds on volume (ex. up to 5 GB for uplink). Once the QoS subscription is established, the Network App developer receives notifications when usage thresholds (data volume) are exceeded. In both scenarios the Network App can use the received notifications to adjust the

application's behaviour to improve service experience. A high-level demonstration can be seen in Figure 15. In the examples folder of the SDK repository a python script [32] is provided to demonstrate its usage. To use the QoSawareness library in the Network App, the developer must follow the subsequent steps:

- Initialise a web server to retrieve notifications from the NEF Emulator (or the 5G-API).
- Start NEF emulator to simulate an environment where user devices are moving between / connecting to different network Cells. Then create a GBR subscription for these user devices via the Network App. To facilitate testing, the NEF emulator supports the following scenario: If two user devices are connected to the same cell the NEF emulator will send notifications, informing the Network App that the QoS targets cannot be achieved. If only one user device is connected to a cell, the NEF emulator sends notifications that the QoS targets are achieved.

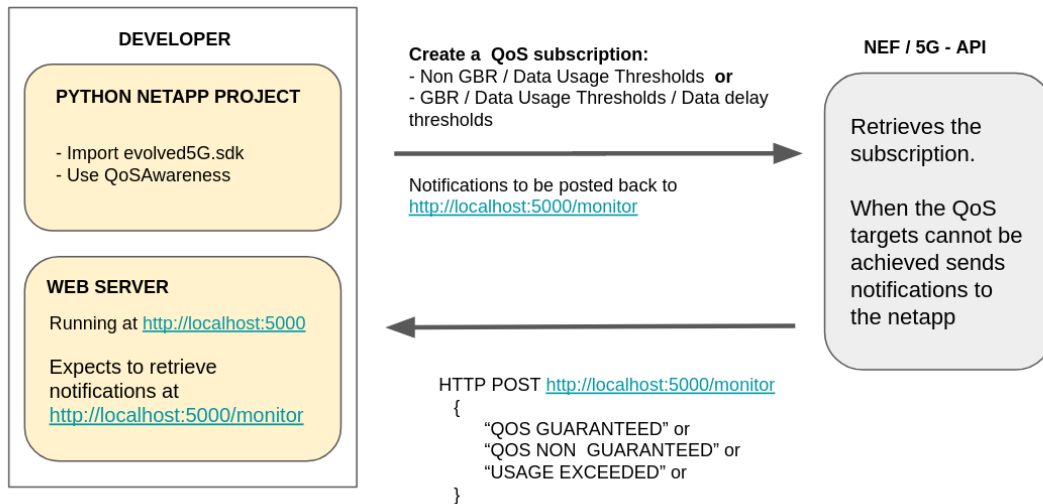


Figure 15 QoS Awareness- Subscribe to monitor changes to the QoS thresholds

QoSawareness interacts with the CAPIF tool in order to retrieve 5G-APIs related with QoS monitoring, retrieve access tokens for this APIs and make the relevant calls. All these details are transparent for the developers, speeding up their development.

TSNManager: This Python class allows the Network App developer to interact with the TSN FrontEnd in order to apply Time-Sensitive Networking (TSN) standards to time-sensitive Network Apps. It allows the configuration of certain parameters in the underlying TSN infrastructure of the testbed.

These parameters indicate the expected QoS of the communication. More information can be found at [33].

In the examples folder of the SDK repository a Python script [34] is provided to demonstrate its usage. To use the TSNManager the developer must follow the subsequent steps:

- Initialise a Python project and download evolved5G SDK as a python dependency. Use TSNManager class

- First start CAPIF and then TSN FrontEnd to simulate an environment where you can make calls to the TSN API.

CAPIFProviderConnector: This Python class allows a Provider (like the NEF emulator) to interact with CAPIF. The developer that implements a provider (ex. NEF) can use this class to register, onboard and publish endpoints to a CAPIF server. The developer does not need to know the details of the CAPIF APIs, which calls to make to which endpoint, since this is covered by the class itself. In the examples folder of the SDK repository two Python scripts are provided to demonstrate its usage. One example [28] demonstrating how the NEF emulator can expose its 5G-APIs to the CAPIF server and one example [35] demonstrating how the TSN tool can expose its APIs to the CAPIF server.

The developer using CAPIFProviderConnector only provides parameters that specify:

- A folder to store the SSL certificates required for Provider - CAPIF communication.
- The URL of the CAPIF service (port, host and protocol).
- The Provider's credentials that will be used during registration.
- A set of fields related with the creation of the SSL certificates.
- The description of the APIs endpoints that will be exposed via CAPIF.

The registration and onboarding need to happen once. Once this registration is complete, the CAPIF tool can expose the related/registered 5G-APIs to the Network Apps.

CAPIFLogger: This Python class allows a Provider (like the NEF emulator) to capture Log information to CAPIF. This allows to monitor usage from Network Apps (ex. which API and endpoint has been called and when, the input parameters of the request and the output parameters of the response) and performance (like the time it takes to fulfil an API request or status codes indicating success or failures).

In the examples folder of the SDK repository a Python script [59] is provided to demonstrate its usage.

CAPIFAuditor: This Python class allows a Provider (like the NEF Emulator) to query the Log and retrieve information that was saved via the CAPIFLogger class. This information can be used for further analysis like troubleshooting (identifying API calls that fail), capacity planning (identifying API calls that have slower responses) and analytics (calculating statistics like the most used endpoints, average response time, Network App usage)

In the examples folder of the SDK repository a Python script [59] is provided to demonstrate its usage.

4.2.1.3 Dummy Network Application

The existence of many entities in EVOLVED-5G ecosystem e.g., CAPIF, NEF, TSN, as well as the plethora of APIs that a Network Application should use to communicate with 5G network, make the implementation of new Network Applications quite difficult and demanding. For this reason, an additional Network App was developed to act as an

implementation example for the application developers, named after Dummy Network App.

Dummy Network App, following the paradigm of all EVOLVED-5G Network Applications, is a container-based application, including the necessary functions, written in Python, to communicate with CAPIF, NEF and TSN, using SDK libraries. The code can be found on GitHub [36] together with deployment instructions. An example of the application files can be seen at Figure 16.

Dummy Network App consists of four services:

1. **capif_callback_server**: A Python server, using Flask framework, to accept and process CAPIF callbacks.
2. **nef_callback_server**: A Python server, using Flask framework, to accept and process NEF callbacks.
3. **python_application**: This service implements the core functionality of the Network App, containing the necessary python scripts that a developer can use to communicate with the EVOLVED-5G components.
4. **redis_db**: Implementation of a database to store information exchanged with CAPIF, NEF and TSN

```
.
├── src
│   ├── capif_callback_server
│   │   ├── app.py
│   │   ├── Dockerfile
│   │   └── requirements.txt
│   ├── nef_callback_server
│   │   ├── app.py
│   │   ├── Dockerfile
│   │   └── requirements.txt
│   └── python_application
│       ├── capif_onboarding
│       ├── 0_network_app_to_nef.py
│       ├── 0_network_app_to_tsn.py
│       ├── 1_network_app_to_capif.py
│       ├── 2_network_app_to_events.py
│       ├── 3_network_app_discover_service.py
│       ├── 4_network_app_to_security.py
│       ├── 5_network_app_to_service_oauth.py
│       ├── 6_network_app_check_auth_pki.py
│       ├── 7_network_app_to_service_pki.py
│       ├── capif_registration.json
│       ├── credentials.properties
│       ├── Dockerfile
│       ├── events.json
│       ├── invoker_details.json
│       ├── Pipfile
│       ├── prepare.sh
│       ├── requirements.txt
│       ├── security_info.json
│       └── token_request.json
├── cleanup_docker_containers.sh
├── docker-compose.yml
├── env_to_copy.dev
├── redis_cli.sh
├── run.sh
├── stop.sh
└── terminal_to_python_app.sh
LICENSE
README.md
```

Figure 16 List Dummy Network App files

4.2.2 Verification Phase

The verification of a Network App is implemented using a variety of tools, as explained in Section 4.1.2. Besides configuring tools like SonarQube, Trivy and Nmap for performing quality checks on the Network App's code and its containerization, the main

focus of the verification process is to ensure that the Network App effectively communicates with the 5G network. To fulfil this goal an appropriate testing process has been set up based on the testing plans of the relevant 5G capabilities exposure tools offered by the EVOLVED-5G framework, namely, CAPIF Core Function tool, NEF emulator and TSN API.

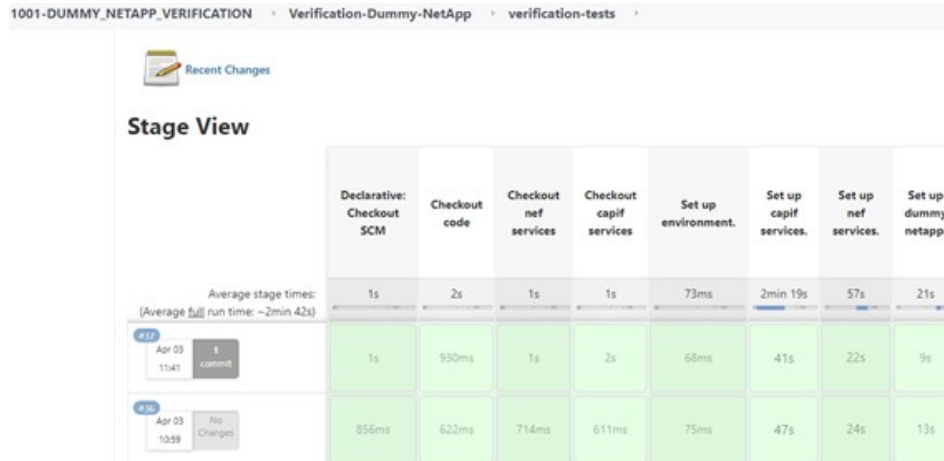


Figure 17 NEF verification pipeline (part I)

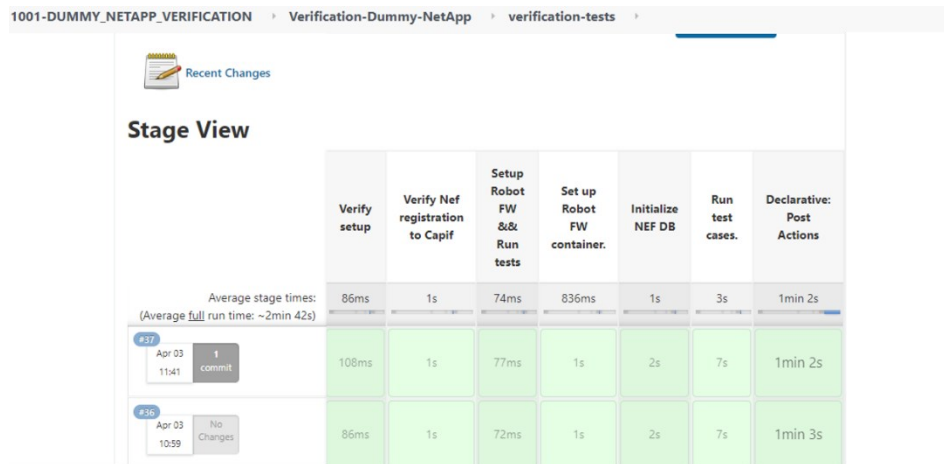


Figure 18 NEF verification pipeline (part II)

The verification of the communication of a Network App with the 5G network is realized through two pipelines in the CI/CD platform of EVOLVED-5G. The two pipelines implement NEF and TSN testing, respectively. Both pipelines require the CAPIF Core function tool in the background as the NEF and TSN APIs depend on CAPIF. The NEF testing pipeline, which is illustrated in two parts in Figure 17 and Figure 18, instantiates several Docker images to implement the communication verification of the different components. It creates the Robot Framework container, in which the code of the Network App retrieved from the EVOLVED-5G repository is also imported. At the same time, the containers of the 5G network tools, i.e., NEF emulator and CAPIF tool, need also to get configured and set up. Once all required containers are alive the actual tests are executed (step 14 “Run test cases” of the pipeline). The TSN testing pipeline is implemented similarly to the NEF pipeline, as depicted in Figure 19.

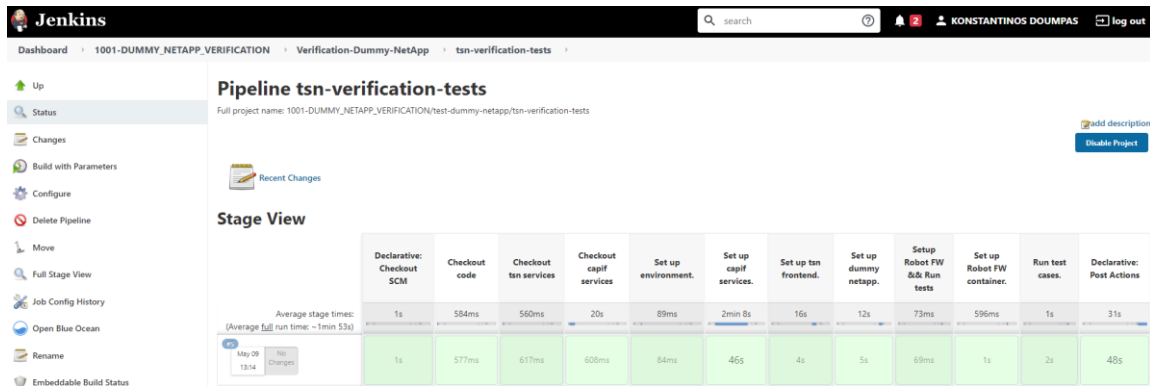


Figure 19 TSN verification pipeline

These verification tests constitute, in practice, integration tests that verify the proper invocation of the relevant CAPIF, NEF and TSN APIs by the Network App. For enabling this verification horizontally for all Network Apps, common implementation guidelines are provided by the Dummy Network Application (please, refer to Section 4.2.1.3) that was implemented for this purpose, among others. The verification process has been designed based on the Dummy Network Application. However, at the same time it is parameterized to accept any Network App as the target of the verification. Provided that a Network App implements the communication with the 5G network as exemplified by the Dummy Network Application, the verification of the Network App to 5G network communication will succeed.

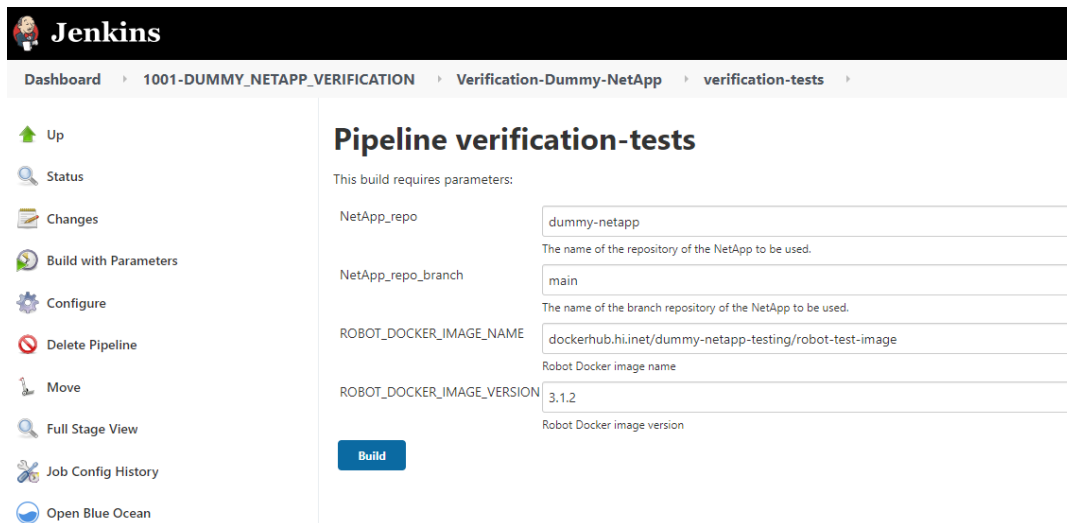


Figure 20 Parameterization of the NEF verification pipeline

Figure 20, depicts the parameters of the NEF relevant verification pipeline in the EVOLVED-5G CI/CD platform. The *NetApp_repo* and *NetApp_repo_branch* parameters specify the Network App to be verified. For the case of the example, the Dummy Network App is the one to be tested. In this figure, the parameterization of the NEF testing pipeline is shown. The TSN testing pipeline follows the same structure, as well.

Details about the specifics of the tests that verify the communication of the Network App with the CAPIF, NEF and TSN APIs are given in the following sections.

4.2.2.1 Network App-CAPIF Interaction

A Network App interacts with CAPIF in order to a) register itself to CAPIF and b) make use of services exposed by CAPIF. These interactions are described below:

Registering the Network app to CAPIF

Network App developers have to register and onboard their Network Apps to CAPIF server as a first step. The registration and onboarding need to happen once. Once this registration is complete, the Network App is recognized by the CAPIF tool and has access to view and use the available 5G-APIs. The registration can happen by calling the following CLI command `evolved5G register_and_onboard_to_capif --config_file_full_path= "path to file"`. This CLI command makes use of the CAPIFInvokerConnector SDK class, that was described at section 4.1.1.2.2. An example of the config file can be found at [37].

The configuration file contains the path to a folder where the certificates required for the Network App - CAPIF communication are to be stored.

Using services exposed by CAPIF

The Network App interacts with CAPIF in order to retrieve and use the exposed endpoints. This is manifested with the usage of the SDK Libraries that encapsulate all the calls to the CAPIF APIs.

The ServiceDiscoverer class described in section 4.1.1.2.2 allows to:

- a) to get access tokens for the specific Network App
- b) to retrieve the available endpoints. Most of the classes of the SDK libraries, as described in section 4.1.1.2.2 (LocationSubscriber, ConnectionMonitor, QosAwareness, TSNManager) use the ServiceDiscoverer in order to make the relevant CAPIF API calls.

This communication is transparent for the Network App developer since all the communication is handled by the SDK itself. In scenarios where the Network App does not have access to make relevant calls (for example because the Network App is not registered to the CAPIF server) relevant error messages are raised.

For the verification of the communication of the Network App with CAPIF, the tests check that the Network App successfully registers with CAPIF. A successful registration issues a certificate for the Network App, which is used in subsequent message exchanges with CAPIF. However, the rest of communication with CAPIF is transparent to the Network App, as it is actually implemented in the background, every time the NEF API is invoked by the Network App.

At step 8 (“*Setup dummy NetApp*”) of the verification pipeline of Figure 17, the Network App container is instantiated (code illustrated in Figure 21). Besides the registration to CAPIF that takes place at that point, the Network App also exposes a callback API, which may receive messages from CAPIF asynchronously. This is also automatically checked.

```

55     stage("Checkout capif services"){
56         when {
57             expression { RUN_CAPIF_LOCALLY == 'true' }
58         }
59         steps{
60             checkout([$class: 'GitSCM',
61                     branches: [[name: 'develop']],
62                     doGenerateSubmoduleConfigurations: false,
63                     extensions: [[$class: 'RelativeTargetDirectory', relativeTargetDir: "capif-services"]],
64                     gitTool: 'Default',
65                     submoduleCfg: [],
66                     userRemoteConfigs: [[url: 'https://github.com/EVOLVED-5G/CAPIF_API_Services.git']]
67             ])
68         }
69     }
70
71     stage("Set up environment."){
72         stages{
73
74             stage("Set up capif services."){
75                 when {
76                     expression { RUN_CAPIF_LOCALLY == 'true' }
77                 }
78                 steps {
79                     dir ("./capif-services") {
80                         sh """
81                             ls -la && cd services
82                             sed -i "s/image: mongo:6.0.2/image: mongo:4.4.17/g" docker-compose.yml
83                             ./run.sh
84                         """
85                     }
86                 }
87             }

```

Figure 21 CAPIF invocation in the verification pipelines

4.2.2.2 Network App-NEF Emulator Interaction

A Network App interacts with the NEF emulator in order to simulate calls to a 5G Network. This can speed up development time since the Network App developers can quickly create and test scenarios, using a local installation of the NEF emulator, in their own workstations.

Requirements for Network App-NEF Emulator Interaction

The Network App needs to have registered and onboarded to CAPIF (as explained in section 4.2.2.1. Provided these, the Network App can now make direct calls to the NEF emulator.

Using NEF endpoints:

The NEF APIs can be separated into two main categories:

- APIs related with monitoring devices in the network (their location or their connectivity)
- APIs related with controlling and monitoring network parameters (Quality of Service)

The Network app can make direct calls to these APIs (provided that the relevant access tokens have been received first from CAPIF) and retrieve information from the NEF emulator either synchronously (e.g., get the location of a given device) or asynchronously (e.g., get a notification when a given device changes location) as explained in section 4.1.1.2.2.

The same interaction can be achieved by using the relevant SDK classes: LocationMonitoring, ConnectionMonitoring, QosAwareness. As mentioned earlier, these

classes hide all the implementation details and all the communication with CAPIF and NEF is handled by the SDK itself.

For verifying the communication of the Network App with NEF, an elaborated testing plan has been designed and implemented covering a variety of test cases. For the monitoring event API, various test cases check that the Network App successfully subscribed and remains properly subscribed to three different suites of functionalities offered, that is (i) UE reachability, (ii) location reporting and (iii) loss of connectivity. An exhaustive list of test cases for creating, manipulating and monitoring the subscription of the Network App to these NEF APIs are executed. A similar approach is followed for checking the subscription to the second NEF API, i.e., the Quality of Service (QoS) API.

Similarly, to CAPIF, also in this case, the instantiation of the Network App container at step 8 of the verification pipeline exposes the NEF callback API of the Network App and verifies that notifications sent by NEF may asynchronously get received by the Network App.

```

89         stage("Set up nef services."){
90             when {
91                 expression { RUN_NEF_LOCALLY == 'true' }
92             }
93             steps {
94                 dir ("./nef-services") {
95                     sh """
96                         make prepare-dev-env
97                         make build-no-cache
98                         make upd
99                     """
100                 }
101             }
102         }

89         stage("Set up nef services."){
90             when {
91                 expression { RUN_NEF_LOCALLY == 'true' }
92             }
93             steps {
94                 dir ("./nef-services") {
95                     sh """
96                         make prepare-dev-env
97                         make build-no-cache
98                         make upd
99                     """
100                 }
101             }
102         }

```

Figure 22 NEF invocation in the verification pipeline

4.2.2.3 Network App-TSN APIs Interaction

A Network App interacts with the TSN APIs in order to apply Time-Sensitive Networking (TSN) standards to time-sensitive Network Apps.

Requirements for Network App-TSN FrontEnd Interaction:

The Network App needs to have registered and onboarded to CAPIF (as explained in section 3.2.2.1. Provided these, the Network APP can now make direct calls to the TSN API.

Using TSN API:

The TSN APIs allows to:

Retrieve the details of a set of preconfigured TSN profiles (that is, a set of parameters).
Apply such profiles, or customized profiles to address the specific Network App needs.
The related parameters are explained in detail at [33].

The Network app can make direct calls to these APIs (provided that the relevant access tokens have been received first from CAPIF) or make use of the TSNManager SDK class, as explained in section 4.1.1.2.2, that hides all the implementation details and takes care of the communication with CAPIF and TSN.

The proper utilization of the TSN API by the Network App is verified by relevant unit tests that check the TSN API invocation by the Network App. The tests check three different functionalities, as show in Figure 23, implemented by the Network App: (i) that applied profiles can be successfully retrieved, (ii) that new profiles may successfully be applied, (iii) that profiles can be deleted.

```

12  *** Test Cases ***
13  Initialize TSN Manager
14      [Tags]    initialize_tsn_manager
15      ${resp}=  Run Keyword    0_network_app_to_tsn.initialize_tsn
16      Log To Console    ${resp}
17      ${text}=  Set Variable If    "${resp}" == "${None}"    ${EMPTY}    ${resp}
18      Should Be Equal    ${resp}    ${text}
19
20  Valid retrieval of TSN profiles
21      [Tags]    valid_retrieval_of_tsn_profiles
22      ${resp}=  Run Keyword    0_network_app_to_tsn.get_profiles
23      Log To Console    ${resp}
24      ${count}=  Get Length    ${resp}
25      Should Be True    ${count} > 0
26
27  Apply TSN profile
28      [Tags]    apply_tsn_profile
29      ${identifier} ${clearance_token}=  Run Keyword    0_network_app_to_tsn.apply_tsn_profile
30      Log To Console    ${identifier}
31      Log To Console    ${clearance_token}
32      ${text}=  Set Variable If    "${identifier}" == "${None}"    ${EMPTY}    ${identifier}
33      Should Be Equal    ${identifier} ${text}
34      Should Not Be Empty    ${clearance_token}
35      Set Suite Variable    ${clearance_token}
36      Set Suite Variable    ${identifier}
37
38
39  Clear Profile Configuration
40      [Tags]    apply_tsn_profile
41      Log To Console    ${identifier}
42      Log To Console    ${clearance_token}
43      Run Keyword    0_network_app_to_tsn.clear_profile_configuration    ${identifier}    ${clearance_token}

```

Figure 23 Implementation of the TSN testing plan in the TSN verification pipeline

Any failure in at least one of these tests results in failure of the verification.

4.3 SECURITY

The scope of this section to focus on security aspects related to the Workspace environment, especially during development and verification phases. It describes the different measures implemented to avoid jeopardize any data of the project.

4.3.1 Development Phase

In the development phase, primary the focus is on the GitHub as it stores everything related to the Network App.

A public organization called EVOLVED-5G has been created on GitHub where all Network App repositories as well as the tools implemented during the project, such as, SDK, NEF, CAPIF, TSN, Dummy Network App, and the CI/CD pipelines are stored. Anyone searching for this project either internal or external to the consortium is able to access the organization and navigate throughout the content of the different publicly available repositories. However, it is not possible for any external developer to modify or even download the code from public repositories, to guarantee safety it is mandatory that anyone external or internal to EVOLVED-5G to be added to the GitHub organization. Every organization in GitHub is granted with a set of roles for security purposes meaning, the developer must be granted with some type of role in order to contribute to a repository enhancement within the organization.

Each repository in the EVOLVED-5G GitHub can be configured and customize as the repository creator pleases, finding different set of rules and, giving for example, the option of allowing just specific users to perform modification in the code. Additionally, GitHub also allows configuring Pull Requests (e.g., process where two different branches are merged), in order to block such process until another user (not the same launching the Pull Request) or an admin review and accept or not the changes.

Security measures are also in place to clone or create a Network App repository. In order to clone a repository, there are two main options, first one is HTTPs where a username and password is provided to authenticate the user, the second one is via SSH key previously generated by the user. Finally, to create a repository in EVOLVED-5G GitHub using the SDK, is mandatory for each user to create a personal access token and copy it in the configuration file asked when the generate command is executed, otherwise an Authentication failure message will be return to the user.

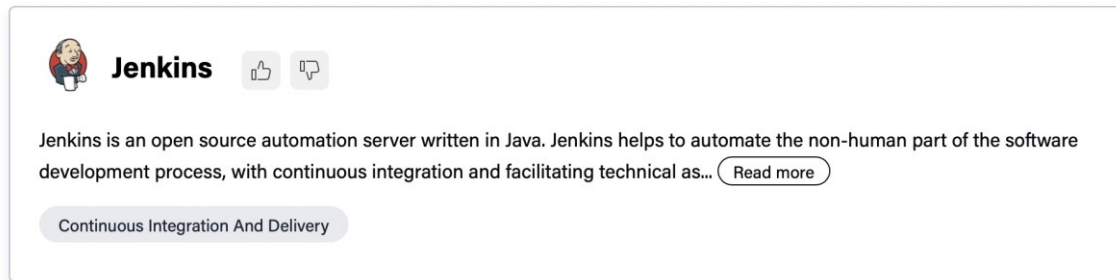
4.3.2 Verification Phase

The verification phase, although a distinct stage in the Network App lifecycle, it is in fact executed as a final step of the development phase. This means that the Network App developer invokes the verification during the development using the exact same resources and security mechanisms described in the previous section. At the same time, for the execution of the verification pipeline from the CLI tool of the EVOLVED-5G facility, certain security measures are applied by the CI/CD platform. These are explained in the next section.

Regarding the communication of the Network App with NEF and CAPIF during the verification, it is implemented over TLS, as explained in detail in Sections 4.3.2.4 and 4.3.2.5. In summary, on one hand certificates issued by CAPIF enable the authenticated usage of CAPIF services, and on the other hand, a proper authentication mechanism provides the necessary tokens to Network Apps for consuming the NEF APIs.

4.3.2.1 Security in Jenkins

Jenkins is an industry standard for CICD automation, massively adopted by software companies. According to 6sense [38], the Jenkins market share is shown in the following picture:



Market Share of Jenkins

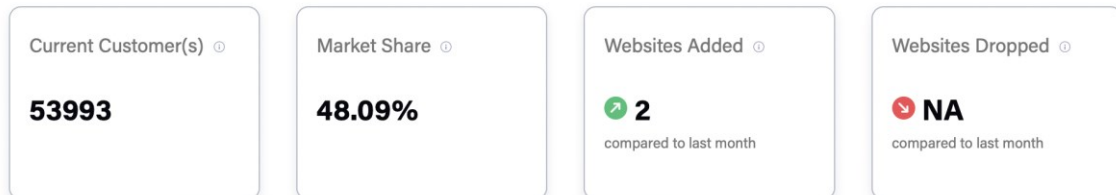


Figure 24 Jenkins Market Share numbers

Jenkins includes security in its design, and security management is large described at Jenkins documentation [39].

Telefónica's CICD environment follows strict security compliance processes that include both Authentication and Authorization management.

- Authentication (users prove who they are) is done using a security realm. The security realm determines user identity and group memberships.
- Authorization (users are permitted to do something) is done by an authorization strategy. This controls whether a user (directly or through group memberships) has a permission.

Access to all EVOLVED-5G pipelines is restricted to registered users that have the appropriate permissions.

4.3.2.2 Security in Open Repository

GitHub has security features to keep code and secrets secure. EVOLVED-5G organization has chosen the free plan which include Dependabot alerts and security updates, Security overview for repositories and security policy among others. A full list of features as well as further information is available in [40].

The container image artifact repository has a PRO X license with security built in features like Vulnerability Scanning and Premium Vulnerability Database (powered by VulnDB [41]). It is deployed in Telefónica's internal Network therefore, it is not publicly exposed.

4.3.2.3 Security in OpenShift

OpenShift is a commercial product licensed by Red Hat. This container platform includes the expected security design and implementation, and it is described in Product Documentation and books like [42].

Telefónica CI/CD environment includes OpenShift v4 as the container platform. This deployment includes security management with Authentication and Authorization enabled.

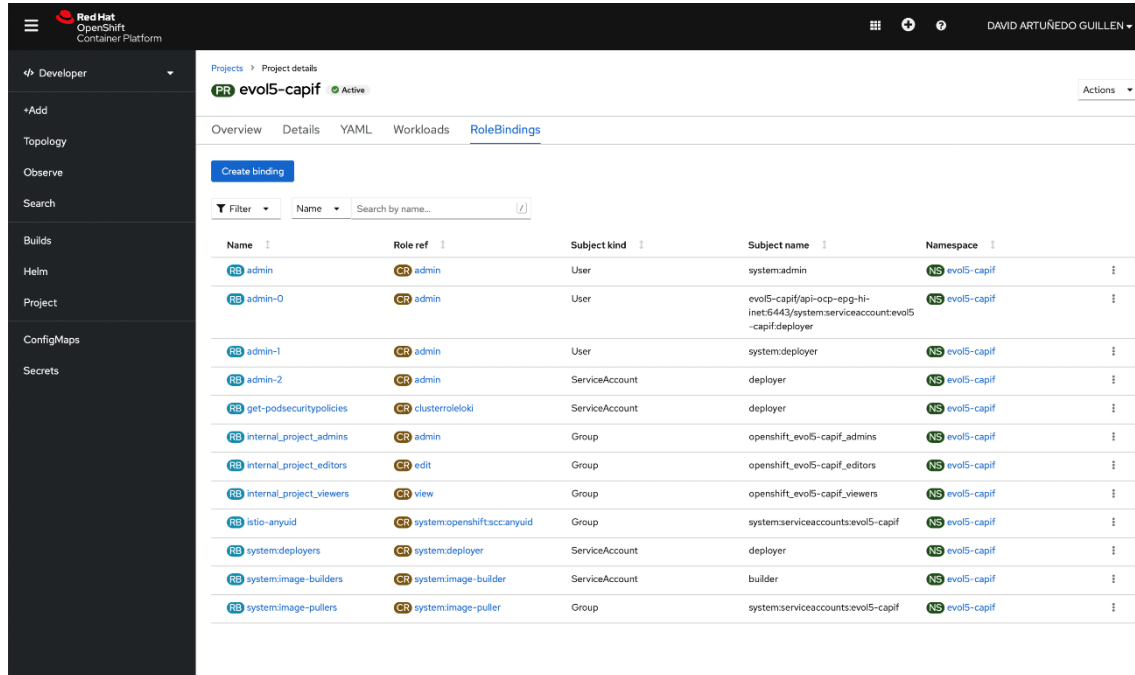


Figure 25 OpenShift console showing User Identity (upper right corner) and project permissions

OpenShift container platform has been described in D3.2 [43] Section 2.6.1, and the update will be described in D3.4.

4.3.2.4 Security in NEF

The security aspects of the Network Exposure Function (NEF) are defined in 3GPP TS 33.501 [44]. This specification outlines the requirements that should be met in the interface (N33) between the NEF and an Application Function (AF), which are the Network Apps in the context of the project. Table 2 provides an overview of these security aspects, including how to provision them and whether the NEF Emulator developed in the project supports them.

Table 2 NEF Security features

Requirements	Provisioning	NEF Emulator
Integrity protection, replay protection and confidentiality protection for communication between the NEF and Application Function shall be supported	TLS shall be used to provide integrity protection, replay protection and confidentiality protection for the interface between the NEF and the Application Function. The support of TLS is mandatory (Protection of the NEF – AF interface)	Supported

Mutual authentication between the NEF and Application Function shall be supported.	Mutual authentication based on client and server certificates shall be performed between the NEF and AF using TLS	Supported (only server side)
The NEF shall be able to determine whether the Application Function is authorised to interact with the relevant Network Functions	The NEF shall authorise the requests from Application Function using OAuth-based authorisation mechanism, the specific authorisation mechanisms shall follow the provisions given in RFC 6749	Supported
SUPI shall not be sent outside the 3GPP operator domain by NEF.	The 3GPP system stores within the subscription data the association between the GPSI and the corresponding SUPI. The GPSI is either an MSISDN or an External Identifier	Supported
Internal 5G Core information such as DNN, S-NSSAI etc., shall not be sent outside the 3GPP operator domain.	Out of scope	Out of scope

NEF Emulator supports server-side authentication based on certificates, for the authentication of the external applications (i.e., Network Apps). Additionally, the TLS protocol is used to provide integrity, replay and confidentiality protection of the NEF-AF interface (N33). After the successful authentication, the API requests from external applications are authorized using OAuth-based authorization. The use of external identifiers is an important aspect of the NEF APIs, as it ensures that the SUPI is not sent outside the 5G NPN domain. Instead, external identifiers are used, which are mapped to the corresponding SUPI within the subscription data. This provides an additional layer of security, as sensitive subscriber information is not exposed to external entities.

NEF and CAPIF communication are secured using TLS mutual authentication as described in the next section.

4.3.2.5 Security in CAPIF

Security in CAPIF is described in TS 33.310 [45]. It describes the security requirements for all CAPIF reference points, namely from CAPIF1/CAPIF-1e to CAPIF 7/CAPIF-7e. At the transport layer, the security procedures for CAPIF-1/CAPIF-1e reference points mandate that mutual authentication based on client and server certificates shall be performed between the CAPIF Core Function (CCF) and the API Provider/Invoker, using TLS. It is noted that TLS provides integrity protection, replay protection and

confidentiality protection for all the reference points of the CAPIF architecture. Certificates to be used shall follow the profiles given in 3GPP TS 33.310.

At the application/user layer, CAPIF shall be able to authenticate and authorize each user asking for API services and shall coordinate authentication and authorization between API Invoker and AEF. It is noted that, the authentication process refers to the process of verifying the identity of a user by obtaining some sort of credentials (e.g., username, password). Authorization, on the contrary, is the process of allowing an authenticated user to access resources by checking whether the user has access rights to those resources. API invocation between API Invokers and API Providers is specified in CAPIF2/CAPIF-2e reference point. This reference point allows the selection of Security methods between API Invoker and API Providers. For those processes PSK, PKI, or OAuth tokens can be used. More precisely, based on the specifications (TS 33.310), the following methods have been defined:

- **Method 1:** Use of TLS with PSK. A Pre-Shared Key (PSK) is used for the API Invoker and AEF interaction.
- **Method 2:** Use of TLS with PKI. Mutual authentication is provided. It is assumed that both API invoker and AEF are pre-provisioned with certificates created by a certificates engine.
- **Method 3:** Use of TLS with OAuth token. The CCF shall perform the functionalities of the Authorization and token protocol endpoints, the API invoker shall perform the functions of the resource owner, client and redirection endpoints functionalities, while the AEF shall perform the resource server functions.

All additional reference points in CAPIF mandates to use TLS mutual authentication. Support for TLS has been added in CAPIF Release 2.0 and Security API for managing Security contexts has been added in Release 3.0, as described in section 4.1.2.5.

4.3.2.6 Security in TSN

Security in the TSN infrastructure is achieved at two different levels: by exposing the TSN FrontEnd API through CAPIF, and by accessing the actual TSN hardware from a separate component (the TSN AF), that is located in the trusted environment.

The TSN FrontEnd makes use of the OAUTH implementation provided by CAPIF. First, the TSN FrontEnd onboards the API to CAPIF, receiving a public key as part of the process. Every invoker then discovers the API also receives a secure token generated by CAPIF, which must be sent as part of every request sent to the TSN FrontEnd. The TSN FrontEnd makes use of the CAPIF public key in order to assess the validity of the token, guaranteeing that the user is trusted.

The TSN FrontEnd does not have any access to the actual TSN infrastructure, instead, every request is processed and sent to an internal component, the TSN AF, through a private connection only available as part of the trusted environment. This enhances security, as well as allows the usage of the same TSN FrontEnd regardless of the existence or specific details of the available TSN hardware.

5 VALIDATION ENVIRONMENT

The Validation environment provides the functionality required for the execution of the Validation phase, where the Network App along with the Vertical App are tested in real or near real network conditions. The Validation environment makes use of the equipment

available in the 5G-NPN, including real 5G-NR deployments, devices and deterministic communication capabilities in the infrastructure.

The Validation environment has been described as part of Deliverable D3.1 – “Implementations and integrations towards EVOLVED-5G framework realization” [15], with minor updates presented in Deliverable D2.3 [46], in both documents as part of Section 5. This is the point of reference that we use to explain the delta between D3.3 and D3.1 and D2.3, respectively.

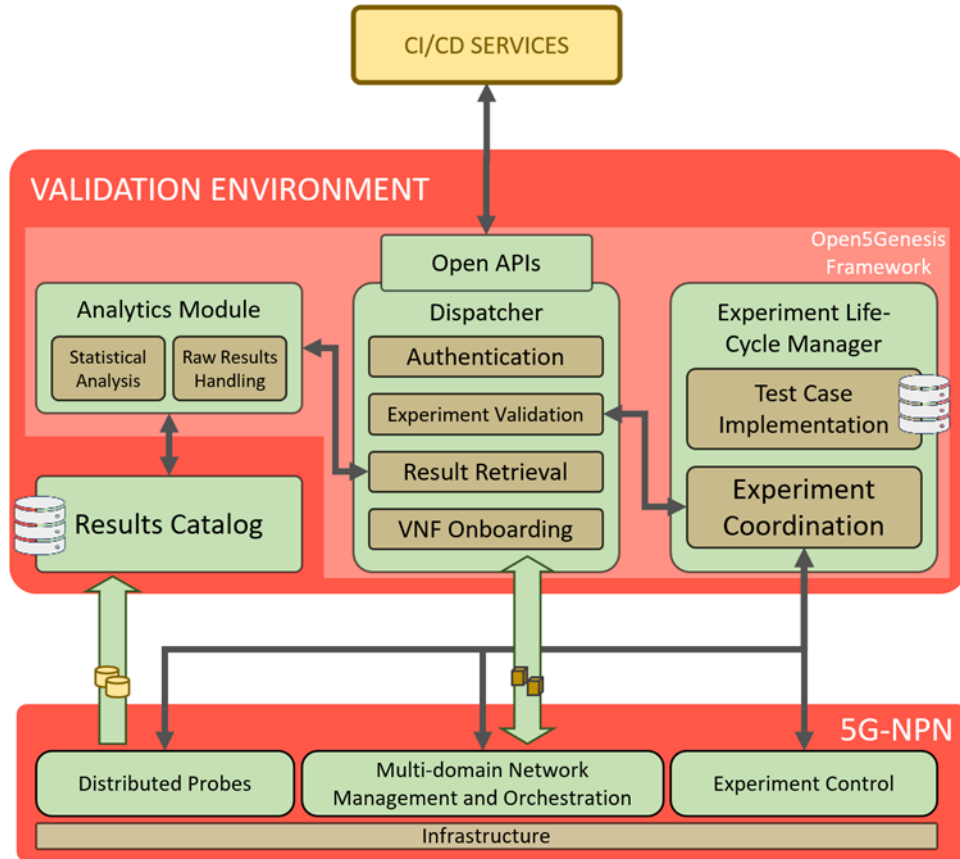


Figure 26 Validation Environment architecture

5.1 COMPONENT DESCRIPTION

Figure 26 shows the components of the Validation environment. execution requests that correspond to certain steps of the Validation (specifically steps 7 and 26 in Figure 27) are sent by the CI/CD Services to the front-end of the environment, which is implemented by the **Dispatcher** in the form of the Open APIs. The Dispatcher provides functionality for handling authentication, results retrieval and experiment execution, along with optional functionalities related to the onboarding of VNFs, which can be useful for the deployment of components of the Vertical App, in case these are provided as virtual machines.

The Dispatcher acts as a single entry point to the Validation environment, redirecting when necessary, the request to other internal components that are specifically tailored for providing specific functionalities related to the execution of experiments and retrieval of results. These internal components are:

- The **Experiment Life-Cycle Manager (ELCM)**, which is able to coordinate the execution of experiments in the 5G-NPN, orchestrating the different hardware and software components that are to be used during testing.

- The **Analytics Module**, which is able to process the raw results stored within the **Results Catalog** during an experiment execution, providing raw or aggregated measurements that can be presented either in a graphical interface or as JSON payloads.

5.2 IMPLEMENTATION

5.2.1 Validation Phase

From the point of view of the Network App developer, the execution of the Validation phase follows the same systematic and sequential design as Verification. However, there are distinct differences both from the point of view of the steps to follow and the procedural logic as detailed below:

- Network App developers request the execution of the Validation pipeline available in the CI/CD Services through the SDK (CLI tool).
- As a result, the Network App developer receives the information generated during the Validation phase as a report (pdf) in their inbox email (logs in the case of the Verification).
- Internally, the execution of the pipelines is also similar, however, in the case of the Validation, several steps involve direct use of the 5G-NPN infrastructure as well as the deployment of the Network App on the K8 clusters of the platforms (Málaga and Athens).
- The Network App developer can access any necessary physical equipment provided by the real network and radio deployments of the platforms for the execution of tests on the Vertical App.

The CI/CD services execute a set of tests over the Network App (including all the tests from the Verification phase plus additional ones related to scaling, open source software usage or interaction with the Vertical App) on both the source code and deployed instances of the application. In this case, the deployed Network App resides in the Kubernetes cluster of the Validation environment (platform), which is made available to the CI/CD services by VPN access. For the steps that make use of the physical equipment in the Validation environment, the CI/CD services request the execution, through the Dispatcher but effectively performed by the ELCM, of specific test cases defined in the platform. These test cases implement:

- The platform assessment tests, which perform an initial validation of the functionality and performance of the Validation environment, ensuring that all equipment performs within the expected thresholds before the deployment and testing of the Network App. This is step 7 in Figure 27, below.
- Any tests defined by the Vertical (i.e., the Vertical App developer and/or the Network App developer), which are related to performance and functionality assessment of the Network App plus Vertical Application, as part of step 26 in Figure 27. Such tests are defined and implemented prior to the execution of the Validation phase, through collaboration between the Vertical and the platform owners.



After the finalization of the execution for each of these steps, the CI/CD services retrieve the generated results, including the KPIs obtained during the Network App – VApp integration tests (step 26) and the performance of the platform (for example, measured throughput and latency in the radio, from step 7), which are then included as part of the Validation Report provided to the Vertical.

It is worth to mention that just the Network App is verified and validated during verification and validation phases although, during the Validation phase the Vertical App is required to validate the proper communication of the Network App against a real 5G System as well as the interaction with the Vertical App.

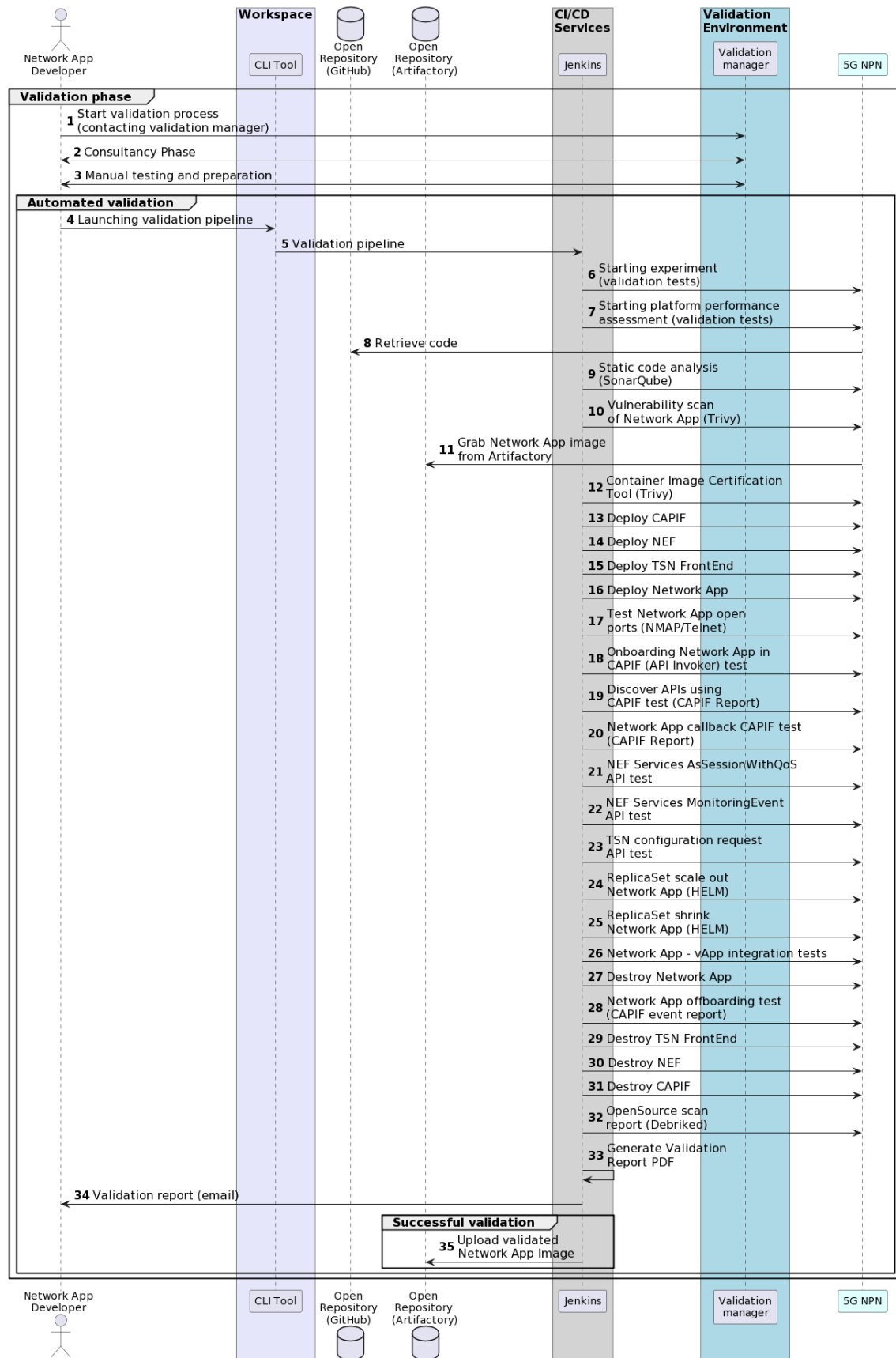


Figure 27 Validation workflow

Figure 27 shows the Validation process workflow in a visual manner. The process starts with an initial consultation phase (steps 1 to 3) in which Network App developers,

Vertical App developers and Validation managers (Platform Owners) are in contact. During this phase, the specific details and goals of the Validation are defined considering the interests and particular needs of the Network and Vertical App developers, required functionalities (e.g., for interfacing with specific devices) are implemented and initial, manual tests are performed. Ensuring the correct execution of the Automated validation, which is initiated in step 4.

The CLI tool (step 5) initiate the execution of the Validation pipeline, which include a set of actions and tests that are executed in the 5G-NPN (step 6).

First, an assessment of the status of the platform is performed (step 7), in order to ensure the correct functionality of all equipment. These tests are performed in the 5G-NPN by the Open5Genesis Framework, using the APIs described in Section 5.2.1.1.

Steps 8 to 12 include the retrieval of the Network App code and container image, and the usage of SonarQube and Trivy tools for an initial code analysis and container vulnerability scan.

The CAPIF Core Function (step 13), NEF Emulator (step 14), TSN FrontEnd (step 15) and Network App instance (step 16) are deployed in the Kubernetes environment of the 5G-NPN. Then it is assessed that the communication ports described (in the docker compose) as part of the Network App are indeed being used (step 17).

The pipeline continues with three sets of tests dedicated to the interaction of the Network App with the CAPIF and NEF services, and to test its scalability:

- CAPIF Core Function: Steps 18 to 20 test the correct onboarding, discoverability of services and reception of callbacks.
- NEF Emulator: Steps 21 and 22 assess the correct usage of the available NEF APIs
- TSN APIs: Step 23 confirm that a configuration request has been sent to the TSN FrontEnd
- Scalability: Steps 24 and 25 test support for scaling up and down the Network App.

Step 26 includes the execution of any customized tests that have been defined by the Vertical. The execution of these tests is offloaded to the Validation Environment (in particular, the Open5Genesis Framework deployed in the 5G-NPN).

The following steps are devoted to the clean-up procedures of the environment. These include destroying the Network App instance (step 27) and assessment of the correct offboarding to CAPIF (step 28), and the removal of the TSN FrontEnd (step 29), NEF (step 30) and CAPIF instances (step 31).

In step 32, Debricked [47] is used for generating a report on all the open-source licensed assets used as part of the Network App. The results of all previous steps are included in the report generated in step 33.

In every case, the Vertical receives this report via email (step 34). If all the results obtained in each of the tests performed are satisfactory, then the Network App is considered *Successfully Validated*, and the container image is uploaded to the Open Repository (artifact).

5.2.1.1 ELCM

The ELCM, which is available as open-source software in the EVOLVED-5G GitHub organization [48] is the main orchestrator and execution engine in the Validation environment. It is in charge of both the platform assessment tests and the Network App –

Vertical App tests that can be defined by the Verticals. The ELCM coordinates the execution of test cases, which are composed by a set of tasks that corresponds to the actions required for an experiment execution. Separate task types are able to perform different kind of actions, such as:

- Controlling the execution flow of an experiment depending on certain conditions.
- Retrieving or generating values that can, in turn, be used by other tasks.
- Making use of interfaces for controlling external (to the ELCM) software or hardware components.
- Delegating the execution of actions to other orchestrators, such as OpenTAP [49] or Robot Framework [50], and gathering the results.

Among others.

The initial set of enhancements and updates that have been applied to the existing code inherited from the 5Genesis H2020 project has been detailed in Deliverable 3.1 [15] Section 5.5. From this point, several additional modifications have been performed, which are detailed below.

- The 3.2 series of releases added support for verdict handling, this is, to label an experiment execution as *Pass*, *Inconclusive*, or *Fail* (among others), depending on the criteria applied to each independent steps that form the execution. This corresponds to the same idea in acceptance testing, where a test may *Pass* or *Fail* depending on certain conditions. Additionally, these releases added support for generic evaluation of expressions as part of a test case, reducing the need of adding custom code for simple computations.
- The 3.3 series did not include new functionality but added support for version 2 of the test case definition format (while retaining backwards compatibility with the previous version). Version 2 improves readability of the test cases and is the base of future enhancements implemented in the 3.6 release.
- Releases 3.4 and 3.5 include support for the execution of tests defined in Robot Framework, and updates to the NEF emulator integration.
- The 3.6 series contains a completely overhauled experiment execution logic. This release introduces the concept of child tasks, allowing the definition of experiments with a more complex execution flow akin to the capabilities of imperative programming languages. In addition, support for task labels (both automatically and manually defined) and an improved log viewer in the ELCM dashboard ease the study and debugging of experiment executions.

5.2.1.2 CI/CD services - Validation environment Interaction

As mentioned previously, the execution of the validation phase is started via the CI/CD services, which is also responsible for retrieving the generated results in order to prepare the report that is provided to the Network App developers as well as to detect issues during the process that can lead to a premature finalization of the validation. To this end, the CI/CD services make use of several endpoints handled by the ELCM and Analytics Module:

- Handled by the ELCM: Experiment execution request, execution status information, execution logs and KPIs of interest (Table 3).
- Handled by the Analytics Module: Aggregated measurements of the selected KPIs (Table 4).

Table 3 Open APIs experiment management endpoints

API prefix: /elcm		
Endpoint	Method	Description
/experiment/run	POST	Creates and queues a new experiment execution, based on the contents of the received experiment descriptor. Replies with the following response JSON: {"ExecutionId": <id>} Where <id> is a unique execution identification that can be used as input in other endpoints.
/execution/<id>/status	GET	Returns a JSON that contains general information about the status of the selected execution id, with the following schema: { "Coarse": Global status or current stage of execution, "Status": Global status or status within the current stage, "PerCent": Percentage of completion of current stage, "Messages": List of global messages generated by the execution, "Verdict": Current or final verdict of the execution }
/execution/<id>/logs	GET	Returns a JSON that contains all the log messages generated by the execution, separated by stage: { "Status": Either "Success" or "Not Found", "PreRun": Messages generated during Pre-Run stage, "Executor": Messages generated during the Run stage, "PostRun": Messages generated during Post-Run stage }
/execution/<id>/kpis	GET	Returns a list of pairs (measurement, KPI) that are of interest from all the results generated by an experiment execution, with the format: {"KPIs": [(measurement, kpi), (measurement, kpi), ...]}

Table 4 Open APIs result retrieval endpoints

API prefix: /result_catalog		
Endpoint	Method	Description
/statistical_analysis/<datasource>	GET	Returns a JSON structure that contains the statistical analysis of the selected KPIs, where <datasource> is the internal database to query. Accepts the following URL parameters: experimentid : Execution ID of the experiment, the same as returned by the experiment management endpoints. measurement : Measurement to obtain (table) kpi : KPI to obtain (column)
/get_data/<datasource>/<id>	GET	Returns a JSON structure that contains the raw measurements of the selected KPIs. Accepts the following URL parameters: measurement : Measurement to obtain (table). Returns all measurements by default

		<p>remove_outliers: 'none', 'zscore' or 'mad'. Defaults to none</p> <p>match_series: Synchronize data from multiple measurements (default: false)</p> <p>max_lag: Time threshold for synchronization (default: 1s)</p> <p>limit: Maximum number of rows to return (default: none)</p> <p>offset: Number of rows to skip on the returned results (default: none)</p> <p>additional_clause: Custom InfluxDb clause (default: none)</p> <p>chunked: Whether to retrieve the results from the server in chunks (default: false)</p> <p>chunk_size: Number of records per chunk (default: 10000, if <i>chunked</i> is enabled)</p>
--	--	---

5.3 SECURITY

Security on the Validation environment is handled by the Dispatcher component, which is in charge of assessing the validity, in terms of format correctness and authorization, of any received requests before redirecting their processing to the corresponding element internally. In addition to this, in the context of EVOLVED-5G, access to the environment is limited to the CI/CD services, which make use of secure connections provided by VPN access.

6 EVOLVED-5G INFRASTRUCTURE EVOLUTION

6.1 COMPONENT DESCRIPTION

This section aims to provide a detailed explanation of the main upgrades that have been made to the platforms from descriptions already provided in D3.1 and used in the EVOLVED-5G project to compose the 5G-NPN Infrastructure. Additionally, this section describes the NEF emulator and CAPIF Core Function tool and the enhancements that have been performed since then.

6.1.1 Athens platform evolution

A key principle of the Athens platform is to provide multiple set-ups in a modular way to support and enhance a variety of test cases and experiments related to EVOLVED-5G. A detailed description of the infrastructure regarding the Athens platform has been provided both in D2.2 and in D3.1. However, the already existing implementation has undergone some architectural updates since then, which are mainly reflected to the new interconnection with a dedicated 10G dark fibre, between the two sites that compose Athen's platform, namely NCSR and Cosmote, and the result is having two fully operational 5G SA networks. Thus, the NCSR campus site includes two radio access networks that are connected to different 5G cores, which enable further research in inter-PLMN handover and roaming scenarios.

Moreover, several updates have taken place on top of the NEF Emulator as part of the infrastructure evolution, and these updates are described in detail in section 6.1.5.

The first 5G SA network is based on ATHONET 5G SA Core and ERICSSON BBU/RRU/RAN, being deployed both at the COSMOT and NCSR Demokritos

campuses. More specifically, the ATHONET 5G Core is located in Cosmote/OTE Academy premises and is used to drive two ERICSSON BBU units, one deployed at Cosmote campus and one deployed at NCSR Demokritos campus. Each one of these two BBUs is controlling 3 ERICSSON RRU/RAN units at each domain, therefore realising a large scale 5G network with 6 indoor/outdoor cells/RAN units in total for both sites.

The latter 5G SA network is deployed only at NCSR Demokritos campus and is based on Amarisoft 5G RAN and a variety of potential 5G SA Core implementations, such as Amarisoft, Open5GS [60] and Athonet, are supported.

Although the Athens platform extension involves important infrastructure updates, it is important to acknowledge that these updates may not be fully utilized within the project's timeframe.

6.1.1.1 Core Network

Open5GS: Open5GS is an open-source 5G core network and a highly suitable option for the Athens platform due to its support for distributed NF deployments, which aligns with the evolution toward the distributed 6G architecture. One of the key advantages of using Open5GS is the ability to deploy UPF in different locations within the testbed, such as at the edge site and the core site, and associate them with different network slices (e.g., S-NSSAI). Overall, this approach allows for greater flexibility and enables the support of multiple user planes in three dimensions, including network slicing, traffic steering, and Application Function (AF) traffic influence.

Amarisoft 5G SA Core: The Amarisoft 5G Core network solution provides essential network functions for the operation of a 5G network, Access and Mobility Management Function (AMF), Authentication Server Function (AUSF), Session Management Function (SMF), User plane Function (UPF), UDM (Unified Data Management) and 5G-EIR (5G Equipment Identity Register) all integrated within the same software component.

Athonet 5G SA Core: ATHONET 5G Stand Alone (SA) core network includes two UPFs (User Plane Function) to emulate the edge and core 5G network data plane. The network also features 3GPP (3rd Generation Partnership Project) Control Plane Network Functions, including the Access and Mobility Management Function (AMF), Session Management Function (SMF), Authentication Server Function (AUSF), and User Data Management (UDM) Function. These functions enable the management and control of the network. Additionally, the network supports 3GPP interfaces, including N1, N2, N3, N4, and N6, which enable communication between network functions. This setup is hosted at COSMOTE Cloud facilities, providing a secure and reliable infrastructure.

6.1.1.2 5G RAN

In addition, the RAN part of the network has been updated by adding new equipment to enable inter PLMN capabilities between two domains.

Ericsson 5G RAN: The RAN is based on the Ericsson BBU 6630 which is a baseband unit that provides high-performance connectivity for mobile networks. The unit is compatible with various radio units, including the 4408, which is designed to provide high-capacity and low-latency connectivity for outdoor deployments. In addition to the radio unit, the system also includes the Indoor Radio Unit (IRU) 8848 and Dot 4479 B78L, which are essential components for the indoor deployment of a 5G network (depicted in Figure 3). The GPS system is used for synchronization purposes and ensures

accurate timing and location data for network operations. Together, these components form a powerful and reliable radio access network that delivers high-speed connectivity and low latency.

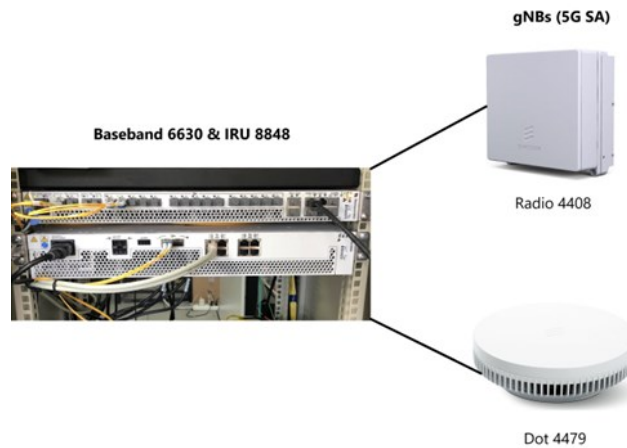


Figure 28 Ericsson 5G RAN

Amarisoft 5G RAN: The Amarisoft 5G NR can operate in FDD/TDD frequency bands below 6 GHz with up to 50 MHz of bandwidth. It supports various subcarrier spacing options for both data and synchronization signals and can operate in MIMO configurations up to 4x4 in DL. The aforementioned MIMO layers can be also complimented either in one 5G cell with 50MHz bandwidth and 2x2 MIMO configuration, or three 5G cells with 20MHz bandwidth and 2x2 MIMO configuration each. This flexibility allows for the deployment of the 5G network in different scenarios depending on the available resources and the specific requirements of the use case.

6.1.1.3 Kubernetes cluster implementation

In order to validate the Network Apps and execute validation tests, they need to be deployed in containerized infrastructure, such as Kubernetes and be executed in order to test the interaction with NEF and CAPIF. The Deployment of Kubernetes (K8s) cluster in Athens Platform consists of 3 virtual machines (VMs), 1 master node and 2 working nodes each one with the following characteristics, 2x vCPUs, 4GB RAM. Access to the nodes can be achieved via Virtual Private Network (VPN) connectivity and the overall setup of the cluster has been updated to K8s v1.26.

Compared to the setup that has been described in D3.2 the cluster has undergone some updates to be ready and functional for the validation phase. More specifically the Calico plugin has been updated to Cilium, geared to provide a flexibility towards the connectivity of the network through specific configurations and policies. Additionally, a NGINX Ingress Controller has been deployed, following the NodePort service exposure approach to route incoming requests to a certain service. The high-level architecture of the overall deployment can be seen in Figure 29 below.

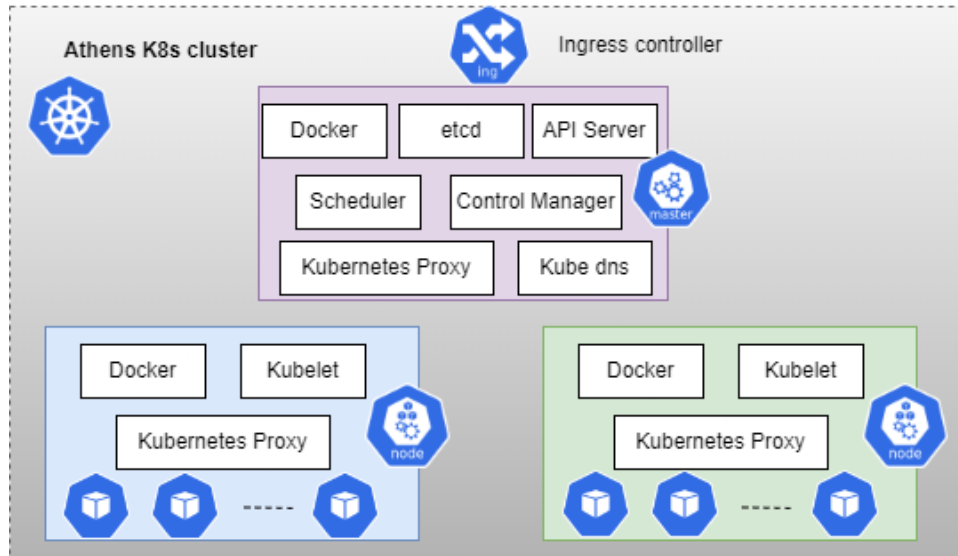


Figure 29 Athens Kubernetes Cluster

6.1.2 Málaga platform evolution

Work on the evolution of the Málaga platform, since the status described in Deliverables D2.2 [51] and D3.1 [15], can be divided in three main topics: Improvements to the Kubernetes cluster, work on the TSN implementation and exposure, and updates to the radio equipment along with the acquisition of additional UEs, which are presented in the following sub-sections.

6.1.2.1 Kubernetes cluster implementation

In the Málaga cluster, the nodes are accessed by means of a Load Balancer implemented with MetalLB [52], which allows balancing the traffic load between the services exposed in the different nodes. There is also an Ingress controller implemented with contour that allows a more elaborate way of exposing the services.

A role-based policy (RBAC) is used to allow the isolation of users using the cluster.

Containerd is used as container runtime, and for networking between the different elements of the system, Calico is used. In addition, KubeVirt is used for the virtualization of VMs in the system over the containerized infrastructure.

Finally, for system monitoring, Prometheus is used together with Grafana for the visualization of the measurements taken by the system.

6.1.2.2 Time-Sensitive Networking (TSN) over 5G

The TSN over 5G infrastructure in Málaga has evolved with respect to the one already introduced in D5.1. The current architecture of the TSN over 5G testbed at the Málaga platform is represented in Figure 30.

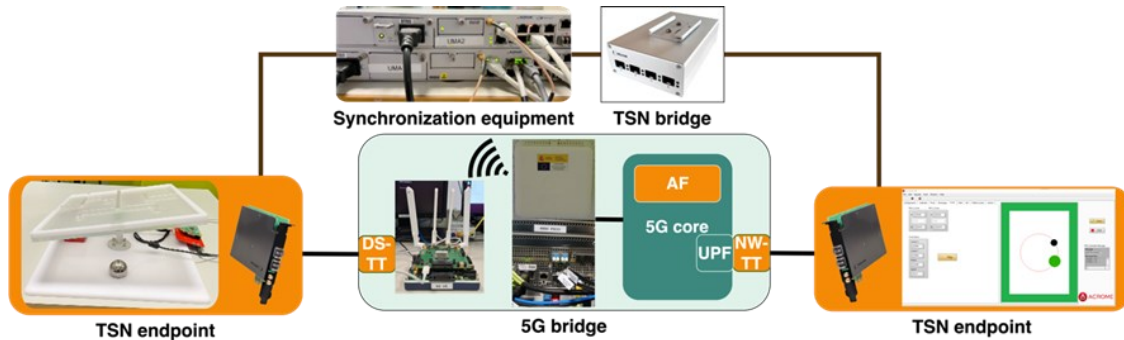


Figure 30 TSN over 5G tested at the Málaga platform

On the one hand, the existing TSN hardware is composed of one TSN Bridge, two TSN endpoints and TSN translators (DS-TT and NW-TT). In addition, we have added new hardware components to the actual ones, which are synchronization equipment in order to deal with time synchronization challenge and a ball balancing table as a visual demonstrator.

On the other hand, the TSN AF has suffered significant changes, since it is the main TSN development coming from EVOLVED-5G project. The main objective of the TSN AF is to maintain the TSN traffic requirements during a TSN session (end-to-end traffic) by reconfiguring the 5G network. We implement a Zero-Touch approach, which is a closed loop for reconfiguration. Automata Learning is used to learn the network behaviour and construct different automata by monitoring the 5G and TSN networks. The idea is to take advantage of these automata to predict possible deviations and apply the network configuration to continues meeting the TSN traffic requirements.

This component, along with the TSN FrontEnd described in Section 6.1.4, allows users to request certain QoS for the TSN traffic session in advance. It allows also to apply a network configuration beforehand. Moreover, we have developed a TSN backend to be able to apply the configuration to the Nokia equipment available at Málaga platform. Note that the FrontEnd can work without a real TSN backend. Nonetheless, in this case the TSN FrontEnd will only accept best-effort requests. In addition, The TSN FrontEnd can work with CAPIF, both for publishing the API and for securing access to the endpoints.

6.1.2.3 Radio evolution, user equipment and devices

In Málaga's site and within Universidad of Málaga (UMA) premises, the Nokia radio infrastructure has been upgraded with the new software version of radio equipment SBTS22R3. This release includes all the new features required in order to meet the latest 3GPP Release 16 requirements.

Additionally, the 4 Remote Radio Heads (RRHs) that provide outdoor 5G Standalone (SA) and Non-Standalone (NSA) and 2 pico RRHs that provide indoor 5G SA have been changed in order to support the new subband 42 of band n78 of Telefonica 5G in Spain and a bandwidth of 100 MHz.

UMA has acquired new UEs and modems for testing the SA mode and millimeter wave (mmWave) performance, which can be seen in Figure 31:

- **Telit's FT980-WW advanced LTE/5G wireless router:** Supports mmWave bands n257, n258, n260, n261. Supports Standalone (SA) & Non-Standalone (NSA) network. Supports 5G FR1 DL 4x4 MIMO DL and 5G FR1 UL 2x2 MIMO.
- **One Plus 11 5G:** Supports Standalone (SA) & Non-Standalone (NSA) network.

- **Google Pixel 7 5G:** Supports Standalone (SA) & Non-Standalone (NSA) network.



Figure 31 FT980-WW, One Plus 11 5G and Google Pixel 7 5G

6.1.3 CAPIF Core Function Tool evolution

CAPIF tool has been extensively described in D3.1 [15] and D3.2 [43]. At the time of D3.2 publication, CAPIF Release 1.0 was described, with basic APIs for API Invokers registration, API Publication and API Discovery. CAPIF Core Function tool has continue its evolution adding functionality and APIs specified in TS 29.222 [18].

CAPIF Release 2.0 was published by 12/09/2022 and Release notes were published at [53]. This release included the following changes:

- **CAPIF Provider Management API:** implementation of API Providers registration in CAPIF Core Function.
- **CAPIF Events API:** CAPIF Events subscription management and Event notifications to API Invokers and API Providers.
- **TLS Enabled:** TLS with mutual authentication is a strong requirement in most 3GPP specifications.

In order to support TLS, CAPIF Core Function has included a Certificate Authority to expedite and manage Certificates. For this purpose, Easy_RSA project has been included [54].

CAPIF Release 2.1 was published on 03/10/22 with bugfixing and some code improvements.

A major refactoring of CAPIF Core Function was performed for Release 3.0. This release was published on 27th January 2023, with Release Notes available at [55].

Besides code refactoring in this release, new APIs were added to this version:

- **Security API:** This API enables the creation of Security Contexts for managing access from API Invokers to published APIs by API Providers. The security options implemented included PKI (Certificates) and OAuth Tokens.
- **Logging Service API:** This API enables the registration of API invocations by API Providers into CAPIF Core Function for auditing and charging purposes.
- **Auditing Service API:** This API enables querying CAPIF Core Function to retrieve API logs submitted by API Providers.

Release 3.1 followed on 10/03/2023 including improvements on clearing entities in CAPIF Core Function when API Providers and API Invokers are deleted from CAPIF Core Function.

With this release, CAPIF Core Function includes all the functionality required by EVOLVED-5G verification, validation and certification processes. There are no major releases planned within the context of EVOLVED-5G project.

6.1.3.1 CAPIF Core Function Tool implementation

Implementation of CAPIF Core Function has been performed using open-source tools and projects, taking as reference input CAPIF API Release 17.0 descriptions by 3GPP from [56].

Besides developing the CAPIF Core Function (CCF) API Services, it has been defined the Test Plans for testing every API endpoint and it has been implemented automated tests. A testing strategy has been defined to improve the code quality and the interaction of the provided CCF and any API Invoker or Provider. This testing strategy is composed of two steps:

- **Test plan document elaboration:** In this step, test plans are described, including various behaviour scenarios (considering both success and failure cases). The test plan structure includes clarifications on the pre-conditions, the action that takes place, and the post-conditions (response/result expected). The horizon of the test plans that have been defined, moves beyond the request-response information that is available in the related 3GPP specifications, in a sense that behavioural scenarios are defined to stress test the implementation integrity.
- **Test implementation and execution:** This step continues after finishing the elaboration of the test plan documentation. Each test suite is implemented and included into an automation pipeline that checks the status of the code after every deployment in the platform.

The tools that have been used for implementing and testing the CAPIF API services are the following:

OpenApi Generator: This software program allows generation of API clients SDKs (Software Development Kit tools) or API servers given an OpenAPI specification. Moreover, it is possible to generate code in more than 20 different programming languages. Having the YAML files of the CAPIF services (as described above), we used OpenApi generator to automatically create code that implements HTTP/HTTPS Endpoints that act as Servers that accept HTTP requests.

MongoDB: Mongo is a non-SQL and open-source database tool used to provide storage to different CAPIF core functionalities.

Nginx: Nginx is an open-source web serving technology that we use as a reverse proxy to forward requests to the different CAPIF modules (Figure 32Figure 1)

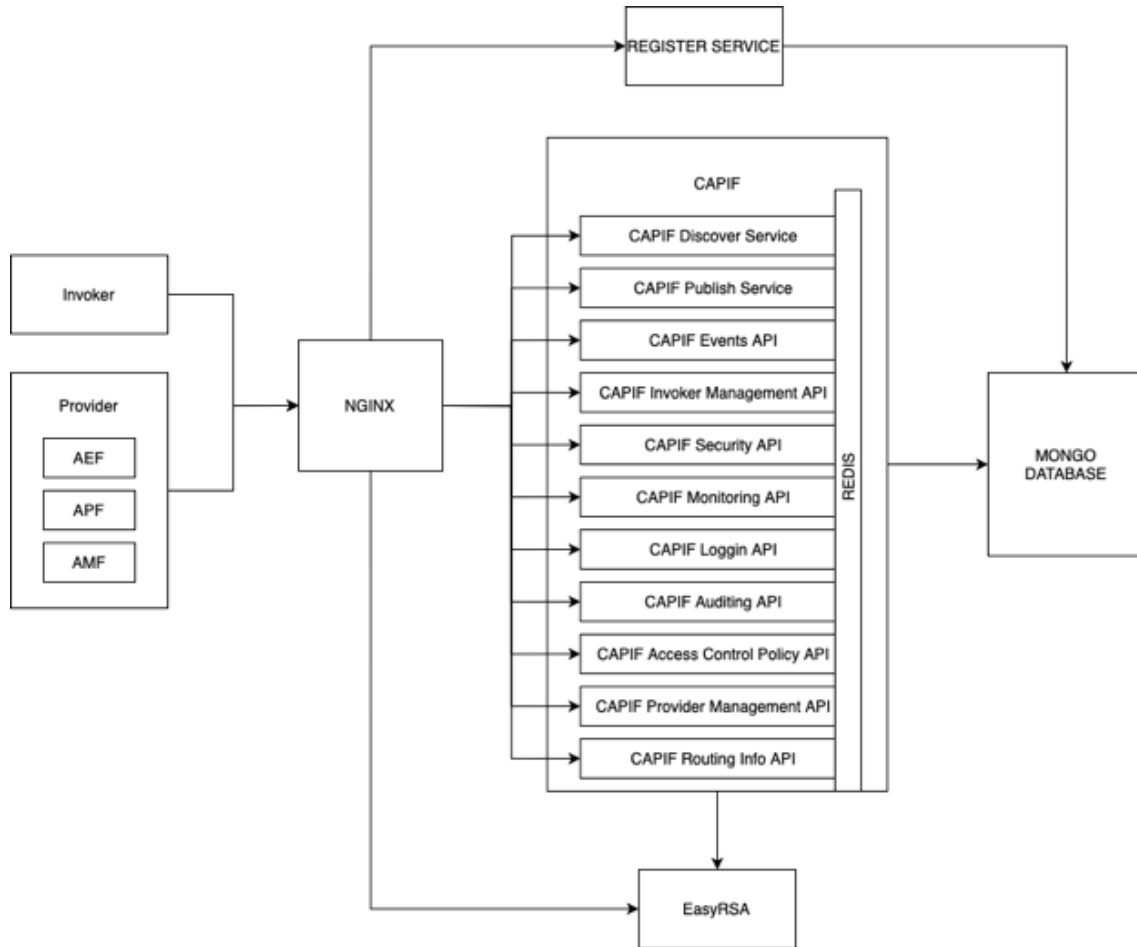


Figure 32 NGINX implements TLS mutual authentication and routes API requests to CAPIF Core Function Services

Flask: Flask is a micro-service web framework written in Python used to build CAPIF. The main advantage of this framework is its modularity. This feature allows us to run different CAPIF services and mix them directly with other services as database with relative ease.

Robot framework: The Robot Framework is a free and open automation testing framework. It is suitable for test automation as well as robotic process automation (RPA). We have implemented a CAPIF Test plan that covers all CAPIF APIs implemented with all possible responses, both successful ones and failure responses.

Easy RSA: Easy RSA Certification Authority has been integrated into CAPIF implementation, to enable CAPIF Core Function to generate its own CA Root and Certificates for TLS mutual authentication. API Invokers and API Providers receive their own certificate during the CAPIF onboarding process. They use this certificate for establishing TLS security layer with CAPIF Core Function. API Invokers and API Providers verify CAPIF Core Function certificates using the CA Root generated by CAPIF Core Function, and CAPIF validates API Invokers and API providers validating the Certificate presented in TLS negotiation.

OAuth2.0: OAuth2.0 is an open standard for authorization and is based on the concept of access tokens. A token is a string that the OAuth client uses to make requests to the resource server. Once a component registers and authenticates itself with credentials, CAPIF core function issues an access token. The component can use that token, within its requests, to access specific resource APIs. As in OAuth, various formats for the tokens can be supported, 3GPP specifications propose JSON Web Token (JWT) as the most

suitable format for token exchange in CAPIF ecosystem. JWT format offers stateless tokens, thus, saving memory on server side as the session information is not stored, and also ensures authenticity of the client by data signing. JWT is the mandatory format of tokens in this standard, and the tokens are stored in a cookie.

Docker: Docker is an open-source containerization tool for building, running, and managing containers, where software is deployed. We use Docker to build each service of the CAPIF Core Function. In this way, we can keep each CAPIF service development isolated from other services. This design pattern is known as a micro-services-oriented architecture and is widely used in software development due to its innumerable advantages like better fault isolation and improved scalability, among others.

6.1.4 TSN FrontEnd

The TSN FrontEnd has been designed as a unified, re-usable front-end for requesting network configurations related to TSN and deterministic communication. The TSN FrontEnd can expose a set of pre-defined configuration, which can then be customized by end-users. On every configuration request, the TSN FrontEnd pre-process the requirements and generates a set of values that are then sent to the TSN AF, which is in charge of making use of the actual hardware in order to achieve the requested parameters.

6.1.4.1 TSN FrontEnd implementation

The TSN AF has been implemented as a containerized Flask application, easing the deployment. It has been published as open-source software (Apache 2.0 license) and is available in the EVOLVED-5G GitHub organization [33].

The application acts as a server that exposes an API with the following endpoints (Table 5):

Table 5 TSN FrontEnd API endpoints

API prefix: /tsn/api/v1		
Endpoint	Method	Description
/profile	GET	When used without parameters the endpoint returns a list of available profiles, with the following format: {"profiles": ["<profile1>", "<profile2>", ..., "best_effort"]} When used with the <i>name</i> =<profile_name> URL parameter, the endpoint returns the default configuration values of the selected profile, with the format: {"<profile_name>": {"<parameter1>": <value1>, "<parameter2>": <value2>, ...}}
/apply	POST	Applies the specified configuration to the selected traffic identifier. The endpoint expects to receive a payload with the format: {"identifier": <identifier>, "profile": <profile>, "overrides": <overrides>} Where: <ul style="list-style-type: none"> - <i>Identifier</i> is a unique identifier defined by the user for the configuration. - <i>profile</i> is the name of the base profile to use. The values in this profile will be used as default, when not overridden.

		<ul style="list-style-type: none"> - <i>overrides</i> is a dictionary of values that will be overridden from the used profile. May be empty.
/clear	POST	<p>Disables the configuration applied by a previous usage of /apply. The endpoint expects to receive a payload with the format:</p> <pre>{"identifier": <identifier>, "token": <token>}</pre> <p>Where:</p> <ul style="list-style-type: none"> - <i>identifier</i> is the same unique identifier used when requesting the configuration through /apply <p><i>token</i> is a value returned by the TSN AF as part of the response to the /apply request.</p>

End users make use of the /profile endpoints in order to receive information about the available profiles already pre-configured. Then, they can make use of the /apply endpoint for requesting a particular configuration.

On a successful request, the /apply will reply with a payload with the following format:

```
{"message": "Success", "token": "<token>"}
```

The token included is not related to the authentication procedure implemented through the CAPIF integration but, is a separate measure that ensures that other third parties with access to the TSN FrontEnd cannot interact with the configuration of other experimenters only by knowing the *identifier* of the configuration. This extra validation is included because the TSN FrontEnd can be deployed without any CAPIF integration, if decided by testbed administrators.

After a configuration is no longer necessary, end users can disable it by making use of the /clear endpoint.

6.1.4.1.1 Integration with the TSN infrastructure

In order to increase security and to allow re-usability of the TSN FrontEnd in different environments, the TSN FrontEnd does not have access to the TSN hardware. Further, the TSN FrontEnd is also prepared to work on environments where no TSN capabilities are available.

This is achieved by the definition of a separate entity, the TSN Application Function (AF), that is in charge of accessing the equipment and apply the configuration by any means necessary, and which only needs to expose a small API that can be used by the TSN FrontEnd in order to send the requested configuration values. Information about this API can be seen in Table 6.

Table 6 TSN AF API endpoints

Endpoint	Method	Description
/apply	POST	<p>Applies the specified configuration. The expected payload has the following format:</p> <pre>{"identifier": <identifier>, "token": <token>, "values": <values>}</pre> <p>Where:</p> <ul style="list-style-type: none"> - <i>identifier</i> is a unique configuration identifier. - <i>token</i> is a unique token related to the configuration. - <i>values</i> is a dictionary of configuration values.

/clear	POST	Disables the configuration applied by a previous usage of /apply. For convenience, the same payload used for /apply is sent.
--------	------	--

In order to keep the implementations as open and extendable as possible, there are no preconceptions about how the TSN AF makes use of the information sent in the payload, or the specific values included in the *values* dictionary. It is only expected that the TSN AF will reply with

- HTTP status 200 on successful requests.
- Any other status code along with a JSON payload that includes a *detail* message indicating the issue, in case of a failed request.

Additionally, the TSN FrontEnd can work in *BackEnd-less* mode, for cases where the underlying infrastructure has no access to TSN capabilities. In this case, the TSN FrontEnd can listen for requests, but will only accept best-effort configurations.

6.1.5 NEF Emulator evolution

Beyond the exposure capabilities, NEF Emulator has been evolved to conform with the other components of the ecosystem. NEF Emulator is compatible with CAPIF Core Function which adds an extra layer of protection to the network infrastructure, preventing direct access by untrusted parties but also ensuring the discoverability of the services. In addition, NEF Emulator's microservice composition is restructured with the addition of a reverse proxy (NGINX) that handles all the traffic in a secure way. The use of RESTful APIs throughout all layers of the simulator allows for most components, such as cells, UEs, Northbound APIs, OAuth2 tokens, and more, to be accessible as APIs. This enables third-party applications, proprietary equipment, and AL/ML services to interact with the components of each layer, providing flexibility and ease of integration. To demonstrate the use of RESTful APIs and third-party applications, an Android companion app has been developed to simulate the movement of a UE by dynamically changing its coordinates.

6.1.5.1 Companion application

The companion app provides the GPS coordinates of the UE to the NEF emulator, which allows for dynamic movement of the UE. It has been specifically designed for Android devices and requires GPS availability (outdoors) as well as connectivity to the NEF emulator via either Wi-Fi or a mobile network (Figure 33).

The configuration details for the companion app are described below. Note here that a convention is to use the "companionapp.com" as the domain part of the external identifier.

- User ID: The external identifier of the UE in NEF emulator. This value should always be "<somevalue>@companionapp.com"
- NEF emulator hostname: The hostname or IP address of the deployed NEF emulator
- NEF emulator port: The port number of the deployed NEF emulator
- NEF emulator username: The username of the deployed NEF emulator to acquire the login token
- NEF emulator password: The password of the deployed NEF emulator to acquire the login token

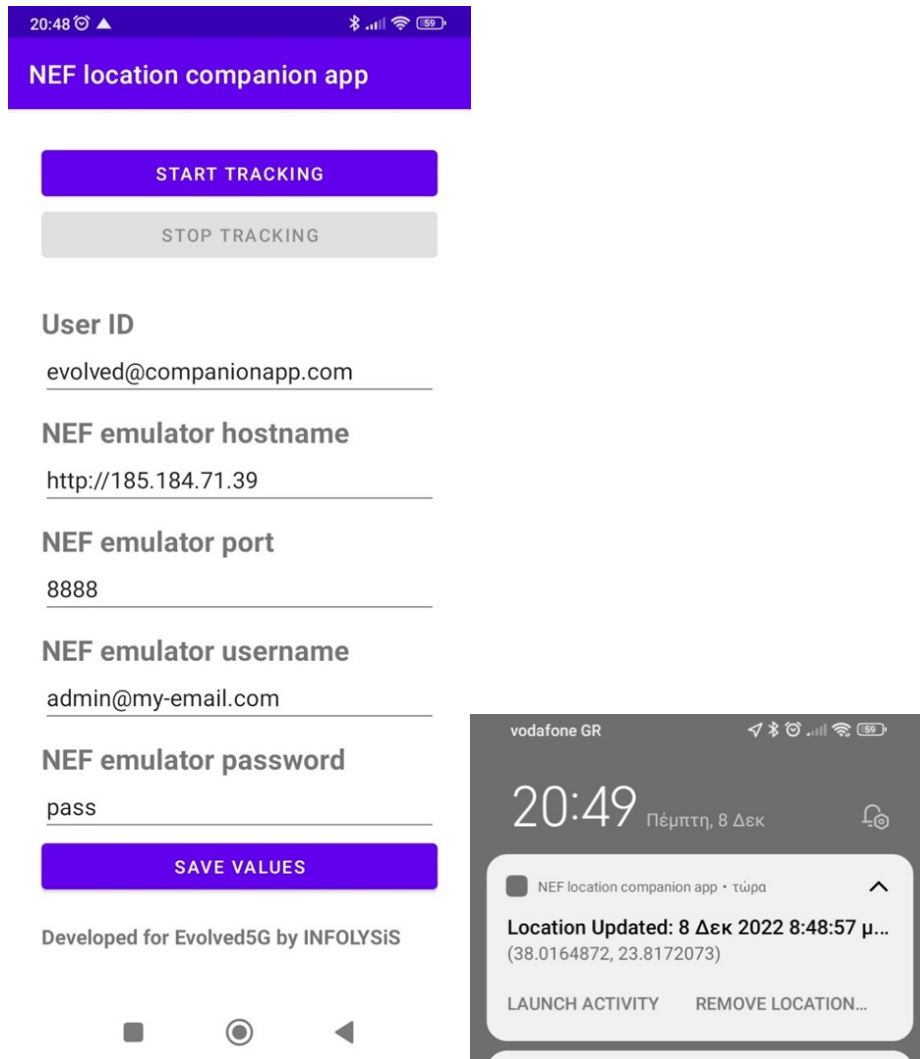


Figure 33 Companion App

After saving the values and the "START TRACKING" button is pressed, the companion app will begin sending GPS coordinates to the NEF emulator (periodically every 10 seconds). Additionally, the app will continue to send GPS coordinates in the background, which is indicated by a permanent notification as depicted in Figure 33.

Communication with the NEF Emulator is facilitated through the following APIs:

1. Login Access Token API (HTTP POST): This API is used to obtain the login token for accessing the NEF Emulator.
2. Create UE API (HTTP POST): This API is used to obtain the SUPI of the UE. If the external identifier provided does not exist, a new one is generated randomly. If it already exists, the existing one is returned.
3. Update UE API (HTTP PUT): This API is used continuously by the app to update the GPS coordinates of the UE being tracked, as long as tracking is enabled.

6.1.5.2 NEF Emulator Implementation

As mentioned, the addition of a reverse proxy (NGINX) is an important aspect of the evolution of the NEF emulator since it covers most of the security requirements (Figure 34). NEF Emulator's server-side authentication mechanism relies on self-signed certificates created using OpenSSL protocol, during the build process (i.e., the certificate

is signed by the server's private key). The TLS handshake procedure is a built-in feature of the NGINX solution, while the authorisation of the network apps is achieved with jwt tokens generated from NEF (when NEF is used without CAPIF).

In Figure 34, the interaction with a network app and the NEF Emulator through the reverse-proxy is demonstrated. During deployment (i.e., docker compose up), the exposed ports of the container for both HTTP and HTTPS can be dynamically configured through environmental variables. The ports of the docker network that are assigned in the reverse proxy service are always 80 for HTTP and 443 for HTTPS protocols. If a network app sends an HTTP request to the reverse proxy (step 1), subsequently the request then is redirected with a 301 Moved Permanently response (step 2) to a secure HTTPS request (step 3) to the proxy service. In case the network app uses HTTPS directly, steps 1-2 are omitted.

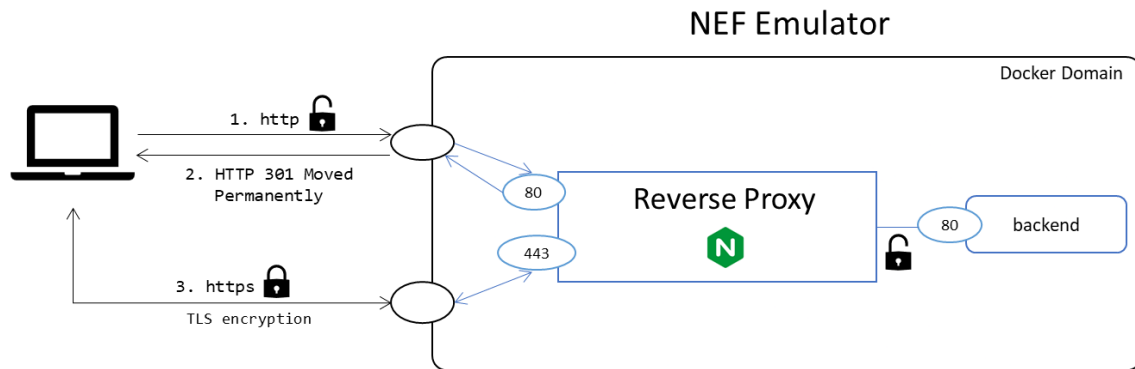


Figure 34 NEF Emulator

In conclusion, NEF Emulator will keep upgrading to ensure its compliance with the latest versions of the various technologies it depends on, such as Python, Docker Compose, and CAPIF, among others.

7 CONCLUSION

This deliverable describes final results of activities performed in tasks T3.1, T3.2 and T3.3, during the second period of WP3, concretely from M13 to M28.

This deliverable presents a final upgrade of Deliverable D3.1 and includes additional information about the current EVOLVED-5G architecture and the different environments and processes the Network App will follow.

Task 3.1 aims to design and implement the development tools for the workspace environment. Such tools, offers a set of functionalities and features to create, verify and launch the validation phase of the Network Apps. In this deliverable, it is described the updates for the final releases of the different tools involved in the Workspace environment including security features as well as new implementations as TSN.

Task 3.2 focuses on the design of the validation environment. In this environment the developer' Network Application is challenged among different validation tools implemented to validate the feasibility of the Network App in the EVOLVED-5G framework towards one of the final phase (certification). Furthermore, this task will provide to developers with a report notifying weather their Network App has successfully pass or not the validation phase, in case of a positive report, the validation phase is in charge to store the validated Network App in a repository for the certification phase.

Task 3.3 supports the integrations between all the phases for the Network App. In D3.1 the requirements where fed by WP2 while now, due to requirements and updates in the implementations, the architecture has evolved, and the modification are reflected in this deliverable. Also, feedback from WP4 and WP5 is required to align the architecture with the requirements coming from developers and certification entity. Finally, the infrastructure evolution is considered in this Task, describing and presenting all the new upgrades related to Athens and Málaga platform in this deliverable.

Regarding next steps, it must be considered that some other modules, will also be part of the integrated EVOLVED-5G platform (i.e., the Certification environment and the Marketplace) but are designed and developed as part of T3.4, and thus, they will be covered in deliverable D3.4, to be released in M30.

REFERENCES

- [1] EVOLVED-5G WIKI, from <https://wiki.evolved-5g.eu/>
- [2] EVOLVED-5G FORUM from <https://forum.evolved-5g.eu/>
- [3] EVOLVED-5G Library from <https://forum.evolved-5g.eu/c/library/>
- [4] Vue.js - The Progressive JavaScript Framework | Vue.js, from <https://vuejs.org>
- [5] Node.js, from <https://nodejs.org/en>
- [6] PostgreSQL: The world's most advanced open-source database, from <https://www.postgresql.org/>
- [7] MySQL, from <https://www.mysql.com/>
- [8] MariaDB Foundation - MariaDB.org, from <https://mariadb.org/>
- [9] SQLite Home Page, from <https://www.sqlite.org/index.html>
- [10] GraphQL | A query language for your API, from <https://graphql.org/>
- [11] Git, from <https://git-scm.com/>
- [12] Analytics, from <https://analytics.google.com/analytics/web/>
- [13] Slack is your productivity platform | Slack, from <https://slack.com/>
- [14] Manage Your Team's Projects From Anywhere | Trello, from <https://trello.com/>
- [15] EVOLVED-5G, Deliverable 3.1 "Implementations and integrations towards EVOLVED-5G framework realisation (intermediate)" from <https://evolved-5g.eu/wp-content/uploads/2022/01/EVOLVED-5G-D3.1-v1.0.pdf>
- [16] EVOLVED-5G, from <https://github.com/EVOLVED-5G/>
- [17] EVOLVED-5G, Deliverable 5.3 "NetApp Certification and Release to Marketplace (intermediate)" from https://evolved-5g.eu/wp-content/uploads/2023/02/EVOLVED-5G-D5.3-Final_version.pdf
- [18] Common API Framework for 3GPP Northbound APIs (3GPP TS 29.222 version 16.5.0 Release 16). https://www.etsi.org/deliver/etsi_ts/129200_129299/129222/16.05.00_60/ts_129222v160500p.pdf
- [19] 5G System; Network Exposure Function Northbound APIs; Stage 3 (3GPP TS 29.522 version 16.4.0 Release 16). https://www.etsi.org/deliver/etsi_ts/129500_129599/129522/16.04.00_60/ts_129522v160400p.pdf
- [20] Time-Sensitive Networking (TSN) Task Group, from <https://1.ieee802.org/tsn/>
- [21] 3GPP TS 23.501 V16.16.0 (2023-03) 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS); Stage 2 (Release 16). https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-gg0.zip
- [22] 3GPP TS 23.501 V17.8.0 (2023-03) 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; System architecture for the 5G System (5GS); Stage 2 (Release 17). https://www.3gpp.org/ftp/Specs/archive/23_series/23.501/23501-h80.zip
- [23] Procedures for the 5G System (5GS) (3GPP TS 23.502 version 16.7.0 Release 16). https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/16.07.00_60/ts_123502v160700p.pdf
- [24] System architecture for the 5G System (5GS) (3GPP TS 23.501 version 17.5.0 Release 17).

- https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/17.05.00_60/ts_123501v170500p.pdf
- [25] Terraform|HashiCorp Developer, from <https://developer.hashicorp.com/terraform>
 - [26] Helm, from <https://helm.sh/>
 - [27] EVOLVED-5G/SDK-CLI: SDK-CLI for EVOLVED-5G H2020 project, from <https://github.com/EVOLVED-5G/SDK-CLI>
 - [28] SDK-CLI/netapp_capif_connector_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/netapp_capif_connector_examples.py
 - [29] SDK-CLI/netapp_service_discovery_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/netapp_service_discovery_examples.py
 - [30] SDK-CLI/location_subscriber_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/location_subscriber_examples.py
 - [31] SDK-CLI/connection_monitor_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/connection_monitor_examples.py
 - [32] SDK-CLI/ qos_awareness_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/qos_awareness_examples.py
 - [33] EVOLVED-5G/TSN_FrontEnd, from https://github.com/EVOLVED-5G/TSN_FrontEnd
 - [34] SDK-CLI/tsn_manager_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/tsn_manager_examples.py
 - [35] SDK-CLI/ tsn_capif_connector_examples.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/tsn_capif_connector_examples.py
 - [36] EVOLVED-5G/dummy-network-application, from <https://github.com/EVOLVED-5G/dummy-network-application>
 - [37] SDK-CLI/netapp_capif_connector_config_file.json at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/netapp_capif_config/netapp_capif_connector_config_file.json
 - [38] Jenkins - Market Share, Competitor Insights in Continuous Integration And Delivery, from <https://6sense.com/tech/continuous-integration/jenkins-market-share>
 - [39] Securing Jenkins, from <https://www.jenkins.io/doc/book/security/>
 - [40] GitHub security features - GitHub Docs, from <https://docs.github.com/en/code-security/getting-started/github-security-features>
 - [41] VulnDB, from <https://vuln.db.cyberriskanalytics.com/>
 - [42] OpenShift Security from <https://www.redhat.com/rhdc/managed-files/cl-openshift-security-guide-ebook-us287757-202103.pdf>
 - [43] EVOLVED-5G, Deliverable 3.2 “NetApp Certification Tools and Marketplace development(intermediate)” from https://evolved-5g.eu/wp-content/uploads/2022/09/EVOLVED-5G-D3.2_FINAL.pdf

- [44] Security architecture and procedures for 5G System (3GPP TS 33.501 version 17.5.0 Release 17), https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/17.05.00_60/ts_133501v170500p.pdf
- [45] Network Domain Security (NDS); Authentication Framework (AF) (3GPP TS 33.310 version 17.3.0 Release 17), https://www.etsi.org/deliver/etsi_ts/133300_133399/133310/17.03.00_60/ts_133310v170300p.pdf
- [46] EVOLVED-5G, Deliverable 2.3 “Overall framework for NetApp development and evaluation” from https://evolved-5g.eu/wp-content/uploads/2022/12/EVOLVED-5G_D2.3.pdf
- [47] Debricked - Your Partner in Open Source, from <https://debricked.com/>
- [48] GitHub - EVOLVED-5G/ELCM: Experiment Lifecycle Manager. Developed by the University of Málaga, from <https://github.com/EVOLVED-5G/ELCM>
- [49] OpenTAP, from <https://opentap.io/>
- [50] Robot Framework, from <https://robotframework.org/>
- [51] EVOLVED-5G, Deliverable 2.2 “Design of NetApps development and evaluation environments” from https://evolved-5g.eu/wp-content/uploads/2021/11/EVOLVED-5G-D2.2-v1.0_final.pdf
- [52] MetalLB, bare metal load-balancer for Kubernetes, from <https://metallb.universe.tf/>
- [53] Release CAPIF Release 2.0 · EVOLVED-5G/CAPIF_API_Services · GitHub, from https://github.com/EVOLVED-5G/CAPIF_API_Services/releases/tag/2.0
- [54] Easy RSA, from <https://easy-rsa.readthedocs.io/en/latest/>
- [55] Release CAPIF Release 3.0 · EVOLVED-5G/CAPIF_API_Services · GitHub, from https://github.com/EVOLVED-5G/CAPIF_API_Services/releases/tag/3.0
- [56] 5G APIs · GitLab, from https://forge.3gpp.org/rep/all/5G_APIs#capif-common-api-framework
- [57] EVOLVED-5G, Deliverable 2.1 “Overall Framework Design and Industry 4.0 Requirements” from https://evolved-5g.eu/wp-content/uploads/2021/11/EVOLVED-5G-D2.1_v1.4.pdf
- [58] EVOLVED-5G/NetworkApp-Template, from <https://github.com/EVOLVED-5G/NetworkApp-template>
- [59] SDK-CLI/nef_logger_and_audit_example.py at master · EVOLVED-5G/SDK-CLI · GitHub, from https://github.com/EVOLVED-5G/SDK-CLI/blob/master/examples/nef_logger_and_audit_example.py
- [60] Open5GS from <https://open5gs.org/>
- [61] SDK-CLI/HISTORY.rst at master · EVOLVED-5G/SDK-CLI · GitHub, from <https://github.com/EVOLVED-5G/SDK-CLI/blob/master/HISTORY.rst>
- [62] EVOLVED-5G, Deliverable 6.1 “Training Material and SMEs and Startup Acceleration Program Activities (Intermediate)” from <https://evolved-5g.eu/wp-content/uploads/2022/09/EVOLVED-5G-D6.1.pdf>
- [63] Robot Framework, from <https://robotframework.org/>